Model-Assisted Pattern Search(MAPS) Manual for Release 0.4a

Christopher Siefert*
College of William and Mary

July, 2000

Abstract

This is the manual for Release 0.4a of the C++ implementation of Model-Assisted Pattern Search. This document will hopefully get you on your feet when it comes to executing this algorithm. It should explain all you need to know about how to get this program to compile, link and run successfully. The HTML version of this document was created using LaTeX2HTML. Since it is not perfect, there will be mistakes in the html file. Read the postscript if you think the html is wrong. If you have any questions at all, feel free to email the author for a clarification.

This work is was supported in part by a Batten Scholarship from the College of William and Mary and by grants from the National Science Foundation (CCR-9734044 and EIA-9712718).

*Department of Computer Science, University of Illinois (`cmsief@acm.org`)

Contents

1 The Basics

1.1 Finding the MAPS algorithm

You can find a current copy of the MAPS algorithm and creator-maintained dependent files at the web site *http://www.cs.wm.edu/~va/software/maps/*. Be sure to download the gzipped tarball with all the necessary files.

There are basically five categories of files that are included in the tarball. The first category contains the maps implementation files (which are discussed in Section 4). These contain the implementation for things unique to MAPS. Secondly, there are the Pattern Search files. They are versions of Dolan and Shepherd's Pattern Search implementation. These files are different from the ones found in the Pattern Search section of Virginia Torczon's software library, so don't expect those versions to compile with MAPS. Next we have the C. Siefert's Matrix and Vector class implementations, which are derived from R. Pozo's Template Numerical Toolkit. These are stock versions and are used 'as-is'. They can be found at *http://www.cs.wm.edu/~va/software/Vectors/*. The fourth category includes S. Park's Lehmer pseudo-random number generation libraries and can be found at *http://www.cs.wm.edu/~va/software/park/*. The final category contains a few auxiliary utilities for maps which are also found on the maps web site; these are detailed in the appendices.

1.2 Finding other dependencies

In order to perform the statistical techniques necessary to manipulate the approximation to the objective function, MAPS uses CLAPACK, available from netlib at *http://www.netlib.org/clapack/*. If you already have CLAPACK and the CBLAS on your machine, then link MAPS to those. If not, download `netlibfiles.tgz` from the MAPS software page. If you have a copy of A. Padula's krigifier, and you downloaded `netlibfiles.tgz` with it, then you need not download the files over again. In this case, make sure that the library files are in the MAPS directory or in the linker path.

1.3 System Requirements

MAPS is designed to use the UNIX fork-exec paradigm for execution of your objective function. So you will need some kind of UNIX or POSIX-compliant OS (Linux works fine) to run this. If you're using a another operating system, you'll have to find some kind of GNU port, or else reimplement the function EvalF in `maps.cc` to work with your operating system.

You'll also need a C++ compiler, that includes support for templates. This software was developed using egcs-g++ version 2.95.2.

2 Quick Install and Test Run

2.1 Installation

Unzip and untar the file. You can do this with the command:

`tar zxvf maps.tgz`

If your version of tar doesn't support the 'z' option (gtar does), then do the following:

`gunzip maps.tgz` (This will create maps.tar)

`tar xvf maps.tar`

If you downloaded `netlibfiles.tgz`, then install these libraries by running:

`make install`

This assumes that your version of tar supports the 'z' option. If not modify the Makefile to `gunzip` and then `tar xvf`. The next step is to compile and link the MAPS software. Do this by running:

`make`

For the trial run, you'll need an objective function. An implementation of a Shekel family function has been included. It should be automatically compiled by the Makefile.

If everything has succeeded up to this point, then you're all set for the example. If there was a problem, please refer to the Section 5.

2.2 Trial Run

There are two ways to go through a trial run. If you just want to verify MAPS' functionality, the fast method should be sufficient. For this, type:

```
cat sample.dat | ./maps
```

The second method involves answering a series of questions. They go as follows:

```
Welcome to the MAPS Test Program!
Please provide the following information:
```

The prompts for the MAPS algorithm are listed below in order. Also listed are responses to those prompts for a trial run of MAPS.

Prompt	Response
Random Seed=	123456789
Number of Runs of MAPS=	1
Number of Variables=	4
Total Budget[V]=	20
Initial Design Budget[N] (less than total)=	10
Initial Mesh Size[Delta]=	0.01
Correlation Family(default=-1,EI=0,GI=1,EP=2,GP=3,CI=4)=	-1
Using default correlation...	
Pattern Search(default=-1,Compass=0, MultiCompass=1)=	-1
Using default pattern search...	
Assumed Trend (default=-1, Constant=0, Quadratic=1)=	-1
Using default trend...	
Weighting Scheme(default=-1,WM=0,WTV=1,WDM=2,WDG=3,WD=4)=	-1
Using default weights...	
Program name of function to optimize=	./shekel
Lower bound (space separated)=	0 0 0 0
Upper bound (space separated)=	10 10 10 10

Regardless of which method you used, the major section of the results should look something like this:

```
**********
fn=./shekel V=20 N=10
Random seed=123456789
******MAPS ITERATION HISTORY******
# for parameters V=20 N=10 P=4 delta=0.01
# on function: ./shekel


NUM          1          2          3          4        VAL
   1      0.11       3.14       5.51       5.09 -0.1371344
   2      2.14       2.26       1.85       1.31 -0.34895367
   3       7.5       6.63       3.94       0.67 -0.10299594
   4      5.49       0.76       6.88        4.5 -0.12256296
   5      9.65       4.03       4.15       8.32 -0.11180869
   6      3.65       5.32       9.17       7.37 -0.14660693
   7      8.67       9.78       0.72       6.78 -0.078442401
   8      6.07       1.43       8.51       9.36 -0.082282933
   9      4.81       8.22       7.62       3.94 -0.17825804
  10      1.08        7.3       2.31       2.03 -0.12213042
  11      2.15       2.26       1.85       1.31 -0.34759741
  12      2.03       2.26       1.85       1.31 -0.36410903
  13      1.48       2.26       1.85       1.31 -0.43621371
  14      0.96       2.26       1.85       1.31 -0.45991384
  15      1.05       2.26       1.85       1.31 -0.46101096
  16      1.04       2.26       1.85       1.31 -0.46100608
  17      1.06       2.26       1.85       1.31 -0.46098669
  18      1.05       2.25       1.85       1.31 -0.46463045
  19      1.05       2.24       1.85       1.31 -0.4682923
  20      1.05       1.13       1.85       1.31 -1.0288096

Best Point found is #20, value=-1.0288096
**********************************
```

Your results need not be the same as those listed above. Indeed, due to differences in architectures, operating systems and compilers, it is unlikely that your results will be exactly like those listed above. At this point, you're set to run MAPS on your own problems. Refer to section 3.1 for information on how to use your own objective functions with MAPS.

3 Customizing MAPS

3.1 Choosing the Function to Optimize

MAPS assumes that the function you wish to optimize is represented by a *stand-alone* executable file. This means you can write your function in the language of your choice, so long as you can create some sort of executable file with it. Your program may require on-disk datafiles — that is fine, so long as your program takes care of file i/o without requiring user input. MAPS will work even if you have to “wrap” your function in a shell script, so long as you make the script executable.

The term “fork-exec paradigm” refers to the way UNIX and UNIX-like operating systems handle spawning another process. When MAPS wishes to evaluate a function, it *forks* or spawns off child process, and *execs* or runs your program as that process. MAPS then sends the program the point it wishes to evaluate the function at on its *standard input.* Your program should return the function value at that point on its *standard output.* If your program outputs anything else to its standard output, MAPS will probably crash. This is because of the method by which MAPS is communicating with your program. Consider yourself warned. If the function evaluation succeeds at a given point, MAPS expects your program to return zero. Likewise if evaluation fails, MAPS expects a non-zero return value. Included in Appendix C is a sample C++ objective function, that fits these specifications.

When specifying the function name for MAPS, it is important to make sure that either (1) the program is in a directory in your path or (2) that you specify a full or relative (`./` etc.) path name. If you don't, chances are MAPS won't find your program.

One minor flaw of this current implementation is that it doesn't directly support objective functions with command-line parameters. If your program does require parameters, it is fairly trivial to write a shell script to “wrapper” it, so that MAPS can function properly. Appendix D has a sample script for this precise purpose.

3.2 The Basic Options

The program you see running when you run `maps` is actually the `testMPG` test program for MAPS (see Appendix B for more information). Customizing the test program itself is dealt with in Section 3.4. For now we'll assume that your running the program 'as-is'. Prompts for input will come up in the following order. Items preceded by three asterisks (***) are options depending on your previous choices.

`Random Seed=`

You'll need to enter a seed for the random number generator, in order to generate the initial design for the problem. Any fairly large seed should work fine — just remember what the seed is, in case you need to reproduce the results.

`Number of Runs of MAPS=`

If you're just trying to run the problem through MAPS to get an answer, enter one here. Otherwise, if you want to see the results of more than one MAPS run on the problem (for algorithm analysis, perhaps), enter the appropriate higher value here.

`Number of Variables=`

This is the number of variables your optimization problem has.

`Total Budget[V]=`

This is the number of function evaluations you're willing to allot MAPS to optimize the function.

`Initial Design Budget[N] (less than total)=`

This is the number of function values you want to allocate the initial design for MAPS. If you're not sure, use about a third to a half of the total budget.

`Initial Mesh Size[Delta]=`

This is the resolution to use for the initial mesh for MAPS. The best thing to do is to keep it about one-hundredth the size of the smallest variable space, or smaller.

`Correlation Family(default=-1,EI=0,GI=1,EP=2,GP=3,CI=4)=`

This option lets you choose which correlation family to use for the kriging in the MAPS algorithm. If you don't know (or have no preference), just use the default (which is Gaussian Isotropic). The other options stand for Exponential Isotropic(EI), Gaussian Isotropic(GI), Exponential Product(EP), Gaussian Product(GP) and Cubic Isotropic(CI). These correlations are *not* created equal, so change from the default only if you really know what you are doing. Section 4.7 lists the formulas for these functions.

```
Pattern Search(default=-1,Compass=0,MultiCompass=1)=
***MultiCompass Params(#searches,maxcalls,stream)=
```

Since MAPS requires that the search criterion be optimized, one of two types of pattern search are available for that purpose. The program defaults to a compass search. If you choose 'MultiCompass,' which is a multi-started compass search, you will be given the option to choose three parameters for it. The first parameter is the number of searches to run, the second is the number of calls to let the compass search run for (-1 makes them run until convergence, which is the best option to use), and the last is the random stream. Be sure to use a number between 10 and 255 for the stream.

`Assumed Trend (default=-1, Constant=0, Quadratic=1)=`

At the moment, MAPS attempts to model the data from the objective function using either a constant (default) or a quadratic trend. Constant trend estimation should be suitable for most applications, but you might want to use the quadratic trend in certain cases (ie. if your objective function is vaguely quadratic).

```
Weighting Scheme(default=-1,WM=0,WTV=1,WDM=2,WDG=3,WD=4)=-1
***Weighting Params (w_0,m)=
***Weighting Params (w_0,w_i,i%)=
***Weighting Params (w_0,dm)=
***Weighting Params (w_0,t%,m_b,m_g)=
***Weighting Params (w_0,m_b,m_g)=
```

The weighting scheme lets you choose the balance to strike between the accuracy of the approximation to your function and the desire to explore more fully the function space. The default setting uses a dynamic weighting scheme (Weight Dynamic) that compares the function value at the last evaluated point to its expected function value. If value compare favorably, then the weight on the approximation is increased. Likewise, if the values compare unfavorably, then the weight on the design criterion is increased. The default setting starts with a weight of $w_0 = 2.0$, and multiplies it by $m_b = 1.25$ if the last value was "bad" and $m_g = 0.75$ if the last value was good. There are four other weighting schemes included in the current release of MAPS — Weight Multiple(WM), Weight Target Value(WTV), Weight Delta Multiple(WDM) and Weight Dynamic Guess(WDG). The first weighting scheme lets you specify the initial weight w_0, and a multiple m which is what the previous iteration's weight is multiplied by to yield the next iteration's weight. Weight Target Value lets you specify the initial weight w_0, the weight of "insignificance," w_i, and the percent of the non-design function evaluations at which to reach the weight of insignificance. The Weight Delta Multiple scheme takes two parameters, the initial weight w_0 and the multiple dm to multiply the weight by after a grid refinement. The Weight Dynamic Guess scheme is non-intuitive. You specify w_0, a tolerance t, and two multiples m_b for the "bad" multiplier, and m_g for the "good" multiplier. If the last function value was within t% of the explored function value range of the approximation's expected function value at that point, then the current weight is multiplied by the "good" multiplier, else it is multiplied by the "bad" multiplier.

`Program name of function to optimize=`

The FULL PATH NAME of the program you wish to optimize. This means that you either have to include the entire path (peg. `/home/bob/func_26`) or qualify the program name with a `./` signifying the current directory (peg. `./func_26`).

`Lower bound (space separated)=`
`Upper bound (space separated)=`

These are the bounds of your problem, listed lower bounds first, each number separated by spaces and NOT by commas.

3.3 Custom Datafiles for testMPG

Included with this distribution of MAPS is a file called `sample.dat` (mentioned in Section 2.2). This file can be customized to fit your needs. While you must stick to the parameter order discussed in Section 2.2, the format is quite flexible. Any number of parameters can be placed on a line, and comments may be added simply by preceding the comment with a `#` character.

3.4 Writing a Test Program

The Model-Assisted Pattern Search algorithm is encapsulated into a C++ class. This means that in order to use MAPS, you have to write a program, to instantiate this class, and run its functions. A sample version of such a program (testMPG.cc) is included with this distribution. This program should allow customization of about 90% of MAPS' features without modification of the code. However, feel free to customize this to suit your needs. More information on testMPG.cc is included in Appendix B.

3.4.1 Introduction for C and Pascal Programmers

Though MAPS is written in C++, there is no reason why a C or Pascal programmer couldn't get MAPS to work. MAPS is implemented as a collection of "classes," which are analogous to C structs and Pascal records, except for the fact that they have attached functions. In fact, like references to attached data members of a struct or record, class member functions are referred to with the same "dot" notation. For example, if one has created a maps class object in this manner:

```
maps myobject;
```

Then one would refer to the RunSearch member function of the maps class as such:

```
myobject.RunSearch();
```

3.4.2 The Basics of a MAPS Test Program

In order to write a simplistic test program, the easiest thing to do would be to create a maps object using the maps(long V1, long P1, const Bounds &B1, char*fn) constructor, then call RunSearch(), and finally call RetrieveBestSeen(mp_vector* bpoint, double* bvalue). The following example, which demonstrates this, assumes that the program has already stored the evaluation budget in Budget, the dimension of the space in Dim, the bounds of the space in TheBounds, and the name of the executable is stored in the string pointed to by myfunc.

```
mp_vector best_point(Dim);          /*Declares a Vector of size Dim*/
double best_value;                  /*Best value returned*/

maps mysearch(Budget,Dim,TheBounds,myfunc);
/*Declares the maps class object*/

mysearch.RunSearch();
/*Runs the Search*/

mysearch.RetrieveBestSeen(&best_point, &best_value);
/*Gets the best point, and best value seen.
  The '&' in this passes a pointer to the best_point
  and best_value objects.
*/
```

As a note, it is important to include the correct header files in your test program using the C/C++ #include compiler directive. The only files you need to include in a basic test program are maps_general.h and maps.h. If you are using a more advanced testing scheme, be sure to include the appropriate headers.

3.4.3 Advanced Testing — Choosing the Specifics

A good example of an advanced test program can be found in Appendix B. What follows here is a brief explanation of how to tweak the various parameters of MAPS.

For the most part, in order to reset a particular parameter of MAPS, one creates an object with the desired characteristics, and then "feeds" it to the maps class object using the appropriate reset functions. This sort of model holds for making the appropriate settings for the Search Criterion, and other classes with class objects as members. For the following example, assume that we want to change from the default grid to another one that uses a special set of settings. First, we would create the new grid (a PSGrid class object) and set it appropriately, and then call the ResetGrid function of the maps class object. The following example illustrates this, assuming that a maps class object called `mysearch` has been declared, and that we wish to change its grid.

```
PSGrid newgrid;                 /*Declare the new grid*/

/*Here is where it is set with the special settings*/

mysearch.ResetGrid(newgrid);    /*Changes the grid to the new grid*/
```

One particular change that one might desire to make, though it may not be immediately obvious how to do it, is changing the correlation function used for the parameter estimation for the kriging. In order to do this, one must create a special KrigApprox class object, "feed" that to a special SearchCriterion class object, and then "feed" that in turn to the maps class object. This is done in `testMPG.cc` (in Appendix B), and the following code is taken from it, assuming that `DIM` is the dimension of the space, `COR` is the number of the desired correlation function, and `M` is the maps class object.

```
  maps M;                             /*Create the maps Object*/

  KrigApprox KA(DIM);                 /*Create a KrigApprox Object*/
  KA.ResetCorrelationFamily(COR);     /*Set the Correlation Family*/
  SearchCriterion SC(KA,b);           /*Create the special Search Criterion*/
  M.ResetSearchCriterion(SC);         /*Modify MAPS*/
```

3.5 MAPS as a Callable Subroutine (testClassic)

In addition to the `testMPG` interface, the MAPS software also comes with a second interface called `testClassic`. This interface allows you to call MAPS from another program. The included file `sample2.cc` contains simple example of using this interface for MAPS.

To use the `testClassic` interface, you create an objective function:

```
int mfcn(long DIM, double * x, double * f)
```

For this function, `DIM` is the dimension of the problem, `x` is a pointer to the evaluation site, and `f` is a pointer to the function value to be returned.

Then, in your program you have to give MAPS bounding information by calling:

```
void setbound(long DIM, double * lower, double * upper);
```

After setting the bounds, call MAPS using the function:

```
bool msearch(long DIM, double * initial_guess, long max_fevals, double
tol);
```

If don't have a good guess for the initial starting point, send NULL in for `initial_guess`. Likewise, if you don't have a particular termination tolerance in mind, send in -1 for `tol`.

4 This Implementation of MAPS

4.1 Breakdown of Information

This implementation of MAPS is broken down into a half-dozen or so major C++ classes, each with a header and an implementation file. Two of them are "abstract base classes", and hence have no complete implementation. The classes used in the implementation are derived from these abstract base classes via inheritance. The major functions of each class will be dealt with individually in the following sections. The below manual sections are only brief overviews — much more comprehensive documentation can be found in the code itself.

There one feature common to almost all of the classes listed below. These classes each have a debug(char*) function which outputs a near-complete picture of the state of the object at a given point in time. The string passed to debug, via the character pointer, will be output before each text line of its output.

4.2 Class: maps

The maps class is the 'superclass' of this program, so to speak. It contains the implementation of the maps algorithm proper, and pointers to objects of a variety of types. These pointers to objects include instantiations of InitialDesign, PSGrid, SearchCriterion and PatternSearch classes.

MAPS has four different constructors, a default constructor, which does very little, a copy constructor and two special-purpose constructors. For most applications, the simpler special-purpose constructor of MAPS, will do a fine job. Its syntax is as follows:

maps(long V1, long P1, const Bounds &B1, char* fn)

The four parameters are (in order), the function evaluation budget, the dimension of the variable space, an object containing the bounds of the problem, and the name of the objective function (passed in as a character pointer). The more complicated special-purpose constructor allows the caller to completely set most of MAPS' initial state.

Like most other classes discussed below, the maps class also contains a collection of "Reset" functions, which allow for the caller to reset major state elements of MAPS before the search is run. They all have basically the same syntax, allowing the user to pass in some kind of object, and returning a bool indicating success or failure.

The RunMAPS function actually runs the Model-Assisted Pattern Search. It has one optional command-line parameter which is the name of the file prefix for iteration-by-iteration graphs of the Approximation and Search Criterion. Due to limits in visualization technology, this only works if your function has a one or two dimensional domain. These graphs are generated by the user-callable GenerateMAPSGraphics function, which can also be used to generate pictures of the final approximation and search criterion after RunMAPS has been called.

RetrieveBestSeen and GenerateIterationHistory are two other functions that allow the user to retrieve information following a successful run of MAPS. The former simply returns the best point and its value, whereas the latter outputs an 'iteration history' to standard output. A sample history is shown below:

```
******MAPS ITERATION HISTORY******
# for parameters V=14 N=5 P=2 delta=0.01
# on function: ./tk
NUM      1        2      VAL
   1    0.42    0.71  62.2503
   2    0.67    0.33  42.6565
   3    0.26    0.96  79.4194
   4    0.87    0.08   177.17
   5    0.73    0.41  34.4587
   6    0.59    0.49 -6.39761
   7    0.49    0.43  52.0737
   8    0.63    0.57  63.6789
   9    0.61    0.45 -58.9764
  10    0.63    0.43  17.3618
  11    0.57    0.43  40.8011
  12    0.65    0.49  50.9157
  13    0.61    0.47 -9.92294
  14    0.49    0.53  60.0616

Best Point found is #9, value=-58.9764
**********************************
```

One can easily extract data from a file of these histories with use of system utilities such as `grep`, `cut`, and `awk`. Refer to Appendix B.3 for details on doing this.

4.3 Classes: InitialDesign and Derivatives

The InitialDesign class is an abstract base class that deals with the issues of generating an initial design for MAPS. Specific types of initial designs are derived from this class. A derived class must keep to the original functionality of the InitialDesign class, and include implementations for functions like GenerateDesign, which will actually generate the points to be used by MAPS. It is important to note that derived classes *must* call InitialDesign::GenerateDesign in their own implementation of GenerateDesign. This call ensures that the initial design is on the MAPS grid.

4.3.1 LatinHCDesign

LatinHCDesign is derived from the InitialDesign class, and implements Latin hypercube sampling on the region of interest. A Latin hypercube of n points divides each dimension into n bins, and places the points randomly within the bins, such that no two points share a bin in a given dimension. This implementation uses a shuffling algorithm to distributes points among bins, and then places the point with the selected bin with a uniform distribution. This implementation of the Latin Hypercube sampling has a time and memory complexity of $O(n \cdot p)$, where p is the dimension of the space.

4.4 Class: PSGrid

The PSGrid class stores all the information concerning the underlying grid, and its fineness. It provides a function SnapToGrid, which ensures that a point is within a certain numerical tolerance of a grid point. This is used by MAPS after getting the initial design, and returning a point from the optimized Search Criterion. Also included is the function GetDelta, which allows the user to retrieve the current grid fineness, and the function GetCore allows the user to find the core pattern of a given point.

4.5 Class: SearchCriterion

The SearchCriterion class is charged with the duty of keeping a copy of the approximation to the objective function, as well as keeping track of the design criterion and the current weighting scheme. The formula used to compute the value of the Search Criterion is as follows :

$$S_c(x) = \hat{f}_c(x) - w_c D_c(x)$$

where S_c is the search criterion, $\hat{f}_c$ is the current approximation to the function, D_c is the design criterion, and w_c is the weight currently placed on the design criterion. The approximation itself and design criterion are kept in an object of a derived class of class Approximate, to which SearchCriterion has a pointer.

Keeping track of the weighting scheme is the primary role for the SearchCriterion class. It currently has five types of weighting schemes. The first is called "Weight Multiple," the second "Weight Target Value," the third "Delta Multiple," the fourth "Dynamic Guess," and the fifth "Dynamic." The first operates with a user specified starting weight, and a user specified multiple. During each iteration of MAPS, the current weight is multiplied by the weight multiple, to yield a new weight. The "Weight Target Value" scheme works off of a user-specified starting value, a user specified value of weight 'insignificance' and a user specified iteration at which the weight is to become insignificant. The correct multiplier is then computed, and updates are handled much like in the "Weight Multiple" scheme. The "Delta Multiple" scheme works much like the "Weight

Multiple," except the weight is only reduced when delta is refined. The final two weighting schemes, "Dynamic Guess" and "Dynamic" both attempt to change the weight from from iteration to iteration depending on the current status of the approximation. The first uses a user-specified threshold percentage of the function range. The second uses a fixed distance of half the distance between the expected value at the last point and the best value as threshold. Experimentally, the final criterion (Dynamic) has shown the most promise.

In terms of interacting with the SearchCriterion class, the two most important functions are EvaluateSearchCriterion and UpdateSearchCriterion. They serve to allow MAPS to update and use the Search Criterion.

4.6 Classes: Approximate and Derivatives

The Approximate class is an abstract base class which keeps track of the Approximation and the Design Criterion. The major functions in this class are UpdateApprox and Evaluate Approx. Like their counterparts in the SearchCriterion class, they serve to allow MAPS to use the approximation to the objective function.

4.6.1 Class: NullApprox

The NullApprox class, which is derived from Approximate is designed for test purposes only. It returns zero for both the approximation and design criterion at every point, effectively turning MAPS into a very computationally expensive pattern search. **DO NOT USE THIS CLASS FOR A REAL OPTIMIZATION PROBLEM**.

4.6.2 Class: KrigApprox

This class is the default derivative of Approximate. Basically, the KrigApprox class contains a ParamEstimate object, which takes care of the parameter estimation that it needs done. Using those values, this class uses kriging to generate an approximation to the objective function. Besides inheriting Approximate's functionality, it also has a variety of other public member functions that serve to allow the user to pick the type of trend and correlation to use for generating the approximation.

4.7 Class: ParamEstimate

The ParamEstimate class handles the most mathematically sensitive part of this process, the parameter estimation. The class currently supports Isotropic and Product correlation functions, of both Gaussian and Exponential varieties. It also supports constant, quadratic, and user-specified custom trend estimation. Matrix inversion for computation of the trend is first attempted using a Cholesky factorization (thus necessitating CLAPACK). If this fails, the matrix is modified, and the factorization attempted again. The singular value decomposition (SVD) is used as a last resort for pseudo-inversion.

For the benefit of the user who may not be that familiar with this area of statistics, the following formulas represent the supported correlation functions:

Exponential Isotropic: $r_\theta(s,t) = e^{-\theta||s-t||}$
Gaussian Isotropic: $r_\theta(s,t) = e^{-\theta||s-t||^2}$
Exponential Product: $r_\theta(s,t) = \prod_{j=1}^{P} e^{-\theta_j|s_j-t_j|}$
Gaussian Product: $r_\theta(s,t) = \prod_{j=1}^{P} e^{-\theta_j|s_j-t_j|^2}$
Cubic Isotropic: $r_\theta(s,t) = \begin{cases} 1 - 1.5\frac{||s-t||}{\theta} + .5\left(\frac{||s-t||}{\theta}\right)^3 & \text{, when } ||s-t|| < \theta \\ 0 & \text{, when } ||s-t|| \geq \theta \end{cases}$

4.8 Other Software Included

In addition to the implementation of all of the above-mentioned classes for MAPS, there are a few additional files included, some of which are detailed in Appendices. Besides this software, implementations of the pattern search, compass search and a multi-start compass search are included. Also included are Steven Park and David Geyer's implementation of a Lehmer pseudo-random number generator and associated libraries for generating random variates from several common distributions. See "Random Number Generators: Good Ones Are Hard To Find" (Communications of the ACM, October 1988) by Steven Park and Keith Miller for more details.

5 Troubleshooting

5.1 MAPS will not compile/link

5.1.1 My system does not have g++/gcc/g77

Just replace a few lines in the Makefile with the appropriate substitutes. Suppose your C++, C and FORTRAN compilers are called `xlC`, `xlc` and `xlf` in that order. Then change:

```
CC=gcc
CXX=g++
FC=g77
```

to:

```
CC=xlc
CXX=xlC
FC=xlf
```

5.1.2 Netlib `f2c` files will not compile

This is a known bug of the Netlib `f2c` files on systems like Solaris. If this is the case, edit `libf2c/makefile.u` and remove the `-x` option from the `ld` line.

5.1.3 No support for bools

If your C++ compiler does not support the 'bool' data type, be sure to compile with the preprocessor macro `NO_BOOLS`. This will typedef the bools to ints, and solve the problem. If you're using the makefile, add `-DNO_BOOLS` to the `$(DFLAGS)` line.

5.1.4 Operator Declaration Errors in vec.h and cmat.h

If you start getting compiler errors like:

```
vec.h:338: declaration of `operator >>' as non-function
vec.h:338: parse error before `<'
vec.h:338: declaration of `operator >>' as non-function
vec.h:338: parse error before `<'
cmat.h:397: declaration of `operator >>' as non-function
cmat.h:397: parse error before `<'
```

Be sure to include the preprocessor macro `OLD_LIBC`. This error is due to the fact that your compiler doesn't fully support templates, but this problem is corrected easily through this macro. If you're using the makefile, add `-DOLD_LIBC` to the `$(DFLAGS)` line.

5.1.5 Compiler Warnings

The compilation of MAPS will generate a number of warnings, but this is intended. There are class implementations that do not use all of their parameters, but the parameters are necessary for inheritance purposes. Hence, the compiler will probably spit out a warning, but this in no way detracts from the software's functionality.

5.1.6 Linker errors

The makefile is included for your benefit, and everything should work well. The makefile assumes that CLAPACK, BLAS, and the F2CLIBS are set up correctly as linkable libraries and are either in your library path or in the current directory. If you've changed the names or directory, edit the `$(LDFLAGS)` line in the makefile to reflect this. If your system administrator has installed these libraries system-wide, you should be fine so long as you get the names correct. If not, be sure to make the necessary changes in the makefile, or else MAPS won't link.

If it doesn't work, always remember to link from most general to least general. So for this program, link your files first, then MAPS, followed by CLAPACK, the CBLAS, the F2CLIBS, and finally the C math libraries.

5.2 MAPS does not run correctly

5.2.1 MAPS hangs

Check to make sure you specified the dimension of the objective function correctly. If your objective function expects more variables that MAPS gives it, you'll probably hang the application.

5.2.2 Execlp errors

Errors such as:

`execlp: No such file or directory`

are caused by incorrectly entering the name of the objective function. be sure to enter either a full (`/home/user/program`) or relative (`./program`) path for the program representing the objective function.

5.3 Using the testClassic interface

5.3.1 My objective function isn't in C++!

Cross-compilation is not an easy thing to do, and regardless of what kind of trick you use, it is bound to be system-dependent. Consult with your local compiler guru. Be careful to note the name-mangling done by all compilers involved.

5.3.2 Linker "undefined reference" errors

Chances are this is a cross-compilation issue, and thus has to be dealt with accordingly. If it is not, be sure that you have a function called `mfcn`, which has the proper calling syntax.

6 The Theory Behind MAPS

Information on MAPS theory and results can be found in *Model-Assisted Pattern Search* (C. Siefert 2000). The full text of this document can be found in postscript form at: *http://www.cs.wm.edu/~va/CS495#Chris.*

7 Concluding Remarks

This implementation was made possible with the assistance of Virginia Torczon, Michael W. Trosset, Anthony Padula, D. Erin Parker, Anne Shepherd and Elizabeth Dolan. Financial support was provided by the Roy Charles Center and the National Science Foundation (CCR-9734044 and EIA-9712718).

A mdat2tex — a MAPS to TEX Converter

Also included in this distribution is `mdat2tex`, an awk script that converts files containing MAPS iteration histories into usable LATEX tables. It does a fairly good job of these conversions, and the tables will require only minimal editing to look good in a LATEX document. The easiest way to run `mdat2tex` is as follows:

```
awk -F mdat2tex.awk mydata.dat > myoutput.tex
```

The following is a the LATEX source for a sample table generated by `mdat2tex`:

```
% From file: sample.dat, this is table #0
\begin{center}\begin{tabular}{c|r|r|r|r|r}
\textbf{Iteration} &\textbf{$x_0$} &\textbf{$x_1$} &\textbf{$x_2$} &\textbf{$x_3
$} &\textbf{Value}\\
\hline
1& 9.8& 2.1& 4.2& 7.1& -0.192076\\
2& 2.8& 6.1& 6.7& 3.3& -0.351249\\
3& 1.5& 5.1& 2.6& 9.6& -0.249708\\
4& 6.3& 1.2& 8.7& 0.8& -0.49091\\
5& 7.1& 4.4& 9.9& 0.0& -0.22272\\
6& 6.7& 1.6& 8.7& 0.8& -0.566242\\
7& 7.1& 1.6& 9.1& 0.8& -0.544828\\
8& 6.9& 2& 8.7& 1.2& -0.586722\\
9& 7.1& 2& 8.3& 1& -0.700908\\
10& 7.7& 2& 7.7& 0.8& -0.838886\\
11& 7.7& 2& 7.1& 1& -0.775674\\
12& 8.1& 2& 7.7& 0.4& -0.720954\\
13& 7.7& 1.8& 7.7& 1& -0.986294\\
14& 7.9& 1.2& 7.7& 1.4& -1.30526\\
\end{tabular}\end{center}
\begin{center} Best Point is \#14, value=-1.30526
\end{center}
```

Run through LaTeX, it looks like this:

Iteration	x_0	x_1	x_2	x_3	**Value**
1	9.8	2.1	4.2	7.1	-0.192076
2	2.8	6.1	6.7	3.3	-0.351249
3	1.5	5.1	2.6	9.6	-0.249708
4	6.3	1.2	8.7	0.8	-0.49091
5	7.1	4.4	9.9	0.0	-0.22272
6	6.7	1.6	8.7	0.8	-0.566242
7	7.1	1.6	9.1	0.8	-0.544828
8	6.9	2	8.7	1.2	-0.586722
9	7.1	2	8.3	1	-0.700908
10	7.7	2	7.7	0.8	-0.838886
11	7.7	2	7.1	1	-0.775674
12	8.1	2	7.7	0.4	-0.720954
13	7.7	1.8	7.7	1	-0.986294
14	7.9	1.2	7.7	1.4	-1.30526

Best Point is #14, value=-1.30526

B testMPG.cc — a Sample Test Program

`testMPG.cc` is a simple shell of a test program that allows the user to easily customize most of the functionality of MAPS. It allows the user to select the seed for the random number generator (to allow for reproducible results), the number of runs to make, the evaluation budget (V), the number of sites (N) to allocate to initial design, the starting delta, and the correlation family to use for the kriging.

B.1 testMPG's command-line parameters

In addition, `testMPG.cc` supports three different optional command line parameters. The `--help` parameter prints out a list of the available parameters. The `--graphics` parameter will create data useful for the creation of graphs of the final approximation and search criterion. The `--iteration` parameter creates data files which can be used for the creation of graphs of the approximation and search criterion for each iteration of MAPS. With this parameter, you specify the file name prefix for the output files. The `--graphics` parameter uses the name of the objective program as its file name prefix for the output files. For more information about MAPS' output conventions, refer to Section 4.2.

B.2 Using --iteration and --graphics to Create Graphs with GNUplot

Using the `--iteration` and/or the `--graphics` command-line options for `testMPG` will allow you to create graphics rather easily from the software's data output using GNUplot, assuming that your objective function is one or two dimensional. For functions of one variable, type the following at the GNUplot prompt:

```
gnuplot> plot "fpref.MAPS.GRAPH.dat" with lines
```

where "fpref" is the file prefix you specify to `testMPG`, and "GRAPH" is either "AP" or "SC" depending on whether you want to view the approximation or search criterion graphs.

To produce plots for functions of two variables, type the following at the GNUplot prompt:

```
gnuplot> set parametric
gnuplot> splot "fpref.MAPS.GRAPH.dat" with lines
```

Again, "fpref' is the file prefix you specify to `testMPG`, and "GRAPH" is either "AP" or "SC" depending on whether you want to view the approximation or search criterion graphs.

Also, you can visualize the evaluation sites superimposed on either the approximation or the search criterion. To do so, type:

```
gnuplot> plot "fpref.MAPS.GRAPH.dat" with lines, "fpref.MAPS.PTS.dat" with points
```

for one variable functions. For two variable functions, type:

```
gunplot> set parametric
gnuplot> splot "fpref.MAPS.GRAPH.dat" with lines, "fpref.MAPS.PTS.dat" with points
```

where "fpref" is the file prefix you specify to `testMPG`, and "GRAPH" is either "AP" or "SC" depending on whether you want to view the approximation or search criterion graphs.

B.3 Getting What You Want from testMPG

In normal operation (no parameters), `testMPG` will output complete iteration histories for each run of the MAPS algorithm. This is probably more data than is usually necessary or desired. Use of system utilities, such as `grep`, can make it easy to extract what is required from this wealth of data. To extract the best value found only, use the following commands, where mydata.dat is the output from `testMPG`:

```
cat mydata.dat | grep Best | cut -f2 -d=
```

If you want to see which iteration MAPS found the best point on for all of your trials, and want these numbers tallied, try:

```
cat mydata.dat | grep Best | cut -f2 -d# | cut -f1 -d',' | sort
-n | uniq -c
```

Or if you want to see the final fineness of the grid, and have this number tallied, try:

```
cat mydata.dat | grep delta | cut -f5 -d= | sort -n | uniq -c
```

C A Sample Objective Function

The C++ source that follows represents a sample objective function that fits the form required by MAPS. Compile this "function" independently from MAPS.

```
#include <iostream.h>

int main() {
  double x,y;
  double f;

  cin>>x>>y;

  f=x+y;

  cout<<f;
  return 0;
}
```

Included with this distribution is a C++ program that implements the Shekel family of objective functions. The file `shekel.cc`, written by Erin Parker contains the implementation of this family of four-dimensional functions.

D wrapper.sh — a "Wrapper" for Objective Functions

`wrapper.sh` is a simple shell script that will allow the use of an objective function that requires command-line parameters with MAPS. Like all other software in this distribution, feel free to modify as you see fit. The code is shown below:

```
#!/bin/sh
# Sample wrapper for a program with command-line arguments,
# so it can work with MAPS
```

```
# Chris Siefert, College of William and Mary

my_objective_function arg1 arg2 arg3
exit $?
# This returns the return value of the objective function
```