

The Network–Enabled Optimization System (NEOS) – a means of solving optimization problems over the internet

Max E. Jerrell
Northern Arizona Univeristy
70 Mc Connell Circle
Flagstaff, AZ 86011-5066, USA
max.jerrell@nau.edu

Wendy A. Campione
Northern Arizona Univeristy
70 Mc Connell Circle
Flagstaff, AZ 86011-5066, USA
wendy.campione@nau.edu

Abstract

Many optimization methods are available at the present time. The software that implements a particular method may not be available to all users or the software may require a compiler not readily available to all users. The software may require extensive modifications before it can run on a particular site. The Network–Enabled Optimization System (NEOS) is a system that has been designed to reduce these problems and make high quality optimization methods available to a large number of users. One of the goals of NEOS is to eliminate the need for the user to have extensive programming knowledge. Another goal is to make a large number of computer facilities available to the user. This goal is achieved by permitting optimization problems to be submitted to NEOS over the Internet. NEOS then directs one or more computers to solve the problem. NEOS also uses automatic differentiation. This relieves the user from coding expressions for the derivatives or from risking the approximation error inherent in numerical differentiation. A major accomplishment of the NEOS project is to take advantage of function partial separability when possible. Previously this had been considered quite difficult because of the difficulty in computing derivative information. NEOS use of automatic differentiation has eliminated this problem. This research presents an overview of NEOS and shows how it can be used to optimize econometric functions and other functions.

1 Introduction

Many numerical optimization methods are available at the present time and the number of good techniques continues to grow. This fact presents the potential user with both opportunities and problems. The opportunities are the greater number of techniques. This creates some difficulties, however. It is difficult for all but specialists to know of the existence of many methods much less the strengths and weaknesses a particular method.

Suppose that the user has selected a promising method, she is then faced with the task of obtaining and installing the software. The installation may or may not be difficult, depending on what modifications are needed to make the software work with a particular compiler and operating system. The user then must read the documentation to learn how write an interface routine(s) to the software and then she must write the interface code. This code can require sophisticated computer language skills. The chosen method will certainly require code to compute function values and in many cases the code needed to compute derivative values.

The Network-Enabled Optimization System (NEOS) was designed to reduce much of the user effort mentioned in the preceding paragraphs [7]. A large repository of information about optimization methods is maintained and continually updated at the NEOS site. This repository is discussed in Section 2.

NEOS also allows users access to some very high quality optimization software and permits them to compile and execute programs using NEOS machines. The designers of NEOS were particularly sensitive to problems that users face when adapting software and have made a concerted effort to provide an interface that is easy to use. Far easier than writing one's own interface code in most cases. NEOS can accept programs written in C, Fortran, AMPL, or GAMS.

Users submit problems to NEOS over the Internet. NEOS then directs one or more computers to work on the problem. It is NEOS's responsibility to provide the interface between the user's code and the remote computers. NEOS also uses automatic differentiation (AD) to compute accurate derivatives for programs written in C or Fortran. The AD code provided (ADIFOR, ADOL-C, and ADIC) is very sophisticated, but interfacing the AD routines with optimization software is quite challenging. NEOS handles the interface to the AD code removing this responsibility from the user. These details are discussed in Section 3 and examples are given in Section 4.

2 Locating optimization methods

A large and ever increasing number of optimization methods are now available. A non-specialist is apt to find it most difficult to keep current in the field. Reading journals in the field is possible but does not seem to be very practicable. A more practical alternative is to locate a document which specifically discusses a number of prominent techniques, such as the *Software Optimization Guide* by Moré and Wright [10]. Even the best guide of this sort will go out of date and, given the rapid advances in optimization methodology, this may happen quickly. An extensive list of optimization techniques, much of it taken from Moré and Wright's, guide is maintained at the NEOS web site at <http://www-neos.mcs.anl.gov/> as the NEOS Guide. The topics covered in the guide are linear programming, integer programming, network optimization, quadratic programming, unconstrained optimization, bound-constrained optimization, nonlinear programming, nonlinear least squares, global optimization, miscellaneous optimization problems, modeling languages and optimization systems, and software libraries for optimization.

Specific optimization methods are discussed for each of these categories. For example, the section on global optimization contains discussions on ASA (adaptive simulated annealing by Lester Inberger, free for academic use), global optimization for Mathematica, LGO (a Lipschitz global optimization package), MCS(multivariate coordinate search in Matlab), OptiA (a package that provides unconstrained and constrained optimization, quadratic programming, global optimization, nons-

mooth optimization, minimax optimization, multicriteria optimization), OPTECH (a commercial guided stochastic search method, available for reduced cost for academic use), QAPP (software for quadratic assignment problems).

In addition to the collection of optimization methods maintained in NEOS Guide, it also contains a number of case studies and examples of how NEOS can be used. Also available there is a preprint, *Optimization case studies in the NEOS guide* by Czyżyk, Wisniewski, and Wright [4], that gives an overview of some of these case studies and examples. This preprint discusses the Diet problem (a classical linear programming problem), a portfolio optimization problem, and a cutting-stock problem.

A recent case study describes how NEOS was able to solve a difficult quadratic assignment test problem [5] using computational pool of 2510 processors. The problem was solved in less than seven days using this pool where previously it had been thought the problems would require unreasonable amounts of processor time (in terms of years).

3 NEOS solvers

Users can submit jobs to NEOS in one of three ways: by e-mail, by using a Java Submission tool, or by using a Web server tool. Submission by e-mail is a bit primitive and does not provide the interactive environment the other tools do. The Web server tool requires a Unix interface which may not be available to many users. The Java Submission tool is available on any platform for which a Java compiler exists. This research was conducted using the Java Submission tool.

When the Java submission tool is started the user is presented with the screen shown in Figure 1. This screen shows the categories of optimization techniques available at the NEOS site. Clicking on one of the choices presents a second menu that lists the techniques available within that category. A list of the available methods by category is given in Table 1. NEOS uses the term solver to designate a technique. If the solver accepts GAMS or AMPL input it is so designated. If no particular input is indicated, the solver will accept C and/or Fortran input.

One of the great benefits of NEOS is that many of these solvers are able to use automatic differentiation (AD) programs to compute derivative values. Automatic differentiation eliminates the approximation error found in numerical differentiation methods. Thus AD can compute derivative values with the same precision as analytical expressions. AD computes these values automatically, as the name suggests, relieving the user of the effort of finding the analytic expressions by hand and then writing the code for the expressions [11]. Note that both these tasks tend to be error prone. There is also evidence that AD may be more efficient than the analytical expressions produced by symbolic algebra packages such as Mathematica [3]. An example of a problem that shows the benefits of AD is given in Section 4.1.3.

NEOS makes three AD packages available. ADOL-C is a C/C++ package available in the public domain [6]. ADIFOR (a Fortran package) [1] and ADIC (a C program) [2], both public domain AD packages developed by mathematicians and computer scientists at the Argonne National Labs. All three of these packages are recognized to be of the highest quality and can take advantage

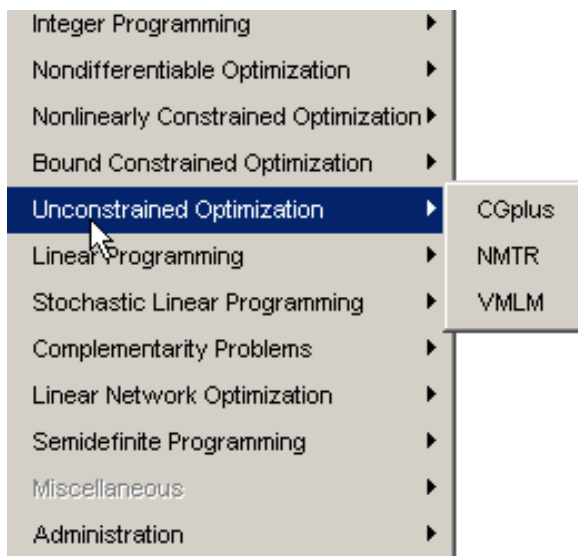


Figure 1: The initial NEOS screen showing the categories of techniques available

of problem structure, such as sparsity. All three packages are available for downloading from the Internet, but they also require good programming skills to interface with users programs. NEOS provides this interface in a manner that it is transparent to the user.

AD allows NEOS to take advantage of function partial separability. Let $x \in R^n$, then if $f(x)$ can be written as

$$f(x) = \sum_1^{n_f} f_i(x)$$

where each element function, f_i , only depends on a few components of x , then $f(x)$ is said to be partially separable with n_f elements. Gropp and Moré [7] report significant reduction in the number of function evaluations required for convergence if this separation is used with software capable of taking advantage of it. This reduction becomes greater the larger the size of the dimension of the problem n . To take advantage of this requires that the software be able to compute the function gradient and provide the separability structure. An example how a partially separable function can be submitted to NEOS is given in Section 4.1.1.

Problem type	Solvers
Integer programming	BonaisG(AMPL), BonsaiG(LP), BonsaiG, NINLP(AMPL), MIQP, XPRESS(AMPL), XPRESS-MP(mp-model), XPRESS-MP, INTEGER APPS
Nondifferentiable optimization	AMPL-PRO, CONOPT(GAMS), DONLP2(AMPL), FILTER(AMPL), LANCELOT(AMPL), LANCELOT, LOQO(AMPL), MINOS(AMPL), MINOS(GAMS), MOSEK(AMPL), NITRO(AMPLE), SNOPT(AMPL), SNOPT(GAMS) SNOPT
Nonlinearly constrained optimization	L-BFGS-B(AMPL), L-BFGS-B,
Bound constrained optimization	CGplus, NMTR, VMLM
LANCELOT, TRON(AMPL) TRON	BDMLP(GAMS), BDMPD(AMPL), BMDPD(LP), BPMPD, HOPDM, MOSEK(AMPL), PCx(AMPL), PCx, XPRESS(AMPL), XPRESS-MP(mp-model), XPRESS-MP/BARRIER, XPRESS-MP/SIMPLEX
Unconstrained optimization	MLES(GAMS), PATH(AMPL), PATH(GAMS), PATH
Linear programming	NETFLO, RELAX4
Complementarity problems	CSDP, DSDP, SDPA, SDPT3, SeDuMi
Linear network optimization	
Semidefinite programming	

Table 1: Solvers available at the NEOS site

4 Examples

4.1 Judge's function

Consider the hypothetical nonlinear least squares problem from Judge et al. [8], which has become something of a standard test problem for nonlinear econometric estimation methods. The problem is to find values of the parameters β_1 and β_2 that minimize

$$S(\beta) = \sum_{t=1}^n (y_t - \beta_1 - \beta_2 x_{1t} - \beta_2^2 x_{2t})^2$$

for generated data. Here β_1 and β_2 are considered to be intervals. This is an unconstrained optimization problem and the NEOS programs only require a subroutine to compute the value of $S(\beta)$ and a subroutine to specify the starting point. The input screen for solving this problem using the CGplus (conjugate gradient) technique is shown in Figure 2. This program requires a subroutine `fcn(n, x, f)` used to compute function values and a routine `initpt(n, x)` to provide a starting point to the optimization algorithm. The screen shows two radio buttons labeled *Function Only* and *Function and Gradient*. Clicking on the Function Only button instructs NEOS to use an automatic differentiation method. This research always uses automatic differentiation when available. The two required Fortran routines for Judge's function are shown next. The NEOS output for the problem is given in Appendix A.1.

```
subroutine fcn(n, x, f)
integer n
double precision f
double precision x(n)
double precision q(20), x1(20), x2(20)
integer nobs, i

data q /4.284, 4.149, 3.877, 0.533, 2.211, 2.389, 2.145,
c      3.231, 1.998, 1.379, 2.106, 1.428, 1.011, 2.179,
c      2.858, 1.388, 1.651, 1.593, 1.046, 2.152/

data x1 /0.286, 0.973, 0.384, 0.276, 0.973, 0.543, 0.957, 0.948,
c       0.543, 0.797, 0.936, 0.889, 0.006, 0.828, 0.399, 0.617,
c       0.939, 0.784, 0.072, 0.889/

data x2 /0.645, 0.585, 0.310, 0.058, 0.455, 0.779, 0.259, 0.202,
C       0.028, 0.099, 0.142, 0.296, 0.175, 0.180, 0.842, 0.039,
C       0.103, 0.620, 0.158, 0.704/

nobs = 20
f = 0.0d0
do 100 i = 1, nobs
  f = f + ( q(i) - x(1) - x(2)*x1(i) - (x(2)**2)*x2(i) )**2
100 continue
return
END

subroutine initpt (n, x)
```

Initial Point Subroutine	D:\Submit Client\Judge\fcn.f	Browse
Function Type	<input checked="" type="radio"/> Function Only <input type="radio"/> Function and Gradient	
Function Subroutine	D:\Submit Client\Judge\initp	Browse
Comments	<div style="border: 1px solid black; height: 30px; width: 100%;"></div>	

Figure 2: The input screen for solving Judge's function using CGplus

```

integer n
double precision x(n)
x(1) = 1.0d0
x(2) = 1.1d0
return
end

```

4.1.1 Example – by Rosen and Suzuki [12]

Consider the constrained nonlinear optimization problem

$$\min f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

with constraints

$$\begin{aligned} c_1(x) &= -2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geq 0 \\ c_2(x) &= -x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8 \geq 0 \\ c_3(x) &= -x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 100 \geq 0 \end{aligned}$$

NEOS requires five subroutines be provided to submit this problem to be worked on by the nonlinear constrained optimization solvers. The five subroutines are `fcn(x, n, nf, fvec)`, a routine used

to provide function values, `cfcn(n, x, m, c)` a routine to provide values of $c(m)$ constraints, `cbound(m, cl, cu)` a routine to place bounds on the constraints, `xbound(n, x)` to place bounds on the values the variables can have, and `initpt(n, x)` which provides a starting point.

The partially separable structure of the function is provided through the array `fvec` in the parameter list the function `fcfn`. The input screen for this problem is shown in Figure 3. This screen shows a box labeled *elements* which is where the dimension n_f of `fvec` is set. The screen also has a box that determines the number of constraint equations m . The routines needed for the problem are shown next.

NEOS provides a number of solvers with GAMS and MINOS input, but only two with Fortran input, Lancelot and SNOPT. ADIFOR, the Fortran automatic differentiation could not compile the constraint routine `cfcn` when invoked by the Lancelot solver. ADIFOR was able to compile the constraint routine when invoked by SNOPT.

SNOPT is maintained by research personnel at the University of California, San Diego and Stanford University. The package was written by Philip Gill (`pgill@ucad.edu`), Walter Murray (`Walter@SOL-Walter.Stanford.edu`), and Michael Saunders (`Mike@SOL-Michael.Stanford.edu`). These authors can be reached via e-mail for further information regarding the algorithm.

```

subroutine initpt(n, x)
  integer n
  double precision x(n)
  x(1) = 0.0d0
  x(2) = 0.0d0
  x(3) = 0.0d0
  x(4) = 0.0d0
  return
end

subroutine fcn(n, x, nf, fvec)
  integer n, nf
  double precision fvec(nf), x(n)
  fvec(1) = x(1)**2 - 5.0d0*x(1)
  fvec(2) = x(2)**2 - 5.0d0*x(2)
  fvec(3) = 2.0d0*x(3)**2 - 21.0d0*x(3)
  fvec(4) = x(4)**2 + 7.0d0*x(4)
  return
end

subroutine cfcn( n, x, m, c)
  integer m, n
  double precision x(4), c(3)

  c(1) = -2.0d0*x(1)**2 - x(2)**2 - x(3)**2
1      - 2.0d0*x(1) + x(2) + x(4) + 5.0d0
  c(2) = -x(1)**2 - x(2)**2 - x(3)**2 - x(4)**2
1      - x(1) + x(2) -x(3) + x(4) + 8.0d0
  c(3) = -x(1)**2 - 2.0d0*x(2)**2 - x(3)**2 -2.0d0*x(4)**2
1      + x(1) + x(4) + 10.0d0
  return

```

end

```
subroutine cbound(m,cl,cu)
  integer m
  double precision cl(m), cu(m)
  cl(1) = 0.0d0
  cl(2) = 0.0d0
  cl(3) = 0.0d0
  return
end
```

```
subroutine xbound(n,xl,xu)
  integer n
  double precision xl(n), xu(n)

  return
end
```

4.1.2 A GAMS example

The following is an example of how NEOS can accept GAMS input. The example is taken from the NEOS site and is described in the GAMS program documentation as shown below. This is a constrained optimization problem is solved here using the SNOPT optimization program.

```
$TITLE AREA OF HEXAGON TEST PROBLEM (HIMMEL16,SEQ=36)
* THE PHYSICAL PROBLEM IS TO MAXIMIZE THE AREA OF A HEXAGON IN WHICH THE
* DIAMETER MUST BE LESS THAN OR EQUAL TO ONE. THE FORMULATION IN HIMMELBLAU
* IS DIFFERENT FROM THE ONE GIVEN HERE, BECAUSE CERTAIN FIXED VARIABLES
* HAVE BEEN ELIMINATED. HOWEVER, THE FORMULATION GIVEN HERE IS MORE NATURAL
* AND EASIER TO UNDERSTAND. IT MAKES USE OF THE FACT THAT ONE VERTEX IS FIXED
* AT THE ORIGIN AND USES THE VECTOR SCALAR PRODUCT TO CALCULATE THE AREAS OF
* ALL TRIANGLES ORIGINATING FROM THE ORIGIN. ALL TERMS IN X(1) AND Y(1) THUS
* VANISH WHEN THE ALGEBRAIC EXPRESSSION IS SIMPLIFIED.
*
* THE PROBLEM APPEARS MANY OTHER PLACES, E.G. AS EXAMPLE 108 IN W. HOCK AND
* K. SCHITTKOWSKI: TEST EXAMPLES FOR NONLINEAR PROGRAMMING CODES, LECTURE
* NOTES IN ECONOMICS AND MATHEMATICAL SYSTEMS, 187, SPRINGER VERLAG, 1981,
* AND AS THE TEST EXAMPLE IN P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND
* M. WRIGHT: USER'S GUIDE FOR SOL/NPSOL: A FORTRAN PACKAGE FOR NONLINEAR
* PROGRAMMING, TECH. REP. 83-12, DEPT. OF OPERATION RESEARCH, STANFORD
* UNIVERSITY.
*
* REFERENCE: HIMMELBLAU D M, APPLIED NONLINEAR PROGRAMMING, MCGRAW HILL,
* NEW YORK, 1972. PROBLEM NO 16.

SET I INDICES FOR THE 6 POINTS /1*6/;
ALIAS (I,J);
VARIABLE X(I) X-COORDINATES OF THE POINTS
         Y(I) Y-COORDINATES OF THE POINTS
```

File Help	
Type of Input	<input checked="" type="radio"/> Fortran <input type="radio"/> SIF
Number of Variables	4
Number of Constraints	3
Number of Element Functions	4
Initial Point Subroutine	C:\SubmitClient\Suzuki\initp Browse
Function Subroutine	C:\SubmitClient\Suzuki\fcn.fi Browse
Constraint Subroutine	C:\SubmitClient\Suzuki\cfcn. Browse
Bounds on Variables Subroutine	C:\SubmitClient\Suzuki\vbou Browse
Bounds on Constraints Subroutine	C:\SubmitClient\Suzuki\cbou Browse
Options specification file	Browse
SIF file	Browse
Comments	
<input type="button" value="Submit to NEOS"/> <input type="button" value="Close"/>	

Figure 3: The input screen for solving Rosen and Suzuki's problem using SNOPT

```

        AREA(I)  AREA OF THE I'TH TRIANGLE ( 0 -> P(I) -> P(I++1) -> 0
        TOTAREA  TOTAL AREA OF THE HEXAGON

EQUATIONS  AREADEF(I)  AREA DEFINITION FOR TRIANGLE I.
           MAXDIST(I,J)  MAXIMAL DISTANCE BETWEEN I AND J
           OBJ1          FIRST DEFINITION OF OBJECTIVE
           OBJ2          SECOND DEFINITION OF OBJECTIVE;

MAXDIST(I,J)$ (ORD(I) LT ORD(J)).. SQR(X(I)-X(J))+SQR(Y(I)-Y(J)) =L= 1;

AREADEF(I).. AREA(I) =E= 0.5*(X(I)*Y(I++1)-Y(I)*X(I++1)) ;

OBJ1..      TOTAREA =E= 0.5*SUM(I,X(I)*Y(I++1)-Y(I)*X(I++1));

OBJ2..      TOTAREA =E= SUM(I,AREA(I));

MODEL SMALL /MAXDIST,OBJ1/
        LARGE /MAXDIST,OBJ2,AREADEF/ ;

*
*  INITIAL CONDITIONS
*
X.FX("1") = 0; Y.FX("1") = 0; Y.FX("2") = 0;
X.L("2") = 0.5; X.L("3") = 0.5; X.L("4") = 0.5;
X.L("5") = 0; X.L("6") = 0;
Y.L("3") = 0.4; Y.L("4") = 0.8; Y.L("5") = 0.8;
Y.L("6") = 0.4;

*
*syntax for setting GAMS options
*
option iterlim=50;

SOLVE LARGE USING NLP MAXIMIZING TOTAREA;

```

And the options file

```

BEGIN.OPT
*Only solver-specific options here*

Feasible exit on

END.OPT

```

4.1.3 Example - the ordinal probit model

One of the virtues of automatic differentiation occurs when the function is relatively simple but the derivatives are very complex. We need only provide the simple expression of the function and let the process deal with the complicated derivatives. The response function of the ordinal probit model can be written as

$$Y_i = \beta' x_i + u_i,$$

where x_i is a vector of characteristics, Y_i is the response variable, and u_i is an error term. Suppose there are m categories and the response variable fits into the j -th category if $\alpha_{j-1} \leq Y \leq \alpha_j$. See Maddala [9] for a more complete description of the problem and for the derivation of the likelihood function, gradient, and Hessian that are used below.

The log of the maximum likelihood function for the categorical probit model can be written as

$$L = \sum_{i=1}^n \sum_{j=1}^m Z_{ij} \log[\Phi(\alpha_j - \beta'x_i) - \Phi(\alpha_{j-1} - \beta'x_i)],$$

where $Z_{ij} = 1$ if the i -th observation falls in the j -th category and $Z_{ij} = 0$ otherwise. The summations run over m categories and n observations. The parameters to be estimated are the vectors α and β . $\Phi(\cdot)$ is the cumulative standard normal probability distribution function and $\phi(x) = \frac{\partial \Phi}{\partial x}$. The function is relatively simple, but the derivatives are quite complex. In what follows let $Y_{i,j} = \alpha_j - \beta'x_i$, $\phi_{i,j} = \phi(\alpha_j - \beta'x_i)$, and $\Phi_{i,j} = \Phi(\alpha_j - \beta'x_i)$. It will also be useful to let $\delta_{j,k} = 1$ if $j = k$ and $\delta_{j,k} = 0$ otherwise. The gradient and Hessian are

$$\begin{aligned} \frac{\partial L}{\partial \beta} &= \sum_{i=1}^n \sum_{j=1}^m Z_{ij} \frac{\phi_{i,j-1} - \phi_{i,j}}{\Phi_{i,j} - \Phi_{i,j-1}} x_i \\ \frac{\partial L}{\partial \alpha_k} &= \sum_{i=1}^n \sum_{j=1}^m Z_{ij} \frac{\delta_{j,k} \phi_{i,j} - \delta_{j-1,k} \phi_{i,j-1}}{\Phi_{i,j} - \Phi_{i,j-1}} \\ \frac{\partial^2 L}{\partial \beta \partial \beta'} &= \sum_{i=1}^n \sum_{j=1}^m \frac{Z_{ij}}{(\Phi_{i,j} - \Phi_{i,j-1})^2} \\ &\quad \times [(\Phi_{i,j} - \Phi_{i,j-1})(Y_{i,j-1} \phi_{i,j-1} - Y_{i,j} \phi_{i,j}) - (\phi_{i,j-1} - \phi_{i,j})^2] x_i x_i' \\ \frac{\partial^2 L}{\partial \beta \partial \alpha_k} &= \sum_{i=1}^n \sum_{j=1}^m \frac{Z_{ij}}{(\Phi_{i,j} - \Phi_{i,j-1})^2} \\ &\quad \times [(\Phi_{i,j} - \Phi_{i,j-1})(Y_{i,j} \phi_{i,j} \delta_{j,k} - Y_{i,j-1} \phi_{i,j-1} \delta_{j-1,k}) \\ &\quad - (\phi_{i,j-1} - \phi_{i,j})(\phi_{i,j} \delta_{j,k} - \phi_{i,j-1} \delta_{j-1,k})] x_i \\ \frac{\partial^2 L}{\partial \alpha_k \partial \alpha_l} &= \sum_{i=1}^n \sum_{j=1}^m \frac{Z_{ij}}{(\Phi_{i,j} - \Phi_{i,j-1})^2} \\ &\quad \times [(\Phi_{i,j} - \Phi_{i,j-1})(Y_{i,j-1} \phi_{i,j-1} \delta_{j-1,k} \delta_{j-1,l} - Y_{i,j} \phi_{i,j} \delta_{j,k} \delta_{j,l}) \\ &\quad - (\phi_{i,j} \delta_{j,k} - \phi_{i,j-1} \delta_{j-1,k})(\phi_{i,j} \delta_{j,l} - \phi_{i,j-1} \delta_{j-1,l})] \end{aligned}$$

This problem illustrates some important benefits of the automatic differentiation capability found in the NEOS solvers. The problem requires the cumulative normal distribution function and its derivatives. The cumulative normal distribution function is not an intrinsic function found in most compilers. NEOS permits the specification of external functions (the complementary error function `Erfc` used to compute the cumulative normal which is given in the code below) and provides the interface so that the AD routines can compute the derivative information in such a manner that requires no additional effort by the user.

The second thing to notice is that while the function itself is quite simple, the expressions for the derivatives are quite complex. One suspects that most people would find the mathematics oppressive, time consuming, and error prone. Converting these expressions to error free computer code would also be laborious. AD eliminates all of these problems—all the user need do is to write the expression to compute the function value. The code for the problem follows.

THE NEOS FUNCTIONS

```

subroutine initpt(n, x)
integer n
double precision x(n)
  x(1) = -2.0d-02
  x(2) = 1.2d0
  x(3) = 0.0d-02
  x(4) = 0.0d-02
  x(5) = 0.0d-02
end

```

```

subroutine fcn (n, x, f)
integer n
double precision x(n), f, arg, L
double precision Erfc
integer y(100)
double precision xobs1(100), xobs2(100), xobs3(100)
integer i, nobs

```

```

data y /1,2,1,2,1,1,1,1,3,3,1,1,1,1,1,1,3,1,3,2,2,1,
1 3,1,1,3,2,1,2,2,3,2,1,1,1,2,1,3,1,1,2,3,2,1,
2 1,1,1,3,2,2,1,3,1,3,2,1,1,3,1,1,1,3,1,3,2,1,
3 3,3,3,1,3,2,1,1,2,2,1,1,3,2,1,1,2,1,3,2,1,1,
4 3,2,2,3,3,1,1,2,3,2,1,1/

```

```

data xobs1 /-3.70D-01,3.10D-01,3.40D-01,4.50D-01,-3.30D-01,
1 4.40D-01,2.70D-01,-4.70D-01,-4.90D-01,-2.40D-01,-1.10D-01,
2 2.40D-01,-3.00D-02,-2.00D-01,-2.70D-01,-2.60D-01,2.40D-01,
3 -2.60D-01,-3.50D-01,2.70D-01,-1.10D-01,-1.60D-01,-4.10D-01,
4 4.40D-01,-1.40D-01,5.00D-02,1.90D-01,1.90D-01,4.90D-01,
5 3.80D-01,4.30D-01,-1.00D-01,-2.30D-01,4.90D-01,3.70D-01,
6 2.80D-01,4.60D-01,-1.60D-01,4.40D-01,-2.20D-01,1.00D-02,
7 -1.40D-01,8.00D-02,-1.30D-01,-4.30D-01,-2.80D-01,4.90D-01,
8 3.80D-01,3.00D-01,2.80D-01,2.00D-01,2.20D-01,-2.00D-01,
9 -2.00D-01,4.70D-01,3.20D-01,3.50D-01,-4.50D-01,2.80D-01,
1 -3.40D-01,-8.00D-02,2.10D-01,-3.70D-01,-1.00D-01,-4.60D-01,
2 1.50D-01,-4.60D-01,4.70D-01,2.90D-01,1.10D-01,4.70D-01,
3 1.00D-01,-7.00D-02,-1.30D-01,2.10D-01,2.80D-01,-5.00D-02,
4 -6.00D-02,2.40D-01,-1.40D-01,2.20D-01,-1.50D-01,-1.90D-01,
5 7.00D-02,3.00D-01,5.00D-01,1.40D-01,-1.70D-01,4.70D-01,
6 -1.30D-01,-1.10D-01,4.70D-01,6.00D-02,2.70D-01,5.00D-02,
7 -3.40D-01,-2.60D-01,-2.50D-01,3.50D-01,4.40D-01/

```

```

data xobs2 /-1.00D-01,2.80D-01,-1.40D-01,-9.00D-02,3.00D-02,
1 -4.20D-01,-3.20D-01,-3.30D-01,4.40D-01,2.60D-01,-2.70D-01,
2 -8.00D-02,4.00D-02,-9.00D-02,-3.80D-01,-3.70D-01,4.70D-01,
3 -4.70D-01,1.00D-02,1.40D-01,9.00D-02,1.20D-01,2.80D-01,
4 -4.90D-01,-5.00D-02,4.50D-01,1.90D-01,-1.80D-01,1.60D-01,
5 2.70D-01,4.80D-01,-1.50D-01,-4.80D-01,-2.60D-01,-3.10D-01,
6 1.40D-01,-1.30D-01,3.80D-01,1.00D-02,-1.20D-01,4.60D-01,
7 1.20D-01,-3.20D-01,-3.30D-01,1.90D-01,-4.50D-01,-2.00D-02,
8 4.70D-01,4.00D-01,3.10D-01,-4.00D-01,2.90D-01,-2.30D-01,
9 4.40D-01,-1.10D-01,-1.60D-01,0.00D+00,3.40D-01,-7.00D-02,
1 -1.20D-01,-4.50D-01,1.40D-01,-1.60D-01,2.20D-01,2.10D-01,
2 -3.70D-01,4.60D-01,3.20D-01,5.00D-01,1.00D-02,3.20D-01,
3 1.20D-01,-3.40D-01,-1.50D-01,3.30D-01,1.70D-01,-6.00D-02,
4 -2.20D-01,4.50D-01,1.10D-01,-3.50D-01,-2.50D-01,1.00D-02,
5 -3.70D-01,4.70D-01,1.50D-01,4.00D-02,-2.40D-01,4.80D-01,
6 3.00D-02,3.00D-01,2.60D-01,9.00D-02,-1.50D-01,-4.50D-01,
7 -7.00D-02,4.50D-01,8.00D-02,-3.70D-01,-7.00D-02/

```

```

data xobs3 /8.00D-02,4.80D-01,-6.00D-02,-1.90D-01,1.80D-01,
1 1.80D-01,4.80D-01,-4.40D-01,-4.30D-01,-3.60D-01,-2.00D-02,
2 -1.00D-01,2.90D-01,2.50D-01,-1.40D-01,2.80D-01,1.10D-01,
3 9.00D-02,-4.40D-01,-1.90D-01,1.40D-01,3.20D-01,2.20D-01,
4 2.90D-01,-1.00D-01,1.10D-01,3.80D-01,2.80D-01,2.70D-01,
5 3.70D-01,2.70D-01,-2.90D-01,1.80D-01,2.00D-02,1.70D-01,
6 1.60D-01,-1.00D-01,-3.00D-01,4.00D-01,3.30D-01,4.70D-01,
7 -2.20D-01,-3.50D-01,-1.10D-01,2.50D-01,2.90D-01,3.10D-01,
8 2.90D-01,3.20D-01,3.50D-01,1.80D-01,-4.00D-02,-1.50D-01,
9 8.00D-02,6.00D-02,-2.60D-01,3.10D-01,-3.10D-01,1.60D-01,
1 1.50D-01,2.60D-01,-2.80D-01,-2.90D-01,1.10D-01,-3.70D-01,
2 1.10D-01,-3.70D-01,-4.10D-01,4.00D-02,5.00D-01,-4.00D-01,
3 -1.00D-01,-1.70D-01,-9.00D-02,-1.00D-02,8.00D-02,3.90D-01,
4 -2.80D-01,-1.70D-01,-1.10D-01,-1.90D-01,4.00D-02,-4.00D-01,
5 1.00D-01,-1.30D-01,3.50D-01,4.20D-01,-4.70D-01,4.30D-01,
6 -1.90D-01,1.70D-01,-2.40D-01,-3.20D-01,3.50D-01,4.20D-01,
7 -3.30D-01,-9.00D-02,2.00D-01,1.20D-01,2.00D-01/

```

```
nobs = 100
```

```
L = 0.0d0
```

```
do 100 i = 1, nobs
```

```
arg = x(3)*xobs1(i) + x(4)*xobs2(i) + x(5)*xobs3(i)
```

```
if (y(i) .EQ. 1) then
```

```
L = L - dlog(0.5d0*Erfc(-(x(1) - arg)/dsqrt(2.0d0)) )
```

```
else if (y(i) .EQ. 2) then
```

```
L = L - dlog(0.5d0*Erfc(-(x(2) - arg)/dsqrt(2.0d0)) )
```

```
1 - 0.5d0*Erfc(-(x(1) - arg)/dsqrt(2.0d0)) ) )
```

```
else
```

```
L = L - dlog(1.0d0 - 0.5*Erfc(-(x(2) - arg)/dsqrt(2.0d0)) ) )
```

```
endif
```

```
100 continue
```

```
f = L
```

```
end
```

```
double precision function Erfc (x)
```

```
double precision x
```

```
double precision z, t, temp, res
```

```
/* the Erfc code is taken from W. H. Press, S. A. Teukolsky,
```

W. T. Vetterling, and B. P. Flannery,
 Numerical Recipes in C - The Art of Scientific Computing, 2ed.,
 Cambridge University Press, 1994

```

*/
z = abs(x)
t = 1.0d0/(1.0d0 + 0.5d0*z)
temp = t*exp(-z*z - 1.26551223d0
c      + t*(1.00002368d0
c      + t*(0.37409196d0
c      + t*(0.09678418d0
c      + t*(-0.18628806d0
c      + t*(0.27886807d0
c      + t*(-1.13520398d0
c      + t*(1.48851587d0
c      + t*(-0.82215223d0
c      + t*0.17087277d0 ))))))) )

if (x .lt. 0) temp = 2.0d0 - temp

Erfc = temp

return
END

```

To test this we created $n = 100$ observations using the ordinal probit model with five parameters. Table 2 shows the parameter values used in creating the data. The table also shows the estimates that this data has given using AD software we have written and used in prior research. The answers NEOS gives are identical the estimates produced by our own code. The NEOS output is shown in Appendix A.3.

parameter	value	estimate
α_1	0.0	-0.149
α_2	3.0	2.74
β_1	1.0	1.03
β_2	10.0	10.91
β_3	-5.0	-5.41

Table 2: Results for the ordered probit model

5 Conclusions

NEOS can be a very useful resource for numerical optimization. A large repository of information concerning optimization methods is maintained at the NEOS site. NEOS also makes good optimization software available and does so in a manner that is user friendly. Potential users can experiment with various optimization software to see if it will meet their needs. Finally, NEOS should provide an excellent teaching tool that can be used as an easy introduction to optimization methods.

References

- [1] C. Bischof, A. Carle, P. Khademi, and A. Mauer. The ADIFOR2.0 System for the Automated Differentiation of Fortran 77 Programs, Argonne Preprint ANL-MCS-P481-1194, and CRPC Technical Report CRPC-TR94491.
- [2] C. Bischof, L. Roh, and A. Mauer. Adic – an extensible automatic differentiation tool for ansi-c. Argonne Preprint ANL/MCS-P626-1196.
- [3] George C. Corliss. Application of differentiation arithmetic. In Ramon E. Moore, editor, *Reliability in Computing*, volume 19 of *Perspectives in Computing*, pages 127–148. Academic Press, Boston, 1988.
- [4] Joseph Czyzyk, Timothy Wisniewski, and Stephen J. Wright. Optimization case studies in the neos guide, July 1998. Preprint ANL/MCS-P704-0198, January 1998 (Revised July 1998), Mathematics and Computer Science Division, Argonne National Laboratory, 9700 Cass Ave., Argonne, IL 60439.
- [5] Nugent C. E., T. E. Vollman, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16:150–173, 1965.
- [6] A. Griewank, D. Juedes, and J. Utke. Adol-c: A package for the automatic differentiation of algorithms written in c/c++”. *ACM Transactions on Mathematical Software*, 23:131–167, 1996.
- [7] W. Gropp and J. J. Moré. Optimization environments and the neos server. In M. D. Buhman and A. Iserles, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 167–182. Cambridge University Press, Cambridge, 1997.
- [8] G. G. Judge, R. C. Hill, W. E. Griffiths, H. Lütkepohl, and T. Lee. *Introduction to the Theory and Practice of Econometrics*. John Wiley & Sons, New York, 1982.
- [9] G. S. Maddala. *Limited-Dependent and Qualitative Variables in Econometrics*. Cambridge University Press, Cambridge, 1983.
- [10] Jorge Moré and Stephen J. Wright. *Software Optimization Guide*. SIAM, Philadelphia, 1993.
- [11] L. B. Rall. *Automatic differentiation - Techniques and Applications*, volume 120 of *Springer Lecture Notes in Computer Science*. Springer-Verlag, New York, 1981.
- [12] J. B. Rosen and S. Suzuki. Construction of nonlinear programming test problems. *CACM*, 8:113, 1965.

Appendix

Appendix A.1 Output for Judge's function

NEOS Output

*** Conjugate Gradient method ***

Number of variables	2
Initial function value	1.63076007D+01
Final function value	1.60817301D+01
Initial gradient 2 norm	5.95024656D+00
Final gradient 2 norm	2.33618671D-04
Number of iterations	6
Number of function evaluations	15
Number of gradient evaluations	15
Total time	9.99699906D-03

The solution is:

index	x(index)
1	8.64823428D-01
2	1.23572675D+00

Appendix A.2: Output for the GAMS example

You chose to use GAMS SNOPT solver for nonlinearly constrained optimization problems
 GAMS Rev 117 Linux/Intel 0201 12:19:55 PAGE 1
 AREA OF HEXAGON TEST PROBLEM (HIMMEL16,SEQ=36)

COMPILATION TIME = 0.000 SECONDS 0.7 Mb LNX195-117
 GAMS Rev 117 Linux/Intel 0201 12:19:55 PAGE 2
 AREA OF HEXAGON TEST PROBLEM (HIMMEL16,SEQ=36)
 Model Statistics SOLVE LARGE USING NLP FROM LINE 64

MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	22
BLOCKS OF VARIABLES	4	SINGLE VARIABLES	19
NON ZERO ELEMENTS	97	NON LINEAR N-Z	84
DERIVATIVE POOL	7	CONSTANT POOL	9
CODE LENGTH	616		

GENERATION TIME = 0.000 SECONDS 1.8 Mb LNX195-117

EXECUTION TIME = 0.000 SECONDS 1.8 Mb LNX195-117
 GAMS Rev 117 Linux/Intel 0201 12:19:55 PAGE 3
 AREA OF HEXAGON TEST PROBLEM (HIMMEL16,SEQ=36)

S O L V E S U M M A R Y

MODEL	LARGE	OBJECTIVE	TOTAREA
TYPE	NLP	DIRECTION	MAXIMIZE
SOLVER	SNOPT	FROM LINE	64

**** SOLVER STATUS 1 NORMAL COMPLETION
 **** MODEL STATUS 2 LOCALLY OPTIMAL
 **** OBJECTIVE VALUE 0.6750

RESOURCE USAGE, LIMIT	0.020	1000.000
ITERATION COUNT, LIMIT	41	50
EVALUATION ERRORS	0	0

SNOPT-Link Oct 16, 2000 LNX.SN.SN 19.5 017.041.039.LNX SNOPT 5.3-5(2)

GAMS/SNOPT, Large Scale Nonlinear SQP Solver
 S N O P T 5.3-5(2) (Sep 1999)
 P. E. Gill, UC San Diego

W. Murray and M. A. Saunders, Stanford University

Work space allocated -- 0.04 Mb

EXIT - Optimal Solution found.

Major, Minor Iterations	26	41
Funobj, Funcon calls	31	31
Superbasics	3	
Aggregations	6	
Interpreter Usage	0.00	0.0%

Work space used by solver -- 0.03 Mb

---- EQU MAXDIST MAXIMAL DISTANCE BETWEEN I AND J

	LOWER	LEVEL	UPPER	MARGINAL
1.2	-INF	0.122	1.000	.
1.3	-INF	0.473	1.000	.
1.4	-INF	1.000	1.000	0.160
1.5	-INF	1.000	1.000	0.042
1.6	-INF	0.312	1.000	.
2.3	-INF	0.122	1.000	.
2.4	-INF	0.607	1.000	.
2.5	-INF	1.000	1.000	0.172
2.6	-INF	0.607	1.000	.
3.4	-INF	0.312	1.000	.
3.5	-INF	1.000	1.000	0.042
3.6	-INF	1.000	1.000	0.160
4.5	-INF	0.412	1.000	.
4.6	-INF	1.000	1.000	0.100
5.6	-INF	0.412	1.000	.

	LOWER	LEVEL	UPPER	MARGINAL
---- EQU OBJ2	.	.	.	1.000

OBJ2 SECOND DEFINITION OF OBJECTIVE

---- EQU AREADEF AREA DEFINITION FOR TRIANGLE I.

	LOWER	LEVEL	UPPER	MARGINAL
1	.	.	.	1.000
2	.	.	.	1.000
3	.	.	.	1.000
4	.	.	.	1.000
5	.	.	.	1.000
6	.	.	.	1.000

---- VAR X X-COORDINATES OF THE POINTS

	LOWER	LEVEL	UPPER	MARGINAL
--	-------	-------	-------	----------

1	.	.	.	EPS
2	-INF	0.349	+INF	5.4893E-7
3	-INF	0.677	+INF	8.9698E-8
4	-INF	0.737	+INF	.
5	-INF	0.175	+INF	.
6	-INF	-0.248	+INF	.

---- VAR Y Y-COORDINATES OF THE POINTS

	LOWER	LEVEL	UPPER	MARGINAL
1	.	.	.	7.7178E-7
2	.	.	.	EPS
3	-INF	0.120	+INF	.
4	-INF	0.676	+INF	.
5	-INF	0.985	+INF	.
6	-INF	0.501	+INF	2.3778E-7

---- VAR AREA AREA OF THE I'TH TRIANGLE (0 -> P(I) -> P(I++1) -> 0

	LOWER	LEVEL	UPPER	MARGINAL
1	-INF	.	+INF	EPS
2	-INF	0.021	+INF	EPS
3	-INF	0.185	+INF	EPS
4	-INF	0.304	+INF	EPS
5	-INF	0.166	+INF	EPS
6	-INF	.	+INF	EPS

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR TOTAREA	-INF	0.675	+INF	.

TOTAREA TOTAL AREA OF THE HEXAGON

**** REPORT SUMMARY :

0	NONOPT
0	INFEASIBLE
0	UNBOUNDED
0	ERRORS

EXECUTION TIME = 0.000 SECONDS 0.7 Mb LNX195-117

USER: MCS Division Argonne National Labs G000517:0709CS-LNX DC2747

**** FILE SUMMARY

INPUT ./MODEL
OUTPUT ./solve.out

Appendix A.3: Output for the ordinal probit model

Language:
User routines taken in FORTRAN

Function type:
Function provided without gradient

Your comments

Polycat.f program for publication ***NEOS Output***

*** Conjugate Gradient method ***

Number of variables	5
Initial function value	1.12871515D+02
Final function value	3.22197990D+01
Initial gradient 2 norm	4.24075674D+01
Final gradient 2 norm	4.32706490D-04
Number of iterations	20
Number of function evaluations	44
Number of gradient evaluations	44
Total time	1.09966996D+00

The solution is:

index	x(index)
1	-1.49321648D-01
2	2.74455726D+00
3	1.03342779D+00
4	1.09102090D+01
5	-5.40759463D+00