Computer code for derivative free gradient projection rotation algorithms

What follows is a discription of some specific computer code for orthogonal and oblique derivative free gradient projection (GP) rotation algorithms. Also included are examples of how to use these and background references. The code used is Matlab (1995). This was choosen because it is a very transparant matrix language that is easily translated to S, R, and other environments or may be used directly. The code itself in editable form may be downloaded from `http://www.stat.ucla/research`. See Jennrich (2003) for a more complete discussion of derivative free GP algorithms and there relation to GP algorithms using exact derivatives. This paper includes results on precision and speed.

A derivative free GP algorithm for orthogonal rotation

The general orthogonal rotation problem is to minimize

$$f(T) \quad \text{given} \quad T'T = I$$

where T need not be square. For orthogonal rotation in factor analysis T is square and

$$f(T) = Q(AT)$$

where Q is a factor analysis rotation criterion and A is an initial loading matrix.

The computer code has three parts. The first is the basic GP minimization algorithm for orthogonal rotation. The second computes the gradient of f at T using numerical derivatives and the third defines the function f. The first two are used without change for any criterion. The third is criterion specific and must be produced for each criterion of interest, for example one for quartimax and another for varimax.

- What follows is Matlab code for the basic GP algorithm.

```
function [Th,table]=GPorth(T)

al=1;
table=[];
```

```
for iter=0:100
   f=ff(T);
   G=Gf(T);
   M=T'*G;
   S=(M+M')/2;
   Gp=G-T*S;
   s=norm(Gp,'fro');

   table=[table;iter f log10(s) al];
   if s<10^(-5),break,end

   al=2*al;
   for i=0:10
      X=T-al*Gp;
      [U,D,V]=svd(X,0);
      Tt=U*V';
      ft=ff(Tt);
      if ft<f-.5*s^2*al,break,end
      al=al/2;
      end
   T=Tt;
   end
Th=T;
```

We will comment on some of the more interesting lines of code. The first line

```
function [Th,table]=GPorth(T)
```

defines the calling sequence for the basic GP algorithm which is named GPorth. The input matrix `T` is an initial value for the rotation matrix. The output matrix `Th` is the optimum value for the rotation matrix produced by the GPorth algorithm. The output value `table` is a convergence history.

The two code lines

```
f=ff(T);
G=Gf(T);
```

are calls to the subroutines ff and Gf. These calls produce the value `f` of f at `T` and the value `G` of the gradient of f at `T`.

The remaining lines are standard code for the GP algorithm.

- What follows is Matlab code to compute the gradient of f at T using numerical differentiation, specifically the symmetric difference formula.

```
function G=Gf(T)

k=size(T,1);
ep=.0001;
Z=zeros(k,k);
G=Z;

for r=1:k
   for s=1:k
      dT=Z;
      dT(r,s)=ep;
      G(r,s)=(ff(T+dT)-ff(T-dT))/(2*ep);
      end
   end
```

The first line

```
function G=Gf(T)
```

defines the calling sequence for the gradient subroutine Gf. The input matrix `T` is a rotation matrix and the output matrix `G` is an approximation to the gradient of f at `T`.

The code line

```
ep=.0001;
```

defines the increment used for the symmetric difference approximations to the partial derivatives of $f(T)$ with respect to the components of T. These are computed one component at a time in the double "for loop" that appears at the end of the code.

The code line

```
G(r,s)=(ff(T+dT)-ff(T-dT))/(2*ep);
```

produces the symmetric difference approximation to the derivative of $f(T)$ at the (r, s) component of T.

- What follows is Matlab code to compute the value of f at an arbitrary T using the quartimax criterion. Because the basic GP algorithm is designed to minimize rather than maximize, the value of f is the negative of that given by the usual quartimax criterion.

```
function f=ff(T)

global A

L=A*T;
L2=L.^2;

f=-sum(sum(L2.*L2));
```

The first line of code

```
function f=ff(T)
```

defines the calling sequence for the subroutine ff that computes the value `f` of f at the input rotation matrix `T`.

The second line

```
global A
```

identifies the initial loading matrix `A`. This might be explicitly inserted here, but it is usually more convenient to obtain it from elsewhere using a global command.

The code line

```
f=-sum(sum(L2.*L2));
```

gives the value `f` of f at `T`. This is the negative of the quartimax criterion at `L`.

An example: Thurstone's box problem

What follows is Matlab code for quartimax rotation of the initial loading matrix from Thurstone's (1947, p. 136) box problem. This uses the subroutines from the previous section. They may be used without change for any quartimax rotation problem. For some other form of orthogonal rotation say varimax only the ff subroutine needs to be changed.

```
global A

A=[
.659 -.736  .138
.725  .180 -.656
.665  .537  .500
.869 -.209 -.443
.834  .182  .508
.836  .519  .152
.856 -.452 -.269
.848 -.426  .320
.861  .416 -.299
.880 -.341 -.354
.889 -.417  .436
.875  .485 -.093
.667 -.725  .109
.717  .246 -.619
.634  .501  .522
.936  .257  .165
.966 -.239 -.083
.625 -.720  .166
.702  .112 -.650
.664  .536  .488
];

T=eye(3);

[T,table]=GPorth(T);

table
L=A*T
```

The matrix `A` is the initial loading matrix from Thurstone's box problem. The `global A` statement in the first line makes `A` available to the ff subroutine. The statement

```
T=eye(3);
```

sets the initial rotation matrix to the identity. The subroutine call

```
[T,table]=GPorth(T);
```

replaces `T` by its optimal quartimax value and generates `table`, the convergence history. The last line

```
L=A*T
```

generates the quartimax rotation `L` of the initial loading matrix `A`.

The output from this problem is:

```
table =

         0  -10.7152   -0.1406    1.0000
    1.0000  -13.2425    0.3893    2.0000
    2.0000  -14.1458    0.0407    0.2500
    3.0000  -14.1964   -0.4122    0.0625
    4.0000  -14.2029   -0.7978    0.0625
    5.0000  -14.2041   -1.0890    0.0625
    6.0000  -14.2046   -1.6858    0.1250
    7.0000  -14.2046   -2.2221    0.1250
    8.0000  -14.2046   -2.5689    0.0625
    9.0000  -14.2046   -3.1207    0.1250
   10.0000  -14.2046   -3.4477    0.0625
   11.0000  -14.2046   -4.0147    0.1250
   12.0000  -14.2046   -4.3231    0.0625
   13.0000  -14.2046   -4.9043    0.1250
   14.0000  -14.2046   -5.1958    0.0625


L =

    0.0105   -0.9934   -0.0899
```

```
0.1585   -0.1673   -0.9671
0.9823   -0.0950   -0.0819
0.1250   -0.5971   -0.7893
0.8696   -0.4716   -0.0904
0.8757   -0.1410   -0.4523
0.0679   -0.8114   -0.5886
0.4067   -0.9079   -0.1157
0.5771   -0.1424   -0.8065
0.1013   -0.7233   -0.6946
0.5001   -0.9497   -0.0468
0.7413   -0.1403   -0.6636
0.0056   -0.9838   -0.1200
0.2142   -0.1194   -0.9474
0.9551   -0.1083   -0.0392
0.7823   -0.4054   -0.4393
0.3627   -0.7531   -0.5463
0.0162   -0.9662   -0.0521
0.1077   -0.2067   -0.9346
0.9744   -0.0926   -0.0908
```

The algorithm converged smoothly in 14 iterations. To the precision displayed the rotated loading matrix `L` is identical to that produced using the orthogonal GP algorithm with exact derivatives.

A derivative free GP algorithm for oblique rotation

The general oblique rotation problem is to minimize

$$f(T) \quad \text{given} \quad \text{dg}(T'T) = I$$

where T is p by k with $p \geq k$. In factor analysis applications T is square and

$$f(T) = Q(A(T')^{-1})$$

where Q is a factor analysis rotation criterion and A is an initial loading matrix.

As in the orthogonal case, the Matlab code consists of three subroutines. These are listed without comment.

- The basic GPoblq subroutine.

```
function [T,table]=GPoblq(T)

al=1;
table=[];
for iter=0:100
   f=ff(T);
   G=Gf(T);
   Gp=G-T*diag(sum(T.*G));
   s=norm(Gp,'fro');

   table=[table;iter f log10(s) al];
   if s<10^(-5),break,end

   al=2*al;
   for i=0:10
      X=T-al*Gp;
      v=1./sqrt(sum(X.^2));
      Tt=X*diag(v);
  ft=ff(Tt);
      if ft<f-.5*s^2*al,break,end
      al=al/2;
  end
   T=Tt;
   end
```

- The Qf subroutine is exactly as it was for the orthogonal case.

```
function G=Gf(T)

k=size(T,1);
ep=.0001;
Z=zeros(k,k);
G=Z;

for r=1:k
   for s=1:k
      dT=Z;
  dT(r,s)=ep;
```

```
  G(r,s)=(ff(T+dT)-ff(T-dT))/(2*ep);
  end
   end
```

- The ff subroutine for quartimin rotation.

```
function f=ff(T)

global A

[p,k]=size(A);
L=A*inv(T');
L2=L.^2;
N=ones(k,k)-eye(k);

f=sum(sum(L2.*(L2*N)));
```

An example: Thurstone's box problem

The following Matlab code produces a quartimin rotation of the initial loading matrix from Thurstone's box problem.

```
global A

A=[
.659 -.736  .138
.725  .180 -.656
.665  .537  .500
.869 -.209 -.443
.834  .182  .508
.836  .519  .152
.856 -.452 -.269
.848 -.426  .320
.861  .416 -.299
.880 -.341 -.354
.889 -.417  .436
.875  .485 -.093
.667 -.725  .109
```

```
.717  .246 -.619
.634  .501  .522
.936  .257  .165
.966 -.239 -.083
.625 -.720  .166
.702  .112 -.650
.664  .536  .488
];

T=eye(3);

[T,table]=GPoblq(T);

table
L=A*inv(T')
phi=T'*T
```

The output from this code is

```
table =

         0    8.9209    0.1505    1.0000
    1.0000    8.9009   -0.0199    0.0156
    2.0000    8.8719    0.1131    0.0312
    3.0000    8.7644    0.4425    0.0625
    4.0000    8.6325    0.5982    0.0312
    5.0000    8.4566    0.5186    0.0156
    6.0000    8.0945    0.6242    0.0312
    7.0000    7.0348    0.8246    0.0625
    8.0000    6.2796    0.8494    0.0312
    9.0000    5.7333    0.6539    0.0156
   10.0000    5.1788    0.5326    0.0312
   11.0000    4.6907    0.4400    0.0625
   12.0000    4.0249    0.7080    0.1250
   13.0000    3.5970    0.4900    0.0312
   14.0000    3.3853    0.3835    0.0312
   15.0000    3.1567    0.3907    0.0625
   16.0000    3.0617    0.1910    0.0312
   17.0000    3.0118    0.0280    0.0312
```

```
   18.0000    2.9720   -0.0078    0.0625
   19.0000    2.9615   -0.3226    0.0156
   20.0000    2.9562   -0.4933    0.0312
   21.0000    2.9524   -0.6438    0.0625
   22.0000    2.9518   -0.9349    0.0156
   23.0000    2.9515   -1.1187    0.0312
   24.0000    2.9513   -1.2807    0.0625
   25.0000    2.9512   -1.5887    0.0156
   26.0000    2.9512   -1.7796    0.0312
   27.0000    2.9512   -1.9144    0.0625
   28.0000    2.9512   -2.2498    0.0156
   29.0000    2.9512   -2.4469    0.0312
   30.0000    2.9512   -2.5444    0.0625
   31.0000    2.9512   -2.9094    0.0156
   32.0000    2.9512   -3.1135    0.0312
   33.0000    2.9512   -3.2996    0.0312
   34.0000    2.9512   -3.4657    0.0625
   35.0000    2.9512   -3.7777    0.0156
   36.0000    2.9512   -3.9692    0.0312
   37.0000    2.9512   -4.0959    0.0625
   38.0000    2.9512   -4.4391    0.0156
   39.0000    2.9512   -4.6370    0.0312
   40.0000    2.9512   -4.7232    0.0625
   41.0000    2.9512   -5.0972    0.0156


L =

   -0.0996   -1.0236    0.0171
   -0.0071    0.0428   -1.0100
    1.0129    0.0332    0.0504
   -0.0548   -0.4493   -0.7723
    0.8563   -0.3740    0.0693
    0.8356    0.0487   -0.3604
   -0.1029   -0.7227   -0.5375
    0.3221   -0.8817    0.0312
    0.4628    0.0852   -0.7838
   -0.0766   -0.6043   -0.6583
```

```
    0.4278   -0.9289    0.1229
    0.6594    0.0772   -0.6073
   -0.1088   -1.0080   -0.0174
    0.0595    0.0956   -0.9868
    0.9899    0.0072    0.0947
    0.7137   -0.2427   -0.3283
    0.2203   -0.6354   -0.4597
   -0.0847   -1.0022    0.0557
   -0.0592   -0.0113   -0.9769
    1.0034    0.0365    0.0394


phi =

    1.0000   -0.2568   -0.3216
   -0.2568    1.0000    0.3366
   -0.3216    0.3366    1.0000
```

To the precision displayed the rotated loading matrix `L` and the factor correlation matrix `phi` are identical to those produced using the oblique GP algorithm with exact derivatives.

References

Jennrich, R.I. (2001). A simple general proceedure for orthogonal rotation. *Psychometrika, 66*, 289-306.

Jennrich, R.I. (2002). A simple general method for oblique rotation. *Psychometrika, 67*, 7-19.

Jennrich, R.I. (2003). Derivative free gradient projection algorithms for rotation. Submitted to Psychometrika.

Matlab (1995). The MathWorks Inc., 24 Prime Park Way, Natick, MA, 01760.

Thurstone, L.L. (1947). *Multiple Factor Analysis*. Chicago: University of Chicago Press.