Peer-methods

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 return_values Struct Reference

Public Attributes

- double $*$ **yT**
- int **yT_size**
- double $*$ **y**
- int **y_rows**
- int **y_cols**
- double $*$ **t**
- int **t_size**

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

- peerMethods.h

Chapter 4

File Documentation

4.1 peerMethods.h File Reference

The library provides an implementation for the main function for solving peer method.

Classes

- struct return_values

Macros

- #define **STAGES** 2

Functions

- void initReturnStruct (return_values ∗rv)

  Initialize the struct return_values.
- int saveResultsInFile (const char ∗fileName, return_values result)

  Save the struct return_values in a file.
- void defineLMatrix (double ∗∗L, int ∗LSize, double Delta_x)

  Spatial semi-discretization of the PDE (Partial Differential Equation). Build the matrix L to perform the spatial semi-discretization of the ODE.
- double ∗ Sherratt (const double ∗y0, int y0Size, const double ∗L, int Lsize, int ∗sherrattSize)

  Applies the Sherratt method.
- double ∗ RungeKutta4th (double h, double t0, const double ∗y0, int y0Size, const double ∗L, int Lsize, int ∗ySize)

  Returns the numerical approximation of the solution of the equation at the next time step, given the current one, by using Runge Kutta 4th order method.
- void fPeerClassic_twoStages (int N, double ∗t_span, int t_span_size, const double ∗L, int Lsize, const double ∗y0, int y0_size, return_values ∗collect_result)

  Time discretization using peer methods. The method solve a large system of ODEs (Ordinary Differential Equations) by using peer methods.
- void ∗ Malloc (size_t size)

  Function wrapper for malloc() function.

- void * Calloc (size_t nmemb, size_t size)

  Function wrapper for calloc() function.
- void initializeRandomVector (double *vector, int N)

  Initialize a vector with random values. NOTE: the seed must be initialized in the calling method.
- void initializeRandomMatrix (double *matrix, int M, int N)

  Initialize a matrix with random values. NOTE: the seed must be initialized in the calling method.
- int initMatrixByRowWithValuesFromVector (double *matrix, int M, int N, double *vector, int vector_size)

  Using a vector to initialize the matrix. The matrix and the vector must have the same dimension. For example, if the matrix A has 3 x 4 elements, the vector B must have 12 elements.
- void initVectorWAnotherVector (double *newVector, double *oldVector, int n)

  Using a vector to initialize another vector. The vectors must have the same dimension.
- void freeEverything (void *arg1,...)

  Free a variable number of pointers.
- double * intervalDiscretization (double first, double last, double step, int *N)

  Provide the discretization of an interval starting with first and ending with last.
- double * eyeD (int N)

  Create identity matrix, a matrix in which the values on the main diagonal have value 1.
- double * onesD (int N)

  Create array of all ones.
- double * zerosD (int N)

  Create array of all zeros.
- double * zerosMatrixD (int M, int N)

  Create matrix of all zeros.
- double * diagD (double *vector, int size, int k, int *matrix_size)

  Create diagonal matrix with all the elements of vector on the k-th diagonal.
- double * packThreeMatrices (int n, double *A, double *B, double *C)

  Packing three square matrices side by side with the same dimension into a new big one.
- double * threeBlockDiagD (int n, double *A, double *B, double *C, int *blckSize)

  Create a square block diagonal matrix made up of three matrices.
- double * packThreeVectors (int n, double *A, double *B, double *C, int *newDimension)

  Packing three vectors side by side into one.
- double * **linspace** (double x1, double x2, int n)

Variables

- double **a**
- double **B1**
- double **B2**
- double **F**
- double **H**
- double **S**
- double **d**
- double **D**
- double **L**
- int **M**

4.1.1 Detailed Description

The library provides an implementation for the main function for solving peer method.

Author

Vincenzo Iannucci

Version

0.1

Date

2022-11-29

4.1.2 Function Documentation

4.1.2.1 Calloc()

```
void* Calloc (
            size_t nmemb,
            size_t size )
```

Function wrapper for calloc() function.

Parameters

in	*nmemb*	number of elements to allocate
in	*size*	Size of the memory allocated

Returns

a pointer to the allocated memory

4.1.2.2 defineLMatrix()

```
void defineLMatrix (
            double ** L,
            int * LSize,
            double Delta_x )
```

Spatial semi-discretization of the PDE (Partial Differential Equation). Build the matrix L to perform the spatial semi-discretization of the ODE.

Parameters

out	*L*	returning pointer to the matrix
in	*LSize*	return the size of the matrix
in	*Delta_x*	the value of the delta

4.1.2.3 diagD()

```
double* diagD (
            double * vector,
            int size,
            int k,
            int * matrix_size )
```

Create diagonal matrix with all the elements of vector on the k-th diagonal.

Parameters

vector	pointer to the vector
size	size of the vector
k	the number of diagonal
matrix_size	the size of the output matrix

Returns

pointer to the new matrix allocated by rows

4.1.2.4 eyeD()

```
double* eyeD (
            int N )
```

Create identity matrix, a matrix in which the values on the main diagonal have value 1.

Parameters

N	size of the output array

Returns

pointer to the new array

4.1.2.5 fPeerClassic_twoStages()

```
void fPeerClassic_twoStages (
            int N,
            double * t_span,
            int t_span_size,
            const double * L,
            int Lsize,
            const double * y0,
            int y0_size,
            return_values * collect_result )
```

Time discretization using peer methods. The method solve a large system of ODEs (Ordinary Differential Equations) by using peer methods.

Parameters

in	*N*	the size of the time grid
in	*t_span*	an array representing the time grid itself
in	*t_span_size*	the spatial dimension of the time grid
in	*L*	pointer to the matrix L deriving from the spatial semi-discretization
in	*LSize*	size of the matrix
in	*y0*	pointer to the y0 vector, containing the initial conditions
in	*y0Size*	size of the y0 vector
out	*collect_result*	size of the result vector

Returns

a struct containing.

4.1.2.6 freeEverything()

```
void freeEverything (
            void * arg1,
             ... )
```

Free a variable number of pointers.

Parameters

in	*arg1*	pointer

4.1.2.7 initializeRandomMatrix()

```
void initializeRandomMatrix (
            double * matrix,
```

```
            int M,
            int N )
```

Initialize a matrix with random values. NOTE: the seed must be initialized in the calling method.

Parameters

in	*matrix*	pointer to the first element of the matrix
in	*M*	number of rows
in	*N*	number of columns

4.1.2.8 initializeRandomVector()

```
void initializeRandomVector (
            double * vector,
            int N )
```

Initialize a vector with random values. NOTE: the seed must be initialized in the calling method.

Parameters

in	*vector*	pointer to the vector
in	*N*	dimension of the vector

4.1.2.9 initMatrixByRowWithValuesFromVector()

```
int initMatrixByRowWithValuesFromVector (
            double * matrix,
            int M,
            int N,
            double * vector,
            int vector_size )
```

Using a vector to initialize the matrix. The matrix and the vector must have the same dimension. For example, if the matrix A has 3 x 4 elements, the vector B must have 12 elements.

Parameters

	[in	out] matrix pointer to the matrix
in	*M*	rows of the matrix
in	*N*	columns of the matrix
in	*vector*	pointer to the vector
in	*vector_size*	size of the vector (must be equal to M x N)

Returns

0 if ok, 1 otherwise

4.1.2.10 initReturnStruct()

```
void initReturnStruct (
            return_values * rv )
```

Initialize the struct return_values.

Parameters

rv	pointer to the struct return_values

4.1.2.11 initVectorWAnotherVector()

```
void initVectorWAnotherVector (
            double * newVector,
            double * oldVector,
            int n )
```

Using a vector to initialize another vector. The vectors must have the same dimension.

Parameters

	[in	out] newVector pointer to the new vector
in	*oldVector*	pointer to the old vector
in	*n*	size of the vectors

4.1.2.12 intervalDiscretization()

```
double* intervalDiscretization (
            double first,
            double last,
            double step,
            int * N )
```

Provide the discretization of an interval starting with first and ending with last.

Parameters

first	starting of the interval
last	ending of the interval
step	the spacing between each value of the array
N	the size of the final array returned

Returns

pointer to an array of double, representing the discretized interval

4.1.2.13 Malloc()

```
void* Malloc (
            size_t size )
```

Function wrapper for malloc() function.

Parameters

in	*size*	Size of the memory allocated

Returns

a pointer to the allocated memory

4.1.2.14 onesD()

```
double* onesD (
            int N )
```

Create array of all ones.

Parameters

N	

Returns

pointer to the new array

4.1.2.15 packThreeMatrices()

```
double* packThreeMatrices (
            int n,
            double * A,
            double * B,
            double * C )
```

Packing three square matrices side by side with the same dimension into a new big one.

Parameters

n	number of rows
A	pointer the first matrix
B	pointer the second matrix
C	pointer the third matrix

Returns

pointer to the new matrix

4.1.2.16 packThreeVectors()

```
double* packThreeVectors (
            int n,
            double * A,
            double * B,
            double * C,
            int * newDimension )
```

Packing three vectors side by side into one.

Parameters

n	number of rows
A	pointer the first matrix
B	pointer the second matrix
C	pointer the third matrix
newDimension	the size of the output vector

Returns

pointer to the new vector

4.1.2.17 RungeKutta4th()

```
double* RungeKutta4th (
            double h,
            double t0,
            const double * y0,
            int y0Size,
            const double * L,
            int Lsize,
            int * ySize )
```

Returns the numerical approximation of the solution of the equation at the next time step, given the current one, by using Runge Kutta 4th order method.

Parameters

in	*h*	constant grid spacing (the space between the values of the discrete grid)
in	*t0*	starting time
in	*y0*	vector representing value of the y function in y(t0)
in	*y0Size*	size of the y0 vector
in	*L*	pointer to the matrix L
in	*LSize*	size of the matrix
out	*ySize*	size of the resulting vector

Returns

a vector that approximates the value of the solution at the next time step t0 + h, y(t0 + h)

4.1.2.18 saveResultsInFile()

```
int saveResultsInFile (
            const char * fileName,
            return_values result )
```

Save the struct return_values in a file.

Parameters

in	*fileName*	the name of the file
out	*rv*	pointer to the struct return

Returns

0 if ok, 1 otherwise.

4.1.2.19 Sherratt()

```
double* Sherratt (
            const double * y0,
            int y0Size,
            const double * L,
            int Lsize,
            int * sherrattSize )
```

Applies the Sherratt method.

Parameters

in	*y0*	pointer to the y0 vector
in	*y0Size*	size of the y0 vector
in	*L*	pointer to the matrix L
in	*LSize*	size of the matrix
out	*sherrattSize*	returing size of the vector calculated by the function

Returns

a pointer the resulting vector after applying the Sherratt method.

4.1.2.20 threeBlockDiagD()

```
double* threeBlockDiagD (
            int n,
            double * A,
            double * B,
            double * C,
            int * blckSize )
```

Create a square block diagonal matrix made up of three matrices.

Parameters

n	number of rows
A	pointer the first matrix
B	pointer the second matrix
C	pointer the third matrix
blockSize	the size of the output matrix

Returns

pointer to the output matrix

4.1.2.21 zerosD()

```
double* zerosD (
            int N )
```

Create array of all zeros.

Parameters

N	dimension of the array

Returns

pointer to the new array

4.1.2.22 zerosMatrixD()

```
double* zerosMatrixD (
            int M,
            int N )
```

Create matrix of all zeros.

Parameters

M	the number of rows
N	the number of columns

Returns

pointer to the new matrix allocated by rows

Chapter 5

Example Documentation

5.1 linspace

Generate linearly spaced vector. (double x1, double x2, int n) generates n points. The spacing between the points is (x2-x1)/(n-1).

Returns

pointer to the new vector

Index