UNIVERSITY OF HELSINKI
DEPARTMENT OF PHYSICS

SCIENTIFIC COMPUTING II

Final Project Report
Velocity Filter Simulation

Student: Caike Crepaldi

Professor: Antti Kuronen

December 2015

Contents

1 Introduction

This project deals with the problem of simulating the motion of a charged particle in a velocity selector (or Wien filter).

The program reads data from the command line and from an input file and uses these data in order to simulate the motion of particles of different configurations (different masses and charges). At the end of the simulation, the program prints the number of particles that got through the filter, its number identification, masses, charges, last velocity and position, in an output file.

1.1 About the situation

The velocity selector has a perpendicular electric and magnetic fields, where:

$$\vec{E} = E\vec{k} \tag{1}$$

$$\vec{B} = b\vec{i} \tag{2}$$

According to the theory, only the particles that have the velocity:

$$v = \frac{E}{B} \tag{3}$$

are capable of passing through the filter.
This is described in the figure 1.

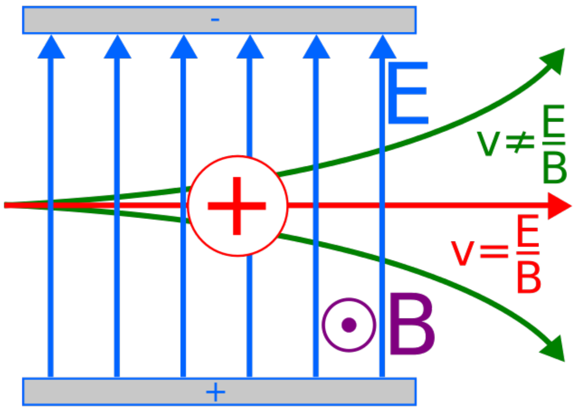

Figure 1: Magnetic and Electric fields in the velocity selector.

In the velocity selector, the particle is affected by the Lorentz force:

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{4}$$

In this project, only the influence of the Lorentz force is considered and the particle motion is limited to the Newtonian motion.

In this program, we use the following values as the dimension parameters of the selector box:

$$L_y = 7.6\ cm$$

$$L_x = L_y = 1.9\ cm$$

Where L_i is the length of the box in the i-axis.

The simulated particles start at the center of the left end of the box with an initial velocity set by the user.

2 Methods

The adopted method to do the simulation is called "Verlet algorithm" and allows the calculation of the values of position, velocity and acceleration of the particle at the next time step using the values from the current time step.

The Verlet algorithm is decribed as:

$$x_{i+1} = x_i + v_i\Delta t + \frac{1}{2}a_i\Delta t^2 \tag{5}$$

$$a_{i+1} = F(x_{i+1})/m \tag{6}$$

$$v_{i+1} = v_i + \frac{1}{2}(a_i + a_{i+1})\Delta t \quad (7)$$

Using this algorithm inside a do-loop, the program can, over the several iterations, verify if the particle got through the filter or if it hit the wall, thus stopping the simulation.

3 Implementation of the Methods

3.1 Program's Modularity

The program was written in Fortran 95/2003 and consists of 4 *.f90 files, 1 main.f90 file (with the proper program code) and other 3 external procedure files (with modules, subroutines and functions).

- The file vfs.f90 contains the module vfs with the subroutine print_help and some parameter values. This module holds the external procedures and parameters related to the inner functioning of the main program itself.
- The file lorentz.f90 contains the module lorentz with the function florentz and function cross. This module holds the external procedures related to the calculation of the forces that affect the particle motion (in our case, just the Lorentz force). One function (cross) evaluates the cross product between two 3-dimensional vectors. This function is later used inside the florentz function (present in the same module) in order to auxiliate the calculation of the Lorentz force acting in the particle.
- The file verlet.f90 contains the module verlet with the Verlet algorithm-based subroutine (advance_step) that evaluates the position, velocity and acceleration of the particle at the next time step.
- The file main.f90 holds the true program's code. It uses the 3 modules with their external functions, subroutines and parameters, in order to do the simulation and interact with the user's I/O.

In order to compile the program, use the use the make command in the ../src/ folder:

Command 1

```
$ make
$ make clean
```

The *make clean* command removes the *.o and *.mod files, leaving only the source codes (*.f90) and the vfsim program file.

See the Makefile or information regarding the compilation and linking commands.

Note: The Makefile contains compilation and linking commands that need the gfortran compiler. The user needs to have the GNU gfortran installed in order to compile and link the program using the Makefile.

3.2 Program's I/O

3.2.1 The Input

The program deals with 2 types of input: The command line arguments and the input.dat file.

When calling the program using the shell, the program needs 3 arguments related with the filter configuration in order to do the simulation:

Command 2

```
$ ./vfsim dt B E
```

Where:

- dt is the discrete time step (in seconds);
- B is the magnetic field strength in the x-axis (in T);
- E is the eletric field streght in the z-axis (in V/m);

If the program is called with the incorrect number of arguments, it will print information about the program and its usage (using the print_help subroutine).

Also, in the folder ../run/ there must be a file input.dat with the individual configuration of each particle (particle number identification, mass, charge, initial velocity in the three dimensions). This file must follow the format:

input_example.dat

```
npar
1 0 120000 0 -5 23
2 130 150000 78 -12 50
3 ...
.
.
.
n vx vy vz m q
```

Where:

- npar is the total number of particles to be simulated using the program;
- n is the particle identification number (label);
- vi (i=x,y,z) is the initial velocity of the particle in the i-axis (in m/s);
- m is the particle mass (in units of electron mass);
- q is the particle charge (in units of electron charge);

The program does the calculations in SI units, so the values of m and q are converted to kg and C, respectively.

3.2.2 The Output

The program's output can be read in the output.dat file, that shall be created after the program's successful execution in the ../run/ folder.

The output file will follow the format:

output_example.dat

```
n vx vy vz m q
.
.
.
count
```

Where:

- n is the identification number (label) of the particle that got through the filter;
- vi (i=x,y,z) is the final velocity of the successful particle in the i-axis (in m/s);
- m is the successful particle mass (in units of electron mass);
- q is the successful particle charge (in units of electron charge);
- count is the total number of sucessful particles;

3.3 The Verlet Algorithm Implementation

The Verlet algorithm implementation is done using 2 do-loops: one reads through each particle's configuration in the input file and the other iterates the Verlet algorithm, thus simulating the particle's motion in the filter. Inside the second do-loop, the program verifies if the particle got through the filter or hit the walls and then calls the advance_step subroutine, changing the values of position, velocity and acceleration.

See a small part of the main.f90 code where the Verlet algorithm is implemented and the true simulation happens:

main.f90

```fortran
! Do the simulation
    ****************************************************

count=0 ! counter of successful particles
! (particles that get to the other side of the filter)

particle_loop: do p=1,npart

   ! particle init. (reads data from the p-th line of the input file)
   read(42,*,iostat=ios),n,v_i,m,q
   if (ios/=0) then ! Houston, we have a problem!
      print '(a)',"Error reading the configuration of a particle."
      print '(a)',"Incorrect type! Please use real number."
      stop
   end if

   q=q*ec ! q is now the particle charge in Coulumb (SI)

   m=abs(m*me) ! m is the particle mass in kg (SI) (mass must be non-
      negative)

   x=[L(1)/2.0_rk,0.0_rk,L(3)/2.0_rk] ! initial position (center)

   v=v_i ! initial velocity

   a=florentz(q,E,B,v)/m ! calculates the first acceleration

   simulation_loop: do
      ! this is where the true simulation happens
      if ( ( abs(x(1)) >= L(1) ) .or. ( abs(x(3)) >= L(3) ) ) then
         write(6,'(a,x,i0,x,a)') "The particle", n, "hit the wall!"
         exit ! particle hits the wall of the filter
      else if (x(2)>=L(2)) then
         count=count+1 ! particle passes through the filter
         write(64,*) n,v,x,m,q
         write(64,*) new_line(arg)
         write(6,'(a,x,i0,x,a)') "The particle", n, "passed throught!"
         exit
      end if
      ! uses the verlet algorithm to update the values
      call advance_step(E,B,q,m,a,v,x,dt)
   end do simulation_loop
end do particle_loop
```

See the advance_step function with the Verlet algorithm in Fortran syntax:

verlet.f90

```
  pure subroutine advance_step(E,B,q,m,a,v,x,dt) ! verlet algorithm
    real(rk),intent(in),dimension(3) :: E,B
    real(rk),intent(in) :: dt,q,m
    real(rk),intent(inout),dimension(3) :: x,v,a
    real(rk),dimension(3) :: x0,a0,v_n

    x0=x ! stores old position

    x=x+(v*dt)+(a*(dt**2)) ! calculates new position

    a0=a ! stores old acceleration
    v_n=(x-x0)/dt ! uses x in order to get a new temporary v (v_n) to
       calculate
                  ! the new acceleration using the lorentz force
    a=florentz(q,E,B,v_n)/m ! calculates new acceleration with v_n

    v=v+0.5_rk*(a0+a)*dt ! calculates new velocity using both the new a
                         ! and the old a (a0)

  end subroutine advance_step
```

4 Results

The program receives 3 command line arguments, dt, B and E. All of those arguments must be integers. dt is the time step (Δt in the method's equations) in seconds, B is the magnetic field in the x-axis (in T) and E is the electric field in the z-axis (in V/m).

You can notice that almost all variables are in SI, the only exception is m and q that are read in units of electron mass/charge. But this is later converted to C and kg in the code.

All calculations in the code are done using SI variables so the output is, naturally, in SI units.

In order to allow the user to check the results in the shell, the program prints the initial configuration of the problem and warns the user if a particle hit the wall or passed through the filter.

Below is an example of the program's execution with $dt = 1e-10$, $B = 0.1$ and $E = 12000$. There are several values of initial velocity, mass and charge being tested using the input file. With the output file's help we can see which configuration allowed the particle to get through the filter.

Command 3

```
$ ./vfsim 1e-10 0.1 12000
 Problem Configuration:

L [m] =  0.019 0.076 0.019


 dt [s] =
   1.0000000000000000E-010


 B [T] =
  0.10000000000000001          0.0000000000000000          0.0000000000000000
 E [V/m] =
   0.0000000000000000        0.0000000000000000        12000.000000000000


 The particle 1 hit the wall!
 The particle 2 passed throught!
 The particle 3 passed throught!
 The particle 4 hit the wall!
 The particle 5 hit the wall!
 The particle 6 hit the wall!
 The particle 7 hit the wall!
 The particle 8 hit the wall!
 The particle 9 passed throught!
 The particle 10 passed throught!
 The particle 11 passed throught!
 The particle 12 passed throught!
 The particle 13 passed throught!
 The particle 14 passed throught!
 The particle 15 hit the wall!
 The particle 16 passed throught!
 The particle 17 passed throught!
 The particle 18 passed throught!
 The particle 19 passed throught!
```

input.dat

```
19
1 0 1 0 1 -1
2 0 1e3 0 3 0
3 0 1 0 1 0
4 0 12e4 0 1 -1
5 0 0 12e4 3 -1
6 22 1e2 15e4 10 -5
7 4e2 2e4 20e4 23 -10
8 0 0.5 10e4 40 -15
9 0 12e6 0 1e3 0
10 1e3 15e6 1e2 3e4 -10
11 1e3 12e4 0 1e2 -1
12 0 12e4 1e3 1e3 -1
13 0 12e4 0 1e4 -1
14 3.4e3 12e4 1e2 1e3 -25
15 0 12e4 22e2 1e2 -42
16 0 12e4 0 1e3 -13
17 0 12e4 0 1e3 42
18 42.666 12e4 0 1e3 25
19 42.666 13e4 0 1e3 25
```

output.dat

```
Output syntax:  n vx vy vz xx xy xz m q
           2   0.0000000000000000        1000.0000000000000
              0.0000000000000000        9.4999997876584526E-003
              7.6000000001155490E-002   9.4999997876584526E-003
              2.7328150336278240E-030   0.0000000000000000


           3   0.0000000000000000        1.0000000000000000
              0.0000000000000000        9.4999997876584526E-003
              7.5999998323637366E-002      9.4999997876584526E-003
              9.1093834454260801E-031      0.0000000000000000


           9     0.0000000000000000           12000000.000000000
              0.0000000000000000           9.4999997876584526E-003
              7.6800000000000063E-002      9.4999997876584526E-003
              9.1093834454260801E-028      0.0000000000000000


          10     1000.0000000000000           14993346.096252315
             444944.84125902667          9.5050997876584224E-003
             7.6488363656542935E-002     1.0657191091109745E-002
             2.7328150336278240E-026    -1.6021765974585869E-018


          11     1000.0000000000000           119999.99999987912
            -2.2012771458869792E-008      1.0133399787654691E-002
             7.6007999999999867E-002     9.4999997876591379E-003
             9.1093834454260801E-029    -1.6021765974585869E-019


          12     0.0000000000000000           120988.49424638534
             151.25857062230153          9.4999997876584526E-003
             7.6007793126834575E-002     9.4437552244521997E-003
             9.1093834454260801E-028    -1.6021765974585869E-019


          13     0.0000000000000000           120000.00000000000
            -8.9223396166144598E-009      9.4999997876584526E-003
             7.6007999999994927E-002     9.4999997876583225E-003
             9.1093834454260801E-027    -1.6021765974585869E-019


          14     3400.0000000000000           119923.24268685412
            -65.022050004829751           1.1653559787656651E-002
             7.6007620856617372E-002     9.5001660989656577E-003
             9.1093834454260801E-028    -4.0054414936464673E-018


          16     0.0000000000000000           119999.99999979952
            -3.1061608808528384E-007      9.4999997876584526E-003
             7.6007999999998632E-002     9.4999997876593148E-003
             9.1093834454260801E-028    -2.0828295766961630E-018


          17     0.0000000000000000           120000.00000088850
             6.0870211973586279E-007     9.4999997876584526E-003
             7.6007999999999215E-002     9.4999997876596860E-003
             9.1093834454260801E-028     6.7291417093260651E-018


          18     42.665999999999997           120000.00000160796
```

```
                      6.5525270642738664E-007     9.5270244320608943E-003
                      7.6007999999998591E-002     9.4999997876621424E-003
                      9.1093834454260801E-028     4.0054414936464673E-018


                 19      42.665999999999997            114197.65711217311
                      -8217.5396885076716            9.5270158988608935E-003
                      7.6001898367075299E-002     9.4636499963278321E-003
                      9.1093834454260801E-028     4.0054414936464673E-018


12
```

There are spaces in between the result of each particles in order to allow the user to read the output more easily. This can be removed in the source code together with the first line (output syntax) if the user wants to read the values with another program or use the output file as input in another program.

We can compare the values of mass and charge of the particles in both files and see the that they were converted to SI units.

Using the same system configuration as the previous example, let's try to see the effect that mass has over the particles that pass through:

input_mass.dat

```
13
1 0 1 0 25 -5
2 0 10 0 25 -5
3 0 1e3 0 25 -5
4 0 1e4 0 25 -5
5 0 1e5 0 25 -5
6 0 12e4 0 25 -5
7 0 14e4 0 25 -5
8 0 18e4 0 25 -5
9 0 1e6 0 25 -5
10 0 1e7 0 25 -5
11 0 1e8 0 25 -5
12 0 1e8 0 1e2 -5
13 0 1e8 0 1e3 -5
```

output_mass.dat

```
Output syntax:   n vx vy vz xx xy xz m q
                  6      0.0000000000000000            35667.482788352034
                       77222.515120530705            9.4999997876584526E-003
                       7.6001507597207252E-002     9.5281575448111465E-003
                       2.2773458613565200E-029    -8.0108829872929346E-019


                 13      0.0000000000000000            99752909.544611663
                      7021251.6241407385           9.4999997876584526E-003
                      7.9924316953263561E-002     1.2660804945438543E-002
                      9.1093834454260801E-028    -8.0108829872929346E-019


2
```

We can see that, under normal conditions, only the right velocity (12×10^4 m/s) pass through the filter. However, if we increase the mass a lot, even a velocity really far from the right one (10^8 m/s) can pass through the filter.

5 Conclusions

The Fortran implementation of the problem was done successfully. The I/O features of Fortran allows us to deal easily with command line data and with file data.

We can then conclude, with the results presented in this report, that the charged particles that pass through the filter have their velocity determined by the equation 3. Also, if the particle is not charged ($q = 0$), it will always pass through the filter, as expected. However, by increasing the mass of a charged particle, we can force it pass through the filter with a velocity far from the right one.