Appendix A

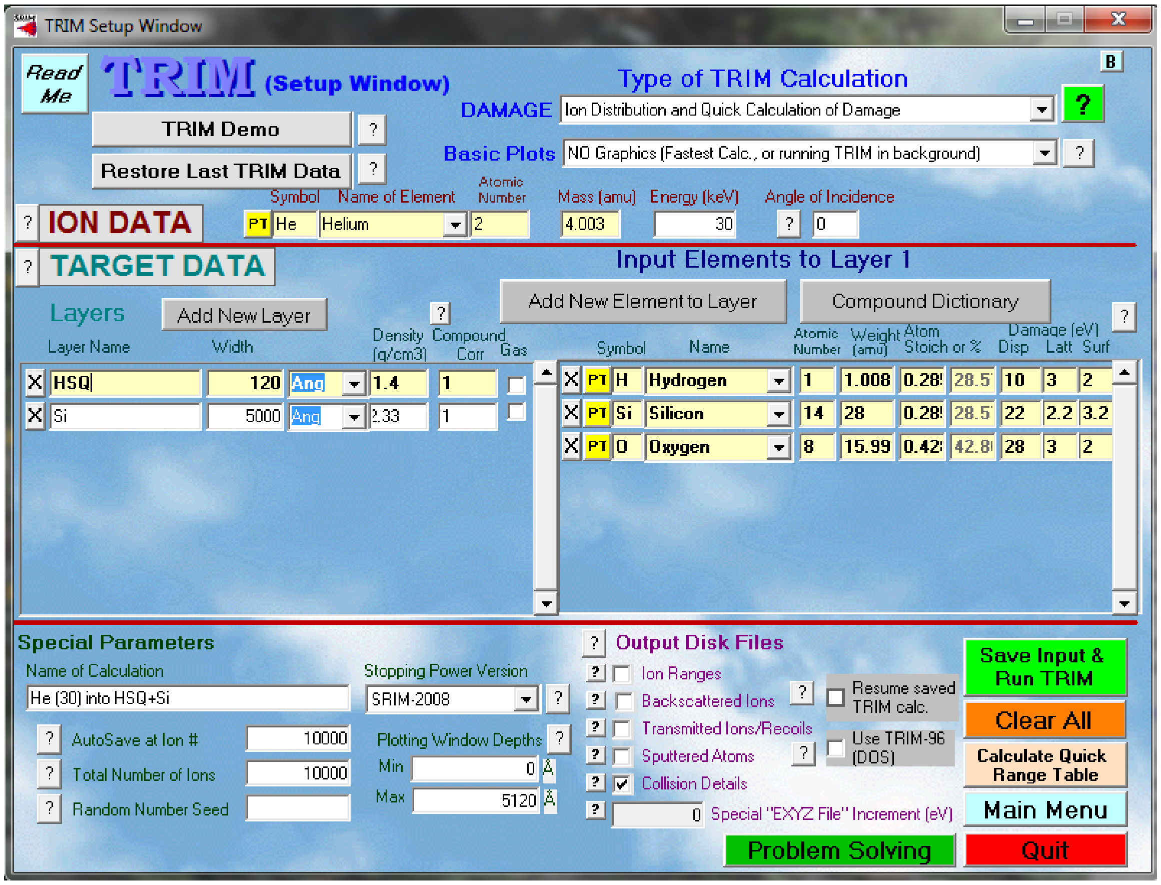
PSF Simulation Procedure and Code

This appendix details the simulation process. First, SRIM [30][1] is used to generate ion trajectories and associated data. Next, a script is used to format the SRIM output for input to the "full" simulator. Also, a separate input file is given to the full simulator, and the format of this file will be presented. Finally, the full code of the simulator will be presented.

A.1 SRIM

A TRIM Calculation within SRIM is performed to generate a set of ion trajectories and associated collision details in order to determine the energy losses at each step in a trajectory, some of which may be interpreted as lost to the target medium (for example, the resist film or the substrate) along the trajectory and some of which may

[1]SRIM Version 2008.04 was used for this work. This detail is not expected to be of technical significance, but may be helpful if certain user-interface elements are renamed or otherwise altered in future versions of the software.



Type of TRIM Calculation

DAMAGE: Ion Distribution and Quick Calculation of Damage

Basic Plots: NO Graphics (Fastest Calc., or running TRIM in background)

ION DATA

Symbol	Name of Element	Atomic Number	Mass (amu)	Energy (keV)	Angle of Incidence
PT He	Helium	2	4.003	30	0

TARGET DATA

Layers

Layer Name	Width		Density (g/cm3)	Compound Corr	Gas
HSQ	120	Ang	1.4	1	☐
Si	5000	Ang	2.33	1	☐

Input Elements to Layer 1

Symbol	Name	Atomic Number	Weight (amu)	Atom Stoich	or %	Damage (eV) Disp	Latt	Surf
PT H	Hydrogen	1	1.008	0.28	28.5	10	3	2
PT Si	Silicon	14	28	0.28	28.5	22	2.2	3.2
PT O	Oxygen	8	15.99	0.42	42.8	28	3	2

Special Parameters

Name of Calculation: He (30) into HSQ+Si

Stopping Power Version: SRIM-2008

AutoSave at Ion #: 10000

Total Number of Ions: 10000

Random Number Seed:

Plotting Window Depths: Min 0 Å, Max 5120 Å

Output Disk Files

- ☐ Ion Ranges
- ☐ Backscattered Ions
- ☐ Transmitted Ions/Recoils
- ☐ Sputtered Atoms
- ✓ Collision Details
- 0 Special "EXYZ File" Increment (eV)

☐ Resume saved TRIM calc.

☐ Use TRIM-96 (DOS)

Figure A.1: The TRIM Setup Window within SRIM. The option to output collision details to disk is selected, with data to be stored for the primary ions but not for recoils (not shown, but selected when checking the Collision Details option). The type of TRIM calculation is the simplest: ion distribution and quick calculation of damage. The density of HSQ has been specified as 1.4 g/cm^3, different than the density that SRIM calculated automatically based on the input stoichiometry.

be interpreted as lost to secondary electrons as an inelastic process. The output for this information is made available by checking the "Collision Details" option under Output Disk Files in the TRIM Setup Window, as shown in Figure A.1.

Figure A.1 is a screenshot of an example setup window for TRIM that will produce a "COLLISON.txt" [sic] file with the collision details. The calculation done will use the simplest type of TRIM calculation, yielding the ion distribution and quick calculation of damage. Alternatively, full damage cascades (useful for heavy ions) or even monolayer collision steps (useful for calculations about surface sputtering) can be selected as the type of TRIM calculation. Individual material layers, in this case HSQ resist and a Si substrate, are specified by (1) elemental composition and stoichiometry and (2) layer density. Finally, the incident ion species and landing energy are specified.

The COLLISON.txt file records basic information about ion-atom collision kinetics along the trajectories of each of the simulated primary ions. A recorded collision is chosen to be one that results in some displacement of target atoms, that is, a collision that potentially produces a recoil cascade. This choice produces fewer path segments associated with each ion trajectory than would result if all elastic nuclear collisions were recorded. The file also contains the spatial coordinates for each collision and the instantaneous electronic stopping power of the ion in the target at the point of collision.

A notable property of the instantaneous energy associated with an ion for a given collision is that the ion's energy throughout the trajectory is "modified by including a straggling component (random variation) in the calculation, so that the actual ion electronic energy loss between specific collisions can not be directly obtained from this number. Because of statistical fluctuations, the ion energy may actually increase between collisions." [30] It is for this reason that I use the average of the instantaneous stopping powers at the collision points that flank each segment of a trajectory to estimate the energy available for generating secondary electrons, rather than using the difference of the instantaneous energies of an ion at the flanking collision points. This practice will be made explicit when presenting the full simulation code later in this section.

Listing 2 Distilling the SRIM output of interest.

```
import re

f = open('sim_COLLISON.txt','r')
fp = open('sim_COLLISION-fancy.txt','w')

# pass annoying non-data line that starts with \xb3
for line in f:
    if line[0]=='\xb3': break

for line in f:
    if line[0]=='\xb3':
        m = re.split('\xb3',line)
        ionNum, energy, z, y, x, SP = m[1], m[2], m[3], m[4], m[5], m[6]
        fp.write(ionNum+`` ''+energy+`` ''+z+`` ''+y+`` ''+x+`` ''+SP+``\n'')

f.close()
fp.close()
```

A.2 Extracting the needed SRIM output

The COLLISON.txt file consists of a large header and then a section for each simulated ion that includes a summary for the ion and a set of rows that correspond to each collision. The data of interest in each row is the ion number and it's instantaneous energy, three-coordinate position, and instantaneous stopping power. A Python script `COLLISION-scraper`(Listing 2) is used to extract this data and produce a simple, single-space-delimited set of lines with the desired details for each collision. The script assumes a renamed copy of the original COLLISON.txt file within the directory containing the script and uses an imported module (`re`) for regular expressions.

A.3 Input to the simulator

There are two inputs to the full simulator, called `jprox-hebeam` because it is a written-in-Java "proximity-effect" (derivable from the PSF, which is the actual output) simulator for helium ion beam exposure. The first input is the formatted SRIM output, discussed in the last section. The second input is a text file that specifies information about certain configuration parameters, the sample, what output is desired, and the

Listing 3 Setting bounds on the computation.

```
private void configure(Scanner input) {
  prune = (input.nextInt() != 0) ? true: false;
  out.println("Prune paths? (1->true, 0->false): "+prune);
  Particle.maxRank = input.nextInt();
  out.println("How many levels of secondaries? "
      + "(0 means no secondaries): " + Particle.maxRank);
  if (Particle.maxRank > 0) {
    Particle.cutoffE = input.nextDouble();
    out.println("Cutoff energy (eV): "+Particle.cutoffE);
  }
}
```

incident beam. The extraction from the input file and the meaning of each line of input will now be described.

A.3.1 Configuring for efficiency

The first set of inputs sets bounds on what is necessary to compute with regard to electron trajectories (see Listing 3). The first option is whether to prune paths. Pruning a path means to stop following an electron once its energy and position are such that there is only an infinitesimal chance that the electron could subsequently reach a plane, for example within the resist film, for which the spatial distribution of energy dissipation is being recorded. Thus, determining the rest of the electron's trajectory is a waste of time in that context. The saved time could be put toward simulating a greater number of incident ions, for example, to obtain a better estimate of the point spread function (PSF).

The second input specifying what is necessary to compute is the number of generations of secondary electrons (SEs). That is, may the primary particles generate SEs? If so, may those SEs generate SEs? And so on. In order to test the sensitivity of a simulated PSF on the presence or absence of a model for SEs, that is, a mechanism for dissipating energy beyond the primary trajectories, this input provides a convenient means of doing so. Furthermore, the sensitivity associated with allowing different numbers of generations of SEs may be tested. In practice, for simulating PSFs, there is generally a significant difference between including zero versus one generation of SEs and no significant difference between including one versus multiple generations of SEs.

The final input that bounds the extent of computation of SE trajectories is that of cutoff energy. This parameter of course had no meaning when the number of generations of SEs to compute is set to zero.[2] When SEs are included in the simulation, there is a certain energy below which an SE will stop its traversal of the target and will simply deposit all of its energy in the cell of the global mesh grid that corresponds to its current position. Why not allow the electron to continue until it reaches "zero" energy? The physics governing the behavior of projectile electrons in solids is not well known as the energy approaches $\sim$ 100 eV. In fact the classic Bethe stopping power equation does not correspond well with experimental data below electron energies of $\sim$ 10 keV. The semi-empirical Joy-Luo equation that my code uses has good correspondence down to $\sim$ 100 eV but not far below that. David Joy, in using this equation in code presented in his book on Monte Carlo modeling of electron beams, specifies a cutoff energy of 50 eV, where the term "cutoff energy" is used as here. This parameter allows a choice for this cutoff.

A.3.2 Describing the sample

The second set of inputs obtains information about the sample, a.k.a. the target (See Listing 4). The first input is the number of layers of the sample. Here already one encounters a design decision for the simulation, which corresponds to that of SRIM: a sample is composed of a sequence of layers, each of different material composition. In a

[2]Here, the context is that of helium-ion-beam simulation, for which primary particle trajectories are imported from SRIM rather than simulated along with the SEs, as would be the case for an electron-beam simulation.

Listing 4 Obtain sample, layer, and species information.

```
private void getSampleInfo(Scanner input) {
  int numLayers = input.nextInt();
  out.println("Number of layers: "+numLayers);
  sample = new Sample(numLayers);
  for (int i = 0; i < numLayers; i++) {
    out.println("Layer "+(i+1)+":");
    Layer layer = new Layer();
    layer.thickness = input.nextDouble();
    out.println("  Thickness (A): "+layer.thickness);
    layer.numSpecies = input.nextInt();
    out.println("  Number of atom species in layer: "
        + layer.numSpecies);
    layer.species = new Species[layer.numSpecies];
    for (int j = 0; j < layer.numSpecies; j++) {
      out.println("  Species "+(j+1)+":");
      Species species = new Species();
      species.atomicNumber = input.nextDouble();
      out.println("    Atomic number: "+species.atomicNumber);
      species.atomicWeight = input.nextDouble();
      out.println("    Atomic weight: "+species.atomicWeight);
      species.speciesDensity = input.nextDouble();
      out.println("    Density of species in layer (g/cm^3): "
          + species.speciesDensity);
      species.update();
      out.println("    Species excitation energy (eV): "
          + species.excitationEnergy);
      layer.species[j] = species;
    }
    layer.update();
    out.format("  Mean excitation energy (eV): %.1f%n",
        layer.meanExcitationEnergy);
    sample.layer[i] = layer;
  }
  sample.update();
}
```

sense, although the coordinate space is three-dimensional, the effective material space is one-dimensional. The substrate material is simply the bottom layer. Why prompt for the number of layers? Can this number not be inferred from subsequent input about the properties of the layers simply by "chunking" that input or by demanding appropriate delimiters in the formatting of the input text file? The number of layers is requested to simplify both the algorithm for acquisition of the layers' information in a structured way, and also to simplify the structure of the input text file, to rid both of extraneous ornamentation.

A sample is created as an instance of a Sample class (Listing 5), in the style of object-oriented programming. Instances of the Layer class (Listing 6) are then

Listing 5 The Sample class.

```
public class Sample {
  int numLayers;
  Layer[] layer;

  public Sample(int numLayers) {
    this.numLayers = numLayers;
    layer = new Layer[numLayers];
  }

  /**
   * Only update when layer array is initialized and
   * each layer has set thickness.
   */
  public void update() {
    double z = 0;
    for (Layer L : layer) {
      L.inz = z;
      L.outz = L.inz + L.thickness;
      z = L.outz;
    }
  }

  int getLayerIndex(Coord r) {
    double z = r.z;
    int index = -1;
    for (Layer L : layer) {
      if (z < L.inz) {return index;}
      else {index += 1;}
    }
    if (z < layer[layer.length - 1].outz) {return index;}
    else {return index+1;}
  }
}
```

instantiated using the input stream and fed to the sample object. The sample object updates itself with any derivative information based on the input that will be used subsequently by the simulation.

One by one, each layer is created by taking in the layer thickness; the number of distinct atom species in the layer[3]; and for each atom species the atomic number, atomic weight, and density of the species in the layer. A short Python script for generating the atom species information in the proper format is included as Listing 7. Each layer object, just as the sample object does once all layers are added to it, updates itself after creation to calculate and store derivative information based on

[3]For example, this number would be 3 for hydrogen silsesquioxane, the three being hydrogen (H), silicon (Si), and oxygen(O).

Listing 6 The Layer class.

```
public class Layer {
  double thickness;
  int numSpecies;
  Species[] species;
  double meanExcitationEnergy;
  double effectiveElectronDensity;
  double inz, outz;

  /**
   * Only update when species array is initialized and set.
   * Not required for update: thickness, inz, outz.
   */
  void update() {
    numSpecies = species.length;
    for (int i = 0; i < numSpecies; i++) {
      effectiveElectronDensity += species[i].electronDensity;
      meanExcitationEnergy += (species[i].electronDensity *
          Math.log(species[i].excitationEnergy));
    }
    meanExcitationEnergy /= effectiveElectronDensity;
    meanExcitationEnergy = Math.exp(meanExcitationEnergy);
  }
}
```

the text file input. Each layer is thus made aware of its z coordinate closest to $z = 0$, where the beam is incident on the sample, and its z coordinate farthest from $z = 0$, so that a particle object may ask in which layer it is. Each layer knows its effective electron density and its mean excitation energy. Furthermore, a species object within a layer (see Listing 8) also updates itself with quantities such as its electron density and excitation energy. The excitation energy is found by a table lookup (Listing 9).

The `getSampleInfo` method of a simulation (Listing 4) is thus a nested sequence of instantiation of various objects — one sample, its layers, and each layer's set of species — based on minimal input, and each of these objects are updated appropriately with secondary information based on the minimal input that ensures the efficient delivery of needed information to the simulation when subsequent methods are called.

Listing 7 Get species density from the total density and the stoichiometry.

```
#HSQ -> (H8Si8O12)n
names = 'H', 'Si', 'O'
numbers = 1, 14, 8
weights = 1.008, 28.09, 16.000 # atomic weights of H, Si, O
stoichs = 8, 8, 12
d = 1.4 # total density, in g/cm^3

if len(weights) != len(stoichs) and len(weights) != len(names):
    print "error: weights and stoichs must be equal in size"
    import sys; sys.exit(1)
material_weight = sum([s*w for s,w in zip(stoichs,weights)])

# A human-friendly format
for n,w,s in zip(names, weights, stoichs):
    print "density of species",n,"is", d*w*s/ material_weight

# For copy-and-paste into a simulation input file
print len(names)
for Z,w,s in zip(numbers, weights, stoichs):
    print Z
    print w
    print d*w*s / material_weight
```

Listing 8 The Species class.

```
public class Species {
  double atomicNumber;
  double atomicWeight;
  // density of species in layer (g/cm3)
  double speciesDensity;
  // atoms/(Angstrom)^3
  double atomDensity;
  // electrons/(Angstrom)^3
  double electronDensity;
  double excitationEnergy;

  /**
   * Only update when atomicNumber, atomicWeight,
   * and speciesDensity are initialized.
   */
  void update() {
    atomDensity = (Standard.N_A * speciesDensity /
        (1.0E24 * atomicWeight));
    this.electronDensity = atomicNumber * atomDensity;
    excitationEnergy = Standard.mean_I[(int) (atomicNumber-1)];
  }
}
```

Listing 9 Table of mean ionization energies for the elements.

```
/**
 * Table of mean ionization energies for the elements.
 * Source: NIST
 * http://physics.nist.gov/PhysRefData/XrayMassCoef/tab1.html
 */
public static final double[] mean_I =
{19.2, 41.8, 40, 63.7, 76, 78, 82, 95,
115, 137, 149, 156, 166, 173, 173, 180, 174, 188, 190, 191, 216, 233,
245, 257, 272, 286, 297, 311, 322, 330, 334, 350, 347, 348, 343, 352,
363, 366, 379, 393, 417, 424, 428, 441, 449, 470, 470, 469, 488, 488,
487, 485, 491, 482, 488, 491, 501, 523, 535, 546, 560, 574, 580, 591,
614, 628, 650, 658, 674, 684, 694, 705, 718, 727, 736, 746, 757, 790,
790, 800, 810, 823, 823, 830, 825, 794, 827, 826, 841, 847, 878, 890 };
```

A.3.3 Allocating resources for the desired output

After getting material information about the sample, the program determines how to digitize the geometry of the sample and what region of the sample is of interest (Listing 10). The lateral dimension of each voxel can be different than the vertical dimension. This difference can save memory without loss of accuracy. For example, a small lateral voxel spacing (e.g., 1 nm) may be desired to resolve the spatial distribution of forward scattering. However, a larger vertical voxel spacing (e.g., 5 nm) may be acceptable because thinner voxels would need to be averaged anyway for a simulated PSF to be compared to a PSF obtained experimentally by top-down scanning electron microscopy.

Obtaining the z values (vertical coordinates) of interest prior to the simulation, for example a set of planes within a resist film, saves time and memory without compromising accuracy. The set of z values saves time because calculation of a given trajectory may be pruned when the particle no longer can physically return to any region of interest. The set of z values saves memory because the energy dissipated in mesh cells not within the region of interest need not be remembered, so corresponding locations in memory need not be allocated.

Listing 10 Building the mesh grid for tracking the spatial distribution of energy dissipation.

```
private void getMeshInfo(Scanner input) {
  DR = input.nextDouble();
  DY = DR;
  out.println("dr, dy mesh size (A): "+DR);
  DZ = input.nextDouble();
  out.println("dz mesh size (A): "+DZ);
  numZs = input.nextInt();
  out.println("Number of z values for output: "+numZs);
  Irz = new double[numZs * MAX_R_INDEX]; // could be + numZs. Eh.
  Eyz = new double[numZs * MAX_R_INDEX];
  Irz_sec = new double[numZs * MAX_R_INDEX];
  Eyz_sec = new double[numZs * MAX_R_INDEX];
  for (int i=0; i < (numZs * MAX_R_INDEX); i++) {
    Irz[i] = 0; Eyz[i] = 0; Irz_sec[i] = 0; Eyz_sec[i] = 0;
  }
  for (int i=0; i <= MAX_Z_INDEX; i++) {
    z[i] = 0; Ez[i] = 0; Ez_sec[i] = 0;
  }
  zMax = 0;
  for (int i = 0; i < numZs; i++) {
    double zValue = input.nextDouble();
    out.format("  z[%d]: %.1f%n", i, zValue);
    int zIndex = (int) (zValue/DZ + 0.5);
    if (zIndex < MAX_Z_INDEX) {
      // +1 so can check as a boolean
      z[zIndex] = i+1;
      zMax = Math.max(zMax, zValue);
    } else {
      System.err.println(""+zValue+" out of mesh. "
          + "Please input another value.");
    }
  }
}
```

Listing 11 Get auxiliary information about the primary beam, in addition to the formatted SRIM output.

```
private void getBeamInfo(Scanner input) {
  //beamEnergy = input.nextDouble();
  //out.println("Beam energy (eV): "+beamEnergy);
  //beamDiameter = input.nextDouble();
  //out.println("Beam diameter (A): "+beamDiameter);
  ionCount = input.nextInt();
  out.println("Number of ions: "+ionCount);
  initialStoppingPower = input.nextDouble();
  out.println("Initial elec. stopping power in top layer (eV/A):
      "+initialStoppingPower);
}
```

A.3.4 Elaborating on the beam

To interface the simulation to the SRIM output, two additional parameters are needed relating to the beam (Listing 11). The first parameter is the total number of primary-beam ions, which may alternately be derived from the formatted SRIM output but serves here as a check for the user to be sure that the correct SRIM output file is paired with this input file for a simulation; if the number of ions specified is different than the number of ions counted from the formatted SRIM output,then an assertion statement in the simulation will trigger an error.

The second parameter needed about the beam is the electronic stopping power of an ion at it's landing energy in the top layer. This parameter is needed here because the SRIM output tabulates electronic stopping power for an ion at each collision only. Because the electronic stopping power at each point along a trajectory leg is estimated as the average of the stopping powers given for the endpoints of the trajectory leg, a value for the initial stopping power is needed to estimate the stopping power along the first trajectory leg, that is the leg prior to the first collision in the sample.

A.3.5 An example input file

Listing 12 is an example input file. Each input parameter is placed on a separate line without annotation. The file specifies that electron paths should be pruned, one generation of SEs is allowed, and the cutoff energy for SE generation is 20 eV. There are two (2) layers in the sample. The first layer has a thickness of 12 nm (120 Å) and is composed of three (3) atomic species: hydrogen (atomic number 1 and atomic weight 1.008) with a density in the layer of 0.0266 g/cm^3, silicon (atomic number 14 and atomic weight 28.09) with a density in the layer of 0.741 g/cm^3, and oxygen (atomic number 8 and atomic weight 16.0) with a density in the layer of 0.633 g/cm^3. The ten numbers describing the material of this layer may be generated by running the code of Listing 7: the material is HSQ of density 1.4 g/cm^3. The second layer has a thickness of 1 μm and is composed of one atomic species: silicon with a density of 2.33 g/cm^3. The lateral mesh grid spacing is 1 nm (10 Å) and the vertical spacing is 5 nm. There is one plane of interest for tabulating energy dissipation for plotting a PSF, namely that at $z = 5$ nm, or to be more precise given the mesh grid spacing, the slice $z = 5 - 10$ nm within the 12 nm film of HSQ. Finally, there are 10,000 ions in the incident beam to be accounted for, and the instantaneous electronic stopping power of an ion at its landing energy in the first layer (the HSQ) is 60.42 eV/nm (6.042 eV/Å).

A.4 Full code listing

This section lists each of the Java classes that comprise the `jprox-hebeam` program.

Listing 12 Example input file to simulation.

```
1
1
20
2
120
3
1
1.008
0.0265772722136
14
28.09
0.740630532223
8
16.0
0.632792195563
10000
1
14
28.09
2.33
10.0
50.0
1
50.0
10000
6.042
```

A.4.1 General methods and constants

Standard.java

```java
package sim;


/**
 * Provides a few static final methods and constants for general use.
 * @author Donny
 *
 */
public class Standard {
  public static final double PI = Math.PI;
  public static final double TWOPI = 2 * PI;
  public static final double PIOVERTWO = PI / 2;
  public static final double Q = 1.60219E-19;
  public static final double ESU = 4.80324E-10;
  public static final double N_A = 6.022E23;
  /* scaling constant for SE generation rate of HeIons,
   * from Ramachandra 2009. */
```