

To the Graduate Council.

I am submitting here with a thesis written by Densu Aktas entitled "Enhancement of PROBEPIC Code and Application to Langmuir Probes." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

Trevor M. Moeller, Major Professor

We have read this thesis
and recommend its acceptance.

Roy Schulz

Gary Flandro

Accepted for the Council.

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

**Enhancement of PROBEPIC Code and
Application to Langmuir Probes**

A Thesis Presented for
The Master of Science
Degree
The University of Tennessee, Knoxville

Densu Aktas
May 2010

Copyright © 2010 by Densu Aktas
All rights reserved.

DEDICATIONS

This thesis is dedicated to my Mother,

Sara Alexis

Whom I love very much

“Happy indeed is the scientist who not only has the pleasures which I have enumerated, but who also wins the recognition of fellow scientists and of the mankind which ultimately benefits from his endeavors.”

-Irving Langmuir, inventor of the Langmuir Probe, also coined the term "plasma".

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of my friends, especially Nicholas A. Lister, and my grandma. I also want to thank the members of the First Baptist Church in Nashville, Tennessee, who provided important financial and spiritual support, especially Mr. Morris Early “grandpa.”

I would like to thank my major advisor, Dr. Trevor M. Moeller, for his assistance and guidance, and my committee for their patience and support.

I would also like to thank my mentor and friend, Mr. Paul Gloyer. Without his help, I never would have been able to make it through all of the challenges that I faced in the last two years. His guidance and council could never be repaid. He served as a wonderful example and showed me how to be a professional engineer.

Tanrı hepsini korusun ve kutsasin.

ABSTRACT

The Langmuir probe is used as a diagnostic device to determine the properties of plasma, which is a state of matter that is represented by partially or wholly ionized gas containing free charged particles; free moving electrons and ions. This probe is used to determine plasma characteristics such as temperature and density by carefully analyzing the probe voltage-current (V-I) characteristics. However, real world effects, such as finite length geometry, cause the measurements from these probes to deviate from the theoretical predictions for ideal probes.

To understand these discrepancies and predict the performance of real probes, computer simulations using the Particle-in-Cell (PIC) and Monte Carlo techniques have been used. These simulations provide insight to help understand experiments that can provide detailed information that is difficult to measure, and help design new experiments and probes.

The current effort expands upon the work performed by Thomas Markusic that resulted in the creation of a 2D/3V PIC code, PROBEPI, in 1996 at UTSL, to simulate a Langmuir probe to analyze the plasma behavior. In the current effort, Markusic's code is updated from C to C++ and also converted into FORTRAN. In addition to resolving coding issues, this effort identified

several opportunities to enhance the effectiveness of the tool. By adding a ray tracing model and modifying the boundary condition models, the accuracy of the tool was improved.

Chapter I provides an introduction to the effort, including a discussion of plasma characteristics and the Langmuir probe theory. This chapter also contains brief information about Markusic's simulation, code upgrades and the code conversion process. Chapter II is focused on the description of the PIC code and the improvements and updates made in PROBEPIC. The issues encountered during the creation of the new version of the PROBEPIC code are also addressed along with their solutions. Chapter III presents the results that show the conversion was successful and that the new results are in agreement with both previous data and theoretical predictions. Finally, Chapter IV contains conclusions and some recommendations for future work.

TABLE OF CONTENTS

CHAPTER I	1
Introduction and General Information	1
1.1 Langmuir Probe Description	4
1.1.1 Langmuir Probe	5
1.1.2 Thomas Markusic's Simulation	14
1.2 PIC Monte Carlo Description	17
1.3 Approach	21
1.3.1 Transformed Probe PIC to FORTRAN and the Changes Made	21
CHAPTER II	25
Description of POBEPIC	25
2.1 Details on Probe PIC	26
2.2 Improvements/Updates	28
2.3 Comparison of New/Old Langmuir Simulation	39
CHAPTER III	40
Results	40
CHAPTER IV	46
Conclusions and Recommendations	46
4.1 Conclusions	46
4.2 Recommendations	47
BIBLIOGRAPHY	48
APPENDICES	50
Vita	323

LIST OF TABLES

Table 1: Quiescent plasma simulation conditions in PROBEPICT [2]	17
Table 2: PROBEPICT sub-program description [2].....	23
Table 3: A sample of editable parameter.txt	37

LIST OF FIGURES

Figure 1: A simple Langmuir probe set up [6]	7
Figure 2: Schematic of a typical Langmuir probe	8
Figure 3: Typical Voltage–Current (V–I) characteristic for a Langmuir probe	9
Figure 4: The view of the objects in a typical PIC code [2]	15
Figure 5: A typical cycle, a onetime step, in a particle simulation program (PIC) [7]	20
Figure 6: PROBEPIE computational domain layout created [2]	27
Figure 7: Field boundary conditions and probe geometry in PROBEPIE	28
Figure 8: The computational grid in PROBEPIE (before correction)	29
Figure 9: The computational grid after the correction of grid.c in PROBEPIE	30
Figure 10: Particle ray tracing shown in animation window	33
Figure 11: The rectangular domain in the original PROBEPIE	34
Figure 12: Cylindrical domain boundary with particle injection over 90 degrees	36
Figure 13: The locations where the particles injected and hit the probe	38
Figure 14: Voltage–Current (V–I) characteristics of original and updated PROBEPIE code	41
Figure 15: Probe grid with 50 percent lower density	44
Figure 16: Effects of 50 percent reduced grid density	45

CHAPTER I

Introduction and General Information

Plasma can be defined as quasi-neutral ionized gas with a roughly equal number of positively and negatively charged particles. This collection of free-moving electrons and ions makes electromagnetic effects important for the plasma physical behavior. Unlike gasses, liquids or solids, plasma does not contain only neutral molecules. Instead, it contains ions/atoms/molecules and accounts for more than 99 percent of the known visible universe. Even the solar system is permeated with plasma, and the earth is completely surrounded by plasma trapped within its magnetic field.

The American chemist, Irving Langmuir, was the first scientist to apply the term “plasma” to describe an ionized gas in the mid-twenties. From his research, Langmuir invented a new probe device to measure the properties of plasma [1]. Even today, the Langmuir probe is still an important tool in plasma diagnostics and research. Furthermore, his basic work with this probe opened the door to the development of many other methods to measure the composition and thermodynamic properties of

plasma, e.g. interferometer (optical and microwave), spectroscopy, spectral intensity and electrostatic probes [6].

With proper design and careful data analysis techniques, the Langmuir probe can provide important information concerning plasma parameters. This information includes the ion density, the plasma electron density, the plasma potential, the plasma electron temperature, and the plasma electron energy distribution function (EEDF) [8]. Langmuir's method accomplishes this with the measurement of the electron and ion currents to a small metal electrode, i.e. Langmuir probe, as different voltages are applied to the probe. This yields a curve called the probe characteristic of the plasma (described in section 1.1.1).

Since the masses of electrons and positive ions are very different, they respond to forces on different time scales. This, combined with their opposite charges, produces the varying response seen in the probe characteristic curve. For example, large negative probe voltages produce a small magnitude current for the heavy slow positive ions, while large positive voltages produce a high magnitude current due to the fast electrons. However in equilibrium plasmas, the number of electrons equals the number of ions (i.e. $n_e = n_i$) in the plasma to satisfy the requirement of overall charge neutrality.

While the Langmuir probe theory provides an explanation of the ideal behavior of a Langmuir probe, the real world effects result in measurements that deviate from the ideal voltage–current characteristic curve [2]. Since these differences can impact the data interpretation and understanding of plasma, it is important to have the means to predict the real world effects prior to testing.

To accomplish this, researchers [2, 3, 7] have applied modern computational tools to simulate the interaction of a Langmuir probe with plasma. Using particle in cell (PIC) and Monte Carlo techniques, it is now possible to predict the probe characteristic curve to some degree of accuracy. A notable example of such a tool is the PROBEPIC code developed by Thomas Markusic and documented in his Master of Science Thesis [2]. PROBEPIC extends the work of previous treatments [3] by simulating the behavior of finite length probes. Markusic used this tool to conduct a parametric study on the Langmuir probe and interpret available experimental data [2]

The purpose of the current research presented in this thesis is to update, enhance and evaluate the PROBEPIC code to improve its accuracy and operation on modern computers. Specifically, the code was updated from C language to C++ and then transformed into the FORTRAN language to provide two updated versions of the

PROBEPIC code. As part of this effort, a variety of minor issues were resolved and simple refinements were made. Additionally, the current research provided opportunities to refine underlying assumptions in the original code by evaluating its performance. These features were addressed by implementing enhanced models that improve the accuracy of the tool. With the enhanced version of PROBEPIC, updated voltage-current characteristics for a reference Langmuir probe were generated and are presented in Chapter III.

1.1 Langmuir Probe Description

As previously mentioned, many fundamental plasma parameters can be determined by placing a small conducting probe into the plasma and observing the current to the probe as a function of the difference between the probe and plasma space potentials (i.e. voltages). This simple electrostatic device is known as a Langmuir probe. In this section the Langmuir probe theory is explained, and the computational simulation of the Langmuir probe pursued by Thomas Markusic in PROBEPIC is described.

1.1.1 Langmuir Probe

In plasma, since the electrons have smaller masses, they have higher thermal speeds than ions at the same temperature. As a result the electrons will migrate to the probe faster than the ions, causing the probe to collect the electrons at a higher rate. Eventually, this process establishes an electric current to the probe and causes the floating potential of the probe to become less than the plasma potential, which is the potential of the plasma with respect to the walls of the device at a given location [4]. Although plasma potential, V_p , is usually a few volts positive with respect to the device walls, electron and ion densities away from the probe are considered to be essentially equal to each other.

In general, between any uninterrupted neutral plasma and a boundary wall (at a lower potential than the plasma) there is a region where the electrons are repelled and the positive ions are attracted. The thickness of this area is called the plasma sheath. Debye shielding length, λ_D , is a characteristic distance in plasma beyond which the electric field of a charged particle is shielded by opposite sign charged particles, and it is calculated by the following equation [5].

$$\lambda_D = \left(\frac{\epsilon_0 * k * T_e}{e^2 * n_e} \right)^{1/2} = 7.4 * 10^3 * \left(\frac{T_{eV}}{n_e} \right)^{1/2} \quad (1.1)$$

$$T_{eV} = \frac{k * T_e}{e} \quad (1.2)$$

where k is the Boltzmann constant, T_e is the electron temperature in Kelvin, T_{eV} is the electron temperature in electron volts ($11,600 \text{ K} \equiv 1 \text{ eV}$), e is the electron charge and n_e is the electron density in (m^{-3}).

Initially, it was thought that plasma potential could be determined by measuring the potential on the probe relative to the reference electrode. However, later on, it was determined that the plasma potential had to be determined from the floating potential, V_f , of the probe. Although the plasma potential and the floating potential are completely different, the real difference between them could not be measured until Irving Langmuir and Harold Mott Smith provided an experimental method to determine the plasma potential in 1920s [2, 6]. From this work, they developed the Langmuir Probe to provide measurements of electron density, temperature and ion density from simple electric measurements.

A typical Langmuir probe set up uses a ramp voltage that is applied to the probe positioned in the plasma to generate a probe current, which is typically recorded with an oscilloscope, to determine the voltage–current (V–I) characteristic. The data from the oscilloscope is then sent to a computer for further analysis which allows one to determine electron temperature and density. Figure1 shows the typical experimental set up for Langmuir probe applications.

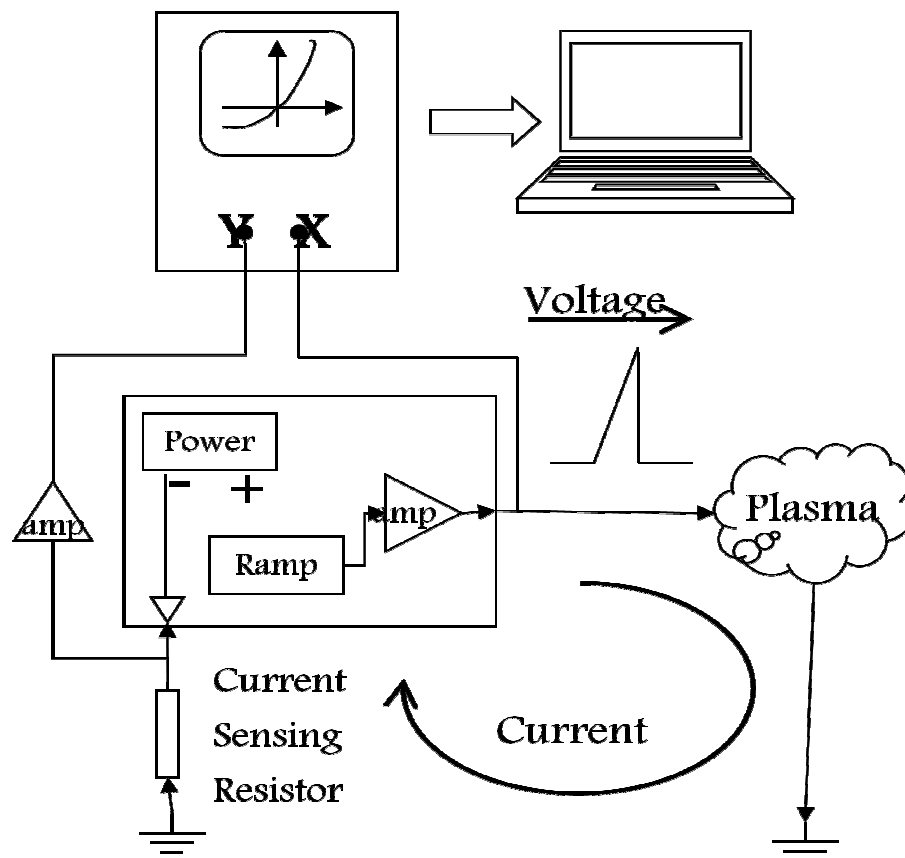


Figure 1. A simple Langmuir probe set up [6]

In general, the Langmuir probe is nothing more than a sphere, cylinder or a disk which is made out of a conducting substance immersed into plasma. Figure 2 illustrates the typical Langmuir probe shape. The typical construction of a Langmuir probe includes two sections, the probe shaft and the probe tip. The probe tip is usually a metal, such as tungsten, and is normally insulated by mica or quartz. Only a small section of the probe (i.e. probe tip) is exposed to the plasma and electrically biased with respect to a reference electrode to collect electron and/or positive ion currents. With this configuration, the probe is exposed to a large flux of ions and electrons, the magnitude of which depends upon the density and the temperature of the particles and the voltage applied to the probe. The charged plasma particles collide with the exposed probe surface, and this allows the Langmuir probe to draw electrical current.

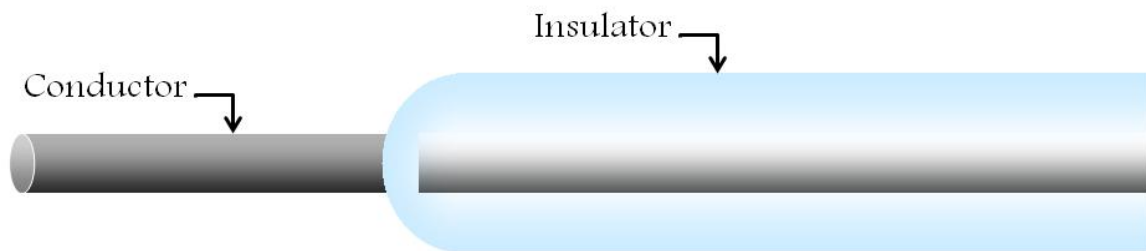


Figure 2. Schematic of a typical Langmuir probe

Irving Langmuir's method consists of obtaining the voltage-current (V-I) characteristic of the probe when the bias voltage, the voltage between the probe and reference electrode, is swept from negative to the positive potential. Figure 3 illustrates the typical current-voltage characteristic for a Langmuir probe. The probe current-voltage characteristic shown in Figure 3 can be typically divided into several regions as described below [5].

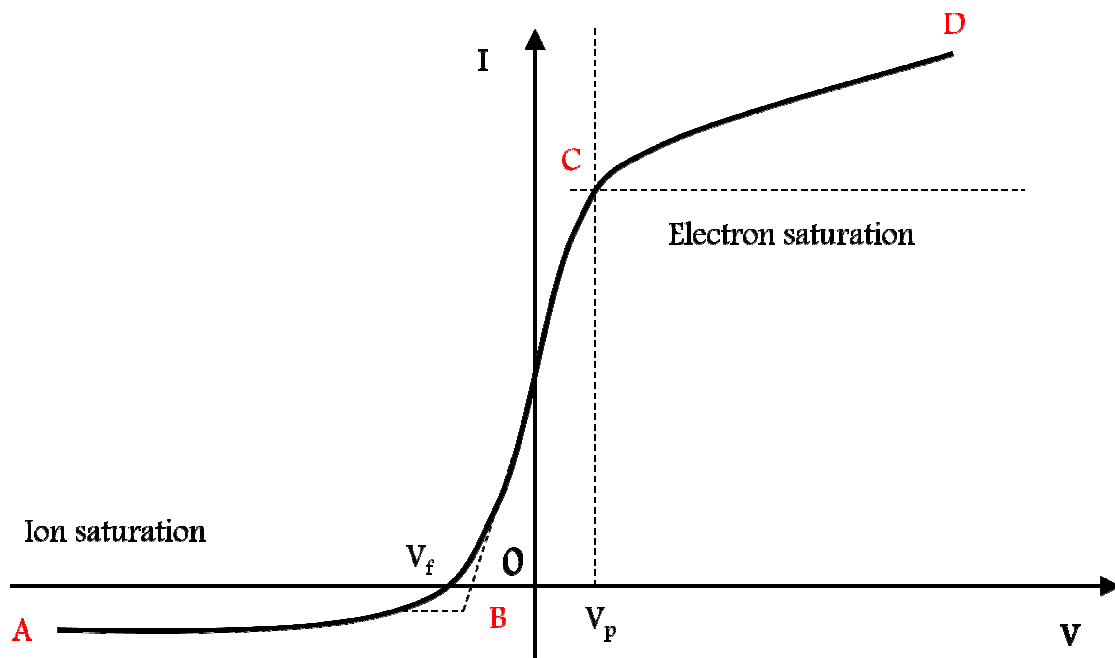


Figure 3: Typical Voltage-Current (V-I) characteristic for a Langmuir probe

As seen in Figure3, the floating potential is the point where the current collected is zero, designated V_f ($V=V_f$). At this point, the ion and electron currents are equal and the probe current is zero. The floating potential can be found with the equation below [5].

$$V_f = -\frac{k * T_e}{e} * \ln \left(\frac{m_i}{4 * \pi * m_e} \right)^{1/2} \quad (1.3)$$

where m_i is the mass of ion and m_e is the mass of electron.

If the probe potential is less than the floating potential, $V < V_f$, then the probe current is mainly positive ion current. Ion saturation occurs rapidly as V is reduced below V_f , as shown between points A and B (Figure 3). On the other hand, if the probe potential is greater than the plasma potential, $V > V_p$, the result is space charge limited electron current, indicating electron saturation, as shown between points C and D (Figure 3). However, if the probe potential is between the floating and the plasma potential, $V_f < V < V_p$, then the current is mostly due to electron-diffusion to the probe. While theory for a probe in ideal plasma indicates that the current in this region would vary linearly with the probe voltage, in reality the current varies exponentially with the probe voltage [5].

When the probe potential equals the plasma potential, $V = V_p$, the electrons and ions diffuse to the probe as if they were unaffected by its presence. The local zero condition ($V = V_p$) is characterized by the lack of a plasma sheath. On either side of this point, the difference in voltage introduces electric forces that create a boundary region known as the plasma sheath, a region where charge neutrality does not exist.

When there is no plasma sheath, the charged particles can migrate to the probe freely due to their thermal velocities. In this case, it is assumed that the probe current, I_p , does not interrupt the plasma equilibrium, and the probe diameter, r_p , is considered less than the electron mean free path. Furthermore in this case, the electrons are considered to be in thermal equilibrium at a temperature T_e with a Maxwellian kinetic energy distribution. In this case the electron current diffusing to the probe can be found with the following equation [5].

$$I_p = \frac{1}{4} * A * n * e * \bar{u} \quad (1.4)$$

where n is the electron density in the immediate vicinity of the probe surface when $V < V_p$, A is the surface area of the probe, e is electron charge and $\bar{u}$ is average electron speed. Here, n can be found using the following equation [5].

$$n = n_e * \exp\left(\frac{-e * (V_p - V)}{k * T_e}\right) = n_e * \exp\left(\frac{(V_p - V)}{k * T_{ev}}\right) \quad (1.5)$$

where k is Boltzmann constant, T_e is electron temperature, n_e is the electron density in the undisturbed plasma at potential V_p , V_p is plasma potential (also called space potential) and V is probe potential. The average electron speed can be calculated with the equation below [5].

$$\bar{u} = \sqrt{\left(\frac{8 * k * T_e}{\pi * m}\right)} = \sqrt{\left(\frac{8 * e * T_{ev}}{\pi * m}\right)} = 6.7 \times 10^5 \sqrt{T_{ev}} \text{ ms}^{-1} \quad (1.6)$$

Taking the natural logarithm of equation (1.5) and with the help of the equation (1.4), the following formula can be found [5].

$$\ln I_p = \ln I_{ps} - \frac{V_p}{T_{ev}} - \frac{V}{T_{ev}} \quad (1.7)$$

where I_{ps} is the probe current when $V = V_p$, and it is calculated using the following equation [5].

$$I_{ps} = \frac{1}{4} * A * n_e * e * \bar{u} \quad (1.8)$$

Then the slope of the curve $\ln I_p$ versus probe potential, V , is given by the formula [5].

$$\frac{d \ln I_p}{dV} = \frac{1}{T_{ev}} \quad (1.9)$$

By differentiating the logarithm of the electron current with respect to the probe voltage, V , for $V < 0$, the electron temperature can be obtained. In this case, the slope of voltage versus $\ln I_p$ is a straight line in the region between the plasma potential and the floating potential. It is important to note that the straight line is achieved only if the distribution is Maxwellian. Again, this slope determines the electron temperature using the equation (1.9). This equation also shows that the positive ion current is small and can be neglected in most of the region except very close the V_f [5]. Also using the value obtained for T_{ev} , the mean electron speed can be obtained from equation (1.6) and the electron density can be obtained from equation (1.8) [5].

The aim of this sub-section was to give the reader a brief description of the Langmuir probe and the basic theory to show how it is applied to determine properties of the plasma. More detailed information can be found in Markusic's thesis [2].

1.1.2 Thomas Markusic's Simulation

During his graduate study, Thomas Markusic created Particle-in-Cell (PIC) simulations to show the behavior of the finite length Langmuir probes [2]. He used PIC as a computational method to simulate the plasma phenomena. The PIC contains particles, grids and boundaries. In PIC, many discrete particles are used to simulate the collective behavior of plasma to come as close to physical reality as possible. A spatial grid on which the electromagnetic fields are computed is superimposed on the computational domain. The charges of the particles in the grid cells are weighted to the grid nodes; the electric field is calculated on the nodes using Poisson's equation; and the charge particles are accelerated by the electric field [7]. Figure4 illustrates the objects in the PIC code. The boundaries in Figure4 are the computational boundaries; the grid is a mathematical object but particles and boundaries represent the real physical objects.

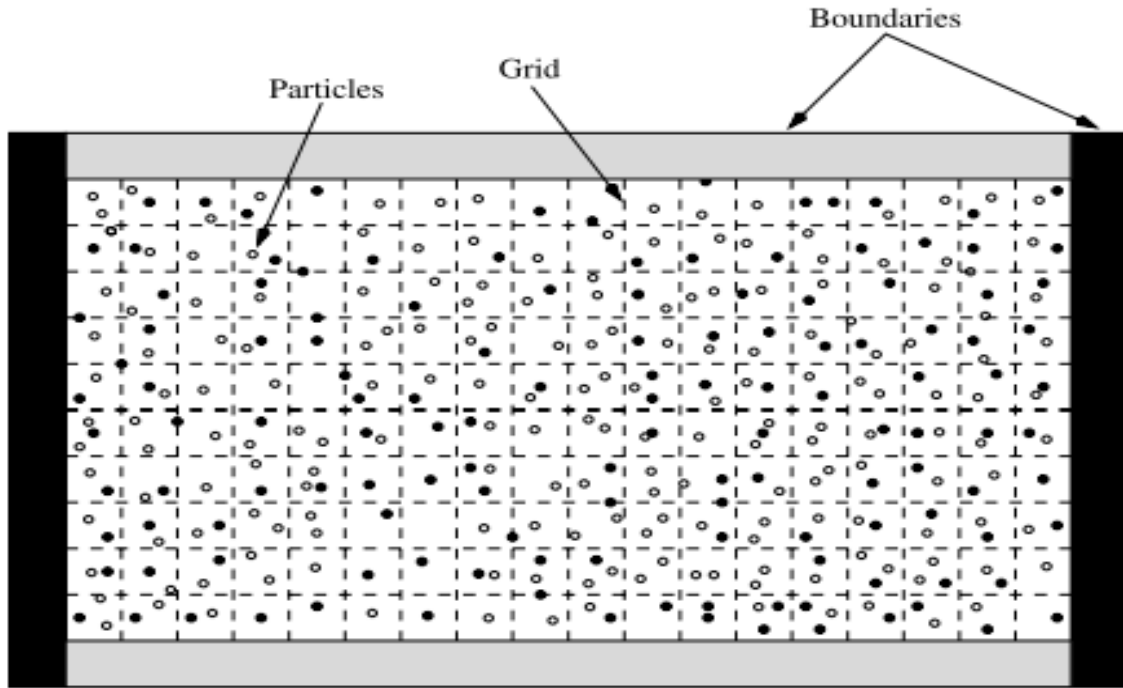


Figure 4: The view of the objects in a typical PIC code [2]

Markusic's thesis work was related to probe simulations in quiescent plasmas and ion beams, and the simulations were shown to be in agreement with both theory and experimental results [2]. Markusic's objective was to provide an accurate modeling of the physical processes using robust algorithms.

Based on electric probe theory, approximate solutions were available for two limiting cases during Markusic's work. One was for very thin sheaths, and the other one was for very thick sheaths. According to the thin sheath case, space charge limited current collection must be assumed. For this case, any particles crossing the sheath are

collected. Moreover, the Debye shielding length is much smaller than the probe radius ($\lambda_D < r_p$). According to that approximation, the resulting voltage–current characteristic would have a very sharp knee at point C in Figure 3.

For the very thick sheath case, orbital motion limited (OML) current collection must be assumed. OML current is the current collected by the probe when undisturbed particles are not capable of reaching the probe on the basis of kinetic energy and the intervening barrier of effective potentials. In this case, the influence of the sheath is neglected, and the particle orbits are computed using the space charge electric field of the probe. For this case, the Debye shielding length is much greater than the probe radius ($\lambda_D \gg r_p$). With this approximation, the resulting voltage–current characteristic does not have a very sharp knee (at point C in Figure 3); rather, a very subtle change from positive to negative curvature occurs.

After examining the two limiting cases, Markusic chose to model the interactions of a probe with plasma that has properties consistent with the very thick sheath case, since OML theory was the most relevant to the Langmuir probe theory and analysis created by I. Langmuir. Table 1 shows the plasma simulation conditions used in PROBE-PIC simulation to determine the V–I characteristic and assess its accuracy [2].

Table 1: Quiescent plasma simulation conditions in PROBEPIC [2]

Hydrogen ($m_i/m_e=1836$) and Light Ion ($m_i/m_e=100$) Conditions	
N [cm ⁻³]	1.0x10 ⁹
T [eV]	2.0
Plasma potential [V]	0.0
Probe aspect ratio	45.92

Table1 shows that the simulations were conducted both for hydrogen plasma that has an ion to electron mass ratio of 1836 and a light ion simulation with an artificial ion to electron mass ratio of 100 to verify that the ion collection was being properly modeled with fewer time steps.

1.2 PIC Monte Carlo Description

The Monte Carlo method is a technique that involves using random numbers and probability to solve problems. Through the Monte Carlo method, many approximate solutions for a variety of mathematical problems can be performed via statistical sampling experiments on a computer. The method applies to problems with

absolutely no probabilistic content, as well as to those with inherent probabilistic structure [9].

Modeling physical problems via Monte Carlo method allows the examination of very complex systems. Solving equations describing the interactions between two particles is a very simple process. However, solving the same equations for hundreds or thousands of particles is currently impossible. However, with Monte Carlo methods, a large system can be sampled in a number of random configurations, and that data can be used to describe the system as a whole.

Monte Carlo simulation is a method for iteratively evaluating a deterministic model using sets of random numbers as inputs. Therefore, the generation of pseudo-random numbers is an important element of any PIC Monte Carlo simulation. The quality of the random number plays a very important role in the statistical accuracy of the simulation. The ability to generate uncorrelated numbers with a uniform deviation is determined by the quality of a random number generator. Note that the fastest generators take a lot of space in the computer memory [7, 9].

For the PROBEPIC version developed in this thesis, standard random number generators were examined to determine their quality and usability. Based on this

evaluation, the “random_number.c” algorithm used in the original PROBEPIC was replaced. In the new version of PROBEPIC, “random_number.c” [2] was changed to “random_number.cpp” for the C++ version and “random_number.f90” for the FORTRAN version. These improved generators allow one to send an array of any size to be filled with random numbers of a uniform deviation. This allows one to store many values for later use rather than calculating them every time they are called in the PROBEPIC simulation. When all of the random numbers in the array are used, the array will be sent back to the algorithm random_number.cpp (or random_number.f90) with a new seed and initialization to generate another array of random numbers [2].

The particle simulation method has been studied for a long time and physicists working with the simulation of charged particles have developed the particle-in-cell (PIC) method based on the theory of charge assignment and force interpolation. This important theory has made it possible to obtain smooth distributions of charge density and current density needed for Maxwell’s equation of electromagnetic fields [7]. PIC and Monte Carlo plasma simulations have existed separately for several decades, with PIC applied primarily to hot fully ionized plasmas without collision, and Monte Carlo method applied to cool weakly ionized gases. However, these capabilities have now been

merged together and can be applied to both plasmas with collisions and to electron flow in semiconductors [7].

A PIC program contains four main modules which are used to move the particles through the phase space (velocity and spatial coordinates). These modules are the particle mover, the charge weighter, the field solver and the force weighter. At each step in time, each of these modules is executed sequentially (Figure 5) [7]. In a characteristic period of the plasma, there may be several thousands of time steps. The order of these modules is shown in Figure5 [7].

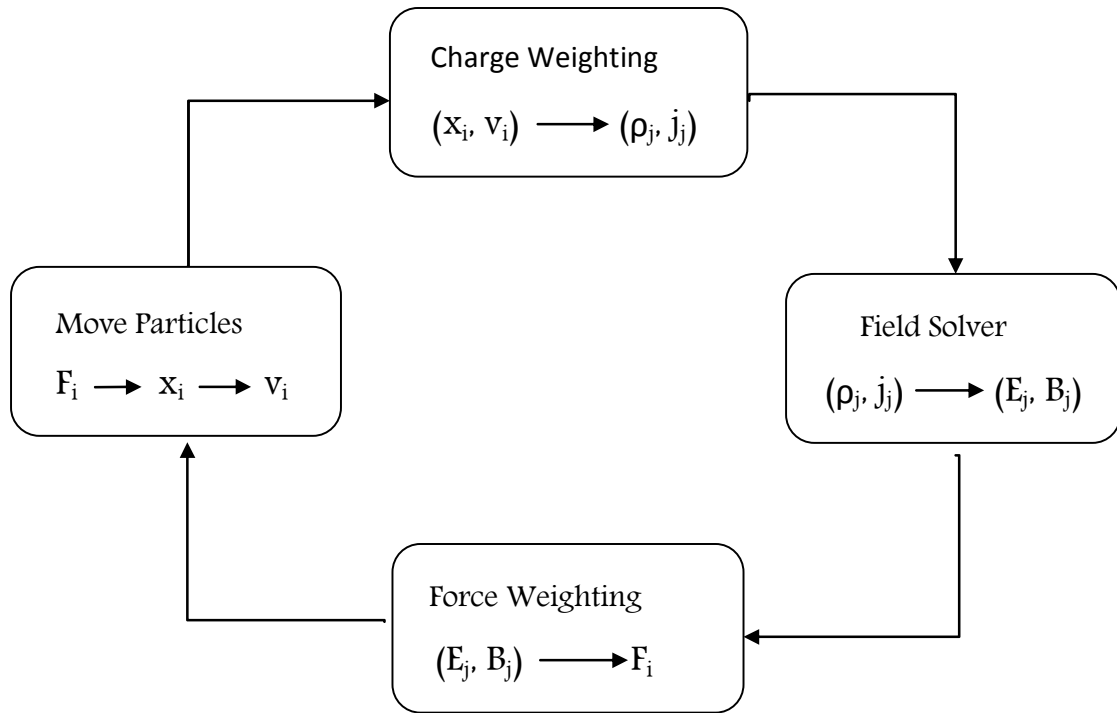


Figure 5: A typical cycle, a onetime step, in a particle simulation program (PIC) [7]

Every particle is moved; the charges and velocities of the particles are weighted to the grid cells to determine the charge density and current density on the grid points; the field solver then calculates the electromagnetic fields on the grid points; these field quantities are used to calculate the force on each particle; and then the cycle is repeated for the next time step.

1.3 Approach

The first objective of this thesis work was to translate PROBE_{PIC}, which was created in C code by T. Markusic in 1996 [2], into FORTRAN to improve its compatibility with other tools. Next, the effort refined the PROBE_{PIC} code to improve its accuracy, and upgrades were made to assumptions in the earlier code. To verify the capabilities of the new PROBE_{PIC} code, it was used to simulate a Langmuir probe reference case to create a voltage–current (V–I) plot that can be compared with theory for an ideal probe. An overview of the transformation of PROBE_{PIC} to FORTRAN is discussed in this section, with a more detailed discussion provided in Chapter II.

1.3.1 Transformed Probe PIC to FORTRAN and the Changes Made

T. Markusic created the PIC code, PROBE_{PIC}, for the simulation of a Langmuir probe to determine the plasma characteristics as described in his thesis [2]. His

programming philosophy was to develop a concise, intuitively accessible code without much redundancy. He used the C programming language since it was a popular language that allowed for longer variable name lengths than other programming languages of the day. C code also allows for a complicated data structure, giving the programmer control over dynamic memory allocation.

PROBEPIC was comprised of twenty-two individual subprograms, each of which performs an individual task. Markusic designed these functions to be independent of any external variables. Table2 shows these independent sub-programs with their functions listed next to them. To be able to run PROBEPIC with its sub-programs, Markusic was pushing the computational capabilities of the day. The computations in Markusic's thesis were primarily done with a C4 super computer at a high performance computer center [2]. However, advances in computer technology now allow PROBEPIC to be run on a desktop computer in roughly the same amount of time. In the present effort, the sub-programs given in Table2 were debugged and upgraded to C++ , and then they were converted into FORTRAN with documentation so that they can be run by users familiar with either programming language.

Table 2: PROBEPIE sub-program description [2]

Sub-program	Function
boundary.c	Handles particle/Boundary interactions for thermal particles
boundary_beam.c	Handles particle/ Boundary interactions for beam particles
charge_weight.c	The charge weighter
field_solver.c	The electric field solver
force_weight.c	The force weighter
graphics.c	Graphical diagnostic output
grid.c	Generates the computational grid
initialize.c	Distributes the initial loading of thermal super particles throughout the computational domain
initialize_beam.c	Distributes the initial loading of beam super particles throughout the computational domain
inject_beam.c	Fluxes beam particles across boundaries into the computational domain
inject_part.c	Fluxes thermal particles across boundaries into the computational domain
locate.c	Determines what grid-cell a particle is in
make_LU.c	Performs LU decomposition of difference matrix for subsequent use in field solver
make_velocity_table.c	Creates arrays of velocities with for fluxing into the computational domain
maxwell.c	Creates arrays of velocities with for fluxing into the Maxwellian distribution
mover.c	The mover
output_data.c	Generates PROBEPIE output file
parameter.c	Determines some necessary parameters used in other sub-programs from initial plasma conditions
probepic.c	PROBEPIE main program
probepic.h	Header file for PROBEPIE
random_number.c	Provides uniform random numbers for all PROBEPIE sub-programs
reset_grid.c	Resets charge density, electric field, etc. at grid-points to zero at beginning of each time-step

During the conversion process, some issues were encountered; these issues and their solutions are explained in detail in Chapter II. Appendix1 contains printed listings of these programs in both computer languages.

CHAPTER II

Description of POBEPIC

The charged particles making up plasma move under the influence of the electromagnetic fields that they generate, together with any external fields that may be present. Thus, in simulating the evolution of plasmas one seeks to model the mutual interaction of all particles and fields in a self-consistent manner. The Particle-in-Cell (PIC) method is a widely used technique for modeling plasma systems; the method (detailed in Chapter I) simulates the motion of plasma particles and calculates all macro-quantities such as number density, current density, etc., from the position and velocity of these particles [7].

T. Markusic, using the PIC code, desired to simulate plasma movement without using an experimental set up to predict the results as close as possible to the real conditions. He coined PROBEPIC as the name of the code in 1996. With the present thesis work, his code is updated into C++ and converted into FORTRAN. In this section, these processes are explained in detail. Additionally, a discussion is provided for the

modeling improvements that were incorporated to update assumptions in the earlier code.

2.1 Details on Probe PIC

Markusic created PROBE PIC, which was a 2D/3V PIC code to simulate the behavior of a Langmuir probe. 2D/3V means that the electric field has two components while the particles are free to move in three dimensions [7]. The code consists of twenty-two independent sub-programs, which are listed in Table 2.

The computational domain in PROBE PIC consists of a cylindrical region which contains a cylindrical Langmuir probe, as illustrated in Figure 6. The probe consists of a cylindrical conducting wire partially covered by an alumina insulator [2]. The computational domain surrounding the probe provides enough room for a plasma sheath to naturally form around the probe. Here, the sheath is generated by the interaction of the plasma with the boundary material.

In PROBE PIC it is assumed that the plasma is axisymmetric which allows the calculation for the fields only in the r and z direction on the half plane as shown in Figure 7. In PROBE PIC, the computational grid plane can be divided into two sections;

interior grid points and surface grid points. Here, the interior grid points lie within the plasma, probe conductor and insulator, while surface grid points lie at the edges of the computational domain and on the surface of the probe conductor and insulator [2].

Figure7 also contains the geometry of the probe (in meters) for the reference case used in this effort.

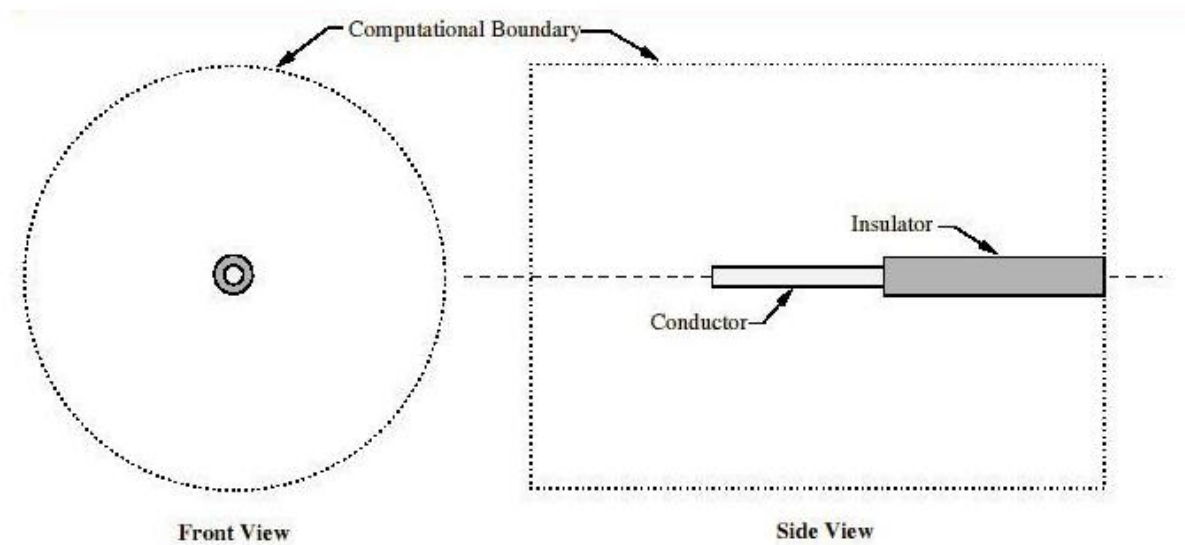


Figure 6: PROBE-PIC computational domain layout created [2]

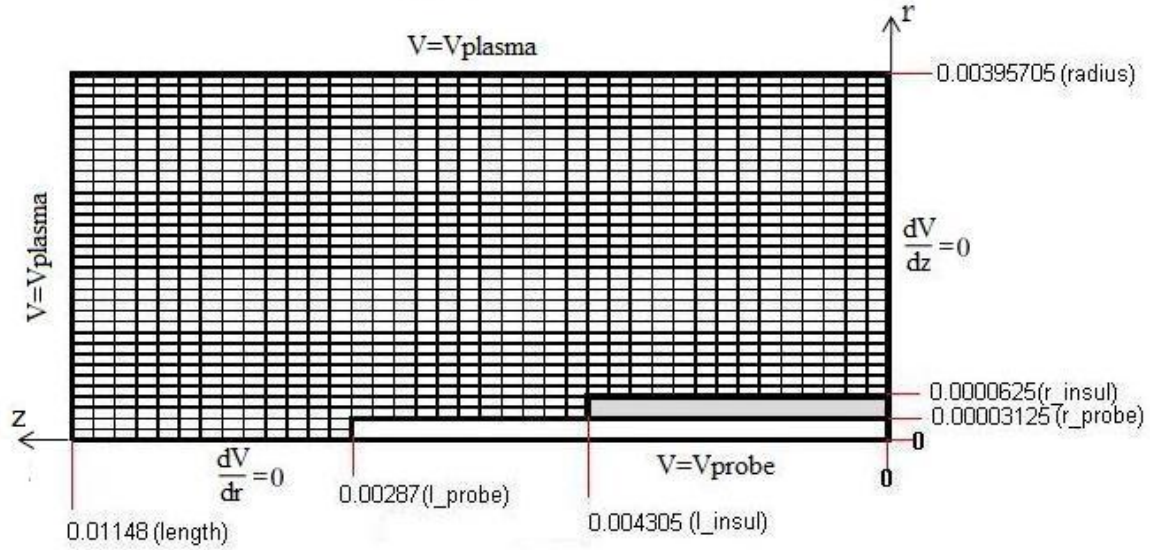


Figure 7. Field boundary conditions and probe geometry in PROBEPICT

2.2 Improvements/Updates

During the conversion of PROBEPICT, several complications occurred. These are described in this section, and their solutions are presented. It was thought that early discrepancies between the C++ (and FORTRAN) version and the original C version occurred because of the technology differences between when the code was written (in 1996) and the present or because the available electronic copy of the code was not the latest version of PROBEPICT published in Markusic's thesis. With several corrections to the available electronic version of the code, similar results to the original data presented in Markusic's thesis [2] were obtained. These corrections are detailed below.

One of the problems occurred with “grid.c”, which is the code used to generate the computational grid. Two of the computational domain parameters, “cell1_count” and “cell6_count”, were hardcoded in this program instead of being set as parameters in “parameter.c”, which is the code used to specify necessary parameters used in other sub-programs to establish the initial plasma conditions. Before the correction of “parameter.c”, the value of “cell-width”, as originally set in “parameter.c”, leads to the domain illustrated below in Figure8, where the grid did not fill the domain. However, the solution was applied by setting the cell_width equal to length divided by $ngp\_horize-1$, covering the entire domain by the grid as shown in Figure9.

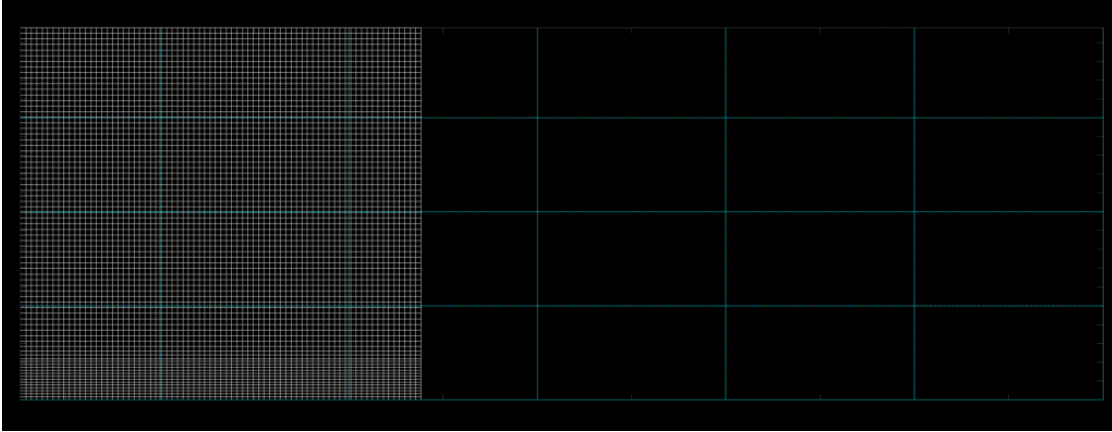


Figure 8: The computational grid in PROBEPIIC (before correction)

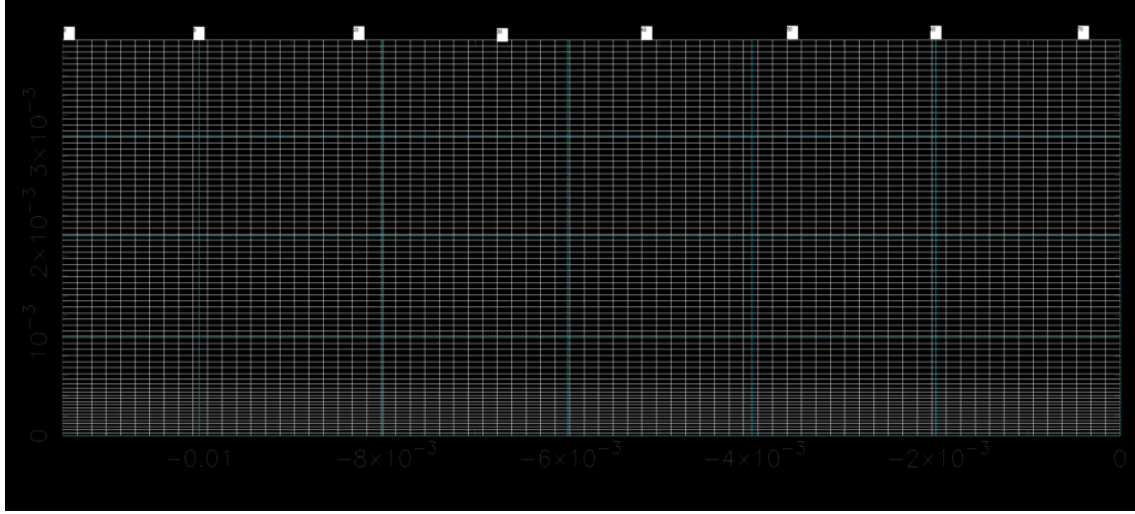


Figure 9: The computational grid after the correction of grid.c in PROBEPIC

The next issue was found in “inject_part.c”, which is a sub-program used to move (i.e. flux) the thermal particles across boundaries into the computational domain. The variable “count” had been constantly incremented and used as an index in order to browse the array of uniform random numbers. At a certain point, “count” exceeds the dimension of “random_num” regardless of how big this was set. The solution to this problem was to constantly check the value of “count” and reset it to zero when it reaches the dimension of “random_num”, the length of the random number array.

Another issue occurred in “charge_weight.c”, which is used to weight charges on particles to the grid points. During the process of a call to “locatex” and “locatey” (used to determine which cell a particle occupies), erroneous results for points near the upper

bound of the grid were obtained. The solution was achieved by decreasing the value of the second input parameter by one to `locatex(.,ngp2-1,...)` and `locatey(.,ngp1-1,...)`, properly defining the boundary.

After these corrections were implemented, the code started running and the individual sub-programs gave the anticipated results. Note that some sub-programs were run separately in test programs to observe if they work properly. For example, did the particle velocity distribution function used in the injector routine provide the correct distribution.

However, when the subroutines were combined together under a main-program and run to create the simulation, the result was not as it was expected to be. Although the results did not match those in Markusic's thesis, they followed an astonishing path; each point on the voltage-current characteristic was a factor of approximately $-5/3$ from the expected value (Markusic's result). At first the scale difference was believed to be the result of an erroneous data entry, e.g. a wrong geometrical dimension such as the probe length or width, the computational domain, etc., and/or maybe a wrong physical constant. To address this issue, the values of `ngp1`, `ngp2`, `numpart`, `NPTS` and `V_TABLE_RES`, etc. in "`lprobe60.h`", which is one of the header files in `PROBEPIC`

regarding probe geometry, were parametrically changed to observe the level of difference occurring in the program. Using this insight, some parameters were adjusted and the difference between the new simulation results and Markusic's results became a factor of -1 . The change from $-5/3$ to -1 is attributed to corrections in erroneous geometry input. It was tempting to assume that the -1 occurred because of a matter of sign convention in voltage-current characteristic, but the author was unsatisfied with this explanation.

At this point the original PROBEPIC code was reexamined, and some more issues were found in the logic of the program. The first of which was that the codes was not correctly detecting when the particle lies inside or outside the computational domain. This was resolved by making improvements in grid generation allowing higher density of cells in the vicinity of the probe.

However, in the original program, an important approximation had not been taken into consideration. A particle was considered to hit the probe when it falls exactly in a cell on the probe. In this case, probe current is generated, otherwise it is not. However, the high speed electrons and ions that have trajectories intersecting the probe and not residing in on a probe cell at the end of a time step were not being included in

the probe current. This was resolved by adding a model to extend the particle trajectory in a straight line (i.e. particle ray tracing) to determine if it intersects the cylinder represented by the probe during a time step, see Figure10. With the inclusion of the particles intersecting the probe, the code started giving the correct results without an arbitrary -1 factor.

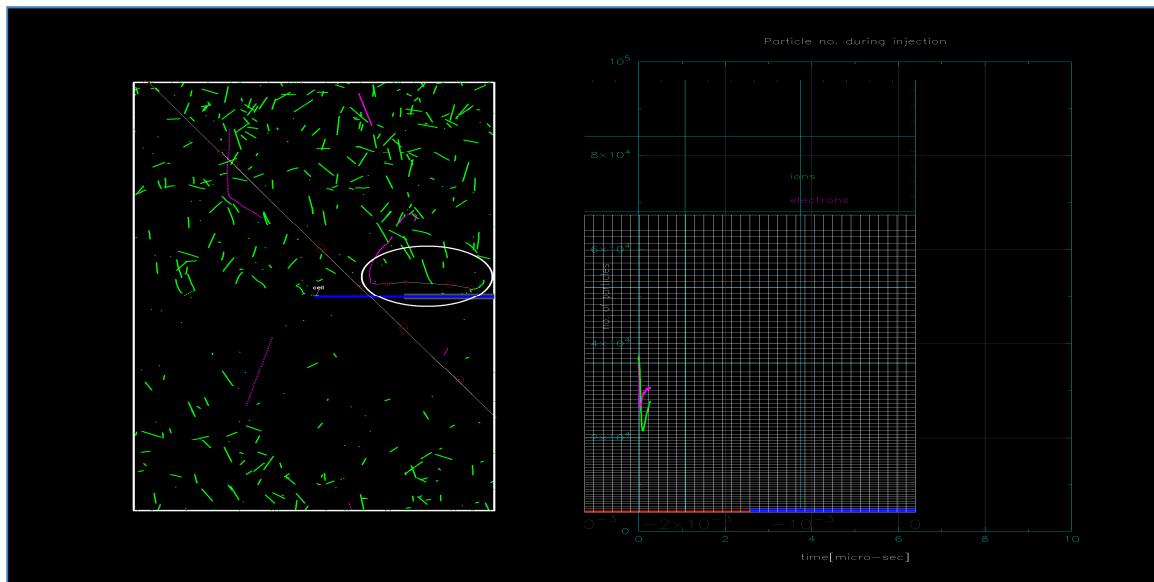


Figure 10: Particle ray tracing shown in animation window

While working on this particle collection issue, another logic issue involving the particle injection was identified. As mentioned previously, the PROBEPIC computational domain is a cylinder. The particle injection routine, “inject_part()”, was intended to place new particles on the domain boundaries from six directions, i.e. top/bottom, front/back, left side/right side, by using a virtual rectangular domain containing the cylinder; the particles were placed on the faces of this rectangular domain as shown in Figure 11.

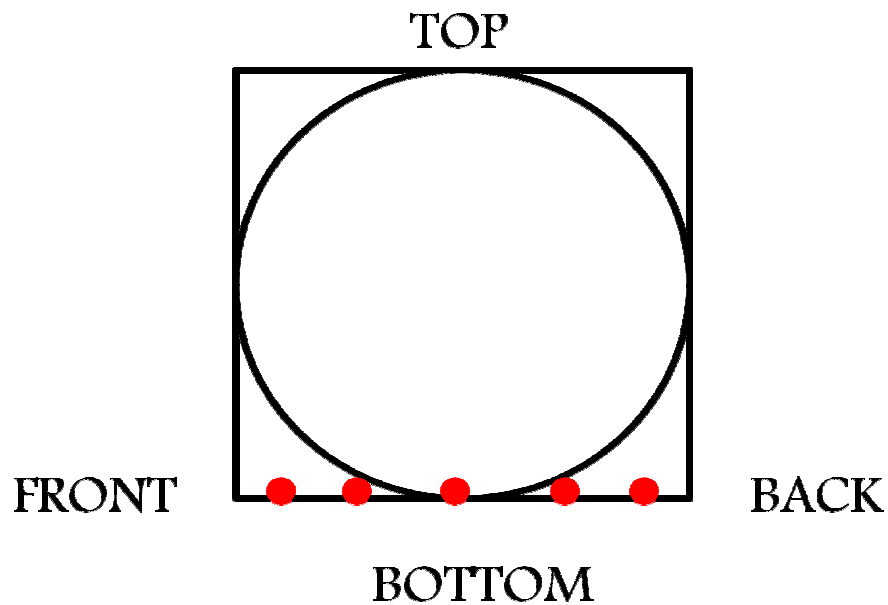


Figure 11. The rectangular domain in the original PROBEPIC

The particles will next travel for a “dt” interval of time, and then their new positions are checked to be inside/outside cylindrical domain. For a low speed particle, the “dt” interval may be too short for the particle to reach the cylindrical domain. For this case the particle is not actually injected into the domain, despite the initial intention. Only the particles fast enough to reach the domain will be injected into the cylinder, i.e. low velocities in the velocity distribution of the injected particles were underrepresented. Therefore, the “inject_part()” routine was modified to inject the new particles directly along the cylindrical domain boundary over an angle of 90 degrees (for computational efficiency) as illustrated in Figure12. Then the rest of the circle was covered by particles injected from three remaining sides of the virtual rectangular domain.

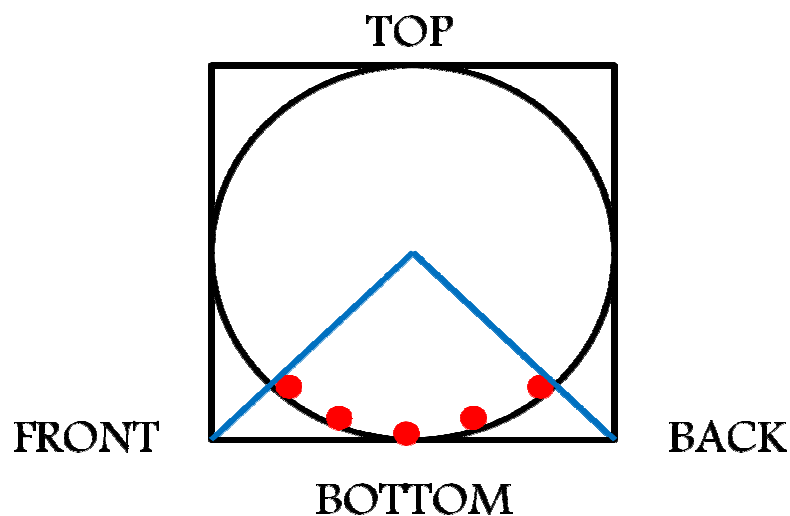


Figure 12: Cylindrical domain boundary with particle injection over 90 degrees

Additionally, a small issue was identified with the particle movement routine. In the original code, the particle movement under constant acceleration was implemented in the “mover()” routine using the equation below.

$$S(t) = S_0 + V_0 t + A_0 t^2$$

where A_0 is $[e * (\vec{E} + \vec{v} \times \vec{B})/m]$ in general and $[(e * \vec{E})/m]$ for the present case with no applied $\vec{B}$ - field. This equation was replaced with the following equation

$$S(t) = S_0 + V_0 t + A_0 \frac{t^2}{2}$$

The time step parameter, “dt”, was chosen empirically to satisfy the condition that a particle not travel a distance greater than a cell size. This requirement is fulfilled on average.

Another item that was adjusted was that almost all of the parameters that control the simulation process that were hard coded in the original routine “parameter()” were moved to an editable input file, “parameter.txt”, giving the user the option of performing multiple simulations without the requirement of recompiling the program each time it is run. A sample “parameter.txt” input file is listed in Table3. One can easily see that beside each numerical value contained in “parameter.txt”, an

explanation of the input is provided. The user can simply change the parameter(s), such as “wpdt” for time step, or vertical grid density with “number of cells per probe radius”, etc. in the input file, and then the user can launch the PROBEPIIC.

Table 3: A sample of editable parameter.txt

0.1	percent of particles displayed in animation, 0.0,...,100.0 ; 0.0 means NO animation
0.00003125	[m] probe radius
45.92	probe ASPECT RATIO: length/radius
3	number of cells per probe radius
40000	initial number of ions in domain
6	number of injected ions from BACK at each time step
6	number of injected ions from FRONT at each time step
12	number of injected ions from SIDE at each time step
40000	initial number of electrons in domain
5	number of injected electrons from BACK at each time step
5	number of injected electrons from FRONT at each time step
8	number of injected electrons from SIDE at each time step
2.0	T[eV] temperature
1.0e9	[cm ⁽⁻³⁾] density number
1.673e-27	m _{ion}
1.6e-19	q _{ion}
9.11e-31	m _{elec}
-1.6e-19	q _{elec}
0.019	[sec] wpdt
0.0	plasma potential

Additionally, the original program did not allow the probe to be divided into more than one cell in the radial direction. The routine “grid()” was modified to allow this to be increased, allowing the grid to be denser in the vicinity of the probe than the rest of the domain. A screen shot from the animation window is given in Figure13, which shows the electrons/ions trajectories, the locations where the particles are injected, and where they hit the probe.

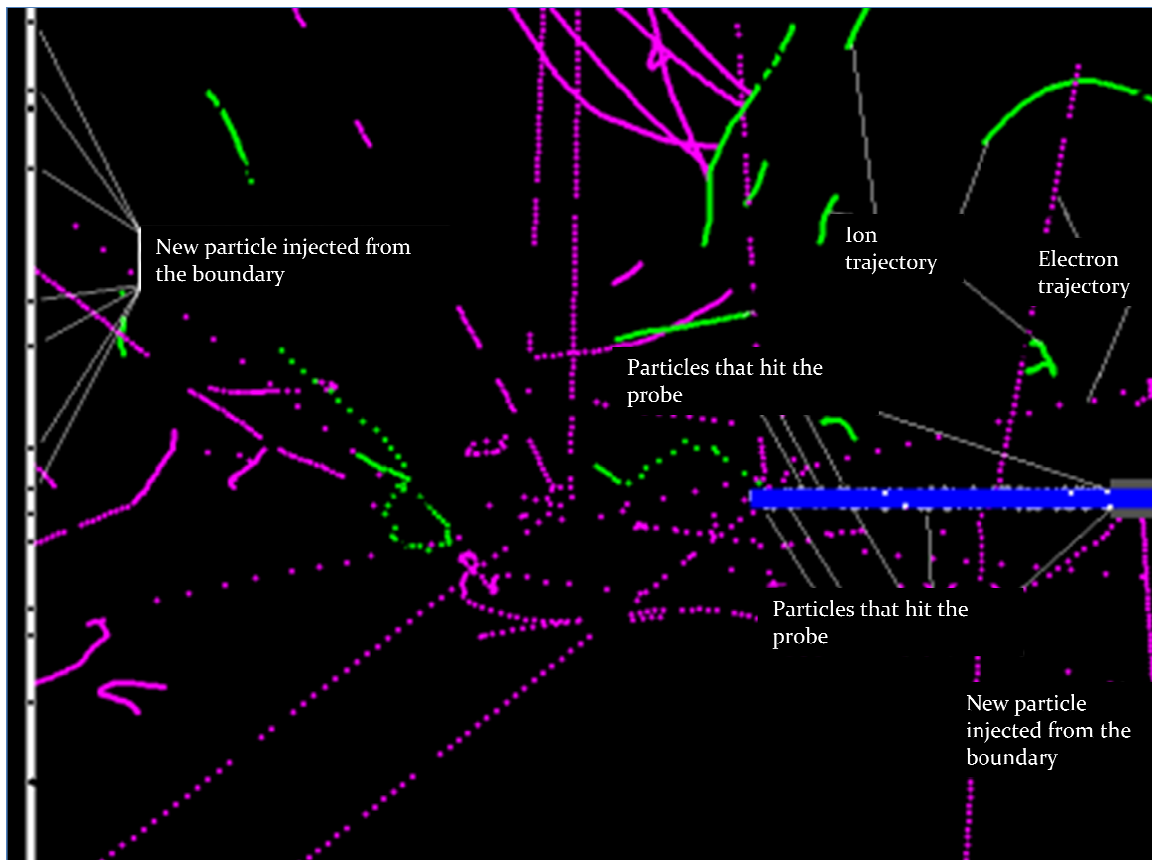


Figure 13: The locations where the particles injected and hit the probe

Note that, the horizontal (axial) grid density is hardcoded to 72 cells (73 points) in the file “lprobe60.cpp (lprobe60.F90)” by the variable called “ngp2”. If the horizontal grid density change is desired, then “ngp2” must be changed to a new value, and the program must then be re-compiled. This is the only situation when a re-compile is needed, and it can be done easily. If a change in the vertical grid is desired, then the 4th numerical value in the text file, “parameter.txt”, should be changed from 1 to 2, or 3, or 4, or more.

2.3 Comparison of New/Old Langmuir Simulation

The first version of the PIC code (PROBEPIC) has been updated and converted into C++ and FORTRAN. Both the C++ and FORTRAN versions of PROBEPIC are running properly together with their subroutines to provide results that are close to the theoretical model of the probe presented in Ref.2. The simulation results of the original and updated versions of PROBEPIC are similar to each other, providing verification that the code conversion was achieved properly. A comparison of the voltage-current characteristics is shown in Chapter III.

CHAPTER III

Results

In the original version of the PROBEPIE code, the simulations were conducted in a quiescent plasma in the orbital limited motion (OLM) limit. Similarly, in the updated version of the code, the plasma conditions were also chosen to correspond to the OLM domain used in Markusic's thesis [2]. Theoretical results are readily available for this case. By comparing the new results with the results from the original version and the theoretical results for an ideal probe, the effectiveness of the updated PROBEPIE can be evaluated.

In Markusic's thesis [2], simulations were conducted both for hydrogen plasma (ion to electron mass ratio of 1836) and a light ion simulation with an artificial ion to electron mass ratio of 100. The "Light ion" experiment was conducted to verify that the ion collection was being properly modeled with fewer time steps. However, an objective of this thesis was to determine if the code is error free and the conversion to FORTRAN was successful. For the updated version, due to time constraints only the Hydrogen plasma (1836 mass ratio) experiment was conducted to verify that the C^{++} and

FORTTRAN versions of the code run properly side by side. Figure14 illustrates the updated simulation results on a voltage–current characteristic plot. The dashed and solid curves are the theoretical results for an ideal Langmuir probe in the OLM limit. Simulation results reported by Markusic [2] are shown as open squares and circles. The results from the updated code are shown as solid red squares.

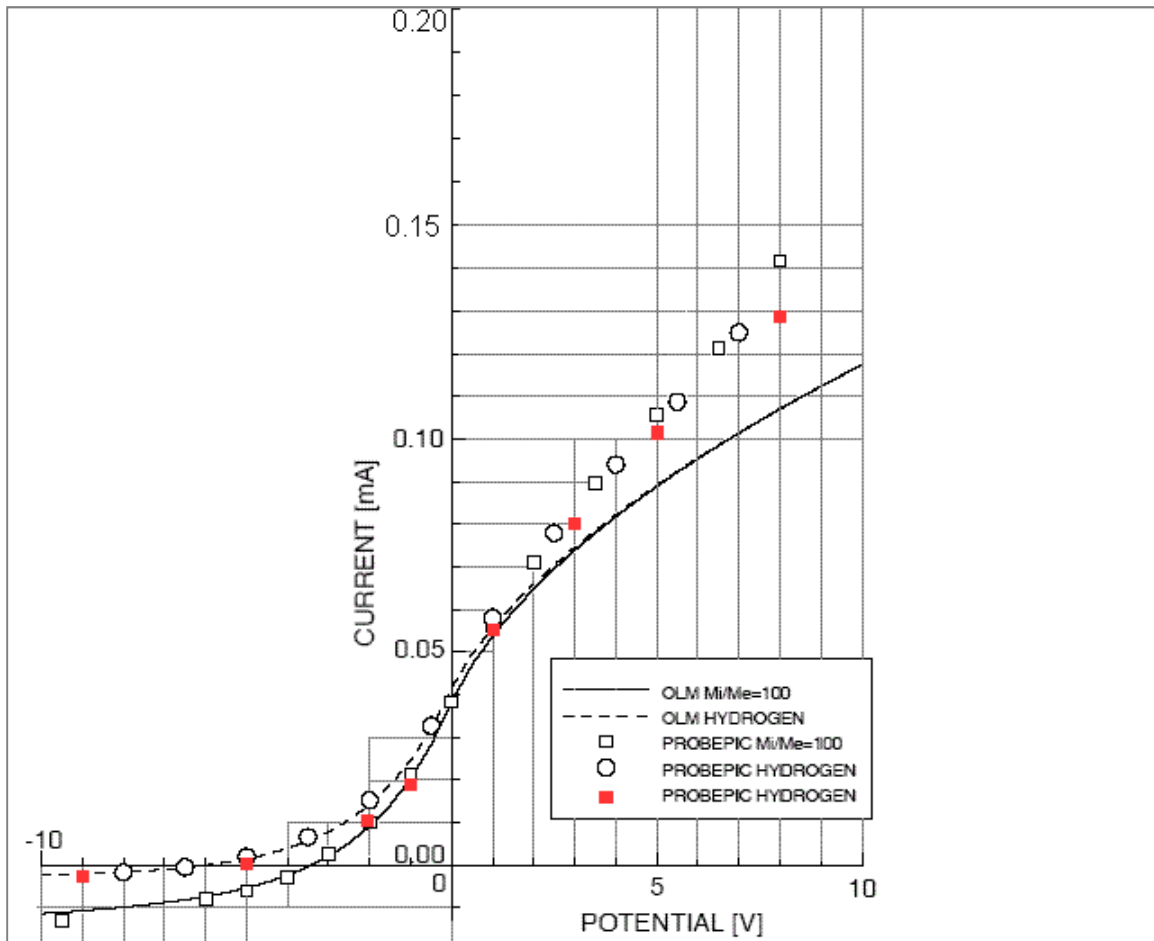


Figure 14: Voltage-Current (V-I) characteristics of original and updated PROBEPIC code

As can be seen in Figure14, if the probe potential is less than the plasma potential, estimated at about 1 Volt, then the PROBEPIE results show good agreement with the Langmuir theory. On the other hand, all of the PROBEPIE results diverge from the expected theoretical results in the electron saturation region (voltages above the plasma potential). The reason for this is not clear. The Langmuir theory always gives the maximum current that could be measured since the ideal probe is assumed, therefore, the PROBEPIE data should always appear below the upper orbital limited motion (OLM) threshold [2] curves shown.

According to conclusion drawn by Markusic [2], the disagreement between the Langmuir theory and PROBEPIE had a physical reason. The Langmuir theory was developed for an infinitely long cylindrical probe. However, for PROBEPIE and the laboratory experiments [3], the finite length probes were used. This suggests that the probe aspect ratio may produce a significant difference in the voltage-current characteristic. Also the sharp edges on finite length probe tips on conductors may produce an intense electric field in the vicinity of the tip that more efficiently draws in charged particles than realized in an “infinite” length probe. Note that since both probe types exhibit orbital limited motion (OML) current collection, the resulting voltage-

current characteristic does not have a sharp knee when the probe potential is equal to the plasma potential. Rather, there occurred a very smooth curve from positive to negative, as is expected (see Chapter I).

However, it should be noted that it appears that the enhancements made to PROBEPIC in the current effort did produce an improvement in the accuracy of the predictions. By comparing the new data points to the earlier data points the gap between the predictions and the Langmuir theory is reduced by about 30% in the electron saturation region. While some of this improvement appears to be from the enhanced particle collection and injection models, the difference between PROBEPIC predictions and Langmuir theory appears to be sensitive to grid density.

A lower grid density was examined to determine if the same results could be obtained in a shorter computational time. Grid density was lowered 50 percent as seen in Figure 15. When the grid density was reduced, it was observed that the predicted current changed significantly, diverging farther away from the theoretical points, as seen in Figure 16. The 50 percent reduction in grid density increased the current corresponding to a probe voltage of 5 V from about 0.10 mA to 0.12 mA. This increased divergence from theory (as seen in Figure 16) indicates a high sensitivity of the

predicted current on the grid density. Although the grid change lessened the computational time, this experiment showed that lowering the grid density had a big impact on the result. Because of this observed sensitivity, it is not known if the baseline grid density was sufficient to provide the best current prediction. To determine the optimum grid density, it will be necessary to perform additional work.

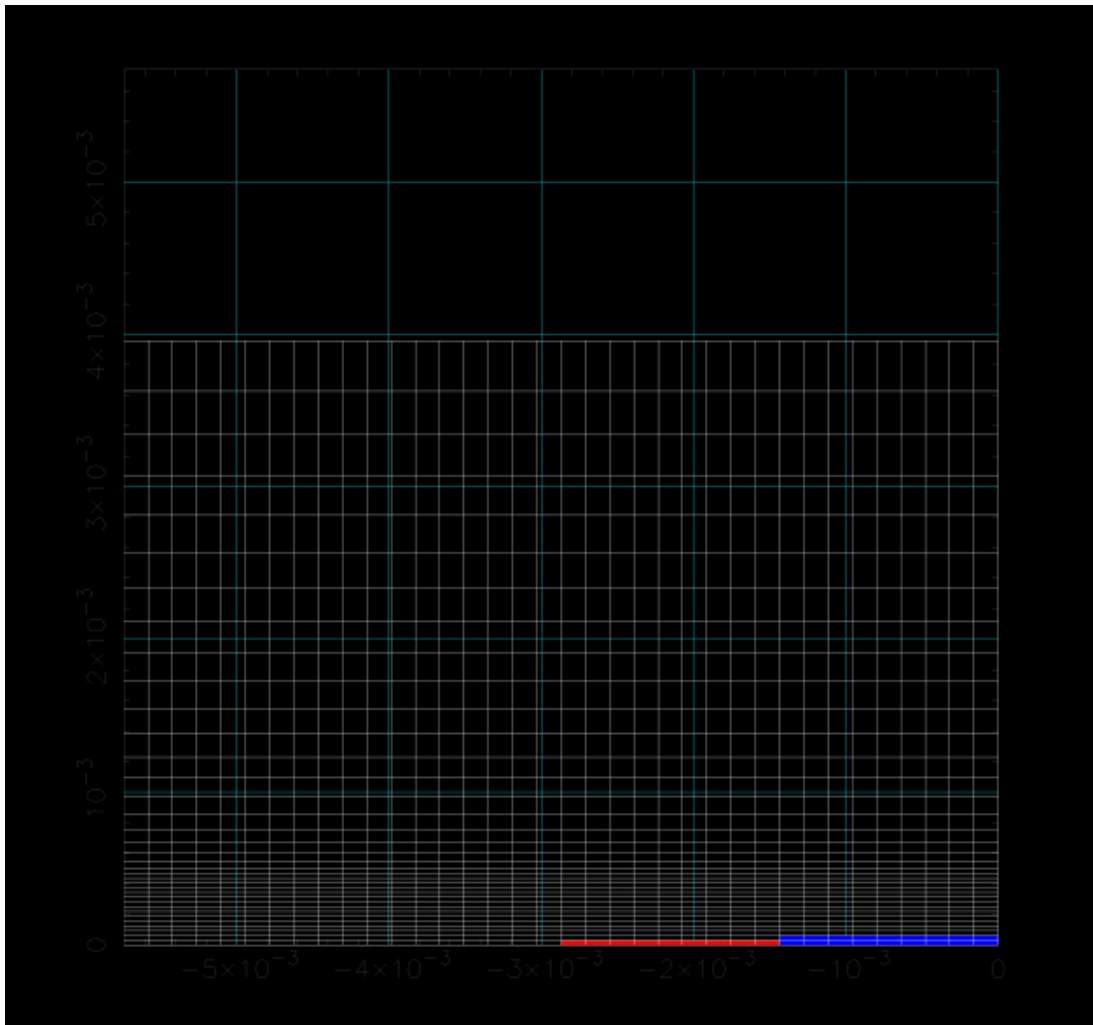


Figure 15: Probe grid with 50 percent lower density

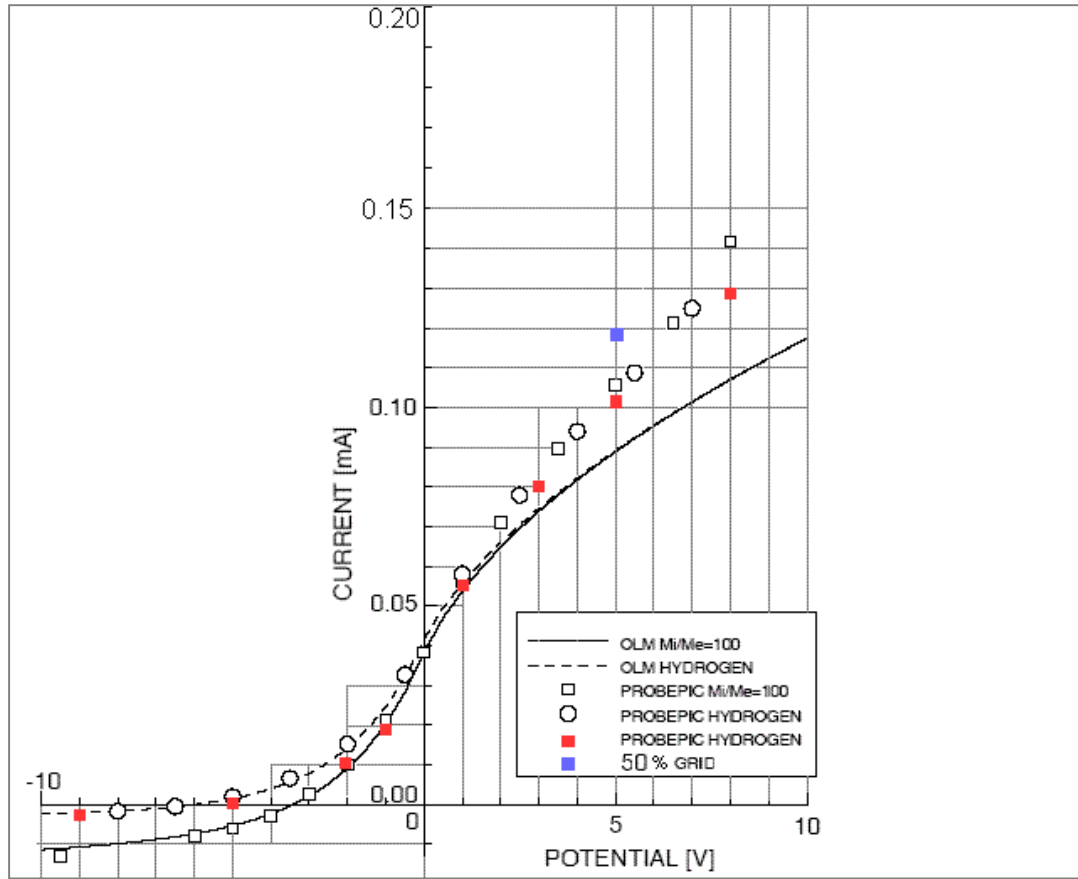


Figure 16: Effects of 50 percent reduced grid density

In future work, the number of injected particles should also be examined to determine the impact of this parameter on the current prediction, since the low grid density experiment also confirmed that the running time also depends on the number of particles in the domain, rather than just the number of mesh points. This sensitivity is the expected result of the need to move and test for position of all the particles at each time step.

CHAPTER IV

Conclusions and Recommendations

4.1 Conclusions

Markusic created the Particle-in-Cell code (PROBEPIC) to simulate the behavior of finite length Langmuir probes via particle simulation. In the present thesis work, Markusic's code has been converted from C into both C++ and FORTRAN versions. During conversion to C++ and FORTRAN, several errors in the available electronic version of Markusic's code were identified and corrected. After the code conversions, the simulation of hydrogen plasma with parameters that correspond to the OML case showed that new versions of PROBEPIC agree well with Markusic's original results. The correction of errors and code improvements in the updated version resulted in simulated currents closer to theory in the electron saturation region of the V-I characteristic, a region with the largest departure from theory. Also, the updated version is more user friendly with the addition of a user input file. Some changes were also made to allow the code to be more applicable to other probe geometries.

4.2 Recommendations

The updated version of PROBEPIIC can be further improved in future work by enhancing code speed with the algorithm refinements. In addition modifying the code to include a hybrid model, i.e. implementing a fluid model for the electrons eliminating the need for two different time scales [2] should be considered. Additional simulations, such as simulating plasma/probe interactions in the thin sheath limit, are also of interest. Conducting simulations with alternate Langmuir probe geometries such as increasing the probe aspect ratio and applying the code to other probe types, such as a Faraday probe, are recommended.

Finally, the lower grid density (50 percent less) was examined to see if the same results could be obtained in a shorter computational time. The results showed that lowering the grid density has a significant effect on the current prediction, causing it to diverge away from the theoretical results. In the future, different grid densities should be examined to find the optimum grid density to provide the best match between the predicted data points and the theoretical values. For this kind of experiments, higher capacity computer systems will be required.

BIBLIOGRAPHY

- [1] H. Mott-Smith, Jr and I. Langmuir. Phys.Rev.28, 27. 1926.
- [2] Markusic, Thomas. Particle Simulation of a Langmuir Probe in Quiescent and Flowing Plasmas. Masters Thesis. Tullahoma, 1996.
- [3] Keefer, D. and D.D. Semak. "Measurements of Radial and Axial Distributions of Ion Thruster Plasma Parameters Using a Langmuir Probe (unpublished).
- [4] Merlino, Robert L. "Understanding Langmuir Probe Current-Voltage Characteristics." 2007.
- [5] Hoag, J.B. and S.A. Korff. Electron and Nuclear Physics. New York: Van Nostrand, 1952.
- [6] Fei, Yee Yue. "The I-V Ramp Circuit for Langmuir Probe." Techpos . 2004.
- [7] Birdsall, C.K. and A.B. Langdon. Plasma Physics via Computer Simulation. New York: Adam Hilger, 1991.
- [8] Azooz, A. A. Four free parameter empirical parametrization of glow discharge Langmuir probe data. Review of Scientific Instruments. American Institute of Physics, October 3, 2008.
- [9] Wang, J. and J. Brophy. 3-D Monte-Carlo Particle-in-Cell Simulations of Ion Thruster Plasma Interactions. San Diego, CA: AIAA, 1995.

APPENDICES

Appendix-I

C++ Code for PROBEPIE New Version

boundary.cpp:

```
//#include "lprobe60.h"

/* This function sorts out which particles have crossed a
boundary, i.e.
the probe or extent of computational domain. Particles crossing the
probe boundary are counted as current */

void boundary (PARTICLEDEF particle[numpart], long int *num_part, FLOAT
q)
{
long int i, i2, numtemp;
FLOAT r_part, vzold, vyold;

numtemp=*num_part; i2=-1;

for (i=0; i < numtemp; i++){
i2=i2+1;
r_part=particle[i2].r;

/* OUT OF COMPUTATIONAL DOMAIN */

if (particle[i2].x > 0.0 || particle[i2].x < -length ||
particle[i2].y > radius || particle[i2].y < -radius ||
particle[i2].z > radius || particle[i2].z < -radius){

particle[i2].x=particle[*num_part-1].x;
particle[i2].y=particle[*num_part-1].y;
particle[i2].z=particle[*num_part-1].z;
particle[i2].r=particle[*num_part-1].r;
particle[i2].vx=particle[*num_part-1].vx;
particle[i2].vy=particle[*num_part-1].vy;
particle[i2].vz=particle[*num_part-1].vz;
particle[i2].q=particle[*num_part-1].q;
particle[i2].m=particle[*num_part-1].m;
i2=i2-1; *num_part = *num_part - 1;
}

/* CONTACT WITH PROBE */

else if (r_part <= r_probe && particle[i2].x >= -(l_probe+l_insul) &&
particle[i2].x <= -l_insul){
particle[i2].x=particle[*num_part-1].x;
```

```

particle[i2].y=particle[*num_part-1].y;
particle[i2].z=particle[*num_part-1].z;
particle[i2].r=particle[*num_part-1].r;
particle[i2].vx=particle[*num_part-1].vx;
particle[i2].vy=particle[*num_part-1].vy;
particle[i2].vz=particle[*num_part-1].vz;

particle[i2].m=particle[*num_part-1].m;

current=current + particle[i2].q;
if(q == q_elec ) curre=curre + particle[i2].q;
else if(q == q_ion) curri=curri + particle[i2].q;

particle[i2].q=particle[*num_part-1].q;
i2=i2-1;*num_part = *num_part - 1;
}

/* CONTACT WITH CYLINDRICAL SURFACE OF INSULATOR */

else if (r_part <= r_insul && particle[i2].x >-(l_insul-cell_width)){
    vzold=particle[i2].vz;
    vyold=particle[i2].vy;
    particle[i2].vz=(vzold*(-particle[i2].z*particle[i2].z +
particle[i2].y*particle[i2].y)
-
2.0*vyold*particle[i2].y*particle[i2].z)/(particle[i2].r*particle[i2].r
) ;
    particle[i2].vy=(vyold*(particle[i2].z*particle[i2].z -
particle[i2].y*particle[i2].y)
-
2.0*vzold*particle[i2].y*particle[i2].z)/(particle[i2].r*particle[i2].r
) ;
}

/* CONTACT WITH FLAT FACE OF INSULATOR */

else if (r_part <= r_insul && particle[i2].x >=-(l_insul)){
    particle[i2].vx=-particle[i2].vx;
}
}
}

```

boundary_v2.cpp:

```

//#include "lprobe60.h"

/* This function sorts out which particles have crossed a
boundary, i.e.
the probe or extent of computational domain. Particles crossing the

```

```

    probe boundary are counted as current    */

extern void gotoXY( short x, short y ); // in PROBEPIC.cpp
extern void SetColor(int ForgC); // in PROBEPIC.cpp

void boundary (PARTICLEDEF particle[numpart],long int *num_part,float
q)

{
    long int i,i2,numtemp;
    float r_part,vzold,vyold;
    float prev_avgc_dte, prev_avgc_dti ;

    //! the previous non-zero average values on dti/dte will remain on
    screen
    //! but dimmed, in case on the present timestep the probe will not be
    hitted
    prev_avgc_dte =prev_avgc_dti = 0.0;
    if (avgc_dte != 0.0) prev_avgc_dte = avgc_dte;
    if (avgc_dti != 0.0) prev_avgc_dti = avgc_dti;

    // average electron current on this dte time step
    avgc_dte = 0.0;

    numtemp=*num_part;i2=-1;

    for (i=0;i < numtemp;i++){
        i2=i2+1;
        r_part=particle[i2].r;

        /* OUT OF COMPUTATIONAL DOMAIN    */

        if (particle[i2].x >0.0 || particle[i2].x < -length ||    // OX
limits (horizontal)
            particle[i2].y >radius || particle[i2].y < -radius ||    // OY
limits (vertical)
            particle[i2].z >radius || particle[i2].z < -radius){ // OZ
limits (depth)
            // the particle 'i2' is removed: it is overwritten by the
            last one, *num_part-1

            particle[i2]=particle[*num_part-1]; // does this actually move
            the components or just copy the address ?

            // particle[i2].x=particle[*num_part-1].x;
            // particle[i2].y=particle[*num_part-1].y;
            // particle[i2].z=particle[*num_part-1].z;
            // particle[i2].r=particle[*num_part-1].r;
            // particle[i2].vx=particle[*num_part-1].vx;
            // particle[i2].vy=particle[*num_part-1].vy;
            // particle[i2].vz=particle[*num_part-1].vz;
            // particle[i2].q=particle[*num_part-1].q;

```



```

//  particle[i2].m=particle[*num_part-1].m;

    ///particle[i2].cellx=particle[*num_part-1].cellx;
    ///particle[i2].celly=particle[*num_part-1].celly;

    i2=i2-1;*num_part = *num_part - 1;
}

/* CONTACT WITH PROBE, but not it's tip(!)  */
else if (r_part <= r_probe
        &&
        particle[i2].x >=-(l_probe+l_insul)
        &&
        particle[i2].x <=-l_insul){

    cpgsci(1);          // white
    cpgpt(1,&particle[i2].x,&particle[i2].y,-1); // graphic symbol
#(-1)

    //! average current on a N*dti time interval
    current=current + particle[i2].q;

    if(q == q_elec ) {
        // average electron current on a N*dti time interval
        curre=curre + particle[i2].q;
        // average electron current on a dte time step
        avgc_dte = avgc_dte +particle[i2].q;

        hitElectrons++;
        // INSTANT charge released to probe is highlited ("bright
white")
        // only when is different from the previous one
        if (particle[i2].q == elec_charge) {
            SetColor(8); // 8=gray
            gotoXY(60-4,38+cursorY); printf("%11.3g",
particle[i2].q);
        }else {
            SetColor(15); // 15=bright white
            gotoXY(60-4,38+cursorY); printf("%11.3g",
particle[i2].q);
        }
        //elec_charge = particle[i2].q*1.0E6;      // latest
electron discharge
        elec_charge = particle[i2].q;      // latest electron
discharge [eV]
    } else {
        if(q == q_ion) {
            // average ion current on a N*dti time interval
            curri=curri + particle[i2].q;
            //average ion current on a dti time step
            avgc_dti = avgc_dti +particle[i2].q;

```

```

        hitIons++;
        // INSTANT charge released to probe is highlited
("bright white")
        // only when is different from the previous one
        if (particle[i2].q == ion_charge) {
            SetColor(8); // 8=gray
            gotoXY(60-4,39+cursorY); printf("%11.3g",
particle[i2].q);
        }else {
            SetColor(15); // 15=bright white
            gotoXY(60-4,39+cursorY); printf("%11.3g",
particle[i2].q);
        }
        //ion_charge = particle[i2].q*1.0E6; // latest ion
discharge
        ion_charge = particle[i2].q; // latest ion discharge
[ev]
    }
}

//particle[i2].q=particle[*num_part-1].q; // removed acc. to
LAB_REPORT

// the particle 'i2' is removed: it is overwritten by the
last one, *num_part-1
particle[i2]=particle[*num_part-1]; // does this actually move
the components or just copy the address ?
//
it actually moves, tested

//particle[i2].x=particle[*num_part-1].x;
// particle[i2].y=particle[*num_part-1].y;
// particle[i2].z=particle[*num_part-1].z;
// particle[i2].r=particle[*num_part-1].r;
// particle[i2].vx=particle[*num_part-1].vx;
// particle[i2].vy=particle[*num_part-1].vy;
// particle[i2].vz=particle[*num_part-1].vz;
// particle[i2].m=particle[*num_part-1].m;
// particle[i2].q=particle[*num_part-1].q; // moved here acc. to
LAB_REPORT
//particle[i2].cellx=particle[*num_part-1].cellx;
//particle[i2].celly=particle[*num_part-1].celly;

i2=i2-1;
*num_part = *num_part - 1; // remove the last particle as it
was duplicated at 'i2' rank

SetColor(15); // 15=bright white
gotoXY(20-2,38+cursorY); printf("%3d", hitElectrons);
//gotoXY(39-7,38+cursorY); printf("%12g", (curre/dti)*1.0E6);
gotoXY(20-2,39+cursorY); printf("%3d", hitIons);
//gotoXY(39-7,39+cursorY); printf("%12g", (curri/dti)*1.0E6);

```

```

    } // if

    //
    // REMOVED ACC. to LAB-REPORT
    //
    // /*      CONTACT WITH CYLINDRICAL SURFACE OF INSULATOR      */
    //
    //else if (r_part <= r_insul && particle[i2].x >= -(l_insul-
cell_width)){
    //    vzold=particle[i2].vz;
    //    vyold=particle[i2].vy;
    //    particle[i2].vz=(vzold*(-particle[i2].z*particle[i2].z +
particle[i2].y*particle[i2].y)
    //    //-
    2.0*vyold*particle[i2].y*particle[i2].z)/(particle[i2].r*particle[i2].r
    );
    //    particle[i2].vy=(vyold*(particle[i2].z*particle[i2].z -
particle[i2].y*particle[i2].y)
    //    //-
    2.0*vzold*particle[i2].y*particle[i2].z)/(particle[i2].r*particle[i2].r
    );
    //}

    // /*      CONTACT WITH FLAT FACE OF INSULATOR      */

    // else if (r_part <= r_insul && particle[i2].x >= -(l_insul)){
    //    particle[i2].vx=-particle[i2].vx; // (3) insulator particle
reflexion
    // }
}; // for

    // average current on this dti time step
    if (q == q_ion) { // a ion time step has finished

        gotoXY(20-2,39+cursorY); printf("%3d", hitIons);
        if (hitIons == 0) {
            // clean the displaying fields when are not ions
            gotoXY(38,39+cursorY); printf(" ");
            gotoXY(58,39+cursorY); printf(" ");
        }

        // average ion current on a dti time step
        gotoXY(39-7,39+cursorY);
        if (avgc_dti != 0.0) { // ions hitted the probe
            SetColor(15); // 15=bright white
            printf("%12.4g", (avgc_dti/dti)*1.0E6);
        } else { // ions did not hit the probe on this time step
            SetColor(8); // 8=gray
            if (prev_avgc_dti != 0.0) printf("%12.4g",
(prev_avgc_dti/dti)*1.0E6);
        }
    } // q == q_ion

```

```

        // average electron current on a dte time step
        gotoXY(39-7,38+cursorY);
        if (avgc_dte != 0.0) { // electrons hitted the probe
            SetColor(15); // 15=bright white
            printf("%12.4g", (avgc_dte/dte)*1.0E6);
        } else { // electrons did not hit the probe on this time step
            SetColor(8); // 8=gray
            if (prev_avgc_dte != 0.0) printf("%12.4g",
            (prev_avgc_dte/dte)*1.0E6);
        }

        //// dimm instant charges
        //      SetColor(8); // 8=gray
        // // electron
        // gotoXY(60-4,38+cursorY); printf("%11.3g", elec_charge);
        // //// ion
        // //gotoXY(60-4,39+cursorY); printf("%11.3g", ion_charge);

        // restore color for the rest of the routines
        SetColor(15); // 15=bright white

        // average current on this dti time step
        if (q == q_ion) avgc_dti = 0.0; // a ion time step has
finished

}; // boundary

```

charge_weight.cpp.

```

#include "lprobe60.h"

/* This function weights the charge to the array */

// found in locate.cpp
//extern void locatex(GRIDDEF xx[ngp1+1][ngp2+1], unsigned long n,
float x, unsigned long *jj);
//extern void locatey(GRIDDEF xx[ngp1+1][ngp2+1], unsigned long n,
float x, unsigned long *jj);
//extern void locatex(GRIDDEF xx[ngp1][ngp2], unsigned long n, float x,
unsigned long *jj);
extern void locatex(GRIDDEF** xx, unsigned long n, float x, unsigned
long *jj); /* 02Fed,2010 */
extern void locatex_2(GRIDDEF** xx, unsigned short n, float x, unsigned
short *jj, unsigned short jStart); /* 11Mar,2010 */
//extern void locatey(GRIDDEF xx[ngp1][ngp2], unsigned long n, float x,
unsigned long *jj);
extern void locatey(GRIDDEF** xx, unsigned long n, float y, unsigned
long *jj); /* 02Fed,2010 */
extern void locatey_2(GRIDDEF** xx, unsigned short n, float y, unsigned
short *jj, unsigned short jStart); /* 11Mar,2010 */

```

```

void qweight(GRIDDEF** array,PARTICLEDEF particle[numpart],long int
&number) /* 02Feb, 2010 */
//void qweight(GRIDDEF array[ngp1][ngp2],PARTICLEDEF
particle[numpart],long int number)
//void qweight(GRIDDEF array[ngp1+1][ngp2+1],PARTICLEDEF
particle[numpart],long int number)
{
//unsigned long positionx,positiony;
unsigned short positionx,positiony;
long int count1;
float dr,dr1,dr2,dz,dz1,dz2;

unsigned long positionx2;

for(count1=0;count1 < number;count1++){

    if (particle[count1].r <= radius){

        //locatex(array,ngp2-1,particle[count1].x,&positionx);

        locatex_2(array,ngp2,particle[count1].x,&positionx,particle[count1].cel
lx);
        //locatex_2(array,ngp2-
1,particle[count1].x,&positionx,particle[count1].cellx);

        /*** TEST for new locatex_2: does it provides the same positionx
as */
        //if ((particle[count1].cellx != 0) && (particle[count1].cellx
!= positionx)) {
            //locatex_2(array,ngp2-
1,particle[count1].x,&positionx2,particle[count1].cellx);
            //if (positionx != positionx2)
            //    positionx2=positionx; // instead of printing a
message
        //}

        //locatey_2(array,ngp1,fabs(particle[count1].y),&positiony,particle[cou
nt1].celly); // 13Mar, 2010

        locatey_2(array,ngp1,particle[count1].r,&positiony,particle[count1].cel
ly); // original

        //locatey_2(array,ngp1-
1,particle[count1].y,&positiony,particle[count1].celly);

        //locatey(array,ngp1-1,particle[count1].r,&positiony); // page 81
Markusic
        //locatey(array,ngp1-1,particle[count1].y,&positiony); /* 06Mar,
2010*/

        if ((positionx+1) < ngp2 && (positiony+1) < ngp1) { // particle
in a domain cell ?

```

```

/*    SAVE CELL LOCATION FOR FORCE WEIGHT    */
particle[count1].cellx=positionx;
particle[count1].celly=positiony;

dr=array[positiony+1][positionx].y*array[positiony+1][positionx].y
- array[positiony][positionx].y
  *array[positiony][positionx].y;
dr1=particle[count1].r - array[positiony][positionx].y;
dr2=array[positiony+1][positionx].y - particle[count1].r;

dz=array[positiony][positionx+1].x -
array[positiony][positionx].x;
dz1=particle[count1].x - array[positiony][positionx].x;
dz2=array[positiony][positionx+1].x - particle[count1].x;

array[positiony][positionx].q_dens=array[positiony][positionx].q_dens +
particle[count1].q *
      dr2 * dz2
      * (2.0 * array[positiony+1][positionx].y + 3.0 *
array[positiony][positionx].y
      - particle[count1].r) / (2.0 * dr * dz);

array[positiony][positionx+1].q_dens=array[positiony][positionx+1].q_de
ns + particle[count1].q *
      dr2 * dz1
      * (2.0 * array[positiony+1][positionx+1].y + 3.0
* array[positiony][positionx].y
      - particle[count1].r) / (2.0 * dr * dz);

array[positiony+1][positionx].q_dens=array[positiony+1][positionx].q_de
ns + particle[count1].q *
      dr1 * dz2
      * (3.0 * array[positiony+1][positionx].y + 2.0 *
array[positiony][positionx].y
      - particle[count1].r) / (2.0 * dr * dz);

array[positiony+1][positionx+1].q_dens=array[positiony+1][positionx+1].
q_dens + particle[count1].q *
      dr1 * dz1
      * (3.0 * array[positiony+1][positionx+1].y + 2.0
* array[positiony][positionx+1].y
      - particle[count1].r) / (2.0 * dr * dz);
} // if particle in domain
else // particle not in domain limits: remove it, as in
'boundary.cpp'
{
    ///printf("\ncharge_weight.cpp: cellx=%d celly=%d
\n",positionx,positiony);
    //particle[count1].x=particle[number-1].x;
    //particle[count1].y=particle[number-1].y;
    //particle[count1].z=particle[number-1].z;
    //particle[count1].r=particle[number-1].r;

```

```

        //particle[count1].vx=particle[number-1].vx;
        //particle[count1].vy=particle[number-1].vy;
        //particle[count1].vz=particle[number-1].vz;
        //particle[count1].q=particle[number-1].q;
        //particle[count1].m=particle[number-1].m;
        //particle[count1].cellx=particle[number-1].cellx;
        //particle[count1].celly=particle[number-1].celly;

        particle[count1]=particle[number-1]; // does this actually
move the components or just copy the address ?
//
it actually moves, tested

        count1=count1-1;
        number = number - 1;
    }
    //{};
}
}; // for
}; // qweight

```

cpgplot.h.

```

#ifndef cpgplot_h
#define cpgplot_h

#ifdef __cplusplus
extern "C" {
#endif

typedef int Logical;

void cpgarro(float x1, float y1, float x2, float y2);
void cpgask(Logical flag);
void cpgaxis(const char *opt, float x1, float y1, float x2, float y2,
float v1, float v2, float step, int nsub, float dmajl, float dmajr,
float fmin, float disp, float orient);
int cpgband(int mode, int posn, float xref, float yref, float *x, float
*y, char *ch_scalar);
void cpgbbuf(void);
int cpgbeg(int unit, const char *file, int nxsub, int nysub);
void cpgbin(int nbin, const float *x, const float *data, Logical
center);
void cpgbox(const char *xopt, float xtick, int nxsub, const char *yopt,
float ytick, int nysub);
void cpgcirc(float xcent, float ycent, float radius);
void cpgclos(void);
void cpgconb(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, const float *c, int nc, const float *tr, float blank);
void cpgconf(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, float c1, float c2, const float *tr);

```

```

void cpgconl(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, float c, const float *tr, const char *label, int intval,
int minint);
void cpgcons(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, const float *c, int nc, const float *tr);
void cpgcont(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, const float *c, int nc, const float *tr);
void cpgctab(const float *l, const float *r, const float *g, const
float *b, int nc, float contra, float bright);
int cpgcurs(float *x, float *y, char *ch_scalar);
void cpgdraw(float x, float y);
void cpgebuf(void);
void cpgend(void);
void cpgenv(float xmin, float xmax, float ymin, float ymax, int just,
int axis);
void cpgeras(void);
void cpgerr1(int dir, float x, float y, float e, float t);
void cpgerrb(int dir, int n, const float *x, const float *y, const
float *e, float t);
void cpgerrx(int n, const float *x1, const float *x2, const float *y,
float t);
void cpgerry(int n, const float *x, const float *y1, const float *y2,
float t);
void cpgetxt(void);
void cpggray(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, float fg, float bg, const float *tr);
void cpghi2d(const float *data, int nxv, int nyv, int ix1, int ix2, int
iy1, int iy2, const float *x, int ioff, float bias, Logical center,
float *ylims);
void cpghist(int n, const float *data, float datmin, float datmax, int
nbin, int pgflag);
void cpgiden(void);
void cpgimag(const float *a, int idim, int jdim, int i1, int i2, int
j1, int j2, float a1, float a2, const float *tr);
void cpglab(const char *xlbl, const char *ylbl, const char *toplbl);
void cpglcur(int maxpt, int *npt, float *x, float *y);
void cpgldev(void);
void cpglen(int units, const char *string, float *xl, float *yl);
void cpgline(int n, const float *xpts, const float *ypts);
void cpgmove(float x, float y);
void cpgmtxt(const char *side, float disp, float coord, float fjust,
const char *text);
void cpgncur(int maxpt, int *npt, float *x, float *y, int symbol);
void cpgnumb(int mm, int pp, int form, char *string, int
*string_length);
void cpgolin(int maxpt, int *npt, float *x, float *y, int symbol);
int cpgopen(const char *device);
void cpgpage(void);
void cpgpanl(int nxc, int nyc);
void cpgpap(float width, float aspect);
void cpgpixl(const int *ia, int idim, int jdim, int i1, int i2, int j1,
int j2, float x1, float x2, float y1, float y2);
void cpgpnts(int n, const float *x, const float *y, const int *symbol,
int ns);

```



```

void cpgpoly(int n, const float *xpts, const float *ypts);
void cpgpt(int n, const float *xpts, const float *ypts, int symbol);
void cpgpt1(float xpt, float ypt, int symbol);
void cpgptxt(float x, float y, float angle, float fjust, const char
*text);
void cpgqah(int *fs, float *angle, float *barb);
void cpgqcf(int *font);
void cpgqch(float *size);
void cpgqci(int *ci);
void cpgqcir(int *icilo, int *icihi);
void cpgqclp(int *state);
void cpgqcol(int *ci1, int *ci2);
void cpgqcr(int ci, float *cr, float *cg, float *cb);
void cpgqcs(int units, float *xch, float *ych);
void cpgqdt(int n, char *type, int *type_length, char *descr, int
*descr_length, int *inter);
void cpgqfs(int *fs);
void cpgqhs(float *angle, float *sepn, float *phase);
void cpgqid(int *id);
void cpgqinf(const char *item, char *value, int *value_length);
void cpgqitf(int *itf);
void cpgqls(int *ls);
void cpgqlw(int *lw);
void cpgqndt(int *n);
void cpgqpos(float *x, float *y);
void cpgqtbgi(int *tbci);
void cpgqtxt(float x, float y, float angle, float fjust, const char
*text, float *xbox, float *ybox);
void cpgqvp(int units, float *x1, float *x2, float *y1, float *y2);
void cpgqvsz(int units, float *x1, float *x2, float *y1, float *y2);
void cpgqwin(float *x1, float *x2, float *y1, float *y2);
void cpgrect(float x1, float x2, float y1, float y2);
float cpgrnd(float x, int *nsub);
void cpgrng(float x1, float x2, float *xlo, float *xhi);
void cpgsah(int fs, float angle, float barb);
void cpgsave(void);
void cpgunsa(void);
void cpgscf(int font);
void cpgsch(float size);
void cpgsci(int ci);
void cpgscir(int icilo, int icihi);
void cpgsclp(int state);
void cpgscr(int ci, float cr, float cg, float cb);
void cpgscl(float dx, float dy);
void cpgscrn(int ci, const char *name, int *ier);
void cpgsfs(int fs);
void cpgshls(int ci, float ch, float cl, float cs);
void cpgshs(float angle, float sepn, float phase);
void cpgsitf(int itf);
void cpgslct(int id);
void cpgsls(int ls);
void cpgslw(int lw);
void cpgstbg(int tbci);
void cpgsubp(int nxsub, int nysub);

```

```

void cpgsvp(float xleft, float xright, float ybot, float ytop);
void cpgswin(float x1, float x2, float y1, float y2);
void cpgtbox(const char *xopt, float xtick, int nxsub, const char
*yopt, float ytick, int nysub);
void cpgtext(float x, float y, const char *text);
void cpgtick(float x1, float y1, float x2, float y2, float v, float
tikl, float tikr, float disp, float orient, const char *str);
void cpgupdt(void);
void cpgvect(const float *a, const float *b, int idim, int jdim, int
i1, int i2, int j1, int j2, float c, int nc, const float *tr, float
blank);
void cpgvsiz(float xleft, float xright, float ybot, float ytop);
void cpgvstd(void);
void cpgwedg(const char *side, float disp, float width, float fg, float
bg, const char *label);
void cpgwnad(float x1, float x2, float y1, float y2);

#ifdef __cplusplus
}
#endif

#endif

```

distrib.cpp.

```

// distrib.cpp
// Put particles in bins

//typedef struct {
//    float r,v;
// }GRIDDEF_T;

void locate(GRIDDEF_T *xx, unsigned long n, float x, unsigned long *jj)
{
    unsigned long ju,jm,jl;
    int ascnd;

    jl=0;
    ju=n+1; // n intervals, (n+1) division points
    ascnd=(xx[n-1].r > xx[1].r);
    while (ju-jl > 1) {
        jm=(ju+jl) >> 1;
        if (x > xx[jm].r == ascnd)
            jl=jm;
        else
            ju=jm;
    }
    *jj=jl;
}; // locate

```

```

        void distrib(GRIDDEF_T *bins,unsigned long n,PARTICLEDEF
*particle,unsigned long num_part)
        {
            //TYPE (GRIDDEF_T) :: bins(0:n-1)
            //TYPE (PARTICLEDEF ) particle(0:numpart-1)

            unsigned long i,pos;
            float v;

            // Empty bins
            for(i=0;i<n;i++) bins[i].v = 0.0;

            // PUT DATA IN BINS
            for(i=0;i<num_part;i++){
                v= sqrt(pow(particle[i].vx,2) +pow(particle[i].vy,2)
+pow(particle[i].vz,2) );
                locate(bins, n, v, &pos);
                bins[pos].v ++;
            };
        }; // DISTRIB

```

erfc.cpp.

```

/* specfunc/erfc.c
*
* Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003 Gerard
Jungman
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
(at
* your option) any later version.
*
* This program is distributed in the hope that it will be useful, but
* WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
* General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-
1301, USA.
*/

/* Author: J. Theiler (modifications by G. Jungman) */

/*
* See Hart et al, Computer Approximations, John Wiley and Sons, New
York (1968)
* (This applies only to the erfc8 stuff, which is the part

```

```

* of the original code that survives. I have replaced much of
* the other stuff with Chebyshev fits. These are simpler and
* more precise than the original approximations. [GJ])
*/
#include <config.h>
#include "../specfunc/gsl_math.h"
#include "../specfunc/gsl_errno.h"
#include "../specfunc/gsl_sf_exp.h"
#include "../specfunc/gsl_sf_erf.h"

#include "check.h"

#include "../specfunc/chebyshev.h"
#include "../specfunc/cheb_eval.c"

#define LogRootPi_ 0.57236494292470008706
#define M_SQRTPI 1.77245385090551602729816748334 /* sqrt(pi) */
#define M_PI 3.14159265358979323846264338328 /* pi */
#define GSL_ROOT6_DBL_EPSILON 2.4607833005759251e-03
#define M_SQRT2 1.41421356237309504880168872421 /* sqrt(2) */
#define GSL_DBL_MIN 2.2250738585072014e-308
#define GSL_EUNDRFLW 15 /* underflow */
#define CHECK_UNDERFLOW(r) if (fabs((r)->val) < GSL_DBL_MIN)
GSL_ERROR("underflow", GSL_EUNDRFLW);
#define GSL_ERROR_SELECT_2(a,b) ((a) != GSL_SUCCESS ? (a) : ((b)
!= GSL_SUCCESS ? (b) : GSL_SUCCESS))
#define GSL_LOG_DBL_MAX 7.0978271289338397e+02
#define GSL_LOG_DBL_MIN -7.0839641853226408e+02
#define GSL_EOVRFLW 16 /* overflow */

typedef void gsl_error_handler_t (const char * reason, const char *
file,
                                int line, int gsl_errno);

void gsl_stream_printf (const char *label, const char *file,
                        int line, const char *reason)
{printf("Label: %s, File: %s, Line: %d, Reason: %s",
label,file,line,reason);};

gsl_error_handler_t * gsl_error_handler = NULL;

void gsl_error (const char * reason, const char * file, int line, int
gsl_errno)
{
    if (gsl_error_handler)
    {
        (*gsl_error_handler) (reason, file, line, gsl_errno);
        return ;
    }

    gsl_stream_printf ("ERROR", file, line, reason);

    fflush (stdout);
    fprintf (stderr, "Default GSL error handler invoked.\n");
}

```

```

    fflush (stderr);

    abort ();
}

#define GSL_ERROR(reason, gsl_errno) \
    do { \
        gsl_error (reason, "erfc.cpp", 492, gsl_errno) ; \
        return gsl_errno ; \
    } while (0)

#define GSL_ERROR_VAL(reason, gsl_errno, value) \
    do { \
        gsl_error (reason, "gsl_sf_erf", 538, gsl_errno) ; \
        return value ; \
    } while (0)

double gsl_fdiv (const double x, const double y)
{
    return x / y;
}
double gsl_posinf (void)
{
    return gsl_fdiv (+1.0, 0.0);
}
# define GSL_POSINF (gsl_posinf())
#define OVERFLOW_ERROR(result) do { (result)->val = GSL_POSINF;
(result)->err = GSL_POSINF; GSL_ERROR ("overflow", GSL_EOVRFLW); }
while(0)
#define UNDERFLOW_ERROR(result) do { (result)->val = 0.0; (result)->err
= GSL_DBL_MIN; GSL_ERROR ("underflow", GSL_EUNDRFLW); } while(0)

// #define OVERFLOW_ERROR_2(r1,r2) do { (r1)->val = GSL_POSINF; (r1)-
>err = GSL_POSINF; (r2)->val = GSL_POSINF ; (r2)->err=GSL_POSINF;
GSL_ERROR ("overflow", GSL_EOVRFLW); } while(0)
// #define UNDERFLOW_ERROR_2(r1,r2) do { (r1)->val = 0.0; (r1)->err =
GSL_DBL_MIN; (r2)->val = 0.0 ; (r2)->err = GSL_DBL_MIN; GSL_ERROR
("underflow", GSL_EUNDRFLW); } while(0)

// from exp.c
int gsl_sf_exp_e(const double x, gsl_sf_result * result)
{
    if(x > GSL_LOG_DBL_MAX) {
        OVERFLOW_ERROR(result);
    }
    else if(x < GSL_LOG_DBL_MIN) {
        //UNDERFLOW_ERROR(result);
        do {
            (result)->val = 0.0;
            result->err = GSL_DBL_MIN;
            GSL_ERROR ("underflow", GSL_EUNDRFLW);
        } while(0);
    }
}

```

```

    else {
        result->val = exp(x);
        result->err = 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
}

static double erfc8_sum(double x)
{
    /* estimates erfc(x) valid for 8 < x < 100 */
    /* This is based on index 5725 in Hart et al */

    static double P[] = {
        2.97886562639399288862,
        7.409740605964741794425,
        6.1602098531096305440906,
        5.019049726784267463450058,
        1.275366644729965952479585264,
        0.5641895835477550741253201704
    };
    static double Q[] = {
        3.3690752069827527677,
        9.608965327192787870698,
        17.08144074746600431571095,
        12.0489519278551290360340491,
        9.396034016235054150430579648,
        2.260528520767326969591866945,
        1.0
    };
    double num=0.0, den=0.0;
    int i;

    num = P[5];
    for (i=4; i>=0; --i) {
        num = x*num + P[i];
    }
    den = Q[6];
    for (i=5; i>=0; --i) {
        den = x*den + Q[i];
    }

    return num/den;
}

inline
static double erfc8(double x)
{
    double e;
    e = erfc8_sum(x);
    e *= exp(-x*x);
    return e;
}

```

```

inline
static double log_erfc8(double x)
{
    double e;
    e = erfc8_sum(x);
    e = log(e) - x*x;
    return e;
}

#if 0
/* Abramowitz+Stegun, 7.2.14 */
static double erfcasympsum(double x)
{
    int i;
    double e = 1.;
    double coef = 1.;
    for (i=1; i<5; ++i) {
        /* coef *= -(2*i-1)/(2*x*x); ??? [GJ] */
        coef *= -(2*i+1)/(i*(4*x*x*x*x));
        e += coef;
        /*
        if (fabs(coef) < 1.0e-15) break;
        if (fabs(coef) > 1.0e10) break;

        [GJ]: These tests are not useful. This function is only
        used below. Took them out; they gum up the pipeline.
        */
    }
    return e;
}
#endif /* 0 */

/* Abramowitz+Stegun, 7.1.5 */
static int erfseries(double x, gsl_sf_result * result)
{
    double coef = x;
    double e = coef;
    double del;
    int k;
    for (k=1; k<30; ++k) {
        coef *= -x*x/k;
        del = coef/(2.0*k+1.0);
        e += del;
    }
    result->val = 2.0 / M_SQRTPI * e;
    result->err = 2.0 / M_SQRTPI * (fabs(del) + GSL_DBL_EPSILON);
    return GSL_SUCCESS;
}

/* Chebyshev fit for erfc((t+1)/2), -1 < t < 1
    erfc(0...1)
*/

```

```

static double erfc_xlt1_data[20] = {
    1.06073416421769980345174155056,
    -0.42582445804381043569204735291,
    0.04955262679620434040357683080,
    0.00449293488768382749558001242,
    -0.00129194104658496953494224761,
    -0.00001836389292149396270416979,
    0.00002211114704099526291538556,
    -5.23337485234257134673693179020e-7,
    -2.78184788833537885382530989578e-7,
    1.41158092748813114560316684249e-8,
    2.72571296330561699984539141865e-9,
    -2.06343904872070629406401492476e-10,
    -2.14273991996785367924201401812e-11,
    2.22990255539358204580285098119e-12,
    1.36250074650698280575807934155e-13,
    -1.95144010922293091898995913038e-14,
    -6.85627169231704599442806370690e-16,
    1.44506492869699938239521607493e-16,
    2.45935306460536488037576200030e-18,
    -9.29599561220523396007359328540e-19
};
static cheb_series erfc_xlt1_cs = {
    erfc_xlt1_data,
    19,
    -1, 1,
    12
};

/* Chebyshev fit for erfc(x) exp(x^2), 1 < x < 5, x = 2t + 3, -1 < t <
1
*/
static double erfc_x15_data[25] = {
    0.44045832024338111077637466616,
    -0.143958836762168335790826895326,
    0.044786499817939267247056666937,
    -0.013343124200271211203618353102,
    0.003824682739750469767692372556,
    -0.001058699227195126547306482530,
    0.000283859419210073742736310108,
    -0.000073906170662206760483959432,
    0.000018725312521489179015872934,
    -4.62530981164919445131297264430e-6,
    1.11558657244432857487884006422e-6,
    -2.63098662650834130067808832725e-7,
    6.07462122724551777372119408710e-8,
    -1.37460865539865444777251011793e-8,
    3.05157051905475145520096717210e-9,
    -6.65174789720310713757307724790e-10,
    1.42483346273207784489792999706e-10,
    -3.00141127395323902092018744545e-11,
    6.22171792645348091472914001250e-12,
    -1.26994639225668496876152836555e-12,
    2.55385883033257575402681845385e-13,

```



```

-5.06258237507038698392265499770e-14,
 9.89705409478327321641264227110e-15,
-1.90685978789192181051961024995e-15,
 3.50826648032737849245113757340e-16
};
static cheb_series erfc_x15_cs = {
    erfc_x15_data,
    24,
    -1, 1,
    16
};

/* Chebyshev fit for erfc(x) x exp(x^2), 5 < x < 10, x = (5t + 15)/2, -
1 < t < 1
*/
static double erfc_x510_data[20] = {
    1.11684990123545698684297865808,
    0.003736240359381998520654927536,
    -0.000916623948045470238763619870,
    0.000199094325044940833965078819,
    -0.000040276384918650072591781859,
    7.76515264697061049477127605790e-6,
    -1.44464794206689070402099225301e-6,
    2.61311930343463958393485241947e-7,
    -4.61833026634844152345304095560e-8,
    8.00253111512943601598732144340e-9,
    -1.36291114862793031395712122089e-9,
    2.28570483090160869607683087722e-10,
    -3.78022521563251805044056974560e-11,
    6.17253683874528285729910462130e-12,
    -9.96019290955316888445830597430e-13,
    1.58953143706980770269506726000e-13,
    -2.51045971047162509999527428316e-14,
    3.92607828989125810013581287560e-15,
    -6.07970619384160374392535453420e-16,
    9.12600607264794717315507477670e-17
};
static cheb_series erfc_x510_cs = {
    erfc_x510_data,
    19,
    -1, 1,
    12
};

#if 0
inline
static double
erfc_asymptotic(double x)
{
    return exp(-x*x)/x * erfcasympsum(x) / M_SQRTPI;
}
inline
static double
log_erfc_asymptotic(double x)

```

```

{
    return log(erfcasympsum(x)/x) - x*x - LogRootPi_;
}
#endif /* 0 */

/*-_-_-_-_-_-_-_-_-_-_-_- Functions with Error Codes -_-_-_-_-_-_-_-_-_-_-_-
*-_-*/

int gsl_sf_erfc_e(double x, gsl_sf_result * result)
{
    const double ax = fabs(x);
    double e_val, e_err;

    /* CHECK_POINTER(result) */

    if(ax <= 1.0) {
        double t = 2.0*ax - 1.0;
        gsl_sf_result c;
        cheb_eval_e(&erfc_xlt1_cs, t, &c);
        e_val = c.val;
        e_err = c.err;
    }
    else if(ax <= 5.0) {
        double ex2 = exp(-x*x);
        double t = 0.5*(ax-3.0);
        gsl_sf_result c;
        cheb_eval_e(&erfc_x15_cs, t, &c);
        e_val = ex2 * c.val;
        e_err = ex2 * (c.err + 2.0*fabs(x)*GSL_DBL_EPSILON);
    }
    else if(ax < 10.0) {
        double exterm = exp(-x*x) / ax;
        double t = (2.0*ax - 15.0)/5.0;
        gsl_sf_result c;
        cheb_eval_e(&erfc_x510_cs, t, &c);
        e_val = exterm * c.val;
        e_err = exterm * (c.err + 2.0*fabs(x)*GSL_DBL_EPSILON +
GSL_DBL_EPSILON);
    }
    else {
        e_val = erfc8(ax);
        e_err = (x*x + 1.0) * GSL_DBL_EPSILON * fabs(e_val);
    }

    if(x < 0.0) {
        result->val = 2.0 - e_val;
        result->err = e_err;
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
    }
    else {
        result->val = e_val;
        result->err = e_err;
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
    }
}

```

```

    }

    return GSL_SUCCESS;
}

int gsl_sf_log_erfc_e(double x, gsl_sf_result * result)
{
    /* CHECK_POINTER(result) */

    if(x*x < 10.0*GSL_ROOT6_DBL_EPSILON) {
        const double y = x / M_SQRTPI;
        /* series for -1/2 Log[Erfc[Sqrt[Pi] y]] */
        const double c3 = (4.0 - M_PI)/3.0;
        const double c4 = 2.0*(1.0 - M_PI/3.0);
        const double c5 = -0.001829764677455021; /* (96.0 - 40.0*M_PI +
3.0*M_PI*M_PI)/30.0 */
        const double c6 = 0.02629651521057465; /* 2.0*(120.0 - 60.0*M_PI
+ 7.0*M_PI*M_PI)/45.0 */
        const double c7 = -0.01621575378835404;
        const double c8 = 0.00125993961762116;
        const double c9 = 0.00556964649138;
        const double c10 = -0.0045563339802;
        const double c11 = 0.0009461589032;
        const double c12 = 0.0013200243174;
        const double c13 = -0.00142906;
        const double c14 = 0.00048204;
        double series = c8 + y*(c9 + y*(c10 + y*(c11 + y*(c12 + y*(c13 +
c14*y)))));
        series = y*(1.0 + y*(1.0 + y*(c3 + y*(c4 + y*(c5 + y*(c6 + y*(c7 +
y*series))))));
        result->val = -2.0 * series;
        result->err = 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
    /*
    don't like use of log1p(); added above series stuff for small x
    instead, should be ok [GJ]
    else if (fabs(x) < 1.0) {
        gsl_sf_result result_erf;
        gsl_sf_erf_e(x, &result_erf);
        result->val = log1p(-result_erf.val);
        result->err = 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
    */
    else if(x > 8.0) {
        result->val = log_erfc8(x);
        result->err = 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
    else {
        gsl_sf_result result_erfc;
        gsl_sf_erfc_e(x, &result_erfc);

```

```

        result->val = log(result_erfc.val);
        result->err = fabs(result_erfc.err / result_erfc.val);
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
}

int gsl_sf_erf_e(double x, gsl_sf_result * result)
{
    /* CHECK_POINTER(result) */

    if(fabs(x) < 1.0) {
        return erfseries(x, result);
    }
    else {
        gsl_sf_result result_erfc;
        gsl_sf_erfc_e(x, &result_erfc);
        result->val = 1.0 - result_erfc.val;
        result->err = result_erfc.err;
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
}

int gsl_sf_erf_Z_e(double x, gsl_sf_result * result)
{
    /* CHECK_POINTER(result) */

    {
        const double ex2 = exp(-x*x/2.0);
        result->val = ex2 / (M_SQRT2 * M_SQRTPI);
        result->err = fabs(x * result->val) * GSL_DBL_EPSILON;
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
    }
    // CHECK_UNDERFLOW(result);
    return GSL_SUCCESS;
}

int gsl_sf_erf_Q_e(double x, gsl_sf_result * result)
{
    /* CHECK_POINTER(result) */

    {
        gsl_sf_result result_erfc;
        int stat = gsl_sf_erfc_e(x/M_SQRT2, &result_erfc);
        result->val = 0.5 * result_erfc.val;
        result->err = 0.5 * result_erfc.err;
        result->err += 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return stat;
    }
}

```

```

int gsl_sf_hazard_e(double x, gsl_sf_result * result)
{
    if(x < 25.0)
    {
        gsl_sf_result result_ln_erfc;
        const int stat_l = gsl_sf_log_erfc_e(x/M_SQRT2, &result_ln_erfc);
        const double lnc = -0.22579135264472743236; /* ln(sqrt(2/pi)) */
        const double arg = lnc - 0.5*x*x - result_ln_erfc.val;
        const int stat_e = gsl_sf_exp_e(arg, result);
        result->err += 3.0 * (1.0 + fabs(x)) * GSL_DBL_EPSILON *
fabs(result->val);
        result->err += fabs(result_ln_erfc.err * result->val);
        return GSL_ERROR_SELECT_2(stat_l, stat_e);
    }
    else
    {
        const double ix2 = 1.0/(x*x);
        const double corrB = 1.0 - 9.0*ix2 * (1.0 - 11.0*ix2);
        const double corrM = 1.0 - 5.0*ix2 * (1.0 - 7.0*ix2 * corrB);
        const double corrT = 1.0 - ix2 * (1.0 - 3.0*ix2*corrM);
        result->val = x / corrT;
        result->err = 2.0 * GSL_DBL_EPSILON * fabs(result->val);
        return GSL_SUCCESS;
    }
}

//-*-***** Functions w/ Natural Prototypes *-*-*
//
//
// #include "eval.h"
//
// double gsl_sf_erfc(double x)
// {
//     EVAL_RESULT(gsl_sf_erfc_e(x, &result));
// }
//
// double gsl_sf_log_erfc(double x)
// {
//     EVAL_RESULT(gsl_sf_log_erfc_e(x, &result));
// }
//
#define EVAL_RESULT(fn) \
    gsl_sf_result result; \
    int status = fn; \
    if (status != GSL_SUCCESS) { \
        GSL_ERROR_VAL(#fn, status, result.val); \
    } ; \
    return result.val;

```

```

double gsl_sf_erf(double x)
{
    EVAL_RESULT(gsl_sf_erf_e(x, &result));
}
//
//double gsl_sf_erf_Z(double x)
//{
//    EVAL_RESULT(gsl_sf_erf_Z_e(x, &result));
//}
//
//double gsl_sf_erf_Q(double x)
//{
//    EVAL_RESULT(gsl_sf_erf_Q_e(x, &result));
//}
//
//double gsl_sf_hazard(double x)
//{
//    EVAL_RESULT(gsl_sf_hazard_e(x, &result));
//}
//

```

field_solver.cpp.

```

//#include "lprobe60.h"
//#include "lapack.cpp"

/*    This function computes the elcetric field intensity at
    the gridpoints using the updated source term and the
    previously factored [A] matrix    */
void e_field(GRIDDEF** array) /* 02Feb, 2010 */
//void e_field(GRIDDEF array[ngp1][ngp2])
//void e_field(GRIDDEF array[ngp1+1][ngp2+1])
{

    long int j,k,count1;
    float deltx,jmax,kmax;
    char trans[]="N";

    nrhs=1;
    jmax=ngp_vert;
    kmax=ngp_horiz;
    deltx=cell_width;

    /*    Zero out source term    */
    count1=-1;
    for(j=0;j<ngp_vert;j++){
        for(k=0;k<ngp_horiz;k++){
            count1++;
            b[count1]=0.0;
        }
    }
}

```

```

/*      Update source term      */
count1=-1;
/*for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        count1++;
        if(k==0 && j!=(jmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot * pi *
                ( ((array[j][k].y +
array[j+1][k].y)/2.0)*((array[j][k].y + array[j+1][k].y)/2.0)
                - ((array[j-1][k].y +
array[j][k].y)/2.0)*((array[j-1][k].y + array[j][k].y)/2.0) )
            / (array[j][k+1].x - array[j][k].x);
        }
        else if(j==(jmax-1) && k!=(kmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot * pi *
                ( (array[j][k].y * array[j][k].y ) - (array[j-
1][k].y * array[j-1][k].y ) )
            /
            (array[j][k+1].x - array[j][k].x);
        }
        else if(j<=jprobe && j>0 && k>=kprobe){
            b[count1]=-array[j][k].q_dens /EPS0 - v * pi *
                ( ((array[j][k].y +
array[j+1][k].y)/2.0)*((array[j][k].y + array[j+1][k].y)/2.0)
                - ((array[j-1][k].y +
array[j][k].y)/2.0)*((array[j-1][k].y + array[j][k].y)/2.0) )
            / (array[j][k].x - array[j][k-1].x);
        }
        else if(j<=jprobe && j==0 && k>=kprobe){
            b[count1]=-array[j][k].q_dens /EPS0 - v * pi *
                ( (array[j+1][k].y * array[j+1][k].y ) -
(array[j][k].y * array[j][k].y ) )
            /
            (array[j][k].x - array[j][k-1].x);
        }
        else if(j>jprobe && k==(kmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 -
plasma_pot*2.0*pi*
array[j][ngp_horiz-1].y-cell_height6/2.0 )*cell_width/cell_height6-
array[j][k]
                .q_dens /EPS0;
        }
        else{
            b[count1]=-array[j][k].q_dens /EPS0;
        }
    }
}
*/
for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        count1++;
        if(k==0 && j!=(jmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;
        }
        else if(j==(jmax-1) && k!=(kmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;
        }
    }
}

```

```

    }
    else if(j<=jprobe && j>0 && k>=kprobe){
        b[count1]=-array[j][k].q_dens /EPS0 - V;
    }
    else if(j<=jprobe && j==0 && k>=kprobe){
        b[count1]=-array[j][k].q_dens /EPS0 - V;
    }
    else if(j>jprobe && k==(kmax-1)){
        b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;
    }
    else{
        b[count1]=-array[j][k].q_dens /EPS0;
    }
}

}

/* Call Lapack to solve [a]x=b */
/*printf("%ld %d\n",t_count2,ldab);*/
sgbtrs_(trans,&n1,&lower_width,&upper_width,&nrhs,abd,&ldab,piv,b,&ldb,
&info);
if (info != 0){
    printf("Band Solver Failed, info==>%d\n",info);
    exit(1);
}

/* Put phi into grid (note:this is a waste, but for now....) */
count1=-1;
for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        count1++;
        array[j][k].phi=b[count1];
    }
}

/* Determine electric field at all gridpoints */
for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        if (k==0){
            array[j][k].Ex=1.0/(2.0*deltx)*(-
3.0*array[j][0].phi+4.0*array[j][1].phi-array[j][2].phi);
            array[j][k].Ey=0.0;
        }
        else if (j==jprobe && k >= kprobe){
            array[j][k].Ey=1.0/(2.0*deltx)*(-
3.0*array[j][k].phi+4.0*array[j][k+1].phi-array[j][k+2].phi);
            array[j][k].Ex=0.0;
        }
        else if (k==(ngp_horiz-1)){
            array[j][k].Ex=-1.0/(2.0*deltx)*(-
3.0*array[j][ngp_horiz-1].phi+4.0*array[j][ngp_horiz-2].phi-
array[j][ngp_horiz-3].phi);
            array[j][k].Ey=0.0;
        }
        else if(j==0){

```



```

        array[j][k].Ex=-(array[j][k-1].phi-
array[j][k+1].phi)/(2.0*deltx);
        array[j][k].Ey=0.0;
    }
    else if(j==(ngp_vert-1)){
        array[j][k].Ex=0.0;
        array[j][k].Ey=array[j-1][k].phi/cell_height6;
    }
    else
    {
        array[j][k].Ex=(array[j][k-1].phi-
array[j][k+1].phi)/(2.0*deltx);
        array[j][k].Ey=-((array[j+1][k].phi-
array[j][k].phi)/(array[j+1][k].y-array[j][k].y))*(array[j][k].y-
array[j-1][k].y)/(array[j+1][k].y-array[j-1][k].y)
        -((array[j][k].phi-array[j-
1][k].phi)/(array[j][k].y-array[j-1][k].y))*(array[j+1][k].y-
array[j][k].y)/(array[j+1][k].y-array[j-1][k].y);
    }

}
}
}

```

field_solver_v1.cpp.

```

// #define EXTERN extern
// #include "lprobe60.h"

/*      This function computes the electric field intensity at
        the gridpoints using the updated source term and the
        previously factored [A] matrix      */
void e_field(GRIDDEF** array) /* 02Feb, 2010 */
//void e_field(GRIDDEF array[ngp1][ngp2])
{
    long int j,k,count1;
    float deltx,jmax,kmax;
    char trans[]="N";

    nrhs=1;
    jmax=ngp_vert;
    kmax=ngp_horiz;
    deltx=cell_width;

    /*      Zero out source term      */
    count1=-1;
    for(j=0;j<ngp_vert;j++){
        for(k=0;k<ngp_horiz;k++){
            count1++;
            b[count1]=0.0;
        }
    }
}

```

```

}

/*      Update source term      */
count1=-1;
/*for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        count1++;
        if(k==0 && j!=(jmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot * pi *
                ( ((array[j][k].y +
array[j+1][k].y)/2.0)*((array[j][k].y + array[j+1][k].y)/2.0)
                - ((array[j-1][k].y +
array[j][k].y)/2.0)*((array[j-1][k].y + array[j][k].y)/2.0) )
            / (array[j][k+1].x - array[j][k].x);
        }
        else if(j==(jmax-1) && k!=(kmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot * pi *
                ( (array[j][k].y * array[j][k].y ) - (array[j-
1][k].y * array[j-1][k].y ) )
            /
            (array[j][k+1].x - array[j][k].x);
        }
        else if(j<=jprobe && j>0 && k>=kprobe){
            b[count1]=-array[j][k].q_dens /EPS0 - v * pi *
                ( ((array[j][k].y +
array[j+1][k].y)/2.0)*((array[j][k].y + array[j+1][k].y)/2.0)
                - ((array[j-1][k].y +
array[j][k].y)/2.0)*((array[j-1][k].y + array[j][k].y)/2.0) )
            / (array[j][k].x - array[j][k-1].x);
        }
        else if(j<=jprobe && j==0 && k>=kprobe){
            b[count1]=-array[j][k].q_dens /EPS0 - v * pi *
                ( (array[j+1][k].y * array[j+1][k].y ) -
(array[j][k].y * array[j][k].y ) )
            /
            (array[j][k].x - array[j][k-1].x);
        }
        else if(j>jprobe && k==(kmax-1)){
            b[count1]=-array[j][k].q_dens /EPS0 -
plasma_pot*2.0*pi*
            (
array[j][ngp_horiz-1].y-cell_height6/2.0 )*cell_width/cell_height6-
array[j][k]
                .q_dens /EPS0;
        }
        else{
            b[count1]=-array[j][k].q_dens /EPS0;
        }
    }
}
*/
for(j=0;j<ngp_vert;j++){
    for(k=0;k<ngp_horiz;k++){
        count1++;
        if(k==0 && j!=(jmax-1)){ // upstream boundary except r =
rMax
            b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;

```

```

    }
    else if(j==(jmax-1) && k!=(kmax-1)){ // top boundary except z =
0      b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;
    }
    else if(j<=jprobe && j>0 && k>=kprobe){// body conductor
except axis      b[count1]=-array[j][k].q_dens /EPS0 - V;
    }
    else if(j<=jprobe && j==0 && k>=kprobe){// body conductor
: the axis      b[count1]=-array[j][k].q_dens /EPS0 - V;
    }
    else if(j>jprobe && k==(kmax-1)){// downstream boundary,
above the conductor (includes insulator)      b[count1]=-array[j][k].q_dens /EPS0 - plasma_pot;
    }
    else{// regular interior mesh points ?
      b[count1]=-array[j][k].q_dens /EPS0;
    }
  }
}

/* Call Lapack to solve [a]x=b */
/*printf("%ld %d\n",t_count2,ldab);*/
sgbtrs_(trans,&n1,&lower_width,&upper_width,&nrhs,abd,&ldab,piv,b,&ldb,
&info);
if (info != 0){
  printf("Band Solver Failed, info==>%d\n",info);
  exit(1);
}

/* Put phi into grid (note:this is a waste, but for now....) */
count1=-1;
for(j=0;j<ngp_vert;j++){
  for(k=0;k<ngp_horiz;k++){
    count1++;
    array[j][k].phi=b[count1];
  }
}

/* Determine electric field at all gridpoints */
for(j=0;j<ngp_vert;j++){
  for(k=0;k<ngp_horiz;k++){
    if (k==0){// entire upstream boundary
      array[j][k].Ex=1.0/(2.0*deltx)*(-
3.0*array[j][0].phi+4.0*array[j][1].phi-array[j][2].phi);
      array[j][k].Ey=0.0;
    }// k!=0: away from upstream boundary

    // else if (j==jprobe && k >= kprobe){ // upper part of the
conductor (all its length)

    /* 25 Mar 2010 */

```

```

        else if (j<=jprobe && k >= kprobe){ // body conductor
            array[j][k].Ey=1.0/(2.0*deltx)*(-
3.0*array[j][k].phi+4.0*array[j][k+1].phi-array[j][k+2].phi);
            array[j][k].Ex=0.0;
        }
        else if (k==(ngp_horiz-1)){ // entire downstream boundary
            array[j][k].Ex=-1.0/(2.0*deltx)*(-
3.0*array[j][ngp_horiz-1].phi+4.0*array[j][ngp_horiz-2].phi-
array[j][ngp_horiz-3].phi);
            array[j][k].Ey=0.0;
        } // k!= ngp_horiz-1: away from downstream boundary
        else if (j==0){ // axis r=0
            array[j][k].Ex=-(array[j][k-1].phi-
array[j][k+1].phi)/(2.0*deltx);
            array[j][k].Ey=0.0;
        }
        else if (j==(ngp_vert-1)){ // entire top boundary
            array[j][k].Ex=0.0;
            //array[j][k].Ey=array[j-1][k].phi/cell_height6;
            array[j][k].Ey=array[j-1][k].phi/(array[j][k].y-
array[j-1][k].y); // 07Mar, 2010
            // as now 'cell_height6' is
            not the height of the upper rows of cells, but the cells height
            increase rate
        }
        else // regular mesh points
        {
            array[j][k].Ex=(array[j][k-1].phi-
array[j][k+1].phi)/(2.0*deltx);
            array[j][k].Ey=-((array[j+1][k].phi-
array[j][k].phi)/(array[j+1][k].y-array[j][k].y))*(array[j][k].y-
array[j-1][k].y)/(array[j+1][k].y-array[j-1][k].y)
            -((array[j][k].phi-array[j-
1][k].phi)/(array[j][k].y-array[j-1][k].y))*(array[j+1][k].y-
array[j][k].y)/(array[j+1][k].y-array[j-1][k].y);
        }
    }
}
}
}

```

force_weight.cpp.

```

// #include "lprobe60.h"

/* This function weights the field from the array */

void fweight(GRIDDEF** array, PARTICLEDEF particle[numpart], long int
number) /* 02Feb, 2010 */

```

```

//void fweight(GRIDDEF array[ngp1][ngp2], PARTICLEDEF
particle[numpart], long int number)
//void fweight(GRIDDEF array[ngp1+1][ngp2+1], PARTICLEDEF
particle[numpart], long int number)
{
long int count1;
float dr, dr1, dr2, dz, dz1, dz2;

/*    CLEAN UP OLD ELECTRIC FIELD    */

for (count1=0; count1 < number; count1++){
    particle[count1].Ey=particle[count1].Ex=0.0;
}

for(count1=0; count1 < number; count1++){

    if (particle[count1].r <= radius){
        // particle in a domain cell ?
        if ( (particle[count1].celly+1 < ngp1) &&
            (particle[count1].cellx+1 < ngp2)) {
            dr=array[particle[count1].celly+1][particle[count1].cellx].y*
                array[particle[count1].celly+1][particle[count1].cellx].y
            -
                array[particle[count1].celly][particle[count1].cellx].y
                *array[particle[count1].celly][particle[count1].cellx].y;
            dr1=particle[count1].r -
                array[particle[count1].celly][particle[count1].cellx].y;
            dr2=array[particle[count1].celly+1][particle[count1].cellx].y -
                particle[count1].r;

            dz=array[particle[count1].celly][particle[count1].cellx+1].x -
                array[particle[count1].celly][particle[count1].cellx].x;
            dz1=particle[count1].x -
                array[particle[count1].celly][particle[count1].cellx].x;
            dz2=array[particle[count1].celly][particle[count1].cellx+1].x -
                particle[count1].x;

            particle[count1].Ey=array[particle[count1].celly][particle[count1].cell
x].Ey*dr2 * dz2
                * (2.0 *
array[particle[count1].celly+1][particle[count1].cellx].y + 3.0
                *
array[particle[count1].celly][particle[count1].cellx].y -
                particle[count1].r) / (2.0 * dr * dz)
            +
array[particle[count1].celly][particle[count1].cellx+1].Ey*dr2 * dz1
                * (2.0 *
array[particle[count1].celly+1][particle[count1].cellx+1].y + 3.0 *
array[particle[count1].celly][particle[count1].cellx].y -
                particle[count1].r) /

```

```

        (2.0 * dr * dz ) +
array[particle[count1].celly+1][particle[count1].cellx+1].Ey*
        dr1 * dz1 * (3.0 *
array[particle[count1].celly+1][particle[count1].cellx+1].y +
        2.0 *
array[particle[count1].celly][particle[count1].cellx+1].y -
        particle[count1].r) / (2.0 * dr * dz) +

array[particle[count1].celly+1][particle[count1].cellx].Ey*dr1 * dz2 *
(3.0 *

array[particle[count1].celly+1][particle[count1].cellx].y + 2.0 *

array[particle[count1].celly][particle[count1].cellx].y -
particle[count1].r)
        / (2.0 * dr * dz);

particle[count1].Ex=array[particle[count1].celly][particle[count1].cell
x].Ex*dr2 * dz2
        * (2.0 *
array[particle[count1].celly+1][particle[count1].cellx].y + 3.0
        *
array[particle[count1].celly][particle[count1].cellx].y -
        particle[count1].r) / (2.0 * dr * dz)
        +
array[particle[count1].celly][particle[count1].cellx+1].Ex*dr2 * dz1
        * (2.0 *
array[particle[count1].celly+1][particle[count1].cellx+1].y + 3.0 *

array[particle[count1].celly][particle[count1].cellx].y -
particle[count1].r) /
        (2.0 * dr * dz ) +
array[particle[count1].celly+1][particle[count1].cellx+1].Ex*
        dr1 * dz1 * (3.0 *
array[particle[count1].celly+1][particle[count1].cellx+1].y +
        2.0 *
array[particle[count1].celly][particle[count1].cellx+1].y -
        particle[count1].r) / (2.0 * dr * dz) +

array[particle[count1].celly+1][particle[count1].cellx].Ex*dr1 * dz2 *
(3.0 *

array[particle[count1].celly+1][particle[count1].cellx].y + 2.0 *

array[particle[count1].celly][particle[count1].cellx].y -
particle[count1].r)
        / (2.0 * dr * dz);

    } // celly+1<ngp1
    else
        printf("\nforce_weight.cpp: cellx=%d celly=%d
\n",particle[count1].cellx,particle[count1].celly);
    }; // r<radius

```

```
}; // count1

}; // fweight
```

grid.cpp.

```
//#include "lprobe60.h"

void gridgen(GRIDDEF** array)
//void gridgen(GRIDDEF array[ngp1][ngp2])
//void gridgen(GRIDDEF array[ngp1+1][ngp2+1])
{
    long int count1,count2,cell1_count,cell6_count;
    float index1,index2;

    ngp_vert=ngp1;
    ngp_horiz=ngp2;

    // The description below holds true for original PROBEPIE mesh: on
    // vertical,one cell = probe radius
    // For later versions, each row is divided into 'ncpr' strips
        // the first 15 rows of cells has 'cell_height1' height
            // row #16 of cells has 'cell_height2' height
            // row #17 of cells has 'cell_height3' height
            // row #18 of cells has 'cell_height4' height
            // row #19 of cells has 'cell_height5' height
        // the last rows, #20...#73 of cells has 'cell_height6' height
    unsigned int count_vert = 0; // is incremented when a new vertical row
    is added to mesh

                                                                    // In the end its value
    should be 74*ncpr
    cell1_count=15; // the first 15 rows of cells has 'cell_height1'
    height
    cell1_count=ncpr+ncpr+n0Rows15; // the first rows of cells has
    'r_probe/ncpr' height
                                                                    // they include probe
    (ncpr), insulator(ncpr) and n0Rows15
    index1=-1.0;
    //for(int i=0; i<ncpr; i++)
    for (count1=0;count1 < cell1_count;count1++){ // rows 0...14
        index1=index1+1.0;
        index2=-1.0;
        for (count2=0;count2 < ngp_horiz;count2++){
            // for (count2=0;count2 <= ngp_horiz;count2++){
                index2=index2+1.0;
                array[count_vert][count2].x=-length+index2*cell_width;
                array[count_vert][count2].y=index1*(cell_height1/ncpr);
            } // count2
            count_vert++;
        } // count1 & i
```

```

goto skipSomeVerticalCells;

//index2=-1.0;
for(int i=0; i<ncpr; i++) {
    index2=-1.0;
    for (count2=0;count2 < ngp_horiz;count2++){
        //for (count2=0;count2 <= ngp_horiz;count2++){ // row 15
            index2=index2+1.0;
            array[count_vert][count2].x=-length+index2*cell_width;
            //array[15][count2].y= array[14][count2].y + cell_height2);
            //array[count_vert][count2].y= array[count_vert-
1][count2].y+(i+1)*(cell_height2/ncpr);
            array[count_vert][count2].y= array[count_vert-
1][count2].y+(cell_height2/ncpr);
        }
        count_vert++;
    } // i

//index2=-1.0;
for(int i=0; i<ncpr; i++) {
    index2=-1.0;
    for (count2=0;count2 < ngp_horiz;count2++){
        //for (count2=0;count2 <= ngp_horiz;count2++){ // row 16
            index2=index2+1.0;
            array[count_vert][count2].x=-length+index2*cell_width;
            //array[count_vert][count2].y=array[count_vert-
1][count2].y+(i+1)*(cell_height3/ncpr);
            array[count_vert][count2].y=array[count_vert-
1][count2].y+(cell_height3/ncpr);
        }
        count_vert++;
    } // i

//index2=-1.0;
for(int i=0; i<ncpr; i++) {
    index2=-1.0;
    for (count2=0;count2 < ngp_horiz;count2++){
        //for (count2=0;count2 <= ngp_horiz;count2++){ // row 17
            index2=index2+1.0;
            array[count_vert][count2].x=-length+index2*cell_width;
            //array[count_vert][count2].y=array[count_vert-
1][count2].y+(i+1)*(cell_height4/ncpr);
            array[count_vert][count2].y=array[count_vert-1][count2].y+
(cell_height4/ncpr);
        }
        count_vert++;
    } // i

//index2=-1.0;
for(int i=0; i<ncpr; i++) {
    index2=-1.0;
    for (count2=0;count2 < ngp_horiz;count2++){
        //for (count2=0;count2 <= ngp_horiz;count2++){ // row 18
            index2=index2+1.0;

```



```

        array[count_vert][count2].x=-length+index2*cell_width;
        //array[count_vert][count2].y=array[count_vert-
1][count2].y+(i+1)*(cell_height5/ncpr);
        array[count_vert][count2].y=array[count_vert-1][count2].y+
(cell_height5/ncpr);
    }
    count_vert++;
}; // i

//cell_height6 = (radius - array[18][0].y)/(ngp_vert-18);
//cell6_count=74; // = ngp_vert = ngp1

skipSomeVerticalCells:
    //
    //cell6_count= ngp_vert; // January 29, 2010
    cell6_count= count_vert; // Feb02, 2010
    //ngp_vert=ngp1=count_vert; // Mar05/ 2010: renounce to the
    linearly varying cells on vertical direction
    //goto skipSomeVerticalCells;

index1=-1.0;
    //for(int i=0; i<ncpr; i++)
    for (count1=cell6_count;count1 < ngp_vert;count1++){ // rows
19...73
        //for (count1=19;count1 < cell6_count;count1++){ // rows 19...73
        index1=index1+1.0;
        index2=-1.0;
        for (count2=0;count2 < ngp_horiz;count2++){
            // for (count2=0;count2 <= ngp_horiz;count2++){
            index2=index2+1.0;
            array[count_vert][count2].x=-length+index2*cell_width;
            //array[count_vert][count2].y=array[count_vert-
1][count2].y + (cell_height6/ncpr);
            array[count_vert][count2].y=2*array[count_vert-
1][count2].y -array[count_vert-2][count2].y + cell_height6;
            }; // count2
            count_vert++;
        }; // count1 & i

        // as we had cumulative errors in cells height, the last row of
points on grid
        // need a small correction for y component
        for (count2=0;count2 < ngp_horiz;count2++)
            array[ngp_vert-1][count2].y=radius;

    // // the upper left out-of-boundary corner cell(74,74)
    // array[ngp_vert][ngp_horiz].x = array[0][ngp_horiz-1].x +
cell_width;
    // array[ngp_vert][ngp_horiz].y = array[ngp_vert-1][0].y +
cell_height1;
    //// Add an out-of-boundary row of cells (needed in qweight.cpp)
    // for (count2=0;count2 <= ngp_horiz;count2++){ // row 74
    //     count1 = ngp_vert;
    //     array[count1][count2].x = array[count1-1][count2].x;

```

```

//      array[count1][count2].y = array[count1-1][count2].y +
cell_height6;
//    };

/*    Clean-up grid for next time step    */
for (count1=0;count1 < ngp_vert;count1++){
//for (count1=0;count1 <= ngp_vert;count1++){
    for (count2=0;count2 < ngp_horiz;count2++){
//    for (count2=0;count2 <= ngp_horiz;count2++){

array[count1][count2].q_dens=array[count1][count2].Ex=array[count1][cou
nt2].Ey=0.0;
    }
}

}

```

initialize.cpp.

```

//#include "lprobe60.h"

void initialize(PARTICLEDEF particle[numpart],long int *num_part,long
int init_num,float super_init,float *v_table,float m,float q)
{
    long int i,rannum;
    float theta,phi,v,r;

    for(i=0;i < init_num;i++){

        theta=pi*random_num[rancount];
        rancount ++;
        phi=2.0*pi*random_num[rancount];
        rancount ++;
        rannum=((long)V_TABLE_RES*random_num[rancount]);
        rancount ++;
        v=v_table[rannum];
        rancount ++;
        r=radius * sqrt(random_num[rancount]); // why 'sqrt' ?
        rancount ++;
        /*printf("%i \n", *num_part);*/
        particle[*num_part].x=-length * random_num[rancount];
        rancount++;
        particle[*num_part].y = r*sin(theta);
        particle[*num_part].z = r*cos(theta);
//        particle[*num_part].r =
sqrt(particle[*num_part].y*particle[*num_part].y +
particle[*num_part].z *particle[*num_part].z);
        particle[*num_part].r = r;
        particle[*num_part].vx = v * sin(theta) *sin(phi);
    }
}

```

```

        particle[*num_part].vy = v * cos(theta);
        particle[*num_part].vz = v * sin(theta) * cos(phi);
        particle[*num_part].Ex = 0.0;
        particle[*num_part].Ey = 0.0;
        particle[*num_part].q = q * super_init;
        particle[*num_part].m = m * super_init;
        (*num_part)++;
    }

}

```

inject_part.cpp.

```

#include "lprobe60.h"

extern void locatex_2(GRIDDEF** xx, unsigned short n, float x, unsigned
short *jj, unsigned short jStart); /* 11Mar,2010 */
extern void locatey_2(GRIDDEF** xx, unsigned short n, float y, unsigned
short *jj, unsigned short jStart); /* 11Mar,2010 */

/* This function injects particles at each time step */

void inject(PARTICLEDEF particle[numpart],long int *num_part,
            float super_back,float super_front, float super_side,
// the super-particle weighting (Thesis, page 58)
            long int superinj_back,long int superinj_front, long
int superinj_side, // num. of injected particles at each dti/dte step
            float *v_table,float m,float q)
{
    long int i,rannum;
    float r,theta,phi,v;
    unsigned short positionx,positiony;

        //cpgsci(7);        // Yellow
        cpgsci(0);        // Black

    // NPTS; no. of elements of array 'random_num'
    if (NPTS<rancount+5*superinj_front) rancount=0; //! make sure that
    'racount' stays in [0..NPTS]

    for(i=0;i < superinj_front;i++){
        /*      FRONT      */
        theta=acos(sqrt(1 - random_num[rancount]));
        rancount ++;
        phi=2.0*pi*random_num[rancount];
        rancount ++;
        rannum=((long int)V_TABLE_RES*random_num[rancount]);
        rancount ++;
        v=v_table[rannum];
        particle[*num_part].x=- length;
    }
}

```

```

//      particle[*num_part].y=height * random_num[rancount] - height /
2.0;
                r= radius * random_num[rancount];
        particle[*num_part].y=r*sin(phi); // 13Mar, 2010
        rancount ++;
//      particle[*num_part].z=width * random_num[rancount] - width /
2.0;
        particle[*num_part].z=r*cos(phi); // 13Mar, 2010
        rancount ++;
//
particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
+ particle[*num_part].z
//      * particle[*num_part].z);
        particle[*num_part].r=r; // 13Mar, 2010

        particle[*num_part].vx=v * cos(theta);
        particle[*num_part].vy=v*sin(theta)*cos(phi);
        particle[*num_part].vz=v*sin(theta)*sin(phi);
        particle[*num_part].q= q * super_front;
        particle[*num_part].m= m * super_front;
        particle[*num_part].Ex=0.0;
        particle[*num_part].Ey=0.0;

        particle[*num_part].cellx=0; // 12 Mar, 2010

//locatey_2(grid,ngp1,fabs(particle[*num_part].y),&positiony,0);
        locatey_2(grid,ngp1,particle[*num_part].r,&positiony,0);
        particle[*num_part].celly=positiony;

        cpgpt(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #(-1)
        //cpgpt(1,&particle[*num_part].x,&particle[*num_part].y,855); //
graphic symbol #(-8)

        (*num_part)++;
        if (*num_part >= numpart) {
                printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
                system("PAUSE");
                exit(1);
        };
}

// NPTS; no. of elements of array 'random_num'
if (NPTS<rancount+3) rancount=0; //! make sure that 'rancount' stays in
[0..NPTS]

for(i=0;i < superinj_back;i++){
        /*      BACK      */
        theta=acos(sqrt(1 - random_num[rancount]));
        rancount ++;
        phi=2.0*pi*random_num[rancount];
        rancount ++;
        rannum=((long int)V_TABLE_RES*random_num[rancount]);

```

```

rancount ++;
v=v_table[rannum];

do{ // avoid injecting in the insulator region
    // NPTS; no. of elements of array 'random_num'
    if (NPTS<rancount+2) rancount=0; ///! make sure that
    'racount' stays in [0..NPTS]

        particle[*num_part].x=0.0;
//        particle[*num_part].y=height * random_num[rancount] - height
// 2.0;
        r= radius * random_num[rancount];
        particle[*num_part].y=r*sin(phi); // 13Mar, 2010
rancount ++;
//        particle[*num_part].z=width * random_num[rancount] - width /
2.0;
        particle[*num_part].z=r*cos(phi); // 13Mar, 2010
rancount ++;
//
particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
+ particle[*num_part].z
//        * particle[*num_part].z);
        particle[*num_part].r=r; // 13Mar, 2010
        particle[*num_part].vx=-v * cos(theta);
        particle[*num_part].vy=v*sin(theta)*cos(phi);
        particle[*num_part].vz=v*sin(theta)*sin(phi);
        particle[*num_part].Ex=0.0;
        particle[*num_part].Ey=0.0;
        particle[*num_part].q= q * super_back; // the super-particle
weighting (Thesis, page 58)
        particle[*num_part].m= m * super_back; // the super-particle
weighting (Thesis, page 58)
    }while( particle[*num_part].r <= r_insul);

        particle[*num_part].cellx=ngp2-2; // 12 Mar, 2010

//locatey_2(grid,ngp1,fabs(particle[*num_part].y),&positiony,0);
        locatey_2(grid,ngp1,particle[*num_part].r,&positiony,0);
        particle[*num_part].celly=positiony;

        cpgpu(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #(-1)

        (*num_part)++;
        if (*num_part >= numpart) {
            printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
            system("PAUSE");
            exit(1);
        };
    }

// NPTS; no. of elements of array 'random_num'

```

```

if (NPTS<rancount+20*superinj_side) rancount=0; // make sure that
'racount' stays in [0..NPTS]
for(i=0;i < superinj_side;i++){
    /*      BOTTOM SIDE      */
    theta=acos(sqrt(1 - random_num[rancount]));
    rancount ++;
    phi=2.0*pi*random_num[rancount];
    rancount ++;
    rannum=((long)V_TABLE_RES*random_num[rancount]);
    rancount ++;
    v=v_table[rannum];
    particle[*num_part].x=-length * random_num[rancount];
    rancount ++;
    //    particle[*num_part].y=-radius;
    particle[*num_part].y=-radius*cos(-pi/4+phi/4.0); // 13Mar,2010
    //    particle[*num_part].z=width * random_num[rancount] - width /
    2.0;
    particle[*num_part].z=radius*sin(-pi/4+phi/4.0); // 13Mar,2010
    rancount ++;
    //
    particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
    + particle[*num_part].z
    //
    * particle[*num_part].z);
    particle[*num_part].r=radius; // 13Mar, 2010
    particle[*num_part].vx=v * sin(theta) *sin(phi);
    particle[*num_part].vy=v * cos(theta);
    particle[*num_part].vz=v*sin(theta)*cos(phi);
    particle[*num_part].Ex=0.0;
    particle[*num_part].Ey=0.0;
    particle[*num_part].q= q * super_side; // the super-particle
weighting (Thesis, page 58)
    particle[*num_part].m= m * super_side; // the super-particle
weighting (Thesis, page 58)

    // 12 Mar, 2010
    particle[*num_part].celly=ngp1-2;
    locatex_2(grid,ngp2,particle[*num_part].x,&positionx,0);
    //locatey_2(grid,ngp1,particle[*num_part].r,&positiony,0);
    particle[*num_part].cellx=positionx;

    //cpgpt(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #1

    (*num_part)++;
    if (*num_part >= numpart) {
        printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
        system("PAUSE");
        exit(1);
    };

    /*      TOP Side      */
    theta=acos(sqrt(1 - random_num[rancount]));

```

```

rancount ++;
    phi=2.0*pi*random_num[rancount];
rancount ++;
    rannum=((long)V_TABLE_RES*random_num[rancount]);
rancount ++;
    v=v_table[rannum];
    particle[*num_part].x=-length * random_num[rancount];
rancount ++;
//    particle[*num_part].y= radius;
    particle[*num_part].y= radius*cos(-pi/4+phi/4.0); // 13Mar,2010
//    particle[*num_part].z=width * random_num[rancount] - width /
2.0;
    particle[*num_part].z= radius*sin(-pi/4+phi/4.0); // 13Mar,2010
rancount ++;
//
particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
+ particle[*num_part].z
//    * particle[*num_part].z);
    particle[*num_part].r=radius; // 13Mar, 2010
    particle[*num_part].vx=v * sin(theta) * sin(phi);
    particle[*num_part].vy=-v * cos(theta);
    particle[*num_part].vz=v * sin(theta) * cos(phi);
    particle[*num_part].Ex=0.0;
    particle[*num_part].Ey=0.0;
    particle[*num_part].q= q * super_side; // the super-particle
weighting (Thesis, page 58)
    particle[*num_part].m= m * super_side; // the super-particle
weighting (Thesis, page 58)

    // 12 Mar, 2010
    particle[*num_part].celly=ngp1-2;
    locatex_2(grid,ngp2,particle[*num_part].x,&positionx,0);
    //locatey_2(grid,ngp1,particle[*num_part].r,&positiony,0);
    particle[*num_part].cellx=positionx;

    //cpgpt(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #1

    (*num_part)++;
    if (*num_part >= numpart) {
        printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
        system("PAUSE");
        exit(1);
    };

    // the two bellow inject particles from OZ axis (depth)

/*      FRONT SIDE      */
    theta=acos(sqrt(1 - random_num[rancount]));
rancount ++;
    phi=2.0*pi*random_num[rancount];
rancount ++;
    rannum=((long)V_TABLE_RES*random_num[rancount]);

```

```

rancount ++;
    v=v_table[rannum];
    particle[*num_part].x=-length * random_num[rancount];
rancount ++;
//    particle[*num_part].y=height * random_num[rancount] - height /
2.0;
    particle[*num_part].y= radius*sin(-pi/4+phi/4.0); // 13Mar,2010
rancount ++;
//    particle[*num_part].z = radius;
    particle[*num_part].z = radius*cos(-pi/4+phi/4.0); // 13Mar,2010
//
particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
+ particle[*num_part].z
//    * particle[*num_part].z);
    particle[*num_part].r=radius; // 13Mar, 2010
    particle[*num_part].vx=v * sin(theta) * cos(phi);
    particle[*num_part].vy=v * sin(theta) * sin(phi);
    particle[*num_part].vz=-v * cos(theta);
    particle[*num_part].Ex=0.0;
    particle[*num_part].Ey=0.0;
    particle[*num_part].q= q * super_side; // the super-particle
weighting (Thesis, page 58)
    particle[*num_part].m= m * super_side; // the super-particle
weighting (Thesis, page 58)

    // 12 Mar, 2010
    locatex_2(grid,ngp2,particle[*num_part].x,&positionx,0);
    particle[*num_part].cellx=positionx;

//locatey_2(grid,ngp1,fabs(particle[*num_part].y),&positiony,0);
//particle[*num_part].celly=positiony;
    particle[*num_part].celly=ngp1-2;

//    cpgpu(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #(-1)

    (*num_part)++;
    if (*num_part >= numpart) {
        printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
        system("PAUSE");
        exit(1);
    };

    /*    BACK SIDE    */
    theta=acos(sqrt(1 - random_num[rancount]));
rancount ++;
    phi=2.0*pi*random_num[rancount];
rancount ++;
    rannum=((long)V_TABLE_RES*random_num[rancount]);
rancount ++;
    v=v_table[rannum];

```



```

        particle[*num_part].x=-length * random_num[rancount];
rancount ++;
//        particle[*num_part].y=height * random_num[rancount] - height /
2.0;
        particle[*num_part].y= radius*sin(-pi/4+phi/4.0); // 13Mar,2010
rancount ++;
//        particle[*num_part].z=-radius;
        particle[*num_part].z=-radius*cos(-pi/4+phi/4.0); // 13Mar,2010
//
particle[*num_part].r=sqrt(particle[*num_part].y*particle[*num_part].y
+ particle[*num_part].z
//        * particle[*num_part].z);
        particle[*num_part].r=radius; // 13Mar, 2010
        particle[*num_part].vx=v * sin(theta) * cos(phi);
        particle[*num_part].vy=v * sin(theta) * sin(phi);
        particle[*num_part].vz=v * cos(theta);
        particle[*num_part].Ex=0.0;
        particle[*num_part].Ey=0.0;
        particle[*num_part].q= q * super_side; // the super-particle
weighting (Thesis, page 58)
        particle[*num_part].m= m * super_side; // the super-particle
weighting (Thesis, page 58)

        // 12 Mar, 2010
        locatex_2(grid,ngp2,particle[*num_part].x,&positionx,0);
        particle[*num_part].cellx=positionx;

//locatey_2(grid,ngp1,fabs(particle[*num_part].y),&positiony,0);
//particle[*num_part].celly=positiony;
        particle[*num_part].celly=ngp1-2;

//        cpgpu(1,&particle[*num_part].x,&particle[*num_part].y,-1); //
graphic symbol #(-1)1

        (*num_part)++;
        if (*num_part >= numpart) {
                printf("*** Too many particles injected: increase
'numpart' in lprobe60.h ****");
                system("PAUSE");
                exit(1);
        };
}
}

```

lapack.cpp:

```

#include "../LAPACK/sger.cpp"
#include "../LAPACK/sscal.cpp"
#include "../LAPACK/scopy.cpp"
#include "../LAPACK/xerbla.cpp"
#include "../LAPACK/ilaenv.cpp"

```

```

#include "../LAPACK/slaswp.cpp"
#include "../LAPACK/isamax.cpp"
#include "../LAPACK/sswap.cpp"
#include "../LAPACK/lsame.cpp"
#include "../LAPACK/stbsv.cpp" // uses xerbla, lsame
#include "../LAPACK/sgemv.cpp" // uses xerbla, lsame
#include "../LAPACK/sgemm.cpp"
#include "../LAPACK/strsm.cpp"
#include "../LAPACK/sgbtf2.cpp"
#include "../LAPACK/sgbtrf.cpp"
#include "../LAPACK/sgbtrs.cpp"

```

locate.cpp.

```

// #include "lprobe60.h"

// det. what is the lower horizontal point number on grid, closest to
// 'x'
// we assume that grid's horizontal points are ascending ordered, in
// order to skip setting 'ascnd'
// we assume that grid has max. 65537 horizontal points, which is a
// reasonable assumption
// to speed up the search, we start from the position indicated by
// 'jStart' instead of the middle of the interval [0,n+1]
void locatex_2(GRIDDEF** grid,
               unsigned short n, // total number of grid's
horizontal points
               float x,
               unsigned short *jj, // location of 'x'
               unsigned short jStart)
{
    unsigned short ju,jm,jl;
    //bool ascnd=1;
    //ascnd=(grid[0][n-1].x > grid[0][1].x); // determines how the values
of grid[0][.] are ordered; in our case, ascending(1)
    // if(ascnd) {
        if (x > grid[0][n-1].x) {*jj=n+1; return;} // exceeds upper
limit
        //if (x < grid[0][ 1].x) {*jj=n+1; return;} // exceeds lower
limit
        if (x < grid[0][ 0].x) {*jj=n+1; return;} // exceeds lower
limit
    //} else {
        // if (x < grid[0][n-1].x) {*jj=n+1; return;} // exceeds lower
limit
        // if (x > grid[0][ 1].x) {*jj=n+1; return;} // exceeds upper
limit
    //};

    jl=0;

```

```

    ju=n+1;// n intervals, (n+1) division points
    jm=jStart;
    if (x > grid[0][jStart].x)
        jl=jm;
    else
        ju=jm;

    while (ju-jl > 1) {
        jm=(ju+jl) >> 1; // division by 2
        if (x > grid[0][jm].x)
            jl=jm;
        else
            ju=jm;
    }
    *jj=jl;
}

// det. what is the lower vertical point number on grid, closest to 'y'
// we assume that grid's vertical points are ascending ordered, in
// order to skip seeting 'ascnd'
// we assume theat grid has max. 65537 vertical points, which is a
// reasonable assumption
// to speed up the search, we start from the position indicated by
// 'jStart' instead of the middle of the interval [0,n+1]
void locatey_2(GRIDDEF** grid,
               unsigned short n, // total number of grid's
               vertical points
               float y,
               unsigned short *jj, // location of 'x'
               unsigned short jStart)
{
    unsigned short ju,jm,jl;
    //bool ascnd=1;
    //ascnd=(grid[0][n-1].x > grid[0][1].x); // determines how the values
    // of grid[0][.] are ordered; in our case, ascending(1)
    // if(ascnd) {
        if (y > grid[n-1][0].y) {*jj=n+1; return;} // exceeds upper
        limit
        //if (y < grid[ 1][0].y) {*jj=n+1; return;} // exceeds lower
        limit
        if (y < grid[ 0][0].y) {*jj=n+1; return;} // exceeds lower
        limit
    //} else {
        // if (x < grid[0][n-1].x) {*jj=n+1; return;} // exceeds lower
        limit
        // if (x > grid[0][ 1].x) {*jj=n+1; return;} // exceeds upper
        limit
    //};

    jl=0;
    ju=n+1;// n intervals, (n+1) division points
    jm=jStart;
    if (y > grid[jStart][0].y)
        jl=jm;

```

```

else
    ju=jm;

    while (ju-jl > 1) {
        jm=(ju+jl) >> 1; // division by 2
        if (y > grid[jm][0].y)
            jl=jm;
        else
            ju=jm;
    }
    *jj=jl;
}

```

```

void locatex(GRIDDEF** grid, unsigned long n, float x, unsigned long
*jj) /*02Feb2010*/
//void locatex(GRIDDEF grid[ngp1+1][ngp2+1], unsigned long n, float x,
unsigned long *jj)
{
    unsigned long ju,jm,jl;
    int ascnd;

    jl=0;
    ju=n+1; // n intervals, (n+1) division points
    ascnd=(grid[0][n-1].x > grid[0][1].x); // determines how the values
of grid[0][.] are ordered; in our case, ascending(1)
    /* 07Mar2010*/
    if(ascnd) {
        if (x > grid[0][n-1].x) {*jj=n+1; return;} // exceeds upper
limit
        if (x < grid[0][ 1].x) {*jj=n+1; return;} // exceeds lower
limit
    } else {
        if (x < grid[0][n-1].x) {*jj=n+1; return;} // exceeds lower
limit
        if (x > grid[0][ 1].x) {*jj=n+1; return;} // exceeds upper
limit
    };

    while (ju-jl > 1) {
        jm=(ju+jl) >> 1; // division by 2
        if (x > grid[0][jm].x == ascnd)
            jl=jm;
        else
            ju=jm;
    }
    *jj=jl;
}

```

```

void locatey(GRIDDEF** grid, unsigned long n, float y, unsigned long
*jj) /*02Feb2010*/

```

```

//void locatey(GRIDDEF grid[ngp1+1][ngp2+1], unsigned long n, float x,
unsigned long *jj)
{
    unsigned long ju,jm,jl;
    int ascnd;

    jl=0;
    ju=n+1;// n intervals, (n+1) division points
    ascnd=(grid[n-1][0].y > grid[1][0].y); // determines how the values
of grid[.][0] are ordered; in our case, ascending(1)
    /* 07Mar2010*/
    if(ascnd) {
        if (y > grid[n-1][0].y) {*jj=n+1; return;} // exceeds upper
limit
        if (y < grid[ 1][0].y) {*jj=n+1; return;} // exceeds lower
limit
    } else {
        if (y < grid[n-1][0].y) {*jj=n+1; return;} // exceeds lower
limit
        if (y > grid[ 1][0].y) {*jj=n+1; return;} // exceeds upper
limit
    };

    while (ju-jl > 1) {
        jm=(ju+jl) >> 1;
        if (y > grid[jm][0].y == ascnd)
            jl=jm;
        else
            ju=jm;
    }
    *jj=jl;
}

```

lprobe60.h.

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "float.h" // needed for FLT_MAX, FLT_MIN

/*include<sgidefs.h>*/

// #define FLOAT float // in order to use <windows.h> for goto(X,Y),
cursor moving on screen

// #define xButtonOffset 0.15
// #define yButtonOffset 0.08
//
// #define SWITCHBOXx -0.07
// #define SWITCHBOXy -0.85
// #define EXITBOXx 0.84
// #define EXITBOXy SWITCHBOXy

```

```

// #define PRINTBOXx 0.65
// #define PRINTBOXy SWITCHBOXy

int cursorY; // used for info text display
float maxVe = -FLT_MAX; // max. electron velocity [m/s]
float minVe = FLT_MAX; // =1E37min. electron velocity [m/s]
float maxVi = -FLT_MAX; // max. ion velocity [m/s]
float minVi = FLT_MAX; // =1E37min. ion velocity [m/s]
float wpdtMax=FLT_MAX; //estimate max. possible timestep parameter

unsigned int hitIons, hitElectrons; // num of particles that hit the
probe in 'dti' time interval
// average current on a dti/dte time step
float avgc_dti ,avgc_dte ;
float ion_charge, elec_charge; // latest discharge on the probe

#define EPS0 8.85e-12
#define EPSA103 10.44e-12
// #define pi 3.14159265359
float pi= 3.14159265359; // to use <windows.h> for goto(X,Y) position
cursor on screen

// #define ngp1 74 // num of grid points on the vertical direction (OY)
int ngp1; // num of grid points on the vertical direction
// #define ngp2 74 // num of grid points on the horizontal direction
(OX)
#define ngp2 (4*18+1) // num of grid points on the horizontal
directionn (OX)
// this form is required to divide both
the probe length & domain length in an integer num. of cells

// #define ngp2 201 // instructor's
int ncpr; // input parameter: num. of cells per probe radius
int noRows15; // will replace the first 15*ncpr rows, set by Markusic

// #define ngp ngp1*ngp2 //5476 = ngp1 x ngp2 = 74 x 74
unsigned long int ngp; /* 02 Feb, 2010 */

// #define numpart 40000 // ! will be exceeded in INJECT_PART
// #define numpart 300000 // increased from 100,000 to 300,000 when
running m_ion/m_elec=100
#define numpart 299000 // increased from 100,000 to 300,000 when
running m_ion/m_elec=100
// #define numpart 450000 // instructor's

#define boltz 1.38e-23
#define RAD 0.00378864
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32

```

```

#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define NRANSI
#define NR_END 1
#define FREE_ARG char*

#define NPTS 1000005 // ! # of data points
// #define NPTS 20000000 // instructor's

// #define V_TABLE_RES 1000
#define V_TABLE_RES numpart
// #define V_TABLE_RES 1000000
// #define V_TABLE_RES 685849 // at electron #685850 convergence cannot
// be reached in 10000 iterations

/** Environment Variables */
// #define black 0
// #define white 15
// #define gray 50
// #define gray2 56
// #define lightgray 56
// #define darkorange 51
// #define midblue 52
// #define red 53
// #define gold 55

/* Global Variables */

typedef struct {
    float x,y,q_dens,Ey,Ex,phi;
}GRIDDEF;

typedef struct {
    float x,y,z,r,vx,vy,vz,Ex,Ey,q,m;
    // unsigned long cellx,celly;
    unsigned short cellx,celly; // 0...65537
}PARTICLEDEF;

typedef unsigned long int unlong;

// typedef struct {
//
//     float x1PositionUP;
//     float x2PositionUP;
//     float y1PositionUP;
//     float y2PositionUP;
//
//     char ButtonText[20];
//
//     short TextColor;

```

```

//      short aBorderColorUp;
//      short bBorderColorUp;
//      short cBorderColorUp;
//      short dBorderColorUp;
//
//      short aBorderColorDown;
//      short bBorderColorDown;
//      short cBorderColorDown;
//      short dBorderColorDown;
//
//      short FillColorUp;
//      short FillColorDown;
//
//  } PushButton;

/* GLOBAL VARIABLES */

/* PARTICLE GLOBAL VARIABLES */

PARTICLEDEF *ion,*electron;

/* GRID/GEOMETRY GLOBAL VARIABLES */
long int ngp_vert,ngp_horiz,jprobe,kprobe,jinsul,kinsul;
float cell_height1;
float
cell_height2,cell_height3,cell_height4,cell_height5,cell_height6;
float
cell_height,cell_width,r_probe,r_insul,l_probe,l_insul,radius,length;
//GRIDDEF grid[ngp1][ngp2];
GRIDDEF **grid;// = new GRIDDEF* [ngp1]; /* 02 Feb, 2010 */
// GRIDDEF grid[][ngp2];// = new GRIDDEF* [ngp1]; /* 02 Feb, 2010 */

/* FIELD SOLVER GLOBAL VARIABLES */
int
test1,test2,test3,ldab,*piv,n1,m1,ldb,upper_width,lower_width,info,nrhs
;
// long int jprobe,kprobe,kinsul;
//float
band1[ngp],band2[ngp],band3[ngp],band4[ngp],band5[ngp],*abd,*b,*phi;
float *band1,*band2,*band3,*band4,*band5,*abd,*b,*phi; /* 02 Feb, 2010
*/

/* PLASMA PARAMETER GLOBAL VARIABLES */
float T,n,m_ion,q_ion,m_elec,q_elec,m_part,plasma_pot;
float mi_me_ratio; // to be entered by the user

/* INJECTION SETUP GLOBAL VARIABLES */
long int
injsuperi,injsupere,superii_init,superii_back,superii_front,superii_side,
superei_init,
superei_back,superei_front,superei_side,rancount,elec_iter;
/* 25 Mar 2010 */

```



```

float superii_back_FLOAT, superei_back_FLOAT, superii_front_FLOAT,
    superei_front_FLOAT, superii_side_FLOAT, superei_side_FLOAT;

/*    long int super_beami, ion_sub_inject;    Those are for
<inject_beam.c>    */
float supere, superi, sigma, sigmae, sigmai, wpdt;
float h;    // not used in calculations; set in 'parameter.cpp'
float
wp, dt, superi_init, superi_back, superi_front, superi_side, supere_init, supe
re_back, supere_front, supere_side, *v_table_e, *v_table_i, height, width, *ra
ndom_num, dti, dte;

/*    float beam_vel, gamma_beam, flux_beam, numbeamperstep, super_beam;
Those are for <inject_beam.c>    */


/*    COUNTING GLOBAL VARIABLES    */
long int
tom, numelec, numion, t_count, t_count2, num, ioncount, ioniter, ion_move_count
, ecounter;

/*    GRAPHICS GLOBAL VARIABLES    */
long int view, page;

/*    BOUNDARY GLOBAL VARIABLES    */
float V; // [V] Potential (Fig. 2.4); to be entered by the user
float
current, VOLUME, avecurrent, curri, curre, avecurri, avecurre, ins_charge;

/*    TESTING VARIABLES    */
long int t1, t2;
float *vxdata;
float percentAnimation; //percent of particles displayed in
animation, 0.0, ..., 100.0 ; 0.0 means NO animation

unsigned long int x, y, z, c, n2;

//--- BINS -----
typedef struct {
    float r, v;
}GRIDDEF_T;

#define noBins 100
#define outputFrequency 100

/**    Prototypes    */
//float random_number(long int* seed);
//ulong msmith_ran();
//void ReadInBigArrays(void);
//void gridgen(GRIDDEF array[ngp1][ngp2]);
//void param();

```

```

//void initialize2(PARTICLEDEF particle[numpart],PARTICLEDEF
particle2[numpart]);
//void initialize(PARTICLEDEF particle[numpart],long int *num_part,long
int init_num,float super_init,float *v_table,float m,float q);
//void initialize_beam(PARTICLEDEF particle[numpart],long int
*num_part,long int init_num,float super_init,float m,float q);
//void make_velocity_table(void);
//void injecte(PARTICLEDEF particle[numpart]);
//void injecti(PARTICLEDEF particle[numpart]);
//void inject(PARTICLEDEF particle[numpart],long int *num_part,float
super_back,float super_front,
//      float super_side,long int superinj_back,long int
superinj_front,
//      long int superinj_side,float *v_table,float m,float
q);
//void qweight(GRIDDEF array[ngp1][ngp2],PARTICLEDEF
particle[numpart],long int number);
//void make_LU(GRIDDEF array[ngp1][ngp2]);
//void e_field(GRIDDEF array[ngp1][ngp2]);
//void fweight(GRIDDEF array[ngp1][ngp2],PARTICLEDEF
particle[numpart],long int number);
//void boundary (PARTICLEDEF particle[numpart],long int *num_part,float
q);
///

```

```
//
```

make_LU.cpp.

```
//#include "lprobe60.h"
//#include "lapack.cpp"

/*    This function computes the LU decomposition of the band matrix
generated
    by differencing Poisson's Equation on the grid */
void make_LU(GRIDDEF** array) /* 02Feb, 2010 */
//void make_LU(GRIDDEF array[ngp1][ngp2])
//void make_LU(GRIDDEF array[ngp1+1][ngp2+1])
{
    FILE *abddata, *abddata2;
    int info;
    long int
    i, j, k, jmax, kmax, count, counter1, counter2, counter3, counter4, counter5;

    abddata=fopen("abddata.dat", "w+");
    abddata2=fopen("abddata2.dat", "w+");

    /*    MAKE BANDS    */

    jmax=ngp_vert;
    kmax=ngp_horiz;
    counter1=upper_width-1;
    counter2=0;
    counter3=-1;
    counter4=0;
    counter5=0;

    /*    ****    */
    /*    Initialize all non-relevant positions to zero    */
    for(i=0; i < ngp; i++){
        band1[i]=band2[i]=band3[i]=band4[i]=band5[i]=0.0;
    }

    for (j=0; j < jmax; j++){
        for(k=0; k < kmax; k++){

            if(counter1<(ngp-1)){
                counter1++;
            }

            if(counter2<(ngp-1)){
                counter2++;
            }
        }
    }
}
```

```

        counter4++;counter5++;

        /* y < r_probe */
        if(j <= jprobe){ // at probe level(on vertical)
            if (j==0 && k==1){ // first cell (on lower boundary):
lower-right point
                counter4=0;
            }
            if(j==1 && k==0){ // first cell (on left boundary):
upper-left point
                counter5=0;
            }
            if( k < kprobe){ // in front of the probe
                if(j==0){ // Lower boundary */
                    if(k==0) { // domain lower-left corner
                        band1[counter1]=0.0;
                        band2[counter2]=0.0;
                        band4[counter4]=0.0;
                        band5[counter5]=0.0;
                    }
                    else { // lower boundary in front of the
probe, excluding the lower-left corner
                        band1[counter1]=pi*cell_width;

                        band2[counter2]=pi*array[j+1][k].y*array[j+1][k].y/(4.0*cell_widt
h);

                        band4[counter4]=pi*array[j+1][k].y*array[j+1][k].y/(4.0*cell_widt
h);

                        band5[counter5]=0.0;
                    }
                }
            }
            else{ // Rest to top of probe:
                if(k==0) { // the left boundary at probe
tip level
                    band1[counter1]=0.0;
                    band2[counter2]=0.0;
                    band4[counter4]=0.0;
                    band5[counter5]=0.0;
                }
                else{ // domain in front of the tip probe

                    band1[counter1]=2.0*pi*cell_width*(array[j][k].y+(array[j+1][k].y
-array[j][k].y)/2.0)/(array[j+1][k].y-array[j][k].y);

                    band2[counter2]=pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
                    -(array[j][k].y-(array[j][k].y-
array[j-1][k].y)/2.0)*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0))/cell_width;

```

```

        band4[counter4]=pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
                        -(array[j][k].y-(array[j][k].y-
array[j-1][k].y)/2.0)*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0))/cell_width;

        band5[counter5]=2.0*pi*cell_width*(array[j][k].y-(array[j][k].y-
array[j-1][k].y)/2.0)/(array[j][k].y-array[j-1][k].y);
    }
}
else { /* on and inside probe */

    band1[counter1]=band2[counter2]=band4[counter4]=band5[counter5]=0
.0;

}

} // above the probe level:
else if (j>jprobe && j<(jmax-1)){ // redundant ? strictly above
the prove & strictly below the upper boundary
    if((k > 0) && (k < (kmax-1))){ // away from the left
boundary & away from the right boundary

band1[counter1]=2.0*pi*cell_width*(array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)/(array[j+1][k].y-array[j][k].y);

band2[counter2]=pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width;

band4[counter4]=pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width;

band5[counter5]=2.0*pi*cell_width*(array[j][k].y-(array[j][k].y-
array[j-1][k].y)/2.0)/(array[j][k].y-array[j-1][k].y);
    }
    // on the left OR right boundary
    else if(k==0){ // on the left boundary
        band1[counter1]=0.0;
        band2[counter2]=0.0;
        band4[counter4]=0.0;
        band5[counter5]=0.0;

    }
    else { // on the right boundary
        band1[counter1]=2.0*pi*(
array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0
)*cell_width/(array[j+1][k].y-array[j][k].y);
        band2[counter2]=0.0;

```

```

        band4[counter4]=pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
                        -(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0)*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0))/cell_width;
        band5[counter5]=2.0*pi*( array[j][ngp_horiz-
1].y-(array[j][k].y-array[j-1][k].y)/2.0 )*cell_width/(array[j][k].y-
array[j-1][k].y);
    }
    } // i.e. (j<=jprobe OR j>=(jmax-1): on the upper OR the lower
boundary, (above the probe level ?)
    else{
        /*band1[counter1]=0.0;*/
        band2[counter2]=0.0;
        band4[counter4]=0.0;
        band5[counter5]=0.0;
    }
}
}
/*for(i=0;i<ngp;i++){
    printf("%ld %e      %e      %e      %e
    %e\n",i,band1[i],band2[i],band3[i],band4[i],band5[i]);
}*/

/*    Diagonal    */
for (j=0;j < jmax;j++){
    /*    y < r_probe        */
    if(j <= jprobe){
        for(k=0;k < kmax;k++){
            counter3++;
            if(k >= 0 && (k < kprobe)){
                if(j==0){
                    if(k==0){
                        band3[counter3]=-1.0;
                    }
                    else{
                        band3[counter3]=-
pi*(cell_width+(array[j+1][k].y-array[j][k].y)*(array[j+1][k].y-
array[j][k].y)/(2.0*cell_width));
                    }
                }
            }
            else {
                if(k==0){
                    band3[counter3]=-1.0;
                }
                else{
                    band3[counter3]=-
(2.0*pi*cell_width*(array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)/(array[j+1][k].y-array[j][k].y) +
pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)

```

```

-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width +
pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width +
2.0*pi*cell_width*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0)/(array[j][k].y-array[j-1][k].y));
        }
    }
    }
    else{
        band3[counter3]=-1.0;
    }
}
/*      y > r_probe */
else if(j<(jmax-1)){
    for(k=0;k < kmax;k++){
        counter3++;

        if(k > 0 && (k < (kmax-1)) && (j<(jmax-1))){
            band3[counter3]=-
(2.0*pi*cell_width*(array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)/(array[j+1][k].y-array[j][k].y) +
pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width +
pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width +
2.0*pi*cell_width*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0)/(array[j][k].y-array[j-1][k].y));

        }
        else if(k==0){
            band3[counter3]=-1.0;
        }
        else{
            band3[counter3]=-
(2.0*pi*cell_width*(array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)/(array[j+1][k].y-array[j][k].y) +
pi*((array[j][k].y+(array[j+1][k].y-
array[j][k].y)/2.0)*(array[j][k].y+(array[j+1][k].y-array[j][k].y)/2.0)
-(array[j][k].y-(array[j][k].y-array[j-1][k].y)/2.0)*(array[j][k].y-
(array[j][k].y-array[j-1][k].y)/2.0))/cell_width +
2.0*pi*cell_width*(array[j][k].y-(array[j][k].y-array[j-
1][k].y)/2.0)/(array[j][k].y-array[j-1][k].y));
        }
    }
}

```

```

        }
    }
    else{
        for(k=0;k < kmax;k++){
            counter3++;
            band3[counter3]=-1.0;
        }
    }
}
/*for(i=0;i<ngp;i++){
    printf("%ld %f    %f    %f    %f\n",i, band1[i], band2[i], band3[i], band4[i], band5[i]);
}*/

/*    Form band vector to be submitted to the band solver    */
count=-1;
for(i=0;i<ngp;i++){
    for(j=1;j<=(2*lower_width+upper_width+1);j++){
        count++;
        if(j==lower_width+1)            abd[count]=band1[i];
        else if(j==(lower_width+upper_width))
            abd[count]=band2[i];
        else if(j==(lower_width+upper_width+1))
            abd[count]=band3[i];
        else if(j==(lower_width+upper_width+2))
            abd[count]=band4[i];
        else if(j==(lower_width+upper_width+2+lower_width-1))
            abd[count]=band5[i];
        else    abd[count]=0.0;
    }
}

printf("Going into LU factorization...\n");

/*    Call LAPACK subroutine to perform the LU factorization    */
sgbtrf_(&m1,&n1,&lower_width,&upper_width,abd,&ldab,piv,&info);

if (info==0){
    printf("LU decomposition successful...\n");
}

else{
    printf("LU decomposition failed...\n");
    system("PAUSE");
    exit(1);
}

fwrite(abd,sizeof(float),count,abddata);
fclose(abddata);

```



```

fwrite(piv, sizeof(int), ngp, abddata2);
fclose(abddata2);
}

```

make_velocity_table.cpp.

```

// #include "lprobe60.h"
// #include "erfc.cpp"

static float erff(float x) {
    /* estimates erfc(x) valid for 8 < x < 100 */
    if ((x>8.0) && (x<100.0))
        {return (1.0-(float)erfc8_sum((float)x));}
    else
        {return (1.0 -(float)erfc8((float)x));};
}; // erff

void make_velocity_table(void)
{
    long int i,num_div;
    FLOAT accuracy,delta,deltay,v,p,p_exact;

    num_div = (long int) V_TABLE_RES;
    deltay=1.0/((FLOAT)num_div);
    accuracy=5.0e-6;
    v= -1.0;

    /*      Electrons      */

    for(i=1;i < num_div;i++){
        v_table_e[i-1] = 0.0;
        p=deltay * ((FLOAT)i);
        printf("%f \n",p);
        do {
            v=v + 1.0;

            p_exact=sqrt(2.0 / pi) * pow(m_elec / (boltz * T *
11588.7),1.5)
                * (sqrt(pi / 2.0) * pow(boltz * T *11588.7 / m_elec,1.5)
                *erff(sqrt(m_elec / (2.0 * boltz * T * 11588.7)) * v)
                - boltz * T * 11588.7 * v / m_elec * exp(-m_elec * v * v
                / (2.0 * boltz * T * 11588.7)));

            delta=fabs(p-p_exact);
        } while(delta > accuracy);
        printf("%1d  %1f\n",i,v);
        v_table_e[i-1] = v;
    }

    printf("Done creating electron velocity table...\n");
}

```

```

/*      IONS */
v=-0.01;

for(i=1;i< num_div;i++){

    p=deltay * ((FLOAT)i);
    do {
        v=v + 0.01;

        p_exact=sqrt(2.0 / pi) * pow((m_ion ) / (boltz * T *
11588.7),1.5)
        * (sqrt(pi / 2.0) * pow(boltz * T *11588.7 / (m_ion
),1.5)
        *erff(sqrt((m_ion ) / (2.0 * boltz * T * 11588.7)) * v)
        - boltz * T * 11588.7 * v / (m_ion ) * exp(-(m_ion ) * v
* v
        / (2.0 * boltz * T * 11588.7)));

        delta=fabs(p-p_exact);

    }while(delta >= accuracy);
    printf("%1d  %1f\n",i,v);
    v_table_i[i-1]=v;

}

printf("Done creating ion velocity table...\n");
}

```

make_velocity_table_v2.cpp.

```

static float erff(float x) {
// http://sector7.xor.aps.anl.gov/programming/gsl/html/Error-Function.html
    float erf = (float)gsl_sf_erf((double) x);
    return erf;
    /** estimates erfc(x) valid for 8 < x < 100 */
    //if ((x>8.0) && (x<100.0))
    //{return (1.0-(float)erfc8_sum((float)x));}
    //else
    //{return (1.0 -(float)erfc8((float)x));};
}; // erff

void make_velocity_table(void)
{

int converge;
long int i,num_div,temp;          // LONG_MAX = 2,147,483,647
float  accuracy,prob_delta,p_calc,p_exact,delta_v,v,error;

unsigned long int numIter, maxIter,maxNumIter;

```

```

maxIter = 30000; // do not allow perform more than 'maxIter'
iterations to reach convergence

/* Initialize parameters */

num_div=V_TABLE_RES;
prob_delta=1.0/((float)num_div);
accuracy=0.1*prob_delta;          /* 10 percent of spacing */
p_exact = 0.0;
v = 0.0;
maxNumIter = 0;

/* Electrons */
printf("\nElectrons 1...%d", num_div);
for(i=1; i < num_div; i++){
    //printf("\r%d", i);
    //std::cout << "\r" << i;

    temp = (long) v;    /* The next two lines set the numbers past the
decimal point to zero. */
    v = (float) temp;   /* There is probably a more compact way to do
this. */
    p_exact+=prob_delta;
    if (p_exact > 1.0) p_exact = 1.0; // due to arithmetic round up,
p_exact may exceed 1.0
    delta_v = 1.0;
    converge = 0;
    numIter = 0;

    do {
        numIter++;

        p_calc=sqrt(2.0 / pi) * pow(m_elec / (boltz * T * 11588.7),1.5)
            * (sqrt(pi / 2.0) * pow(boltz * T *11588.7 / m_elec,1.5)
            *erff(sqrt(m_elec / (2.0 * boltz * T * 11588.7)) * v)
            - boltz * T * 11588.7 * v / m_elec * exp(-m_elec * v * v
            / (2.0 * boltz * T * 11588.7)));
        error = p_exact - p_calc;

        if (fabs(error) < accuracy) {
            converge = 1;
        }
        else if (error > 0){
            v+=delta_v;
        }
        else {
            v-=delta_v;
            delta_v*=1.0e-1;
        }
    }while((!converge)&&(numIter<maxIter));    // i.e. converge == 0,
i.e. fabs(error) >= accuracy

    if (numIter >= maxIter) {

```

```

        printf("\n\n**** Routine 'make_velocity_table(void)'
*****\n");
        printf("Convergence at electron #%ld could not be reached
in %ld iterations\n", i, maxIter);
        printf("Try to increase 'maxIter' or decrease
V_TABLE_RES(%ld) in 'lprobe60.h' \n", V_TABLE_RES);
        printf("\nThe program will stop\n");
        system("PAUSE");
        exit(0);
    }

    v_table_e[i-1]=v;

    if (v > maxVe) maxVe = v;
    if (v < minVe) minVe = v;

    if (numIter >maxNumIter) maxNumIter = numIter;
}; // for i
v_table_e[num_div-1]= v_table_e[num_div-2] + v_table_e[num_div-2] -
v_table_e[num_div-3]; /* Avoid singularity v = infinity when
p_exact = 1 */

printf("\nMax. number of iterations performed to achieve convergence:
%ld\n",maxNumIter);
printf("Done creating electron velocity table...\n");

/* Reset parameters */
p_exact = 0.0;
v = 0.0;
maxNumIter = 0;

/* Ions */
printf("Ions 1...%d",num_div);
for(i=1;i < num_div;i++){
    //printf("\r%d",i);
    //std::cout << "\r" << i;

    temp = (long) v; /* The next two lines set the numbers past the
decimal point to zero. */
    v = (float) temp; /* There is probably a more compact way to do
this. */
    p_exact+=prob_delta;
    if (p_exact > 1.0) p_exact = 1.0; // due to arithmetic round up,
p_exact may exceed 1.0
    delta_v = 1.0;
    converge = 0;
    numIter = 0;

    do {
        numIter++;

```

```

        p_calc=sqrt(2.0 / pi) * pow((m_ion ) / (boltz * T *
11588.7),1.5)
        * (sqrt(pi / 2.0) * pow(boltz * T *11588.7 / (m_ion ),1.5)
        *erff(sqrt((m_ion ) / (2.0 * boltz * T * 11588.7)) * v)
        - boltz * T * 11588.7 * v / (m_ion ) * exp(-(m_ion ) * v * v
        / (2.0 * boltz * T * 11588.7)));
        error = p_exact - p_calc;

        if (fabs(error) < accuracy) {
            converge = 1;
        }
        else if (error > 0){
            v+=delta_v;
        }
        else {
            v-=delta_v;
            delta_v*=1.0e-1;
        }
    }while((!converge)&&(numIter<maxIter));    // i.e. converge == 0,
i.e. fabs(error) >= accuracy

    if (numIter >= maxIter) {
        printf("\n\n**** Routine 'make_velocity_table(void)'
*****\n");
        printf("Convergence at ion #%ld could not be reached in %ld
iterations\n", i, maxIter);
        printf("Try to increase 'maxIter' or decrease
V_TABLE_RES(%ld) in 'lprobe60.h' or \n", V_TABLE_RES);
        printf("\nThe program will stop\n");
        system("PAUSE");
        exit(0);
    }

    v_table_i[i-1]=v;

    if (v > maxVi) maxVi = v;
    if (v < minVi) minVi = v;

    if (numIter >maxNumIter) maxNumIter = numIter;

}; // for i
v_table_i[num_div-1]= v_table_i[num_div-2] + v_table_i[num_div-2] -
v_table_i[num_div-3];    /* Avoid singularity v = infinity when
p_exact = 1 */

printf("\nMax. number of iterations performed to achieve convergence:
%ld\n",maxNumIter);
printf("Done creating ion velocity table...\n");

}

```

mover.cpp:

```

#include "lprobe60.h"
extern void SetColor(int ForgC);

/* This function moves the particles */

void mover(PARTICLEDEF particle[numpart], long int num_part, float
timestep)
{
    long int i;
    float accelx, accelr, theta;

    ///// plot the particle[0] trajectory
    //i=0;
    //    // old position
    //    float x1 = particle[i].x;    float y1 = particle[i].y;
    //
    //    accelx=particle[i].q * particle[i].Ex / particle[i].m;
    //    accelr=particle[i].q * particle[i].Ey / particle[i].m;
    //    theta=atan2(particle[i].y,particle[i].z);
    //    particle[i].vx=particle[i].vx + accelx * timestep;
    //    particle[i].vy=particle[i].vy + accelr * timestep * sin(theta);
    //    particle[i].vz=particle[i].vz + accelr * timestep * cos(theta);
    //    //new position:
    //    particle[i].x=particle[i].x + particle[i].vx * timestep; //
horizontal
    //    particle[i].y=particle[i].y + particle[i].vy * timestep; //
vertical
    //    particle[i].z=particle[i].z + particle[i].vz * timestep; // depth
    //    particle[i].r=sqrt(particle[i].y*particle[i].y +
particle[i].z*particle[i].z);
    //
    //    // new position
    //    float x2 = particle[i].x;    float y2 = particle[i].y;
    //    cpmove(x1,y1);
    //    cpdraw(x2,y2);
    //
    //    cppt(1,&x1,&y1,9); // graphic symbol #9
    //    cpdraw(7); // Yellow
    //    cppt(1,&x2,&y2,9); // graphic symbol #9
    //
    //    cprect(x1, x2, // X1, X2 : the horizontal range of the
rectangle.
    //          y1,y2); //Y1, Y2 : the vertical range of the
rectangle.

    /***** DEBUG *****/
    float x0,y0,z0;
    double vx,vy,vz,accely,accelz;
    double A0,V0;

    for(i=0;i < num_part;i++){

        //if ((particle[i].cellx==0) &&(particle[i].celly==0)) {

```

```

        //      printf("particle #%d located at cell (0,0) ?\n",i);
        //      printf("x=%f, y=%f,
z=%f\n",particle[i].x,particle[i].y,particle[i].z);
        //      printf("Ex=%f, Ey=%f,
q=%f\n",particle[i].Ex,particle[i].Ey,particle[i].q);
        //      printf("Vx=%f, Vy=%f,
Vz=%f\n",particle[i].vx,particle[i].vy,particle[i].vz);
        //      printf("radius(y,z)=%f, radius domain =
%f",particle[i].r,radius);
        //      system("PAUSE");
        //}

        ///***** DEBUG *****/
        ///** are there trapped particles ? ***/
        ///x0=particle[i].x; y0=particle[i].y; z0=particle[i].z;
        ///*****/

        accelx=particle[i].q * particle[i].Ex / particle[i].m; // page 48,81
in Markusic
        accelr=particle[i].q * particle[i].Ey / particle[i].m; // Ey is, in
fact, Er ??
        theta=atan2(particle[i].y,particle[i].z);

        //x = x0 + V0*dt + (a0/2)*dt*dt
        accely = accelr * sin(theta);
        accelz = accelr * cos(theta);
        vx=particle[i].vx + accelx * timestep;
        vy=particle[i].vy + accely * timestep;
        vz=particle[i].vz + accelz * timestep;

        x0=particle[i].x;
        y0=particle[i].y;
        z0=particle[i].z;

        particle[i].x=particle[i].x + (vx +accelx/2.0* timestep)*
timestep; // horizontal
        particle[i].y=particle[i].y + (vy +accely/2.0* timestep) *
timestep; // vertical
        particle[i].z=particle[i].z + (vz +accelz/2.0* timestep) *
timestep; // depth
        particle[i].vx=vx;
        particle[i].vy=vy;
        particle[i].vz=vz;

        //  if ((x0-particle[i].x)*(x0-particle[i].x)+
        //      (y0-particle[i].y)*(y0-particle[i].y)+
        //      (z0-particle[i].z)*(z0-particle[i].z) >
cell_width*cell_width) {

        //  V0 = sqrt((vx- accelx * timestep)*(vx- accelx * timestep) +
        //      (vy- accely * timestep)*(vy- accely * timestep) +
        //      (vz- accelz * timestep)*(vz- accelz * timestep));
        //A0 = sqrt(accelx*accelx+accely*accely+accelz*accelz);
        //float wpdt_loc = wp*(sqrt(V0*V0+2*A0*cell_width)-V0)/A0; // dte

```

```

        //if (timestep == dti) wpdt_loc /= 15.0; // dti

        //SetColor(2); // light green
// gotoXY(0,45+cursorY);
        //if (wpdt_loc < wpdt) {
        //    printf("!!!! Please decrease parameter wpdt, in
'parameter.txt' to approx. ");
        //    SetColor(15); // bright white
        //    printf("%f",wpdt_loc);
        //}
        //else // the estimated wpdt may slightly bigger than we already
use
        //    printf("!!!! Please decrease parameter wpdt, in
'parameter.txt'");
        //SetColor(8); // gray
        //    printf("\nThe program will stop.\n");
        //    system("PAUSE");
        //exit(0);
// }

goto skipOriginal;
    particle[i].vx=particle[i].vx + accelx * timestep;
    particle[i].vy=particle[i].vy + accelr * timestep * sin(theta);
    particle[i].vz=particle[i].vz + accelr * timestep * cos(theta);

    //new position:  $x = x_0 + v_0 \cdot dt + a_0 \cdot dt \cdot dt$  ? Shouldn't be
(a0/2)*dt*dt ???
    particle[i].x=particle[i].x + particle[i].vx * timestep; //
horizontal
    particle[i].y=particle[i].y + particle[i].vy * timestep; // vertical
    particle[i].z=particle[i].z + particle[i].vz * timestep; // depth
skipOriginal:
    particle[i].r=sqrt(particle[i].y*particle[i].y +
particle[i].z*particle[i].z);

    ///***** DEBUG detect trpped (non-movig)
particles*****/
    // if ((x0==particle[i].x) && ( y0==particle[i].y) &&
(z0=particle[i].z)) {
        //    particle[i]=particle[i];
    // }

}

}

```

output_data.cpp.

```

#include "lprobe60.h"

```



```

//void output_data(PARTICLEDEF particle1[numpart],PARTICLEDEF
particle2[numpart],long int numpart1,long int numpart2,float me,float
mi)
void output_data(PARTICLEDEF particle1[],PARTICLEDEF particle2[],long
int numpart1,long int numpart2,float me,float mi)
{
    FILE *currdata;
    float nde,ndi;
    long int i2;

    currdata=fopen("current_q100.dat.csv","a");
    // the average current over 'outputFrequency' iterations of 'dti' time
    step
    // when outputFrequency=1, the current has the 'instant' value
    N*q_particle)/dti,
    // where N is the num. of particles that hit the probe in the
    dti time interval
    avecurrent=current/(outputFrequency*dti); // this is the current we
    need to display
                                                    // to
    increase its value, 'outputFrequency' value may be reduced;
    see'lprobe60.h'
    avecurre=curre/(outputFrequency*dti);avecurri=curri/(outputFrequency*dt
    i);

    //// why we divide by 'outputFrequency' ? it does not make sense
    //avecurrent=current/dti; // this is the current we need to display:
    the algebraic sum of 'curre+curri'
    //avecurre=curre/dti; // the current generated by the impact of
    electrons with probe
    //avecurri=curri/dti; // the current generated by the impact of ions
    with probe

    // no. of particles that generated the above currents are: curre/q_elec
    & curri/q_ion

    /*    COMPUTE NUMBER DENSITY    */

    nde=ndi=0.0;

    for(i2=0;i2 < numpart1;i2++){
        if(particle1[i2].r <= radius){ // is the particle inside the
    probe ?
        nde=nde + particle1[i2].m / me;
        }
    }
    nde=nde / (pi*radius*radius*length);

    for(i2=0;i2 < numpart2;i2++){
        if(particle2[i2].r <= radius){
        ndi=ndi + particle2[i2].m / mi;
        }
    }
}

```

```

        ndi=ndi / (pi*radius*radius*length);

        if (t_count2 == 1){
            fprintf(currdata,"    Potential[V], t[micro-sec],electrons
in domain,ions in domain,Electrons number density,Ions number
density>Total current[mA],Electron current[mA],Ion current[mA]\n \n");
        }

        //fprintf(currdata,"%f, %e, %ld, %ld, %e, %e, %e, %e,
%le\n",V,t_count2*(dti*1000000.0),numelec,numion,nde,ndi,avecurre*100
0.0,avecurre,avecurre);
        fprintf(currdata,"%f, %e, %ld, %ld, %e, %e, %e, %e,
%le\n",V,t_count2*(dti*1000000.0),numelec,numion,nde,ndi,avecurre*100
0.,avecurre*1000.,avecurre*1000.); /* 07Mar2010 */
        fclose(currdata);
        //printf(" current: %f [mA]",avecurre*1000.0);
        //printf(" current: %f [mA]",avecurre*1000.); /* 07Mar2010 */
        printf("%6.3f",avecurre*1000.); /* 07Mar2010 */

        t_count=0;
        current=curri=curre=0.0;

        hitElectrons = hitIons = 0;
    }
}

```

parameter.cpp:

```

//#include "lprobe60.h"

/*This function sets initial plasma conditions and computes plasma
parameters to be used throughout the rest of the simulation */

void param()
{
    float avveli,avvele,gammai,gammae;

    float numiperstep_back,numiperstep_front,

    numiperstep_side,numeperstep_back,numeperstep_front,numeperstep_side,fl
uxi_back,fluxi_front,
        fluxi_side,fluxe_back,fluxe_front,fluxe_side,afront,aback,aside;

    /*    GEOMETRY    */
    FILE *paramFile = fopen("param.txt","r");

    fscanf(paramFile,"%f",&percentAnimation);
    while (fgetc(paramFile) != '\n'); // read the comment on the line

    // probe radius [m]

```

```

/*r_probe=1.25e-4;*/
r_probe=0.0; //1.25e-4-3.0*(0.00003125);
    fscanf(paramFile,"%f",&r_probe);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line
/*r_insul=2.0*r_rprobe;*/

//!*****
// Ian.22, 2010
float PROBE_ASPECT_RATIO = 0.0; //45.92;    // probe aspect ratio
// probe length [m]
    fscanf(paramFile,"%f",&PROBE_ASPECT_RATIO);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line
l_probe = PROBE_ASPECT_RATIO * r_probe;
    fscanf(paramFile,"%d",&ncpr); // num. of cells per probe radius
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

/*****      NUMBER OF PARTICLES TO INJECT AT EACH micro-sec      *****/
//for ions
superii_init=40000;
fscanf(paramFile,"%ld",&superii_init);
while (fgetc(paramFile) != '\n');    // read the comment on the line

superii_back=2;
fscanf(paramFile,"%ld",&superii_back);
while (fgetc(paramFile) != '\n');    // read the comment on the line

superii_front=2;
fscanf(paramFile,"%ld",&superii_front);
while (fgetc(paramFile) != '\n');    // read the comment on the line

superii_side=4;
fscanf(paramFile,"%ld",&superii_side);
while (fgetc(paramFile) != '\n');    // read the comment on the line

//superii_init=40000;
//superii_back=2*3;
//superii_front=2*3;
//superii_side=4*3;

// for electrons
superei_init=40000;
fscanf(paramFile,"%ld",&superei_init);
while (fgetc(paramFile) != '\n');    // read the comment on the line

superei_back=5;
fscanf(paramFile,"%ld",&superei_back);
while (fgetc(paramFile) != '\n');    // read the comment on the line

superei_front=5;
fscanf(paramFile,"%ld",&superei_front);
while (fgetc(paramFile) != '\n');    // read the comment on the line

```

```

superei_side=8;
fscanf(paramFile,"%ld",&superei_side);
while (fgetc(paramFile) != '\n'); // read the comment on the line

//*****
// probe length [m]
l_insul=1.0*l_probe; // same/different length as probe (collecting
surface)
/*l_probe=0.00287 + 0.5 * 0.00287;
l_insul=1.0*l_probe;*/

// computational domain limits
length=4.0*l_probe; // length of the computational domain on
OX(horizontal) axis (see boundary.cpp)
// domain OX limits: [-length,0] (see
boundary.cpp)
length=2.0*(l_probe+l_insul);

// a first estimation for domain radius
float ratio3 = 0.00395705 / 0.00003125; // = (domain radius)/ (probe
radius)
//radius=0.00395705; // radius of the computational domain
//radius = 0.7*length; // * (0.0039570499/0.0057399999)
length*0.68938152768957365312846085589653
radius = ratio3 * r_probe; // = OY&OZ axis domain vertical (& 3D depth)
limit: +/- radius; used in boundary.cpp, charge_weight.cpp,
//

boundary_beam.cpp, force_weight.cpp

//height=width=2.0 * radius; // 'height' & 'width' variables not used
but below, to calculate VOLUME
//radius = 0.7*length; // * (0.0039570499/0.0057399999)
length*0.68938152768957365312846085589653

ioniter=70; // not used
ioncount=0; // not used
ion_move_count=100; // not used
ecounter=0; // not used
view=1; // not used

// electrical charges
T=0.0; //2.0; // [eV] temperature
fscanf(paramFile,"%f",&T);
while (fgetc(paramFile) != '\n'); // read the comment on the
line

n=0.0; //1.0e15; //1.0e9[cm(-3)]
fscanf(paramFile,"%f",&n);
while (fgetc(paramFile) != '\n'); // read the comment on the
line
n = n *1.e6; // conversion [cm(-3)] --> [m(-3)]

```

```

m_ion=0.0;//1.673e-27;
    fscanf(paramFile,"%f",&m_ion);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

q_ion=0.0;// = 1.6e-19;
    fscanf(paramFile,"%f",&q_ion);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

//m_elec=0.0;// =9.11e-31;
//    fscanf(paramFile,"%f",&m_elec);
//    while (fgetc(paramFile) != '\n');    // read the comment on the
line

/*****
/*  mi/me = any_ratio entered by the user */
*****/
m_elec = m_ion/mi_me_ratio;

q_elec=0.0;//-1.6e-19;
    fscanf(paramFile,"%f",&q_elec);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

wpdt=0.0;//0.009 [sec];
    fscanf(paramFile,"%f",&wpdt);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

plasma_pot=0.0;
    fscanf(paramFile,"%f",&plasma_pot);
    while (fgetc(paramFile) != '\n');    // read the comment on the
line

//float ratio1 = 15;    // = (close-to-probe domain radius)/ (probe
radius) =
//                //    num. of horizontal strips in the close-to-
probe domain
h=6.9*sqrt((T*11588.7)/(n*1.0e-6)) * 0.01;    // not used in
calculations ?

//float dh=(radius-(ngp1-1)*r_probe)/(10+5*(ngp1-20));    // height
increase rate in the linear domain
//float ratio2 = (19*0.00003125+10*0.00000588)/0.00003125;    // (linar
domain radius) / (probe radius)
float dh = (0.00000588/0.00003125)*r_probe;    // 0.18816*r_probe

// max. number of row cells that can be added vertically
int M;
for (int N=1;N<=15;N++) {
    M=ngp2-N-2*ncpr;
    //dh =length-r_probe*(2+(float)(N+M)/ncpr);
    dh =radius-r_probe*(2+(float)(N+M)/ncpr);

```

```

        if (dh <=0.0) break;
        dh = 2*dh/M/(M+1);
        noRows15=N;
};

M = ngp2-noRows15-2*ncpr;
//dh =length-r_probe*(2+(float)(noRows15+M)/ncpr);
dh =radius-r_probe*(2+(float)(noRows15+M)/ncpr);
dh = 2*dh/M/(M+1);
ngp1 = ncpr+ncpr+noRows15+M;

/* 15 rows 1...15 */      cell_height1=r_probe ;// ncpr*15 rows of
height cell_height1/ncpr
// linearly spaced cells
/* 'ncpr' rows      16 */ cell_height2=cell_height1 +dh;      // equally
spaced at cell_height2/ncpr
/* 'ncpr' rows      17 */ cell_height3=cell_height2 +dh;      // equally
spaced at cell_height3/ncpr
/* 'ncpr' rows      18 */ cell_height4=cell_height3 +dh;      // equally
spaced at cell_height4/ncpr
/* 'ncpr' rows      19 */ cell_height5=cell_height4 +dh;      // equally
spaced at cell_height5/ncpr
/* the rest up to ncpr*73 */ cell_height6=cell_height5 +dh; // =
cell_height5 + 0.00000588

/* the increase rate of cell's height, up to ncpr*73 */ cell_height6=dh;
// = cell_height5 + 0.00000588
/* !!!!!!!!!!!!!!!!!!!!! routine 'e_field' uses 'cell_height6'
!!!!!!!!!!!!!!!!!!!!!!!! */

        radius = r_probe+ // probe
                r_probe+ // insulator
                noRows15*(r_probe/ncpr)+ // cell height as for
probe & insulator
                //cell_height2 + cell_height3 + cell_height4 +
cell_height5;// dh increase rate in cell height
                M*r_probe/ncpr + M*(M+1)/2*dh;

        height=width=2.0 * radius; // 'height' & 'width' variables not
used but below, to calculate VOLUME

        // ngp2=4*N+1 form is required to divide both the probe length &
domain length in an integer num. of cells
        cell_width = length/(ngp2-1); // ngp2=ngp_horiz
        //cell_width = 2*(l_probe+l_insul)/(ngp2-1); // ngp2=ngp_horiz

// insulator radius [m]
r_insul=r_probe + cell_height1;

/* TIME STEPS */
wp=2.0*pi*9000.0*sqrt(n*1.0e-6); // (omega)plasma frequency:
1.7882259e+9

```

```

elec_iter=15;
dt=wpdt/wp;dte=dt;dti=15.0*dte;      // dte = 5e-12[sec] timestep
electrons                                     // dti = 7.5e-11[sec]
timestep ions; used to calculate the total current (see boundary.cpp)

///// Lab report page 10: (1) dt = 1.0e-11 [sec]
//dt=1.0e-11;dte=dt;dti=15.0*dte;

t_count=0;

sigmae=sqrt((boltz*T*11588.7)/m_elec);
sigmai=sqrt((boltz*T*11588.7)/m_ion);

/* Boundary Fluxing Setup */

// the super-particle weighting (Thesis, page 58)
afront=4.0* radius * radius;
aside=2.0 * radius * length;
aback=4.0 * radius * radius - pi*r_insul*r_insul; // subtract the non-
flow area

avveli=sqrt((8.0*boltz*T*11589.7)/(pi*m_ion)); // Eq. (4.7)
avvele=sqrt((8.0*boltz*T*11589.7)/(pi*m_elec)); // Eq. (4.7)
gammai=n*avveli/4.0;                          // Eq. (4.6)
gammae=n*avvele/4.0;                          // Eq. (4.6)

fluxi_back=gammai*aback;
fluxe_back=gammae*aback;
fluxi_front=gammai*afront;
fluxe_front=gammae*afront;
fluxi_side=gammai*aside;
fluxe_side=gammae*aside;

numiperstep_back=fluxi_back*dti;
numeperstep_back=fluxe_back*dte;
numiperstep_front=fluxi_front*dti;
numeperstep_front=fluxe_front*dte;
numiperstep_side=fluxi_side*dti;
numeperstep_side=fluxe_side*dte;

///// Original
//superi_back=numiperstep_back/((float)superii_back);
//supere_back=numeperstep_back/((float)superei_back);
//superi_front=numiperstep_front/((float)superii_front);
//supere_front=numeperstep_front/((float)superei_front);
//superi_side=numiperstep_side/((float)superii_side);
//supere_side=numeperstep_side/((float)superei_side);

/* 25 Mar 2010 */

```

```

superi_back=numiperstep_back/((dte*superii_back)/1.0e-6); // num. inj.
part. at each ION time step: (dte*superii_back)/1.0e-6
supere_back=numeperstep_back/((dte*superei_back)/1.0e-6); // num. inj.
part. at each ELECTRON time step: (dte*superei_back)/1.0e-6
superi_front=numiperstep_front/((dte*superii_front)/1.0e-6); // idem
supere_front=numeperstep_front/((dte*superei_front)/1.0e-6); // idem
superi_side=numiperstep_side/((dte*superii_side)/1.0e-6); // idem
supere_side=numeperstep_side/((dte*superei_side)/1.0e-6); // idem

/*****      NUMBER OF PARTICLES TO INJECT AT EACH electron/ion TIME
STEP, respectively *****/
// (not used anymore as integer variables but as FP variables as input
parameters in 'inject_part')
superii_back = superii_back_FLOAT = (dte*superii_back)/1.0e-6;
superei_back= superei_back_FLOAT= (dte*superei_back)/1.0e-6;
superii_front= superii_front_FLOAT= (dte*superii_front)/1.0e-6;
superei_front=superei_front_FLOAT= (dte*superei_front)/1.0e-6;
superii_side= superii_side_FLOAT= (dte*superii_side)/1.0e-6;
superei_side= superei_side_FLOAT= (dte*superei_side)/1.0e-6;

/* Initial Loading Injection Setup */
// VOLUME not used anywhere else: perhaps the volume of a rectangular
box minus the volumes of probe & insulator
VOLUME=length*height*width - (pi* r_probe * r_probe * l_probe) - (pi*
r_insul * r_insul * l_insul);
superi_init=n * ((pi* radius * radius * length)- // vol. domain
                (pi* r_probe * r_probe * l_probe) - // vol. probe
                (pi* r_insul * r_insul * l_insul)) // vol.
insulator
                / ((float)superii_init);
supere_init=n * ((pi* radius * radius * length)-
                (pi* r_probe * r_probe * l_probe) - (pi* r_insul * r_insul *
l_insul))
                / ((float)superei_init);
injsuperi=superii_init;
injsupere=superei_init;

/* Parameteters needed for field solver */
jprobe=((long int)(r_probe/(cell_height1/ncpr))); // no. of cells
covering half of probe width(i.e. above 0X=0) : 1

// grid point vertical limit for probe
kprobe=((long int)((length-l_probe-l_insul)/cell_width)); // no. of
cells in front of the probe : 45

// grid point horizontal limit for probe tip

// <----- length ----->
// -----=====HHHHHHHHHHHHHHHHHH
// <-- l_probe --><-- l_insul -->

```



```
fclose(paramFile);

}
```

plot_1.cpp:

```
// Default Color Representation:
www.cse.iitb.ac.in/~cs101/Year2003/resources/notesPgplot/chapter5.html

void PGEX30(float xMax) { // bins
    //          int max_bin,i;
    cpgenv(0.0,xMax,0.0,1.0,0,-2);

    // Draw first a grid at low brightness(14=dark gray)

    cpgsci(14); // Dark Gray
    cpgbox("G",0.0,0,"G",0.0,0); // automatic

    // and then a frame and axes at full brightness.
    cpgsci(5); // cyan
    cpgbox("ABCTSN",0.0,0,"ABCTSNV",0.0,0); // automatic

    // Call PGLAB to label the graph in a different color (3=green).
    cpgsci(3); // green
    cpgsci(1); // white
    cpglab("velocity(W)","f(W) ","Velocity distribution during
injection");
}; // PGEX30

void PGEX31(float tMax) {
    cpgenv (0.0,tMax,0.0,100000.0,0,-2); // Lab-report.pdf (page
12)
    //cpgenv(0.0,tMax,30000.0,(0.0+numpart),0,-2); //! Lab-
report.pdf (page 12)

    // Draw first a grid at low brightness(14=dark gray)
    cpgsci(14); // Dark Gray
    cpgbox("G",0.0,0,"G",0.0,0); // automatic
    // and then a frame and axes at full brightness.
    cpgsci(5); // cyan
    cpgbox("ABCTSN",0.0,0,"ABCTSNV",0.0,0); // automatic

    // Call PGLAB to label the graph in a different color (3=green).
    cpgsci(3); // green
    cpgsci(1); // white
    cpglab("time[micro-sec]","no. of particles ","Particle no. during
injection");
    // http://www.starlink.rl.ac.uk/star/docs/sc15.htx/node10.html
};
```

```

//void PGEX32(GRIDDEF grid[ngp1+1][ngp2+1])      {
//void PGEX32(GRIDDEF grid[][ngp2]) {
void PGEX32(GRIDDEF** grid)  {
    // draw the grid
    // (grid.x,grid.y) is the left-down corner of the cell
    // grid[][0],grid[][1],...,grid[][ngp2-1] advances on OX
    // grid[0][],grid[1][],...,grid[ngp1-1][] advances on OY

    static float xPlot[4],yPlot[4];
    float minX,maxX,minY,maxY;
    int nx,ny;

    // determine the (x,y) limits
    minX=maxX=grid[0][0].x;
    minY=maxY=grid[0][0].y;
    for(nx =0;nx<ngp2;nx++)
        for(ny=0;ny<ngp1;ny++) {
            if (grid[ny][nx].x > maxX) maxX = grid[ny][nx].x;
            if (grid[ny][nx].x < minX) minX = grid[ny][nx].x;
            if (grid[ny][nx].y > maxY) maxY = grid[ny][nx].y;
            if (grid[ny][nx].y < minY) minY = grid[ny][nx].y;
        };
    maxX =0.0;
    maxY =radius;
    float d = maxY-minY;
    if ((maxX-minX) > d) d = maxX-minX;

    cpgsci(14); // Dark Gray
    cpghenv(minX,minX+d,minY,minY+d,
            1, // the scales on OX and OY will be equal
            //0, // the scales on OX and OY will not be equal
            0); // draw box and label it with coordinates
    // cpghenv(minX,maxX,minY,maxY,
    //      1, // the scales on OX and OY will be equal
    //      //0); // draw box and label it with coordinates
    cpgbox("AP",0.0,0,"AP",0.0,0); // automatic

    cpgsci(5); // cyan
    cpgbox("G",0.0,0,"G",0.0,0); // automatic

    goto label1; // plot the cells in color bands, as they were generated
    in 'grid.cpp'
    // draw the cells trigonometric-wise
    cpgsci(1); // White
    for(nx =0;nx<ngp2-1;nx++)
        for(ny=0;ny<ngp1-1;ny++) {
            xPlot[0]=grid[ny][nx].x;      yPlot[0]=grid[ny][nx].y;
            xPlot[1]=grid[ny][nx+1].x;
            yPlot[1]=grid[ny][nx+1].y;
            xPlot[2]=grid[ny+1][nx+1].x;
            yPlot[2]=grid[ny+1][nx+1].y;
            xPlot[3]=grid[ny+1][nx].x;
            yPlot[3]=grid[ny+1][nx].y;
            cppline(4,xPlot,yPlot);

```

```

    }

label1:
    cpghlw(1); // linewidth; the thinnest
    // draw probe
    cpghsci(2); // Red

    //xPlot[2]= grid[0][ngp2-1].x;
    xPlot[2]= -l_insul;
    yPlot[2] = r_probe;

    //
    // grid[0][0]      +-----+ grid[0][ngp2-1]
    // |      +-----+. insul. |
    // +-----+--- probe ---+-----+ (0,0)
    //      \.
    xPlot[0]= -(l_probe+l_insul);
    yPlot[0] = 0.0; // = grid[0][0].y = grid[0][ngp2-1].y ;

    //xPlot[1]= grid[0][ngp2-1].x; yPlot[1] = grid[0][ngp2-1].y;
    //xPlot[3]= -l_insul;          yPlot[3] = r_insul;
    cpghrect (xPlot[0], xPlot[2], yPlot[0], yPlot[2]);

    // draw insulator
    cpghsci(4); // Blue

    xPlot[2]= 0.0; // = grid[0][ngp2-1].x;
    yPlot[2] = r_insul;

    //
    // grid[0][0]      +-----+ grid[0][ngp2-1]
    // |      +-----+. insul. |
    // +-----+--- probe ---+-----+ (0,0)
    //      /
    xPlot[0]= -l_insul;
    yPlot[0] = 0.0; // = grid[0][0].y = grid[0][ngp2-
1].y;
    //xPlot[1]= grid[0][ngp2-1].x; yPlot[1] = grid[0][ngp2-1].y;
    //xPlot[3]= -l_insul;          yPlot[3] = r_insul;
    cpghrect (xPlot[0], xPlot[2], yPlot[0], yPlot[2]);

    cpghupdt(); // output the graph

///// Draw first a grid at low brightness(14=dark gray)
//      cpghsci(14); // Dark Gray
//      cpghbox("G",0.0,0,"G",0.0,0); // automatic
///// and then a frame and axes at full brightness.
//      cpghsci(5); // cyan
//      cpghbox("ABCTSN",0.0,0,"ABCTSNV",0.0,0); // automatic
//
///// Call PGLAB to label the graph in a different color (3=green).
//      cpghsci(3); // green
//      cpghsci(1); // white
//      cpghlab("time[sec]","no. of particles ","Particle no. during
injection");

float index1,index2;

```

```

long int count1,count2;

cpgsci(1); // White
index1=-1.0;
for (count1=0;count1 < ngp1-1;count1++){ // rows
    index1=index1+1.0;
    index2=-1.0;
    for (count2=0;count2 < ngp2-1;count2++){ // columns
//    for (count2=0;count2 <= ngp_horiz;count2++){
        index2=index2+1.0;
        //grid[count1][count2].x=-length+index2*cell_width;
        //grid[count1][count2].y=index1*cell_height1;

                xPlot[0]=grid[count1][count2].x;
yPlot[0]=grid[count1][count2].y;
                xPlot[1]=grid[count1][count2+1].x;
yPlot[1]=grid[count1][count2+1].y;
                xPlot[2]=grid[count1+1][count2+1].x;
yPlot[2]=grid[count1+1][count2+1].y;
                xPlot[3]=grid[count1+1][count2].x;
yPlot[3]=grid[count1+1][count2].y;
                cppline(4,xPlot,yPlot);

    } // for count2
} // for count1
cpgupdt(); // output the graph

}

```

PROBEPIC.cpp.

```

// see "VC project settings.txt" for the right configuration

// I could not remove the error below for DEBUG mode, where some
libraries should be replaced the right ones.
// http://msdn.microsoft.com/en-us/library/aa267384\(VS.60\).aspx
// LINK : warning LNK4098: defaultlib 'LIBCMT' conflicts with use of
other libs; use /NODEFAULTLIB:library
// use /NODEFAULTLIB:libcmt.lib in Project->Properties -> Linker ->
Command Line
//Configuration Properties
//-> General -> CLR Support : NO
//-> General -> Use of ATL : NO
//-> C/C++ -> Preprocessor -> Definitions: WIN32; _DEBUG; _CONSOLE
//-> C/C++ -> Code Generation -> Runtime Library -> /MT
//-> C/C++ -> Precompiled headers: NO
//-> C/C++ -> Advanced: __cdecl
//-> C/C++ -> Command Line: /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D
"_UNICODE" /D "UNICODE" /Gm /EHsc /MT /Fo"Debug\\" /Fd"Debug\vc80.pdb"
/W3 /nologo /c /Zi /Gd /TP /errorReport:prompt

```

```

//-> Linker -> General -> Additional Library Directories: .\lib
//-> Linker -> General -> Incremental linking: YES
//-> Linker -> Input -> Additional dependencies:
//    GrWin.lib
//    pgplot.lib
//    cpghplot.lib
//    mkl_core.lib
//    libmmds.lib (Multi-threaded static library (built with /MD))
//or libmmt.lib (Multi-threaded static library (/MT))
//or libm.lib (Static library)
//or libmdd.lib & libmdd.dll (Debug Multi-threaded dynamic library
(/MDd))
//or libmmd.lib & libmmd.dll (Multi-threaded dynamic library (/MD))
// http://software.intel.com/en-us/articles/libraries-provided-by-
intelr-c-compiler-for-windows-and-intel-parallel-composer/

//
//Inherited:
//    kernel32.lib
//    user32.lib
//    gdi32.lib
//    winspool.lib
//    comdlg32.lib
//    advapi32.lib
//    shell32.lib
//    ole32.lib
//    oleaut32.lib
//    uuid.lib
//    odbc32.lib
//    odbccp32.lib
//
//-> Linker -> Command line:
///OUT:"C:\Documents and
Settings\contul1\Desktop\cpghplot\GrWinTest\Debug\GrWinTest.exe"
/INCREMENTAL /NOLOGO /LIBPATH:".lib" /MANIFEST
/MANIFESTFILE:"Debug\GrWinTest.exe.intermediate.manifest" /DEBUG
/ASSEMBLYDEBUG /PDB:"c:\Documents and
Settings\contul1\Desktop\cpghplot\GrWinTest\debug\GrWinTest.pdb"
/MACHINE:X86 /ERRORREPORT:PROMPT GrWin.lib pgplot.lib cpghplot.lib
mkl_core.lib libmmds.lib kernel32.lib user32.lib gdi32.lib
winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib
oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

#include <iostream>      // for 'cout' in "make_velocity_table_v2.cpp"
#include ".\PGPLOT\cpghplot.h"

#include "lprobe60.h"
//#include <stdlib.h>
/*  Prototypes  */
#include "erfc.cpp"
#include "lapack.cpp"

#include "distrib.cpp"
//extern float random_number(long int* seed);

```

```

// #include "random_number.cpp"
#include "random_number_v1.cpp"
// extern unlong msmith_ran();
// extern void ReadInBigArrays(void);
// extern void gridgen(GRIDDEF array[ngp1][ngp2]);
#include "grid.cpp"
// extern void param();
#include "parameter.cpp"
// extern void initialize2(PARTICLEDEF particle[numpart], PARTICLEDEF
particle2[numpart]);
// extern void initialize(PARTICLEDEF particle[numpart], long int
*num_part, long int init_num, float super_init, float *v_table, float
m, float q);
#include "initialize.cpp"
// extern void initialize_beam(PARTICLEDEF particle[numpart], long int
*num_part, long int init_num, float super_init, float m, float q);
// extern void make_velocity_table(void);

// #include "make_velocity_table.cpp"
#include "make_velocity_table_v2.cpp"

// extern void injecte(PARTICLEDEF particle[numpart]);
// extern void injecti(PARTICLEDEF particle[numpart]);
// extern void inject(PARTICLEDEF particle[numpart], long int
*num_part, float super_back, float super_front,
// float super_side, long int superinj_back, long int
superinj_front,
// long int superinj_side, float *v_table, float m, float
q);
#include "inject_part.cpp"
// extern void qweight(GRIDDEF array[ngp1][ngp2], PARTICLEDEF
particle[numpart], long int number);
#include "charge_weight.cpp"
// extern void make_LU(GRIDDEF array[ngp1][ngp2]);
#include "make_LU.cpp"
// extern void e_field(GRIDDEF array[ngp1][ngp2]);
#include "field_solver_v1.cpp"
// extern void fweight(GRIDDEF array[ngp1][ngp2], PARTICLEDEF
particle[numpart], long int number);
#include "force_weight.cpp"
// extern void boundary (PARTICLEDEF particle[numpart], long int
*num_part, float q);
// #include "boundary.cpp"
#include "boundary_v2.cpp" // see Lab-report.pdf page 10
/* void mover (PARTICLEDEF array[numpart], long int num_part); */
// extern void mover(PARTICLEDEF array[numpart], long int num_part, float
timestep);
#include "mover.cpp"
// extern void graph(GRIDDEF array[ngp1][ngp2], PARTICLEDEF
particle1[numpart], PARTICLEDEF particle2[numpart]);
// extern float gasdev(long int *idum);
/* old random number generator from recipes */
/* float ran1(long int *idum); */

```

```

//extern void output_data(PARTICLEDEF particle1[numpart],PARTICLEDEF
particle2[numpart],long int numpart1,long int numpart2,float me,float
mi);
#include "output_data.cpp"
//extern void locatex(GRIDDEF xx[ngp1][ngp2], unsigned long int n,
float x, unsigned long int *jj);
//extern void locatey(GRIDDEF xx[ngp1][ngp2], unsigned long int n,
float x, unsigned long int *jj);
#include "locate.cpp"
//extern void reset_grid(GRIDDEF array[ngp1][ngp2]);
#include "reset_grid.cpp"
//extern void recombination (PARTICLEDEF heavy_part[numpart],long int
*num_heavy, PARTICLEDEF elec[numpart],long int *num_part);

//#include ".\PGPLOT\cpgplot.h"
#include "plot_1.cpp"
#include <string.h>

//#include <stdio.h>
#include <conio.h>      // needed for 'kbhit'
//#include <system.h>

#include <windows.h> // needed for gotoXY
//using namespace std;

// position cursor on screen; (0,0) is top-left position
// http://www.cplusplus.com/forum/beginner/4234/
// http://www.dreamincode.net/code/snippet921.htm has
complete/extensive example(s)
void gotoXY( short x, short y ) {
HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
COORD position = { x, y };
SetConsoleCursorPosition( hStdout, position );
}

short int whereX()
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    //COORD result;
    if (!GetConsoleScreenBufferInfo(
        GetStdHandle( STD_OUTPUT_HANDLE ),
        &csbi
    ))
        return -1;
    return csbi.dwCursorPosition.X;
}

short int whereY()
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    //COORD result;
    if (! GetConsoleScreenBufferInfo(
        GetStdHandle( STD_OUTPUT_HANDLE ),

```

```

        &csbi
    ))
    return -1;
    return csbi.dwCursorPosition.Y;
}

// http://www.daniweb.com/forums/thread216345.html
//0 = Black      8 = Gray
//1 = Blue       9 = Light Blue
//2 = Green      10 = Light Green
//3 = Aqua       11 = Light Aqua
//4 = Red        12 = Light Red
//5 = Purple     13 = Light Purple
//6 = Yellow     14 = Light Yellow
//7 = White      15 = Bright White
// 16 Black     17 Blue
// Foreground is the text color (range = 0-31)
// Background is the screen color (range = 0-7)

//This will set the foreground color for printing in a console window.
void SetColor(int ForgC)
{
    WORD wColor;
    //We will need this handle to get the current background attribute
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;

    //We use csbi for the wAttributes word.
    if(GetConsoleScreenBufferInfo(hStdOut, &csbi))
    {
        //Mask out all but the background attribute, and add in the
        foreground color
        wColor = (csbi.wAttributes & 0xF0) + (ForgC & 0x0F);
        SetConsoleTextAttribute(hStdOut, wColor);
    }
    return;
}

float xPlot,yPlot,tMax;
double dx,dy;
char string1[10];
char string_t[30],string_pot[30],string_elec[30];

//using namespace std;// for 'cout' in "make_velocity_table_v2.cpp"
char* strBOOL(bool T){
    if(T) return "ON ";
    return "OFF";
};

```



```

/* This is the main program for a two dimensional PIC simulation of a
Langmuir probe. */

int main(void)
{
    // handling keypress (for animation ON/OFF)
    bool animationON=0;
    char key1;

    /* FILE *diagnostic; */
    /* FILE *efdata; */
    long int seed,i,i3,iter,i4,i5,i6;
    /* float *diag_vector; */

    /* Load simulation parameter initialization values */
    printf("Enter mi/me ratio [Hydrogen=1836]: ");
    scanf("%f",&mi_me_ratio);

    param(); // see 'parameter.cpp'
    printf("\n");
    printf("Number density, n[m-3]: %7.2e\n",n);
    printf("Plasma potential [V]: %f\n", plasma_pot);
    printf("Plasma temperature, T[eV]: %f\n", T);
    printf("Probe Aspect Ratio: %5.2f\n", l_probe/r_probe);
    printf("Electron time step [micro-sec]: %8.2g\n", dte*1.0e6);
    printf("Average Current measuring interval : %d x (ion time steps) \n",
    outputFrequency);
    printf("\n");
    cursorY = 5; // manages the info text on the screen (vertically)

    //ngp1 = 74* ncpr; // vertically, no. of grid points will be
multiple of 74
    //ngp1 = (ncpr+ncpr+noRows15* ncpr) +ncpr +ncpr +ncpr+ncpr; //
no. of grid points vertically

    ngp = ngp1*ngp2; //5476 = ngp1 x ngp2 = 74 x 74

    grid= new GRIDDEF* [ngp1]; /* 02 Feb, 2010 */
    for(i=0; i<ngp1; i++) grid[i] = new GRIDDEF[ngp2] ;

    band1= new float[ngp];
    band2= new float[ngp];
    band3= new float[ngp];
    band4= new float[ngp];
    band5= new float[ngp];

    /* FIELD SOLVER GLOBAL VARIABLES */
    test1=0,test2=0,test3=0,ldab=3*ngp2+1,n1=ngp,m1=ngp,ldb=ngp,upper_width
=ngp2,lower_width=ngp2;
    x=521288629,y=362436069,z=16163801,c=1,n2=1131199299;

    /* Large array initialization */

```

```

electron=(PARTICLEDEF *)malloc(numpart * sizeof(PARTICLEDEF));
ion=(PARTICLEDEF *)malloc(numpart * sizeof(PARTICLEDEF));
abd=(float *)malloc((2*lower_width+upper_width+2)*ngp*sizeof(float));
b=(float *)malloc(ngp*sizeof(float));
phi=(float *)malloc(ngp*sizeof(float));
v_table_e=(float *)malloc((V_TABLE_RES+1)*sizeof(float));
v_table_i=(float *)malloc((V_TABLE_RES+1)*sizeof(float));
random_num=(float *)malloc((NPTS+1)*sizeof(float));
piv=(int *)malloc(ngp*sizeof(int));

/*efdata=fopen("efdat30.dat","w+");*/

/* Variable initialization */
numelec=numion=t_count=t_count2=0;
current=0.0;
V=0.0;

/*prefposition(20,5);
prefsize(900,700);40000
vinit("X11");
mapcolor(50,100,100,100);
mapcolor(56,175,175,175);
mapcolor(51,139,69,0);
mapcolor(52,20,20,200);
mapcolor(53,255,0,0);
mapcolor(55,238,201,0); */

/* Initialize array of uniform random numbers */
seed=100;
//random_number(&seed); // void random_number(long int seed)
random_number(seed); // void random_number(long int seed)
rancount=0;

/* Generate computational cells */
printf("Generate computational cells ...");
gridgen(grid); // see 'grid.cpp'
printf(" DONE\n");

/* Generate look-up tables for ion and electron speeds */
printf("Generate look-up tables for %d speeds...", V_TABLE_RES);
make_velocity_table();
//printf(" DONE\n");
printf("Electron speed range [m/s]: %4d ...
%6d", (int)minVe, (int)maxVe);
printf("\n      Ion speed range [m/s]: %4d ...
%6d\n", (int)minVi, (int)maxVi);

```

```

if (maxVe*dte > cell_width){
    SetColor(2); // light green
    printf("\n\n! ! ! ! Please decrease parameter 'wpdt', in
'parameter.txt', to approx. %f\n", wp*cell_width/maxVe);
    printf("The program will stop.\n");
    system("PAUSE");
    exit(0);
}
if (maxVi*dti > cell_width){
    SetColor(2); // light green
    printf("\n\n! ! ! ! Please decrease parameter 'wpdt', in
'parameter.txt', to approx. %f\n", wp*cell_width/maxVi/elec_iter);
    printf("The program will stop.\n");
    system("PAUSE");
    exit(0);
}
// Try to decrease running time by increasing the time step
wpdtMax = wp*cell_width/maxVi/elec_iter;
if (wpdtMax > wp*cell_width/maxVe ) wpdtMax = wp*cell_width/maxVe
;

if (wpdt < wpdtMax ){
    printf("\nTo speed up the simulation, parameter 'wpdt', in
'parameter.txt', \n");
    printf("may be increased to approx. ");
    SetColor(15); // bright white
    printf("%f\n", wpdtMax);
    int kkk =whereY();
    SetColor(7); // white
    system("PAUSE");
    gotoXY(0, kkk-0);
}
//printf("Supere_init = %d \n", superei_init);
//printf("got here 2 %d \n", V_TABLE_RES);

/* Generate an initial uniform fill of the computational domain with
electrons */
printf("Generate an initial uniform fill of the computational domain
with electrons...");
rancount=0;
initialize(electron, &numelec, superei_init, supere_init, v_table_e, m_elec,
q_elec);
printf("\nDONE
\n");

/* Refresh array of uniform random numbers */
seed=101;
//random_number(&seed); // void random_number(long int seed)
random_number(seed); // void random_number(long int seed)

printf("\nGenerate an initial uniform fill of the computational domain
with ions...");
rancount=0;
initialize(ion, &numion, superii_init, superi_init, v_table_i, m_ion, q_ion);

```

```

printf("\nDONE
\n");
//!-----
//!      SET UP BINS TO PUT electrons IN
GRIDDEF_T bins[noBins];
float tMax,min_val,max_val,delta,velocity;
int max_bin;

//! minimum speed
min_val = 0.0;
//! get the maximum speed
max_val = 0.0;
for(i=0;i<numelec;i++){
    velocity= sqrt(pow(electron[i].vx,2) +pow(electron[i].vy,2)
+pow(electron[i].vz,2) );
    if (velocity > max_val) max_val = velocity;
}; // for i

// set the lower limit for each bin
delta = (max_val-min_val)/noBins;  //! mean # of points in a bin
for(i=0;i<noBins;i++){
    bins[i].r = min_val + i*delta;  //! the lower limit of the bin
    bins[i].v = 0.0;                //! shows how many data points fall in it
}; // for i

//! PUT v_table_e DATA IN BINS
distrib(bins,noBins,electron,numelec);

//! get the bin with the maximum number of particles in it
max_bin = 0;
for(i=0;i<noBins;i++){
    if (bins[i].v > bins[max_bin].v) max_bin = i;
}; // for i
//!-----

printf("initial number electrons ==>%ld    initial number ions
==>%ld\n",numelec,numion);

//----- CPGPLOT sequence -----
//! Call PGOPEN to initiate PGPLOT and open the output device; PGOPEN
//! will prompt the user to supply the device name and type. Always
//! check the return code from PGOPEN.
    if (cpgopen("/CGW") < 0) exit(1);
    //if(cpgbeg(0, "/CGW", 1, 1) != 1) exit(EXIT_FAILURE);
//!          /GW:
//!          /CGW:
//!          /GWD: crash
//!          /CGWD: crash

    //!if(PGBEG(0, "/WV", 1, 1) /= 1) STOP !UNIT, FILE, NXSUB, NYSUB
    //!!CALL PGASK(1) ! FLAG=1(TRUE) : 'prompt state' of PGPLOT is set
to ON
    cpgbbuf(); //! Begin saving graphical output commands in an
internal buffer; the

```

```

        ///! commands are held until a matching PGEBUF call
    //cpgask(1); ///! FLAG=1(FALSE) :prompt when plot before plotting on
another page

    //check grid
    PGEX32(grid);    // see 'plot_1.cpp'

    cpgask(1); ///! FLAG=0(FALSE):do not prompt when plot before
plotting on another page
        ///! FLAG=1(TRUE) :do        prompt when plot before
plotting on another page
    //cpgeras();
    cpgsubp(2, 1);    ///! two adjacent panels
    ///!
    ///! Print information about device.
    ///!CALL PGEX0
    ///!
    ///! Call the drawing subroutines

    ///!!      Set manually the color table to default values
    //  cpgscr(0, 0.0, 0.0, 0.3); // changes color index 0 (background)
to dark blue
    //  cpgscr(0, 0.0, 0.0, 0.0); // changes color index 0 (background)
to black
    //cpgscr(5, 0.00, 1.00, 1.00); // cyan
    //cpgscr(1, 1.00, 1.00, 1.00); // white
    //cpgscr(3, 0.00, 1.00, 0.00); // green
    //cpgscr(6, 1.00, 0.00, 1.00); // magenta
    //cpgscr(14, 0.33, 0.33, 0.33); // dark gray
    ///!-----
    ///! select Left Panel
    cpgpanl(2, 1);
    //cpgeras(); ///! Erase all graphics from the current page or
panel.

    PGEX30(max_val);    ///! draw axes

    ///! draw graph of electron speed distribution
    cpgsci(6);          ///! magenta
    for(i=0;i<noBins;i++) {
        xPlot = bins[i].r;
        yPlot = bins[i].v/bins[max_bin].v;
        cpgpt(1,&xPlot,&yPlot,17); // graphic symbol #17
    };

    ///! draw text
    cpgsci(1);          ///! white
    sprintf(string1,"%f", 0.0); // zero seconds
    strcpy(string_t," t = " );
    strcat(string_t,string1);strcat(string_t," sec");
    cpgptxt(2000000.0,0.65,0.0, 0.0,string_t); ///! X, Y, ANGLE,
FJUST, TEXT)

    sprintf(string1,"%d", numelec);

```

```

        strcpy(string_elec,"No. electrons: ");
strcat(string_elec,string1);
        cpgptxt(2000000.0,0.50,0.0, 0.0,string_elec); //! X, Y, ANGLE,
FJUST, TEXT)

//! flush any pending commands to the      output device.
        cpgupdt();
//!-----
-----

/*
iter=12001;
for(i3 = 0;i3 < 18 ; i3++)

diagnostic=fopen("output.dat","wb");
for(i=0;i<V_TABLE_RES;i++){
diag_vector[i]=v_table_e[i];
}
fwrite(diag_vector,sizeof(float),V_TABLE_RES,diagnostic);
fclose(diagnostic);
exit(1);

*/

//printf("LU decomposition of the difference matrix for the electric
field solver...");
make_LU(grid);
printf(" \n");

printf("***** Entering main program *****\n");
printf("Enter potential [V]: "); scanf("%f",&V);
int duration;
printf("Enter the duration of the experiment [micro-sec]: ");
scanf("%d",&duration);
tMax=(float)duration;
iter = duration/(dti*1000000.0); // dt[sec] -> dt[micro-sec]
printf("\nThis simulation will perform %d ion
iterations\n",iter*elec_iter);
printf("The output file is: current_q100.dat.csv ('append' mode)\n");
printf("The program may be stopped at any time by pressing
<CTRL>C\n\n");

        if (percentAnimation <= 1.0e-6)      // animation disabled from
parameter file
                printf("The trajectories animation was disabled (see
'param.txt').\n\n");
        else
                printf("The trajectories animation may be turned ON/OFF by
pressing any key.\n\n");

//printf("Progress: ");
//printf("\n");
//printf("\nInstant current: ----- electrons ; ");

```



```

        cpgptxt(tMax/3.0,75000.0,0.0, 0.0," ions"); // X, Y, ANGLE,
FJUST, TEXT)
        //!-----
-----

        //// old position(s)
// float xIon0_1; float yIon0_1;
// float xIonN_1; float yIonN_1;
// float xElec0_1; float yElec0_1;
// float xElecN_1; float yElecN_1;
// // new position
// float xIon0_2; float yIon0_2;
// float xIonN_2; float yIonN_2;
// float xElec0_2; float yElec0_2;
// float xElecN_2; float yElecN_2;

        animationON = true;
        if ((percentAnimation > 1.0e-6) && (animationON == true)) {
            gotoXY(10,45+cursorY); printf("ON ");
        } else {
            gotoXY(10,45+cursorY); printf("OFF");
        }

        //!***** PERFORM SIMULATION with 'dti' time step
        *****

        // average current on a dti/dte time step
        avgc_dti = 0.0;
        avgc_dte = 0.0;

        for (i = 1;i <= iter; i++){
            if (rancount > (NPTS - outputFrequency)){
                seed=seed + 100;
                //random_number(&seed);
                random_number(seed);
                rancount=0;
            }; // if

            for(i6=0;i6 < elec_iter;i6++){ // elect_iter=15
                reset_grid(grid); // grid[][].q_dens=0.0;
                qweight(grid,electron,numelec); // adjust
grid[][].q_dens

                // locate electron[].cellx/celly; if outside domain, remove
                qweight(grid,ion,numion); // adjust
grid[][].q_dens

                // locate
ion[].cellx/celly; if outside domain, remove
                e_field(grid); // Call Lapack to solve [a]x=b
                // adjust
grid[][].Ex/Ey

                // uses 'cell_height6'
as the value of upper row cells height

```



```

        fweight(grid,electron,numelec); // re-calculate
electron[].Ex/Ey from grid[][].Ex/Ey
        // it adds particles and locate them:
electron[].cellx/celly
        inject(electron,&numelec,
                supere_back,
supere_front,supere_side, // the super-particle weighting (Thesis, page
58)
                // num. of injected particles at
each dte step:
                /* superei_back, */ (unsigned long
int)(superei_back_FLOAT*i)-(unsigned long int)(superei_back_FLOAT*(i-
1)),
                /* superei_front,*/ (unsigned long
int)(superei_front_FLOAT*i)-(unsigned long int)(superei_front_FLOAT*(i-
1)),
                /* superei_side, */ (unsigned long
int)(superei_side_FLOAT*i)-(unsigned long int)(superei_side_FLOAT*(i-
1)),
                v_table_e,m_elec,q_elec); // put
particles on boundaries with Ex=Ey=0
// electron[].q=super_X * q_elec

        mover(electron,numelec,dte); // moves particles with
acc=acc[q,Ex/Ey,m)
        boundary(electron,&numelec,q_elec); // collects the
electron[].q current when on probe

        // the insulator is disabled (comented code)
        //mover(electron,numelec,dte); // moves particles with
acc=acc[q,Ex/Ey,m)

        if ((percentAnimation > 1.0e-6) && (animationON
== true)) {
                // select Left Panel (maxwellian)
                cpgpanl(1, 1);
                dx=length/10.0;
                // some space arround the drawing area
                dy=(grid[ngp_vert-1] [0].y)/10.0; // some
space arround the drawing area
                // world coordinates:
                cpgswin(-length-dx,0.0+dx, // 0x axis
                        -radius-dy, radius+dy);
                // 0y axis

                // plot the electrons(magenta)
                cpgsci(6); //! magenta
                for (unsigned long int count1=0 ; count1 <
numelec ; count1 += 100/percentAnimation)
                {
                        //xPlot = electron[count1].x;
                        //yPlot = electron[count1].y;

```

```

//cpgpt(1,&xPlot,&yPlot,1); //
graphic symbol #1
//cpgpt(1,&xPlot,&yPlot,-1); //
graphic symbol #(-1)

cpgpt(1,&electron[count1].x,&electron[count1].y,-1); // graphic symbol
#(-1)

// http://www.astro.rug.nl/~kuijken/pgplot.html#PGPIXL
}
}

}; // electron iteration

/***** ion iteration *****/
fweight(grid,ion,numion); // re-calculate ion[].Ex/Ey from
grid[][][].Ex/Ey
// it adds new particles and locate them:
ion[].cellx/celly=0
inject(ion,&numion,
        superi_back,superi_front,superi_side, // the
super-particle weighting (Thesis, page 58)
        // num. of injected ions at each dti step:
        /* superii_back, */ (unsigned long
int)(superii_back_FLOAT*i)-(unsigned long int)(superii_back_FLOAT*(i-
1)),
        /* superii_front,*/ (unsigned long
int)(superii_front_FLOAT*i)-(unsigned long int)(superii_front_FLOAT*(i-
1)),
        /* superii_side, */ (unsigned long
int)(superii_side_FLOAT*i)-(unsigned long int)(superii_side_FLOAT*(i-
1)),
        v_table_i,m_ion,q_ion); // put particles on
boundaries with Ex=Ey=0

//
ion[].q=super_X * q_ion
mover(ion,numion,dti); // moves particles with
acc=acc[q,Ex/Ey,m)
boundary(ion,&numion,q_ion); // collects the ion[].q
current when on probe
//mover(ion,numion,dti); // moves particles with
acc=acc[q,Ex/Ey,m)

// moved in boundary.cpp
//gotoXY(17,36); printf("%6d electrons; %f [micro
A]", (int)(curre/q_elec), curre*1.0E6);
//gotoXY(17,37); printf("%6d ions; %f [micro
A]", (int)(curri/q_ion), curri*1.0E6);
/***** Turn animation ON/OFF *****/

//if (System::Threading::Console::KeyAvailable ==
false) animationON++;

```

```

////http://www.computing.net/answers/programming/c-detecting-
keystrokes/8706.html
    //key1 = getche(); // outer test
    //putch('\x8'); // put backspace char

    // http://www.azmeer.info/?p=32
    key1='\00'; // show that the keyboard was not pressed
    if (kbhit()) {
        key1=getch(); // remove the character from
buffer
        //
http://nemesis.lonestar.org/reference/telecom/codes/ascii.html
        if (key1 == '\03') exit(0); // <CTRL><C> was
pressed

        if (animationON) {
            animationON=false;
            gotoXY(10,45+cursorY); printf("OFF");
        } else {
            animationON=true;
            gotoXY(10,45+cursorY); printf("ON ");
        }
        key1='\01'; // show that the keyboard was
pressed
    }
/*****/

// is trajectories aniamtion percentage set in the parameters file ?
if (percentAnimation <= 1.0e-6) goto L1234; // skip display
particle trajectories
// is animation turned ON from the console ?
if(animationON == false) {
    // was the animation just now turned ON/OFF ? If so, erase the
screen
    if (key1=='\01') {
        // select Left Panel (maxwellian/trajectories)
        cpgpanl(1, 1);
        cpgeras();
    }
    goto L1234;
};

    // ***** ions trajectories ***** //

    cpgsch (1.0/10.0); // character drawing size; default = 1.0

    // select Left Panel (maxwellian)
    cpgpanl(1, 1);

    // was the animation just now turned ON/OFF ? If so, erase the
screen
    if (key1=='\01') cpgeras();

    // erase the old graph by, clearing the panel, every 60 iterations
    //if ((i==1) || (i%120==0)) cpgeras();

```

```

        // restore world->screen transformation
        dx=length/10.0;                                // some space
around the drawing area
        dy=(grid[ngp_vert-1][0].y)/10.0; // some space around the
drawing area
        // world coordinates:
        cpgswin(-length-dx,0.0+dx, // 0x axis
                -radius-dy, radius+dy); // 0y axis

        // draw the domain
        cpgsci(1); // White
        cpgmove(0.0,0.0);
        cpgdraw(0.0,+radius);
        cpgdraw(-length,+radius);
        cpgdraw(-length,-radius);
        cpgdraw(0.0,-radius);
        cpgdraw(0.0,0.0);

        // re-draw the probe/insulator 'damaged' by the particles
        // draw the probe (conductor) in Blue
        //
http://www.cse.iitb.ac.in/~cs101/Year2003/resources/notesPgplot/chapter5.html
        cpgsci(4); // Blue
        cpgrect(-l_insul - l_probe, 0.0, // X1, X2 : the horizontal
range of the rectangle.
        //cpgrect(- l_probe, 0.0, // X1, X2 : the horizontal range of
the rectangle.
                -r_probe, r_probe); //Y1, Y2 : the vertical range
of the rectangle.

        // draw the insulator in Dark Gray, if present
        if (l_insul > 0.0) {
            cpgsci(14); // Dark Gray
            //cpgrect(- l_insul, 0.0, // X1, X2 : the horizontal range of
the rectangle.
            // -r_insul, r_insul); //Y1, Y2 : the vertical range
of the rectangle.
            cpgrect(- l_insul, 0.0, // X1, X2 : the horizontal range of the
rectangle.
                    -r_insul, -r_probe); //Y1, Y2 : the vertical range
of the rectangle.
            cpgrect(- l_insul, 0.0, // X1, X2 : the horizontal range of the
rectangle.
                    r_insul, r_probe); //Y1, Y2 : the vertical range
of the rectangle.
        };

        // // plot the electrons(magenta) then the ions(green), 1%
        // cpgsci(6); //! magenta

```

```

//      //cpgsci(4);      //! blue
//      //for (unsigned long int count1=0 ; count1 < numelec ; count1++)
//      //for (unsigned long int count1=0 ; count1 < numelec ; count1 +=
100)
//      for (unsigned long int count1=0 ; count1 < numelec ; count1 +=
100/percentAnimation)
/////      for (unsigned long int count1=0 ; count1 < numpart ; count1 +=
100/percentAnimation)
//      {
//          xPlot = electron[count1].x;
//          yPlot = electron[count1].y;
//          //cpgpt(1,&xPlot,&yPlot,1); // graphic symbol #1
//          //cpgpt(1,&xPlot,&yPlot,-1); // graphic symbol #(-1)
//
//          cpgpt(1,&electron[count1].x,&electron[count1].y,-1); //
graphic symbol #(-1)
//
//          http://www.astro.rug.nl/~kuijken/pgplot.html#PGPIXL
//      }
//

      cpgsci(3);      //! green
      //cpgsci(2);      //! red
      for (unsigned long int count1=0 ; count1 < numelec ; count1 +=
100/percentAnimation)
      //for (unsigned long int count1=0 ; count1 < numpart ; count1 +=
100/percentAnimation)
      {
          //xPlot = ion[count1].x;
          //yPlot = ion[count1].y;
          //cpgpt(1,&xPlot,&yPlot,1); // graphic symbol #1
          //cpgpt(1,&xPlot,&yPlot,-1); // graphic symbol #1

          cpgpt(1,&ion[count1].x,&ion[count1].y,-1); // graphic
symbol #1
      }

      //! flush any pending commands to the      output device.
      cpgupdt();

      cpgsch (1.0); // restore character drawing size to default =
1.0

      goto L1234; //! skip display velocity distribution

//-----
      // select Left Panel (maxwellian)
      cpgpanl(1, 1);
      // restore world->screen transformation
      cpgswin(0.0,max_val,0.0,1.0);
      // erase the old graph by re-drawing it in background color
      cpgsci(0);      //! black
      cpgptxt(2000000.0,0.65,0.0, 0.0,string_t); //! X, Y, ANGLE,
FJUST, TEXT)

```

```

        cpgptxt(2000000.0,0.55,0.0, 0.0,string_pot); //! X, Y, ANGLE,
FJUST, TEXT)
        cpgptxt(2000000.0,0.50,0.0, 0.0,string_elec); //! X, Y, ANGLE,
FJUST, TEXT)
        for(int iPG=0;iPG<noBins;iPG++) {
                xPlot = bins[iPG].r;
                yPlot = bins[iPG].v/bins[max_bin].v;
                cpgpt(1,&xPlot,&yPlot,17); // graphic symbol #17
        };
        //cpgupdt(); // display now !

        // compute and draw the new graph
        distrib(bins,noBins,electron,numelec); //! PUT v_table_e DATA IN
BINS
        //! get the bin with the maximum number of particles in it
        max_bin = 0;
        for(int iPG=0;iPG<noBins;iPG++){
                if (bins[iPG].v > bins[max_bin].v) max_bin = iPG;
        }; // for iPG

        // redraw grid damaged by the graph
        cpgsci(14); // Dark Gray
        cpgbox("G",0.0,0,"G",0.0,0); // grid
        // draw graph of speed distribution
        cpgsci(6); //! magenta
        for(int iPG=0;iPG<noBins;iPG++) {
                xPlot = bins[iPG].r;
                yPlot = bins[iPG].v/bins[max_bin].v;
                cpgpt(1,&xPlot,&yPlot,17); // graphic symbol #17
        };

        cpgsci(1); // white
        strcpy(string1,""); sprintf(string1,"%8.3E", i*dti); // seconds
        strcpy(string_t," t = ") ;
        strcat(string_t,string1);strcat(string_t," sec");
        cpgptxt(2000000.0,0.65,0.0, 0.0,string_t); //! X, Y, ANGLE,
FJUST, TEXT)

        strcpy(string1,""); sprintf(string1,"%6.2f", V);
        strcpy(string_pot," Potential ="); strcat(string_pot,string1);
        cpgptxt(2000000.0,0.55,0.0, 0.0,string_pot); //! X, Y, ANGLE,
FJUST, TEXT)

        sprintf(string1,"%d", numelec);
        strcpy(string_elec,"No. electrons: ");
        strcat(string_elec,string1);
        cpgptxt(2000000.0,0.50,0.0, 0.0,string_elec); //! X, Y, ANGLE,
FJUST, TEXT)

        //! flush any pending commands to the output device.
        cpgupdt();

        //!-----
L1234:

```

```

        /*** ion iteration */
        //fweight(grid,ion,numion);
        //inject(ion,&numion,superi_back,superi_front,superi_side,
        //
superii_back,superii_front,superii_side,v_table_i,m_ion,q_ion);
        //mover(ion,numion,dti);
        //boundary(ion,&numion,q_ion);

//!-----
        //! select Right Panel (ion/electron)
        cpgpanl(2, 1);
        //! restore world->screen transformation
        cpgswin(0.0,tMax,0.0,100000.0);        //! PGEX31 must be modified
too when these values change
        //cpgswin(0.0,tMax,30000.0,(0.0+numpart)); //! Lab-report.pdf
(page 12)
        cpgslw(4); //! line width
        cpgsci(6); //! magenta for electrons
        xPlot=i*(dti*1000000.0); // dt[sec] -> dt[micro-sec]
        yPlot=(float)numelec;
        cpgpt(1,&xPlot,&yPlot,-2);        //! -1, -2 : a dot of size given
by cpgsch(size)
        cpgsci(3); //! green for ions
        yPlot=(float)numion;
        cpgpt(1,&xPlot,&yPlot,-2);        //! -1, -2 : a dot of size given
by cpgsch(size)
        cpgupdt(); //! display now !
//!-----

        t_count++; // will be reset to zero in ' output_data'
        t_count2++; //

        // Progress:
        gotoXY(12,34+cursorY); printf("%6.3f",
i*100.0/iter); // percent
        gotoXY(39,34+cursorY); printf("%6.3f",i*dti*1.0E6);
// time
        //gotoXY(9,37); printf("%6d electrons; %f [micro
A]");
        //gotoXY(19,38); printf("%6d ions; %f [micro
A]");

        //gotoXY(67,44+cursorY); // average current,
displayed in 'output_data.cpp'
        gotoXY(67-4,44-3+cursorY); // average current,
displayed in 'output_data.cpp'

        //if (percentAnimation <= 1.0e-6) // animation disabled from
parameter file
        // printf("\rANIMATION:OFF %f %% electrons:%d
ions:%d",i*100.0/iter,numelec,numion);
        //else
        // printf("\rANIMATION:%s %f %% electrons:%d
ions:%d",strBOOL(animationON),i*100.0/iter,numelec,numion);

```

```

        if (t_count == outputFrequency || t_count2==1) {
output_data(electron,ion,numelec,numion,m_elec,m_ion);
        // clean the displaying field
        gotoXY(18,41+2+cursorY); printf("
",numelec);
        gotoXY(18,42+2+cursorY); printf("      ",numion);
    }

    gotoXY(24,41+2+cursorY); printf("%6d",numelec);
    gotoXY(24,42+2+cursorY); printf("%6d",numion);

}; // i=0..iter-1

//}; // V=0... 18*0.5

/*fclose(efdata);*/
/*vexit();*/
exit(1);

}

//#undef IA
//#undef IM
//#undef AM
//#undef IQ
//#undef IR
//#undef NTAB
//#undef NDIV
//#undef EPS
//#undef RNMX
//#undef NRANSI

```

random_number.cpp.

```

//#include "lprobe60.h"

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7

```



```

#define RNMX (1.0-EPS)

float random_number(long *idum)
{
    int j;
    long k;
    static long iy=0;
    static long iv[NTAB];
    float temp;

    if (*idum <= 0 || !iy) {
        if (-(*idum) < 1) *idum=1;
        else *idum = -(*idum);
        for (j=NTAB+7;j>=0;j--) {
            k=(*idum)/IQ;
            *idum=IA*(*idum-k*IQ)-IR*k;
            if (*idum < 0) *idum += IM;
            if (j < NTAB) iv[j] = *idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ;
    *idum=IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0) *idum += IM;
    j=iy/NDIV;
    iy=iv[j];
    iv[j] = *idum;
    if ((temp=AM*iy) > RNMX) return RNMX;
    else return temp;
}
#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX

```

random_number_v1.cpp:

```

//#define EXTERN extern
//#include "lprobe60.h"

/* Common Block Declarations */
struct klotz0_1_ {
    float buff[607];
    long int ptr;
};

#define klotz0_1 (*(struct klotz0_1_ *) &klotz0_)

```

```

#define min(a,b) (a<b)?a:b

struct klotz1_1_ {
    float xbuff[1024];
    long int first, xptr;
};

//#define klotz1_1 (*(struct klotz1_1_ *) &klotz1_)

/* Initialized data */
struct {
    long int fill_1[1214];
    long int e_2;
    } klotz0_ = { {0}, 0 };

//struct {
//    float fill_1[1024];
//    long int e_2[2];
//    float e_3;
//    } klotz1_ = { {0}, 0, 0, 0. };

int zufall_(long int n, float *a)
//long int n;
//float *a;
{
    long int buffsz = 607;

    long int left, aptr, bptr, aptr0, i, k, q;
    float t;
    long int nn, vl, qq, k273, k607, kptr;

/* portable lagged Fibonacci series uniform random number */
/* generator with "lags" -273 und -607: */
/* W.P. Petersen, IPS, ETH Zuerich, 19 Mar. 92 */

    aptr = 0;
    nn = n;

L1:
    if (nn <= 0) {
        return 0;
    }

/* factor nn = q*607 + r */
    q = (nn - 1) / 607;
    left = buffsz - klotz0_1.ptr;

```

```

    if (q <= 1) {
/* only one or fewer full segments */

        if (nn < left) {
            kptr = klotz0_1.ptr;
            for (i = 0; i < nn; ++i) {
                a[i + aptr] = klotz0_1.buff[kptr + i];
            }
            klotz0_1.ptr += nn;
            return 0;
        } else {
            kptr = klotz0_1.ptr;
/*pragma _CRI ivdep*/
            for (i = 0; i < left; ++i) {
                a[i + aptr] = klotz0_1.buff[kptr + i];
            }
            klotz0_1.ptr = 0;
            aptr += left;
            nn -= left;
/* buff -> buff case */
            vl = 273;
            k273 = 334;
            k607 = 0;
            for (k = 0; k < 3; ++k) {
/*pragma _CRI ivdep*/
                for (i = 0; i < vl; ++i) {
                    t = klotz0_1.buff[k273+i]+klotz0_1.buff[k607+i];
                    klotz0_1.buff[k607+i] = t - (float) ((long int) t);
                }
                k607 += vl;
                k273 += vl;
                vl = 167;
                if (k == 0) {
                    k273 = 0;
                }
            }
            goto L1;
        }
    } else {
/* more than 1 full segment */

        kptr = klotz0_1.ptr;
/*pragma _CRI ivdep*/
        for (i = 0; i < left; ++i) {
            a[i + aptr] = klotz0_1.buff[kptr + i];
        }
        nn -= left;
        klotz0_1.ptr = 0;
        aptr += left;

/* buff -> a(aptr0) */

```

```

        vl = 273;
        k273 = 334;
        k607 = 0;
        for (k = 0; k < 3; ++k) {
            if (k == 0) {
/*pragma _CRI ivdep*/
                for (i = 0; i < vl; ++i) {
                    t = klotz0_1.buff[k273+i]+klotz0_1.buff[k607+i];
                    a[aptr + i] = t - (float) ((long int) t);
                }
                k273 = aptr;
                k607 += vl;
                aptr += vl;
                vl = 167;
            } else {
/*pragma _CRI ivdep*/
                for (i = 0; i < vl; ++i) {
                    t = a[k273 + i] + klotz0_1.buff[k607 + i];
                    a[aptr + i] = t - (float) ((long int) t);
                }
                k607 += vl;
                k273 += vl;
                aptr += vl;
            }
        }
        nn += -607;

/* a(aptr-607) -> a(aptr) for last of the q-1 segments */

        aptr0 = aptr - 607;
        vl = 607;

        for (qq = 0; qq < q-2; ++qq) {
            k273 = aptr0 + 334;
/*pragma _CRI ivdep*/
            for (i = 0; i < vl; ++i) {
                t = a[k273 + i] + a[aptr0 + i];
                a[aptr + i] = t - (float) ((long int) t);
            }
            nn += -607;
            aptr += vl;
            aptr0 += vl;
        }

/* a(aptr0) -> buff, last segment before residual */

        vl = 273;
        k273 = aptr0 + 334;
        k607 = aptr0;
        bptr = 0;
        for (k = 0; k < 3; ++k) {
            if (k == 0) {
/*pragma _CRI ivdep*/
                for (i = 0; i < vl; ++i) {

```

```

        t = a[k273 + i] + a[k607 + i];
        klotz0_1.buff[bptr + i] = t - (float) ((long int) t);
    }
    k273 = 0;
    k607 += vl;
    bptr += vl;
    vl = 167;
} else {
/*pragma _CRI ivdep*/
    for (i = 0; i < vl; ++i) {
        t = klotz0_1.buff[k273 + i] + a[k607 + i];
        klotz0_1.buff[bptr + i] = t - (float) ((long int) t);
    }
    k607 += vl;
    k273 += vl;
    bptr += vl;
}
}
goto L1;
}
} /* zufall_ */

```

```

long int zufalli_(long int seed)
//long int seed;
{

```

```

    /* Initialized data */

```

```

    long int kl = 9373;
    long int ij = 1802;

```

```

    /* Local variables */
    long int i, j, k, l, m;
    float s, t;
    long int ii, jj;

```

```

/* generates initial seed buffer by linear congruential */
/* method. Taken from Marsaglia, FSU report FSU-SCRI-87-50 */
/* variable seed should be 0 < seed <31328 */

```

```

    if (seed != 0) {
        ij = seed;
    }

```

```

    i = ij / 177 % 177 + 2;
    j = ij % 177 + 2;
    k = kl / 169 % 178 + 1;
    l = kl % 169;

```

```

    for (ii = 0; ii < 607; ++ii) {
        s = 0.;
        t = .5;
        for (jj = 1; jj <= 24; ++jj) {

```

```

        m = i * j % 179 * k % 179;
        i = j;
        j = k;
        k = m;
        l = (l * 53 + 1) % 169;
        if (l * m % 64 >= 32) {
            s += t;
        }
        t *= (float).5;
    }
    klotz0_1.buff[ii] = s;
}
return 0;
} /* zufalli_ */

long int zufallsv_(float *svblk)
//float *svblk;
{
    long int i;

    /* saves common blocks klotz0, containing seeds and */
    /* polong inter to position in seed block. IMPORTANT: svblk must be */
    /* dimensioned at least 608 in driver. The entire contents */
    /* of klotz0 (polong inter in buff, and buff) must be saved. */

    /* Function Body */
    svblk[0] = (float) klotz0_1.ptr;
    /*pragma _CRI ivdep*/
    for (i = 0; i < 607; ++i) {
        svblk[i + 1] = klotz0_1.buff[i];
    }

    return 0;
} /* zufallsv_ */

long int zufallrs_(float *svblk)
//float *svblk;
{
    long int i;

    /* restores common block klotz0, containing seeds and pointer */
    /* to position in seed block. IMPORTANT: svblk must be */
    /* dimensioned at least 608 in driver. The entire contents */
    /* of klotz0 must be restored. */

    klotz0_1.ptr = (long int) svblk[0];
    /*pragma _CRI ivdep*/
    for (i = 0; i < 607; ++i) {
        klotz0_1.buff[i] = svblk[i + 1];
    }

```

```

    }

    return 0;
} /* zufallrs_ */

void random_number(long int seed)
{
    /*long int tom;*/
    // extern long int fischet_(), zufalli_(), normalt_(), zufallt_();

    zufalli_(seed);
    zufall_(NPTS, random_num);

    /*for(tom=0;tom<100;tom++) prlong
    intf("random_number[%d]=%f\n",tom,random_num[tom]);*/

}; //random_number

```

reset_grid.cpp.

```

//#include "lprobe60.h"

void reset_grid(GRIDDEF** array) /* 02Feb.2010*/
//void reset_grid(GRIDDEF array[ngp1+1][ngp2+1])
{
    long int count1,count2;

    /* CLEAN UP GRID */

    for (count1=0;count1 < ngp_vert;count1++){
        for (count2=0;count2 < ngp_horiz;count2++){
            //for (count1=0;count1 <= ngp_vert;count1++){
            // for (count2=0;count2 <= ngp_horiz;count2++){
                array[count1][count2].q_dens=0.0;
            }
        }
    }
}

```

Appendix-II

FORTTRAN Code for PROBEPI New Version

BOUNDARY_v2.F90:

```
! -----
MODULE BOUNDARY_v2
! -----
!   This routine sorts out which particles have crossed a boundary,i.e.
!   the probe or extent of computational domain.  Particles crossing
the
!   probe boundary are counted as current
!
USE LPROBE60_1
USE Console_API ! only for debug
USE MSFLIB

CONTAINS

! -----
SUBROUTINE boundary(particle,num_part,q, textWindow, graphWindow)
! -----
!TYPE (PARTICLEDEF ) particle(0:numpart-1)
TYPE (PARTICLEDEF ) particle(0:)
INTEGER num_part
REAL q
INTEGER textWindow, graphWindow

! - - - local declarations - - -
INTEGER numtemp
  INTEGER :: i,i2
  REAL r_part
  !REAL avgc_dti, avgc_dte ! average current on a dti/dte time step
  LOGICAL resultOK
  REAL prev_avgc_dte, prev_avgc_dti

TYPE (rccoord) crsPos
INTEGER result
CHARACTER key1

! - - - begin - - -
crsPos%row = 1      ! Quickwin text window origin
crsPos%col = 1

! the previous non-zero average values on dti/dte will remain on screen
! but dimmed, in case on the present timestep the probe will not be
hitted
prev_avgc_dte =0.0
prev_avgc_dti =0.0
```



```

if (avgc_dte /= 0.0) prev_avgc_dte = avgc_dte
if (avgc_dti /= 0.0) prev_avgc_dti = avgc_dti

! average current on this dte time step
avgc_dte = 0.0

numtemp = num_part
i2 = -1
DO i = 0, numtemp-1
  i2 = i2+1
  r_part = particle(i2)%r

!      OUT OF COMPUTATIONAL DOMAIN

  IF ((particle(i2)%x > 0.0) .OR. (particle(i2)%x < -length) .OR. &
      (particle(i2)%y > radius) .OR. (particle(i2)%y < -radius)
      .OR. &
      (particle(i2)%z > radius) .OR. (particle(i2)%z < -radius))
  THEN

    ! the particle 'i2' is removed: it is overwritten by the
    last one, *num_part-1
    particle(i2)=particle(num_part-1)
    !particle(i2)%x = particle(num_part-1)%x
    !particle(i2)%y = particle(num_part-1)%y
    !particle(i2)%z = particle(num_part-1)%z
    !particle(i2)%r = particle(num_part-1)%r
    !particle(i2)%vx = particle(num_part-1)%vx
    !particle(i2)%vy = particle(num_part-1)%vy
    !particle(i2)%vz = particle(num_part-1)%vz
    !particle(i2)%q = particle(num_part-1)%q
    !particle(i2)%m = particle(num_part-1)%m
    i2 = i2-1
    num_part = num_part - 1

!      CONTACT WITH PROBE , but not it's tip(!)

    ELSE IF ((r_part <= r_probe) .AND. (particle(i2)%x >= -
(l_probe+l_insul)) .AND. &
      (particle(i2)%x <= -l_insul))
    THEN

      !particle(i2)%x = particle(num_part-1)%x
      !particle(i2)%y = particle(num_part-1)%y
      !particle(i2)%z = particle(num_part-1)%z
      !particle(i2)%r = particle(num_part-1)%r
      !particle(i2)%vx = particle(num_part-1)%vx
      !particle(i2)%vy = particle(num_part-1)%vy
      !particle(i2)%vz = particle(num_part-1)%vz
      !particle(i2)%m = particle(num_part-1)%m
      !particle(i2)%q = particle(num_part-1)%q

```

```

! some additional tests when Animation=OFF may be needed here
! in order to speed-up computations

        result = SETACTIVEQQ (graphWindow)
        !result = FOCUSQQ (graphWindow)
        CALL pgsci(1)    ! White
        CALL pgpt(1,particle(i2)%x,particle(i2)%y,-1) ! graphic symbol
#(-1)
        result = SETACTIVEQQ (textWindow)
        !result = FOCUSQQ (textWindow)

!! BEGIN DEBUG
!CALL WRITE_$ _TO_CONSOLE('Part. ')
!CALL WRITE_numI$_TO_CONSOLE(i2)
!CALL WRITE_$ _TO_CONSOLE(' hit probe')
!!result = SYSTEMQQ ('PAUSE') ! mesage is displayed in Windows console
!key1=GETCHARQQ( ) ! pause
!CALL WRITE_$ _TO_CONSOLE("\n"C) ! CR
!! END DEBUG

! average current on a N*dti time interval
current = current + particle(i2)%q

IF (q == q_elec ) THEN
! average electron current on a N*dti time interval
curre = curre + particle(i2)%q
! average electron current on a dte time step
avgc_dte = avgc_dte +particle(i2)%q

        hitElectrons = hitElectrons +1
        ! INSTANT charge released to probe is highlited ("bright
white")
        ! only when is different from the previous one
        IF (particle(i2)%q == elec_charge) THEN
            txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
        ELSE
            txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
        ENDIF
        CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(60-5), crsPos)
        WRITE(*,100) particle(i2)%q
        elec_charge =particle(i2)%q ! latest electron discharge

ELSE IF (q == q_ion) THEN
! average ion current on a N*dti time interval
curri = curri + particle(i2)%q
! average ion current on a dti time step
avgc_dti = avgc_dti +particle(i2)%q

        hitIons = hitIons +1
        ! INSTANT charge released to probe is highlited ("bright
white")
        ! only when is different from the previous one
        IF (particle(i2)%q == ion_charge) THEN

```

```

        txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
    ELSE
        txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
    ENDIF
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(60-5), crsPos)
    WRITE(*,100) particle(i2)%q
    ion_charge = particle(i2)%q ! latest ion discharge
END IF

! remove particle from simulation
particle(i2)=particle(num_part-1)

i2 = i2-1
num_part = num_part - 1

txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow

CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(20-1), crsPos)
WRITE(*,'(I4,$)') hitElectrons

CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(20-1), crsPos)
WRITE(*,'(I4,$)') hitIons

! This section removed to check on particle trapping
!! CONTACT WITH CYLINDRICAL SURFACE OF INSULATOR
!
! ELSE IF ((r_part <= r_insul) .AND. (particle(i2)%x >-(l_insul-
cell_width))) THEN
!     vzold = particle(i2)%vz
!     vyold = particle(i2)%vy
!
!     particle(i2)%vz = (vzold*(-particle(i2)%z*particle(i2)%z +
particle(i2)%y*particle(i2)%y) &
!
!     -
2.0*vyold*particle(i2)%y*particle(i2)%z)/(particle(i2)%r*particle(i2)%r
)
!
!     particle(i2)%vy = (vyold*(particle(i2)%z*particle(i2)%z -
particle(i2)%y*particle(i2)%y) &
!
!     -
2.0*vzold*particle(i2)%y*particle(i2)%z)/(particle(i2)%r*particle(i2)%r
)
!
!! CONTACT WITH FLAT FACE OF INSULATOR
!
! ELSE IF ((r_part <= r_insul) .AND. (particle(i2)%x >=-(l_insul)))
THEN
!     particle(i2)%vx = -particle(i2)%vx
END IF

END DO ! i

! in case no particles hit the probe

```

```

    result = SETACTIVEQQ (textWindow)

! average current on this dti time step
IF (q == q_ion) THEN ! a ion time step has finished
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(20-1), crsPos)
    WRITE(*, '(I4,$)') hitIons
    IF (hitIons == 0) THEN
        CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(39-6), crsPos)
        WRITE(*, '("      ",$)')
        CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(62-6), crsPos)
        WRITE(*, '("      ",$)')
    ENDIF
ENDIF

! average ion current on a dti time step
CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(39-6), crsPos)
if (avgc_dti /= 0.0) THEN ! ions hit the probe
    txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
    WRITE(*,102) (avgc_dti/dti)*1.0E6
ELSE ! ions did not hit the probe on this time step
    txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
    if (prev_avgc_dti /= 0.0) WRITE(*,102) (prev_avgc_dti/dti)*1.0E6
ENDIF

! average electron current on a dte time step
CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(39-6), crsPos)
if (avgc_dte /= 0.0) THEN ! electrons hit the probe
    txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
    WRITE(*,102) (avgc_dte/dte)*1.0E6
ELSE
    txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
    if (prev_avgc_dte /= 0.0) WRITE(*,102) (prev_avgc_dte/dte)*1.0E6
ENDIF

! ! dimm instant charges
! txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
! ! electron
! CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(60-5), crsPos)
! WRITE(*,100) elec_charge
! ! ion
! CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(60-5), crsPos)
! WRITE(*,100) ion_charge

! restore color for the rest of the routines
txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow

! average current on this dti time step
IF (q == q_ion) avgc_dti = 0.0 ! a ion time step has finished

100 FORMAT(G11.3,$)
102 FORMAT(G12.4,$)

```

END SUBROUTINE boundary

END MODULE BOUNDARY_v2

CHARGE_WEIGHT.F90:

```
! -----
MODULE BOUNDARY_v2
! -----
!   This routine sorts out which particles have crossed a boundary,i.e.
!   the probe or extent of computational domain.  Particles crossing
!   the
!   probe boundary are counted as current
!
USE LPROBE60_1
USE Console_API ! only for debug
USE MSFLIB

CONTAINS

! -----
SUBROUTINE boundary(particle,num_part,q, textWindow, graphWindow)
! -----
!TYPE (PARTICLEDEF ) particle(0:numpart-1)
TYPE (PARTICLEDEF ) particle(0:)
INTEGER num_part
REAL q
INTEGER textWindow, graphWindow

! - - - local declarations - - -
INTEGER numtemp
  INTEGER :: i,i2
  REAL r_part
  !REAL avgc_dti, avgc_dte ! average current on a dti/dte time step
  LOGICAL resultOK
  REAL prev_avgc_dte, prev_avgc_dti

TYPE (rcCOORD) crsPos
INTEGER result
CHARACTER key1

! - - - begin - - -
crsPos%row = 1      ! Quickwin text window origin
crsPos%col = 1

! the previous non-zero average values on dti/dte will remain on screen
! but dimmed, in case on the present timestep the probe will not be
hitted
```

```

prev_avgc_dte =0.0
prev_avgc_dti =0.0
if (avgc_dte /= 0.0) prev_avgc_dte = avgc_dte
if (avgc_dti /= 0.0) prev_avgc_dti = avgc_dti

! average current on this dte time step
avgc_dte = 0.0

numtemp = num_part
i2 = -1
DO i = 0,numtemp-1
    i2 = i2+1
    r_part = particle(i2)%r

!      OUT OF COMPUTATIONAL DOMAIN

    IF ((particle(i2)%x > 0.0) .OR. (particle(i2)%x < -length) .OR. &
        (particle(i2)%y > radius) .OR. (particle(i2)%y < -radius)
    .OR. &
        (particle(i2)%z > radius) .OR. (particle(i2)%z < -radius))
    THEN

        ! the particle 'i2' is removed: it is overwritten by the
last one, *num_part-1
        particle(i2)=particle(num_part-1)
        !particle(i2)%x = particle(num_part-1)%x
        !particle(i2)%y = particle(num_part-1)%y
        !particle(i2)%z = particle(num_part-1)%z
        !particle(i2)%r = particle(num_part-1)%r
        !particle(i2)%vx = particle(num_part-1)%vx
        !particle(i2)%vy = particle(num_part-1)%vy
        !particle(i2)%vz = particle(num_part-1)%vz
        !particle(i2)%q = particle(num_part-1)%q
        !particle(i2)%m = particle(num_part-1)%m
        i2 = i2-1
        num_part = num_part - 1

!      CONTACT WITH PROBE , but not it's tip(!)

        ELSE IF ((r_part <= r_probe) .AND. (particle(i2)%x >= -
(l_probe+l_insul)) .AND. &
                (particle(i2)%x <= -l_insul))
    THEN

        !particle(i2)%x = particle(num_part-1)%x
        !particle(i2)%y = particle(num_part-1)%y
        !particle(i2)%z = particle(num_part-1)%z
        !particle(i2)%r = particle(num_part-1)%r
        !particle(i2)%vx = particle(num_part-1)%vx
        !particle(i2)%vy = particle(num_part-1)%vy
        !particle(i2)%vz = particle(num_part-1)%vz
        !particle(i2)%m = particle(num_part-1)%m

```

```

!particle(i2)%q = particle(num_part-1)%q

! some additional tests when Animation=OFF may be needed here
! in order to speed-up computations

        result = SETACTIVEQQ (graphWindow)
        !result = FOCUSQQ (graphWindow)
        CALL pgsci(1) ! White
        CALL pgpt(1,particle(i2)%x,particle(i2)%y,-1) ! graphic symbol
#(-1)
        result = SETACTIVEQQ (textWindow)
        !result = FOCUSQQ (textWindow)

!! BEGIN DEBUG
!CALL WRITE_$ _TO_CONSOLE('Part. ')
!CALL WRITE_numI$_TO_CONSOLE(i2)
!CALL WRITE_$ _TO_CONSOLE(' hit probe')
!!result = SYSTEMQQ ('PAUSE') ! mesage is displayed in Windows console
!key1=GETCHARQQ( ) ! pause
!CALL WRITE_$ _TO_CONSOLE("\n"C) ! CR
!! END DEBUG

! average current on a N*dti time interval
current = current + particle(i2)%q

IF (q == q_elec ) THEN
! average electron current on a N*dti time interval
curre = curre + particle(i2)%q
! average electron current on a dte time step
avgc_dte = avgc_dte +particle(i2)%q

        hitElectrons = hitElectrons +1
        ! INSTANT charge released to probe is highlited ("bright
white")

        ! only when is different from the previous one
        IF (particle(i2)%q == elec_charge) THEN
                txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
        ELSE
                txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
        ENDIF
        CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(60-5), crsPos)
        WRITE(*,100) particle(i2)%q
        elec_charge = particle(i2)%q ! latest electron discharge

ELSE IF (q == q_ion) THEN
! average ion current on a N*dti time interval
curri = curri + particle(i2)%q
! average ion current on a dti time step
avgc_dti = avgc_dti +particle(i2)%q

        hitIons = hitIons +1
        ! INSTANT charge released to probe is highlited ("bright
white")

```

```

        ! only when is different from the previous one
        IF (particle(i2)%q == ion_charge) THEN
            txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
        ELSE
            txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
        ENDIF
        CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(60-5), crsPos)
        WRITE(*,100) particle(i2)%q
        ion_charge = particle(i2)%q ! latest ion discharge
    END IF

    ! remove particle from simulation
    particle(i2)=particle(num_part-1)

    i2 = i2-1
    num_part = num_part - 1

    txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow

    CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(20-1), crsPos)
    WRITE(*,'(I4,$)') hitElectrons

    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(20-1), crsPos)
    WRITE(*,'(I4,$)') hitIons

! This section removed to check on particle trapping
!! CONTACT WITH CYLINDRICAL SURFACE OF INSULATOR
!
! ELSE IF ((r_part <= r_insul) .AND. (particle(i2)%x >-(l_insul-
cell_width))) THEN
!     vzold = particle(i2)%vz
!     vyold = particle(i2)%vy
!
!     particle(i2)%vz = (vzold*(-particle(i2)%z*particle(i2)%z +
particle(i2)%y*particle(i2)%y) &
!
!     -
2.0*vyold*particle(i2)%y*particle(i2)%z)/(particle(i2)%r*particle(i2)%r
)
!
!     particle(i2)%vy = (vyold*(particle(i2)%z*particle(i2)%z -
particle(i2)%y*particle(i2)%y) &
!
!     -
2.0*vzold*particle(i2)%y*particle(i2)%z)/(particle(i2)%r*particle(i2)%r
)
!
!! CONTACT WITH FLAT FACE OF INSULATOR
!
! ELSE IF ((r_part <= r_insul) .AND. (particle(i2)%x >=-(l_insul)))
THEN
!     particle(i2)%vx = -particle(i2)%vx
    END IF

END DO ! i

```



```

! in case no particles hitted the probe
result = SETACTIVEQQ (textWindow)

! average current on this dti time step
IF (q == q_ion) THEN ! a ion time step has finished
  CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(20-1), crsPos)
  WRITE(*, '(I4,$)') hitIons
  IF (hitIons == 0) THEN
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(39-6), crsPos)
    WRITE(*, '("      ",$)')
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(62-6), crsPos)
    WRITE(*, '("      ",$)')
  ENDIF
ENDIF

! average ion current on a dti time step
CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(39-6), crsPos)
if (avgc_dti /= 0.0) THEN ! ions hitted the probe
  txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
  WRITE(*,102) (avgc_dti/dti)*1.0E6
ELSE ! ions did not hit the probe on this time step
  txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
  if (prev_avgc_dti /= 0.0) WRITE(*,102) (prev_avgc_dti/dti)*1.0E6
ENDIF

! average electron current on a dte time step
CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(39-6), crsPos)
if (avgc_dte /= 0.0) THEN ! electrons hitted the probe
  txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow
  WRITE(*,102) (avgc_dte/dte)*1.0E6
ELSE
  txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
  if (prev_avgc_dte /= 0.0) WRITE(*,102) (prev_avgc_dte/dte)*1.0E6
ENDIF

! ! dimm instant charges
! txtColor = SETTEXTCOLOR(INT2(8)) ! 8=gray
! ! electron
! CALL SETTEXTPOSITION (INT2(38-1+cursorY), INT2(60-5), crsPos)
! WRITE(*,100) elec_charge
! ! ion
! CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(60-5), crsPos)
! WRITE(*,100) ion_charge

! restore color for the rest of the routines
txtColor = SETTEXTCOLOR(INT2(14)) ! 14=yellow

! average current on this dti time step
IF (q == q_ion) avgc_dti = 0.0 ! a ion time step has finished

100 FORMAT(G11.3,$)
102 FORMAT(G12.4,$)

```

END SUBROUTINE boundary

END MODULE BOUNDARY_v2

Console_API.f90.

MODULE Console_API

!USE MSFLIB
!use msfwin

CHARACTER(1), PARAMETER, PUBLIC :: CTRL_C = #03

CONTAINS

```
! /*****  
***\  
!* FUNCTION      : GetConsoleTextAttribute  
!\*****  
***/  
integer*2 function GetConsoleTextAttribute( hConsole )  
use msfwin  
  
! /*****  
***\  
!* INPUTS       : hConsole - handle to an open console  
!* RETURNS      : Current text attributes WORD.  
!* GLOBAL VARS  : None.  
!* LOCAL VARS   : csbi - Console screen buffer information structure  
!* COMMENTS     : Gets the previous console color attributes  
!\*****  
***/  
integer hConsole  
  
    type (T_CONSOLE_SCREEN_BUFFER_INFO)    csbi  
  
        i = GetConsoleScreenBufferInfo( hConsole, csbi)  
        GetConsoleTextAttribute = csbi%wAttributes  
    !end type  
END FUNCTION GetConsoleTextAttribute  
  
! position cursor on screen; (0,0) is top-left position  
! http://www.cplusplus.com/forum/beginner/4234/  
! http://www.dreamincode.net/code/snippet921.htm has complete/extensive  
example(s)
```

```

SUBROUTINE gotoXY(x,y )
use msfwin
    INTEGER x,y
    integer hStdout
    type(T_COORD) position
    LOGICAL result

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE)
    position%x = x
    position%y = y
    result = SetConsoleCursorPosition( hStdout, position )
END SUBROUTINE gotoXY

INTEGER FUNCTION whereX()
use msfwin
    type(T_CONSOLE_SCREEN_BUFFER_INFO) csbi;
    type(T_COORD) result;

    if (.NOT.(GetConsoleScreenBufferInfo(GetStdHandle( STD_OUTPUT_HANDLE
), csbi))) THEN
        whereX = -1
    ELSE
        whereX = result%X
    ENDIF
END FUNCTION whereX

INTEGER FUNCTION whereY()
use msfwin
    type(T_CONSOLE_SCREEN_BUFFER_INFO) csbi;
    type(T_COORD) result;

    if (.NOT.(GetConsoleScreenBufferInfo(GetStdHandle( STD_OUTPUT_HANDLE
), csbi))) THEN
        whereY = -1
    ELSE
        whereY = result%Y
    ENDIF
END FUNCTION whereY

! http://www.daniweb.com/forums/thread216345.html
!0 = Black      8 = Gray
!1 = Blue       9 = Light Blue
!2 = Green      10 = Light Green
!3 = Aqua       11 = Light Aqua
!4 = Red        12 = Light Red
!5 = Purple     13 = Light Purple
!6 = Yellow     14 = Light Yellow
!7 = White      15 = Bright White
! 16 Black      17 Blue
! Foreground is the text color (range = 0-31)

```

```

! Background is the screen color (range = 0-7)

!This will set the foreground color for printing in a console window.
!SUBROUTINE SetColor(ForgC)      ! name SetColor already defined in
MSFLIB
SUBROUTINE SetFColor(ForgC)      ! name SetColor already defined in
MSFLIB
use msfwin
INTEGER ForgC
INTEGER(2) wColor
! We will need this handle to get the current background attribute
integer hStdOut
type(T_CONSOLE_SCREEN_BUFFER_INFO) csbi

hStdOut = GetStdHandle(STD_OUTPUT_HANDLE)

! We use csbi for the wAttributes word.
if(GetConsoleScreenBufferInfo(hStdOut, csbi)) THEN
! Mask out all but the background attribute, and add in the
forgournd color
wColor = IAND(csbi%wAttributes,#F0) + IAND(ForgC, #0F)
i=SetConsoleTextAttribute(hStdOut, wColor)
ENDIF
!return
END SUBROUTINE SetFColor
!END SUBROUTINE SetColor

SUBROUTINE SetFColor_QWIN(ForgC,hStdOut)      ! name SetColor already
defined in MSFLIB
use msfwin
INTEGER ForgC
INTEGER(2) wColor
! We will need this handle to get the current background attribute
integer hStdOut
type(T_CONSOLE_SCREEN_BUFFER_INFO) csbi

!hStdOut = GetStdHandle(STD_OUTPUT_HANDLE)
hStdOut = GetStdHandle(hStdOut)

! We use csbi for the wAttributes word.
if(GetConsoleScreenBufferInfo(hStdOut, csbi)) THEN
! Mask out all but the background attribute, and add in the
forgournd color
wColor = IAND(csbi%wAttributes,#F0) + IAND(ForgC, #0F)
i=SetConsoleTextAttribute(hStdOut, wColor)
ENDIF
!return
END SUBROUTINE SetFColor_QWIN

!CHARACTER, PARAMETER,PUBLIC ::CTRL_C  = #03

! http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2004-11/0961.html

```

```

!
!     Checks the keystroke buffer for a recent console
!     keystroke and returns .TRUE. if there is a character in the
buffer or
!     .FALSE. if there is not.
!
!     To find out the value of the key in the buffer, call GETCHARQQ.
If there is
!     no character waiting in the buffer when you call GETCHARQQ,
GETCHARQQ waits
!     until there is a character in the buffer. If you call PEEKCHARQQ
first, you
!     prevent GETCHARQQ from halting your process while it waits for a
keystroke.
!     If there is a keystroke, GETCHARQQ returns it and resets
PEEKCHARQQ to
!     .FALSE..
!
LOGICAL FUNCTION kbhit()
USE MSFLIB
    kbhit = PEEKCHARQQ( )
    RETURN
END    FUNCTION kbhit

CHARACTER FUNCTION getch()
USE MSFLIB
    getch = GETCHARQQ( )
    RETURN
END    FUNCTION getch

SUBROUTINE WRITE_$_TO_CONSOLE(str)
! http://software.intel.com/en-us/articles/cannot-write-to-window-
created-with-alloconsole
use msfwin
INTEGER hConsole ! the handle to the console window
CHARACTER*(*) str ! the string to be written
INTEGER num_w_char ! num. of characters actually written
LOGICAL result
!CHARACTER*100 tmpstr !/'/'/
!tmpstr = ""
hConsole = GetStdHandle(STD_OUTPUT_HANDLE)

!write(tmpstr,'(A)') ADJUSTL(TRIM(str)) ! error FOR1127: internal
compiler error
!write(tmpstr,'(A)') str
!100 FORMAT(A7)
!tmpstr = trim(tmpstr) // "\nC" ! add 'new line'
!result =
WriteConsole(hConsole, loc(tmpstr), len_Trim(tmpstr), loc(num_w_char), NULL
)

!result =
WriteConsole(hConsole, loc(str), len_Trim(str), loc(num_w_char), NULL)

```

```

result = WriteConsole(hConsole, loc(str), len(str), loc(num_w_char), NULL)
END SUBROUTINE WRITE_$_TO_CONSOLE

```

```

SUBROUTINE WRITE_TO_CONSOLE(str)
! http://software.intel.com/en-us/articles/cannot-write-to-window-
created-with-alloconsole
use msfwin
INTEGER hConsole ! the handle to the console window
CHARACTER*(*) str ! the string to be written
CHARACTER*100 tmpstr /'/ ! as WriteConsole works with the address
! of the string, we need a
different string
! that contains CR at the
end
INTEGER num_w_char ! num. of characters actually written
LOGICAL result
Type(T_COORD) con_wpos
Type (T_CONSOLE_SCREEN_BUFFER_INFO) con_info

hConsole = GetStdHandle(STD_OUTPUT_HANDLE)
if (str .EQ.'') THEN
    CALL WRITE_$_TO_CONSOLE("\nC")
    result = GetConsoleScreenBufferInfo(hConsole, con_info)
    con_wpos%x = con_info%dwCursorPosition%x -1 ! because a blank was
written
    con_wpos%y = con_info%dwCursorPosition%y
    result = SetConsoleCursorPosition(hConsole, con_wpos)
ELSE
    tmpstr = str // "\nC" ! add 'new line' (in C language syntax)
    result =
WriteConsole(hConsole, loc(tmpstr), len(tmpstr), loc(num_w_char), NULL)
ENDIF
END SUBROUTINE WRITE_TO_CONSOLE

```

```

SUBROUTINE CREATE_TEXT_CONSOLE
use msfwin
! Console data declarations
    integer con_lines, con_length
    logical con_status
    integer con_handle
    Type(T_COORD) con_wpos

! Set buffer size variables
    con_length = 80
    con_lines = 90
! Allocate a console
    con_status = AllocConsole() ! get a console window of the
currently set size
    con_handle = GetStdHandle(STD_OUTPUT_HANDLE)
    con_wpos%x = con_length ! must be >= currently set console
window line length

```

```

        con_wpos%y = con_lines      ! must be >= currently set console
window number of lines

        ! Set a console buffer bigger than the console window. This
provides
        ! scroll bars on the console window to scroll through the console
buffer
        con_status = SetConsoleScreenBufferSize(con_handle, con_wpos)

END SUBROUTINE CREATE_TEXT_CONSOLE

SUBROUTINE CLOSE_TEXT_CONSOLE
!   Deallocate the console to free its resources.
use msfwin
    logical con_status
    con_status = FreeConsole()

END SUBROUTINE CLOSE_TEXT_CONSOLE

END MODULE Console_API

```

DISTRIB.F90.

```

! -----
MODULE DISTRIB_1
! -----
USE LPROBE60_1
!   REAL(8),PARAMETER :: min_val=1.0
!   REAL(8),PARAMETER :: max_val=15000.0

TYPE :: GRIDDEF_T
    REAL(4) :: r
    REAL(4) :: v
END TYPE GRIDDEF_T

CONTAINS ! these are public

    SUBROUTINE DISTRIB(bins,n,particle,num_part)
!       USE LPROBE60_1
        TYPE (GRIDDEF_T) :: bins(0:n-1)
        TYPE (PARTICLEDEF ) particle(0:numpart-1)

        INTEGER i,pos
        REAL v

! Empty bins
        DO i = 0,n-1
            bins(i)%v = 0
        END DO
    
```

```

      ! PUT DATA IN BINS
      DO i = 0,num_part
        v= SQRT(particle(i)%vx**2 +particle(i)%vy**2
+particle(i)%vz**2 )
        CALL locate(bins, n-1, v, pos)
        bins(pos)%v = bins(pos)%v+1.0
      END DO
    END SUBROUTINE DISTRIB

```

```

! -----
SUBROUTINE locate(xx,n,x,j)
! -----
!USE DISTRIB_1
!IMPLICIT NONE
! - - - arg types - - -
  INTEGER :: n,j
  REAL(4) :: x
!  TYPE (GRIDDEF_T) :: xx(0:ngp)
  TYPE (GRIDDEF_T) :: xx(0:n-1)

! - - - local declarations - - -
  INTEGER :: ju,jm,jl !ascnd
  LOGICAL (1) ascnd

! - - - begin - - -
  jl = 0
  ju = n+1 ! // n intervals, (n+1) division points
!  ascnd = (xx(n-1)%r > xx(1)%r)
  ascnd = 0
  IF (xx(n-1)%r > xx(1)%r) ascnd = 1
  DO WHILE (ju-jl > 1)
    jm = ISHFT((ju+jl), -1) ! // jm=(ju+jl)/2
!    IF (x > xx(jm+1)%r == ascnd) THEN
    IF ( (x > xx(jm)%r).EQV.ascnd) THEN
      jl = jm
    ELSE
      ju = jm
    END IF
  END DO
  j = jl

END SUBROUTINE locate

END MODULE DISTRIB_1

```

ERF2.FOR.

```

MODULE ERF_2

  CONTAINS

```


C <http://lib.stat.cmu.edu/apstat/195>

```

C      SUBROUTINE CALERF(ARG,RESULT,JINT)
C-----
C
C THIS PACKET COMPUTES THE ERROR AND COMPLEMENTARY ERROR FUNCTIONS
C FOR REAL ARGUMENTS ARG. IT CONTAINS TWO FUNCTION TYPE
C SUBPROGRAMS, ERF AND ERFC (OR DERF AND DERFC), AND ONE
C SUBROUTINE TYPE SUBPROGRAM, CALERF. THE CALLING STATEMENTS
C FOR THE PRIMARY ENTRIES ARE
C
C      Y=ERF(X)      (OR   Y=DERF(X) )
C AND
C      Y=ERFC(X)     (OR   Y=DERFC(X) ).
C
C THE ROUTINE CALERF IS INTENDED FOR INTERNAL PACKET USE ONLY,
C ALL COMPUTATIONS WITHIN THE PACKET BEING CONCENTRATED IN THIS
C ROUTINE. THE FUNCTION SUBPROGRAMS INVOKE CALERF WITH THE
C STATEMENT
C      CALL CALERF(ARG,RESULT,JINT)
C WHERE THE PARAMETER USAGE IS AS FOLLOWS
C
C      FUNCTION          PARAMETERS FOR CALERF
C      CALL              ARG          RESULT          JINT
C      ERF(ARG)          ANY REAL ARGUMENT          ERF(ARG)          0
C      ERFC(ARG)         ABS(ARG) .LT. XMAX          ERFC(ARG)          1
C
C THE MAIN COMPUTATION EVALUATES NEAR MINIMAX APPROXIMATIONS
C FROM "RATIONAL CHEBYSHEV APPROXIMATIONS FOR THE ERROR FUNCTION"
C BY W. J. CODY, MATH. COMP., 1969, PP. 631-638. THIS
C TRANSPORTABLE PROGRAM USES RATIONAL FUNCTIONS THAT THEORETICALLY
C APPROXIMATE ERF(X) AND ERFC(X) TO AT LEAST 18 SIGNIFICANT
C DECIMAL DIGITS. THE ACCURACY ACHIEVED DEPENDS ON THE ARITHMETIC
C SYSTEM, THE COMPILER, THE INTRINSIC FUNCTIONS, AND PROPER
C SELECTION OF THE MACHINE-DEPENDENT CONSTANTS.
C
C *****
C
C EXPLANATION OF MACHINE-DEPENDENT CONSTANTS
C
C XSMALL = ARGUMENT BELOW WHICH ERF(X) MAY BE REPRESENTED
C BY  $2*X/\sqrt{\pi}$  AND ABOVE WHICH  $X*X$  WILL
C NOT UNDERFLOW. A CONSERVATIVE VALUE IS THE
C LARGEST X SUCH THAT  $1.0 + X = 1.0$  TO MACHINE
C PRECISION.
C XMAX = LARGEST ARGUMENT ACCEPTABLE TO ERFC; SOLUTION TO
C EQUATION:  $W(X) * (1-0.5/X**2) = XMIN$ , WHERE
C  $W(X) = \exp(-X*X)/(X*\sqrt{\pi})$ , AND XMIN IS THE
C SMALLEST POSITIVE MACHINE NUMBER (SEE TABLE BELOW).
C
C APPROXIMATE VALUES FOR SOME IMPORTANT MACHINES ARE:
C
C      XSMALL      XMAX      XMIN

```

```

C   IBM 195      (D.P.)   1.39D-17   13.306   5.40D-79
C   CDC 7600    (S.P.)   7.11E-15   25.922   3.13E-294
C   CRAY-1      (S.P.)   7.11E-15   75.326   4.58E-2467
C   UNIVAC 1108 (D.P.)   1.73D-18   26.582   2.78D-309
C   VAX 11/780  (S.P.)   5.96E-8    9.269   2.94E-39
C   VAX 11/780  (D.P.)   1.39D-17   9.269   2.94D-39
C   IBM PC      (S.P.)   5.96E-8    9.194   1.18E-38
C   IBM PC      (D.P.)   1.11D-16   26.543   2.23D-308
C
C *****
C
C ERROR RETURNS
C
C   THE PROGRAM RETURNS  ERFC = 0   FOR  ARG .GT. XMAX.
C
C
C OTHER SUBPROGRAMS REQUIRED (SINGLE PRECISION VERSION)
C
C   ABS, EXP
C
C OTHER SUBPROGRAMS REQUIRED (DOUBLE PRECISION VERSION)
C
C   DABS, DEXP
C
C
C   AUTHOR: W. J. CODY
C           MATHEMATICS AND COMPUTER SCIENCE DIVISION
C           ARGONNE NATIONAL LABORATORY
C           ARGONNE, IL 60439
C
C   LATEST MODIFICATION: JANUARY 8, 1985
C
C-----
C      INTEGER I, JINT
CS     REAL          A, ARG, B, C, D, FOUR, HALF, P, ONE, Q, RESULT, SQRPI,
CS     1             TWO, THRESH, X, XMAX, XDEN, XNUM, XSMALL, Y, YSQ, ZERO
C      DOUBLE PRECISION A, ARG, B, C, D, FOUR, HALF, P, ONE, Q, RESULT, SQRPI,
C      1             TWO, THRESH, X, XMAX, XDEN, XNUM, XSMALL, Y, YSQ, ZERO
C      DIMENSION A(5), B(4), C(9), D(8), P(6), Q(5)
C-----
C   MATHEMATICAL CONSTANTS
C-----
CS     DATA FOUR, ONE, HALF, TWO, ZERO/4.0E0, 1.0E0, 0.5E0, 2.0E0, 0.0E0/
CS     DATA SQRPI/5.6418958354775628695E-1/, THRESH/0.46875E0/
C     DATA FOUR, ONE, HALF, TWO, ZERO/4.0D0, 1.0D0, 0.5D0, 2.0D0, 0.0D0/
C     DATA SQRPI/5.6418958354775628695D-1/, THRESH/0.46875D0/
C-----
C   MACHINE-DEPENDENT PARAMETERS
C-----
CS     DATA XSMALL/4.2E-16/, XMAX/9.269E0/
C     DATA XSMALL/4.2D-16/, XMAX/9.269D0/
C-----
C   COEFFICIENTS FOR APPROXIMATION TO DERF IN FIRST INTERVAL
C-----

```

```

CS    DATA A/3.16112374387056560E00,1.13864154151050156E02,
CS    1      3.77485237685302021E02,3.20937758913846947E03,
CS    2      1.85777706184603153E-1/
CS    DATA B/2.36012909523441209E01,2.44024637934444173E02,
CS    1      1.28261652607737228E03,2.84423683343917062E03/
      DATA A/3.16112374387056560D00,1.13864154151050156D02,
      1      3.77485237685302021D02,3.20937758913846947D03,
      2      1.85777706184603153D-1/
      DATA B/2.36012909523441209D01,2.44024637934444173D02,
      1      1.28261652607737228D03,2.84423683343917062D03/

```

```

C-----
C  COEFFICIENTS FOR APPROXIMATION TO DERFC IN SECOND INTERVAL
C-----

```

```

CS    DATA C/5.64188496988670089E-1,8.88314979438837594E0,
CS    1      6.61191906371416295E01,2.98635138197400131E02,
CS    2      8.81952221241769090E02,1.71204761263407058E03,
CS    3      2.05107837782607147E03,1.23033935479799725E03,
CS    4      2.15311535474403846E-8/
CS    DATA D/1.57449261107098347E01,1.17693950891312499E02,
CS    1      5.37181101862009858E02,1.62138957456669019E03,
CS    2      3.29079923573345963E03,4.36261909014324716E03,
CS    3      3.43936767414372164E03,1.23033935480374942E03/
      DATA C/5.64188496988670089D-1,8.88314979438837594D0,
      1      6.61191906371416295D01,2.98635138197400131D02,
      2      8.81952221241769090D02,1.71204761263407058D03,
      3      2.05107837782607147D03,1.23033935479799725D03,
      4      2.15311535474403846D-8/
      DATA D/1.57449261107098347D01,1.17693950891312499D02,
      1      5.37181101862009858D02,1.62138957456669019D03,
      2      3.29079923573345963D03,4.36261909014324716D03,
      3      3.43936767414372164D03,1.23033935480374942D03/

```

```

C-----
C  COEFFICIENTS FOR APPROXIMATION TO DERFC IN THIRD INTERVAL
C-----

```

```

CS    DATA P/3.05326634961232344E-1,3.60344899949804439E-1,
CS    1      1.25781726111229246E-1,1.60837851487422766E-2,
CS    2      6.58749161529837803E-4,1.63153871373020978E-2/
CS    DATA Q/2.56852019228982242E00,1.87295284992346047E00,
CS    1      5.27905102951428412E-1,6.05183413124413191E-2,
CS    2      2.33520497626869185E-3/
      DATA P/3.05326634961232344D-1,3.60344899949804439D-1,
      1      1.25781726111229246D-1,1.60837851487422766D-2,
      2      6.58749161529837803D-4,1.63153871373020978D-2/
      DATA Q/2.56852019228982242D00,1.87295284992346047D00,
      1      5.27905102951428412D-1,6.05183413124413191D-2,
      2      2.33520497626869185D-3/

```

```

C-----
      X = ARG
CS    Y = ABS(X)
      Y = DABS(X)
      IF (Y .GT. FOUR) GO TO 200
      IF (Y .GT. THRESH) GO TO 100

```

```

C-----
C  EVALUATE ERF FOR ABS(X) .LE. 0.46875

```

```

C-----
      YSQ = ZERO
      IF (Y .GT. XSMALL) YSQ = Y * Y
      XNUM = A(5)*YSQ
      XDEN = YSQ
      DO 20 I = 1, 3
        XNUM = (XNUM + A(I)) * YSQ
        XDEN = (XDEN + B(I)) * YSQ
      20 CONTINUE
      RESULT = X * (XNUM + A(4)) / (XDEN + B(4))
      IF (JINT .NE. 0) RESULT = ONE - RESULT
      GO TO 800
C-----
C  EVALUATE ERFC FOR 0.46875 .LT. ABS(X) .LE. 4.0
C-----
      100 YSQ = Y * Y
          XNUM = C(9)*Y
          XDEN = Y
          DO 120 I = 1, 7
            XNUM = (XNUM + C(I)) * Y
            XDEN = (XDEN + D(I)) * Y
          120 CONTINUE
      CS  RESULT = EXP(-YSQ) * (XNUM + C(8)) / (XDEN + D(8))
          RESULT = DEXP(-YSQ) * (XNUM + C(8)) / (XDEN + D(8))
          GO TO 300
C-----
C  EVALUATE ERFC FOR ABS(X) .GT. 4.0
C-----
      200 RESULT = ZERO
          IF (Y .GE. XMAX) GO TO 300
      220 YSQ = ONE / (Y * Y)
          XNUM = P(6)*YSQ
          XDEN = YSQ
          DO 240 I = 1, 4
            XNUM = (XNUM + P(I)) * YSQ
            XDEN = (XDEN + Q(I)) * YSQ
          240 CONTINUE
          RESULT = YSQ *(XNUM + P(5)) / (XDEN + Q(5))
      CS  RESULT = (EXP(-Y*Y) / Y) * (SQRPI - RESULT)
          RESULT = (DEXP(-Y*Y) / Y) * (SQRPI - RESULT)
C-----
C  FIX UP FOR NEG. ARG., ERF, ETC.
C-----
      300 IF (JINT .EQ. 0) GO TO 350
          IF (X .LT. ZERO) RESULT = TWO - RESULT
          GO TO 800
      350 RESULT = (HALF - RESULT) + HALF
          IF (X .LT. ZERO) RESULT = -RESULT
C-----
      800 RETURN
C----- LAST CARD OF CALERF -----
      END      SUBROUTINE

```

```

CS      REAL FUNCTION ERF(X)
        DOUBLE PRECISION DERF(X)
C-----
C
C PROGRAM TO COMPUTE THE ERROR FUNCTION
C
C   AUTHOR - W. J. CODY
C
C   DATE - JANUARY 8, 1985
C-----
        INTEGER JINT
CS      REAL          X, RESULT
        DOUBLE PRECISION X, RESULT
C-----
        JINT = 0
        CALL CALERF(X, RESULT, JINT)
CS      ERF = RESULT
        DERF = RESULT
        RETURN
C----- LAST CARD OF DERF -----
        END    FUNCTION DERF

        REAL FUNCTION ERFF(X)
C-----
C
C PROGRAM TO COMPUTE THE ERROR FUNCTION
C-----
        INTEGER JINT
CS      REAL          X, RESULT
        REAL X, R_RESULT
        DOUBLE PRECISION DX, RESULT
C-----
        JINT = 0
        DX=X
        CALL CALERF(DX, RESULT, JINT)
CS      ERF = RESULT
        R_RESULT = RESULT
        ERFF = R_RESULT
        RETURN
C----- LAST CARD OF DERF -----
        END    FUNCTION ERFF

CS      REAL FUNCTION ERFC(X)
        DOUBLE PRECISION DERFC(X)
C-----
C
C PROGRAM TO COMPUTE THE COMPLEMENTARY ERROR FUNCTION

```

```

C
C   AUTHOR - W. J. CODY
C
C   DATE - JANUARY 8, 1985
C
C-----
      INTEGER JINT
CS      REAL          X, RESULT
      DOUBLE PRECISION X, RESULT
C-----
      JINT = 1
      CALL CALERF(X,RESULT,JINT)
CS      ERFC = RESULT
      DERFC = RESULT
      RETURN
C----- LAST CARD OF DERFC -----
      END    FUNCTION  DERFC
C
      END MODULE ERF_2

```

FIELD_SOLVER.F90:

```

! -----
MODULE FIELD_SOLVER_1
! -----
!   This function computes the elcetric field intensity at
!   the gridpoints using the updated source term and the
!   previously factored [A] matrix      */
! page 70

USE LPROBE60_1
USE LAPACK_1
!USE MSFLIB  ! for debug

CONTAINS

! -----
SUBROUTINE e_field(array)
! -----
! - - - arg types - - -
!TYPE (GRIDDEF ) array(0:ngp1-1,0:ngp2-1)
TYPE (GRIDDEF ) array(0:,0:)
! - - - local declarations - - -
REAL deltx,jmax,kmax
INTEGER :: j,k,count1
CHARACTER :: trans

!TYPE (rccoord) crsPos
!crsPos%row =1
!crsPos%col=1

```

```

! - - - begin - - -
trans = "N"
nrhs = 1
jmax = ngp_vert
kmax = ngp_horiz
deltx = cell_width

!      Zero out source term      */
count1 = -1
DO j = 0,ngp_vert-1
  DO k = 0,ngp_horiz-1
    count1 = count1+1
    b(count1)=0.0
  END DO
END DO

!      Update source term      */
count1 = -1
DO j = 0,ngp_vert-1
  DO k = 0,ngp_horiz-1
    count1 = count1+1
    IF (k == 0 .AND. j /= (jmax-1)) THEN
      b(count1)=-array(j,k)%q_dens /EPS0 - plasma_pot
    ELSE IF (j == (jmax-1) .AND. k /= (kmax-1)) THEN
      b(count1)=-array(j,k)%q_dens /EPS0 - plasma_pot
    ELSE IF (j <= jprobe .AND. j > 0 .AND. k >= kprobe) THEN
      b(count1)=-array(j,k)%q_dens /EPS0 - V
    ELSE IF (j <= jprobe .AND. j == 0 .AND. k >= kprobe) THEN
      b(count1)=-array(j,k)%q_dens /EPS0 - V
    ELSE IF (j > jprobe .AND. k == (kmax-1)) THEN
      b(count1)=-array(j,k)%q_dens /EPS0 - plasma_pot
    ELSE
      b(count1)=-array(j,k)%q_dens /EPS0
    END IF
  END DO
END DO

!      Call Lapack to solve [a]x=b      */
CALL sgbtrs(trans, n1, lower_width, upper_width, nrhs, abd, ldab,
piv, b, ldb, info)
IF (info /= 0) THEN
!      WRITE (*,901) "Band Solver Failed, info==>",info
!      WRITE (*,*) " Band Solver Failed, info==>",info
!      STOP '(1)'
END IF

!      Put phi into grid (note:this is a waste, but for now....)      */
count1 = -1
DO j = 0,ngp_vert-1
  DO k = 0,ngp_horiz-1
    count1 = count1+1
    array(j,k)%phi = b(count1)
  
```

```

        END DO
    END DO

!    Determine electric field at all gridpoints    */
    DO j = 0,ngp_vert-1
        DO k = 0,ngp_horiz-1
            IF (k == 0) THEN
                array(j,k)%Ex = 1.0/(2.0*deltx)*(-
3.0*array(j,0)%phi+4.0*array(j,1)%phi-array(j,2)%phi)
                array(j,k)%Ey = 0.0
!            ELSE IF (j == jprobe .AND. k >= kprobe) THEN    !// upper part
of the conductor (all its length)
! 29 Mar 2010
                ELSE IF (j <= jprobe .AND. k >= kprobe) THEN    ! // body
conductor
                    array(j,k)%Ey = 1.0/(2.0*deltx)*(-
3.0*array(j,k)%phi+4.0*array(j,k+1)%phi-array(j,k+2)%phi)
                    array(j,k)%Ex = 0.0
                    ELSE IF (k ==(ngp_horiz-1)) THEN
                        array(j,k)%Ex = -1.0/(2.0*deltx)*(-3.0*array(j,ngp_horiz-
1)%phi+4.0*array(j,ngp_horiz-2)%phi-array(j,ngp_horiz-3)%phi)
                        array(j,k)%Ey = 0.0
                    ELSE IF (j == 0) THEN
                        array(j,k)%Ex = -(array(j,k-1)%phi-
array(j,k+1)%phi)/(2.0*deltx)
                        array(j,k)%Ey = 0.0
                    ELSE IF (j ==(ngp_vert-1)) THEN
                        array(j,k)%Ex = 0.0
                        !array(j,k)%Ey = array(j-1,k)%phi/cell_height6
                        array(j,k)%Ey = array(j-1,k)%phi/(array(j,k)%y-array(j-1,k)%y)
! 07Mar,2010

! as now 'cell_height6' is not the
height of the upper rows of cells, but the cells height increase rate
                    ELSE
                        array(j,k)%Ex = (array(j,k-1)%phi-array(j,k+1)%phi)/(2.0*deltx)
                        array(j,k)%Ey = -((array(j+1,k)%phi-
array(j,k)%phi)/(array(j+1,k)%y-array(j,k)%y))*(array(j,k)%y-array(j-
1,k)%y)/(array(j+1,k)%y-array(j-1,k)%y) &
!
+ -((array(k,j)%phi-array(k,j-
1+1)%phi)/(array(k,j)%y-array(k,j-1+1)%y))*(array(k,j+1+1)%y-
array(k,j)%y)/(array(k,j+1+1)%y-array(k,j-1+1)%y)
- ((array(j,k)%phi-array(j-
1,k)%phi)/(array(j,k)%y-array(j-1,k)%y))*(array(j+1,k)%y-
array(j,k)%y)/(array(j+1,k)%y-array(j-1,k)%y)
                        END IF
                    END DO ! k

!    CALL SETTEXTPOSITION (INT2(1), INT2(1), crsPos)
!    WRITE(*,100) j,k
!    100 FORMAT(" j=",I4," k=",I4, $)
    END DO ! j

!901 FORMAT (A,I0)

```



```

END SUBROUTINE e_field
END MODULE FIELD_SOLVER_1

```

FORCE_WEIGHT.F90.

```

! -----
MODULE FORCE_WEIGHT_1
! -----
! This function weights the field from the array */

USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE fweight(array,particle,number)
! -----
! - - - arg types - - -
  INTEGER :: number
  !TYPE (PARTICLEDEF ) particle(0:numpart-1)
  !TYPE (GRIDDEF ) array(0:ngp1-1,0:ngp2-1)
  TYPE (PARTICLEDEF ) particle(0:)
  TYPE (GRIDDEF ) array(0:,0:)
! - - - local declarations - - -
  INTEGER :: count1
  REAL dr,dr1,dr2,dz,dz1,dz2
! - - - begin - - -

!      CLEAN UP OLD ELECTRIC FIELD      */

  DO count1 = 0,number-1
    particle(count1)%Ex = 0.0
    particle(count1)%Ey = 0.0
  END DO

  DO count1 = 0,number-1
    IF (particle(count1)%r <= radius) THEN
      if ( (particle(count1)%celly+1 < ngp1) .AND.
(particle(count1)%cellx+1 < ngp2)) THEN
        dr=array(particle(count1)%celly+1,
particle(count1)%cellx)%y*array(particle(count1)%celly+1,
particle(count1)%cellx)%y - array(particle(count1)%celly,
particle(count1)%cellx)%y *array(particle(count1)%celly,
particle(count1)%cellx)%y

        dr1 = particle(count1)%r -
array(particle(count1)%celly,particle(count1)%cellx)%y
!      dr2 = array(particle(count1)%celly,particle(count1)%cellx+1)%y -
particle(count1)%r

```

```

dr2=array(particle(count1)%celly+1 ,particle(count1)%cellx)%y -
particle(count1)%r

!      dz = array(particle(count1)%celly+1,particle(count1)%cellx)%x -
&
!      array(particle(count1)%celly,particle(count1)%cellx)%x
dz=array(particle(count1)%celly, particle(count1)%cellx+1)%x -
array(particle(count1)%celly, particle(count1)%cellx)%x

dz1 = particle(count1)%x -
array(particle(count1)%celly,particle(count1)%cellx)%x
!      dz2 = array(particle(count1)%celly+1,particle(count1)%cellx)%x -
particle(count1)%x
dz2=array(particle(count1)%celly, particle(count1)%cellx+1)%x -
particle(count1)%x

!      particle(count1)%Ey =
array(particle(count1)%celly,particle(count1)%cellx)%Ey*dr2 * dz2 &
!      * (2.0 *
array(particle(count1)%celly,particle(count1)%cellx+1)%y + 3.0* &
!      array(particle(count1)%celly,particle(count1)%cellx)%y - &
!      particle(count1)%r) / (2.0 * dr * dz) &
!      +
array(particle(count1)%celly+1,particle(count1)%cellx)%Ey*dr2 * dz1 &
!      * (2.0 *
array(particle(count1)%celly+1,particle(count1)%cellx+1)%y + 3.0 * &
!      array(particle(count1)%celly,particle(count1)%cellx)%y -
particle(count1)%r) / &
!      (2.0 * dr * dz ) +
array(particle(count1)%celly+1,particle(count1)%cellx+1)%Ey* &
!      dr1 * dz1 * (3.0 *
array(particle(count1)%celly+1,particle(count1)%cellx+1)%y + &
!      2.0 *
array(particle(count1)%celly+1,particle(count1)%cellx)%y - &
!      particle(count1)%r) / (2.0 * dr * dz) + &
!
array(particle(count1)%celly,particle(count1)%cellx+1)%Ey*dr1 * dz2 *
(3.0 * &
!      array(particle(count1)%celly,particle(count1)%cellx+1)%y +
2.0 * &
!      array(particle(count1)%celly,particle(count1)%cellx)%y -
particle(count1)%r) &
!      / (2.0 * dr * dz)
particle(count1)%Ey=array(particle(count1)%celly,
particle(count1)%cellx)%Ey*dr2 * dz2* (2.0 *
array(particle(count1)%celly+1 ,particle(count1)%cellx)%y + 3.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz) + array(particle(count1)%celly,
particle(count1)%cellx+1)%Ey*dr2 * dz1 * (2.0 *
array(particle(count1)%celly+1, particle(count1)%cellx+1)%y + 3.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz ) +
array(particle(count1)%celly+1, particle(count1)%cellx+1)%Ey* dr1 * dz1

```

```

* (3.0 * array(particle(count1)%celly+1, particle(count1)%cellx+1)%y +
2.0 * array(particle(count1)%celly, particle(count1)%cellx+1)%y -
particle(count1)%r) / (2.0 * dr * dz) + array(particle(count1)%celly+1,
particle(count1)%cellx)%Ey*dr1 * dz2 * (3.0 *
array(particle(count1)%celly+1, particle(count1)%cellx)%y + 2.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz)

!      particle(count1)%Ex =
array(particle(count1)%celly, particle(count1)%cellx)%Ex*dr2 * dz2 &
!      * (2.0 *
array(particle(count1)%celly, particle(count1)%cellx+1)%y + 3.0* &
!      array(particle(count1)%celly, particle(count1)%cellx)%y - &
!      particle(count1)%r) / (2.0 * dr * dz) &
!      +
array(particle(count1)%celly+1, particle(count1)%cellx)%Ex*dr2 * dz1 &
!      * (2.0 *
array(particle(count1)%celly+1, particle(count1)%cellx+1)%y + 3.0 * &
!      array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / &
!      (2.0 * dr * dz ) +
array(particle(count1)%celly+1, particle(count1)%cellx+1)%Ex* &
!      dr1 * dz1 * (3.0 *
array(particle(count1)%celly+1, particle(count1)%cellx+1)%y + &
!      2.0 *
array(particle(count1)%celly+1, particle(count1)%cellx)%y - &
!      particle(count1)%r) / (2.0 * dr * dz) + &
!
array(particle(count1)%celly, particle(count1)%cellx+1)%Ex*dr1 * dz2 *
(3.0 * &
!      array(particle(count1)%celly, particle(count1)%cellx+1)%y +
2.0 * &
!      array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) &
!      / (2.0 * dr * dz)
      particle(count1)%Ex=array(particle(count1)%celly,
particle(count1)%cellx)%Ex*dr2 * dz2* (2.0 *
array(particle(count1)%celly+1 ,particle(count1)%cellx)%y + 3.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz) + array(particle(count1)%celly,
particle(count1)%cellx+1)%Ex*dr2 * dz1 * (2.0 *
array(particle(count1)%celly+1, particle(count1)%cellx+1)%y + 3.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz ) +
array(particle(count1)%celly+1, particle(count1)%cellx+1)%Ex* dr1 * dz1
* (3.0 * array(particle(count1)%celly+1, particle(count1)%cellx+1)%y +
2.0 * array(particle(count1)%celly, particle(count1)%cellx+1)%y -
particle(count1)%r) / (2.0 * dr * dz) + array(particle(count1)%celly+1,
particle(count1)%cellx)%Ex*dr1 * dz2 * (3.0 *
array(particle(count1)%celly+1, particle(count1)%cellx)%y + 2.0 *
array(particle(count1)%celly, particle(count1)%cellx)%y -
particle(count1)%r) / (2.0 * dr * dz)
      ELSE

```

```

                write(*,*) "FORCE_WEIGHT.F90:
cellx=",particle(count1)%cellx," celly=",particle(count1)%celly
                END IF ! celly+1<ngp1
            END IF ! r<radius
        END DO

END SUBROUTINE    fweight
END MODULE FORCE_WEIGHT_1

```

GRID.F90.

```

! -----
MODULE GRID_1
! -----
! page 61,117,

USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE gridgen(array)
! -----
! - - - arg types - - -
    TYPE (GRIDDEF ) array(0:ngp1-1,0:ngp2-1)

! - - - local declarations - - -
    INTEGER :: count1,count2,cell1_count,cell6_count
    REAL index1,index2
    INTEGER count_vert ! is incremented when a new vertical row is added
to mesh
                                ! In the end its value should be
74*ncpr

! - - - begin - - -
    ngp_vert = ngp1
    ngp_horiz = ngp2
    cell1_count=ncpr+ncpr+n0Rows15 ! the first rows of cells has
'r_probe/ncpr' height
                                ! they include probe
(ncpr), insulator(ncpr) and n0Rows15
    !cell6_count = 74
    count_vert = 0

    index1 = -1.0
    DO count1 = 0,cell1_count-1
        index1 = index1+1.0
        index2 = -1.0
        DO count2 = 0,ngp_horiz-1
            index2 = index2+1.0
            array(count_vert,count2)%x = -length+index2*cell_width
            array(count_vert,count2)%y = index1*cell_height1

```

```

        END DO
        count_vert = count_vert +1
    END DO

    cell6_count= count_vert

    index1 = -1.0
    DO count1 = cell6_count, ngp_vert-1
        index1 = index1+1.0
        index2 = -1.0
        DO count2 = 0,ngp_horiz-1
            index2 = index2+1.0
            array(count_vert,count2)%x = -length+index2*cell_width
            array(count_vert,count2)%y = 2*array(count_vert-1,count2)%y -
array(count_vert-2,count2)%y + cell_height6
        END DO
        count_vert = count_vert +1
    END DO

    ! as we had cumulative errors in cells height, the last row of points
    on grid
    ! need a small correction for y component
    DO count2=0,ngp_horiz-1
        array(ngp_vert-1,count2)%y=radius
    END DO

    !      Clean-up grid for next time step      */

    DO count1 = 0,ngp_vert-1
        DO count2 = 0,ngp_horiz-1
            array(count1,count2)%Ey = 0.0
            array(count1,count2)%Ex = 0.0
            array(count1,count2)%q_dens = 0.0
        END DO
    END DO

END SUBROUTINE gridgen
END MODULE GRID_1

```

INITIALIZE.F90:

```

! -----
MODULE INITIALIZE_1
! -----
! init_num: initial # of particles in domain
! rancount: "8*init_num" random values using random_num[]
USE LPROBE60_1

CONTAINS

```

```

! -----
SUBROUTINE
initialize(particle,num_part,init_num,super_init,v_table,m,q)

! -----
! - - - arg types - - -
INTEGER :: num_part,init_num
REAL super_init
!TYPE (PARTICLEDEF ) particle(0:numpart-1)
TYPE (PARTICLEDEF ) particle(0:)
REAL m,q
REAL v_table(:)
! - - - local declarations - - -
INTEGER :: i,rannum
REAL theta,phi,v,r

! - - - begin - - -
!A = (SIZE(random_num)-1)/(8.0*init_num) ! factor that maps
[0..ini_num] over [0..NPTS]
DO i = 0,init_num-1
    theta=pi*random_num(rancount)
    rancount =rancount +1
    phi=2.0*pi*random_num(rancount)
    rancount = rancount + 1
    rannum=V_TABLE_RES*random_num(rancount)
    rancount =rancount +1
    v=v_table(rannum+1)
    rancount =rancount +1

    ! debug
    if (random_num(rancount) <= 0.0) then
        write(*,*) 'random_num(', rancount, '+1)=' ,
random_num(rancount)
    end if
    ! end debug

    r = radius * SQRT(random_num(rancount))
    rancount =rancount +1
    particle(num_part)%x = -length * random_num(rancount)
    rancount = rancount+1
    particle(num_part)%y = r*SIN(theta)
    particle(num_part)%z = r*COS(theta)
    ! particle(num_part)%r =
    SQRT(particle(num_part)%y*particle(num_part)%y + &
    !
particle(num_part)%z*particle(num_part)%z)
    particle(num_part)%r = r
    particle(num_part)%vx = v * SIN(theta) *SIN(phi)
    particle(num_part)%vy = v * COS(theta)
    particle(num_part)%vz = v * SIN(theta) * COS(phi)
    particle(num_part)%Ex = 0.0
    particle(num_part)%Ey = 0.0

```



```

!A = (SIZE(random_num)-1)/(rancount+5.0*superinj_front) ! factor that
maps [0..ini_num] over [0..NPTS]
if (SIZE(random_num)<rancount+5*superinj_front) rancount=0 ! make
sure that 'racount' stays in [0..NPTS]
DO i = 0,superinj_front-1
  !      FRONT      */
  theta = acos(SQRT(1 - random_num(rancount)))
  rancount =rancount +1
  phi = 2.0*pi*random_num(rancount)
  rancount =rancount +1
  rannum = V_TABLE_RES*random_num(rancount)
  rancount =rancount +1
  v = v_table(rannum)
  particle(num_part)%x = -length
!   particle(num_part)%y = height * random_num(rancount) - height /
2.0
                                r = radius * random_num(rancount)
  particle(num_part)%y= r*sin(phi) ! 13Mar, 2010
  rancount =rancount +1
!   particle(num_part)%z = width * random_num(rancount) - width / 2.0
  particle(num_part)%z = r*cos(phi) ! 13Mar, 2010
  rancount =rancount +1
!   particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!   *particle(num_part)%z)
  particle(num_part)%r = r
  particle(num_part)%vx = v * COS(theta)
  particle(num_part)%vy = v*SIN(theta)*COS(phi)
  particle(num_part)%vz = v*SIN(theta)*SIN(phi)
  particle(num_part)%q = q * super_front
  particle(num_part)%m = m * super_front
  particle(num_part)%Ex = 0.0
  particle(num_part)%Ey = 0.0

  particle(num_part)%cellx=0 ! 12 Mar, 2010
  !CALL
locatey_2(grid,ngp1,abs(particle(num_part)%y),positiony,0)
  CALL locatey_2(grid,ngp1,particle(num_part)%r,positiony,0)
  particle(num_part)%celly=positiony

  CALL pgpt(1,particle(num_part)%x,particle(num_part)%y,-1) !
graphic symbol #(-1)

  num_part= num_part +1
  if (num_part >= numpart) THEN
    write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
    STOP
  END IF

END DO

!      BACK      */

```



```

!A = (SIZE(random_num)-1)/(rancount+5.0*superinj_front) ! factor that
maps [0..rancount] over [0..NPTS]
if (SIZE(random_num)<rancount+5) rancount=0 ! make sure that
'racount' stays in [0..NPTS]
DO i = 0,superinj_back-1
  theta = acos(SQRT(1 - random_num(rancount)))
  rancount = rancount + 1
  phi = 2.0*pi*random_num(rancount)
  rancount = rancount + 1
  rannum = V_TABLE_RES*random_num(rancount)
  rancount = rancount + 1
  v = v_table(rannum)

  particle(num_part)%x = 0.0
!   particle(num_part)%y = height * random_num(rancount) - height /
2.0
      r = radius * random_num(rancount)
  particle(num_part)%y = r*sin(phi) ! 13Mar, 2010
  rancount = rancount + 1
!   particle(num_part)%z = width * random_num(rancount) - width /
2.0
      particle(num_part)%z = r*cos(phi) ! 13Mar, 2010
  rancount = rancount + 1
!   particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!   *particle(num_part)%z)
  particle(num_part)%r = r
  particle(num_part)%vx = -v * COS(theta)
  particle(num_part)%vy = v*SIN(theta)*COS(phi)
  particle(num_part)%vz = v*SIN(theta)*SIN(phi)
  particle(num_part)%Ex = 0.0
  particle(num_part)%Ey = 0.0
  particle(num_part)%q = q * super_back
  particle(num_part)%m = m * super_back
DO WHILE ( particle(num_part)%r <= r_insul) ! // avoid
injecting in the insulator region
  !A = (SIZE(random_num)-1)/(rancount+2) ! factor that maps
[0..rancount] over [0..NPTS]
  if (SIZE(random_num)<rancount+2) rancount=0 ! make sure that
'racount' stays in [0..NPTS]
  particle(num_part)%x = 0.0
!   particle(num_part)%y = height * random_num(rancount) - height /
2.0
      r = radius * random_num(rancount)
  particle(num_part)%y = r*sin(phi) ! 13Mar, 2010
  rancount = rancount + 1
!   particle(num_part)%z = width * random_num(rancount) - width /
2.0
      particle(num_part)%z = r*cos(phi) ! 13Mar, 2010
  rancount = rancount + 1
!   particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&

```

```

!      *particle(num_part)%z)
particle(num_part)%r = r
particle(num_part)%vx = -v * COS(theta)
particle(num_part)%vy = v*SIN(theta)*COS(phi)
particle(num_part)%vz = v*SIN(theta)*SIN(phi)
particle(num_part)%Ex = 0.0
particle(num_part)%Ey = 0.0
particle(num_part)%q = q * super_back
particle(num_part)%m = m * super_back
END DO

      particle(num_part)%cellx=ngp2-2 ! 12 Mar, 2010
!      CALL locatey_2(grid,ngp1,abs(particle(num_part)%y),positiony,0)
      CALL locatey_2(grid,ngp1,particle(num_part)%r,positiony,0)
      particle(num_part)%celly=positiony

      CALL pgpt(1,particle(num_part)%x,particle(num_part)%y,-1) !
graphic symbol #(-1)

      num_part = num_part +1
      if (num_part >= numpart) THEN
        write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
        STOP
      END IF
END DO

!      BOTTOM SIDE      */
!A = (SIZE(random_num)-1)/(rancount+19.0*superinj_side) ! factor that
maps [0..rancount] over [0..NPTS]
if (SIZE(random_num)<rancount+20*superinj_side) rancount=0 ! make
sure that 'racount' stays in [0..NPTS]
DO i = 0,superinj_side-1
  theta = acos(SQRT(1 - random_num(rancount)))
  rancount =rancount +1
  phi = 2.0*pi*random_num(rancount)
  rancount =rancount +1
  rannum = V_TABLE_RES *random_num(rancount)
  rancount =rancount +1
  v = v_table(rannum)
  particle(num_part)%x = -length * random_num(rancount)
  rancount =rancount +1
!   particle(num_part)%y = -radius
  particle(num_part)%y = -radius*cos(-pi/4+phi/4.0) ! 13Mar,2010
!   particle(num_part)%z = width * random_num(rancount) - width / 2.0
  particle(num_part)%z = radius*sin(-pi/4+phi/4.0) ! 13Mar,2010
  rancount =rancount +1
!   particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!      *particle(num_part)%z)
particle(num_part)%r = radius
particle(num_part)%vx = v * SIN(theta) *SIN(phi)
particle(num_part)%vy = v * COS(theta)

```

```

particle(num_part)%vz = v*SIN(theta)*COS(phi)
particle(num_part)%Ex = 0.0
particle(num_part)%Ey = 0.0
particle(num_part)%q = q * super_side
particle(num_part)%m = m * super_side

! 12 Mar, 2010
particle(num_part)%celly=ngp1-2
CALL locatex_2(grid,ngp2,particle(num_part)%x,positionx,0)
particle(num_part)%cellx=positionx

num_part = num_part +1
if (num_part >= numpart) THEN
    write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
    STOP
END IF

!          TOP      Side      */

theta = acos(SQRT(1 - random_num(rancount)))
rancount =rancount +1
phi = 2.0*pi*random_num(rancount)
rancount =rancount +1
rannum = V_TABLE_RES *random_num(rancount)
rancount =rancount +1
v = v_table(rannum)
particle(num_part)%x = -length * random_num(rancount)
rancount =rancount +1
!   particle(num_part)%y = radius
particle(num_part)%y = radius*cos(-pi/4+phi/4.0) ! 13Mar,2010
!   particle(num_part)%z = width * random_num(rancount) - width / 2.0
particle(num_part)%z = radius*sin(-pi/4+phi/4.0) ! 13Mar,2010
rancount =rancount +1
!   particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!           *particle(num_part)%z)
particle(num_part)%r = radius
particle(num_part)%vx = v * SIN(theta) * SIN(phi)
particle(num_part)%vy = -v * COS(theta)
particle(num_part)%vz = v * SIN(theta) * COS(phi)
particle(num_part)%Ex = 0.0
particle(num_part)%Ey = 0.0
particle(num_part)%q = q * super_side
particle(num_part)%m = m * super_side

! 12 Mar, 2010
particle(num_part)%celly=ngp1-2
CALL locatex_2(grid,ngp2,particle(num_part)%x,positionx,0)
particle(num_part)%cellx=positionx

num_part = num_part +1

```

```

        if (num_part >= numpart) THEN
            write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
            STOP
        END IF

!           FRONT SIDE           */

        theta = acos(SQRT(1 - random_num(rancount)))
        rancount = rancount + 1
        phi = 2.0*pi*random_num(rancount)
        rancount = rancount + 1
        rannum = V_TABLE_RES * random_num(rancount)
        rancount = rancount + 1
        v = v_table(rannum)
        particle(num_part)%x = -length * random_num(rancount)
        rancount = rancount + 1
!       particle(num_part)%y = height * random_num(rancount) - height /
2.0
        particle(num_part)%y = radius*sin(-pi/4+phi/4.0) ! 13Mar,2010
        rancount = rancount + 1
!       particle(num_part)%z = radius
        particle(num_part)%z = radius*cos(-pi/4+phi/4.0) ! 13Mar,2010
!       particle(num_part)%r =
SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!       *particle(num_part)%z)
        particle(num_part)%r = radius
        particle(num_part)%vx = v * SIN(theta) * COS(phi)
        particle(num_part)%vy = v * SIN(theta) * SIN(phi)
        particle(num_part)%vz = -v * COS(theta)
        particle(num_part)%Ex = 0.0
        particle(num_part)%Ey = 0.0
        particle(num_part)%q = q * super_side
        particle(num_part)%m = m * super_side

        ! 12 Mar, 2010
        CALL locatex_2(grid,ngp2,particle(num_part)%x,positionx,0)
!       CALL locatey_2(grid,ngp1,abs(particle(num_part)%y),positiony,0)
        particle(num_part)%cellx=positionx
!       particle(num_part)%celly=positiony
        particle(num_part)%celly=ngp1-2

        num_part = num_part + 1
        if (num_part >= numpart) THEN
            write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
            STOP
        END IF

!           BACK SIDE           */

        theta = acos(SQRT(1 - random_num(rancount)))
        rancount = rancount + 1
        phi = 2.0*pi*random_num(rancount)

```

```

        rancount = rancount + 1
        rannum = V_TABLE_RES * random_num(rancount)
        rancount = rancount + 1
        v = v_table(rannum)
        particle(num_part)%x = -length * random_num(rancount)
        rancount = rancount + 1
!       particle(num_part)%y = height * random_num(rancount) - height /
2.0
        particle(num_part)%y = radius*sin(-pi/4+phi/4.0) ! 13Mar,2010
        rancount = rancount + 1
!       particle(num_part)%z = -radius
        particle(num_part)%z = -radius*cos(-pi/4+phi/4.0) ! 13Mar,2010
!       particle(num_part)%r =
        SQRT(particle(num_part)%y*particle(num_part)%y + particle(num_part)%z
&
!       *particle(num_part)%z)
        particle(num_part)%r = radius
        particle(num_part)%vx = v * SIN(theta) * COS(phi)
        particle(num_part)%vy = v * SIN(theta) * SIN(phi)
        particle(num_part)%vz = v * COS(theta)
        particle(num_part)%Ex = 0.0
        particle(num_part)%Ey = 0.0
        particle(num_part)%q = q * super_side
        particle(num_part)%m = m * super_side

        ! 12 Mar, 2010
        CALL locatex_2(grid,ngp2,particle(num_part)%x,positionx,0)
!       CALL locatey_2(grid,ngp1,abs(particle(num_part)%y),positiony,0)
!       particle(num_part)%cellx=positionx
!       particle(num_part)%celly=positiony
        particle(num_part)%celly=ngp1-2

        num_part = num_part + 1
        if (num_part >= numpart) THEN
            write(*,*) " *** Too many particles injected: increase
'numpart' in lprobe60.h *****"
            STOP
        END IF
    END DO

END SUBROUTINE inject
END MODULE INJECT_PART_1

```

LAPACK.FOR.

```

        MODULE LAPACK_1
*   http://www.netlib.org/lapack/lapack-3.1.1/html/sgbtrf.f.html
*   ... ..
*   http://www.netlib.org/lapack/lapack-3.1.1/html/lsame.f.html
*
        CONTAINS

```

```

*      .. External Functions ..
      INCLUDE '..\LAPACK_1\ILAENV.FOR'
      INCLUDE '..\LAPACK_1\ISAMAX.FOR'
*
*      ..
*      .. External Subroutines ..
      INCLUDE '..\LAPACK_1\SGBTRF.FOR'
      INCLUDE '..\LAPACK_1\SGBTFF2.FOR'
      INCLUDE '..\LAPACK_1\SGBTRS.FOR'
      INCLUDE '..\LAPACK_1\SCOPY.FOR'
      INCLUDE '..\LAPACK_1\SGEMM.FOR'
      INCLUDE '..\LAPACK_1\SGEMV.FOR'
      INCLUDE '..\LAPACK_1\SGER.FOR'
      INCLUDE '..\LAPACK_1\SLASWP.FOR'
      INCLUDE '..\LAPACK_1\SSCAL.FOR'
      INCLUDE '..\LAPACK_1\SSWAP.FOR'
      INCLUDE '..\LAPACK_1\STRSM.FOR'
      INCLUDE '..\LAPACK_1\STBSV.FOR'
      INCLUDE '..\LAPACK_1\XERBLA.FOR'
      INCLUDE '..\LAPACK_1\IEEECK.FOR'
      INCLUDE '..\LAPACK_1\IPARMQ.FOR'
      INCLUDE '..\LAPACK_1\LSAME.FOR'

      END MODULE LAPACK_1

```

LOCATE.F90.

```

! -----
MODULE LOCATE_1
! -----
USE LPROBE60_1

CONTAINS

SUBROUTINE locatex_2(grid,n,x,jj,jStart)
! -----
! - - - arg types - - -
!TYPE (GRIDDEF ) grid(0:ngp1-1,0:ngp2-1)
TYPE (GRIDDEF ) grid(0:,0:)
INTEGER :: n,jj,jStart
REAL x
! - - - local declarations - - -
INTEGER :: ju,jm,jl

! - - - begin - - -

! // exceeds upper limit
if (x > grid(0,n-1)%x) THEN
  jj=n+1

```

```

        return
    END IF

! exceeds lower limit
    if (x < grid(0, 0)%x) THEN
        jj=n+1
        return
    ENDIF

    jl = 0
    ju = n+1 ! // n intervals, (n+1) division points
    jm = jStart

    if (x > grid(0,jStart)%x) THEN
        jl=jm
    else
        ju=jm
    ENDIF

    DO WHILE (ju-jl > 1)
        jm=(ju+jl)/2 ! jm = ISHFT((ju+jl), -1)
        IF (x > grid(0,jm)%x) THEN
            jl = jm
        ELSE
            ju = jm
        END IF
    END DO
    jj = jl

END SUBROUTINE    locatex_2

SUBROUTINE locatey_2(grid,n,y,jj,jStart)
! -----
! - - - arg types - - -
!TYPE (GRIDDEF ) grid(0:ngp1-1,0:ngp2-1)
TYPE (GRIDDEF ) grid(0:,0:)
INTEGER :: n,jj,jStart
REAL y
! - - - local declarations - - -
INTEGER :: ju,jm,jl

! - - - begin - - -
! // exceeds upper limit
    if (y > grid(n-1,0)%y) THEN
        jj=n+1
        return
    END IF

! exceeds lower limit
    if (y < grid(0, 0)%y) THEN
        jj=n+1
        return
    ENDIF

```

```

jl = 0
ju = n+1 ! // n intervals, (n+1) division points
jm = jStart

if (y > grid(jStart,0)%y) THEN
    jl=jm
else
    ju=jm
ENDIF

DO WHILE (ju-jl > 1)
    jm=(ju+jl)/2 ! jm = ISHFT((ju+jl), -1)
    IF (y > grid(jm,0)%y) THEN
        jl = jm
    ELSE
        ju = jm
    END IF
END DO
jj = jl

END SUBROUTINE locatey_2

! -----
SUBROUTINE locatex(xx,n,x,jj)
! -----
! - - - arg types - - -
TYPE (GRIDDEF ) xx(0:ngp1-1,0:ngp2-1)
INTEGER :: n,jj
REAL x
! - - - local declarations - - -
INTEGER :: ju,jm,jl
LOGICAL ascnd
! - - - begin - - -
jl = 0
ju = n+1 ! // n intervals, (n+1) division points
!ju = n
ascnd = 0
IF (xx(0,n-1)%x > xx(0,1)%x) ascnd = 1
DO WHILE (ju-jl > 1)
    jm=(ju+jl)/2 ! jm = ISHFT((ju+jl), -1)
    IF (x > xx(0,jm)%x .EQV. ascnd) THEN
        jl = jm
    ELSE
        ju = jm
    END IF
END DO
jj = jl

END SUBROUTINE locatex

! -----
SUBROUTINE locatey(xx,n,x,jj)
! -----

```



```

! - - - arg types - - -
TYPE (GRIDDEF ) xx(0:ngp1-1,0:ngp2-1)
INTEGER :: n,jj
REAL x
! - - - local declarations - - -
INTEGER :: ju,jm,jl
LOGICAL ascnd

! - - - begin - - -
jl = 0
ju = n+1 ! // n intervals, (n+1) division points
!ju = n
ascnd = 0
IF (xx(n-1,0)%y > xx(1,0)%y) ascnd = 1

DO WHILE (ju-jl > 1)
  jm=(ju+jl)/2 ! jm = ISHFT((ju+jl),-1)
  IF (x > xx(jm,0)%y .EQV. ascnd) THEN
    jl = jm
  ELSE
    ju = jm
  END IF
END DO
jj = jl

END SUBROUTINE locatey
END MODULE LOCATE_1

```

LPROBE60.F90.

```

! -----
MODULE LPROBE60_1
! -----
! USE MSIMSL          ! needed for ERF()      in make_velocity_table
!USE MSFLIB
! TYPE :: GRIDDEF_T
  TYPE :: GRIDDEF
    REAL(4) :: x
    REAL(4) :: y
    REAL(4) :: q_dens
    REAL(4) :: Ey
    REAL(4) :: Ex
    REAL(4) :: phi
  END TYPE GRIDDEF

! TYPE :: PARTICLEDEF_T
  TYPE :: PARTICLEDEF
    REAL(4) :: x
    REAL(4) :: y
    REAL(4) :: z
    REAL(4) :: r
    REAL(4) :: vx

```

```

REAL(4) :: vy
REAL(4) :: vz
REAL(4) :: Ex
REAL(4) :: Ey
REAL(4) :: q
REAL(4) :: m
INTEGER :: cellx
INTEGER :: celly
END TYPE PARTICLEDEF

! TYPE (GRIDDEF_T) :: GRIDDEF

! TYPE (PARTICLEDEF_T) :: PARTICLEDEF

INTEGER,PARAMETER :: noBins = 100
INTEGER,PARAMETER :: outputFrequency = 100

REAL,PARAMETER :: FLT_MAX = 1.0E+37
REAL :: maxVe = -FLT_MAX ! max. electron velocity [m/s]
REAL :: minVe = FLT_MAX ! =1E37min. electron velocity [m/s]
REAL :: maxVi = -FLT_MAX ! max. ion velocity [m/s]
REAL :: minVi = FLT_MAX ! =1E37min. ion velocity [m/s]
REAL :: wpdtMax=FLT_MAX ! estimate max. possible timestep parameter

INTEGER hitIons, hitElectrons ! num of particles that hit the probe in
'dti' time interval
! average current on a dti/dte time step
REAL avgc_dti ,avgc_dte
REAL ion_charge, elec_charge ! latest discharge on the probe

REAL,PARAMETER :: EPS0 = 8.85e-12
REAL,PARAMETER :: EPSA103 =10.44e-12
REAL,PARAMETER :: pi = 3.14159265359
!INTEGER,PARAMETER :: ngp = 5476 ! = ngp1 x ngp2 = 74 x 74
INTEGER ngp ! = ngp1 x ngp2
!INTEGER,PARAMETER :: ngp1= 74
INTEGER ngp1 ! num of grid points on the vertical direction
!INTEGER,PARAMETER :: ngp2= 74
INTEGER,PARAMETER :: ngp2= (4*18+1) ! num of grid points on the
horizontal directionn (OX)
! this form is required to divide both
the probe length & domain length in an integer num. of cells
INTEGER ncpr ! input parameter: num. of cells per probe radius
INTEGER noRows15 ! will replace the first 15*ncpr rows, set by
Markusic

!INTEGER,PARAMETER :: numpart= 40000 ! will be exceeded in
INJECT_PART
!INTEGER,PARAMETER :: numpart= 100000 ! will be exceeded in
INJECT_PART

```

```

INTEGER,PARAMETER :: numpart= 299000 ! increased from 100,000 to
300,000 when running m_ion/m_elec=100

REAL,PARAMETER :: boltz= 1.38e-23
REAL,PARAMETER :: RAD= 0.00378864
INTEGER,PARAMETER :: IA= 16807
INTEGER,PARAMETER :: IM= 2147483647
REAL,PARAMETER :: AM= (1.0/IM)
INTEGER,PARAMETER :: IQ= 127773
INTEGER,PARAMETER :: IR= 2836
INTEGER,PARAMETER :: NTAB= 32
INTEGER,PARAMETER :: NDIV= (1+(IM-1)/NTAB)
REAL,PARAMETER :: EPS= 1.2e-7
REAL,PARAMETER :: RNMX= (1.0-EPS)
INTEGER,PARAMETER :: NR_END= 1
CHARACTER,POINTER :: FREE_ARG

INTEGER,PARAMETER :: NPTS= 1000005 ! # of data points
!INTEGER,PARAMETER :: NPTS= numpart ! # of data points
INTEGER,PARAMETER :: V_TABLE_RES= numpart
!INTEGER,PARAMETER :: V_TABLE_RES= 1000

! /* PARTICLE GLOBAL VARIABLES */
TYPE (PARTICLEDEF),POINTER :: ion(:),electron(:)

!/* GRID/GEOMETRY GLOBAL VARIABLES */
INTEGER ngp_vert,ngp_horiz,jprobe,kprobe,jinsul,kinsul
REAL
cell_height1,cell_height2,cell_height3,cell_height4,cell_height5,cell_h
eight6,cell_height,cell_width,r_probe,r_insul,l_probe,l_insul,radius,le
ngth
!TYPE (GRIDDEF) :: grid(0:ngp1,0:ngp2)
TYPE (GRIDDEF),POINTER :: grid(:, :)

!/* FIELD SOLVER GLOBAL VARIABLES */
INTEGER test1,test2,test3,info,nrhs

INTEGER,POINTER :: piv(:)
! ALLOCATE( piv(0:ngp)) ! type of INTEGER
!INTEGER piv(0:ngp) ! pointer type not supported by LAPACK's
routine SGBTRF
!INTEGER piv(:) ! pointer type not supported by LAPACK's
routine SGBTRF

!**already in GRID/GEOMETRY GLOBAL VARIABLES** INTEGER(8)
jprobe,kprobe,kinsul
!REAL band1(0:ngp-1),band2(0:ngp-1),band3(0:ngp-1),band4(0:ngp-
1),band5(0:ngp-1)
REAL,POINTER :: band1(:),band2(:),band3(:),band4(:),band5(:)
REAL,POINTER :: b(:),phi(:)

! REAL,POINTER :: abd(:)
! ALLOCATE( abd(0:(2*lower_width+upper_width+2)*ngp) ) ! type of REAL
INTEGER :: ldab != 3*ngp2+1

```

```

INTEGER    n1 ! = ngp
INTEGER    m1 ! = ngp
INTEGER    ldb ! = ngp
INTEGER :: upper_width ! = ngp2
INTEGER :: lower_width ! = ngp2
!REAL :: abd(0:(2*lower_width+upper_width+2)*ngp ) ! pointer type not
supported by LAPACK's routine SGBTRF
REAL,POINTER :: abd(:) ! pointer type not supported by LAPACK's
routine SGBTRF

!/* PLASMA PARAMETER GLOBAL VARIABLES */
REAL  T,n,m_ion,q_ion,m_elec,q_elec,m_part,plasma_pot
REAL  mi_me_ratio ! to be entered by the user

!/* INJECTION SETUP GLOBAL VARIABLES */
INTEGER
injsuperi,injsupere,superii_init,superii_back,superii_front,superii_sid
e,superei_init, &
    superei_back,superei_front,superei_side,rancount,elec_iter
!/* 25 Mar 2010 */
REAL superii_back_FLOAT,superei_back_FLOAT,superii_front_FLOAT, &
    superei_front_FLOAT,superii_side_FLOAT,superei_side_FLOAT

!/* EXTERN long int super_beami,ion_sub_inject; Those are for
<inject_beam.c> */
REAL  sigma,sigmae,sigmai,wpdt,wp,dt,height,width,dti,dte, &
    supere_back,supere_front,supere_side, &
    superi_back,superi_front,superi_side
REAL h ! // not used in calculations; set in 'parameter.f90'
REAL supere,superi,superi_init,supere_init
REAL,POINTER :: v_table_e(:),v_table_i(:),random_num(:)

!/* COUNTING GLOBAL VARIABLES */
INTEGER
tom,numelec,numion,t_count,t_count2,num,ioncount,ioniter,ion_move_count
,ecounter

!/* BOUNDARY GLOBAL VARIABLES */
REAL
V,current,VOLUME,avecurent,curri,curre,avecurre,ins_charge

!/* TESTING VARIABLES */
INTEGER t1,t2
REAL,POINTER :: vxdata
INTEGER x,y,z,c,n2

!*****
!/* GRAPHICS GLOBAL VARIABLES */
REAL percentAnimation ! percent of particles displayed in animation,
0.0,...,100.0 ; 0.0 means NO animation
INTEGER cursorY; ! used for info text display

INTEGER view, page

```

```

!REAL,PARAMETER :: xButtonOffset= 0.15
!REAL,PARAMETER :: yButtonOffset =0.08
!REAL,PARAMETER :: SWITCHBOXx= -0.07
!REAL,PARAMETER :: SWITCHBOXy= -0.85
!REAL,PARAMETER :: EXITBOXx= 0.84
!REAL,PARAMETER :: EXITBOXy= SWITCHBOXy
!REAL,PARAMETER :: PRINTBOXx 0.65
!REAL,PARAMETER :: PRINTBOXy= SWITCHBOXy

! TYPE :: PUSHBUTTON_T
!   REAL(4) :: x1PositionUP
!   REAL(4) :: x2PositionUP
!   REAL(4) :: y1PositionUP
!   REAL(4) :: y2PositionUP
!!   CHARACTER (LEN=20) :: ButtonText
!   INTEGER(2) :: TextColor
!   INTEGER(2) :: aBorderColorUp
!   INTEGER(2) :: bBorderColorUp
!   INTEGER(2) :: cBorderColorUp
!   INTEGER(2) :: dBorderColorUp
!   INTEGER(2) :: aBorderColorDown
!   INTEGER(2) :: bBorderColorDown
!   INTEGER(2) :: cBorderColorDown
!   INTEGER(2) :: dBorderColorDown
!   INTEGER(2) :: FillColorUp
!   INTEGER(2) :: FillColorDown
! END TYPE
! TYPE (PUSHBUTTON_T) :: PushButton

! /* Environment Variables */
#define black 0
#define white 15
#define gray 50
#define gray2 56
#define lightgray 56
#define darkorange 51
#define midblue 52
#define red 53
#define gold 55

! *****

END MODULE ! LPROBE60_1

```

MAKE_LU.F90.

```

! -----
MODULE MAKE_LU_1
! -----

```

```

!      This function computes the LU decomposition of the band matrix
generated
!      by differencing Poisson's Equation on the grid  */
! page 126

USE LPROBE60_1
USE LAPACK_1

CONTAINS

! -----
SUBROUTINE make_LU(array)
! -----
! - - - arg types - - -
TYPE (GRIDDEF ) array(0:ngp1-1,0:ngp2-1)
! - - - local declarations - - -
INTEGER :: i,j,k,jmax,kmax,count,counter1,counter2
INTEGER :: counter3,counter4,counter5
INTEGER :: info
!INTEGER :: SIZEOF_FLOAT = 4
!INTEGER :: SIZEOF_INT = 4

INTEGER :: abddata=100
INTEGER :: abddata2=200

REAL      dz,      dzPlus2, dzMinus2
REAL      rPlus2,   rMinus2,drPlus2,drMinus2,drSqr
!  INTEGER auxMat(0:7),shift      ! N*LDBA-LDBA*LDBA=LDBA*(N-LDBA)
INTEGER shift
INTEGER,POINTER :: auxMat(:)      ! N*LDBA-LDBA*LDBA=LDBA*(N-LDBA)

!  INTEGER abdEqv(1:2*lower_width+upper_width+2,1:ngp )
!  INTEGER abdEqv(1:(2*lower_width+upper_width+2)*ngp )
!  EQUIVALENCE (abdEqv,abd(1))

! - - - begin - - -
!  abddata = FOPEN("abddata.dat","w+")
!  abddata2 = FOPEN("abddata2.dat","w+")
jmax = ngp_vert
kmax = ngp_horiz
counter1 = upper_width-1
counter2 = 0
counter3 = -1
counter4 = 0
counter5 = 0
!      **** */
!      Initialize all non-relevant positions to zero  */

DO i = 0,ngp-1
    band1(i)=0.0
    band2(i)=0.0
    band3(i)=0.0

```

```

        band4(i)=0.0
        band5(i)=0.0
    END DO

DO j = 0, jmax-1
    DO k = 0, kmax-1
        IF (counter1 < (ngp-1)) THEN
            counter1 = counter1+1
        END IF
        IF (counter2 < (ngp-1)) THEN
            counter2 = counter2+1
        END IF
        counter4 = counter4+1
        counter5 = counter5+1
!   y < r_probe      */
        IF (j <= jprobe) THEN
            if (j==0 .AND. k==1) counter4=0
            if (j==1 .AND. k==0) counter5=0

            if( k < kprobe) THEN
                IF (j == 0) THEN ! /* Lower boundary      */
!                   IF (k == 1) THEN
!                       IF (k == 0) THEN
!                           band1(counter1)=0.0
!                           band2(counter2)=0.0
!                           band4(counter4)=0.0
!                           band5(counter5)=0.0
!                       ELSE ! k==1
!                           band1(counter1)=pi*cell_width

band2(counter2)=pi*array(j+1,k)%y*array(j+1,k)%y/(4.0*cell_width)

band4(counter4)=pi*array(j+1,k)%y*array(j+1,k)%y/(4.0*cell_width)

!
band2(counter2)=pi*array(k,j+1)%y*array(k,j+1)%y/(4.0*cell_width)
!
band4(counter4)=pi*array(k,j+1)%y*array(k,j+1)%y/(4.0*cell_width)
band5(counter5)=0.0
                END IF ! k==1
                ELSE ! j==0 !      Rest to top of probe      */
                IF (k == 0) THEN
                    band1(counter1)=0.0
                    band2(counter2)=0.0
                    band4(counter4)=0.0
                    band5(counter5)=0.0
                ELSE ! k==0
                    dz = (array(j+1,k)%x-array(j-1,k)%x)/2.0 !
(4.37)+(4.32)+(4.31)
                    dz = cell_width !
(4.37)+(4.32)+(4.31)
                    dzPlus2 = array(j+1,k)%x-array(j,k)%x ! (4.38)
                    dzMinus2 = array(j,k)%x-array(j-1,k)%x ! (4.39)

```

```

(4.30)      rPlus2 = (array(j+1,k)%y+array(j,k)%y)/2.0      !
(4.31)      rMinus2 = (array(j-1,k)%y+array(j,k)%y)/2.0    !
            drPlus2 = rPlus2 - rMinus2
            ! (4.35)
(4.35)      drPlus2 = array(j+1,k)%y-array(j,k)%y          !
(4.36)      drMinus2 = array(j,k)%y-array(j-1,k)%y         !
            drSqr = rPlus2*rPlus2 - rMinus2*rMinus2        ! (4.34)

!           dz = (array(k,j+1)%x-array(k,j-1)%x)/2.0 !
(4.37)+(4.32)+(4.31)
!           dz = cell_width                                !
(4.37)+(4.32)+(4.31)
!           dzPlus2 = array(k,j+1)%x-array(k,j)%x         ! (4.38)
!           dzMinus2 = array(k,j)%x-array(k,j-1)%x        ! (4.39)
!
!           rPlus2 = (array(k,j+1)%y+array(k,j)%y)/2.0    !
(4.30)      rMinus2 = (array(k,j-1)%y+array(k,j)%y)/2.0    !
(4.31)      drPlus2 = rPlus2 - rMinus2
            ! (4.35)
(4.35)      drPlus2 = array(k,j+1)%y-array(k,j)%y          !
(4.36)      drMinus2 = array(k,j)%y-array(k,j-1)%y         !
            drSqr = rPlus2*rPlus2 - rMinus2*rMinus2        ! (4.34)

            if ((array(j+1,k)%y-array(j,k)%y) == 0.0) then
                write(*,*) 'zero denominator'
            end if
!           band1(counter1)=2.0*pi*dz*rPlus2/drPlus2      ! (4.45)

            band1(counter1)=2.0*pi*cell_width*(array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0) &
                                /(array(j+1,k)%y-array(j,k)%y)

!           band2(counter2)=pi*drSqr/dzPlus2              !
(4.47)      band2(counter2)=pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) &
                                -(array(j,k)%y-(array(j,k)%y-
array(j-1,k)%y)/2.0)*(array(j,k)%y-(array(j,k)%y-array(j-
1,k)%y)/2.0))/cell_width

!           band2(counter2)=pi*drSqr/dzMinus2              !
(4.48)      band4(counter4)=pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) &

```



```

                                -(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-(array(j,k)%y-array(j-
1,k)%y)/2.0))/cell_width

                                if ((array(j,k)%y-array(j-1,k)%y) == 0.0) then
                                    write(*,*) 'zero denominator'
                                end if
!                                band1(counter1)=2.0*pi*dz*rMinus2/drMinus2    ! (4.46)
                                band5(counter5)=2.0*pi*cell_width*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0) &
                                    /(array(j,k)%y-array(j-1,k)%y)

!                                !band1(counter1)=2.0*pi*dz*rPlus2/drPlus2    ! (4.45)
!
                                band1(counter1)=2.0*pi*cell_width*(array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0) &
                                    /(array(k,j+1)%y-array(k,j)%y)
!
!                                !band2(counter2)=pi*drSqr/dzPlus2                !
(4.47)
!                                band2(counter2)=pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) &
!                                    -(array(k,j)%y-(array(k,j)%y-
array(k,j-1)%y)/2.0)*(array(k,j)%y-(array(k,j)%y-array(k,j-
1)%y)/2.0))/cell_width
!
!                                !band2(counter2)=pi*drSqr/dzMinus2                !
(4.48)
!                                band4(counter4)=pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) &
!                                    -(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-(array(k,j)%y-
array(k,j-1)%y)/2.0))/cell_width
!
!                                !band1(counter1)=2.0*pi*dz*rMinus2/drMinus2    !
(4.46)
!                                band5(counter5)=2.0*pi*cell_width*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0) &
!                                    /(array(k,j)%y-array(k,j-1)%y)
                                END IF    ! k==0
                                END IF ! j==0
                                ELSE ! k<probe /*      on and inside probe      */
                                    band1(counter1)=0.0
                                    band2(counter2)=0.0
                                    band4(counter4)=0.0
                                    band5(counter5)=0.0
                                END IF ! k < kprobe
                                ELSE IF (j > jprobe .AND. j < (jmax-1)) THEN
                                    IF ((k > 0) .AND. (k < (kmax-1))) THEN
                                        if ((array(j+1,k)%y-array(j,k)%y) == 0.0) then
                                            write(*,*) 'zero denominator'
                                        end if

```

```

        band1(counter1)=2.0*pi*cell_width*(array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0) &
                                /(array(j+1,k)%y-array(j,k)%y)
        band2(counter2)=pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) &
                                - (array(j,k)%y-(array(j,k)%y-
array(j-1,k)%y)/2.0)*(array(j,k)%y-(array(j,k)%y-array(j-
1,k)%y)/2.0))/cell_width
        band4(counter4)=pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) &
                                - (array(j,k)%y-(array(j,k)%y-
array(j-1,k)%y)/2.0)*(array(j,k)%y-(array(j,k)%y-array(j-
1,k)%y)/2.0))/cell_width
        if ((array(j,k)%y-array(j-1,k)%y) == 0.0) then
            write(*,*) 'zero denominator'
        end if
        band5(counter5)=2.0*pi*cell_width*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0) &
                                /(array(j,k)%y-array(j-1,k)%y)

!
        band1(counter1)=2.0*pi*cell_width*(array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0) &
!
!                                /(array(k,j+1)%y-array(k,j)%y)
!        band2(counter2)=pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) &
!
!                                - (array(k,j)%y-(array(k,j)%y-
array(k,j-1)%y)/2.0)*(array(k,j)%y-(array(k,j)%y-array(k,j-
1)%y)/2.0))/cell_width
!        band4(counter4)=pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) &
!
!                                - (array(k,j)%y-(array(k,j)%y-
array(k,j-1)%y)/2.0)*(array(k,j)%y-(array(k,j)%y-array(k,j-
1)%y)/2.0))/cell_width
!        if ((array(k,j)%y-array(k,j-1)%y) == 0.0) then
!            write(*,*) 'zero denominator'
!        end if
!        band5(counter5)=2.0*pi*cell_width*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0) &
!
!                                /(array(k,j)%y-array(k,j-1)%y)

ELSE IF (k == 0) THEN
    band1(counter1)=0.0
    band2(counter2)=0.0
    band4(counter4)=0.0
    band5(counter5)=0.0
ELSE
    if ((array(j+1,k)%y-array(j,k)%y) == 0.0) then
        write(*,*) 'zero denominator'
    end if

```

```

        band1(counter1)=2.0*pi*(
array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0 )*cell_width &
        /(array(j+1,k)%y-array(j,k)%y)
        band2(counter2)=0.0
        band4(counter4)=pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) &
        -(array(j,k)%y-(array(j,k)%y-
array(j-1,k)%y)/2.0)*(array(j,k)%y-(array(j,k)%y-array(j-
1,k)%y)/2.0))/cell_width
        if ((array(j,k)%y-array(j-1,k)%y) == 0.0) then
            write(*,*) 'zero denominator'
        end if
        band5(counter5)=2.0*pi*( array(j,ngp_horiz-1)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0 )*cell_width &
        /(array(j,k)%y-array(j-1,k)%y)

!           if ((array(k,j+1)%y-array(k,j)%y) == 0.0) then
!               write(*,*) 'zero denominator'
!           end if
!           band1(counter1)=2.0*pi*(
array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0 )*cell_width &
!               /(array(k,j+1)%y-array(k,j)%y)
!           band2(counter2)=0.0
!           band4(counter4)=pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) &
!               -(array(k,j)%y-(array(k,j)%y-
array(k,j-1)%y)/2.0)*(array(k,j)%y-(array(k,j)%y-array(k,j-
1)%y)/2.0))/cell_width
!           if ((array(k,j)%y-array(k,j-1)%y) == 0.0) then
!               write(*,*) 'zero denominator'
!           end if
!           band5(counter5)=2.0*pi*( array(ngp_horiz-1,j+1)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0 )*cell_width &
!               /(array(k,j)%y-array(k,j-1)%y)
        END IF
    ELSE ! j > jprobe .AND. j <(jmax-1)
!           band1(counter1)=0.0
!           band2(counter2)=0.0
!           band4(counter4)=0.0
!           band5(counter5)=0.0

    END IF
END DO
END DO

! /* Diagonal */

DO j = 0,jmax-1
!     y < r_probe */
    IF (j <= jprobe) THEN
        DO k = 0,kmax-1
            counter3 = counter3+1
            IF (k >= 0 .AND. (k < kprobe)) THEN
                IF (j == 0) THEN
                    IF (k == 0) THEN

```

```

band3(counter3)=-1.0
ELSE
band3(counter3)=-pi*(cell_width+(array(j+1,k)%y-
array(j,k)%y)*(array(j+1,k)%y-array(j,k)%y)/(2.0*cell_width))
!
band3(counter3)=-pi*(cell_width+(array(k,j+1)%y-
array(k,j)%y)*(array(k,j+1)%y-array(k,j)%y)/(2.0*cell_width))
END IF ! k == 0
ELSE ! j==0
IF (k == 0) THEN
band3(counter3)=-1.0
ELSE
if ((array(j,k)%y-array(j-1,k)%y) == 0.0) then
write(*,*) 'zero denominator'
end if
band3(counter3)=-
(2.0*pi*cell_width*(array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)/(array(j+1,k)%y-array(j,k)%y) +
pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) -
(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0))/cell_width +
pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) -
(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0))/cell_width +
2.0*pi*cell_width*(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0) &
/(array(j,k)%y-array(j-1,k)%y))
!
if ((array(k,j)%y-array(k,j-1)%y == 0.0) .OR.
(array(k,j+1)%y-array(k,j)%y == 0.0)) then
!
write(*,*) 'zero denominator'
!
end if
!
band3(counter3)=-
(2.0*pi*cell_width*(array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)/(array(k,j+1)%y-array(k,j)%y) &
!
+
pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) -
(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0))/cell_width &
!
+
pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) -
(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0))/cell_width &
!
+
2.0*pi*cell_width*(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0) &
!
/(array(k,j)%y-array(k,j-1)%y))
END IF ! k == 0
END IF ! j == 0
ELSE ! (k >= 0 .AND. (k < kprobe)
band3(counter3)=-1.0
END IF ! (k >= 0 .AND. (k < kprobe)
END DO

```

```

!      y > r_probe */
      ELSE IF (j < (jmax-1)) THEN
        DO k = 0, kmax-1
          counter3 = counter3+1
          IF (k > 0 .AND. (k < (kmax-1)) .AND. (j < (jmax-1))) THEN
            if ((array(j,k)%y-array(j-1,k)%y) == 0.0) then
              write(*,*) 'zero denominator'
            end if
            band3(counter3)=-
(2.0*pi*cell_width*(array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)/(array(j+1,k)%y-array(j,k)%y) +
pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) -
(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0))/cell_width +
pi*((array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) -
(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0))/cell_width +
2.0*pi*cell_width*(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0) &
/(array(j,k)%y-array(j-1,k)%y))

!      if ((array(k,j)%y-array(k,j-1)%y == 0.0) .OR.
(array(k,j+1)%y-array(k,j)%y == 0.0)) then
!      write(*,*) 'zero denominator'
!      end if
!      band3(counter3)=-
(2.0*pi*cell_width*(array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)/(array(k,j+1)%y-array(k,j)%y) &
!      +
pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) -
(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0))/cell_width &
!      +
pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) -
(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0))/cell_width &
!      +
2.0*pi*cell_width*(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0) &
!      /(array(k,j)%y-array(k,j-1)%y))

      ELSE IF (k == 0) THEN
        band3(counter3)=-1.0
      ELSE
        if (array(j,k)%y-array(j-1,k)%y == 0.0 .OR.
array(j+1,k)%y-array(j,k)%y == 0.0) then
          write(*,*) 'zero denominator'
        end if
        band3(counter3)=-
(2.0*pi*cell_width*(array(j,k)%y+(array(j+1,k)%y-
array(j,k)%y)/2.0)/(array(j+1,k)%y-array(j,k)%y) &
+
pi*((array(j,k)%y+(array(j+1,k)%y-

```

```

array(j,k)%y)/2.0)*(array(j,k)%y+(array(j+1,k)%y-array(j,k)%y)/2.0) -
(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0)*(array(j,k)%y-
(array(j,k)%y-array(j-1,k)%y)/2.0))/cell_width &
+
2.0*pi*cell_width*(array(j,k)%y-(array(j,k)%y-array(j-1,k)%y)/2.0) &
/(array(j,k)%y-array(j-1,k)%y))

!           if ((array(k,j)%y-array(k,j-1)%y) == 0.0) then
!               write(*,*) 'zero denominator'
!           end if
!           band3(counter3)=-
(2.0*pi*cell_width*(array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)/(array(k,j+1)%y-array(k,j)%y) &
+
pi*((array(k,j)%y+(array(k,j+1)%y-
array(k,j)%y)/2.0)*(array(k,j)%y+(array(k,j+1)%y-array(k,j)%y)/2.0) -
(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0)*(array(k,j)%y-
(array(k,j)%y-array(k,j-1)%y)/2.0))/cell_width &
+
2.0*pi*cell_width*(array(k,j)%y-(array(k,j)%y-array(k,j-1)%y)/2.0) &
/(array(k,j)%y-array(k,j-1)%y))
!
END IF ! k == 0
END DO
ELSE ! j < (jmax-1)
DO k = 0, kmax-1
counter3 = counter3+1
band3(counter3)=-1.0
END DO
END IF ! j < (jmax-1)
END DO

! Form band vector 'abd' to be submitted to the band solver */

!http://www.intel.com/software/products/mkl/docs/webhelp/appendices/mkl
_appB_MA.html
!http://www.viswiki.com/en/Packed_storage_matrix
!
! The band storage scheme for LU factorization is a more compact way
than an m-by-n rectangular
! array by exploiting a special structure of the matrix.
! These are most notably used in BLAS and LAPACK.
! In Fortran, column-major ordering of storage is assumed: elements of
the same column occupy successive storage locations.
! In C/C++ , line ordering storage is assumed. Therefore, an array
transpose is needed when translating
! a program from FORTRAN to C/C++, or viceversa
!
!
! An example of packed storage for banded matrix:
!
! real:: A(m,n) ! a banded matrix with kl subdiagonals and ku
superdiagonals
! real:: AP(-kl:ku,n) ! packed storage for A

```

```

!      ! the band of A is stored column-by-column in AP. Some elements
of AP are unused.
!      ! unpacking the matrix AP to A
!      do j=1,n
!          forall(i=max(1,j-kl):min(m,j+ku)) A(i,j) = AP(i-j,j)
!      end do
!      print *,AP(0,:) ! the diagonal

count = -1
DO i = 0,ngp-1
    DO j = 1,(2*lower_width+upper_width+1) ! columns
        count = count+1 ! rows
        IF (j == lower_width+1) THEN
            abd(count)=band1(i)
        ELSE IF (j ==(lower_width+upper_width)) THEN
            abd(count)=band2(i)
        ELSE IF (j ==(lower_width+upper_width+1)) THEN
            abd(count)=band3(i)
        ELSE IF (j ==(lower_width+upper_width+2)) THEN
            abd(count)=band4(i)
        ELSE IF (j ==(lower_width+upper_width+2+lower_width-1)) THEN
            abd(count)=band5(i)
        ELSE
            abd(count)=0.0
        END IF
    END DO
END DO

goto 100 ! skip transposing

! If the C/C++ arrangement is
! 11 12 13 14 | 15 16
! 21 22 23 24 | 25 26
! 31 32 33 34 | 35 36
! 41 42 43 44 | 45 46
!
! then the FORTARN arrangement as a 1-dim array should be:
!
! |11 21 31 41 | 12 22 32 42 | 13 23 33 43 | 14 24 34 44 | 15 25 35 45
| 16 26 36 46|
!
! which is the equivalent of the following 2-dim array
! 11 21 31 41
! 12 22 32 42
! 13 23 33 43
! 14 24 34 44
! -----
! 15 25 35 45
! 16 26      36 46
!
!that is the transposed matrix.
! Below is the algorithm for transposing LDABxN matrices.

! Generate the test matrix above:

```

```

!n1=6
!LDAB=4
!do i=0,4-1      !LDAB-1; LDAB=4
!do j=0,6-1      !N-1    ; N=6
!abd(6*i+j) = (i+1)*10+j+1
!end do
!end do

! 1. transpose the square part
! 11 21 31 41 | 15 16
! 12 22 32 42 | 25 26
! 13 23 33 43 | 35 36
! 14 24 34 44 | 45 46
!do i=0,4-1      !LDAB-1
!do j=i,4-1      !LDAB-1
!aux=abd(6*i+j)
!abd(6*i+j)=abd(6*j+i)
!abd(6*j+i) = aux
do i=0,LDAB-1    !LDAB-1
do j=i,LDAB-1    !LDAB-1
aux=abd(n1*i+j)
abd(n1*i+j)=abd(n1*j+i)
abd(n1*j+i) = aux
end do
end do
! The 1-dim array result after transpose is:
! |11 21 31 41 15 16| 12 22 32 42 25 26| 13 23 33 43 35 36| 14 24 34
44 45 46|

! 2. transpose the non-square part and save. In 1-dim storage this is
! |15 25 35 45 | 16 26 36 46 |
!ALLOCATE( auxMat(0:4*(6-4)-1)) ! LDAB*(N-LDAB)
!do i=0,4-1 ! LDAB-1
!do j=4,6-1 ! LDAB,...,N-1
!auxMat(4*(j-4)+i) = abd(6*i+j) ! LDAB=4, N=6
ALLOCATE( auxMat(0:LDAB*(n1-LDAB)-1)) ! LDAB*(N-LDAB)
do i=0,LDAB-1 ! LDAB-1
do j=LDAB,n1-1 ! LDAB,...,N-1
auxMat(LDAB*(j-LDAB)+i) = abd(n1*i+j) ! LDAB=4, N=6
end do
end do

! 3. re-position(shift to left) the elements in the 1-dim array:
!shift = 6-4 ! N-LDAB
!do i=1,4-1 ! 1,...,LDAB-1
!do j=0,4-1 ! 1,...,LDAB-1
!abd(6*i+j-i*shift)=abd(6*i+j)
shift = n1-LDAB ! N-LDAB
do i=1,LDAB-1 ! 1,...,LDAB-1
do j=0,LDAB-1 ! 1,...,LDAB-1
abd(n1*i+j-i*shift)=abd(n1*i+j)
end do
end do
! The 1-dim array result after shift is:

```



```

!   |11 21 31 41 | 12 22 32 42 | 13 23 33 43 | 14 24 34 44 | * * * * |
!
! 4. copy the saved part to the end
!count1=4*(6-4) ! the dimension of the saved part
!do i=1, count1
!abd(4*4-1+i)=auxMat(i-1)
count1=LDAB*(n1-LDAB) ! the dimension of the saved part
do i=1, count1
abd(LDAB*LDAB-1+i)=auxMat(i-1)
end do

DEALLOCATE(auxMat)

! end transpose process


100 WRITE (*,901) " Going into LU factorization..."

!      Call LAPACK subroutine to perform the LU factorization      */
! http://www.nag.co.uk/numeric/F1/manual/pdf/F07/f07bdf.pdf
!
http://www.intel.com/software/products/mkl/docs/WebHelp/lle/funcn\_gbtrf.html
! http://www.mpi-hd.mpg.de/astrophysik/HEA/internal/Numerical\_Recipes/f2-3.pdf
! http://www.fi.muni.cz/~xsvobod2/misc/lapack/

!*****
! Band storage      from f07bdf.pdf
!line 1
! need not be set

!! line 2
!abd(11)=-3.66
!abd(16)=-2.13
!! line 3
!abd(7)=2.54
!abd(12)=-2.73
!abd(17)=4.07
!! line 4
!abd(3)=-0.23
!abd(8)=2.46
!abd(13)=2.46
!abd(18)=-3.82
!! line 5
!abd(4)=-6.98
!abd(9)=2.56
!abd(14)=-4.78
!
! CALL SGBTRF( 4, 4, 1, 2, abd, 5,      piv, info) ! SUCCESSFULL
!!      SGBTRF( M, N, KL KU, AB, LDAB, IPIV, INFO )
!*****

```

```

CALL SGBTRF( m1, n1, lower_width, upper_width, abd, ldab, piv, info)
!   SGBTRF( M,  N,          KL,          KU,          AB, LDAB, IPIV, INFO
)
!   M      (input) INTEGER
!           The number of rows of the matrix A.  M >= 0.
!
!   N      (input) INTEGER
!           The number of columns of the matrix A.  N >= 0.
!
!   KL     (input) INTEGER
!           The number of subdiagonals within the band of A.  KL >= 0.
!
!   KU     (input) INTEGER
!           The number of superdiagonals within the band of A.  KU >= 0.
!
!   AB     (input/output) REAL array, dimension (LDAB,N)
!           On entry, the matrix A in band storage, in rows KL+1 to
!           2*KL+KU+1; rows 1 to KL of the array need not be set.
!           The j-th column of A is stored in the j-th column of the
!           array AB as follows:
!           AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)
!
!           On exit, details of the factorization: U is stored as an
!           upper triangular band matrix with KL+KU superdiagonals in
!           rows 1 to KL+KU+1, and the multipliers used during the
!           factorization are stored in rows KL+KU+2 to 2*KL+KU+1.
!           See below for further details.
!
!   LDAB   (input) INTEGER
!           The leading dimension of the array AB.  LDAB >= 2*KL+KU+1.
!
!   IPIV   (output) INTEGER array, dimension (min(M,N))
!           The pivot indices; for 1 <= i <= min(M,N), row i of the
!           matrix was interchanged with row IPIV(i).
!
!   INFO   (output) INTEGER
!           = 0: successful exit
!           < 0: if INFO = -i, the i-th argument had an illegal value
!           > 0: if INFO = +i, U(i,i) is exactly zero. The factorization
!               has been completed, but the factor U is exactly
!               singular, and division by zero will occur if it is used
!               to solve a system of equations.
!
IF (info == 0) THEN
    WRITE (*,901) " LU decomposition successful..."
ELSE
    WRITE (*,901) " LU decomposition failed..."
    STOP '(1)'
END IF

! http://cplusplus.com/reference/clibrary/cstdio/fwrite/
!   pFile = fopen ( "myfile.bin" , "wb" );
!   fwrite (buffer , 1 , sizeof(buffer) , pFile );

```

```

! buffer
!   Pointer to the array of elements to be written.
! size=1
!   Size in bytes of each element to be written.
! count
!   Number of elements, each one with a size of size bytes.

      OPEN (abddata, FILE='abddata.dat', FORM='BINARY') ! binary
sequential file
      DO i=0,count
        WRITE(abddata) abd(i)
      END DO
      CLOSE (abddata)

      OPEN (abddata2, FILE='abddata2.dat',FORM='BINARY') ! binary
sequential file
      DO i=0,ngp-1
        WRITE(abddata2) piv(i)
! CALL fwrite(piv, SIZEOF_INT,ngp,abddata2)
      END DO
      CLOSE (abddata2)

901 FORMAT (99A)
END SUBROUTINE make_LU

!INCLUDE 'sgbtrf.f.'

END MODULE MAKE_LU_1

```

MAKE_VELOCITY_TABLE_v2.F90.

```

! -----
MODULE MAKE_VELOCITY_TABLE_v2
! -----
USE LPROBE60_1
USE ERF_2

CONTAINS

! -----
SUBROUTINE make_velocity_table()
! -----
USE MSFLIB ! needed for SYSTEMQQ
! - - - local declarations - - -
  INTEGER :: i,num_div,temp,converge
  REAL accuracy,prob_delta,delta_v,v,p_calc,p_exact,error

  INTEGER numIter, maxIter,maxNumIter
  maxIter = 30000 ! do not allow perform more than 'maxIter' iterations
to reach convergence

```

```

! - - - begin - - -
  num_div = V_TABLE_RES
  prob_delta = 1.0/num_div
  accuracy = 0.1*prob_delta          !10 percent of spacing */
  p_exact = 0.0
  v = 0.0
maxNumIter = 0

!      Electrons */
WRITE(*, '(" Electrons 1...", I6, $)') num_div

  DO i = 1, num_div-1

    temp = v ! The next two lines set the numbers past the decimal
    point to zero.
    v = temp ! There is probably a more compact way to do this.
    p_exact = p_exact + prob_delta
    if (p_exact > 1.0) p_exact = 1.0 ! due to arithmetic round up,
    p_exact may exceed 1.0
    delta_v = 1.0
    converge = 0
    numIter = 0

    DO WHILE ((converge == 0) .AND. (numIter < maxIter))      ! i.e.
    converge == 0, i.e. fabs(error) >= accuracy.
      numIter = numIter+1

      p_calc = SQRT(2.0 / pi) * (m_elec / (boltz * T * 11588.7))**1.5
&
      * (SQRT(pi / 2.0) * (boltz * T * 11588.7 / m_elec)**1.5 &
      *erff(SQRT(m_elec / (2.0 * boltz * T * 11588.7)) * v) &
      - boltz * T * 11588.7 * v / m_elec * exp(-m_elec * v * v &
      / (2.0 * boltz * T * 11588.7)))

      error = p_exact - p_calc
      if (ABS(error) < accuracy) THEN
        converge = 1
      else if (error > 0) THEN
!          delta_v = delta_v*2.0      ! try to increase the step
size to speed-up computation
        v = v + delta_v
      else ! try again with a smaller step
        v = v - delta_v
        delta_v = delta_v/10.0      ! decrease the step size
      end if
!      end if

    END DO !while

    if (numIter >= maxIter) THEN
      WRITE(*, *) " **** Routine 'make_velocity_table(void)'
*****"

```

```

        WRITE(*,*) " Convergence at electron #",i," could not be
reached in ",maxIter," iterations"
        WRITE(*,*) " Try to increase ''maxIter'' or decrease
V_TABLE_RES(",V_TABLE_RES," in ''lprobe60.f90''")
        WRITE(*,*) " The program will stop"
        result = SYSTEMQQ('PAUSE') ! MSFLIB
        STOP
    ENDIF

    v_table_e(i-1) = v

    if (v > maxVe) maxVe = v
    if (v < minVe) minVe = v

    if (numIter >maxNumIter) maxNumIter = numIter

END DO      ! i

! Avoid singularity v = infinity when p_exact = 1
v_table_e(num_div-1)= v_table_e(num_div-2) + v_table_e(num_div-2) -
v_table_e(num_div-3)

WRITE(*,*)
WRITE(*, '(" Max. number of iterations performed to achieve convergence:
",I5)') maxNumIter
    WRITE (*,903) " Done creating electron velocity table..."

! Reset parameters */
p_exact = 0.0
v = 0.0
maxNumIter = 0

!      IONS */
WRITE(*, '(" Ions 1...",I6,$)') num_div

DO i = 1,num_div-1

    temp = v ! The next two lines set the numbers past the decimal
point to zero.
    v = temp ! There is probably a more compact way to do this.
    p_exact = p_exact + prob_delta
    if (p_exact > 1.0) p_exact = 1.0 ! due to arithmetic round up,
p_exact may exceed 1.0
    delta_v = 1.0
    converge = 0
    numIter = 0;

    ! DO WHILE (converge == 0)
    DO WHILE ((converge == 0) .AND. (numIter<maxIter))      ! i.e.
converge == 0, i.e. fabs(error) >= accuracy.
        numIter = numIter +1

```

```

        p_calc = SQRT(2.0 / pi) * (m_ion / (boltz * T * 11588.7))**1.5
&
        * (SQRT(pi / 2.0) * (boltz * T * 11588.7 / m_ion)**1.5 &
        *erff(SQRT(m_ion / (2.0 * boltz * T * 11588.7)) * v) &
        - boltz * T * 11588.7 * v / m_ion * exp(-m_ion * v * v &
        / (2.0 * boltz * T * 11588.7)))
        error = p_exact - p_calc
        if (ABS(error) < accuracy) THEN
            converge = 1
        else if (error > 0) THEN
            v = v + delta_v
        else
            v = v - delta_v
            delta_v = delta_v / 10.0
        end if
!      end if

      END DO !while

      if (numIter >= maxIter) THEN
        WRITE(*,*) " **** Routine 'make_velocity_table(void)'"
*****
        WRITE(*,*) " Convergence at ion #",i," could not be reached
in ",maxIter," iterations"
        WRITE(*,*) " Try to increase 'maxIter' or decrease
V_TABLE_RES(",V_TABLE_RES," in 'lprobe60.f90'"
        WRITE(*,*) " The program will stop"
        result = SYSTEMQQ('PAUSE') ! MSFLIB
        STOP
      ENDIF

      v_table_i(i-1) = v

      if (v > maxVi) maxVi = v
      if (v < minVi) minVi = v

      if (numIter >maxNumIter) maxNumIter = numIter

    END DO

! Avoid singularity v = infinity when p_exact = 1
v_table_i(num_div-1)= v_table_i(num_div-2) + v_table_i(num_div-2) -
v_table_i(num_div-3)

WRITE(*,*)
WRITE(*,')(" Max. number of iterations performed to achieve convergence:
",I5') maxNumIter
!WRITE(*,*) "Max. number of iterations performed to achieve
convergence: ",maxNumIter
      WRITE (*,903) " Done creating ion velocity table..."

901 FORMAT (F10.6,A)
902 FORMAT (I1,A,F10.6)
903 FORMAT (99A)

```

```

END SUBROUTINE make_velocity_table
END MODULE MAKE_VELOCITY_TABLE_v2

```

MOVER.F90.

```

! -----
MODULE MOVER_1
! -----
USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE mover(particle,num_part,timestep)
! -----
! - - - arg types - - -
!TYPE (PARTICLEDEF ) particle(0:numpart-1) ! is it different from
num_part ?
TYPE (PARTICLEDEF ) particle(0:) ! 27Mar2010
INTEGER num_part
REAL timestep
! - - - local declarations - - -
    INTEGER :: i
    REAL accelx,accelr,theta

REAL vx,vy,vz

! - - - begin - - -
DO i = 0,num_part-1
    accelx = particle(i)%q * particle(i)%Ex / particle(i)%m
    accelr = particle(i)%q * particle(i)%Ey / particle(i)%m
    theta = atan2(particle(i)%y,particle(i)%z)

    !x = x0 + V0*dt + (a0/2)*dt*dt
    accely = accelr * sin(theta)
    accelz = accelr * cos(theta)
    vx=particle(i)%vx + accelx * timestep
    vy=particle(i)%vy + accely * timestep
    vz=particle(i)%vz + accelz * timestep

    particle(i)%x=particle(i)%x + (vx +accelx/2.0* timestep)*
timestep ! horizontal
    particle(i)%y=particle(i)%y + (vy +accely/2.0* timestep) *
timestep ! vertical
    particle(i)%z=particle(i)%z + (vz +accelz/2.0* timestep) *
timestep ! depth

    particle(i)%vx=vx
    particle(i)%vy=vy
    particle(i)%vz=vz

```

```

!particle(i)%vx = particle(i)%vx + accelx * timestep
!particle(i)%vy = particle(i)%vy + accelr * timestep * SIN(theta)
!particle(i)%vz = particle(i)%vz + accelr * timestep * COS(theta)
!particle(i)%x = particle(i)%x + particle(i)%vx * timestep
!particle(i)%y = particle(i)%y + particle(i)%vy * timestep
!particle(i)%z = particle(i)%z + particle(i)%vz * timestep
particle(i)%r = SQRT(particle(i)%y*particle(i)%y +
particle(i)%z*particle(i)%z)
END DO

END SUBROUTINE mover
END MODULE MOVER_1

```

OUTPUT_DATA.F90.

```

! -----
MODULE OUTPUT_DATA_1
! -----
USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE output_data(particle1,particle2,numpart1,numpart2,me,mi)
! -----
! - - - arg types - - -
!TYPE (PARTICLEDEF ) particle1(0:numpart-1)
!TYPE (PARTICLEDEF ) particle2(0:numpart-1)
TYPE (PARTICLEDEF ) particle1(0:)
TYPE (PARTICLEDEF ) particle2(0:)
INTEGER numpart1,numpart2
REAL me,mi
! - - - local declarations - - -
  INTEGER :: i2
  REAL nde,ndi
  INTEGER :: currdata = 1

! - - - begin - - -
! currdata = FOPEN("current_q100.dat","a")
  OPEN (currdata, FILE='current_q100.dat.csv',ACCESS='APPEND',
RECL=132) ! ASCII file
!avecurrent = current/(500.0*dti) ! what 500 is ?
!avecurre = curre/(500.0*dti)
!avecurri = curri/(500.0*dti)
avecurrent = current/(outputFrequency*dti)

!write(*,*)
!write(*,*) "current=",current
!write(*,*) "outputFrequency=",outputFrequency
!write(*,*) "dti=",dti
!write(*,*)

```



```

    avecurre = curre/(outputFrequency*dti)
    avecurri = curri/(outputFrequency*dti)

!          COMPUTE NUMBER DENSITY          */

    ndi = 0.0
    nde = 0.0
    DO i2 = 0,numpart1-1
        IF (particle1(i2)%r <= radius) THEN
            nde = nde + particle1(i2)%m / me
        END IF
    END DO

    nde = nde / (pi*radius*radius*length)

    DO i2 = 0,numpart2-1
        IF (particle2(i2)%r <= radius) THEN
            ndi = ndi + particle2(i2)%m / mi
        END IF
    END DO

    ndi = ndi / (pi*radius*radius*length)

    IF (t_count2 == 1) THEN
!        WRITE (currdata,901) "    V            t            ne            ni            nde
ndi            j            je            ji//"
!WRITE (currdata,*) "V, t[micro-sec],            ne,            ni,            nde,
ndi,            j,            je,            ji"
        WRITE (currdata,*) "Potential[V], t[micro-sec],electrons in
domain,ions in domain,Electrons number density,Ions number
density,Total current[mA],Electron current[mA],Ion current[mA] /"
        END IF

!    WRITE (currdata,902) V,"    ",t_count2,"    ",numelec,"    ",numion,"
",nde,"    ",ndi,"    ",avecurrent,"    ",avecurre,"    ",avecurri
        WRITE (currdata,902)
V,t_count2*(dti*1000000.0),numelec,numion,nde,ndi,avecurrent*1000.,avec
urre*1000.,avecurri*1000. ! /* 07Mar2010 */
        CLOSE (currdata)

!write(*,'( " current: ",F9.6, " [mA]",$)') avecurrent*1000.0
WRITE(*,'(F6.3,$)') avecurrent*1000.0 ! /* 07Mar2010 */

    t_count = 0
    curre = 0.0
    curri = 0.0
    current = 0.0

    hitElectrons = 0
    hitIons = 0

901 FORMAT (A )

```

```

!902 FORMAT
(E8.3,1X,I5,3X,I5,1X,I5,2X,E8.3,2X,E8.3,2X,E12.5,2X,E12.5,1X,E12.5)
902 FORMAT (E10.2,' ',',E8.3,' ',',I5,' ',',I5,' ',',E8.3,' ',',E8.3,' ',
',E12.5,' ',',E12.5,' ',',E12.5)
END SUBROUTINE output_data
END MODULE OUTPUT_DATA_1

```

PARAMETER.F90:

```

! -----
MODULE PARAMETER_1
! -----
! This function sets initial plasma conditions and computes plasma
! parameters to be used throughout the rest of the simulation */

USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE param()
! -----
! - - - local declarations - - -
REAL avveli,avvele,gammai,gammae
REAL numiperstep_back,numiperstep_front, &
numiperstep_side,numeperstep_back,numeperstep_front,numeperstep_side,fl
uxi_back,fluxi_front, &
fluxi_side,fluxe_back,fluxe_front,fluxe_side,afront,aback,aside
INTEGER :: paramFile = 2
REAL :: PROBE_ASPECT_RATIO = 0.0 ! 45.92; // probe aspect ratio
REAL ratio3

integer M ! max. number of row cells that can be added vertically
INTEGER i

! - - - begin - - -

OPEN (paramFile, FILE='param.txt', ACTION='READ') ! ASCII file

!READ(paramFile,'(F4.2)') percentAnimation
READ(paramFile,*) percentAnimation

r_probe=0.0 !1.25e-4-3.0*(0.00003125);
READ(paramFile,*) r_probe

READ(paramFile,*) PROBE_ASPECT_RATIO

l_probe = PROBE_ASPECT_RATIO * r_probe

READ(paramFile,*) ncpr ! // num. of cells per probe radius

```

```

!   NUMBER OF PARTICLES TO INJECT AT EACH micro-sec      */
superii_init = 40000
READ(paramFile,*) superii_init

superii_back = 2
READ(paramFile,*) superii_back

superii_front = 2
READ(paramFile,*) superii_front

superii_side = 4
READ(paramFile,*) superii_side

superei_init = 40000
READ(paramFile,*) superei_init

superei_back = 5
READ(paramFile,*) superei_back

superei_front = 5
READ(paramFile,*) superei_front

superei_side = 8
READ(paramFile,*) superei_side

!   GEOMETRY      */

! probe length [m]
l_insul=1.0*l_probe ! same/different length as probe (collecting
surface)

! computational domain limits
length=2.0*(l_probe+l_insul) ! length of the computational domain on
OX(horizontal) axis (see boundary.cpp)
! domain OX limits: [-
length,0] (see boundary.cpp)

!// a first estimation for domain radius
ratio3 = 0.00395705 / 0.00003125 !// = (domain radius)/ (probe radius)
!//radius=0.00395705; // radius of the computational domain
!//radius = 0.7*length; // * (0.0039570499/0.0057399999)
length*0.68938152768957365312846085589653
radius = ratio3 * r_probe !// = OY&OZ axis domain vertical (& 3D
depth) limit: +/- radius; used in boundary.cpp, charge_weight.cpp,

!
!! r_probe=1.25e-4;*/
! r_probe = 1.25e-4 - 3.0*(0.00003125) ! = 0.3125e-4
!! r_insul=2.0*r_rprobe;*/
!
! l_probe = 0.00287
! l_insul = 1.5*l_probe
!

```

```

! radius = 0.00395705
! length = 4.0*l_probe
! width = 2.0 * radius
! height = width
! ioniter = 70
! ioncount = 0
! ion_move_count = 100
! ecounter = 0
! view = 1

! electrical charges
T=0.0 ! [eV] temperature
READ(paramFile,*) T

n = 0.0 ! [cm-3]
READ(paramFile,*) n
n = n *1.e6 ! conversion [cm-3] --> [m-3]

m_ion = 0.0 ! 1.673e-27
READ(paramFile,*) m_ion

q_ion = 0.0 ! 1.6e-19
READ(paramFile,*) q_ion

! m_elec = 0.0 !9.11e-31
! READ(paramFile,*) m_elec

! /*****
!/*  mi/me = any_ratio entered by the user */
! /*****
m_elec = m_ion/mi_me_ratio

q_elec = 0.0 ! -1.6e-19
READ(paramFile,*) q_elec

wpdt = 0.0 ! 0.009
READ(paramFile,*) wpdt

plasma_pot = 0.0 ! [V]
READ(paramFile,*) plasma_pot

h = 6.9*SQRT((T*11588.7)/(n*1.0e-6)) * 0.01 ! not used in
calculations

dh = (0.00000588/0.00003125)*r_probe ! 0.18816*r_probe

! max. number of row cells that can be added vertically
DO i=1,15
  M=ngp2-i-2*ncpr
  dh =radius-r_probe*(2+(i+M+0.0)/ncpr)
  if (dh <= 0.0) EXIT
  dh = 2*dh/M/(M+1)
  noRows15=i

```

```

END DO
  M = ngp2-noRows15-2*ncpr
  dh =radius-r_probe*(2+(noRows15+M+0.0)/ncpr)
  dh = 2*dh/M/(M+1)
  ngp1 = ncpr+ncpr+noRows15+M

  cell_height1=r_probe ! ncpr*15 rows of height cell_height1/ncpr

  !cell_height2 = 0.00003713
  !cell_height3 = 0.00004301
  !cell_height4 = 0.00004889
  !cell_height5 = 0.00005477
  cell_height6 = 0.00006065

  ! the increase rate of rest, up to ncpr*73 */
  cell_height6=dh
  !* !!!!!!!!!!!!!!!!!!!!! routine 'e_field' uses 'cell_height6'
  !!!!!!!!!!!!!!!!!!!!! */

  radius = r_probe+ & !// probe
           r_probe+ & !// insulator
           noRows15*(r_probe/ncpr)+ & !// cell height as for
probe & insulator
           M*r_probe/ncpr + M*(M+1)/2*dh

  height=2.0 * radius ! 'height' & 'width' variables not used but
below, to calculate VOLUME
  width = height ! 'height' & 'width' variables not used but below,
to calculate VOLUME

  ! ngp2=4*N+1 form is required to divide both the probe length &
domain length in an integer num. of cells
  cell_width = length/(ngp2-1) ! ngp2=ngp_horiz

  ! insulator radius [m]
  r_insul = r_probe + cell_height1

  ! /* TIME STEPS */
  wp = 2.0*pi*9000.0*SQR(n*1.0e-6) ! // (omega)plasma frequency:
1.7882259e+9
  elec_iter = 15
  dt = wpdt/wp
  dte = dt ! dte = 5e-12[sec] timestep electrons
  dti = 15.0*dte ! dti = 7.5e-11[sec] timestep ions; used to
calculate the total current (see boundary.f90)

  t_count = 0

  sigmae = SQR((boltz*T*11588.7)/m_elec)
  sigmai = SQR((boltz*T*11588.7)/m_ion)

  ! Boundary Fluxing Setup */

```

```

afront = 4.0* radius * radius
aside = 2.0 * radius * length
aback = 4.0 * radius * radius - pi*r_insul*r_insul
avveli = SQRT((8.0*boltz*T*11589.7)/(pi*m_ion))
avvele = SQRT((8.0*boltz*T*11589.7)/(pi*m_elec))
gammai = n*avveli/4.0
gammae = n*avvele/4.0

fluxi_back = gammai*aback
fluxe_back = gammae*aback
fluxi_front = gammai*afront
fluxe_front = gammae*afront
fluxi_side = gammai*aside
fluxe_side = gammae*aside

numiperstep_back = fluxi_back*dti
numeperstep_back = fluxe_back*dte
numiperstep_front = fluxi_front*dti
numeperstep_front = fluxe_front*dte
numiperstep_side = fluxi_side*dti
numeperstep_side = fluxe_side*dte

! superi_back = numiperstep_back/superii_back
! supere_back = numeperstep_back/superei_back
! superi_front = numiperstep_front/superii_front
! supere_front = numeperstep_front/superei_front
! superi_side = numiperstep_side/superii_side
! supere_side = numeperstep_side/superei_side

!/* 25 Mar 2010 */
superi_back=numiperstep_back/((dte*superii_back)/1.0e-6) ! num. inj.
part. at each ION time step: (dte*superii_back)/1.0e-6
supere_back=numeperstep_back/((dte*superei_back)/1.0e-6) ! num. inj.
part. at each ELECTRON time step: (dte*superei_back)/1.0e-6
superi_front=numiperstep_front/((dte*superii_front)/1.0e-6) ! idem
supere_front=numeperstep_front/((dte*superei_front)/1.0e-6) ! idem
superi_side=numiperstep_side/((dte*superii_side)/1.0e-6) ! idem
supere_side=numeperstep_side/((dte*superei_side)/1.0e-6) ! idem

!*****      NUMBER OF PARTICLES TO INJECT AT EACH electron/ion TIME
STEP, respectively *****/
! (not used anymore as integer variables but as FP variables as input
parameters in 'inject_part')
superii_back_FLOAT = (dte*superii_back)/1.0e-6
superii_back = superii_back_FLOAT
superei_back_FLOAT = (dte*superei_back)/1.0e-6
superei_back = superei_back_FLOAT
superii_front_FLOAT = (dte*superii_front)/1.0e-6
superii_front = superii_front_FLOAT
superei_front_FLOAT = (dte*superei_front)/1.0e-6
superei_front = superei_front_FLOAT
superii_side_FLOAT = (dte*superii_side)/1.0e-6

```

```

superii_side= superii_side_FLOAT
superei_side_FLOAT= (dte*superei_side)/1.0e-6
superei_side= superei_side_FLOAT

! Initial Loading Injection Setup */

    VOLUME = length*height*width - (pi* r_probe * r_probe * l_probe) -
    (pi* r_insul * r_insul * l_insul)
    superi_init = n * ((pi* radius * radius * length)- &
    (pi* r_probe * r_probe * l_probe) - (pi* r_insul * r_insul *
    l_insul)) &
    / superii_init
    superei_init = n * ((pi* radius * radius * length)- &
    (pi* r_probe * r_probe * l_probe) - (pi* r_insul * r_insul *
    l_insul)) &
    / superei_init
    injsuperi = superii_init
    injsupere = superei_init

! Parameteters needed for field solver */
jprobe=((r_probe/(cell_height1/ncpr))) ! no. of cells covering half
of probe width(i.e. above 0X=0) : 1

    ! grid point vertical limit for probe
kprobe=((length-l_probe-l_insul)/cell_width)) ! no. of cells in front
of the probe : 45

    ! grid point horizontal limit for probe tip

! <----- length ----->
! -----=====HHHHHHHHHHHHHHHHHH
! <-- l_probe --><-- l_insul ->

CLOSE(paramFile)

END SUBROUTINE param
END MODULE PARAMETER_1

```

PLOT_1.F90.

```

MODULE PLOT_1 ! PGPLOT routines

USE DISTRIB_1 ! needed for type GRIDDEF_T

CONTAINS
!-----
-
! This subroutine tests PGQINF and displays the information returned on

```

```

! the standard output.
!-----
-
      SUBROUTINE PGEX0
      CHARACTER*64 VALUE
      INTEGER LENGTH
      REAL X, Y, X1, X2, Y1, Y2
!
! Information available from PGQINF:
!
      CALL PGQINF('version', VALUE, LENGTH)
      WRITE (*,*) 'version=', VALUE(:LENGTH)
      CALL PGQINF('state', VALUE, LENGTH)
      WRITE (*,*) 'state=', VALUE(:LENGTH)
      CALL PGQINF('user', VALUE, LENGTH)
      WRITE (*,*) 'user=', VALUE(:LENGTH)
      CALL PGQINF('now', VALUE, LENGTH)
      WRITE (*,*) 'now=', VALUE(:LENGTH)
      CALL PGQINF('device', VALUE, LENGTH)
      WRITE (*,*) 'device=', VALUE(:LENGTH)
      CALL PGQINF('file', VALUE, LENGTH)
      WRITE (*,*) 'file=', VALUE(:LENGTH)
      CALL PGQINF('type', VALUE, LENGTH)
      WRITE (*,*) 'type=', VALUE(:LENGTH)
      CALL PGQINF('dev/type', VALUE, LENGTH)
      WRITE (*,*) 'dev/type=', VALUE(:LENGTH)
      CALL PGQINF('hardcopy', VALUE, LENGTH)
      WRITE (*,*) 'hardcopy=', VALUE(:LENGTH)
      CALL PGQINF('terminal', VALUE, LENGTH)
      WRITE (*,*) 'terminal=', VALUE(:LENGTH)
      CALL PGQINF('cursor', VALUE, LENGTH)
      WRITE (*,*) 'cursor=', VALUE(:LENGTH)
!
! Get view surface dimensions:
!
      CALL PGQVSZ(1, X1, X2, Y1, Y2)
      X = X2-X1
      Y = Y2-Y1
      WRITE (*,100) X, Y, X*25.4, Y*25.4
100 FORMAT (' Plot dimensions (x,y; inches): ',F9.2,', ',F9.2, '
(mm): ',F9.2,', ',F9.2)
!      1 (mm): ',F9.2,', ',F9.2)
      END SUBROUTINE ! PGEX0

!-----
-
! This example illustrates the use of PGENV, PGLAB, PGPT, PGLINE.
!-----
-
      SUBROUTINE PGEX1
      INTEGER I
      REAL XS(5),YS(5), XR(100), YR(100)
      DATA XS/1.,2.,3.,4.,5./

```



```

DATA YS/1.,4.,9.,16.,25./
!
! Call PGENV to specify the range of the axes and to draw a box, and
! PGLAB to label it. The x-axis runs from 0 to 10, and y from 0 to 20.
!
CALL PGENV(0.,10.,0.,20.,0,1)
CALL PGLAB('(x)', '(y)', 'PGPLOT Example 1:  $y = x^2$ ')
!
! Mark five points (coordinates in arrays XS and YS), using symbol
! number 9.
!
CALL PGPT(5,XS,YS,9)
!
! Compute the function at 60 points, and use PGLINE to draw it.
!
DO 10 I=1,60
    XR(I) = 0.1*I
    YR(I) = XR(I)**2
10 CONTINUE
CALL PGLINE(60,XR,YR)
END SUBROUTINE ! PGEX1
!-----

!-----
! This example illustrates the use of PGBOX and attribute routines to
! mix colors and line-styles.
!-----
! Default Color Representation:
www.cse.iitb.ac.in/~cs101/Year2003/resources/notesPgplot/chapter5.html

SUBROUTINE PGEX30(xMax) ! bins
!
    USE DISTRIB_1
    !TYPE (GRIDDEF_T) :: xx(0:99)
    INTEGER max_bin

!
    REAL PI
!
    PARAMETER (PI=3.14159265359)
    INTEGER I
    REAL XR(360), YR(360)
    REAL ARG

!
! Call PGENV to initialize the viewport and window; the
! AXIS argument is -2, so no frame or labels will be drawn.
!
CALL PGENV(0.,720.,-2.0,2.0,0,-2)

!
CALL PGENV(1.0,n-1.0,yMin,yMax,0,-2)
CALL PGENV(0.0,xMax,0.0,1.0,0,-2)
!CALL PGSAVE
!

! Draw first a grid at low brightness(14=dark gray)

CALL PGSCI(14) ! Dark Gray

```

```

        !CALL PGBOX('G',30.0,0,'G',0.2,0)
        CALL PGBOX('G',0.0,0,'G',0.0,0) ! automatic

! and then a frame and axes at full brightness.
    CALL PGSCI(5)      ! cyan
    !CALL PGBOX('ABCTSN',90.0,3,'ABCTSNV',0.0,0)
    CALL PGBOX('ABCTSN',0.0,0,'ABCTSNV',0.0,0)      ! automatic
!
! Call PGLAB to label the graph in a different color (3=green).
    CALL PGSCI(3)      ! green
    CALL PGSCI(1)      ! white
    CALL PGLAB('velocity(W)','f(W) ','Velocity distribution during
injection')
!
! Compute the function to be plotted: a trig function of an
! angle in degrees, computed every 2 degrees.
    !DO 20 I=1,360
    !    XR(I) = 2.0*I
    !    ARG = XR(I)/180.0*PI
    !    YR(I) = SIN(ARG) + 0.5*COS(2.0*ARG) +
0.5*SIN(1.5*ARG+PI/3.0)
!    1          0.5*SIN(1.5*ARG+PI/3.0)
    !20 CONTINUE
!
! Change the color (6=magenta), line-style (2=dashed), and line
! width and draw the function.
    !CALL PGSCI(6)      ! magenta
    !CALL PGSLS(2)      ! continuous line
    !CALL PGSBW(3)      ! line width
!    CALL PGLINE(360,XR,YR)

    RETURN

!
! Restore attributes to defaults.
    !CALL PGUNSA

    END  SUBROUTINE ! PGEX30
!-----

! Default Color Representation:
www.cse.iitb.ac.in/~cs101/Year2003/resources/notesPgplot/chapter5.html
    SUBROUTINE PGEX31(tMax)
!        USE DISTRIB_1
        TYPE (GRIDDEF_T) :: xx(0:99)
        INTEGER max_bin

!        REAL PI
!        PARAMETER (PI=3.14159265359)
        INTEGER I
        REAL XR(360), YR(360)
        REAL ARG
!

```

```

! Call PGENV to initialize the viewport and window; the
! AXIS argument is -2, so no frame or labels will be drawn.
!   CALL PGENV(0.,720.,-2.0,2.0,0,-2)

!   CALL PGENV(1.0,n-1.0,yMin,yMax,0,-2)
CALL PGENV(0.0,tMax,10000.0,80000.0,0,-2) ! Lab-report.pdf (page
12)
!CALL PGSAVE
!
! Draw first a grid at low brightness(14=dark gray)

CALL PGSCI(14) ! Dark Gray
!CALL PGBOX('G',30.0,0,'G',0.2,0)
CALL PGBOX('G',0.0,0,'G',0.0,0) ! automatic

! and then a frame and axes at full brightness.
CALL PGSCI(5) ! cyan
!CALL PGBOX('ABCTSN',90.0,3,'ABCTSNV',0.0,0)
CALL PGBOX('ABCTSN',0.0,0,'ABCTSNV',0.0,0) ! automatic
!
! Call PGLAB to label the graph in a different color (3=green).
CALL PGSCI(3) ! green
CALL PGSCI(1) ! white
CALL PGLAB('time[micro-sec]','no. of particles ','Particle no.
during injection')
!
! Compute the function to be plotted: a trig function of an
! angle in degrees, computed every 2 degrees.
!DO 20 I=1,360
!   XR(I) = 2.0*I
!   ARG = XR(I)/180.0*PI
!   YR(I) = SIN(ARG) + 0.5*COS(2.0*ARG) +
0.5*SIN(1.5*ARG+PI/3.0)
!   1 0.5*SIN(1.5*ARG+PI/3.0)
!20 CONTINUE
!
! Change the color (6=magenta), line-style (2=dashed), and line
! width and draw the function.
!CALL PGSCI(6) ! magenta
!CALL PGSLS(2) ! continuous line
!CALL PGSLW(3) ! line width
! CALL PGLINE(360,XR,YR)

RETURN

! Restore attributes to defaults.
!CALL PGUNSA

END SUBROUTINE ! PGEX31
!-----

SUBROUTINE PGEX32 !(grid)

```

```
USE LPROBE60_1
```

```

! draw the grid
! (grid.x,grid.y) is the left-down corner of the cell
! grid[][0],grid[][1],...,grid[][ngp2-1] advances on OX
! grid[0][],grid[1][],...,grid[ngp1-1][] advances on OY

!TYPE (GRIDDEF ) grid(0:ngp1-1,0:ngp2-1)

REAL xPlot(0:4),yPlot(0:4)
REAL minX,maxX,minY,maxY
INTEGER nx,ny
REAL d
REAL index1,index2
INTEGER count1,count2

! determine the (x,y) limits
maxX=grid(0,0)%x
minX=maxX
maxY=grid(0,0)%y
minY=maxY

DO nx =0,ngp2-1
  DO ny=0,ngp1-1
    if (grid(ny,nx)%x > maxX) maxX = grid(ny,nx)%x
    if (grid(ny,nx)%x < minX) minX = grid(ny,nx)%x
    if (grid(ny,nx)%y > maxY) maxY = grid(ny,nx)%y
    if (grid(ny,nx)%y < minY) minY = grid(ny,nx)%y
  END DO
END DO

maxX =0.0
maxY =radius
d = maxY-minY;
if ((maxX-minX) > d) d = maxX-minX

CALL pgsci(14) ! Dark Gray
CALL pgenv(minX,minX+d,minY,minY+d, &
  1, & ! the scales on OX and OY will be equal
  ! 0, // the scales on OX and OY will not be equal
  0) ! draw box and label it with coordinates
CALL pgbox("AP",0.0,0,"AP",0.0,0) ! automatic

CALL pgsci(5) ! cyan
CALL pgbox("G",0.0,0,"G",0.0,0) ! automatic

CALL pgslw(1) ! linewidth; the thinnest
! draw probe
CALL pgsci(2) ! Red

!xPlot[2]= grid[0][ngp2-1].x;
xPlot(2)= -l_insul
yPlot(2) = r_probe
!
/

```

```

! grid[0][0]          +--/-----+ grid[0][ngp2-1]
! |          +-----+. insul. | /
! +-----+-- probe --+-----+ (0,0)
!          \.
!          xPlot(0)= -(l_probe+l_insul)
!          yPlot(0) = 0.0 != grid[0][0].y = grid[0][ngp2-1].y ;

CALL pgrect (xPlot(0), xPlot(2), yPlot(0), yPlot(2))

! draw insulator
CALL pgsci(4) ! Blue
!          xPlot(2)= 0.0 != grid[0][ngp2-1].x;
!          yPlot(2) = r_insul
!          \
! grid[0][0]          +-----+ grid[0][ngp2-1]
! |          +-----+. insul. | /
! +-----+-- probe --+-----+ (0,0)
!          /
!          xPlot(0)= -l_insul
!          yPlot(0) = 0.0 != grid[0][0].y = grid[0][ngp2-
1].y;
CALL pgrect (xPlot(0), xPlot(2), yPlot(0), yPlot(2))

CALL pgupdt() ! output the graph

CALL pgsci(1) ! White
index1=-1.0
DO count1=0,ngp1-2 ! rows
  index1=index1+1.0
  index2=-1.0
  DO count2=0, ngp2-2 ! columns
    index2=index2+1.0

    xPlot(0)=grid(count1,count2)%x
    yPlot(0)=grid(count1,count2).y
    xPlot(1)=grid(count1,count2+1)%x
    yPlot(1)=grid(count1,count2+1)%y;
    xPlot(2)=grid(count1+1,count2+1)%x
    yPlot(2)=grid(count1+1,count2+1)%y
    xPlot(3)=grid(count1+1,count2)%x
    yPlot(3)=grid(count1+1,count2)%y
    CALL pgline(4,xPlot,yPlot)

  END DO ! count2
END DO ! count1
CALL pgupdt() ! output the graph

END SUBROUTINE PGEX32

END MODULE PLOT_1

```

PROBEPIC.F90:

```

! To use PGPLOT.LIB in a project:
! Insert .. -> Files into project... -> File type: .lib ->
C:\MSDEV\LIB\PGPLOT.LIB
! -----
PROGRAM PROBEPIC
! -----
!
! This is the main program for a two dimensional PIC simulation of a
Langmuir probe.  */

USE LPROBE60_1

USE RANDOM_NUMBER_V1_1
!USE BOUNDARY_1
USE BOUNDARY_v2          ! insulator removed

!USE BOUNDARY_BEAM_1
USE CHARGE_WEIGHT_1
USE FORCE_WEIGHT_1
USE GRID_1
USE INITIALIZE_1
!USE INITIALIZE_BEAM_1
!USE INJECT_BEAM_1
USE LOCATE_1
USE INJECT_PART_1
USE MAKE_LU_1

!USE MAKE_VELOCITY_TABLE_1
USE MAKE_VELOCITY_TABLE_v2    ! does not generate zero-speed particles

USE MOVER_1
USE OUTPUT_DATA_1
USE PARAMETER_1
USE RESET_GRID_1
USE FIELD_SOLVER_1

USE DISTRIB_1                ! put particles in bins
USE PLOT_1                   ! PGPLOT routines

USE Console_API              ! console & keyboard management
routines
!USE MSFLIB                  ! provides MSFLIB
use msfwin ! provides Fortran interfaces to Win32 routines

!USE DFWIN ! provides Fortran interfaces to Win32 routines

!INCLUDE 'flib.fi'
!PROGRAM Windows_Handle_Sample
!INCLUDE 'flib.fd'

```

```

! - - - local declarations - - -
  INTEGER PGOPEN ! need for graphics

  !INTEGER :: seed ! already defined in the system module MSFLIB, as
SUBROUTINE SEED(ARG)
  INTEGER :: seed0
  INTEGER :: i,iter,i6,max_bin
  INTEGER :: duration
  !BYTE :: CR = 13
  TYPE (GRIDDEF_T) :: bins(0:noBins-1)
  REAL tMax,min_val,max_val,delta,velocity
  CHARACTER(8) string,string2,string3

!REAL xPlot,yPlot
REAL dx,dy
!CHARACTER(10) string1
!CHARACTER(30) string_t,string_pot,string_elec
INTEGER iPG

! handling keypress (for animation ON/OFF)
LOGICAL :: animationON = .TRUE.
CHARACTER key1
!LOGICAL(4) :: result ! used by SYSTEMQQ
INTEGER kkk,count1;

! API related
character*100 lpszFileName
integer hConsole
!INTEGER(2) top, left, bottom, right
INTEGER(2) txtColor
  TYPE (rccoord) crsPos
  INTEGER(4) textWindow, graphWindow
  CHARACTER ASCII_code

! Console data declarations
  integer con_lines,con_length
  logical con_status, lstat
  integer con_handle, std_handle
  Type(T_COORD) con_wpos
  character*100 str
  integer num_w_char
  integer lpcRead, lpvReserved
  Type (T_CONSOLE_SCREEN_BUFFER_INFO) con_info
  type(T_INPUT_RECORD) inRec(0:127)
  !type(T_KEY_EVENT_RECORD) keRec ! uCHAR%CHAR is the key
user pressed
  LOGICAL resultOK
  integer i00
  INTEGER fdwMode

```

```

! - - - begin - - -

! create a Windows console to be able to turn animation ON/OFF
CALL CREATE_TEXT_CONSOLE

! /* Enable the window and mouse input events. */

!     fdwMode = 0
!     resultOK = SetConsoleMode(GetStdHandle(STD_INPUT_HANDLE),
fdwMode)
!WRITE(*,*) "Disable mouse & window ",resultOK

! "CreateFile(...OPEN_EXISTING/OPEN_NEW...)" does not work unless
CREATE_TEXT_CONSOLE is previously called
!     lpszFileName = "CONOUT$C"
!     hConsole = CreateFile(lpszFileName, IOR(GENERIC_WRITE ,
GENERIC_READ),
&
IOR(FILE_SHARE_READ , FILE_SHARE_WRITE),
&
! !             NULL_SECURITY_ATTRIBUTES, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0)
!             NULL_SECURITY_ATTRIBUTES, OPEN_NEW,
FILE_ATTRIBUTE_NORMAL, 0)

CALL WRITE_TO_CONSOLE('') ! <=> new line
CALL WRITE_$TO_CONSOLE('Press any key ON THIS WINDOW to switch
animation ON/OFF')
CALL WRITE_TO_CONSOLE('') ! <=> new line
CALL WRITE_TO_CONSOLE('') ! <=> new line

!
!CALL WRITE_$TO_CONSOLE('|Al doileas sir    |')
!CALL WRITE_$TO_CONSOLE('|cu 4 spatii    |')
!
!!CALL WRITE_$TO_CONSOLE("\nC)      ! works too
!CALL WRITE_TO_CONSOLE('') ! <=> new line
!
!CALL WRITE_$TO_CONSOLE('|Al treilea sir|')
!CALL WRITE_TO_CONSOLE('|Al 4-lea sir|')

!     CALL OUTTEXT ('Hello, everyone')
!
!     write (7,*) " "
!     write (7,*) "This should be in CONSOLE "
!     write (*,*) "Press any key when done viewing "
!     read (*,*)
!
!! Deallocate the console to free its resources.
!

```



```

!      con_status = FreeConsole()

! top-left text postion in Quickwin window
crsPos%row=1
crsPos%col=1

! Load simulation parameter initialization values */
WRITE (*, '(" Enter mi/me ratio [Hydrogen=1836]: ",$)')
READ(*,*) mi_me_ratio

! unit numebr of the Quickwin text window
textWindow = GETACTIVEQQ ( )

CALL param() ! see 'parameter.f90'

WRITE (*,*)
WRITE (*, '(" Number density,  $n[m^{(-3)}]$ : ",E7.2)') n
WRITE (*, '(" Plasma potential [V]: ",F5.2)') plasma_pot
WRITE (*, '(" Plasma temperature, T[eV]: ",F5.2)') T
WRITE (*, '(" Probe Aspect Ratio: ",F5.2)') l_probe/r_probe
WRITE (*, '(" Electron time step [micro-sec]: ", G8.3)') dte*1.0e6
WRITE (*, '(" Average Current measuring interval : ",I3," x (ion time
steps) ")') outputFrequency
WRITE (*,*)
cursorY = 3 ! manages the info text on the screen (vertically)

      ngp = ngp1*ngp2 ! = ngp1 x ngp2 = 74 x 74

      ALLOCATE(grid(0:ngp1-1,0:ngp2-1))
      ALLOCATE(band1(0:ngp-1))
      ALLOCATE(band2(0:ngp-1))
      ALLOCATE(band3(0:ngp-1))
      ALLOCATE(band4(0:ngp-1))
      ALLOCATE(band5(0:ngp-1))

!      !IF (PGOPEN('?') .LE. 0) STOP
!      IF (PGOPEN('/WV') .LE. 0) STOP
!
!      !if(PGBEG(0, "/WV", 2, 1) /= 1) STOP !UNIT, FILE, NXSUB, NYSUB
!      !CALL PGASK(1) ! FLAG=1(TRUE) : 'prompt state' of PGPLOT is set to
ON
!      !CALL PGBBUF
!      !CALL PGSUBP(2, 1)

! FIELD SOLVER GLOBAL VARIABLES */
test1 = 0
test2 = 0
test3 = 0

ldab = 3*ngp2+1
n1 = ngp
m1 = ngp
ldb = ngp

```

```

upper_width = ngp2
lower_width = ngp2

x = 521288629
y = 362436069
z = 16163801
c = 1
n2 = 1131199299

!/* Large array initialization */
ALLOCATE( electron(0:numpart-1)) ! type of PARTICLEDEF
ALLOCATE(      ion(0:numpart-1)) ! type of PARTICLEDEF
ALLOCATE( abd(0:(2*lower_width+upper_width+2)*ngp) ) ! type of REAL
ALLOCATE(    b(0:ngp-1))      ! type of REAL
ALLOCATE(  phi(0:ngp-1))      ! type of REAL

ALLOCATE( v_table_e(0:V_TABLE_RES))      ! type of REAL
ALLOCATE( v_table_i(0:V_TABLE_RES))      ! type of REAL
ALLOCATE( random_num(0:NPTS)) ! type of REAL    0...1,000,005
ALLOCATE( piv(0:ngp-1))      ! type of INTEGER

! Variable initialization */
t_count2 = 0
t_count = 0
numion = 0
numelec = 0
current=0.0
V = 0.0 ! potential

! Initialize array of uniform random numbers */
seed0 = 100 ! Number of integers the processor uses to hold the seed
value.
CALL random_number(seed0)
!CALL random_seed(seed2)
!CALL random_number(random_num) ! stores    1,000,005 random
numbers
rancount = 0

! Generate computational cells */
WRITE (*, '(" Generate computational cells ...",$)')
CALL gridgen(grid)
WRITE (*,*) "DONE"

! Generate look-up tables for ion and electron speeds */
! WRITE (*,*) " Generate look-up tables for
",V_TABLE_RES,"ion/electron speeds..."
WRITE (*,*) "Generate look-up tables for ion/electron speeds..."
! initializes an array of one hundred thousand speeds for each
species using (4.17)
CALL make_velocity_table() ! Creates arrays of velocities with for
fluxing into the computational domain

```

```

WRITE(*, '(" Electron speed range [m/s]: ",I4," ... ",I6)')
INT(minVe),INT(maxVe)
WRITE(*, '(" Ion speed range [m/s]: ",I4," ... ",I6)')
INT(minVi),INT(maxVi)

IF (maxVe*dte > cell_width) THEN
    !CALL SetFColor(2) ! light green
    txtColor = SETTEXTCOLOR(INT2(2)) ! green
    WRITE(*, '(! ! ! ! ! Please decrease parameter ''wpdt'', in
''param.txt'', to approx.", F8.6)') wp*cell_width/maxVe
    WRITE(*,*) " The program will stop."
    result = SYSTEMQQ('PAUSE') ! MSFLIB
    STOP
ENDIF

IF (maxVi*dti > cell_width) THEN
    !CALL SetFColor(2) ! light green
    txtColor = SETTEXTCOLOR(INT2(2)) ! green
    WRITE(*, '(! ! ! ! ! Please decrease parameter ''wpdt'', in
''param.txt'', to approx.", F8.6)') wp*cell_width/maxVi/elec_iter
    WRITE(*,*) " The program will stop."
    result = SYSTEMQQ('PAUSE') ! MSFLIB
    STOP
ENDIF

! Try to decrease running time by increasing the time step
wpdtMax = wp*cell_width/maxVi/elec_iter
if (wpdtMax > wp*cell_width/maxVe ) wpdtMax = wp*cell_width/maxVe

IF (wpdt < wpdtMax) THEN
    WRITE(*,*)
    WRITE(*, '(" To speed up the simulation, parameter ''wpdt'', in
''param.txt''")')

    WRITE(*, '(" may be increased to approx.", $)')
    !CALL SetFColor(15) ! bright white
    txtColor = SETTEXTCOLOR(INT2(14)) ! 15=white, 14=yellow

    WRITE(*,*) wpdtMax
    !kkk =whereY()
    CALL GetTextPosition (crsPos)
    kkk= crsPos%row

    !CALL SetFColor(7) ! white
    txtColor = SETTEXTCOLOR(INT2(15)) ! 15=white 7=gray

    !result = SYSTEMQQ('PAUSE') ! creates a new Windows window
    WRITE(*, '(" Press any key to continue ...", $)')
    key1=GETCHARQQ( ) ! read/remove the pressed key from buffer

    !CALL gotoXY(0, kkk-0)
    crsPos%row=0
    crsPos%col=0
    CALL SETTEXTPOSITION (kkk-2, INT2(1), crsPos)

```

```

ENDIF

! Refresh array of uniform random numbers */
!seed = 101
!CALL random_number(seed)

!CALL random_seed(seed2)
!CALL random_number(random_num) ! stores      1,000,005 random
numbers
  rancount = 0
! Generate an initial uniform fill of the computational domain with
electrons */
WRITE(*, '("Generate an initial uniform fill of the computational domain
with electrons...", $)')
  CALL initialize(electron, numelec, superei_init, supere_init,
v_table_e, m_elec, q_elec)
WRITE(*, *)
WRITE(*, *) "DONE"

! Refresh array of uniform random numbers */
  seed0 = 101
  CALL random_number(seed0)
WRITE(*, '("Generate an initial uniform fill of the computational
domain with ions...", $)')
  rancount = 0
! Generate an initial uniform fill of the computational domain with
ions
  CALL initialize(ion, numion, superii_init, superi_init, v_table_i,
m_ion, q_ion)
WRITE(*, *)
WRITE(*, *) "DONE"

!-----
! SET UP BINS TO PUT electrons IN
! minimum speed
min_val = 0.0
! get the maximum speed
max_val = 0.0
DO i = 0, numelec-1
  velocity= SQRT(electron(i)%vx**2 +electron(i)%vy**2
+electron(i)%vz**2 )
  if (velocity > max_val) max_val = velocity
END DO

! set the lower limit for each bin
delta = (max_val-min_val)/noBins ! mean # of points in a bin
DO i = 0, noBins-1 ! bins[0], ..., bins[99]
  !start = min_val + i*delta
  !bins(i)%r = start ! the lower limit of the bin
  bins(i)%r = min_val + i*delta ! the lower limit of the bin
  bins(i)%v = 0.0 ! shows how many data points fall in it
END DO

! PUT v_table_e DATA IN BINS

```

```

CALL DISTRIB(bins,noBins,electron,numelec)

! get the bin with the maximum number of particles in it
max_bin = 0.0
DO i = 0,noBins-1
  if (bins(i)%v > bins(max_bin)%v) max_bin = i
END DO
!-----

WRITE(*,*) " initial number electrons :",numelec," initial number ions
: ",numion

!----- CPGPLOT sequence -----

!
! Call PGOPEN to initiate PGPLOT and open the output device; PGOPEN
! will prompt the user to supply the device name and type. Always
! check the return code from PGOPEN.
  IF (PGOPEN('/WX') .LE. 0) STOP
!
!           /WV: 640 x 480
!           /WS: 800 x 600
!           /WX: 1024 x 768
!           /WZ: 1280 x 1024

! unit numebr of the Quickwin graphics window
graphWindow = GETACTIVEQQ ( )

!if(PGBEG(0, "/WV", 1, 1) /= 1) STOP !UNIT, FILE, NXSUB, NYSUB
!!CALL PGASK(1) ! FLAG=1(TRUE) : 'prompt state' of PGPLOT is set to
ON
CALL PGBBUF ! Begin saving graphical output commands in an internal
buffer; the
! commands are held until a matching PGEBUF call
! check grid
CALL PGEX32 !(grid) ! see 'plot_1.f90'

CALL PGASK(1) ! FLAG=0(FALSE) :do not prompt when plot befor
plotting on another page
! FLAG=1(TRUE) :do prompt when plot before
plotting on another page
CALL PGSUBP(2, 1) ! two adjacent panels
!
! Print information about device.
!CALL PGEX0
!
! Call the drawing subroutines

! Set manually the color table to default values
!CALL PGSCR(0, 0.0, 0.0, 0.3) ! changes color index 0
(background) to dark blue
!CALL PGSCR(0, 0.0, 0.0, 0.0) ! changes color index 0
(background) to black

```

```

        !CALL PGSCR(5, 0.00, 1.00, 1.00) ! cyan
        !CALL PGSCR(1, 1.00, 1.00, 1.00) ! white
        !CALL PGSCR(3, 0.00, 1.00, 0.00) ! green
        !CALL PGSCR(6, 1.00, 0.00, 1.00) ! magenta
        !CALL PGSCR(14, 0.33, 0.33, 0.33) ! dark gray
!-----
        ! select Left Panel

        CALL PGPANL(2, 1)
        !CALL PGERAS ! Erase all graphics from the current page or panel.

!! the user is required to hit <ENTER> on the text window
!result = SETACTIVEQQ (textWindow)
!result = FOCUSQQ (textWindow)

        CALL PGEX30(max_val)      ! draw axes

        ! draw graph of velocities distribution
        CALL PGSCI(6)      ! magenta
        DO I=0,noBins-1
            CALL PGPT(1,bins(I)%r,bins(I)%v/bins(max_bin)%v,17)
        END DO

        ! draw a marker
        !CALL PGPT (1,2000000.0,0.5,8) ! (N, XPTS, YPTS, SYMBOL)

        ! draw text
        CALL PGSCI(1)      ! white
        !write(string,'(I6)') numelec
        !CALL PGPTXT (2000000.0,0.5,0.0, 0.0,'No. electrons: '//string)
! X, Y, ANGLE, FJUST, TEXT)

        write(string3,'(E8.3)') 0.0
        write(string,'(I6)') numelec
        CALL PGPTXT (2000000.0,0.65,0.0, 0.0,' t = '//string3//' sec') !
X, Y, ANGLE, FJUST, TEXT)
        CALL PGPTXT (2000000.0,0.50,0.0, 0.0,'No. electrons: '//string)
! X, Y, ANGLE, FJUST, TEXT)

! flush any pending commands to the output device.
        CALL PGUPDT

!-----
-----

        !WRITE (*,*) " initial number electrons ==>",numelec,"    initial
number ions ==>",numion

        CALL make_LU(grid) ! LU decomposition of the difference matrix for
the electric field solver

        result = SETACTIVEQQ (textWindow)
        result = FOCUSQQ (textWindow)
        WRITE (*,*)

```



```

CALL PGASK(1) ! FLAG=1(TRUE) :prompt when plot befor plotting on
another page
! select Right Panel (ion/electron)
CALL PGPANL(1, 1)
!CALL PGPANL(2, 1)
!CALL PGERAS ! Erase all graphics from the current page or panel.
CALL PGEX31(tMax) ! draw axes
CALL PGSCI(6) ! magenta for electrons
CALL PGPTXT (tMax/3.0,73000.0,0.0, 0.0,' electrons') ! X, Y,
ANGLE, FJUST, TEXT)
CALL PGSCI(3) ! green for ions
CALL PGPTXT (tMax/3.0,75000.0,0.0, 0.0,' ions') ! X, Y, ANGLE,
FJUST, TEXT)

!-----

result = SETACTIVEQQ (textWindow)
!result = FOCUSQQ (textWindow)

! the real-time numerical values will be displayed in yellow
txtColor = SETTEXTCOLOR(INT2(14)) ! yellow

animationON = .true.
if ((percentAnimation > 1.0e-6) .AND. (animationON)) THEN
!CALL gotoXY(10,45+cursorY)
CALL SETTEXTPOSITION (INT2(45+cursorY), INT2(11), crsPos)
WRITE(*, '(" ON ",$)')
else
!CALL gotoXY(10,45+cursorY)
CALL SETTEXTPOSITION (INT2(45+cursorY), INT2(11), crsPos)
WRITE(*, '(" OFF", $)')
ENDIF

write(*,*)

!***** PERFORM SIMULATION with 'dti' time step
*****

! average current on a dti/dte time step
avgc_dti = 0.0
avgc_dte = 0.0

DO i = 1,iter
IF (rancount > (NPTS - outputFrequency)) THEN ! is 500 for
safety (array out of range) ?
seed0 = seed0 + 100
CALL random_number(seed0)
rancount = 0
END IF

! electron iteration (page 40)
DO i6 = 0,elec_iter-1 ! 0...15-1
CALL reset_grid(grid) ! grid[][].q_dens=0.0

```



```

        CALL qweight(grid, electron, numelec) !// adjust
grid[][] .q_dens
                                                                    !//
locate electron[].cellx/celly; if outside domain, remove
        CALL qweight(grid, ion, numion) !// adjust grid[][] .q_dens
                                                                    !// locate
ion[].cellx/celly; if outside domain, remove
        CALL e_field(grid) ! Call Lapack to solve the Poisson equation
is solved yielding the electric field at the grid points
                                                                    ! // adjust
grid[][] .Ex/Ey
                                                                    ! // uses 'cell_height6'
as the value of upper row cells height
        CALL fweight(grid, electron, numelec) !// re-calculate
electron[].Ex/Ey from grid[][] .Ex/Ey

        ! Fluxes thermal particles across boundaries into the
computational domain
        ! it adds particles and locate them:
electron[].cellx/celly
        CALL inject(electron, numelec, &
                    supere_back, supere_front, supere_side, & ! the
super-particle weighting (Thesis, page 58)
                    INT(superei_back_FLOAT*i)-
INT(superei_back_FLOAT*(i-1)), & ! superei_back,
                    INT(superei_front_FLOAT*i)-
INT(superei_front_FLOAT*(i-1)), & ! superei_front,
                    INT(superei_side_FLOAT*i)-
INT(superei_side_FLOAT*(i-1)), & ! superei_side,
                    v_table_e, m_elec, q_elec)
        CALL mover(electron, numelec, dte) ! Integrate the Lorentz
force equation to determine the new position and velocity for each
particle
                                                                    !//
moves particles with acc=acc[q,Ex/Ey,m)
        CALL boundary(electron, numelec, q_elec, textWindow, graphWindow)
! // collects the electron[].q current when on probe

! // the insulator is disabled (commented code)

        ! Plot the electrons
        if ((percentAnimation > 1.0e-6) .AND.
(animationON)) THEN
                ! select Left Panel (maxwellian)
                result = SETACTIVEQQ (graphWindow)
                !result = FOCUSQQ (graphWindow)
                CALL pgpanl(1, 1)
                dx=length/10.0
                ! some space around the drawing area
                dy=(grid(ngp_vert-1,0)%y)/10.0 ! some
space around the drawing area
                ! world coordinates:
                CALL pgswin(-length-dx,0.0+dx, & ! 0x
axis

```

```

                                -radius-dy, radius+dy) ! Oy
axis
                                ! plot the electrons(magenta) then the
ions(green), 1%
                                CALL pgsci(6)      ! magenta
                                DO count1=0, numelec-1, 100/percentAnimation

                                CALL
pgpt(1,electron(count1)%x,electron(count1)%y,-1) ! graphic symbol #(-1)

                                ! http://www.astro.rug.nl/~kuijken/pgplot.html#PGPIXL
                                END DO
                                ENDIF ! animation

                                END DO ! i6, electron iteration

                                ! ***** ion iteration (page 40)

                                result = SETACTIVEQQ (graphWindow)      ! needed for 'inject'
routine
                                CALL fweight(grid, ion, numion) ! // re-calculate ion[].Ex/Ey
from grid[][].Ex/Ey
                                ! Fluxes thermal particles across boundaries into the
computational domain
                                ! it adds new particles and locate them: ion[].cellx/celly=0
                                CALL inject(ion, numion, &
                                superi_back, superi_front,superi_side, & ! the
super-particle weighting (Thesis, page 58)
                                INT(superii_back_FLOAT*i)-
INT(superii_back_FLOAT*(i-1)), & ! superii_back,
                                INT(superii_front_FLOAT*i)-
INT(superii_front_FLOAT*(i-1)), & !superii_front,
                                INT(superii_side_FLOAT*i)-
INT(superii_side_FLOAT*(i-1)), & !superii_side,
                                v_table_i, m_ion, q_ion) ! put particles on
boundaries with Ex=Ey=0
                                CALL mover(ion, numion, dti) ! moves particles with
acc=acc[q,Ex/Ey,m)
                                CALL boundary(ion, numion, q_ion,textWindow,graphWindow) !
collects the ion[].q current when on probe

!/*****
!!! did the user hit a key ? Then, switch ON/OFF animation

                                ! http://www.azmeer.info/?p=32
                                key1=#00 ! show that the keyboard was not pressed

resultOK= PeekConsoleInput( &      ! resultOK=true but inRec%Event=0,
always
                                GetStdHandle(STD_INPUT_HANDLE), &
                                !GETHANDLEFRAMEQQ(),      & !// handle of QuickWin input
buffer

```

```

!GETHANDLECHILDQQ(GETHANDLEQQ(QWIN$FRAMEWINDOW)), & !///  

handle of QuickWin console window input buffer      inRec,  

& !// address of the buffer for peek data  

!GETHANDLEQQ(QWIN$FRAMEWINDOW), & !///  

console window input buffer      inRec, & !///  

address of the buffer for peek data  

inRec(0), & !// address of buffer to  

receive data  

128, & !// number of records to  

read  

loc(lpcRead) & !// address of number of  

records read  

)  

! discard all unread records in a console's input  

buffer  

resultOK=FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))  

! resultOK = PEEKCHARQQ( ) ! does not read the buffer in  

Quicwin mode (resultOK==false, always)  

if ((inRec(0)%EventType == KEY_EVENT) .AND. (lpcRead  

/= 0)) THEN  

!   

http://nemesis.lonestar.org/reference/telecom/codes/ascii.html  

if (key1 == CHAR(#03)) STOP ! <CTRL><C> was  

pressed  

result = SETACTIVEQQ (textWindow)  

!result = FOCUSQQ (textWindow)  

if (animationON) THEN  

animationON =.false.  

!CALL gotoXY(10,45+cursorY)  

CALL SETTEXTPOSITION (INT2(45-1+cursorY),  

INT2(10), crsPos)  

WRITE(*, '(" OFF", $)')  

CALL WRITE_$ _TO_CONSOLE("\r"C) ! <=>  

begining of row  

CALL WRITE_$ _TO_CONSOLE('ANIMATION: OFF')  

else  

animationON=.true.  

!CALL gotoXY(10,45+cursorY)  

CALL SETTEXTPOSITION (INT2(45-1+cursorY),  

INT2(10), crsPos)  

WRITE(*, '(" ON ", $)')  

CALL WRITE_$ _TO_CONSOLE("\r"C) ! <=>  

begining of row  

CALL WRITE_$ _TO_CONSOLE('ANIMATION: ON ')  

END IF  

key1=#01 ! show that the keyboard was pressed

```

```

END IF ! lpcRead

! /*****
! is trajectories animation percentage set from the parameters file ?
if (percentAnimation <= 1.0e-6) goto 1234 ! skip display particle
trajectories
! is animation turned ON from the console ?
if (.NOT.(animationON)) THEN
    ! was the animation just now turned ON/OFF ? If so, erase the
screen
    if (key1==CHAR(#01)) THEN
        ! select Left Panel (maxwellian/trajectories)
        result = SETACTIVEQQ (graphWindow)
        !result = FOCUSQQ (graphWindow)
        CALL pgpanl(1, 1)
        CALL pgeras()
    ENDIF
    goto 1234
ENDIF
ENDIF

! ***** ions trajectories ***** //

result = SETACTIVEQQ (graphWindow)
!result = FOCUSQQ (graphWindow)

CALL pgsch (1.0/10.0) ! character drawing size; default = 1.0

! select Left Panel (maxwellian)
CALL pgpanl(1, 1)

! was the animation just now turned ON/OFF ? If so, erase the
screen
if (key1==CHAR(#01)) CALL pgeras()

! restore world->screen transformation
dx=length/10.0 ! some space around
the drawing area
dy=(grid(ngp_vert-1,0)%y)/10.0 ! some space around the
drawing area
! world coordinates:
CALL pgswin(-length-dx,0.0+dx, & ! 0x axis
            -radius-dy, radius+dy) ! 0y axis

! draw the domain
CALL pgsci(1) ! White
CALL pgmove(0.0,0.0)
CALL pgdraw(0.0,+radius)
CALL pgdraw(-length,+radius)
CALL pgdraw(-length,-radius)
CALL pgdraw(0.0,-radius)
CALL pgdraw(0.0,0.0)

```

```

! re-draw the probe/insulator 'damaged' by the particles
! draw the probe (conductor) in Blue
!
http://www.cse.iitb.ac.in/~cs101/Year2003/resources/notesPgplot/chapter
5.html
CALL pgsci(4) ! Blue
CALL pgrect(- l_insul - l_probe, 0.0, & ! X1, X2 : the
horizontal range of the rectangle.
-r_probe, r_probe) ! Y1, Y2 :
the vertical range of the rectangle.

! draw the insulator in Dark Gray, if present
if (l_insul > 0.0) THEN
CALL pgsci(14) ! Dark Gray
CALL pgrect(- l_insul, 0.0, & ! X1, X2 : the horizontal
range of the rectangle.
-r_insul, -r_probe) ! Y1, Y2 : the
vertical range of the rectangle.
CALL pgrect(- l_insul, 0.0, & ! X1, X2 : the horizontal
range of the rectangle.
r_insul, r_probe) ! Y1, Y2 : the
vertical range of the rectangle.
END IF

CALL pgsci(3) ! green
DO count1=0,numelec-1,100/percentAnimation
CALL pgpt(1,ion(count1)%x, ion(count1)%y,-1) ! graphic symbol
#(-1)
END DO

! flush any pending commands to the output device.
CALL pgupdt()

CALL pgsch (1.0) ! restore character drawing size to default = 1.0

goto 1234 ! skip display velocity distribution
!-----
! select Left Panel (maxwellian)
CALL PGPANL(1, 1)
! restore world->screen transformation
CALL PGSWIN(0.0,max_val,0.0,1.0)
! erase the old value
CALL PGSCI(0) ! black
CALL PGPTXT (2000000.0,0.65,0.0, 0.0,' t = '//string3//' sec') !
X, Y, ANGLE, FJUST, TEXT)
CALL PGPTXT (2000000.0,0.55,0.0, 0.0,' Potential
='//string2//'V') ! X, Y, ANGLE, FJUST, TEXT)
CALL PGPTXT (2000000.0,0.50,0.0, 0.0,'No. electrons: '//string)
! X, Y, ANGLE, FJUST, TEXT)
DO Ipg=0,noBins-1
CALL PGPT(1,bins(Ipg)%r,bins(Ipg)%v/bins(max_bin)%v,17)

```

```

END DO
  CALL PGUPDT ! display now !

  ! new value
  CALL DISTRIB(bins,noBins,electron,numelec)
  ! get the bin with the maximum number of particles in it
  max_bin = 0.0
  DO Ipg = 0,noBins-1
    if (bins(Ipg)%v > bins(max_bin)%v) max_bin = Ipg
  END DO

  ! redraw grid damaged by the graph
  CALL PGSCI(14) ! Dark Gray
  CALL PGBOX('G',0.0,0,'G',0.0,0) ! grid
  ! draw graph
  CALL PGSCI(6) ! magenta
  DO Ipg=0,noBins-1
    CALL PGPT(1,bins(Ipg)%r,bins(Ipg)%v/bins(max_bin)%v,17)
  END DO

  ! convert from FLOAT to STRING
  write(string3,'(E8.3)') i*dti
  write(string,'(I6)') numelec
  write(string2,'(F4.1)') V
  ! draw text
  CALL PGSCI(1) ! white
  CALL PGPTXT (2000000.0,0.65,0.0, 0.0,' t = '//string3//' sec') !
X, Y, ANGLE, FJUST, TEXT)
  CALL PGPTXT (2000000.0,0.55,0.0, 0.0,' Potential
='//string2//'V') ! X, Y, ANGLE, FJUST, TEXT)
  CALL PGPTXT (2000000.0,0.5,0.0, 0.0,'No. electrons: '//string) !
X, Y, ANGLE, FJUST, TEXT)

  CALL PGUPDT ! display now !
!-----
1234 CONTINUE
  ! select Right Panel (ion/electron)
  CALL PGPANL(2, 1)
  ! restore world->screen transformation
  CALL PGSWIN(0.0,tMax,0.0,100000.0) ! Lab-report.pdf (page 12)
  CALL PGSLW(4) ! line width
  CALL PGSCI(6) ! magenta for electrons
  !CALL PGPT(1,i*dti,numelec+0.0,17) ! 17: symbol number
  CALL PGPT(1,i*(dti*100000.0) ,numelec+0.0,-2) ! dt[sec] ->
dt[micro-sec]
  CALL PGSCI(3) ! green for ions
  !CALL PGPT(1,i*dti,numion+0.0,17)
  CALL PGPT(1,i*(dti*100000.0),numion+0.0,-2) ! dt[sec] ->
dt[micro-sec]
  CALL PGUPDT ! display now !
!-----

  t_count = t_count+1 ! will be reset to 0 in 'output_data'
  t_count2 = t_count2+1 ! =i =total number of time steps

```

```

!///// Progress:
result = SETACTIVEQQ (textWindow)
result = FOCUSQQ (textWindow)

! Percent
!CALL gotoXY(12,34+cursorY)
CALL SETTEXTPOSITION (INT2(34-1+cursorY), INT2(12+1), crsPos)
WRITE(*, '(F6.3,$)') i*100.0/iter

! Time
!CALL gotoXY(39,34+cursorY)
CALL SETTEXTPOSITION (INT2(34-1+cursorY), INT2(39+1), crsPos)
if (i*dti*1.0E6 < 0.001) THEN
    WRITE(*, '(G9.3,$)') i*dti*1.0E6
ELSE
    WRITE(*, '(F6.3,"    ",$)') i*dti*1.0E6
ENDIF

! Average current [mA]
!CALL gotoXY(67,44+cursorY) ! average current, displayed in
'output_data.cpp'
! CALL SETTEXTPOSITION (INT2(43+cursorY), INT2(62+2), crsPos)
CALL SETTEXTPOSITION (INT2(40+cursorY), INT2(62+2), crsPos)

IF (t_count == outputFrequency .OR. t_count2 == 1) THEN      !
500: outputFrequency
    CALL output_data(electron, ion, numelec, numion, m_elec, m_ion)
    ! clean the displayed fields 'hit electrons/ions'
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(39-6), crsPos)
    WRITE(*, '("          ",$)')
    CALL SETTEXTPOSITION (INT2(39-1+cursorY), INT2(62-6), crsPos)
    WRITE(*, '("          ",$)')
END IF

! Particles in domain
CALL SETTEXTPOSITION (INT2(41+1+cursorY), INT2(24+2), crsPos)
WRITE(*, '(I6,$)') numelec
CALL SETTEXTPOSITION (INT2(42+1+cursorY), INT2(24+2), crsPos)
WRITE(*, '(I6,$)') numion

END DO ! i
!write(*,*)

CALL PGERAS ! Erase all graphics from the current page or panel.

! END DO ! i3

! Finally, call PGCL0S to terminate things properly.
CALL PGCL0S
! Deallocate the console to free its resources.
resultOK = FreeConsole()

```



```

!!                                     NULL_SECURITY_ATTRIBUTES, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0)
!                                     NULL_SECURITY_ATTRIBUTES, OPEN_NEW,
FILE_ATTRIBUTE_NORMAL, 0)
!   if( hConsole == -1 ) then
!       ierror = GetLastError ()
!       if (ierror .eq. ERROR_ALREADY_EXISTS) then
!           ierror = 1
!       end if
!       write(*,*)
!       write (*,*) ("Error: Unable to open console.")
!       write(*,*)
!       stop 1
!   end if
!
!
!!1010 CONTINUE

!! Set buffer size variables
!   con_length = 80
!   con_lines  = 90
!!   Allocate a console
!   con_status = AllocConsole() ! get a console window of the
currently set size
!   con_handle = GetStdHandle(STD_OUTPUT_HANDLE)
!   con_wpos%x = con_length    ! must be >= currently set console
window line length
!   con_wpos%y = con_lines     ! must be >= currently set console
window number of lines
!
!   ! Set a console buffer bigger than the console window. This
provides
!   ! scroll bars on the console window to scroll through the console
buffer
!
!   con_status = SetConsoleScreenBufferSize(con_handle, con_wpos)
!
!   ! Write to the screen as needed. Add a READ to pause before
deallocation
!
!!// redirect unbuffered STDOUT to the console
!
!!   std_Handle = GetStdHandle(STD_OUTPUT_HANDLE)
!!   con_handle = _open_osfhandle(std_Handle, _O_TEXT)
!!   fp = fdopen( con_handle, "w" )
!!   stdout = fp
!!   setvbuf( stdout, NULL, _IONBF, 0 )
!
!!!   lpszFileName = "CONOUT$"C
!!   hConsole = CreateFile(lpszFileName, IOR(GENERIC_WRITE ,
GENERIC_READ),
&
!!                                     IOR(FILE_SHARE_READ , FILE_SHARE_WRITE),
&

```

```

!!                                     NULL_SECURITY_ATTRIBUTES, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0)
!!!                                     NULL_SECURITY_ATTRIBUTES, OPEN_NEW,
FILE_ATTRIBUTE_NORMAL, 0)
!
!
!OPEN(7,file="CONOUT$", ACTION='WRITE')
!
!      con_wpos%x = 5 ! 6 characters from left
!      con_wpos%y = 5 ! 6 lines down
!      !lstat = SetConsoleCursorPosition(hConsole, con_wpos) ! merge as
gotoXY
!      !lstat = SetConsoleCursorPosition(con_handle, con_wpos) ! merge
as gotoXY
!
!
!
!str = ""
!write(str,'(A)') ADJUSTL(TRIM('Asta ar trebui sa fie in CONSOLE'))
!str = trim(str) // "\n"C
!!result =
WriteConsole(hConsole, loc(str), len_Trim(str), loc(num_w_char), NULL)
!result =
WriteConsole(con_handle, loc(str), len_Trim(str), loc(num_w_char), NULL)

!!      Deallocate the console to free its resources.
!      con_status = FreeConsole()

!result = SETACTIVEQQ (graphWindow)
!!result = FOCUSQQ (graphWindow)

!ASCII_code = CHAR(#00) ! NULL
!key1 = CHAR(#00) ! NULL
!!key1 = 0 ! NULL
!

!WRITE(*,*)
!WRITE(*,*) GETHANDLEQQ(QWIN$FRAMEWINDOW)
!WRITE(*,*) GETHANDLEFRAMEQQ()
!WRITE(*,*) GetStdHandle(STD_INPUT_HANDLE) ! provides the Quickwin
console handle

!resultOK = ReadConsole(
!                                     & ! it works
!                                     !GETHANDLEQQ(QWIN$FRAMEWINDOW), & !// handle of
QuickWin input buffer
!                                     !GETHANDLEFRAMEQQ(), & !// handle of QuickWin
input buffer
!                                     GetStdHandle(STD_INPUT_HANDLE), & !// handle of
QuickWin input buffer
!                                     loc(inRec), & !// address of buffer
to receive data
!                                     1, & !// number of
characters to read

```

```

!                                loc(lpcRead),                & !// address of number
of characters read
!                                lpvReserved                & !// reserved
!                                )
!
!WRITE(*,*)
!WRITE(*,100) "ReadConsole: ",resultOK,lpcRead
!100 FORMAT (1X,A,1X,L1,I3)

!
!!!ASCII_code = inRec%Event%KeyEvent%AsciiChar
!!!key1 = inRec%Event%KeyEvent%AsciiChar
!

!WRITE(*,*)  ++++++-----
!WRITE(*,100) "PEEK: ",resultOK,lpcRead
!100 FORMAT (1X,A,1X,L1,I3)

!WRITE(*,102) "KEY_EVENT=",KEY_EVENT,  &
!"MOUSE_EVENT=",MOUSE_EVENT,          &
!"WINDOW_BUFFER_SIZE_EVENT=",WINDOW_BUFFER_SIZE_EVENT,&
!"MENU_EVENT=",MENU_EVENT,            &
!"FOCUS_EVENT=",FOCUS_EVENT
!102 FORMAT (1X,5(A,I3))

!WRITE(*,*)
!WRITE(*,101) "Event=",inRec%EventType," KEY_EVENT=", KEY_EVENT
!101 FORMAT (1X,A,I3,A,I3)

!! reads data from a console input buffer and removes it from the
buffer.
!! WAITS (execution stopped) for an event to happen
!      resultOK = ReadConsoleInput(          &
!      GetStdHandle(STD_INPUT_HANDLE),      & !// handle of
QuickWin input buffer
!!      GETHANDLEFRAMEQQ(), & !// handle of QuickWin
input buffer
!!      GETHANDLEQQ(QWIN$FRAMEWINDOW), & !// handle of
QuickWin console window input buffer      inRec,
& !// address of the buffer for peek data
!      inRec(0), & !// address of buffer to receive data
!      128, & !/* size of read buffer */
!      loc(lpcRead) & !// address of number
of characters read
!      )

!WRITE(*,*)
!write(*,103) resultOK,lpcRead
!103 FORMAT (1X,L1," ReadConsoleInput read ",I2," chars:")
!

```

```

!          DO i00 = 0, lpcRead-1
!              SELECT CASE (inRec(i00)%EventType)
!                  case (KEY_EVENT) !/* keyboard input */
!                      Write(*,104)
!                      inRec(i00)%Event%KeyEvent%AsciiChar
!                      key1 =
!                      inRec(i00)%Event%KeyEvent%AsciiChar
!104                      FORMAT(1X, "Event ",I2," is KBD:",A2)
!
!                  case (MOUSE_EVENT:FOCUS_EVENT) ! /* known event */
!                      Write(*,105) i00,inRec(i00)%EventType
!105                      FORMAT(1X, "Event ",I2," is ",I2)
!
!                  case default ! unrecognized event */
!                      Write(*,106) i00,inRec(i00)%EventType
!106                      FORMAT(1X, "Event ",I2," is ",I2, " invalid")
!
!              END SELECT
!          END DO

!          GetStdHandle(STD_INPUT_HANDLE),      & !// handle of
QuickWin input buffer
!          loc(inRec),                          & !// address of buffer
to receive data
!          1,                                  & !// number of
characters to read
!          loc(lpcRead),                      & !// address of number
of characters read
!          lpvReserved                        & !// reserved
!          )

!!          if (ASCII_code /= "\0"C ) THEN
!!          if (key1 /= CHAR(#00) ) THEN ! CHAR(#00) is not
considered 0 ???
!!          if (inRec%Event%KeyEvent%AsciiChar /= "\0"C ) THEN

!          if (inRec%EventType == KEY_EVENT) THEN
!WRITE(*,*) "KEY_EVENT =", KEY_EVENT
!          if (lpcRead /= 0 ) THEN
!WRITE(*,*) "KEYSTROKE ", lpcRead

!          if (resultOK) THEN
!key1=GETCHARQQ( ) ! remove the character from
buffer

!WRITE(*,*) "KEYSTROKE"

!resultOK = ReadConsole(
!          GetStdHandle(STD_INPUT_HANDLE),      & ! replaces GETCHARQQ( )
QuickWin input buffer
!          loc(inRec),                          & !// address of buffer
to receive data

```

```

!           1,           & !// number of
characters to read
!           loc(lpcRead),           & !// address of number
of characters read
!           lpvReserved           & !// reserved
!           )
!key1 = inRec%Event%KeyEvent%AsciiChar

!WRITE(*,*) key1
! To discard all unread records in a console's input buffer, use the
FlushConsoleInputBuffer function

```

random_number_v1.f90.

```

! -----
MODULE RANDOM_NUMBER_V1_1
! -----
USE LPROBE60_1

!/* Common Block Declarations */
  TYPE :: KLOTZ0_1_
    SEQUENCE
    REAL :: buff(0:607-1)
    INTEGER :: ptr
  END TYPE KLOTZ0_1_

  TYPE :: KLOTZ1_1_
    REAL :: xbuff(0:1024-1)
    INTEGER :: first
    INTEGER :: xptr
  END TYPE KLOTZ1_1_

!/* Initialized data */

  TYPE :: type_klotz0_
    SEQUENCE
    INTEGER :: fill_1(0:1214-1)
    INTEGER :: e_2
  END TYPE type_klotz0_

  TYPE(type_klotz0_) :: klotz0_
  DATA klotz0_ % fill_1, klotz0_ % e_2 / 1214*0, 0 /

!  TYPE :: klotz1_
!    REAL :: fill_1(1024)
!    INTEGER :: e_2(2)
!    REAL :: e_3
!  END TYPE klotz1_
!  TYPEDEF(klotz1_) :: klotz1_ = { {0}, 0, 0, 0.0 }

```

```

#define klotz0_1 (*(struct klotz0_1_ *) &klotz0_)
TYPE(klotz0_1_) :: klotz0_1
EQUIVALENCE (klotz0_1, klotz0_)

#define klotz1_1 (*(struct klotz1_1_ *) &klotz1_)
!TYPE(klotz1_1_) :: klotz1_1 = klotz1_

CONTAINS

! -----
SUBROUTINE random_number(seed)
! -----
! - - - arg types - - -
  INTEGER :: seed
! - - - local declarations - - -

! - - - begin - - -
  CALL zufalli_(seed)
  CALL zufall_(NPTS, random_num)

END SUBROUTINE random_number

! -----
FUNCTION zufall_(n,a) RESULT (output_4)
! -----
IMPLICIT NONE
! - - - arg types - - -
  INTEGER :: output_4
  INTEGER :: n
  REAL :: a(0:)
! - - - local declarations - - -
  INTEGER :: buffsz, left, aptr, bptr, aptr0, i, k, q, nn, vl, qq, k273, k607, kptr
  !INTEGER :: klotz0_1%ptr, t, zufalli_, seed, kl,
  !INTEGER :: svblk, zufallrs_, zufallsv_
  !INTEGER :: ij, j, l, m, ii, jj, s
  REAL :: t
  INTEGER :: t_int

! - - - begin - - -
  buffsz = 607

!/* portable lagged Fibonacci series uniform random number */
!/* generator with "lags" -273 und -607: */
!/* W.P. Petersen, IPS, ETH Zuerich, 19 Mar. 92 */

  aptr = 0
  nn = n
10  CONTINUE ! L1
  IF (nn .LE. 0) THEN
    output_4 = 0
    RETURN
  END IF

```

```

!/* factor nn = q*607 + r */

q = (nn - 1) / 607
left = buffsz - klotz0_1%ptr

IF (q <= 1) THEN

!/* only one or fewer full segments */

IF (nn < left) THEN
  kptr = klotz0_1%ptr
  DO i = 0,nn-1
    a(i + aptr+1) = klotz0_1%buff(kptr + i)
  END DO

  klotz0_1%ptr = klotz0_1%ptr + nn
  output_4 = 0
  RETURN
ELSE
  kptr = klotz0_1%ptr
  DO i = 0,left-1
    a(i + aptr) = klotz0_1%buff(kptr + i)
  END DO
  klotz0_1%ptr = 0
  aptr = aptr + left
  nn = nn - left
!/* buff -> buff case */
  vl = 273
  k273 = 334
  k607 = 0
  DO k = 0,3-1
    DO i = 0,vl-1
      t = klotz0_1%buff(k273+i)+klotz0_1%buff(k607+i)
      t_int = t ! the integer part
      klotz0_1%buff(k607+i) = t - t_int ! fractionary part, (FLOAT)
((long int) t);
    END DO ! i
    k607 = k607 + vl
    k273 = k273 + vl
    vl = 167
    IF (k == 0) k273 = 0
  END DO ! k
  GO TO 10 ! L1
END IF
ELSE

!/* more than 1 full segment */

  kptr = klotz0_1%ptr
  DO i = 0,left-1
    a(i + aptr) = klotz0_1%buff(kptr + i)
  END DO
  nn = nn - left
  klotz0_1%ptr = 0

```

```

    aptr = aptr + left

!/* buff -> a(aptr0) */

    vl = 273
    k273 = 334
    k607 = 0
    DO k = 0,3-1
        IF (k == 0) THEN
            DO i = 0,vl-1
                t = klotz0_1%buff(k273+i)+klotz0_1%buff(k607+i)
                t_int = t ! the integer part
                a(aptr + i) = t - t_int !(FLOAT) ((long int) t)
            END DO ! i
            k273 = aptr
            k607 = k607 + vl
            aptr = aptr + vl
            vl = 167
        ELSE
            DO i = 0,vl-1
                t = a(k273 + i) + klotz0_1%buff(k607 + i)
                t_int = t ! the integer part
                a(aptr + i) = t - t_int !(FLOAT) ((long int) t)
            END DO ! i
            k607 = k607 + vl
            k273 = k273 + vl
            aptr = aptr + vl
        END IF
    END DO ! k
    nn = nn +( -607)

!/* a(aptr-607) -> a(aptr) for last of the q-1 segments */

    aptr0 = aptr - 607
    vl = 607
    DO qq = 0,q-2-1
        k273 = aptr0 + 334
        DO i = 0,vl-1
            t = a(k273 + i) + a(aptr0 + i)
            t_int = t ! intger part
            a(aptr + i) = t - t_int !(FLOAT) ((long int) t)
        END DO ! i
        nn = nn +( -607)
        aptr = aptr + vl
        aptr0 = aptr0 + vl
    END DO ! qq

!/* a(aptr0) -> buff, last segment before residual */

    vl = 273
    k273 = aptr0 + 334
    k607 = aptr0
    bptr = 0
    DO k = 0,3-1

```



```

        IF (k == 0) THEN
            DO i = 0,vl-1
                t = a(k273 + i) + a(k607 + i)
                t_int = t ! integer part
                klotz0_1.buff(bptr + i) = t - t_int !(FLOAT) ((long int) t);
            END DO ! i
            k273 = 0
            k607 = k607 + vl
            bptr = bptr + vl
            vl = 167
        ELSE
            DO i = 0,vl-1
                t = klotz0_1.buff(k273 + i) + a(k607 + i)
                t_int = t ! integer part
                klotz0_1.buff(bptr + i) = t - t_int !(FLOAT) ((long int) t);
            END DO ! i
            k607 = k607 + vl
            k273 = k273 + vl
            bptr = bptr + vl
        END IF
    END DO
    GO TO 10 ! L1
END IF

END FUNCTION zufall_

FUNCTION zufalli_(seed) RESULT (output_4)
IMPLICIT NONE
! - - - arg types - - -
    INTEGER :: output_4
    INTEGER :: seed
! - - - local declarations - - -
    INTEGER :: kl = 9373
    INTEGER :: ij = 1802
    INTEGER :: i, j, k, l, m
    REAL :: s, t
    INTEGER :: ii, jj

!/* generates initial seed buffer by linear congruential */
!/* method. Taken from Marsaglia, FSU report FSU-SCRI-87-50 */
!/* variable seed should be 0 < seed <31328 */

! - - - begin - - -
    IF (seed /= 0) ij = seed

    i = MOD(ij / 177,177) + 2
    j = MOD(ij,177) + 2
    k = MOD(kl / 169,178) + 1
    l = MOD(kl,169)
    DO ii = 0,607-1
        s = 0.0
        t = 0.5
        DO jj = 1,24
            m = MOD(MOD(i * j,179) * k,179)

```

```

        i = j
        j = k
        k = m
        l = MOD((l * 53 + 1),169)
        IF (MOD(l * m,64) >= 32) s = s + t
        t = t *0.5
    END DO ! jj
        klotz0_1%buff(ii) = s
    END DO ! ii
    output_4 = 0
    RETURN
END FUNCTION zufalli_

```

```

FUNCTION zufallsv_(svblk) RESULT (output_4)
IMPLICIT NONE
! - - - arg types - - -
    INTEGER :: output_4
    REAL :: svblk(0:)
! - - - local declarations - - -
    INTEGER :: i

!/*  saves common blocks klotz0, containing seeds and */
!/*  polong inter to position in seed block. IMPORTANT: svblk must be */
!/*  dimensioned at least 608 in driver. The entire contents */
!/*  of klotz0 (polong inter in buff, and buff) must be saved. */

! - - - begin - - -
    svblk(0) = klotz0_1%ptr
    DO i = 0,607-1
        svblk(i + 1) = klotz0_1%buff(i)
    END DO
    output_4 = 0
    RETURN
END FUNCTION zufallsv_

```

```

FUNCTION zufallrs_(svblk) RESULT (output_4)
IMPLICIT NONE
! - - - arg types - - -
    INTEGER :: output_4
    REAL :: svblk
! - - - local declarations - - -
    INTEGER :: i

!/*  restores common block klotz0, containing seeds and pointer */
!/*  to position in seed block. IMPORTANT: svblk must be */
!/*  dimensioned at least 608 in driver. The entire contents */
!/*  of klotz0 must be restored. */

! - - - begin - - -

    klotz0_1%ptr = svblk(0)
    DO i = 0,607-1

```

```

        klotz0_1%buff(i) = svblk(i + 1)
    END DO
    output_4 = 0
    RETURN
END FUNCTION zufallrs_

END MODULE RANDOM_NUMBER_V1_1

```

RESET_GRID.F90.

```

! -----
MODULE RESET_GRID_1
! -----
USE LPROBE60_1

CONTAINS

! -----
SUBROUTINE reset_grid(array)
! -----
! - - - arg types - - -
    TYPE (GRIDDEF ) array(0:ngp1-1,0:ngp2-1)
! - - - local declarations - - -
    INTEGER :: count1,count2

! - - - begin - - -
!     CLEAN UP GRID      */
    DO count1 = 0,ngp_vert-1
        DO count2 = 0,ngp_horiz-1
            array(count1,count2)%q_dens = 0.0
        END DO
    END DO

END SUBROUTINE reset_grid
END MODULE RESET_GRID_1

```

MSFLIB.F90.

```

$freeform
MODULE msflib
!* fgraph.fi - declare constants and functions for graphics library
!*
!* Copyright (c) 1987-1995 Microsoft Corporation. All rights
reserved.
!*
!* Purpose:
!* This file declares the graphics library functions and
!* the constants that are used with them.
!*

```

```

!* NOTE: When modifying this file, you must follow the following rules
to
!* ensure compatibility with fixed form and free form source:
!* 1) All comments must start with '!'
!* 2) All code must be within columns 7 to 72
!* 3) For continuation, place a '&'amp;' in column 73,
!*      and a '&' in column 6 of the continuation line

```

```

$if .not. defined (_MSFORTRAN_)
$define _MSFORTRAN_ = 100
$endif

```

```

$if .not. defined($MSFLIB$ProgramUnitNumber)
$define $MSFLIB$ProgramUnitNumber = -1
$endif
$if .not. defined($MSFLIB$FIProgUnitNum)
$define $MSFLIB$FIProgUnitNum = -2
$endif

```

```

$if .not. defined (__MSFLIB_FLIB_INCLUDE)

```

```

!      User-visible declarations for FORTRAN Graphics Library

```

```

INTERFACE
  FUNCTION arc(x1,y1,x2,y2,x3,y3,x4,y4)
    INTEGER*2 arc[C,ALIAS:"__arc"],x1,y1,x2,y2,x3,y3,x4,y4
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  FUNCTION arc_w(wx1,wy1,wx2,wy2,wx3,wy3,wx4,wy4)
    INTEGER*2 arc_w[C,ALIAS:"__arc_w"]
    DOUBLE PRECISION wx1,wy1,wx2,wy2,wx3,wy3,wx4,wy4
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  FUNCTION getarcinfo(lpstart,lpend,lppaint)
    INTEGER*2 getarcinfo[C,ALIAS:"__getarcinfo"]
    STRUCTURE/xycoord/
      INTEGER*2 xcoord
      INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/lpstart[REFERENCE]
    RECORD/xycoord/lpend[REFERENCE]
    RECORD/xycoord/lppaint[REFERENCE]
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  SUBROUTINE clearscreen[C,ALIAS:"__FQclearscreen"](area)
    INTEGER*2 area

```

```

    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION displaycursor(toggle)
        INTEGER*2 displaycursor[C,ALIAS:"__FQdisplaycursor"],toggle
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION ellipse(control,x1,y1,x2,y2)
        INTEGER*2 ellipse[C,ALIAS:"__ellipse"],control,x1,y1,x2,y2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION ellipse_w(control,wx1,wy1,wx2,wy2)
        INTEGER*2 ellipse_w[C,ALIAS:"__ellipse_w"],control
        DOUBLE PRECISION wx1,wy1,wx2,wy2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION floodfill(x,y,boundary)
        INTEGER*2 floodfill[C,ALIAS:"__floodfill"],x,y,boundary
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION floodfillrgb(x,y,bcolor)
        INTEGER*2 floodfillrgb[C,ALIAS:"__floodfillrgb"],x,y
        INTEGER*4 bcolor
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION floodfill_w(wx1,wy1,boundary)
        INTEGER*2 floodfill_w[C,ALIAS:"__floodfill_w"],boundary
        DOUBLE PRECISION wx1,wy1
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION floodfillrgb_w(wx,wy,bcolor)
        INTEGER*2 floodfillrgb_w[C,ALIAS:"__floodfillrgb_w"]
        DOUBLE PRECISION wx,wy
        INTEGER*4 bcolor
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION getactivepage()
        INTEGER*2 getactivepage[C,ALIAS:"__getactivepage"]
    END FUNCTION

```

```

END INTERFACE

INTERFACE
    FUNCTION getbkcolor()
        INTEGER*4 getbkcolor[C,ALIAS:"__getbkcolor"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION getcolor()
        INTEGER*2 getcolor[C,ALIAS:"__getcolor"]
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE
getcurrentposition[ALIAS:"__f_getcurrentposition@4"](s)
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
getcurrentposition_w[ALIAS:"__f_getcurrentposition_wxy@4"](s)
        STRUCTURE/wxycoord/
            DOUBLE PRECISION wx
            DOUBLE PRECISION wy
        END STRUCTURE
        RECORD/wxycoord/s[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE getfillmask[C,ALIAS:"__getfillmask"](mask)
        INTEGER*1 mask[REFERENCE](8)
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE getimage[C,ALIAS:"__getimage"](x1,y1,x2,y2,image)
        INTEGER*2 x1,y1,x2,y2
        INTEGER*1 image[REFERENCE](*)
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
getimage_w[C,ALIAS:"__getimage_w"](wx1,wy1,wx2,wy2,image)
        DOUBLE PRECISION wx1,wy1,wx2,wy2
        INTEGER*1 image[REFERENCE](*)

```

```

        END SUBROUTINE
    END INTERFACE

    INTERFACE
        FUNCTION getlinestyle()
            INTEGER*2 getlinestyle[C,ALIAS:"__getlinestyle"]
        END FUNCTION
    END INTERFACE

$if _MSFORTRAN_ .lt. 300
    INTERFACE
        SUBROUTINE getphyscoord[ALIAS:"__f_getphyscoord@12"](x,y,s)
            INTEGER*2 x,y
            STRUCTURE/xycoord/
                INTEGER*2 xcoord
                INTEGER*2 ycoord
            END STRUCTURE
            RECORD/xycoord/s[REFERENCE]
        END SUBROUTINE
    END INTERFACE
$else
    INTERFACE GETPHYSCOORD
        SUBROUTINE $$MSFLIB$GETPHYSCOORD$22
&
        & [alias:'__f_getphyscoord@12'](x,y,s)
            INTEGER*2 x
            INTEGER*2 y
            STRUCTURE/xycoord/
                INTEGER*2 xcoord
                INTEGER*2 ycoord
            END STRUCTURE
            RECORD/xycoord/s[REFERENCE]
        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETPHYSCOORD$24
&
        & [alias:'__f_getphyscoord@12'](x,y,s)
            INTEGER*2 x
            INTEGER*4 y
            STRUCTURE/xycoord/
                INTEGER*2 xcoord
                INTEGER*2 ycoord
            END STRUCTURE
            RECORD/xycoord/s[REFERENCE]
        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETPHYSCOORD$42
&
        & [alias:'__f_getphyscoord@12'](x,y,s)
            INTEGER*4 x
            INTEGER*2 y
            STRUCTURE/xycoord/
                INTEGER*2 xcoord
                INTEGER*2 ycoord
            END STRUCTURE
            RECORD/xycoord/s[REFERENCE]

```

```

        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETPHYSCOORD$44
&
& [alias:'__f_getphyscoord@12'](x,y,s)
    INTEGER*4 x
    INTEGER*4 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

INTERFACE
    FUNCTION getpixel(x,y)
        INTEGER*2 getpixel[C,ALIAS:"__getpixel"],x,y
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION getpixel_w(wx,wy)
        INTEGER*2 getpixel_w[C,ALIAS:"__getpixel_w"]
        DOUBLE PRECISION wx,wy
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE getpixels[C, ALIAS:"__getpixels"](n, x, y, c)
        INTEGER*4 n                ! input : size of arrays
        INTEGER*2 x[REFERENCE](*)  ! input : x coordinates
        INTEGER*2 y[REFERENCE](*)  ! input : y coordinates
        INTEGER*2 c[REFERENCE](*)  ! input : palette indices
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION gettextcolor()
        INTEGER*2 gettextcolor[C,ALIAS:"__gettextcolor"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION gettextcursor()
        INTEGER*2 gettextcursor[C,ALIAS:"__gettextcursor"]
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE gettextposition[ALIAS:"__fq_gettextposition@4"](s)
        STRUCTURE/rccoord/

```



```

        INTEGER*2 row
        INTEGER*2 col
    END STRUCTURE
    RECORD/rccoord/s[REFERENCE]
END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
gettextwindow[C,ALIAS:"__gettextwindow"](r1,c1,r2,c2)
        INTEGER*2 r1[REFERENCE],c1[REFERENCE]
        INTEGER*2 r2[REFERENCE],c2[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE getvideoconfig[C,ALIAS:"__getvideoconfig"](s)
        STRUCTURE/videoconfig/
            INTEGER*2 numxpixels      ! number of pixels on X axis
            INTEGER*2 numypixels      ! number of pixels on Y axis
            INTEGER*2 numtextcols      ! number of text columns
available
            INTEGER*2 numtextrows      ! number of text rows
available
            INTEGER*2 numcolors        ! number of actual colors
            INTEGER*2 bitsperpixel     ! number of bits per pixel
            INTEGER*2 numvideopages    ! number of available video
pages
            INTEGER*2 mode              ! current video mode
            INTEGER*2 adapter           ! active display adapter
            INTEGER*2 monitor           ! active display monitor
            INTEGER*2 memory            ! adapter video memory in K
bytes
        END STRUCTURE
        RECORD/videoconfig/s[REFERENCE]
    END SUBROUTINE
END INTERFACE

$if _MSFORTRAN_ .lt. 300
    INTERFACE
        SUBROUTINE getviewcoord[ALIAS:"__f_getviewcoord@12"](x,y,s)
            INTEGER*2 x,y
            STRUCTURE/xycoord/
                INTEGER*2 xcoord
                INTEGER*2 ycoord
            END STRUCTURE
            RECORD/xycoord/s[REFERENCE]
        END SUBROUTINE
    END INTERFACE
$else
    INTERFACE GETVIEWCOORD
        SUBROUTINE $$MSFLIB$GETVIEWCOORD$22
&
        & [alias:'__f_getviewcoord@12'](x,y,s)

```

```

    INTEGER*2 x
    INTEGER*2 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$GETVIEWCOORD$24
&
& [alias:'__f_getviewcoord@12'](x,y,s)
    INTEGER*2 x
    INTEGER*4 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$GETVIEWCOORD$42
&
& [alias:'__f_getviewcoord@12'](x,y,s)
    INTEGER*4 x
    INTEGER*2 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$GETVIEWCOORD$44
&
& [alias:'__f_getviewcoord@12'](x,y,s)
    INTEGER*4 x
    INTEGER*4 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

INTERFACE
    SUBROUTINE
getviewcoord_w[ALIAS:"__f_getviewcoord_w@12"](wx,wy,s)
    DOUBLE PRECISION wx,wy
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]

```

```

        END SUBROUTINE
    END INTERFACE

    INTERFACE
        FUNCTION getvisualpage()
            INTEGER*2 getvisualpage[C,ALIAS:"__getvisualpage"]
        END FUNCTION
    END INTERFACE

$if _MSFORTRAN_ .lt. 300
    INTERFACE
        SUBROUTINE getwindowcoord[ALIAS:"__f_getwindowcoord@12"](x,y,s)
            INTEGER*2 x,y
            STRUCTURE/wxycoord/
                DOUBLE PRECISION wx
                DOUBLE PRECISION wy
            END STRUCTURE
            RECORD/wxycoord/s[REFERENCE]
        END SUBROUTINE
    END INTERFACE
$else
    INTERFACE GETWINDOWCOORD
        SUBROUTINE $$MSFLIB$GETWINDOWCOORD$22
&
        & [alias:'__f_getwindowcoord@12'](x,y,s)
            INTEGER*2 x
            INTEGER*2 y
            STRUCTURE/wxycoord/
                DOUBLE PRECISION wx
                DOUBLE PRECISION wy
            END STRUCTURE
            RECORD/wxycoord/s[REFERENCE]
        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETWINDOWCOORD$24
&
        & [alias:'__f_getwindowcoord@12'](x,y,s)
            INTEGER*2 x
            INTEGER*4 y
            STRUCTURE/wxycoord/
                DOUBLE PRECISION wx
                DOUBLE PRECISION wy
            END STRUCTURE
            RECORD/wxycoord/s[REFERENCE]
        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETWINDOWCOORD$42
&
        & [alias:'__f_getwindowcoord@12'](x,y,s)
            INTEGER*4 x
            INTEGER*2 y
            STRUCTURE/wxycoord/
                DOUBLE PRECISION wx
                DOUBLE PRECISION wy
            END STRUCTURE
            RECORD/wxycoord/s[REFERENCE]

```

```

        END SUBROUTINE
        SUBROUTINE $$MSFLIB$GETWINDOWCOORD$44
&
& [alias:'__f_getwindowcoord@12'](x,y,s)
    INTEGER*4 x
    INTEGER*4 y
    STRUCTURE/wxycoord/
        DOUBLE PRECISION wx
        DOUBLE PRECISION wy
    END STRUCTURE
    RECORD/wxycoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

INTERFACE
    FUNCTION getwritemode()
        INTEGER*2 getwritemode[C,ALIAS:"__getwritemode"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION grstatus()
        INTEGER*2 grstatus[C,ALIAS:"__grstatus"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION imagesize(x1,y1,x2,y2)
        INTEGER*4 imagesize[C,ALIAS:"__imagesize"]
        INTEGER*2 x1,y1,x2,y2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION imagesize_w(wx1,wy1,wx2,wy2)
        INTEGER*4 imagesize_w[C,ALIAS:"__imagesize_w"]
        DOUBLE PRECISION wx1,wy1,wx2,wy2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION lineto(x,y)
        INTEGER*2 lineto[C,ALIAS:"__lineto"],x,y
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION lineto_w(wx,wy)
        INTEGER*2 lineto_w[C,ALIAS:"__lineto_w"]
        DOUBLE PRECISION wx,wy
    END FUNCTION

```

```

END INTERFACE

INTERFACE
  INTEGER*4 FUNCTION
loadimage[ALIAS:"__f_loadimage@16"](fname,x,y)
  CHARACTER*(*) fname[REFERENCE]
  INTEGER*4 x,y
  END FUNCTION
END INTERFACE

INTERFACE
  INTEGER*4 FUNCTION loadimage_w[C,
ALIAS:"__loadimage_w"](fname,wx,wy)
  CHARACTER*(*) fname[REFERENCE]
  DOUBLE PRECISION wx, wy
  END FUNCTION
END INTERFACE

$if _MSFORTRAN_ .lt. 300
  INTERFACE
    SUBROUTINE moveto[ALIAS:"__f_moveto@12"](x,y,s)
      INTEGER*2 x,y
      STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
      END STRUCTURE
      RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
  END INTERFACE
$else
  INTERFACE MOVETO
  SUBROUTINE $$MSFLIB$MOVETO$22[alias:'__f_moveto@12'](x,y,s)
  INTEGER*2 x
  INTEGER*2 y
  STRUCTURE/xycoord/
    INTEGER*2 xcoord
    INTEGER*2 ycoord
  END STRUCTURE
  RECORD/xycoord/s[REFERENCE]
  END SUBROUTINE
  SUBROUTINE $$MSFLIB$MOVETO$24[alias:'__f_moveto@12'](x,y,s)
  INTEGER*2 x
  INTEGER*4 y
  STRUCTURE/xycoord/
    INTEGER*2 xcoord
    INTEGER*2 ycoord
  END STRUCTURE
  RECORD/xycoord/s[REFERENCE]
  END SUBROUTINE
  SUBROUTINE $$MSFLIB$MOVETO$42[alias:'__f_moveto@12'](x,y,s)
  INTEGER*4 x
  INTEGER*2 y
  STRUCTURE/xycoord/
    INTEGER*2 xcoord

```

```

        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$MOVETO$44[alias:'__f_moveto@12'](x,y,s)
    INTEGER*4 x
    INTEGER*4 y
    STRUCTURE/xycoord/
        INTEGER*2 xcoord
        INTEGER*2 ycoord
    END STRUCTURE
    RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

INTERFACE
    SUBROUTINE moveto_w[ALIAS:"__f_moveto_w@12"](wx,wy,s)
        DOUBLE PRECISION wx,wy
        STRUCTURE/wxycoord/
            DOUBLE PRECISION wx
            DOUBLE PRECISION wy
        END STRUCTURE
        RECORD/wxycoord/s[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE outtext[ALIAS:"__fq_outtext@8"](text)
        CHARACTER(*) text[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION pie(i,x1,y1,x2,y2,x3,y3,x4,y4)
        INTEGER*2 pie[C,ALIAS:"__pie"],i,x1,y1,x2,y2,x3,y3,x4,y4
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION pie_w(i,wx1,wy1,wx2,wy2,wx3,wy3,wx4,wy4)
        INTEGER*2 pie_w[C,ALIAS:"__pie_w"],i
        DOUBLE PRECISION wx1,wy1,wx2,wy2,wx3,wy3,wx4,wy4
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION polygon(control,lppoints,cpoints)
        INTEGER*2 polygon[C,ALIAS:"__polygon"],control,cpoints
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/lppoints[REFERENCE](*)

```

```

    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION polygon_w(control, lppoints, cpoints)
        INTEGER*2 polygon_w[C, ALIAS: "__polygon_w"], control, cpoints
        STRUCTURE/wxycoord/
            DOUBLE PRECISION wx
            DOUBLE PRECISION wy
        END STRUCTURE
        RECORD/wxycoord/lppoints[REFERENCE](*)
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE putimage[C, ALIAS: "__putimage"](x, y, image, action)
        INTEGER*2 x, y, action
        INTEGER*1 image[REFERENCE](*)
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
putimage_w[C, ALIAS: "__putimage_w"](wx, wy, image, action)
        DOUBLE PRECISION wx, wy
        INTEGER*1 image[REFERENCE](*)
        INTEGER*2 action
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION rectangle(control, x1, y1, x2, y2)
        INTEGER*2 rectangle[C, ALIAS: "__rectangle"]
        INTEGER*2 control, x1, y1, x2, y2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION rectangle_w(control, wx1, wy1, wx2, wy2)
        INTEGER*2 rectangle_w[C, ALIAS: "__rectangle_w"], control
        DOUBLE PRECISION wx1, wy1, wx2, wy2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION remappalette(index, color)
        INTEGER*4 remappalette[C, ALIAS: "__remappalette"], color
        INTEGER*2 index
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION remappalettergb(index, color)
        INTEGER*4 remappalettergb[C, ALIAS: "__remappalettergb"], color

```

```

        INTEGER*2 index
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION remapallpalette(colors)
        INTEGER*2 remapallpalette[C,ALIAS:"__remapallpalette"]
        INTEGER*4 colors[REFERENCE](*)
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION remapallpalettergb(colors)
        INTEGER*2 remapallpalettergb[C,ALIAS:"__remapallpalettergb"]
        INTEGER*4 colors[REFERENCE](*)
    END FUNCTION
END INTERFACE

INTERFACE
    INTEGER*4 FUNCTION
saveimage[ALIAS:"__f_saveimage@24"] (fname, x1, y1, x2, y2)
        CHARACTER*(*) fname[REFERENCE]
        INTEGER*4 x1, y1, x2, y2
    END FUNCTION
END INTERFACE

INTERFACE
    INTEGER*4 FUNCTION saveimage_w[C,
ALIAS:"__saveimage_w"] (fname, wx1, wy1, wx2, wy2)
        CHARACTER*(*) fname[REFERENCE]
        DOUBLE PRECISION wx1, wy1, wx2, wy2
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE
scrolltextwindow[C,ALIAS:"__FQscrolltextwindow"] (rows)
        INTEGER*2 rows
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION selectpalette(number)
        INTEGER*2 selectpalette[C,ALIAS:"__selectpalette"], number
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setactivepage(page)
        INTEGER*2 setactivepage[C,ALIAS:"__setactivepage"], page
    END FUNCTION
END INTERFACE

```



```

INTERFACE
  FUNCTION setbkcolor(color)
    INTEGER*4 setbkcolor[C,ALIAS:"__setbkcolor"],color
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE setcliprpn[C,ALIAS:"__setcliprpn"](x1,y1,x2,y2)
    INTEGER*2 x1,y1,x2,y2
  END SUBROUTINE
END INTERFACE

INTERFACE
  FUNCTION setcolor(color)
    INTEGER*2 setcolor[C,ALIAS:"__setcolor"]
    INTEGER*2 color
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE setfillmask[C,ALIAS:"__setfillmask"](mask)
    INTEGER*1 mask[REFERENCE](8)
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE setlinestyle[C,ALIAS:"__setlinestyle"](mask)
    INTEGER*2 mask
  END SUBROUTINE
END INTERFACE

INTERFACE
  FUNCTION setpixel(x,y)
    INTEGER*2 setpixel[C,ALIAS:"__setpixel"],x,y
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION setpixel_w(wx,wy)
    INTEGER*2 setpixel_w[C,ALIAS:"__setpixel_w"]
    DOUBLE PRECISION wx,wy
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE setpixels[C, ALIAS:"__setpixels"](n, x, y, c)
    INTEGER*4 n           ! input : size of arrays
    INTEGER*2 x[REFERENCE](*) ! input : x coordinates
    INTEGER*2 y[REFERENCE](*) ! input : y coordinates
    INTEGER*2 c[REFERENCE](*) ! input : palette indices
  END SUBROUTINE
END INTERFACE

```

```

INTERFACE
  FUNCTION settextcolor(index)
    INTEGER*2 settextcolor[C,ALIAS:"__settextcolor"],index
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION settextcursor(attr)
    INTEGER*2 settextcursor[C,ALIAS:"__settextcursor"],attr
  END FUNCTION
END INTERFACE

$if _MSFORTRAN_ .lt. 300
  INTERFACE
    SUBROUTINE
      settextposition[ALIAS:"__fq_settextposition@12"](row, col, s)
        INTEGER*2 row,col
        STRUCTURE/rcCOORD/
          INTEGER*2 row
          INTEGER*2 col
        END STRUCTURE
        RECORD/rcCOORD/s[REFERENCE]
      END SUBROUTINE
    END INTERFACE
$else
  INTERFACE SETTEXTPOSITION
    SUBROUTINE $$MSFLIB$SETTEXTPOSITION$22
&
    & [alias:'__fq_settextposition@12'](row, col, s)
      INTEGER*2 row
      INTEGER*2 col
      STRUCTURE/rcCOORD/
        INTEGER*2 row
        INTEGER*2 col
      END STRUCTURE
      RECORD/rcCOORD/s[REFERENCE]
      END SUBROUTINE
      SUBROUTINE $$MSFLIB$SETTEXTPOSITION$24
&
    & [alias:'__fq_settextposition@12'](row, col, s)
      INTEGER*2 row
      INTEGER*4 col
      STRUCTURE/rcCOORD/
        INTEGER*2 row
        INTEGER*2 col
      END STRUCTURE
      RECORD/rcCOORD/s[REFERENCE]
      END SUBROUTINE
      SUBROUTINE $$MSFLIB$SETTEXTPOSITION$42
&
    & [alias:'__fq_settextposition@12'](row, col, s)
      INTEGER*4 row
      INTEGER*2 col
      STRUCTURE/rcCOORD/

```

```

        INTEGER*2 row
        INTEGER*2 col
    END STRUCTURE
    RECORD/rccoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$SETTEXTPOSITION$44
&
& [alias:'__fq_settextposition@12'](row, col, s)
    INTEGER*4 row
    INTEGER*4 col
    STRUCTURE/rccoord/
        INTEGER*2 row
        INTEGER*2 col
    END STRUCTURE
    RECORD/rccoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

INTERFACE
    FUNCTION settextrows(rows)
        INTEGER*2 settextrows[C,ALIAS:"__settextrows"],rows
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE
settextwindow[C,ALIAS:"__settextwindow"](r1,c1,r2,c2)
        INTEGER*2 r1,c1,r2,c2
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION setvideomode(mode)
        INTEGER*2 setvideomode[C,ALIAS:"__setvideomode"],mode
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setvideomoderows(mode,rows)
        INTEGER*2 setvideomoderows[C,ALIAS:"__setvideomoderows"]
        INTEGER*2 mode,rows
    END FUNCTION
END INTERFACE

$if _MSFORTRAN_ .lt. 300
INTERFACE
    SUBROUTINE setvieworg[ALIAS:"__f_setvieworg@12"](x,y,s)
        INTEGER*2 x,y
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE

```

```

        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
END INTERFACE
$else
    INTERFACE SETVIEWORG
    SUBROUTINE $$MSFLIB$SETVIEWORG$22
&
    & [alias:'__f_setvieworg@12'](x,y,s)
        INTEGER*2 x
        INTEGER*2 y
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$SETVIEWORG$24
&
    & [alias:'__f_setvieworg@12'](x,y,s)
        INTEGER*2 x
        INTEGER*4 y
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$SETVIEWORG$42
&
    & [alias:'__f_setvieworg@12'](x,y,s)
        INTEGER*4 x
        INTEGER*2 y
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    SUBROUTINE $$MSFLIB$SETVIEWORG$44
&
    & [alias:'__f_setvieworg@12'](x,y,s)
        INTEGER*4 x
        INTEGER*4 y
        STRUCTURE/xycoord/
            INTEGER*2 xcoord
            INTEGER*2 ycoord
        END STRUCTURE
        RECORD/xycoord/s[REFERENCE]
    END SUBROUTINE
    END INTERFACE
$endif

    INTERFACE
        SUBROUTINE setviewport[C,ALIAS:"__setviewport"](x1,y1,x2,y2)

```

```

        INTEGER*2 x1,y1,x2,y2
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION setvisualpage(page)
        INTEGER*2 setvisualpage[C,ALIAS:"__setvisualpage"],page
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setwindow(finvert,wx1,wy1,wx2,wy2)
        INTEGER*2 setwindow[C,ALIAS:"__setwindow"]
        LOGICAL*2 finvert
        DOUBLE PRECISION wx1,wy1,wx2,wy2
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setwritemode(wmode)
        INTEGER*2 setwritemode[C,ALIAS:"__setwritemode"],wmode
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION wrapon(option)
        INTEGER*2 wrapon[C,ALIAS:"__wrapon"],option
    END FUNCTION
END INTERFACE

```

! FONTS

```

INTERFACE
    FUNCTION getfontinfo(fi)
        INTEGER*2 getfontinfo[ALIAS:"__f_getfontinfo@4"]
        STRUCTURE/fontinfo/
            INTEGER*4 type          ! b0 set = vector,clear = bit map
            INTEGER*4 ascent        ! pix dist from top to baseline
            INTEGER*4 pixwidth      ! character width in pixels, 0=prop
            INTEGER*4 pixheight     ! character height in pixels
            INTEGER*4 avgwidth      ! average character width in pixels
            CHARACTER*81 filename   ! file name including path
            CHARACTER*32 facename   ! font name
            LOGICAL*1 italic        ! .true. - font is italic
            LOGICAL*1 emphasized    ! .true. - font is emphasized
            LOGICAL*1 underline     ! .true. - font is underline
        END STRUCTURE
        RECORD/fontinfo/fi[REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION getgttexttent(text)

```

```

        INTEGER*2 getgtexttent[ALIAS:"__f_getgtexttent@8"]
        CHARACTER*(*) text[REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE getgtextvector[ALIAS:"__f_getgtextvector@8"](x,y)
        INTEGER*2 x[REFERENCE],y[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE outgtext[ALIAS:"__f_outgtext@8"](text)
        CHARACTER*(*) text[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION registerfonts(filename)
        INTEGER*2 registerfonts[ALIAS:"__f_registerfonts@8"]
        CHARACTER*(*) filename[REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION initializefonts()
        INTEGER*2 initializefonts[ALIAS:"__initializefonts"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setfont(options)
        INTEGER*2 setfont[ALIAS:"__f_setfont@8"]
        CHARACTER*(*) options[REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE settextfont[ALIAS:"__f_settextfont@4"] (fontname)
        CHARACTER*(*) fontname[REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE setgtextvector[C,ALIAS:"__setgtextvector"](x,y)
        INTEGER*2 x,y
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE unregisterfonts[C,ALIAS:"__unregisterfonts"]()
    END SUBROUTINE
END INTERFACE

```

```

INTERFACE
  SUBROUTINE
    setgtextrotation[C,ALIAS:"__setgtextrotation"](degrees)
      INTEGER*4 degrees
    END SUBROUTINE
  END INTERFACE

INTERFACE
  FUNCTION getgtextrotation()
    INTEGER*4 getgtextrotation[C,ALIAS:"__getgtextrotation"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION getcolorrgb()
    INTEGER*4 getcolorrgb[C,ALIAS:"__getcolorrgb"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION getbkcolorrgb()
    INTEGER*4 getbkcolorrgb[C,ALIAS:"__getbkcolorrgb"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION gettextcolorrgb()
    INTEGER*4 gettextcolorrgb[C,ALIAS:"__gettextcolorrgb"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION getpixelrgb(x,y)
    INTEGER*4 getpixelrgb[C,ALIAS:"__getpixelrgb"]
    INTEGER*2 x
    INTEGER*2 y
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION getpixelrgb_w(wx, wy)
    INTEGER*4 getpixelrgb_w[C,ALIAS:"__getpixelrgb_w"]
    REAL*8 wx
    REAL*8 wy
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE setpixelsrgb[C,ALIAS:"__setpixelsrgb"](n,x,y,color)
    INTEGER*4 n
    INTEGER*2 x [REFERENCE](*)
    INTEGER*2 y [REFERENCE](*)
    INTEGER*4 color [REFERENCE](*)

```

```

    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE getpixelsrgb[C,ALIAS:"__getpixelsrgb"](n,x,y,color)
        INTEGER*4 n
        INTEGER*2 x [REFERENCE](*)
        INTEGER*2 y [REFERENCE](*)
        INTEGER*4 color [REFERENCE](*)
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION setcolorrgb(color)
        INTEGER*4 setcolorrgb[C,ALIAS:"__setcolorrgb"]
        INTEGER*4 color
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setbkcolorrgb(color)
        INTEGER*4 setbkcolorrgb[C,ALIAS:"__setbkcolorrgb"]
        INTEGER*4 color
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setpixelrgb(x,y,color)
        INTEGER*4 setpixelrgb[C,ALIAS:"__setpixelrgb"]
        INTEGER*2 x
        INTEGER*2 y
        INTEGER*4 color
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION setpixelrgb_w(x,y,color)
        INTEGER*4 setpixelrgb_w[C,ALIAS:"__setpixelrgb_w"]
        REAL*8      x
        REAL*8      y
        INTEGER*4    color
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION settextrcolorrgb(color)
        INTEGER*4 settextrcolorrgb[C,ALIAS:"__settextcolorrgb"]
        INTEGER*4 color
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION rgbtointeger(red,green,blue)
        INTEGER*4 rgbtointeger[C,ALIAS:"__rgbtointeger"]

```



```

        INTEGER*4 red, green, blue
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE integertorgb[C, ALIAS: "__integertorgb"] (rgb, red,
green, blue)
        INTEGER*4 rgb
        INTEGER*4 red    [REFERENCE]
        INTEGER*4 green  [REFERENCE]
        INTEGER*4 blue   [REFERENCE]
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
setdblclkinterval[C, ALIAS: "_setdblclkintervalqq"](interval)
        INTEGER interval
    END SUBROUTINE
END INTERFACE

INTERFACE
    FUNCTION unregistermouseevent(Unit, MouseEvents)
        INTEGER
unregistermouseevent[C, ALIAS: "_unregistermouseeventqq"]
        INTEGER Unit
        INTEGER MouseEvents
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION registermouseevent(Unit, MouseEvents, CallBackRoutine)
        INTEGER registermouseevent[C, ALIAS: "_registermouseeventqq"]
        INTEGER Unit
        INTEGER MouseEvents
        EXTERNAL CallBackRoutine
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION waitonmouseevent(MouseEvents, KeyState, x, y)
        INTEGER waitonmouseevent[C, ALIAS: "_waitonmouseeventqq"]
        INTEGER MouseEvents
        INTEGER KeyState [REFERENCE]
        INTEGER x [REFERENCE]
        INTEGER y [REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    SUBROUTINE
setframewindow[C, ALIAS: "__setframewindow"](X, Y, width, height)
        INTEGER X
        INTEGER Y

```

```

        INTEGER width
        INTEGER height
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE setmessageqq[C,ALIAS:"__f_setmessageqq"](msg, id)
        CHARACTER*(*) msg[reference]
        INTEGER*4      id
    END SUBROUTINE
END INTERFACE

INTERFACE
    SUBROUTINE
setstatusmessage[C,ALIAS:"__f_setstatusmessage@8"](msg, id)
        CHARACTER*(*) msg[reference]
        INTEGER*4      id
    END SUBROUTINE
END INTERFACE

! We only include the following code once inside the same program unit.
! This code is shared with flib.fi.
$if _MSFORTRAN_ .lt. 300 ! if using /4fps1, DO NOT include the code
below
$undefine $MSFLIB$FIProgUnitNum
$define $MSFLIB$FIProgUnitNum = -1
$endif
$if $MSFLIB$FIProgUnitNum .ne. $MSFLIB$ProgramUnitNumber
$undefine $MSFLIB$FIProgUnitNum
$define $MSFLIB$FIProgUnitNum = $MSFLIB$ProgramUnitNumber

INTERFACE
    FUNCTION INCHARQQ()
        INTEGER*2 INCHARQQ[C,ALIAS:"__inchar"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WOPENQQ(name)
        INTEGER*4 WOPENQQ[C,ALIAS:"__wopen"]
        CHARACTER*(*) name
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGCLOSEQQ(handle)
        INTEGER*4 WGCLOSEQQ[C,ALIAS:"__wgclose"], handle
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGSETACTIVEQQ(handle)
        INTEGER*4 WGSETACTIVEQQ[C,ALIAS:"__wgsetactive"], handle
    END FUNCTION

```

```

        END INTERFACE

        INTERFACE
            FUNCTION WGGETACTIVEQQ()
                INTEGER*4 WGGETACTIVEQQ[C,ALIAS:"__wggetactive"]
            END FUNCTION
        END INTERFACE
$endif

$endif ! $if .not. defined (__MSFLIB_FLIB_INCLUDE)
!***** Module Header
!*****
!*
!* Copyright (c) 1990-1995  Microsoft Corporation
!*
!* Module Name: FLIB.FI
!*
!* This module provides interface descriptions for subprograms found
!* in the various Fortran libraries.
!*
!* The functional areas covered are the following.
!*
!*   Math error support
!*   Signal support
!*   Coprocessor Control
!*   Time and Date Support
!*   Command Line Arguments
!*   Random Numbers
!*   Running Programs and System Commands
!*   Drives and Directories
!*   Files
!*
!*
!*   Errors
!*   Environment
!*   Beep and Sleep
!*   Sorting and Searching Arrays
!*   Configuration
!*   QuickWin Support
!*   Access to Windows Handles for QuickWin components
!*   QuickWin Default Menu Support
!*
!*
!*****
*****

!* NOTE: When modifying this file, you must follow the following rules
to
!* ensure compatibility with fixed form and free form source:
!* 1) All comments must start with '!'
!* 2) All code must be within columns 7 to 72
!* 3) For continuation, place a '&' in column 73,

```

```

!*      and a '&' in column 6 of the continuation line

$if .not. defined (_MSFORTRAN_)
$define _MSFORTRAN_ = 100
$endif

$if .not. defined($MSFLIB$ProgramUnitNumber)
$define $MSFLIB$ProgramUnitNumber = -1
$endif
$if .not. defined($MSFLIB$FIProgUnitNum)
$define $MSFLIB$FIProgUnitNum = -2
$endif

$if .not. defined (__MSFLIB_FGRAPH_INCLUDE)

! -----
! Math Error Support
! -----
! MATHERRQQ -- user callbacks cannot be in the module MSFLIB

! -----
! Signal support
! -----
      INTERFACE
        INTEGER*4 FUNCTION SIGNALQQ[c,alias:'_signal'](SIGNAL, HANDLER)
          INTEGER*4 SIGNAL[value]
          integer*4 handler
          EXTERNAL HANDLER
        END FUNCTION
      END INTERFACE

      INTERFACE
        INTEGER*4 FUNCTION RAISEQQ[c,alias:'_raise'](SIGNAL)
          INTEGER*4 SIGNAL[value]
        END FUNCTION
      END INTERFACE

! -----
! Coprocessor Control
! -----

      INTERFACE
        SUBROUTINE LCWRQQ(CONTROL)
          INTEGER*2 CONTROL
        END SUBROUTINE
      END INTERFACE

      INTERFACE
        SUBROUTINE SCWRQQ(CONTROL)
          INTEGER*2 CONTROL
        END SUBROUTINE
      END INTERFACE

```

```

INTERFACE
  SUBROUTINE SSWRQQ(STATUS)
    INTEGER*2 STATUS
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE SETCONTROLFPQQ[alias: '_LCWRQQ@4'](CONTROL)
    INTEGER*2 CONTROL
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE GETCONTROLFPQQ[alias: '_SCWRQQ@4'](CONTROL)
    INTEGER*2 CONTROL
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE GETSTATUSFPQQ[alias: '_SSWRQQ@4'](STATUS)
    INTEGER*2 STATUS
  END SUBROUTINE
END INTERFACE

! -----
! Time and Date Support
! -----

INTERFACE
  SUBROUTINE GETTIM(IHR, IMIN, ISEC, I100TH)
    INTEGER*2 IHR, IMIN, ISEC, I100TH
  END SUBROUTINE
END INTERFACE

INTERFACE
  LOGICAL FUNCTION SETTIM(IHR, IMIN, ISEC, I100TH)
    INTEGER*2 IHR, IMIN, ISEC, I100TH
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE GETDAT(IYR, IMON, IDAY)
    INTEGER*2 IYR, IMON, IDAY
  END SUBROUTINE
END INTERFACE

INTERFACE
  LOGICAL FUNCTION SETDAT(IYR, IMON, IDAY)
    INTEGER*2 IYR, IMON, IDAY
  END FUNCTION
END INTERFACE

! -----
! Command Line Arguments
! -----

```

```

        INTERFACE
            INTEGER*4 FUNCTION NARGS()
        END FUNCTION
    END INTERFACE

$if _MSFORTRAN_ .LT. 300
    INTERFACE
        SUBROUTINE GETARG(N, BUFFER, STATUS)
            INTEGER*2 N
            CHARACTER*(*) BUFFER
            INTEGER*2 STATUS
        END SUBROUTINE
    END INTERFACE

    ! Note that EXIT does not have an interface for FORTRAN 77
    compiles.
    ! Thus, EXIT can be used with or without an argument
    ! (so long as it is used consistently within a program unit).
$else
    INTERFACE
        SUBROUTINE GETARG(N, BUFFER, STATUS)
            INTEGER*2 N
            CHARACTER*(*) BUFFER
            INTEGER*2, OPTIONAL :: STATUS
        END SUBROUTINE
    END INTERFACE

    INTERFACE
        SUBROUTINE EXIT(EXITVALUE)
            INTEGER*4, OPTIONAL :: EXITVALUE
        END SUBROUTINE
    END INTERFACE
$endif

! -----
! Random Numbers
! -----

$if _MSFORTRAN_ .LT. 300
    INTERFACE
        SUBROUTINE RANDOM(ARG)
            REAL*4 ARG
        END SUBROUTINE
    END INTERFACE
$else
    ! Random is a generic interface for use with the RANDOM in portlib
    INTERFACE RANDOM
        SUBROUTINE $$MSFLIB$RANDOM(ARG)
    !MS$ATTRIBUTES ALIAS: '_RANDOM@4' :: $$MSFLIB$RANDOM
        REAL*4 ARG
    END SUBROUTINE
    END INTERFACE
$endif

```

```

INTERFACE
  SUBROUTINE SEED(ARG)
    INTEGER*4 ARG
  END SUBROUTINE
END INTERFACE

```

```

! -----
! Running Programs and System Commands
! -----

```

```

INTERFACE
  INTEGER*4 FUNCTION RUNQQ(FILENAME, COMMANDLINE)
    CHARACTER*(*) FILENAME, COMMANDLINE
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  LOGICAL*4 FUNCTION SYSTEMQQ(COMMANDLINE)
    CHARACTER*(*) COMMANDLINE
  END FUNCTION
END INTERFACE

```

```

! -----
! Drives and Directories
! -----

```

```

INTERFACE
  LOGICAL*4 FUNCTION GETDRIVESIZEQQ(DRIVE, TOTAL, AVAIL)
    CHARACTER*(*) DRIVE
    INTEGER*4 TOTAL, AVAIL
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  CHARACTER*26 FUNCTION GETDRIVESQQ()
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  INTEGER*4 FUNCTION GETDRIVEDIRQQ(BUFFER)
    CHARACTER*(*) BUFFER
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  LOGICAL*4 FUNCTION CHANGEDRIVEQQ(DRIVE)
    CHARACTER*(*) DRIVE
  END FUNCTION
END INTERFACE

```

```

INTERFACE
  LOGICAL*4 FUNCTION CHANGEDIRQQ(DIR)
    CHARACTER*(*) DIR
  END FUNCTION
END INTERFACE

```

```

INTERFACE
    LOGICAL*4 FUNCTION MAKEDIRQQ(DIR)
        CHARACTER*(*) DIR
    END FUNCTION
END INTERFACE

INTERFACE
    LOGICAL*4 FUNCTION DELDIRQQ(DIR)
        CHARACTER*(*) DIR
    END FUNCTION
END INTERFACE

INTERFACE
    INTEGER*4 FUNCTION FULLPATHQQ(NAME, FULLPATH)
        CHARACTER*(*) NAME, FULLPATH
    END FUNCTION
END INTERFACE

INTERFACE
    INTEGER*4 FUNCTION SPLITPATHQQ(PATH, DRIVE, DIR, NAME, EXT)
        CHARACTER*(*) PATH, DRIVE, DIR, NAME, EXT
    END FUNCTION
END INTERFACE

! -----
! Files
! -----

INTERFACE
    INTEGER*4 FUNCTION GETFILEINFOQQ(FILES, BUFFER, HANDLE)
        CHARACTER*(*) FILES
        STRUCTURE / FILE$INFO /
            INTEGER*4      CREATION          ! Creation time (-1 on
FAT)
            INTEGER*4      LASTWRITE         ! Last write to file
            INTEGER*4      LASTACCESS        ! Last access (-1 on FAT)
            INTEGER*4      LENGTH            ! Length of file
            INTEGER*2      PERMIT            ! File access mode
            CHARACTER*255  NAME              ! File name
        END STRUCTURE
        RECORD / FILE$INFO / BUFFER
        INTEGER*4 HANDLE
    END FUNCTION
END INTERFACE

INTERFACE
    LOGICAL*4 FUNCTION SETFILETIMEQQ(NAME, TIMEDATE)
        CHARACTER*(*) NAME
        INTEGER*4 TIMEDATE
    END FUNCTION
END INTERFACE

INTERFACE

```



```

        LOGICAL*4 FUNCTION SETFILEACCESSQQ(NAME, ACCESS)
            CHARACTER*(*) NAME
            INTEGER*4 ACCESS
        END FUNCTION
    END INTERFACE

    INTERFACE
        INTEGER*4 FUNCTION DELFILESQQ(FILES)
            CHARACTER*(*) FILES
        END FUNCTION
    END INTERFACE

    INTERFACE
        LOGICAL*4 FUNCTION RENAMEFILEQQ(OLDNAME, NEWNAME)
            CHARACTER*(*) OLDNAME, NEWNAME
        END FUNCTION
    END INTERFACE

    INTERFACE
        INTEGER*4 FUNCTION FINDFILEQQ(FILE, ENV, BUF)
            CHARACTER*(*) FILE, ENV, BUF
        END FUNCTION
    END INTERFACE

    INTERFACE
        SUBROUTINE PACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY,IHR, IMIN,
ISEC)
            INTEGER*4 TIMEDATE
            INTEGER*2 IYR, IMON, IDAY, IHR, IMIN, ISEC
        END SUBROUTINE
    END INTERFACE

    INTERFACE
        SUBROUTINE UNPACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY,IHR, IMIN,
ISEC)
            INTEGER*4 TIMEDATE
            INTEGER*2 IYR, IMON, IDAY, IHR, IMIN, ISEC
        END SUBROUTINE
    END INTERFACE

    INTERFACE
        LOGICAL*4 FUNCTION COMMITQQ(UNIT)
            INTEGER*4 UNIT
        END FUNCTION
    END INTERFACE

    ! -----
    ! Keyboard
    ! -----

    INTERFACE
        CHARACTER*1 FUNCTION GETCHARQQ()
        END FUNCTION
    END INTERFACE

```

```

INTERFACE
  LOGICAL*4 FUNCTION PEEKCHARQQ()
  END FUNCTION
END INTERFACE

INTERFACE
  INTEGER*4 FUNCTION GETSTRQQ(BUFFER)
  CHARACTER*(*) BUFFER
  END FUNCTION
END INTERFACE

! -----
! Errors
! -----

INTERFACE
  INTEGER*4 FUNCTION GETLASTERRORQQ()
  END FUNCTION
END INTERFACE

INTERFACE
  SUBROUTINE SETERRORMODEQQ(PROMPT)
  LOGICAL*4 PROMPT
  END SUBROUTINE
END INTERFACE

! -----
! Environment
! -----

INTERFACE
  INTEGER*4 FUNCTION GETENVQQ(NAME, VALUE)
  CHARACTER*(*) NAME, VALUE
  END FUNCTION
END INTERFACE

INTERFACE
  LOGICAL*4 FUNCTION SETENVQQ(NAMVAL)
  CHARACTER*(*) NAMVAL
  END FUNCTION
END INTERFACE

! -----
! Beep and Sleep
! -----

INTERFACE
  SUBROUTINE SLEEPQQ(DURATION)
  INTEGER*4 DURATION
  END SUBROUTINE
END INTERFACE

```

```

INTERFACE
    SUBROUTINE BEEPQQ(FREQUENCY, DURATION)
        INTEGER*4 FREQUENCY, DURATION
    END SUBROUTINE
END INTERFACE

! -----
! Sorting and Searching Arrays
! -----

INTERFACE
    SUBROUTINE SORTQQ(ADRARRAY, LENGTH, SIZE)
        INTEGER*4 ADRARRAY, LENGTH, SIZE
    END SUBROUTINE
END INTERFACE

INTERFACE
    INTEGER*4 FUNCTION BSEARCHQQ(ADRKEY, ADRARRAY, LENGTH, SIZE)
        INTEGER*4 ADRKEY, ADRARRAY, LENGTH, SIZE
    END FUNCTION
END INTERFACE

! -----
! Configuration
! -----

INTERFACE
    INTEGER*4 FUNCTION GETCONFIGQQ()
    END FUNCTION
END INTERFACE

! -----
! QuickWin Support
! -----

$PACK:1

INTERFACE
    FUNCTION GETWINDOWCONFIG(wc)
        LOGICAL GETWINDOWCONFIG[ALIAS:"__f_getwindowconfig@4"]
        STRUCTURE/windowconfig/
        INTEGER*2 numxpixels
        INTEGER*2 numypixels
        INTEGER*2 numtextcols
        INTEGER*2 numtextrows
        INTEGER*2 numcolors
        INTEGER*4 fontsize
        CHARACTER*(80) title
        INTEGER*2 bitsperpixel
        INTEGER*2 numvideopages
        INTEGER*2 mode
        INTEGER*2 adapter
        INTEGER*2 monitor
        INTEGER*2 memory
        INTEGER*2 environment
    END FUNCTION
END INTERFACE

```

```

        CHARACTER*(32) extendfontname
        INTEGER*4 extendfontsize
        INTEGER*4 extendfontattributes
    END STRUCTURE
    RECORD/windowconfig/wc[REFERENCE]
END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION SETWINDOWCONFIG(wc)
        LOGICAL SETWINDOWCONFIG[ALIAS:"__f_setwindowconfig@4"]
        STRUCTURE/windowconfig/
            INTEGER*2 numxpixels
            INTEGER*2 numypixels
            INTEGER*2 numtextcols
            INTEGER*2 numtextrows
            INTEGER*2 numcolors
            INTEGER*4 fontsize
            CHARACTER*(80) title
            INTEGER*2 bitsperpixel
            INTEGER*2 numvideopages
            INTEGER*2 mode
            INTEGER*2 adapter
            INTEGER*2 monitor
            INTEGER*2 memory
            INTEGER*2 environment
            CHARACTER*(32) extendfontname
            INTEGER*4 extendfontsize
            INTEGER*4 extendfontattributes
        END STRUCTURE
        RECORD/windowconfig/wc[REFERENCE]
    END FUNCTION
END INTERFACE

$PACK

INTERFACE
    FUNCTION APPENDMENUQQ(menuID, flags, text, routine)
        LOGICAL APPENDMENUQQ[C]
        INTEGER*4 menuID, flags
        CHARACTER*(*) text[REFERENCE]
        EXTERNAL routine
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION INSERTMENUQQ(menuID, itemID, flags, text, routine)
        LOGICAL INSERTMENUQQ[C]
        INTEGER*4 menuID, itemID, flags
        CHARACTER*(*) text[REFERENCE]
        EXTERNAL routine
    END FUNCTION
END INTERFACE

INTERFACE

```

```

        FUNCTION DELETEMENUQQ(menuID,itemID)
            LOGICAL DELETEMENUQQ[C]
            INTEGER*4 menuID,itemID
        END FUNCTION
    END INTERFACE

    INTERFACE
        FUNCTION MODIFYMENUFLAGSQQ(menuID,itemID,flags)
            LOGICAL MODIFYMENUFLAGSQQ[C]
            INTEGER*4 menuID,itemID,flags
        END FUNCTION
    END INTERFACE

    INTERFACE
        FUNCTION MODIFYMENUSTRINGQQ(menuID,itemID,text)
            LOGICAL MODIFYMENUSTRINGQQ[C]
            INTEGER*4 menuID,itemID
            CHARACTER*(*) text[REFERENCE]
        END FUNCTION
    END INTERFACE

    INTERFACE
        FUNCTION MODIFYMENUROUTINEQQ(menuID,itemID,routine)
            LOGICAL MODIFYMENUROUTINEQQ[C]
            INTEGER*4 menuID,itemID
            EXTERNAL routine
        END FUNCTION
    END INTERFACE

    INTERFACE
        FUNCTION SETWINDOWMENUQQ(menuID)
            LOGICAL SETWINDOWMENUQQ[C]
            INTEGER*4 menuID
        END FUNCTION
    END INTERFACE

    INTERFACE
        SUBROUTINE YIELDQQ()
        END SUBROUTINE
    END INTERFACE

```

```

! We only include the following code once inside the same program unit.
! This code is shared with fgraph.fi.
$if _MSFORTRAN_ .lt. 300      ! if using /4fps1, always include the code
below
$undefine $MSFLIB$FIProgUnitNum
$define $MSFLIB$FIProgUnitNum = -2
$endif
$if $MSFLIB$FIProgUnitNum .ne. $MSFLIB$ProgramUnitNumber
$undefine $MSFLIB$FIProgUnitNum
$define $MSFLIB$FIProgUnitNum = $MSFLIB$ProgramUnitNumber

```

```

    INTERFACE
        FUNCTION INCHARQQ()

```

```

        INTEGER*2 INCHARQQ[C,ALIAS:"__inchar"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGOPENQQ(name)
        INTEGER*4 WGOPENQQ[C,ALIAS:"__wgopen"]
        CHARACTER*(*) name
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGCLOSEQQ(handle)
        INTEGER*4 WGCLOSEQQ[C,ALIAS:"__wgclose"], handle
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGSETACTIVEQQ(handle)
        INTEGER*4 WGSETACTIVEQQ[C,ALIAS:"__wgsetactive"], handle
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION WGGETACTIVEQQ()
        INTEGER*4 WGGETACTIVEQQ[C,ALIAS:"__wggetactive"]
    END FUNCTION
END INTERFACE
$endif

INTERFACE
    FUNCTION SETACTIVEQQ(lunit)
        INTEGER*4 SETACTIVEQQ[C,ALIAS:"__wgsetactiveunit"], lunit
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION GETACTIVEQQ()
        INTEGER*4 GETACTIVEQQ[C,ALIAS:"__wggetactiveunit"]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION FOCUSQQ(IUNIT)
        INTEGER*4 FOCUSQQ, IUNIT
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION INQFOCUSQQ(IUNIT)
        INTEGER*4 INQFOCUSQQ, IUNIT
    END FUNCTION
END INTERFACE

```

```

INTERFACE
    FUNCTION GETHANDLEQQ(IUNIT)
        INTEGER*4 GETHANDLEQQ, IUNIT
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION GETHWNDQQ(IUNIT)
        INTEGER*4 GETHWNDQQ, IUNIT
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION GETUNITQQ(IHANDLE)
        INTEGER*4 GETUNITQQ, IHANDLE
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION ABOUTBOXQQ(STR)
        INTEGER*4 ABOUTBOXQQ[C, ALIAS: "__QWINSetAboutString"]
        CHARACTER*(*) STR[REFERENCE]
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION CLICKQQ(ITEM)
        INTEGER*4 CLICKQQ[C, ALIAS: "__QWINMenuClick"], ITEM
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION CLICKMENUQQ(ITEM)
        INTEGER*4 CLICKMENUQQ[C, ALIAS: "__QWINMenuClick"]
        INTEGER*4 ITEM
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION SETWSIZEQQ(IUNIT, WINFO)
        STRUCTURE /QWINFO/
            INTEGER*2 TYPE           ! request type
            INTEGER*2 X              ! x coordinate for upper left
            INTEGER*2 Y              ! y coordinate for upper left
            INTEGER*2 H              ! window height
            INTEGER*2 W              ! window width
        END STRUCTURE
        INTEGER*4 SETWSIZEQQ, IUNIT
        RECORD /QWINFO/ WINFO
    END FUNCTION
END INTERFACE

INTERFACE

```

```

FUNCTION GETWSIZEQQ(IUNIT, IREQ, WINFO)
  STRUCTURE /QWINFO/
    INTEGER*2 TYPE          ! request type
    INTEGER*2 X              ! x coordinate for upper left
    INTEGER*2 Y              ! y coordinate for upper left
    INTEGER*2 H              ! window height
    INTEGER*2 W              ! window width
  END STRUCTURE
  INTEGER*4 GETWSIZEQQ, IUNIT
  INTEGER*4 IREQ
  RECORD /QWINFO/ WINFO
END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION MESSAGEBOXQQ(MSG, CAPTION, MTYPE)
    CHARACTER*(*) MSG[REFERENCE]
    CHARACTER*(*) CAPTION[REFERENCE]
    INTEGER*4 MESSAGEBOXQQ[C, ALIAS:"__QWINMsgBox"], MTYPE
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION GETEXITQQ()
    INTEGER*4 GETEXITQQ[C, ALIAS:"__QWINGetExit"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION SETEXITQQ(exitmode)
    INTEGER*4 SETEXITQQ[C, ALIAS:"__QWINSetExit"]
    INTEGER*4 exitmode
  END FUNCTION
END INTERFACE

! -----
! Access to Windows Handles for QuickWin components
! -----

INTERFACE
  FUNCTION GETHANDLEFRAMEQQ()
    INTEGER*4 GETHANDLEFRAMEQQ[C, ALIAS:"__QWINTGethFrame"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION GETHANDLECLIENTQQ()
    INTEGER*4 GETHANDLECLIENTQQ[C, ALIAS:"__QWINTGethClient"]
  END FUNCTION
END INTERFACE

INTERFACE
  FUNCTION GETHANDLECHILDQQ(QUICKHND)
    INTEGER*4 GETHANDLECHILDQQ[C, ALIAS:"__QWINGethWnd"]
    INTEGER*4 QUICKHND

```



```

        END FUNCTION
    END INTERFACE

$if _MSFORTRAN_ .GE. 300
! -----
! Unused Routine -- simply returns, used to avoid "unused" warnings
! -----
        INTERFACE
            SUBROUTINE UNUSEDQQ[C, REFERENCE, VARYING, ALIAS:"__FFunusedqq"]()
            END SUBROUTINE
        END INTERFACE
$endif

! -----
! QuickWin Default Menu Support
! -----
! INITIALMENU -- user callbacks cannot be in the module MSFLIB
! INITIALSETTINGS -- user callbacks cannot be in the module MSFLIB

        INTERFACE
            SUBROUTINE WINPRINT()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINSAVE()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINEXIT()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINCOPY()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINPASTE()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINSIZETOFIT()
            END SUBROUTINE
        END INTERFACE

        INTERFACE
            SUBROUTINE WINFULLSCREEN()
            END SUBROUTINE
        END INTERFACE

```

```

INTERFACE
  SUBROUTINE WINSTATE()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINCASCADE()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINTILE()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINARRANGE()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WININPUT()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINCLEARPASTE()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINSTATUS()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WININDEX()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINUSING()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINABOUT()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINSELECTTEXT()
  END SUBROUTINE
END INTERFACE

```

```

INTERFACE
  SUBROUTINE WINSELECTGRAPHICS()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE WINSELECTALL()
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE NUL()
  END SUBROUTINE
END INTERFACE

$endif ! $if .not. defined (__MSFLIB_FGRAPH_INCLUDE)
!* fgraph.fd - declare constants and functions for graphics library
!*
!* Copyright (c) 1987-1995 Microsoft Corporation. All rights
reserved.
!*
!* Purpose:
!* This file declares the graphics library functions and
!* the constants that are used with them.
!*

!* NOTE: When modifying this file, you must follow the following rules
to
!* ensure compatibility with fixed form and free form source:
!* 1) All comments must start with '!'
!* 2) All code must be within columns 7 to 72
!* 3) For continuation, place a '&' in column 73,
!* and a '&' in column 6 of the continuation line

$if .not. defined (_MSFORTRAN_)
$define _MSFORTRAN_ = 100
$endif
$if _MSFORTRAN_ .LT. 300
$NOTSTRICT ! uses structures which are non-standard conforming
$endif

$if .not. defined($MSFLIB$ProgramUnitNumber)
$define $MSFLIB$ProgramUnitNumber = -1
$endif
$if .not. defined($MSFLIB$FDProgUnitNum)
$define $MSFLIB$FDProgUnitNum = -2
$endif

$if .not. defined(__MSFLIB_FLIB_INCLUDE)

! User-visible declarations for FORTRAN Graphics Library

```

```

!   Structure for getvideoconfig() as visible to user
STRUCTURE/videoconfig/
    INTEGER*2 numxpixels      ! number of pixels on X axis
    INTEGER*2 numypixels      ! number of pixels on Y axis
    INTEGER*2 numtextcols     ! number of text columns available
    INTEGER*2 numtextrows     ! number of text rows available
    INTEGER*2 numcolors       ! number of actual colors
    INTEGER*2 bitsperpixel    ! number of bits per pixel
    INTEGER*2 numvideopages   ! number of available video pages
    INTEGER*2 mode            ! current video mode
    INTEGER*2 adapter         ! active display adapter
    INTEGER*2 monitor         ! active display monitor
    INTEGER*2 memory          ! adapter video memory in K bytes
END STRUCTURE

!   Return value of getcurrentposition(), etc.
STRUCTURE/xycoord/
    INTEGER*2 xcoord
    INTEGER*2 ycoord
END STRUCTURE

!   Structure for text position
STRUCTURE/rccoord/
    INTEGER*2 row
    INTEGER*2 col
END STRUCTURE

!   ERROR HANDLING

!   Status info returned by grstatus()
INTEGER*2 $GRPARAMETERALTERED, $GRCLIPPED, $GRNOOUTPUT, $GROK,
&
&          $GRERROR, $GRMODENOTSUPPORTED, $GRNOTINPROPERMODE,
&
&          $GRINVALIDPARAMETER, $GRFONTFILENOTFOUND,
&
&          $GRINVALIDFONTFILE, $GRCORRUPTEDFONTFILE,
&
&          $GRINSUFFICIENTMEMORY, $GRINVALIDIMAGEBUFFER,
&
&          $GRNOBITMAPFILE, $GRFILEREADERROR,
&
&          $GRIMPROPERBITMAPFORMAT, $GRBITMAPTOOLARGE,
&
&          $GRBITMAPDISPLAYERROR, $GRIMAGEREADERROR,
&
&          $GRFILEOPENERROR, $GRFILEWRITERROR
!   Successful
PARAMETER($GROK                                = 0)  ! success
!   Warnings
PARAMETER($GRNOOUTPUT                          = 1)  ! nothing drawn
PARAMETER($GRCLIPPED                          = 2)  ! output was partially
clipped

```

```

        PARAMETER($GRPARAMETERALTERED      = 3) ! input parameters
adjusted
!      Errors
        PARAMETER($GRERROR                  = -1) ! generic graphics
error
        PARAMETER($GRMODENOTSUPPORTED       = -2) ! video mode not
supported
        PARAMETER($GRNOTINPROPERMODE        = -3) ! not supported in
current mode
        PARAMETER($GRINVALIDPARAMETER       = -4) ! bad parameter
        PARAMETER($GRFONTFILENOTFOUND       = -5) ! font file not found
        PARAMETER($GRINVALIDFONTFILE        = -6) ! not a font file
        PARAMETER($GRCORRUPTEDFONTFILE      = -7) ! inconsistent font
file
        PARAMETER($GRINSUFFICIENTMEMORY     = -8) ! out of memory
        PARAMETER($GRINVALIDIMAGEBUFFER     = -9) ! bad image buffer
detected
        PARAMETER($GRNOBITMAPFILE           = -10) ! bitmap file not
found
        PARAMETER($GRFILEREADERROR          = -11) ! error reading bitmap
file
        PARAMETER($GRIMPROPERBITMAPFORMAT   = -12) ! bitmap format not
supported
        PARAMETER($GRBITMAPTOOLARGE        = -13) ! bitmap size > window
config
        PARAMETER($GRBITMAPDISPLAYERROR     = -14) ! error displaying
bitmap
        PARAMETER($GRIMAGEREADERROR         = -15) ! error reading image
        PARAMETER($GRFILEOPENERROR          = -16) ! error opening bitmap
file
        PARAMETER($GRFILEWRITERROR          = -17) ! error writing bitmap
file

!      SETUP AND CONFIGURATION

!      Arguments to setvideomode()
        INTEGER*2 $MAXRESMODE,$MAXCOLORMODE,$DEFAULTMODE,$TEXTBW40,
&
        &          $TEXTC40,$TEXTBW80,$TEXTC80,$TEXTMONO,$MRES16COLOR,
&
        &          $HRES16COLOR,$ERESNOCOLOR,$ERESCOLOR,$VRES2COLOR,
&
        &          $VRES16COLOR,$MRES256COLOR,$ORESCOLOR,$ORES256COLOR,
&
        &          $VRES256COLOR,$SRES16COLOR,$SRES256COLOR,$XRES16COLOR,&
        &          $XRES256COLOR,$ZRES16COLOR,$ZRES256COLOR
        PARAMETER($MAXRESMODE  =-3) ! graphics mode with highest
resolution
        PARAMETER($MAXCOLORMODE =-2) ! graphics mode with most colors
        PARAMETER($DEFAULTMODE  =-1) ! restore screen to original mode
        PARAMETER($TEXTBW40     =0) ! 40 x 25 text, 16 grey
        PARAMETER($TEXTC40      =1) ! 40 x 25 text, 16/8 color
        PARAMETER($TEXTBW80     =2) ! 80 x 25 text, 16 grey

```

```

PARAMETER($TEXTC80      =3)      ! 80 x 25 text, 16/8 color
PARAMETER($TEXTMONO     =7)      ! 80 x 25 text, BW
PARAMETER($MRES16COLOR  =13)     ! 320 x 200, 16 color
PARAMETER($HRES16COLOR  =14)     ! 640 x 200, 16 color
PARAMETER($ERESNOCOLOR  =15)     ! 640 x 350, BW
PARAMETER($ERESCOLOR    =16)     ! 640 x 350, 4 or 16 color
PARAMETER($VRES2COLOR   =17)     ! 640 x 480, BW
PARAMETER($VRES16COLOR  =18)     ! 640 x 480, 16 color
PARAMETER($MRES256COLOR =19)     ! 320 x 200, 256 color
PARAMETER($ORESCOLOR    =64)     ! 640 x 400, 1 of 16 colors
(Olivetti)

!      The following 8 modes require VESA SuperVGA BIOS extensions
PARAMETER($ORES256COLOR =#0100)   ! 640 x 400, 256 color
PARAMETER($VRES256COLOR =#0101)   ! 640 x 480, 256 color

!      WARNING: DO NOT attempt to set the following modes without
ensuring
!      that your monitor can safely handle that resolution. Otherwise,
you
!      may risk damaging your display monitor! Consult your owner's
manual
!      for details.
!      Note: _MAXRESMODE and _MAXCOLORMODE never select SRES, XRES, or
ZRES
!      modes.

!      Requires NEC MultiSync 3D or equivalent, or better
PARAMETER($SRES16COLOR  =#0102)   ! 800 x 600, 16 color
PARAMETER($SRES256COLOR =#0103)   ! 800 x 600, 256 color

!      Requires NEC MultiSync 4D or equivalent, or better
PARAMETER($XRES16COLOR  =#0104)   ! 1024 x 768, 16 color
PARAMETER($XRES256COLOR =#0105)   ! 1024 x 768, 256 color

!      Requires NEC MultiSync 5D or equivalent, or better
PARAMETER($ZRES16COLOR  =#0106)   ! 1280 x 1024, 16 color
PARAMETER($ZRES256COLOR =#0107)   ! 1280 x 1024, 256 color

!      VIDEOCONFIG adapter values

!      These constants can be used to determine the type of the active
!      adapter, using either simple comparisons or the bitwise-AND
!      operator
INTEGER*2 $QUICKWIN, $MDPA, $EGA, $VGA, $OEGA, $OVGA, $SVGA
PARAMETER($QUICKWIN =#0000) ! Quick Win Graphics is device
independent
PARAMETER($MDPA      =#0001) ! Monochrome Display Adapter (MDPA)
PARAMETER($EGA       =#0004) ! Enhanced Graphics Adapter (EGA)
PARAMETER($VGA       =#0008) ! Video Graphics Array (VGA)
PARAMETER($OEGA      =#0044) ! Olivetti Enhanced Graphics Adapter
(OEGA)
PARAMETER($OVGA      =#0048) ! Olivetti Video Graphics Array
(OVGA)

```

```

        PARAMETER($SVGA      =#0088) ! Super VGA with VESA BIOS support
(SVGA)

!     VIDEOCONFIG monitor values

!     These constants can be used to determine the type of monitor in
use,
!     using either simple comparisons or the bitwise-AND operator
        INTEGER*2 $MONO,$COLOR,$ENHCOLOR,$ANALOGMONO,
&
        &          $ANALOGCOLOR,$ANALOG
        PARAMETER($MONO      =#0001) ! Monochrome
        PARAMETER($COLOR      =#0002) ! Color (or Enhanced emulating
color)
        PARAMETER($ENHCOLOR   =#0004) ! Enhanced Color
        PARAMETER($ANALOGMONO =#0008) ! Analog Monochrome only
        PARAMETER($ANALOGCOLOR=#0010) ! Analog Color only
        PARAMETER($ANALOG     =#0018) ! Analog Color + Analog
Monochrome

!     COORDINATE SYSTEMS

!     OUTPUT ROUTINES

!     Control parameters for rectangle(), polygon(), ellipse(), and
pie()
        INTEGER*2 $GBORDER,$GFILLINTERIOR
        PARAMETER($GBORDER    =2) ! draw outline only
        PARAMETER($GFILLINTERIOR=3) ! fill using current fill mask

!     Parameters for clearsreen()
        INTEGER*2 $GCLEARSCREEN,$GVIEWPORT,$GWINDOW
        PARAMETER($GCLEARSCREEN =0)
        PARAMETER($GVIEWPORT    =1)
        PARAMETER($GWINDOW      =2)

! TEXT

!     Parameters for displaycursor()
        INTEGER*2 $GCURSOROFF,$GCURSORON
        PARAMETER($GCURSOROFF =0)
        PARAMETER($GCURSORON  =1)

!     Parameters for wrapon()
        INTEGER*2 $GWRAPOFF,$GWRAPON
        PARAMETER($GWRAPOFF    =0)
        PARAMETER($GWRAPON     =1)

!     Parameters for scrolltextwindow()
        INTEGER*2 $GSCROLLUP,$GSCROLLDOWN
        PARAMETER($GSCROLLUP   =1)
        PARAMETER($GSCROLLDOWN =-1)

```

```

!      Request maximum number of rows in _settextrrows() and
_setvideomoderows()
      INTEGER*2 $MAXTEXTROWS
      PARAMETER($MAXTEXTROWS =-1)

!      "Action verbs" for putimage(), setwritemode()
      INTEGER*2 $GPSET,$GPRESET,$GAND,$GOR,$GXOR
      PARAMETER($GPSET      =3)
      PARAMETER($GPRESET    =2)
      PARAMETER($GAND       =1)
      PARAMETER($GOR        =0)
      PARAMETER($GXOR       =4)

!      Color values are used with setbkcolor in graphics modes and also
by
!      remappalette and remapallpalette. Also known as palette colors.
!      Not to be confused with color indices (aka. color attributes).

!      Universal color values (all color modes):
      INTEGER*4 $BLACK,$BLUE,$GREEN,$CYAN,$RED,$MAGENTA,$BROWN,
&
&          $WHITE,$GRAY,$LIGHTBLUE,$LIGHTGREEN,$LIGHTCYAN,
&
&          $LIGHTRED,$LIGHTMAGENTA,$YELLOW,$BRIGHTWHITE
      PARAMETER($BLACK      =#000000)
      PARAMETER($BLUE       =#200000)
      PARAMETER($GREEN      =#002000)
      PARAMETER($CYAN       =#202000)
      PARAMETER($RED        =#000020)
      PARAMETER($MAGENTA    =#200020)
      PARAMETER($BROWN     =#002020)
      PARAMETER($WHITE      =#303030)
      PARAMETER($GRAY       =#202020)
      PARAMETER($LIGHTBLUE  =#3f0000)
      PARAMETER($LIGHTGREEN =#003f00)
      PARAMETER($LIGHTCYAN  =#3f3f00)
      PARAMETER($LIGHTRED   =#00003f)
      PARAMETER($LIGHTMAGENTA =#3f003f)
      PARAMETER($YELLOW     =#003f3f)
      PARAMETER($BRIGHTWHITE =#3f3f3f)

!      The following is obsolescent and defined only for backwards
!      compatibility
      INTEGER*4 $LIGHTYELLOW
      PARAMETER($LIGHTYELLOW =#153f3f)

!      Mono mode F ($ERESNOCOLOR) color values:
      INTEGER*4 $MODEFOFF,$MODEFOFFTOON,$MODEFOFFTOHI,$MODEFONTTOOFF,
&
&          $MODEFON,$MODEFONTTOHI,$MODEFHITOOFF,$MODEFHITOON,
&
&          $MODEFHI
      PARAMETER($MODEFOFF    =0)
      PARAMETER($MODEFOFFTOON =1)

```



```

PARAMETER($MODEFOFFTOHI =2)
PARAMETER($MODEFONT0OFF =3)
PARAMETER($MODEFON      =4)
PARAMETER($MODEFONT0HI  =5)
PARAMETER($MODEFHIT0OFF =6)
PARAMETER($MODEFHIT0ON  =7)
PARAMETER($MODEFHI      =8)

!      Mono mode 7 ($TEXTMONO) color values:
      INTEGER*4 $MODE7OFF,$MODE7ON,$MODE7HI
PARAMETER($MODE7OFF      =0)
PARAMETER($MODE7ON       =1)
PARAMETER($MODE7HI       =2)

!      QWIN Message IDS

      INTEGER*4 QWIN$MSG_TERM ,QWIN$MSG_EXITQ, QWIN$MSG_FINISHED,
&
& QWIN$MSG_PAUSED, QWIN$MSG_RUNNING, QWIN$MSG_FILEOPENDLG,
&
& QWIN$MSG_BMPSAVEDLG, QWIN$MSG_INPUTPEND,
&
& QWIN$MSG_PASTEINPUTPEND, QWIN$MSG_MOUSEINPUTPEND,
&
& QWIN$MSG_SELECTTEXT, QWIN$MSG_SELECTGRAPHICS,
&
& QWIN$MSG_PRINTABORT, QWIN$MSG_PRINTLOAD,
QWIN$MSG_PRINTNODEFAULT, &
& QWIN$MSG_PRINTDRIVER, QWIN$MSG_PRINTINGERROR, QWIN$MSG_PRINTING,
&
& QWIN$MSG_PRINTCANCEL, QWIN$MSG_PRINTINPROGRESS,
&
& QWIN$MSG_HELPNOTAVAIL, QWIN$MSG_TITLETEXT

PARAMETER (QWIN$MSG_TERM =1)
PARAMETER (QWIN$MSG_EXITQ =2)
PARAMETER (QWIN$MSG_FINISHED=3)
PARAMETER (QWIN$MSG_PAUSED =4)
PARAMETER (QWIN$MSG_RUNNING =5)
PARAMETER (QWIN$MSG_FILEOPENDLG =6)
PARAMETER (QWIN$MSG_BMPSAVEDLG =7)
PARAMETER (QWIN$MSG_INPUTPEND =8)
PARAMETER (QWIN$MSG_PASTEINPUTPEND =9)
PARAMETER (QWIN$MSG_MOUSEINPUTPEND =10)
PARAMETER (QWIN$MSG_SELECTTEXT =11)
PARAMETER (QWIN$MSG_SELECTGRAPHICS =12)
PARAMETER (QWIN$MSG_PRINTABORT =13)
PARAMETER (QWIN$MSG_PRINTLOAD =14)
PARAMETER (QWIN$MSG_PRINTNODEFAULT =15)
PARAMETER (QWIN$MSG_PRINTDRIVER =16)
PARAMETER (QWIN$MSG_PRINTINGERROR =17)
PARAMETER (QWIN$MSG_PRINTING =18)
PARAMETER (QWIN$MSG_PRINTCANCEL =19)
PARAMETER (QWIN$MSG_PRINTINPROGRESS =20)

```

```

PARAMETER (QWIN$MSG_HELPNOTAVAIL =21)
PARAMETER (QWIN$MSG_TITLETEXT =22)

!   WINDOW COORDINATE SYSTEM

!   Structure for window coordinate pair
STRUCTURE/wxycoord/
    DOUBLE PRECISION wx    ! window x coordinate
    DOUBLE PRECISION wy    ! window y coordinate
END STRUCTURE

!   Window coordinate entry points for graphics output routines

STRUCTURE/fontinfo/
    INTEGER*4 type          ! b0 set = vector,clear = bit map
    INTEGER*4 ascent        ! pix dist from top to baseline
    INTEGER*4 pixwidth      ! character width in pixels, 0=prop
    INTEGER*4 pixheight     ! character height in pixels
    INTEGER*4 avgwidth      ! average character width in pixels
    CHARACTER*81 filename   ! file name including path
    CHARACTER*32 facename   ! font name
    LOGICAL*1 italic        ! .true. - font is italic
    LOGICAL*1 emphasized    ! .true. - font is emphasized (bold)
    LOGICAL*1 underline     ! .true. - font is underline
END STRUCTURE

!   Font parameters

INTEGER*2 $NO_SPACE,$FIXED_SPACE,$PROP_SPACE
PARAMETER ($NO_SPACE      = 0)

PARAMETER ($FIXED_SPACE = 1)
PARAMETER ($PROP_SPACE  = 2)

INTEGER*2 $NO_FONT_MAP,$VECTOR_MAP,$BIT_MAP
PARAMETER ($NO_FONT_MAP = 0)
PARAMETER ($VECTOR_MAP  = 1)
PARAMETER ($BIT_MAP     = 2)

INTEGER MOUSE$LBUTTONDOWN, MOUSE$MOVE, MOUSE$LBUTTONUP,
&
& MOUSE$RBUTTONDOWN, MOUSE$RBUTTONUP, MOUSE$LBUTTONDBLCLK,
&
& MOUSE$RBUTTONDBLCLK, MOUSE$KS_LBUTTON, MOUSE$KS_RBUTTON,
&
& MOUSE$KS_SHIFT, MOUSE$KS_CONTROL, MOUSE$BADUNIT, MOUSE$BADEVENT

PARAMETER(MOUSE$LBUTTONDOWN      = #1)
PARAMETER(MOUSE$LBUTTONUP        = #2)
PARAMETER(MOUSE$LBUTTONDBLCLK    = #4)
PARAMETER(MOUSE$RBUTTONDOWN      = #8)

```

```

        PARAMETER(MOUSE$RBUTTONUP           = #10)
        PARAMETER(MOUSE$RBUTTONDBLCLK       = #20)
        PARAMETER(MOUSE$MOVE                 = #40)

        PARAMETER(MOUSE$KS_LBUTTON          = #1)
        PARAMETER(MOUSE$KS_RBUTTON          = #2)
        PARAMETER(MOUSE$KS_SHIFT            = #4)
        PARAMETER(MOUSE$KS_CONTROL          = #8)

        PARAMETER(MOUSE$BADUNIT              = -2)
        PARAMETER(MOUSE$BADEVENT            = -1)

$endif ! $if .not. defined(__MSFLIB_FLIB_INCLUDE)
!***** Module Header
!*****
!*
!* Copyright (c) 1990-1995 Microsoft Corporation
!*
!* Module Name: FLIB.FD
!*
!* This module provides parameter and structure type declarations
!* needed to communicate with subprograms found in the various Fortran
!* libraries.
!*
!* The functional areas covered are the following.
!*
!*   Data Type Codes
!*   Math error support
!*   Signal support
!*   Coprocessor Control
!*   Random Numbers
!*   Drives and Directories
!*   Files
!*   Keyboard
!*   Errors
!*   Sorting and Searching Arrays
!*   Configuration
!*   QuickWin Support
!*
!*****
!*****

!* NOTE: When modifying this file, you must follow the following rules
to
!* ensure compatibility with fixed form and free form source:
!*   1) All comments must start with '!'
!*   2) All code must be within columns 7 to 72
!*   3) For continuation, place a '&' in column 73,
!*       and a '&' in column 6 of the continuation line

$if .not. defined (_MSFORTRAN_)
$define _MSFORTRAN_ = 100
$endif

```

```

$if .not. defined($MSFLIB$ProgramUnitNumber)
$define $MSFLIB$ProgramUnitNumber = -1
$endif
$if .not. defined($MSFLIB$FDProgUnitNum)
$define $MSFLIB$FDProgUnitNum = -2
$endif

$if .not. defined(__MSFLIB_FGRAPH_INCLUDE)

! -----
! Data Type Codes
! -----

      INTEGER*4 TY$REAL4
      INTEGER*4 TY$REAL8
      INTEGER*4 TY$CMPLX8
      INTEGER*4 TY$CMPLX16

      PARAMETER (TY$REAL4      = 1)
      PARAMETER (TY$REAL8      = 2)
      PARAMETER (TY$CMPLX8     = 3)
      PARAMETER (TY$CMPLX16    = 4)

! -----
! Math Error Support
! -----
! Codes for function failure.
      INTEGER*4 MTH$_DOMAIN           ! Argument domain error
      INTEGER*4 MTH$_SINGULARITY      ! Argument Singularity
      INTEGER*4 MTH$_OVERFLOW         ! Overflow range error
      INTEGER*4 MTH$_UNDERFLOW        ! Underflow range error
      INTEGER*4 MTH$_TLOSS            ! Total loss of precision
      INTEGER*4 MTH$_PLOSS            ! Partial loss of precision

      PARAMETER (MTH$_DOMAIN      = 1)
      PARAMETER (MTH$_SINGULARITY = 2)
      PARAMETER (MTH$_OVERFLOW    = 3)
      PARAMETER (MTH$_UNDERFLOW   = 4)
      PARAMETER (MTH$_TLOSS       = 5)
      PARAMETER (MTH$_PLOSS       = 6)

! Math error information structure.
      STRUCTURE /MTH$_INFO/
      INTEGER*4 ERRCODE           ! INPUT : One of the MTH$ values above
      INTEGER*4 FTYPE             ! INPUT : One of the TY$ values above
      UNION
      MAP
      REAL*4 R4ARG1               ! INPUT : First argument
      REAL*4 R4ARG2               ! INPUT : Second argument (if any)
      REAL*4 R4RES                 ! OUTPUT : Desired result
      END MAP
      MAP

```

```

        REAL*8 R8ARG1      ! INPUT : First argument
        REAL*8 R8ARG2      ! INPUT : Second argument (if any)
        REAL*8 R8RES       ! OUTPUT : Desired result
    END MAP
    MAP
        COMPLEX*8 C8ARG1    ! INPUT : First argument
        COMPLEX*8 C8ARG2    ! INPUT : Second argument (if any)
        COMPLEX*8 C8RES     ! OUTPUT : Desired result
    END MAP
    MAP
        COMPLEX*16 C16ARG1  ! INPUT : First argument
        COMPLEX*16 C16ARG2  ! INPUT : Second argument (if any)
        COMPLEX*16 C16RES   ! OUTPUT : Desired result
    END MAP
    END UNION
END STRUCTURE

```

```

! -----
! Signal support
! -----
    INTEGER*4 SIG$ERR          ! SIGNALQQ return value
    INTEGER*4 SIG$NSIG
    INTEGER*4 SIG$INT
    INTEGER*4 SIG$IILL
    INTEGER*4 SIG$FPE
    INTEGER*4 SIG$SEGV
    INTEGER*4 SIG$TERM
    INTEGER*4 SIG$USR1
    INTEGER*4 SIG$USR2
    INTEGER*4 SIG$USR3
    INTEGER*4 SIG$BREAK
    INTEGER*4 SIG$ABORT

    PARAMETER (SIG$ERR   = -1)
    PARAMETER (SIG$NSIG  = 23)
    PARAMETER (SIG$INT   =  2)
    PARAMETER (SIG$IILL  =  4)
    PARAMETER (SIG$FPE   =  8)
    PARAMETER (SIG$SEGV  = 11)
    PARAMETER (SIG$TERM  = 15)
    PARAMETER (SIG$USR1  = 16)
    PARAMETER (SIG$USR2  = 17)
    PARAMETER (SIG$USR3  = 20)
    PARAMETER (SIG$BREAK = 21)
    PARAMETER (SIG$ABORT = 22)

    INTEGER*4 FPE$INVALID
    INTEGER*4 FPE$DENORMAL
    INTEGER*4 FPE$ZERODIVIDE
    INTEGER*4 FPE$OVERFLOW
    INTEGER*4 FPE$UNDERFLOW
    INTEGER*4 FPE$INEXACT
    INTEGER*4 FPE$UNEMULATED

```

```

INTEGER*4 FPE$SQRTNEG
INTEGER*4 FPE$STACKOVERFLOW
INTEGER*4 FPE$STACKUNDERFLOW
INTEGER*4 FPE$EXPLICITGEN                                ! RAISEQQ( SIGFPE )

PARAMETER ( FPE$INVALID          = #81)
PARAMETER ( FPE$DENORMAL         = #82)
PARAMETER ( FPE$ZERODIVIDE       = #83)
PARAMETER ( FPE$OVERFLOW        = #84)
PARAMETER ( FPE$UNDERFLOW       = #85)
PARAMETER ( FPE$INEXACT         = #86)
PARAMETER ( FPE$UNEMULATED      = #87)
PARAMETER ( FPE$SQRTNEG         = #88)
PARAMETER ( FPE$STACKOVERFLOW   = #8a)
PARAMETER ( FPE$STACKUNDERFLOW = #8b)
PARAMETER ( FPE$EXPLICITGEN     = #8c)

! -----
! Coprocessor control
! -----

INTEGER*2 FPCW$MCW_EM
INTEGER*2 FPCW$INEXACT
INTEGER*2 FPCW$UNDERFLOW
INTEGER*2 FPCW$OVERFLOW
INTEGER*2 FPCW$ZERODIVIDE
INTEGER*2 FPCW$INVALID
INTEGER*2 FPCW$DENORMAL

INTEGER*2 FPCW$MCW_PC
INTEGER*2 FPCW$64
INTEGER*2 FPCW$53
INTEGER*2 FPCW$24

INTEGER*2 FPCW$MCW_IC
INTEGER*2 FPCW$AFFINE
INTEGER*2 FPCW$PROJECTIVE

INTEGER*2 FPCW$MCW_RC
INTEGER*2 FPCW$NEAR
INTEGER*2 FPCW$DOWN
INTEGER*2 FPCW$UP
INTEGER*2 FPCW$CHOP

INTEGER*2 FPSW$MSW_EM
INTEGER*2 FPSW$INVALID
INTEGER*2 FPSW$DENORMAL
INTEGER*2 FPSW$ZERODIVIDE
INTEGER*2 FPSW$OVERFLOW
INTEGER*2 FPSW$UNDERFLOW
INTEGER*2 FPSW$INEXACT

PARAMETER ( FPCW$MCW_EM          = #003F) ! exception mask
PARAMETER ( FPCW$INVALID        = #0001) ! invalid

```

```

PARAMETER (FPCW$DENORMAL      = #0002) ! denormal
PARAMETER (FPCW$ZERODIVIDE    = #0004) ! zero divide
PARAMETER (FPCW$OVERFLOW      = #0008) ! overflow
PARAMETER (FPCW$UNDERFLOW     = #0010) ! underflow
PARAMETER (FPCW$INEXACT       = #0020) ! inexact (precision)

PARAMETER (FPCW$MCW_PC        = #0300) ! precision control mask
PARAMETER (FPCW$64            = #0300) ! 64 bits
PARAMETER (FPCW$53            = #0200) ! 53 bits
PARAMETER (FPCW$24            = #0000) ! 24 bits

PARAMETER (FPCW$MCW_IC        = #1000) ! infinity control mask
PARAMETER (FPCW$AFFINE        = #1000) ! affine
PARAMETER (FPCW$PROJECTIVE     = #0000) ! projective

PARAMETER (FPCW$MCW_RC        = #0C00) ! rounding control mask
PARAMETER (FPCW$CHOP          = #0C00) ! chop
PARAMETER (FPCW$UP            = #0800) ! up
PARAMETER (FPCW$DOWN          = #0400) ! down
PARAMETER (FPCW$NEAR          = #0000) ! near

PARAMETER (FPSW$MSW_EM        = #003F) ! exception mask
PARAMETER (FPSW$INVALID       = #0001) ! invalid
PARAMETER (FPSW$DENORMAL      = #0002) ! denormal
PARAMETER (FPSW$ZERODIVIDE    = #0004) ! zero divide
PARAMETER (FPSW$OVERFLOW      = #0008) ! overflow
PARAMETER (FPSW$UNDERFLOW     = #0010) ! underflow
PARAMETER (FPSW$INEXACT       = #0020) ! inexact (precision)

! -----
! Random Numbers
! -----

INTEGER*4 RND$TIMESEED

PARAMETER (RND$TIMESEED = -1)

! -----
! Drives and Directories
! -----

CHARACTER*1 FILE$CURDRIVE
INTEGER*4   FILE$MAXNAME
INTEGER*4   $MAXPATH

PARAMETER (FILE$CURDRIVE = ' ')
PARAMETER (FILE$MAXNAME = 255)
PARAMETER ($MAXPATH = 260)

! -----
! Files
! -----

STRUCTURE / FILE$INFO /
  INTEGER*4      CREATION      ! Creation time (-1 on FAT)

```

```

        INTEGER*4      LASTWRITE      ! Last write to file
        INTEGER*4      LASTACCESS     ! Last access (-1 on FAT)
        INTEGER*4      LENGTH         ! Length of file
        INTEGER*2      PERMIT         ! File access mode
        CHARACTER*255   NAME          ! File name
END STRUCTURE

INTEGER*4 FILE$NORMAL
INTEGER*4 FILE$READONLY
INTEGER*4 FILE$HIDDEN
INTEGER*4 FILE$SYSTEM
INTEGER*4 FILE$VOLUME
INTEGER*4 FILE$DIR
INTEGER*4 FILE$ARCHIVE
INTEGER*4 FILE$FIRST
INTEGER*4 FILE$LAST
INTEGER*4 FILE$ERROR
INTEGER*4 FILE$INVALID
INTEGER*4 FILE$CURTIME

PARAMETER (FILE$NORMAL      = #0000)
PARAMETER (FILE$READONLY   = #0001)
PARAMETER (FILE$HIDDEN     = #0002)
PARAMETER (FILE$SYSTEM     = #0004)
PARAMETER (FILE$VOLUME     = #0008)
PARAMETER (FILE$DIR        = #0010)
PARAMETER (FILE$ARCHIVE    = #0020)

PARAMETER (FILE$FIRST      = -1)
PARAMETER (FILE$LAST       = -2)
PARAMETER (FILE$ERROR      = -3)

PARAMETER (FILE$INVALID    = -1) ! For PACKTIMEQQ and
UNPACKTIMEQQ
PARAMETER (FILE$CURTIME    = -1) ! For SETFILETIMEQQ

! -----
! Keyboard
! -----

! -----
! Errors
! -----

INTEGER*4 ERR$ZERO
INTEGER*4 ERR$PERM
INTEGER*4 ERR$NOENT
INTEGER*4 ERR$SRCH
INTEGER*4 ERR$INTR
INTEGER*4 ERR$IO
INTEGER*4 ERR$NXIO
INTEGER*4 ERR$2BIG
INTEGER*4 ERR$NOEXEC

```


INTEGER*4 ERR\$BADF
 INTEGER*4 ERR\$CHILD
 INTEGER*4 ERR\$AGAIN
 INTEGER*4 ERR\$NOMEM
 INTEGER*4 ERR\$ACCES
 INTEGER*4 ERR\$FAULT
 INTEGER*4 ERR\$NOTBLK
 INTEGER*4 ERR\$BUSY
 INTEGER*4 ERR\$EXIST
 INTEGER*4 ERR\$XDEV
 INTEGER*4 ERR\$NODEV
 INTEGER*4 ERR\$NOTDIR
 INTEGER*4 ERR\$ISDIR
 INTEGER*4 ERR\$INVAL
 INTEGER*4 ERR\$NFILE
 INTEGER*4 ERR\$MFILE
 INTEGER*4 ERR\$NOTTY
 INTEGER*4 ERR\$TXTBSY
 INTEGER*4 ERR\$FBIG
 INTEGER*4 ERR\$NOSPC
 INTEGER*4 ERR\$SPIPE
 INTEGER*4 ERR\$ROFS
 INTEGER*4 ERR\$MLINK
 INTEGER*4 ERR\$PIPE
 INTEGER*4 ERR\$DOM
 INTEGER*4 ERR\$RANGE
 INTEGER*4 ERR\$UCLEAN
 INTEGER*4 ERR\$DEADLOCK

PARAMETER (ERR\$ZERO	= 0)
PARAMETER (ERR\$PERM	= 1)
PARAMETER (ERR\$NOENT	= 2)
PARAMETER (ERR\$SRCH	= 3)
PARAMETER (ERR\$INTR	= 4)
PARAMETER (ERR\$IO	= 5)
PARAMETER (ERR\$NXIO	= 6)
PARAMETER (ERR\$2BIG	= 7)
PARAMETER (ERR\$NOEXEC	= 8)
PARAMETER (ERR\$BADF	= 9)
PARAMETER (ERR\$CHILD	= 10)
PARAMETER (ERR\$AGAIN	= 11)
PARAMETER (ERR\$NOMEM	= 12)
PARAMETER (ERR\$ACCES	= 13)
PARAMETER (ERR\$FAULT	= 14)
PARAMETER (ERR\$NOTBLK	= 15)
PARAMETER (ERR\$BUSY	= 16)
PARAMETER (ERR\$EXIST	= 17)
PARAMETER (ERR\$XDEV	= 18)
PARAMETER (ERR\$NODEV	= 19)
PARAMETER (ERR\$NOTDIR	= 20)
PARAMETER (ERR\$ISDIR	= 21)
PARAMETER (ERR\$INVAL	= 22)
PARAMETER (ERR\$NFILE	= 23)
PARAMETER (ERR\$MFILE	= 24)

```

PARAMETER (ERR$NOTTY           = 25)
PARAMETER (ERR$TXTBSY          = 26)
PARAMETER (ERR$FBIG            = 27)
PARAMETER (ERR$NOSPC           = 28)
PARAMETER (ERR$SPIPE           = 29)
PARAMETER (ERR$ROFS            = 30)
PARAMETER (ERR$MLINK           = 31)
PARAMETER (ERR$PIPE            = 32)
PARAMETER (ERR$DOM             = 33)
PARAMETER (ERR$RANGE           = 34)
PARAMETER (ERR$UCLEAN          = 35)
PARAMETER (ERR$DEADLOCK        = 36)

LOGICAL*4 ERR$HARDPROMPT      ! For SETERRORMODEQQ
LOGICAL*4 ERR$HARDFAIL

PARAMETER (ERR$HARDPROMPT     = .TRUE.)
PARAMETER (ERR$HARDFAIL      = .FALSE.)

! -----
! Sorting and Searching Arrays
! -----

INTEGER*4 SRT$REAL4
INTEGER*4 SRT$REAL8
INTEGER*4 SRT$INTEGER1
INTEGER*4 SRT$INTEGER2
INTEGER*4 SRT$INTEGER4

PARAMETER (SRT$REAL4          = #00010000)
PARAMETER (SRT$REAL8          = #00020000)
PARAMETER (SRT$INTEGER1       = #00030000)
PARAMETER (SRT$INTEGER2       = #00040000)
PARAMETER (SRT$INTEGER4       = #00050000)

! -----
! Configuration
! -----

INTEGER*4 CFG$WINNT
PARAMETER (CFG$WINNT          = #00000001)

! -----
! QuickWin Support
! -----
$PACK:1
! structure for setwindowconfig/getwindowconfig
STRUCTURE /windowconfig/
  INTEGER*2 numxpixels
  INTEGER*2 numypixels
  INTEGER*2 numtextcols
  INTEGER*2 numtextrows
  INTEGER*2 numcolors

```

```

        INTEGER*4 fontsize
        CHARACTER*(80) title
        INTEGER*2 bitsperpixel
        INTEGER*2 numvideopages
        INTEGER*2 mode
        INTEGER*2 adapter
        INTEGER*2 monitor
        INTEGER*2 memory
        INTEGER*2 environment
        CHARACTER*(32) extendfontname
        INTEGER*4 extendfontsize
        INTEGER*4 extendfontattributes
    END STRUCTURE
$PACK

!     Parameters for APPENDMENUQQ, INSERTMENUQQ, MODIFYMENUFLAGSQQ
    INTEGER*4 $MENUMGRAYED, $MENUDISABLED, $MENUENABLED
    INTEGER*4 $MENUSEPARATOR, $MENCHECKED, $MENUUNCHECKED
    PARAMETER($MENUMGRAYED=1)
    PARAMETER($MENUDISABLED=2)
    PARAMETER($MENUENABLED=4)
    PARAMETER($MENUSEPARATOR=8)
    PARAMETER($MENCHECKED=16)
    PARAMETER($MENUUNCHECKED=32)

! Clicking on menus.
    INTEGER*4 QWIN$STATUS
    INTEGER*4 QWIN$TILE
    INTEGER*4 QWIN$CASCADE
    INTEGER*4 QWIN$ARRANGE

    PARAMETER (QWIN$STATUS   = 1)
    PARAMETER (QWIN$TILE     = 2)
    PARAMETER (QWIN$CASCADE  = 3)
    PARAMETER (QWIN$ARRANGE  = 4)

! Extended font information

    INTEGER*4 QWIN$EXTENDFONT
    INTEGER*4 QWIN$EXTENDFONT_NORMAL
    INTEGER*4 QWIN$EXTENDFONT_UNDERLINE
    INTEGER*4 QWIN$EXTENDFONT_BOLD
    INTEGER*4 QWIN$EXTENDFONT_ITALIC

    PARAMETER (QWIN$EXTENDFONT           =#FFFFFFFE)
    PARAMETER (QWIN$EXTENDFONT_NORMAL    =#80000000)
    PARAMETER (QWIN$EXTENDFONT_BOLD      =#00000001)
    PARAMETER (QWIN$EXTENDFONT_UNDERLINE =#00000002)
    PARAMETER (QWIN$EXTENDFONT_ITALIC    =#00000004)

! Size/Move a window
    STRUCTURE /QWINFO/
        INTEGER*2 TYPE           ! request type
        INTEGER*2 X              ! x coordinate for upper left

```

```

        INTEGER*2 Y           ! y coordinate for upper left
        INTEGER*2 H           ! window height
        INTEGER*2 W           ! window width
END STRUCTURE

INTEGER*2 QWIN$MIN
INTEGER*2 QWIN$MAX
INTEGER*2 QWIN$RESTORE
INTEGER*2 QWIN$SET

PARAMETER (QWIN$MIN      = 1)
PARAMETER (QWIN$MAX      = 2)
PARAMETER (QWIN$RESTORE = 3)
PARAMETER (QWIN$SET      = 4)

INTEGER*4 QWIN$FRAMEWINDOW
INTEGER*4 QWIN$NOACTIVIEWINDOW
PARAMETER (QWIN$FRAMEWINDOW = #80000000)
PARAMETER (QWIN$NOACTIVIEWINDOW = #C0000000)

INTEGER*4 QWIN$FRAMEMAX
INTEGER*4 QWIN$FRAMECURR
INTEGER*4 QWIN$CHILDMAX
INTEGER*4 QWIN$CHILDCURR
INTEGER*4 QWIN$SIZEMAX
INTEGER*4 QWIN$SIZECURR

PARAMETER (QWIN$FRAMEMAX = 1)
PARAMETER (QWIN$FRAMECURR = 2)
PARAMETER (QWIN$CHILDMAX = 3)
PARAMETER (QWIN$CHILDCURR = 4)
PARAMETER (QWIN$SIZEMAX = 3)
PARAMETER (QWIN$SIZECURR = 4)

!Get/Set Exit
INTEGER*4 QWIN$EXITPROMPT, QWIN$EXITNOPERSIST, QWIN$EXITPERSIST

PARAMETER(QWIN$EXITPROMPT= 1)      ! prompt before exiting
PARAMETER(QWIN$EXITNOPERSIST= 2)   ! exit immediately
PARAMETER(QWIN$EXITPERSIST= 3)     ! do not exit appilcation

! Message Boxes
INTEGER*4 MB$ABORTRETRYIGNORE
INTEGER*4 MB$DEFBUTTON1
INTEGER*4 MB$DEFBUTTON2
INTEGER*4 MB$DEFBUTTON3
INTEGER*4 MB$ICONASTERISK
INTEGER*4 MB$ICONEXCLAMATION
INTEGER*4 MB$ICONHAND
INTEGER*4 MB$ICONINFORMATION
INTEGER*4 MB$ICONQUESTION
INTEGER*4 MB$ICONSTOP
INTEGER*4 MB$OK
INTEGER*4 MB$OKCANCEL

```

```

INTEGER*4 MB$RETRYCANCEL
INTEGER*4 MB$SYSTEMMODAL
INTEGER*4 MB$TASKMODAL
INTEGER*4 MB$YESNO
INTEGER*4 MB$YESNOCANCEL

PARAMETER (MB$ABORTRETRYIGNORE = #00000002)
PARAMETER (MB$DEFBUTTON1      = #00000000)
PARAMETER (MB$DEFBUTTON2      = #00000100)
PARAMETER (MB$DEFBUTTON3      = #00000200)
PARAMETER (MB$ICONASTERISK     = #00000040)
PARAMETER (MB$ICONEXCLAMATION = #00000030)
PARAMETER (MB$ICONHAND         = #00000010)
PARAMETER (MB$ICONINFORMATION = #00000040)
PARAMETER (MB$ICONQUESTION     = #00000020)
PARAMETER (MB$ICONSTOP         = #00000010)
PARAMETER (MB$OK               = #00000000)
PARAMETER (MB$OKCANCEL         = #00000001)
PARAMETER (MB$RETRYCANCEL      = #00000005)
PARAMETER (MB$SYSTEMMODAL      = #00001000)
PARAMETER (MB$TASKMODAL        = #00002000)
PARAMETER (MB$YESNO            = #00000004)
PARAMETER (MB$YESNOCANCEL      = #00000003)

INTEGER*4 MB$IDOK
INTEGER*4 MB$IDCANCEL
INTEGER*4 MB$IDABORT
INTEGER*4 MB$IDRETRY
INTEGER*4 MB$IDIGNORE
INTEGER*4 MB$IDYES
INTEGER*4 MB$IDNO

PARAMETER (MB$IDOK      = 1)
PARAMETER (MB$IDCANCEL = 2)
PARAMETER (MB$IDABORT  = 3)
PARAMETER (MB$IDRETRY  = 4)
PARAMETER (MB$IDIGNORE = 5)
PARAMETER (MB$IDYES    = 6)
PARAMETER (MB$IDNO     = 7)

! InitialMenu should not be typed here, so that when InitialMenu is
! defined by the user it can include FLIB.FD. Nor should it be
external.
!      LOGICAL*4 INITIALMENU[EXTERN]

$endif ! $if .not. defined(__MSFLIB_FGRAPH_INCLUDE)

END MODULE msflib

```

Vita

Denise Su (Densu) A. Aktas was born in 1987 in Istanbul. She finished her high school in 2001 and entered the university to study Electronic Engineering degree at Marmara University in Istanbul. She graduated from the university in 2005. During her studying engineering, she won several scholarships from honor organizations both in her country and abroad. Then she got her second degree in Electromechanical Engineering Technology at MTSU in 2007 to combine her electrical and mechanical knowledge for technical applications. Because of her professors' recommendations, she entered the graduate school at UT Space Institute in 2007 to pursue Masters Degree. She completed the degree requirements for both Aviation Systems and Mechanical Engineering. She plans to pursue her PhD degree also in Mechanical Engineering specializing in electric propulsion.