Bolt Documentation

Release 1.0

Shyam Sundar Sankaran, Mani Chandra

May 02, 2021

Contents:

CHAPTER 1

About Bolt:

`Bolt` is a flexible framework for solving kinetic theory formulations, making use of the finite volume and/or advective semi-lagrangian method. Additionally, it also consists of a linear solver which is primarily used in verifying the results given by the nonlinear solver. The code is open-source and developed by the research division at Quazar Technologies, Delhi where it used to study device physics and astrophysical plasmas

The code is written in Python and features an easy-to-use interface, where the user provides input through a `physical system` object which holds details about the system solved. The `physical_system` object is declared by defining the advection terms, and the source term for the system of interest. Additionally, the physical simulation also requires details such as the domain information, and initial conditions.

`Bolt` is capable of running on CPUs and GPUs, and has been parallelized to be able to run efficiently across several nodes/devices.

CHAPTER 2

Doc Contents

2.1 Home

2.1.1 Overview

What is Bolt?

`Bolt` is an open-source Python based framework for solving kinetic theory formulations uptil 5-dimensional phase space on a range of devices using the finite volume method, and/or the advective Semi-Lagrangian approach originally proposed by Cheng&Knorr. The framework is designed to solve a range of physical systems where the domain of interest can be mapped on to a rectangular grid. It is designed to target a range of hardware platforms via use of the ArrayFire library, and is completely parallelized to run on large clusters by use of the PETSc library.

The current release has the following capabilities:

- Dimensionality - Upto 2D3V phase space dimensionality
- Interpolation Methods - Linear, Cubic Spline
- Reconstruction Methods - minmod, PPM, WENO5
- Riemann Solvers - 1st Order Upwind flux, Local Lax-Friedrics flux
- Platforms - CPUs, OpenCL Devices, CUDA Devices
- Temporal Discretisation:
  - Time Splitting: Strang, Lie, SWSS
  - Time Stepping : Explicit - RK2, RK4, RK6
- Precision - Double
- Solution Files Exported - HDF5 (.h5, .hdf5)

2.2 Theory

In `Bolt`, the defined system is evolved by stepping the complete probability distribution function, from which physical parameters of interest about the system may be obtained by coarse-graining the system via use of the `compute_moments` method. The implementation allows us to deal with a wide range of boundary conditions with ease, in addition to capturing the dynamics of short-range interactions via a collision operator(input as a source). `Bolt` is capable of performing accurate simulations of systems that are governed by the form:

$$\frac{\partial f}{\partial t} + A_{q1}\frac{\partial f}{\partial q_1} + A_{q2}\frac{\partial f}{\partial q_2} + A_{p1}\frac{\partial f}{\partial p_1} + A_{p2}\frac{\partial f}{\partial p_2} + A_{p3}\frac{\partial f}{\partial p_3} = S(f)$$

`Bolt` can make use of the finite-volume method and/or the non-conservative semi-lagrangian method.

2.2.1 Finite Volume Method

To explore this method in detail, we'll first need to define the generalized conservative equations:

$$\frac{\partial f}{\partial t} + \frac{\partial (C_{q1} f)}{\partial q_1} + \frac{\partial (C_{q2} f)}{\partial q_2} + \frac{\partial (C_{p1} f)}{\partial p_1} + \frac{\partial (C_{p2} f)}{\partial p_2} + \frac{\partial (C_{p3} f)}{\partial p_3} = S(f)$$

The conservative equations are multiplied by the volume element of a discrete grid zone in phase space $\Delta v = dq_1 dq_2 dp_1 dp_2 dp_3$, and using the divergence theorem gives use the finite volume formulation:

$$\partial_t \bar{f} + \frac{\bar{F}_{q1}^{q-right} - \bar{F}_{q1}^{q-left}}{\Delta dq1} + \frac{\bar{F}_{q2}^{q-top} - \bar{F}_{q2}^{q-bottom}}{\Delta dq2} + \frac{\bar{F}_{p1}^{p-right} - \bar{F}_{p1}^{p-left}}{\Delta dp1} + \frac{\bar{F}_{p2}^{p-top} - \bar{F}_{p2}^{p-bottom}}{\Delta dp2} + \frac{\bar{F}_{p3}^{p-front} - \bar{F}_{p3}^{p-back}}{\Delta dp3} = \bar{S}$$

Where $\bar{f} = (\int f \Delta v)/\int \Delta v$, $\bar{S} = (\int S \Delta v)/\int \Delta v$, $\bar{F}_{q1} = (\int f C_{q1} dq_2 dp_1 dp_2 dp_3)/\int dq_2 dp_1 dp_2 dp_3$, $\bar{F}_{q2} = (\int f C_{q2} dq_1 dp_1 dp_2 dp_3)/\int dq_2 dp_1 dp_2 dp_3$, $\bar{F}_{p1} = (\int f C_{p1} dq_1 dq_2 dp_2 dp_3)/\int dq_1 dq_2 dp_2 dp_3$, $\bar{F}_{p2} = (\int f C_{p2} dq_1 dq_2 dp_1 dp_3)/\int dq_1 dq_2 dp_1 dp_3$, $\bar{F}_{p3} = (\int f C_{p3} dq_1 dq_2 dp_1 dp_2)/\int dq_1 dq_2 dp_1 dp_2$.

The locations right, left, top, bottom, front, and back are mentioned whether they are considered in q-space or p-space since we are considering 5-dimensional phase space here. The appropriate value at the boundaries is obtained by using a reconstruction operator which takes in the cell-centered values and constructs a polynomial interpolant from which the edge states can be computed. The time derivative term $\partial_t \bar{f}$ is then passed to an appropriate integrator to evolve the system in time.

2.2.2 Semi-Lagrangian Method

In this approach, a probability distribution function is given a grid-based Eulerian representation which is then evolved via Lagrangian dynamics. The CFL time step restriction of a regular finite difference or finite volume scheme is removed in a semi-Lagrangian framework, allowing for a cheaper and more flexible numerical realization. However, the downside is that the method is non-conservative

A detailed overview of the advective semi-Lagrangian method is given in:

- *The integration of the Vlasov equation in configuration space.* Cheng, C. Z., & Knorr, G. (1976). Journal of Computational Physics, 22(3), 330-351.<http://www.sciencedirect.com/science/article/pii/002199917690053X>

2.3 Installation

2.3.1 Downloading the Source

`Bolt` is distributed using the git version control system, and is hosted on Github. The repository can be cloned using:

```
git clone https://github.com/QuazarTech/Bolt.git
```

2.3.2 Dependencies

Overview

`Bolt` has a hard dependency on Python 3+ and the following Python packages:

1. mpi4py
2. numpy
3. h5py
4. pytest
5. scipy
6. matplotlib
7. petsc4py
8. arrayfire

Before installing the above python packages, the following libraries need to be installed so that their python wrappers can function:

Building ArrayFire

- Clone the arrayfire repository
- Build using the instructions that have been provided here

Building PETSc

- Clone the petsc repository
- We suggest that you install PETSc using the following:

```
./configure --prefix=/path/to/petsc_installation/ --with-debugging=0 COPTFLAGS="-
↪O3 -march=native" CXXOPTFLAG S="-O3 -march=native" --with-hdf5=1 --download-
↪hdf5 --with-clean=1 --with-memalign=64 --known-level1-dcache-size=32768 --known-
↪level1-dcache-linesize=64 --known-level1-dcache-assoc=8 --with-hypre=1 --
↪download-mpich=1 --with-64-bit-indices
```

- If you are keen on modifying the above build parameters, detailed instructions for the same may be found here

Below are instructions for building the PETSc stack on a few machines that we've tested on:

- On BRC HPC Savio:

```
python2 './configure' '--with-debugging=0' 'COPTFLAGS=-O3 -qopt-report=5 -qopt-
↪report-phase=vec -xhost' 'CXXOPTFLAGS=-O3 -qopt-report=5 -qopt-report-phase=vec␣
↪-xhost' '--with-hdf5=1' '--with-clean=1' '--with-mpi-dir=/global/software/sl-6.
↪x86_64/modules/intel/2016.1.150/openmpi/1.10.2-intel/' '--with-blas-lapack-dir=/
↪global/software/sl-6.x86_64/modules/langs/intel/2016.1.150/mkl/lib/intel64' '--
↪with-memalign=64' '--known-level1-dcache-size=32768' --known-level1-dcache-
↪linesize=64' 'known-level1-dcache-assoc=8' '--with-hypre=1' '--download-hypre=1
↪' '--with-64-bit-indices'
```

2.3.3 Installation

Before running `Bolt` it is first necessary to either install the software using the provided `setup.py` installer(TODO) or add the root directory to `PYTHONPATH` using:

```
user@computer ~/Bolt$ export PYTHONPATH=.:$PYTHONPATH
```

Once the build of ArrayFire and PETSc is completed install the python dependencies using:

```
user@computer ~/Bolt$ pip install -r requirements.txt
```

2.4 Getting started with Bolt

2.4.1 Overview

`Bolt` is organized such that a system is defined by making use of the `physical_system` class. The object created by `physical_system` is then passed as an argument to the solver objects.

Physical System

An instance of the `physical_system` object may be initialized by using:

```
system = physical_system(domain,
                         boundary_conditions,
                         params,
                         initialize,
                         advection_terms,
                         source,
                         moments
                        )
```

The arguments in the above command are all python modules/functions, where the details regarding the system being solved have been provided.

A detailed breakdown of what is to be contained in these files is demonstrated in the tutorials.

Solvers

The solver objects may be declared by using:

```
nls = nonlinear_solver(system)
ls  = linear_solver(system)
```

The physical system defined is then evolved using the various time-stepping methods available under each solver:

```
for time_index, t0 in enumerate(time_array):
  print('Computing For Time =', t0)
  nls.strang_timestep(dt)
  ls.RK2_step(dt)
```

The abstracted information about the system may be obtained by using the `compute_moments` method available under each solver:

```
density_nls = nls.compute_moments('density')
density_ls  = ls.compute_moments('density')
```

The data about the evolved system can be dumped to file by making use of the methods `dump_distribution_function`,‘‘dump_moments‘‘ and `dump_EM_fields`

Running in Parallel

`Bolt` can be run in parallel across multiple nodes. To do so prefix the python command being executed with `mpirun -n <nodes/devices>`. Make sure that `num_devices` is set correctly under `params` when running on nodes which contain more than a single accelerator(NOTE: The parallelization has only been implemented for the nonlinear solver. The linear solver can only take advantage of shared memory parallelism)

2.4.2 Tutorial Notebook

This notebook covers the basics of setting up and interacting with the primary features of the code. We consider the example problem of a 1D1V setup of the non-relativistic Boltzmann equation in which we observe the damping of the density with time.

2.4.3 Example Scripts

A wide range of examples are available under the `example_problems` subdirectory of the main code repository, which you can browse here.

These examples cover a wider range of use cases, including larger multidimensional problems designed for parallel execution. Most folders also have a `README` which gives context for the case that has been setup. Basic post-processing and plotting scripts are also provided with many problems.

These simulation and processing scripts may be useful as a starting point for implementing different problems and equation sets.

2.5 Units

`Bolt` handles dimensionless quantities normalized using some reference quantities. It’s to be ensured that all input quantities are normalized appropriately when passing to `physical_system`. This is to be done under the parameter and domain files under the respective problem folder.

Let us now illustrate this choice of normalization that can be adapted, when dealing with a purely collisonless case with electrostatic fields. Note that this is just one of the possible ways in which the independant units can be arrived at:

The equations governing by the system under consideration is given by:

$$\begin{aligned}\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} + \frac{qE}{m}\frac{\partial f}{\partial v} &= 0\\ E &= -\frac{\partial \phi}{\partial x}\\ \nabla^2\phi = -\frac{\rho}{\epsilon_0} &= -\frac{ne}{\epsilon_0}\end{aligned}$$

- As we'd stated earlier, we have a choice of declaring a few variables with respect to absolute reference quantities(independant quantities), and find that the remaining dependant quantities can be expressed in terms of these. For this example, we'll choose appropriate reference units for time t, velocity v, charge e and mass m.
- When dealing with a plasma at constant mean density n_e , it is convenient to normalize times by introducing the electron plasma frequency $\omega_{pe} = \sqrt{\frac{ne^2}{m\epsilon_0}}$. Then all times are normalized using ${\omega_{pe}}^{-1}$. For the sake of convenience, let's call this normalization factor t_0 (where $t_0 = {\omega_{pe}}^{-1}$):

$$t = t_0\bar{t}$$

Similarly, we introduce the scaling factor v_0 for the velocity. So the velocity can be expressed as:

$$v = v_0\bar{v}$$

- Expressing the charge and the mass interms of our reference units e_0 and m_0 which are typically taken as the electron charge and mass:

$$\begin{aligned}e &= e_0\bar{e}\\ m &= m_0\bar{m}\end{aligned}$$

Now substituting these back into Vlasov-Boltzmann equation, we get:

$$\begin{aligned}\frac{1}{t_0}\frac{\partial f}{\partial \bar{t}} + v_0\bar{v}\frac{\partial f}{\partial x} + \frac{e_0}{m_0 v_0}\frac{\bar{q}E}{\bar{m}}\frac{\partial f}{\partial \bar{v}} &= 0\\ \implies \frac{\partial f}{\partial \bar{t}} + v_0 t_0\bar{v}\frac{\partial f}{\partial x} + \frac{e_0 t_0}{m_0 v_0}\frac{\bar{q}E}{\bar{m}}\frac{\partial f}{\partial \bar{v}} &= 0\\ \implies \frac{\partial f}{\partial \bar{t}} + \bar{v}\frac{\partial f}{\partial (\frac{x}{v_0 t_0})} + \frac{\bar{q}}{\bar{m}}\frac{E}{(\frac{m_0 v_0}{e_0 t_0})}\frac{\partial f}{\partial \bar{v}} &= 0\\ \implies \frac{\partial f}{\partial \bar{t}} + \bar{v}\frac{\partial f}{\partial \bar{x}} + \frac{\bar{q}\bar{E}}{\bar{m}}\frac{\partial f}{\partial \bar{v}} &= 0\end{aligned}$$

Thus, we find that the normalization constant for the distance x and electric field E come out in terms of the independantly chosen references:

$$\begin{aligned}x &= x_0\bar{x}; where\ x_0 = v_0 t_0\\ E &= E_0\bar{E}; where\ E_0 = \frac{m_0 v_0}{e_0 t_0}\end{aligned}$$

Now, let's take a look at the appropriate normalizations that need to be applied for the electric potential:

$$\begin{aligned}E &= -\frac{\partial \phi}{\partial x}\\ \implies E_0\bar{E} &= -\frac{1}{x_0}\frac{\partial \phi}{\partial \bar{x}}\\ \implies \bar{E} &= -\frac{\partial (\frac{\phi}{E_0 x_0})}{\partial \bar{x}}\\ \implies \bar{E} &= -\frac{\partial \bar{\phi}}{\partial \bar{x}}\end{aligned}$$

Hence, we get $\phi_0 = E_0 x_0 = \frac{m_0 v_0^2}{e_0 t_0}$

The table below gives a list of the normalizations we had used in this case, clearly distinguishing between the dependant and the independant quantites:

Independant Quantities:

Physical Quantity	Reference Unit
Time	t_0
Velocity	v_0
Charge	e_0
Mass	m_0

Dependant Quantities:

Physical Quantity	Reference Unit
Distance	$v_0 t_0$
Electric Field	$\frac{m_0 v_0}{e_0 t_0}$
Electric Potential	$\frac{m_0 v_0^2}{e_0 t_0}$

2.5.1 Exploration of the Plasma Scales:

In this section, we indent to explore the different scales that can exist in plasmas. We hope to elaborate the different time scales, length scales and velocity scales from which the scale that resolves the system under consideration appropriately can be chosen. Derivations have been performed wherever appropriate to provide context:

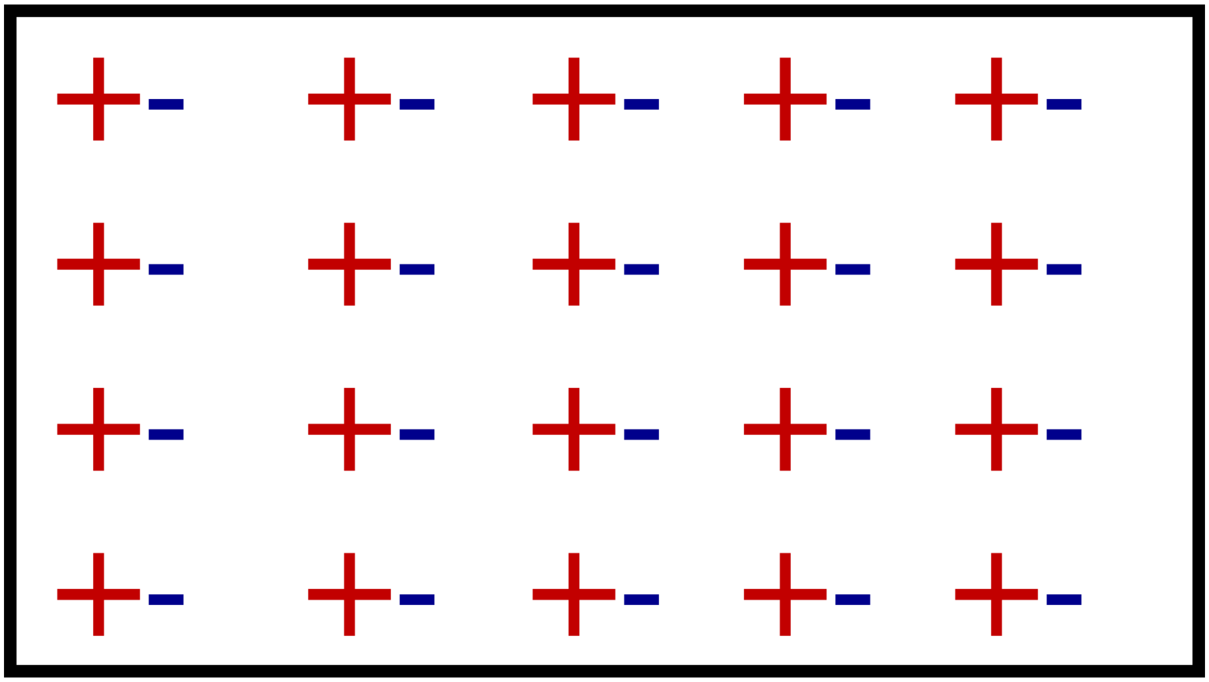
Temporal Scales:

The following timescales can be taken when dealing with a plasma:

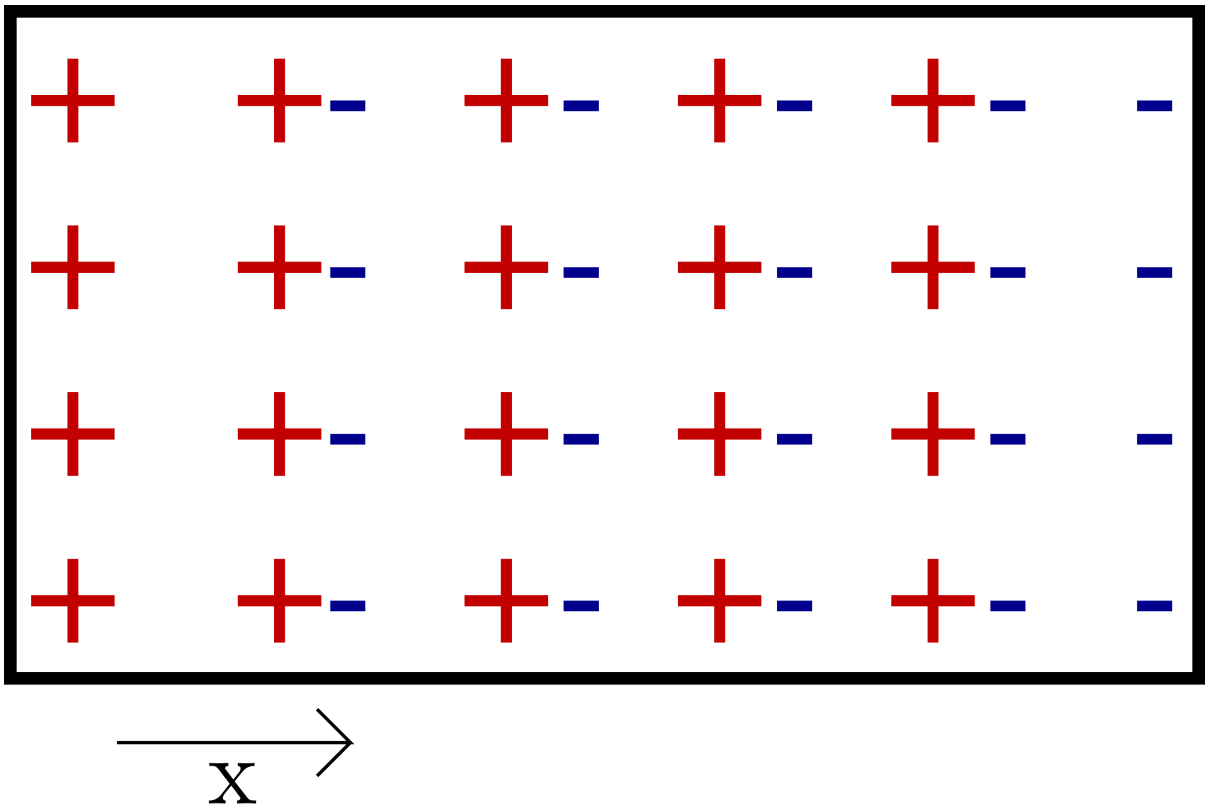
- Plasma Frequency

If one displaces by a group of charged particles from an electrically neutral plasma, the Coulomb force pulls the electrons back which results in a simple harmonic oscillation given by the plasma frequency. Below we derive this frequency.

Let us start by considering a charge neutral plasma where the positive charges and negative charges are next to each other.



Now if we move the negative charges by x, then we will end up with the following:



Thus, now there is a slab of positive and negative charges which would be exerting a field. Let us consider the field created by the positive slab of charges:

By Gauss' Law:

$$\int \vec{E} \cdot d\vec{A} = \frac{q}{\epsilon_0}$$

Now the charges are given by the number density multiplied by the volume of the segment, which can be expressed in terms of the area A and displacement x

$$q = neAx$$
$$\implies \int \vec{E} d\vec{A} = \frac{neAx}{\epsilon_0}$$
$$\implies E = \frac{nex}{\epsilon_0}$$

The force acting on an electron would be:

$$F = m_e a = -eE$$
$$\implies a = -\frac{ne^2}{m_e \epsilon_0} x = -\omega^2 x$$
$$\implies \omega = \sqrt{\frac{ne^2}{m_e \epsilon_0}}$$

- Gyrofrequency

Since the force acting on a charged particle in a magnetic field is always perpendicular to the direction of motion, the particle executes circular motion. The gyrofrequency is the angular frequency of thus circular motion of the charged particle in the plane perpendicular to the magnetic field. In the section below on length scales, we derive the gyroradius which we'll be using in obtaining the gyrofrequency. With the radius of gyration, we can calculate the time period of the motion executed, from which the angular frequency can be obtained:

$$T = \frac{2\pi r}{v_\perp}$$
$$\omega = \frac{2\pi}{T} = \frac{v_\perp}{r}$$
$$\implies \omega = \frac{qB}{m}$$

- The Alfvén time

An Alfvén wave in a plasma is a low-frequency travelling oscillation of the ions and the magnetic field

The Alfven time τ_A characteri is an important timescale for wave phenomena, and characterizes the timescale for this wave. It is related to the Alfvén velocity(which we derive in the section below) by

$$\tau_A = \frac{a}{v_A}$$

Where a is the characteristic length scale of the system in consideration.

Length Scales:

The following length scales can exist in a plasma:

- Thermal deBroglie Wavelength

$$\lambda = \frac{h}{p}$$

where h is the planck constant, and p is the momentum of the particle

The relation between the momentum and kinetic energy is given by:

$$E_k = \frac{p^2}{2m}$$

The effective kinetic energy derived with the statistics of Fermi gas is given as $E_k = \pi k_B T$. Hence, we get the thermal deBroglie wavelength as

$$\lambda = \frac{h}{\sqrt{2m\pi k_B T}}$$

- Classical Distance of Closest Approach

The potential energy possessed by 2 particles of charge e_1 and e_2 separated by distance r is given by:

$$F = \frac{1}{4\pi\epsilon}\frac{e_1 e_2}{r}$$

Now, this energy is to be balanced by the thermal energy of the plasma $E_{thermal} = kT$. Hence at the distance of closest approach:

$$\frac{1}{4\pi\epsilon}\frac{e_1 e_2}{r} = kT \implies r = \frac{1}{4\pi\epsilon}\frac{e_1 e_2}{kT}$$

- Gyroradius

This is the radius of the circle in which the charge particle oscillates when subjected to a magnetic field. sThe force on a moving charged particle in a magnetic field is given by the Lorentz force:

$$\vec{F} = e(\vec{v} \times \vec{B})$$

The force would always act perpendicular to the direction of motion, and would hence cause the particle to move in a circle in the plane perpendicular to the magnetic field. Equating this force to the centripetal force, we get:

$$\frac{mv_\perp^2}{r} = qv_\perp B$$
$$\implies r = \frac{mv_\perp}{qB}$$

- Debye Length

The plasma Debye length λ_D is the characteristic distance over which electrostatic potentials are screened out or attenuated by a redistribution of the charged particles. A charge in a plasma will attract opposite charges and repel like charges to the point that its electric eld is screened by the charges it has attracted, so particles outside the screening charges are unaware of the presence of the interior charge.

For this derivation, it is assumed that the ions and electrons have the same temperature T and number density n prior to the addition of another, positive, point charge. The charge of ions will be e andthe charge of electrons will be -

Thus, we have a plasma with temperature T and number density n, and we add a positive point charge. The particles will move around until they reach thermal equilibrium, at which point their probability of being in astate of energy U is proportional to the Boltzmann factor

$$P(U) \propto e^{-\frac{U}{kT}}$$

Now, the potential energy of a single particle from the new charge is $U = eV$,so the distribution function is given by

$$f(v) = n_0\sqrt{\frac{m}{2\pi kT}}e^{-frac\frac{mv^2}{2}+eVkT}$$

The integralof the distribution function is the total particle number density, so we have:

$$\int_{-\infty}^{\infty} f(v) = n = n_0 e^{-\frac{eV}{kT}}$$

Now that we have the number density, we can get the charge density via $\rho = n_i e + n_e(-e)$.

NOT TOO CLEAR ABOUT THE FOLLOWING SECTION

$$\rho = n_i e + n_e(-e) = e(n_i - n_e) = en_0(e^{-\frac{eV}{kT}} - e^{\frac{eV}{kT}}) = -en_0 \sinh(\frac{eV}{kT})$$

This allows us to write down Poisson's equation

$$\nabla^2 V = -\frac{\rho}{\epsilon} = \frac{en_0 \sinh(\frac{eV}{kT})}{\epsilon}$$

Since $eV << kT$, we can expand the RHS using Taylor series:

$$\nabla^2 V = \frac{en_0}{\epsilon}\frac{eV}{kT} = \frac{n_0 e^2}{\epsilon kT} V$$

Expressing this as:

$$\nabla^2 V = \frac{V}{\lambda_D^2}; where\ \lambda_D = \frac{\epsilon kT}{n_0 e^2}$$

This equation has the solution:

$$V = V_0 e^{-\frac{x}{\lambda_D}}$$

From this form of the solution it is clear what the physical meaning of λ_D is. Inside of λ_D, charges feel the potential due to the central charge. Outside of this Debye length, the potential falls o exponentially, and charges are no longer aware of the presence of the central charge. The charge is, eectively, screened by the surrounding charges.

Reference

- Plasma Skin Depth

The plasma skin depth is the depth in a collisionless plasma to which low-frequency electromagnetic radiation can penetrate (as defined by attenuation of the wave amplitude by a factor of $1/e$)

In a traditional plasma, the expression for plasma skin depth is given by $l_s = \frac{c}{\omega_p}$ where c is the speed of light in vacuum.

$$l_{debye} = c\sqrt{\frac{\epsilon m}{ne^2}}$$

Reference

Velocity Scales:

- Thermal Velocity

Equating the kinetic energy and thermal energy of the plasma, we obtain the thermal velocity which we use for scaling the velocity terms

$$\frac{1}{2} m{v_0}^2 = \frac{1}{2} kT \implies v_0 = \sqrt{\frac{kT}{m}}$$

- Sound Velocity:

$$c_s = \sqrt{\gamma RT}$$

- Alfven Velocity

NOT TOO CLEAR ON THIS. WikiPage derivation seems strange.

Alfvén waves are a fundamental physical phenomenon in all kinds of magnetized plasmas. Alfvén waves contribute to a variety of physical processes in space plasmas.

In plamas dominated by Alfvén waves, tension is due to the magnetic field. The plasma behaves like air except it is affected by magnetic fields. The dynamics are dominated by the energy density and pressure of the magnetic field. In this case, the appropriate sound speed is the Alfven speed

$$v_a = \sqrt{\frac{B^2}{\rho\mu_0}}$$

Reference

2.6 Quick-Reference:

`bolt.lib.physical_system`	
`bolt.lib.linear.linear_solver`	This is the module which contains the functions of the linear solver of Bolt.
`bolt.lib.nonlinear.nonlinear_solver`	This is the module where the main solver object for the nonlinear solver of bolt is defined.

CHAPTER 3

Other Links

Learn more about Bolt by visiting the

- Code repository: http://github.com/QuazarTech/Bolt
- Documentation: http://qbolt.rtfd.io