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. exceptions No known exceptions.

Returns:

The result of evaluating the function for x.

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):
`java.lang.ArithmeticException` - 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):
`java.lang.ArithmeticException` - 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):
`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):
`java.lang.ArithmeticException` - 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):
`java.lang.ArithmeticException` - 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):
`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

`java.lang.ArithmeticException` - 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):

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