Simulation of particles in Earth's magnetic field

C++ code summary

The main function

3 operations:

- Make objects for the telescope(position), and for the particles.
- Distribute accordingly the particles(initial population state).
- Simulate with or without wave interaction.

1. MAKE OBJECTS

Using the structs for the Telescope and the Particles:

```
//Position of the Particle Telescope
Telescope ODPT(Constants::telescope_lamda, Constants::L_shell);
//Single particle struct
Particles single;
//Vector of structs for particle distribution
std::vector<Particles> eql_dstr(Constants::test_pop, single);
```

STRUCTS USED

Telescope	Particles

`struct Telescope`
`{`
`Telescope(real lat, real L_parameter);`
`//Constructor. Initialize position of satellite.`
`bool crossing(real p1_lamda, real p2_lamda, real p_L_shell); //True if particle crosses`
`//Function to push back detected particles.`
`void store(int id, real lamda, real alpha, real aeq, real time);`
`//Satellite's position parameters`
`real L_shell, latitude;`
`//Vectors to store detected particles.`
`std::vector<real> lamda , uper , upar, alpha, aeq, eta, time;`
`std::vector<int> id; };`
	`struct Particles`
	`{`
	`//Member function to initialize particle population.`
	`void initialize(real eta0, real aeq0, real alpha0, real lamda0, real Ekev0, real Blam0, real zeta0, real deta_dt0, real time0);`
	`//Member function to push_back new state if needed.`
	`void save_state(real new_aeq, real new_alpha, real new_lamda, real new_deta_dt, real new_time);`
	`//Member variables.`
	`std::vector<real> lamda , zeta, uper , upar, ppar, pper, alpha, aeq, eta, M_adiabatic, deta_dt, Ekin, time;`
	`};`

2. DISTRIBUTE PARTICLES (Latitude, eq. P.A,eta)

- Here, we can distribute the particles in many ways, in Energy, gyrophase, latitude etc.

```
//Declare some variables
real eta0,aeq0,lamda0;
real Blam0,salpha0,alpha0;
//Bmag_dipole in functions.h
real Beq0 = Bmag_dipole(0);
```

```
for(int e=0, p=0; e<Constants::eta_dstr; e++)
{
	eta0 = (Constants::eta_start_d + e*Constants::eta_step_d)
* Constants::D2R;
	for(int a=0; a<Constants::aeq_dstr; a++)
	{
		aeq0 = (Constants::aeq_start_d + a*Constants::aeq_step_d)
	* Constants::D2R;
		for(int l=0; l<Constants::lamda_dstr; l++, p++)
		{
			lamda0 = (Constants::lamda_start_d +
		l*Constants::lamda_step_d) * Constants::D2R;
			//Find P.A at lamda0.
			Blam0=Bmag_dipole(lamda0);
			salpha0=sin(aeq0)*sqrt(Blam0/Beq0);
			if( (salpha0<-1) || (salpha0>1) || (salpha0==0)
		)
			//Exclude these particles.
			{ eql_dstr.pop_back(); p--; continue; }
			eql_dstr[p].initialize(eta0,aeq0,alpha0,lamda0,
		Constants::Ekev0,Blam0,0,0,0);
		}
	}
}

//Population of particles that will be tracked. Some particles were
excluded from the initial population due to domain issues.
		int64_t track_pop = eql_dstr.size();
```

3. SIMULATE(Wave particle interaction, or adiabatic motion)

- At this point we can choose whether the particles will bounce without interacting with a wave. There are 2 codes for wave interaction. One of them uses Ray tracing.
- Also, with an OpenMP Work sharing implementation, user can decide how many threads can execute the program.
- Just including the right function inside the for loop, will do the desirable job.

```
//Parallelism Work sharing
int realthreads;
double wtime = omp_get_wtime();
#pragma omp parallel
{
    int id = omp_get_thread_num();
   if(id==0){ realthreads = omp_get_num_threads();}
   #pragma omp for schedule(static)
//Void Function for particle's motion. Involves RK4 for Nsteps.
//Detected particles are saved in ODPT object, which is passed
here by reference.
  for(int p=0; p<track_pop; p++)
        {
             wpi(p, eql_dstr[p], ODPT); //--change this
function--//
        }
}
              //--With any of these 3 functions--//
```

NoWPI	WPI where wave is everywhere	WPI when there's Ray Tracing

- Each one of these 3 functions include a 4^{th} order Runge Kutta method(RK4)
- Within this method, Bell's or Li's equations are used to estimate the position, the speed and other time and spatial rates of the particle.
- For Ray Tracing, there is a need to interpolate the ray to fit in the simulation's time scales. This is done outside of the main algorithm, either in python or C++ and saved as a h5 file. Importing this file, means we have the Ray in the desirable time scales.

Last part would be to make an h5 file output for further processing and plotting.

Simulation of particles in Earth's magnetic field

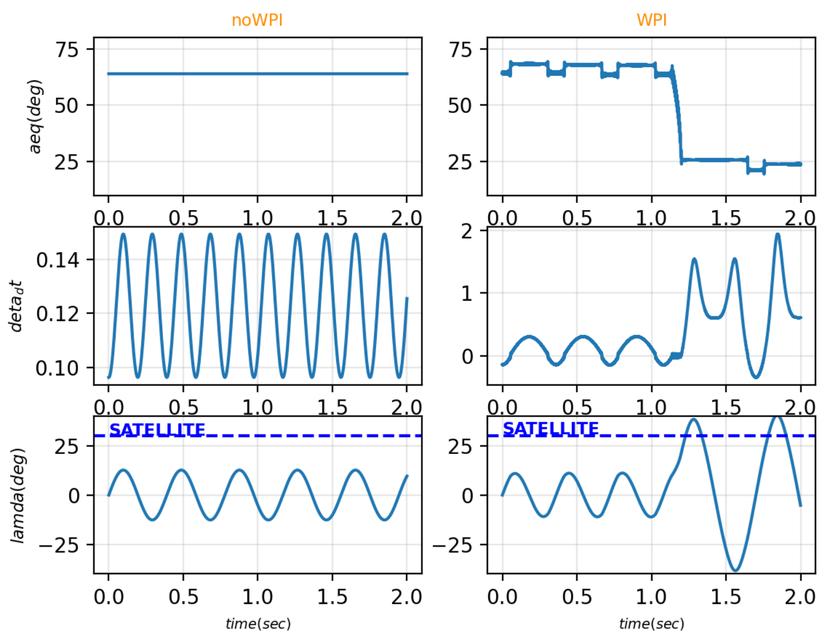
Matplotlib plots

Comparing plots between WPI and noWPI.

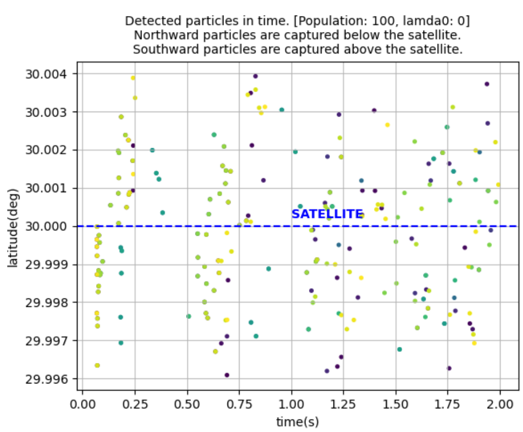
Plots on the left are for a single-random particle that happens to oscillate around -20deg and 20 deg

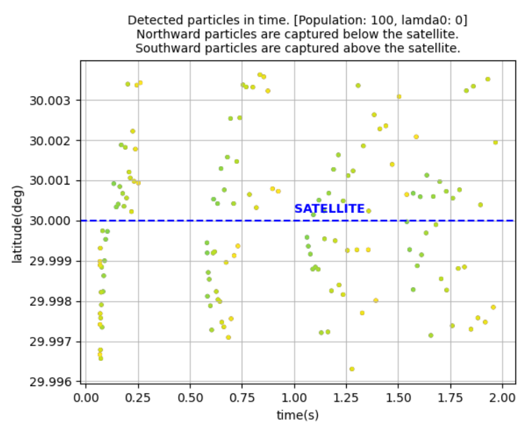
Plots on the right are for a single-random particle that happens to oscillate and achieve resonance with the wave. This particle would not be detected by the satellite if there was noWPI.

Equatorial pitch angle(deg) and time rate of eta is also shown.

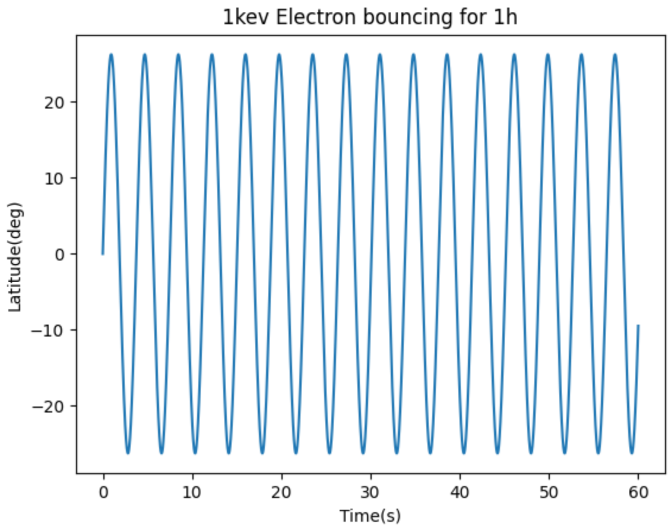


WPI (some new particles are detected)





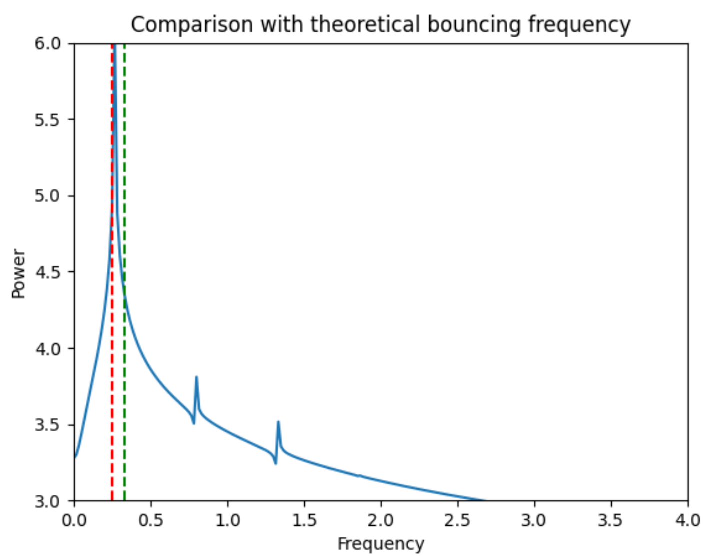
1kev electron oscillating without interacting with a wave, for 1 hour between –25 and 25 deg.



The Fast Fourier Transform can give us the oscillation frequency.

(Red-dotted line) The theoretical [Orlova1,Shprits2,2011] oscillation frequency

(Green-dotted line) Using the calculator https://solenelejosne.com/bounce/



Scalability for multiple threads.