Tp2

- **Author: MITCHOZOUNOU Jean-CLaude**
  - **Institute: UM6P**

Newton avec région de confiance (Cas multidimensionnel)

1- Gradient conjugué linéaire

1.1 **Gradient simple**

```
Grad_Mat.sci


function [dN,dNQ,niter,L_f] = Grad_Mat(c,Q,eps,MaxIter,verbose,delt0)
// résout le système linéaire Q*dN+b=0 par la descente du gradient simple.
// on présume que Q est la matrice hessienne et c le gradient d'une
fonction
```

```
// à  minimiser.
//
//
// Entrée:
//    c(1,n), Q(n,n) paramètres de la fonction quadratique à minimiser
//        q(delt) = 0.5*delt'*Q*delt + c*delt
//        nabla q(delt) = delt'*Q + c
//    eps tolérance d'arrêt
//    MaxIter  nombre maximum d'itérations
//    verbose  imprime les itérations (>0) ou non (<=0)
//    delt0(n,1)  point de départ des itérations
//
// Sortie:
//    dN: la solution  :dN*Q+b=0
//    niter: nombre total d'itérations
//    L_f: liste des valeurs de fonctions au fil des itérations
//

  niter = 0;
  L_f = [];
  [bidon,dim] = size(c)

  delt = delt0;
  deltQ = delt'*Q;
  nablaq = c + deltQ;

  pad = '';
  if verbose >0,
    s='   ';
    for i=1:verbose,
      pad = pad + s
    end
    mprintf("%s iter    q(delt)  ||nabla q(delt)||\n",pad)
    mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
        norm(nablaq))
  end

  norm2nablaq = nablaq*nablaq';
  pasprecis = norm2nablaq>eps^2;

  while pasprecis & (niter < MaxIter)
    niter = niter + 1;
    p = -nablaq';
    pQ = p'*Q;
    pQp = pQ*p

    theta=norm2nablaq/(pQp);
    if theta<0.0
      warning("Q not positive ")
    end

    delt = delt + theta*p;
    deltQ = deltQ + theta*pQ;
```

```
    nablaq = nablaq + theta*pQ;

    norm2nablaq = nablaq*nablaq'
    pasprecis = norm2nablaq>eps^2;
    if verbose>0,
      mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
          norm(nablaq))
    end
    L_f(niter) = c*delt + 0.5*deltQ*delt;
  end
  dN = delt;
  dNQ = deltQ;
endfunction


 // reproduit les calculs du tableau de  f(x,y)=2x^2-2xy+y^2+2x-2y

H=[4 -2;-2 2];
b=[2 -2];
n=2;
delt0=[10;5];

xstar = -H\b';  // Solution optimale

verbose = 1;
MaxIter = 20;



[dNG,dNQ,ngc,L_f] = Grad_Mat(b,H,1e-8,MaxIter,verbose,delt0);
```

Output

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/Grad_Mat.sci', -1)
   iter    q(delt)  ||nabla q(delt)||
     0  135.0000000  3.4176015e+01
     1  19.7783784  5.7267917e+00
     2   2.1745662  5.2214865e+00
     3  -0.5149828  8.7495180e-01
     4  -0.9258980  7.9775017e-01
     5  -0.9886785  1.3367706e-01
     6  -0.9982703  1.2188202e-01
     7  -0.9997357  2.0423474e-02
     8  -0.9999596  1.8621403e-02
     9  -0.9999938  3.1203431e-03
    10  -0.9999991  2.8450187e-03
    11  -0.9999999  4.7673287e-04
    12  -1.0000000  4.3466821e-04
    13  -1.0000000  7.2836294e-05
    14  -1.0000000  6.6409562e-05
    15  -1.0000000  1.1128089e-05
    16  -1.0000000  1.0146199e-05
    17  -1.0000000  1.7001738e-06
    18  -1.0000000  1.5501585e-06
    19  -1.0000000  2.5975629e-07
    20  -1.0000000  2.3683661e-07

-->
```

1.2 **Gradient conjugué linéaire avec matrice**

GC_Mat.sci

```
function [dN,dNQ,niter,L_f] = GC_TR(c,Hv,x,eps,Delta,MaxIter,verbose)
    // résout le système linéaire Hv*dN+b=0 par la descente du gradient
simple.
    // on présume que Hv est la matrice hessienne et c le gradient d'une
fonction
    // à  minimiser.
    //
    //
    // Entrée:
    //    c(1,n), Hv(n,n) paramètres de la fonction quadratique à minimiser
    //        q(delt) = 0.5*delt'*Hv*delt + c*delt
    //        nabla q(delt) = delt'*Hv + c
    //        x  la variable dont depend Hv
    //    eps tolérance d'arrêt
    //    MaxIter  nombre maximum d'itérations
    //    verbose  imprime les itérations (>0) ou non (<=0)
    //    delt0(n,1)  point de départ des itérations
    //
    // Sortie:
    //    dN: la solution  :dN*Q+b=0
    //    niter: nombre total d'itérations
    //    L_f: liste des valeurs de fonctions au fil des itérations
    //
```

```
        niter = 0;
        L_f = [];
        [bidon,dim] = size(c)

        delt = delt0;


        deltQ = Hv(x,delt);
        nablaq = c + deltQ;
        d = -nablaq'
        bet = 0
        pad = '';
        if verbose >0,
          s='   ';
          for i=1:verbose,
            pad = pad + s
          end
          mprintf("%s iter    q(delt)  ||nabla q(delt)||\n",pad)
          mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
              norm(nablaq))
        end
        dQ = Hv(x,d);
        dQd = dQ*d;

        norm2nablaq = nablaq*nablaq';
        pasprecis = norm2nablaq>eps^2;



        sortie = %F
        while (~sortie & pasprecis) & (niter < MaxIter)
          niter = niter + 1;

          dQ = Hv(x,d);
          dQd = dQ*d;


          // Calcul de thetamax: Resoudre || delt + theta*d||<=Delta
          A = d'*d
          B = 2*(d'*delt)
          C = delt'*delt - Delta
          Discriminant = B^2 -4*A*C
          if Discriminant <0
              sortie = %T
          else
              racines = [-B-sqrt(Discriminant)/(2*A),-
B+sqrt(Discriminant)/(2*A)] // Racine
              thetamax = max(racines)
              if thetamax<0
                  thetamax = 1
              end
              theta=(-nablaq*d)/(dQd);
              if (theta<=0.0)|| (theta>thetamax)
```

```
                theta = thetamax
            end
            delt = delt + theta*d;
            deltQ = deltQ + theta*dQ;
            nablaq = nablaq + theta*dQ;

            bet = (dQ*nablaq')/(dQd)
            d = -nablaq' + bet*d;

            norm2nablaq = nablaq*nablaq'
            pasprecis = norm2nablaq>eps^2

        end


        if verbose>0,
       mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
              norm(nablaq))
        end
        L_f(niter) = c*delt + 0.5*deltQ*delt;
      end
      dN = delt;
      dNQ = deltQ;
endfunction
```

Output

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/GC_Mat.sci', -1)
   iter    q(delt)  ||nabla q(delt)||
     0   0.0000000  2.0579822e+00
     1  -0.3675485  2.1293138e+00
     2  -0.7681128  2.1284560e+00
     3  -1.1031739  1.2278968e+00
     4  -1.1967173  5.8375108e-01
     5  -1.2131550  2.0060845e-15

-->
```

1.3 **Gradient conjugué linéaire sans matrice: La fonction**

GC_Hv.sci

```
function [dN,dNQ,niter,L_f] = GC_Hv(c,Hv,x,eps,MaxIter,verbose,delt0)
// résout le système linéaire Hv*dN+b=0 par la descente du gradient simple.
// on présume que Hv est la matrice hessienne et c le gradient d'une
fonction
// à  minimiser.
```

```
//
//
// Entrée:
//    c(1,n), Hv(n,n) paramètres de la fonction quadratique à minimiser
//        q(delt) = 0.5*delt'*Hv*delt + c*delt
//        nabla q(delt) = delt'*Hv + c
//        x  la variable dont depend Hv
//    eps tolérance d'arrêt
//    MaxIter  nombre maximum d'itérations
//    verbose  imprime les itérations (>0) ou non (<=0)
//    delt0(n,1)  point de départ des itérations
//
// Sortie:
//    dN: la solution  :dN*Q+b=0
//    niter: nombre total d'itérations
//    L_f: liste des valeurs de fonctions au fil des itérations
//

  niter = 0;
  L_f = [];
  [bidon,dim] = size(c)

  delt = delt0;


  deltQ = Hv(x,delt);
  nablaq = c + deltQ;
  d = -nablaq'
  bet = 0
  pad = '';
  if verbose >0,
    s='   ';
    for i=1:verbose,
      pad = pad + s
    end
    mprintf("%s iter    q(delt)  ||nabla q(delt)||\n",pad)
    mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
        norm(nablaq))
  end

  norm2nablaq = nablaq*nablaq';
  pasprecis = norm2nablaq>eps^2;

  while pasprecis & (niter < MaxIter)
    niter = niter + 1;
    d = -nablaq' + bet*d;

    dQ = Hv(x,d);
    dQd = dQ*d;

    theta=(-nablaq*d)/(dQd);

    if theta<0.0
```

```
        warning("Q not positive ")
      end

      delt = delt + theta*d;
      deltQ = deltQ + theta*dQ;
      nablaq = nablaq + theta*dQ;

      bet = (dQ*nablaq')/(dQd)
      norm2nablaq = nablaq*nablaq'
      pasprecis = norm2nablaq>eps^2;
      if verbose>0,
        mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
  0.5*deltQ*delt), ...
             norm(nablaq))
      end
      L_f(niter) = c*delt + 0.5*deltQ*delt;
    end
    dN = delt;
    dNQ = deltQ;
  endfunction
```

Output

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/GC_Hv.sci', -1)
Warning : redefining function: GC_Hv                    . Use funcprot(0) to avoid this message
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  2.4035384e+00
      1  -0.5803006  2.6571814e+00
      2  -1.1104406  2.2711391e+00
      3  -1.5406532  1.7025778e+00
      4  -1.7099427  6.2816576e-01
      5  -1.7281704  1.3478936e-15

-->
```

Appel simultané de GC_Mat et GC_Hv

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/jc.sci', -1)
Warning : redefining function: GC_Mat                   . Use funcprot(0) to avoid this message
Warning : redefining function: GC_Hv                    . Use funcprot(0) to avoid this message
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  2.2707848e+00
      1  -0.5014701  2.4961653e+00
      2  -1.0995013  2.1444699e+00
      3  -1.4250745  1.6553742e+00
      4  -1.5672691  5.5478412e-01
      5  -1.5825016  1.7945528e-15
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  2.2707848e+00
      1  -0.5014701  2.4961653e+00
      2  -1.0995013  2.1444699e+00
      3  -1.4250745  1.6553742e+00
      4  -1.5672691  5.5478412e-01
      5  -1.5825016  1.7945528e-15

-->
```

norm(dNCGMat - dNCGHv)

```
 "Norm(dNCGMat - dNCGHv) = "

  0.

-->
```

2- Région de confiance

2.1 **Adaptation de** *GC_Hv : GC_TR.sci*

```
function [dN,dNQ,niter,L_f] = GC_TR(c,Hv,x,eps,Delta,MaxIter,verbose)
    // résout le système linéaire Hv*dN+b=0 par la descente du gradient
simple.
    // on présume que Hv est la matrice hessienne et c le gradient d'une
fonction
    // à  minimiser.
    //
    //
    // Entrée:
    //    c(1,n), Hv(n,n) paramètres de la fonction quadratique à minimiser
    //        q(delt) = 0.5*delt'*Hv*delt + c*delt
    //        nabla q(delt) = delt'*Hv + c
    //        x  la variable dont depend Hv
    //    eps tolérance d'arrêt
    //    MaxIter  nombre maximum d'itérations
    //    verbose  imprime les itérations (>0) ou non (<=0)
    //    delt0(n,1)  point de départ des itérations
    //
    // Sortie:
    //    dN: la solution  :dN*Q+b=0
    //    niter: nombre total d'itérations
    //    L_f: liste des valeurs de fonctions au fil des itérations
    //

      niter = 0;
      L_f = [];
      [bidon,dim] = size(c)

      delt = delt0;


      deltQ = Hv(x,delt);
      nablaq = c + deltQ;
      d = -nablaq'
      bet = 0
      pad = '';
      if verbose >0,
        s='   ';
```

```
        for i=1:verbose,
          pad = pad + s
        end
        mprintf("%s iter    q(delt)  ||nabla q(delt)||\n",pad)
        mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
            norm(nablaq))
      end
      dQ = Hv(x,d);
      dQd = dQ*d;

      norm2nablaq = nablaq*nablaq';
      pasprecis = norm2nablaq>eps^2;



      sortie = %F
      while (~sortie & pasprecis) & (niter < MaxIter)
        niter = niter + 1;

        dQ = Hv(x,d);
        dQd = dQ*d;


        // Calcul de thetamax: Resoudre || delt + theta*d||<=Delta
        A = d'*d
        B = 2*(d'*delt)
        C = delt'*delt -Delta^2
        Discriminant = B^2 -4*A*C
        if Discriminant <0
            sortie = %T
        else
            racines = [-B-sqrt(Discriminant)/(2*A),-
B+sqrt(Discriminant)/(2*A)] // Racine
            thetamax = max(racines)
            if thetamax<0
                //sortie = %T
                thetamax = 1
            end
            theta=(-nablaq*d)/(dQd);
            if (theta<=0.0)|| (theta>thetamax)
                theta = thetamax
            end
            delt = delt + theta*d;
            deltQ = deltQ + theta*dQ;
            nablaq = nablaq + theta*dQ;

            bet = (dQ*nablaq')/(dQd)
            d = -nablaq' + bet*d;


            norm2nablaq = nablaq*nablaq'
            pasprecis = norm2nablaq>eps^2
```

```
        end


        if verbose>0,
       mprintf("%s %3d  %10.7f  %10.7e\n",pad,niter, (c*delt +
0.5*deltQ*delt), ...
              norm(nablaq))
        end
        L_f(niter) = c*delt + 0.5*deltQ*delt;
      end
      dN = delt;
      dNQ = deltQ;
endfunction
```

- Exécution des trois instances de problèmes oú la taille de région de confiance est ajustéé pour valider que votre implantation semble fournir des r´esultats corrects

```
// Générateur de problèmes

n = 15;

// valeurs propres extrêmes
lambda_1 = 1;
lambda_n = 20;
range = lambda_n-lambda_1;

lambda = lambda_1:range/(n-1):lambda_n
// Matrice diagonale avec les valeurs propres étalées entre lambda_1 et
lambda_n
Lambda = diag(lambda);

// Générons une matrice de rotation "aléatoire".
M=rand(n,n);
[Q,R] = qr(M);

// H est une rotation de la matrice diagonale Lambda
H = Q*Lambda*Q';

// un vecteur b aléatoire
b = rand(1,n);
delt0=zeros(n,1);


function [Hv] = Hv(x,v)
  Hv = v'*H;
endfunction

verbose = 1;
MaxIter = 20;
```

```
x = zeros(1,n);
v = 1
exec ("GC_Hv.sci",0);

[dNCGHv,dNQ,ngc,L_fGC] = GC_Hv(b,Hv,x,1e-8,MaxIter,verbose,delt0);


exec ("GC_TR.sci",0);

Delta = (1.01)*norm(dNCGHv);

[dNTR,dNQ,ngc] = GC_TR(b,Hv,x,1e-8,Delta,MaxIter,verbose);

nor = norm(dNCGHv-dNTR) // devrait être exactement 0

disp(nor)
```

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/test3.sce', -1)
Warning : redefining function: GC_Hv                    . Use funcprot(0) to avoid this messag
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  1.7935610e+00
      1  -0.3406179  2.4069946e+00
      2  -0.8960431  1.6571100e+00
      3  -1.0283054  8.5982444e-01
      4  -1.0804693  5.4484094e-01
      5  -1.1046950  2.9662239e-01
      6  -1.1109847  1.5576651e-01
      7  -1.1121443  7.4101788e-02
      8  -1.1124710  3.8949485e-02
      9  -1.1125702  1.4256698e-02
     10  -1.1125803  3.1813802e-03
     11  -1.1125808  7.0038929e-04
     12  -1.1125809  1.9993183e-04
     13  -1.1125809  7.0166216e-05
     14  -1.1125809  2.3592911e-05
     15  -1.1125809  5.6142642e-17
Warning : redefining function: GC_TR                    . Use funcprot(0) to avoid this messag
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  1.7935610e+00
      1  -0.3406179  2.4069946e+00
      2  -0.8960431  1.6571100e+00
      3  -1.0283054  8.5982444e-01
      4  -1.0804693  5.4484094e-01
      5  -1.1046950  2.9662239e-01
      6  -1.1109847  1.5576651e-01
      7  -1.1121443  7.4101788e-02
      8  -1.1124710  3.8949485e-02
      9  -1.1125702  1.4256698e-02
```


```
      7  -1.1121443  7.4101788e-02
      8  -1.1124710  3.8949485e-02
      9  -1.1125702  1.4256698e-02
     10  -1.1125803  3.1813802e-03
     11  -1.1125808  7.0038929e-04
     12  -1.1125809  1.9993183e-04
     13  -1.1125809  7.0166216e-05
     14  -1.1125809  2.3592911e-05
     15  -1.1125809  5.6142642e-17
```


```
 0.
```

```
Delta = (0.99)*norm(dNCGHv);

[dNTR,dNQ,ngc] = GC_TR(b,Hv,x,1e-8,Delta,MaxIter,verbose);

disp(norm(dNCGHv-dNTR)) // devrait être proche de 0
```

```
Warning : redefining function: GC_Hv                    . Use funcpr
    iter     q(delt)  ||nabla q(delt)||
      0   0.0000000  1.9884190e+00
      1  -0.4353743  1.9584381e+00
      2  -0.7446707  1.5958482e+00
      3  -0.9284409  8.2998053e-01
      4  -0.9770177  6.7293991e-01
      5  -1.0113794  4.8495007e-01
      6  -1.0246714  2.5681960e-01
      7  -1.0287792  1.1503343e-01
      8  -1.0294367  5.3625791e-02
      9  -1.0295971  2.2406671e-02
     10  -1.0296211  9.8200617e-03
     11  -1.0296271  3.6393841e-03
     12  -1.0296277  1.1194005e-03
     13  -1.0296278  3.5574903e-04
     14  -1.0296278  5.6106018e-06
     15  -1.0296278  3.4963986e-17
Warning : redefining function: GC_TR                    . Use funcpr
    iter     q(delt)  ||nabla q(delt)||
      0   0.0000000  1.9884190e+00
      1  -0.4353743  1.9584381e+00
      2  -0.7446707  1.5958482e+00
      3  -0.9284409  8.2998053e-01
      4  -0.9770177  6.7293991e-01
      5  -1.0113794  4.8495007e-01
      6  -1.0246714  2.5681960e-01
      7  -1.0287792  1.1503343e-01
      8  -1.0294367  5.3625791e-02
      9  -1.0294367  5.3625791e-02

   0.0086555
```

```
Delta = (0.5)*norm(dNCGHv);

[dNTR,dNQ,ngc] = GC_TR(b,Hv,x,1e-8,Delta,MaxIter,verbose);

disp(norm(dNCGHv-dNTR)) // devrait être loin de 0
```

```
-->exec('/home/jc1er/UM6P_QFM_S2/Stochastic_Optimization/TP2_Stochastique/t
Warning : redefining function: GC_Hv                    . Use funcprot(0) to
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  2.3067099e+00
      1  -0.8649064  2.8533763e+00
      2  -1.3930177  1.5415050e+00
      3  -1.5358305  1.1576399e+00
      4  -1.6391729  7.7278634e-01
      5  -1.6864069  6.2790000e-01
      6  -1.7089099  2.3409912e-01
      7  -1.7120269  1.0019909e-01
      8  -1.7124522  3.7114689e-02
      9  -1.7125436  1.4853339e-02
     10  -1.7125525  5.3899460e-03
     11  -1.7125548  2.4535531e-03
     12  -1.7125551  5.1183429e-04
     13  -1.7125551  1.2578844e-04
     14  -1.7125551  1.4991997e-05
     15  -1.7125551  4.9403284e-17
Warning : redefining function: GC_TR                    . Use funcprot(0) to
    iter    q(delt)  ||nabla q(delt)||
      0   0.0000000  2.3067099e+00
      1  -0.8649064  2.8533763e+00
      2  -1.3930177  1.5415050e+00
      3  -1.3930177  1.5415050e+00

   0.6604050

-->
```

2.2 **Test de l'algorithme**