INTEROPT: Philosophy

INTEROPT is a computer program specifically designed to solve difficult optimization problems. It establishes a convenient, easy-to-understand, dialogue with the user, in which the user is prompted with natural language questions, and asked to describe his/her optimization problem. INTEROPT then writes a computer program based on this dialogue, and runs that program to solve the specified problem. INTEROPT has several capabilities that are unique in optimization theory:

•Ability to find global minima: Often, an optimization problem suffers from local minima. That is, we are interested, not in a minimum of a function, but in the best minimum of that function. INTEROPT is usually able to find such minima.

•Ability to deal with continuous variables

Probably the best known technique for solving optimization problems with local minima is Simulated Annealing; and INTEROPT is based on a derivation of this theory. However, Simulated Annealing, as previously described in the literature, works best on problems in combinatorial optimization; that is, problems in which the variables take on only a few (usually 2) possible values.

Although completely general purpose and easy to use, INTEROPT does have some limitations:

•Limited number of variables Currently, INTEROPT has only been successful in solving optimization problems with less than 30 variables

•Inability to deal with constraints So far, we have been unable to formulate INTEROPT to handle constrained optimization problems, with the following exception: Since the user may specify the range of possible values of each variable (and thus define the search space), it is often possible to formulate constraints as conditions on this search space.

•INTEROPT places heavy demands on computer resources. Don't even try to run INTEROPT without at least 4 Mbytes of user memory. Furthermore, because of the statistical search methods used, the faster the computer, the better.

INTEROPT was written with the following philosophy: to develop an optimization program which can solve hard optimization problems as effectively and as fast as possible, while requiring as little software development as possible on the part of the user. Following this philosophy, INTEROPT talks to the user, asking questions like How many variables are there in this problem?

and the user replies with simple responses. A sample dialogue is given below.

Using INTEROPT

Most of the use of INTEROPT is self explanatory, and the interactive help feature is generally sufficient. However, in this section, we provide an overview of the operation of INTEROPT, and discuss some terminology which might otherwise be confusing.

Modes of Operation

INTEROPT operates in basically two modes, function minimization and data fitting. Actually, data fitting is simply a user-friendly front-end to the function minimization capability, where the function to be minimized is the means-squared error between the data provided and the specified function.

Function minimization

In this mode, the uses is asked to specify a function, and the range over which to search. The global minimum of the function, within the range, will be found. For example, suppose we are to find the minimum of

$$y = \frac{\sin x}{x}$$

$$\text{for}(\pi < x < 3\pi)$$

Then, we simply specify the function when asked to, and specify the bounds of the parameter x, and INTEROPT will find the value of x which minimizes y.

Data fitting

When fitting data, we are to find the values of certain parameters which minimize the mean-squared-error between the data and the function.
Before we continue with this discussion, we need to carefully define a couple of terms:

TERMINOLOGY:

The terms *variable* and *parameter* are used in different ways in INTEROPT, and may be confused. When minimizing a function, as in the example above, we are solving for the value of the variable x which minimizes y. However, consider a problem in function fitting (again in one dimension, just to make the explanation clearer). Suppose we are to find the values of p, s, and m which minimize the fit of

$$y = p\exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$$

to some x, y pairs of data which the user has stored in a file. In this context, x is our variable, and p, μ, and σ are the parameters of the fit. Unlike the function minimization case, in the data fitting case, INTEROPT is solving for the parameters, and is given values for the variables.

Using vectors.

The variables or the parameters, in both the cases of function minimization and data fitting, may be vectors. In addition, the outputs may be vectors in the case of data fitting. This is discussed more in the next section.

Format of data files

In the MSE fitting case, the data to be fit must be specified in an ascii data file. The outputs are specified first. In the data fitting example given above, the first two lines of the data file might be
1.0 0.2
0.8 0.3
for a Gaussian with mean $\mu = 0.2$. In a fitting problem with vector-valued outputs, for example

$$y_1 = p_1 \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$$

$$y_2 = p_2 \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$$

the first two lines of the data file might be
1.0 1.0 0.2
0.9 0.8 0.3
In this example, the data correspondence is
$y_1\ y_2\ x$.

Specifying a function

A function is specified in the syntax of the C programming language.

Function minimization

When asked to type in the function to be minimized, the user should type an equation of the form.
y = f (x). For example:

y = a*a + b * exp (3 * x * x)

Data fitting

When asked to type in the function to be fit, the user should type the equation(s) in the same form, likewise using the syntax of C, but in this case, the user must specify which of the possible outputs is computed. For example,
y[0] = a*a + b1 * exp (9 * x * x); \
y[1] = a*a + b2 * exp (3 * x * x)
The use of the array structure for y is required here, since INTEROPT can find the set of parameters which simultaneously fit several outputs. Note that no semicolon is required after the *last* line.

Continuation lines

When specifying a function to INTEROPT, lines may be continued by terminating the line with a \ (backslash). **NOTE: Some shells (csh in particular) treat backslash as a special symbol. In such a case, the backslash must be escaped (type two backslashes)**

Complex expressions

The expressions entered as functions are arbitrarily complex C expressions, and may contain IF statements and local variables. In this case, enclose the entire expression in curly brackets. For example

```
{\
double xtemp;\
xtemp = x ·z;\
if (xtemp < 0.0) y[0] = 3.14159;\
else y[0] = ln(xtemp);\
}
```

Note that no semicolon is required after the last line.

Peculiar Conditions

INTEROPT cannot deal with singularities. In the case of positive singularities, (for example, y = sin(x) / x), INTEROPT CAN fail with the famous “Floating point exception” error message. It is necessary for the user to be sure that no such singularities

exist within the range of possible values. The example of 1/(sin(x)/x) is particularly bad, since the singularity actually occurs at the minimum.

Negative singularities cause a different problem: INTEROPT tries very hard to find the minimum of the function. A negative singularity is therefore extremely attractive, and INTEROPT will end up in the singularity every time.

The other condition which can cause problems is the case of transcendental functions such as *exp* or *ln*, in which the arguments can only take on particular ranges of values. That is, exponents with arguments larger than 50 or smaller than -50 are not permitted, nor can one have logarithms with negative arguments. These two special cases (ln and exp) are explicitly trapped and handeled by INTEROPT. The expression entered by the user is scanned before compilation and converted into another function call which will not allow illegal argument values. If such arguments are passed to the function, an error message is given, a “reasonable” value is returned, and the program continues to run. NOTE: Deep within the guts of INTEROPT, the exp function is called, even though it may not be called explicitly by the user. Therefore, the user will occasionally see the *argument out of range to exp function* error message occur. Just ignore such warnings.

Installing INTEROPT

In this section, we provide some description of the files and directory structures used by INTEROPT,

Installing INTEROPT on Unix systems

INTEROPT is set up to run very effectively on UNIX and UNIX look-alike systems. As discussed earlier, speed of operation was a primary consideration, and therefore, the most effective means of software generation was used. INTEROPT actually write programs, and then, via operating system calls, compiles and executes those programs. To install INTEROPT on a UNIX system, simply copy the existing files in the directory structure specified, using FTP, rcp, or whatever other file transfer program you wish, or reading from the distribution media.

Disk space required

Disk space requirements for INTEROPT are minimal. A fully configured system generally requires around 500Kbytes.

Directories

Within the home directory for INTEROPT (which we refer to henceforth as HOME/), several subdirectories are required.
src: This directory contains the source of the INTEROPT main program and subroutines. If you are on a machine with a system-level installation, you may use symbolic links for all of these EXCEPT opt.c. This file is modified, and must actually exist in your directory
obj: This is a temporary directory used to hold object files during compiles. It may be empty when you first receive INTEROPT.
·*include*: This directory contains include (.h) files used by INTEROPT. It may be a symbolic link.
·*demos*: This directory may not exist in your installation. It contains examples of data fitting and function minimization. It is not required.
interopt.workdir: This directory is generated automatically by INTEROPT and used to hold temporary results. It may or may not exist when you receive your distribution. If it does not exist, INTEROPT will automatically make it. Afterward, under certain conditions, INTEROPT will try to make this directory even though it already exists. If that happens, you will see the message:
interopt.workdir: directory exists
Just ignore this message.

Source files

INTEROPT uses a number of files, some automatically generated, and some modified at run time. In the following, only the files which the user may have questions about are documented.
src/opt.c
This file is modified by INTEROPT, and, when compiled and linked, becomes the program which actually performs the minimization.

Executable files

Two programs are of primary interest to the user: INTEROPT itself, and OPT. INTEROPT is run by the user, and generates OPT automatically as a result of an interactive dialogue with the user.

Results files

After optimizations have been run, answers may be found in two places:
parameters.h
Answers.dat

Answers.dat is the more useful of these output files. Furthermore, it will contain accumulated results in the case of fitting several sets of data to the same function. parameters.h contains only the last result. Parameters are identified in *Answers.dat* by the logical name given by the user. *Answers. dat* is re-initialized at the beginning of each INTEROPT run.

Log files

INTEROPT logs all commands which are typed to it into interopt.log. This file may be renamed, edited if needed, and used for input (by redirecting standard in) on later runs. Using this file is the preferred means of operation, since interactive dialogues take so long.

Installing INTEROPT on non-Unix machines

INTEROPT is designed to make optimal use of computer performance by writing, compiling, and running other programs as subprocesses of INTEROPT itself. For that reason, its structure is highly operating-system dependent.
We currently only support
•MacIntosh running OS-X
•SUN's running SOLARIS
•PC's running LINUX.

Running Interopt

Running Interopt from another directory

Once INTEROPT is installed in the system-wide INTEROPT directory, you will need to make a special directory in which to run it, and you will need some of the files from the system directory. The easiest way to do this is to create a directory in your area and call it INT. Then, within this directory, establish symbolic links to the system INTEROPT directory. For example, suppose the system INTEROPT directory is /usr/local/interopt.
From your home directory, type

```
mkdir myinteropt
cd myinteropt
ln -s /usr/local/interopt/src src
ln -s /usr/local/interopt/include include
ln -s /usr/local/interopt/libinteropt.a libinteropt.a
mkdir demos
alias interopt /usr/local/interopt/interopt
```

In this structure, you would put your data files in your demos directory, and would run INTEROPT by typing

```
interopt
```

or, if you are redirecting standard input,

```
interopt < demos/commandfile
```

Of course, you don't have to use a demos subdirectory, you can put your data files anywhere you wish.

INTEROPT control parameters

INTEROPT has only 3 control parameters. They are specified on the command line as follows:

`-d ds`

`ds` is an optional "rate" term specified on the command line. The smaller this number is, the slower INTEROPT will run. However, slower runs will generally result in more accurate answers. This parameter, called ds, specifies the amount of variance of the energy which is removed per iteration. This parameter is exactly analogous to the entropy in a physical process. Slower is equal to more careful. The default value for `ds` is.01.

Example:

```
interopt -d .001
```

or, if redirecting standard input

```
interopt -d .001 < commandfile
```

Datasets

In the application of INTEROPT to data fitting, it is often useful to be able to have a number of data sets in the same input file. The number of elements in a single data set is one of the questions answered in the interactive dialogue. However, if the file contains many such data sets, this command-line argument may be used to specify that fact. In the case of multiple data sets, the sets are not separated at all in the input file. The second N elements simply follows the first N.

·-h

Help. If this switch is specified, a help message will be given, and the program initiated.

Example Dialogs

In the examples below, the information typed by the user is shown in red, comments to the reader of this documentation are in blue.

Defining a function and finding its minimum

```
% interopt

Your answers will be logged to interopt.log (this allows
you to save interopt.log, rename it, and use it later as
stdin)

Do you want to optimize a function?y
mkdir: interopt.workdir: File exists (note this is not an
error message: you will get this if you have already run
interopt in this directory)
```

```
How many variables does the function have?1

What is the name of variable number 0?x

Enter the function to be minimized

use the form
 y = f(x)
y = x * x -x

I'm now compiling your function. Please wait
I'm now compiling the main program. Please wait

OK. We got that far. Now, tell me the max and min values
on each variable


What is the lower limit of x?-2

What is the upper limit of x?2
I am now linking your program together
cc -o opt   interopt.workdir/opt.o obj/sa.o
interopt.workdir/H.o obj/helps.o \
interopt.workdir/reportI.o obj/genreps.o obj/Wsubs.o -lm

I am now running your optimization
```

Fitting data

```
% interopt
Your answers will be logged to interopt.log

Do you want to optimize a function?n

Well then, would you like to fit some data?y
mkdir: interopt.workdir: File exists

How many parameters does the fit have?3

How many variables does the fit have?1

How many outputs does the fit have?1

Data file name?demos/datairene

How many data points are in that file?9
```

```
Now I need to know the parameters over which I should
minimize

What is the name of parameter number 0
a

What is the name of parameter number 1
b

What is the name of parameter number 2
c

Now, I need to know the names of the variables

DONT USE y!


What is the name of variable number 0
x

Enter the function to be fit
use the form
 y[i] = f(x), with a different y[i] for each output
y[0] = a + b * exp ( c * x )

I'm now compiling your function. Please wait
I'm now compiling the main program. Please wait

OK. We got that far. Now, tell me the max and min values
on each variable


What is the lower limit of a?30

What is the upper limit of a?50

What is the lower limit of b?-20

What is the upper limit of b?0

What is the lower limit of c?-0.1

What is the upper limit of c?0
I am now linking your program together
```

```
cc -o opt   interopt.workdir/opt.o obj/sa.o
interopt.workdir/H.o obj/helps.o \
interopt.workdir/reportI.o obj/genreps.o obj/Wsubs.o -lm

I am now running your optimization
```

The data file for this example (demos/datairene) is as follows: (remember that these are y,x pairs... had there been more variables than just x, they would have been in order, y, x1,x2,...).

```
28.03 0.0
35.46 60.0
37.97 120.0
39.15 180.0
39.47 240.0
39.51 300.0
39.96 360.0
40.17 420.0
40.17 480.0
```