UNIVERSITY OF CALIFORNIA, SAN DIEGO

Implantation, Flux and Recoil Distributions for Plasma Species Impinging on Tokamak Divertor Materials.

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Engineering Sciences (Aerospace Engineering)

by

Nathan David Moshman

Committee in charge:

Professor Sergei Krasheninnikov, Chair
Professor Farhat Beg
Professor Alexander Pigarov

2009

The Thesis of Nathan David Moshman is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

Table of Contents

List of Symbols

a	screening distance
a_0	Bohr radius
(α, β, γ)	angular displacement from (x,y,z) directions respectively
$(\alpha', \beta', \gamma')$	angular displacement from (x,y,z) after hard collision
Δ	is a correction factor
ΔE_l	local energy loss from weak inelastic collisions
ΔE_{nl}	non-local energy loss from weak inelastic collisions
$\Delta\phi$	interval of fluence
e	charge of electron
E	energy of particle between collisions
E'	kinetic energy of the projectile after collision
E_{bb}	bulk binding energy of solid target material
E_{b0}	initial kinetic energy of recoil
E_c	is the center of mass energy, particle and target atoms
E_0	initial kinetic energy of incident particle
ε_0	permittivity of free space
n	local atomic number density
J_{scal}	scalar particle flux distribution
λ	local mean free path
m_{target}	atomic mass of target atom
m_{proj}	atomic mass of incident plasma species
n	local atomic density
N_H	number of particle histories in TRIDYN simulation
p	radius of impact
p_{max}	impact parameter.
p_k^{weak}	radius of additional inelastic collsions
$\phi(r)$	'Kr-C' screening function
ϕ_{impl}	implanted particle density distribution
ϕ_{recoil}	target recoil source density distributio
ϕ_0	incident fluence
φ	azimuthal angular location of the impact
(Ψ_r, ϕ_r)	recoil polar and azimuthal deflection angles
(Ψ, ϕ)	particle polar and azimuthal deflection angle in the lab frame
ρ_{proj}	radii of curvature of projectile trajectory at closest approach
ρ_{target}	radii of curvature of recoil trajectory at closest approach
r	separation in
$<R>$	average implanted particle depth
R_c,	distance of closest approach.
R_E	energy reflection coefficient
R_N	reflection coefficient

r_p	random number from 0 to 1.
r_φ	a random number from 0 to 1.
S_{el}	electronic stopping cross-section
t	distance of the asymptotic deflection point from the plane of the target atom before it experiences recoil
T	kinetic energy transferred from incident projectile to lattice atom
θ	deflection angle of the center-of-mass system
V(r)	electrostatic interaction potential
V'(r/a)	spatial derivative of the interaction potential.
(x, y, z)	particle position,
ϒ	sputtering yield
Z_{proj}	charge of incident plasma species
Z_{target}	charge of target material

List of Figures

List of Graphs

Acknowledgements

I would like to acknowledge my thesis committee. Professor Krasheninnikov for presenting me with this opportunity at UCSD. Professor Pigarov for many useful discussions and Professor Beg for useful instruction.

I would also like to thank Roman Smirnov, post-doc in the MAE Department, for helpful conversations as well as my family and friends for their constant and needed encouragement.

Chapter 1 is a summary and synthesis of W. Moller and W. Eckstein. Computer Physics Communication 51 (1988) 355-368[2].

ABSTRACT OF THE THESIS

Implantation, Flux and Recoil Distributions for Plasma Species Impinging on Tokamak Divertor Materials.

by

Nathan David Moshman

Master of Science in Engineering Sciences (Aerospace Engineering)

University of California, San Diego, 2009

Professor Sergei Krasheninnikov, Chair

A time-dependent one-dimensional continuum transport code, WALL-PSI, is being developed by [12] to be a physics component of the integrated core-edge-wall project. WALL-PSI calculates wall

temperature, concentrations of trapped plasma species and other quantities needed to couple tokamak edge plasma to coolant facing components (CFCs). One purpose of developing this code is to help estimate suitability and longevity of divertor plate materials for ITER. For every possible edge plasma species impinging on every potential wall target material combination there will be a unique implanted particle density distribution (ϕ_{impl}), recoil density distribution (ϕ_{recoil}) and scalar flux distribution (J_{scal}) that will change with incident energy and angle. The reflection coefficient (R_N), energy reflection coefficient (R_E), average projectile range <R>, and sputtering yield (Υ) are also needed for integration into WALL-PSI. A binary collision code, TRIDYN, is used to generate the data over an energy range from 0.1eV to 5000eV and normal to tangential incidence. Numeric routines are written to run many cases automatically, synthesize and analyze the data. In some cases analytic functions are fit to the data and various trends are reported. C-functions are written which take, as input: projectile and target species, energy and angle of incidence and depth and return estimations of local implanted particle density, local implanted recoil density, local particle flux, reflection coefficient, energy reflection coefficient and sputtering yield. Trends in data are reported and C-functions are used in WALL-PSI.

Introduction

To simulate the interaction of fast moving projectiles with a solid target it is necessary to use the binary collision approximation, meaning that one projectile at a time is introduced into a solid target and the projectile can only strongly collide with one lattice atom at a time. This is much less computationally expensive than molecular dynamics model and has shown to yield accurate statistics after many particles [1]. One such code widely used for this type of problem is TRIM (Transport of Ions in Matter). This code starts with a 3D grid of target atoms spaced according to their density, known from chemical tables, that also supply other useful properties of the solid such as molecular mass, surface binding energy, displacement energy, heat of dissociation and standard enthalpies. The incident projectile experiences one collision per lattice cell volume. The projectile will then have a change in velocity where the speed reduction is due to energy transferred to the lattice atom and the change in direction follows a Monte Carlo deflection. If the energy transferred to the lattice atom is greater than the displacement energy of the solid target than, by good intuition, the lattice atom should break from its bonds with its neighboring lattice atoms and start to move through the solid much like the original incident projectile. Such a displaced

target atom is called a recoil. When incident fluence or particle energy is substantially large enough, the tracking of recoils will become non-trivial to an accurate solution of the problem. The code TRIM is not able to simulate this regime.

The code TRIDYN is essentially TRIM but includes dynamic changes to the target. While TRIM can account for incident projectile interactions with previously implanted projectiles TRIDYN can track recoil atoms from previous incident projectiles and simulate changes in local density for subsequent projectile tracking [2]. These dynamic changes in density also affect collision statistics locally and overall. The statistics from TRIDYN simulations are synthesized into the quantities of interest.

Chapter 1 Computational Procedure in Tridyn

1.1 Particle Tracking

At any moment during a simulation every moving atom, whether projectile or recoil, is defined by its position (x, y, z), directional angles (α, β, γ) where α is defined as angular displacement from the longitudinal direction x normal to target surface, β and γ are angles measured from the transverse y and z-axes respectively, and its kinetic energy E [2].

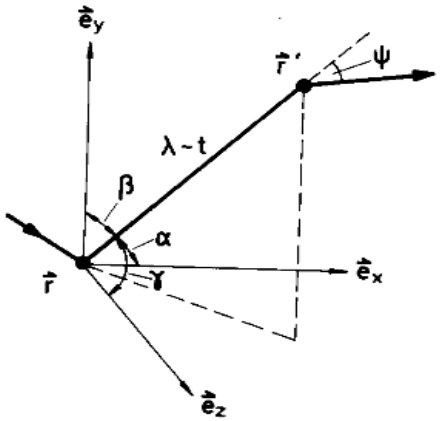


Figure 1.1 Path of a moving atom with two collisions. Its direction is define by the directional angles with respect to the unit vectors e_x, e_y and e_z. By definition e_x points in inward direction normal to the surface

1.2 Hard Collisions

One collision takes place per atomic volume. An atomic volume is defined as a cylinder with length equal to the mean free path $\lambda = n^{1/3}$, where n is the local atomic density which is a dynamic and spatial

variable updated after each projectile history, and radius equal to p_{max} also called impact parameter. The actual radius of impact, p, is determined with a Monte Carlo step where

$$p = p_{max} * sqrt(r_p) \quad (1)$$

where r_p is a random number from 0 to 1. The azimuthal angular location of the impact $\varphi=2\pi r_\varphi$ where r_φ is a random number from 0 to 1.

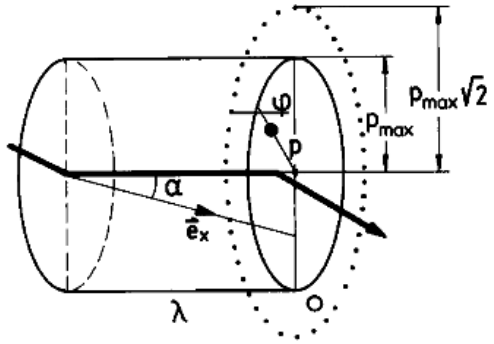


Figure 1.2 Scattering geometry for an elastic two-body interaction with the target atom being initially at rest

The radial distance from the target atom where the impact takes place, R_c, can also be called the distance of closest approach. Its determination is more sophisticated than a Monte Carlo step. Involved is the universal two-body 'Kr-C' interaction potential. This potential is a mean fit to individual inter-atomic potentials for a variety of combinations of scattering pairs. The derivations and details are left to [3],[4],[5],[6].

From a center-of-mass reference frame at the distance of closest approach for a two body collision conservation of energy at R_c yields the following relation [7]

$$1\text{-}V(R_c)/E_c - (p/R_c)^2 = 0 \quad (2)$$

Here E_c is the center of mass energy.

$$E_c = (m_{target}/(m_{proj} + m_{target}))*E \quad (3)$$

and the electrostatic interaction potential

$$V(r)=[(Z_{proj}*Z_{target}*e^2)/(4\pi\varepsilon_0 r)]*\phi(r/a) \quad (4)$$

a is the screening distance [4]

$$a=.8853a_0/(Z_{proj}^{1/2} + Z_{target}^{1/2})^{2/3} a_0 \quad (5)$$

a_0 is the Bohr radius (0.529 Å) and $\phi(R/a)$ is the 'Kr-C' screening function

$$\phi(r/a)=0.191e^{-.279r/a} + 0.474e^{-0.637r/a} + 0.335e^{-1.919r/a} \quad (6)$$

Equation (6) is solved iteratively via Newton's method from R_c.

Now that the location of scattering is known the new position, directional angles and kinetic energy must be determined for the incident particle and the recoil target atom. Let the deflection angle of the center-of-mass system be θ. [8] showed that

$$\cos(\theta/2) = (p/a + \rho + \Delta)/(R_c/a + \rho) \quad (7)$$

where $\rho = \rho_{proj} + \rho_{target}$ where ρ_{proj} and ρ_{target} are the radii of curvature of the trajectories of the projectile and recoiling target atom

respectively at closest approach. These radii can be related to the CM energy, V(R_c/a) and V'(Rc/a) the spatial derivative of the interaction potential. Δ is a correction factor that approaches the value for Rutherford scattering when the reduced energy is large.

Translating back from the CM frame to the lab frame gives the polar deflection angle Ψ

$$\tan(\Psi) = \sin(\theta)/(m_{proj}/m_{target} + \cos(\theta)) \quad (8)$$

The longitudinal distance from the projectile to the original plane of the target atom, t, also the distance of the asymptotic deflection point from the plane of the target atom before it experiences recoil is

$t=R_c*\sin(\theta/2)$

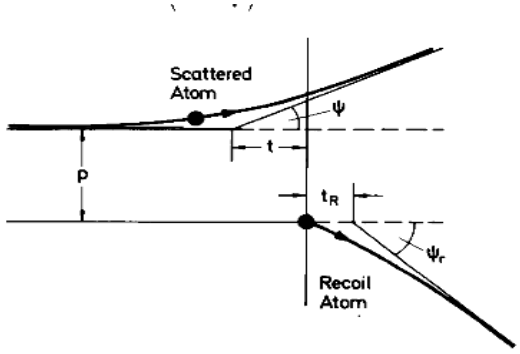


Figure 1.3 Scattering geometry for an elastic two-body interaction with the target atom being initially at rest

Now the new position and direction angles r', α', β', and γ' are [2]:

$$r' = r + (\lambda - t)*(\cos(\alpha), \cos(\beta), \cos(\gamma)) \quad (9)$$

$$\cos(\alpha') = \cos(\Psi)*\cos(\alpha) + \sin(\Psi)*\cos(\phi)*\sin(\alpha) \quad (10)$$

$$\cos(\beta') = \cos(\Psi)*\cos(\beta) - (\sin(\Psi)/\sin(\alpha))*[\cos(\phi)*\cos(\alpha)*\cos(\beta)- \sin(\phi)*\cos(\gamma)] \quad (11)$$

$$\cos(\gamma') = \cos(\Psi)*\cos(\gamma) - (\sin(\Psi)/\sin(\alpha))*[\cos(\phi)*\cos(\alpha)*\cos(\gamma)+\sin(\phi)*\cos(\beta)] \quad (12)$$

1.3 Recoils

Every hard collision generates a recoil target atom if the energy transferred to it via the incident projectile T is greater than the bulk binding E_{bb} energy of the solid specified in TRIDYN's data tables. The initial energy of the recoil is the $E_{b0} = T\text{-}E_{bb}$. It's initial direction is determined based on the directional angles of the incident projectile (α, β, γ) and it's deflection angles (Ψ_r , ϕ_r) defined by $\tan(\Psi_r) = \sin(\theta)/(1\text{-}\cos(\theta))$ and $\phi_r = \pi - \phi$. The recoil will then be tracked much the same way the incident projectile is, experiencing hard elastic collisions and weak inelastic ones and potentially generating more recoils in a cascade.

The kinetic energy transferred from the incident projectile to the lattice atom is given by [9]

$$T=4*(m_{target} + m_{proj})*E*\sin(\theta/2)^2/(m_{target} + m_{proj})^2 \quad (13)$$

1.4 Weak Inelastic Collsions

The new kinetic energy of the projectile

$$E' = E - T - \Delta E_{nl} - \Delta E_{l} \quad (14)$$

where the last two terms are the non-local and local energy loss from weak inelastic collisions that can occur with up to three nearby lattice atoms. These collisions do not affect the projectile's direction, only add to it's energy loss. For the three weak inelastic collisions a new impact parameter

$$p_k^{weak} = p_{max} * sqrt(k + r_p) \quad (15)$$

where k = 1,2,3. Now each p_k^{weak} represents an additional atomic volume in which one collision occurs. A new mean free path is calculated base on p_k^{weak} and used for the non-local energy loss.

$$\Delta E_{nl} = (\lambda - t) n S_{el} \quad (16)$$

where S_{el} is the electronic stopping cross-section calculated in [10]. The formula for local energy loss [11]. The code will simulate collisions until the particle's energy is below that of the cutoff energy specified by the user in the input file.

1.5 Initial Conditions

The TRIDYN code accepts user input of particle energy and angle of incidence. Constant sets of values of energy E_0 angle of incidence α_0 were used. Initially, the particle's start energy will be E_0 plus the surface binding energy of the target defined in TRIDYN's data tables.

1.6 Data of Relevance from TRIDYN

An incident particle has one of two fates. It can end it's motion in the target where it will be considered stopped or implanted. The first four moments of the distribution of implanted particles will determine a Pearson curve up to an amplitude. Alternatively, the projectile can escape from the surface of the solid becoming a back-scattered particle and contributing to R_E and R_N. It can do this without ever entering the target (e.g. low energy high angle of incidence) or it can collide many times with lattice atoms and end up on a path directed out from the surface of the target. Similarly, recoils which are necessarily generated within the target can end their movement inside or outside the target. The former are known as stopped recoils and the latter are back-sputtered recoils. Stopped recoils contribute to the recoil source distribution ϕ_{recoil} back sputtered recoils contribute

to the sputtering yield Υ.

1.7 Dynamic Relaxation and Code Flow

Every projectile or pseudoparticle can be considered an interval of fluence such that $\Delta\phi = \phi_0/N_H$ where ϕ_0 is the incident fluence and N_H is the number of particle histories. When lattice atoms are displaced from their original position and become recoils they leave behind voids in the lattice decreasing the local density. Where particles and recoil atoms terminate their movement increases the local density. In order to accurately account for saturation phenomena a maximum atomic fraction can be defined in the input file. Depth intervals are adjusted in such a manner as to make the local atomic fraction within acceptable levels. The thickness of the depth interval is always kept between .5x and 1.5 x where x is defined by the user based on the number of depth intervals and the target thickness. When intervals become too large they are split in half and the deepest interval is discarded.

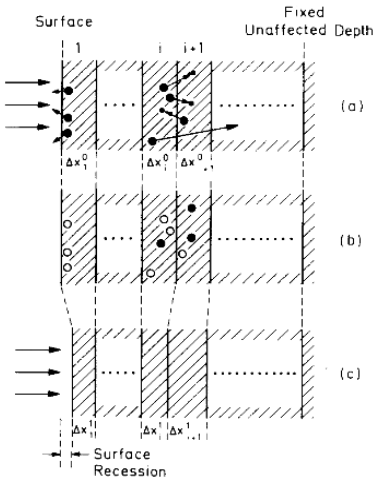


Figure 1.4 Schematic representation of collisional transport and dynamic relaxation: Relocations and sputtering caused by bombardment (a) produce vacancies and additional atoms (b), which are allowed to relax(c)

When an interval is too small it is joined to its neighbor and a new interval is introduced to the simulation space with the atomic fractions of the intervals at the beginning of the simulation. New local atomic densities are then calculated and used for the next particle tracking, specifically to calculate the local mean free path. In this way, TRIDYN is dynamically accounting for the presence of implanted particles, voids, generated recoils and saturation phenomena. An outline of the major loops of the code is shown.

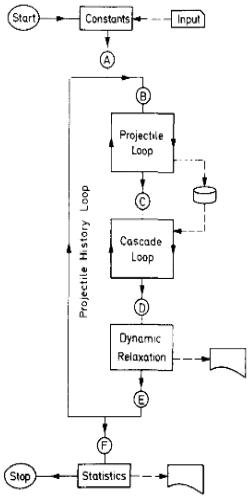


Figure 1.5 Block flow chart of TRIDYN A to F denote the connection points of the different units

Chapter 2 Implementing Tridyn

2.1 Tridyn Output Files of Interest

Recall that particle reflection coefficient (R_N), energy reflection coefficient (R_E), profile of implanted particles ϕ_{impl}, average projectile range $<R_N>$, profile of scalar flux $J_{scal,}$ distribution of recoil atoms ϕ_{recoil}, and sputtering yields (Υ) are needed. TRIDYN has several output files automatically generated for any case run and several longer output files that can be specified to be written or not, based on user input. The automatically generated output file 'output.dat' contains R_N, R_E, $<R_N>$, Υ as well as the three higher moments of the implanted particle distribution: standard deviation, skewness and kurtosis. These three moments combined with the first moment, the mean, make it possible to generate an analytic Pearson curve that matches the shape of the data array for the implanted particle distribution. The output file 'depth_proj.dat' gives the number of particle stops at a given depth. A simple manipulation of this data gives the implanted source distribution ϕ_{impl}. Similarly 'depth_recoil.dat' gives the number recoil stops at a given depth. A simple manipulation of this data gives the stopped recoil source distribution ϕ_{recoil}. The scalar flux is, by far, the most time and memory consuming array to obtain. Originally it was thought that TRIDYN

would generate this data which was false. The next best option was to modify the source code to generate this data. After much effort a stable version of the code with editable source code was not found. The next best option was to use a stable version of the code without source to specify the generation of the output files 'traj_stop_p.dat' and 'traj_back_p.dat' which give the location of every hard collision for every stopped and backscattered particle respectively. This data was then used to generate the scalar flux profiles.

2.2 Running many Tridyn Simulations Automatically

The method used for running TRIDYN cases requires a Windows operating system, Python version 2.4 or 2.5 and the tcl scripting language used to run a TRIDYN.exe executable as well as a stable version of TRIDYN. Both tcl and Python are open source and available to download online.

Several thousand cases were simulated. There are four candidate target materials of potential interest to ITER: Carbon, Beryllium, Tungsten and Lithium. This is due to their thermal and other material properties. There are at least eight different species of plasma that might impinge on the wall of the tokamak either from the fusion processes or from degradation of the wall. These species are

Hydrogen, Deuterium, Tritium, Helium and the four candidate wall elements. The interaction of each different incident species with each different wall material must be simulated by TRIDYN separately. There are 8x4=32 combinations of species. For each combination a range of incident energy from .1 eV to 5000 eV is of interest. This is discretized nearly geometrically, and not linearly, in the following set of 15 energies {0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0 1000.0 2000.0 5000.0} (eV). At each magnitude of energy a range of angle of incidence is of interest. This is discretized in the following set of nine values. {0 15 30 45 55 65 75 80 85} (degrees). Therefore, there are 32x15x9=4320 separate TRIDYN simulations to run. An automatic procedure of some kind is necessary.

A TRIDYN input file [B1] is a short name list declaring values of input variables, some mandatory, others optional. An explanation of those variables is included in the comments of the sample input file. Cases are run within a Python script [B3]. The script starts with an input template [B2], which is identical to the final input file except the values for the projectile species, energy and angle of incidence as well as the target species, target thickness and cutoff energy have string variable names. The Python script seeks out those string names and replaces them with the desired values of each variable in four

imbedded ‘for’ loops. The script runs a TRIDYN simulation with the command Os.system(“tclsh Run.tcl tri.inp”). In this way all 4320 of the unique cases can be run automatically.

The number of particle histories, or pseudoprojectiles, is always 100000. The number of components is always 2, 1 projectile and 1 homogeneous target. The incident fluence is always 1e16/cm^2. The target thickness is determined with trial and error. It was found that for the majority of cases a thickness of 30 Å was acceptable for and energy range of .1eV to 2eV, 60 Å for 5eV to 20eV, 500 Å 50 eV to 200 eV, 1000Å for 500eV to 1000eV and 4000Å for 2000eV to 5000eV. Since Tungsten is so much larger than every other element involved, simulations involving Tungsten usually had to be dealt with separately.

A case of normal incidence was run preliminarily at each energy level and with each combination of target and projectile to determine the necessary target thickness. These cases reveal the maximum depth of projectile at a given energy. If the number of stops in ‘depth_proj.dat’ did not reach zero by the furthest depth bin, the case would have to be run again with a larger target thickness. For some cases the typical target thickness made the cells too thick to resolve the implantation profile and consequently the thickness was reduced.

TRIDYN limits the minimum cell thickness to .1 Angstroms so if the target needed to be thinner than 20 Angstroms, the number of cells was decreased accordingly from 200 until there was a resolvable implantation and flux profile. The maximum number of particle collision histories that can be written to the largest output file, ‘trajec_all.dat’, is 10000. The largest number gives the most accurate profile so this variable is set to 10000 and the number of counts is translated to particles/cm^2 by multiplying by number of histories over incident fluence (10000/1e16). The cutoff energy is the maximum energy a particle can have after a collision to end the collision loop. This always must be set to a value below the bulk binding energy of the material so that the particle or recoil has a chance to move through the material. It was found that for large incident energies, a cutoff energy which was four orders of magnitude smaller than incident energy made the simulations run prohibitively slow. A difference of three orders of magnitude was found to be more appropriate and have a negligible effect on the results. TRIDYN imposes a minimum value allowed for the cutoff energy of .01 eV. Since the case of .1 eV is of interest, the best possible situation is one order of magnitude difference between Ec and E0. EC was held at .01 eV until the case of E0=10eV, and the difference of three orders of

magnitude is maintained for larger values of E0.

After a simulation terminated, the output files of interest were in the same working directory as the Python script which ran the case. If another simulation ran subsequently it would overwrite the output files of interest. To avoid this, the output files were copied to a directory named by for the projectile and target combination. [B3] Additionally the energy level and angle of incidence were appended to the names of the output files so that every single case had a distinguishable set of output files. All output files were stored on an external 500 GB hard drive name LaCie.

Chapter 3 Post-processing Tridyn data

3.1 Scalar Data

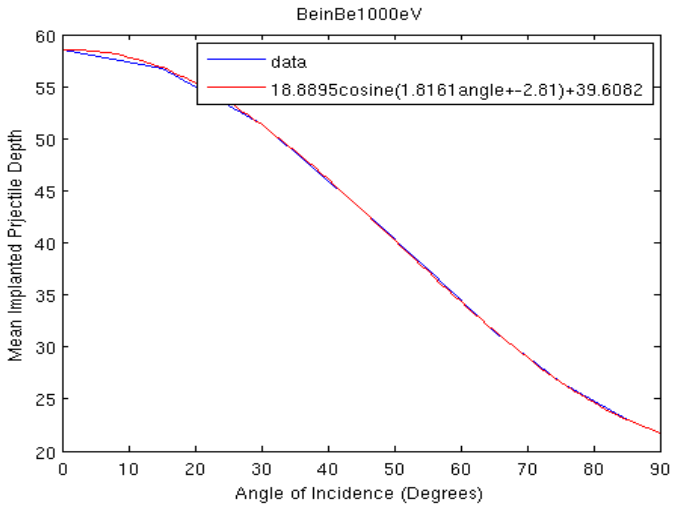
Post-processing was done in Matlab 7.3.1. The m-file ‘postpdata.m’ [A1] reads through the output file ‘output(Energy)(angle).dat’ in a given directory and extracts the mean depth<R>, the reflection coefficient Rn, the energy reflection coefficient Re, and the sputtering yield and writes them in column form to ‘outputdatatable.dat’ within that directory. It does this for every directory, or every combination of projectile and target. The three higher moments of the implanted distribution: standard deviation, skewness and kurtosis are also copied and rewritten. The script ‘columntoblock.m’[A2] reads ‘outputdatatable.dat’ and writes the four quantities of interest to matrix form in a separate data file. This was done to have the data in a more convenient form to copy to a C function.

3.2 Analytic Fits to Scalar Data

‘fitimplanteddata.m’ [A8] is used to demonstrate that, at a given energy level, the quantities of interest follow analytic sinusoid curves according to angle of incidence. Nine data points are read at once from ‘outputdatatable.dat’ corresponding to a quantity of interest’s

(e.g. the mean depth) value at a given energy, for each simulated angle of incidence. The Nelder-Meade optimization routine is implemented and the analytic curve used for fitting has the form A*cos(b*theta+c)+d where theta is the angle of incidence. It was found that a good initial guess for 'A' was .5*(mean(0)-mean(85)), a good guess for 'b' was 2, a good guess for c was 15 degrees, and a good guess for 'd' was .5*(mean(0)-mean(85)).

Graph 3.1 Data and analytic sinusoid fit of mean depth vs. angle of incidence of the case of Beryllium into Beryllium, initial energy 1keV.



3.3 Implanted Particle Distribution and Pearson Curves

'getimplantarray.dat' [A3] reads the output file 'depth_proj(Energy)(angle).dat' and extracts two arrays. The first is an array of positions within the target and the second is the corresponding number of stops at that location. The number of stops

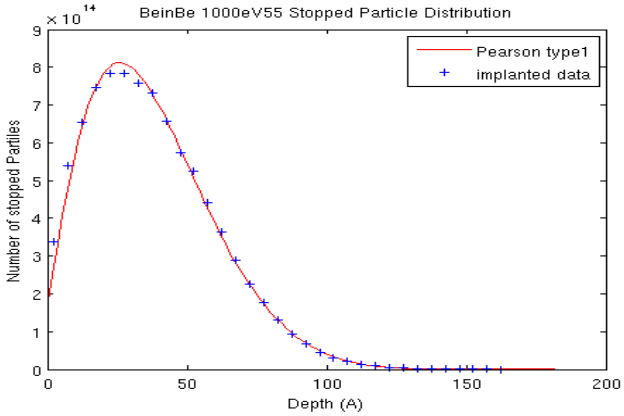
is translated into a density or implanted source array by multiplying the number of stops by the incident flux and dividing by the number of incident particles and the width of one depth bin. In this way the integral of the implanted source distribution over the target equals (1-Rn)*incident flux. These arrays are then written to the file 'SOURCEimplant.txt' for each Energy and angle within the directory.

'Pearsplot_implant.m' [A4] is an m-file used to verify that the analytic Pearson family of curves can accurately match the shape of the implanted source array. Pearson curves are a family of frequency distribution functions that satisfy the differential equation

$$df/dy = (y - a)f(y)/[b_0 + b_1y + b_2y^2] \quad (17)$$

Given the first four moments of the implanted distribution (mean, standard deviation, skewness and kurtosis), an analytic Pearson curve may be defined. It was found that only Pearson type 1 and 2 curves were needed. The amplitude of the curve was yet undefined so a simple function fit was performed using the Nelder-Meade algorithm and an initial guess of the maximum number of stops which showed to work in most cases. The Pearson analytic curve and the actual data array obtained from 'SOURCEimplant.txt' were plotted on the same axes for each case and saved accordingly with a distinguishing name.

Graph 3.2 Implanted particle density distribution vs. depth for the case of Beryllium into Beryllium 1keV, 55° angle of incidence. An analytic Pearson type 1 curve is included.



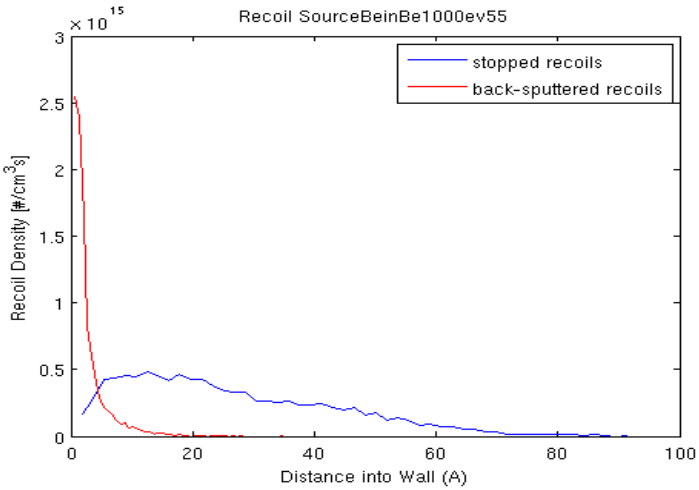
3.4 Recoil Source Distribution

'getrecoilsource.m' [A5] reads the output file 'partic_stop_r.dat' and produces the recoil source array. The output file 'partic_stop_r.dat' lists the starting and ending location of 10000 stopped recoil atoms. The ending locations are read into an array and sorted into a histogram with 100 bins, the width of which are determined by dividing the maximum depth by 100. The histogram number of counts is multiplied by the incident flux and divided by the number of incident particles and the bin width.

This gives the recoil source array in units of cm^-3s^-1. The array is written to a separate data file 'SOURCErecoil.dat' and will eventually be used in a C function.

Graph 3.3 Recoil source density distribution vs. depth for the case of Beryllium into Beryllium 1keV, 55° angle of incidence. Red curve is

sputtered recoils blue curve is implanted recoils.



3.5 Particle Flux Distributions and Analytic Fitting

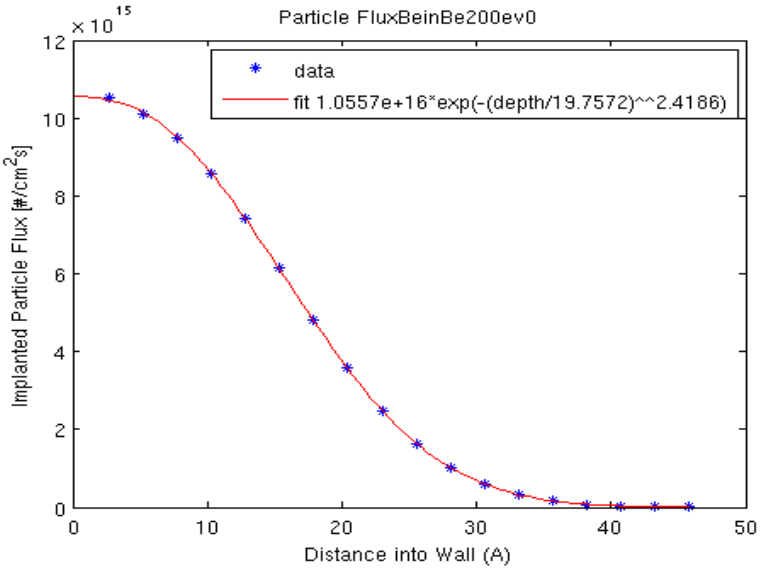
'getflux.m' [A6] takes, by far, the most time to run. This is because the script must analyze the largest output files generated by TRIDYN; 'trajec_back_p.dat' and 'trajec_stop_p.dat'. These two files list the location of every hard collision that a projectile experiences. The former lists all collisions for back-scattered projectiles and the latter all stopped projectiles. The maximum depth of a projectile, which is known from the output file 'depth_proj.dat', is a good approximation for the maximum flux depth. Therefore an array can be defined, which will encompass the entire flux profile, which starts 10 Angstroms above the surface of the target and extends a little past the maximum flux depth. It was shown that this interval would span all collisions. The particle's initial location is read as well as the

subsequent location. If the locations exist in different depth bins of the flux array then the number of crossings stored in each bin between the two locations would be increased by one. This basically counts the number of times a particle passes through the imaginary plane between two bins in the flux array. The actual flux profile is calculated by multiplying the number of passes in each bin by the incident flux and dividing by the number of particle histories. The array is then written to a separate data file ‘fluxarray.dat’ where it will be used later in a C-function. The higher the incident energy, the larger the number of collisions. Consequently, a single output file for a high energy case can be more than 100MB in size. It typically takes one to two hours to analyze that much data and with hundreds of high energy cases it takes weeks to get all the flux arrays.

‘fluxfit.m’ [A7] is used to plot the flux array calculated in ‘getflux.m’ and a fitted analytic function on the same axes. The flux array is read from ‘fluxarray.dat’. The analytic function is of the form A*exp(-(x/L)^p) where A,L and p are fitted parameters and x is the depth into the target. The Nelder-Meade optimization algorithm is used to minimize the sum of the squares of the difference in each data point to the analytic approximation. It was found that a good initial guess for ‘A’ was the value in the first flux bin multiplied by 1.1. A

good initial guess for ‘L’ was one-third of the maximum flux depth and an acceptable initial guess for p was 2.

Graph 3.4 Particle flux distribution vs. depth for the case of Beryllium into Beryllium 200ev, normal incidence. Exponential power analytic fit is included.



Chapter 4 C-Functions for WALLPSI

4.1 Functions for Scalar Data

Seven C-functions were written to be used in the WALLPSI code. Four of the functions take, as input, target, projectile, energy and angle of incidence and return a scalar quantity, either mean (meanfunc.c), reflection coefficient (rcfunc.c), energy reflection coefficient (Ercfunc.c) or sputtering yield (Esput.c). The header file containing 15x9 matrices for each species combination, “Scalars.h”, is included in each of these four functions. Based on the input, the functions look up the table of values associated with each target-projectile combination. The four points in energy-angle space that surround the input are determined and a geometric weighting is applied to interpolate a return value.

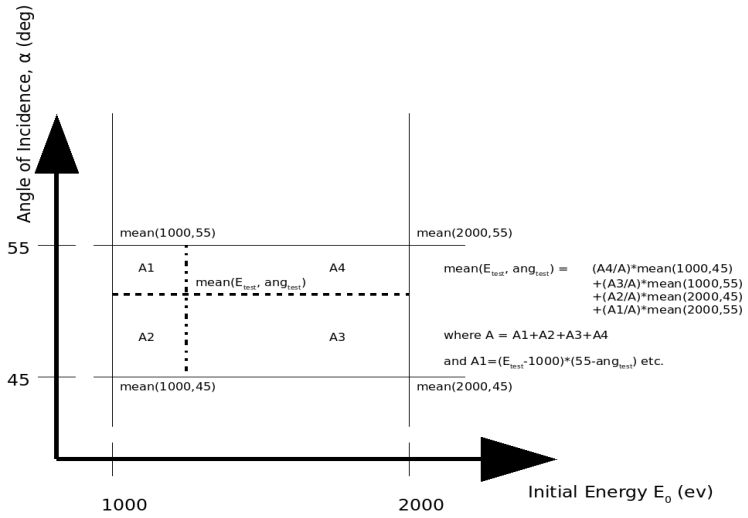


Figure 4.1 An example of how interpolation of mean depth is based on geometric weighting of calculated mean from surrounding points in energy-angle grid.

4.2 Functions for Array Quantities

The other three C-functions to go into WALLPSI take, as input, target, projectile, energy, angle of incidence and depth and return interpolated values of local flux, implanted particle density and recoil source density.

For the local flux (Arrayflux.c), interpolating between four arrays which surround the energy-angle point is awkward because the flux arrays can have very different spatial resolution between energy levels. Instead, the analytic flux curves of the form A*exp(-(x/L)^p) that were calculated in fluxfit.m are used. Include files for each target-projectile combination have the three parameters, A, L, and p calculated for each energy and angle case. No spatial interpolation is necessary since the analytic curve can give a flux value exactly at the depth input. A geometrically weighted sum from the four surrounding flux curves is then computed and returned as the local flux.

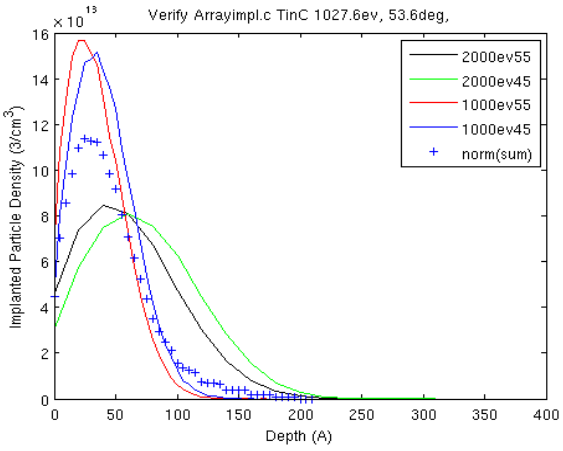
A different interpolation approach was used for the local implanted particle density (Ephrayim) [C2]. Since the integral of the implanted distribution is normalized to (R_N) conserving particle number is necessary. When all four arrays have the same spatial resolution, the arrays are simply summed ϕ_int and normalized by the factor (1-R_{int})*F_0/integral(ϕ_int**)** where R_{int} is the iterpolated reflection

coefficient for the input energy and angle, calculated in rcfunc.c. The local density is determined by linearly interpolating the value of the normalized array's two nearest values.

If the spatial resolution of any of the arrays aren't equal then an additional subroutine is implemented in order to sum the arrays. Define beta as the multiplicative factor that equates the two resolutions. For the coarser array, each of it's values in divided by beta and added to the spatially overlapping values in the finer array. If the finer array's bin overlaps two elements of the coarser array, then a weighted sum is added to the finer array's bin value.

Once the summed array has been generated, it is normalized as described above and the local density is determined by linearly interpolating as before.

Graph 4.1 Implanted particle distributions for the four cases surrounding test point on Energy-angle grid. '+' is the interpolated array based on geometric weighting and normalized by interpolated reflection coefficient.



Arrayrec.c calculates local recoil source density. The way the the recoil source arrays were generated makes every energy-angle case have a different spatial resolution. The disparity between resolution between certain energy levels for the flux arrays was dramatic, up to a factor of 5. The disparity for the recoils is much less, no more than a factor of 2. Therefore, a reasonably similar resolution is shared by nearby arrays. For the four arrays surrounding the point in energy angle space, the array element containing the depth input is calculated from each respective spatial resolution. Then the geometrically weighted sum (based on energy-angle) is returned. The recoils aren’t conserved like the implanted projectiles since there is no way to tell how many recoils will be generated for a given simulation, but the number of incident particles is fixed. Knowing the reflection coefficient then fixes the number of implanted particles.

Chapter 5 Summary and Discussion

A Python Script was written which can be used to run many TRIDYN simulations sequentially. 8 separate Matlab scripts were written that manipulate TRIDYN output to obtain particle flux into wall, trapped recoil source density, implanted particle density with first Four moments and scalar quantities {rc, Erc, Esput}. Analytic function fitting was doe in Matlab with Nelder-Meade algorithm. It was found that the particle flux distribution in the wall follows and exponential power curve. The implanted particle density distribution follows a Pearson Type 1 or 2 curve and the first four moments of the implanted density distribution follow a sinusoidal curve as a function of angle of incidence at constant initial energy.

7 separate C-functions for use in WALLPSI. Four of the C-functions take, as input, incident particle and target species, incident energy and angle and return mean implanted particle depth, reflection coefficient, energy reflection coefficient and sputtering yield. The other three functions take as input incident particle and target species, incident energy and angle and return local implanted particle density, local trapped recoil source density and local particle flux.

Appendix A

A1 postpdata.m

```
clear all;

energy={0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0};
angle={0 15 30 45 55 65 75 80 85};
en=[0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0 1000.0
2000.0 5000.0];
an=[0 15 30 45 55 65 75 80 85];
dirname={'WinC' 'WinBe'};
        %'BeinC' 'BeinBe' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe' 'DinC'
'DinW'...         %not BeinC, NAN errors in data file
        %'HinBe' 'HinC' 'HinW' 'HeinBe' 'HeinC' 'HeinW' 'LiinBe' 'LiinC'
'LiinW'...
        %'TinBe' 'TinC' 'TinW' 'WinC' 'WinBe' 'WinW'};

for k=1:length(dirname)
   filew=['/home/nmoshman/Research/299Research/externaldata/outp
ut/outputdata_implanteddist/LASToutputdatatable' dirname{k}
'12_3_08.txt'];
   fid = fopen(filew,'w');
   fprintf(fid, 'Energy   Angle   mean   standev gamma   beta   nBS
BSE   nrecoil Total_Sput_Yield\n');
   fclose(fid)
   for i=1:length(energy)
      for j=1:length(angle)
         if en(i)>.5
            filename=['/media/LaCie/ALLTridynoutputfiles/' dirname{k}
'/output' num2str(energy{i}) '.0ev' num2str(angle{j}) '.dat'];
         else
            filename=['/media/LaCie/ALLTridynoutputfiles/' dirname{k}
'/output' num2str(energy{i}) 'ev' num2str(angle{j}) '.dat'];
         end

         fid=fopen(filename,'r');
         %read 63 header lines and ignore all numeric data until the
         %string 'AREAL' is reached DIFFERENT
         %LENGTHS FOR DIFFERENT ENERGIES
         C=textscan(fid,'%[^A]','headerLines',63);
```

```
        C=textscan(fid,'%s %s %s %s %s %s %s',1); %areal densities
and fluences
        C=textscan(fid,'%s %f',1);
        C=textscan(fid,'%s %s %s %s %s %s',1);
        C=textscan(fid,'%f %f %f %f %f %f %f',2);
        C=textscan(fid,'%s %s %s %s %s %s %s %s',1);  %energy
losses projectiles
        C=textscan(fid,'%s %s %s %s %s %s',1);
        C=textscan(fid,'%f %f %f %f %f',1);
        C=textscan(fid,'%s %s %s %s %s %s %s %s',1);  %energy
losses recoils
        C=textscan(fid,'%s %s %s %s %s %s',1);
        C=textscan(fid,'%f %f %f %f %f',2);
        C=textscan(fid,'%s %s %s %s %s %s %s %s',1); %Statistics
```

```
C=textscan(fid,'%s %f %7s %f %s',1); np=C{2}; totE=C{4};
        C=textscan(fid,'%s %s %f %7s %f %s',1); nimpl=C{3};
implE=C{5};
        C=textscan(fid,'%s %s %f %7s %f %s',1); nreem=C{3};
reemE=C{5};
        C=textscan(fid,'%s %s %f %7s %f %s',1); nbs(i,j)=C{3};
bsE(i,j)=C{5};
        C=textscan(fid,'%s %s %f %7s %f %s',1); ntran=C{3};
tranE=C{5};
        C=textscan(fid,'%s %s %f ',1); nrecoil(i,j)=C{3};
        C=textscan(fid,'%s %s %s %f %7s %f %s',1); nrec1=C{4};
rec1E=C{6};
        C=textscan(fid,'%s %s %s %f %7s %f %s',1); nrec2=C{4};
rec2E=C{6};
        C=textscan(fid,'%s %s %s %s %s %f %7s %f %s',1);
nrectot=C{6}; rectotE=C{8};

        C=textscan(fid,'%s %s %f ',1);  %reemission particles

        C=textscan(fid,'%s %s %s %s %f %s %s',1);

        C=textscan(fid,'%s %s %s %s %f %s %s',1);

        C=textscan(fid,'%s %s %s %s %s %f %7s %f %s',1);
%allbackgenrecoil

        Ebsput_tot(i,j)=C{8};
        %check
        %C=textscan(fid,'%s %s %11s %s %s %s',1)
%transgenerateрecoil(1),(2)

        C=textscan(fid,'%202c',1);
        %check for both recoil and NON recoil cases



        C=textscan(fid,'%s',1);
        C{1};


        tf=strcmp(C{1},'BACK.SPUTTERED'); %checks for 2by1 back
sputtered text line
```

```
            if (tf==1) %2lines for back sputtered recoils
                C=textscan(fid,'%s %s %s %f %7s %f %s',1);

                Ebsput_21(i,j)=C{6};
                C=textscan(fid,'%s' ,1);
                ttf=strcmp(C{1},'BACK.SPUTTERED'); %checks for 2by2
back sputtered text line
                if (ttf==1)
                    C=textscan(fid,'%s %s %s %f %7s %f %s',1);
                    Ebsput_22(i,j)=C{6};
                    C=textscan(fid,'%991c',1);
                    C{1};
                    %check, at line of distribution data
                end
                if (ttf==0)
                    C=textscan(fid,'%982c',1);
                    C{1};
                end
            end
            if(tf==0)
                C=textscan(fid,'%982c',1);
                C{1};
            end


            C=textscan(fid,'%f %f %f %f %f %f',1);
            mmean(i,j)=C{2};
            stdv(i,j)=C{3};
            gamma(i,j)=C{4};
            beta(i,j)=C{5};
            spread(i,j)=C{6};

            fclose(fid);


            fid = fopen(filew,'a+');
            fprintf(fid, '%f %f %6.5f %6.5f %7.6f %5.5f %d %5.1f %d
%5.1f\n', ...
                en(i), an(j), mmean(i,j), stdv(i,j), gamma(i,j), beta(i,j),
nbs(i,j), bsE(i,j), nrecoil(i,j),Ebsput_tot(i,j));
            fclose(fid);
        end
```

```
    end
end
%now data matrices have energy row indices and angle column
indices
```

A2 columntoblock.m

```
%read data tables and write data to matrix block comma-
delimited format


dirname={'BeinBe' 'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe'
'DinC' 'DinW'...        %not BeinC, NAN errors in data file
        'HinBe' 'HinC' 'HinW' 'HeinBe' 'HeinC' 'HeinW' 'LiinBe' 'LiinC'
'LiinW'...
        'TinBe' 'TinC' 'TinW' 'WinC' 'WinBe' 'WinW'};

for n=1:length(dirname)
   clear C;
   fileread=['/home/nmoshman/Research/299Research/externaldata/o
utput/outputdata_implanteddist/LASToutputdatatable' dirname{n}
'12_3_08.txt'];
   fidr=fopen(fileread,'r');
   filewrite=['/home/nmoshman/Research/299Research/externaldata/o
utput/outputdata_implanteddist/blockmatrix' dirname{n}
'12_26_08.txt'];
   fidw=fopen(filewrite,'a+');
   %read 10 strings, columns headers
   C=textscan(fidr,'%s %s %s %s %s %s %s %s %s %s',1);
   %read data in 9 row chunks, corresponds to 9 angles per energy
level
   C=textscan(fidr,'%f %f %f %f %f %f %f %f %f %f',135); %9angles
times 15 energy levels

   fprintf(fidw,'mean= \n');
   for i=1:15
      fprintf(fidw,'%f, %f, %f, %f, %f, %f, %f, %f, %f, \n', C{3}(9*i-
8:9*i));
   end

   %{
   fprintf(fidw,'\n standev= \n');
   for i=1:15
      fprintf(fidw,'%f, %f, %f, %f, %f, %f, %f, %f, %f, \n', C{4}(9*i-
8:9*i));
   end
```

```
    fprintf(fidw,'\n skewness= \n');
    for i=1:15
        fprintf(fidw,'%f, %f, %f, %f, %f, %f, %f, %f, %f, \n', C{5}(9*i-
8:9*i));
    end

    fprintf(fidw,'\n kurtosis= \n');
    for i=1:15
        fprintf(fidw,'%f, %f, %f, %f, %f, %f, %f, %f, %f, \n', C{6}(9*i-
8:9*i));
    end
    %}

    fprintf(fidw,'\n Reflection Coeff \n');
    for i=1:15
        fprintf(fidw,'%d, %d, %d, %d, %d, %d, %d, %d, %d, \n', C{7}(9*i-
8:9*i)/100000);
    end

    % the quantity is the total energy reflected divided by
    %(number particles*incident energy)
    fprintf(fidw,'\n Energy Reflection Coeff \n');
    for i=1:15
        fprintf(fidw,'%d, %d, %d, %d, %d, %d, %d, %d, %d, \n', (C{8}(9*i-
8:9*i))./(100000*(C{1}(9*i-8:9*i))));
    end

    fprintf(fidw,'\n Total Sputtering Yield \n');
    for i=1:15
        fprintf(fidw,'%d, %d, %d, %d, %d, %d, %d, %d, %d, \n',
C{10}(9*i-8:9*i));
    end

    fclose(fidr);
    fclose(fidw);
end
```

A3 getimplantarray.m

```
clear all;

dirname={ 'BeinBe' 'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe'
'DinC' ...
    'DinW' 'HeinBe' 'HeinC' 'HeinW' 'HinBe' 'HinC' 'HinW' 'LiinBe'
'LiinC'...
    'LiinW' 'TinBe' 'TinC' 'TinW' 'WinBe' 'WinC' 'WinW'};


energy={0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0};
angle={0 15 30 45 55 65 75 80 85};
en=[0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0 1000.0
2000.0 5000.0];
an=[0 15 30 45 55 65 75 80 85];
for n=1:9%length(dirname)

    for i=1:length(energy)
        for j=1:length(angle)


            if energy{i}>.5
                filename=['/media/LaCie/ALLTridynoutputfiles/' dirname{n}
'/depth_proj' num2str(energy{i}) '.0ev' num2str(angle{j}) '.dat'];
            else
                filename=['/media/LaCie/ALLTridynoutputfiles/' dirname{n}
'/depth_proj' num2str(energy{i}) 'ev' num2str(angle{j}) '.dat'];
            end




            fid = fopen(filename,'r');

            C=textscan(fid, '%f %f %f %f %f %f %f %f
%f',1,'headerLines',7);



            k=1;
```

```
x(1)=C{1};

stops(1)=C{2}; %first elements of array
%if (ttf==1)
%   C=textscan(fid, '%f %f %f %f %f %f %f %f %f',1); %1 more
header line in some cases

%end

ttf=isnan(C{1});
%initial index of x_start, want to keep array of the data to find
max and discretize data accordingly
while(ttf==0)

   C=textscan(fid,'%f %f %f %f %f %f %f %f %f',1);

   %ttf=iszero(C{1});
   if (C{1}==0.0), k=k+1; x(k)=C{1}; stops(k)=C{2}; break,
end
   k=k+1;
   x(k)=C{1};
   stops(k)=C{2};



end
%normalize data

%chop off last element in array, is the total
stops=stops(1:length(stops)-1);
x=x(1:length(x)-1);

%turn into source array
dx=x(2)-x(1);
implsrc=((1e16)/(100000*dx))*stops;
%sum(implsrc*dx)
%satisfied integral of implsrd*dx = (nstop/nincident*dx)*flux0
fclose(fid);
%plot(x,implsrc,'+');


x=x(1:k-2);        %chops off last point in array
```

```
            stops=stops(1:k-2);
            filewrite=['/home/nmoshman/Research/299Research/externald
ata/output/' dirname{n} '/SOURCEimplant1_10_08.txt'];
            fidw=fopen(filewrite,'a+');
            fprintf(fidw,'\n %s %f %s %f %s %f \n','energy=',en(i),
'angle=',an(j), 'dx=',...
                2*x(1)); % $ sign will be marker for scanning document later
            for g=1:length(x)
                fprintf(fidw,'%e, ',implsrc(g));
            end

            fclose(fidw);
            %plot(x,stops,'+')

        end
    end
end
```

A4 Pearsplot_implant.m

```
function Pearsplot_implant

%script for plotting implanted distribution with Pearson fit.

%read in data
clear all;
dirname={   'BeinBe' 'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe'
'DinC' ...
   'DinW' 'HeinBe' 'HeinC' 'HeinW' 'HinBe' 'HinC' 'HinW' 'LiinBe'
'LiinC'...
   'LiinW' 'TinBe' 'TinC' 'TinW' 'WinBe' 'WinC' 'WinW'};

en=[0.1 0.2 0.5 1.0 2.0 5.0];% 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0]; %15 energy levels
an=[0 15 30 45 55 65 75 80 85]; % 9 angles

for n=1:length(dirname)
   filename=['/home/nmoshman/Research/299Research/externaldata/o
utput/outputdata_implanteddist/LowEoutputdatatable' dirname{n}
'10_27_08.txt'];
   fid = fopen(filename,'r');
   filename_implarray=['/home/nmoshman/Research/299Research/ext
ernaldata/output/' dirname{n} '/LowEimplantedarray.txt'];
   fidr=fopen(filename_implarray,'r');


   for i=1:length(en);
      for j=1:length(an)
         clear xbin;
         clear stops;
         clear x;
         clear f;
         %%%%%%%%%%%first get the implanted particle data array
         C=textscan(fidr,'%[^l] %4s %[^.] %7c',1);%will always read to
the energy/angle header line

         C=textscan(fidr,'%f %f',1);
         xbin(1)=C{1};
         stops(1)=C{2};
```

```
        tf=isnan(C{1})+isempty(C{1});
        k=2;
        while(tf==0)
            C=textscan(fidr,'%f %f',1);

            tf=isempty(C{1})+isempty(C{1});
            if (tf>0), break, end;

            xbin(k)=C{1};
            stops(k)=C{2};
            k=k+1;

        end
        %stops will have several 0 elements, I want to chop those off
        for g=1:length(stops)
            iszero=isequal(stops(g),0.0);
            if (iszero==1)
                stops=stops(1:g);
                xbin=xbin(1:g);
                break;
            end
        end
        stops=(1e16/100000).*stops;


        %C=textscan(fidr,'%s %f %s %f',1);
        %C{1}
        %C{2}
        %C{3}
        %C{4}


        %%%%%%%%generate Pearson fits type 1 or 2
        %scan the 1 header line 9 columns
        C=textscan(fid, '%s %s %s %s %s %s %s %s
%s',1,'headerLines',1);
        mean=str2double(C{3});
        standev=str2double(C{4});
        gamma=str2double(C{5});
        beta=str2double(C{6});

        %mean=58.50480; standev=27.61400; gamma=0.355410;
```

```
beta=2.81706;

        CA=1/(2*(5*beta-6*gamma*gamma-9));

        b0=-1*standev*standev*(4*beta-3*gamma*gamma)*CA;
        b1=-1*gamma*standev*(beta+3)*CA;
        b2=-1*(2*beta-3*gamma*gamma-6)*CA;

        K=1; %just to have it set for now
        xspace=max(xbin);%max(xbin);
        %x=0:xspace/100:xspace;


        %test for Pearson type 1
        if
(gamma~=0.0)&&(gamma^2+1)<=beta<(3+1.5*gamma^2)

            type=1;
            A1=.5*((b1/b2)+((b1/b2)^2-(4*b0/b2))^.5);
            A2=-.5*((b1/b2)-((b1/b2)^2-(4*b0/b2))^.5);


            m1=(A1+b1)/(b2*(A1+A2));
            m2=(A2-b1)/(b2*(A1+A2));
            %(m1+1)*A2
            %(m2+1)*A1 these must be equal
            x=0:(A2+mean)/100:(A2+mean);
            f=K*((x-mean+A1).^m1).*(A2+mean-x).^m2;


        elseif (gamma==0.0)&&(1<=beta<=3)
            %disp(en(i),an(j),'Pearson type 2')
            type=2;
            A=sqrt(-1*b0/b2);
            m=1/(2*b2);
            x=0:(mean+A)/100:(mean+A);
            f=K*(A^2-(x-mean).^2).^m;


        elseif (gamma~=0.0)&&(beta==(3+1.5*gamma^2))
            disp('Pearson type 3')
            type=3;
```

```
        elseif
(0.0<=gamma^2<=32.0)&&(beta>(39*gamma^2+48+6*(gamma^2
+4)^1.5)/(32-gamma^2))
            disp('Pearson type 4')
            type=4;

        elseif
(0.0<=gamma^2<=32.0)&&(beta==(39*gamma^2+48+6*(gamma^
2+4)^1.5)/(32-gamma^2))
            disp('Pearson type 5')
            type=5;

        elseif
(gamma~=0.0)&&((3+1.5*gamma^2)<beta<(39*gamma^2+48+6*(g
amma^2+4)^1.5)/(32-gamma^2))
            disp('Pearson type 6')
            type=6;
        elseif (gamma==0.0)&&(beta>3.0)
            disp('Pearson type 7')
            type=7;

        elseif (gamma~=0.0)&&(beta<((6/5)*gamma^2+(9/5)))
            disp('Pearson type 8')
            type=8;
        else
            disp('NONE OF THESE')
            type=9;
        end

        %fminsearch fit the amplitude for the Pearson dist

        options=optimset('Display','off');
        if (en(i)>501.0)
            Starting=[max(stops)/1e20];
        else
            Starting=[max(stops)];
        end

        if (type==1)
            parameter_hat=fminsearch(@mycurve1, Starting, options,
xbin, stops);
            fitted=parameter_hat(1).*((x-mean+A1).^m1).*(A2+mean-
```

```
x).^m2;
        end

        if (type==2)
            parameter_hat=fminsearch(@mycurve2, Starting, options,
xbin, stops);
            fitted=parameter_hat(1).*(A^2-(x-mean).^2).^m;
        end


        fname=['/home/nmoshman/Research/299Research/externaldat
a/output/' dirname{n} '/impldatawfit' num2str(en(i)) 'ev'
num2str(an(j)) '.fig'];
        plot(x,fitted,'r'); %for demo purposes
        hold on;
        %normstops=stops/sum(stops);
        plot(xbin,stops,'+');
        string1=['Pearson type' num2str(type)];
        legend(string1,'implanted data');
        title([dirname{n} ' ' num2str(en(i)) 'eV' num2str(an(j)) '
Stopped Particle Distribution']);
        xlabel('Depth (A)');
        ylabel('Number of stopped Partiles');

        %%Create figure for mobile species


        hgsave(fname);
        hold off;
      end
   end
   fclose(fid);
   fclose(fidr);
end

function sse=mycurve1(parameter,xbin,output)
Fitted_Curve=parameter(1).*((xbin-mean+A1).^m1).*(A2+mean-
xbin).^m2;
output=stops;
error=Fitted_Curve-stops;
sse=sum(error.^2);
end
```

```
function sse=mycurve2(parameter,xbin,output)
Fitted_Curve=parameter(1).*(A^2-(xbin-mean).^2).^m;
output=stops;
error=Fitted_Curve-stops;
sse=sum(error.^2);
end

end
```

A5 getrecoilsource.m

```
clear all;
dirname={   %'BeinBe' };
   'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe' 'DinC' ...
   'DinW' 'HeinBe' 'HeinC' 'HeinW' 'HinBe' 'HinC' 'HinW' 'LiinBe'
'LiinC'...
   'LiinW' 'TinBe' 'TinC'...
   'TinW' 'WinBe' 'WinC' 'WinW'};


energy={0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0};
angle={0 15 30 45 55 65 75 80 85};
en=[0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0 1000.0
2000.0 5000.0];
an=[0 15 30 45 55 65 75 80 85];
for n=1:length(dirname)

   for i=1:length(energy)
      for j=1:length(angle)


         if energy{i}>.5
            filename_stop=['/media/LaCie/ALLTridynoutputfiles/'
dirname{n} '/partic_stop_r' num2str(energy{i}) '.0ev'
num2str(angle{j}) '.dat'];
         else
            filename_stop=['/media/LaCie/ALLTridynoutputfiles/'
dirname{n} '/partic_stop_r' num2str(energy{i}) 'ev'
num2str(angle{j}) '.dat'];
```

```
        end
        %fid = fopen('testdoc.dat','r');
        fid = fopen(filename_stop,'r');

        %scan the 5 header lines and column headers
        C=textscan(fid, '%s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s',1,'headerLines',5); %20 column headings

        %initialize recoil source array and sputtering yield to zero

        sputtering_yield=0;
        ttf=strcmp(C{1},'ende');
        k=1; %initial index of x_start, want to keep array of the data to
find max and discretize data accordingly
        while(ttf==0)

            %C=textscan(fid, '%f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f',1);
            C=textscan(fid,'%s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s',1);

            ttf=strcmp(C{1},'ende');
            if (ttf==1), break, end

            %recoil_energy=str2double(C{4});
            %start_loc(k)=str2double(C{5});
            end_loc(k) = str2double(C{8});
            k=k+1;


            %ithink this line is just wrong 12_25
            %sputtering_yield=sputtering_yield+recoil_energy;

        end
        fclose(fid);
        dx=max(end_loc)/100;
        edges=0:dx:max(end_loc);
        %translate data to recoil source array in /cm3s
        rec_source=histc(end_loc,edges)*1e16/(100000*dx);

        %plot(edges,rec_source);
```

```
            filewrite=['/home/nmoshman/Research/299Research/externald
ata/output/' dirname{n} '/SOURCErecooil12_25_08.txt'];
            fidw=fopen(filewrite,'a+');
            fprintf(fidw,'%s %f %s %f \n','energy=',en(i), 'angle=',an(j) );
            for g=1:length(edges)
                fprintf(fidw,'%f %e\n',edges(g),rec_source(g));
            end

            fclose(fidw);



            %check this recoil array agains the one obtained from
            %depth_recoil.dat


        end %iterates angle

    end %iterates energy

end %iterates directory name
```

A6 getflux.m

```
%clear all; clc;

%script for generating a flux array from particle history data
%for every elastic collision x y z and time are given

clear all;
energy={0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0};
angle={0 15 30 45 55 65 75 80 85};
en=[0.1 0.2 0.5 1.0 2.0 5.0 10.0 20.0 50.0 100.0 200.0 500.0 1000.0
2000.0 5000.0];
an=[0 15 30 45 55 65 75 80 85];

dirname={   'BeinLi' }; % as of 1_7 need to still do BeinLi 5000ev45-
85




for k=1:length(dirname)
   for i=15:15%length(energy)
      for j=4:9%1:length(angle)
         if energy{i}>.5
            filename1=['/media/LaCie/ALLTridynoutputfiles/'
dirname{k} '/trajec_back_p' num2str(energy{i}) '.0ev'
num2str(angle{j}) '.dat']
            filename2=['/media/LaCie/ALLTridynoutputfiles/'
dirname{k} '/trajec_stop_p' num2str(energy{i}) '.0ev'
num2str(angle{j}) '.dat'];
            %filename1=['/home/nmoshman/Research/299Research/ext
ernaldata/BeinBecase/trajec_back_p' num2str(energy{i}) '.0ev'
num2str(angle{j}) '.dat']
            %filename2=['/home/nmoshman/Research/299Research/ext
ernaldata/BeinBecase/trajec_stop_p' num2str(energy{i}) '.0ev'
num2str(angle{j}) '.dat'];
         else
            %filename1=['/home/nmoshman/Research/299Research/ext
ernaldata/BeinBecase/trajec_back_p' num2str(energy{i}) 'ev'
num2str(angle{j}) '.dat']
            %filename2=['/home/nmoshman/Research/299Research/ext
```

```
ernaldata/BeinBecase/trajec_stop_p' num2str(energy{i}) 'ev'
num2str(angle{j}) '.dat'];
            filename1=['/media/LaCie/ALLTridynoutputfiles/'
dirname{k} '/trajec_back_p' num2str(energy{i}) 'ev'
num2str(angle{j}) '.dat']
            filename2=['/media/LaCie/ALLTridynoutputfiles/'
dirname{k} '/trajec_stop_p' num2str(energy{i}) 'ev'
num2str(angle{j}) '.dat'];

        end

        time=clock
        %to track the time it takes for runs

        if en(i)<4.0
            tt=60.0;
        end

        if en(i)>4.0 && en(i) <25.0
            tt=60.0;
        end

        if en(i)<250.0 && en(i)>25.0
            tt=500.0;
            %tt=50.0
        end
        if en(i)>250.0 && en(i)<1500.0
            tt=1000.0;
            %tt=100.0
        end
        if en(i)>1500.0
            tt=4000.0;
            %tt=200.0
        end
        % particles get tracked from x= -3.39A
        dx=(tt+10.0)/201;
        xbin=-10.0:dx:tt;

        particlesb=0; %counter can be displayed on command window
to monitor progress
        particless=0;
        xold= -3.3990626;
```

```
oldindex=1;
sflux(1:1:202)=0;
bflux(1:1:202)=0;

fid=fopen(filename1,'r');
%scan the 7 header lines
C=textscan(fid, '%s %s %s %s' , 1);
C=textscan(fid, '%s %s %s %s %s',1);
C=textscan(fid, '%f %s %s %s' , 1);
C=textscan(fid, '%s %s %s %s %s %s %s %s' , 1);
C=textscan(fid, '%f %f %f %f %f %f %f %f' , 1);
C=textscan(fid, '%s %s %s %s %s' , 1);

%a condition which is true returns 1 like isstrprop

C=textscan(fid, '%s %s %s %s %s' , 1);
%break condition if C contains strings from first line in file
ttf=strcmp(C{1},'ende');

while(ttf==0)

    C=textscan(fid, '%s %s %s %s %s' , 1);
    ttf=strcmp(C{1},'ende');
    if (ttf==1), break, end

    for z=1:5
        D{z}=str2double(C{z});
    end
    tf=isempty('D{1}');
    ft=isnan(D{1});
    ft=sum(ft);
    %if statement which tests for string or number types
    %means line is a string not data
    if ((tf==1) || (ft>.5))
        %read the rest of the header line, should be 3 more
        %strings
        D=textscan(fid, '%s %s %s',1);
        %read 8 floats in line just below string headers
        D=textscan(fid, '%f %f %f %f %f %f %f %f' , 1);

        D=textscan(fid, '%s %s %s %s %s' , 1);
        %read first line of new particle data and continue loop
```

```
        D=textscan(fid, '%f %f %f %f %f' , 1);

        particlesb=particlesb+1;
    end
    xnew=D{2};

    newindex=((xnew+10.0)/dx)+1;
    newindex=floor(newindex);

    if xnew>xold
        for zz=(oldindex+1):1:newindex
            bflux(zz)=bflux(zz)+1;
        end
    else
        for zz=(oldindex+1):1:newindex
            bflux(zz)=bflux(zz)+1;
        end

    end
    xold=xnew;
    oldindex=newindex;

end



fclose(fid);


fid=fopen(filename2,'r');
%scan the 7 header lines
C=textscan(fid, '%s %s %s %s' , 1);
C=textscan(fid, '%s %s %s %s %s',1);
C=textscan(fid, '%f %s %s %s' , 1);
C=textscan(fid, '%s %s %s %s %s %s %s %s' , 1);
C=textscan(fid, '%f %f %f %f %f %f %f %f' , 1);
C=textscan(fid, '%s %s %s %s %s' , 1);

%a condition which is true returns 1 like isstrprop

C=textscan(fid, '%s %s %s %s %s' , 1);
```

```
%break condition if C contains strings from first line in file
ttf=strcmp(C{1},'ende');

while(ttf==0)

    C=textscan(fid, '%s %s %s %s %s' , 1);
    ttf=strcmp(C{1},'ende');
    if (ttf==1), break, end

    for zzz=1:5
        D{zzz}=str2double(C{zzz});
    end
    tf=isempty('D{1}');
    ft=isnan(D{1});
    ft=sum(ft);
    %if statement which tests for string or number types
    %means line is a string not data
    if ((tf==1) || (ft>.5))
        %read the rest of the header line, should be 3 more
        %strings
        D=textscan(fid, '%s %s %s',1);
        %read 8 floats in line just below string headers
        D=textscan(fid, '%f %f %f %f %f %f %f %f' , 1);

        D=textscan(fid, '%s %s %s %s %s' , 1);
        %read first line of new particle data and continue loop
        D=textscan(fid, '%f %f %f %f %f' , 1);

        particless=particless+1;

    end
    xnew=D{2};


    newindex=((xnew+10.0)/dx)+1;
    newindex=floor(newindex);

    if xnew>xold
        for yy=(oldindex+1):1:newindex
            sflux(yy)=sflux(yy)+1;
        end
    else
```

```
                for yy=(oldindex+1):1:newindex
                    sflux(yy)=sflux(yy)+1;
                end

            end
            xold=xnew;
            oldindex=newindex;

        end
        fclose(fid);

        totpart=particless+particlesb; %total number of particle
histories
        totflux=(1e16/totpart)*(sflux+bflux);  %pseudoparticle
conversion
        %sflux=(1e16/10000)*sflux;
        %bflux=(1e16/particlesb+1)*bflux;
         plot(xbin,totflux);
        %hold on;
        %plot(xbin,sflux,'r');
        %plot(xbin,bflux,'k');

        %legend('total flux', 'stopped flux', 'back scattered flux');
        title(['Particle Flux' dirname{k} num2str(energy{i}) 'ev'
num2str(angle{j})]);
        fname=['/home/nmoshman/Research/299Research/externaldat
a/output/' dirname{k} '/' num2str(energy{i}) 'ev' num2str(angle{j})
'.fig'];
        xlabel('Distance into Wall (A)');
        ylabel('Implanted Particle Flux [#/cm^2s]');
        hgsave(fname);

        fidw=fopen(['/home/nmoshman/Research/299Research/externa
ldata/output/' dirname{k} '/fluxarrays12_8_08.txt'],'a+');
        fprintf(fidw, '\n %5.1f %2.1f %5.5f \n', en(i), an(j), dx);
        dlmwrite(['/home/nmoshman/Research/299Research/externald
ata/output/' dirname{k} '/fluxarrays12_8_08.txt'], totflux, 'precision',
'%.6e', '-append');
        fclose(fidw);
      end
    end
end
```

A7 fluxfit.m

```
function fluxfit


%read in flux array

clear all;
energy={0.1 0.2 0.5 1.0 2.0 5.0};% 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0};
angle={0 15 30 45 55 65 75 80 85};
en=[0.1 0.2 0.5 1.0 2.0 5.0];% 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0];
an=[0 15 30 45 55 65 75 80 85];



dirname={ 'BeinBe' 'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe'
'DinC' ...
    'DinW' 'HeinBe' 'HeinC' 'HeinW' 'HinBe' 'HinC' 'HinW' 'LiinBe'
'LiinC'...
    'LiinW' 'TinBe' 'TinC' 'TinW' 'WinBe' 'WinC' 'WinW' };

for k=1:length(dirname)
    fid=fopen(['/home/nmoshman/Research/299Research/externaldata/
output/' dirname{k} '/fluxarrays12_8_08.txt'],'r');
end

C=textscan(fid, '%f %f', 1);

for i=1:length(en)  % loops energy levels
    C=textscan(fid, '%f', 201, 'delimiter',',');
    for s=1:201
        fluxar{s}=C{1}(s);
        fluxarr(s)=fluxar{s};
    end

    iter=[100,.5,50];  %need good initial guess


    if en(i)<25.0
        tt=60.0;
```

```
    end

    if (en(i)<250.0 && en(i)>25.0)
        tt=500.0;
        %tt=50.0
    end
    if (en(i)>250.0 && en(i)<1500.0)
        tt=1000.0;
        %tt=100.0
    end
    if en(i)>1500.0
        tt=4000.0;
        %tt=200.0
    end
    dx=tt/200;
    x=0:dx:tt;

    [tf locmax] = ismember(max(fluxarr),fluxarr);
    loczero=length(nonzeros(fluxarr))  %number of non-zero elements
of flux array



    %fluxarr=fluxarr(locmax:loczero); %flux array is chopped to span
its max to its first zero
    %x=x(locmax:loczero);
    Starting=[1.1*fluxarr(1),(loczero*dx)/3, 2];



    %%%%all in agreement up to here
    options=optimset('Display','iter');
    parameter_hat=fminsearch(@mycurve, Starting, options, x, fluxarr)
    fitted=parameter_hat(1).*exp(-
(x./parameter_hat(2).^parameter_hat(3)));


    plot(x,fluxarr,'b');
    hold on
    plot(x,fitted,'r');
    title(['Particle Flux' dirname{k} num2str(energy{i}) 'ev'
num2str(angle{j})]);
```

```
    fname=['/home/nmoshman/Research/299Research/externaldata/out
put/' dirname{k} '/' num2str(energy{i}) 'ev' num2str(angle{j}) '.fig'];
    xlabel('Distance into Wall (A)');
    ylabel('Implanted Particle Flux [#/cm^2s]');
    hgsave(fname);



end

fclose(fid);

function sse=mycurve(parameter,input,output)
Fitted_Curve=parameter(1)*exp(-
1*(input./parameter(2)).^parameter(3));
output=fluxarr;
error=Fitted_Curve-output;
sse=sum(error.^2);
end

end
```

A8 fitimplanteddata.m

```
function fitimplanteddata
%script for reading outputdatatable(directory name)

%read in data
clear all;
dirname={ 'BeinBe' 'BeinC' 'BeinW' 'CinBe' 'CinC' 'CinW' 'DinBe'
'DinC' ...
   'DinW' 'HeinBe' 'HeinC' 'HeinW' 'HinBe' 'HinC' 'HinW' 'LiinBe'
'LiinC'...
   'LiinW' 'TinBe' 'TinC' 'TinW' 'WinBe' 'WinC' 'WinW'};

en=[0.1 0.2 0.5 1.0 2.0 5.0];% 10.0 20.0 50.0 100.0 200.0 500.0
1000.0 2000.0 5000.0]; %15 energy levels
an=[0 15 30 45 55 65 75 80 85]; % 9 angles



for n=1:length(dirname)
   filename=['/home/nmoshman/Research/299Research/externaldata/o
utput/outputdata_implanteddist/LowEoutputdatatable' dirname{n}
'10_27_08.txt'];
   %filenamewrite=['/home/nmoshman/Research/299Research/externa
ldata/output/' dirname{n} '/LowEmeanfitdata.txt'];
   %filenamewrite=['/home/nmoshman/Research/299Research/externa
ldata/output/' dirname{n} '/LowEstandevfitdata.txt'];
   %filenamewrite=['/home/nmoshman/Research/299Research/externa
ldata/output/' dirname{n} '/LowEgammafitdata.txt'];
   filenamewrite=['/home/nmoshman/Research/299Research/external
data/output/' dirname{n} '/LowEbetafitdata.txt'];

   fid=fopen(filename,'r');
   mdt=fopen(filenamewrite,'a+');
   C=textscan(fid,'%s %s %s %s %s %s %s %s %s %s',1); %read
headers
   for i=1:length(en)
      for j=1:length(an)
         %reads in the
         C=textscan(fid,'%f %f %f %f %f %f %f %f %f %f',1);
         mean(j)=C{3};
         sd(j)=C{4};
```

```
            gamma(j)=C{5};
            beta(j)=C{6};


        end
        %fname=['/home/nmoshman/Research/299Research/externaldata
/output/' dirname{n} '/meanvanglefit' num2str(en(i)) 'ev'
num2str(an(j)) '.fig'];
        %fname=['/home/nmoshman/Research/299Research/externaldata
/output/' dirname{n} '/standevvanglefit' num2str(en(i)) 'ev'
num2str(an(j)) '.fig'];
        %fname=['/home/nmoshman/Research/299Research/externaldata
/output/' dirname{n} '/skewnessvanglefit' num2str(en(i)) 'ev'
num2str(an(j)) '.fig'];
        fname=['/home/nmoshman/Research/299Research/externaldata/o
utput/' dirname{n} '/kurtosisvanglefit' num2str(en(i)) 'ev'
num2str(an(j)) '.fig'];

        %function fitting code

        angp=0:1:90;
        options=optimset('Display','off');

        %{

        %do it for mean
        plot(an(1:9),mean(1:9));
        hold on;
        Starting=[.5*(mean(1)-mean(9)) 2 15 .5*(mean(1)+mean(9))];
        parameter_hat=fminsearch(@mymeancurve, Starting, options,
an, mean);
        fitted=parameter_hat(1).*cosd(parameter_hat(2)*angp +
parameter_hat(3))+parameter_hat(4);

        plot(angp,fitted,'r');
        A=num2str(parameter_hat(1));
        b=num2str(parameter_hat(2));
        phase=num2str(parameter_hat(3));
        level=num2str(parameter_hat(4));
        fitstring=[ A 'cosine(' b 'angle+' phase ')+' level];
        legend('data',fitstring);
```

```
        fname=['/home/nmoshman/Research/299Research/externaldata/o
utput/' dirname{n} '/meanvanglefit' num2str(en(i)) 'eV.fig'];
        xlabel('Angle of Incidence (Degrees)');
        ylabel('Mean Implanted Prjectile Depth');
        title([dirname{n} num2str(en(i)) 'eV']);

        hgsave(fname);
        %write mean data to data file
        fprintf(mdt,'%f %f %f %f \n', parameter_hat(1), parameter_hat(2),
parameter_hat(3), parameter_hat(4));

        %}

        %{

        %do it for standev
        plot(an(1:9),sd(1:9)); %looks like cosine
        hold on;
        Starting=[.5*(sd(9)-sd(1)) .8 80 .5*(sd(1)+sd(9))];
        parameter_hat=fminsearch(@mysdcurve, Starting, options, an,
sd);
        fitted=parameter_hat(1).*cosd(parameter_hat(2)*angp +
parameter_hat(3))+parameter_hat(4);

        plot(angp,fitted,'r');
        A=num2str(parameter_hat(1));
        b=num2str(parameter_hat(2));
        phase=num2str(parameter_hat(3));
        level=num2str(parameter_hat(4));
        fitstring=[ A 'cosine(' b 'angle+' phase ')+' level];
        legend('data',fitstring);

        fname=['/home/nmoshman/Research/299Research/externaldata/o
utput/' dirname{n} '/standevvanglefit' num2str(en(i)) 'eV.fig'];
        xlabel('Angle of Incidence (Degrees)');
        ylabel('Stand Dev of Implanted Prjectile Depth');
        title([dirname{n} num2str(en(i)) 'eV']);

        hgsave(fname);
        %write stan dev data to data file
        fprintf(mdt,'%f %f %f %f \n', parameter_hat(1), parameter_hat(2),
parameter_hat(3), parameter_hat(4));
```

```
        %}

        %{
        %do it for gamma
        plot(an(1:9),gamma(1:9));   %looks like -cosine 1.1 to 1.45
        hold on;
        Starting=[.5*(gamma(1)-gamma(9)) 1.3 55
.5*(gamma(1)+gamma(9))];
        parameter_hat=fminsearch(@mygammacurve, Starting, options,
an, gamma);
        fitted=parameter_hat(1).*cosd(parameter_hat(2)*angp +
parameter_hat(3))+parameter_hat(4);

        plot(angp,fitted,'r');
        A=num2str(parameter_hat(1));
        b=num2str(parameter_hat(2));
        phase=num2str(parameter_hat(3));
        level=num2str(parameter_hat(4));
        fitstring=[ A 'cosine(' b 'angle+' phase ')+' level];
        legend('data',fitstring, 'Location', 'NorthWest');

        fname=['/home/nmoshman/Research/299Research/externaldata/o
utput/' dirname{n} '/gammavanglefit' num2str(en(i)) 'eV.fig'];
        xlabel('Angle of Incidence (Degrees)');
        ylabel('Skewness of Implanted Prjectile Depth');
        title([dirname{n} num2str(en(i)) 'eV']);

        hgsave(fname);
        %write gamma data to data file
        fprintf(mdt,'%f %f %f %f \n', parameter_hat(1), parameter_hat(2),
parameter_hat(3), parameter_hat(4));

        %}



        %do it for beta
        plot(an(1:9),beta(1:9));   %looks like -cosine 1.1 to 1.45
        hold on;
        Starting=[.5*(beta(1)-beta(9)) 1.45 35 .5*(beta(1)+beta(9))];
        parameter_hat=fminsearch(@mybetacurve, Starting, options, an,
```

```
beta);
        fitted=parameter_hat(1).*cosd(parameter_hat(2)*angp +
parameter_hat(3))+parameter_hat(4);

        plot(angp,fitted,'r');
        A=num2str(parameter_hat(1));
        b=num2str(parameter_hat(2));
        phase=num2str(parameter_hat(3));
        level=num2str(parameter_hat(4));
        fitstring=[ A 'cosine(' b 'angle+' phase ')+' level];
        legend('data',fitstring,'Location','Northeast');

        fname=['/home/nmoshman/Research/299Research/externaldata/o
utput/' dirname{n} '/betavanglefit' num2str(en(i)) 'eV.fig'];
        xlabel('Angle of Incidence (Degrees)');
        ylabel('Kurtosis of Implanted Prjectile Depth');
        title([dirname{n} num2str(en(i)) 'eV']);

        hgsave(fname);
        %write beta data to data file
        fprintf(mdt,'%f %f %f %f \n', parameter_hat(1), parameter_hat(2),
parameter_hat(3), parameter_hat(4));









        hold off;



    end

    fclose(fid);
    fclose(mdt);
end


function sse=mymeancurve(parameter,angp,output)

Fitted_Curve=parameter(1).*cosd(parameter(2)*angp +
```

```
parameter(3))+parameter(4);
output=mean;
error=Fitted_Curve-output;
sse=sum(error.^2);
end

function sse=mysdcurve(parameter,angp,output)

Fitted_Curve=parameter(1).*cosd(parameter(2)*angp +
parameter(3))+parameter(4);
output=sd;
error=Fitted_Curve-output;
sse=sum(error.^2);
end

function sse=mybetacurve(parameter,angp,output)

Fitted_Curve=parameter(1).*cosd(parameter(2)*angp +
parameter(3))+parameter(4);
output=beta;
error=Fitted_Curve-output;
sse=sum(error.^2);
end

function sse=mygammacurve(parameter,angp,output)

Fitted_Curve=parameter(1).*cosd(parameter(2)*angp +
parameter(3))+parameter(4);
output=gamma;
error=Fitted_Curve-output;
sse=sum(error.^2);
end

end
```

Appendix B

B1 example of a TRIDYN input file

```
 10 eV D -> C
 &TRI_INP
   case_e0 = 0
   e0     =  0.1, 0.00
   ncp    =  2,
   symbol = He, Li
   nh =    100000,
   idrel = 1,
   flc   = 1.00
   sfin  = 0,
   ipot  = 1,
   isbv  = 3,
   imcp =0,
   ttarget =  30.0
   nqx     =  200.0,
   e_cutoff= 0.01,   0.01,
   qu      = 0.0 ,   1.0,
   case_alpha=0
   alpha0 = 15  , 0.000
   qubeam = 1.000, 0.000
   qumax  = 1.00, 1.000
   ioutput_hist = 10000,10000,10000,10000,10000,10000
   ioutput_part = 10000,10000,10000,10000,10000,10000
   lparticle_p = .true.
   lparticle_r = .true.
   lmatrices = .false.
   ltraj_p = .true.
/
```

B2 template for TRIDYN input file

```
 10 eV D -> C
 &TRI_INP
   case_e0 = 0
   e0     =  ENERGY, 0.00
%'ENERGY' is replaced by numeric value
   ncp    =  2,
   symbol = PROJECTILE, TARGET
%replaced by chemical symbol
   nh =    100000,
   idrel = 1,
   flc   = 1.00
   sfin  = 0,
   ipot  = 1,
   isbv  = 3,
   imcp =0,
   ttarget =  TTHICKNESS
    %'TTHICKNESS is replaced
   nqx     =  NUMCELLS,          %
'NUMCELLS' is replaced
   e_cutoff= ENCUTOFF,   ENCUTOFF,
%'ENCUTOFF'is replaced
   qu      = 0.0 ,   1.0,
   case_alpha=0
   alpha0 = ANGLE  , 0.000
 %'ANGLE' is replaced
   qubeam = 1.000, 0.000
   qumax  = 1.00, 1.000
   ioutput_hist = 10000,10000,10000,10000,10000,10000
   ioutput_part = 10000,10000,10000,10000,10000,10000
   lparticle_p = .true.
   lparticle_r = .true.
   lmatrices = .false.
   ltraj_p = .true.
/
```

B3 runtricases.py

```
import string
import os
import re
import sys

from string import *
from shutil import *
#script for modifying and running tri.inp in a sequential manner

#target array
target=['Li'];                  #['C','W','Be'];
#projectile array

projectile= ['He', 'H', 'Li', 'T'];
#energy array

energy=[ 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 200.0,
500.0, 1000.0, 2000.0, 5000.0];
#angle array

angle=[0, 15, 30, 45, 55, 65, 75, 80, 85];
#names to be replaced in template

names=["ENERGY", "PROJECTILE", "TARGET", "NUMCELLS",
"ENCUTOFF", "ANGLE", "TTHICKNESS"];

def write_file(target, projectile, energy, angle, names,
file1="inputtemplate.txt", file2="tri.inp"):

    nqx=200.0

    if energy<=4.0:
        tthickness =   30.0;

    if energy>4.0 and energy<=100.0:
        tthickness = 200.0;

    if energy >1000.0:
       tthickness =10000.0;
```

```
    if energy >=50.0 and energy <= 1000.0:
        tthickness = 2000.0;

    if energy<4:
        ecutoff=.01;

    if energy>4 and energy<100:
        ecutoff=.1;

    if energy>=100:
        ecutoff=1.0;

    f = open("inputtemplate.txt")
    w = open("tri.inp","w")
    s = f.readline()
    while s:
            for i in range(len(names)):
                    s  = s.replace(str((names[0])), str(energy));
                    s  = s.replace(str((names[1])), str((projectile)));
                    s  = s.replace(str((names[2])), str((target)));
                    s  = s.replace(str((names[6])), str(tthickness));
                    s  = s.replace(str((names[5])), str(angle));
                    s  = s.replace(str((names[3])), str(nqx));
                    s  = s.replace(str((names[4])), str(ecutoff));
            w.write(s)
            s = f.readline()
    f.close()
    w.close()

for i in range(len(target)):
    for j in range(len(projectile)):        #so to start out of H directory
        for k in range(len(energy)):
            for l in range(len(angle)):

                write_file(target[i], projectile[j], energy[k], angle[l], names,
file1="inputtemplate.txt", file2="tri.inp");
                print target[i], projectile[j], energy[k], angle[l]
                os.system("tclsh Run.tcl tri.inp");

                time.sleep(5);
                dirname = str(projectile[j])+"in"+str(target[i]);
```

```
            simname = str(energy[k])+"ev"+str(angle[l])

            drive = str("C:\\Users\\Nathan\\Documents\\Tridyn\\");

            nf_output =
str(drive)+str(dirname)+"\output"+simname+".dat"

            nf_energy_analyse =
str(drive)+str(dirname)+"\energy_analyse"+simname+".dat"

            nf_depth_recoil =
str(drive)+str(dirname)+"\depth_recoil"+simname+".dat"

            nf_depth_proj =
str(drive)+str(dirname)+"\depth_proj"+simname+".dat"

            nf_trajec_stop_p =
str(drive)+str(dirname)+"\\trajec_stop_p"+simname+".dat"

            nf_trajec_back_p =
str(drive)+str(dirname)+"\\trajec_back_p"+simname+".dat"

            nf_partic_back_r =
str(drive)+str(dirname)+"\partic_back_r"+simname+".dat"

            nf_partic_stop_r =
str(drive)+str(dirname)+"\partic_stop_r"+simname+".dat"

            copyfile("output.dat", nf_output);
            copyfile("energy_analyse.dat", nf_energy_analyse)
            copyfile("depth_recoil.dat", nf_depth_recoil)
            copyfile("depth_proj.dat", nf_depth_proj)
            copyfile("trajec_stop_p.dat", nf_trajec_stop_p)
            copyfile("trajec_back_p.dat", nf_trajec_back_p)
            copyfile("partic_back_r.dat", nf_partic_back_r)
            copyfile("partic_stop_r.dat", nf_partic_stop_r)
```

Appendix C

C1 meanfunc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mean.data"

double meanfunc(char* targ, char* proj, float energy, float angle);

int main(void){

char targ[1][3]={"C"};
char proj[1][3]={"T"};

/*check the function's return value for the interpolated mean*/

printf("\n%f\n",  meanfunc( (char*) targ, (char*) proj, (float) 1027.6,
(float) 53.6));

return(0);

}

double meanfunc(char* targ, char* proj, float energy, float angle)
{
  printf("\n%s\n", targ);
  int i,j;
  double earray[15] = {0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0,
100.0, 200.0, 500.0, 1000.0, 2000.0, 5000.0};
  double angarray[9] = {0.0, 15.0, 30.0, 45.0, 55.0, 65.0, 75.0, 80.0,
85.0};

  char target[4][3] = {"Be", "C", "W", "Li"};
  char projectile[8][3] = {"Be", "C", "D", "He", "H", "Li", "T", "W"};

  int casenum;
  double ediffold=1000.0;
  double ediffnew;
  int emindex;
  double adiffold=100.0;
```

```
  double adiffnew;
  int amindex;
 int ilow, ihigh;
  int jlow, jhigh;


  /*get differences to weight appropriately*/
  double interp_mean;
  double wt1, wt2, wt3, wt4, area;
  double mean[15][9];
  double *imean,*jmean;
  jmean=&mean[0][0];

  //handle high angles of incidence

  if (angle>85.0){
    angle=85.0;
  }

/*null terminator, must define the array to be one byte larger than the
*/

  if (strcmp(targ,target[0])==0) {
    if (strcmp(proj,projectile[0])==0) {
      casenum = 1; /*BeinBe*/
      imean = &mean1[0][0];
    }
    else if (strcmp(proj,projectile[1])==0) {
      casenum = 2; /*CinBe*/
      imean = &mean2[0][0];
    }
    else if (strcmp(proj,projectile[2])==0) {
      casenum = 3; /*DinBe*/
      imean = &mean3[0][0];
    }
    else if (strcmp(proj,projectile[3])==0) {
      casenum = 4; /*HeinBe*/
      imean = &mean4[0][0];
    }
    else if (strcmp(proj,projectile[4])==0) {
      casenum = 5; /*HinBe*/
```

```
    imean = &mean5[0][0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 6; /*LiinBe*/
    imean = &mean6[0][0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 7; /*TinBe*/
    imean = &mean7[0][0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 8; /*WinBe*/
    imean = &mean8[0][0];
   }
 }
 else if (strcmp(targ,target[1])==0) {
   if (strcmp(proj,projectile[0])==0) {
    casenum = 9; /*BeinC*/
    imean = &mean9[0][0];
   }
   else if (strcmp(proj,projectile[1])==0) {
    casenum = 10; /*CinC*/
    imean = &mean10[0][0];
   }
   else if (strcmp(proj,projectile[2])==0) {
    casenum = 11; /*DinC*/
    imean = &mean11[0][0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 12; /*HeinC*/
    imean = &mean12[0][0];
   }
   else if (strcmp(proj,projectile[4])==0) {
    casenum = 13; /*HinC*/
    imean = &mean13[0][0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 14; /*LiinC*/
    imean = &mean14[0][0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 15; /*TinC*/
```

```
    imean = &mean15[0][0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 16; /*WinC*/
    imean = &mean16[0][0];
   }
 }
 else if (strcmp(targ,target[2])==0) {
   if (strcmp(proj,projectile[0])==0) {
    casenum = 17; /*BeinW*/
    imean = &mean17[0][0];
   }
   else if (strcmp(proj,projectile[1])==0) {
    casenum = 18; /*CinW*/
    imean = &mean18[0][0];
   }
   else if (strcmp(proj,projectile[2])==0) {
    casenum = 19; /*DinW*/
    imean = &mean19[0][0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 20; /*HeinW*/
    imean = &mean20[0][0];
   }
   else if (strcmp(proj,projectile[4])==0) {
    casenum = 21; /*HinW*/
    imean = &mean21[0][0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 22; /*LiinW*/
    imean = &mean22[0][0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 23; /*TinW*/
    imean = &mean23[0][0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 24; /*WinW*/
    imean = &mean24[0][0];
   }
 }
 else if (strcmp(targ,target[3])==0) {
```

```
    if (strcmp(proj,projectile[0])==0) {
      casenum = 25; /*BeinLi*/
      imean = &mean25[0][0];
    }
    else if (strcmp(proj,projectile[1])==0) {
      casenum = 26; /*CinLi*/
      imean = &mean26[0][0];
    }
    else if (strcmp(proj,projectile[2])==0) {
      casenum = 27; /*DinLi*/
      imean = &mean27[0][0];
    }
    else if (strcmp(proj,projectile[3])==0) {
      casenum = 28; /*HeinLi*/
      imean = &mean28[0][0];
    }
    else if (strcmp(proj,projectile[4])==0) {
      casenum = 29; /*HinLi*/
      imean = &mean29[0][0];
    }
    else if (strcmp(proj,projectile[5])==0) {
      casenum = 30; /*LiinLi*/
      imean = &mean30[0][0];
    }
    else if (strcmp(proj,projectile[6])==0) {
      casenum = 31; /*TinLi*/
      imean = &mean31[0][0];
    }
    else if (strcmp(proj,projectile[7])==0) {
      casenum = 32; /*WinLi*/
      imean = &mean32[0][0];
    }
  }

/*  fill in 2D parameter matrix */
  for (i=0;i<15*9;i++) {
  jmean[i]=imean[i];
  }

  /*finds closest energy grid point */
  for (i=0; i<15; i=i+1) {
    ediffnew = fabs(earray[i] - energy);
```

```
   if (ediffnew < ediffold) {
     ediffold = ediffnew;
     emindex = i;
   }
  }

  /*find closest angle grid point */
  for (j=0; j<9; j=j+1) {
   adiffnew = fabs(angarray[j] - angle);
   if (adiffnew < adiffold) {
     adiffold = adiffnew;
     amindex = j;
   }
  }

  /*
  at this point we have the data to intepolate from the nearest values
in the array on energy and angle
  want a weighted linear interpolation, should depend on 4 nearest
points
  bracket the Energy and angle input with the correct values in the
earray and angarray
  */
  if (earray[emindex] < energy) {
   ilow = emindex;
   ihigh = emindex+1;
  }
  else {
   ihigh = emindex;
   ilow = emindex-1;
  }
  if (angarray[amindex] < angle) {
   jlow = amindex;
   jhigh = amindex+1;
  }
  else {
   jhigh=amindex;
   jlow=amindex-1;
  }

  area= (earray[ihigh] - earray[ilow]) * (angarray[jhigh] -
angarray[jlow]);
```

```
  wt1 = ((energy - earray[ilow]) * (angle - angarray[jlow])) / area;
  wt2 = ((energy - earray[ilow]) * (angarray[jhigh] - angle)) / area;
  wt3 = ((earray[ihigh] - energy) * (angle - angarray[jlow])) / area;
  wt4 = ((earray[ihigh] - energy) * (angarray[jhigh] - angle)) / area;

  /*interpolate largest area means farthest away*/

  interp_mean = wt4*mean[ilow][jlow] + wt3*mean[ilow][jhigh] +
wt2*mean[ihigh][jlow] + wt1*mean[ihigh][jhigh];
  printf("%f\n", interp_mean);
  return (interp_mean);
}
```

C2 Arrayimpl.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "SOURCEimplantBeinBe.h"
#include "SOURCEimplantCinBe.h"
#include "SOURCEimplantDinBe.h"
#include "SOURCEimplantHeinBe.h"
#include "SOURCEimplantHinBe.h"
#include "SOURCEimplantLiinBe.h"
#include "SOURCEimplantTinBe.h"
#include "SOURCEimplantWinBe.h"
#include "SOURCEimplantBeinC.h"
#include "SOURCEimplantCinC.h"
#include "SOURCEimplantDinC.h"
#include "SOURCEimplantHeinC.h"
#include "SOURCEimplantHinC.h"
#include "SOURCEimplantLiinC.h"
#include "SOURCEimplantTinC.h"
#include "SOURCEimplantWinC.h"
#include "SOURCEimplantBeinW.h"
#include "SOURCEimplantCinW.h"
#include "SOURCEimplantDinW.h"
#include "SOURCEimplantHeinW.h"
#include "SOURCEimplantHinW.h"
#include "SOURCEimplantLiinW.h"
#include "SOURCEimplantTinW.h"
#include "SOURCEimplantWinW.h"
#include "SOURCEimplantBeinLi.h"
#include "SOURCEimplantCinLi.h"
#include "SOURCEimplantDinLi.h"
#include "SOURCEimplantHeinLi.h"
#include "SOURCEimplantHinLi.h"
#include "SOURCEimplantLiinLi.h"
#include "SOURCEimplantTinLi.h"
#include "SOURCEimplantWinLi.h"
#include "rcfunc.h"

double implfunc(char* targ, char* proj, double energy, double angle,
double depth);
```

```
int main(void){
  char targ[1][3]={"C"};
  char proj[1][3]={"T"};

  /*check the function's return value for the interpolated flux at
depth*/

  printf("\n%e\n", implfunc((char*) targ,(char*) proj, (double) 1027.6,
(double) 53.6, (double) 23.0));

  return(0);
}

double implfunc(char* targ, char* proj, double energy, double angle,
double depth)

{
  double implarr[15][9][200];
  double dximpl[15];
  double *iimplarr,*jimplarr;
  double *idximpl,*jdximpl;

  jimplarr = &implarr[0][0][0];
  jdximpl = &dximpl[0];

  int i,j,klow, khigh;

  double earray[15] = {0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0,
100.0, 200.0, 500.0, 1000.0, 2000.0, 5000.0};

  double angarray[9] = {0.0, 15.0, 30.0, 45.0, 55.0, 65.0, 75.0, 80.0,
85.0};

/*null terminator, must define the array to be one byte larger than the
largest string*/

  char target[4][3] = {"Be", "C", "W", "Li"};
  char projectile[8][3] = {"Be", "C", "D", "He", "H", "Li", "T", "W"};
```

```
  int casenum;

  double ediffold=1000.0;
  double ediffnew;
  int emindex;

  double adiffold=100.0;
  double adiffnew;
  int amindex;

  int ilow, ihigh;
  int jlow, jhigh;

  double fluence = 1.0e16;
  double rc_int, integral_sum, dbig, dsm, beta, norm, breadth, density;
  double interp_arr[200];
  double xbig[50];
  double xsm[200];

  //max size for xsm greater than largest difference in adjacent dx
values (.15 to 5 in Be in Be)

  //handle high angles of incidence

  if (angle>85.0){
    angle=85.0;
  }

  if (strcmp(targ,target[0])==0) {
    if (strcmp(proj,projectile[0])==0) {
      casenum = 1; //BeinBe
      iimplarr = &implarrBeinBe[0][0][0];
      idximpl = &dxBeinBe[0];
    }
    else if (strcmp(proj,projectile[1])==0) {
      casenum = 2; //CinBe
      iimplarr = &implarrCinBe[0][0][0];
      idximpl = &dxCinBe[0];
    }
    else if (strcmp(proj,projectile[2])==0) {
      casenum = 3; //DinBe
```

```
    iimplarr = &implarrDinBe[0][0][0];
    idximpl = &dxDinBe[0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 4; //HeinBe
    iimplarr = &implarrHeinBe[0][0][0];
    idximpl = &dxHeinBe[0];
   }

   else if (strcmp(proj,projectile[4])==0) {
    casenum = 5; //HinBe
    iimplarr = &implarrHinBe[0][0][0];
    idximpl = &dxHinBe[0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 6; //LiinBe
    iimplarr = &implarrLiinBe[0][0][0];
    idximpl = &dxLiinBe[0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 7; //TinBe
    iimplarr = &implarrTinBe[0][0][0];
    idximpl = &dxTinBe[0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 8; //WinBe
    iimplarr = &implarrWinBe[0][0][0];
    idximpl = &dxWinBe[0];
   }
  }
  else if (strcmp(targ,target[1])==0) {
   if (strcmp(proj,projectile[0])==0) {
    casenum = 9; //BeinC
    iimplarr = &implarrBeinC[0][0][0];
    idximpl = &dxBeinC[0];
   }
   else if (strcmp(proj,projectile[1])==0) {
    casenum = 10; //CinC
    iimplarr = &implarrCinC[0][0][0];
    idximpl = &dxCinC[0];
   }
   else if (strcmp(proj,projectile[2])==0) {
```

```
    casenum = 11; //DinC
    iimplarr = &implarrDinC[0][0][0];
    idximpl = &dxDinC[0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 12; //HeinC
    iimplarr = &implarrHeinC[0][0][0];
    idximpl = &dxHeinC[0];
   }
   else if (strcmp(proj,projectile[4])==0) {
    casenum = 13; //HinC
    iimplarr = &implarrHinC[0][0][0];
    idximpl = &dxHinC[0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 14; //LiinC
    iimplarr = &implarrLiinC[0][0][0];
    idximpl = &dxLiinC[0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 15; //TinC
    iimplarr = &implarrTinC[0][0][0];
    idximpl = &dxTinC[0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 16; //WinC
    iimplarr = &implarrWinC[0][0][0];
    idximpl = &dxWinC[0];
   }
  }
  else if (strcmp(targ,target[2])==0) {
   if (strcmp(proj,projectile[0])==0) {
    casenum = 17; //BeinW
    iimplarr = &implarrBeinW[0][0][0];
    idximpl = &dxBeinW[0];
   }
   else if (strcmp(proj,projectile[1])==0) {
    casenum = 18; //CinW
    iimplarr = &implarrCinW[0][0][0];
    idximpl = &dxCinW[0];
   }
   else if (strcmp(proj,projectile[2])==0) {
```

```
    casenum = 19; //DinW
    iimplarr = &implarrDinW[0][0][0];
    idximpl = &dxDinW[0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 20; //HeinW
    iimplarr = &implarrHeinW[0][0][0];
    idximpl = &dxHeinW[0];
   }
   else if (strcmp(proj,projectile[4])==0) {
    casenum = 21; //HinW
    iimplarr = &implarrHinW[0][0][0];
    idximpl = &dxHinW[0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 22; //LiinW
    iimplarr = &implarrLiinW[0][0][0];
    idximpl = &dxLiinW[0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 23; //TinW
    iimplarr = &implarrTinW[0][0][0];
    idximpl = &dxTinW[0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 24; //WinW
    iimplarr = &implarrWinW[0][0][0];
    idximpl = &dxWinW[0];
   }
  }
  else if (strcmp(targ,target[3])==0) {
   if (strcmp(proj,projectile[0])==0) {
    casenum = 25; //BeinLi
    iimplarr = &implarrBeinLi[0][0][0];
    idximpl = &dxBeinLi[0];
   }
   else if (strcmp(proj,projectile[1])==0) {
    casenum = 26; //CinLi
    iimplarr = &implarrCinLi[0][0][0];
    idximpl = &dxCinLi[0];
   }
   else if (strcmp(proj,projectile[2])==0) {
```

```
    casenum = 27; //DinLi
    iimplarr = &implarrDinLi[0][0][0];
    idximpl = &dxDinLi[0];
   }
   else if (strcmp(proj,projectile[3])==0) {
    casenum = 28; //HeinLi
    iimplarr = &implarrHeinLi[0][0][0];
    idximpl = &dxHeinLi[0];
   }
   else if (strcmp(proj,projectile[4])==0) {
    casenum = 29; //HinLi
    iimplarr = &implarrHinLi[0][0][0];
    idximpl = &dxHinLi[0];
   }
   else if (strcmp(proj,projectile[5])==0) {
    casenum = 30; //LiinLi
    iimplarr = &implarrLiinLi[0][0][0];
    idximpl = &dxLiinLi[0];
   }
   else if (strcmp(proj,projectile[6])==0) {
    casenum = 31; //TinLi
    iimplarr = &implarrTinLi[0][0][0];
    idximpl = &dxTinLi[0];
   }
   else if (strcmp(proj,projectile[7])==0) {
    casenum = 32; //WinLi
    iimplarr = &implarrWinLi[0][0][0];
    idximpl = &dxWinLi[0];
   }
}

//fill in 3D implant array matrix
for (i=0;i<15*9*200;i++){
  jimplarr[i]=iimplarr[i];
}

//fill in 2D dx array
for (i=0;i<15;i++){
  jdximpl[i]=idximpl[i];
}

//finds closest energy grid point
```

```
  for (i=0; i<15; i=i+1) {
    ediffnew = fabs(earray[i] - energy);
    if (ediffnew < ediffold) {
      ediffold = ediffnew;
      emindex = i;
    }
  }

  //find closest angle grid point
  for (j=0; j<9; j=j+1) {
    adiffnew = fabs(angarray[j] - angle);
    if (adiffnew < adiffold) {
      adiffold = adiffnew;
      amindex = j;
    }
  }
  /*
  at this point we have the data to intepolate from the nearest   values
in the array on energy and angle
  want a weighted linear interpolation, should depend on 4 nearest
points
 bracket the Energy and angle input with the correct values in the
earray and angarray
  */

  if (earray[emindex] < energy) {
    ilow = emindex;
    ihigh = emindex+1;
  }
  else {
    ihigh = emindex;
    ilow = emindex-1;
  }

 if (angarray[amindex] < angle) {
    jlow = amindex;
    jhigh = amindex+1;
  }
  else {
    jhigh=amindex;
    jlow=amindex-1;
  }
```

```
  //get interpolated reflection coefficient from rcfunc.c

  rc_int=rcfunc(targ, proj, energy, angle);

  //add all the historam arrays of implanted density together case of
same step size

  //interp_array is the sum of the 4 arrays from ilowjlow etc., the
quantity is normalized

  if (dximpl[ilow]==dximpl[ihigh]){
    dsm=dximpl[ilow];
    for (i=0;i<50;i++){
      xsm[i]=dsm*i;
    }

    integral_sum = 0.0;
    for (i=0;i<50;i++){
      interp_arr[i]=implarr[ilow][jlow][i]+implarr[ilow][jhigh][i]+implar
r[ihigh][jlow][i]+implarr[ihigh][jhigh][i];
      integral_sum=integral_sum + interp_arr[i]*dsm;
    }

    //normalized with the interpolated reflection coefficient
    norm=(1.0-rc_int)*(fluence/integral_sum);
    for (i=0;i<50;i++){
      interp_arr[i]=norm*interp_arr[i];
    }
    //determine depth index, if past array max, set =0
    klow= depth/dsm;
    khigh= depth/dsm + 1;

    if (klow>breadth){
      interp_arr[klow] = 0.0;
      interp_arr[khigh]=0.0;
    }
  }

  //must add four histograms of different precision because the arrays
have different depth cell size, first for
```

```
  //this normalizes low e arrays
  else {
    dbig=dximpl[ihigh];
    dsm = dximpl[ilow];

    for (i=0;i<50;i++){
      xbig[i]=dbig*i;
    }

    breadth=xbig[49]/dsm+1;
    beta=dbig/dsm;

    for (i=0;i<breadth;i++){
      xsm[i]=dsm*i;
    }

    // i is index of histogram with smaller precision
    i=1; j=1;
    while (i<breadth) {
      interp_arr[i]=implarr[ilow][jlow][i]+implarr[ilow][jhigh][i]+
                    (1.0/beta)*(implarr[ihigh][jlow][j]+implarr[ihigh][jhigh][j
]);
      i++;
      if (beta*i>xbig[j]){
        interp_arr[i]=implarr[ilow][jlow][i]+implarr[ilow][jhigh][i]+
                (1.0/beta)*(((xbig[j]-
xsm[i])/dsm)*(implarr[ihigh][jlow][j]+implarr[ihigh][jhigh][j]))+

                (1.0/beta)*(((xsm[i+1]-
xbig[j])/dsm)*(implarr[ihigh][jlow][j+1]+implarr[ihigh][jhigh][j+1]))
        i++;
        j++;
      }
    }

    integral_sum = 0.0;
    for (i=0;i<breadth;i++){
      integral_sum=integral_sum + interp_arr[i]*dsm;
    }

    //normalized with the interpolated reflection coefficient
    norm=(1.0-rc_int)*(fluence/integral_sum);
```

```
    for (i=0;i<breadth;i++){
      interp_arr[i]=norm*interp_arr[i];
    }

      //determine index of depth input integer division rounds down to
zero
    //determine depth index, if past array max, set =0

    klow= depth/dsm;
    khigh= depth/dsm + 1;
    if (klow>breadth){
      interp_arr[klow] = 0.0;
      interp_arr[khigh]=0.0;
    }
  }

  // end of loop defining interp_arr[i]
  //do linear interpolation
  //simple line using xlow,xhigh, shifting line to the left deltax/2 so that
points are at center of 'bin'

  density= ((interp_arr[khigh]-interp_arr[klow])/(dsm))*(depth-
(xsm[klow]-dsm/2))+interp_arr[klow];

    printf("\n%e\n%f\n",integral_sum,norm);
    return(density);
}
```

References

1. W. Moller and W. Eckstein, Nucl. Instr. and Meth. B2 (1984) 814-818.

2. W. Moller and W. Eckstein, CPC 51 (1988) 355-388.

3. J. Lindhard, V. Nielsen, and M. Scharff, Kgl. Dan. Vid. Selsk. Mat.-Fys. Medd. 36 (1968) 10.

4. O. B. Firsov, Sov. Phys. JETP 6 (1958) 534.

5. W. D. Wilson, L.G. Haggmark and J.P. Biersack, Phys. Rev. B 15 (1977) 2458.

6. W. Moller, G. Pospiech and G. Schrieder, Nucl. Instr. and Meth. 130 (1975) 265.

7. farside.ph.utexas.edu/teaching/336k/lectures/node65.htm

8. J.P. Biersach and L.G. Haggmark, Nucl. Instr. and Meth. 174 (1980) 257.

9. farside.ph.utexas.edu/teaching/336k/lectures/node66.htm

10. J. Lindhard and M. Scharff, Phy. Rev. 124 (1961) 128.

11. O.S. Oen and M.T. Robinson, Nucl. Instr. and Meth. 132 (1976) 647.

12. A. Yu. Pigarov, S.I. Krasheninnikov. “Coupled plasma-wall modeling with WALL-PSI code” UCSD.

13. A.Yu. Pigarov. “Wall and Plasma-Surface Interaction (WallPSI) Module” FACETS All Hand Meeting 2007.