Guide to Using run1.cpp

Samuel Johnson Stoever
Cornell University
sj324@cornell.edu

Summer 2008 UCLA REU Program

1 Preliminaries

1.1 Language, Software, and Hardware

The simulation was written in C++ corresponding to the ANSI/ISO C++ standard. The program makes use of various C++ Standard Library files: `iostream`, `cstdlib`, `fstream`, and `cmath`. The program has been successfully compiled (using the GNU Compiler Collection (GCC)) and run on a Mac OS X 10.5.4 computer with a 2.8 GHz Intel Core Duo processor and 4 GB of RAM, and on a Ubuntu 8.04 'Hardy Heron' Linux computer with a 1.6 GHz AMD Turion 64 processor and 2 GB of RAM.

1.2 Goals and Outline

The program was the result from an REU project entitled 'Monte Carlo Simulations of Extragalactic H and He Cloud Ionization and Heating from Quasar Emissions' at UCLA during the summer of 2008. The research advisor was Prof. Furlanetto of UCLA.

The program starts with a photon with pre-determined energy, the program then selects if the photon ionizes an H I , He I, or He II atom via the random generator and photoionization cross-section data. The ionization event then creates a photoelectron with a definable energy which then interacts with other matter in the 'cloud' - we randomly select processes out of: electron-impact ionization of H I, He I, or HeII, electron-impact excitation of H I, He I, or He II, and electron-electron coulomb interactions that cause the energetic electron to lose energy and heat the 'cloud.' The energy of the main photoelectron is followed, as are the creation of secondary electrons (through the ionization processes), excitation to Lyα producing levels in the H atom, and how much heat is imparted on the gas via electron-electron interactions. The effects of the secondary electrons on the gas are also followed until the energies of all electrons are below the 10.2 eV threshold (where no more excitations or ionizations can occur).

1.3 Numerical Recipes

The simulation makes use of code provided by the 'Numerical Recipes' series of books, written by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Plannery. The random number generator used, `ran1`, is taken from a 2nd edition, 'Numerical Recipes in C', while the interpolating functions, `spline` and `splint`, are taken from the 3rd edition printing, 'Numerical Recipes in C++', in order to take advantage of the object-oriented programming capabilities of C++. Discussions of these functions and classes follow in their appropriate sections of the guide.

1.4 Compiling

As long as the required files are present, and the required libraries (C++ Standard Library) are available, one can simply compile via the GCC:

```
g++ -o run1 run1.cpp
```

and run the program through the usual way (e.g., `./run1`).

1.5 Warnings

The code has not been tested on 64-bit compilers, and some of the definitions instituted by the NR3.h header file may not work well with 64-bit compilers.

2 General Physical Parameters and Constants

2.1 Units

The program considers all activity to take place within the realm of CGS units.

2.2 Particle Densities

We use the following functions to describe the particle densities in our monte carlo simulation (where is $\Omega_b h^2 \approx 0.024$). The Hydrogen density (represented by `nSUBh` in the code) is given by

$$n_H = \left(8.56 \times 10^{-6}\right) \cdot \Omega_b h^2 \cdot (1+z)^3 \;\; cm^{-3} \; .$$

The Helium density (represented by `nSUBhe` in the code) is given by:

$$n_{He} = \frac{1}{4}\left(\frac{0.24}{1-0.24}\right) \times n_H,$$

and finally, the electron density (represented by `nSUBe` in the code) is given by:

$$n_e = \left(9.83 \times 10^{-6}\right) \cdot \Omega_b h^2 \cdot (1+z)^3 \;\; cm^{-3} \; .$$

These are all obviously functions of z (the redshift age of the universe), which is represented in the program via the variable `zAge`. The value for n_e is used in the calculation for the Coulomb Logarithm (used in the electron-electron coulomb interaction cross-section), while all three of these values are used to determine interaction probabilities within in the monte carlo code.

2.3 The Coulomb Logarithm

We require the use of the coulomb logarithm to calculate cross-section data for the electron-electron heating interactions. We obtain this function from Spitzer 1962, and it is defined as

$$\ln \Lambda$$

where

$$\Lambda = \frac{3}{2e^3}\left[\frac{k^3 T^3}{\pi n_e}\right]^{\frac{1}{2}}$$

for electron-electron interactions. In the expression above, e is the elementary charge, k is the Bolzmann constant, T is the temperature, and n_e is the electron density. When calculating the Coulomb Logarithm, the program takes into account `temp` and `nSUBe`, the temperature and the electron density, respectively. Due to the nature of logarithms, the Coulomb Logarithm is fairly insensitive to even moderate changes in T and n_e. $\ln \Lambda$ is represented by `coulLog` in the program.

Calculation of `coulLog` requires the use of < `cmath` > for `log()` and `pow()`.

2.4 Ionization Potentials

This program requires the use of ionization potentials primarily to describe the energy of the primary photoelectron that is produced as a product of the initial photoionization. The ionization potentials are stored in the array `ionPotential` where the ionization potentials for H I, He I, and He II are stored as `ionPotential`[0], `ionPotential`[1], and `ionPotential`[2], respectively. The values in use in the array are {13.598, 24.586, 54.4}.

2.5 Excitation Energies for Hydrogen I

The excitation energies required for the Hydrogen I electron to make a jump from the $n = 1$ state to the n^{th} state can be given by an analytic equation

$$E_n = 13.6 \cdot \left(1 - \frac{1}{n^2}\right) \; eV \; .$$

The electron incident on the Hydrogen atom must then lose E_n one it has been selected to excite the atom to the n^{th} state.

2.6 Excitation Energies for Helium I

The excitation energies for the He I atom were retreived from the 'NIST Atomic Spectra Database Levels Form' (http://physics.nist.gov/PhysRefData/ASD/levels_form.html)(retreived 13 August, 2008). The data are stored in the array `exitHeIenergies`. Refer to the source code to view the values contained.

2.7 Excitation Energies for Helium II

The excitation energies required for the Helium II electron to make a jump from the $n = 1$ state to the n^{th} state can be given by an analytic equation

$$E_n = 54.4 \cdot \left(1 - \frac{1}{n^2}\right) \; eV \; .$$

2.8 Energy Distributions of Secondary Electrons after electron-impact ionization

In order to calculate the enrgy loss of the primary and to tabulate the energies of the secondary electrons produced in this process, we decided to use the probability density

$$\frac{1}{1 + (\varepsilon/\bar{\varepsilon})^{2.1}}$$

so that our function is

$$g(\varepsilon_{max}) = \int_0^{\varepsilon_{max}} \frac{1}{1 + (\varepsilon/\bar{\varepsilon})} d\varepsilon \tag{1}$$

where ϵ_{max} is given by

$$\varepsilon_{max} = \frac{1}{2} (E - I)$$

and where E is the energy of the incident electron and I is the ionization potential. The constant $\bar{\varepsilon}$ is equal to 8 eV, 15.8 eV, and 32.6 eV for H I, He I, and He II, respectively. Because of the probabilistic nature of the energy distribution, the difficulty of the integral, and the desire to not institute a root finding function, we generated a table for $g(\varepsilon_{max})$ from each value $\varepsilon_{max} \in [1, 600]$ for each element, and wrote a procedure (contained in the function `electronEnergy`) that would use these tables to find the closest integer energy to the electron energy suggested by the generated random number. If one has the desire to find a more precise function, the aforementioned tables can easily be interpolated. The information presented in this section was taken from Dalgarno, Yan, and Liu (1999), some of which was adopted from the work contained in Opal, Peterson, and Beaty (1971).

3 Header Files and Packages in `run1.cpp`

3.1 `iostream`, `cstdlib`, `cmath`, `ctime`, `fstream`, and `iomanip`

These header libraries are from the C++ Standard Library.

3.2 `nr3.h`

This file is the Library file from the 'Numerical Recipes in C++' book (see section 1.3). The file is required to run our interpolating program, and provides compiler independent definitions for certain data structures used therein.

3.3 `interp_1d.h`

This file contains our interpolation program. The source code was taken from 'Numerical Recipes in C++' (see section 1.3). The interpolation file creates the class template that can be constructed into a function by feeding VecDoub (a data strcture defined in *nr3.h*) arrays via the constructor

```
Spline_interp  function(xVecDoub, yVecDoub);
```

and the interpolated value at a given x is called by:

```
function.interp(x);
```

Also, see utilities `gen1` and `gen2`.

3.4 `photoion.cpp`

This file provides the function `photoion_CrossSection(double E, int species)`, which gives the photoionization cross-sections for a photon with energy E. The variable `species` is 0 for HI, 1 for HeI, and 2 for HeII.

3.5 `elecexitHeII.cpp`, `elecexitHeI.cpp`, and `elecexitHeIshell3.cpp`

These files provide the electron impact excitation cross-sections at different energy levels for HeII and HeI.

3.6 `elecInteracts.cpp`

This file provides the function for the cross-section of the electron-electron Coulomb heating interactions.

3.7 `random1.cpp`

This file contains the random generator function `ran1` as described in the book, Numerical Recipes in C, 2nd ed. (see Section 1.3). This routine uses the system time as the seed for the random number generator. Once the function has been seeded, a random number is generated with the function call:

```
randomnun = ran1(idup) .
```

3.8 `exitHe1function.cpp`

This file contains the array `exitHeIenergies[]` which contains the excitation energies for a number of excitation transistions in the HeI atom. The specific transitions are documented in the file itself.

3.9 `energyLoss.cpp`

This file contains the function `energyLoss(int index1, int nlevel)` that provides the the energy loss from the ground state to the `nlevel` state. `index1 = 0` corresponds to HI and `index1 = 1` corresponds to HeII.

3.10 `giantData.cpp`

This file contains 3 large arrays, each corresponding to values obtained from the interpolation of the integral in Equation 1 (see Section 2.8). The integral is a complicated one and it would take an excessive amount of time to accurately integrate this function at a certain value every time it is needed. The integration was carried out in Mathematica 6.0.

Along with the arrays, this file contains the function `electronEnergy(int specimen, double randomnum, double energy1)` that processes the interpolation arrays, and returns the energy of the secondary electron produced.

4 Executable Files

Although the `run1.cpp` file is the main executable to run the simulation, it is only designed to simulate photons of one particular energy level per run. Because of this I have designed a simple script generator, `scriptGen.cpp`, that will generate a script that will run `./run1` for each desired energy, that will output the results to a different file. The `scriptGen` routine also writes the program `mean.cpp` into the script, so that the use of `scriptGen` also generates the final file that can be plotted, the mean of the results generated by `run1.cpp` at each value of the starting photon energy.

4.1 Example Simulation Run

We now describe how to run a simulation that encompasses every integral energy starting at 1 eV and ending at $900\,eV$, for initial photon energy, at $50,000$ trials per energy level. We then want the mean energy converted into Lyα photons, and the mean energy converted into heat via electron-electron Coulomb heating. We start by editing the scriptGen.cpp file, and setting the variables: `begin` = 1, `end` = 900, `increment` = 1, and `iterations` = `50000`. We then compile necessary files:

```
g++ -o run1 run1.cpp
g++ -o mean mean.cpp
g++ -o scriptGen scriptGen.cpp
```

We now use the script generator to obtain an appropriate executable script:

```
./scriptGen -fhm1.dat -gpm1.dat
```

we then change the permissions of the script files so that we can run it, and then we run the script:

```
chmod 777 script1.sh
./script1.sh
```

The simulation should then run, and the plottable files are located in the subfolder `meanoutput` under `hm1.dat` and `pm1.dat` for the electron heating energy and the Lyman α energy, respectively.

5 Command Line Arguments

5.1 For `run1.cpp`

An example for the usage of command line arguments for `run1.cpp` could be:

```
./run1 -i2000 -o1000 -hdata/heat.dat -ldata/lyalpha.dat .
```

This will make `run1` iterate 2000 times with an initial photon energy of 1000 eV where it will output the heating energy (per iteration) and energy put into lyman α photons into the files `heat.dat` and `lyalpha.dat`, respectively, which are both in the subdirectory `data`.

It should be noted that there can be no space between the '`-x`' command and the parameters of the command, i.e., 2000.

The available commands are

Command	Typical Usage	Effect
`-i`	`-i40000`	The program will simulate the outcome of 40000 photons at the given energy
`-e`	`-e900`	The initial photon will have an energy of 900 eV
`-h`	`-hdata/file2.dat`	Will output Coulomb heating energy results to `data/file2.dat`
`-l`	`-ldata/file3.dat`	Will output the amount of heat released to Lyman α photons to the file.
`-z`	`-z3.5`	Will set the age of the universe to $z = 3.5$

5.2 For `scriptGen.cpp`

The `scriptGen.cpp` utility program has two command line options, `−f` and `−g`. These two options set the filenames (without the .dat postfix) for the outputs of the `mean` program, which will be in the `meanoutput` folder. The `−f` option sets the output file for electron-electron Coulomb heating, and the $-g$ option sets the output file for Lyman α energy. For example calling

```
./scriptGen −fpm1 −ghm1
```

sets the outputs to `meanoutput/pm1.dat` and `meanoutput/hm1.dat`.

5.3 For `mean.cpp`

Since the `mean` program is generally only used when using `scriptGen` we omit a detailed description of the command line arguments. In short, there are 6 options, `−e`, `−i`, `−o`, `−j`, `−p`, `−f`. All but the first and last options deal with the input and output files of the program.The `−e` inputs the current energy of the initial photon (e.g., $-e900$) to the program, and the `−f` option flags the program to erase the outputfile before writing new lines, so it should only be used for the first calling of `mean` for a given simulation at multiple energies.