NCSI:Basic

1.0

Generated by Doxygen 1.8.5

Wed Feb 5 2014 12:49:45

Contents

1 Non-canonical Symplectic Integrator: Basic

Minimalist implementation of guiding center trajectory calculation. A fourth-order Runge-Kutta and noncanonical symplectic algorithm are present for the calculation of charged particle guiding center trajectories in electric and magnetic fields. An axisymmetric tokamak field is used for demonstration.

Installation

The NCSI:Basic code uses the `BOOST::program_options` library for option specification and the `Eigen` library for linear algebra tasks. Eigen is a header-only library available at `http://eigen.tuxfamily.org/` . The compiler needs to know where to find the header files, so placing the Eigen package somewhere like /usr/local/include is a good idea.

Usage

After compiling and linking the executable "driver", see a summary of runtime options using driver –help

The options may be specified on the command line with –option or -O. Alternatively, one may specify input options using an input file. See sample_input.cfg for an example. Units are discussed in the documentation.

Upgrade to PRO Today!

Full version of NCSI includes:

- Additional ODE systems (oscillators, magnetic field line flow)
- Additional integrators (linear multistep methods, implicit midpoint)
- Improved class hierarchy (ImplicitIntegrator, MultistepIntegrator)
- Automatic differentiation (with `ADOLC`)
- Field writing routines (with Python and Sympy)
- Unittests (using `gTest`)

2 Users Guide

2.1 Command Line Use

To call code_solver from the command line, one must specify several “manditory" options. From the command line, call: code_solver –help to see a help message summarizing the manditory options and several commonly used options. For a full list of options, consult input_parser.cc.

2.2 Configuration File Use

Alternatively, one may specify all manditory options and any optional options within a configuration file and pass the configuration file to code_solver. The syntax is: code_solver input.cfg

Within the input file, one specifies the options using: option_name=option_value

For multitoken options, the present method is to repeatedly specify the option: multivalue_option=option_value1 multivalue_option=option_value2

2.3 A note on units

Units are specified in the input_sample.cfg file, and conversions from standard units should be straight forward. The system of units listed emerges by starting with cgs units, then normalizing the vector potential A by (mc/e) and scalar potential phi by (m/e) where c is the speed of light in [cm/s], e is the particle charge in [StatCoulombs], and m is the particle mass in [g], After normalizing these potentials, the units of A become [length/time] or [cm/s] and B become [1/time] or [1/s]. To obtain order 1 quantities for typical fusion test particles, it is helpful to scale the time units from [seconds] to [10^-8 seconds]. A choice of B0 = 1 then APPROXIMATELY corresponds to a magnetic field of 1 Tesla for a proton test particle. That is: B0_SI = 1 [Tesla], B0_cgs = 10^4 Gauss, B0_norm = (1.0447*10^8) [s^-1], B0_norm 1 [10^-8 s^-1]. A choice of B0 = 1 in normalized units corresponds to slightly less than 1 Tesla to the extent that (mc/e) for a proton is greater than 10^4.

As a final note, one may more generally consider the coordinates by normalizing by B0, R0. The units of length then becomes R0's and the units of time 1/B0's. Setting these parameters to 1.0 in the input file accomplishes this goal. However, the current units have been specified to be more intuitive.

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

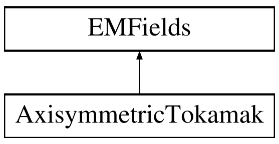
5.1 File List

Here is a list of all documented files with brief descriptions:

6 Class Documentation

6.1 AxisymmetricTokamak Class Reference

Inheritance diagram for AxisymmetricTokamak:



Public Member Functions

- AxisymmetricTokamak (const double kB0, const double kR0)

  Construct AxisymmetricTokamak which derives from EMFields. Set constants b0, r0.
- void VectorPotentialA (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &a) const

  Evaluates Vector potential A.
- void GradA (const double kt, const Eigen::VectorXd &kx, Eigen::MatrixXd &grad_a) const

  Evaluates Matrix of derivatives of vector potential A.
- void BHat (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &b_hat) const

  Evaluates Unit vector in the direction of the magnetic field.
- void GradBHat (const double kt, const Eigen::VectorXd &kx, Eigen::MatrixXd &grad_b_hat) const

  Evaluates Gradient matrix of magnetic field unit vector.
- void GradPhi (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &grad_phi) const

Evaluates Gradient of scalar potential phi.

- void GradModB (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &grad_mod_b) const

  Evaluates Gradient of magnetic field magnitude.

Private Attributes

- const double kB0_

  Magnetic field on-axis amplitude in [Tesla].
- const double kR0_

  Major radius of tokamak in [cm].

6.1.1 Constructor & Destructor Documentation

6.1.1.1 AxisymmetricTokamak::AxisymmetricTokamak (const double *kB0*, const double *kR0*)

Construct AxisymmetricTokamak which derives from EMFields. Set constants b0, r0.

Parameters

in	*kB0*	Magnetic field strength in [Tesla] (see documentation)
in	*kR0*	Major radius of tokamak in [cm]

6.1.2 Member Function Documentation

6.1.2.1 void AxisymmetricTokamak::BHat (const double *kUnusedt*, const Eigen::VectorXd & *kx*, Eigen::Vector3d & *b_hat*) const [virtual]

Evaluates Unit vector in the direction of the magnetic field.

Parameters

in	*kUnusedt*	Current time (fields time independent)
in	*kx*	Position in cartesian coordinates
out	*b_hat*	Unit vector in the direction of the magnetic field

Implements EMFields.

6.1.2.2 void AxisymmetricTokamak::GradA (const double *kUnusedt*, const Eigen::VectorXd & *kx*, Eigen::MatrixXd & *grad_a*) const [virtual]

Evaluates Matrix of derivatives of vector potential A.

Parameters

in	*kUnusedt*	Current time (fields time independent)
in	*kx*	Position in cartesian coordinates
out	*grad_a*	Matrix of derivatives of vector potential A

Implements EMFields.

6.1.2.3 void AxisymmetricTokamak::GradBHat (const double *kUnusedt*, const Eigen::VectorXd & *kx*, Eigen::MatrixXd & *grad_b_hat*) const [virtual]

Evaluates Gradient matrix of magnetic field unit vector.

Parameters

`in`	*kUnusedt*	Current time (fields time independent)
`in`	*kx*	Position in cartesian coordinates
`out`	*grad_b_hat*	Gradient matrix of magnetic field unit vector

Implements EMFields.

6.1.2.4 void AxisymmetricTokamak::GradModB (const double *kUnusedt*, const Eigen::VectorXd & *kx*, Eigen::Vector3d & *grad_mod_b*) const `[virtual]`

Evaluates Gradient of magnetic field magnitude.

Parameters

`in`	*kUnusedt*	Current time (fields time independent)
`in`	*kx*	Position in cartesian coordinates
`out`	*grad_mod_b*	Gradient of magnetic field magnitude

Implements EMFields.

6.1.2.5 void AxisymmetricTokamak::GradPhi (const double *kUnusedt*, const Eigen::VectorXd & *kUnusedx*, Eigen::Vector3d & *grad_phi*) const `[virtual]`

Evaluates Gradient of scalar potential phi.

Parameters

`in`	*kUnusedt*	Current time (fields time independent)
`in`	*kUnusedx*	Position in cartesian coordinates
`out`	*grad_phi*	Gradient of scalar potential phi

Implements EMFields.

6.1.2.6 void AxisymmetricTokamak::VectorPotentialA (const double *kUnusedt*, const Eigen::VectorXd & *kx*, Eigen::Vector3d & *a*) const `[virtual]`

Evaluates Vector potential A.

Parameters

`in`	*kUnusedt*	Current time (fields time independent)
`in`	*kx*	Position in cartesian coordinates
`out`	*a*	Vector potential A

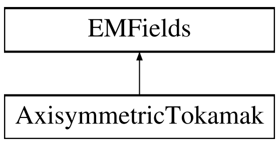
Implements EMFields.

The documentation for this class was generated from the following files:

- axisymmetric_tokamak.h
- axisymmetric_tokamak.cc

6.2 EMFields Class Reference

Inheritance diagram for EMFields:



Public Member Functions

- virtual void VectorPotentialA (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &a) const =0

  Fetch vector potential A.
- virtual void GradA (const double kt, const Eigen::VectorXd &kx, Eigen::MatrixXd &grad_a) const =0

  Fetch gradient matrix of the vector potential A.
- virtual void BHat (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &b_hat) const =0

  Fetch unit vector in direction of magnetic field.
- virtual void GradBHat (const double kt, const Eigen::VectorXd &kx, Eigen::MatrixXd &grad_b_hat) const =0

  Fetch gradient matrix of magnetic field unit vector.
- virtual void GradPhi (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &grad_phi) const =0

  Fetch gradient of scalar potential.
- virtual void GradModB (const double kt, const Eigen::VectorXd &kx, Eigen::Vector3d &grad_mod_b) const =0

  Fetch gradient of magnetic field magnitude.

The documentation for this class was generated from the following file:

- em_fields.h

6.3 GuidingCenter Class Reference

Public Member Functions

- GuidingCenter (EMFields ∗em_fields, const double kMu)

  Constructor which sets magnetic moment and initializes fields.
- int VectorField (const double kt, const Eigen::VectorXd &kx, Eigen::VectorXd &fx) const

  Evaluates vector field of ODE. f in dot{x} = f(x)
- EMFields ∗ em_fields () const

  Return pointer to em_fields instance.
- int kDimen () const

  Return dimension of ODE system.
- double kMu () const

  Return value of mu.

Private Attributes

- EMFields ∗ em_fields_

  Pointer to class defining electromagnetic fields.
- const double kMu_

  Magnetic moment.

Static Private Attributes

- static const int kDimen_ = 4

  Dimension of ODE system.

6.3.1 Constructor & Destructor Documentation

6.3.1.1 GuidingCenter::GuidingCenter (EMFields ∗ *em_fields*, const double *kMu*)

Constructor which sets magnetic moment and initializes fields.

Parameters

`in`	*em_fields*	
`in`	*kMu*	Magnetic moment in [cm^2/10^-8 s]

6.3.2 Member Function Documentation

6.3.2.1 int GuidingCenter::VectorField (const double *kt*, const Eigen::VectorXd & *kx*, Eigen::VectorXd & *fx*) const

Evaluates vector field of ODE. f in dot{x} = f(x)

Evaluation of guiding center equations of motion.

Letting x be particle position and u the velocity: dot{x}_i = (B^dag_i u - (b x E^dag)_i) / B^dag_

    dot{u} = (B^dag dot E^dag)/B^dag_par

Parameters

`in`	*kt*	Time
`in`	*kx*	Position of guiding center particle [x u]
`out`	*fx*	Right hand side of dot{x} = f(x)

Returns

zero if success

The documentation for this class was generated from the following files:

- guiding_center.h
- guiding_center.cc

6.4 InputParser Class Reference

Public Member Functions

- int ReadInput (int argc, char **argv)
- template<typename T >
  int GetValue (const char *key, T &value_out) const

Private Attributes

- po::variables_map variables_map_
  Program Options variables map.

6.4.1 Member Function Documentation

6.4.1.1 template<typename T > int InputParser::GetValue (const char * *key*, T & *value_out*) const [inline]

Accessors Retrieve the value at key and output it to value_out

6.4.1.2 int InputParser::ReadInput (int *argc*, char ** *argv*)

Read command line arguments and notify variables map of runtime options.

Parameters

`in`	*argc*	Number of input arguments
`in`	*argv*	Input arguments

Returns

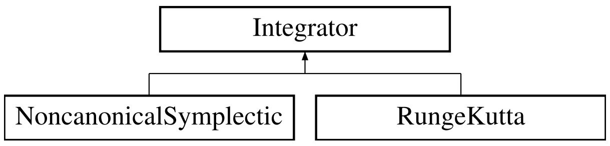
Zero for success

The documentation for this class was generated from the following files:

- input_parser.h
- input_parser.cc

6.5 Integrator Class Reference

Inheritance diagram for Integrator:



Public Member Functions

- Integrator (const double kdt, const GuidingCenter &kGuidingCenter)

  Constructor - Save model, stepsize, and dimension.
- virtual int Step (double &t, Eigen::VectorXd &x)=0

  Time-advance map from (t_k, x_k) -> (t_{k+1}, x_{k+1}).
- virtual void Reset ()

  Method used in multistep integrators.

Protected Attributes

- const double kdt_

  Numerical Step Size.
- const GuidingCenter & kGuidingCenter_

  ODE model.
- const int kDimen_

  Dimension of the ODE system.

6.5.1 Constructor & Destructor Documentation

6.5.1.1 Integrator::Integrator (const double *kdt*, const GuidingCenter & *kGuidingCenter*) `[inline]`

Constructor - Save model, stepsize, and dimension.

Parameters

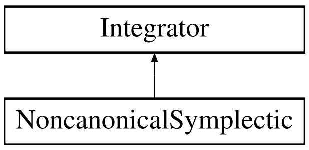
in	*kdt*	Numerical Step Size
in	*kGuidingCenter*	ODE Model instance

The documentation for this class was generated from the following file:

- integrator.h

6.6 NoncanonicalSymplectic Class Reference

Inheritance diagram for NoncanonicalSymplectic:



Public Member Functions

- NoncanonicalSymplectic (const double kdt, const GuidingCenter &kGuidingCenter, const double kNewton-Tolerance, const double kMaxIterations)
- int Step (double &t, Eigen::VectorXd &x)

  Advance t and x forward in time.
- void Reset ()

  Method used in multistep integrators.

Protected Member Functions

- int StoreHistory (const Eigen::VectorXd &kx)

  StoreHistory updates member x_history_ by knocking off the oldest data and putting on the newest.
- int InitialStep (double &t, Eigen::VectorXd &x) const
- int UpdateRule (const double kt, const Eigen::VectorXd &kx, Eigen::VectorXd &error) const

  Function which should evaluate to zero(vector) when the algorithm is satisfied.
- int NewtonGuess (double &t, Eigen::VectorXd &x) const

  Guess for initializing NewtonSolver. Uses ForwardEuler.
- int Jacobian (const double kt, const Eigen::VectorXd &kx, Eigen::MatrixXd &jacobian) const

  Calculates Jacobian matrix used in nonlinear solve. This base-class version uses a centered finite difference on the update rule.

Protected Attributes

- const double kMu_

  Magnetic moment of particle.
- const double kNewtonTolerance_

  Error threshold for nonlinear solve.
- const double kMaxIterations_

  Maximum allowable Newton iterations.
- bool needs_initialization_

  Flag for needing initial conditions.
- Eigen::MatrixXd x_history_

  Stores previous positions.
- EMFields $*$ em_fields_

  Field definitions pulled from GuidingCenter.

6.6.1 Constructor & Destructor Documentation

6.6.1.1 NoncanonicalSymplectic::NoncanonicalSymplectic (const double *kdt*, const GuidingCenter & *kGuidingCenter*, const double *kNewtonTolerance*, const double *kMaxIterations*)

Constructor - Initializes members and base class Integrator. Sets size of x_history_ to have a number of rows equal to the dimension of the ODE system and number of columns equal to the number of steps in the multistep method. In this case, two.

Parameters

in	*kdt*	Numerical Step Size
in	*kGuidingCenter*	ODE being modeled
in	*kNewton-Tolerance*	Error threshold for nonlinear solve
in	*kMaxIterations*	Maximum number of nonlinear solve iterations

6.6.2 Member Function Documentation

6.6.2.1 int NoncanonicalSymplectic::InitialStep (double & *t*, Eigen::VectorXd & *x*) const [protected]

InitialStep advances t and x using RK4. To be used while needs_initialization_ is true.

Parameters

in,out	*t*	Simulation time. Advanced by kdt_.
in,out	*x*	Position. At in: x(t=t_k) At out: x(t=t_k+1)

Returns

0 upon success

6.6.2.2 int NoncanonicalSymplectic::Jacobian (const double *kt*, const Eigen::VectorXd & *kx*, Eigen::MatrixXd & *jacobian*) const [protected]

Calculates Jacobian matrix used in nonlinear solve. This base-class version uses a centered finite difference on the update rule.

Parameters

in	*kt*	Simulation time at newest point.
in	*kx*	Position at newest point.
out	*jacobian*	derivative of the update rule w.r.t the new position

Returns

Integer code which is 0 if successful.

6.6.2.3 int NoncanonicalSymplectic::NewtonGuess (double & *t*, Eigen::VectorXd & *x*) const [protected]

Guess for initializing NewtonSolver. Uses ForwardEuler.

Parameters

in,out	*t*	Simulation time. Advanced by step size kdt
in,out	*x*	Position to advance to initial guess for solver

Returns

Zero for success

6.6.2.4 int NoncanonicalSymplectic::Step (double & *t*, Eigen::VectorXd & *x*) [virtual]

Advance t and x forward in time.

If this is the first step, some additional initial conditions need to be generated. At present, this is performed using RK4. Otherwise, solve the implicit map defined by the discrete Euler-Lagrange equations.

Parameters

`in`	*t*	Physical time at this step. For time-dependent systems.
`in,out`	*x*	Vector of current coordinates. Will be updated to x(t+h).

Returns

Integer code which is 0 if successful.

Implements Integrator.

6.6.2.5 int NoncanonicalSymplectic::StoreHistory (const Eigen::VectorXd & *kx*) [protected]

StoreHistory updates member x_history_ by knocking off the oldest data and putting on the newest.

Parameters

`in`	*kx*	New position to store

Returns

0 upon success

6.6.2.6 int NoncanonicalSymplectic::UpdateRule (const double *kt*, const Eigen::VectorXd & *kx*, Eigen::VectorXd & *error*) const [protected]

Function which should evaluate to zero(vector) when the algorithm is satisfied.

Parameters

`in`	*kt*	Simulation time at proposed new position
`in`	*kx*	New position to test
`out`	*error*	Error vector which is zero for satisfied update rule.

Returns

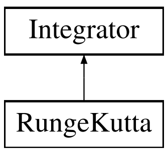
Zero for success

The documentation for this class was generated from the following files:

- noncanonical_symplectic.h
- noncanonical_symplectic.cc

6.7 RungeKutta Class Reference

Inheritance diagram for RungeKutta:



Public Member Functions

- RungeKutta (const double kdt, const GuidingCenter &kGuidingCenter, const Eigen::MatrixXd a_coefficients, const Eigen::VectorXd b_coefficients, const Eigen::VectorXd c_coefficients)

  Constructor for general runge-kutta methods. Specify the coefficients in eigen vectors/matrices.
- RungeKutta (const double kdt, const GuidingCenter &kGuidingCenter, const int kOrder)

  Constructor which implements common methods for convenience. Specify an order with an integer, and this constructor will set the corresponding coefficients.
- int Step (double &t, Eigen::VectorXd &x)

  Explicit advance x(t) forward in time by step size kdt. RK: x_{n+1} = x_n + sum_{i=1}^s b_i k_i k_i = h f(t_n + c_i h, y_n + sum_{j=1}^{i-1} a_ij k_j.
- Eigen::MatrixXd a_coefficients () const
- Eigen::VectorXd b_coefficients () const

  Access b_ coefficients vector.
- Eigen::VectorXd c_coefficients () const

  Access c_ coefficients vector.

Private Attributes

- Eigen::MatrixXd a_

  Specifies for a_ij coefficients of the RK method.
- Eigen::VectorXd b_

  Specifies b_i coefficients of the RK method.
- Eigen::VectorXd c_

  Specifies c_i coefficients of the RK method.
- Eigen::MatrixXd k_

  Temporary space for k_i evaluations.
- Eigen::VectorXd xtemp_

  Temporary space used at internal stages.
- Eigen::VectorXd ftemp_

  Temporary space for function evaluations.

Additional Inherited Members

6.7.1 Constructor & Destructor Documentation

6.7.1.1 RungeKutta::RungeKutta (const double *kdt*, const **GuidingCenter** & *kGuidingCenter*, const Eigen::MatrixXd *a_coefficients*, const Eigen::VectorXd *b_coefficients*, const Eigen::VectorXd *c_coefficients*)

Constructor for general runge-kutta methods. Specify the coefficients in eigen vectors/matrices.

Parameters

in	*kdt*	Numerical step size
in	*kGuidingCenter*	ODE system being modeled
in	*a_coefficients*	Matrix with a_{ij} coefficients in Butch. tableau
in	*b_coefficients*	Vector with b_i coefficients in Butcher tableau
in	*c_coefficients*	Vector with c_i coefficients in Butcher tableau

6.7.1.2 RungeKutta::RungeKutta (const double *kdt*, const **GuidingCenter** & *kGuidingCenter*, const int *kOrder*)

Constructor which implements common methods for convenience. Specify an order with an integer, and this constructor will set the corresponding coefficients.

Parameters

in	*kdt*	Numerical step size
in	*kGuidingCenter*	ODE system being solved
in	*kOrder*	Order of Runge-Kutta method. 2 and 4 are implemented.

6.7.2 Member Function Documentation

6.7.2.1 Eigen::MatrixXd RungeKutta::a_coefficients ( ) const [inline]

Accessors Access a_ coefficients matrix

6.7.2.2 int RungeKutta::Step (double & *t*, Eigen::VectorXd & *x*) [virtual]

Explicit advance x(t) forward in time by step size kdt. RK: x_{n+1} = x_n + sum_{i=1}^s b_i k_i k_i = h f(t_n + c_i h, y_n + sum_{j=1}^{i-1} a_ij k_j.

Parameters

in,out	*t*	Simulation time. Advanced by kdt_
in,out	*x*	Position. At in: x(t=t_k) At out: x(t=t_{k+1})

Implements Integrator.

The documentation for this class was generated from the following files:

- runge-kutta.h
- runge-kutta.cc

7 File Documentation

7.1 axisymmetric_tokamak.cc File Reference

EMFields derived class which implements the axisymmetric fields in Qin_2009.

```
#include "axisymmetric_tokamak.h"
```

7.1.1 Detailed Description

EMFields derived class which implements the axisymmetric fields in Qin_2009.

Date

Feb 2014

Author

C. Leland Ellison

7.2 axisymmetric_tokamak.h File Reference

EMFields derived class which implements the axisymmetric fields in Qin_2009 in cylindrical coordinates.

```
#include "em_fields.h"
#include <Eigen/Dense>
```

Classes

- class AxisymmetricTokamak

7.2.1 Detailed Description

EMFields derived class which implements the axisymmetric fields in Qin_2009 in cylindrical coordinates.

Date

Feb 2014

Author

C. Leland Ellison

7.3 driver.cc File Reference

Implements driver for integrating guiding center trajectories.

```
#include <stdlib.h>
#include <iostream>
#include <Eigen/Dense>
#include <ctime>
#include "input_parser.h"
#include "guiding_center.h"
#include "em_fields.h"
#include "axisymmetric_tokamak.h"
#include "integrator.h"
#include "runge-kutta.h"
#include "noncanonical_symplectic.h"
```

Functions

- void PrintState (double t, const Eigen::VectorXd &x, int n_digits)
  Prints a line to standard out giving [time x[0] x[1] ....].
- int main (int argc, char ∗argv[])
  Body of the driver. Use program options to specify ode, integrator, dt, and n_steps.

7.3.1 Detailed Description

Implements driver for integrating guiding center trajectories.

Date

Feb 2014

Author

C. Leland Ellison

7.3.2 Function Documentation

7.3.2.1 void PrintState (double *t*, const Eigen::VectorXd & *x*, int *n_digits*)

Prints a line to standard out giving [time x[0] x[1] ....].

Parameters

in	*t*	Time
in	*x*	Position vector
in	*n_digits*	Number of digits to display in output

7.4 em_fields.h File Reference

Header for EMFields abstract base class which defines the interfaces for electromagnetic field quantities used in calculating, for instance, guiding center trajectories.

```
#include <Eigen/Dense>
```

Classes

- class EMFields

7.4.1 Detailed Description

Header for EMFields abstract base class which defines the interfaces for electromagnetic field quantities used in calculating, for instance, guiding center trajectories.

Date

Feb 2014

Author

C. Leland Ellison

7.5 guiding_center.cc File Reference

Implementation of GuidingCenter system.

```
#include "guiding_center.h"
```

7.5.1 Detailed Description

Implementation of GuidingCenter system.

Date

Feb 2014

Author

C. Leland Ellison

7.6 guiding_center.h File Reference

Header for GuidingCenter system describing the motion of charged magnetized particles in magnetic fields.

```
#include <Eigen/Dense>
#include "em_fields.h"
```

Classes

- class GuidingCenter

7.6.1 Detailed Description

Header for GuidingCenter system describing the motion of charged magnetized particles in magnetic fields. Guiding center equations of motion dot{x}_i = (B^dag_i u - (b x E^dag)_i) / B^dag_

dot{u} = (B^dag dot E^dag)/B^dag_par

Date

Feb 2014

Author

C. Leland Ellison

7.7 input_parser.cc File Reference

Implementation of InputParser class. Sets runtime options.

```
#include "input_parser.h"
```

7.7.1 Detailed Description

Implementation of InputParser class. Sets runtime options.

Date

Jan 2014

Author

C. Leland Ellison

Version

0.1

7.8 input_parser.h File Reference

Interface for input parser class. Recieves command line input from driver and outputs essential information for running the driver.

```
#include <stdlib.h>
#include <fstream>
#include <boost/program_options.hpp>
```

Classes

- class InputParser

7.8.1 Detailed Description

Interface for input parser class. Recieves command line input from driver and outputs essential information for running the driver.

Date

Jan 2014

Author

C. Leland Ellison

Version

0.1

7.9 integrator.h File Reference

Header for Integrator abstract base class. Step is pure virtual. Declares members kdt_, model_ and kDimen_.

```
#include "guiding_center.h"
#include <Eigen/Dense>
```

Classes

- class Integrator

7.9.1 Detailed Description

Header for Integrator abstract base class. Step is pure virtual. Declares members kdt_, model_ and kDimen_.

Date

Feb 2014

Author

C. Leland Ellison

7.10 noncanonical_symplectic.cc File Reference

Implementation of integrator resulting from a midpoint discretization of the guiding center Lagrangian. Implements a step which interfaces with the base class integrators and defines the update rule.

```
#include "noncanonical_symplectic.h"
```

7.10.1 Detailed Description

Implementation of integrator resulting from a midpoint discretization of the guiding center Lagrangian. Implements a step which interfaces with the base class integrators and defines the update rule.

Date

Jan 2014

Author

C. Leland Ellison

7.11 noncanonical_symplectic.h File Reference

Header for integrator resulting from a midpoint discretization of the guiding center Lagrangian.

```
#include "guiding_center.h"
#include "em_fields.h"
#include "integrator.h"
#include <iostream>
#include <Eigen/Dense>
#include "runge-kutta.h"
```

Classes

- class NoncanonicalSymplectic

7.11.1 Detailed Description

Header for integrator resulting from a midpoint discretization of the guiding center Lagrangian.

Date

Feb 2014

Author

C. Leland Ellison

7.12 runge-kutta.cc File Reference

Implementation of general Runge-Kutta.

```
#include "runge-kutta.h"
```

7.12.1 Detailed Description

Implementation of general Runge-Kutta.

Date

October 2013

Author

C. Leland Ellison

Version

0.1

7.13 runge-kutta.h File Reference

Header for general Runge-Kutta integrator.

```
#include "integrator.h"
#include <Eigen/Dense>
#include "guiding_center.h"
#include <iostream>
```

Classes

- class RungeKutta

7.13.1 Detailed Description

Header for general Runge-Kutta integrator.

Date

Feb 2014

Author

C. Leland Ellison

Version

0.1

Index