CMAT Newsletter: July 2008, Part II

Wolfgang M. Hartmann

July 2008

Contents

1 Illustrations

The section *Fitting Quantal Response Models* is based on the work of Ying So and it was presented at our tutorial at the 1995 ASA joint meeting in Orlando. At the same tutorial I presented the application from section *Diffusion of Tetracycline Hydrochloride* using the software PROC NLP in SAS/OR and the NLP functions in SAS/IML.
The CMAT code of section *Neural Nets with One Hidden Layer* is based on a publicly available SAS macro by Warren Sarle.

1.1 ML Estimation: Examples by Clarke (1987)

1.1.1 Mitcherlitz Equation on Pattinson Data

The model in Clarke (1987, p.224) is

$$f(x_i, \theta) = \theta_3 + \theta_2 \exp(\theta_1 x_i)$$

The following are the Pattinson (1981) data used in Clarke (1987):

```
print "G.P.Y. Clarke (1987): Mitcherlitz Equation";
mitch = [ 1   3.183,  2   3.059,  3   2.871,  4   2.622,
          5   2.541,  6   2.184,  7   2.110,  8   2.075,
          9   2.018, 10   1.903, 11   1.770, 12   1.762,
         13   1.550 ];
cnam = [" x y "];
mitch = cname(mitch,cnam);
```

We specify the negative log-likelihood function:

$$l(\theta) = \log(\sigma) + \frac{1}{2}(\frac{x - \mu(\theta)}{\sigma})^2, \quad \text{with} \quad \mu(\theta) = \theta_3 + \theta_2 \exp(\theta_1 x_i)$$

The `nlp` specification follows:

```
function fmitchml(x) global(mitch) {
  mu = x[4] + x[3] * exp(x[2] * mitch[,1]);
  loglik = log(x[1]) + .5*((mitch[,2] - mu)/x[1]).**2;
  f = loglik[+];
  return(f);
}

bc = cons(4,2,.); cons[1,1] = 1.e-12;
x0 = [ .1 -.1 1. 1. ];
pnam = [" sigma th1 th2 th3 "];
x0 = cname(x0,pnam);

print "Trust-Region Algorithm: ML";
mopt = [ "tech"   "trureg" ,
         "print"         3 ];
< xr,rp,der1,der2 > = nlp(fmitchml,x0,mopt,bc);
print "Xopt=",xr;     print "RP=",rp;
print "DER1=", der1;  print "DER2=", der2;
```

The iteration history follows:

```
                    Trust Region Optimization
                    Without Parameter Scaling
               Gradient Computed by Finite Differences
          Hessian Computed by Finite Differences (dense)
                    (Using Only Function Calls)

 Iteration Start:
 N. Variables             4
 Criterion       397.0676620        Max Grad Entry  8410.023279
 TR Radius       1.000000000


Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
   1*   0    3   0  142.6687 254.3989 3019.21 25105.7  0.10003
   2*   0    4   0  60.46836 82.20037 1183.90 1243.25  0.10512
   3*   0    5   0  19.98902 40.47934 452.177 656.187  0.11171
   4*   0    6   0  0.827850 19.16117 164.037 376.798  0.12132
   5*   0    7   0 -7.978590 8.806441 52.1365 243.386  0.12797
   6*   0   11   0 -17.97489 9.996298 66.2129 23.9794  1.52490
   7*   0   12   0 -20.18709 2.212202 378.072 67.2497  0.38122
   8*   0   14   0 -24.45118 4.264089 133.827 598.295  0.03849
   9    0   15   0 -25.89750 1.446317 37.2613 314.506  0.04031
  10    0   19   0 -27.32614 1.428646 65.0396 8.13024  0.31280
  11    0   20   0 -28.27502 0.948878 47.5394 33.5743  0.12968
  12    0   21   0 -28.80040 0.525380 253.589 0.00000  0.28200
  13    0   22   0 -29.14097 0.340567 8.83469 0.00000  0.19969
  14    0   23   0 -29.19834 0.057372 33.5668 3.27322  0.07217
  15    0   24   0 -29.20972 0.011385 5.04417 0.00000  0.07252
  16    0   25   0 -29.21029 5.7e-004 0.56264 0.00000  0.03503
  17    0   26   0 -29.21029 2.4e-006  2e-003 0.00000  0.01066
  18    0   27   0 -29.21029 3.0e-010  3e-006 0.00000   6e-004

 Successful Termination After    18 Iterations
 GCONV convergence criterion satisfied.
 Criterion      -29.21029458        Max Grad Entry  2.9127e-006
 Ridge (lambda)  0.000000000        TR Radius       0.000624645
 Act.dF/Pred.dF  4.302549508
 N. Function Calls        28        N. Gradient Calls         2
 N. Hessian Calls         20        Preproces. Time           0
 Time for Method           0        Effective Time            0
```

The tables of parameter estimates and the Hessian matrix follow:

```
                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------
```

```
     Parameter     Estimate   Gradient

      1 sigma    0.06412340  2.91e-006
      2 th1     -0.10305545 -2.16e-006
      3 th2      2.51899317 -6.78e-008
      4 th3      0.96312800 -1.21e-007

Value of Objective Function =     -29.2103

                Hessian Matrix
                **************

      Symmetric Matrix: Dense Storage

S |           1          2          3          4
-------------------------------------------------
1 |   6323.2499 -19.282826 -0.3279992 -0.5845209
2 |  -19.282826  185589.64  10998.017  23322.116
3 |  -0.3279992  10998.017  989.63839  1653.5970
4 |  -0.5845209  23322.116  1653.5970  3161.6238
```

The following code specifies approximate standard errors and the approximate correlation matrix of parameter estimates:

```
mle = xr; sopt = [ "grad" "hess" ];
< gopt,hopt > = fider(fmitchml,mle,sopt);
print "gopt=", gopt;
print "hopt=", hopt;

v = inv(hopt);
ase = sqrt(dia2vec(v)); tab = mle‘ -> ase;
tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);
d = diag(1. / ase);
corr = d * v * d;
corr = cname(corr,pnam); corr = rname(corr,pnam);

print "Estimates:",mle;
print "Standard Errors:",tab;
print "Correlations:",corr;
```

```
Estimates:
   |     sigma       th1       th2       th3
-------------------------------------------
 1 |   0.06412  -0.10306    2.5190   0.96313

Standard Errors:
        |   Estimate     STDERR
------------------------------
  sigma |    0.06412    0.01258
```

```
    th1 |   -0.10306    0.02230
    th2 |     2.5190    0.23240
    th3 |    0.96313    0.28116

Correlations:
    SYM |     sigma       th1       th2       th3
------------------------------------------------
  sigma |   1.00000
    th1 |   0.00426   1.00000
    th2 |   0.00394   0.92282   1.00000
    th3 |  -0.00419  -0.98400  -0.97222    1.0000
```

1.1.2 Richards Function on Nelder Data

The Richards function is given in Clarke (1987, p.226)

$$f(x_i, \theta) = \theta_4 - \theta_3 \log(1. + \exp(-\frac{\theta_2}{\theta_3} - \frac{\theta_1}{\theta_3} x_i))$$

The following data are from Nelder (1961) but are given in Clarke (1987) too:

```
print "G.P.Y. Clarke (1987): Richards Function";
nelder = [  -2.15   2.5451, -1.50   3.6652,  -.85   4.5110,
             -.08   5.6556,   .52   6.3982,  1.10   7.0397,
             2.28   7.8241,  3.23   8.2558,  4.00   8.4784,
             4.65   8.4121,  5.00   8.4784 ];
pnam = [" x y "];
nelder = cname(nelder,pnam);
```

The `nlp` specification for the negative log-likelihood function follows:

```
print "Maximum Likelihood Estimation";
function frichml(x) global(nelder) {
  w = 1. + exp(-x[3]/x[4] - x[2]*nelder[,1] / x[4]);
  mu = x[5] - x[4] * log(w);
  v = log(x[1]) + .5 *((nelder[,2] - mu) / x[1]) .** 2;
  f = v[+];
  return(f);
}

bc = cons(5,2,.);
bc[1,1] = 1.e-12; bc[4,1] = 1.e-12;

x0 = [ .1 1. -1. 1. 1. ];
pnam = [" sigma th1 th2 th3 th4 "];
x0 = cname(x0,pnam);

print "Trust-Region Algorithm: ML";
mopt = [ "tech"   "trureg" ,
         "print"         3 ];
< xr,rp,der1,der2 > = nlp(frichml,x0,mopt,bc);
print "Xopt=",xr;     print "RP=",rp;
print "DER1=", der1;  print "DER2=", der2;
```

The iteration history follows:

```
                    Trust Region Optimization
                    Without Parameter Scaling
               Gradient Computed by Finite Differences
             Hessian Computed by Finite Differences (dense)
                    (Using Only Function Calls)


 Iteration Start:
 N. Variables              5
 N. Bound. Constr.         2        N. Mask  Constr.          0
 Criterion       23675.63075        Max Grad Entry  473909.0671
 N. Active Constraints     0        TR Radius       1.000000000


Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
  1*   0    2   0  11002.90 12672.73  180742 2968.97  1.00000
  2*   0    3   0  4937.381 6065.517 67663.4 1584.82  1.00012
  3*   0    4   0  2111.973 2825.407 24703.2 821.053  1.00872
  4*   0    5   0  842.0891 1269.884 8691.91 409.876  1.02435
  5*   0    6   0  298.4665 543.6226 2874.55 199.587  1.05268
  6*   0    7   0  83.00837 215.4581 854.195 94.0314  1.09124
  7*   0    8   0  7.357272 75.65110 199.751 37.1669  1.13019
  8*   0   11   0 -19.08662 26.44389 60.7628 17.6754  3.63103
  9*   0   13   0 -20.60162 1.515000 407.233 697.064  0.12355
 10*   0   14   0 -26.07121 5.469591 173.536 147.840  0.12550
 11    0   15   0 -27.56660 1.495383 49.1202 29.2072  0.12629
 12    0   16   0 -28.08124 0.514645 11.8963 12.5331  0.12632
 13    0   17   0 -28.23798 0.156734 5.25356 0.00000  0.12641
 14    0   18   0 -28.24741 9.4e-003 0.38698 0.00000  0.11886
 15    0   19   0 -28.24748 7.1e-005  2e-003 0.00000  0.02629
 16    0   20   0 -28.24748 4.9e-009  3e-005 0.00000   3e-003

 Successful Termination After    16 Iterations
 GCONV convergence criterion satisfied.
 Criterion      -28.24747843        Max Grad Entry  2.8933e-005
 N. Active Constraints     0        Ridge (lambda)  0.000000000
 TR Radius       0.002808353        Act.dF/Pred.dF  1.528095205
 N. Function Calls        21        N. Gradient Calls         2
 N. Hessian Calls         18        Preproces. Time           1
 Time for Method           0        Effective Time            1
```

The tables of parameter estimates and the Hessian matrix follow:

```
                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------
```

```
Parameter     Estimate   Gradient   Active BC

 1 sigma    0.04651629  2.89e-005
 2 th1      1.64098753  2.71e-007
 3 th2     -2.39220641  4.92e-007
 4 th3      1.76818791  8.61e-008
 5 th4      8.55448140 -4.49e-007

 Value of Objective Function =     -28.2475

                Hessian Matrix
                **************

          Symmetric Matrix: Dense Storage

S |           1          2          3          4          5
--------------------------------------------------------------
1 |   10167.465 -1.3368009 -0.8633067 -0.3318032 -6.3062497
2 |  -1.3368009  3898.8341 -1418.6074 -279.59754 -602.24648
3 |  -0.8633067 -1418.6074  1960.1710 -1011.8422  2562.8372
4 |  -0.3318032 -279.59754 -1011.8422  923.41257 -1942.4388
5 |  -6.3062497 -602.24648  2562.8372 -1942.4388  5083.7307
```

The following is the code for specifying approximate standard errors and approximate correlation matrix of parameter estimates follow:

```
mle = xr; sopt = [ "grad" "hess" ];
< gopt,hopt > = fider(frichml,mle,sopt);
print "gopt=", gopt;
print "hopt=", hopt;

v = inv(hopt);
ase = sqrt(dia2vec(v)); tab = mle‘ -> ase;
tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);
d = diag(1. / ase);
corr = d * v * d;
corr = cname(corr,pnam); corr = rname(corr,pnam);

print "Estimates:",mle;
print "Standard Errors:",tab;
print "Correlations:",corr;
```

The tables of approximate standard errors and approximate correlation matrix of parameter estimates follow:

```
Estimates:
   |     sigma       th1       th2       th3       th4
-------------------------------------------------------
 1 |   0.04652    1.6410   -2.3922    1.7682    8.5545
```

```
Standard Errors:
        |   Estimate     STDERR
------------------------------
  sigma |    0.04652    0.00992
    th1 |     1.6410    0.07156
    th2 |    -2.3922    0.13604
    th3 |     1.7682    0.23678
    th4 |     8.5545    0.04590

Correlations:
    SYM |     sigma       th1       th2       th3       th4
------------------------------------------------------------
  sigma |   1.00000
    th1 |   0.00459   1.00000
    th2 |   0.00420   0.95834   1.00000
    th3 |   0.00497   0.94905   0.91795   1.00000
    th4 |   0.00464   0.62332   0.49209   0.77466   1.00000
```

1.2 ML Estimation: Error Rates of Two Medical Diagnostic Tests

This example shows how to use the `nlp` function to evaluate the accuracy of two diagnostic tests. The accuracy of any test is determined by its sensitivity (the ability of the test to correctly diagnose the presence of disease) and specificity (the ability of the test to correctly diagnose the absence of disease). The false positive rate α is defined as the proportion of non-diseased people who have positive outcomes of the test. Likewise, the false negative rate β is defined as the proportion of diseased people who have negative outcomes of the test. Therefore, sensitivity is $1 - \beta$ and specificity is $1 - \alpha$.

Hui and Walter (1980) showed that, if data are available from two populations with different disease prevalences, you can use maximum likelihood to estimate the error rates of both tests and also the prevalences in both populations. In this example, the Tine test (a new test) and the Mantoux test (the standard test), are compared for detection of tuberculosis in low and high prevalence popupations. Both skin tests are applied simultaneously to the arm of each person, with the evaluation made after 48 hours. The results were:

	Population 1			Population 2		
	Tine test			Tine test		
Mantoux test	Positive	Negative	Total	Positive	Negative	Total
Positive	14	4	18	887	31	918
Negative	9	528	537	37	367	404
Total	23	532	555	924	398	1322

Let

- N_i be the sample size from the i^{th} population, $i = 1, 2$
- α_j be the false positive rate of the j^{th} test, $j = 1, 2$
- β_j be the false negative rate of the j^{th} test, $j = 1, 2$
- θ_i be the prevalence rate for the i^{th} population, $i = 1, 2$, with $\theta_1 \neq \theta_2$

Under the assumption of conditional independence between the tests, the likelihood function is the product of two independent multinomials and is given by

$$\begin{aligned} L &= \prod_{i=1}^{2} \{\theta_i (1-\beta_1)(1-\beta_2) + (1-\theta_i)\alpha_1\alpha_2\}^{N_i P_{i,1,1}} \\ &\quad \times \{\theta_i (1-\beta_1)\beta_2 + (1-\theta_i)\alpha_1(1-\alpha_2)\}^{N_i P_{i,1,2}} \\ &\quad \times \{\theta_i \beta_1 (1-\beta_2) + (1-\theta_i)(1-\alpha_1)\alpha_2\}^{N_i P_{i,2,1}} \\ &\quad \times \{\theta_i \beta_1 \beta_2 + (1-\theta_i)(1-\alpha_1)(1-\alpha_2)\}^{N_i P_{i,2,2}} \end{aligned}$$

where $P_{i,j,j'}$ is the observed proportion of the sample i with test outcomes j and j' in tests 1 and 2, respectively:

- $j = 1(2)$ if test 1 is positive (negative) and
- $j' = 1(2)$ if test 2 is positive (negative).

The likelihood can be maximized by numerical optimization methods with the solution restricted to the unit hypercube. In practice, it is common to approach this problem by minimizing the negative log of the likelihood. This is the approach taken here.
The negative log likelihood function can be minimized using `nlp`. For this example, the trust region technique is used with `nlp` to calculate the maximum likelihood parameter estimates and `fider` is used to compute the second-order derivative.

```
function fhuiwal(z) global(data) {
  /* pieces of the log likelihood */
  p = cons(8,1);
  p[1] = log(z[5]*(1-z[2])*(1-z[4]) + (1-z[5])*z[1]*z[3]);
  p[2] = log(z[5]*(1-z[2])*z[4]     + (1-z[5])*z[1]*(1-z[3]));
  p[3] = log(z[5]*z[2]*(1-z[4])     + (1-z[5])*(1-z[1])*z[3]);
  p[4] = log(z[5]*z[2]*z[4]         + (1-z[5])*(1-z[1])*(1-z[3]));
  p[5] = log(z[6]*(1-z[2])*(1-z[4]) + (1-z[6])*z[1]*z[3]);
  p[6] = log(z[6]*(1-z[2])*z[4]     + (1-z[6])*z[1]*(1-z[3]));
  p[7] = log(z[6]*z[2]*(1-z[4])     + (1-z[6])*(1-z[1])*z[3]);
  p[8] = log(z[6]*z[2]*z[4]         + (1-z[6])*(1-z[1])*(1-z[3]));
  /* F is the negative log likelihood function */
  f = -data * p;
  return (f);
}
```

```
/* Supply the data values for test1 and test2:                          */
/* first  four values correspond to pop1: (+,+), (+,-), (-,+), (-,-)  */
/* second four values correspond to pop2: (+,+), (+,-), (-,+), (-,-)  */
data = [ 14.0 4.0 9.0 528.0 887.0 31.0 37.0 367.0 ];

/* Constrain the parameters to the unit hypercube */
bc = cons(6,1,.001) -> cons(6,1,.999);

/* Provide a starting point x0.  The parameters are:  */
x0 = [ .01 .05 .01 .05 .05 .70 ];
pnam = [" alpha1 beta1 alpha2 beta2 theta1 theta2 "];
x0 = cname(x0,pnam);

print "Trust-Region Algorithm: ML";
mopt = [ "tech"   "trureg" ,
         "print"         3 ];
< xr,rp,der1,der2 > = nlp(fhuiwal,x0,mopt,bc);
print "Xopt=",xr;     print "RP=",rp;
print "DER1=", der1;  print "DER2=", der2;
```

The output shows there were no convergence problems and prints the estimates of the error rates, the prevalences, and the estimated variance-covariance matrix. Note that you need to subtract the estimated error rates from 1 to obtain the respective estimates for sensitivity and specificity.

```
                Trust Region Optimization
                Without Parameter Scaling
           Gradient Computed by Finite Differences
       Hessian Computed by Finite Differences (dense)
                (Using Only Function Calls)


 Iteration Start:
 N. Variables              6
 N. Bound. Constr.        12        N. Mask  Constr.          0
 Criterion       1218.723512        Max Grad Entry  355.8748373
 N. Active Constraints     0        TR Radius       1.000000000


Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
   1    0    3   0  1208.864 9.859474 259.907 8941.93  0.02968
   2    0    4   0  1207.675 1.188848 64.2961 0.00000  0.02783
   3    0    5   0  1207.617 0.058087 5.73393 0.00000  0.01422
   4    0    6   0  1207.617 3.5e-004 0.05911 0.00000   2e-003
   5    0    7   0  1207.617 2.7e-008  2e-005 0.00000   1e-004

 Successful Termination After     5 Iterations
 GCONV convergence criterion satisfied.
 Criterion       1207.616749        Max Grad Entry  1.5158e-005
 N. Active Constraints     0        Ridge (lambda)  0.000000000
```

```
TR Radius       0.000139531        Act.dF/Pred.dF  1.016132951
N. Function Calls         8        N. Gradient Calls         2
N. Hessian Calls          7        Preproces. Time           0
Time for Method           0        Effective Time            0

                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

       Parameter     Estimate   Gradient   Active BC

        1 alpha1   0.00668137  1.52e-005
        2 beta1    0.03387585  0.0000000
        3 alpha2   0.01586428  0.0000000
        4 beta2    0.03117227 -1.48e-005
        5 theta1   0.02683959  0.0000000
        6 theta2   0.71679214 -8.89e-006

        Value of Objective Function =      1207.62

                       Hessian Matrix
                       **************

             Symmetric Matrix: Dense Storage

  S |           1          2          3          4          5
  ------------------------------------------------------------
  1 |   75931.582 -618.91465 -164.14334  12747.427  2588.6667
  2 |  -618.91465  23798.136  10051.775 -706.38428 -522.24166
  3 |  -164.14334  10051.775  36645.312 -478.88769  1236.7929
  4 |   12747.427 -706.38428 -478.88769  28078.882 -487.97031
  5 |   2588.6667 -522.24166  1236.7929 -487.97031  19880.571
  6 |   1670.5061 -854.37704  1535.7543 -490.36164  0.0000000

  S |           6
  ---------------
  1 |   1670.5061
  2 |  -854.37704
  3 |   1535.7543
  4 |  -490.36164
  5 |   0.0000000
  6 |   6311.7641
```

The following is the CMAT code for computing asymptotic standard errors and the correlation matrix of parameter estimates under normality assumptions:

```
mle = xr; sopt = [ "grad" "hess" ];
```

```
  < gopt,hopt > = fider(fhuiwal,mle,sopt);
  print "gopt=", gopt;
  print "hopt=", hopt;

  v = inv(hopt);
  ase = sqrt(dia2vec(v)); tab = mle‘ -> ase;
  tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);
  d = diag(1. / ase);
  corr = d * v * d;
  corr = cname(corr,pnam); corr = rname(corr,pnam);

  print "Estimates:",mle;
  print "Standard Errors:",tab;
  print "Correlations:",corr;
```

```
Estimates:
   |    alpha1     beta1    alpha2     beta2    theta1    theta2
-------------------------------------------------------------------
 1 |   0.00668   0.03388   0.01586   0.03117   0.02684   0.71679
```

```
Standard Errors:
         |   Estimate     STDERR
--------------------------------
  alpha1 |    0.00668    0.00380
   beta1 |    0.03388    0.00695
  alpha2 |    0.01586    0.00562
   beta2 |    0.03117    0.00623
  theta1 |    0.02684    0.00713
  theta2 |    0.71679    0.01279
```

```
Correlations:
     SYM |    alpha1     beta1    alpha2
-------------------------------------
  alpha1 |   1.00000
   beta1 |  -0.00535   1.00000
  alpha2 |   0.01273  -0.35154   1.00000
   beta2 |  -0.28189   0.02904  -0.00399
  theta1 |  -0.07626   0.04382  -0.05897
  theta2 |  -0.09107   0.11255  -0.13378

     SYM |     beta2    theta1    theta2
-------------------------------------
   beta2 |    1.0000
  theta1 |   0.04196   1.00000
  theta2 |   0.06255   0.01710    1.0000
```

Using the estimated variance-covariance matrix to obtain the standard deviations for the estimates, the maximum likelihood estimates ($\pm$ one standard error) are:

- $\hat{\alpha_1} = .0067 \pm .0038$

- $\hat{\beta_1} = .0339 \pm .0069$
- $\hat{\alpha_2} = .0159 \pm .0056$
- $\hat{\beta_2} = .0312 \pm .0062$
- $\hat{\theta_1} = .0268 \pm .0071$
- $\hat{\theta_2} = .7168 \pm .0128$

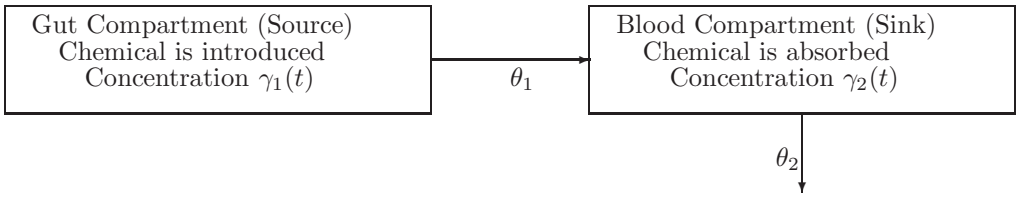
These results match those given by Hui and Walter.

1.3 Diffusion of Tetracycline Hydrochloride

The CMAT code for this example can be found in files `cmat/tnlp/tnlp160.inp` and `cmat/tnlp/tnlp164.inp`.

1.3.1 Description of the Problem

This example shows how to use the `nlp` and `ode` functions to estimate the parameters of a two-compartment model for the diffusion of the drug Tetracycline Hydrochloride. In a system of compartments, the rates of flow of the drug between the compartments follow first-order differential equations.
A tetracycline compound was administered to a subject orally, and measurements of the concentration of tetracycline hydrochloride in the blood serum were taken over the next 16 hours. The data consist of two variables: time in hours since the adminstration of drug and concentration in micrograms per milliliter.
The input is a bolus in the gut, and the drug is then absorbed into the blood. Let $\gamma_1(t)$ and $\gamma_2(t)$ be the concentrations at time t in the gut and in the serum, respectively. Let θ_1 and θ_2 be the transfer parameters. The model is depicted as follows:



Two-Compartment Model for Tetracyline Diffusion

The rates of flow of the drug are described by the following pair of ordinary differential equations:

$$\frac{\partial \gamma_1(t)}{\partial t} \quad = \quad \dot{\gamma}_1(t) = -\theta_1 \gamma_1(t)$$

$$\frac{\partial \gamma_2(t)}{\partial t} \quad = \quad \dot{\gamma}_2(t) = \theta_1 \gamma_1(t) - \theta_2 \gamma_2(t)$$

The initial concentration of the drug in the gut is unknown, but the concentration in the serum is assumed to be zero. Let θ_3 be the unknown initial concentration in the gut. The initial conditions are

$$\gamma_1(0) = \theta_3 \quad \text{and} \quad \gamma_2(0) = 0$$

Suppose y_i is the observed serum concentration at t_i. The parameters $\theta_1, \theta_2,$ and θ_3 are estimated by minimizing the sum of squares of the differences between the observed and predicted serum concentrations

$$\sum_i \{y_i - \gamma_2(t_i)\}^2$$

Since CMAT is a matrix language, it is more convenient to express the system of differential equations in matrix notation. Let

$$\gamma(t) = \begin{bmatrix} \gamma_1(t) \\ \gamma_2(t) \end{bmatrix}$$

then

$$\frac{\partial \gamma(t)}{\partial t} = \dot{\gamma}(t) = \mathbf{A}\gamma(t)$$

where the transfer matrix $\mathbf{A}$ and the initial condition are

$$\mathbf{A} = \begin{bmatrix} -\theta_1 & 0 \\ \theta_1 & -\theta_2 \end{bmatrix} \quad \text{and} \quad \gamma(0) = \begin{bmatrix} \theta_3 \\ 0 \end{bmatrix}$$

For fixed $\theta_1, \theta_2,$ and θ_3, the given system of differential equations can be solved by specifying

```
par = [ "print"         2 ,
        "reltol"    1.e-9 ,
        "met"      "rk56" ];
< res, tout > = ode(der,t,c,par,.,.,jac);
```

The input arguments of the ODE subroutine are as follows:

der name of the function returning $\mathbf{A}\gamma(t)$

t vector of time points specifying the limits of integration over connected subintervals. The first component must contain the initial time of the integration.

c initial values $\gamma(0)$

par two-column matrix specifying options

jac name of the function for the Jacobian of the function "der"

Among others we can specify the following options:

"print" 2 controls printed output (2=all output)

"reltol" 1.e-8 relative accuracy criterion for termination

"met" "adam" specifies Adams method

"met" "gear" specifies Gear method

"met" "rk56" specifies Verner's (5,6) order Runge-Kutta method

The output arguments of the ODE subroutine are as follows:

res name of the vector that contains the solutions

tout vector of time points which the solution `res` approximates closer than the input vector `t`.

See the *Reference Manual* of CMAT for more information.

1.3.2 Solving the Basic Problem

Using the data in Bates & Watts (1988, p.281), the parameters of the compartment models are estimated by the specifications given in the following steps:

1. Create the input data. Invoke IML and read in the data.

```
print "LS Estimation with ODE: Bates and Watts (1988):";
print "Example: Tetracycline : Data by Wagner (1967)";
options ps=60 ls=68;
wagner = [ 1 0.7,   2 1.2,   3 1.4,   4 1.4,   6 1.1,
           8 0.8,  10 0.6,  12 0.5,  16 0.3 ];
wagner = cname(wagner, [ "time" "conc" ]);
m = nrow(wagner);
```

2. Function and Jacobian Modules needed by ODE call:

   The `derode` function evaluates system of ODE's, where the rows of `y` correspond to ODE's and `x` corresponds to γ_t:

```
function derode(t,y) global(A) {
  z = A * y;
  return(z);
}
```

The `jacode` function evaluates Jacobian of `derode()`:

```
function jacode(t,y) global(A) {
  return(A);
}
```

3. Define the objective function F0 for the LS estimation. The serum concentrations are computed in the function F0 by calling the ODE subroutine.

```
function f0(theta,time,ipri) global(A) {
  /* transfer matrix A needed by derode() and jacode() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2]; th3 = theta[3];
  A = [ t1m   0. ,
        th1  t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0];
  /* vector of increasing time points
     used for the integration limits */
  t = [0 , time ];
  par = [ "print"      ipri ,
          "task"      "exa" ,
          "minval"       1. ,
          "reltol"    1.e-4 ,
          "met"      "gear" ];
  res = ode(derode,t,c,par,.,.,jacode);
  /* res = ode(derode,t,c,par); */
  return(res[2,]‘);
}
```

4. Make some test runs of F0:

```
print "Test function F0 call";
time = wagner[,1];
t = [0., time]; th0 = [ .1 .3 10 ];
A = [ -.1  0. , .1  -.3 ];
fun = f0(th0,wagner[,1],2);
print "Fun=",fun;
```

```
         Solve Initial Value Problem dY_i/dT = F(Y_i,T)
             Use Adams Method for Non-Stiff Systems
         Start Iteration at T=0  Rel. Precision=0.0001
```

```
  Iter Nstep Nfunc Njac CurrentTime AvStpSiz  LStpSiz AvOrd LOrd

     1     9    18    0  1.00000000 0.111111 0.255994  1.67    3
     2    13    23    0  2.00000000 0.153846 0.232017  2.08    3
     3    18    28    0  3.00000000 0.166667 0.071933  2.39    4
     4    24    34    0  4.00000000 0.166667 0.640334  2.79    4
     5    28    39    0  6.00000000 0.214286 0.078999  2.96    4
     6    36    48    0  8.00000000 0.222222 0.110070  3.11    3
     7    42    55    0  10.0000000 0.238095 0.717648  3.10    3
     8    45    60    0  12.0000000 0.266667 0.564703  3.09    3
     9    51    67    0  16.0000000 0.313725 0.507446  3.08    3
       Sucessful Termination at T=16  Rel. Precision=0.0001
```

```
Fun=
   |         1
--------------
 1 |   0.00000
 2 |   0.81998
 3 |    1.3495
 4 |    1.6712
 5 |    1.8456
 6 |    1.9176
 7 |    1.7930
 8 |    1.5903
 9 |    1.3693
10 |   0.96828
```

5. LS Objective Function needed by LEVMAR call:

   Function F1 is a slightly modified version of F0 used to specify the least squares fit problem and to compute the difference between the observed and predicted serum concentrations.

```
function f1(theta) global(wagner,A,ipri,eps,met) {
  /* note: we fit the concentration in the serum,
     i.e. we compare the gamma_2 values in res[2,]
     with the concentration measurements in wagner[,2] */
  /* transfer matrix A needed by derode() and jacode() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2]; th3 = theta[3];
  A = [ t1m   0. ,
        th1  t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0];
  /* vector of increasing time points
```

```
      used for the integration limits */
    time = wagner[,1]; t = [ 0 , time ];
    par = [ "print"      ipri ,
            "reltol"      eps ,
            "task"      "int" ,
            "minval"      1. ,
            "met"         met ,
            "nolog"          ];
    res = ode(derode,t,c,par,.,.,jacode);
    /* res = ode(derode,t,c,par); */
    dif = res[2,2:10]' - wagner[,2];
    return(dif);
  }


  print "Test run of function F1(): RG56";
  ipri = 3; eps = 1.e-8; met = "rk56";
  th0 = [ .1 .3 10 ];
  f1ini = f1(th0);
  print "F1_ini=",f1ini;


           Solve Initial Value Problem dY_i/dT = F(Y_i,T)
           Use Runge-Kutta (5,6) Verner Method (Jackson)
           Start Iteration at T=0  Rel. Precision=1e-008

             Iter Nstep   Nfunc   Curr.Time   Rel.Error

                0     0       1  0.00000000  1.000e-008
                1     1      33  1.00000000  1.000e-008
                2     1      65  2.00000000  1.000e-008
                3     1      97  3.00000000  1.000e-008
                4     1     129  4.00000000  1.000e-008
                5     1     177  6.00000000  1.000e-008
                6     1     225  8.00000000  1.000e-008
                7     1     273  10.0000000  1.000e-008
                8     1     313  12.0000000  1.000e-008
                9     1     377  16.0000000  1.000e-008
        Sucessful Termination at T=16  Rel. Precision=1e-008


F1_ini=
   |         1
--------------
 1 |   0.12010
 2 |   0.14960
```

```
 3 |   0.27124
 4 |   0.44563
 5 |   0.81756
 6 |   0.99306
 7 |   0.99046
 8 |   0.86935
 9 |   0.66833
```

6. Try solving the problem by general minimization:

   Instead of using the Levenberg-Marquardt algorithm, you can use any of the other optimization algorithms for solving the `least squares` problem specified with function F1. However, you may also solve the related general MINimization problem using the the objective function specified in module F2.

   The following function F2 is the same as F1 except that it returns a scalar which is the sum of squares error between fitted and observed data:

```
function f2(theta) global(wagner,A,ipri,eps,met) {
  /* transfer matrix A needed by derode() and jacode() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2]; th3 = theta[3];
  A = [ t1m   0. ,
        th1  t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0];
  /* vector of increasing time points
     used for the integration limits */
  time = wagner[,1]; t = [ 0 , time ];
  par = [ "print"      ipri ,
          "reltol"      eps ,
          "task"      "int" ,
          "minval"       1. ,
          "met"         met ,
          "nolog"           ];
  res = ode(derode,t,c,par,.,.,jacode);
  /* res = ode(derode,t,c,par); */

  dif = ssq(res[2,2:10]' - wagner[,2]);
  return(dif);
}
```

   Note that for the special case of 2×2 transfer matrix $\mathbf{A}$, the system of ordinary differential equations can be solved analytically by computing the eigenvalue decomposition of $\mathbf{A}$, and PROC NLP can then be used for the

optimization. Refer to Bates & Watts (1988, pp.173-175) or Hartmann (1992) for more details.

7. Prepare and Perform `nlp` and `ode` calls:

   With $\theta_1 = .1, \theta_2 = .3$, and $\theta_3 = 10$ as starting values, the Levenberg-Marquardt algorithm converges quickly. We specify boundary constraunts, requiring that all parameters $\theta \geq 0$.

```
print "start values:  th0 = [ .1 .3 10 ]";
th0 = [ .1 .3 10 ];
tim = time("clock");
ipri = 0; eps = 1.e-8; met = "adam";
bc = cons(3,1,0.) -> cons(3,1,.);
mopt = [ "tech"   "levmar" ,
         "print"         3 ];
< xr, rp > = nlp(f1,th0,mopt,bc);
tim = time("clock") - tim;
print "Time=",tim;
```

```
                      ******************
                      Optimization Start
                      ******************

                     Parameter Estimates
                     -------------------

 Parameter     Estimate   Gradient   Lower BC   Upper BC

  1 X_1      0.10000000  38.271683  0.0000000     .
  2 X_2      0.30000000 -24.093935  0.0000000     .
  3 X_3      10.0000000  0.8337407  0.0000000     .

           Value of Objective Function =      2.07349


                        Hessian Matrix
                        **************

                 Symmetric Matrix: Dense Storage

                S |           1          2          3
                ------------------------------------------
                1 |   873.95662 -235.97285  12.240310
                2 |  -235.97285  141.47402 -5.0027356
                3 |   12.240310 -5.0027356  0.2092783
```

The iteration history is shown as follows:

```
                Levenberg-Marquardt Optimization
                  Scaling Update of More (1978)
              Gradient Computed by Finite Differences
            CRP Jacobian Computed by Finite Differences


 Iteration Start:
 N. Variables              3        N. Equations              9
 N. Bound. Constr.         3        N. Mask  Constr.          0
 Criterion       2.073493759        Max Grad Entry  38.27168224
 N. Active Constraints     0        TR Radius       1167.146748


Iter rest nfun act   optcrit  difcrit maxgrad  lambda      rho
   1    0    2   0  0.284283 1.789211 5.00033 0.00000  0.86844
   2    0    4   0  0.065578 0.218705 4.15760 0.01296  0.87960
   3    0    6   0  0.034129 0.031449 0.75303 0.01725  0.98411
   4    0    9   0  0.025409 8.7e-003 0.72110  4e-003  0.88141
   5    0   11   0  0.021930 3.5e-003 0.16632  9e-003  0.99150
   6    0   14   0  0.019378 2.6e-003 0.33477  3e-003  0.83349
   7    0   16   0  0.018333 1.0e-003 0.31083  8e-004  0.64902
   8    0   17   0  0.017829 5.0e-004 0.03131 0.00000  1.00163
   9    0   18   0  0.017823 5.2e-006  2e-004 0.00000  0.98225
  10    0   19   0  0.017823 3.6e-008  2e-004 0.00000  0.78709
  11    0   20   0  0.017823 2.0e-009  2e-005 0.00000  0.94816
  12    0   23   0  0.017823 7.2e-012  6e-006  5e-003  0.30174

 Successful Termination After    12 Iterations
 GCONV convergence criterion satisfied.
 Criterion       0.017823385        Max Grad Entry  6.4271e-006
 N. Active Constraints     0        Ridge (lambda)  0.004617837
 TR Radius       4.1687e-005        Act.dF/Pred.dF  0.301741460
 N. Function Calls        24        N. Gradient Calls         2
 N. Hessian Calls         14        Preproces. Time           1
 Time for Method           5        Effective Time            7
```

The parameter estimates match those of Bates & Watts (1988, p. 175):

```
                    ********************
                    Optimization Results
```

```
                    ********************

                    Parameter Estimates
                    -------------------

Parameter     Estimate   Gradient   Active BC

 1 X_1      0.18302024  6.43e-006
 2 X_2      0.43449513  1.29e-006
 3 X_3      5.99545670  4.19e-007

 Value of Objective Function =    0.0178234


                  Hessian Matrix
                  **************

       Symmetric Matrix: Dense Storage

      S |           1          2          3
      -------------------------------------
      1 |   95.585789 -25.452984  3.7577047
      2 |  -25.452984  28.829443 -2.3563995
      3 |   3.7577047 -2.3563995  0.2327152
```

8. LS Fit of Tetracycline: Studentized Residuals

   A plot of the Tetracyline data and the fitted response versus time (Figure 16) shows a poor fit, since the fitted curve is too squat. A plot of Studentized residuals versus time (Figure 17) also reveals that the model may be indequate for the data.

```
jtj = jopt‘ * jopt;
hat = jopt * inv(jtj) * jopt‘;
stures = sqrt(ssq(fopt) .* (dia2vec(ide(m) - hat)));
stures = fopt ./ stures;
print "Studentized Residuals", fopt, stures;
```

```
   |         1                          |         1
---------------                       ---------------
 1 |   0.10792                         1 |   0.74849
 2 |  -0.00399                         2 |  -0.02681
 3 |  -0.06519                         3 |  -0.40523
 4 |  -0.06902                         4 |  -0.43811
 5 |   0.03335                         5 |   0.22077
 6 |   0.07417                         6 |   0.45976
 7 |   0.04321                         7 |   0.26579
 8 |  -0.03844                         8 |  -0.24993
 9 |  -0.07080                         9 |  -0.48519
```

1.3.3 Solving the Problem with Dead Time Parameter

1. Introducing an additional parameter for dead time:

   If a system does not respond immediately to the input, the addition of a delay or dead time parameter can improve the fit of the model. Let θ_4 be the dead time parameter. Consider the modified model:

$$\begin{aligned} \dot{\gamma}(\tau) &= \mathbf{A}\gamma(\tau) \\ \gamma(0) &= \gamma_0 \end{aligned}$$

   with

$$\tau = \begin{cases} t - \theta_4 & \text{for } t > \theta_4 \\ 0 & \text{for } t \leq \theta_4 \end{cases}$$

2. To incorporate the dead time parameter, the F1 function for the Levenberg-Marquardt algorithm is modified by changing the specification of the t vector to include the dead time parameter (instead of 0) as the lower limit for the numerical integration of the differential equations.

```
function f10(theta,wagner,ipri) global(A) {
  /* 1. prepare input for ODE */
  /* transfer matrix A needed by derode() and jacode() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2];
  th3 = theta[3]; th4 = theta[4];
  A = [ t1m   0. ,
        th1  t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0. ];
  /* vector of increasing time points
     used for the integration limits */
  time = wagner[,1]; t = [ th4 , time ];
  par = [ "print"      ipri ,
```

```
          "task"      "int" ,
          "minval"       1. ,
          "reltol"    1.e-4 ,
          "met"      "gear" ];
  res = ode(derode,t,c,par,.,.,jacode);
  return(res[2,]');
}

function f11(theta) global(wagner,A,ipri,eps,met) {
  /* dif = f10(theta,wagner[,1]) - wagner[,2]; */
  /* transfer matrix A needed by der() and jac() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2];
  th3 = theta[3]; th4 = theta[4];
  A = [ t1m     0. ,
        th1    t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0. ];
  /* vector of increasing time points
     used for the integration limits */
  time = wagner[,1]; t = [ th4 , time ];
  par = [ "print"      ipri ,
          "task"      "int" ,
          "minval"       1. ,
          "reltol"      eps ,
          "met"         met ,
          "nolog"          ];
  res = ode(derode,t,c,par,.,.,jacode);
  dif = res[2,2:10]' - wagner[,2];
  return(dif);
}
```

3. The arguments for NLPLM have to be modified to reflect the additional parameter θ_4. A constraint is imposed on θ_4 to restrict it in the range between zero and the first time measurement in the data, which is 1. A starting value of .1 for θ_4 is used.

```
m = nrow(wagner);
th0 = [ .1 .3 10 .1 ];
/* to perform ODE, dead time parameter theta[4] must be
   smaller than the first time measurement = .7 */
bc = cons(4,1,0.) -> cons(4,1,.);
bc[4,2] = .7; /* print "BC=",bc; */
ipri = 0; eps = 1.e-8; met = "adam";
mopt = [ "tech"   "levmar" ,
         "print"        3 ];
< xr, rp > = nlp(f11,th0,mopt,bc);

                        ******************
                        Optimization Start
```

****

Parameter Estimates

Parameter	Estimate	Gradient	Lower BC	Upper BC
1 X_1	0.10000000	37.448785	0.0000000	.
2 X_2	0.30000000	-24.030149	0.0000000	.
3 X_3	10.0000000	0.8240406	0.0000000	.
4 X_4	0.10000000	0.1701123	0.0000000	0.7000000

Value of Objective Function = 2.08764

Hessian Matrix

Symmetric Matrix: Dense Storage

S	1	2	3	4
1	864.48305	-233.61791	12.116656	-13.568670
2	-233.61791	139.96633	-4.9481496	-0.4288472
3	12.116656	-4.9481496	0.2070553	-0.1126972
4	-13.568670	-0.4288472	-0.1126972	0.7832142

4. The iteration history does not show any convergence problems in the optimization.

Levenberg-Marquardt Optimization
Scaling Update of More (1978)
Gradient Computed by Finite Differences
CRP Jacobian Computed by Finite Differences

Iteration Start:			
N. Variables	4	N. Equations	9
N. Bound. Constr.	5	N. Mask Constr.	0
Criterion	2.087641605	Max Grad Entry	37.44878512
N. Active Constraints	0	TR Radius	1137.182678

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	lambda	rho
1	0	2	0	0.514668	1.572974	25.1347	0.00000	0.75432
2	0	4	1	0.212573	0.302095	13.2358	0.01630	0.82072
3	0	6	0’	0.079012	0.133560	3.34982	0.01669	0.97253
4	0	8	0	0.032153	0.046859	2.84165	0.01321	0.74765
5	0	12	0	0.015512	0.016641	1.80673	2e-004	0.61678
6	0	13	0	5.0e-003	0.010481	9e-003	0.00000	0.99904
7	0	14	0	5.0e-003	8.0e-006	2e-003	0.00000	0.74891

```
 8    0   18   0  5.0e-003 2.2e-009  8e-004 0.02396  0.05076
 9    0   19   0  5.0e-003 1.2e-008  9e-004 0.05416  1.49553
10    0   20   0  5.0e-003 1.9e-008  1e-003 0.11063  0.27026
11    0   21   0  5.0e-003 2.3e-008  3e-004 0.02050  1.35776
12    0   29   0  5.0e-003 2.5e-012  3e-004 1292.10  5.27292
13    0   30   0  5.0e-003 4.0e-012  2e-004 994.914  2.60020
14    0   31   0  5.0e-003 3.5e-012  2e-004 410.124  1.34910
15    0   64   0  5.0e-003 0.000000  2e-004  3e+017  1.00000

Successful Termination After    15 Iterations
XCONV convergence criterion satisfied.
Criterion       0.005023148        Max Grad Entry  0.000217209
N. Active Constraints     0        Ridge (lambda)  3.3599e+017
TR Radius       1.4733e-017        Act.dF/Pred.dF  1.000000000
N. Function Calls        65        N. Gradient Calls         2
N. Hessian Calls         17        Preproces. Time           0
Time for Method          12        Effective Time           13

                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

       Parameter     Estimate   Gradient   Active BC

        1 X_1      0.14854360  1.39e-004
        2 X_2      0.71824401  2.17e-004
        3 X_3      10.1351892  1.20e-005
        4 X_4      0.41371618  6.56e-007

        Value of Objective Function =   0.00502315

                      Hessian Matrix
                      **************

             Symmetric Matrix: Dense Storage

        S |           1          2          3          4
        ---------------------------------------------------
        1 |   172.45466 -29.388540  3.1611804 -5.5203780
        2 |  -29.388540  12.141842 -0.9393271 -0.2548978
        3 |   3.1611804 -0.9393271  0.0816783 -0.0541781
        4 |  -5.5203780 -0.2548978 -0.0541781  0.9293510
```

We save some of the optimal results and compute the Jacobian:

```
thopt = xr;
```

```
fopt = f11(thopt);              print "Fopt=",fopt;
jopt = fider(f11,thopt,"jaco"); print "Jopt=",jopt;
jtj = jopt‘ * jopt;
```

5. LS Fit of Tetracycline with Dead Time: Data and Fit with Inference Band

   The Tetracycline data and the fitted model with dead time are plotted in Figure 18. The approximate inference bands for the expected response are also given in the same plot.

```
n = ncol(thopt); m = nrow(wagner);
fres = fopt + wagner[,2]; print "fres=",fres;
sigma = ssq(fopt) / (m-n);
sigm2 = sqrt(sigma); print "m,n,sigm2=",m,n,sigm2;

/* Inference Band */
function f12(theta) global(xtim,A,ipri,eps,met) {
  /* ycon = f10(theta,xtim); */
  /* transfer matrix A needed by der() and jac() */
  th1 = theta[1]; t1m = -th1; t2m = -theta[2];
  th3 = theta[3]; th4 = theta[4];
  A = [ t1m   0. ,
        th1  t2m ];
  /* vector  of initial values y(0) = gamma(0) */
  c = [ th3 , 0. ];
  /* vector of increasing time points
     used for the integration limits */
  t = [ th4 , xtim ];
  par = [ "print"      ipri ,
          "reltol"      eps ,
          "task"      "int" ,
          "minval"      1. ,
          "met"        met ,
          "nolog"         ];
  ycon = ode(derode,t,c,par,.,.,jacode);
  nc = ncol(ycon);
  return(ycon[2,2:nc]‘);
}

ipri = 0; eps = 1.e-10; met = "rk56";
a = thopt[4] + .1;
xtim = [ a : .2 : 16. ]‘; /* print "XTIM=",xtim; */
yvelo = f12(thopt);
jac = fider(f12,thopt,"jaco");
jtjopt = jopt‘ * jopt;
dhat = sqrt(dia2vec(jac * inv(jtjopt) * jac‘));
```

```
   /* correct bug: here n=2 */
   quan = fmis(.95, . ,n,m-n);
   fact = sigm2 * sqrt(2. * quan);
   yupp = yvelo + fact * dhat;
   ylow = yvelo - fact * dhat;

   pmat = xtim -> yvelo -> ylow -> yupp;
   cnam = [ "x" "y" "y_low" "y_upp" ];
   pmat = cname(pmat,cnam);
   print "Inference Band=", pmat;
```

	x	y	y_low	y_upp
1	0.51372	0.14419	-0.23597	0.52434
2	0.71372	0.39707	0.14546	0.64868
3	0.91372	0.60815	0.44124	0.77506
4	1.1137	0.78326	0.66133	0.90518
5	1.3137	0.92742	0.81920	1.0356
6	1.5137	1.0450	0.93544	1.1546
7	1.7137	1.1398	1.0265	1.2531
8	1.9137	1.2150	1.1005	1.3295
9	2.1137	1.2735	1.1609	1.3862
10	2.3137	1.3177	1.2094	1.4260
11	2.5137	1.3497	1.2471	1.4523
12	2.7137	1.3713	1.2747	1.4680
13	2.9137	1.3841	1.2929	1.4754
14	3.1137	1.3895	1.3023	1.4767
15	3.3137	1.3885	1.3038	1.4733
16	3.5137	1.3823	1.2984	1.4662
17	3.7137	1.3717	1.2872	1.4561
18	3.9137	1.3573	1.2713	1.4433
19	4.1137	1.3400	1.2519	1.4280
20	4.3137	1.3201	1.2298	1.4104
21	4.5137	1.2982	1.2059	1.3906
22	4.7137	1.2748	1.1806	1.3689
23	4.9137	1.2501	1.1546	1.3455
24	5.1137	1.2244	1.1281	1.3207
25	5.3137	1.1980	1.1014	1.2946
26	5.5137	1.1711	1.0747	1.2675
27	5.7137	1.1439	1.0482	1.2396
28	5.9137	1.1165	1.0219	1.2112
29	6.1137	1.0892	0.99598	1.1824
30	6.3137	1.0619	0.97043	1.1534
31	6.5137	1.0348	0.94527	1.1244
32	6.7137	1.0080	0.92051	1.0954
33	6.9137	0.98146	0.89614	1.0668

34	7.1137	0.95533	0.87216	1.0385
35	7.3137	0.92961	0.84854	1.0107
36	7.5137	0.90434	0.82527	0.98342
37	7.7137	0.87956	0.80231	0.95680
38	7.9137	0.85527	0.77965	0.93089
39	8.1137	0.83150	0.75727	0.90573
40	8.3137	0.80826	0.73516	0.88135
41	8.5137	0.78554	0.71331	0.85778
42	8.7137	0.76337	0.69171	0.83503
43	8.9137	0.74174	0.67039	0.81309
44	9.1137	0.72064	0.64934	0.79195
45	9.3137	0.70008	0.62858	0.77159
46	9.5137	0.68005	0.60814	0.75197
47	9.7137	0.66055	0.58803	0.73306
48	9.9137	0.64156	0.56828	0.71484
49	10.114	0.62308	0.54890	0.69726
50	10.314	0.60510	0.52995	0.68025
51	10.514	0.58761	0.51140	0.66383
52	10.714	0.57061	0.49326	0.64796
53	10.914	0.55408	0.47557	0.63258
54	11.114	0.53800	0.45833	0.61768
55	11.314	0.52238	0.44153	0.60323
56	11.514	0.50720	0.42523	0.58917
57	11.714	0.49245	0.40938	0.57552
58	11.914	0.47812	0.39394	0.56230
59	12.114	0.46419	0.37910	0.54929
60	12.314	0.45067	0.36481	0.53652
61	12.514	0.43753	0.35080	0.52426
62	12.714	0.42477	0.33713	0.51241
63	12.914	0.41238	0.32400	0.50075
64	13.114	0.40034	0.31135	0.48933
65	13.314	0.38865	0.29915	0.47815
66	13.514	0.37730	0.28736	0.46724
67	13.714	0.36628	0.27588	0.45667
68	13.914	0.35558	0.26484	0.44632
69	14.114	0.34519	0.25434	0.43604
70	14.314	0.33510	0.24419	0.42600
71	14.514	0.32530	0.23421	0.41639
72	14.714	0.31579	0.22465	0.40693
73	14.914	0.30655	0.21580	0.39731
74	15.114	0.29759	0.20723	0.38795
75	15.314	0.28889	0.19849	0.37928
76	15.514	0.28044	0.19023	0.37064
77	15.714	0.27223	0.18312	0.36135
78	15.914	0.26427	0.17621	0.35233

6. LS Fit of Tetracycline with Dead Time: Studentized Residuals:

   The studentized residuals are plotted in Figure 19. The modified model fit the description of the data better than the model without the dead parameter. The distribution of the studentized residuals appears to be more uniform than that of the previous model.

```
dhat = dia2vec(jopt * inv(jtjopt) * jopt');
print "Dhat=", dhat;
stures = sigm2 * sqrt(cons(m,1,1.) - dhat);
stures = fopt ./ stures;
print "Studendized Residuals=", fopt, stures;
```

```
   |         1                                |         1
---------------                              ---------------
 1 |  -0.01220                                1 |   -1.8704
 2 |   0.04215                                2 |    1.5331
 3 |  -0.01271                                3 |  -0.36085
 4 |  -0.04983                                4 |   -1.3912
 5 |   0.00474                                5 |   0.13928
 6 |   0.04495                                6 |    1.1733
 7 |   0.03352                                7 |   0.86941
 8 |  -0.02794                                8 |  -0.76904
 9 |  -0.03910                                9 |   -1.0875
```

1.3.4 Comparing Different Optimization Methods

The following CMAT input

```
ipri = 0; eps = 1.e-8;
vmet = [ "adam" "gear" "dyna" "rk45" "rk56" ];
bc = cons(3,1,0.) -> cons(3,1,.);
vtech = [ "levmar" "trureg" "nrridg" "dbldog" ];

for (it = 1; it <= 4; it++)
for (im = 1; im <= 5; im++) {
  tech = vtech[it]; met = vmet[im];
  tim = time("clock");
  mopt = [ "tech"       tech ,
           "maxit"      1000 ,
           "maxfu"      3000 ,
           "print"         3 ];
  < xr, rp > = nlp(f1,th0,mopt,bc);
  tim = time("clock") - tim;
  print "NLPtech=",tech," ODEmeth=",met," Time=",tim;
}
```

generates a table showing some computing times when optimizing the least squares objective function F1:

Table 1: Least Squares Approach: Time, Iterations, Function Calls

NLP Technique	Adams	Gear	RK(4,5)	RK(5,6)
LEVMAR	5.750,12,24	6.735,13,22	19.360,13,22	18.187,13,22
TRUREG	4.438, 9,11	5.484,10,12	16.344,10,12	15.656,10,12
NRRIDG	5.937,12,18	5.828,10,16	17.313,10,16	15.531,10,16
DBLDOG	9.250,29,36	10.703,30,38	30.906,30,38	29.156,30,38

The following CMAT input

```
ipri = 0; eps = 1.e-8;
vmet = [ "adam" "gear" "dyna" "rk45" "rk56" ];
bc = cons(3,1,0.) -> cons(3,1,.);
vtech = [ "trureg" "nrridg" "dbldog" ];

for (it = 1; it <= 3; it++)
for (im = 1; im <= 5; im++) {
  tech = vtech[it]; met = vmet[im];
  tim = time("clock");
  mopt = [ "tech"       tech ,
           "maxit"      1000 ,
           "maxfu"      3000 ,
           "print"         3 ];
  < xr, rp > = nlp(f2,th0,mopt,bc);
  tim = time("clock") - tim;
  print "NLPtech=",tech," ODEmeth=",met," Time=",tim;
}
```

generates a table showing some computing times when optimizing the general minimization function F2:
The results for the `"dyna"` ODE method are basically the same asthbat for the `"adam"` method. Obviously the trust region optimization technique using the least squares specification is recommended most for this kind of problem. Since double dogleg algorithm is using only the gradient but not the Jacobian, the computer times for the two runs are very similar.

Table 2: Minimization Approach: Time, Iterations, Function Calls

NLP Technique	Adams	Gear	RK(4,5)	RK(5,6)
TRUREG	12.891,11,21	15.656,12,23	46.172,12,23	42.687,12,23
NRRIDG	30.094,21,25	20.672,15,18	61.266,15,18	57.437,15,18
DBLDOG	9.094,29,35	10.562,30,38	30.547,30,38	29.156,30,38

1.4 Using QP and NLP for Computing Efficient Frontier

The CMAT code for this example can be found in files `cmat/tnlp/tnlp161.inp` and `cmat/tnlp/tnlp162.inp`.

1.4.1 Introduction to the Markowitz Model

The Markowitz model can be described by two equivalent approaches:

- Select a portfolio of stocks that offers the greatest return R_p for a given level of risk σ_p. This is a linear optimization problem with a quadratic constraint.
- Select a portfolio of stocks with lowest risk σ_p for a given level of return R_p. This is a quadratic optimization problem with a linear constraint.

This paper describes the second approach.
We want to select up to n stocks into a portfolio where each of the stocks is to be chosen with a weight x_j, $0 \leq x_j \leq 1$, $j = 1, \ldots, n$, and $\sum_{j=1}^{n} x_j = 1$. The basic information needed for our analysis is a data set containing the stock returns over a time period in the past. We describe the data with r_{ij}, where $i = 1, \ldots, N$ refers to the time points and $j = 1, \ldots, n$ refers to the stocks. Based on this data set, we can use PROC CORR to compute

- the *stock return vector* $\mathbf{r} = (r_j)$, that is, the vector of the arithmetic means

$$r_j = \frac{1}{N} \sum_{i=1}^{N} r_{ij}$$

- the covariance matrix $\mathbf{S} = (\sigma_{jk})$ of the stock return values

$$\sigma_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (r_{ij} - r_j)(r_{ik} - r_k) \quad , \quad j, k = 1, \ldots, n.$$

Two important functions depend on the weights x_j of the portfolio stocks:

- the *expected portfolio return* $R_p(x)$

$$R_p(x) = \sum_{j=1}^{n} x_j r_j$$

- the *expected portfolio risk*

$$\sigma_p(x) = \sqrt{\sum_{j=1}^{n} \sum_{k=1}^{n} \sigma_{jk} x_j x_k}$$

1.4.2 The Basic Minimum Risk Model

Here we only want to compute the optimal weights x_j that minimize the portfolio risk

$$\min_x \sigma_p^2(x) = \sigma_p^2(\hat{x})$$

subject to the basic set of constraints

$$0 \leq x_j \leq 1, \quad j = 1, \ldots, n \quad \text{lower and upper boundary constraints}$$

$$\sum_{j=1}^{n} x_j = 1 \quad \text{linear equality constraint}$$

This is a quadratic optimization problem $\min_x \sigma_p^2(x) = \min_x \mathbf{x}^T\mathbf{S}\mathbf{x}$ with a positive (semi)definite (covariance) matrix $\mathbf{S}$ subject to a set of linear constraints that can be solved with any of the optimization algorithms available in PROC NLP and IML. We first illustrate the use of the quadratic optimization technique in PROC NLP. Later we will also show how to use the IML subroutines for nonlinear optimization. Note that the objective function for the quadratic optimization techniques in PROC NLP and IML is $\frac{1}{2}\mathbf{x}^T\mathbf{S}\mathbf{x}$, which is half of the squared *portfolio risk*. That means we have to compute the square root of two times the value of the objective function to obtain the *portfolio risk* $\sigma_p(x)$.
The following data set contains the monthly returns of ten selected stocks during the period January 1978 until December 1987.

```
print "Stock Returns: CRSP Monthly Data, Jan 78 - Dec 87";
options NOECHO;
%inc "c:\\cmat\\tdata\\stock.dat";
options ECHO;

vlab = [ "Gerber Corporation" "Tandy Corporation"    "General Mills"
         "Con Edison"         "Weyerhauser"          "IBM"
         "DEC"                "Mobil Corporation"    "Texaco"
         "CPL" ];
vnam = [ "Gerber"   "TandyC"   "GenMills"
```

```
          "CEdison"  "Weyerh"   "IBM"
          "DEC"      "Mobil"    "Texaco"   "CPL" ];
   print "VName=",vnam;
   stock = cname(stock,vnam);
   nr = nrow(stock); nc = nvar= ncol(stock);
   print "nr,nc=",nr,nc;
```

The data contains 120 observations (rows, i.e. ten years each with twelfe months) for 10 stocks (columns).
The `univar` and `bivar` functions are used to compute the mean and covariance matrix of the data:

```
   print "Compute COV matrix and MEAN vector";
   sopt = [ "ari" "std" "med" ];
   univ = univar(stock,sopt);
   print "Univar=",univ;
   mean = univ[1,];
   print "Mean=",mean;
   cov = bivar(stock,"cov");
   print "COV=",cov;
```

The following is the result of the `univar` call:

```
Univar=
           |     Gerber     TandyC   GenMills    CEdison
-----------------------------------------------------------
  Ari_Mean |    0.01543    0.02501    0.01523    0.01851
  Std_Dev  |    0.08711    0.12757    0.06328    0.05027
  Median   |    0.01500    0.02250    0.01150    0.01950

           |     Weyerh        IBM        DEC      Mobil
-----------------------------------------------------------
  Ari_Mean |    0.00963    0.00962    0.01975    0.01619
  Std_Dev  |    0.08507    0.05902    0.09914    0.08031
  Median   |   -0.00200    0.00200    0.02400    0.01250

           |     Texaco        CPL
---------------------------------
  Ari_Mean |    0.01194    0.01270
  Std_Dev  |    0.07970    0.05395
  Median   |    0.01050    0.01350
```

For space reasons we do not show the output of the covariance matrix.

The following CMAT code shows how to create a feasible starting point $x0_j = 1/n$ and how to specify the lower and upper bounds `bc`, $0 \leq x_j \leq 1$, as well as the equality constraint `lc`, $\sum_{j=1}^{n} x_j = 1$, for the estimates:

```
/* Creating initial values and constraints */
/* 1. parms observation with initial values */
r = 1. / nc;
x0 = cons(1,nc,r);
/* 2. lower bounds: weights must be nonnegative */
/* 3. upper bounds: weights must be smaller than 1 */
bc = cons(nc,1,0.) -> cons(nc,1,1.);
/* 4. linear equality constraint: weights must sum up to 1 */
lc = cons(1,nc+2,1.);
```

Using the null space QP algorithm we run the constrained optimization:

```
print "Compute Minimum Risk Portfolio";
print "Without Additional Constraints on Minimum Return";
optn = [ "qpnusp"       ,
         "print"      3 ];
< xr, rp > = qp(cov,.,lc,optn,bc,x0);
print "XR=",xr;
print "RP=",rp;
if (rp[1] < 0) print "Error NLP:", rp[1];
```

The following output shows the definition of the QP:

```
                        ******************
                        Optimization Start
                        ******************

                       Parameter Estimates
                       -------------------

 Parameter      Estimate   Gradient   Lower BC   Upper BC

   1 X1       0.10000000  0.0022481  0.0000000  1.0000000
   2 X2       0.10000000  0.0041447  0.0000000  1.0000000
   3 X3       0.10000000  0.0014994  0.0000000  1.0000000
   4 X4       0.10000000  7.78e-004  0.0000000  1.0000000
   5 X5       0.10000000  0.0029964  0.0000000  1.0000000
   6 X6       0.10000000  0.0016441  0.0000000  1.0000000
   7 X7       0.10000000  0.0033524  0.0000000  1.0000000
   8 X8       0.10000000  0.0023685  0.0000000  1.0000000
   9 X9       0.10000000  0.0019405  0.0000000  1.0000000
```

```
 10 X10      0.10000000  0.0010951  0.0000000  1.0000000

          Value of Objective Function =   0.00110338

                        Linear Constraints
                        ------------------

[ 1]ACT 1.0000000 ==  +  1.00000 * X1       +  1.00000 * X2
                      +  1.00000 * X3       +  1.00000 * X4
                      +  1.00000 * X5       +  1.00000 * X6
                      +  1.00000 * X7       +  1.00000 * X8
                      +  1.00000 * X9       +  1.00000 * X10
                      ( -1e-016 )
```

The iteration history does not show any problems:

```
        Null Space Active Set Method of Quadratic Problem
                     Using Dense Hessian


   Iteration Start:
   N. Variables             10
   N. Bound. Constr.        20        N. Mask  Constr.          0
   N. Linear Constr.         1        Lin. Equ. Constr.         1
   Criterion       0.001103375        Max Grad Entry  0.001604353
   N. Active Constraints     1

  Iter rest nfun act   optcrit  difcrit maxgrad   alpha    slope
     1    0    2   2  6.3e-004 4.7e-004  5e-004 0.68599  -1e-003
     2    0    3   3  6.2e-004 1.7e-005  4e-004 0.19315  -1e-004
     3    0    4   4  6.0e-004 1.1e-005  2e-004 0.28289  -5e-005
     4    0    5   5  6.0e-004 1.9e-006  1e-004 0.15203  -1e-005
     5    0    6   6  6.0e-004 3.9e-006  2e-005 0.79126  -8e-006
     6    0    7   6  6.0e-004 1.2e-007  2e-019 1.00000  -2e-007

   Successful Termination After     6 Iterations
   ABSGCONV convergence criterion satisfied.
   Criterion       0.000598330        Max Grad Entry  6.5052e-019
   N. Active Constraints     6        Slope SDirect. -2.3209e-007
   N. Function Calls         9        N. Gradient Calls         7
   Preproces. Time           0        Time for Method           0
   Effective Time            0
```

We obtain the following optimal stock weights $\hat{x}_j$, which correspond to the minimum portfolio risk of $\sigma_p = \sqrt{2 * 0.0005983304}$:

```
                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

          Parameter      Estimate   Gradient   Active BC

            1 X1       0.00000000  0.0012921  Lower BC
            2 X2       0.00000000  0.0017323  Lower BC
            3 X3       0.13962066  0.0011967
            4 X4       0.33107360  0.0011967
            5 X5       0.00000000  0.0014926  Lower BC
            6 X6       0.23000302  0.0011967
            7 X7       0.00000000  0.0014646  Lower BC
            8 X8       0.00000000  0.0012159  Lower BC
            9 X9       0.17172687  0.0011967
           10 X10      0.12757585  0.0011967

          Value of Objective Function =   0.00059833


            Linear Constraints Evaluated at Solution
            ----------------------------------------

 [ 1]ACT    1.00000 * X1       +  1.00000 * X2
        +  1.00000 * X3       +  1.00000 * X4
        +  1.00000 * X5       +  1.00000 * X6
        +  1.00000 * X7       +  1.00000 * X8
        +  1.00000 * X9       +  1.00000 * X10      -  1.00000
        =  2.2204e-016


XR=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00000   0.00000   0.13962   0.33107   0.00000

   |         6         7         8         9        10
---------------------------------------------------------
 1 |   0.23000   0.00000   0.00000   0.17173   0.12758


RP=
```

```
          |         1
----------------------
  Failure |    3.0000
   F_Crit |   0.00060
   N_Iter |    6.0000
   N_Func |    9.0000
   N_Grad |    7.0000
  N_BCact |    5.0000
  N_LCact |   1.00000
   unused |    .
    G_Max |    7e-019
   O_Time |   0.00000
  FG_Time |   0.00000
   unused |   0.00000
```

To compute the expected portfolio return $R_p(\hat{x})$ for the minimum risk solution $\hat{x}$ we multiply the vector of optimal stock weights $\hat{x}_j$ with the average stock returns r_j

$$R_p(\hat{x}) = \sum_{j=1}^{n} \hat{x}_j r_j$$

We assume that the average returns of the past are the expected future (monthly) returns. (In reality, this can be questionable and points to the limits of the Markowitz model.)

```
print "Compute Expected Portfolio Return";
_rmax = mean[<>];
_retn = retn0 = mean * xr';
_risk = risk0 = sqrt(2. * rp[2]);
xr0 = xr;  res0 = [ 0. retn0  retn0  risk0 ];
print "Maximum Possible Expected Return=",_rmax;
print "Expected Return for Minimum Risk Solution=",_retn;
print "Optimal Risk Value for Minimum Risk Solution=",_risk;
```

For our example we obtain $R_p(\hat{x}) = 0.014137$.

```
Compute Expected Portfolio Return
Maximum Possible Expected Return= 0.02501
Expected Return for Minimum Risk Solution= 0.01414
Optimal Risk Value for Minimum Risk Solution= 0.03459
```

Since all stock weights x_j are between 0 and 1 and add up to 1, the expected portfolio return $R_p(\hat{x})$ for the minimum risk solution must be somewhere between $\min_j r_j$ and $\max_j r_j$

$$\min_j r_j \leq R_p(\hat{x}) \leq \max_j r_j$$

For our example we find $\min_j r_j = 0.00962$ and $\max_j r_j = 0.02501$ in the _TYPE_= MEAN observation of the OCOV data set.

1.4.3 Lower Bound on the Expected Portfolio Return

In our first application, we computed optimal weights $\hat{x}_j$ minimizing the portfolio risk without requiring a minimum acceptable level of expected portfolio return. We can, however, subject the minimization process to an additional linear inequality constraint that requires the expected portfolio return $R_p(x, pr)$ to be larger than or equal to a specified lower bound pr

$$R_p(x, pr) = \sum_{j=1}^{n} x_j r_j \geq pr$$

If the value of pr is chosen to be larger than $\max_j r_j = 0.02501$, no feasible solution to the problem exists. It also makes no sense to permit the expected portfolio return $R_p(x, pr)$ to be smaller than the one obtained by the minimum risk solution, $R_p(\hat{x})$. We therefore select

$$pr(perc) = R_p(\hat{x}) + perc * (\max_j r_j - R_p(\hat{x})) \quad \text{with} \quad 0 < perc < 1$$

We specify an additional linear (ineqality) constraint, which requires a minimum risk solution for a return corresponding to $perc = .5$. As starting point `x0` we use the former optimal point `xr` which is of course not feasible w.r.t. the linear inequality constraint, however, the QP call will generate a fdeasible starting point:

```
x0 = xr;
rhs = _retn + .5 *(_rmax - _retn);
lc2 = cons(2,nc+2); lc2[1,] = lc;
lc2[2,] = rhs -> mean -> .;
print "LC2=",lc2;
```

```
LC2=
   |         1         2         3         4         5         6
---------------------------------------------------------------------
 1 |   1.00000   1.00000   1.00000   1.00000   1.00000   1.00000
 2 |   0.01957   0.01543   0.02501   0.01523   0.01851   0.00963

   |         7         8         9        10        11        12
---------------------------------------------------------------------
 1 |   1.00000   1.00000   1.00000   1.00000   1.00000   1.00000
 2 |   0.00962   0.01975   0.01619   0.01194   0.01270     .
```

```
optn = [ "qpnusp"      ,
         "print"      3 ];
< xr, rp > = qp(cov,.,lc2,optn,bc,x0);
print "XR=",xr;
print "RP=",rp;
if (rp[1] < 0) print "Error NLP:", rp[1];
```

For $perc = .5$ (corresponding to $R_p = 0.01957$, which is half of the interval between $R_p(\hat{x}) = 0.014137$ and $\max_j r_j = 0.02501$), we obtain a solution with a slightly higher risk value $\sigma_p = \sqrt{2 * 0.001107}$ than in the first application ($\sigma_p = \sqrt{2 * 0.000598}$):

```
                          ******************
                          Optimization Start
                          ******************

                         Parameter Estimates
                         -------------------

 Parameter      Estimate   Gradient   Lower BC   Upper BC

   1 X1       0.15904963  0.0028156  0.0000000  1.0000000
   2 X2       0.33093189  0.0079084  0.0000000  1.0000000
   3 X3       0.00000000  0.0017789  0.0000000  1.0000000
   4 X4       0.04542447  6.13e-004  0.0000000  1.0000000
   5 X5       0.05517296  0.0040675  0.0000000  1.0000000
   6 X6       0.00000000  0.0021135  0.0000000  1.0000000
   7 X7       0.23662084  0.0054871  0.0000000  1.0000000
   8 X8       0.17280021  0.0031743  0.0000000  1.0000000
   9 X9       0.00000000  0.0023885  0.0000000  1.0000000
  10 X10      0.00000000  9.77e-004  0.0000000  1.0000000

           Value of Objective Function =   0.00258204


                          Linear Constraints
                          ------------------

 [ 1]ACT 1.0000000 ==  +  1.00000 * X1       +  1.00000 * X2
                       +  1.00000 * X3       +  1.00000 * X4
                       +  1.00000 * X5       +  1.00000 * X6
                       +  1.00000 * X7       +  1.00000 * X8
                       +  1.00000 * X9       +  1.00000 * X10
                       (  1e-016 )
 [ 2]ACT 0.0195728 <=  +  0.01543 * X1       +  0.02501 * X2
```

```
                  +  0.01523 * X3       +  0.01851 * X4
                  +   1e-002 * X5       +   1e-002 * X6
                  +  0.01975 * X7       +  0.01619 * X8
                  +  0.01194 * X9       +  0.01270 * X10
                  ( -2e-018 )


     Null Space Active Set Method of Quadratic Problem
                  Using Dense Hessian


 Iteration Start:
 N. Variables             10
 N. Bound. Constr.        20        N. Mask  Constr.          0
 N. Linear Constr.         2        Lin. Equ. Constr.         1
 Criterion       0.002582039        Max Grad Entry  0.003524371
 N. Active Constraints    5+

Iter rest nfun act   optcrit  difcrit maxgrad   alpha    slope
   1    0    2   4' 2.6e-003 2.6e-005  3e-003 1.00000  -3e-005
   2    0    3   5  2.5e-003 7.5e-005  3e-003 0.02271  -3e-003
   3    0    4   6  1.9e-003 5.9e-004  2e-003 0.20806  -3e-003
   4    0    5   7  1.4e-003 4.6e-004  2e-003 0.34514  -2e-003
   5    0    6   8  1.1e-003 3.1e-004  3e-004 0.74544  -7e-004
   6    0    7   8  1.1e-003 1.8e-005  4e-019 1.00000  -4e-005

 Successful Termination After     6 Iterations
 ABSGCONV convergence criterion satisfied.
 Criterion       0.001106983        Max Grad Entry  1.8431e-018
 N. Active Constraints     8        Slope SDirect. -3.5179e-005
 N. Function Calls         9        N. Gradient Calls         7
 Preproces. Time           0        Time for Method           0
 Effective Time            0


                  ********************
                  Optimization Results
                  ********************

                  Parameter Estimates
                  -------------------

       Parameter      Estimate   Gradient   Active BC

         1 X1       0.00000000  0.0011477  Lower BC
         2 X2       0.18393296  0.0039429
```

```
 3 X3       0.00000000  0.0013548  Lower BC
 4 X4       0.68348036  0.0018754
 5 X5       0.00000000  0.0018504  Lower BC
 6 X6       0.00000000  9.02e-004  Lower BC
 7 X7       0.04948184  0.0022703
 8 X8       0.08310484  0.0011385
 9 X9       0.00000000  5.93e-004  Lower BC
10 X10      0.00000000  0.0014942  Lower BC

Value of Objective Function =   0.00110698


  Linear Constraints Evaluated at Solution
  ----------------------------------------

[ 1]ACT    1.00000 * X1       +  1.00000 * X2
        +  1.00000 * X3       +  1.00000 * X4
        +  1.00000 * X5       +  1.00000 * X6
        +  1.00000 * X7       +  1.00000 * X8
        +  1.00000 * X9       +  1.00000 * X10      -  1.00000
        =  1.1102e-016
[ 2]ACT    0.01543 * X1       +  0.02501 * X2
        +  0.01523 * X3       +  0.01851 * X4
        +   1e-002 * X5       +   1e-002 * X6
        +  0.01975 * X7       +  0.01619 * X8
        +  0.01194 * X9       +  0.01270 * X10      -  0.01957
        = -3.2526e-018


XR=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00000   0.18393   0.00000   0.68348   0.00000

   |         6         7         8         9        10
---------------------------------------------------------
 1 |   0.00000   0.04948   0.08310   0.00000   0.00000


RP=
          |         1
----------------------
  Failure |    3.0000
   F_Crit |   0.00111
   N_Iter |    6.0000
   N_Func |    9.0000
```

```
 N_Grad |    7.0000
N_BCact |    6.0000
N_LCact |    2.0000
 unused |    .
  G_Max |    2e-018
 O_Time |   0.00000
FG_Time |   0.00000
 unused |   0.00000


 print "Compute Expected Portfolio Return";
 _retn = mean * xr';
 _risk = sqrt(2. * rp[2]);
 print "RETN=",_retn," RISK=",_risk;


Compute Expected Portfolio Return
RETN= 0.01957 RISK= 0.04705
```

The following CMAT module `effront` computes a so-called efficient frontier, which is a series of minimum risk solutions for a list of different *perc* values defining the interval of the expected portfolio return $R_p(x, pr)$. The arguments of the module define:

cov covariance matrix of the data,

mean mean vector of the data,

lst list of *perc* values, which should be between 0 and 1,

tech string scalar specifying the QP method,

prit int value specifying the amount of printed output.

In applications of the module, the values of the number list `lst` should be sorted in increasing order since the module uses the optimal parameter weights from one `qp` run as the starting values of the subsequent `qp` run:

```
function effront(cov,mean,lst,tech,prit)
{
  /*--- Compute Efficient Frontier ---*/
  nc = ncol(cov); nlst = ncol(lst);

  /* Creating initial values and constraints */
  /* 1. parms observation with initial values */
  r = 1. / nc;
  x0 = cons(1,nc,r);
```

```
/* 2. lower bounds: weights must be nonnegative */
/* 3. upper bounds: weights must be smaller than 1 */
bc = cons(nc,1,0.) -> cons(nc,1,1.);
/* 4. linear equality constraint: weights must sum up to 1 */
lc = cons(1,nc+2,1.);

print "Compute Minimum Risk Portfolio";
print "Without Constraints on Minimum Return";
optn = [ tech           ,
         "print"   prit ];
< xr, rp > = qp(cov,.,lc,optn,bc,x0);
print "XR=",xr;
print "RP=",rp;
if (rp[1] < 0) print "Error NLP:", rp[1];

/* Use OCOV (MEAN) and EST (PARMS) data sets */
print "Compute Expected Portfolio Return";
_imax = mean[<:>]; _rmax = mean[_imax];
_retn = retn0 = mean * xr`;
_risk = risk0 = sqrt(2. * rp[2]);
rhs0 = _retn + .5 *(_rmax - _retn);
xr0 = xr;  res0 = [ 0. retn0  retn0  risk0 ];
print "Maximum Possible Expected Return=",_rmax;
print "Expected Return for Minimum Risk Solution=",_retn;
print "Optimal Risk Value for Minum Risk Solution=",_risk;

/*-------------------------------------------------------------*/

lc2 = cons(2,nc+2); lc2[1,] = lc;
lc2[2,] = rhs0 -> mean -> .;
pret = cons(nlst+1,nc+4);
pret[1,] = res0 -> xr0;
for (ilst = 1; ilst <= nlst; ilst++) {
  x0 = xr; perc = lst[ilst];
  /* This is feasible starting point:
     x0 = cons(1,nc,0.); x0[_imax] = 1.;
     print "New X0=",x0; */
  rhs = retn0 + perc * (_rmax - retn0);
  lc2[2,1] = rhs;
  /* print "Perc=",perc," RHS=",rhs," Lc2=",lc2; */

  print "Lower Bound=",rhs," for Portfolio Return, Perc.=",perc;
  optn = [ tech           ,
           "lce"    1.e-6 ,
           "print"      0 ];
  < xr, rp > = qp(cov,.,lc2,optn,bc,x0);
```

```
    /* print "XR=",xr; print "RP=",rp; */
    if (xr == . || rp[1] < 0) {
      print "Error NLP:", rp[1];
      print "Old X0=",x0;
      res = [ perc rhs . . ];
      pret[ilst+1,] = res -> cons(1,nc,.);
      xr = cons(1,nc,0.); xr[_imax] = 1.;
      continue;
     }
    _retn = mean * xr';
    _risk = sqrt(2. * rp[2]);
    print "Restricted Minimum Risk Value=", _risk;
    print "Expected Return of Portfolio=", _retn;

    res = [ perc rhs _retn _risk ];
    pret[ilst+1,] = res -> xr;
  }
  return(pret);
}
```

We now want to plot the portfolio returns versus the risk levels for 21 optimal solutions, starting with the minimum risk solution for $perc = 0$, followed by 20 optimizations for the different values of required expected portfolio returns corresponding to the `lst` values $perc$ =0.05, 0.10, 0.15, ..., 1.0:

```
lst = [ .05 : .05 : 1. ]; nlst = ncol(lst);
tech = "qpnusp"; prit = 3;
eff1 = effront(cov,mean,lst,tech,prit);
eff1 = rname(eff1,rnam);
eff1 = cname(eff1,cnam);
print "Efficient Frontier=",eff1;
```

```
Efficient Frontier=
          |    Percent        RHS     Return       Risk     Gerber
----------------------------------------------------------------------
  MINRisk |    0.00000    0.01414    0.01414    0.03459    0.00000
    Risk1 |    0.05000    0.01468    0.01468    0.03473    0.00000
    Risk2 |    0.10000    0.01522    0.01522    0.03513    0.00000
    Risk3 |    0.15000    0.01577    0.01577    0.03575    0.00000
    Risk4 |    0.20000    0.01631    0.01631    0.03656    0.00290
    Risk5 |    0.25000    0.01686    0.01686    0.03757    0.00879
    Risk6 |    0.30000    0.01740    0.01740    0.03876    0.01469
    Risk7 |    0.35000    0.01794    0.01794    0.04014    0.01842
    Risk8 |    0.40000    0.01849    0.01849    0.04186    0.01860
```

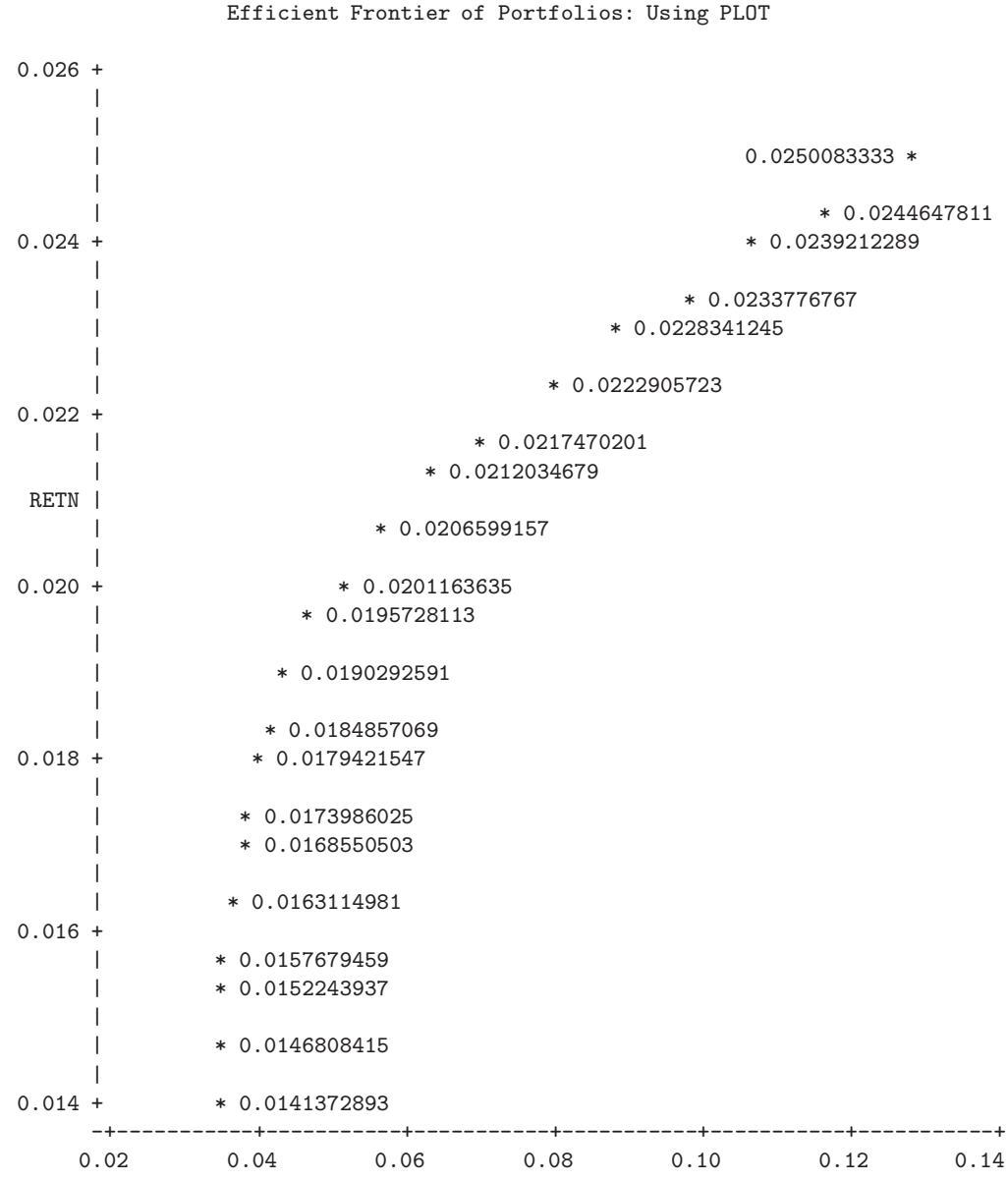
Risk9	0.45000	0.01903	0.01903	0.04400	0.01030
Risk10	0.50000	0.01957	0.01957	0.04705	0.00000
Risk11	0.55000	0.02012	0.02012	0.05127	0.00000
Risk12	0.60000	0.02066	0.02066	0.05655	0.00000
Risk13	0.65000	0.02120	0.02120	0.06310	0.00000
Risk14	0.70000	0.02175	0.02175	0.07073	0.00000
Risk15	0.75000	0.02229	0.02229	0.07918	0.00000
Risk16	0.80000	0.02283	0.02283	0.08821	0.00000
Risk17	0.85000	0.02338	0.02338	0.09766	0.00000
Risk18	0.90000	0.02392	0.02392	0.10741	0.00000
Risk19	0.95000	0.02446	0.02446	0.11740	0.00000
Risk20	1.0000	0.02501	0.02501	0.12757	0.00000

	TandyC	GenMills	CEdison	Weyerh	IBM
MINRisk	0.00000	0.13962	0.33107	0.00000	0.23000
Risk1	0.00000	0.14510	0.39391	0.00000	0.20065
Risk2	0.00000	0.14984	0.45189	0.00000	0.17127
Risk3	0.00233	0.14967	0.49822	0.00000	0.14038
Risk4	0.01608	0.13800	0.52448	0.00000	0.10190
Risk5	0.02967	0.12547	0.55006	0.00000	0.06304
Risk6	0.04327	0.11295	0.57563	0.00000	0.02418
Risk7	0.06179	0.09117	0.60366	0.00000	0.00000
Risk8	0.08843	0.05414	0.63573	0.00000	0.00000
Risk9	0.12514	0.00000	0.67885	0.00000	0.00000
Risk10	0.18393	0.00000	0.68348	0.00000	0.00000
Risk11	0.24489	0.00000	0.68294	0.00000	0.00000
Risk12	0.32603	0.00000	0.64789	0.00000	0.00000
Risk13	0.41464	0.00000	0.58536	0.00000	0.00000
Risk14	0.49826	0.00000	0.50174	0.00000	0.00000
Risk15	0.58188	0.00000	0.41812	0.00000	0.00000
Risk16	0.66551	0.00000	0.33449	0.00000	0.00000
Risk17	0.74913	0.00000	0.25087	0.00000	0.00000
Risk18	0.83275	0.00000	0.16725	0.00000	0.00000
Risk19	0.91638	0.00000	0.08362	0.00000	0.00000
Risk20	1.00000	0.00000	0.00000	0.00000	0.00000

	DEC	Mobil	Texaco	CPL
MINRisk	0.00000	0.00000	0.17173	0.12758
Risk1	0.00000	0.01904	0.16154	0.07977
Risk2	0.00226	0.04147	0.14849	0.03477
Risk3	0.01640	0.05408	0.13892	0.00000
Risk4	0.02424	0.07033	0.12206	0.00000

```
 Risk5 |    0.03222    0.08587    0.10487    0.00000
 Risk6 |    0.04019    0.10142    0.08768    0.00000
 Risk7 |    0.04509    0.12063    0.05924    0.00000
 Risk8 |    0.04495    0.14588    0.01228    0.00000
 Risk9 |    0.04763    0.13808    0.00000    0.00000
Risk10 |    0.04948    0.08310    0.00000    0.00000
Risk11 |    0.05155    0.02062    0.00000    0.00000
Risk12 |    0.02608    0.00000    0.00000    0.00000
Risk13 |    0.00000    0.00000    0.00000    0.00000
Risk14 |    0.00000    0.00000    0.00000    0.00000
Risk15 |    0.00000    0.00000    0.00000    0.00000
Risk16 |    0.00000    0.00000    0.00000    0.00000
Risk17 |    0.00000    0.00000    0.00000    0.00000
Risk18 |    0.00000    0.00000    0.00000    0.00000
Risk19 |    0.00000    0.00000    0.00000    0.00000
Risk20 |    0.00000    0.00000    0.00000    0.00000
```

Column 3 shows the expected portfolio return, column 4 the risk of the portfolio, and the columns 5 to 15 show the optimal weights for the 21 optimizations. This summary of results also indicates how the required increase in the portfolio return also increases the portfolio risk value and affects the diversity of the stock selection.


Efficient Frontier of Portfolios: Using PLOT
0.026
0.024
0.022
0.020
0.018
0.016
0.014
RETN
0.0250083333
0.0244647811
0.0239212289
0.0233776767
0.0228341245
0.0222905723
0.0217470201
0.0212034679
0.0206599157
0.0201163635
0.0195728113
0.0190292591
0.0184857069
0.0179421547
0.0173986025
0.0168550503
0.0163114981
0.0157679459
0.0152243937
0.0146808415
0.0141372893
0.02
0.04
0.06
0.08
0.10
0.12
0.14
RISK
CRSP Monthly Data, Jan 78 - Dec 87

1.4.4 Compute Tangential Portfolio

To compute the *Capital Market Line* and the *Tangential Portfolio* we need to specify the expected return R_f on the risk-free asset. For this example we use the mean value of past risk-free monthly returns which is $R_f = .007161$. There are many other ways to estimate the expected return R_f on the risk-free asset (Elton & Gruber, 1981). The *Capital Market Line* is defined as the line through the point of the risk-free asset which is tangent to the *Efficient Frontier*. The point where the capital market line touches the curve of the efficient frontier defines the *Tangential Portfolio*. We can use PROC NLP to compute the *Tangential Portfolio* point by maximizing the following (nonlinear) objective function

$$\max_x \frac{R_p(x) - R_f}{\sigma_p(x)}$$

where $R_p(x)$ is the *expected portfolio return*

$$R_p(x) = \sum_{j=1}^{n} x_j r_j$$

and $\sigma_p(x)$ is the *expected portfolio risk*

$$\sigma_p(x) = \sqrt{\sum_{j=1}^{n}\sum_{k=1}^{n} \sigma_{jk} x_j x_k}$$

We have written the following module `tanport` which uses the earlier generated data and creates a return object `tanp` containing the two points of the risk-free and tangential portfolios.
The following input arguments must be specified:

cov covariance matrix of the data,

mean mean vector of the data,

pret the return table from the `effront` call.

R_f the expected risk-free return R_f,

tech string scalar specifying the QP method,

prit int value specifying the amount of printed output.

Note, that the `nlp` function in `tanport` is using finite difference (numerical) first and second order derivatives and may be computational expensive for very large applications.

```
function tanport(pret,tech,prit) global(cov,mean,R_f)
{
  /*--- Compute Tangential Portfolio ---*/
  nc = ncol(cov); nlst = nrow(pret);
  tanp = cons(2,nc+3); nc4 = nc + 4;
  ind = [ 1 3 4 ] -> [ 5 : nc4 ];
  tanp[1,] = pret[1,ind];
```

```
  /* Creating initial values and constraints */
  nint = nlst / 2; nint = (int)nint;
  nc4 = nc+4; ind = [ 5 : nc4 ];
  x0 = pret[nint,ind];
  bc = cons(nc,1,0.) -> cons(nc,1,1.);
  lc = cons(1,nc+2,1.);

  function ftang(x) global(cov,mean,R_f)
  {
    sigm = sqrt(x * cov * x');
    R_p = x * mean';
    tangp = (R_p - R_f) / sigm;
    return(tangp);
  }

  mopt = [ "tech"      tech ,
           "max"            ,
           "print"     prit ];
  < xr,rp > = nlp(ftang,x0,mopt,bc,lc);
  print "XR=",xr; print "RP=",rp;

  if (rp[1] < 0) {
    print "Error NLP:", rp[1];
    tanp = .;
  } else {
    _risk = sqrt(xr * cov * xr');
    _retn = mean * xr';
    print "Tangential Portfolio Risk= ", _risk;
    print "Tangential Portfolio Return= ", _retn;
    tanp[2,] = [ R_f _retn _risk ] -> xr;
  }
  return(tanp);
}
```

Note also, that the global arguments of the inner function `ftang` must be global arguments of the outer function. The following call of the module `tanport` generates the tangential portfolio:

```
pret = eff1;  R_f = .007161;
tech = "trureg"; prit = 3;
tanp = tanport(pret,tech,prit);
```

The following shows the output of the `tanport` call:

```
                    ******************
                    Optimization Start
                    ******************
```

```
                    Parameter Estimates
                    -------------------

Parameter      Estimate   Gradient   Lower BC   Upper BC

  1 X_1      0.00000000  0.1910605  0.0000000  1.0000000
  2 X_2      0.18393296  0.0617060  0.0000000  1.0000000
  3 X_3      0.00000000  0.1623224  0.0000000  1.0000000
  4 X_4      0.68348036  0.1699034  0.0000000  1.0000000
  5 X_5      0.00000000 -0.0157392  0.0000000  1.0000000
  6 X_6      0.00000000  0.0968653  0.0000000  1.0000000
  7 X_7      0.04948184  0.1492296  0.0000000  1.0000000
  8 X_8      0.08310484  0.2084576  0.0000000  1.0000000
  9 X_9      0.00000000  0.1831224  0.0000000  1.0000000
 10 X_10     0.00000000  0.0918813  0.0000000  1.0000000

          Value of Objective Function =     0.263785


                     Linear Constraints
                     ------------------

[ 1]ACT 1.0000000 ==  +  1.00000 * X_1      +  1.00000 * X_2
                      +  1.00000 * X_3      +  1.00000 * X_4
                      +  1.00000 * X_5      +  1.00000 * X_6
                      +  1.00000 * X_7      +  1.00000 * X_8
                      +  1.00000 * X_9      +  1.00000 * X_10
                      (  4e-016 )


                  Trust Region Optimization
                  Without Parameter Scaling
            Gradient Computed by Finite Differences
         Hessian Computed by Finite Differences (dense)
                  (Using Only Function Calls)

   Iteration Start:
   N. Variables            10
   N. Bound. Constr.       20       N. Mask  Constr.          0
   N. Linear Constr.        1       Lin. Equ. Constr.         1
   Criterion       0.263784800       Max Grad Entry  0.076507120
   N. Active Constraints    6+       TR Radius       1.000000000


  Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
     1    0    2   5' 0.268550 4.8e-003 0.04861 0.00000  1.00000
     2    0    3   5  0.270668 2.1e-003  2e-003 0.00000  0.15782
     3    0    4   5  0.270673 4.6e-006  1e-005 0.00000  0.06824

   Successful Termination After     3 Iterations
```

```
ABSGCONV convergence criterion satisfied.
Criterion       0.270672968        Max Grad Entry  1.0801e-005
N. Active Constraints     5        Ridge (lambda)  0.000000000
TR Radius       0.068237469        Act.dF/Pred.dF  1.004309568
N. Function Calls         5        N. Gradient Calls         2
N. Hessian Calls          5        Preproces. Time           0
Time for Method           1        Effective Time            1


                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

          Parameter      Estimate   Gradient   Active BC

            1 X_1      0.01836736  0.1687852
            2 X_2      0.09657008  0.1688005
            3 X_3      0.04253182  0.1687808
            4 X_4      0.64558689  0.1687909
            5 X_5      0.00000000 -0.0233571  Lower BC
            6 X_6      0.00000000  0.1031017  Lower BC
            7 X_7      0.04502064  0.1687927
            8 X_8      0.15192322  0.1687875
            9 X_9      0.00000000  0.1676175  Lower BC
           10 X_10     0.00000000  0.0864063  Lower BC

          Value of Objective Function =     0.270673


            Linear Constraints Evaluated at Solution
            ----------------------------------------

 [ 1]ACT    1.00000 * X_1      +  1.00000 * X_2
        +  1.00000 * X_3      +  1.00000 * X_4
        +  1.00000 * X_5      +  1.00000 * X_6
        +  1.00000 * X_7      +  1.00000 * X_8
        +  1.00000 * X_9      +  1.00000 * X_10     -  1.00000
        =  4.7184e-016


XR=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.01837   0.09657   0.04253   0.64559   0.00000

   |         6         7         8         9        10
---------------------------------------------------------
```

```
 1 |   0.00000   0.04502   0.15192   0.00000   0.00000
```

```
RP=
          |         1
----------------------
  Failure |    3.0000
   F_Crit |   0.27067
   N_Iter |    3.0000
   N_Func |    5.0000
   N_Grad |    2.0000
   N_Jacm |   0.00000
   N_Hess |    5.0000
   N_Fnlc |   0.00000
   N_Jnlc |   0.00000
   O_Time |   1.00000
   FG_Tim |    2.0000
   unused |   0.00000
```

```
Tangential Portfolio Risk=  0.04242
Tangential Portfolio Return=  0.01864
```

```
   rnam2 = [ "MINRisk" "TangPort" ];
   cnam2 = [" Percent Return Risk "] -> vnam;
   tanp = rname(tanp,rnam2);
   tanp = cname(tanp,cnam2);
   print "Tangential Portfolio", tanp;
```

```
Tangential Portfolio
           |   Percent    Return      Risk        X1        X2
-------------------------------------------------------------
   MINRisk |   0.00000   0.01146   0.22606   0.20301   0.13301
  TangPort |   0.00716   0.28513   0.34513   0.27895   0.00000

           |        X3        X4        X5        X6        X7
-------------------------------------------------------------
   MINRisk |   0.05669   0.14032   0.17784   0.00000   0.09447
  TangPort |   0.00000   0.28305   0.01474   0.03295   0.23051

           |        X8        X9       X10
-----------------------------------------
   MINRisk |   0.07781   0.00000   0.11685
  TangPort |   0.01371   0.14609   0.00000
```

1.5 Fitting Quantal Response Models

This section is based on the work of Ying So and it was presented at our tutorial at the 1995 ASA joint meeting in Orlando. This example can be found in `tnlp212.inp`.

1.5.1 General Remarks

Quantal response data (sometimes called dose-response data, or quantal assay data) may arise in a variety of different areas. For example in efficacy studies of a drug, subjects are observed if they have a response (that is, a positive outcome) from using the drug.

A two parameter model such the logit model is often used for quantal response data. However, low dose extrapolation based on the logit model is often unsatisfactory, and models with extra parameters are often preferred. The Aranda-Ordaz (1981) asymmetric model and the quantit model of Copenhaver and Mielke (1977) are three parameter models that contain the logit model as a special case. The probability $P(x)$ of response to dose x is given by

- Logit Model

$$P(x) = \frac{1}{1 + \mathrm{e}^{-(\alpha+\beta x)}}$$

- Aranda-Ordaz Asymmetric Model

$$P(x) = \begin{cases} 1 - (1 + \lambda \mathrm{e}^{\alpha+\beta x})^{-1/\lambda} & \text{if } \lambda \mathrm{e}^{\alpha+\beta x} > -1 \\ 1 & \text{otherwise} \end{cases}$$

$$\lambda = 1 \quad \Longrightarrow \quad P(x) = \frac{1}{1+\mathrm{e}^{-(\alpha+\beta x)}}$$

- Quantit Model

$$\int_{0.5}^{P(x)} \frac{dz}{1 - |2z - 1|^{\nu+1}} = \alpha + \beta x, \quad \nu > -1$$

$$\nu = 1 \quad \Longrightarrow \quad P(x) = \frac{1}{1+\mathrm{e}^{-4(\alpha+\beta x)}}$$

Here, α and β are the location and scale parameters. The Aranda-Ordaz model has an extra parameter λ, and the model treats responses and nonresponses asymmetrically. The quantit model has an extra shape parameter ν. The underlying distribution for the quantit model is called the *omega* distribution, which is a symmetric distribution and includes the double-exponential (ν=0), logistic ($\nu = 1$), and uniform (ν=∞) distributions as special cases (Morgan 1992).

For quantal response data, each observation corresponds to a test dose; that is, for the ith dose, x_i is given to n_i subjects of whom r_i respond. Parameters in the quantal response models are estimated by maximizing the log likelihood for the data

$$\text{log likelihood} = \sum_i \{r_i \log(P(x_i)) + (n_i - r_i) \log(1 - P(x_i))\}$$

The following data (Martin, 1942) are used to illustrate the fitting of these models using CMAT.

Log Concentration (x_i)	Number of Subjects (n_i)	Number Responded (r_i)
0.71	49	16
1.00	48	18
1.31	48	34
1.48	49	47
1.61	50	47
1.70	48	48

The `nlp` function can be used for fitting:

1. the logit model: unconstrained fit
2. the Aranda-Ordaz model with a nonlinear constraint specification
3. the quantit model: boundary constrained fit where the objective function applies the `quad` function in a simple iterative method for solving an inverse problem.

1.5.2 Fitting the Logit Model

The parameters α and β are estimated by maximizing the log likelihood

$$l_1(\alpha, \beta) = \sum_i \{r_i \log(P_i) + (n_i - r_i) \log(1 - P_i)\}$$

where

$$P_i = \frac{1}{1 + \mathrm{e}^{-\alpha - \beta x_i}}$$

The log likelihood can be written as

$$l_1(\alpha, \beta) = \sum_{i=1}^{k} \{r_i(\alpha + \beta x_i) - n_i \log(1 + \mathrm{e}^{\alpha + \beta x_i})\}$$

Note that there are no boundary, linear, or nonlinear constraints in the optimization. Let $(\alpha_0, \beta_0) = (0, 0)$ be the starting value. The quasi-Newton algorithm is used for the optimization. The convariance matrix is obtained as the inverse of the negative Hessian evaluated at the maximum likelihood estimates.
The CMAT code for fitting the logit model is the following:

```
assay = [ 1   .71  49  16 ,
          2  1.00  48  18 ,
          3  1.31  48  34 ,
          4  1.48  49  47 ,
          5  1.61  50  47 ,
          6  1.70  48  48 ];
cnam = [" obs x n r "];
assay = cname(assay,cnam);

function f212_3(x) global(assay) {
  t = x[1] + x[2] * assay[,2];
  t1 = assay[,4] .* t;
  t2 = assay[,3] .* log(1. + exp(t));
```

```
  loglik = t1 - t2;
  f = loglik[+];
  return(f);
}

print "Call NLP to maximize the log likelihood";
x0 = [ 0. 0. ];
mopt = [ "tech"    "trureg" ,
         "max"              ,
         "print"          4 ];
< mle,rp > = nlp(f212_3,x0,mopt);

pnam = [" Alpha Beta "];
mle = cname(mle,pnam);
print "Xopt=", mle;  print "RP=", rp;
```

Results of the model fit are

- table of starting values:

```
                          ******************
                          Optimization Start
                          ******************

                         Parameter Estimates
                         -------------------

                   Parameter     Estimate   Gradient

                    1 X_1      0.00000000  64.000000
                    2 X_2      0.00000000  110.58500

              Value of Objective Function =     -202.399
```

- optimization history:

```
                      Trust Region Optimization
                      Without Parameter Scaling
                 Gradient Computed by Finite Differences
           Hessian Computed by Finite Differences (dense)
                    (Using Only Function Calls)

      Iteration Start:
      N. Variables             2
      Criterion      -202.3989767        Max Grad Entry  110.5850010
      TR Radius       1.000000000


     Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
        1    0    2   0 -150.5319 51.86712 22.2403 13.5561  1.00000
        2    0    4   0 -123.5601 26.97172 9.33134 0.93133  3.20091
        3    0    5   0 -119.2234 4.336748 2.18167 0.00000  3.26565
```

```
  4    0    6   0 -119.0934 0.129987 0.11348 0.00000  1.79683
  5    0    7   0 -119.0931 2.6e-004  3e-004 0.00000  0.33736
  6    0    8   0 -119.0931 1.7e-009  9e-007 0.00000  0.01459

Successful Termination After     6 Iterations
GCONV convergence criterion satisfied.
Criterion      -119.0931376        Max Grad Entry  8.7478e-007
Ridge (lambda)  0.000000000        TR Radius       0.014589254
Act.dF/Pred.dF  1.010888714
N. Function Calls         9        N. Gradient Calls         2
N. Hessian Calls          8        Preproces. Time           0
Time for Method           0        Effective Time            0
```

- The maximum likelihood estimates:

```
                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

               Parameter     Estimate   Gradient

                1 X_1     -4.45093726 -8.75e-007
                2 X_2      4.46023238  0.0000000

          Value of Objective Function =     -119.093

                         Hessian Matrix
                         **************

                Symmetric Matrix: Dense Storage

                     S |           1          2
                     --------------------------
                     1 |  -37.350524 -42.606529
                     2 |  -42.606529 -52.365980
```

- Approximate standard errors and correlations:

```
  sopt = [ "grad" "hess" ];
  < gopt,hopt > = fider(f212_3,mle,sopt);
  print "gopt=", gopt;
  print "hopt=", hopt;

  v = inv(-hopt);
  ase = sqrt(dia2vec(v)); tab = mle' -> ase;
  tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);
  d = diag(1. / ase);
  corr = d * v * d;

  print "*** Logit Model ***";
  print "Estimates:",mle;
  print "Standard Errors:",tab;
  print "Correlations:",corr;
```

```
Standard Errors:
        |   Estimate     STDERR
------------------------------
  Alpha |    -4.4509    0.61033
   Beta |     4.4602    0.51545

Correlations:
 S |         1         2
```

```
------------------------
 1 |   1.00000
 2 |  -0.96339    1.0000
```

1.5.3 Fitting the Aranda-Ordaz Asymmetric Model

The parameters α, β, and λ are estimated by maximizing the log likelihood

$$l_2(\alpha, \beta, \lambda) = \sum_i \{r_i \log(P_i) + (n_i - r_i) \log(1 - P_i)\}$$

where

$$P_i = \begin{cases} 1 - (1 + \lambda e^{\alpha + \beta x_i})^{-1/\lambda} & \text{if } \lambda e^{\alpha+\beta x_i} > -1 \\ 1 & \text{otherwise} \end{cases}$$

For k doses (observations), there are $3k$ nonlinear constraints:

$$\begin{aligned} \lambda e^{\alpha+\beta x_i} &> -1 \\ P_i &> .000001 \\ 1 - P_i &> .000001, \quad i = 1, \ldots, k \end{aligned}$$

Nonlinear constraints are defined as a multi-valued function of the parameters such that the nonlinear constraints are equivalent to having function values greater than or equal to zero. The nonlinear constraints can be expressed as

$$\left\{ \begin{array}{c} 1 + \lambda e^{\alpha+\beta x_i} > 0. \\ P_i - .000001 > 0. \\ .999999 - P_i > 0. \end{array} \right\}$$

and the nonlinear constraints function contains the expression to the left of the inequality sign.

Let $(\alpha_0, \beta_0, \lambda_0) = (-4.5, 4.5, 1)$ be the starting value. Note that this corresponds to the maximium likelihood estimates for the logit model. Again, the quasi-Newton algorithm is used for the optimization.

The CMAT code for fitting the Aranda-Ordaz model is the following:

```
function f212_4(x) global(assay) {
  c = 1. + x[3] .* exp(x[1] + x[2] * assay[,2]);
  f = .;
  if (c > 1.e-8) {
    tt = -1. / x[3]; q = c .** tt;
    if (q > 1.e-6 && q < .999999) {
      t1 = assay[,4] .* log(1. - q);
      t2 = (assay[,3] - assay[,4]) .* log(q);
      loglik = t1 + t2;
      f = loglik[+];
  } }
  return(f);
}

/* define nonlinear constraints */
function c212_4(x) global(assay) {
```

```
  c = 1. + x[3] .* exp(x[1] + x[2] * assay[,2]);
  if (c <   1.e-8) c = 1.e-8;
  tt = -1. / x[3]; q = c .** tt;
  if (q <   1.e-6) q = 1.e-6;
  if (q > .999999) q = .999999;
  c0 = c - 1.e-8;
  c1 = q - 1.e-6;
  c2 = .999999 - q;
  con = c0 |> c1 |> c2;
  return(con);
}

print "Call NLP to maximize the log likelihood";
x0 = [ -4.5 4.5 1. ];
mopt = [ "tech"    "qadpen" ,
         "max"              ,
         "print"          4 ];
< xr,rp > = nlp(f212_4,x0,mopt,.,.,c212_4);
print "Xopt=", xr;    print "RP=", rp;
```

Results of the model fit are

- Table of starting values:

```
                          ******************
                          Optimization Start
                          ******************

                         Parameter Estimates
                         -------------------

                   Parameter     Estimate   Gradient

                    1 X_1     -4.50000000  0.1378881
                    2 X_2      4.50000000  0.0084796
                    3 X_3      1.00000000 -3.1560483

                Value of Objective Function =     -119.096

                     Values of Nonlinear Constraints
                     -------------------------------

                       Constraint   Residual
                           1  1    1.2711725
                           2  2    2.0000000
                           3  3    5.0349746
                           4  4    9.6711376
                           5  5    16.564614
                           6  6    24.336065
                           7  7    0.7866743
                           8  8    0.4999990
```

```
 9  9    0.1986097
10 10    0.1033995
11 11    0.0603687
12 12    0.0410903
13 13    0.2133237
14 14    0.4999990
15 15    0.8013883
16 16    0.8965985
17 17    0.9396293
18 18    0.9589077
```

- The optimization history:

```
Quadratic Penalty Method (Gould, 1989)
Gradient Computed by Finite Differences
Hessian Computed by Finite Differences (dense)
(Using Only Function Calls)
Jacobian Nonlinear Constraints (dense)
(Computed by Finite Differences)


 Iteration Start:
 N. Variables              3
 N. NonLin Constr.        18        NonLin EquConstr.         0
 Criterion      -119.0963570        Max Grad Entry  0.000000000
 N. Active Constraints     0        Max Const Viol. 0.000000000


Iter  act nfun        mu optcrit difcrit  conmax maxgrad  alpha
   0   0     2 0.1000000-119.096 0.00000 0.00000 3.15605 0.0000
       0     7 ridge    -117.344 1.75189 0.00000 5.22057 0.2476
       0     9 ridge    -115.629 1.71590 0.00000 19.0414 0.1000
       0    10 ridge    -115.032 0.59684 0.00000 14.1900 1.0000
       0    11 ridge    -114.959 0.07227 0.00000 8.00062 2.0000
       0    14 ridge    -114.928 0.03144 0.00000 4.76793 1.4797
       0    16 ridge    -114.912 0.01634 0.00000 0.22325 1.1159
       0    18 ridge    -114.912  7e-005 0.00000  2e-004 1.0024
   1   0*   19 0.0010000-114.912 4.18474 0.00000  1e-006 1.0000
   2   0*   20 1.00e-005-114.912 -9e-012 0.00000  2e-006 1.0000
   3   0*   21 1.00e-007-114.912  2e-012 0.00000  3e-006 1.0000
       0    37          -114.912  1e-014 0.00000  3e-006 6e-004
       0-   66          -114.912 0.00000 0.00000  3e-006 0.0000
   4   0*   67 1.00e-008-114.912 -7e-012 0.00000  1e-006 1.0000
       0    76          -114.912  7e-012 0.00000  2e-006 1.3173
       0-  111          -114.912 0.00000 0.00000  2e-006 0.0000

 Successful Termination After     4 Iterations
 Criterion      -114.9116125        Max Grad Entry  0.000000000
 N. Active Constraints     0
 N. Function Calls       112        N. Gradient Calls        23
 N. Hessian Calls         17        N. Line Searches         13
```

```
Preproces. Time           0        Time for Method          1
Effective Time            1
```

- maximium likelihood estimates:

```
                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

                 Parameter     Estimate   Gradient

                  1 X_1     -2.63984895 -7.86e-007
                  2 X_2      2.02708665 -1.58e-006
                  3 X_3     -0.40770962 -6.77e-007

               Value of Objective Function =     -114.912

                  Values of Nonlinear Constraints
                  -------------------------------

                    Constraint   Residual
                        1  1    0.8772760
                        2  2    0.7790817
                        3  3    0.5858659
                        4  4    0.4154779
                        5  5    0.2392416
                        6  6    0.0869838
                        7  7    0.7253176
                        8  8    0.5421033
                        9  9    0.2694456
                       10 10    0.1159842
                       11 11    0.0299531
                       12 12    0.0025036
                       13 13    0.2746804
                       14 14    0.4578947
                       15 15    0.7305524
                       16 16    0.8840138
                       17 17    0.9700449
                       18 18    0.9974944

                           Hessian Matrix
                           **************

                  Symmetric Matrix: Dense Storage

                 S |           1          2          3
                 ------------------------------------
                 1 |  -335.04088 -488.86776  362.57947
                 2 |  -488.86776 -734.79185  567.90034
```

```
3 |   362.57947  567.90034 -472.60286
```

Since none of the nonlinear constraints is active at the maximum liklihood estimates, the covariance matrix can be approximated by the inverse of the negative Hessian evaluated at the maximum likelihood estimates. The IML code for computing the standard errors and correlations is similiar to that for the logit model, and is given as follows:

```
print "Compute Standard Errors and COV Matrix";

sopt = [ "grad" "hess" ];
< gopt,hopt > = fider(f212_4,mle,sopt);
print "gopt=", gopt;
print "hopt=", hopt;

v = inv(-hopt); ase = sqrt(dia2vec(v));
ase = sqrt(dia2vec(v)); tab = mle‘ -> ase;
tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);
d = diag(1. / ase);
corr = d * v * d;

print "*** Aranda-Ordaz Model ***";
print "Estimates:",mle;
print "Standard Errors:",tab;
print "Correlations:",corr;
```

The approximate standard errors and correlations are:

```
Standard Errors:
         |   Estimate     STDERR
---------------------------------
   Alpha |    -2.6398    0.58885
    Beta |     2.0271    0.61358
  Lambda |   -0.40771    0.31743

Correlations:
 S |         1         2         3
-----------------------------------
 1 |   1.00000
 2 |  -0.97431   1.00000
 3 |  -0.83989   0.93610   1.00000
```

1.5.4 Fitting the Quantit Model

The parameters α, β, and ν are estimated by maximizing the log likelihood

$$l_3(\alpha,\beta,\nu) = \sum_i \{r_i \log(P_i) + (n_i - r_i)\log(1 - P_i)\}$$

where

$$\int_{0.5}^{P_i} \frac{dz}{1 - |2z-1|^{\nu+1}} = \alpha + \beta x_i, \quad \nu > -1$$

Note that there is no closed form for P_i. The evaluation of P_i is carried out by an iterative Newton scheme, which is nested within the global iteration of (α, β, ν).
Suppose α, β, and ν are known. By a change of variable $t = 2z$,

$$P_i = \begin{cases} .5 + .5u & \text{if} \quad \alpha + \beta x_i \geq 0 \\ .5 - .5u & \text{if} \quad \alpha + \beta x_i < 0 \end{cases}$$

where u is the solution of the equation

$$\int_0^u \frac{.5dt}{1 - t^{\nu+1}} = |\alpha + \beta x_i|$$

Let

$$g(u) = \int_0^{u_i} \frac{.5dt}{1 - t^{\nu+1}} \quad \text{and} \quad c_i = |\alpha + \beta x_i|$$

The derivative of $g(u)$ is $g'(u) = \frac{.5}{1-u^{\nu+1}}$. IML has a subroutine called QUAD that computes definite integrals. With QUAD you can solve $g(u) = c_i$ by an iterative Newton scheme. With a starting value u_0, the solution u is obtained iteratively:

$$u_{m+1} = u_m - \frac{g(u_m) - c_i}{g'(u_m)}$$

Iteration stops when u_{m+1} is sufficiently close to u_m (for example, $|u_{m+1} - u_m|$ ¡ 1e-6). The solution of $g(u) = c_i$ is $u = u_{m+1}$.
The starting value $(\alpha_0, \beta_0, \nu_0) = (-1, 1, 1)$ is used, which corresponds roughly to the maximum likelihood estimates for the logit model. Again, the quasi-Newton algorithm is used for the optimization. The boundary constraint of $\nu > 1$ is imposed in the estimation.
The CMAT code for fitting the Quantit model is the following:

```
/* integrand function */
function qf0(y) global(nup1) {
  v = .5 / (1. - y .** nup1);
  return(v);
}
```

The response probability $P(x)$ in the quantit model does not have a closed form. The function module for the quantit model

```
function fquant(x) global(ipr,assay,nup1) {
  nobs = nrow(assay);
  logl = 0.; nup1 = x[3] + 1;
  for (i = 1; i <= nobs; i++) {
    /* g(u)=int(0 to u)dt/(1-t**(nu+1)) */
    /* solve u when g(u)=alpha + beta*y */
    hx = assay[i,2]; hn = assay[i,3]; hr = assay[i,4];
    c = x[1] + x[2] * hx;
    if (ipr) print "next obs: i,c=",i,c;
    flag = 0;
    if (c < 0) { c = -c; flag = 1; }
    if (c < 1.e-10) p = .5;
    else {
```

```
      /* Newton method to solve u iteratively:
         adjust u so that area z matches c */
      err = 1;
      u = 0.99999999;
      while (abs(err) > 1e-6) {
        /* call quad(z,"qf0",{0.}||u) msg='no'; */
        ab = [ 0. u ];
        optn = cons(6,1,.);
        optn[1] = 1.e-7; optn[2] = 1.e-3;
        z = quad(qf0,ab,optn);
        /* optn = [ 1.e-8 1.e-6 . 1 ];
           zv = quad(qf0,ab,optn);
           z = zv[1]; aer = zv[2]; nfun = zv[3]; */
        err = z - c;
        der = qf0(u);
        u = u - err / der;
        if (ipr) print "err,area,der,u=",err,z,der,u;
      }
      p= .5 *(1 + u);
      if (flag) p = 1. - p;
    }
    t1 = hr * log(p);
    t2 = (hn - hr) * log(1.-p);
    if (ipr) print "result: i,p,t1,t2=",i,p,t1,t2;
    logl += t1 + t2;
  }
  return(logl);
}

print "Call NLP to maximize the log likelihood";
x0 = [ -1. 1. 1. ];  nup1 = x0[3] + 1.;
bc = cons(3,2,.); bc[3,1] = -1.;

ipr = 1;
f0 = fquant(x0);
print "F0=",f0;
ipr = 0;

mopt = [ "tech"    "trureg" ,
         "max"              ,
         "print"          4 ];
< mle,rp > = nlp(fquant,x0,mopt,bc);
mle = cname(mle,pnam);
print "Xopt=", mle;    print "RP=", rp;
```

Results of the quantit model fit are

- Table of starting values:

```
                          ******************
                          Optimization Start
```

```
******************

Parameter Estimates
-------------------

Parameter     Estimate   Gradient   Lower BC   Upper BC

 1 X_1     -1.00000000  12.414603     .          .
 2 X_2      1.00000000  21.773557     .          .
 3 X_3      1.00000000  3.0910907 -1.0000000     .

           Value of Objective Function =     -119.628
```

- Optimization history: trust region algorithm:

```
                Trust Region Optimization
                Without Parameter Scaling
          Gradient Computed by Finite Differences
       Hessian Computed by Finite Differences (dense)
                (Using Only Function Calls)


  Iteration Start:
  N. Variables             3
  N. Bound. Constr.        1        N. Mask  Constr.          0
  Criterion      -119.6281315       Max Grad Entry  21.77355862
  N. Active Constraints    0        TR Radius       1.000000000


 Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
    1    0    2   0 -118.1159 1.512216 1.36995 0.00000  1.00000
    2    0    5   0 -117.4469 0.669010 5.40781 0.00000  2.00775
   3*    0    7   0 -116.6419 0.804961 14.4504 1.06281  1.88212
    4    0    8   0 -116.2851 0.356869 0.36654 0.00000  1.88344
    5    0    9   0 -116.1738 0.111307 2.88726 0.00000  1.45282
    6    0   10   0 -116.1438 0.029981 0.75527 0.00000  1.58655
    7    0   11   0 -116.1377 6.1e-003 0.36460 0.00000  1.17163
    8    0   12   0 -116.1370 6.8e-004 0.07157 0.00000  0.80234
    9    0   13   0 -116.1370 1.9e-005  3e-003 0.00000  0.36866
   10    0   14   0 -116.1370 2.5e-008  3e-006 0.00000  0.07258

  Successful Termination After    10 Iterations
  GCONV convergence criterion satisfied.
  Criterion      -116.1369996       Max Grad Entry  3.1598e-006
  N. Active Constraints    0        Ridge (lambda)  0.000000000
  TR Radius       0.072584002       Act.dF/Pred.dF  0.972311948
  N. Function Calls        15       N. Gradient Calls         2
  N. Hessian Calls         12       Preproces. Time           3
  Time for Method          27       Effective Time           33
```

- Optimization history: Quasi Newton algorithm: This run is about twice as fast as trust region, since quasi Newton does not need to compute the Hessian

matrix. The computing time, however, could be reduced even more by slightly changing the CMAT code.

```
                Dual Quasi-Newton Optimization
    Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
              Gradient Computed by Finite Differences


    Iteration Start:
    N. Variables              3
    N. Bound. Constr.         1        N. Mask  Constr.          0
    Criterion      -119.6281315        Max Grad Entry  21.77355719
    N. Active Constraints     0


   Iter rest nfun act   optcrit  difcrit maxgrad   alpha    slope
      1    0    4   0 -119.4155 0.212638 4.44892  3e-003 -146.454
      2    0    5   0 -119.0164 0.399091 1.71149 0.10000 -5.90599
      3    0    7   0 -117.4285 1.587894 21.9755 4.73012 -0.60501
      4    0    8   0 -116.9244 0.504062 11.8815 1.00000 -0.98625
      5    0   10   0 -116.7344 0.190039 12.0442 0.51747 -0.73094
      6    0   12   0 -116.3257 0.408689 18.4741 2.61262 -0.36370
      7    0   14   0 -116.2117 0.114066 3.83224 1.46279 -0.15546
      8    0   16   0 -116.1691 0.042518 1.93490 1.42499 -0.04983
      9    0   17   0 -116.1505 0.018651 3.78431 3.16786 -0.03355
     10    0   18   0 -116.1440 6.4e-003 2.20529 1.99705 -0.02335
     11    0   19   0 -116.1372 6.9e-003 0.19143 1.15460 -0.01393
     12    0   21   0 -116.1370 1.5e-004 0.06318 1.00000  -3e-004
     13    0   23   0 -116.1370 9.8e-007 0.03517 1.00000  -3e-006

    Successful Termination After    13 Iterations
    GCONV convergence criterion satisfied.
    Criterion      -116.1369999        Max Grad Entry  0.035167085
    N. Active Constraints     0        Slope SDirect. -2.7794e-006
    N. Function Calls        24        N. Gradient Calls        17
    N. Line Searches         13        Preproces. Time           1
    Time for Method          13        Effective Time           15
```

- maximum likelihood estimates:

```
                        ********************
                        Optimization Results
                        ********************

                        Parameter Estimates
                        -------------------

          Parameter     Estimate   Gradient   Active BC

           1 X_1     -0.81175994  4.74e-006
           2 X_2      0.81088337  6.85e-006
```

```
3 X_3      9.88180051 -8.76e-008

Value of Objective Function =     -116.137

                Hessian Matrix
                **************

      Symmetric Matrix: Dense Storage

     S |           1          2          3
     -------------------------------------
     1 |  -1689.9310 -2239.0267 -4.4075941
     2 |  -2239.0267 -3166.7312 -7.2265755
     3 |  -4.4075941 -7.2265755 -0.0271515
```

- approximate standard errors and correlations:

```
sopt = [ "grad" "hess" ];
< gopt,hopt > = fider(fquant,mle,sopt);
print "gopt=", gopt;
print "hopt=", hopt;

v = inv(-hopt);
ase = sqrt(dia2vec(v)); tab = mle‘ -> ase;
tab = rname(tab,pnam); tab = cname(tab,[" Estimate STDERR "]);

d = diag(1. / ase);
corr = d * v * d;

print "*** Quantit Model ***";
print "Estimates:",mle;
print "Standard Errors:",tab;
print "Correlations:",corr;
```

```
Standard Errors:
        |   Estimate     STDERR
------------------------------
  Alpha |   -0.81176    0.12832
   Beta |    0.81088    0.11356
     Nu |     9.8818     12.830

Correlations:
 S |         1         2         3
---------------------------------
 1 |   1.00000
 2 |  -0.96836   1.00000
 3 |   0.65695  -0.78273   1.00000
```

The observed data and the fitted models are shown in the following figure.

Logit, Aranda-Ordaz Asymmetric, and Quantit Models

1.6 Neural Nets with One Hidden Layer

This example can be found in `tnlp90.inp`. The following module computes the function vector of of residuals for the least-squares problem:

```
function fneur902(x) global (data,indx,indy,nh) {
  nr = nrow(data);
  xdat = data[,indx]; nx = size(indx);
  ydat = data[,indy]; ny = size(indy);
  npar = size(x); /* print "npar=",npar; */

  na2 = nx * nh;
  nb1 = na2 + 1; nb2 = na2 + nh * ny;
  nc1 = nb2 + 1; nc2 = nb2 + nh;
  nd1 = nc2 + 1; nd2 = nc2 + ny;
  /* mequ= nr * ny = 49 * 3 */
  /* nx * nh + nh * ny + nh + ny =
     = 40 + 30 + 10 + 3 = 83 parms */
  a = x[ 1:na2 ];   a = shape(a,nx,nh);
  b = x[ nb1:nb2 ]; b = shape(b,nh,ny);
  c = x[ nc1:nc2 ]; c1 = cons(nr,1,1.) @ c;
  d = x[ nd1:nd2 ]; d1 = cons(nr,1,1.) @ d;

  sum = c1 + xdat * a;
  h = 1. / (1. + exp(-sum));

  sum = d1 + h * b;
  yhat = 1. / (1. + exp(-sum));

  res = ydat - yhat;
  fun = shape(res,.,1);
  /* print "Fun=",fun; */
  return(fun);
}
```

The following module computes the Jacobian matrix of the least-squares problem:

```
function jneur902(x) global (data,indx,indy,nh) {
  nr = nrow(data);
  xdat = data[,indx]; nx = size(indx);
  ydat = data[,indy]; ny = size(indy);
  npar = size(x);
  /* print "JNEUR2: nr=",nr," npar=",npar," nx=",nx," ny=",ny; */

  na2 = nx * nh;
  nb1 = na2 + 1; nb2 = na2 + nh * ny;
  nc1 = nb2 + 1; nc2 = nb2 + nh;
  nd1 = nc2 + 1; nd2 = nc2 + ny;
  /* nx * nh + nh * ny + nh + ny =
     = 40 + 30 + 10 + 3 = 83 parms */
```

```
a = x[ 1:na2 ];   a = shape(a,nx,nh);
b = x[ nb1:nb2 ]; b = shape(b,nh,ny);
c = x[ nc1:nc2 ]; c1 = cons(nr,1,1.) @ c;
d = x[ nd1:nd2 ]; d1 = cons(nr,1,1.) @ d;

fun = cons(nr*ny,1); /* mequ= nr * ny = 49 * 3 */
jacm = cons(nr*ny,npar);
h = u = cons(nr,nh);
yhat = v = cons(nr,ny);

sum = c1 + xdat * a;
u = exp(-sum);
h = 1. / (1. + u);
sum = d1 + h * b;
v = exp(-sum);
yhat = 1. / (1. + v);
res = ydat - yhat;
fun = shape(res,.,1);

nry = nr * ny;
v = shape(v,nry,1);
yhat = shape(yhat,nry,1);
yind = [ 1:ny:nry ];
/* print "H=",h;  print "Yhat=", yhat; */
vyy = -v .* (yhat .* yhat);
uhh = u .* (h .* h);

/* [1] Deriv w.r.t. xd[ny]: cols 81-83 */
if (ny == 1) {
  jacm[ ,nd1] = vyy;
} else {
  j = nd1;
  for (iy = 1; iy <= ny; iy++, j++) {
    ind = [ iy:ny:nry ];
    jacm[ind,j] = vyy[ind];
} }
/* print "Derivs w.r.t. d[ny] are computed"; */

/* [2] Deriv w.r.t. xb[nh,ny]: cols 41-70 */
if (ny == 1) {
  j = nb1;
  for (ih = 1; ih <= nh; ih++, j++)
  jacm[,j] = vyy .* h[,ih];
} else {
  for (iy = 1; iy <= ny; iy++) {
    ind = [ iy:ny:nry ];
    j = nb1 + iy - 1;
    for (ih = 1; ih <= nh; ih++, j+=ny)
    jacm[ind,j] = vyy[ind] .* h[,ih];
} }
```

```
    /* print "Derivs w.r.t. b[nh,ny] are computed"; */

    /* [3] Deriv w.r.t. D(yhat[iy]) / D(xc[ih]): cols 71-80 */
    j = nc1;
    for (ih = 1; ih <= nh; ih++, j++) {
      tt = uhh[,ih] @ b[ih,]';
      jacm[,j] = vyy .* tt;
    }
    /* print "Derivs w.r.t. c[nh] are computed"; */

    /* [4] Deriv w.r.t. xa[nx,nh]: col 1-40 */
    for (ih = 1; ih <= nh; ih++) {
      tt = uhh[,ih] @ b[ih,]';
      vv = vyy .* tt;
      for (iy = 1; iy <= ny; iy++) {
        ind = [ iy:ny:nry ]; ww = vv[ind];
        j = ih;
        for (ix = 1; ix <= nx; ix++, j+=nh)
        jacm[ind,j] = ww .* xdat[,ix];
    } }
    /* print "Derivs w.r.t. a[nx,nh] are computed"; */

    /* print " * Leaving JNEUR902"; */
    return(jacm);
  }
```

The complete data can be found in `tnlp90.inp`:

```
print "TNLP90: Neural Network Problem, n=83";
cont= [ 1.00      0.00      0.00      0.00     95.29      -.75      1.30 ,
        0.75      0.25      0.00      0.00     68.28      -.85     -2.54 ,
        ..............................................................
        0.34      0.00      0.33      0.33     33.98     33.25    -29.88 ,
        0.34      0.33      0.00      0.33     37.46     27.13    -15.02 ];


  cont[,5] =  cont[,5] / 100.;
  cont[,6] = (cont[,6] +  1.) / 60.;
  cont[,7] = (cont[,7] + 42.) / 55.;
  data = con2 = cont[1:49,];
  cnam = [ "x1":"x4" "y1":"y3" ];
  data = con2 = cname(con2,cnam);
  print "CON2=", con2;


  nr = nrow(con2);
  indx = [ 1:4 ]; indy = [ 5:7 ];
  nx = size(indx); ny = size(indy);
  nh = 10;           /* np = 83; */
```

```
  np = nx * nh + nh * ny + nh + ny;
  print "NX,NY=",nx,ny," NP=",np;


  mequ = nr * ny;
  lbc = cons(np,1,-30.);
  ubc = cons(np,1, 30.);
  bc = lbc -> ubc; /* print "BC=",bc; */


  x0 = cons(1,np,1.);
  fun2 = fneur902(x0);
  /* print "Fun2=",fun2; */
  jac21 = jneur902(x0);
  /* print "JAC21=",jac21; */
  jac22 = fider(fneur902,x0);
  /* print "JAC22=", jac22; */


  jres = jac21 - jac22;
  jres = ssq(jres);
  print "Squared sum of Difference=",jres;


NX,NY= 4 3 NP= 83
Squared sum of Difference= 7.754e-014
```

The optimization function creates a singular Hessian and a large number of local minima. Any fit with a nonzero objective function represents a local and not the global minimum:

```
  print "Levenberg-Marquardt LS Algorithm";
  mopt = [ "tech"      "levmar" ,
           "print"            3 ,
           "maxit"         3000 ,
           "maxfu"         8000 ];
  < xr, rp > = nlp(fneur902,x0,mopt,bc,.,.,jneur902);


                        ******************
                        Optimization Start
                        ******************

                       Parameter Estimates
                       -------------------

 Parameter      Estimate   Gradient   Lower BC   Upper BC

   1 Par_1    1.00000000  1.18e-004 -30.000000  30.000000
```

```
 2 Par_2    1.00000000  1.18e-004 -30.000000  30.000000
 3 Par_3    1.00000000  1.18e-004 -30.000000  30.000000
.....................................................
81 Par_81   1.00000000  0.0019650 -30.000000  30.000000
82 Par_82   1.00000000  0.0018040 -30.000000  30.000000
83 Par_83   1.00000000  0.0012727 -30.000000  30.000000

          Value of Objective Function =      33.2668


               Levenberg-Marquardt Optimization
                 Scaling Update of More (1978)
                 User Specified Jacobian (dense)


  Iteration Start:
  N. Variables             83        N. Equations            147
  N. Bound. Constr.       166        N. Mask  Constr.          0
  Criterion       33.26677150        Max Grad Entry  0.001964988
  N. Active Constraints     0        TR Radius       1.000000000


 Iter rest nfun act   optcrit  difcrit maxgrad  lambda      rho
    1*   0    3  82  23.40551 9.861258  2e-038  4e-014  0.32744

  Successful Termination After     1 Iterations
  ABSGCONV convergence criterion satisfied.
  Criterion       23.40551370        Max Grad Entry  2.1190e-038
  N. Active Constraints    82        Ridge (lambda)  4.4409e-014
  TR Radius       2.000000000        Act.dF/Pred.dF  0.327441722
  N. Function Calls         4        N. Gradient Calls         2
  N. Hessian Calls          3        Preproces. Time           3
  Time for Method           1        Effective Time            5
```