Class LinearAlgebra

```
java.lang.Object
  └LinearAlgebra
```

```
public class LinearAlgebra
extends java.lang.Object
```

A library of linear algebra algorithms originally created in Python by Massimo Di Pierro and ported to Java. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

private static double	**ap**
private static int	**ns**
private static int	**p**
private TestMatrix	**portfolio**
private double	**portfolio_return**
private double	**portfolio_risk**
private static double	**rp**

Constructor Summary

LinearAlgebra()

Method Summary

TestMatrix	**Cholesky**(TestMatrix A) Returns a TestMatrix object with the Cholesky algorithm applied.
TestMatrix	**exp**(TestMatrix x) Returns the exponent of a TestMatrix object.
TestMatrix	**getMarkovitzPortfolio**() Get method to return Markovitz portfolio value.
double	**getMarkovitzPortfolioReturn**() Get method to return Markovitz portfolio return.
double	**getMarkovitzPortfolioRisk**() Get method to return Markovitz portfolio risk.
boolean	**is_almost_symmetric**(TestMatrix x) Returns a boolean value indicating whether the matrix is almost symmetric.
boolean	**is_almost_zero**(TestMatrix A) Returns a boolean value indicating if a matrix is almost zero.
boolean	**is_positive_definite**(TestMatrix A) Returns a boolean value indicating if a TestMatrix is positive definite.
LinearAlgebra	**Markovitz**(TestMatrix mu, TestMatrix A, double r_free) Calculates the Markovitz portfolio, risk and return.
double	**norm**(double A) Returns the norm of a double value.
double	**norm**(TestMatrix A) Returns the norm of a TestMatrix.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

ap

private static double **ap**

---

rp

```
private static double rp
```

---

ns

```
private static int ns
```

---

p

```
private static int p
```

---

portfolio

```
private TestMatrix portfolio
```

---

portfolio_return

```
private double portfolio_return
```

---

portfolio_risk

```
private double portfolio_risk
```

Constructor Detail

LinearAlgebra

```
public LinearAlgebra()
```

Method Detail

is_almost_symmetric

```
public boolean is_almost_symmetric(TestMatrix x)
```

Returns a boolean value indicating whether the matrix is almost symmetric.

Parameters:

x - The TestMatrix object to be examined.

Returns:
The boolean result of the test.
Exception(s):
No known exceptions.
See Also:
TestMatrix

---

is_almost_zero

```
public boolean is_almost_zero(TestMatrix A)
```

Returns a boolean value indicating if a matrix is almost zero.

Parameters:
A - The TestMatrix object to be examined.
Returns:
Boolean result of the test.
Exception(s):
No known exceptions.
See Also:
TestMatrix

---

norm

```
public double norm(double A)
```

Returns the norm of a double value.

Parameters:
A - The value to be examined.
Returns:
The norm of A.
Exception(s):
No known sinces

---

norm

```
public double norm(TestMatrix A)
```

Returns the norm of a TestMatrix. Needs work. Not properly implemented.

Parameters:
A - The TestMatrix object to be examied.
Returns:
The norm of the matrix.

Exception(s):
Norm will always be zero. Not properly implemented.

---

exp

```
public TestMatrix exp(TestMatrix x)
```

Returns the exponent of a TestMatrix object.

Parameters:
`x` - The TestMatrix object to apply the function to.
Returns:
The exponent TestMatrix.
Exception(s):
Algorithm may fail to converge, division by zero errors.
See Also:
`TestMatrix`

Cholesky

```
public TestMatrix Cholesky(TestMatrix A)
```

Returns a TestMatrix object with the Cholesky algorithm applied.

Parameters:
`A` - The TestMatrix object to apply Cholesky to.
Returns:
A TestMatrix with Cholesky applied.
Exception(s):
Can't take a square root of a negative number.
See Also:
`TestMatrix`

---

is_positive_definite

```
public boolean is_positive_definite(TestMatrix A)
```

Returns a boolean value indicating if a TestMatrix is positive definite.

Parameters:
`A` - The TestMatrix to test for positive definite.
Returns:
The boolean result of the algorithm.
Exception(s):
Run time error possible.

See Also:
TestMatrix

Markovitz

```
public LinearAlgebra Markovitz(TestMatrix mu,
                               TestMatrix A,
                               double r_free)
```

Calculates the Markovitz portfolio, risk and return. Returns a reference to LinearAlgebra from which the Markovitz portfolio TestMatrix, risk and return can be obtained with get methods.

Parameters:
`mu` - Markovitz mu.
`A` - The TestMatrix object.
`r_free` - The risk free rate.
Returns:
LinearAlgebra reference to get portfolio, risk and return
Exception(s):
TestMatrix should be symmetric. Rows in mu should mirror columns in A
See Also:
getMarkovitzPortfolio(), getMarkovitzPortfolioRisk(), getMarkovitzPortfolioReturn()

---

getMarkovitzPortfolio

```
public TestMatrix getMarkovitzPortfolio()
```

Get method to return Markovitz portfolio value.

Returns:
Portfolio TestMatrix object
Exception(s):
Markovitz must be run and the value set prior to using this algorithm.
See Also:
Markovitz(TestMatrix, TestMatrix, double), TestMatrix

---

getMarkovitzPortfolioRisk

```
public double getMarkovitzPortfolioRisk()
```

Get method to return Markovitz portfolio risk.

Returns:
Markovitz portfolio risk.

Exception(s):

Markovitz must be run and the value set prior to using this algorithm.

See Also:

`Markovitz(TestMatrix, TestMatrix, double)`

---

getMarkovitzPortfolioReturn

`public double` **`getMarkovitzPortfolioReturn`**`()`

Get method to return Markovitz portfolio return.

Returns:

Markovitz portfolio return.

Exception(s):

Markovitz must be run and the value set prior to using this algorithm.

See Also:

`Markovitz(TestMatrix, TestMatrix, double)`

Class RunMe

```
java.lang.Object
  └RunMe
```

```
public class RunMe
extends java.lang.Object
```

Used to demonstrate the functionality of the mathematical library. Algorithms originally created in Python by Massimo Di Pierro and ported to Java. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
TestMatrix, LinearAlgebra, TestFunctionAbstract, TestFunction, TestFunction2, TestFunction3, TestFunction4, TestFunction5, TestFunction6, TestFunction7, Code Repository

Field Summary

private static LinearAlgebra	**LA**
private static TestFunction3	**P**
private static TestFunction4	**Q**
private static java.text.DecimalFormat	**twelveD**
private static java.text.DecimalFormat	**twoD**
private static TestFunction	**Y**
private static TestFunction2	**Z**

Constructor Summary

RunMe()

Method Summary

static void	**main**(java.lang.String[] args) Runs all test methods.
static void	**Test1**() Tests inverse matrix as implemented in class using c++.
static void	**Test10**() Tests optimize bisection for a function extended from TestFunctionAbstract.
static void	**Test11**() Tests optimize newton for a function extended from TestFunctionAbstract.
static void	**Test12**() Tests optimize secant for a function extended from TestFunctionAbstract.
static void	**Test13**() Tests optimize newton stabilized for a function extended from TestFunctionAbstract.
static void	**Test14**() Tests optimize golden search for a function extended from TestFunctionAbstract.
static void	**Test15**() Tests first and second derivatives for a function extended from TestFunctionAbstract.
static void	**Test16**() Tests for basic TestMatrix math functionality.
static void	**Test2**() Tests Cholesky as implemented in test096 from Massimo Ei Pierro's numeric.py.
static void	**Test3**() Tests Markovitz as implemented in the orignal Markovitz by Massimo Di Pierro in numeric.py
static void	**Test35**() Tests the condition number for doubles.
static void	**Test4**() Tests fit least squares for TestFunctionAbstract array of functions.
static void	**Test5**() Tests solve fixed point for a function extended from TestFunctionAbstract.
static void	**Test6**() Tests solve bisection for a function extended from TestFunctionAbstract.
static void	**Test7**() Tests solve solve newton for a function extended from TestFunctionAbstract.

static void	**Test8**() Tests solve secant for a function extended from TestFunctionAbstract.
static void	**Test9**() Tests solve newton stabilized for a function extended from TestFunctionAbstract.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

Y

private static TestFunction **Y**

Z

private static TestFunction2 **Z**

P

private static TestFunction3 **P**

Q

private static TestFunction4 **Q**

LA

private static LinearAlgebra **LA**

---

twoD

private static java.text.DecimalFormat **twoD**

---

twelveD

```
private static java.text.DecimalFormat twelveD
```

Constructor Detail

RunMe

```
public RunMe()
```

Method Detail

Test1

```
public static void Test1()
```

Tests inverse matrix as implemented in class using c++.

Exception(s):
Fails when method is incorrect.
See Also:
TestMatrix, TestMatrix.invMatrix(), TestMatrix.mulMatrix(TestMatrix)

Test2

```
public static void Test2()
```

Tests Cholesky as implemented in test096 from Massimo Ei Pierro's numeric.py.

Exception(s):
Fails when method is incorrect.
See Also:
TestMatrix, LinearAlgebra, LinearAlgebra.Cholesky(TestMatrix)

Test3

```
public static void Test3()
```

Tests Markovitz as implemented in the orignal Markovitz by Massimo Di Pierro in numeric.py

Exception(s):
Fails when method is incorrect.
See Also:
TestMatrix, LinearAlgebra, LinearAlgebra.Markovitz(TestMatrix, TestMatrix, double), LinearAlgebra.getMarkovitzPortfolio(), LinearAlgebra.getMarkovitzPortfolioReturn(),

LinearAlgebra.getMarkovitzPortfolioReturn()

Test35

```
public static void Test35()
```

Tests the condition number for doubles.

Exception(s):
Fails when method is incorrect. The condition number for test matrix is not implemented.
See Also:
TestMatrix, TestMatrix.condition_number()

Test4

```
public static void Test4()
```

Tests fit least squares for TestFunctionAbstract array of functions.

Exception(s):
Not yet implemented.
See Also:
TestFunctionAbstract, TestFunctionAbstract.fit_least_squares()

Test5

```
public static void Test5()
```

Tests solve fixed point for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.solve_fixed_point(double), TestFunction3

Test6

```
public static void Test6()
```

Tests solve bisection for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.

See Also:
TestFunctionAbstract, TestFunctionAbstract.solve_bisection(double, double), TestFunction4

Test7

```
public static void Test7()
```

Tests solve solve newton for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.solve_newton(double), TestFunction4

Test8

```
public static void Test8()
```

Tests solve secant for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.solve_secant(double), TestFunction4

Test9

```
public static void Test9()
```

Tests solve newton stabilized for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.solve_newton_stabilized(double, double), TestFunction4

Test10

```
public static void Test10()
```

Tests optimize bisection for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.optimize_bisection(double, double), TestFunction4

Test11

```
public static void Test11()
```

Tests optimize newton for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.optimize_newton(double), TestFunction4

Test12

```
public static void Test12()
```

Tests optimize secant for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.optimize_secant(double), TestFunction4

Test13

```
public static void Test13()
```

Tests optimize newton stabilized for a function extended from TestFunctionAbstract.

Exception(s):
Fails when method is incorrect.
See Also:
TestFunctionAbstract, TestFunctionAbstract.optimize_newton_stabilized(double, double), TestFunction4

Test14

```
public static void Test14()
```

Tests optimize golden search for a function extended from TestFunctionAbstract.

Exception(s):

Fails when method is incorrect.

See Also:

TestFunctionAbstract, TestFunctionAbstract.optimize_golden_search(double, double), TestFunction4

Test15

```
public static void Test15()
```

Tests first and second derivatives for a function extended from TestFunctionAbstract.

Exception(s):

Fails when method is incorrect.

See Also:

TestFunctionAbstract, TestFunctionAbstract.f(double), TestFunctionAbstract.Df(double), TestFunctionAbstract.DDf(double), TestFunction2

Test16

```
public static void Test16()
```

Tests for basic TestMatrix math functionality.

Exception(s):

Fails when method is incorrect.

See Also:

TestMatrix, TestMatrix.addMatrix(double), TestMatrix.addMatrix(TestMatrix), TestMatrix.changeMe(int, int, double), TestMatrix.condition_number(), TestMatrix.copyMe(), TestMatrix.divMatrix(double), TestMatrix.invMatrix(), TestMatrix.mulMatrix(double), TestMatrix.mulMatrix(TestMatrix), TestMatrix.mulMatrixScalar(TestMatrix), TestMatrix.printMe(), TestMatrix.subMatrix(double), TestMatrix.subMatrix(TestMatrix)

---

main

```
public static void main(java.lang.String[] args)
```

Runs all test methods.

Parameters:

`args` - Default for Java.

Exception(s):

Fails for incorrect methods.

Class TestFunction

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction
```

```
public class TestFunction
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula x*x-5.0*x. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract

ap, h, ns, rp

Constructor Summary

TestFunction()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function x*x-5.0*x.

Methods inherited from class TestFunctionAbstract

condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction

```
public TestFunction()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function x*x-5.0*x.

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction2

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction2
```

```
public class TestFunction2
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula x*x-5.0*x. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction2()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function x*x-5.0*x.

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction2

```
public TestFunction2()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function x*x-5.0*x.

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction3

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction3
```

```
public class TestFunction3
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula (x-2)*(x-5)/10. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction3()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function (x-2)*(x-5)/10.

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction3

```
public TestFunction3()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function (x-2)*(x-5)/10.

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction4

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction4
```

```
public class TestFunction4
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula (x-2)*(x-5). All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction4()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function (x-2)*(x-5).

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction4

```
public TestFunction4()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function (x-2)*(x-5).

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction5

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction5
```

```
public class TestFunction5
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula x. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction5()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function x.

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction5

```
public TestFunction5()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function x.

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction6

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction6
```

```
public class TestFunction6
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula 5+0.8*x+0.3*x*x+Math.sin(x). All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction6()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function 5+0.8*x+0.3*x*x+Math.sin(x).

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dg, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton,

optimize secant, solve bisection, solve fixed point, solve newton stabilized, solve newton, solve secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction6

```
public TestFunction6()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function 5+0.8*x+0.3*x*x+Math.sin(x).

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunction7

```
java.lang.Object
  └TestFunctionAbstract
      └TestFunction7
```

```
public class TestFunction7
extends TestFunctionAbstract
```

TestFunctionAbstract extended as the formula 2. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
Code Repository

Field Summary

Fields inherited from class TestFunctionAbstract
ap, h, ns, rp

Constructor Summary

TestFunction7()

Method Summary

double	**f**(double x) Implementation of the abstract method f with the function 2.

Methods inherited from class TestFunctionAbstract
condition_number, condition_number, DDf, Df, Dq, fit_least_squares, g, optimize_bisection, optimize_golden_search, optimize_newton_stabilized, optimize_newton, optimize_secant, solve_bisection, solve_fixed_point, solve_newton_stabilized,

solve_newton, solve_secant

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TestFunction7

```
public TestFunction7()
```

Method Detail

f

```
public double f(double x)
```

Implementation of the abstract method f with the function 2.

Specified by:
f in class TestFunctionAbstract

Parameters:
x - Value used to evaluate the function with.

Returns:
The result of evaluating the function for x.

Exception(s):
No known exceptions.

Class TestFunctionAbstract

```
java.lang.Object
  └TestFunctionAbstract
```

Direct Known Subclasses:

TestFunction, TestFunction2, TestFunction3, TestFunction4, TestFunction5, TestFunction6, TestFunction7

```
public abstract class TestFunctionAbstract
extends java.lang.Object
```

An abstract method requiring extension of the method f (a function) which can then be used with this library of algorithms originally created in Python by Massimo Di Pierro and ported to Java. All code released under BSD licensing.

Version:

0.1

Author:

Ruthann Sudman

See Also:

Code Repository

Field Summary

private static LinearAlgebra	**A**
static double	**ap**
static double	**h**
static int	**ns**
static double	**rp**

Constructor Summary

TestFunctionAbstract()

Method Summary

double	**condition_number**(double x) Evaluates the condition number of the abstract function f.
double	**condition_number**(TestMatrix f) Evaluates the condition number of the TestMatrix f.
double	**DDf**(double x) The second derivative for the abstract function f.
double	**Df**(double x) The first derivative for the abstract function f.
double	**Dg**(double x) The first derivative of the function g.
abstract double	**f**(double x) An abstract function method to be extended by daughter classes.
TestMatrix	**fit_least_squares**() Evaluates the abstract function f for fit least squares.
double	**g**(double x) The abstract function f plus x
double	**optimize_bisection**(double a, double b) Optimized bisection for the abstract function f in (a,b).
double	**optimize_golden_search**(double a, double b) Optimizes golden search for the abstract function f in (a,b).
double	**optimize_newton_stabilized**(double a, double b) Optimization of newton stabilized for the abstract function f.
double	**optimize_newton**(double x_guess) Newton optimized for the abstract function f.
double	**optimize_secant**(double x) Optimized secant for the abstract function f.
double	**solve_bisection**(double a, double b) Solves bisection for the abstract function f.
double	**solve_fixed_point**(double x) Solves fixed point for the abstract function f.
double	**solve_newton_stabilized**(double a, double b) Solves newton stabilized for the abstract function f in (a,b).
double	**solve_newton**(double x_guess) Solves newton for the abstract function f.

double	**solve_secant**(double x) Solves secant for the abstract function f.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

h

public static double **h**

---

ap

public static double **ap**

---

rp

public static double **rp**

---

ns

public static int **ns**

A

private static LinearAlgebra **A**

Constructor Detail

TestFunctionAbstract

public **TestFunctionAbstract**()

Method Detail

f

```
public abstract double f(double x)
```

An abstract function method to be extended by daughter classes.

Parameters:
x - The value used to evaluate the function.
Returns:
The result of evaluating the function for x.
Exception(s):
No known exceptions.

Df

```
public double Df(double x)
```

The first derivative for the abstract function f.

Parameters:
x - The value used to evaluate the first derivative.
Returns:
The result of evaluating the first derivative for x.
Exception(s):
No known exceptions.

DDf

```
public double DDf(double x)
```

The second derivative for the abstract function f.

Parameters:
x - The value used to evaluate the second derivative.
Returns:
The result of evaluating the second derivative for x.
Exception(s):
No known exceptions.

---

g

```
public double g(double x)
```

The abstract function f plus x

Parameters:
x - The value used to evaluate the second derivative.

Returns:
The result of evaluating the new function for x.
Exception(s):
No known exceptions.

Dg

```
public double Dg(double x)
```

The first derivative of the function g.

Parameters:
`x` - The value used to evaluate the first derivative of g.
Returns:
The result of evaluating the first derivative of g
Exception(s):
No known exceptions.

---

condition_number

```
public double condition_number(double x)
```

Evaluates the condition number of the abstract function f.

Parameters:
`x` - The value used to evaluate the condition number.
Returns:
The condition number for the abstract function f.
Exception(s):
Does not work when the f(x) evaluates to zero.

---

condition_number

```
public double condition_number(TestMatrix f)
```

Evaluates the condition number of the TestMatrix f.

Parameters:
`f` - The TestMatrix to be evaluated for condition number.
Returns:
The condition number for the TestMatrix f.
Exception(s):
This function has not been properly implemented.
See Also:
TestMatrix

fit_least_squares

```
public TestMatrix fit_least_squares()
```

Evaluates the abstract function f for fit least squares.

Returns:
A TestMatrix of the least squares fit
Exception(s):
This function has not been properly implemented, returns 0.
See Also:
TestMatrix

---

solve_fixed_point

```
public double solve_fixed_point(double x)
```

Solves fixed point for the abstract function f.

Parameters:
x - The value used to solve fixed point.
Returns:
Fixed point of the abstract function f for x.
Exception(s):
Does not work when the first derivative is greater than or equal to 1. Does not work if fixed point does not converge for x.

---

solve_bisection

```
public double solve_bisection(double a,
                              double b)
```

Solves bisection for the abstract function f.

Parameters:
a - The low value to examine the function.
b - The high value to examine the function.
Returns:
Bisection for abstract function f in (a,b).
Exception(s):
f(a) and f(b) must have opposite signs. Does not work when bisection does not converge for f in range (a,b).

---

solve_newton

```
public double solve_newton(double x_guess)
```

Solves newton for the abstract function f.

Parameters:
`x_guess` - The result guess for newton.
Returns:
Newton for abstract function f in x.
Exception(s):
Does not work when newton does not converge for f in x.

---

solve_secant

```
public double solve_secant(double x)
```

Solves secant for the abstract function f.

Parameters:
`x` - The value used to evaluate the abstract function f for secant.
Returns:
Secant of the abstract function f in x.
Exception(s):
If the norm of the function is less than the absolute function. If the secant does not converge for abstract function f in x.

---

solve_newton_stabilized

```
public double solve_newton_stabilized(double a,
                                      double b)
```

Solves newton stabilized for the abstract function f in (a,b).

Parameters:
`a` - The low value for f.
`b` - The high value for f.
Returns:
Newton stabilized for the abstract function f in (a,b).
Exception(s):
f(a) and f(b) must evaluate with opposite signs. Does not work if newton stabilized does not converge.

---

optimize_bisection

```
public double optimize_bisection(double a,
                                 double b)
```

Optimized bisection for the abstract function f in (a,b).

Parameters:
`a` - The low value.
`b` - The high value.
Returns:
Optimized bisection for the abstract function f in (a,b).
Exception(s):
Df(a) and Df(b) must evaluate with opposite signs. Does not work when bisection does not converge for f in (a,b).

---

optimize_newton

```
public double optimize_newton(double x_guess)
```

Newton optimized for the abstract function f.

Parameters:
`x_guess` - The guess for newton.
Returns:
Newton optimized for the abstract function f in x.
Exception(s):
Does not work if newton does not converge for f in x.

---

optimize_secant

```
public double optimize_secant(double x)
```

Optimized secant for the abstract function f.

Parameters:
`x` - The value used to evaluate secant for f.
Returns:
Optimized secant for the abstract function f.
Exception(s):
Does not work if DDf(x) is less than absolute precision. Does not work if optimize secant does not converge for f in x.

---

optimize_newton_stabilized

```
public double optimize_newton_stabilized(double a,
                                         double b)
```

Optimization of newton stabilized for the abstract function f.

Parameters:

a - The low value.

b - The high value.

Returns:

Optimized newton stabilized for the abstract function f.

Exception(s):

Df(a) and Df(b) must evaluate with opposite signs. Does not work if newton does not converge for the abstract function f in (a,b).

---

optimize_golden_search

```
public double optimize_golden_search(double a,
                                     double b)
```

Optimizes golden search for the abstract function f in (a,b).

Parameters:

a - The low value.

b - The high value.

Returns:

The optimized golden search for abstract function f.

Exception(s):

Does not work if golden search cannot be optimized for the abstract function f in (a,b).

Class TestMatrix

```
java.lang.Object
  └TestMatrix
```

```
public class TestMatrix
extends java.lang.Object
```

An implementation of basic matrix math which can be used with a library of linear algebra algorithms originally created in Python by Massimo Di Pierro and ported to Java. All code released under BSD licensing.

Version:
0.1
Author:
Ruthann Sudman
See Also:
LinearAlgebra, Code Repository

Field Summary

private int	**myCols**
private double[][]	**myData**
private int	**myRows**

Constructor Summary

TestMatrix(int rows, int columns) TestMatrix constructor, initializing the original matrix to all 0.

Method Summary

TestMatrix	**addMatrix**(double x) Add a value to all elements in the TestMatrix.

TestMatrix	**addMatrix**(TestMatrix otherData) Add two TestMatrices together.
void	**changeMe**(int row, int column, double myvalue) Updates a specific value in the myData.
double	**condition_number**() Return the condition number of myData.
TestMatrix	**copyMe**() Return a copy of myData.
TestMatrix	**divMatrix**(double x) Divide all elements in a TestMatrix by x.
int	**getColumns**() Get method that returns the number of columns in the TestMatrix.
double	**getMe**(int row, int column) Obtain a specific value in the myData.
int	**getRows**() Get method that returns the number of rows in the TestMatrix.
TestMatrix	**identity**() Construct a diagonal matrix identical in size to myData.
TestMatrix	**invMatrix**() The inverse of a TestMatrix object.
TestMatrix	**mulMatrix**(double x) Multiply all elements in a TestMatrix by a value.
TestMatrix	**mulMatrix**(TestMatrix otherData) Multiply two TestMatrices together.
double	**mulMatrixScalar**(TestMatrix B) Do a scalar multiplication of two matrices.
double	**norm**() Return the norm of myData
void	**printMe**() Print out the myData in a single line.
TestMatrix	**subMatrix**(double x) Subtract a specific value from all elements in the TestMatrix.
TestMatrix	**subMatrix**(TestMatrix otherData) Subtract one TestMatrix from another.
private void	**swapMe**(int r1, int c1, int r2, int c2) Swap two values in myData.
private void	**updateAddMe**(int row, int column, double myvalue) Adds a value to a specific value in the myData.
void	**updateSubMe**(int row, int column, double myvalue)

	Subtracts a specific value in the myData.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

myRows

```
private int myRows
```

---

myCols

```
private int myCols
```

---

myData

```
private double[][] myData
```

Constructor Detail

TestMatrix

```
public TestMatrix(int rows,
                  int columns)
```

TestMatrix constructor, initializing the original matrix to all 0.

Parameters:

`rows` - Number of rows in the TestMatrix.
`columns` - Number of Columns in the TestMatrix.

Exception(s):

TestMatrix need to have at least 1 row and 1 column.

Method Detail

getRows

```
public int getRows()
```

Get method that returns the number of rows in the TestMatrix.

Returns:
Number of rows in the TestMatrix myRows.
Exception(s):
No known exceptions.

---

getColumns

```
public int getColumns()
```

Get method that returns the number of columns in the TestMatrix.

Returns:
The number of columns in the TestMatrix myCols.
Exception(s):
No known exceptions.

---

changeMe

```
public void changeMe(int row,
                     int column,
                     double myvalue)
```

Updates a specific value in the myData.

Parameters:
`row` - The row of the value to update.
`column` - The column of the value to update.
`myvalue` - The update value.
Exception(s):
row and column must be in the bounds of the matrix myData.

---

updateAddMe

```
private void updateAddMe(int row,
                         int column,
                         double myvalue)
```

Adds a value to a specific value in the myData.

Parameters:
`row` - The row of the value to add to.
`column` - The column of the value to add to.
`myvalue` - The value to add to the original number.

Exception(s):
row and column must be in the bounds of the matrix myData.

---

updateSubMe

```
public void updateSubMe(int row,
                        int column,
                        double myvalue)
```

Subtracts a specific value in the myData.

Parameters:
`row` - The row of the value to subtract from.
`column` - The column of the value to subtract.
`myvalue` - The value to subtract from the original number.
Exception(s):
row and column must be in the bounds of the matrix myData.

---

getMe

```
public double getMe(int row,
                    int column)
```

Obtain a specific value in the myData.

Parameters:
`row` - The row of the desired value.
`column` - The column of the desired value.
Returns:
The desired value to return from myData.
Exception(s):
row and column must be in the bounds of the matrix myData.

---

printMe

```
public void printMe()
```

Print out the myData in a single line.

Exception(s):
Printing does not work well for TestMatrices with column = 1.

---

addMatrix

```
public TestMatrix addMatrix(TestMatrix otherData)
```

Add two TestMatrices together.

Parameters:
`otherData` - The TestMatrix to add to myData.
Returns:
The added TestMatrx.
Exception(s):
The rows and columns of both matrices must be equal.

---

addMatrix

```
public TestMatrix addMatrix(double x)
```

Add a value to all elements in the TestMatrix.

Parameters:
`x` - The value to add to all elements of the TestMatrix.
Returns:
The TestMatrix with the addition of x performed.
Exception(s):
No known exceptions

---

subMatrix

```
public TestMatrix subMatrix(TestMatrix otherData)
```

Subtract one TestMatrix from another.

Parameters:
`otherData` - The matrix to subtract from myData.
Returns:
The subtracted TestMatrix.
Exception(s):
The rows and columns of both matrices must be equal.

---

subMatrix

```
public TestMatrix subMatrix(double x)
```

Subtract a specific value from all elements in the TestMatrix.

Parameters:
`x` - The value to subtract from all elements in myData.

Returns:
myData with x subtracted.
Exception(s):
No known exceptions.

---

mulMatrix

`public TestMatrix mulMatrix(double x)`

Multiply all elements in a TestMatrix by a value.

Parameters:
`x` - The value to multiply all elements in myData by.
Returns:
myData multiplied by x.
Exception(s):
No known exceptions.

---

mulMatrixScalar

`public double mulMatrixScalar(TestMatrix B)`

Do a scalar multiplication of two matrices.

Parameters:
`B` - The TestMatrix to be multiplied with myData.
Returns:
The scalar multiplication of myData and TestMatrix B.
Exception(s):
The number of rows for both TestMatrices must be one and the number of columns must be equal OR the number of columns for both TestMatrices must be one and the number of rows must be equal.

---

mulMatrix

`public TestMatrix mulMatrix(TestMatrix otherData)`

Multiply two TestMatrices together.

Parameters:
`otherData` - The TestMatrix to multiply with myData.
Returns:
The multiplication of myData and TestMatrix otherData.
Exception(s):
The number of columns in myData must be equal to the number of rows in otherData.

---

divMatrix

```
public TestMatrix divMatrix(double x)
```

Divide all elements in a TestMatrix by x.

Parameters:
`x` - The value to divide all myData elements by.
Returns:
The muliplication of myData and x.
Exception(s):
No known exceptions.

---

swapMe

```
private void swapMe(int r1,
                    int c1,
                    int r2,
                    int c2)
```

Swap two values in myData.

Parameters:
`r1` - The row of the first value to swap.
`c1` - The column of the first value to swap.
`r2` - The row of the second value to swap.
`c2` - The column of the second value to swap.
Exception(s):
r1, c1, r2 and c2 must be indexes in range for myData.

---

copyMe

```
public TestMatrix copyMe()
```

Return a copy of myData. One cannot simply return myData because that would be returning a double array and not a TestMatrix object.

Returns:
TestMatrix

---

invMatrix

```
public TestMatrix invMatrix()
```

The inverse of a TestMatrix object.

Returns:
The inverse of myData.

---

identity

```
public TestMatrix identity()
```

Construct a diagonal matrix identical in size to myData.

Returns:
The identity matrix for myData
Exception(s):
The size of the identity matrix will be identical to myData. Size is not customizable.

---

norm

```
public double norm()
```

Return the norm of myData

Returns:
Norm of matrix myData
Exception(s):
This function is not properly implemented.

---

condition_number

```
public double condition_number()
```

Return the condition number of myData.

Returns:
The condition number of myData.
Exception(s):
This function is not properly implemented.
