The University of Connecticut **Fall 2009**
Dept. of ECE **KRP**

Problem Set # 3

Part A: Analytical (Due September 22, 2009)

1. The Armijo step size rule can be combined with a quadratic interpolation scheme as follows. Given s, s_{LB} (a lower bound on s), β, and σ such that $0 < s_{LB} < s$, $0 < \beta < 1$, $0 < \sigma < 1/2$, the current point $\underline{x}_k$ and the direction $\underline{d}_k$, compute $f(\underline{x}_k + s\underline{d}_k)$. The procedure is:

   a) Fit a parabola $h(\alpha) = a\alpha^2 + b\alpha + c$ such that $h(0) = f(\underline{x}_k)$, $h(s) = f(\underline{x}_k + s\underline{d}_k)$, and $h'(0) = \underline{g}_k^T \underline{d}_k$, where $\underline{g}_k$ is the gradient of $f(\underline{x})$ at $\underline{x} = \underline{x}_k$. Determine a, b, and c. The minimum point is $\alpha_k^* = -b/2a$.

   b) The initial step size for the Armijo rule is determined from:

$$s_k = \begin{cases} s & \text{if } f(\underline{x}_k + \alpha^* \underline{d}_k) \geq f(\underline{x}_k + s\underline{d}_k) \\ \alpha^* & \text{if } f(\underline{x}_k + \alpha^* \underline{d}_k) < f(\underline{x}_k + s\underline{d}_k) \\ s_{LB} & \text{otherwise} \end{cases}$$

   c) Apply Armijo step size rule.

   Devise an algorithm based on a), b) and c) above.

2. Use Golden Section Search to determine (within an interval of 0.6) the optimal solution to: min $(e^x - x)$ subject to $-1 \leq x \leq 3$.

3. (*How to Estimate the rate of convergence and number of iterations?*)

   *C*onsider the problem

$$\text{min}imize\ f(x,y) = 5x^2 - xy + 5y^2 - 11x + 11y + 11$$

   a) Find a point satisfying the first-order necessary conditions for a solution.
   b) Show that the point is a global minimum.
   c) What would be the rate of convergence of steepest descent for this problem?
   d) Starting at $x=y=0$, how many steepest descent iterations would it take (at most) to reduce the function value to 10^{-11}?

4. (*How to effectively reduce the condition number or improve convergence rate?*)

   Let $\underline{\xi}_1, \underline{\xi}_2, \ldots, \underline{\xi}_n$ denote the eigenvectors of the symmetric positive definite nxn matrix Q. For the quadratic problem considered in class, suppose $\underline{x}_0$ is chosen so that $\underline{g}_0$ belongs to a subspace M spanned by a subset of the $\{\underline{\xi}_i\}$. Show that for the method of steepest descent $\underline{g}_k \in M$ for all k. Find the rate of convergence in this case.

5. (*Why Steepest Descent can tolerate inaccurate line searches?*)

   Suppose we use the method of steepest descent to minimize the quadratic function $f(\underline{x}) = \frac{1}{2}(\underline{x} - \underline{x}^*)^T Q(\underline{x} - \underline{x}^*)$ but we allow a tolerance of $\pm\delta\alpha_k, \delta \geq 0$ in the line search, that is,

$$\underline{x}_{k+1`} = \underline{x}_k - \alpha_k \underline{g}_k,$$

   where

$$(1-\delta)\overline{\alpha}_k \leq \alpha_k \leq (1+\delta)\overline{\alpha}_k$$

and $\bar{\alpha}_k$ minimizes $f(\underline{x}_k - \alpha \underline{g}_k)$ over α.

a) Find the convergence rate of the algorithm in terms of $\lambda_{\max}$ and $\lambda_{\min}$, the largest and the smallest eigenvalues of Q and the tolerance δ. (*Hint:* Assume the extreme cas $\alpha_k = (1 \pm \delta)\bar{\alpha}_k$).

b) What is the largest δ that guarantees convergence of the algorithm? Explain the result geometrically.

c) Does the sign of δ make any difference?

6. (*Stopping Criterion and Aitken's acceleration process*)

A question that arises in using an algorithm such as the steepest descent to minimize an objective function *f* is when to stop the iterative process. In other words, how can one tell when the current point is close to a solution? If, as with the steepest descent, it is known that convergence is linear, this knowledge can be used to develop a stopping criterion. Let $\{f_k\}_{k=0}^{\infty}$ be the sequence of values obtained by the algorithm. We assume that $f_k \to f^*$ linearly, but both f^* and the convergence ratio β are unknown. However, we know that, at least approximately,

$$f_{k+1} - f^* = \beta(f_k - f^*)$$

and

$$f_k - f^* = \beta(f_{k-1} - f^*)$$

These two equations can be solved for β and f^*.

a) Show that

$$f^* = \frac{f_k^2 - f_{k-1} f_{k=1}}{2f_k - f_{k-1} - f_{k+1}}$$

$$\beta = \frac{f_{k+1} - f_k}{f_k - f_{k-1}}$$

b) Motivated by the above, we form the sequence $\{f_k^*\}$ defined by

$$f_k^* = \frac{f_k^2 - f_{k-1} f_{k=1}}{2f_k - f_{k-1} - f_{k+1}}$$

as the original sequence is generated. This procedure of generating $\{f_k^*\}$ from $\{f_k\}$ is called the Aitken δ^2-process. If $|f_k - f^*| = \beta^k + O(\beta^k)$, show that $|f_k^* - f^*| = O(\beta^k)$, which means that $\{f_k^*\}$ converges to f^* faster than $\{f_k\}$ does. The iterative search for the minimum can then be terminated when $(f_k - f_k^*)$ is smaller than some prescribed tolerance.

7. Problem 1.2.6 of Bertsekas, Page 55.

Part B: Computational (Due October 6, 2009)

Extensively test the computer subroutines LSEARCH (Golden section + quadratic interpolation) and LSEARCH2 (Armijo + quadratic interpolation) for minimizing a scalar function $(0,\infty)$. The program LSEARCHoperates in three steps:

1. Finds three point pattern;
2. Use golden section search until the uncertain interval is cut sufficiently short;
3. Use quadratic fit to obtain the final result.

Sample MATLAB code is in huskyct for LSEARCH and LSEARCH2, as well as some old Fortran code for LSEARCH only below. My recommendation is that you use the attached code as a guideline for writing your own code. The inputs of LSEARCH (for the Fortran version below) include: FNCL (function value at x=0), EST (a priori estimate of the optimal), DELTA (step length to obtain a three point pattern), and HOF (value of the function for a given x). The outputs include: ALPHA (the optimal step size), FNCL (optimal function value), and NFCNS (number of function evaluations). You should write a routine that computes EST and DELTA. The choice of a rich set of test functions that demonstrate your effort and ideas will be critical. Test functions should include at least 10 scalar functions of your choice. These functions should span polynomials, ratio of polynomials, trigonometric and logarithmic functions. Furthermore, you should consider functions that are piecewise polynomial and discontinuous. Note that all functions must have derivatives less than zero at zero.

1. Examine 2 or more techniques for picking the initial parameters EST and DELTA. For this part, assume the following values for two parameters: PG=30 and PQ = 10. Discuss your results in detail. Tell which technique you recommend on the basis of numerical testing using your set of test functions.
2. Determine the "best" value of PG if PQ is fixed at 10, 100, 1000, and 10000 (Note that PG > PQ). For this work, assume EST is as you found in part 1, but take DELTA = EST.

Note that you will need to write specific code for each function to be minimized. Also, you may need a function that returns $df(x)/dx$, especially if you plan to get EST from $df(x)/dx|_{x=0}$. **Do this and all future computer assignments in Teams of two.**

All future programs will be graded as follows:

a. Working program 25%

b. Program and Algorithm Documentation 25%

c. Extensive Testing 25%

d. Discussion of Results 25%

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     Subroutines for line search algorithms
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine lsearch(fxk,gradf,xk,dk,alpha,nfcnev)
      include 'param.dim'
      real xk(nn),dk(nn),gradf(nn),fxk,alpha,eta,est,
     $                gfder,delta,s,sigma,scale
      integer nfcnev,flag1,lines,nfcnf
      common/line/lines,eta,flag1,s,sigma,scale
      common/gradn/n,h,idiff
      if (lines.eq.1) then
c     **    Check function and derivative
            if (fxk.eq.0.) fxk=fxk+eta
102         gfder=dot(n,gradf,dk)
            if (gfder.le.0) then
                  if (gfder.eq.0) gfder=gfder-eta
                  if (flag1.eq.1) call init1(n,fxk,gfder,est,delta)
                  if (flag1.eq.2) then
                     call init2(n,xk,dk,fxk,gfder,nfcnf,est,delta)
                     nfcnev=nfcnev+nfcnf
                  endif
            else
                  do 120 i=1,n
                        dk(i)=-gradf(i)
120               continue
                  print *,' '
c                 print *,'****** ERROR :Gradient TOO BIG *****'
c                 STOP
            endif
            call lserch(n,xk,dk,alpha,fxk,nfcns,est,delta,pq,pg,tol)
            nfcnev=nfcnev+nfcns
      else
```

```
            call lserc2(n,gradf,dk,nfcnf,s,sigma,scale,alpha,fxk,xk)
            nfcnev=nfcnev+nfcnf
        endif
        return
        end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     SUBROUTINE line search : uses GOLDEN SECTION rule and
c                              QUADRATIC INTERPOLATION
c
        SUBROUTINE LSERCH(n,xk,dk,alpha,fxk,nfcns,est,delta,pq,pg,tol)
        include 'param.dim'
        integer n
        real xk(nn),dk(nn)
        real *8 dble,x0,y0,s1,s2
        common/a3/nfcnev,ndrv,iter,ipr,ifg
        alpmax = 10000
        alpmin = 1e-05
        nalpha = 0
        alpha = 0.0
        tau = 1.61803
        rtau = 0.61803
        nfcns=0
        z0 = 0.0
        hz0 = fxk
c
c     Three point pattern set up
c           if delta = est use h(0,0) point
c           if delta .ne. est use h(est-delta) point
c
        if(abs(delta-est).le.tol*est) goto 400
        z1 = est
        hz1 = hof(n,xk,z1,dk)
        nfcns=nfcns+1
        if(hz1 .ge. hz0) goto 100
```

```
380 z0 = z1 - delta
      if(z0.gt.alpmin)          goto 390
      z0=alpmin
      hz0=hof(n,xk,z0,dk)
      nfcns=nfcns+1
      goto 500
390       hz0 = hof(n,xk,z0,dk)
      nfcns=nfcns+1
      if(hz0.gt.hz1)    goto 156
      z2=z1
      z1 = z0
      hz2=hz1
      hz1 = hz0
      goto 380
400       z0 = 0.0
      hz0 = fxk
155       z1 = delta + z0
      hz1 = hof(n,xk,z1,dk)
      nfcns=nfcns+1
      if(hz1.ge.hz0)    goto 100
156 r = 1.0
      l = 1
300       l = l + 1
      r = r*tau
      z2 = z1 + r*delta
c
c     Check if z2 is still within aloowable range
c
           if(z2.le.alpmax)    goto 302
c
c            Have    hit the boundary in an attempt to set up a 3 point pattern
c            set alpha to alpmax and see if hof(alpmax) < hof(z1)
c
      z2 = alpmax
      hz2 = hof(n,xk,z2,dk)
```

```
      nfcns=nfcns+1
      if(hz2.gt.hz1)    goto 500
c
c     hz2>hz1 ==>3 point pattern found( not in GS): goto Quadratic fit,
c              hz2<hz1 ==>alpmax is best alpha in this direction
c
      goto 305
302 hz2 = hof(n,xk,z2,dk)
      nfcns=nfcns+1
      if(hz2.ge.hz1)    goto 200
      z0 = z1
      hz0 = hz1
      z1 = z2
      hz1 = hz2
      if(I.lt.15)    goto 300
      print *,'could not bracket minimum in 15 steps'
305       alpha = z2
      hz1 = hz2
306 print *,' Could not set up 3-pt pattern; returning best alpha'
      goto 900
100       z2 = z1
      hz2 = hz1
      z1 = z0 + rtau*rtau*(z2-z0)
      hz1 = hof(n,xk,z1,dk)
      nfcns=nfcns+1
      if(hz1.lt.hz0)    goto 200
      if(abs(z1).ge.alpmin)    goto 100
747       continue
      alpha = 0.0
      hz1 = hz0
      goto 900
c
c     Start the GOLDEN SECTION search to PG accuracy
c
200       continue
```

```
      iflg = 1
      alpha = z1
450       ppg = ((z2-z0)/(2.0*z1))*100.0
451       nalpha = nalpha + 1
      if(nalpha.lt.11)          goto 505
      alpha = z1
      goto 900
505       if(ppg.le.pg)    goto 500
      if(abs(z1).lt.alpmin)    goto 747
      if(iflg.eq.1)    goto 250
      z = z1
      hz = hz1
      z1 = z0 + rtau*(z-z0)
      hz1 = hof(n,xk,z1,dk)
      nfcns=nfcns+1
      goto 800
250   z = z0 + rtau*(z2-z0)
      hz = hof(n,xk,z,dk)
      nfcns=nfcns+1
800 if(hz.gt.hz1)    goto 600
      z0 = z1
      hz0 = hz1
      z1 = z
      hz1 = hz
      iflg = 1
      goto 450
600       z2 = z
      hz2 = hz
      iflg = 0
      goto 450
c
c     Start QUADRATIC fit to PQ accuracy
c
500       continue
510 alpha = z1
```

```
      ppq = ((z2-z0)/(2.0*z1))*100.0
455 if(nalpha.gt.20)          goto 900
      if(ppq.le.PQ)    goto 900
      if(abs(z1).lt.alpmin)    goto 747
      nalpha = nalpha + 1
      A32 = z2 - z1
      A21 = z1 - z0
      s1 = dble(hz0 - hz1)/dble(A21)
      s2 = dble(hz2 - hz1)/dble(A32)
      if(s1.lt.0.0000000001.AnD.s2.lt.0.0000000001) goto 900
456       x0= dble(A32)*s1 - dble(A21)*s2
      y0= s1 + s2
      x0= (x0/y0) * 0.5D0
      amin = dble(z1) + x0
      if(amin.lt.alpmax)    goto 512
      alpha = z1
      goto 900
512 if(abs(amin-z1).le.tol*z1) goto 900
      hmin = hof(n,xk,amin,dk)
      nfcns=nfcns+1
      if(amin.gt.z1)    goto 540
      if(hmin.ge.hz1)    goto 530
      z2 = z1
      hz2 = hz1
525 z1 = amin
      hz1 = hmin
      goto 510
530       z0 = amin
      hz0 = hmin
      goto 510
540       if(hmin.ge.hz1)    goto 550
      z0 = z1
      hz0 = hz1
      goto 525
550       z2 = amin
```

```
      hz2 = hmin
      goto 510
900 fxk=hz1
      do 50 i=1,n
          xk(i)=xk(i)+alpha*dk(i)
50    continue
      return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     SUBROUTINE line search : uses ARMIJO STEP SIZE rule and
c                              QUADRATIC INTERPOLATION
c
      SUBROUTINE LSERC2(n,gradf,dk,nfcnf,s,sigma,scale,alpha,fxk,
     $                                    xk)
      include 'param.dim'
      integer n,nfcnf
      real hof,gradf(nn),dk(nn),s,sigma,scale,alpha,lambdak,xk(nn),
     $           temp,galpha,glambda,fxk,l
      l=.1
      alpmax=1000
      alpmin=1e-07
      nfcnf=0
      alpha=s
      temp=dot(n,gradf,dk)
100 galpha=hof(n,xk,alpha,dk)
      nfcnf=nfcnf+1
      if (nfcnf.ge.100) then
          print *,'# of fn evaluations exceeded; nfcnf =',nfcnf
          goto 200
      endif
c     execute this while condition on alpha holds; else exit
      if (galpha.gt.(fxk+sigma*alpha*temp)) then
          lambdak=-0.5*(alpha*alpha*temp)/(gaplha-alpha*temp-fxk)
          glambda=hof(n,xk,lambdak,dk)
```

```
            nfcnf=nfcnf+1
c     update alpha
            if (glambda.lt.galpha) then
                  if (lambdak.ge.l*alpha) then
                        alpha=lambdak
                  else
                        alpha=l*alpha
                  endif
            else
                  alpha=alpha/scale
            endif
c     check if we have hit the boundary; then exit
            if (alpha.gt.alpmax) then
                  alpha=alpmax
                  galpha=hof(n,xk,alpha,dk)
                  nfcnf=nfcnf+1
                  goto 200
            endif
            if (alpha.lt.alpmin) then
                  alpha=alpmin
                  galpha=hof(n,xk,alpha,dk)
                  nfcnf=nfcnf+1
                  goto 200
            endif
            goto 100
      else
200         fxk=galpha
            do 50 i=1,n
                  xk(i)=xk(i)+alpha*dk(i)
50          continue
      endif
      return
      end
```