Table of Contents

Guaranteed Automatic Integration Library (GAIL) 2.1 User Guide 2
Functions 2
Installation 2
Tests 2
Website 3
Functions 3
1-D approximation 3
1-D integration 3
High dimension integration 3
1-D minimization 3
funappx_g 3
Syntax 3
Description 3
Guarantee 5
Examples 5
See Also 6
References 6
funmin_g 6
Syntax 6
Description 6
Guarantee 8
Examples 8
See Also 9
References 9
integral_g 9
Syntax 9
Description 10
Guarantee 11
Examples 12
See Also 12
References 12
meanMC_g 12
Syntax 12
Description 13
Guarantee 14
Examples 15
See Also 15
References 15
meanMCBer_g 15
Syntax 16
Description 16
Guarantee 16
Examples 17
See Also 17
References 17
cubMC_g 17
Syntax 17
Description 18
Guarantee 20

Examples 20
See Also 21
References 21
cubLattice_g 21
Syntax 21
Description 21
Guarantee 24
Examples 24
See Also 25
References 25
cubSobol_g 25
Syntax 25
Description 25
Guarantee 27
Examples 27
See Also 28
References 28
Installation Instructions 28

Guaranteed Automatic Integration Library (GAIL) 2.1 User Guide

GAIL (Guaranteed Automatic Integration Library) is created, developed, and maintained by Fred Hickernell (Illinois Institute of Technology), Sou-Cheng Choi (NORC at the University of Chicago and IIT), Yuhan Ding (IIT), Lan Jiang (IIT), Lluis Antoni Jimenez Rugama (IIT), Xin Tong (University of Illinois at Chicago), Yizhi Zhang (IIT), and Xuan Zhou (IIT).

GAIL is a suite of algorithms for integration problems in one, many, and infinite dimensions, and whose answers are guaranteed to be correct.

```
%help GAU
```

```
Error: File: /Users/terrya/GAIL_Dev/GAIL_Matlab/Documentation/gail_ug2_1.m
Unexpected MATLAB operator.
```

Functions

Installation

Tests

We provide quick doctests for each of the functions above. To run doctests in **funappx_g**, for example, issue the command **doctest funappx_g**.

We also provide unit tests for MATLAB version 8 or later. To run unit tests for **funmin_g**, for instance, execute **run(ut_funmin_g);**

To run all the fast doctests and unit tests in the suite, execute the script **runtests.m**.

A collection of long tests are contained in **longtests.m**.

Website

For more information about GAIL, visit [Gailteam]

Functions

1-D approximation

1-D integration

High dimension integration

1-D minimization

funappx_g

`1-D guaranteed function recovery on a closed interval [a,b]`

Syntax

fappx = **funappx_g**(f)

fappx = **funappx_g**(f,a,b,abstol)

fappx = **funappx_g**(f,'a',a,'b',b,'abstol',abstol)

fappx = **funappx_g**(f,in_param)

[fappx, out_param] = **funappx_g**(f,...)

Description

fappx = **funappx_g**(f) approximates function f on the default interval [0,1] by an approximated function fappx within the guaranteed absolute error tolerance of 1e-6. Input f is a function handle. The statement y = f(x) should accept a vector argument x and return a vector y of function values that is of the same size as x.

fappx = **funappx_g**(f,a,b,abstol) for a given function f and the ordered input parameters that define the finite interval [a,b], and a guaranteed absolute error tolerance abstol.

fappx = **funappx_g**(f,'a',a,'b',b,'abstol',abstol) recovers function f on the finite interval [a,b], given a guaranteed absolute error tolerance abstol. All four field-value pairs are optional and can be supplied in different order.

fappx = **funappx_g**(f,in_param) recovers function f on the finite interval [in_param.a,in_param.b], given a guaranteed absolute error tolerance in_param.abstol. If a field is not specified, the default value is used.

[fappx, out_param] = **funappx_g**(f,...) returns an approximated function fappx and an output structure out_param.

Input Arguments

- f --- `input function`
- in_param.a --- `left end point of interval, default value is 0`
- in_param.b --- `right end point of interval, default value is 1`
- in_param.abstol --- `guaranteed absolute error tolerance, default value is 1e-6`

Optional Input Arguments (Recommended not to change very often)

- in_param.nlo --- `lower bound of initial number of points we used, default value is 10`
- in_param.nhi --- `upper bound of initial number of points we used, default value is 1000`
- in_param.nmax --- `when number of points hits the value, iteration will stop, default value is 1e7`
- in_param.maxiter --- `max number of interations, default value is 1000`

Output Arguments

- fappx --- `approximated function`
- out_param.f --- `input function`
- out_param.a --- `left end point of interval`
- out_param.b --- `right end point of interval`
- out_param.abstol --- `guaranteed absolute error tolerance`
- out_param.nlo --- `a lower bound of initial number of points we use`
- out_param.nhi --- `an upper bound of initial number of points we use`
- out_param.nmax --- `when number of points hits the value, iteration will stop`
- out_param.maxiter --- `max number of iterations`
- out_param.ninit --- `initial number of points we use for each sub interval`
- out_param.exit --- `this is a number defining the conditions of success or failure satisfied when finishing the algorithm. The algorithm is considered successful (with out_param.exit == 0) if no other flags arise warning that the results are certainly not guaranteed. The initial value is 0 and the final value of this parameter is encoded as follows:`

```
1  |If reaching overbudget. It states whether
the max budget is attained without reaching the
```

guaranteed error tolerance.

2 |If reaching overiteration. It states whether the max iterations is attained without reaching the guaranteed error tolerance.|

- out_param.iter --- number of iterations
- out_param.npoints --- number of points we need to reach the guaranteed absolute error tolerance
- out_param.errest --- an estimation of the absolute error for the approximation
- out_param.nstar --- final value of the parameter defining the cone of functions for which this algorithm is guaranteed for each subinterval; nstar = ninit-2 initially

Guarantee

For $[a, b]$, there exists a partition

$$P = \{[t_0, t_1], [t_1, t_2], \ldots, [t_{L-1}, t_L]\}, a = t_0 < t_1 < \cdots < t_L = b.$$

If the function to be approximated, f satisfies the cone condition

$$\|f''\|_\infty \le \frac{2\text{nstar}}{t_l - t_{l-1}} \left\| f' - \frac{f(t_l) - f(t_{l-1})}{t_l - t_{l-1}} \right\|_\infty$$

for each sub interval $[t_{l-1}, t_l]$, where $1 \le l \le L$, then the pp output by this algorithm is guaranteed to satisfy

$$\|f - ppval(pp,)\|_\infty \le \text{abstol}$$

Examples

Example 1

```
f = @(x) x.^2; [pp, out_param] = funappx_g(f)

% Approximate function x^2 with default input parameter to make the error
% less than 1e-6.
```

Example 2

```
[pp, out_param] = funappx_g(@(x) x.^2,0,100,1e-7,10,1000,1e8)

% Approximate function x^2 on [0,100] with error tolerence 1e-7, cost
% budget 10000000, lower bound of initial number of points 10 and upper
% bound of initial number of points 100
```

Example 3

```
clear in_param; in_param.a = -20; in_param.b = 20; in_param.nlo = 10;
in_param.nhi = 100; in_param.nmax = 1e8; in_param.abstol = 1e-7;
```

```
[pp, out_param] = funappx_g(@(x) x.^2, in_param)

% Approximate function x^2 on [-20,20] with error tolerence 1e-7, cost
% budget 1000000, lower bound of initial number of points 10 and upper
% bound of initial number of points 100
```

Example 4

```
clear in_param; f = @(x) x.^2;
[pp, out_param] = funappx_g(f,'a',-10,'b',50,'nmax',1e6,'abstol',1e-8)

% Approximate function x^2 with error tolerence 1e-8, cost budget 1000000,
% lower bound of initial number of points 10 and upper
% bound of initial number of points 100
```

See Also

References

[1] Nick Clancy, Yuhan Ding, Caleb Hamilton, Fred J. Hickernell, and Yizhi Zhang, The Cost of Deterministic, Adaptive, Automatic Algorithms: Cones, Not Balls, Journal of Complexity 30 (2014), pp. 21-45.

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

funmin_g

```
Guaranteed global minimum value of univariate function on a closed
interval [a,b] and the subset containing optimal solutions
```

Syntax

fmin = **funmin_g**(f)

fmin = **funmin_g**(f,a,b,abstol,TolX)

fmin = **funmin_g**(f,'a',a,'b',b,'abstol',abstol,'TolX',TolX)

fmin = **funmin_g**(f,in_param)

[fmin, out_param] = **funmin_g**(f,...)

Description

fmin = **funmin_g**(f) finds minimum value of function f on the default interval [0,1] within the guaranteed absolute error tolerance of 1e-6 and the X tolerance of 1e-3. Default initial number of points is 100 and default cost budget is 1e7. Input f is a function handle.

fmin = **funmin_g**(f,a,b,abstol,TolX) finds minimum value of function f with ordered input parameters that define the finite interval [a,b], a guaranteed absolute error tolerance abstol and a guaranteed X tolerance TolX.

fmin = **funmin_g**(f,'a',a,'b',b,'abstol',abstol,'TolX',TolX) finds minimum value of function f on the interval [a,b] with a guaranteed absolute error tolerance abstol and a guaranteed X tolerance TolX. All five field-value pairs are optional and can be supplied in different order.

fmin = **funmin_g**(f,in_param) finds minimum value of function f on the interval [in_param.a,in_param.b] with a guaranteed absolute error tolerance in_param.abstol and a guranteed X tolerance in_param.TolX. If a field is not specified, the default value is used.

[fmin, out_param] = **funmin_g**(f,...) returns minimum value fmin of function f and an output structure out_param.

Input Arguments

- f --- `input function`
- in_param.a --- `left end point of interval, default value is 0`
- in_param.b --- `right end point of interval, default value is 1`
- in_param.abstol --- `guaranteed absolute error tolerance, default value is 1e-6.`
- in_param.TolX --- `guaranteed X tolerance, default value is 1e-3.`

```
%  tional Input Arguments (Recommended not to change very often) |
%
% * in_param.nlo --- |lower bound of initial number of points we used,
%  default value is 10|
%
% * in_param.nhi --- |upper bound of initial number of points we used,
%  default value is 1000|
%
% * in_param.nmax --- |cost budget, default value is 1e7.|
%
% *Output Arguments*
%
% * out_param.f --- |input function|
%
% * out_param.a --- |left end point of interval|
%
% * out_param.b --- |right end point of interval|
%
% * out_param.abstol --- |guaranteed absolute error tolerance|
%
% * out_param.TolX --- |guaranteed X tolerance|
%
% * out_param.nlo --- |a lower bound of initial number of points we use|
%
% * out_param.nhi --- |an upper bound of initial number of points we use|
%
% * out_param.nmax --- |cost budget|
%
```

```
% * out_param.ninit --- |initial number of points we use|
%
% * out_param.tau --- |latest value of tau|
%
% * out_param.npoints --- |number of points needed to reach the guaranteed
%  absolute error tolerance or the guaranteed X tolerance|
%
% * out_param.exitflag --- |the state of program when exiting
%            0  Success
%            1  Number of points used is greater than out_param.nmax|
%
% * out_param.errest --- |estimation of the absolute error bound|
%
% * out_param.volumeX --- |the volume of intervals containing the point(s)
%  where the minimum occurs|
%
% * out_param.tauchange --- |it is 1 if out_param.tau changes, otherwise
%  it is 0|
%
% * out_param.intervals --- |the intervals containing point(s) where the
%  minimum occurs. Each column indicates one interval where the first
%  row is the left point and the second row is the right point.|
%
```

Guarantee

If the function to be minimized, f satisfies the cone condition

$$\|f''\|_\infty \le \frac{\tau}{b-a}\left\|f' - \frac{f(b)-f(a)}{b-a}\right\|_\infty,$$

then the fmin output by this algorithm is guaranteed to satisfy

$$|\min f - \mathrm{fmin}| \le \mathrm{abstol}$$

or

\mathrm{volumeX} \le \mathrm{TolX},

provided the flag $\mathrm{exceedbudget} = 0.$

Examples

```
% *Example 1*

f=@(x) (x-0.3).^2+1; [fmin,out_param] = funmin_g(f)

% Minimize function (x-0.3)^2+1 with default input parameter.
```

Example 2

```
f=@(x) (x-0.3).^2+1;
```

```
[fmin,out_param] = funmin_g(f,-2,2,1e-7,1e-4,10,10,1000000)

% Minimize function (x-0.3)^2+1 on [-2,2] with error tolerence 1e-4, X
% tolerance 1e-2, cost budget 1000000, lower bound of initial number of
% points 10 and upper bound of initial number of points 10
```

Example 3

```
clear in_param; in_param.a = -13; in_param.b = 8;
in_param.abstol = 1e-7; in_param.TolX = 1e-4;
in_param.nlo = 10; in_param.nhi = 100;
in_param.nmax = 10^6;
[fmin,out_param] = funmin_g(f,in_param)

% Minimize function (x-0.3)^2+1 on [-13,8] with error tolerence 1e-7, X
% tolerance 1e-4, cost budget 1000000, lower bound of initial number of
% points 10 and upper bound of initial number of points 100
```

Example 4

```
f=@(x) (x-0.3).^2+1;
[fmin,out_param] = funmin_g(f,'a',-2,'b',2,'nhi',100,'nlo',10,'nmax',1e6,'abstol',

% Minimize function (x-0.3)^2+1 on [-2,2] with error tolerence 1e-4, X
% tolerance 1e-2, cost budget 1000000, lower bound of initial number of
% points 10 and upper bound of initial number of points 100
```

See Also

References

[1] Xin Tong. A Guaranteed, Adaptive, Automatic Algorithm for Univariate Function Minimization. MS thesis, Illinois Institute of Technology, 2014.

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

integral_g

```
1-D guaranteed function integration using trapezoidal rule
```

Syntax

q = **integral_g**(f)

q = **integral_g**(f,a,b,abstol)

q = **integral_g**(f,'a',a,'b',b,'abstol',abstol)

q = **integral_g**(f,in_param)

[q, out_param] = **integral_g**(f,...)

Description

q = **integral_g**(f) computes q, the definite integral of function f on the interval [a,b] by trapezoidal rule with in a guaranteed absolute error of 1e-6. Default starting number of sample points taken is 100 and default cost budget is 1e7. Input f is a function handle. The function y = f(x) should accept a vector argument x and return a vector result y, the integrand evaluated at each element of x.

q = **integral_g**(f,a,b,abstol) computes q, the definite integral of function f on the finite interval [a,b] by trapezoidal rule with the ordered input parameters, and guaranteed absolute error tolerance abstol.

q = **integral_g**(f,'a',a,'b',b,'abstol',abstol) computes q, the definite integral of function f on the finite interval [a,b] by trapezoidal rule within a guaranteed absolute error tolerance abstol. All four field-value pairs are optional and can be supplied.

q = **integral_g**(f,in_param) computes q, the definite integral of function f by trapezoidal rule within a guaranteed absolute error in_param.abstol. If a field is not specified, the default value is used.

[q, out_param] = **integral_g**(f,...) returns the approximated integration q and output structure out_param.

Input Arguments

```
|
%
% * f --- |input function|
%
% * in_param.a --- |left end of the integral, default value is 0|
%
% * in_param.b --- |right end of the integral, default value is 1|
%
% * in_param.abstol --- |guaranteed absolute error tolerance, default value
%  is 1e-6|
%

%  tional Input Arguments (Recommended not to change very often) |
%
% * in_param.nlo --- |lowest initial number of function values used, default
%  value is 10|
%
% * in_param.nhi --- |highest initial number of function values used,
%  default value is 1000|
%
% * in_param.nmax --- |cost budget (maximum number of function values),
%  default value is 1e7|
%
% * in_param.maxiter --- |max number of interations, default value is 1000|
%
% *Output Arguments*
%
```

```
% * q --- |approximated integral|
%
% * out_param.f --- |input function|
%
% * out_param.a --- |low end of the integral|
%
% * out_param.b --- |high end of the integral|
%
% * out_param.abstol --- |guaranteed absolute error tolerance|
%
% * out_param.nlo --- |lowest initial number of function values|
%
% * out_param.nhi --- |highest initial number of function values|
%
% * out_param.nmax --- |cost budget (maximum number of function values)|
%
% * out_param.maxiter --- |max number of iterations|
%
% * out_param.ninit --- |initial number of points we use, computed by nlo
%  and nhi|
%
% * out_param.exceedbudget --- |it is true if the algorithm tries to use
%   more points than cost budget, false otherwise.|
%
% * out_param.tauchange --- |it is true if the cone constant has been
%  changed, false otherwise. See [1] for details. If true, you may wish to
%  change the input in_param.ninit to a larger number.|
%
% * out_param.iter --- |number of iterations|
%
% * out_param.npoints --- |number of points we need to
%  reach the guaranteed absolute error tolerance abstol.|
%
% * out_param.errest --- |approximation error defined as the differences
%  between the true value and the approximated value of the integral.|
%
% * out_param.nstar --- |final value of the parameter defining the cone of
%  functions for which this algorithm is guaranteed; nstar = ninit-2
%  initially and is increased as necessary|
%
```

Guarantee

If the function to be integrated, f satisfies the cone condition

$$\|f''\|_1 \le \frac{\text{nstar}}{2(b-a)}\left\|f' - \frac{f(b)-f(a)}{b-a}\right\|_1,$$

then the q output by this algorithm is guaranteed to satisfy

$$\left\|\int_a^b f(x)dx - q\right\|_1 \le \text{abstol}$$

provided the flag $\text{exceedbudget} = 0$.

And the upper bound of the cost is

$$\sqrt{\frac{\text{nstar}*(b-a)^2 \text{Var}(f')}{2 \times \text{abstol}}} + 2 \times \text{nstar} + 4.$$

Examples

Example 1

```
f = @(x) x.^2; [q, out_param] = integral_g(f)

% Integrate function x with default input parameter to make the error less
% than 1e-7.
```

Example 2

```
[q, out_param] = integral_g(@(x) exp(-x.^2),'a',1,'b',2,'nlo',100,'nhi',10000,'abs

% Integrate function x^2 with starting number of points 52, cost budget
% 10000000 and error toerence 1e-8
```

See Also

References

[1] Nick Clancy, Yuhan Ding, Caleb Hamilton, Fred J. Hickernell, and Yizhi Zhang, The Cost of Deterministic, Adaptive, Automatic Algorithms: Cones, Not Balls, Journal of Complexity 30 (2014), pp. 21-45.

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

meanMC_g

```
MEANMC_G Monte Carlo method to estimate the mean of a random variable.
```

Syntax

tmu = **meanMC_g**(Yrand)

tmu = **meanMC_g**(Yrand,abstol,reltol,alpha)

tmu = **meanMC_g**(Yrand,'abstol',abstol,'reltol',reltol,'alpha',alpha)

[tmu, out_param] = **meanMC_g**(Yrand,in_param)

Description

tmu = **meanMC_g**(Yrand) estimates the mean, mu, of a random variable Y to within a specified generalized error tolerance, tolfun:=max(abstol,reltol*|mu|), i.e., `mu - tmu` <= tolfun with probability at least 1-alpha, where abstol is the absolute error tolerance, and reltol is the relative error tolerance. Usually the reltol determines the accuracy of the estimation, however, if the `mu` is rather small, the abstol determines the accuracy of the estimation. The default values are abstol=1e-2, reltol=1e-1, and alpha=1%. Input Yrand is a function handle that accepts a positive integer input n and returns an n x 1 vector of IID instances of the random variable Y.

tmu = **meanMC_g**(Yrand,abstol,reltol,alpha) estimates the mean of a random variable Y to within a specified generalized error tolerance tolfun with guaranteed confidence level 1-alpha using all ordered parsing inputs abstol, reltol, alpha.

tmu = **meanMC_g**(Yrand,'abstol',abstol,'reltol',reltol,'alpha',alpha) estimates the mean of a random variable Y to within a specified generalized error tolerance tolfun with guaranteed confidence level 1-alpha. All the field-value pairs are optional and can be supplied in different order, if a field is not supplied, the default value is used.

[tmu, out_param] = **meanMC_g**(Yrand,in_param) estimates the mean of a random variable Y to within a specified generalized error tolerance tolfun with the given parameters in_param and produce the estimated mean tmu and output parameters out_param. If a field is not specified, the default value is used.

Input Arguments

- Yrand --- `the function for generating n IID instances of a random variable Y whose mean we want to estimate. Y is often defined as a function of some random variable X with a simple distribution. The input of Yrand should be the number of random variables n, the output of Yrand should be n function values. For example, if Y = X.^2 where X is a standard uniform random variable, then one may define Yrand = @(n) rand(n,1).^2.`

- in_param.abstol --- `the absolute error tolerance, which should be positive, default value is 1e-2.`

- in_param.reltol --- `the relative error tolerance, which should be between 0 and 1, default value is 1e-1.`

- in_param.alpha --- `the uncertainty, which should be a small positive percentage. default value is 1%.`

```
%  Optional input parameters:|
%
% * in_param.fudge --- |standard deviation inflation factor, which should
%  be larger than 1, default value is 1.2.|
%
% * in_param.nSig --- |initial sample size for estimating the sample
%  variance, which should be a moderate large integer at least 30, the
%  default value is 1e4.|
%
% * in_param.n1 --- |initial sample size for estimating the sample mean,
%  which should be a moderate large positive integer at least 30, the
%  default value is 1e4.|
%
```

```
% * in_param.tbudget --- |the time budget in seconds to do the two-stage
%  estimation, which should be positive, the default value is 100 seconds.|
%
% * in_param.nbudget --- |the sample budget to do the two-stage
%  estimation, which should be a large positive integer, the default
%  value is 1e9.|
%
% *Output Arguments*
%
% * tmu --- |the estimated mean of Y.|
%
% * out_param.tau --- |the iteration step.|
%
% * out_param.n --- |the sample size used in each iteration.|
%
% * out_param.nremain --- |the remaining sample budget to estimate mu. It was
%  calculated by the sample left and time left.|
%
% * out_param.ntot --- |total sample used.|
%
% * out_param.hmu --- |estimated mean in each iteration.|
%
% * out_param.tol --- |the reliable upper bound on error for each iteration.|
%
% * out_param.var --- |the sample variance.|
%
% * out_param.exit --- |the state of program when exiting.
%
%                    0   Success
%
%                    1   Not enough samples to estimate the mean|
%
% * out_param.kurtmax --- |the upper bound on modified kurtosis.|
%
% * out_param.time --- |the time elapsed in seconds.|
%
% * out_param.flag --- |parameter checking status
%
%                         1  checked by meanMC_g|
%
```

Guarantee

This algorithm attempts to calculate the mean, mu, of a random variable to a prescribed error tolerance, tolfun:= max(abstol,reltol*|mu|), with guaranteed confidence level 1-alpha. If the algorithm terminated without showing any warning messages and provide an answer tmu, then the follow inequality would be satisfied:

Pr(`mu-tmu` <= tolfun) >= 1-alpha

The cost of the algorithm, N_tot, is also bounded above by N_up, which is defined in terms of abstol, reltol, nSig, n1, fudge, kurtmax, beta. And the following inequality holds:

Pr (N_tot <= N_up) >= 1-beta

Please refer to our paper for detailed arguments and proofs.

Examples

Example 1

```
% Calculate the mean of x^2 when x is uniformly distributed in
% [0 1], with the absolute error tolerance = 1e-3 and uncertainty 5%.

  in_param.reltol=0; in_param.abstol = 1e-3;in_param.reltol = 0;
  in_param.alpha = 0.05; Yrand=@(n) rand(n,1).^2;
  tmu = meanMC_g(Yrand,in_param)
```

Example 2

```
% Calculate the mean of exp(x) when x is uniformly distributed in
% [0 1], with the absolute error tolerance 1e-3.

  tmu = meanMC_g(@(n)exp(rand(n,1)),1e-3,0)
```

Example 3

```
% Calculate the mean of cos(x) when x is uniformly distributed in
% [0 1], with the relative error tolerance 1e-2 and uncertainty 0.05.

  tmu = meanMC_g(@(n)cos(rand(n,1)),'reltol',1e-2,'abstol',0,'alpha',0.05)
```

See Also

References

[1] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen, Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling, Monte Carlo and Quasi-Monte Carlo Methods 2012 (J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, eds.), Springer-Verlag, Berlin, 2014. arXiv:1208.4318 [math.ST]

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

meanMCBer_g

```
Monte Carlo method to estimate the mean of a Bernoulli random variable to
within a specified absolute error tolerance with guaranteed confidence
level 1-alpha.
```

Syntax

pHat = **meanMCBer_g**(Yrand)

pHat = **meanMCBer_g**(Yrand,abstol,alpha,nmax)

pHat = **meanMCBer_g**(Yrand,'abstol',abstol,'alpha',alpha,'nmax',nmax)

[pHat, out_param] = **meanMCBer_g**(Yrand,in_param)

Description

pHat = **meanMCBer_g**(Yrand) estimates the mean of a Bernoulli random variable Y to within a specified absolute error tolerance with guaranteed confidence level 99%. Input Yrand is a function handle that accepts a positive integer input n and returns a n x 1 vector of IID instances of the Bernoulli random variable Y.

pHat = **meanMCBer_g**(Yrand,abstol,alpha,nmax) estimates the mean of a Bernoulli random variable Y to within a specified absolute error tolerance with guaranteed confidence level 1-alpha using all ordered parsing inputs abstol, alpha and nmax.

pHat = **meanMCBer_g**(Yrand,'abstol',abstol,'alpha',alpha,'nmax',nmax) estimates the mean of a Bernoulli random variable Y to within a specified absolute error tolerance with guaranteed confidence level 1-alpha. All the field-value pairs are optional and can be supplied in different order.

[pHat, out_param] = **meanMCBer_g**(Yrand,in_param) estimates the mean of a Bernoulli random variable Y to within a specified absolute error tolerance with the given parameters in_param and produce the estimated mean pHat and output parameters out_param.

Input Arguments

- Yrand --- `the function for generating IID instances of a Bernoulli random variable Y whose mean we want to estimate.`
- pHat --- `the estimated mean of Y.`
- in_param.abstol --- `the absolute error tolerance, the default value is 1e-2.`
- in_param.alpha --- `the uncertainty, the default value is 1%.`
- in_param.nmax --- `the sample budget, the default value is 1e9.`

Output Arguments

- out_param.n --- `the total sample used.`
- out_param.time --- `the time elapsed in seconds.`

Guarantee

If the sample size is calculated according Hoeffding's inequality, which equals to ceil(log(2/out_param.alpha)/(2*out_param.abstol^2)), then the following inequality must be satisfied:

Pr(`p-pHat` <= abstol) >= 1-alpha.

Here p is the true mean of Yrand, and pHat is the output of MEANMCBER_G

Also, the cost is deterministic.

Examples

```
*Example 1*

%   Calculate the mean of a Bernoulli random variable with true p=1/90,
%   absolute error tolerance 1e-3 and uncertainty 0.01.
%
    in_param.abstol=1e-3; in_param.alpha = 0.01; p=1/9;Yrand=@(n) rand(n,1)<p;
    pHat = meanMCBer_g(Yrand,in_param)
```

```
*Example 2*

%   Using the same function as example 1, with the absolute error tolerance
%   1e-4.
%
    pHat = meanMCBer_g(Yrand,1e-4)
```

```
*Example 3*

%   Using the same function as example 1, with the absolute error
%   tolerance 1e-2 and uncertainty 0.05.
%
    pHat = meanMCBer_g(Yrand,'abstol',1e-2,'alpha',0.05)
```

See Also

References

[1] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen, Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling, Monte Carlo and Quasi-Monte Carlo Methods 2012 (J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, eds.), Springer-Verlag, Berlin, 2014. arXiv:1208.4318 [math.ST]

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

cubMC_g

```
Monte Carlo method to evaluate a multidimensional integral.
```

Syntax

[Q,out_param] = **cubMC_g**(f,hyperbox)

Q = **cubMC_g**(f,hyperbox,measure,abstol,reltol,alpha)

Q = **cubMC_g**(f,hyperbox,'measure',measure,'abstol',abstol,'reltol',reltol,'alpha',alpha)

[Q out_param] = **cubMC_g**(f,hyperbox,in_param)

Description

[Q,out_param] = **cubMC_g**(f,hyperbox) estimates the integral of f over hyperbox to within a specified generalized error tolerance, tolfun = max(abstol, reltol*|I|), i.e., | I - Q | <= tolfun with probability at least 1-alpha, where abstol is the absolute error tolerance, and reltol is the relative error tolerance. Usually the reltol determines the accuracy of the estimation, however, if the | I | is rather small, the abstol determines the accuracy of the estimation. The default values are abstol=1e-2, reltol=1e-1, and alpha=1%. Input f is a function handle that accepts an n x d matrix input, where d is the dimension of the hyperbox, and n is the number of points being evaluated simultaneously. The input hyperbox is a 2 x d matrix, where the first row corresponds to the lower limits and the second row corresponds to the upper limits.

Q = **cubMC_g**(f,hyperbox,measure,abstol,reltol,alpha) estimates the integral of function f over hyperbox to within a specified generalized error tolerance tolfun with guaranteed confidence level 1-alpha using all ordered parsing inputs f, hyperbox, measure, abstol, reltol, alpha, fudge, nSig, n1, tbudget, nbudget, flag. The input f and hyperbox are required and others are optional.

Q = **cubMC_g**(f,hyperbox,'measure',measure,'abstol',abstol,'reltol',reltol,'alpha',alpha) estimates the integral of f over hyperbox to within a specified generalized error tolerance tolfun with guaranteed confidence level 1-alpha. All the field-value pairs are optional and can be supplied in different order. If an input is not specified, the default value is used.

[Q out_param] = **cubMC_g**(f,hyperbox,in_param) estimates the integral of f over hyperbox to within a specified generalized error tolerance tolfun with the given parameters in_param and produce output parameters out_param and the integral Q.

Input Arguments

- f --- `the integrand.`
- hyperbox --- `the integration hyperbox. The default value is [zeros(1,d); ones(1,d)], the default d is 1.`
- in_param.measure --- `the measure for generating the random variable, the default is 'uniform'. The other measure could be handled is 'normal'/'Gaussian'. The input should be a string type, hence with quotes.`
- in_param.abstol --- `the absolute error tolerance, the default value is 1e-2.`
- in_param.reltol --- `the relative error tolerance, the default value is 1e-1.`
- in_param.alpha --- `the uncertainty, the default value is 1%.`

```
%  Optional input parameters:|
%
% * in_param.fudge --- |the standard deviation inflation factor, the
%  default value is 1.2.|
%
% * in_param.nSig --- |initial sample size for estimating the sample
```

```
%  variance, which should be a moderate large integer at least 30, the
%  default value is 1e4.|
%
% * in_param.n1 --- |initial sample size for estimating the sample mean,
%  which should be a moderate large positive integer at least 30, the
%  default value is 1e4.|
%
% * in_param.tbudget --- |the time budget to do the estimation, the
%  default value is 100 seconds.|
%
% * in_param.nbudget --- |the sample budget to do the estimation, the
%  default value is 1e9.|
%
% * in_param.flag --- |the value corresponds to parameter checking status.
%
%                       0   not checked
%
%                       1   checked by meanMC_g
%
%                       2   checked by cubMC_g|
%
% *Output Arguments*
%
% * Q --- |the estimated value of the integral.|
%
% * out_param.n --- |the sample size used in each iteration.|
%
% * out_param.ntot --- |total sample used.|
%
% * out_param.nremain --- |the remaining sample budget to estimate I. It was
%  calculated by the sample left and time left.|
%
% * out_param.tau --- |the iteration step.|
%
% * out_param.hmu --- |estimated integral in each iteration.|
%
% * out_param.tol --- |the reliable upper bound on error for each iteration.|
%
% * out_param.kurtmax --- |the upper bound on modified kurtosis.|
%
% * out_param.time --- |the time elapsed in seconds.|
%
% * out_param.var --- |the sample variance.|
%
% * out_param.exit --- |the state of program when exiting.
%
%                     0   success
%
%                     1   Not enough samples to estimate the mean
%
%                     10  hyperbox does not contain numbers
%
%                     11  hyperbox is not 2 x d
%
```

```
%                     12  hyperbox is only a point in one direction
%
%                     13  hyperbox is infinite when measure is 'uniform'
%
%                     14  hyperbox is not doubly infinite when measure
%                         is 'normal'|
%
```

Guarantee

This algorithm attempts to calculate the integral of function f over a hyperbox to a prescribed error tolerance tolfun:= max(abstol,reltol*|I|) with guaranteed confidence level 1-alpha. If the algorithm terminated without showing any warning messages and provide an answer Q, then the follow inequality would be satisfied:

Pr(`Q-I` <= tolfun) >= 1-alpha

The cost of the algorithm, N_tot, is also bounded above by N_up, which is a function in terms of abstol, reltol, nSig, n1, fudge, kurtmax, beta. And the following inequality holds:

Pr (N_tot <= N_up) >= 1-beta

Please refer to our paper for detailed arguments and proofs.

Examples

Example 1

```
% Estimate the integral with integrand f(x) = sin(x) over the interval [1;2]
%

 f=@(x) sin(x);interval = [1;2];
 Q = cubMC_g(f,interval,'uniform',1e-3,1e-2)
```

Example 2 Estimate the integral with integrand f(x) = exp(-x1^2-x2^2) over the hyperbox [0 0;1 1], where x is a vector x = [x1 x2].

```
 f=@(x) exp(-x(:,1).^2-x(:,2).^2);hyperbox = [0 0;1 1];
 Q = cubMC_g(f,hyperbox,'measure','uniform','abstol',1e-3,'reltol',1e-13)
```

Example 3

```
% Estimate the integral with integrand f(x) = 2^d*prod(x1*x2*...*xd)+0.555
% over the hyperbox [zeros(1,d);ones(1,d)], where x is a vector
% x = [x1 x2... xd].
%

  d=3;f=@(x) 2^d*prod(x,2)+0.555;hyperbox =[zeros(1,d);ones(1,d)];
  in_param.abstol = 1e-3;in_param.reltol=1e-3;
  Q = cubMC_g(f,hyperbox,in_param)
```

Example 4

```
% Estimate the integral with integrand f(x) = exp(-x1^2-x2^2) in the
```

```
% hyperbox [-inf -inf;inf inf], where x is a vector x = [x1 x2].
%

 f=@(x) exp(-x(:,1).^2-x(:,2).^2);hyperbox = [-inf -inf;inf inf];
 Q = cubMC_g(f,hyperbox,'normal',0,1e-2)
```

See Also

References

[1] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen, Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling, Monte Carlo and Quasi-Monte Carlo Methods 2012 (J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, eds.), pp. 105-128, Springer-Verlag, Berlin, 2014 DOI: 10.1007/978-3-642-41095-6_5

[2] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

cubLattice_g

```
is a Quasi-Monte Carlo method using rank-1 Lattices cubature over a d-
dimensional region to integrate within a specified generalized error
tolerance with guarantees under Fourier coefficients cone decay assump-
tions.
```

Syntax

[q,out_param] = **cubLattice_g**(f,d)

q = **cubLattice_g**(f,d,abstol,reltol,measure,shift,mmin,mmax,fudge,transform,toltype,theta)

q = **cubLattice_g**(f,d,'abstol',abstol,'reltol',reltol,'measure',measure,'shift',shift,'mmin',mmin,'mmax',mmax,'fudge',fudge,'transf

q = **cubLattice_g**(f,d,in_param)

Description

[q,out_param] = **cubLattice_g**(f,d) estimates the integral of f over the d-dimensional region with an error guaranteed not to be greater than a specific generalized error tolerance, tolfun:=max(abstol,reltol*|integral(f)|). The generalized tolerance function can aslo be cosen as tolfun:=theta*abstol+(1-theta)*reltol*|integral(f)| where theta is another input parameter. Input f is a function handle. f should accept an n x d matrix input, where d is the dimension of the hypercube, and n is the number of points being evaluated simultaneously. The input d is the dimension in which the function f is defined. Given the construction of our Lattices, d must be a positive integer with 1<=d<=250.

q = **cubLattice_g**(f,d,abstol,reltol,measure,shift,mmin,mmax,fudge,transform,toltype,theta) estimates the integral of f over a d-dimensional region. The answer is given within the generalized error tolerance tolfun. All parameters should be input in the order specified above. If an input is not specified, the default value is used. Note that if an input is not specified, the remaining tail cannot be specified either.

q = **cubLattice_g**(f,d,'abstol',abstol,'reltol',reltol,'measure',measure,'shift',shift,'mmin',mmin,'mmax',mmax,'fudge',fudge,'transf estimates the integral of f over a d-dimensional region. The answer is given within the generalized error tolerance tolfun. All the field-value pairs are optional and can be supplied with any order. If an input is not specified, the default value is used.

q = **cubLattice_g**(f,d,in_param) estimates the integral of f over the d-dimensional region. The answer is given within the generalized error tolerance tolfun.

Input Arguments

- f --- `the integrand whose input should be a matrix nxd where n is the number of data points and d the dimension. By default it is the quadratic function.`
- d --- `dimension of domain on which f is defined. d must be a positive integer 1<=d<=250. By default it is 1.`
- in_param.abstol --- `the absolute error tolerance, abstol>0. By default it is 1e-4.`
- in_param.reltol --- `the relative error tolerance, which should be in [0,1]. Default value is 1e-1.`
- in_param.measure --- `for f(x)*mu(dx), we can define mu(dx) to be the measure of a uniformly distributed random variable in [0,1)^d or normally distributed with covariance matrix I_d. By default it is 'uniform'. The only possible values are 'uniform' or 'normal'.`

```
%  Optional input parameters:|
%
% * in_param.shift --- |the Rank-1 lattices can be shifted to avoid the origin
%  or other particular points. By default we consider a uniformly [0,1)
%  random shift.|
%
% * in_param.mmin --- |the minimum number of points to start is 2^mmin. The
%  cone condition on the Fourier coefficients decay requires a minimum
%  number of points to start. The advice is to consider at least mmin=10.
%  mmin needs to be a positive integer with mmin<=mmax. By default it is 10.|
%
% * in_param.mmax --- |the maximum budget is 2^mmax. By construction of our
%  Lattices generator, mmax is a positive integer such that mmin<=mmax<=26.
%  The default value is 24.|
%
% * in_param.fudge --- |the positive function multiplying the finite
%  sum of Fast Fourier coefficients specified in the cone of functions.
%  For more information about this parameter, refer to the references.
%  By default it is @(x) 5*2.^-x.|
```

```
%
% * in_param.transform --- |the algorithm is defined for continuous periodic funct
%  input function f is not, there are 5 types of transform to periodize it
%  without modifying the result. By default it is Baker. The options:
%    'id' : no transformation. Choice by default.
%    'Baker' : Baker's transform or tent map in each coordinate. Preserving
%              only continuity but simple to compute.
%    'C0' : polynomial transformation only preserving continuity.
%    'C1' : polynomial transformation preserving the first derivative.
%    'C1sin' : Sidi transform with sinus preserving the first derivative.
%              This is in general a better option than 'C1'.|
%
% * in_param.toltype --- |this is the tolerance function. There are two
%  choices, 'max' (chosen by default) which takes
%  max(abstol,reltol*|integral(f)|) and 'comb' which is a linear combination
%  theta*abstol+(1-theta)*reltol*|integral(f)|. Theta is another
%  parameter that can be specified (see below). For pure absolute error,
%  either choose 'max' and set reltol=0 or choose 'comb' and set
%  theta=1.|
%
% * in_param.theta --- |this input is parametrizing the toltype
%  'comb'. Thus, it is only afecting when the toltype
%  chosen is 'comb'. It stablishes the linear combination weight
%  between the absolute and relative tolerances
%  theta*abstol+(1-theta)*reltol*|integral(f)|. Note that for theta=1,
%  we have pure absolute tolerance while for theta=0, we have pure
%  relative tolerance. By default, theta=1.|
%
% *Output Arguments*
%
% * q --- |the estimated value of the integral.|
%
% * out_param.n --- |number of points used when calling cubLattice_g for f.|
%
% * out_param.bound_err --- |predicted bound on the error based on the cone
%  condition. If the function lies in the cone, the real error should be
%  smaller than this predicted error.|
%
% * out_param.time --- |time elapsed in seconds when calling cubLattice_g for f.|
%
% * out_param.exitflag --- |this is a binary vector stating whether
%  warning flags arise. These flags tell about which conditions make the
%  final result certainly not guaranteed. One flag is considered arisen
%  when its value is 1. The following list explains the flags in the
%  respective vector order:|
%

%                    1    If reaching overbudget. It states whether
%                    the max budget is attained without reaching the
%                    guaranteed error tolerance.
%
%                    2   If the function lies outside the cone. In
%                    this case, results are not guaranteed. Note that
%                    this parameter is computed on the transformed
```

```
%                     function, not the input function. For more
%                     information on the transforms, check the input
%                     parameter in_param.transfrom; for information about
%                     the cone definition, check the article mentioned
%                     below.|
%
```

Guarantee

This algorithm computes the integral of real valued functions in dimension d with a prescribed generalized error tolerance. The Fourier coefficients of the integrand are assumed to be absolutely convergent. If the algorithm terminates without warning messages, the output is given with guarantees under the assumption that the integrand lies inside a cone of functions. The guarantee is based on the decay rate of the Fourier coefficients. For more details on how the cone is defined, please refer to the references below.

Examples

Example 1

```
% Estimate the integral with integrand f(x) = x1.*x2 in the interval [0,1)^2:

  f = @(x) prod(x,2); d = 2;
  q = cubLattice_g(f,d,1e-5,1e-1,'uniform','transform','C1sin')
```

Example 2

```
% Estimate the integral with integrand f(x) = x1.^2.*x2.^2.*x3.^2+0.11
% in the interval R^3 where x1, x2 and x3 are normally distributed:

  f = @(x) x(:,1).^2.*x(:,2).^2.*x(:,3).^2+0.11; d = 3;
  q = cubLattice_g(f,d,1e-3,1e-3,'normal','transform','C1sin')
```

Example 3

```
% Estimate the integral with integrand f(x) = exp(-x1^2-x2^2) in the
% interval [0,1)^2:

  f = @(x) exp(-x(:,1).^2-x(:,2).^2); d = 2;
  q = cubLattice_g(f,d,1e-3,1e-1,'uniform','transform','C1')
```

Example 4

```
% Estimate the price of an European call with S0=100, K=100, r=sigma^2/2,
% sigma=0.05 and T=1.

  f = @(x) exp(-0.05^2/2)*max(100*exp(0.05*x)-100,0); d = 1;
  q = cubLattice_g(f,d,1e-4,1e-1,'normal','fudge',@(m) 2.^-(2*m),'transform','C1si
```

Example 5

```
% Estimate the integral with integrand f(x) = 8*x1.*x2.*x3.*x4.*x5 in the interval
% [0,1)^5 with pure absolute error 1e-5.

  f = @(x) 8*prod(x,2); d = 5;
  q = cubLattice_g(f,d,1e-5,0)
```

See Also

References

[1] Lluis Antoni Jimenez Rugama and Fred J. Hickernell: Adaptive Multidimensional Integration Based on Rank-1 Lattices (2014). Submitted for publication: arXiv:1411.1966.

[2] Sou-Cheng T. Choi, Fred J. Hickernell, Yuhan Ding, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

cubSobol_g

```
is a Quasi-Monte Carlo method using Sobol' cubature over the d-dimen-
sional region to integrate within a specified generalized error tol-
erance with guarantees under Walsh-Fourier coefficients cone decay as-
sumptions.
```

Syntax

[q,out_param] = **cubSobol_g**(f,d)

q = **cubSobol_g**(f,d,abstol,reltol,measure,mmin,mmax,fudge,toltype,theta)

q = **cubSobol_g**(f,d,'abstol',abstol,'reltol',reltol,'measure',measure,'mmin',mmin,'mmax',mmax,'fudge',fudge,'toltype',toltype,'the

q = **cubSobol_g**(f,d,in_param)

Description

[q,out_param] = **cubSobol_g**(f,d) estimates the integral of f over the d-dimensional region with an error guaranteed not to be greater than a specific generalized error tolerance, tolfun:=max(abstol,reltol*|integral(f)|). The generalized tolerance function can aslo be cosen as tolfun:=theta*abstol+(1-theta)*reltol*|integral(f)| where theta is another input parameter. Input f is a function handle. f should accept an n x d matrix input, where d is the dimension of the hypercube, and n is the number of points being evaluated simultaneously. The input d is the dimension in which the function f is defined. Given the construction of Sobol', d must be a positive integer with 1<=d<=1111.

q = **cubSobol_g**(f,d,abstol,reltol,measure,mmin,mmax,fudge,toltype,theta) estimates the integral of f over a d-dimensional region. The answer is given within the generalized error tolerance tolfun. All parameters should be input in the order specified above. If an input is not specified, the default value is used. Note that if an input is not specified, the remaining tail cannot be specified either.

q = **cubSobol_g**(f,d,'abstol',abstol,'reltol',reltol,'measure',measure,'mmin',mmin,'mmax',mmax,'fudge',fudge,'toltype',toltype,'the
estimates the integral of f over a d-dimensional region. The answer is given within the generalized error

tolerance tolfun. All the field-value pairs are optional and can be supplied with any order. If an input is not specified, the default value is used.

q = **cubSobol_g**(f,d,in_param) estimates the integral of f over the d-dimensional region. The answer is given within the generalized error tolerance tolfun.

Input Arguments

- f --- the integrand whose input should be a matrix nxd where n is the number of data points and d the dimension. By default it is the quadratic function.
- d --- dimension of domain on which f is defined. d must be a positive integer 1<=d<=1111. By default it is 1.
- in_param.abstol --- the absolute error tolerance, abstol>0. By default it is 1e-4.
- in_param.reltol --- the relative error tolerance, which should be in [0,1]. Default value is 1e-1.
- in_param.measure --- for f(x)*mu(dx), we can define mu(dx) to be the measure of a uniformly distributed random variable in [0,1)^d or normally distributed with covariance matrix I_d. By default it is 'uniform'. The only possible values are 'uniform' or 'normal'.

```
%  Optional input parameters:|
%
% * in_param.mmin --- |the minimum number of points to start is 2^mmin. The
%  cone condition on the Fourier coefficients decay requires a minimum
%  number of points to start. The advice is to consider at least mmin=10.
%  mmin needs to be a positive integer with mmin<=mmax. By default it is 10.|
%
% * in_param.mmax --- |the maximum budget is 2^mmax. By construction of the
%  Sobol' generator, mmax is a positive integer such that mmin<=mmax<=53.
%  The default value is 24.|
%
% * in_param.fudge --- |the positive function multiplying the finite
%  sum of Fast Walsh coefficients specified in the cone of functions.
%  For more information about this parameter, refer to the references.
%  By default it is @(x) 5*2.^-x.|
%
% * in_param.toltype --- |this is the tolerance function. There are two
%  choices, 'max' (chosen by default) which takes
%  max(abstol,reltol*|integral(f)|) and 'comb' which is a linear combination
%  theta*abstol+(1-theta)*reltol*|integral(f)|. Theta is another
%  parameter that can be specified (see below). For pure absolute error,
%  either choose 'max' and set reltol=0 or choose 'comb' and set
%  theta=1.|
%
% * in_param.theta --- |this input is parametrizing the toltype
%  'comb'. Thus, it is only afecting when the toltype
%  chosen is 'comb'. It stablishes the linear combination weight
%  between the absolute and relative tolerances
%  theta*abstol+(1-theta)*reltol*|integral(f)|. Note that for theta=1,
```

```
%  we have pure absolute tolerance while for theta=0, we have pure
%  relative tolerance. By default, theta=1.|
%
% *Output Arguments*
%
% * q --- |the estimated value of the integral.|
%
% * out_param.n --- |number of points used when calling cubSobol_g for f.|
%
% * out_param.pred_err --- |predicted bound on the error based on the cone
%  condition. If the function lies in the cone, the real error should be
%  smaller than this predicted error.|
%
% * out_param.time --- |time elapsed in seconds when calling cubSobol_g for f.|
%
% * out_param.exitflag --- |this is a binary vector stating whether
%  warning flags arise. These flags tell about which conditions make the
%  final result certainly not guaranteed. One flag is considered arisen
%  when its value is 1. The following list explains the flags in the
%  respective vector order:|
%

%                     1    If reaching overbudget. It states whether
%                     the max budget is attained without reaching the
%                     guaranteed error tolerance.
%
%                     2   If the function lies outside the cone. In
%                     this case, results are not guaranteed. Note that
%                     this parameter is computed on the transformed
%                     function, not the input function. For more
%                     information on the transforms, check the input
%                     parameter in_param.transfrom; for information about
%                     the cone definition, check the article mentioned
%                     below.|
%
```

Guarantee

This algorithm computes the integral of real valued functions in dimension d with a prescribed generalized error tolerance. The Walsh-Fourier coefficients of the integrand are assumed to be absolutely convergent. If the algorithm terminates without warning messages, the output is given with guarantees under the assumption that the integrand lies inside a cone of functions. The guarantee is based on the decay rate of the Walsh-Fourier coefficients. For more details on how the cone is defined, please refer to the references below.

Examples

Example 1

```
% Estimate the integral with integrand f(x) = x1.*x2 in the interval [0,1)^2:

  f = @(x) x(:,1).*x(:,2); d = 2;
  q = cubSobol_g(f,d,1e-5,1e-1,'uniform')
```

Example 2

```
% Estimate the integral with integrand f(x) = x1.^2.*x2.^2.*x3.^2+0.11
% in the interval R^3 where x1, x2 and x3 are normally distributed:

  f = @(x) x(:,1).^2.*x(:,2).^2.*x(:,3).^2+0.11; d = 3;
  q = cubSobol_g(f,d,1e-3,1e-3,'normal')
```

Example 3

```
% Estimate the integral with integrand f(x) = exp(-x1^2-x2^2) in the
% interval [0,1)^2:

  f = @(x) exp(-x(:,1).^2-x(:,2).^2); d = 2;
  q = cubSobol_g(f,d,1e-3,1e-1,'uniform')
```

Example 4

```
% Estimate the price of an European call with S0=100, K=100, r=sigma^2/2,
% sigma=0.05 and T=1.

  f = @(x) exp(-0.05^2/2)*max(100*exp(0.05*x)-100,0); d = 1;
  q = cubSobol_g(f,d,1e-4,1e-1,'normal','fudge',@(m) 2.^-(2*m))
```

Example 5

```
% Estimate the integral with integrand f(x) = 8*x1.*x2.*x3.*x4.*x5 in the interval
% [0,1)^5 with pure absolute error 1e-5.

  f = @(x) 8*prod(x,2); d = 5;
  q = cubSobol_g(f,d,1e-5,0)
```

See Also

References

[1] Fred J. Hickernell and Lluis Antoni Jimenez Rugama: Reliable adaptive cubature using digital sequences (2014). Submitted for publication: arXiv:1410.8615.

[2] Sou-Cheng T. Choi, Fred J. Hickernell, Yuhan Ding, Lan Jiang, Lluis Antoni Jimenez Rugama, Xin Tong, Yizhi Zhang and Xuan Zhou, "GAIL: Guaranteed Automatic Integration Library (Version 2.1)" [MATLAB Software], 2015. Available from http://code.google.com/p/gail/

If you find GAIL helpful in your work, please support us by citing the above paper and software.

Installation Instructions

1. Unzip the contents of the zip file to a directory and maintain the existing directory and subdirectory structure. (Please note: If you install into the **toolbox** subdirectory of the MATLAB program hierarchy, you will need to click the button "Update toolbox path cache" from the File/Preferences... dialog in MATLAB.)

2. In MATLAB, add the GAIL directory to your path. This can be done by running **GAIL_Install.m**. Alternatively, this can be done by selecting **File/Set Path...** from the main or Command window menus, or with the command **pathtool**. We recommend that you select the "Save" button on this dialog so that GAIL is on the path automatically in future MATLAB sessions.

3. To check if you have installed GAIL successfully, type **help funappx_g** to see if its documentation shows up.

Alternatively, you could do this:

1. Download DownloadInstallGail_2_1.m and put it where you want GAIL to be installed.

2. Execute it in MATLAB.

To uninstall or reinstall GAIL, execute **GAIL_Uninstall**. To reinstall GAIL, execute **GAIL_Install**.

Published with MATLAB® R2014a