temfpy

dev-team temfpy

Nov 20, 2020

CONTENTS

1 Nonlinear equations 3

2 Optimization 9

3 Uncertainty quantification 15

4 Acknowledgement 19

5 Bibliography 21

Bibliography 23

Python Module Index 25

Index 27

`temfpy` is an open-source package providing test models and functions for standard numerical components in computational economic models.

With `conda` available on your path, installing and testing `temfpy` is as simple as typing

```
$ conda install -c opensourceeconomics temfpy
$ python -c "import temfpy; temfpy.test()"
```

Supported by

CHAPTER

ONE

NONLINEAR EQUATIONS

We provide a variety of non-linear equations used for testing numerical optimization algorithms.

`temfpy.nonlinear_equations.`**`broyden`**(*x*, *a=3, 0.5, 2, 1*, *jac=False*)

Broyden tridiagonal function.

$$\begin{aligned} x &\mapsto \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T \\ F_1(x) &= x_1(a_1 - a_2 x_1) - a_3 x_2 + a_4 \\ F_i(x) &= x_i(a_1 - a_2 x_i) - x_{i-1} - a_3 x_{i+1} + a_4 \\ i &= 2, 3, \dots, p-1 \\ F_p(x) &= x_p(a_1 - a_2 x_p) - x_{p-1} + a_4 \end{aligned}$$

Parameters

- **x** (*array_like*) – Input domain with dimension $p > 1$.
- **a** (*array_like, optional*) – The default array is (3, 0.5, 2, 1).
- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain.
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian.

References

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import broyden
>>>
>>> np.random.seed(123)
>>> p = np.random.randint(3,20)
>>> x = np.zeros(p)
>>> np.allclose(broyden(x), np.repeat(1,p))
True
```

`temfpy.nonlinear_equations.`**`chandrasekhar`**(*x, y, c, jac=False*)

Discretized version of Chandrasekhar's H-equation.

$$x \mapsto \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T$$
$$F_i(x) = x_i - \left(1 - \frac{c}{2p} \sum_{j=1}^{p} \frac{y_i x_j}{y_i + y_j}\right)^{-1}$$
$$i = 1, 2, \dots, p$$

Parameters

- **x** (*array_like*) – Input domain with dimension p.
- **y** (*array_like,*) – Array of constants with dimension p
- **c** (*float*) – Constant parameter
- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian. Numerically derived Jacobian only if dimension $p > 1$.

References

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import chandrasekhar
>>>
>>> np.random.seed(123)
>>> p = np.random.randint(1,20)
>>> x = np.repeat(2,p)
>>> y = np.repeat(1,p)
>>> c = 1
>>> np.allclose(chandrasekhar(x,y,c), np.zeros(p))
True
```

`temfpy.nonlinear_equations.`**`exponential`**(*x, a=10, b=1, jac=False*)

Exponential function.

$$x \mapsto \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T$$
$$F_1(x) = e^{x_1} - b$$
$$F_i(x) = \frac{i}{a}(e^{x_i} + x_{i-1}) - b$$
$$i = 2, 3, \dots, p$$

Parameters

- **x** (*array_like*) – Input domain with dimension p.
- **a** (*float, optional*) – The default value is 10.
- **b** (*float, optional*) – The default value is 1.

- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian.

References

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import exponential
>>>
>>> np.random.seed(123)
>>> p = np.random.randint(1,20)
>>> x = np.zeros(p)
>>> np.allclose(exponential(x), np.zeros(p))
True
```

`temfpy.nonlinear_equations.`**`rosenbrock_ext`**(*x*, *a=10, 1*, *jac=False*)

Extended-Rosenbrock function.

$$\begin{aligned} x \mapsto & \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T \\ F_{2i-1}(x) &= a_1(x_{2i} - x_{2i-1}^2) \\ F_{2i}(x) &= a_2 - x_{2i-1}, \\ i &= 1, 2, 3, \dots, \frac{p}{2} \end{aligned}$$

Parameters

- **x** (*array_like*) – Input domain with even dimension $p > 1$.
- **a** (*array_like, optional*) – The default array is (10,1)
- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian.

References

BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function.

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import rosenbrock_ext
>>>
>>> np.random.seed(123)
>>> p = 2*np.random.randint(1,20)
>>> x = np.zeros(p)
>>> compare = np.resize([0,1], p)
>>> np.allclose(rosenbrock_ext(x), compare)
True
```

temfpy.nonlinear_equations.**trig_exp**(*x*, *a=3, 2, 5, 4, 3, 2, 8, 4, 3*, *jac=False*)

Trigonometrical exponential function.

$$
\begin{aligned}
x &\mapsto \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T \\
F_1(x) &= a_1 x_1^3 + a_2 x_2 - a_3 + \sin(x_1 - x_2)\sin(x_1 + x_2) \\
F_i(x) &= -x_{i-1} e^{x_{i-1} - x_i} + x_i (a_4 + a_5 x_i^2) + a_6 x_{i+1} \\
&\quad + \sin(x_i - x_{i+1}) \sin(x_i + x_{i+1}) - a_7 \\
&\quad i = 2, 3, \dots, p-1 \\
F_p(x) &= -x_{p-1} e^{x_{p-1} - x_p} + a_8 x_p - a_9
\end{aligned}
$$

Parameters

- **x** (*array_like*) – Input domain with dimension $p > 1$.
- **a** (*array_like, optional*) – The default array is (3,2,5,4,3,2,8,4,3).
- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian.

References

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import trig_exp
>>>
>>> np.random.seed(123)
>>> p = np.random.randint(3,20)
>>> x = np.zeros(p)
>>> compare = np.insert(np.array([-5,-3]), 1, np.repeat(-8, p-2))
>>> np.allclose(trig_exp(x), compare)
True
```

temfpy.nonlinear_equations.**troesch**(*x*, *rho=10*, *a=2*, *jac=False*)

Troesch function.

$$
\begin{aligned}
& x \mapsto \begin{pmatrix} F_1(x) & F_2(x) & \dots & F_p(x) \end{pmatrix}^T \\
& h = \frac{1}{p+1} \\
F_1(x) &= ax_1 + \rho h^2 \sinh(\rho x_1) - x_2, \\
F_i(x) &= ax_i + \rho h^2 \sinh(\rho x_i) - x_{i-1} - x_{i+1} \\
& i = 2, 3, \dots, p-1 \\
F_p(x) &= ax_p + \rho h^2 \sinh(\rho x_p) - x_{p-1}
\end{aligned}
$$

Parameters

- **x** (*array_like*) – Input domain with dimension $p > 1$.
- **rho** (*float, optional*) – The default value is 10
- **a** (*float, optional*) – The default value is 2
- **jac** (*bool*) – If True, an additional array containing the numerically and the analytically derived jacobian are returned. The default is False.

Returns

- *array_like* – Output domain
- *array_like* – Only if $jac = True$. Tuple containing the analytically derived Jacobian and the numerically derived Jacobian.

References

Examples

```
>>> import numpy as np
>>> from temfpy.nonlinear_equations import troesch
>>>
>>> np.random.seed(123)
>>> p = np.random.randint(1,20)
>>> x = np.zeros(p)
>>> np.allclose(troesch(x), np.zeros(p))
True
```

CHAPTER

TWO

OPTIMIZATION

We provide a host of models and functions that are often used for testing and benchmarking exercises in the numerical optimization literature.

`temfpy.optimization.`**`ackley`**`(`*x*, *a=20*, *b=0.2*, *c=6.283185307179586*`)`

Ackley function.

$$f(x) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Parameters

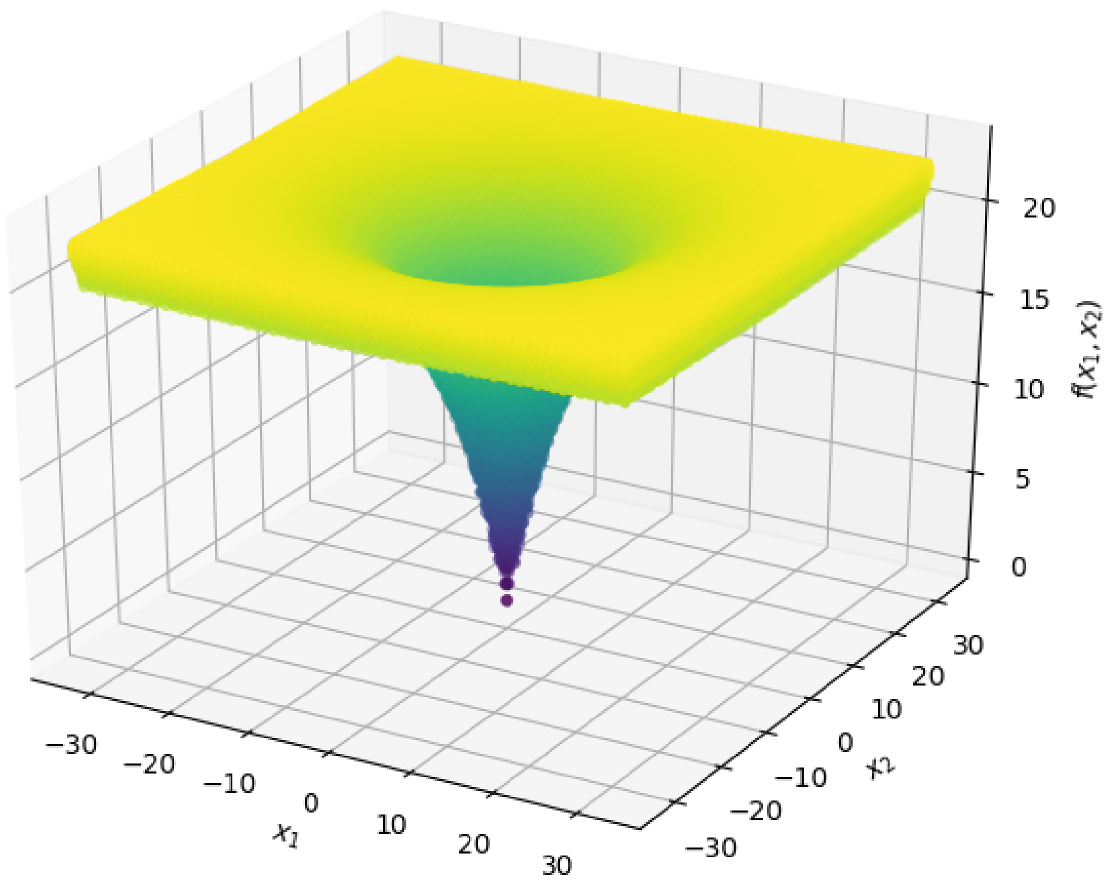
- **x** (*array_like*) – Input domain with dimension d. It is usually evaluated on the hypercube $x_i \in [-32.768, 32.768]$, for all $i = 1, \ldots, d$.
- **a** (*float, optional*) – The default value is 20.
- **b** (*float, optional*) – The default value is 0.2.
- **c** (*float, optional*) – The default value is 2.

Returns Output domain

Return type float

Notes

This function was proposed by David Ackley in [A1987] and used in [B1996] and [M2005]. It is characterized by an almost flat outer region and a central hole or peak where modulations become more and more influential. The function has its global minimum $f(x) = 0$ at $x = \begin{pmatrix}0 & \ldots & 0\end{pmatrix}^T$.



20
15
10
5
0
$f(x_1, x_2)$
−30
−20
−10
0
10
20
30
x_1
30
20
10
0
−10
−20
−30
x_2

References

Examples

```
>>> from temfpy.optimization import ackley
>>> import numpy as np
>>>
>>> x = [0, 0]
>>> y = ackley(x)
>>> np.testing.assert_almost_equal(y, 0)
```

`temfpy.optimization.`**`rastrigin`**(*x*, *a=10*)

Rastrigin function.

$$f(x) = ad + \sum_{i=1}^{d} \left(x_i^2 - 10\cos(2\pi x_i)\right)$$

Parameters

- **x** (*array_like*) – Input domain with dimension d. It is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, \ldots, d$.
- **a** (*float, optional*) – The default value is 10.

Returns Output domain

Return type float

Notes

The function was first proposed by Leonard Rastrigin in [R1974]. It produces frequent local minima as it is highly multimodal. However, the location of the minima are regularly distributed. The function has its global minimum $f(x) = 0$ at $x = \begin{pmatrix} 0 & \ldots & 0 \end{pmatrix}^T$.

References

Examples

```
>>> from temfpy.optimization import rastrigin
>>> import numpy as np
>>>
>>> x = [0, 0]
>>> y = rastrigin(x)
>>> np.testing.assert_almost_equal(y, 0)
```

`temfpy.optimization.`**`rosenbrock`**(*x*)

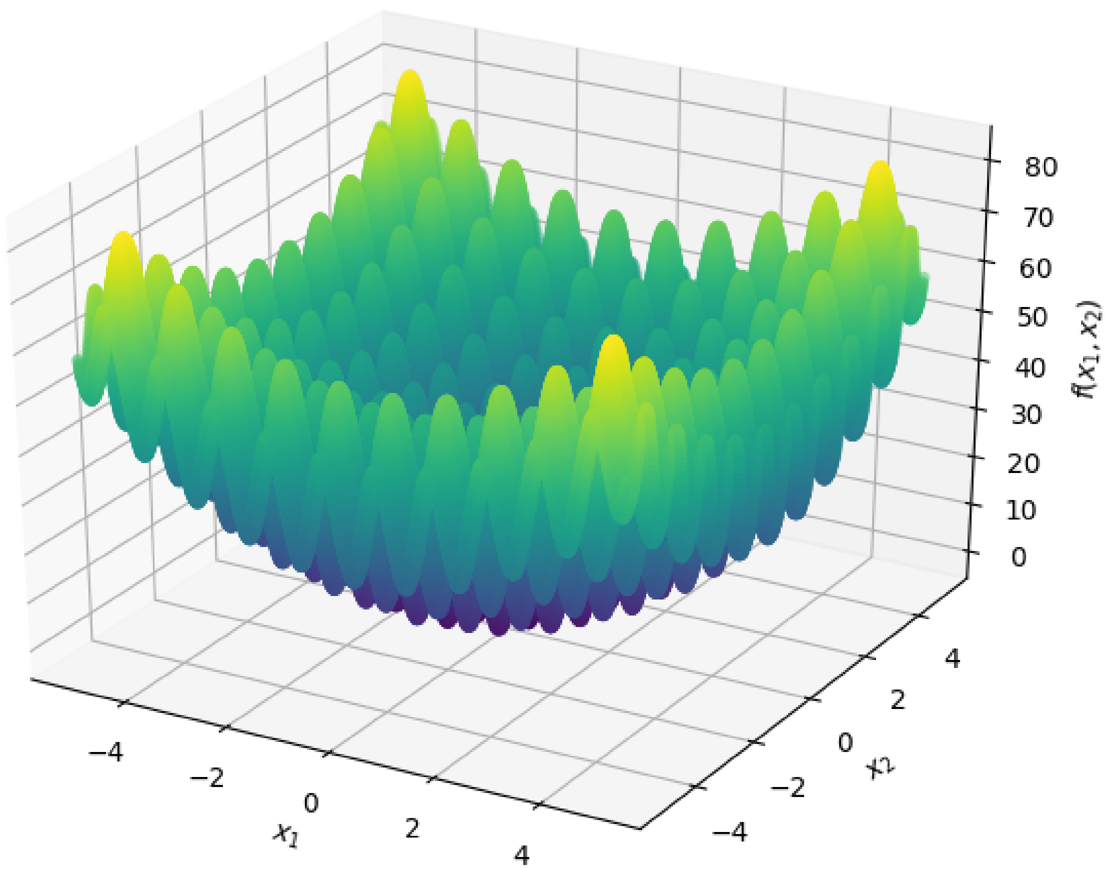
Rosenbrock function.

$$f(x) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i^2)\right]$$

Parameters **x** (*array_like*) – Input domain with dimension $d > 1$.

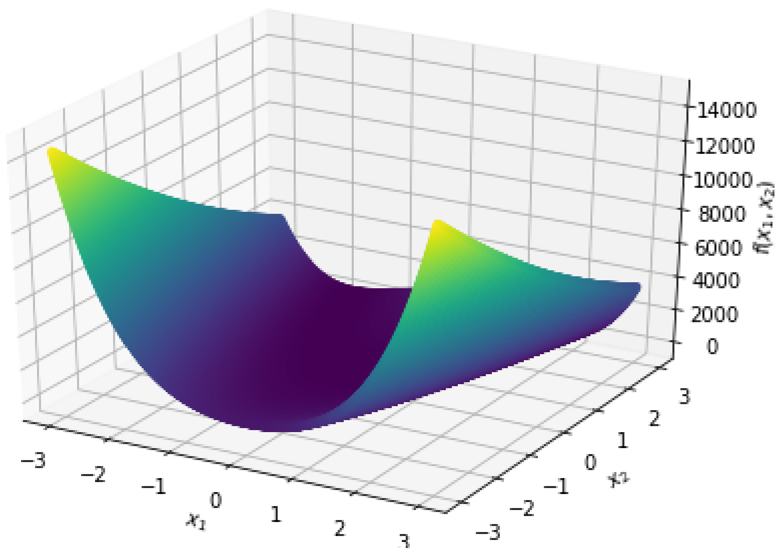
Returns Output domain

Return type float


80
70
60
50
40
30
20
10
0
$f(x_1, x_2)$
−4
−2
0
2
4
x_1
4
2
0
−2
−4
x_2

Notes

The function was first proposed by Howard H. Rosenbrock in [R1960] and is often also referred to, due to its shape, as Rosenbrock's valley or Rosenbrock's Banana function. The function has its global minimum at $x = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T$



References

Examples

```
>>> from temfpy.optimization import rosenbrock
>>> import numpy as np
>>>
>>> x = [1, 1]
>>> y = rosenbrock(x)
>>> np.testing.assert_almost_equal(y, 0)
```

CHAPTER

THREE

UNCERTAINTY QUANTIFICATION

We provide a host of models and functions that are often used for testing and benchmarking exercises in the uncertainty quantification literature.

`temfpy.uncertainty_quantification.`**`borehole`**`(x)`

Borehole function.

$$f(x) = \frac{2\pi x_1(x_2 - x_3)}{\ln(x_4/x_5)\left(1 + \frac{2x_1x_6}{\ln(x_4/x_5)x_5^2x_7} + \frac{x_1}{x_8}\right)}$$

Parameters **x** (*array_like*) – Core parameters of the model with dimension 8.

Returns Flow rate in m^3/yr.

Return type float

Notes

The Borehole function was developed by Harper and Gupta [H1983] to model steady state flow through a hypothetical borehole. It is widely used as a testing function for a variety of methods due to its simplicity and quick evaluation (e.g. [X2013]). Harper and Gupta used the function originally to compare the results of a sensitivity analysis to results based on Latin hypercube sampling.

References

Examples

```
>>> from temfpy.uncertainty_quantification import borehole
>>> import numpy as np
>>>
>>> x = [1, 2, 3, 4, 5, 6, 7, 8]
>>> y = borehole(x)
>>> np.testing.assert_almost_equal(y, 34.43500403827335)
```

`temfpy.uncertainty_quantification.`**`eoq_model`**`(x, r=0.1)`

Economic order quantity model.

$$y = \sqrt{\frac{24x_0x_2}{rx_1}}$$

Parameters

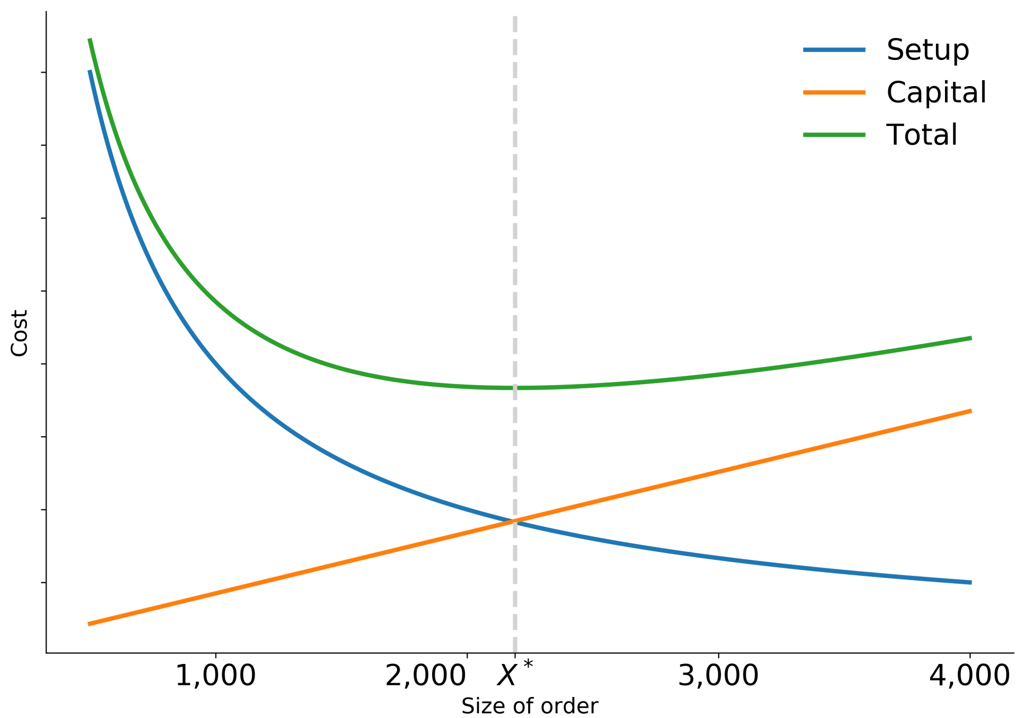
- **x** (*array_like*) – Core parameters of the model.
- **r** (*float, optional*) – Annual interest rate (default value is 0.1).

Returns **y** – Optimal order quantity.

Return type float

Notes

This function computes the optimal economic order quantity (EOQ) based on the model presented in [H1990]. The EOQ minimizes the holding costs as well as ordering costs. The core parameters of the model are the units per months x_0, the unit price of items in stock x_1, and the setup costs of an order x_2. The annual interest rate r is treated as an additional parameter. A historical perspective on the model is provided by [E1990]. A brief description with the core equations is available in [W2020]. The figure below illustrates the core trade-off in the model. Holding x_1 and x_2 constant, an increase in x_0 results in a decrease in the setup cost per unit, but an increase in capital cost increases as the stock of inventory increase.



References

Examples

```
>>> from temfpy.uncertainty_quantification import eoq_model
>>> import numpy as np
>>>
>>> x = [1, 2, 3]
>>> y = eoq_model(x, r=0.1)
>>> np.testing.assert_almost_equal(y, 18.973665961010276)
```

`temfpy.uncertainty_quantification.`**`ishigami`**(*x*, *a=7*, *b=0.05*)

Ishigami function.

$$f(x) = \sin(x_1) + a\sin^2(x_2) + bx_3^4\sin(x_1)$$

Parameters

- **x** (*array_like*) – Core parameters of the model with dimension 3.
- **a** (*float, optional*) – The default value is 7, as used by Sobol' and Levitan in [S1999].
- **b** (*float, optional*) – The default value is 0.05, as used by Sobol' and Levitan.

Returns Output domain

Return type float

Notes

This function was specifically developed by Ishigami and Homma [I1990] as a test function used for uncertainty analysis. It is characterized by its strong nonlinearity and nonmonotonicity. Sobol' and Levitan note that the Ishigami function has a strong dependence on x_2.

References

Examples

```
>>> from temfpy.uncertainty_quantification import ishigami
>>> import numpy as np
>>>
>>> x = [1, 2, 3]
>>> y = ishigami(x)
>>> np.testing.assert_almost_equal(y, 10.037181146302519)
```

`temfpy.uncertainty_quantification.`**`simple_linear_function`**(*x*)

Uncomplicated linear function.

This function computes the sum of all elements of a given array.

Parameters **x** (*array_like*) – Array of summands

Examples

```
>>> from temfpy.uncertainty_quantification import simple_linear_function
>>> import numpy as np
>>>
>>> x = [1, 2, 3]
>>> y = simple_linear_function(x)
>>> np.testing.assert_almost_equal(y, 6)
```

CHAPTER FOUR

ACKNOWLEDGEMENT

`temfpy` is developed and maintained as part of the OpenSourceEconomics initiative. We build on the work in [S2013], [B2016], and [W2020b].

Project Manager

- Philipp Eisenhauer (peisenha)

Developers

- Luis Wardenbach (luward)
- Mila Kiseleva (milakis)
- Leiqiong Wan (loikein)
- Manuel Huth (manuhuth)

CHAPTER

FIVE

BIBLIOGRAPHY

BIBLIOGRAPHY

[V2009] Varadhan, R., and Gilbert, P. D. (2009). BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(1):1–26, 2009.

[V2009] Varadhan, R., and Gilbert, P. D. (2009). BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(1):1–26, 2009.

[V2009] Varadhan, R., and Gilbert, P. D. (2009). BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(1):1–26, 2009.

[V2009] Varadhan, R., and Gilbert, P. D. (2009).

[V2009] Varadhan, R., and Gilbert, P. D. (2009). BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(1):1–26, 2009.

[V2009] Varadhan, R., and Gilbert, P. D. (2009). BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(1):1–26, 2009.

[A1987] Ackley, D. H. (1987). A connectionist machine for genetic hillclimbing. Boston, MA: Kluwer Academic Publishers.

[B1996] Back, T. (1996). Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms. Oxford, UK: Oxford University Press.

[M2005] Molga, M., and Smutnicki, C. (2005). Test functions for optimization needs. Retrieved June 2020, from http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.

[R1974] Rastrigin, L. A. (1974). Systems of extremal control. Moscow, Russia: Mir.

[R1960] Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. The Computer Journal, Volume 3, Issue 3, Pages 175-184

[H1983] Harper, W. V., and Gupta, S. K. (1983) Sensitivity/uncertainty analysis of a borehole scenario comparing Latin hypercube sampling and deterministc sensitivity approaches. Office of Nuclear Waste Isolation, Battelle Memorial Institute.

[X2013] Xiong, S., and Qian, P. Z., and Wu, C. J. (2013). Sequential design and analysis of high-accuracy and low-accuracy computer codes. Technometrics, 55(1), 37-46.

[H1990] Harris, F. W. (1990). How many parts to make at once. Operations Research, 38(6), 947–950.

[E1990] Erlenkotter, D. (1990). Ford Whitman Harris and the economic order quantity model. Operations Research, 38(6), 937–946.

[W2020] Economic order quantity. (2020, April 3). In Wikipedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Economic_order_quantity&oldid=948881557

[I1990] Ishigami, T., and Homma, T. (1990). An importance quantification technique in uncertainty analysis for computer models. In: Uncertainty Modeling and Analysis, 1990. Proceedings., First International Symposium on (pp. 398-403).

[S1999] Sobol', I. M., and Levitan, Y. L (1999). On the use of variance reducing multipliers in Monte Carlo computations of a global sensitivity index. Computer Physics Communications, 117(1), 52-61.

[B2016] A. Bűrmen, J. Puhan, J. Olenšek, G. Cijan, and T. Tuma. Pyopus-simulation, optimization, and design. *EDA Laboratory, Faculty of Electrical Engineering, University of Ljubljana. Retrieved May, 2020, from http://spiceopus.si/pyopus/index.html*, 2016.

[R1960] H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[S2013] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. *Retrieved May, 2020, from http://www.sfu.ca/ ssurjano*, 2013.

[V2009] R. Varadhan and P. Gilbert. Bb: an r package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of statistical software*, 32(1):1–26, 2009.

[W2020b] Wikipedia. Test functions for optimization. *Retrieved May, 2020 from https://en.wikipedia.org/w/index.php?title=Test_functions_for_optimization&oldid=937963949*, 2020.

PYTHON MODULE INDEX

t

temfpy.nonlinear_equations, 3
temfpy.optimization, 9
temfpy.uncertainty_quantification, 15

INDEX

A

ackley() (*in module temfpy.optimization*), 9

B

borehole() (*in module temfpy.uncertainty_quantification*), 15
broyden() (*in module temfpy.nonlinear_equations*), 3

C

chandrasekhar() (*in module temfpy.nonlinear_equations*), 3

E

eoq_model() (*in module temfpy.uncertainty_quantification*), 15
exponential() (*in module temfpy.nonlinear_equations*), 4

I

ishigami() (*in module temfpy.uncertainty_quantification*), 17

M

module
    temfpy.nonlinear_equations, 3
    temfpy.optimization, 9
    temfpy.uncertainty_quantification, 15

R

rastrigin() (*in module temfpy.optimization*), 11
rosenbrock() (*in module temfpy.optimization*), 11
rosenbrock_ext() (*in module temfpy.nonlinear_equations*), 5

S

simple_linear_function() (*in module temfpy.uncertainty_quantification*), 17

T

temfpy.nonlinear_equations
    module, 3
temfpy.optimization
    module, 9
temfpy.uncertainty_quantification
    module, 15
trig_exp() (*in module temfpy.nonlinear_equations*), 6
troesch() (*in module temfpy.nonlinear_equations*), 6