Performances of DESCON, L-BFGS, L-CG-DESCENT and of CONOPT, KNITRO, MINOS, SNOPT, IPOPT for solving the problem PALMER1C

Neculai Andrei
Center for Advanced Modeling and Optimization
Academy of Romanian Scientists, Bucharest, Romania

Technical Report No.3/2019
June 9, 2019
Bucharest, Romania

Abstract. In this technical report I present the performances of DESCON (Andrei, 2013), L-BFGS (Liu and Nocedal, 1989) and L-CG-DESCENT (Hager and Zhang, 2013) for solving the problem PALMER1C. At the same time, the performances and the solution of this problem given by CONOPT, KNITRO, MINOS, SNOPT and IPOPT, included in GAMS technology, are presented. Appendices A, B and C presents the Fortran, C and GAMS codes for this problem.

1. Introduction

Consider the unconstrained optimization problem PALMER1C:

$$\min\{f(x) : x \in \mathbb{R}^8\},$$

where

$$f(x) = \sum_{i=1}^{35}\left(y_i - \left(x_1 + x_2(z_i^2) + x_3(z_i^4) + x_4(z_i^6) + x_5(z_i^8) + x_6(z_i^{10}) + x_7(z_i^{12}) + x_8(z_i^{14})\right)\right)^2,$$

and the parameters y_i and z_i, $i = 1,\ldots,35$, are as in Table 1:

Table 1. Parameters of the problem PALMER1C

Parameters $z_i, (i = 1,\ldots,35)$	Parameters $y_i, (i = 1,\ldots,35)$
z(1)= -1.788963	y(1)= 78.596218
z(2)= -1.745329	y(2)= 65.77963
z(3)= -1.658063	y(3)= 43.96947
z(4)= -1.570796	y(4)= 27.038816
z(5)= -1.483530	y(5)= 14.6126
z(6)= -1.396263	y(6)= 6.2614
z(7)= -1.308997	y(7)= 1.538330

z(8)= -1.218612	y(8)= 0.000000
z(9)= -1.134464	y(9)= 1.188045
z(10)= -1.047198	y(10)= 4.6841
z(11)= -0.872665	y(11)= 16.9321
z(12)= -0.698132	y(12)= 33.6988
z(13)= -0.523599	y(13)= 52.3664
z(14)= -0.349066	y(14)= 70.1630
z(15)= -0.174533	y(15)= 83.4221
z(16)= 0.0000000	y(16)= 88.3995
z(17)= 1.788963	y(17)= 78.596218
z(18)= 1.745329	y(18)= 65.77963
z(19)= 1.658063	y(19)= 43.96947
z(20)= 1.570796	y(20)= 27.038816
z(21)= 1.483530	y(21)= 14.6126
z(22)= 1.396263	y(22)= 6.2614
z(23)= 1.308997	y(23)= 1.538330
z(24)= 1.218612	y(24)= 0.000000
z(25)= 1.134464	y(25)= 1.188045
z(26)= 1.047198	y(26)= 4.6841
z(27)= 0.872665	y(27)= 16.9321
z(28)= 0.698132	y(28)= 33.6988
z(29)= 0.523599	y(29)= 52.3664
z(30)= 0.349066	y(30)= 70.1630
z(31)= 0.174533	y(31)= 83.4221
z(32)= -1.8762289	y(32)= 108.18086
z(33)= -1.8325957	y(33)= 92.733676
z(34)= 1.8762289	y(34)= 108.18086
z(35)= 1.8325957	y(35)= 92.733676

This is a positive definite quadratic optimization problem of dimension $n = 8$ with a large condition number of about 10^{12}. The eigenvalues of this problem are in the range from 0.0002 up to 2×10^{8}. This is an ill-conditioned problem, for which the convergence is very slow, even if its dimension is very small.

The description of the problem in AMPL language is given at the address:
https://vanderbei.princeton.edu/ampl/nlmodels/cute/palmer1c.mod
The initial point is: $x(i) = 1,\ (i = 1,\ldots,8)$.

The performances of DESCON, L-BFGS (*m=8*) and L-CG-DESCENT are illustrated in the following tables. Observe that there are some differences among the solutions obtained by these algorithms. However, the optimal value of the objective function f is the same for all algorithms considered in this numerical study.

For solving this problem, the performances of DESCONa [1] are given in Table 2.

Table 2. Performances of DESCONa

#iter	*#fg*	*cpu(s)*	*vfo*
937	5004	0.02	0.097594

The solution obtained by DESCONa is as follows:

```
x(1)= 0.8829720532991E+02
x(2)=-0.1643843812955E+03
x(3)= 0.1460603050440E+03
x(4)=-0.1034224338789E+03
x(5)= 0.5592546440655E+02
x(6)=-0.1783851227088E+02
x(7)= 0.3042154111949E+01
x(8)=-0.2140580573342E+00
```

The performances of L-BFGS [2, 3] with *m=8* are given in Table 3:

Table 3. Performances of L-BFGS (*m=8*)

#iter	*#fg*	*cpu(s)*	*vfo*
5350	6511	0.06	0.097594

The solution of the problem given by L-BFGS (*m=8*) is:

```
x(1)= 0.8829735123552E+02
x(2)=-0.1643868112967E+03
x(3)= 0.1460716820046E+03
x(4)=-0.1034432170695E+03
x(5)= 0.5594302750442E+02
x(6)=-0.1784595250574E+02
x(7)= 0.3043693771710E+01
x(8)=-0.2141820897549E+00
```

Performances of L-CG-DESCENT [4] with standard Wolfe line search are presented in Table 4.

Table 4. Performances of L-CG-DESCENT

	#iter	*#f*	*#g*	*cpu(s)*	*vfo*
TRUE	51302	83296	143343	134.94	0.097594
memory=0	51302	83296	143343	138.10	0.097594
memory=5	14242	17103	32229	36.41	0.097594
memory=9	12	23	24	0.04	0.097594

The solution obtained by L-CG-DESCENT (*memory*=9) is:

```
x(1)= 0.8829717175E+02
x(2)=-0.1643839719E+03
x(3)= 0.1460596677E+03
x(4)=-0.1034224930E+03
x(5)= 0.5592610073E+02
x(6)=-0.1783894601E+02
x(7)= 0.3042268999E+01
x(8)=-0.2140689274E-01
```

L-CG-DESCENT is a development of CG-DESCENT by Hager and Zhang [5, 6].
In Table 4 TRUE corresponds to setting in L-CG-DESCENT the parameter LBFGS = TRUE.
In L-CG-DESCENT the parameter "*memory*" represents the number of search directions in the memory of the limited memory conjugate gradient algorithm.
#iter, *#fg*, *#f*, *#g*, *cpu(s)* and *vfo* represents the number of iterations, the number of function and its gradient evaluations (in DESCONa and L-BFGS), the number of function evaluations, the number of gradient evaluations (in L-CG-DESCENT), the CPU time in seconds and function value in optimal point, respectively.
When memory=0, then L-CG-DESCENT reduces to version CG-DESCENT 5.3 of CG-DESCENT.

Observe that for solving this problem, L-CG-DESCENT with *memory=9* is the best variant of the algorithm.
Setting LBFGS parameter to TRUE in L-CG-DESCENT, yields L-BFGS. However, as we can see the L-BFGS Fortran code with *m=8* of Liu and Nocedal [2] is way faster then the limited memory BFGS implemented in L-CG-DESCENT.
Subject to the CPU time metric, both DESCON and L-CG-DESCENT with *memory=9* have similar performances.

It is worth seeing the performances of solvers CONOPT, KNITRO, MINOS, SNOPT and IPOPT for solving this ill-conditioned unconstrained optimization problem. These optimizers, included in GAMS technology, are described in [7]. Details on GAMS representation of optimization models are presented in [8].
Table 5 presents the performances and the solution given by CONOPT:

Table 5. Performances of CONOPT
#iter = 10, cpu = 0.022 seconds, vfo = 0.0975979913

```
X(1) =  88.2971762002
X(2) = -164.3840192523
X(3) =  146.0598092567
X(4) = -103.4226614763
X(5) =  55.9261888449
X(6) = -17.8389658845
X(7) =  3.0422702655
X(8) = -0.2140688425
```

Table 6 presents the performances and the solution given by KNITRO:

Table 6. Performances of KNITRO
#iter = 1, cpu = 0.133 seconds, vfo = 0.0975979913

```
X(1) =  88.2971761790
X(2) = -164.3840187082
X(3) =  146.0598069387
X(4) = -103.4226577230
X(5) =  55.9261859138
X(6) = -17.8389646986
X(7) =  3.0422700261
X(8) = -0.2140688235
```

Table 7 presents the performances and the solution given by MINOS:

Table 7. Performances of MINOS
#iter = 25, cpu = 0.015 seconds, vfo = 0.0975979913

```
X(1) =  88.2971762001
X(2) = -164.3840192513
X(3) =  146.0598092565
X(4) = -103.4226614787
X(5) =  55.9261888477
X(6) = -17.8389658858
X(7) =  3.0422702658
X(8) = -0.2140688426
```

Table 8 presents the performances and the solution given by SNOPT:

Table 8. Performances of SNOPT
#iter = 25, cpu = 0.209 seconds, vfo = 0.0975979913

```
X(1) =  88.2971762001
X(2) = -164.3840192514
X(3) =  146.0598092569
X(4) = -103.4226614794
X(5) =  55.9261888483
X(6) = -17.8389658860
X(7) =  3.0422702658
X(8) = -0.2140688426
```

Table 9 presents the performances and the solution given by IPOPT:

Table 9. Performances of IPOPT
#iter = 1, cpu = 0.077 seconds, vfo = 0.0975979913

```
X(1) =  88.2971762829
X(2) = -164.3840214187
X(3) =  146.0598186039
X(4) = -103.4226767374
X(5) =  55.9262008369
X(6) = -17.8389707600
X(7) =  3.0422712537
X(8) = -0.2140689215
```

References

[1] Andrei, N., (2013). Another conjugate gradient algorithm with guaranteed descent and conjugacy conditions for large-scale unconstrained optimization. *Journal of Optimization Theory and Applications, 159*, 159-182.

[2] Liu, D.C., & Nocedal, J., (1989). On the limited-memory BFGS method for large optimization. *Mathematical Programming, 45*, 503-528.

[3] Nocedal, J., (1980). Updating quasi-Newton matrices with limited storage. *Mathematics of Computation 35*, 773-782.

[4] Hager, W.W., & Zhang, H., (2013). The limited memory conjugate gradient method. *SIAM Journal on Optimization, 23*, 2150-2168.

[5] Hager, W.W., & Zhang, H., (2005). A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, *16*, (2005) 170-192.
[6] Hager, W.W., & Zhang, H., (2006a). Algorithm 851: CG-DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software, 32*(1), 113-137.
[7] Andrei, N., (2017) *Continuous Nonlinear Optimization for Engineering Applications in GAMS Technology.* Springer Optimization and Its Applications 121, New York, NY, USA: Springer Science + Business Media.
[8] Andrei, N., (2013). *Nonlinear Optimization Applications using the GAMS Technology.* Springer Optimization and its Applications Series. Vol. 81, New York, NY, USA: Springer Science + Business Media.

APPENDIX A

The Fortran expression of the problem PALMER1C.

```
cF81                                PALMERC1
*                       Initial point x0=[1,1,...,1]
*
81    continue
c
      z(1)= -1.788963
      z(2)= -1.745329
      z(3)= -1.658063
      z(4)= -1.570796
      z(5)= -1.483530
      z(6)= -1.396263
      z(7)= -1.308997
      z(8)= -1.218612
      z(9)= -1.134464
      z(10)= -1.047198
      z(11)= -0.872665
      z(12)= -0.698132
      z(13)= -0.523599
      z(14)= -0.349066
      z(15)= -0.174533
      z(16)= 0.0000000
      z(17)= 1.788963
      z(18)= 1.745329
      z(19)= 1.658063
      z(20)= 1.570796
      z(21)= 1.483530
      z(22)= 1.396263
      z(23)= 1.308997
      z(24)= 1.218612
      z(25)= 1.134464
      z(26)= 1.047198
      z(27)= 0.872665
      z(28)= 0.698132
      z(29)= 0.523599
      z(30)= 0.349066
      z(31)= 0.174533
      z(32)= -1.8762289
```

```
      z(33)= -1.8325957
      z(34)= 1.8762289
      z(35)= 1.8325957
c
      y(1)= 78.596218
      y(2)= 65.77963
      y(3)= 43.96947
      y(4)= 27.038816
      y(5)= 14.6126
      y(6)= 6.2614
      y(7)= 1.538330
      y(8)= 0.000000
      y(9)= 1.188045
      y(10)= 4.6841
      y(11)= 16.9321
      y(12)= 33.6988
      y(13)= 52.3664
      y(14)= 70.1630
      y(15)= 83.4221
      y(16)= 88.3995
      y(17)= 78.596218
      y(18)= 65.77963
      y(19)= 43.96947
      y(20)= 27.038816
      y(21)= 14.6126
      y(22)= 6.2614
      y(23)= 1.538330
      y(24)= 0.000000
      y(25)= 1.188045
      y(26)= 4.6841
      y(27)= 16.9321
      y(28)= 33.6988
      y(29)= 52.3664
      y(30)= 70.1630
      y(31)= 83.4221
      y(32)= 108.18086
      y(33)= 92.733676
      y(34)= 108.18086
      y(35)= 92.733676
c
c Function

      do i=1,35
        temp(i) = y(i) - (x(1)           + x(2)*z(i)**2
     +                     + x(3)*z(i)**4  + x(4)*z(i)**6
     +                   + x(5)*z(i)**8  + x(6)*z(i)**10
     +                   + x(7)*z(i)**12 + x(8)*z(i)**14)
      end do

      f = 0.d0

      do i=1,35
        f = f + (temp(i))**2
      end do

c Gradient
```

```
      g(1) = 0.d0
      do i=1,35
        g(1) = g(1) - 2.d0*temp(i)
      end do

      g(2) = 0.d0
      do i=1,35
        g(2) = g(2) - 2.d0*temp(i) * (z(i)**2)
      end do

      g(3) = 0.d0
      do i=1,35
        g(3) = g(3) - 2.d0*temp(i) * (z(i)**4)
      end do

      g(4) = 0.d0
      do i=1,35
        g(4) = g(4) - 2.d0*temp(i) * (z(i)**6)
      end do

      g(5) = 0.d0
      do i=1,35
        g(5) = g(5) - 2.d0*temp(i) * (z(i)**8)
      end do

      g(6) = 0.d0
      do i=1,35
        g(6) = g(6) - 2.d0*temp(i) * (z(i)**10)
      end do

      g(7) = 0.d0
      do i=1,35
        g(7) = g(7) - 2.d0*temp(i) * (z(i)**12)
      end do

      g(8) = 0.d0
      do i=1,35
        g(8) = g(8) - 2.d0*temp(i) * (z(i)**14)
      end do

      return
      end
```

APPENDIX B

The C expression of the problem PALMER1C

```
//                              PALMER1C  (function)
double myvalue
(
 double   *x,
 INT       n
 )
{
      double f, z[35], y[35], temp[35] ;
```

```
        INT i ;

        z[0]= -1.788963f;
        z[1]= -1.745329f;
        z[2]= -1.658063f;
        z[3]= -1.570796f;
        z[4]= -1.483530f;
        z[5]= -1.396263f;
        z[6]= -1.308997f;
        z[7]= -1.218612f;
        z[8]= -1.134464f;
        z[9]= -1.047198f;
        z[10]= -0.872665f;
        z[11]= -0.698132f;
        z[12]= -0.523599f;
        z[13]= -0.349066f;
        z[14]= -0.174533f;
        z[15]= 0.0000000f;
        z[16]= 1.788963f;
        z[17]= 1.745329f;
        z[18]= 1.658063f;
        z[19]= 1.570796f;
        z[20]= 1.483530f;
        z[21]= 1.396263f;
        z[22]= 1.308997f;
        z[23]= 1.218612f;
        z[24]= 1.134464f;
        z[25]= 1.047198f;
        z[26]= 0.872665f;
        z[27]= 0.698132f;
        z[28]= 0.523599f;
        z[29]= 0.349066f;
        z[30]= 0.174533f;
        z[31]= -1.8762289f;
        z[32]= -1.8325957f;
        z[33]= 1.8762289f;
        z[34]= 1.8325957f;
//
        y[0]= 78.596218f;
        y[1]= 65.77963f;
        y[2]= 43.96947f;
        y[3]= 27.038816f;
        y[4]= 14.6126f;
        y[5]= 6.2614f;
        y[6]= 1.538330f;
        y[7]= 0.00000f;
        y[8]= 1.188045f;
        y[9]= 4.6841f;
        y[10]= 16.9321f;
        y[11]= 33.6988f;
        y[12]= 52.3664f;
        y[13]= 70.1630f;
        y[14]= 83.4221f;
        y[15]= 88.3995f;
        y[16]= 78.596218f;
        y[17]= 65.77963f;
        y[18]= 43.96947f;
```

```
      y[19]= 27.038816f;
      y[20]= 14.6126f;
      y[21]= 6.2614f;
      y[22]= 1.538330f;
      y[23]= 0.000000f;
      y[24]= 1.188045f;
      y[25]= 4.6841f;
      y[26]= 16.9321f;
      y[27]= 33.6988f;
      y[28]= 52.3664f;
      y[29]= 70.1630f;
      y[30]= 83.4221f;
      y[31]= 108.18086f;
      y[32]= 92.733676f;
      y[33]= 108.18086f;
      y[34]= 92.733676f;

/* Function for   PALMER1C */

      for (i = 0; i <= 34; i++)
      {
        temp[i] = y[i] - (x[0]                  + x[1]*pow(z[i],2.0f)
                       + x[2]*pow(z[i],4.0f)  + x[3]*pow(z[i],6.0f)
                       + x[4]*pow(z[i],8.0f)  + x[5]*pow(z[i],10.0f)
                       + x[6]*pow(z[i],12.0f) + x[7]*pow(z[i],14.0f));
      }

      f = 0.0f ;

      for(i = 0; i <= 34; i++)
      {
       f += pow((temp[i]),2.0f);
      }
      return (f) ;
}

//                                PALMER1C (gradient)

void mygrad
(
 double  OUT  *g,
 double IN    *x,
 INT        n
 )
{
      double z[35], y[35], temp[35];
      INT i;

      z[0]= -1.788963f;
      z[1]= -1.745329f;
      z[2]= -1.658063f;
      z[3]= -1.570796f;
      z[4]= -1.483530f;
      z[5]= -1.396263f;
      z[6]= -1.308997f;
      z[7]= -1.218612f;
      z[8]= -1.134464f;
```

```
        z[9]= -1.047198f;
        z[10]= -0.872665f;
        z[11]= -0.698132f;
        z[12]= -0.523599f;
        z[13]= -0.349066f;
        z[14]= -0.174533f;
        z[15]= 0.0000000f;
        z[16]= 1.788963f;
        z[17]= 1.745329f;
        z[18]= 1.658063f;
        z[19]= 1.570796f;
        z[20]= 1.483530f;
        z[21]= 1.396263f;
        z[22]= 1.308997f;
        z[23]= 1.218612f;
        z[24]= 1.134464f;
        z[25]= 1.047198f;
        z[26]= 0.872665f;
        z[27]= 0.698132f;
        z[28]= 0.523599f;
        z[29]= 0.349066f;
        z[30]= 0.174533f;
        z[31]= -1.8762289f;
        z[32]= -1.8325957f;
        z[33]= 1.8762289f;
        z[34]= 1.8325957f;
//
        y[0]= 78.596218f;
        y[1]= 65.77963f;
        y[2]= 43.96947f;
        y[3]= 27.038816f;
        y[4]= 14.6126f;
        y[5]= 6.2614f;
        y[6]= 1.538330f;
        y[7]= 0.00000f;
        y[8]= 1.188045f;
        y[9]= 4.6841f;
        y[10]= 16.9321f;
        y[11]= 33.6988f;
        y[12]= 52.3664f;
        y[13]= 70.1630f;
        y[14]= 83.4221f;
        y[15]= 88.3995f;
        y[16]= 78.596218f;
        y[17]= 65.77963f;
        y[18]= 43.96947f;
        y[19]= 27.038816f;
        y[20]= 14.6126f;
        y[21]= 6.2614f;
        y[22]= 1.538330f;
        y[23]= 0.000000f;
        y[24]= 1.188045f;
        y[25]= 4.6841f;
        y[26]= 16.9321f;
        y[27]= 33.6988f;
        y[28]= 52.3664f;
        y[29]= 70.1630f;
```

```
  y[30]= 83.4221f;
  y[31]= 108.18086f;
  y[32]= 92.733676f;
  y[33]= 108.18086f;
  y[34]= 92.733676f;

  for (i = 0; i <= 34; i++)
  {
    temp[i] = y[i]
            - (x[0]
            +  x[1]*pow(z[i],2.0f)
            +  x[2]*pow(z[i],4.0f)  + x[3]*pow(z[i],6.0f)
            +  x[4]*pow(z[i],8.0f)  + x[5]*pow(z[i],10.0f)
            +  x[6]*pow(z[i],12.0f) + x[7]*pow(z[i],14.0f));
  }

g[0] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[0] += -2.0f * temp[i];
  }

  g[1] = 0.0f ;
  for(i = 0; i <= 34; i++)
{
    g[1] += -2.0f * temp[i] * pow(z[i],2.0f);
  }

  g[2] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[2] += -2.0f * temp[i] * pow(z[i],4.0f);
  }

  g[3] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[3] += -2.0f * temp[i] * pow(z[i],6.0f);
  }

  g[4] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[4] += -2.0f * temp[i] * pow(z[i],8.0f);
  }

  g[5] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[5] += -2.0f * temp[i] * pow(z[i],10.0f);
  }

  g[6] = 0.0f;
  for(i = 0; i <= 34; i++)
{
    g[6] += -2.0f * temp[i] * pow(z[i],12.0f);
  }
```

```
      g[7] = 0.0f;
      for(i = 0; i <= 34; i++)
    {
        g[7] += -2.0f * temp[i] * pow(z[i],14.0f);
      }

      return;
}
```

APPENDIX C

Expression of PALMER1C in GAMS

```
*   PALMER1C
*   July 10, 2019
*
*   Solution of PALMER1C using some optimizers from GAMS.
*
 Equations objcon;
 Variables x1, x2, x3, x4, x5, x6, x7, x8, obj;

Scalars
* z(i)  (35 elements)
z1 /-1.788963/,
z2 /-1.745329/,
z3 /-1.658063/,
z4 /-1.570796/,
z5 /-1.483530/,
z6 /-1.396263/,
z7 /-1.308997/,
z8 /-1.218612/,
z9 /-1.134464/,
z10 /-1.047198/,
z11 /-0.872665/,
z12 /-0.698132/,
z13 /-0.523599/,
z14 /-0.349066/,
z15 /-0.174533/,
z16 /0.0000000/,
z17 /1.788963/,
z18 /1.745329/,
z19 /1.658063/,
z20 /1.570796/,
z21 /1.483530/,
z22 /1.396263/,
z23 /1.308997/,
z24 /1.218612/,
z25 /1.134464/,
z26 /1.047198/,
```

```
z27 /0.872665/,
z28 /0.698132/,
z29 /0.523599/,
z30 /0.349066/,
z31 /0.174533/,
z32 /-1.8762289/,
z33 /-1.8325957/,
z34 /1.8762289/,
z35 /1.8325957/,


* y(i)  (35 elements)
y1   /78.596218/,
y2   /65.77963/,
y3   /43.96947/,
y4   /27.038816/,
y5   /14.6126/,
y6   /6.2614/,
y7   /1.538330/,
y8   /0.000000/,
y9   /1.188045/,
y10  /4.6841/,
y11  /16.9321/,
y12  /33.6988/,
y13  /52.3664/,
y14  /70.1630/,
y15  /83.4221/,
y16  /88.3995/,
y17  /78.596218/,
y18  /65.77963/,
y19  /43.96947/,
y20  /27.038816/,
y21  /14.6126/,
y22  /6.2614/,
y23  /1.538330/,
y24  /0.000000/,
y25  /1.188045/,
y26  /4.6841/,
y27  /16.9321/,
y28  /33.6988/,
y29  /52.3664/,
y30  /70.1630/,
y31  /83.4221/,
y32  /108.18086/,
y33  /92.733676/,
y34  /108.18086/,
y35  /92.733676/;

* Initial point :
x1.L = 1;
x2.L = 1;
```

```
x3.L = 1;
x4.L = 1;
x5.L = 1;
x6.L = 1;
x7.L = 1;
x8.L = 1;

* Objective function (to be minimized):
objcon.. obj =e= power( y1-( x1    + x2*power(z1,2)    + x3*power(z1,4)    + x4*power(z1,6)+
        x5*power(z1,8)+   x6*power(z1,10)   + x7*power(z1,12)   + x8*power(z1,14)) , 2)
         + power( y2-( x1      + x2*power(z2,2)      + x3*power(z2,4)      + x4*power(z2,6)+
        x5*power(z2,8)+   x6*power(z2,10)   + x7*power(z2,12)   + x8*power(z2,14)) , 2)
         + power( y3-( x1      + x2*power(z3,2)      + x3*power(z3,4)      + x4*power(z3,6)+
        x5*power(z3,8)+   x6*power(z3,10)   + x7*power(z3,12)   + x8*power(z3,14)) , 2)
         + power( y4-( x1      + x2*power(z4,2)      + x3*power(z4,4)      + x4*power(z4,6)+
        x5*power(z4,8)+   x6*power(z4,10)   + x7*power(z4,12)   + x8*power(z4,14)) , 2)
         + power( y5-( x1      + x2*power(z5,2)      + x3*power(z5,4)      + x4*power(z5,6)+
        x5*power(z5,8)+   x6*power(z5,10)   + x7*power(z5,12)   + x8*power(z5,14)) , 2)
         + power( y6-( x1      + x2*power(z6,2)      + x3*power(z6,4)      + x4*power(z6,6)+
        x5*power(z6,8)+   x6*power(z6,10)   + x7*power(z6,12)   + x8*power(z6,14)) , 2)
         + power( y7-( x1      + x2*power(z7,2)      + x3*power(z7,4)      + x4*power(z7,6)+
        x5*power(z7,8)+   x6*power(z7,10)   + x7*power(z7,12)   + x8*power(z7,14)) , 2)
         + power( y8-( x1      + x2*power(z8,2)      + x3*power(z8,4)      + x4*power(z8,6)+
        x5*power(z8,8)+   x6*power(z8,10)   + x7*power(z8,12)   + x8*power(z8,14)) , 2)
         + power( y9-( x1      + x2*power(z9,2)      + x3*power(z9,4)      + x4*power(z9,6)+
        x5*power(z9,8)+   x6*power(z9,10)   + x7*power(z9,12)   + x8*power(z9,14)) , 2)
         + power( y10-( x1    + x2*power(z10,2)    + x3*power(z10,4)    + x4*power(z10,6)+
        x5*power(z10,8)+  x6*power(z10,10)  + x7*power(z10,12)  + x8*power(z10,14)) , 2)
         + power( y11-( x1    + x2*power(z11,2)    + x3*power(z11,4)    + x4*power(z11,6)+
        x5*power(z11,8)+  x6*power(z11,10)  + x7*power(z11,12)  + x8*power(z11,14)) , 2)
         + power( y12-( x1    + x2*power(z12,2)    + x3*power(z12,4)    + x4*power(z12,6)+
        x5*power(z12,8)+  x6*power(z12,10)  + x7*power(z12,12)  + x8*power(z12,14)) , 2)
         + power( y13-( x1    + x2*power(z13,2)    + x3*power(z13,4)    + x4*power(z13,6)+
        x5*power(z13,8)+  x6*power(z13,10)  + x7*power(z13,12)  + x8*power(z13,14)) , 2)
         + power( y14-( x1    + x2*power(z14,2)    + x3*power(z14,4)    + x4*power(z14,6)+
        x5*power(z14,8)+  x6*power(z14,10)  + x7*power(z14,12)  + x8*power(z14,14)) , 2)
         + power( y15-( x1    + x2*power(z15,2)    + x3*power(z15,4)    + x4*power(z15,6)+
        x5*power(z15,8)+  x6*power(z15,10)  + x7*power(z15,12)  + x8*power(z15,14)) , 2)
         + power( y16-( x1    + x2*power(z16,2)    + x3*power(z16,4)    + x4*power(z16,6)+
        x5*power(z16,8)+  x6*power(z16,10)  + x7*power(z16,12)  + x8*power(z16,14)) , 2)
         + power( y17-( x1    + x2*power(z17,2)    + x3*power(z17,4)    + x4*power(z17,6)+
        x5*power(z17,8)+  x6*power(z17,10)  + x7*power(z17,12)  + x8*power(z17,14)) , 2)
         + power( y18-( x1    + x2*power(z18,2)    + x3*power(z18,4)    + x4*power(z18,6)+
        x5*power(z18,8)+  x6*power(z18,10)  + x7*power(z18,12)  + x8*power(z18,14)) , 2)
         + power( y19-( x1    + x2*power(z19,2)    + x3*power(z19,4)    + x4*power(z19,6)+
        x5*power(z19,8)+  x6*power(z19,10)  + x7*power(z19,12)  + x8*power(z19,14)) , 2)
         + power( y20-( x1    + x2*power(z20,2)    + x3*power(z20,4)    + x4*power(z20,6)+
        x5*power(z20,8)+  x6*power(z20,10)  + x7*power(z20,12)  + x8*power(z20,14)) , 2)
         + power( y21-( x1    + x2*power(z21,2)    + x3*power(z21,4)    + x4*power(z21,6)+
        x5*power(z21,8)+  x6*power(z21,10)  + x7*power(z21,12)  + x8*power(z21,14)) , 2)
```

```
 + power( y22-( x1  + x2*power(z22,2)  + x3*power(z22,4)  + x4*power(z22,6)+
x5*power(z22,8)+ x6*power(z22,10) + x7*power(z22,12) + x8*power(z22,14)) , 2)
 + power( y23-( x1  + x2*power(z23,2)  + x3*power(z23,4)  + x4*power(z23,6)+
x5*power(z23,8)+ x6*power(z23,10) + x7*power(z23,12) + x8*power(z23,14)) , 2)
 + power( y24-( x1  + x2*power(z24,2)  + x3*power(z24,4)  + x4*power(z24,6)+
x5*power(z24,8)+ x6*power(z24,10) + x7*power(z24,12) + x8*power(z24,14)) , 2)
 + power( y25-( x1  + x2*power(z25,2)  + x3*power(z25,4)  + x4*power(z25,6)+
x5*power(z25,8)+ x6*power(z25,10) + x7*power(z25,12) + x8*power(z25,14)) , 2)
 + power( y26-( x1  + x2*power(z26,2)  + x3*power(z26,4)  + x4*power(z26,6)+
x5*power(z26,8)+ x6*power(z26,10) + x7*power(z26,12) + x8*power(z26,14)) , 2)
 + power( y27-( x1  + x2*power(z27,2)  + x3*power(z27,4)  + x4*power(z27,6)+
x5*power(z27,8)+ x6*power(z27,10) + x7*power(z27,12) + x8*power(z27,14)) , 2)
 + power( y28-( x1  + x2*power(z28,2)  + x3*power(z28,4)  + x4*power(z28,6)+
x5*power(z28,8)+ x6*power(z28,10) + x7*power(z28,12) + x8*power(z28,14)) , 2)
 + power( y29-( x1  + x2*power(z29,2)  + x3*power(z29,4)  + x4*power(z29,6)+
x5*power(z29,8)+ x6*power(z29,10) + x7*power(z29,12) + x8*power(z29,14)) , 2)
 + power( y30-( x1  + x2*power(z30,2)  + x3*power(z30,4)  + x4*power(z30,6)+
x5*power(z30,8)+ x6*power(z30,10) + x7*power(z30,12) + x8*power(z30,14)) , 2)
 + power( y31-( x1  + x2*power(z31,2)  + x3*power(z31,4)  + x4*power(z31,6)+
x5*power(z31,8)+ x6*power(z31,10) + x7*power(z31,12) + x8*power(z31,14)) , 2)
 + power( y32-( x1  + x2*power(z32,2)  + x3*power(z32,4)  + x4*power(z32,6)+
x5*power(z32,8)+ x6*power(z32,10) + x7*power(z32,12) + x8*power(z32,14)) , 2)
 + power( y33-( x1  + x2*power(z33,2)  + x3*power(z33,4)  + x4*power(z33,6)+
x5*power(z33,8)+ x6*power(z33,10) + x7*power(z33,12) + x8*power(z33,14)) , 2)
 + power( y34-( x1  + x2*power(z34,2)  + x3*power(z34,4)  + x4*power(z34,6)+
x5*power(z34,8)+ x6*power(z34,10) + x7*power(z34,12) + x8*power(z34,14)) , 2)
 + power( y35-( x1  + x2*power(z35,2)  + x3*power(z35,4)  + x4*power(z35,6)+
x5*power(z35,8)+ x6*power(z35,10) + x7*power(z35,12) + x8*power(z35,14)) , 2);

Model palmer1c /all/;
solve palmer1c minimizing obj using nlp;

file palmer1co /palmer1.dat/
put palmer1co;
put x1.l:25:10, x2.l:25:10, x3.l:25:10, x4.l:25:10, x5.l:25:10, x6.l:25:10, x7.l:25:10, x8.l:25:10 /;
put obj.l:25:10 /;
* End palmer1c
```

-----oooOooo-----