

Peer-methods

Generated by Doxygen 1.8.17

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 return_values Struct Reference	5
3.1.1 Detailed Description	5
4 File Documentation	7
4.1 peerMethods.h File Reference	7
4.1.1 Detailed Description	9
4.1.2 Function Documentation	9
4.1.2.1 Calloc()	9
4.1.2.2 defineLMatrix()	9
4.1.2.3 diagD()	10
4.1.2.4 eyeD()	10
4.1.2.5 fPeerClassic_twoStages()	11
4.1.2.6 freeEverything()	11
4.1.2.7 initializeRandomMatrix()	12
4.1.2.8 initializeRandomVector()	12
4.1.2.9 initMatrixByRowWithValuesFromVector()	12
4.1.2.10 initReturnStruct()	13
4.1.2.11 initVectorWAnotherVector()	13
4.1.2.12 intervalDiscretization()	13
4.1.2.13 Malloc()	14
4.1.2.14 onesD()	14
4.1.2.15 packThreeMatrices()	15
4.1.2.16 packThreeVectors()	15
4.1.2.17 peerMethodsDaxpy()	16
4.1.2.18 peerMethodsDgemv()	16
4.1.2.19 peerMethodsDscal()	17
4.1.2.20 RungeKutta4th()	17
4.1.2.21 saveResultsInFile()	18
4.1.2.22 Sherratt()	18
4.1.2.23 threeBlockDiagD()	19
4.1.2.24 zerosD()	19
4.1.2.25 zerosMatrixD()	19
5 Example Documentation	21
5.1 linspace	21
Index	23

Chapter 1

Class Index

1.1 Class List

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

return_values	This struct contains the returning values from peer methods function	5
-------------------------------	--	---

Chapter 2

File Index

2.1 File List

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

[peerMethods.h](#)

The library provides an implementation for the main function for solving peer method [7](#)

Chapter 3

Class Documentation

3.1 return_values Struct Reference

This struct contains the returning values from peer methods function.

```
#include <peerMethods.h>
```

Public Attributes

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

3.1.1 Detailed Description

This struct contains the returning values from peer methods function.

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](#)

This struct contains the returning values from peer methods function.

Macros

- `#define STAGES 2`
- `#define a 1.5`
- `#define B1 0.45`
- `#define B2 0.3611`
- `#define F 0.802`
- `#define H 0.802`
- `#define S 0.0002`
- `#define d 500`
- `#define D 0.802`

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.
- void [peerMethodsDaxpy](#) (const int N, const double alpha, const double *X, const int incX, double *Y, const int incY)
Constant times a vector plus a vector.
- void [peerMethodsDscal](#) (const int N, const double alpha, double *X, const int incX)
Scales a vector by a constant.
- void [peerMethodsDgemv](#) (const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, double *Y, const int incY)
*Performs matrix-vector operations $y := y + \alpha * A * x$.*
- 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 *blkSize)
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

- int **M**

4.1.1 Detailed Description

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

Author

Vincenzo Iannucci

Version

0.2

Date

01-30-2023 (MM-DD-YYYY)

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	<i>nmemb</i>	number of elements to allocate
in	<i>size</i>	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	<i>L</i>	returning pointer to the matrix
in	<i>LSize</i>	return the size of the matrix
in	<i>Delta_x</i>	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

<i>vector</i>	pointer to the vector
<i>size</i>	size of the vector
<i>k</i>	the number of diagonal
<i>matrix_size</i>	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

<i>N</i>	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	<i>N</i>	the size of the time grid
in	<i>t_span</i>	an array representing the time grid itself
in	<i>t_span_size</i>	the spatial dimension of the time grid
in	<i>L</i>	pointer to the matrix L deriving from the spatial semi-discretization
in	<i>LSize</i>	size of the matrix
in	<i>y0</i>	pointer to the y0 vector, containing the initial conditions
in	<i>y0Size</i>	size of the y0 vector
out	<i>collect_result</i>	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	<i>arg1</i>	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	<i>matrix</i>	pointer to the first element of the matrix
in	<i>M</i>	number of rows
in	<i>N</i>	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	<i>vector</i>	pointer to the vector
in	<i>N</i>	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

	<i>[in</i>	out] matrix pointer to the matrix
--	------------	-----------------------------------

Parameters

in	M	rows of the matrix
in	N	columns of the matrix
in	<i>vector</i>	pointer to the vector
in	<i>vector_size</i>	size of the vector (must be equal to $M \times N$)

Returns

0 if ok, 1 otherwise

4.1.2.10 initReturnStruct()

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

Initialize the struct `return_values`.

Parameters

<i>rv</i>	pointer to the struct <code>return_values</code>
-----------	--

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

	<i>[in</i>	out] newVector pointer to the new vector
in	<i>oldVector</i>	pointer to the old vector
in	<i>n</i>	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

<i>first</i>	starting of the interval
<i>last</i>	ending of the interval
<i>step</i>	the spacing between each value of the array
<i>N</i>	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

<i>N</i>	
----------	--

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

<i>n</i>	number of rows
<i>A</i>	pointer the first matrix
<i>B</i>	pointer the second matrix
<i>C</i>	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

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

Returns

pointer to the new vector

4.1.2.17 peerMethodsDaxpy()

```
void peerMethodsDaxpy (
    const int N,
    const double alpha,
    const double * X,
    const int incX,
    double * Y,
    const int incY )
```

Constant times a vector plus a vector.

Parameters

<i>N</i>	number of elements in input vector(s)
<i>alpha</i>	scalar
<i>X</i>	double precision array
<i>incX</i>	storage spacing between elements of X
<i>Y</i>	double precision array
<i>incY</i>	storage spacing between elements of Y

4.1.2.18 peerMethodsDgemv()

```
void peerMethodsDgemv (
    const int M,
    const int N,
    const double alpha,
    const double * A,
    const int lda,
    const double * X,
    const int incX,
    double * Y,
    const int incY )
```

Performs matrix-vector operations $y := y + \alpha A * x$.

Parameters

<i>M</i>	specifies the number of rows of the matrix A
<i>N</i>	specifies the number of columns of the matrix A
<i>alpha</i>	scalar
<i>A</i>	double precision array containing the matrix coefficients
<i>lda</i>	specifies the second dimension of A

Parameters

<i>X</i>	double precision array containing the elements of the vector x
<i>incX</i>	storage spacing between elements of X
<i>Y</i>	double precision array resulting elements from matrix-vector multiplication
<i>incY</i>	storage spacing between elements of Y

4.1.2.19 peerMethodsDscal()

```
void peerMethodsDscal (
    const int N,
    const double alpha,
    double * X,
    const int incX )
```

Scales a vector by a constant.

Parameters

<i>N</i>	number of elements in input vector(s)
<i>alpha</i>	scalar
<i>X</i>	double precision array
<i>incX</i>	storage spacing between elements of X

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

Returns

a vector that approximates the value of the solution at the next time step $t_0 + h$, $y(t_0 + h)$

4.1.2.21 saveResultsInFile()

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

Save the struct `return_values` in a file.

Parameters

in	<i>fileName</i>	the name of the file
out	<i>rv</i>	pointer to the struct return

Returns

0 if ok, 1 otherwise.

4.1.2.22 Sherratt()

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

Applies the Sherratt method.

Parameters

in	<i>y0</i>	pointer to the y0 vector
in	<i>y0Size</i>	size of the y0 vector
in	<i>L</i>	pointer to the matrix L
in	<i>LSize</i>	size of the matrix
out	<i>sherrattSize</i>	returning size of the vector calculated by the function

Returns

a pointer the resulting vector after applying the Sherratt method.

4.1.2.23 threeBlockDiagD()

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

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

Parameters

<i>n</i>	number of rows
<i>A</i>	pointer the first matrix
<i>B</i>	pointer the second matrix
<i>C</i>	pointer the third matrix
<i>blkSize</i>	the size of the output matrix

Returns

pointer to the output matrix

4.1.2.24 zerosD()

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

Create array of all zeros.

Parameters

<i>N</i>	dimension of the array
----------	------------------------

Returns

pointer to the new array

4.1.2.25 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

Calloc
 peerMethods.h, 9

defineLMatrix
 peerMethods.h, 9

diagD
 peerMethods.h, 10

eyeD
 peerMethods.h, 10

fPeerClassic_twoStages
 peerMethods.h, 11

freeEverything
 peerMethods.h, 11

initializeRandomMatrix
 peerMethods.h, 12

initializeRandomVector
 peerMethods.h, 12

initMatrixByRowWithValuesFromVector
 peerMethods.h, 12

initReturnStruct
 peerMethods.h, 13

initVectorWAnotherVector
 peerMethods.h, 13

intervalDiscretization
 peerMethods.h, 13

Malloc
 peerMethods.h, 14

onesD
 peerMethods.h, 14

packThreeMatrices
 peerMethods.h, 15

packThreeVectors
 peerMethods.h, 15

peerMethods.h, 7
 Calloc, 9
 defineLMatrix, 9
 diagD, 10
 eyeD, 10
 fPeerClassic_twoStages, 11
 freeEverything, 11
 initializeRandomMatrix, 12
 initializeRandomVector, 12
 initMatrixByRowWithValuesFromVector, 12
 initReturnStruct, 13
 initVectorWAnotherVector, 13
 intervalDiscretization, 13
 Malloc, 14
 onesD, 14
 packThreeMatrices, 15
 packThreeVectors, 15
 peerMethodsDaxpy, 16
 peerMethodsDgemv, 16
 peerMethodsDscal, 17
 RungeKutta4th, 17
 saveResultsInFile, 18
 Sherratt, 18
 threeBlockDiagD, 18
 zerosD, 19
 zerosMatrixD, 19
peerMethodsDaxpy
 peerMethods.h, 16
peerMethodsDgemv
 peerMethods.h, 16
peerMethodsDscal
 peerMethods.h, 17

return_values, 5

RungeKutta4th
 peerMethods.h, 17

saveResultsInFile
 peerMethods.h, 18

Sherratt
 peerMethods.h, 18

threeBlockDiagD
 peerMethods.h, 18

zerosD
 peerMethods.h, 19

zerosMatrixD
 peerMethods.h, 19