Optimization

ADDENDUM

Explanatory Supplement for the DEMONLP Program

This material will be of interest to users of the QuickPak Scientific DEMONLP program. It is a more detailed overview and explanation about how to properly interact with the software when setting up your own constrained optimization applications. Remember that DEMONLP is a lengthy and complex computer program. It was designed to serve as a "shell" for solving your own non–linear programming (NLP) problems.

Problem Statement

The formal definition of the multi–variable, constrained, non–linear programming problem which can be solved with the DEMONLP computer program is as follows:

Solve for a vector $\bar{x} = (x_1, x_2, ..., x_{NX})$ of *design variables* which minimize or maximize a non–linear, scalar *cost* or *objective function* given by

$$f(\bar{x}) = f(x_1, x_2, ..., x_{NX})$$

subject to a vector of linear and/or non–linear *equality constraints*

$$\bar{h}_j(\bar{x}) = \bar{h}_j(x_1, x_2, ..., x_{NX}) = 0 \qquad j = 1 \text{ to } NH$$

and a vector of linear and/or non–linear *inequality constraints*

$$\bar{g}_i(\bar{x}) = \bar{g}_i(x_1, x_2, ..., x_{NX}) \le 0 \qquad i = 1 \text{ to } NG$$

where NX is the total number of design variables, NH is the total number of equality constraints, and NG is the total number of inequality constraints.

Practically speaking, we want to find the "best" solution to our problem which also satisfies one or more physical or design constraints. These constraints may be due to the types of materials we use to solve the problem, the physics which define the behavior of our dynamical system, or other assumptions we have made concerning our problem.

Global Arrays

The following is a brief description of the global arrays used in the software. Do not change the size, contents, or type of any of these arrays. They are absolutely required for any application you may write.

N(50) = integer array of position pointers for the control variable, constraint, and other working arrays

METHOD(35) = integer array which describes the types of algorithms DEMONLP will use, input and output options, and algorithm termination and control criteria

TOL(35) = an array of algorithm iteration and convergence tolerances.

Global Variables

The software also uses three global variables to communicate with other options of the program. The purpose of these variables is to provide information about how well the algorithm is performing. Any or all of these variables can be eliminated from your application. However, you will not have access to either the iteration information and/or the immediate results option of the computer program.

IPRT = print intermediate results flag *(1 = yes, 0 = no)*

NPI = integer number of constrained iterations

IFN = integer number of unconstrained iterations

The intermediate printout will be of interest to users who understand the terminology of optimization. Most of the information provided consists of a minimization summary at each unconstrained optimization step with data about the value of the Lagrange multipliers, constraints, augmented function, etc.

Main Program Variables and Arrays

As described in the original DEMONLP documentation, the only main program variables you need to modify for your own custom applications are NX, NG and NH. Do not modify any other variables defined in the main program. This includes such items as NF, MAXF and MAXG. It is useful to think of these as *reserved* variable names and it is best not to use them in your user subroutine or any other code you add to this program.

Furthermore, do not modify or eliminate the following statements:

```
DIM X(MAXX), F(MAXF), G(MAXG), H(MAXH)
DIM SHARED FFD(NX)
```

The array variable names X, F, G, H, and FFD are reserved for DEMONLP. Do not use these variable names in any code that you add to the "shell" software.

The main program performs several calculations and array dimensioning which are essential to the successful performance of this computer program. This includes such statements as:

```
NC = NG + NH
IF (NX > NC) THEN NC = NX
```

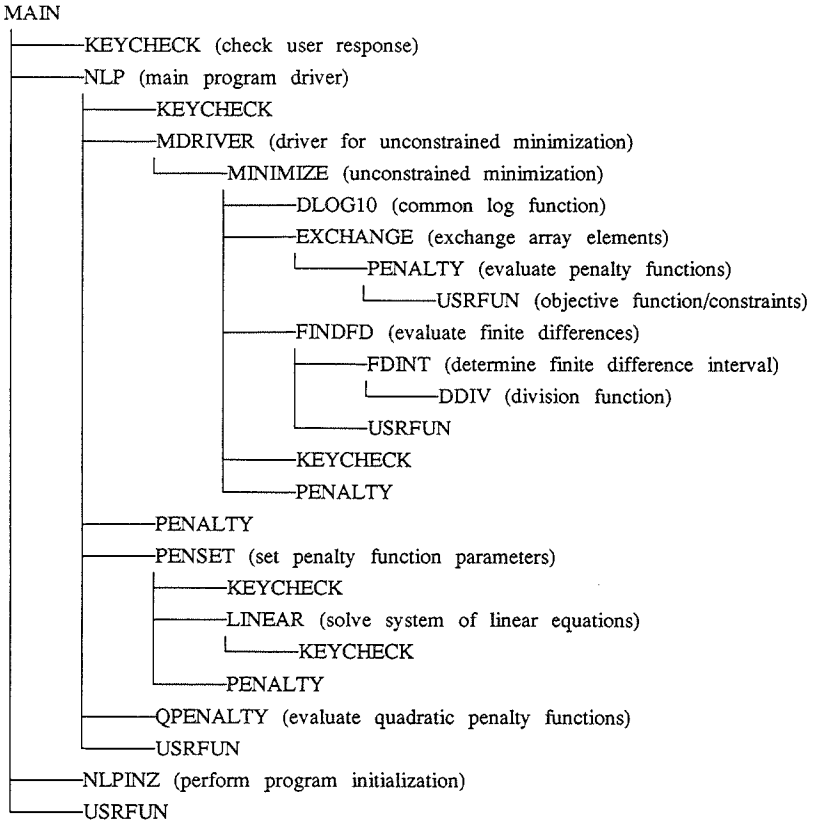
and

```
MAXX = NX * (7 + NC) + NC ^ 2 + NX * (NX + 1) / 2
```

Do not remove these statements from your computer program.

Functions and Subroutines

Each and every one of the functions and subroutines contained in the DEMONLP program is required in order for the software to work properly. The majority of information is passed among the subroutines and function through parameter lists and "working" arrays. The following diagram illustrates the relationships and hierarchy between the main program and all functions and subroutines. A brief description of each function and subroutine is provided in the first occurrence of the function or subroutine name. The basic tree diagram was created with Don Malin's XREF software.



There is no need for the user to directly interact with any of these subroutines or functions other than the QuickBASIC main driver subroutine via the statement

CALL NLP(X(), F(), G(), H(), IERR)

Program Initialization

The user initialization of the variables NX, NG, and NH is mandatory if the software is to operate correctly. These variables define the *size* and *type* of NLP problem by specifying the number of control variables and type of constraints.

The programmer must also specify the maximum number of non–linear iterations the software is allowed to perform when searching for the solution. There are several criteria which the user can modify which control the convergence behavior of the DEMONLP algorithm.

In addition, initial guesses for the control variables must be provided by the user. These guesses give the algorithm a starting point from which to begin the optimization search. The closer this initial guess is to the actual solution, the quicker and more likely the algorithm will converge.

The type of method used for computing derivatives must also be selected by the user. For the one–sided finite difference method, the software will compute approximate partial derivatives with the *forward finite difference* equation

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

and for the symmetric finite difference method the formula is

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i + \delta x_i) - f(x_i - \delta x_i)}{2\,\delta x_i}$$

In these equations δx_i is called the *finite difference* or *perturbation* interval. The "best" value for the perturbation interval is determined in program DEMONLP with an algorithm due to Gill, Wright, and Murray.

Program initialization is performed by the statement

```
CALL NLPINZ(X(), F(), G(), H(), IER)
```

Do not eliminate this subroutine call statement from your program. Several parameters are set to "default" values in this subroutine and can be changed by the user. The first parameter set defines the convergence and termination criteria for the algorithm. These four elements of the TOL array are defined as follows:

TOL(1) = convergence tolerance on function decrease

TOL(4) = convergence tolerance on constraint error

TOL(7) = convergence tolerance on step size

TOL(8) = termination tolerance for the unconstrained method

The default values set in subroutine NLPINZ should work fine for most problems. Setting these parameters to smaller values may improve the solution and will always require more computing time. Sometimes it is better to relax these values until a solution is found, and then "tighten" one or more parameters to improve the "relaxed" results.

The second set of algorithm parameters involve penalty constants and Lagrange multipliers. These parameters are very "problem dependent", and should only be changed if you have additional insight into your particular problem and you understand the concepts of penalty functions and Lagrange multipliers. For example, you might change the default values to those found during the solution of a similar type of problem, or when "restarting" a difficult NLP problem. These constants are in the following elements of the TOL array:

TOL(16) = initial value of equality constraint penalty constants

TOL(17) = initial value of inequality constraint penalty constants

TOL(18) = initial value of equality constraint Lagrange multiplier

TOL(19) = initial value of inequality constraint Lagrange multiplier

Error and Diagnostic Information

The DEMONLP program provides the user with a variety of error and diagnostic messages. These messages may appear during initialization and while the software is running. The initialization error messages have to do with the problem setup and array allocation. If these types of errors occur, check the problem definition variables NX, NG and NH. If array allocation errors occur, you have probably modified or overwritten one or more main program statements.

During computation a fatal error may occur. The error message displayed may be one of the following:

Singular matrix encountered during solution process !

Error during unconstrained minimization computation !

If either of these messages is displayed, it is usually best to restart the program and try other program options, different initial conditions, or a combination of both.

Several diagnostic messages may also appear while the main program is running. These errors may be one or more of the following:

Zero element in gradient-potential error in CV definition !

Gradient vector has nearly equal elements !

Gradient is out of acceptable range !

These messages are provided for the user's information only and are usually not fatal. After displaying the message and receiving a user response, the software will attempt to proceed with the computations.

Programming Example

The most important part of successful non–linear programming is setting up your problem correctly. This process involves the following steps:

- Defining the objective function
- Selecting the control variables
- Recognizing the problem constraints
- Providing a reasonable set of initial conditions.

To help illustrate this process, we will discuss and formulate a classic problem of space flight mechanics. The problem we have chosen involves the determination of optimum gravity–assisted interplanetary trajectories. These types of trajectories are being used for the Galileo mission to Jupiter and the Ulysses space mission to the Sun.

These types of trajectories consist of the following sequence of events:

- Launch from the origin planet into an orbit about the Sun
- Interplanetary cruise to a second intermediate planet
- Gravity–assisted flyby of the intermediate planet
- Interplanetary cruise to the destination planet

The intermediate planet is used to alter the trajectory is a way which sends the spacecraft on an intercept trajectory with the destination planet. This is accomplished by using the gravity of the intermediate planet to change both the direction and energy of the spacecraft's *heliocentric* or Sun–centered trajectory.

The heliocentric speed may be increased or decreased by flying either in front of or behind the intermediate planet. The trajectory of the spacecraft relative to the Sun is an ellipse while the trajectories relative to the departure, intermediate, and destination planets are hyperbolas.

For this problem, we would like to minimize the launch energy required to perform this type of space mission. The financial cost and complexity of the launch vehicle and spacecraft systems are directly related to the energy required. The scalar objective function for this problem is as follows:

$$\mathcal{E} = f(D_l, D_f, D_a) \qquad (1)$$

where $\mathcal{E}$ is the launch energy and D_l, D_f, and D_a are the dates of the launch, flyby, and arrival, respectively. Note that this equation defines our scalar objective or cost function and the control or design variables for this problem. We could eliminate one or more control variables, but a general rule when setting up NLP problems is to include as many control variables as possible. If we wanted to maximize the launch energy our objective function would simply be $-\mathcal{E}$.

The maximum turn angle possible during a planetary flyby is a function of the planet's radius R_p, gravitational constant μ, and the hyperbolic speed during encounter V_∞. The correct mathematical relationship is:

$$\psi_{max} = 2 \sin^{-1}\left\{ \frac{1}{1 + \frac{R_p V_\infty^2}{\mu}} \right\} \qquad (2)$$

This is the turn angle when the spacecraft just "grazes" the planet's surface. If the planet has an atmosphere, the value of R_p must be increased accordingly.

The actual turn angle during the encounter is a function of the incoming hyperbolic velocity vector $V_{\infty in}$ and the hyperbolic velocity vector $V_{\infty out}$ after the flyby. The dot product of the two unit velocity vectors provides this angle:

$$\psi = \cos^{-1}\left[\hat{V}_{\infty in} \bullet \hat{V}_{\infty out} \right] \qquad (3)$$

where

$$\hat{V}_{\infty} = \frac{V_{\infty}}{|V_{\infty}|} = \text{unit velocity vector}$$

$$|V_{\infty}| = V_{\infty} = \sqrt{V_{\infty_x}^2 + V_{\infty_y}^2 + V_{\infty_z}^2}$$

The constraints imposed upon the solution of this problem are due to physics and geometry. Since we must obey the law of conservation of energy relative to the flyby planet during the gravity–assist, the magnitude of the incoming hyperbolic velocity vector must be equal to the magnitude of the outgoing hyperbolic velocity vector. The dynamical *equality* constraint which must be satisfied is

$$V_{\infty in} = V_{\infty out} \qquad (4)$$

The planet and spacecraft actually exchange energy during the flyby. However, this is not perceivable by the inhabitants, but is quite noticeable to the spacecraft!

For a physically realizable flyby trajectory to exist, the spacecraft must pass above the planet's surface and atmosphere. Therefore, the following geometric *inequality* constraint must also be enforced during the solution process:

$$\psi_{max} - \psi \geq 0 \qquad (5)$$

Finally, the orbital energy per unit mass for any hyperbolic orbit is given by

$$\mathcal{E} = V_{\infty}^{2} \qquad (6)$$

To summarize the problem definition for this example:

- The objective function F(1) is defined by Equation (1) and computed with Equation (6) where V_{∞} corresponds to the launch energy
- The three elements of the control variable array X are the launch, flyby, and arrival dates as specified in Equation (1)
- The single "energy" equality constraint H(1) is given by Equation (4)
- The single "geometric" inequality constraint G(1) is defined by Equation (5)

The solutions for the two trajectory "legs" between the departure, flyby, and arrival planets are determined with Lambert's theorem. The solution consists of the heliocentric trajectory which connects any two planet position vectors and satisfies the time–of–flight between the planets. The time–of–flight is the difference between the two calendar dates for each leg of the complete transfer trajectory. The solution of Lambert's problem also provides the necessary hyperbolic velocity vectors at each planet.

Initial conditions for this problem can be obtained by a two step process. The first step consists of solving the minimum energy *ballistic* interplanetary transfer problem. This is the orbit between the launch and destination planets without a gravity–assist from a third planet. This problem can be solved with a combination of Lambert's theorem and an unconstrained, multi–variable optimization algorithm such as the QuickPak Scientific DEMOMNZ2 program. For this case the objective function depends only on the launch and arrival dates, and there are no problem constraints.

In the second step, a flyby planet is selected and the minimum energy ballistic transfers between both legs of a flyby trajectory are computed. During these calculations the launch and arrival dates are held fixed to the values found for the ballistic transfer, and only the flyby date is allowed to change. This step determines if there is a candidate flyby date which occurs between the launch and arrival dates of the original ballistic trajectory. If there appears to be a flyby opportunity, DEMONLP is then used to determine the trajectory which has the minimum launch energy. Many times trajectories are found which do not reduce the launch energy relative to ballistic missions, but do permit opportunities which are not available for ballistic missions. Gravity-assist trajectories often require longer interplanetary transfer times than ballistic opportunities.

Once a converged trajectory is found, the radius of closest approach during the actual flyby can be computed from

$$R_{ca} = \frac{\mu}{V_\infty^2} \left[\csc \frac{\psi}{2} - 1 \right] \qquad (7)$$

where V_∞ and ψ are the hyperbolic speed and flyby angle during the encounter, respectively.

During the actual solution of this problem, we find that the minimum launch energy is most sensitive to the flyby date and least sensitive to the arrival date at the destination planet. This indicates that the launch energy depends mainly on the first leg of the gravity-assisted trajectory. However, we would not want to eliminate the arrival date as a control variable. Remember, always give an optimization algorithm as much help as possible.

Curve Fit

ADDENDUM

Non-Linear Curve-Fitting

Curve–fitting data to a general analytic function can be useful in many areas of science and engineering. With a functional representation of our data, we can use elementary calculus to determine such things as the rate of change (first derivative) anywhere on the curve, points of minimum and maximum values (second derivative), and the area under the curve (integral) represented by the data. We can also use this function to accurately model our data in other computer simulations.

Program DEMONLF1.BAS is an interactive computer program which uses a combination of unconstrained minimization and least squares to curve fit data to any non–linear equation defined by the user. The user simply supplies a subroutine which defines the function and its gradient, the arrays of x and y data, and initial guesses for each fitting parameter.

The least squares function which we seek to minimize is defined by

$$J = \sum_{i=1}^{N} \left[f(x_i) - y_i \right]^2$$

where f is a user–defined non–linear *fitting* function and y_i is an array of N experimental data points. The N terms within the large brackets are called the *residuals*. These differences are an indication of how well the function fits each y data point. The scalar function J is called the *objective* or *cost* function.

The unconstrained minimization algorithm used in DEMONLF1 also requires the gradient of the objective function with respect to the N fitting parameters. Mathematically this is written as G = ∇J, and is simply the partial derivatives of the objective function with respect to each of the curve–fit parameters.

To illustrate the use of this algorithm for non–linear curve–fitting, we will fit the twelve x and y data points tabulated in the following data table:

Data Table

x	y
1	5.308
2	7.240
3	9.638
4	12.866
5	17.069
6	23.192
7	31.443
8	38.558
9	50.156
10	62.948
11	75.995
12	91.972

The non–linear fitting function we will use is given by:

$$f(b_1,b_2,b_3,x) = \frac{b_1}{1 + b_2 e^{x b_3}}$$

In this equation b_1, b_2, and b_3 are three fitting parameters we would like to calculate, and x represents an array of independent data.

Since we have three fitting parameters, we will also need the three components of the objective function gradient given by

$$G_1 = \frac{\partial J}{\partial b_1} = \sum_{i=1}^{N} 2f(x_i) \frac{1}{1 + b_2 e^{x_i b_3}}$$

$$G_2 = \frac{\partial J}{\partial b_2} = \sum_{i=1}^{N} -2f(x_i) \frac{b_1 e^{x_i b_3}}{\left(1 + b_2 e^{x_i b_3}\right)^2}$$

$$G_3 = \frac{\partial J}{\partial b_3} = \sum_{i=1}^{N} -2f(x_i) \frac{b_1 b_2 x_i e^{x_i b_3}}{\left(1 + b_2 e^{x_i b_3}\right)^2}$$

The QuickBASIC subroutine which calculates the least–squares function and its gradient must be called USERFUNC. For this example it is stored on the QuickPak Scientific disk in the file USRFUN1 and coded as follows:

```
SUB USERFUNC (NPTS, XDATA(), YDATA(), B(),
                                    FX, G()) STATIC

    ' Non–linear least squares function subroutine

    ' Input

    '  NPTS      = number of X and Y data points
    '  XDATA()   = array of X data points
    '  YDATA()   = array of Y data points
    '  B()       = array of fitting parameters

    ' Output

    '  FX  = least squares function value
    '  G() = array of gradient values

    DIM F(NPTS)

    B1 = B(1)
    B2 = B(2)
    B3 = B(3)

    ' evaluate non–linear least squares objective function

    FX = 0#
```

```
    FOR I = 1 TO NPTS
        F(I) = B1 / (1# + B2 * EXP(XDATA(I) * B3)) -
               YDATA(I)
        FX = FX + F(I) ^ 2
    NEXT I

    ' evaluate gradient of objective function

    G(1) = 0#
    G(2) = 0#
    G(3) = 0#

    FOR I = 1 TO NPTS
        G(1) = G(1) + 2# * F(I) * (1# / (1# + B2 *
               EXP(XDATA(I) * B3)))
        G(2) = G(2) - 2# * F(I) * (B1 * EXP(XDATA(I) *
               B3) / (1# + B2 * EXP(XDATA(I) * B3)) ^ 2)
        G(3) = G(3) - 2# * F(I) * (B1 * B2 * XDATA(I) *
               EXP(XDATA(I) * B3) / (1# + B2 *
               EXP(XDATA(I) * B3)) ^ 2)
    NEXT I

    ' erase working array

    ERASE F
END SUB
```

In order for the curve–fit algorithm to perform properly, the value of the objective function must be FX and the elements of the gradient must be loaded into the G array. You may need to add code which prevents such things as divide by zero and other calculations which can create errors during the evaluation of the function and its gradient.

The program will interactively prompt the user for several inputs. The software allows the user to interactively select either the Conjugate Gradient or Quasi–Newton method of minimization. The "best" method for solving your curve fit problem will depend on the fitting function you select and its data. Curve–fitting is a blend of art and science and you may want to experiment with both solution methods.

The elements of the data table are "hardwired" into the QuickBASIC code. It would be quite easy to modify the software to read the number of x and y data points and the actual data values from a disk file. You may want to add the capability to interactively prompt the user for this information.

The software will also prompt you for the maximum number of algorithm iterations allowed and a convergence criteria. 100 to 300 iterations are usually sufficient, and a convergence criteria between 1D–4 and 1D–8 is recommended. You may want to start with a larger convergence criteria and then decrease its value as you gain confidence that the algorithm is working properly.

Using the three initial guesses of 200, 30, and –.4 for the fitting parameters, the software calculates and prints the following results for this example:

Program DEMONLF1

Quasi-Newton Solution

Value of fitting parameter # 1	*196.1862*
Value of fitting parameter # 2	*49.09163*
Value of fitting parameter # 3	*–.3135697*
Minimized function value	*2.587277*
Number of iterations	*42*
Convergence criteria	*.00000001*

The minimization method used to solve the non-linear curve fit problem is a modified version of the MINIMIZ1 subroutine which is described on page 96. Please consult that section of the programmer's manual for additional information about using these types of algorithms.

The correct syntax for calling the subroutine which does all the hard work is

```
CALL MINIMIZ3 (METHOD, N, NPTS, XDATA(),
     YDATA(), EPS, MAXITER, IFLAG, NITER, F, X())
```

Where:

METHOD = method of solution [input]
- 1 = conjugate gradient
- 2 = quasi-Newton

N = number of variables [input]

NPTS = number of X and Y data points [input]

XDATA() = array of X data points [input]
(1 dimensional array; NPTS rows by 1 column)

YDATA() = array of Y data points [input]
(1 dimensional array; NPTS rows by 1 column)

EPS = convergence criteria [input]

MAXITER = maximum number of iterations [input]

IFLAG = diagnostic flag [input]
- 0 = converged
- 1 = maximum number of function evaluations
- 2 = linear search failure
- 3 = search vector failure

NITER = number of algorithm iterations [output]

F = final objective function value [output]

X() = input as the initial guess for the solution vector and output as the final solution vector