MATH693a
Dr. Peter Blomgren
HW05
BFGS

Matteo Polimeno

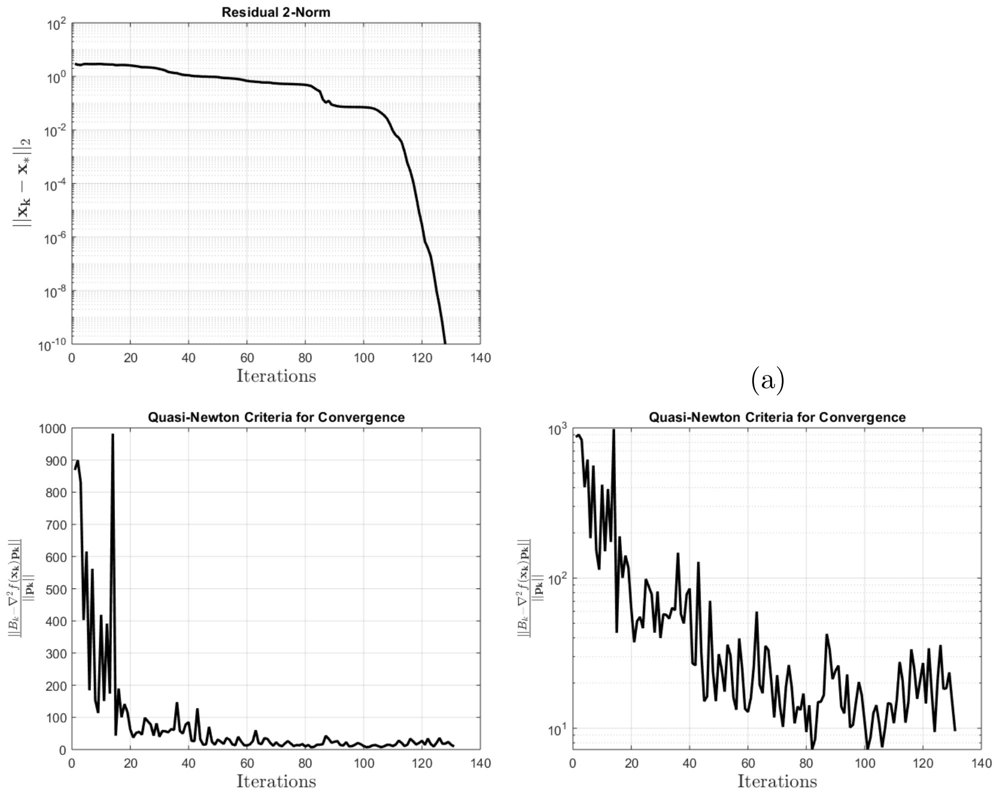
November 28^{th}, 2018

BFGS Algorithm - Implementation and Results

In this HW assignment we grab the file *rosenbrock2Nd.m* from the slides and use it to implement the BFGS algorithm to find the minimum of the function. We compare iteration count and elapsed time of the algorithm to the newton method.

Method	Iterations	Elapsed Time (sec)
Newton	27	0.0608
BFGS	131	0.00121

As we see from the above table, the BFGS algorithm is faster overall in term of elapsed time, but takes more than 100 more iterations compared to the Newton Method. For completeness in our results, we plot the Residual 2-norm for the BFGS and we check the convergence criteria for the algorithm.



b) c)

Figure 1: Euclidean Norm for the Residual a) and the Quasi-Newton Criteria for convergence on a linear scale b) and on a logarithmic scale c)

Appendix - Matlab Code

Provided Code

```
function [R] = rosenbrock_2Nd(x,order)

if( nargin<2 )
  order=0;
end

%
% Initial condition in R18
%
if( order == - 1 )
  xN      = [  1 ; 1 ];
  x0easy = [  1.2 ; 1.2 ];
  x0e2   = (xN + x0easy) / 2;
  x0e3   = (xN + x0e2)  / 2;
  x0e4   = (xN + x0e3) / 2;
  x0hard = [ -1.2 ; 1.0 ];
  x0h2   = (xN + x0hard) / 2;
  x0h3   = (xN + x0h2)  / 2;
  x0h4   = (xN + x0h3) / 2;
  x0h5   = 2*x0hard;
  R      = [x0easy ; x0e2 ; x0e3 ; x0e4 ; ...
            x0hard ; x0h2 ; x0h3 ; x0h4; x0h5];
  return
end

nx = length(x);

% 1D versions
%
% The function and derivatives needed to compute the
% gradient and the Hessian
%
rb2d      = @(x) ( 100*(x(2)-x(1).^2).^2+(1-x(1)).^2 )
    ;
rb2d_x    = @(x) ( -400*(x(2)-x(1).^2).*x(1)-2+2*x(1)
    );
rb2d_xx   = @(x) ( 1200*x(1).^2-400*x(2)+2 );
rb2d_xy   = @(x) ( -400*x(1) );
rb2d_y    = @(x) ( 200*x(2)-200*x(1).^2 );
rb2d_yy   = @(x) ( 200 );
```

```
rb2d_grad = @(x) ( [ rb2d_x(x) ; rb2d_y(x) ] );
rb2d_hess = @(x) ( [ rb2d_xx(x) rb2d_xy(x) ; rb2d_xy(x
    ) rb2d_yy(x)] );

switch order
 case 0
  R = zeros(1,1);
  for k = 1:2:nx
    R = R + rb2d(x(k:(k+1)));
  end
 case 1
  R = zeros(length(x),1);
  for k = 1:2:nx
    R(k:(k+1)) = rb2d_grad(x(k:(k+1)));
  end
 case 2
  R = zeros(length(x),length(x));
  for k = 1:2:nx
    R(k:(k+1),k:(k+1)) = rb2d_hess(x(k:(k+1)));
  end
 otherwise
  error(sprintf('\nCannot compute derivatives of order
      %d.\n',order));
end
```

BFGS Algorithm

```
%code for the BFGS algorithm
x0 = rosenbrock_2Nd([],-1);
xk = x0;
Hk = eye(size(rosenbrock_2Nd(xk,2)));
Bk = eye(size(rosenbrock_2Nd(xk,2)));
abar = 1;
al = 0;
ah = 1;
amax = 5;
c1 = 1e-4;
c2 = 0.9;
rho = .5;
tol = 1e-11;
xstar = ones(18,1);
QNconv = [];
rkvec = [];
kk = 0;
Tin = [];

tic
while norm(rosenbrock_2Nd(xk,1)) > tol
    pk = -Hk*rosenbrock_2Nd(xk,1);
    pt = pk';
    tic
    a = wolfe_strong2(xk,pk,al,ah,amax,c1,c2);
    tin = toc;
    Tin = [Tin;tin];
    xkp1 = xk+a*pk;
    sk = xkp1 - xk;
    yk = rosenbrock_2Nd(xkp1,1)-rosenbrock_2Nd(xk,1);
    yt = yk';
    st = sk';
    rhok = 1/((yt)*sk);
    Hk = (eye(size(Hk))-rhok*sk*yt)*Hk*(eye(size(Hk))-rhok*yk*st) + rhok*sk*st;
    Bk = Bk - (Bk*sk*st*Bk)/(st*Bk*sk) + (yk*yt)/(yt*sk);
    Qconv = norm((Bk-Hk)*pk)/norm(pk);
    QNconv = [QNconv;Qconv];
    xk = xkp1;
    kk = kk + 1;
    rk = norm(xk-xstar);
```

```
    rkvec = [rkvec;rk];
end
toc

gradk = rosenbrock_2Nd(xk,1);
%mu = eig(Hk);
figure(1)
kvec = 1:kk;
plot(kvec,QNconv,'k-','linewidth',2)
title('Quasi-Newton Criteria for Convergence')
xlabel('Iterations','interpreter','latex','fontsize',15)
ylabel('$\frac{||B_{k}-\nabla^{2}f({\bf x_{k}}){\bf p_{k}}||}{||{\bf p_{k}}||}$','interpreter','latex','fontsize',14)
grid on

% % Get fitted values
% coeffs = polyfit(kvec', QNconv, 1);
% fittedY = polyval(coeffs, kvec);

figure(2)
kvec = 1:kk;
semilogy(kvec,QNconv,'k-','linewidth',2)
% hold on
% % Plot the fitted line
% plot(kvec, fittedY, '--', 'LineWidth', 2);
% hold off
title('Quasi-Newton Criteria for Convergence')
xlabel('Iterations','interpreter','latex','fontsize',15)
ylabel('$\frac{||B_{k}-\nabla^{2}f({\bf x_{k}}){\bf p_{k}}||}{||{\bf p_{k}}||}$','interpreter','latex','fontsize',14)
grid on

figure(3)
kvec = 1:kk;
semilogy(kvec,rkvec,'k-','linewidth',2)
title('Residual 2-Norm')
xlabel('Iterations','interpreter','latex','fontsize',15)
ylabel('$||{\bf x_{k}} - {\bf x_{\ast}}||_{2}$','interpreter','latex','fontsize',15)
```

```
ylim([10^-10 10^2])
grid on
```

Linesearch

```
function anew = wolfe_strong2(x,pk,al,ah,amax,c1,c2) % strong wolfe conditions
    aii = [al,ah];
    ii = 1;
    while true
        if (rosenbrock_2Nd(x+aii(2)*pk,0) > rosenbrock_2Nd(x,0) + c1*aii(2)*rosenbrock_2Nd(x,0)) || ((rosenbrock_2Nd(x+aii(2)*pk,0) >= rosenbrock_2Nd(x+aii(1)*pk,0))) && ii > 1
            anew = zoom2(x,pk,aii(1),aii(2),c1,c2);
            break
        end
        if abs(phi_prime(x,pk,aii(2))) <= -c2*phi_prime(x,pk,0)
            anew = aii(2);
            break
        end
        if phi_prime(x,pk,aii(2)) >= 0
            anew = zoom2(x,pk,aii(2),aii(1),c1,c2);
            break
        end
        if (abs(aii(2)-aii(1)) < 1e-14) || (abs(aii(2)) < 1e-14)
            anew = aii(2);
            break
        end
        aii(1) = aii(2);
        aii(2) = (amax+aii(2))/2;
        ii = ii + 1;
    end
end
```

```
function akp1 = interp2(x,pk,al,ah)  %return result of the interpolation
    d1 = phi_prime(x,al) + phi_prime(x,ah) - 3*(rosenbrock_2Nd(x+al*pk,0)-rosenbrock_2Nd(x+ah*pk,0))/(al-ah);
    d2 = sign(ah-al)*(sqrt(d1.^2-phi_prime(x,al).*phi_prime(x,ah)));
    akp1 = ah - (ah-al)*((phi_prime(x,ah)+d2-d1)./(phi_prime(x,ah)-phi_prime(x,al)+2*d2));
end
```

```
function astar = zoom2(x,pk,al,ah,c1,c2)
    go = true;
    while go
        ajj = abs(ah-al)/2;
        if (rosenbrock_2Nd(x+ajj*pk,0) > rosenbrock_2Nd(x,0)+c1*ajj*phi_prime(x,pk,0)) || (rosenbrock_2Nd(x+ajj*pk,0) >= rosenbrock_2Nd(x+al*pk,0))
          ah = ajj;
        else
            if abs(phi_prime(x,pk,ajj)) <= -c2*phi_prime(x,pk,0)
                astar = ajj;
                go = false;
            end
            if (phi_prime(x,pk,ajj)*(ah-al)) >= 0
                ah = al;
            al = ajj;
            end
        end
    end
end

% function akp1 = interp(x,al,ah)  %return result of the interpolation
%     d1 = phi_prime(x,al) + phi_prime(x,ah) - 3*(rosenbrock_2Nd(x+al*dir_newt2(x),0)-rosenbrock_2Nd(x+ah*dir_newt2(x),0))/(al-ah);
%     d2 = sign(ah-al)*(sqrt(d1.^2-phi_prime(x,al).*phi_prime(x,ah)));
%     akp1 = ah - (ah-al)*((phi_prime(x,ah)+d2-d1)./(phi_prime(x,ah)-phi_prime(x,al)+2*d2));
% end
```

```
% function phip = phi_prime(x,alpha)
%       phip = dot(rosenbrock_2Nd(x+alpha*dir_newt2(x)
    ,1),(dir_newt2(x)));
% end

% function pk = dir_newt2(x)
%       pk = -(rosenbrock_2Nd(x,2)\rosenbrock_2Nd(x,1));
% end
```

```
function phip = phi_prime(x,pk,alpha)
    phip = dot(rosenbrock_2Nd(x+alpha*pk,1),pk);
end
```