TaylorPLib_CSharp

Class: Polynomial

Namespace: LibMatrix

(Taylor) Polynomial with derivate degree n (n+1 coefficients):

P_n(x) = f(a) + (x-a)*f'(a)/1! + (x-a)^2*f''(a)/2! + (x-a)^3*f'''(a)/3!
+ ... + (x-a)^n*f^{(n)}(a)/n!

= sum{k=0}{n} (x-a)^k*f^{(k)}(a)/k!

being 'f' the function to be approximated by P_n(x) at point 'a', with its first n derivatives existing on a closed interval I, so that

f(x) = P_n(x) + R_n(x)

the remainder term being R_n(x) = (x-a)^{n+1}*f^{(n+1)}(c)/(n+1)! for some 'c' between 'x' and 'a'.

Another often used form:

f(x0+h) = f(x0) + h*f'(x0)/1! + h^2*f''(x0)/2! + h^3*f'''(x0)/3!
+ ... + h^n*f^{(n)}(x0)/n!

Polynomial - Constructor

Default constructor for the class. Creates the object.
It is a Taylor polynomial of order zero, i.e., it has a constant value:

p(x) = 1

Polynomial - Constructor

Constructor for the class with a derivative order as parameter. Creates the object.
Example, order=3 <==/> 4 coefficients:

p(x) = p_0 + p_1*x + p_2*x^2 + p_3*x^3

Parameters:

order	The derivative order of the Taylor polynomial.

Polynomial - Constructor

Constructor for the class with given params

Parameters:

order	number of items (zerobased)
coeffs	must match order e.g.: order = 3 -> coeffs = new double[4] with values
constant	[optional] 1 = constant, zero = not constant, -1 = unknown

Polynomial - Constructor

Copy constructor.
Polynomial newtp = new Polynomial(oldtp);
Equivalent to Polynomial newtp = oldtp;

Parameters:

p	The Taylor polynomial object to copy from.

Finalize - Methode

Destructor. Cleans up the object.

unsetConst - Methode

fior unseting the constCount

initializePolynomial(System.Int32,System.Double[],System.Int32) - Methode

for initializing the Polynomial

Parameters:

order	The order of Polynomial
coeffs	The Values
constant	default = -1

op_Equality(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the == operator.
Compares two Taylor polynomials.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if equal

op_Inequality(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the != operator.
Compares two Taylor polynomials.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if not equal

Equals(System.Object) - Methode

Overrides the base function Equals

Parameters:

obj	object of Polynomial

Return:	true if coeffs[] are equal

GetHashCode - Methode

Overrides the base function GetHashCode()

Return:	hash of coeffs[]

op_LessThan(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the < operator.
Compares two Taylor polynomials according to the value of the first coefficient.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if a < b

op_LessThanOrEqual(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the <= operator.
Compares two Taylor polynomials according to the value of the first coefficient.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if a <= b

op_GreaterThan(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the > operator.
Compares two Taylor polynomials according to the value of the first coefficient.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if a > b

op_GreaterThanOrEqual(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the >= operator.
Compares two Taylor polynomials according to the value of the first coefficient.

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	true if a >= b

op_Addition(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the + operator

Parameters:

a	Polynomial on the left side
b	Polynomial on the right side

Return:	a + b (if order mathches)

op_UnaryNegation(LibMatrix.Polynomial) - Methode

Implements the unary - Operator

Parameters:

a	

Return:	

op_Subtraction(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Implements the -= operator.
Substracts a Taylor polynomial from the current one using pointers to arrays that store the coefficients.

Parameters:

a	the Taylor polynomial to be substracted from.
b	Polynomial on the right side

Return:	a - b (if order mathches)

op_Multiply(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Multiplies two Taylor polynomials. Implements the * operator for Taylor arithmetic.
The following coefficient propagation rule is applied:

v_k = sum_{j=0}{k} u_j * w_{k-j}

for k = 1...d and v(t) = u(t) * w(t), u, v, w being Taylor polynomials, and d being the derivative degree.

It is assumed that all three Taylor polynomials have the same derivative degree d.
Three different cases are distinguished here: when at least one of the polynomials is a constant polynomial and when both polynomials are not.

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

Parameters:

a	the Taylor polynomial to be multiplied with.
b	the Taylor polynomial to be multiplied by.

Return:	The resulting Polynomail (without changing the order)

op_Multiply(LibMatrix.Polynomial,System.Double) - Methode

Implements the * operator.
Multiplies a Taylor polynomial by a scalar.

Parameters:

a	The Polynomial value to multiply with
d	The scalar value to multiply by

Return:	

op_Division(LibMatrix.Polynomial,LibMatrix.Polynomial) - Methode

Divides a Taylor polynomial (dividend) by another Taylor polynomial (divisor).
Implements the / operator for Taylor arithmetic.

v_k = 1 / w_0 * [u_k - sum_{j=0}{k-1} v_j * w_{k-j}]

for k = 1...d and v(t) = u(t) / w(t), u, v, w being Taylor polynomials, and d being the derivative degree.

It is assumed that all three Taylor polynomials have the same derivative degree d.

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

Parameters:

a	Polynomial as divisor
b	Polynomial as divident

Return:	the resulting Taylor polynomial.

TaylorPLib_CSharp

sqr - Methode

Polynomial PExpect = new Polynomial(3, new double[] { 1, 4, 10, 20 });
Taylor arithmetic.

The following coefficient propagation rule is applied:

v_k = sum_{j=0}{k} u_j * u_{k-j}

for k = 1...d and v(t) = u(t)^2, u and v being a Taylor polynomials, and d being the derivative degree.

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

Return:	The resulting Taylor polynomial.

setSqr - Methode

Sets this Taylor polynomial to its square.

The following coefficient propagation rule is applied:

v_k = sum_{j=0}{k} u_j * u_{k-j}

for k = 1...d and v(t) = u(t)^2, u and v being a Taylor polynomials, and d being the derivative degree.

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

sqrt - Methode

Calculates the square root of a Taylor polynomial. Implements the square root function for Taylor arithmetic.

The following coefficient propagation rule is applied:

v_k = 1 / 2*v_0 * [u_k - sum_{j=1}{k-1} v_j * v_{k-j}]

for k = 1...d and v(t) = sqrt(u(t)), u and v being a Taylor polynomials, and d being the derivative degree. In particular, v_0 = sqrt(u_0).

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

Return:	

setSqrt - Methode

Sets this Taylor polynomial to its square root.

The following coefficient propagation rule is applied:

v_k = 1 / 2*v_0 * [u_k - sum_{j=1}{k-1} v_j * v_{k-j}]

for k = 1...d and v(t) = sqrt(u(t)), u and v being a Taylor polynomials, and d being the derivative degree. In particular, v_0 = sqrt(u_0).

(See Griewank's book, p.222 from "Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation". In Frontiers in Applied Mathematics Nr. 19, SIAM, Philadelphia, PA, 2000)

TaylorPLib_CSharp

print - Methode

Prints out the coefficients of a Taylor polynomial, starting by the independent term.
Prints out to Standard Output (Console)

print(System.String) - Methode

Prints out to a file the coefficients of a Taylor polynomial, starting by the independent term.

Parameters:

filenameWithPath	The output file to write the polynomial to.

eval(System.Double,System.Double) - Methode

Evaluates a Taylor polynomial at a given value with a point of expansion.

Parameters:

x	The value to evaluate the polynomial at.
alpha	alpha The point of expansion.

Return:	The result of the evaluation.

feval - Methode

Returns the first coefficient of the Taylor polynomial, i.e., the evaluation of the function.

Return:	The evaluation of the function at the initial point.

shift - Methode

Implements the SHIFT operator to calculate the derivative of a Taylor polynomial.

The new coefficients are shifted to the left and the last one is zeroed.
E.g.:

```
y(t) = sum_{j=0}^{d} y_j * t^j + O(t^d+1)
     = y_0 + y_1*t + y_2*t^2 + ... + y_d*t^d

y'(t) = y_1 + 2*y_2*t + 3*y_3*t^2 + ... + d*y_d*t^d-1 + 0
```

isConst - Methode

Checks if Polynomial is constant
(if _constant unknown then it will be set)

isConst(System.Double) - Methode

Returns true in case it is near a constant Taylor polynomial;
false otherwise.

Parameters:

eps	The threshold value to compare with.

Return:	\a true if it is a constant Taylor polynomial; \a false otherwise.

isId - Methode

Returns \a true in case it is a constant Taylor polynomial with value 1;
\a false otherwise.

Return:	\a true if it is a constant Taylor polynomial with value 1; \a false otherwise.

TaylorPLib_CSharp

isId(System.Double) - Methode

Returns \a true in case it is near a constant Taylor polynomial with value 1;
\a false otherwise.

Parameters:

eps	The threshold value to compare with.

Return:	\a true if it is a constant Taylor polynomial with value 1; \a false otherwise.

isZero - Methode

Returns \a true in case all coefficients of the Taylor polynomial are zeroed;
\a false otherwise.

Return:	\a true if all coefficients are zeroed; \a false otherwise.

isZero(System.Double) - Methode

Returns \a true in case all coefficients of the Taylor polynomial are lower or equal than
a threshold given as parameter; \a false otherwise.

Parameters:

eps	The threshold value to compare with.

Return:	\a true if all coefficients are almost null; \a false otherwise.

set2Zero - Methode

Sets all coefficients of a Taylor polynomial to zero.

set2Zero(System.Int32) - Methode

Sets the coefficients of a Taylor polynomial to zero, from the order given as parameter on.

Parameters:

order	Derivative order from which to start on (increasingly).

set2const(System.Double) - Methode

Sets a Taylor polynomial to the constant given as parameter.

Parameters:

c	The constant value of type \a double to set the Taylor polynomial to.

setCoeffs(System.Double[]) - Methode

Sets the coefficients of a Taylor polynomial to the ones given as parameter.

Parameters:

c	A vector of coefficients of type \type double.

set2Id - Methode

sets the Polynomial to id

getValueAt(System.Int32) - Methode

Returns value at the given index from the array

Parameters:

index	The index to be analyzed.

Return:	The coefficient at that index.

ToString - Methode

Returns a String of the Polynomial

Return:	String in the Format 2x^2 + -1x^1 + 7

TaylorPLib_CSharp

order - Property

returns order

ncoeff - Property

returns the number of coeffs

Item(System.Int32) - Property

poly[2] = 7, i.e., '[]' also in the left side!)

TaylorPLib_CSharp

Class: Matrix

Namespace: LibMatrix

Matrix Class

_rows - Field

Number of rows

_cols - Field

Number of columns

_dimT - Field

The dimension of the Taylor polynomials

_data - Field

The Taylor Polynomials in an 2 dimensional Matrix Array

nrows - Methode

Number of Rows

Return:	Number of Rows

ncols - Methode

Number of Cols

Return:	Number of Cols

dimT - Methode

Dimension of Polynomials

Return:	Dimension of Polynomials

get(System.Int32,System.Int32) - Methode

Get the Polynomial at the given position

Parameters:

row	row index
col	col index

Return:	Polynomial at given position

isSquare - Methode

Square Matrix

Return:	true if cols == rows

Matrix - Constructor

Default constructor for the class. Creates the object.

It is a 1-by-1 matrix, i.e., it has only one element, which is set to zero:

m(0,0) == 0.0

Matrix - Constructor

Constructor for the class with both the number of rows and columns as parameters.
Creates the object.

Parameters:

rows	The number of rows.
cols	The number of columns.

Matrix - Constructor

Constructor for the class with both the number of rows and columns as parameters,
as well as the dimension of the elements' type, e.g., the Taylor polynomial's grade.
Creates the object.

Parameters:

rows	The number of rows.
cols	The number of columns.
dimT	The dimension

Matrix - Constructor

Copy constructor.

```
Matrix *newm = new Matrix( (*m) );
```

Parameters:

matrix	A Matrix object to copy from.

Matrix - Constructor

Easy constructor for testing.

Parameters:

rows	The number of rows.
cols	The number of columns.
values	an initialised Polynom

Finalize - Methode

Destructor. Cleans up the object.

op_Equality(LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Implements the == operator. It compares two matrices.

Parameters:

a	Matrix on the left side
b	Matrix on the right side

Return:	true if the matrices are equal. Otherwise it returns false.

op_Inequality(LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Implements the != operator.
Compares two matrices.

Parameters:

a	Matrix on the left side
b	Matrix on the right side

Return:	true if not equal

op_Addition(LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Implements the + operator. It adds up two matrices.

Parameters:

a	Matrix on the left side
b	Matrix on the right side

Return:	the resulting Matrix object.

TaylorPLib_CSharp

op_Subtraction(LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Implements the - operator. It substracts two matrices.

Parameters:

a	Matrix on the left side
b	Matrix on the right side

Return:	the resulting Matrix object.

op_UnaryNegation(LibMatrix.Matrix) - Methode

Implements the unary - operator.

Parameters:

a	Matrix to operate on

Return:	the resulting Matrix object.

op_Multiply(LibMatrix.Matrix,System.Double) - Methode

Implements the * operator. It multiplies a matrix by a scalar.

Parameters:

a	The Matrix to multiply with.
alpha	The scalar to multiply by.

Return:	the resulting Matrix object.

op_Multiply(LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Implements the * operator. It multiplies the matrix with another matrix.

Parameters:

a	Matrix on the left side
b	Matrix on the right side

Return:	the resulting Matrix object.

mmCaABbC(System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B + beta * C

with A : m-by-p matrix
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * B
beta	The scalar value that multiplies C
A	an object of type Matrix
B	an object of type Matrix

TaylorPLib_CSharp

bmmCaABbC(System.Int32,System.Int32,System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B + beta * C

with A : m-by-p matrix
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

where the inferior-right block of B is an identity matrix like in:
(* * * 0 0)
(* * * 0 0)
(0 0 0 1 0)
(0 0 0 0 1)

so that a particular block multiplication is needed.

Parameters:

r	The number of rows in B that are of interest (2 in the example above)
c	The number of columns in B that are of interest (3 in the example above)
alpha	The scalar value that multiplies A * B
beta	The scalar value that multiplies C
A	an object of type Matrix
B	an object of type Matrix

mmCasABbC(System.Int32,System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B + beta * C

with A : m-by-p matrix
B : p-by-n matrix
C : r-by-n matrix (only the last r rows from A are interesting)
alpha, beta : real numbers

where A ("special" A) is of the form:

(   )
(X   )
(-------)
(* ... *)
(* ... *)

so that a particular matrix multiplication is needed.

Parameters:

r	The last rows in A that are of interest (2 non-zero-rows in the example above)
alpha	The scalar value that multiplies A * B
beta	beta The scalar value that multiplies C
A	an object of type Matrix
B	an object of type Matrix

TaylorPLib_CSharp

mmCaAsBbC(System.Int32,System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B + beta * C

with A : m-by-p matrix
B : p-by-n matrix
C : m-by-r matrix (only the last r columns from B are interesting).
alpha, beta : real numbers

where B ("special" B) is of the form:

```
(   | * * * )
(   | . . . )
( X | . . . )
(   | . . . )
(   | * * * )
```

so that a particular matrix multiplication is needed.

Parameters:

r	The last columns in B that are of interest (3 non-zero-columns - * in the example above)
alpha	The scalar value that multiplies A * B
beta	The scalar value that multiplies C
A	an object of type Matrix
B	an object of type Matrix

mmCaAUTBPbC(System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix,System.Int32[]) - Methode

Matrix multiplication of the form:

C = alpha * A * UTB + beta * C

where UTB means that only the upper triangular part is of interest. Furthermore, a column pivoting on B is considered.

with A : m-by-p matrix
B : p-by-n matrix
C : m-by-r matrix (only the last r columns from B are interesting).
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * B
beta	The scalar value that multiplies C
A	an object of type Matrix
B	an object of type Matrix
piv	a vector of permutations on the columns of B

mmCaAATbC(System.Double,System.Double,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * A^T + beta * C

with A, C : m-by-m matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * A^T
beta	The scalar value that multiplies C
A	an object of type Matrix. Its transpose is also considered

TaylorPLib_CSharp

mmCaATAbC(System.Double,System.Double,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A^T * A + beta * C

with A, C : m-by-m matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * A^T
beta	The scalar value that multiplies C
A	an object of type Matrix. Its transpose is also considered

mmCaATBbC(System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A^T * B + beta * C

with A : p-by-m matrix
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * A^T
beta	The scalar value that multiplies C
A	an object of type Matrix. Its transpose is also considered
B	an object of type Matrix. Its transpose is also considered

mmCaATBPbC(System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix,System.Int32[]) - Methode

Matrix multiplication of the form:

C = alpha * A^T * B + beta * C

with A : p-by-m matrix
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

and a column pivoting on A^Ts rows

Parameters:

alpha	The scalar value that multiplies A * A^T
beta	The scalar value that multiplies C
A	an object of type Matrix. Its transpose is also considered
B	an object of type Matrix
piv	a vector of permutations on the columns of B

mmCaABTbC(System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B^T + beta * C

with A : m-by-p matrix
B : n-by-p matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * B^T
beta	The scalar value that multiplies C
A	an object of type Matrix.
B	an object of type Matrix. Its transpose is also considered

mmCaABTbC(System.Int32,System.Boolean,System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B^T + beta * C

with A : m-by-p matrix
B : n-by-p matrix
C : m-by-n matrix
alpha, beta : real numbers

After transposing B, either its first or its last rows are considered for multiplication,

```
( * ... * )                  (    X    )
( ------- )        or      ( ------- )
(         )                 ( * ... * )
(    X    )                  ( * ... * )
```

according to A dimensions. I.e., the matrix A has less columns than B^T rows has.

Parameters:

r	The number of rows from B that should be considered.
up	The binary parameter to indicate whether the first or the last r rows
alpha	The scalar value that multiplies A * B^T
beta	The scalar value that multiplies C
A	an object of type Matrix.
B	an object of type Matrix. Its transpose is also considered

TaylorPLib_CSharp

bmmCaABTbC(System.Int32,System.Int32,System.Double,System.Double,LibMatrix.Matrix,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * B^T + beta * C

with A : m-by-p matrix
B : n-by-p matrix
C : m-by-n matrix
alpha, beta : real numbers

where the inferior-right block of A is an identity matrix like in:

(* * * 0 0)
(* * * 0 0)
(0 0 0 1 0)
(0 0 0 0 1)

so that a particular block multiplication is needed.

Parameters:

r	The number of rows in A that are of interest (2 in the example above).
c	The number of columns in A that are of interest (3 in the example above).
alpha	The scalar value that multiplies A * B^T
beta	The scalar value that multiplies C
A	an object of type Matrix.
B	an object of type Matrix. Its transpose is also considered

mmCaIBbC(System.Double,System.Double,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * I * B + beta * C

with I : m-by-p matrix; identity matrix
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies I * B
beta	The scalar value that multiplies C
B	an object of type Matrix

mmCaIBbC(System.Double,System.Double,System.Int32[],System.Boolean,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * I * B + beta * C

with I : m-by-p matrix; identity matrix permuted according to a vector of permutations, piv
B : p-by-n matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies I * B
beta	The scalar value that multiplies C
piv	a vector of permutations on I, of type int
rows	The binary parameter to indicate whether the rows or the columns of I should be permuted (true for the rows; false for the columns)
B	an object of type Matrix

TaylorPLib_CSharp

mmCaAIbC(System.Double,System.Double,LibMatrix.Matrix) - Methode

Matrix multiplication of the form:

C = alpha * A * I + beta * C

with A : m-by-p matrix
I : p-by-n matrix; identity matrix
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * I
beta	The scalar value that multiplies C
A	an object of type Matrix

mmCaAIbC(System.Double,System.Double,LibMatrix.Matrix,System.Int32[],System.Boolean) - Methode

Matrix multiplication of the form:

C = alpha * A * I + beta * C

with A : m-by-p matrix
I : p-by-n matrix; identity matrix permuted according to a vector of permutations, piv
C : m-by-n matrix
alpha, beta : real numbers

Parameters:

alpha	The scalar value that multiplies A * I
beta	The scalar value that multiplies C
A	an object of type Matrix
piv	a vector of permutations on I, of type int
rows	The binary parameter to indicate whether the rows or the columns of I should be permuted (true for the rows; false for the columns)

utxsolve(LibMatrix.Matrix) - Methode

Solves the equation

X U = B

by back-substitution, where:

U : m-by-m upper triangular matrix, non-singular
X : m-by-n matrix
B : m-by-n matrix, overwritten with the solution on output.

The X_ik are calculated making few modifications to the function 'utsolve' for UX=B.

Parameters:

B	an object of type Matrix that is the independent term on input

cpermutem(System.Int32[],System.Boolean) - Methode

Permutes the columns of a matrix given a vector of permutations.

For example, in case a matrix A is permuted after a QR decomposition with column pivoting, then the resulting matrix in the upper triangular matrix R.

Parameters:

piv	a vector of permutations on the columns of A
trans	The boolean parameter to indicate whether to transpose the vector of permutations piv or not (=1, transpose; =0, otherwise). Default is false

TaylorPLib_CSharp

rpermutem(System.Int32[]) - Methode

Permutes the rows of a matrix given a vector of permutations.

Parameters:

piv	a vector of permutations on the rows of A

transpose - Methode

Transposes this matrix in place.

asTranspose - Methode

Creata the transposes of this matrix. This matrix opject remains unchanged.

Return:	The transposed matrix object

shift - Methode

Implements the shift operator to calculate the derivative of Taylor polynomials
in case the elements of the matrix are such, like in:

```
y(t) = sum_{j=0}^{d} y_j * t^j + O(t^d+1)
       = y_0 + y_1*t + y_2*t^2 + ... + y_d*t^d

y'(t) = y_1 + 2*y_2*t + 3*y_3*t^2 + ... + d*y_d*t^d-1
```

Internally, the coefficients are shifted to the left and the last one is zeroed.

isId - Methode

Returns true in case the given matrix is the identity matrix; false otherwise.

Return:	true if the matrix is the identity matrix; false otherwise.

isZero - Methode

Returns true in case the given matrix is the zero matrix; false otherwise.

Return:	true if the matrix is the zero matrix; false otherwise.

set2Id - Methode

Sets a matrix to the identity one:

M = I

set2Id(System.Int32,System.Int32,System.Int32,System.Int32) - Methode

Sets a submatrix to the identity one:

```
e.g. M = (    | 1 0 0 |    )
         ( M1 | 0 1 0 | M2 )
         (    | 0 0 1 |    )
         (       M3        )
```

Parameters:

top	The number of rows at the top to keep unchanged
bottom	The number of rows at the bottom to keep unchanged
left	The number of columns on the left to keep unchanged
right	The number of columns on the right to keep unchanged

set2IdFromIndices(System.Int32,System.Int32,System.Int32,System.Int32) - Methode

Sets a submatrix to the identity one:

```
e.g. M = (    | 1 0 0 |    )
         ( M1 | 0 1 0 | M2 )
         (    | 0 0 1 |    )
         (       M3        )
```

Parameters:

firstRow	The row from which to start on
lastRow	The last row that should be considered
firstCol	The column from which to start on
lastCol	The last column that should be considered

set2Zero - Methode

Sets a matrix to zero entries.

set2Zero(System.Int32,System.Int32,System.Int32,System.Int32) - Methode

Sets a submatrix to zero:

```
e.g. M = (    | 0 0 0 |    )
         ( M1 | 0 0 0 | M2 )
         (    | 0 0 0 |    )
         (       M3        )
```

Parameters:

top	The number of rows at the top to keep unchanged
bottom	The number of rows at the bottom to keep unchanged
left	The number of columns on the left to keep unchanged
right	The number of columns on the right to keep unchanged

set2ZeroFromIndices(System.Int32,System.Int32,System.Int32,System.Int32) - Methode

Sets a submatrix to zero:

```
e.g. M = (    | 0 0 0 |    )
         ( M1 | 0 0 0 | M2 )
         (    | 0 0 0 |    )
         (       M3        )
```

Parameters:

firstRow	The row from which to start on
lastRow	The last row that should be considered
firstCol	The column from which to start on
lastCol	The last column that should be considered

set2Val(System.Double) - Methode

Sets a matrix to the value given as parameter.

Parameters:

v	The double value to set the elements to

set2Val(System.Int32,System.Int32,System.Int32,System.Int32,System.Double) - Methode

Sets a submatrix to the value given as parameter:

```
e.g. M = (    | v v v |    )
         ( M1 | v v v | M2 )
         (    | v v v |    )
         (       M3        )
```

Parameters:

top	The number of rows at the top to keep unchanged
bottom	The number of rows at the bottom to keep unchanged
left	The number of columns on the left to keep unchanged
right	The number of columns on the right to keep unchanged
v	The double value to set the elements to

set2ValFromIndices(System.Int32,System.Int32,System.Int32,System.Int32,System.Double) - Methode

Sets a submatrix to the value given as parameter:

```
e.g. M = (    | v v v |    )
         ( M1 | v v v | M2 )
         (    | v v v |    )
         (       M3        )
```

Parameters:

firstRow	The row from which to start on
lastRow	The last row that should be considered
firstCol	The column from which to start on
lastCol	The last column that should be considered
v	The double value to set the elements to

ToString - Methode

Returns a String of the Matrix

Return:	String of matrix

Item(System.Int32,System.Int32) - Property

Implements the [] operator.

TaylorPLib_CSharp

Class: MathException

Namespace: LibMatrix

Exception Class for own Exceptions in Matrix and Polynomial

MathException - Constructor

Base Constructor

MathException - Constructor

Base Constructor with initializing the Message

Parameters:

message	Message the Exception should throw

what - Methode

Returns the Exception description

Return:	String of the exception description