libridc-0.2

Generated by Doxygen 1.8.10

Mon Dec 14 2015 09:32:11

Contents

1 About RIDC

Revisionist Integral Deferred Correction methods – a family of high-order, parallel time-integrators.

Revisionist integral deferred correction (RIDC) methods are a family of parallel–in–time methods to solve systems of initial values problems. The approach is able to bootstrap lower order time integrators to provide high order approximations in approximately the same wall clock time, hence providing a multiplicative increase in the number of compute cores utilized. Here we provide a C++ framework which automatically produces a parallel–in–time solution of a system of initial value problems given user supplied code for the right hand side of the system and a sequential code for a first-order time step. The user supplied time step routine may be explicit or implicit and may make use of any auxiliary libraries which take care of the solution of any nonlinear algebraic systems which may arise or the numerical linear algebra required. The code contains six examples of increasing complexity which also serve as templates to solve user defined problems.

2 Building and Installing

Prerequisites

There are no prerequisites for building the base RIDC software and examples in `explicit/` and `implicit/`. To build the example in `brusselator_gsl/`, the GNU Scientific Library and headers need to be installed. To build examples in `implicit_mkl/`, `brusselator_mkl/` and `brusselator\_radau\_mkl/`, the Intel Math Kernel Library needs to be installed, and appropriate environment variables initialized.

Required

- A recent C++ compiler that supports (most of) the C++11 standard. This code has been successfully tested with GCC 4.1.x and the Intel Compiler 13.0.x.

Optional

- Intel MKL or GNU Scientific Library are required for some of the examples.

Obtain the Source Code

The RIDC software is hosted at `http://mathgeek.us/software.html`. Users should download the latest `libridc-x.x.tar.gz`, and uncompress the file to your desired location.

Configure the Software

In the top level directory, `./configure --help` gives the possible configuration options. To configure using standard build configuration, type `./configure --prefix=/home/user/opt/libridc`. If you wish to compile and check the MKL and GSL examples, add the configuration flags `--with-intel-mkl` or `--with-gsl` respectively.

Building with Software

The library is built by typing `make && make check && make install`. By default, only the explicit and implicit examples are part of make check, unless the `--with-intel-mkl` or `--with-gsl` flags are added in the configuration step.

3 Contributing

The RIDC software is managed by the GNU build system. As such, the developer release requires GNU autoconf, automake, libtool, m4, make and their respective prerequisites. If there are version mismatches between the R↩ IDC software and the local system, issuing the commands `autoreconf -f` and `automake -a -c` should resolve version errors and warning. To build the documentation, Doxygen must be installed, as well as appropriate Doxygen pre-requisites. For example, to build a PDF manual documenting the source code, Doxygen requires a LaTeX compiler.

Branching

Contributors should fork the git repository hosted at `https://github.com/ongbw/ridc.git`

If this project gets large enough, we will utilize the `git-flow` workflow

Branch Name Pattern	Description
`master`	tip of the `master` branch is always the latest stable release
`development`	tip of the `development` branch is the current state of development and not expected to be stable or even usable
`feature/*`	various feature branches are used to implement new features and should be based off the `development` branch
`release/*`	a release branch is created from the `development` branch and used to prepare a new release and will be merged into `master`
`hotfix/*`	hotfix branches are based off `master` or `development` to fix important and severe bugs and should be merged into `development` and `master` as soon as possible

Releases and release candidates are tagged in the form `release-X.Y.Z(-RCa)`, where `X`, `Y`, and `Z` specify the version with respect to [semantic versioning] and `a` the number of the release candidate of that version.

Commit Messages

Please keep commit messages clean and descriptive as possible. The following are suggested:

- Commit Title must not be longer than 50 characters

  If applicable, the title should start with a category name (such as `docu`, `tests`, ...) followed by a colon (e.g. `"docu: add usage examples for RIDC"`).

- Commit Description must have line wraps at 72 characters

- Please *sign* your commits (i.e. use `git commit -s`)

How to Implement a New Feature?

1. create a fork/clone

2. switch to the `development` branch and pull in the latest changes
3. create a new branch `feature/XYZ` where `XYZ` is a short title of your planned feature (word seperation should be done with underscores, e.g. `feature/my_awesome_feature`)
4. hack and write Unit Tests
5. commit
6. repeat steps 4 and 5 until you feel your feature is in an almost usable state and most of the unit tests pass
7. write documentation for your feature
8. push your feature branch
9. stay tuned on reviews, remarks and suggestions by the other developers

4 Running the Examples

The directory `examples/` includes five examples of utilizing the RIDC library, and one example, `examples/brusselator↩_radau_mkl` that implements a three stage, fifth-order Radau method to provide a basis of comparison with the RIDC integrators. Depending on the options selected in the `./configure` step, some or all of these examples are built and run during during the `make check` process. Alternatively, a user can compile and run an example seperately after the `./configure` step. For example, the subdirectory `examples/explicit/` contains the code to solve a system of ODES using RIDC with an explicit Euler step function. To compile this specific example, move into the `examples/explicit` subdirectory and type `make explicit`. The executable `explicit` takes as input the order required and the number of time steps. For example `./explicit.exe 4 100` solves the system of ODEs using fourth order RIDC with 100 time steps. A shell script `run.sh` is provided to run the RIDC integrator with different numbers of time steps for a convergence study. A simple matlab or octave script `convergence.m` is included in that subdirectory to test the order of convergence. `octave convergence.m` gives the slope and intercept for the linear fit of log of the error versus log of the time step. In this example we obtain a slope of -4.0630 indicating the we indeed have an order 4 method.

5 Using the RIDC Library

To utilize the RIDC library, a main program should specify the **ODE** (p. 19) class, which specifies the system of ODEs to be solved, the **ODE** (p. 19) parameters (including the number of equations, number of time steps, size of the time step, and the initial and final time), as well as the step function: an Euler integerator that advances the solution from time t(n) to t(n+1). This step routine may be complicated requiring large scale linear algebra provided by external external libraries or possibly a nonlinear solve. The `examples/brusselator_gsl` directory contains such an example. This example uses a backward Euler step for a nonlinear system of ODEs. The step function uses a Newton iteration (see the functions `newt` and `jac`) and the GNU scientific library (GSL) to solve for the Newton step. The functions `newt` and `jac` required by step are defined and declared in `brusselator.h`. Finally, the solution is integrated using a call to the `ridc_fe` or `ridc_be` functions.

To link against the RIDC library, include the following arguments to the compilation command:

```
-L/home/user/opt/libridc/lib -I/home/user/opt/libridc/include -lridc
```

6 Hierarchical Index

6.1 Class Hierarchy

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

7 Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

8 File Index

8.1 File List

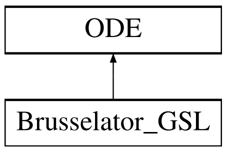
Here is a list of all files with brief descriptions:

9 Data Structure Documentation

9.1 Brusselator_GSL Class Reference

```
#include <brusselator.h>
```

Inheritance diagram for Brusselator_GSL:



Public Member Functions

- **Brusselator_GSL** (int my_neq, int my_nt, double my_ti, double my_tf, double my_dt)
- void **rhs** (double t, double ∗u, double ∗f)
- void **step** (double t, double ∗u, double ∗unew)

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

Private Member Functions

- void **newt** (double t, double ∗uprev, double ∗uguess, double ∗g)
- void **jac** (double t, double ∗u, double ∗J)

9.1.1 Detailed Description

Definition at line 10 of file brusselator.h.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 Brusselator_GSL::Brusselator_GSL (int *my_neq*, int *my_nt*, double *my_ti*, double *my_tf*, double *my_dt*) [inline]

Definition at line 12 of file brusselator.h.

References ODE::dt, ODE::neq, ODE::nt, ODE::tf, and ODE::ti.

9.1.3 Member Function Documentation

9.1.3.1 void Brusselator_GSL::jac (double *t*, double ∗ *u*, double ∗ *J*) [inline], [private]

Helper function to the Jacobian matrix (using finite differences) for advancing the solution from time t(n) to t(n+1) using an implicit Euler step on a system of equations

Returns

(by reference) J the Jacobian for the Newton step

Parameters

t	current time step
u	function value at the current time step
J	Jacobian, returned by reference

Definition at line 155 of file brusselator.h.

References ODE::dt, ODE::neq, and rhs().

Referenced by step().

9.1.3.2 void Brusselator_GSL::newt (double *t*, double ∗ *uprev*, double ∗ *uguess*, double ∗ *g*) [inline], [private]

Helper function to compute the next Newton step for solving a system of equations

Returns

(by reference) g how far from zero we are

Parameters

t	current time step
uguess	current solution guess
uprev	solution at previous time step
g	how far from zero we are, returned by reference

Definition at line 138 of file brusselator.h.

References ODE::dt, ODE::neq, and rhs().

Referenced by step().

9.1.3.3 void Brusselator_GSL::rhs (double *t*, double ∗ *u*, double ∗ *f*) [inline], [virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implements **ODE** (p. 20).

Definition at line 20 of file brusselator.h.

References ODE::neq.

Referenced by jac(), and newt().

9.1.3.4 void Brusselator_GSL::step (double *t*, double ∗ *u*, double ∗ *unew*) [inline], [virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implements **ODE** (p. 20).

Definition at line 47 of file brusselator.h.

References ODE::dt, jac(), ODE::neq, and newt().

9.1.4 Field Documentation

9.1.4.1 double ODE::dt [inherited]

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL(), corr_be(), corr_fe(), jac(), newt(), ridc_be(), ridc_fe(), and step().

9.1.4.2 int ODE::neq [inherited]

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL(), corr_be(), corr_fe(), jac(), newt(), rhs(), ridc_be(), ridc_fe(), and step().

9.1.4.3 int ODE::nt [inherited]

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL(), ridc_be(), and ridc_fe().

9.1.4.4 double ODE::tf [inherited]

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL().

9.1.4.5 double ODE::ti [inherited]

initial time

Definition at line 27 of file ridc.h.

Referenced by Brusselator_GSL(), ridc_be(), and ridc_fe().

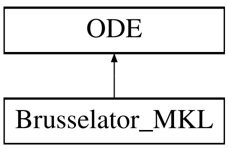
The documentation for this class was generated from the following file:

- examples/brusselator_gsl/**brusselator.h**

9.2 Brusselator_MKL Class Reference

```
#include <brusselator.h>
```

Inheritance diagram for Brusselator_MKL:



Public Member Functions

- **Brusselator_MKL** (int my_neq, int my_nt, double my_ti, double my_tf, double my_dt)
- void **rhs** (double t, double *u, double *f)
- void **step** (double t, double *u, double *unew)

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

Private Member Functions

- void **newt** (double t, double ∗uprev, double ∗uguess, double ∗g)
- void **jac** (double t, double ∗u, double ∗J)

9.2.1 Detailed Description

Definition at line 11 of file brusselator.h.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 Brusselator_MKL::Brusselator_MKL (int *my_neq,* int *my_nt,* double *my_ti,* double *my_tf,* double *my_dt*) [inline]

Definition at line 13 of file brusselator.h.

9.2.3 Member Function Documentation

9.2.3.1 void Brusselator_MKL::jac (double *t,* double ∗ *u,* double ∗ *J*) [inline], [private]

Helper function to compute the Jacobian matrix (using finite differences) for advancing the solution from time t(n) to t(n+1) using an implicit Euler step on a system of equations

Returns

(by reference) J the Jacobian for the Newton step

Parameters

t	current time step
u	solution value at the current iterate
J	Jacobian, returned by reference

Definition at line 141 of file brusselator.h.

References rhs().

9.2.3.2 void Brusselator_MKL::newt (double *t,* double ∗ *uprev,* double ∗ *uguess,* double ∗ *g*) [inline], [private]

Helper function for computing the next Newton step

Returns

(by reference) g distance from the root

Parameters

t	current time step
uguess	current solution guess
uprev	solution at previous time step
g	distance from the root, returned by reference

Definition at line 125 of file brusselator.h.

References rhs().

9.2.3.3 void Brusselator_MKL::rhs (double *t,* double ∗ *u,* double ∗ *f*) [inline], [virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implements **ODE** (p. 20).

Definition at line 21 of file brusselator.h.

9.2.3.4 void Brusselator_MKL::step (double *t*, double ∗ *u*, double ∗ *unew*) [inline], [virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implements **ODE** (p. 20).

Definition at line 48 of file brusselator.h.

References jac(), and newt().

9.2.4 Field Documentation

9.2.4.1 double ODE::dt [inherited]

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩
GSL::newt(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.2.4.2 int ODE::neq [inherited]

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩ GSL::newt(), Brusselator_GSL::rhs(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.2.4.3 int ODE::nt [inherited]

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

9.2.4.4 double ODE::tf [inherited]

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL().

9.2.4.5 double ODE::ti [inherited]

initial time

Definition at line 27 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

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

- examples/brusselator_mkl/**brusselator.h**

9.3 BUTCHER Struct Reference

```
#include <ode.h>
```

Data Fields

- int **S**
- double * **b**
- double * **c**
- double ** **A**

9.3.1 Detailed Description

Definition at line 21 of file ode.h.

9.3.2 Field Documentation

9.3.2.1 double** BUTCHER::A

Definition at line 25 of file ode.h.

Referenced by main(), and newt().

9.3.2.2 double∗ BUTCHER::b

Definition at line 23 of file ode.h.

Referenced by main(), and step().

9.3.2.3 double∗ BUTCHER::c

Definition at line 24 of file ode.h.

Referenced by main(), newt(), and step().

9.3.2.4 int BUTCHER::S

Definition at line 22 of file ode.h.

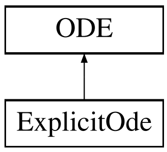
Referenced by jac(), main(), newt(), and step().

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

- examples/brusselator_radau_mkl/**ode.h**

9.4 ExplicitOde Class Reference

Inheritance diagram for ExplicitOde:



Public Member Functions

- **ExplicitOde** (int my_neq, int my_nt, double my_ti, double my_tf, double my_dt)
- void **rhs** (double t, double ∗u, double ∗f)
- void **step** (double t, double ∗u, double ∗unew)

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

9.4.1 Detailed Description

Definition at line 10 of file explicit.cpp.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 ExplicitOde::ExplicitOde (int *my_neq,* int *my_nt,* double *my_ti,* double *my_tf,* double *my_dt*) [inline]

Definition at line 12 of file explicit.cpp.

9.4.3 Member Function Documentation

9.4.3.1 void ExplicitOde::rhs (double *t*, double ∗ *u*, double ∗ *f*) [inline], [virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implements **ODE** (p. 20).

Definition at line 20 of file explicit.cpp.

9.4.3.2 void ExplicitOde::step (double *t*, double ∗ *u*, double ∗ *unew*) [inline], [virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implements **ODE** (p. 20).

Definition at line 26 of file explicit.cpp.

References rhs().

9.4.4 Field Documentation

9.4.4.1 double ODE::dt [inherited]

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_GSL::newt(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.4.4.2 int ODE::neq [inherited]

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_GSL::newt(), Brusselator_GSL::rhs(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.4.4.3 int ODE::nt [inherited]

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

9.4.4.4 double ODE::tf [inherited]

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL().

9.4.4.5 double ODE::ti [inherited]

initial time

Definition at line 27 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

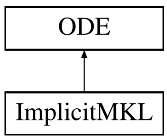
The documentation for this class was generated from the following file:

- examples/explicit/**explicit.cpp**

9.5 ImplicitMKL Class Reference

`#include <implicit.h>`

Inheritance diagram for ImplicitMKL:



Public Member Functions

- **ImplicitMKL** (int my_neq, int my_nt, double my_ti, double my_tf, double my_dt)
- void **rhs** (double t, double ∗u, double ∗f)
- void **step** (double t, double ∗u, double ∗unew)

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

Private Member Functions

- void **newt** (double t, double ∗uprev, double ∗uguess, double ∗g)
- void **jac** (double t, double ∗u, double ∗J)

9.5.1 Detailed Description

Definition at line 11 of file implicit.h.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 ImplicitMKL::ImplicitMKL (int *my_neq*, int *my_nt*, double *my_ti*, double *my_tf*, double *my_dt*) [inline]

Definition at line 13 of file implicit.h.

9.5.3 Member Function Documentation

9.5.3.1 void ImplicitMKL::jac (double *t*, double ∗ *u*, double ∗ *J*) [inline], [private]

Helper function for computing the jacobian matrix (using finite differences) for advancing the solution from time t(n) to t(n+1) using an implicit Euler step on a system of equations

Returns

(by reference) J the Jacobian for the Newton step

Parameters

t	current time step
u	function value at the current time step
J	Jacobian, returned by reference

Definition at line 128 of file implicit.h.

References rhs().

9.5.3.2 void ImplicitMKL::newt (double *t*, double ∗ *uprev*, double ∗ *uguess*, double ∗ *g*) [inline], [private]

Helper function for computing the next Newton step for solving a system of equations

Returns

(by reference) g, how far from zero we are

Parameters

t	current time step
uguess	current solution guess
uprev	solution at previous time step
g	how far from zero we are, returned by reference

Definition at line 111 of file implicit.h.

References rhs().

9.5.3.3 void ImplicitMKL::rhs (double *t*, double ∗ *u*, double ∗ *f*) [inline], [virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step

u	solution u at time t
f	rhs(t,u)

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implements **ODE** (p. 20).

Definition at line 21 of file implicit.h.

9.5.3.4 void ImplicitMKL::step (double *t*, double ∗ *u*, double ∗ *unew*) [inline], [virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implements **ODE** (p. 20).

Definition at line 33 of file implicit.h.

References jac(), and newt().

9.5.4 Field Documentation

9.5.4.1 double ODE::dt [inherited]

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↵
GSL::newt(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.5.4.2 int ODE::neq [inherited]

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩
GSL::newt(), Brusselator_GSL::rhs(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.5.4.3 int ODE::nt [inherited]

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

9.5.4.4 double ODE::tf [inherited]

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL().

9.5.4.5 double ODE::ti [inherited]

initial time

Definition at line 27 of file ridc.h.

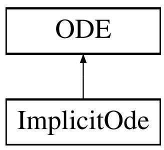
Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

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

- examples/implicit_mkl/**implicit.h**

9.6 ImplicitOde Class Reference

Inheritance diagram for ImplicitOde:



Public Member Functions

- **ImplicitOde** (int my_neq, int my_nt, double my_ti, double my_tf, double my_dt)
- void **rhs** (double t, double ∗u, double ∗f)
- void **step** (double t, double ∗u, double ∗unew)

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

9.6.1 Detailed Description

Definition at line 10 of file implicit.cpp.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 ImplicitOde::ImplicitOde (int *my_neq*, int *my_nt*, double *my_ti*, double *my_tf*, double *my_dt*) [inline]

Definition at line 12 of file implicit.cpp.

9.6.3 Member Function Documentation

9.6.3.1 void ImplicitOde::rhs (double *t*, double $*$ *u*, double $*$ *f*) [inline], [virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implements **ODE** (p. 20).

Definition at line 20 of file implicit.cpp.

9.6.3.2 void ImplicitOde::step (double *t*, double $*$ *u*, double $*$ *unew*) [inline], [virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implements **ODE** (p. 20).

Definition at line 26 of file implicit.cpp.

References rhs().

9.6.4 Field Documentation

9.6.4.1 double ODE::dt [inherited]

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩
GSL::newt(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.6.4.2 int ODE::neq [inherited]

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_GSL::newt(), Brusselator_GSL::rhs(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.6.4.3 int ODE::nt [inherited]

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

9.6.4.4 double ODE::tf [inherited]

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL().

9.6.4.5 double ODE::ti [inherited]

initial time

Definition at line 27 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

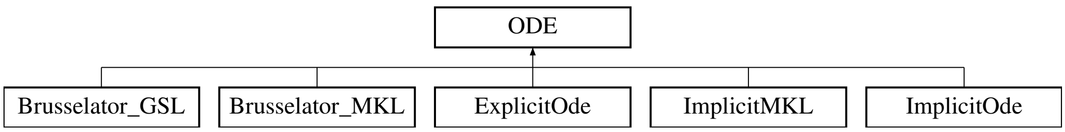
The documentation for this class was generated from the following file:

- examples/implicit/**implicit.cpp**

9.7 ODE Class Reference

#include <ridc.h>

Inheritance diagram for ODE:



Public Member Functions

- virtual void **rhs** (double t, double *u, double *f)=0
- virtual void **step** (double t, double *u, double *unew)=0

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

9.7.1 Detailed Description

Definition at line 17 of file ridc.h.

9.7.2 Member Function Documentation

9.7.2.1 virtual void ODE::rhs (double *t*, double * *u*, double * *f*) [pure virtual]

user implemented rhs function, u'=rhs(t,u)

Returns

(by reference) f: rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)

Implemented in **Brusselator_MKL** (p. 9), **ImplicitMKL** (p. 15), **Brusselator_GSL** (p. 6), **ExplicitOde** (p. 13), and **ImplicitOde** (p. 18).

Referenced by ridc_be(), and ridc_fe().

9.7.2.2 virtual void ODE::step (double *t*, double * *u*, double * *unew*) [pure virtual]

user implemented step function, for advancing the solution from t to t+dt

Returns

(by reference) unew: solution at time t+dt

Parameters

t	current time step
u	solution u at time t
unew	solution at time t+dt

Implemented in **Brusselator_MKL** (p. 10), **Brusselator_GSL** (p. 7), **ImplicitMKL** (p. 16), **ExplicitOde** (p. 13), and **ImplicitOde** (p. 18).

Referenced by corr_be(), corr_fe(), ridc_be(), and ridc_fe().

9.7.3 Field Documentation

9.7.3.1 double ODE::dt

time step

Definition at line 33 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩ GSL::newt(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.7.3.2 int ODE::neq

number of equations

Definition at line 21 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), corr_be(), corr_fe(), Brusselator_GSL::jac(), Brusselator_↩ GSL::newt(), Brusselator_GSL::rhs(), ridc_be(), ridc_fe(), and Brusselator_GSL::step().

9.7.3.3 int ODE::nt

number of time steps

Definition at line 24 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

9.7.3.4 double ODE::tf

final time

Definition at line 30 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL().

9.7.3.5 double ODE::ti

initial time

Definition at line 27 of file ridc.h.

Referenced by Brusselator_GSL::Brusselator_GSL(), ridc_be(), and ridc_fe().

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

- src/**ridc.h**

9.8 PARAMETER Struct Reference

```
#include <ode.h>
```

Data Fields

- int **neq**
- int **nt**
- double **ti**
- double **tf**
- double **dt**

9.8.1 Detailed Description

2014–6–18 **ode.h** (p. 25) Requires at least dt - delta t neq - number of equations

Definition at line 13 of file ode.h.

9.8.2 Field Documentation

9.8.2.1 double PARAMETER::dt

Definition at line 18 of file ode.h.

Referenced by main(), newt(), and step().

9.8.2.2 int PARAMETER::neq

Definition at line 14 of file ode.h.

Referenced by jac(), main(), newt(), rhs(), and step().

9.8.2.3 int PARAMETER::nt

Definition at line 15 of file ode.h.

Referenced by main().

9.8.2.4 double PARAMETER::tf

Definition at line 17 of file ode.h.

Referenced by main().

9.8.2.5 double PARAMETER::ti

Definition at line 16 of file ode.h.

Referenced by main().

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

- examples/brusselator_radau_mkl/**ode.h**

10 File Documentation

10.1 doc/source/0_installing.md File Reference

10.2 doc/source/1_contributing.md File Reference

10.3 doc/source/2_running.md File Reference

10.4 doc/source/3_use.md File Reference

10.5 examples/brusselator_gsl/brusselator.cpp File Reference

```
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#include <stdio.h>
#include <cmath>
#include "ridc.h"
#include "brusselator.h"
```

Functions

- int **main** (int argc, char ∗argv[ ])

10.5.1 Function Documentation

10.5.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 10 of file brusselator.cpp.

References ridc_be().

10.6 examples/brusselator_mkl/brusselator.cpp File Reference

```
#include <stdlib.h>
```

```
#include <omp.h>
#include <time.h>
#include <stdio.h>
#include "mkl.h"
#include "mkl_lapacke.h"
#include <cmath>
#include "ridc.h"
#include "brusselator.h"
```

Functions

- int **main** (int argc, char ∗argv[ ])

10.6.1 Function Documentation

10.6.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 12 of file brusselator.cpp.

References ridc_be().

10.7 examples/brusselator_radau_mkl/brusselator.cpp File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "mkl.h"
#include "mkl_lapacke.h"
#include <cmath>
#include "ode.h"
```

Functions

- void **rhs** (double t, double ∗u, **PARAMETER** param, double ∗f)
- void **newt** (double t, double ∗uprev, double ∗Kguess, double ∗g, **PARAMETER** param, **BUTCHER** rk)
- void **jac** (double t, double ∗uprev, double ∗Kguess, double ∗J, **PARAMETER** param, **BUTCHER** rk)
- void **step** (double t, double ∗uold, **PARAMETER** param, double ∗unew, **BUTCHER** rk)

10.7.1 Function Documentation

10.7.1.1 void jac (double *t,* double ∗ *uprev,* double ∗ *Kguess,* double ∗ *J,* PARAMETER *param,* BUTCHER *rk*)

Helper function for computing the Jacobian matrix (using finite differences) for advancing the solution from time t(n) to t(n+1) using an implicit RK step on a system of equations

Returns

(by reference) J the Jacobian for the Newton step

Parameters

param	structure containing number of equations, number of time steps, initial and final time, time step
t	current time step
uprev	function value at the previous (known) time step
Kguess	iterated guess for the stage values
J	Jacobian, returned by reference
rk	Butcher Tableau coefficients

Definition at line 70 of file brusselator.cpp.

References PARAMETER::neq, newt(), and BUTCHER::S.

Referenced by ImplicitMKL::step(), Brusselator_MKL::step(), and step().

10.7.1.2 void newt (double *t*, double ∗ *uprev*, double ∗ *Kguess*, double ∗ *g*, PARAMETER *param*, BUTCHER *rk*)

Helper function for advancing the solution from time t(n) to t(n+1) using an implicit RK step on a non linear system using a Newton step.

Returns

(by reference) g distance from the root, returned by reference

Parameters

param	structure containing number of equations, number of time steps, initial and final time, time step
t	current time step
uprev	function value at the previous (known) time step
Kguess	iterated guess for the stage values
rk	Butcher Tableau coefficients
g	distance from the root, returned by reference

Definition at line 32 of file brusselator.cpp.

References BUTCHER::A, BUTCHER::c, PARAMETER::dt, PARAMETER::neq, rhs(), and BUTCHER::S.

Referenced by jac(), ImplicitMKL::step(), Brusselator_MKL::step(), and step().

10.7.1.3 void rhs (double *t*, double ∗ *u*, PARAMETER *param*, double ∗ *f*)

rhs function, u'=rhs(t,u)

Returns

(by reference) f rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)
param	structure containing number of equations, number of time steps, initial and final time, time step

Definition at line 9 of file brusselator.cpp.

References PARAMETER::neq.

Referenced by ImplicitMKL::jac(), Brusselator_MKL::jac(), newt(), ImplicitMKL::newt(), Brusselator_MKL::newt(), ExplicitOde::step(), ImplicitOde::step(), and step().

10.7.1.4 void step (double *t*, double ∗ *uold*, PARAMETER *param*, double ∗ *unew*, BUTCHER *rk*)

Definition at line 105 of file brusselator.cpp.

References BUTCHER::b, BUTCHER::c, PARAMETER::dt, jac(), PARAMETER::neq, newt(), rhs(), and BUTCH←
ER::S.

Referenced by main().

10.8 examples/brusselator_gsl/brusselator.h File Reference

```
#include "ridc.h"
#include <stdlib.h>
#include <stdio.h>
#include <cmath>
#include <gsl/gsl_linalg.h>
```

Data Structures

- class **Brusselator_GSL**

10.9 examples/brusselator_mkl/brusselator.h File Reference

```
#include "ridc.h"
#include <stdio.h>
#include "mkl.h"
#include "mkl_lapacke.h"
```

Data Structures

- class **Brusselator_MKL**

Macros

- #define **_BRUSSELATOR_H_**

10.9.1 Macro Definition Documentation

10.9.1.1 #define _BRUSSELATOR_H_

Definition at line 7 of file brusselator.h.

10.10 examples/brusselator_radau_mkl/ode.h File Reference

Data Structures

- struct **PARAMETER**
- struct **BUTCHER**

Functions

- void **rhs** (double t, double ∗u, **PARAMETER** param, double ∗f)
- void **newt** (double t, double ∗uprev, double ∗Kguess, double ∗g, **PARAMETER** param, **BUTCHER** rk)
- void **jac** (double t, double ∗uprev, double ∗Kguess, double ∗J, **PARAMETER** param, **BUTCHER** rk)
- void **step** (double t, double ∗u, **PARAMETER** param, double ∗unew, **BUTCHER** rk)

10.10.1 Function Documentation

10.10.1.1 void jac (double *t*, double ∗ *uprev*, double ∗ *Kguess*, double ∗ *J*, PARAMETER *param*, BUTCHER *rk*)

Helper function for computing the Jacobian matrix (using finite differences) for advancing the solution from time t(n) to t(n+1) using an implicit RK step on a system of equations

Returns

(by reference) J the Jacobian for the Newton step

Parameters

param	structure containing number of equations, number of time steps, initial and final time, time step
t	current time step
uprev	function value at the previous (known) time step
Kguess	iterated guess for the stage values
J	Jacobian, returned by reference
rk	Butcher Tableau coefficients

Definition at line 70 of file brusselator.cpp.

References PARAMETER::neq, newt(), and BUTCHER::S.

Referenced by ImplicitMKL::step(), Brusselator_MKL::step(), and step().

10.10.1.2 void newt (double *t*, double ∗ *uprev*, double ∗ *Kguess*, double ∗ *g*, PARAMETER *param*, BUTCHER *rk*)

Helper function for advancing the solution from time t(n) to t(n+1) using an implicit RK step on a non linear system using a Newton step.

Returns

(by reference) g distance from the root, returned by reference

Parameters

param	structure containing number of equations, number of time steps, initial and final time, time step
t	current time step
uprev	function value at the previous (known) time step
Kguess	iterated guess for the stage values
rk	Butcher Tableau coefficients
g	distance from the root, returned by reference

Definition at line 32 of file brusselator.cpp.

References BUTCHER::A, BUTCHER::c, PARAMETER::dt, PARAMETER::neq, rhs(), and BUTCHER::S.

Referenced by jac(), ImplicitMKL::step(), Brusselator_MKL::step(), and step().

10.10.1.3 void rhs (double *t*, double ∗ *u*, PARAMETER *param*, double ∗ *f*)

rhs function, u'=rhs(t,u)

Returns

(by reference) f rhs(t,u)

Parameters

t	current time step
u	solution u at time t
f	rhs(t,u)
param	structure containing number of equations, number of time steps, initial and final time, time step

Definition at line 9 of file brusselator.cpp.

References PARAMETER::neq.

Referenced by ImplicitMKL::jac(), Brusselator_MKL::jac(), newt(), ImplicitMKL::newt(), Brusselator_MKL::newt(), ExplicitOde::step(), ImplicitOde::step(), and step().

10.10.1.4 void step (double *t,* double ∗ *u,* **PARAMETER** *param,* double ∗ *unew,* **BUTCHER** *rk*)

Definition at line 105 of file brusselator.cpp.

References BUTCHER::b, BUTCHER::c, PARAMETER::dt, jac(), PARAMETER::neq, newt(), rhs(), and BUTCH←ER::S.

Referenced by main().

10.11 examples/brusselator_radau_mkl/radau.cpp File Reference

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "mkl.h"
#include "mkl_lapacke.h"
#include <cmath>
#include "ode.h"
```

Functions

- int **main** (int argc, char ∗argv[ ])

  This is the the main function for the brusselator_radau_mkl example.

10.11.1 Function Documentation

10.11.1.1 int main (int *argc,* char ∗ *argv[ ]*)

This is the the main function for the brusselator_radau_mkl example.

This will pass user given options along with some standard options for this type of problem in to the **PARAMETER** (p. 21) struct and start the solving the problem

Definition at line 17 of file radau.cpp.

References BUTCHER::A, BUTCHER::b, BUTCHER::c, PARAMETER::dt, PARAMETER::neq, PARAMETER::nt, BUTCHER::S, step(), PARAMETER::tf, and PARAMETER::ti.

10.12 examples/explicit/explicit.cpp File Reference

```
#include <stdlib.h>
```

```
#include <omp.h>
#include <time.h>
#include <stdio.h>
#include "ridc.h"
```

Data Structures

- class **ExplicitOde**

Functions

- int **main** (int argc, char *argv[ ])

10.12.1 Function Documentation

10.12.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 38 of file explicit.cpp.

References ridc_fe().

10.13 examples/implicit/implicit.cpp File Reference

```
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#include <stdio.h>
#include "ridc.h"
```

Data Structures

- class **ImplicitOde**

Functions

- int **main** (int argc, char *argv[ ])

10.13.1 Function Documentation

10.13.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 39 of file implicit.cpp.

References ridc_be().

10.14 examples/implicit_mkl/implicit.cpp File Reference

```
#include <stdlib.h>
```

```
#include <omp.h>
#include <time.h>
#include <stdio.h>
#include "ridc.h"
#include "implicit.h"
```

Functions

- int **main** (int argc, char *argv[ ])

10.14.1 Function Documentation

10.14.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 8 of file implicit.cpp.

References ridc_be().

10.15 examples/implicit_mkl/implicit.h File Reference

```
#include "ridc.h"
#include <stdio.h>
#include "mkl.h"
#include "mkl_lapacke.h"
```

Data Structures

- class **ImplicitMKL**

Macros

- #define **_IMPLICIT_H_**

10.15.1 Macro Definition Documentation

10.15.1.1 #define _IMPLICIT_H_

Definition at line 7 of file implicit.h.

10.16 README.md File Reference

10.17 src/ridc.cpp File Reference

```
#include "ridc.h"
#include <stdio.h>
```

Functions

- void **ridc_fe** (**ODE** *ode, int order, double *sol)
- void **ridc_be** (**ODE** *ode, int order, double *sol)

- void **lagrange_coeff** (double ∗x, int Nx, int i, double ∗L)
- double **get_quad_weight** (double ∗L, int Nx, double a, double b)
- void **integration_matrices** (int N, double ∗∗S)
- void **init_unif_nodes** (double ∗x, int Nx, double a, double b)
- void **corr_fe** (**ODE** ∗ode, double ∗uold, double ∗∗fprev, double ∗∗S, int index, int level, double t, double ∗unew)
- void **corr_be** (**ODE** ∗ode, double ∗uold, double ∗∗fprev, double ∗∗S, int index, int level, double t, double ∗unew)

10.17.1 Function Documentation

10.17.1.1 void corr_be (ODE ∗ *ode,* double ∗ *uold,* double ∗∗ *fprev,* double ∗∗ *S,* int *index,* int *level,* double *t,* double ∗ *unew*)

RIDC helper function - solves error equation, updating the solution from time t to time t+param.dt.

Returns

(by reference) unew: solution at time level t + param.dt

Parameters

ode	abstract class containing parameters and step/rhs functions
uold	solution at time level t
fprev	matrix containing derivative information from previous steps, previous level
S	integration matrix (quadrature weights)
index	decides which quadrature weights to use
level	determines size of quadrature stencil
t	current time iterate
unew	solution at the new time level, passed by reference

Definition at line 894 of file ridc.cpp.

References ODE::dt, ODE::neq, and ODE::step().

Referenced by ridc_be().

10.17.1.2 void corr_fe (ODE ∗ *ode,* double ∗ *uold,* double ∗∗ *fprev,* double ∗∗ *S,* int *index,* int *level,* double *t,* double ∗ *unew*)

RIDC helper function - solves error equation, updating the solution from time t to time t+param.dt.

Returns

(by reference) unew: solution at time level t + param.dt

Parameters

ode	abstract class containing parameters and step/rhs functions
uold	solution at time level t
fprev	matrix containing derivative information from previous steps, previous level
S	integration matrix (quadrature weights)
index	decides which quadrature weights to use
level	determines size of quadrature stencil

t	current time iterate
unew	solution at the new time level, passed by reference

Definition at line 846 of file ridc.cpp.

References ODE::dt, ODE::neq, and ODE::step().

Referenced by ridc_fe().

10.17.1.3 double get_quad_weight (double ∗ *L,* int *Nx,* double *a,* double *b*)

RIDC helper function – generates quadrature weight, int(L_{n,i}(x),x=a..b)

Returns

quadrature weights

Parameters

a	range of integration
b	range of integration
Nx	number of quadrature nodes
L	coefficients for Lagrange poly, L[0] + L[1]x + L[2]x^2 + ...

Definition at line 723 of file ridc.cpp.

Referenced by integration_matrices().

10.17.1.4 void init_unif_nodes (double ∗ *x,* int *Nx,* double *a,* double *b*)

RIDC helper function – initializes uniformly spaced quadrature nodes

Returns

(by reference) x: uniformly spaced quadrature nodes

Parameters

Nx	number of quadrature nodes
a	range of integration
b	range of integration
x	quadrature node location (returned by reference)

Definition at line 811 of file ridc.cpp.

Referenced by integration_matrices().

10.17.1.5 void integration_matrices (int *Nx,* double ∗∗ *S*)

RIDC helper function – constructions the integration matrix using get_quad_weight

Returns

(by reference) the integration matrix S

Parameters

Nx	number of quadrature nodes
S	integration matrix (by reference)

Definition at line 765 of file ridc.cpp.

References get_quad_weight(), init_unif_nodes(), and lagrange_coeff().

Referenced by ridc_be(), and ridc_fe().

10.17.1.6 void lagrange_coeff (double * *x*, int *Nx*, int *i*, double * *L*)

RIDC helper function – generates the coefficients for the lagrange interpolatory polynomials.

Returns

(by reference) L: coefficients for the Lagrange intepolatory polynomial. L is a vector of elements such that p(x) = L(0) + L(1)*x + L(2)*x^2 + ...

Parameters

x	quadrature nodes
i	the i^{th} Lagrange polynomial
Nx	number of quadrature nodes
L	coefficients, returned by reference

Definition at line 652 of file ridc.cpp.

Referenced by integration_matrices().

10.17.1.7 void ridc_be (ODE * *ode*, int *order*, double * *sol*)

Main implicit ridc loop that initializes variables, integrates solution from ti to tf by bootstrapping the step function.

Returns

(by reference) sol, the solution at the final time, param.tf

Parameters

ode	abstract class containing parameters and step/rhs functions
order	order of the RIDC method (predictor + number of correctors)
sol	initial condition of the IVP

Definition at line 341 of file ridc.cpp.

References corr_be(), ODE::dt, integration_matrices(), ODE::neq, ODE::nt, ODE::rhs(), ODE::step(), and ODE::ti.

Referenced by main().

10.17.1.8 void ridc_fe (ODE * *ode*, int *order*, double * *sol*)

Main explicit ridc loop that initializes variables, integrates solution from ti to tf by bootstrapping the step function.

Returns

(by reference) sol, the solution at the final time, param.tf

Parameters

ode	abstract class containing parameters and step/rhs functions
order	order of the RIDC method (predictor + number of correctors)
sol	initial condition of the IVP

Definition at line 32 of file ridc.cpp.

References corr_fe(), ODE::dt, integration_matrices(), ODE::neq, ODE::nt, ODE::rhs(), ODE::step(), and ODE::ti.

Referenced by main().

10.18 src/ridc.h File Reference

header file containing explanation of functions for the RIDC integrator

```
#include <omp.h>
#include <cmath>
#include <algorithm>
```

Data Structures

- class **ODE**

Functions

- void **ridc_fe** (**ODE** ∗ode, int order, double ∗sol)
- void **ridc_be** (**ODE** ∗ode, int order, double ∗sol)
- void **lagrange_coeff** (double ∗x, int Nx, int i, double ∗L)
- double **get_quad_weight** (double ∗L, int Nx, double a, double b)
- void **integration_matrices** (int Nx, double ∗∗S)
- void **init_unif_nodes** (double ∗x, int Nx, double a, double b)
- void **corr_fe** (**ODE** ∗ode, double ∗uold, double ∗∗fprev, double ∗∗S, int index, int level, double t, double ∗unew)
- void **corr_be** (**ODE** ∗ode, double ∗uold, double ∗∗fprev, double ∗∗S, int index, int level, double t, double ∗unew)

10.18.1 Detailed Description

header file containing explanation of functions for the RIDC integrator

Author

Ong:Benjamin

Version

Revision 0.2

Date

2015-09-04

10.18.2 Function Documentation

10.18.2.1 void corr_be (ODE ∗ *ode,* double ∗ *uold,* double ∗∗ *fprev,* double ∗∗ *S,* int *index,* int *level,* double *t,* double ∗ *unew*)

RIDC helper function - solves error equation, updating the solution from time t to time t+param.dt.

Returns

(by reference) unew: solution at time level t + param.dt

Parameters

ode	abstract class containing parameters and step/rhs functions
uold	solution at time level t
fprev	matrix containing derivative information from previous steps, previous level
S	integration matrix (quadrature weights)
index	decides which quadrature weights to use
level	determines size of quadrature stencil
t	current time iterate
unew	solution at the new time level, passed by reference

Definition at line 894 of file ridc.cpp.

References ODE::dt, ODE::neq, and ODE::step().

Referenced by ridc_be().

10.18.2.2 void corr_fe (ODE * *ode*, double * *uold*, double ** *fprev*, double ** *S*, int *index*, int *level*, double *t*, double * *unew*)

RIDC helper function - solves error equation, updating the solution from time t to time t+param.dt.

Returns

(by reference) unew: solution at time level t + param.dt

Parameters

ode	abstract class containing parameters and step/rhs functions
uold	solution at time level t
fprev	matrix containing derivative information from previous steps, previous level
S	integration matrix (quadrature weights)
index	decides which quadrature weights to use
level	determines size of quadrature stencil
t	current time iterate
unew	solution at the new time level, passed by reference

Definition at line 846 of file ridc.cpp.

References ODE::dt, ODE::neq, and ODE::step().

Referenced by ridc_fe().

10.18.2.3 double get_quad_weight (double * *L*, int *Nx*, double *a*, double *b*)

RIDC helper function – generates quadrature weight, int(L_{n,i}(x),x=a..b)

Returns

quadrature weights

Parameters

a	range of integration
b	range of integration
Nx	number of quadrature nodes
L	coefficients for Lagrange poly, L[0] + L[1]x + L[2]x^2 + ...

Definition at line 723 of file ridc.cpp.

Referenced by integration_matrices().

10.18.2.4 void init_unif_nodes (double * *x*, int *Nx*, double *a*, double *b*)

RIDC helper function – initializes uniformly spaced quadrature nodes

Returns

(by reference) x: uniformly spaced quadrature nodes

Parameters

Nx	number of quadrature nodes
a	range of integration
b	range of integration
x	quadrature node location (returned by reference)

Definition at line 811 of file ridc.cpp.

Referenced by integration_matrices().

10.18.2.5 void integration_matrices (int *Nx*, double ∗∗ *S*)

RIDC helper function – constructions the integration matrix using get_quad_weight

Returns

(by reference) the integration matrix S

Parameters

Nx	number of quadrature nodes
S	integration matrix (by reference)

Definition at line 765 of file ridc.cpp.

References get_quad_weight(), init_unif_nodes(), and lagrange_coeff().

Referenced by ridc_be(), and ridc_fe().

10.18.2.6 void lagrange_coeff (double ∗ *x*, int *Nx*, int *i*, double ∗ *L*)

RIDC helper function – generates the coefficients for the lagrange interpolatory polynomials.

Returns

(by reference) L: coefficients for the Lagrange intepolatory polynomial. L is a vector of elements such that p(x) = L(0) + L(1)∗x + L(2)∗x^2 + ...

Parameters

x	quadrature nodes
i	the i^{th} Lagrange polynomial
Nx	number of quadrature nodes
L	coefficients, returned by reference

Definition at line 652 of file ridc.cpp.

Referenced by integration_matrices().

10.18.2.7 void ridc_be (ODE ∗ *ode*, int *order*, double ∗ *sol*)

Main implicit ridc loop that initializes variables, integrates solution from ti to tf by bootstrapping the step function.

Returns

(by reference) sol, the solution at the final time, param.tf

Parameters

ode	abstract class containing parameters and step/rhs functions
order	order of the RIDC method (predictor + number of correctors)
sol	initial condition of the IVP

Definition at line 341 of file ridc.cpp.

References corr_be(), ODE::dt, integration_matrices(), ODE::neq, ODE::nt, ODE::rhs(), ODE::step(), and ODE::ti.

Referenced by main().

10.18.2.8 void ridc_fe (ODE * *ode*, int *order*, double * *sol*)

Main explicit ridc loop that initializes variables, integrates solution from ti to tf by bootstrapping the step function.

Returns

(by reference) sol, the solution at the final time, param.tf

Parameters

ode	abstract class containing parameters and step/rhs functions
order	order of the RIDC method (predictor + number of correctors)
sol	initial condition of the IVP

Definition at line 32 of file ridc.cpp.

References corr_fe(), ODE::dt, integration_matrices(), ODE::neq, ODE::nt, ODE::rhs(), ODE::step(), and ODE::ti.

Referenced by main().

Index