

# Package ‘TestFunctions’

January 21, 2024

**Type** Package

**Title** Test Functions for Simulation Experiments and Evaluating  
Optimization and Emulation Algorithms

**Version** 0.2.1

**Author** Collin Erickson

**Maintainer** Collin Erickson <collinberickson@gmail.com>

**Description** Test functions are often used to test computer code.  
They are used in optimization to test algorithms and in  
metamodeling to evaluate model predictions. This package provides  
test functions that can be used for any purpose.

**License** GPL-3

**RoxygenNote** 7.2.3

**Encoding** UTF-8

**Suggests** knitr, rmarkdown, numDeriv, ContourFunctions, covr, ggplot2,  
lhs, testthat

**VignetteBuilder** knitr

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2024-01-21 14:52:59 UTC

## R topics documented:

add_linear_terms . . . . .	3
add_noise . . . . .	3
add_null_dims . . . . .	4
add_zoom . . . . .	5
bananagramacy2Dexp . . . . .	5
funcprofile . . . . .	16
nsin . . . . .	17
numGrad . . . . .	17
numHessian . . . . .	18

RFF	18
RFF_get	19
standard_test_func	20
subtractlm	21
test_func_applyMO	21
TF_ackley	22
TF_bananagramacy2Dexp	23
TF_bananatimesgramacy2Dexp	24
TF_beale	24
TF_beambending	25
TF_branin	25
TF_chengsandu	28
TF_detpep8d	28
TF_easom	29
TF_Gfunction	30
TF_GoldsteinPrice	30
TF_GoldsteinPriceLog	31
TF_gramacy2Dexp	31
TF_gramacy2Dexp3hole	32
TF_gramacy6D	33
TF_griewank	33
TF_hartmann	34
TF_hump	34
TF_levy	35
TF_levytilt	35
TF_linkletter_nosignal	36
TF_logistic	36
TF_logistic15	37
TF_logistic_plateau	38
TF_michalewicz	38
TF_moon_high	39
TF_morris	39
TF_piston	40
TF_quad_peaks	40
TF_quad_peaks_slant	41
TF_rastrigin	41
TF_robotarm	42
TF_RoosArnold	42
TF_steelcolumnstress	43
TF_SWNExpCos	44
TF_vertigrad	44
TF_vertigrad_grad	45
TF_welch	45
TF_wingweight	46
TF_winkel	47

---

add_linear_terms	<i>add_linear_terms: Add linear terms to another function. Allows you to easily change an existing function to include linear terms.</i>
------------------	------------------------------------------------------------------------------------------------------------------------------------------

---

**Description**

add\_linear\_terms: Add linear terms to another function. Allows you to easily change an existing function to include linear terms.

**Usage**

```
add_linear_terms(func, coeffs)
```

**Arguments**

func	Function to add linear terms to
coeffs	Linear coefficients, should have same length as function has dimensions

**Value**

Function with added linear terms

**Examples**

```
banana(c(.1, .2))
add_linear_terms(banana, coeffs=c(10, 1000))(c(.1, .2))
```

---

add_noise	<i>add_noise: Adds noise to any function</i>
-----------	----------------------------------------------

---

**Description**

add\_noise: Adds noise to any function

**Usage**

```
add_noise(func, noise = 0, noise_type = "Gauss")
```

**Arguments**

func	Function to add noise to.
noise	Standard deviation of Gaussian noise
noise_type	Type of noise, only option now is "Gauss" for Gaussian noise.

**Value**

A function that has noise

**Examples**

```
tf <- add_noise(function(x)sin(2*x*pi));curve(tf)
tf <- add_noise(function(x)sin(2*x*pi), noise=.1);curve(tf)
```

---

add_null_dims	<i>add_null_dims: Add null dimensions to another function. Allows you to pass in input data with any number of dimensions and it will only keep the first nactive.</i>
---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

**Description**

add\_null\_dims: Add null dimensions to another function. Allows you to pass in input data with any number of dimensions and it will only keep the first nactive.

**Usage**

```
add_null_dims(func, nactive)
```

**Arguments**

func	Function to add null dimensions to
nactive	Number of active dimensions in func

**Value**

Function that can take any dimensional input

**Examples**

```
banana(c(.1, .2))
# banana(c(.1,.2,.4,.5,.6,.7,.8)) # gives warning
add_null_dims(banana, nact=2)(c(.1,.2,.4,.5,.6,.7,.8))
```

---

add_zoom	<i>add_zoom: Zoom in on region of another function. Allows you to easily change an existing function so that <math>[0,1]^n</math> refers to a subregion of the original function</i>
----------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

### Description

add\_zoom: Zoom in on region of another function. Allows you to easily change an existing function so that  $[0,1]^n$  refers to a subregion of the original function

### Usage

```
add_zoom(func, scale_low, scale_high)
```

### Arguments

func	Function to add linear terms to
scale_low	Vector of low end of scale values for each dimension
scale_high	Vector of high end of scale values for each dimension

### Value

Function with added linear terms

### Examples

```
banana(c(.5, .85))
add_zoom(banana, c(0, .5), c(1,1))(c(.5, .7))
add_zoom(banana, c(.2, .5), c(.8,1))(matrix(c(.5, .7), ncol=2))
ContourFunctions::cf(banana)
ContourFunctions::cf(add_zoom(banana, c(0, .5), c(1,1)))
ContourFunctions::cf(add_zoom(banana, c(.2, .5), c(.8,1)))
```

---

bananagramacy2Dexp	<i>bananagramacy2Dexp: bananagramacy2Dexp function 6 dimensional function. First two dimensions are banana function, next two are the gramacy2Dexp function, last two are null dimensions</i>
--------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

### Description

branin: A function. 2 dimensional function.

**Usage**

```
bananagramacy2Dexp(  
  x,  
  scale_it = T,  
  scale_low = 0,  
  scale_high = 1,  
  noise = 0,  
  ...  
)
```

```
bananatimesgramacy2Dexp(  
  x,  
  scale_it = T,  
  scale_low = 0,  
  scale_high = 1,  
  noise = 0,  
  ...  
)
```

```
gramacy2Dexp(x, scale_it = T, scale_low = -2, scale_high = 6, noise = 0, ...)
```

```
gramacy2Dexp3hole(  
  x,  
  scale_it = T,  
  scale_low = 0,  
  scale_high = 1,  
  noise = 0,  
  ...  
)
```

```
gramacy6D(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
```

```
branin(  
  x,  
  scale_it = T,  
  scale_low = c(-5, 0),  
  scale_high = c(10, 15),  
  noise = 0  
)
```

```
borehole(  
  x,  
  scale_it = T,  
  scale_low = c(0.05, 100, 63070, 990, 63.1, 700, 1120, 9855),  
  scale_high = c(0.15, 50000, 115600, 1110, 116, 820, 1680, 12045),  
  noise = 0  
)
```

```
franke(x, scale_it = F, scale_low = c(0, 0), scale_high = c(1, 1), noise = 0)

zhou1998(x, scale_it = F, scale_low = c(0, 0), scale_high = c(1, 1), noise = 0)

currin1991(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0
)

currin1991b(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0
)

limpoly(x, scale_it = F, scale_low = c(0, 0), scale_high = c(1, 1), noise = 0)

limnonpoly(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0
)

banana(
  x,
  scale_it = T,
  scale_low = c(-20, -10),
  scale_high = c(20, 5),
  noise = 0
)

banana_grad(
  x,
  scale_it = T,
  scale_low = c(-20, -10),
  scale_high = c(20, 5),
  noise = 0
)

gaussian1(
  x,
```

```
    scale_it = F,
    scale_low = c(0, 0),
    scale_high = c(1, 1),
    noise = 0
)

sinumoid(x, scale_it = F, scale_low = c(0, 0), scale_high = c(1, 1), noise = 0)

waterfall(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0
)

sqrtsin(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0,
  freq = 2 * pi
)

powsin(
  x,
  scale_it = F,
  scale_low = c(0, 0),
  scale_high = c(1, 1),
  noise = 0,
  freq = 2 * pi,
  pow = 0.7
)

OTL_Circuit(
  x,
  scale_it = T,
  scale_low = c(50, 25, 0.5, 1.2, 0.25, 50),
  scale_high = c(150, 70, 3, 2.5, 1.2, 300),
  noise = 0
)

GoldsteinPrice(
  x,
  scale_it = T,
  scale_low = c(-2, -2),
  scale_high = c(2, 2),
```

```
    noise = 0
  )

GoldsteinPriceLog(
  x,
  scale_it = T,
  scale_low = c(-2, -2),
  scale_high = c(2, 2),
  noise = 0
)

ackley(
  x,
  scale_it = T,
  scale_low = -32.768,
  scale_high = 32.768,
  noise = 0,
  a = 20,
  b = 0.2,
  c = 2 * pi
)

piston(
  x,
  scale_it = T,
  scale_low = c(30, 0.005, 0.002, 1000, 90000, 290, 340),
  scale_high = c(60, 0.02, 0.01, 5000, 110000, 296, 360),
  noise = 0
)

wingweight(
  x,
  scale_it = T,
  scale_low = c(150, 220, 6, -10, 16, 0.5, 0.08, 2.5, 1700, 0.025),
  scale_high = c(200, 300, 10, 10, 45, 1, 0.18, 6, 2500, 0.08),
  noise = 0
)

welch(x, scale_it = T, scale_low = c(-0.5), scale_high = c(0.5), noise = 0)

robotarm(
  x,
  scale_it = T,
  scale_low = rep(0, 8),
  scale_high = c(rep(2 * pi, 4), rep(1, 4)),
  noise = 0
)
```

```
RoosArnold(x, scale_it = F, scale_low = 0, scale_high = 1, noise = 0)
Gfunction(x, scale_it = F, scale_low = 0, scale_high = 1, noise = 0, ...)
beale(x, scale_it = T, scale_low = -4.5, scale_high = 4.5, noise = 0, ...)
easom(x, scale_it = T, scale_low = -4.5, scale_high = 4.5, noise = 0, ...)
griewank(x, scale_it = T, scale_low = -600, scale_high = 600, noise = 0, ...)
hump(x, scale_it = T, scale_low = -5, scale_high = 5, noise = 0, ...)
levy(x, scale_it = T, scale_low = -10, scale_high = 10, noise = 0, ...)
levytilt(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
michalewicz(x, scale_it = T, scale_low = 0, scale_high = pi, noise = 0, ...)
rastrigin(
  x,
  scale_it = T,
  scale_low = -5.12,
  scale_high = 5.12,
  noise = 0,
  ...
)
moon_high(x, scale_it = F, scale_low = 0, scale_high = 1, noise = 0, ...)
linkletter_nosignal(
  x,
  scale_it = F,
  scale_low = 0,
  scale_high = 1,
  noise = 0,
  ...
)
morris(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
detpep8d(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
hartmann(x, scale_it = F, scale_low = 0, scale_high = 1, noise = 0, ...)
quad_peaks(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
quad_peaks_slant(
  x,
```

```

    scale_it = T,
    scale_low = 0,
    scale_high = 1,
    noise = 0,
    ...
)
SWNExpCos(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
logistic(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
logistic15(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
logistic_plateau(
  x,
  scale_it = T,
  scale_low = 0,
  scale_high = 1,
  noise = 0,
  ...
)
vertigrad(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
vertigrad_grad(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
beambending(
  x,
  scale_it = T,
  scale_low = c(10, 1, 0.1),
  scale_high = c(20, 2, 0.2),
  noise = 0,
  ...
)
chengsandu(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)
steelcolumnstress(
  x,
  scale_it = T,
  scale_low = c(330, 4e+05, 420000, 420000, 200, 10, 100, 10, 12600),
  scale_high = c(470, 6e+05, 780000, 780000, 400, 30, 500, 50, 29400),
  noise = 0,
  ...
)
winkel(x, scale_it = T, scale_low = 0, scale_high = 1, noise = 0, ...)

```

```

boreholeMV(
  x,
  NOD = 51,
  scale_it = T,
  scale_low = c(0.05, 100, 63070, 990, 63.1, 700, 1120, 9855),
  scale_high = c(0.15, 50000, 115600, 1110, 116, 820, 1680, 12045),
  noise = 0
)

test_func_apply(func, x, scale_it, scale_low, scale_high, noise = 0, ...)

```

### Arguments

x	Input value, either a matrix whose rows are points or a vector for a single point. Be careful with 1-D functions.
scale_it	Should the data be scaled from $[0, 1]^D$ to $[\text{scale\_low}, \text{scale\_high}]$ ? This means the input data is confined to be in $[0, 1]^D$ , but the function isn't.
scale_low	Lower bound for each variable
scale_high	Upper bound for each variable
noise	If white noise should be added, specify the standard deviation for normal noise
...	Additional parameters for func
freq	Wave frequency for sqrtsin and powsin
pow	Power for powsin
a	A constant for ackley()
b	A constant for ackley()
c	A constant for ackley()
NOD	number of output dimensions
func	A function to evaluate

### Value

Function values at x

### References

- Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.
- Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.
- Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.
- Dixon, L. C. W. (1978). The global optimization problem: an introduction. *Towards Global Optimization* 2, 1-15.
- Morris, M. D., Mitchell, T. J., & Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3), 243-255.

- Worley, Brian A. Deterministic uncertainty analysis. No. ORNL-6428. Oak Ridge National Lab., TN (USA), 1987.
- Franke, R. (1979). A critical comparison of some methods for interpolation of scattered data. Monterey, California: Naval Postgraduate School. Page 13.
- An, J., & Owen, A. (2001). Quasi-regression. *Journal of complexity*, 17(4), 588-607.
- Currin, C., Mitchell, T., Morris, M., & Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416), 953-963.
- Currin, C., Mitchell, T., Morris, M., & Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416), 953-963.
- Lim, Yong B., Jerome Sacks, W. J. Studden, and William J. Welch. "Design and analysis of computer experiments when the output is highly correlated over the input space." *Canadian Journal of Statistics* 30, no. 1 (2002): 109-126.
- Lim, Yong B., Jerome Sacks, W. J. Studden, and William J. Welch. "Design and analysis of computer experiments when the output is highly correlated over the input space." *Canadian Journal of Statistics* 30, no. 1 (2002): 109-126.
- Haario, H., Saksman, E., & Tamminen, J. (1999). Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*, 14(3), 375-396.
- Joseph, V. R., Dasgupta, T., Tuo, R., & Wu, C. J. (2015). Sequential exploration of complex surfaces using minimum energy designs. *Technometrics*, 57(1), 64-74.
- Ben-Ari, Einat Neumann, and David M. Steinberg. "Modeling data from computer experiments: an empirical comparison of kriging with MARS and projection pursuit regression." *Quality Engineering* 19.4 (2007): 327-338.
- Kenett, Ron S., Shelemyahu Zacks, and Daniele Amberti. *Modern Industrial Statistics: with applications in R, MINITAB and JMP*. John Wiley & Sons, 2013.
- Forrester, A., & Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.
- [http://www.abe.ufl.edu/jjones/ABE\\_5646/2010/Morris.1991](http://www.abe.ufl.edu/jjones/ABE_5646/2010/Morris.1991)
- <http://www.tandfonline.com/doi/pdf/10.1198/TECH.2010.09157?needAccess=true>
- Santner, T. J., Williams, B. J., & Notz, W. (2003). *The Design and Analysis of Computer Experiments*. Springer Science & Business Media.
- Cheng, Haiyan, and Adrian Sandu. "Collocation least-squares polynomial chaos method." In *Proceedings of the 2010 Spring Simulation Multiconference*, p. 80. Society for Computer Simulation International, 2010.
- Kuschel, Norbert, and Rudiger Rackwitz. "Two basic problems in reliability-based structural optimization." *Mathematical Methods of Operations Research* 46, no. 3 (1997): 309-333.
- Prikhodko, Pavel, and Nikita Kotlyarov. "Calibration of Sobol indices estimates in case of noisy output." arXiv preprint arXiv:1804.00766 (2018).
- Winkel, Munir A., Jonathan W. Stallings, Curt B. Storlie, and Brian J. Reich. "Sequential Optimization in Locally Important Dimensions." arXiv preprint arXiv:1804.10671 (2018).
- Morris, M. D., Mitchell, T. J., & Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3), 243-255.

Worley, Brian A. Deterministic uncertainty analysis. No. ORNL-6428. Oak Ridge National Lab., TN (USA), 1987.

### Examples

```

bananagramacy2Dexp(runif(6))
bananagramacy2Dexp(matrix(runif(6*20),ncol=6))
bananatimesgramacy2Dexp(runif(6))
bananatimesgramacy2Dexp(matrix(runif(6*20),ncol=6))
gramacy2Dexp(runif(2))
gramacy2Dexp(matrix(runif(2*20),ncol=2))
gramacy2Dexp3hole(runif(2))
gramacy2Dexp3hole(matrix(runif(2*20),ncol=2))
gramacy6D(runif(6))
gramacy6D(matrix(runif(6*20),ncol=6))
branin(runif(2))
branin(matrix(runif(20), ncol=2))
borehole(runif(8))
borehole(matrix(runif(80), ncol=8))
franke(runif(2))
zhou1998(runif(2))
currin1991(runif(2))
currin1991b(runif(2))
limpoly(runif(2))
limnonpoly(runif(2))
banana(runif(2))
x <- y <- seq(0, 1, len=100)
z <- outer(x, y, Vectorize(function(a, b){banana(c(a, b))}))
contour(x, y, z)
banana_grad(runif(2))
x <- y <- seq(0, 1, len=100)
z <- outer(x, y, Vectorize(function(a, b){sum(banana_grad(c(a, b))^2}))
contour(x, y, z)
gaussian1(runif(2))
sinumoid(runif(2))
x <- y <- seq(0, 1, len=100)
z <- outer(x, y, Vectorize(function(a, b){sinumoid(c(a, b))}))
contour(x, y, z)
waterfall(runif(2))
sqrtsin(runif(1))
curve(sqrtsin(matrix(x,ncol=1)))
powsin(runif(1))#,pow=2)
OTL_Circuit(runif(6))
OTL_Circuit(matrix(runif(60),ncol=6))
GoldsteinPrice(runif(2))
GoldsteinPrice(matrix(runif(60),ncol=2))
GoldsteinPriceLog(runif(2))
GoldsteinPriceLog(matrix(runif(60),ncol=2))
ackley(runif(2))
ackley(matrix(runif(60),ncol=2))
piston(runif(7))
piston(matrix(runif(7*20),ncol=7))
wingweight(runif(10))

```

```
wingweight(matrix(runif(10*20),ncol=10))
welch(runif(20))
welch(matrix(runif(20*20),ncol=20))
robotarm(runif(8))
robotarm(matrix(runif(8*20),ncol=8))
RoosArnold(runif(8))
RoosArnold(matrix(runif(8*20),ncol=8))
Gfunction(runif(8))
Gfunction(matrix(runif(8*20),ncol=8))
beale(runif(2))
beale(matrix(runif(2*20),ncol=2))
easom(runif(2))
easom(matrix(runif(2*20),ncol=2))
griewank(runif(2))
griewank(matrix(runif(2*20),ncol=2))
hump(runif(2))
hump(matrix(runif(2*20),ncol=2))
levy(runif(2))
levy(matrix(runif(2*20),ncol=2))
levytilt(runif(2))
levytilt(matrix(runif(2*20),ncol=2))
michalewicz(runif(2))
michalewicz(matrix(runif(2*20),ncol=2))
rastrigin(runif(2))
rastrigin(matrix(runif(2*20),ncol=2))
moon_high(runif(20))
moon_high(matrix(runif(20*20),ncol=20))
linkletter_nosignal(runif(2))
linkletter_nosignal(matrix(runif(2*20),ncol=2))
morris(runif(20))
morris(matrix(runif(20*20),ncol=20))
detpep8d(runif(2))
detpep8d(matrix(runif(2*20),ncol=2))
hartmann(runif(2))
hartmann(matrix(runif(6*20),ncol=6))
quad_peaks(runif(2))
quad_peaks(matrix(runif(2*20),ncol=2))
quad_peaks_slant(runif(2))
quad_peaks_slant(matrix(runif(2*20),ncol=2))
SWNExpCos(runif(2))
SWNExpCos(matrix(runif(2*20),ncol=2))
curve(logistic, from=-5,to=5)
curve(logistic(x,offset=.5, scl=15))
logistic(matrix(runif(20),ncol=1))
curve(logistic15)
curve(logistic15(x,offset=.25))
logistic15(matrix(runif(20),ncol=1))
curve(logistic_plateau(matrix(x,ncol=1)))
logistic_plateau(matrix(runif(20),ncol=1))
vertigrad(runif(2))
vertigrad(matrix(runif(2*20),ncol=2))
vertigrad_grad(runif(2))
vertigrad_grad(matrix(runif(2*20),ncol=2))
```

```

beambending(runif(3))
beambending(matrix(runif(3*20),ncol=3))
chengsandu(runif(2))
chengsandu(matrix(runif(2*20),ncol=2))
steelcolumnstress(runif(8))
steelcolumnstress(matrix(runif(8*20),ncol=8))
winkel(runif(2))
winkel(matrix(runif(2*20),ncol=2))
boreholeMV(runif(8))
boreholeMV(matrix(runif(80), ncol=8))
x <- matrix(seq(0,1,length.out=10), ncol=1)
y <- test_func_apply(sin, x, TRUE, 0, 2*pi, .05)
plot(x,y)
curve(sin(2*pi*x), col=2, add=TRUE)

```

---

funcprofile

*Profile a function*


---

### Description

Gives details about how linear it is.

### Usage

```
funcprofile(func, d, n = 1000 * d, bins = 30)
```

### Arguments

func	A function with a single output
d	The number of input dimensions for the function
n	The number of points to use for the linear model.
bins	Number of bins in histogram.

### Value

Nothing, prints and plots

### Examples

```
funcprofile(ackley, 2)
```

---

nsin	<i>Wave functions</i>
------	-----------------------

---

**Description**

nsin: Block wave

**Usage**

```
nsin(xx)
```

```
vsin(xx)
```

**Arguments**

xx	Input values
----	--------------

**Value**

nsin evaluated at nsin

**Examples**

```
curve(nsin(2*pi*x), n = 1000)
curve(nsin(12*pi*x), n = 1000)
curve(vsin(2*pi*x), n = 1000)
curve(vsin(12*pi*x), n = 1000)
```

---

numGrad	<i>Create function calculating the numerical gradient</i>
---------	-----------------------------------------------------------

---

**Description**

Create function calculating the numerical gradient

**Usage**

```
numGrad(func, ...)
```

**Arguments**

func	Function to get gradient of.
...	Arguments passed to numDeriv::grad().

**Value**

A gradient function

**Examples**

```
numGrad(sin)
```

---

```
numHessian
```

*Create function calculating the numerical hessian*

---

**Description**

Create function calculating the numerical hessian

**Usage**

```
numHessian(func, ...)
```

**Arguments**

func	Function to get hessian of
...	Arguments passed to numDeriv::hessian().

**Value**

A hessian function

**Examples**

```
numHessian(sin)
```

---

```
RFF
```

*Evaluate an RFF (random wave function) at given input*

---

**Description**

Evaluate an RFF (random wave function) at given input

**Usage**

```
RFF(x, freq, mag, dirr, offset, wave = sin, noise = 0)
```

**Arguments**

x	Matrix whose rows are points to evaluate or a vector representing a single point. In 1 dimension you must use a matrix for multiple points, not a vector.
freq	Vector of wave frequencies
mag	Vector of wave magnitudes
dirr	Matrix of wave directions
offset	Vector of wave offsets
wave	Type of wave
noise	Standard deviation of random normal noise to add

**Value**

Output of RFF evaluated at x

**Examples**

```
curve(RFF(matrix(x,ncol=1),3,1,1,0))
curve(RFF(matrix(x,ncol=1),3,1,1,0, noise=.1), n=1e3, type='p', pch=19)

curve(RFF(matrix(x,ncol=1),c(3,20),c(1,.1),c(1,1),c(0,0)), n=1e3)
```

---

RFF\_get

*Create a new RFF function*


---

**Description**

Create a new RFF function

**Usage**

```
RFF_get(D = 2, M = 30, wave = sin, noise = 0, seed = NULL)
```

**Arguments**

D	Number of dimensions
M	Number of random waves
wave	Type of wave
noise	Standard deviation of random normal noise to add
seed	Seed to set before randomly selecting function

**Value**

A random wave function

**Examples**

```
func <- RFF_get(D=1)
curve(func)

f <- RFF_get(D=1, noise=.1)
curve(f(matrix(x,ncol=1)))
for(i in 1:100) curve(f(matrix(x,ncol=1)), add=TRUE, col=sample(2:8,1))
```

---

standard\_test\_func      *Create a standard test function.*

---

## Description

This makes it easier to create many functions that follow the same template. R CMD check doesn't like the ... if this command is used to create functions in the package, so it is not currently used.

## Usage

```
standard_test_func(
  func,
  scale_it_ = F,
  scale_low_ = NULL,
  scale_high_ = NULL,
  noise_ = 0,
  ...
)
```

## Arguments

func	A function that takes a vector representing a single point.
scale_it_	Should the function scale the inputs from $[0, 1]^D$ to $[\text{scale\_low\_}, \text{scale\_high\_}]$ by default? This can be overridden when actually giving the output function points to evaluate.
scale_low_	What is the default lower bound of the data?
scale_high_	What is the default upper bound of the data?
noise_	Should noise be added to the function by default?
...	Parameters passed to func when evaluating points.

## Value

A test function created using the standard\_test\_func template.

## Examples

```
.gaussian1 <- function(x, center=.5, s2=.01) {
  exp(-sum((x-center)^2/2/s2))
}
gaussian1 <- standard_test_func(.gaussian1, scale_it=FALSE, scale_low = c(0,0), scale_high = c(1,1))
curve(gaussian1(matrix(x,ncol=1)))
```

---

subtractlm	<i>Subtract linear model from a function</i>
------------	----------------------------------------------

---

**Description**

This returns a new function which a linear model has an r-squared of 0.

**Usage**

```
subtractlm(func, d, n = d * 100)
```

**Arguments**

func	A function
d	Number of input dimensions
n	Number of points to use for the linear model

**Value**

A new function

**Examples**

```
subtractlm(ackley, 2)

f <- function(x) {
  if (is.matrix(x)) x[,1]^2
  else x[1]^2
}
ContourFunctions::cf(f)
ContourFunctions::cf(subtractlm(f, 2), batchmax=Inf)
```

---

test_func_applyMO	<i>General function for evaluating a test function with multivariate output</i>
-------------------	---------------------------------------------------------------------------------

---

**Description**

General function for evaluating a test function with multivariate output

**Usage**

```
test_func_applyM0(
  func,
  x,
  numoutdim,
  scale_it,
  scale_low,
  scale_high,
  noise = 0,
  ...
)
```

**Arguments**

func	A function to evaluate
x	Input value, either a matrix whose rows are points or a vector for a single point. Be careful with 1-D functions.
numoutdim	Number of output dimensions
scale_it	Should the data be scaled from $[0, 1]^D$ to $[\text{scale\_low}, \text{scale\_high}]$ ? This means the input data is confined to be in $[0, 1]^D$ , but the function isn't.
scale_low	Lower bound for each variable
scale_high	Upper bound for each variable
noise	If white noise should be added, specify the standard deviation for normal noise
...	Additional parameters for func

**Value**

Function values at x

**Examples**

```
x <- matrix(seq(0,1,length.out=10), ncol=1)
y <- test_func_apply(sin, x, TRUE, 0, 2*pi, .05)
plot(x,y)
curve(sin(2*pi*x), col=2, add=TRUE)
```

---

TF\_ackley

*TF\_ackley: Ackley function for evaluating a single point.*

---

**Description**

TF\_ackley: Ackley function for evaluating a single point.

**Usage**

```
TF_ackley(x, a = 20, b = 0.2, c = 2 * pi)
```

**Arguments**

x	Input vector at which to evaluate.
a	A constant for ackley()
b	A constant for ackley()
c	A constant for ackley()

**Value**

Function output evaluated at x.

**Examples**

```
TF_ackley(c(0, 0)) # minimum of zero, hard to solve
```

---

TF\_bananagramacy2Dexp *TF\_bananagramacy2Dexp: bananagramacy2Dexp function for evaluating a single point.*

---

**Description**

TF\_bananagramacy2Dexp: bananagramacy2Dexp function for evaluating a single point.

**Usage**

```
TF_bananagramacy2Dexp(x)
```

**Arguments**

x	Input vector at which to evaluate.
---	------------------------------------

**Value**

Function output evaluated at x.

**Examples**

```
TF_bananagramacy2Dexp(rep(0, 6))  
TF_bananagramacy2Dexp(rep(1, 6))
```

---

TF\_bananatimesgramacy2Dexp

*TF\_bananatimesgramacy2Dexp: bananatimesgramacy2Dexp function for evaluating a single point.*

---

**Description**

TF\_bananatimesgramacy2Dexp: bananatimesgramacy2Dexp function for evaluating a single point.

**Usage**

TF\_bananatimesgramacy2Dexp(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_bananatimesgramacy2Dexp(rep(0, 6))  
TF_bananatimesgramacy2Dexp(rep(1, 6))
```

---

TF\_beale

*TF\_beale: Beale function for evaluating a single point.*

---

**Description**

TF\_beale: Beale function for evaluating a single point.

**Usage**

TF\_beale(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_beale(rep(0, 2))  
TF_beale(rep(1, 2))
```

---

TF_beambending	<i>TF_beambending: beambending function for evaluating a single point.</i>
----------------	----------------------------------------------------------------------------

---

**Description**

TF\_beambending: beambending function for evaluating a single point.

**Usage**

```
TF_beambending(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_beambending(rep(0,3))  
TF_beambending(rep(1,3))
```

---

TF_branin	<i>Base test function.</i>
-----------	----------------------------

---

**Description**

TF\_branin: A function taking in a single vector. 2 dimensional function. See corresponding function with "TF\_" for more details.

**Usage**

```
TF_branin(  
  x,  
  a = 1,  
  b = 5.1/(4 * pi^2),  
  cc = 5/pi,  
  r = 6,  
  s = 10,  
  tt = 1/(8 * pi)  
)  
  
TF_borehole(x)
```

TF\_franke(x)  
TF\_zhou1998(x)  
TF\_currin1991(x)  
TF\_currin1991b(x)  
TF\_limpoly(x)  
TF\_limnonpoly(x)  
TF\_banana(x)  
TF\_banana\_grad(x, v1, v2)  
TF\_gaussian1(x, center = 0.5, s2 = 0.01)  
TF\_sinumoid(x)  
TF\_sqrtsin(x, freq = 2 \* pi)  
TF\_powsin(x, freq = 2 \* pi, pow = 0.7)  
TF\_OTL\_Circuit(x)  
TF\_boreholeMV(x, NOD = 51)

### Arguments

x	Input vector at which to evaluate.
a	Parameter for TF_branin
b	Parameter for TF_branin
cc	Parameter for TF_branin
r	Parameter for TF_branin
s	Parameter for TF_branin
tt	Parameter for TF_branin
v1	Scale parameter for first dimension
v2	Scale parameter for second dimension
center	Where to center the function, a vector.
s2	Variance of the Gaussian.
freq	Wave frequency for TF_sqrtsin and TF_powsin
pow	Power to raise wave to for TF_powsin.
NOD	number of output dimensions.

**Value**

Function output evaluated at x.

**References**

- Dixon, L. C. W. (1978). The global optimization problem: an introduction. *Towards Global Optimization* 2, 1-15.
- Morris, M. D., Mitchell, T. J., & Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3), 243-255.
- Worley, Brian A. Deterministic uncertainty analysis. No. ORNL-6428. Oak Ridge National Lab., TN (USA), 1987.
- Franke, R. (1979). A critical comparison of some methods for interpolation of scattered data. Monterey, California: Naval Postgraduate School. Page 13.
- An, J., & Owen, A. (2001). Quasi-regression. *Journal of complexity*, 17(4), 588-607.
- Currin, C., Mitchell, T., Morris, M., & Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416), 953-963.
- Currin, C., Mitchell, T., Morris, M., & Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416), 953-963.
- Haario, H., Saksman, E., & Tamminen, J. (1999). Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*, 14(3), 375-396.
- Joseph, V. R., Dasgupta, T., Tuo, R., & Wu, C. J. (2015). Sequential exploration of complex surfaces using minimum energy designs. *Technometrics*, 57(1), 64-74.
- Ben-Ari, Einat Neumann, and David M. Steinberg. "Modeling data from computer experiments: an empirical comparison of kriging with MARS and projection pursuit regression." *Quality Engineering* 19.4 (2007): 327-338.
- Morris, M. D., Mitchell, T. J., & Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3), 243-255.
- Worley, Brian A. Deterministic uncertainty analysis. No. ORNL-6428. Oak Ridge National Lab., TN (USA), 1987.

**Examples**

```
TF_branin(runif(2))
TF_borehole(runif(8))
TF_franke(runif(2))
TF_zhou1998(runif(2))
TF_currin1991(runif(2))
TF_currin1991b(runif(2))
TF_limpoly(runif(2))
TF_limnonpoly(runif(2))
TF_banana(runif(2))
TF_banana_grad(runif(2), v1=40, v2=15)
TF_gaussian1(runif(2))
TF_sinumoid(runif(2))
```

```
TF_sqrtsin(runif(2))
TF_powsin(runif(2))
TF_OTL_Circuit(c(50,25,0.5,1.2,0.25,50))
TF_boreholeMV(runif(8))
```

---

TF\_chengsandu                    *TF\_chengsandu: chengsandu function for evaluating a single point.*

---

### Description

TF\_chengsandu: chengsandu function for evaluating a single point.

### Usage

```
TF_chengsandu(x)
```

### Arguments

x                    Input vector at which to evaluate.

### Value

Function output evaluated at x.

### References

Cheng, Haiyan, and Adrian Sandu. "Collocation least-squares polynomial chaos method." In Proceedings of the 2010 Spring Simulation Multiconference, p. 80. Society for Computer Simulation International, 2010.

### Examples

```
TF_chengsandu(rep(0,2))
TF_chengsandu(rep(1,2))
```

---

TF\_detpep8d                    *TF\_detpep8d: detpep8d function for evaluating a single point.*

---

### Description

TF\_detpep8d: detpep8d function for evaluating a single point.

### Usage

```
TF_detpep8d(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_detpep8d(rep(0,2))  
TF_detpep8d(rep(1,2))
```

---

TF\_easom

*TF\_easom: Easom function for evaluating a single point.*

---

**Description**

TF\_easom: Easom function for evaluating a single point.

**Usage**

```
TF_easom(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_easom(rep(0,2))  
TF_easom(rep(1,2))
```

---

TF_Gfunction	<i>TF_Gfunction: G-function for evaluating a single point.</i>
--------------	----------------------------------------------------------------

---

**Description**

TF\_Gfunction: G-function for evaluating a single point.

**Usage**

```
TF_Gfunction(x, a = (1:length(x) - 1)/2)
```

**Arguments**

x	Input vector at which to evaluate.
a	Parameter for Gfunction

**Value**

Function output evaluated at x.

**Examples**

```
TF_Gfunction(rep(0,8))  
TF_Gfunction(rep(1,8))
```

---

TF_GoldsteinPrice	<i>TF_GoldsteinPrice: Goldstein Price function for evaluating a single point</i>
-------------------	----------------------------------------------------------------------------------

---

**Description**

TF\_GoldsteinPrice: Goldstein Price function for evaluating a single point

**Usage**

```
TF_GoldsteinPrice(x)
```

**Arguments**

x	Input vector at which to evaluate.
---	------------------------------------

**Value**

Function output evaluated at x.

**Examples**

```
TF_GoldsteinPrice(c(0, -1)) # minimum
```

---

TF\_GoldsteinPriceLog    *TF\_GoldsteinPrice: Goldstein Price function for evaluating a single point on a log scale, normalized to have mean 0 and variance 1.*

---

**Description**

TF\_GoldsteinPrice: Goldstein Price function for evaluating a single point on a log scale, normalized to have mean 0 and variance 1.

**Usage**

```
TF_GoldsteinPriceLog(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_GoldsteinPriceLog(c(0, -1)) # minimum
```

---

TF\_gramacy2Dexp        *TF\_gramacy2Dexp: gramacy2Dexp function for evaluating a single point.*

---

**Description**

TF\_gramacy2Dexp: gramacy2Dexp function for evaluating a single point.

**Usage**

```
TF_gramacy2Dexp(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.

**Examples**

```
TF_gramacy2Dexp(rep(0,2))
TF_gramacy2Dexp(rep(1,2))
```

---

TF\_gramacy2Dexp3hole    *TF\_gramacy2Dexp3hole: gramacy2Dexp3hole function for evaluating a single point.*

---

**Description**

TF\_gramacy2Dexp3hole: gramacy2Dexp3hole function for evaluating a single point.

**Usage**

```
TF_gramacy2Dexp3hole(x)
```

**Arguments**

x                      Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.

**Examples**

```
TF_gramacy2Dexp3hole(rep(0,2))
TF_gramacy2Dexp3hole(rep(1,2))
```

---

TF_gramacy6D	<i>TF_gramacy6D: gramacy6D function for evaluating a single point.</i>
--------------	------------------------------------------------------------------------

---

**Description**

From Gramacy and Lee (2009).

**Usage**

TF\_gramacy6D(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Gramacy, Robert B., and Herbert KH Lee. "Adaptive design and analysis of supercomputer experiments." *Technometrics* 51.2 (2009): 130-145.

**Examples**

```
TF_gramacy6D(rep(0,6))
TF_gramacy6D(rep(1,6))
```

---

TF_griewank	<i>TF_griewank: Griewank function for evaluating a single point.</i>
-------------	----------------------------------------------------------------------

---

**Description**

TF\_griewank: Griewank function for evaluating a single point.

**Usage**

TF\_griewank(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_griewank(rep(0,2))
TF_griewank(rep(1,2))
```

---

TF_hartmann	<i>TF_hartmann: hartmann function for evaluating a single point.</i>
-------------	----------------------------------------------------------------------

---

**Description**

TF\_hartmann: hartmann function for evaluating a single point.

**Usage**

```
TF_hartmann(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_hartmann(rep(0,6))
TF_hartmann(rep(1,6))
TF_hartmann(c(.20169, .150011, .476874, .275332, .311652, .6573)) # Global minimum of -3.322368
```

---

TF_hump	<i>TF_hump: Hump function for evaluating a single point.</i>
---------	--------------------------------------------------------------

---

**Description**

TF\_hump: Hump function for evaluating a single point.

**Usage**

```
TF_hump(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_hump(rep(0,2))
TF_hump(rep(1,2))
```

---

`TF_levy`*TF\_levy: Levy function for evaluating a single point.*

---

**Description**

TF\_levy: Levy function for evaluating a single point.

**Usage**

```
TF_levy(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_levy(rep(0,2))
TF_levy(rep(1,2))
```

---

`TF_levytilt`*TF\_levytilt: Levy function with a tilt for evaluating a single point.*

---

**Description**

TF\_levytilt: Levy function with a tilt for evaluating a single point.

**Usage**

```
TF_levytilt(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_levytilt(rep(0,2))
TF_levytilt(rep(1,2))
```

---

```
TF_linkletter_nosignal
```

*TF\_linkletter\_nosignal: Linkletter (2006) no signal function for evaluating a single point.*

---

**Description**

TF\_linkletter\_nosignal: Linkletter (2006) no signal function for evaluating a single point.

**Usage**

```
TF_linkletter_nosignal(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_linkletter_nosignal(rep(0,2))
TF_linkletter_nosignal(rep(1,2))
```

---

```
TF_logistic
```

*TF\_logistic: logistic function for evaluating a single point.*

---

**Description**

TF\_logistic: logistic function for evaluating a single point.

**Usage**

```
TF_logistic(x, offset = 0, scl = 1)
```

**Arguments**

x                    Input vector at which to evaluate.  
 offset              Amount it should be offset  
 scl                  Scale parameter

**Value**

Function output evaluated at x.

**Examples**

```
TF_logistic(0)
TF_logistic(1)
```

---

TF_logistic15	<i>TF_logistic15: logistic15 function for evaluating a single point. Same as logistic except adjusted to be reasonable from 0 to 1.</i>
---------------	-----------------------------------------------------------------------------------------------------------------------------------------

---

**Description**

TF\_logistic15: logistic15 function for evaluating a single point. Same as logistic except adjusted to be reasonable from 0 to 1.

**Usage**

```
TF_logistic15(x, offset = 0.5, scl = 15)
```

**Arguments**

x	Input vector at which to evaluate.
offset	Amount it should be offset
scl	Scale parameter

**Value**

Function output evaluated at x.

**Examples**

```
TF_logistic15(0)
TF_logistic15(1)
curve(Vectorize(TF_logistic15)(x))
```

---

TF_logistic_plateau	<i>TF_logistic_plateau: logistic_plateau function for evaluating a single point.</i>
---------------------	--------------------------------------------------------------------------------------

---

**Description**

TF\_logistic\_plateau: logistic\_plateau function for evaluating a single point.

**Usage**

```
TF_logistic_plateau(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_logistic_plateau(0)
TF_logistic_plateau(.5)
```

---

TF_michalewicz	<i>TF_michalewicz: Michalewicz function for evaluating a single point.</i>
----------------	----------------------------------------------------------------------------

---

**Description**

TF\_michalewicz: Michalewicz function for evaluating a single point.

**Usage**

```
TF_michalewicz(x, m = 10)
```

**Arguments**

x                    Input vector at which to evaluate.  
m                    Parameter for the michalewicz function

**Value**

Function output evaluated at x.

**Examples**

```
TF_michalewicz(rep(0,2))  
TF_michalewicz(rep(1,2))
```

---

TF_moon_high	<i>TF_moon_high: Moon (2010) high-dimensional function for evaluating a single point.</i>
--------------	-------------------------------------------------------------------------------------------

---

**Description**

TF\_moon\_high: Moon (2010) high-dimensional function for evaluating a single point.

**Usage**

```
TF_moon_high(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_moon_high(rep(0,20))  
TF_moon_high(rep(1,20))
```

---

TF_morris	<i>TF_morris: morris function for evaluating a single point.</i>
-----------	------------------------------------------------------------------

---

**Description**

TF\_morris: morris function for evaluating a single point.

**Usage**

```
TF_morris(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

[http://www.abe.ufl.edu/jjones/ABE\\_5646/2010/Morris.1991](http://www.abe.ufl.edu/jjones/ABE_5646/2010/Morris.1991)

**Examples**

```
TF_morris(rep(0,20))
TF_morris(rep(1,20))
```

---

TF\_piston

*TF\_piston: Piston simulation function for evaluating a single point.*

---

**Description**

TF\_piston: Piston simulation function for evaluating a single point.

**Usage**

```
TF_piston(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_piston(c(30,.005,.002,1e3,9e4,290,340)) # minimum of zero, hard to solve
```

---

TF\_quad\_peaks

*TF\_quad\_peaks: quad\_peaks function for evaluating a single point.*

---

**Description**

TF\_quad\_peaks: quad\_peaks function for evaluating a single point.

**Usage**

```
TF_quad_peaks(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_quad_peaks(rep(0,2))  
TF_quad_peaks(rep(1,2))
```

---

TF_quad_peaks_slant	<i>TF_quad_peaks_slant: quad_peaks_slant function for evaluating a single point.</i>
---------------------	--------------------------------------------------------------------------------------

---

**Description**

TF\_quad\_peaks\_slant: quad\_peaks\_slant function for evaluating a single point.

**Usage**

```
TF_quad_peaks_slant(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_quad_peaks_slant(rep(0,2))  
TF_quad_peaks_slant(rep(1,2))
```

---

TF_rastrigin	<i>TF_rastrigin: Rastrigin function for evaluating a single point.</i>
--------------	------------------------------------------------------------------------

---

**Description**

TF\_rastrigin: Rastrigin function for evaluating a single point.

**Usage**

```
TF_rastrigin(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_rastrigin(rep(0,2))
TF_rastrigin(rep(1,2))
```

---

TF_robotarm	<i>TF_robotarm: Robot arm function for evaluating a single point.</i>
-------------	-----------------------------------------------------------------------

---

**Description**

TF\_robotarm: Robot arm function for evaluating a single point.

**Usage**

```
TF_robotarm(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_robotarm(rep(0,8))
TF_robotarm(rep(1,8))
```

---

TF_RoosArnold	<i>TF_RoosArnold: Roos &amp; Arnold (1963) function for evaluating a single point.</i>
---------------	----------------------------------------------------------------------------------------

---

**Description**

TF\_RoosArnold: Roos & Arnold (1963) function for evaluating a single point.

**Usage**

```
TF_RoosArnold(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_RoosArnold(rep(0,8))  
TF_RoosArnold(rep(1,8))
```

---

TF\_steelcolumnstress    *TF\_steelcolumnstress: steelcolumnstress function for evaluating a single point.*

---

**Description**

TF\_steelcolumnstress: steelcolumnstress function for evaluating a single point.

**Usage**

```
TF_steelcolumnstress(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Kuschel, Norbert, and Rudiger Rackwitz. "Two basic problems in reliability-based structural optimization." *Mathematical Methods of Operations Research* 46, no. 3 (1997): 309-333.

Prikhodko, Pavel, and Nikita Kotlyarov. "Calibration of Sobol indices estimates in case of noisy output." arXiv preprint arXiv:1804.00766 (2018).

**Examples**

```
TF_steelcolumnstress(rep(0,8))  
TF_steelcolumnstress(rep(1,8))
```

---

TF_SWNExpCos	<i>TF_SWNExpCos: SWNExpCos function for evaluating a single point.</i>
--------------	------------------------------------------------------------------------

---

**Description**

TF\_SWNExpCos: SWNExpCos function for evaluating a single point.

**Usage**

TF\_SWNExpCos(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Santner, T. J., Williams, B. J., & Notz, W. (2003). The Design and Analysis of Computer Experiments. Springer Science & Business Media.

**Examples**

```
TF_SWNExpCos(rep(0,2))
TF_SWNExpCos(rep(1,2))
```

---

TF_vertigrad	<i>TF_vertigrad: vertigrad function for evaluating a single point.</i>
--------------	------------------------------------------------------------------------

---

**Description**

TF\_vertigrad: vertigrad function for evaluating a single point.

**Usage**

TF\_vertigrad(x)

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_vertigrad(rep(0,2))  
TF_vertigrad(rep(1,2))
```

---

TF_vertigrad_grad	<i>TF_vertigrad_grad: vertigrad_grad function for evaluating a single point.</i>
-------------------	----------------------------------------------------------------------------------

---

**Description**

TF\_vertigrad\_grad: vertigrad\_grad function for evaluating a single point.

**Usage**

```
TF_vertigrad_grad(x)
```

**Arguments**

x                      Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Forrester, A., & Keane, A. (2008). Engineering design via surrogate modelling: a practical guide. John Wiley & Sons.

**Examples**

```
TF_vertigrad_grad(rep(0,2))  
TF_vertigrad_grad(rep(1,2))
```

---

TF_welch	<i>TF_welch: Welch function for evaluating a single point.</i>
----------	----------------------------------------------------------------

---

**Description**

TF\_welch: Welch function for evaluating a single point.

**Usage**

```
TF_welch(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**Examples**

```
TF_welch(rep(0,20)) # minimum of zero, hard to solve
```

---

TF\_wingweight

*TF\_wingweight: Wing weight function for evaluating a single point.*

---

**Description**

TF\_wingweight: Wing weight function for evaluating a single point.

**Usage**

```
TF_wingweight(x)
```

**Arguments**

x                    Input vector at which to evaluate.

**Value**

Function output evaluated at x.

**References**

Forrester, A., & Keane, A. (2008). Engineering design via surrogate modelling: a practical guide. John Wiley & Sons.

**Examples**

```
TF_wingweight(c(150,220,6,-10,16,.5,.08,2.5,1700,.025)) # minimum of zero, hard to solve
```

---

`TF_winkel`*TF\_winkel: winkel function for evaluating a single point.*

---

**Description**

TF\_winkel: winkel function for evaluating a single point.

**Usage**

```
TF_winkel(x)
```

**Arguments**

`x`                    Input vector at which to evaluate.

**Value**

Function output evaluated at `x`.

**References**

Winkel, Munir A., Jonathan W. Stallings, Curt B. Storlie, and Brian J. Reich. "Sequential Optimization in Locally Important Dimensions." arXiv preprint arXiv:1804.10671 (2018).

**Examples**

```
TF_winkel(rep(0,2))  
TF_winkel(rep(1,2))
```

# Index

ackley (bananagramacy2Dexp), 5  
add\_linear\_terms, 3  
add\_noise, 3  
add\_null\_dims, 4  
add\_zoom, 5

banana (bananagramacy2Dexp), 5  
banana\_grad (bananagramacy2Dexp), 5  
bananagramacy2Dexp, 5  
bananatimesgramacy2Dexp  
    (bananagramacy2Dexp), 5  
beale (bananagramacy2Dexp), 5  
beambending (bananagramacy2Dexp), 5  
borehole (bananagramacy2Dexp), 5  
boreholeMV (bananagramacy2Dexp), 5  
branin (bananagramacy2Dexp), 5

chengsandu (bananagramacy2Dexp), 5  
currin1991 (bananagramacy2Dexp), 5  
currin1991b (bananagramacy2Dexp), 5

detpep8d (bananagramacy2Dexp), 5

easom (bananagramacy2Dexp), 5

franke (bananagramacy2Dexp), 5  
funcprofile, 16

gaussian1 (bananagramacy2Dexp), 5  
Gfunction (bananagramacy2Dexp), 5  
GoldsteinPrice (bananagramacy2Dexp), 5  
GoldsteinPriceLog (bananagramacy2Dexp),  
    5  
gramacy2Dexp (bananagramacy2Dexp), 5  
gramacy2Dexp3hole (bananagramacy2Dexp),  
    5  
gramacy6D (bananagramacy2Dexp), 5  
griewank (bananagramacy2Dexp), 5

hartmann (bananagramacy2Dexp), 5  
hump (bananagramacy2Dexp), 5

levy (bananagramacy2Dexp), 5  
levytilt (bananagramacy2Dexp), 5  
limnonpoly (bananagramacy2Dexp), 5  
limpoly (bananagramacy2Dexp), 5  
linkletter\_nosignal  
    (bananagramacy2Dexp), 5  
logistic (bananagramacy2Dexp), 5  
logistic15 (bananagramacy2Dexp), 5  
logistic\_plateau (bananagramacy2Dexp), 5

michalewicz (bananagramacy2Dexp), 5  
moon\_high (bananagramacy2Dexp), 5  
morris (bananagramacy2Dexp), 5

nsin, 17  
numGrad, 17  
numHessian, 18

OTL\_Circuit (bananagramacy2Dexp), 5

piston (bananagramacy2Dexp), 5  
powsin (bananagramacy2Dexp), 5

quad\_peaks (bananagramacy2Dexp), 5  
quad\_peaks\_slant (bananagramacy2Dexp), 5

rastrigin (bananagramacy2Dexp), 5  
RFF, 18  
RFF\_get, 19  
robotarm (bananagramacy2Dexp), 5  
RoosArnold (bananagramacy2Dexp), 5

sinumoid (bananagramacy2Dexp), 5  
sqrtsin (bananagramacy2Dexp), 5  
standard\_test\_func, 20  
steelcolumnstress (bananagramacy2Dexp),  
    5  
subtractlm, 21  
SWNExpCos (bananagramacy2Dexp), 5  
test\_func\_apply (bananagramacy2Dexp), 5

test\_func\_applyM0, 21  
TF\_ackley, 22  
TF\_banana (TF\_branin), 25  
TF\_banana\_grad (TF\_branin), 25  
TF\_bananagramacy2Dexp, 23  
TF\_bananatimesgramacy2Dexp, 24  
TF\_beale, 24  
TF\_beambending, 25  
TF\_borehole (TF\_branin), 25  
TF\_boreholeMV (TF\_branin), 25  
TF\_branin, 25  
TF\_chengsandou, 28  
TF\_currin1991 (TF\_branin), 25  
TF\_currin1991b (TF\_branin), 25  
TF\_detpep8d, 28  
TF\_easom, 29  
TF\_franke (TF\_branin), 25  
TF\_gaussian1 (TF\_branin), 25  
TF\_Gfunction, 30  
TF\_GoldsteinPrice, 30  
TF\_GoldsteinPriceLog, 31  
TF\_gramacy2Dexp, 31  
TF\_gramacy2Dexp3hole, 32  
TF\_gramacy6D, 33  
TF\_griewank, 33  
TF\_hartmann, 34  
TF\_hump, 34  
TF\_levy, 35  
TF\_levytilt, 35  
TF\_limnonpoly (TF\_branin), 25  
TF\_limpoly (TF\_branin), 25  
TF\_linkletter\_nosignal, 36  
TF\_logistic, 36  
TF\_logistic15, 37  
TF\_logistic\_plateau, 38  
TF\_michalewicz, 38  
TF\_moon\_high, 39  
TF\_morris, 39  
TF\_OTL\_Circuit (TF\_branin), 25  
TF\_piston, 40  
TF\_powsin (TF\_branin), 25  
TF\_quad\_peaks, 40  
TF\_quad\_peaks\_slant, 41  
TF\_rastrigin, 41  
TF\_robotarm, 42  
TF\_RoosArnold, 42  
TF\_sinumoid (TF\_branin), 25  
TF\_sqrtsin (TF\_branin), 25  
TF\_steelcolumnstress, 43  
TF\_SWNExpCos, 44  
TF\_vertigrad, 44  
TF\_vertigrad\_grad, 45  
TF\_welch, 45  
TF\_wingweight, 46  
TF\_winkel, 47  
TF\_zhou1998 (TF\_branin), 25  
  
vertigrad (bananagramacy2Dexp), 5  
vertigrad\_grad (bananagramacy2Dexp), 5  
vsin (nsin), 17  
  
waterfall (bananagramacy2Dexp), 5  
welch (bananagramacy2Dexp), 5  
wingweight (bananagramacy2Dexp), 5  
winkel (bananagramacy2Dexp), 5  
  
zhou1998 (bananagramacy2Dexp), 5