# MATLAB-PGSL Version 1.0

User Manual and Source Code Documentation

July 9, 2010

# 1. Introduction

The MATLAB version of PGSL (MATLAB-PGSL) was developed by porting the C version of PGSL 4. Since the Matlab interface to C programs run very slowly due to the overheads involved in loading DLLs (Dynamic Link Libraries), the PGSL code was translated into native Matlab syntax. An important consideration in developing the Matlab code was that it should be fully compatible with GNU Octave. This is to ensure that people using this freeware are able to run their optimization programs without performing any additional steps for portability.

The MATLAB version is not a line-to-line translation of the C code due to many reasons. Matlab is interpreted and does not run as fast as C code. Therefore, some parts of the code had to be modified to improve the speed (for example, some operations on individual array elements were replaced with array functions). However, no claim is made that the code is fully optimized for Matlab.

# 2. Installation

In order to install MATLAB-PGSL, simply unzip the zip file, MATLAB-PGSL.zip to an appropriate folder. In Matlab or GNU Octave change the working directory to the PGSL folder using the command *cd* and you are ready to run PGSL.

# 3. Using MATLAB-PGSL

Using MATLAB-PGSL is very similar to the C version of PGSL 4. There are three key steps involved:

- Define the optimization (search) variables
- Specify the PGSL parameters
- Define the Objective function

Defining the optimization variables involves specifying their lower and upper bounds as well as the precision. These are specified as arguments to the function *ProblemSetup_create*.

The PGSL parameters can be assigned default values by simply specifying the maximum number of evaluations of the objective function. In addition, you need to specify the threshold, which is the minimum value of the objective function below which optimization terminates. These parameters are also specified as arguments to the function *ProblemSetup_create*.

This function is called using the following syntax:

```
setup = ProblemSetup_create( bmin, bmax, prec, numeval,
threshold);
```

where,

- *bmin* is an array containing the lower bound for all the optimization variables
- *bmax* is an array containing the upper bound for all the optimization variables
- *prec* is an array containing the precision of all the optimization variables
- *numeval* is an integer that specifies the maximum number of evaluations of the objective function
- *threshold* is the minimum value of the objective function below which optimization terminates

The above function returns a structure of type *ProblemSetup*. The structure is described in more detail in Section 4.1.1. The structure represents the optimization problem and contains all the information that is needed to run the optimization.

The objective function is defined as a Matlab function with two arguments. The first argument is the *ProblemSetup* structure representing the optimization problem. The second argument is an array containing the values of optimization variables proposed by PGSL. This array represents a potential solution to be evaluated by the objective function. The objective function returns a real number which is the evaluation of the objective function.

A sample objective function is shown below.

```
function ret  = test_parabola_objective (setup, x)

      x1 = x(1);
      x2 = x(2);

      ret = x1*x1 + x2*x2;

end
```

In the above example, `test_parabola_objective` is the name of the objective function. The second argument *x* contains the optimization variables which are assigned to local variables *x1* and *x2*. The return value of the objective function *ret* is the sum of the squares of the values of the optimization variables.

The name of the objective function is arbitrary and is left to the preference of the user. The name of the function that is to be optimized is specified by assigning the function pointer to the attribute *costFunction* of the structure *ProblemSetup*. This is illustrated using the following code:

```
setup = ProblemSetup_create( bmin, bmax, prec, numeval, threshold);
setup.costFunction = @test_parabola_objective;
```

Finally, the optimizer is invoked by calling the function *PGSL_findMinimum*. The function takes a single argument which is the structure *ProblemSetup* containing the representation of the optimization problem. This function implements the PGSL algorithm and calls the user defined objective function repeatedly during the course of searching for the minimum. The function returns when either of the following conditions is met:

- The maximum number of evaluations of the objective function is reached
- The objective function value has become less than the threshold

When the function returns, the best solution is stored in the attribute *minimumPoint* of the structure *ProblemSetup*.This attribute is of type structure *Point*.The structure *Point* contains the following attributes:

- x – an array of real numbers which contains the values of optimization variables
- y – a real number that stores the evaluation of the point

The complete code for solving a sample optimization problem is shown below:

```
function ret  = test_parabola ()

      bmin = [ -100 -100];
      bmax = [  100  100];
      prec = [ 0.001 0.001];

      % The number of evaluations
      numeval = 200 ;

      % The threshold of the objective function
      threshold = -1;

      setup = ProblemSetup_create( bmin, bmax, prec, numeval,
threshold);
      setup.costFunction = @test_parabola_objective;

      setup = PGSL_findMinimum(setup);

      % Print the results
      fprintf(1, ' Minimum found %f\n', setup.minimumPoint.y);
      fprintf(1, ' Variable values\n');
      fprintf(1, 'x1 \t %f \n', setup.minimumPoint.x(1) );
      fprintf(1, 'x2 \t %f \n', setup.minimumPoint.x(2) );

      ret = setup;

end
```

## 3.1 Testing PGSL

There are two test functions provided. These are

- test_parabola
- test_F8

### 3.1.1 test_parabola

This function tests a simple quadratic problem in two variables. The code can be found in test_parabola.m. The lower and upper bounds of all the variables are -100 and +100 respectively. The precision of variables is set to 0.001. The maximum number of evaluations of the objective function is set to 400. These parameters can be changed in the file test_parabola.m and the function can be executed by running the command test_parabola in Matlab.

You will find an output similar to the following

```
 Num eval 402
 Minimum found 0.000023
 Variable values
x1       0.004831
x2       0.000193
```

The actual results will vary from one run to the next. More accurate values can be obtained by increasing the maximum number of evaluations and changing the precision. The theoretical minimum is 0 and the values of x1 and x2 are 0.

### 3.1.1 test_F8

This tests the Griewank's function. This function needs to be called with one argument, that is, the number of variables. For example, like this:

```
test_F8(2)
```

For larger problem sizes, it takes quite a lot of time to complete since the number of evaluations of the objective function has been set to 10000 * numvars. Since Griewank's function has many local minima it takes quite a number of iterations to converge.

The objective function is defined in the file test_F8_objective.m.

# 4. Source code documentation

PGSL code is organized into several functions which are stored in .m files in the PGSL folder. The following naming conventions are used:

- All the functions ending with _create are the equivalents of constructors in object oriented programming. These functions return a structure after initializing the attributes
- All the functions starting with the name of a structure followed by an underscore are the equivalent of member functions (methods) in object oriented programming. For example, the function PAxis_indexOf represents the method indexOf in the structure PAxis.

## 4.1 Data Structure

The structures that are used in MATLAB-PGSL are described in this Section.

### 4.1.1 Structure ProblemSetup

The structure represents an optimization problem. The attributes of this structure are summarized in the following table.

| Attribute | Data type | Description |
|---|---|---|
| numvars | Integer | The number of variables in the optimization problem |
| maxNumEvaluations | Integer | Maximum number of evaluations of the objective function |
| threshold | Real | The threshold of the objective function below which search terminates |
| lowerBounds | Array of real | lower bounds of variables |
| upperBounds | Array of real | upper bounds of variables |
| axes | Array of structure PAxis | The axis representing each variable |
| NS | Integer | PGSL parameter NS, number of iterations in the sampling cycle |
| NPUC | Integer | PGSL parameter NPUC, number of iterations in the probability updating cycle |
| NFC | Integer | PGSL parameter NFC, number of iterations in the focusing cycle |
| NSDC | Integer | PGSL parameter NSDC, number of iterations in the subdomain cycle |
| useStartPoint | Integer | The user should set the following variable to 1, if an initial starting point is to be used |
| numEvaluations | Integer | The number of evaluations of the objective function so far |
| iSDC | Integer | Number of iterations in the subdomain cycle completed so far |
| iS | Integer | Number of iterations in the current sampling cycle completed so far |
| restart | Integer | An internal variable. This variable is set to non-zero whenever there is a restart after convergence |
| minimumPoint | Structure Point | The best solution found so far |
| backupMinimumPoint | Structure Point | An internal variable. This variable stores the previous best minimum point when a restart occurs after convergence |
| costFunction | Function pointer | The pointer (handle) to the objective function. User should assign his objective function to this variable |

### 4.1.2 Structure PAxis

This structure represents an axis or a search dimension corresponding to a variable. The attributes of this structure are summarized in the following table.

| Attribute | Data type | Description |
|---|---|---|
| min | Real | Lower bound for the variable |
| max | Real | Upper bound for the variable |
| precision | Real | Precision of the variable |
| intervals | Array of real | A set of values of the variable at which probabilities are computed for storing the PDF as a histogram |
| prob | Array of real | Probabilities corresponding to the intervals |
| cdf | Array of Real | The cumulative distribution function corresponding to the intervals |

### 4.1.2 Structure Point

This structure represents a solution point. The attributes of this structure are summarized in the following table.

| Attribute | Data type | Description |
|---|---|---|
| x | Array of real | The values of the optimization variables |
| y | real | The evaluation of the point |

## 4.2 Functions

### 4.2.1 PAxis_calculateCDF

Function prototype:
function ret  = PAxis_calculateCDF (axis)

This function computes the CDF of the PDF of a variable.  The argument to the function is a variable of type structure PAxis. Returns an array of real which is the CDF.

The function sums the probabilities and normalizes by dividing by the total. This is an internal function and may not be called directly by the user.

### 4.2.2 PAxis_create

Function prototype:
function ret  = PAxis_create (min, max, precision)

This function is the constructor for creating the PAxis structure. The first argument is the lower bound of the variable. The second argument is the upper bound of the variable. The return value is a variable of type structure PAxis.

The function initializes the variables and creates a PDF which is uniformly distributed between the lower and upper bounds.

### 4.2.3 PAxis_indexOf

Function prototype:
function ret  = PAxis_indexOf (axis, x)

This function returns the index of the interval in which a number lies. Argument axis is a variable of type structure PAxis. Argument x is the value of the variable that is to be located. The return value is the index of the interval in which the number lies. The index starts from 1.

This is an internal function and may not be called directly by the user.

### 4.2.4 PAxis_valueForCDF

Function prototype:
function ret  = PAxis_valueForCDF (axis, t)

This function returns the value of the variable for a specified value of CDF. The argument axis is of type structure PAxis. The argument t is the value of the CDF for the which the value of the variable needs to be computed. The return value is a real number within the lower and upper bounds of the variable.

This is an internal function and may not be called directly by the user. The function locates the interval in which the CDF value lies and then interpolates between the bounds of the interval.

### 4.2.5 PGSL_computeWeightedAverage

Function prototype:
function ret = PGSL_computeWeightedAverage (axis, points)

This function returns the weighted average of the value of a variable for the best five solution points obtained in previous focusing cycles. The argument axis is an integer representing the index of the variable. The argument points is an array of structure Point. The array contains the previous five points.

The return value of the function is a real number which is the weighted average of the values of the variable.

This is an internal function and may not be called directly by the user. The function is called by the PGSL routine PGSL_doSubdomainCycle.

### 4.2.6 PGSL_divideInterval

Function prototype:
function ret = PGSL_divideInterval (axis, index, pvalue, mindx, amin, amax, probBestInterval)

The function updates the PDF of a variable by sub-dividing the best interval as described in the PGSL algorithm.

The argument axis is of type structure PAxis. The argument index is the index of the best interval. The argument pvalue is the pivot value about which division takes place (the best value of the variable). The argument mindx is the minimum interval width. The arguments amin and amax are the upper and lower bounds of the variable. The argument probBestInterval is the probability to be assigned to the best interval.

The return value of the function is of type structure PAxis. The return value is a new axis for the variable with updated PDF.

This is an internal function and may not be called directly by the user.

### 4.2.7 PGSL_doSubdomainCycle

Function prototype:
function ret  = PGSL_doSubdomainCycle (setup)

The function performs the subdomain cycle as described in the PGSL algorithm. The argument setup is of type structure ProblemSetup. The function returns the updated ProblemSetup structure.

This is an internal function and may not be called directly by the user.

### 4.2.8 PGSL_doFocusingCycle

Function prototype:
function ret  = PGSL_doFocusingCycle (setup, minBound, maxBound, startPoint)

The function performs the focusing cycle as described in the PGSL algorithm. The argument setup of type structure ProblemSetup. The arguments minBound and maxBound are arrays containing lower and upper bounds of variables. The argument startPoint is of type structure Point and is the starting point for the focusing cycle. The starting point is randomly generated except when it is specified by the user.

The return value of the function is of type structure ProblemSetup which is the updated problem setup.

This is an internal function and may not be called directly by the user.

### 4.2.9 PGSL_doRestart

Function prototype:
function  ret  = PGSL_doRestart (setup, minimumPoint)

The function re-initialises all the axes after the focusing cycles has converged. The argument setup is of type structure ProblemSetup. The argument minimumPoint is of type Point. It is the best point found so far. The function returns the updated structure ProblemSetup.

This is an internal function and may not be called directly by the user. The function stores the minimumPoint in the variable setup.backupMinimumPoint. The previous point is replaced if the new point is better. Then all the axes are reset to start with the original uniform PDF.

### 4.2.10 PGSL_ findMinimum

Function prototype:
function ret  = PGSL_findMinimum (setup)

This function is the main entry point to PGSL and is called by the user. This should be called after creating the ProblemSetup structure, setting all the parameters, defining the cost function, etc. The argument setup is of type structure ProblemSetup.

The function executes the PGSL algorithm, locates the minimum point and returns the updated ProblemSetup structure. This structure contains the minimum point found in the search.

### 4.2.11 PGSL_generatePoint

Function prototype:
function ret  = PGSL_generatePoint (setup, point)

The function generates a random solution point according to the current PDF. The argument setup is of type structure ProblemSetup. The argument point is of type structure Point.

The return value of the function is of type structure Point. It has randomly generated values for all the variables. The objective function is called to compute the y value of the point.

This is an internal function and may not be called directly by the user.

### 4.2.12 PGSL_getIntervals

Function prototype:
function ret  = PGSL_getIntervals (axes, pt)

The function is used by PGSL to locate the best intervals containing the values of variables in a newly generated point. The argument axes is an  array containing elements of type structure PAxis. The argument  pt is of type structure Point.

The function returns an array of integers. Each integer is the index of the interval corresponding to the value in the point pt

This is an internal function and may not be called directly by the user.

### 4.2.13 PGSL_hasConverged

Function prototype:
function ret  = PGSL_hasConverged (axes)

The function is called by PGSL to determine whether all the axes have converged during the subdomain cycle. It checks for convergence by examining the minimum and maximum of each axis. The function returns either 1 or 0. 1 denotes convergence.

This is an internal function and may not be called directly by the user.

### 4.2.14 PGSL_narrowDown

Function prototype:
function ret  = PGSL_narrowDown (setup, points, numpoints)

The function reduces the subdomain size as described in the PGSL algorithm.
The argument setup is of type structure  ProblemSetup. The argument points is an array containing elements of type structure Point. It contains up to 5 previous best points. The argument numpoints is an integer storing the number of previous best points saved in the array points.

The function returns an array containing the updated axes of all the variables.

This is an internal function and may not be called directly by the user.

### 4.2.15 PGSL_resetAxes

Function prototype:
function ret  = PGSL_resetAxes (setup)

The function resets the bounds of every variable according to the original values after a convergence. The argument setup is of type structure  ProblemSetup. The function returns the new array of updated axes.

This is an internal function and may not be called directly by the user.

### 4.2.16 PGSL_resetAxes

Function prototype:
function ret  = PGSL_sortBestPoints (numPoints, points, newpoint)

The function inserts a new point in the sorted array of 5 best points maintained by PGSL to avoid premature convergence. The argument numPoints is an integer storing the number of points already in the array points. The argument points is an array of structure Point. It contains the existing best points. The argument newpoint is of type structure Point. It contains a new point to be added to the array.

The function returns the updated array of points

The function inserts the new point in the sorted order in the array.

This is an internal function and may not be called directly by the user.

### 4.2.17 PGSL_updateIntervals

Function prototype:
function ret = PGSL_updateIntervals (setup, iFC, prevBestPoint, minBound, maxBound)

The function updates the intervals of the PDF, during the focusing cycle of PGSL. The argument setup is of type structure ProblemSetup. The argument iFC, is current iteration in the focusing cycle. if it is zero, the previous point is dummy and is not used to update the intervals. The argument preBestPoint is of type Structure Point, the best point found in the previous cycle. The arguments minBound, maxBound are the lower and upper bounds of the variable.

The function returns the array containing the updated structure PAxis

This is an internal function and may not be called directly by the user.

### 4.2.18 PGSL_updateProbability

Function prototype:
function ret = PGSL_updateProbability (setup, bestIntervals, negpt)

The function updates the intervals of the PDF, during the probability updating cycle of PGSL. The argument setup is of type structure ProblemSetup. The argument bestIntervals is an array of integers storing the index of intervals containing the best values of variables The argument negpt is of type structure Point. It represents a negative point - a point that is inferior to the current best

The function returns the array containing the updated structure PAxis

This is an internal function and may not be called directly by the user.

### 4.2.19 ProblemSetup_updateMinimum

Function prototype:
function ret  = ProblemSetup_updateMinimum (setup, newpoint)

This function updates the best minimum point found so far. The argument setup is of type structure ProblemSetup. The argument newpoint is of type structure Point. It is a new point that has been evaluated already.

The function returns the minimum point found so far. Before calling this function, the variable setup.numEvaluations should be incremented.

This is an internal function and may not be called directly by the user.