Optimization Homework 3

Ryan Day

February 11, 2019

Contents

1 Written description of program

I implemented Steepest Descent and also the Quasi-Newton method with a BFGS update. I used recursion to call the line updates so that I wouldn't have to fiddle around with while or for loops.

The line search was implemented by first taking a step in the search direction as calculated by the chosen algorithm with $\alpha = 0.7$. If this step is too big, the program will start the line search over from the starting point and use $\alpha_{current}/100$. This will keep happening until the first step is smaller than the initial value. After this, the line search keeps searching by doubling alpha each step, always from the same starting point. Once the value of the alpha is bigger than the alpha of the previous value, it finds the value of the point at $\alpha_{current} * 0.75$, halfway between the last point and the second to last point. It then takes the minimum point and the two points around that one, does a quadratic fit, and finds the alpha of the minimum function value of that fit. It then takes a step and finds a new x0, and checks the gradient at this point. It repeats this process again until each component of the gradient is under the specified tolerance.

2 Did the methods perform as well as I expected?

For the quadratic equation, the Quasi-Newton algorithm converged with three line searches, which is exactly what we would predict with 3 variables. We know that since Quasi-Newton approximates the algorithm as a quadratic, if the function is a quadratic then it will converge very quickly. Steepest Descent converged in 21 line searches, which is fine. This wass not as good as Quasi-Newton, but the function was not elliptical enough to cause the algorithm to take a while to converge.

For the Rosenbrock function, Steepest Descent oscillated back in forth as seen in the figure on page 5. This is predictable because it always tries to go to the minimum but it overshoots it. The Quasi-Newton method had some erratic behavior but converged pretty quickly.

3 Test Results Quadratic function

3.1 Results for Quasi-Newton Method

	Starting Point	Function Value	Search Direction	Step Length	Nobj for search
1	10 10 10	520	-0.99426 0.06414 -0.08552	7.8185	7
2	2.226 10.502 9.3313	154.343	-0.1624 -0.5596 -0.8362	18.3067	9
3	-0.7467 0.2560 -5.977	-14.399	0.05846 0.66920 -0.78849	3.270	6
4	-0.5556 2.4444 -8.5556	-22.3889			1

TOTAL OBJ CALLS = 23
TOTAL GRAD CALLS = 4

3.2 First five steps for Steepest Descent

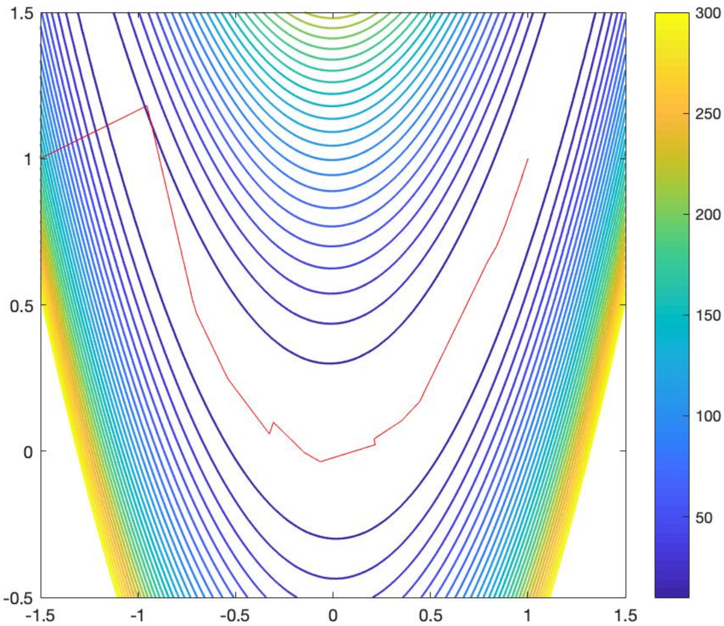
	Starting Point	Function Value	Search Direction	Step Length	Nobj for search
1	10 10 10	520	-0.99426 0.06414 -0.08552	7.8185	7
2	2.226 10.502 9.3313	154.3	0.0335 -0.5723 -0.8194	12.5261	8
3	2.6466 3.3328 -0.9320	38.88	-0.9765 0.1557 -0.1487	2.5121	6
4	0.1934 3.7239 -1.3057	1.5472	0.1236 -0.1598 -0.9793	4.3808	7
5	0.7352 3.0236 -5.5961	-12.99	-0.9819 0.1229 -0.1441	0.9798	4

TOTAL OBJ CALLS = 166
TOTAL GRAD CALLS = 24

4 Test Results Rosenbrocks function

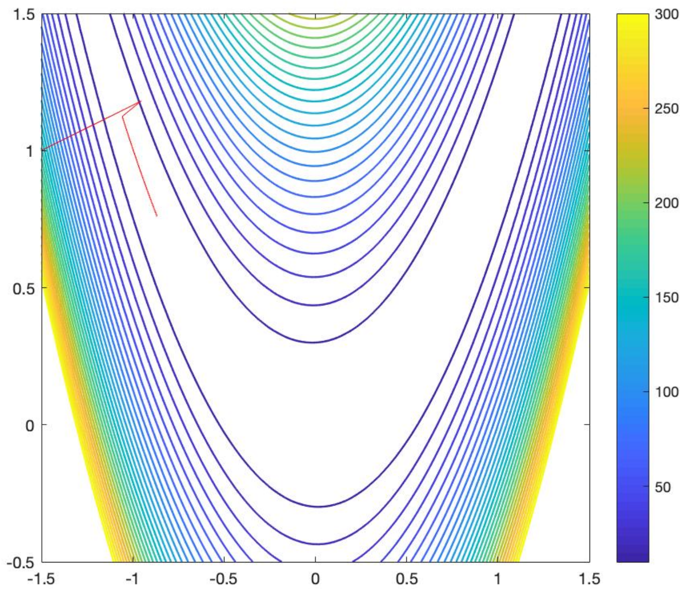
4.1 Results for Quasi-Newton Method

TOTAL OBJ CALLS = 162, TOTAL GRAD CALLS = 22

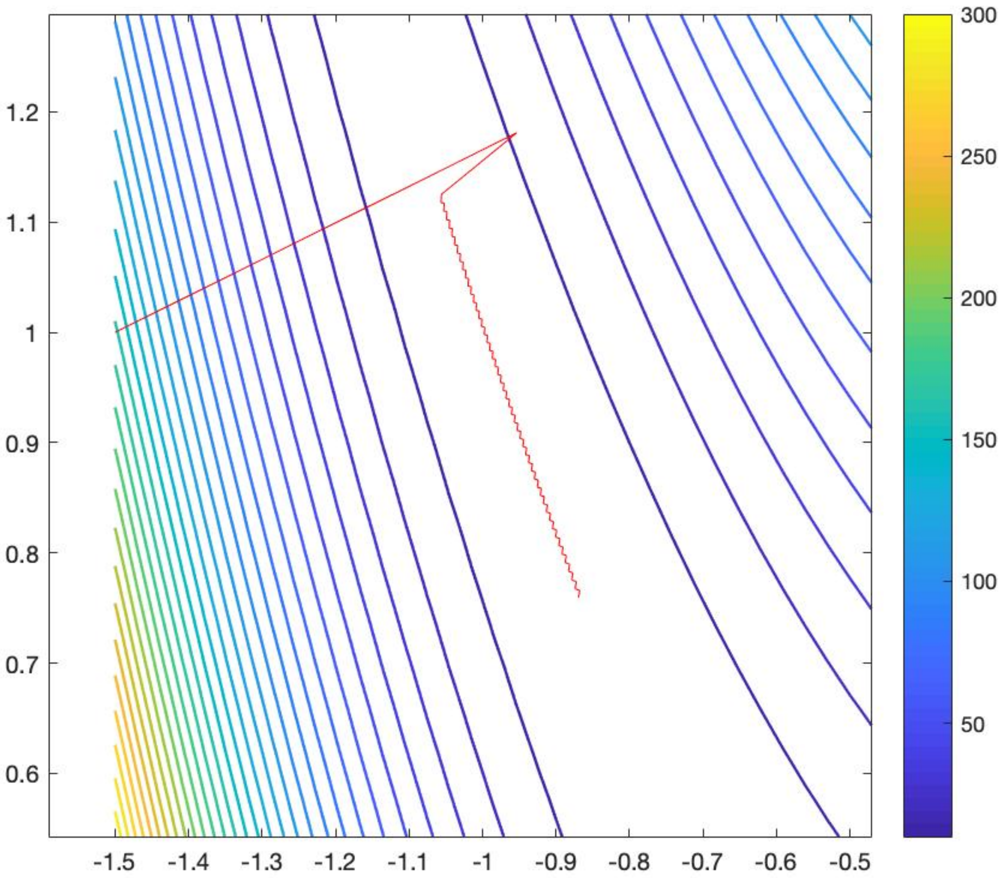


4.2 First 100 Steps for Steepest Descent

TOTAL OBJ CALLS TO CONVERGE = 3931, TOTAL GRAD CALLS TO CONVERGE = 559



Although its hard to see here, the algorithm is taking lots of tiny steps around the line. In the zoomed up view you can see a very small zig zag pattern.



5 Code

5.1 fminun

```
function [xopt, fopt, exitflag] = fminun(obj, gradobj, x0, stoptol,
    algoflag)
    %--- Set constants ---%
    [n,~] = size(x0);
    stoptol_vector = stoptol*ones(1,n);
    alphaInitial = 0.7;
    firstflag = 1;
    %--- Set globals ---%
    global f_a_nobj_history
    global x_history
    global s_history
    %--- Set initial variables ---%
    allx = [];
    allf = [];
    alla = [];
    alls = [];
    nobjlocal = [];
```

```
    N = eye(n);
    %--- Update Direction ---%
    [xopt, fopt, exitflag] = updateDirection(obj, gradobj, x0,...
    stoptol_vector, algoflag, alphaInitial,...
    N, x0, 0, firstflag, allx, allf, alla, alls, nobjlocal);
%    Plot Rosenbrock contour plot
%      if algoflag == 1
%          ContourPlotRosenbrock(x_history(:,1:100))
%      else
%          % Show all steps
%          ContourPlotRosenbrock(x_history)
%      end
end

function [xopt,fopt,exitflag] = updateDirection(obj, gradobj, x0,...
    stoptol_vector, algoflag, alphaInitial, N, xprev, gradprev, firstflag,...
    allx, allf, alla, alls, nobjlocal)
    %--- Initialize variables ---%
    global nobj
    nobj_at_start = nobj;
    f = obj(x0);
    alpha = alphaInitial;
    grad = gradobj(x0);
    %--- Check if gradient meets stoptolerance ---%
    if abs(grad) < stoptol_vector
        xopt = x0;
        fopt = f;
        exitflag = 0;
        global f_a_nobj_history
        global x_history
        global s_history
        f_a_nobj_history = [allf;alla;nobjlocal];
        x_history = allx;
        s_history = alls;
        return
    else
    %--- Check if nobj has been called too many times ---%
%      if nobj > 1000
%          exitflag = 1;
%          xopt = 'algorithm called obj more than 1000 times';
%          fopt = 'algorithm called obj more than 1000 times';
%          return
%      end
    %--- Update search direction --- %
    if algoflag == 1
        s = getSdDirection(grad);
    elseif algoflag == 2
        % First iteration of quasi-newton method runs as steepest descent
        if firstflag == 1
        s = getSdDirection(grad);
        firstflag = 0;
        else
        [s,N] = getQNdirection(grad,N,grad-gradprev,x0-xprev);
        end
    else
```

```
        error('algorithm not defined')
    end
    %--- Do a line search in new direction ---%
    [a4,fn4] = searchLine(x0,f,s,obj,alpha);
    [a3,fn3] = takeBestThreePoints(a4,fn4);
    alpha = quadraticFit(a3,fn3);
    %--- Record variable history ---%
    allx = [allx x0];
    allf = [allf f];
    alls = [alls s];
    alla = [alla alpha];
    nobjlocal = [nobjlocal (nobj-nobj_at_start)];
    %--- Set new variables for new search direction ---%
    xnew = x0 + alpha*s;
    [xopt, fopt, exitflag] = updateDirection(obj, gradobj, xnew, stoptol_vector, algoflag, alphaInitial, N,x0,grad,firstflag,allx,allf,alla,alls,nobjlocal);
    end
end

function [a,fn] = searchLine(x0,f,s,obj,alpha)
    xnew = x0 + alpha*s;
    fnew = obj(xnew);
    if fnew < f
        [a,fn] = searchLineHelper(x0,fnew,s,obj,alpha*2,[0, alpha],[f,fnew]);
    else
        % If the step was too far, try again with a smaller alpha
        [a,fn] = searchLine(x0,f,s,obj,alpha/100);
        return
    end
    % Go one step back and return those four points
    amiddle = a(end)*0.75;
    fmiddle = obj(x0+amiddle*s);
    a = [a((end-2):end-1),amiddle,a(end)];
    fn = [fn((end-2):end-1),fmiddle,fn(end)];
end

function [a,fn] = searchLineHelper(x0,f,s,obj,alpha,ahistory,fhistory)
    xnew = x0 + alpha*s;
    fnew = obj(xnew);
    if fnew < f
        [a,fn] = searchLineHelper(x0,fnew,s,obj,alpha*2,[ahistory alpha],[fhistory,fnew]);
    else
        a = [ahistory alpha];
        fn = [fhistory fnew];
    end
end

function [astar] = quadraticFit(a,fn)
    % Calculate Alpha of minimum of the quadratic approximation
    num = fn(1)*(a(2)^2-a(3)^2)+fn(2)*(a(3)^2-a(1)^2)+fn(3)*(a(1)^2-a(2)^2);
    den = 2*(fn(1)*(a(2)-a(3))+fn(2)*(a(3)-a(1))+fn(3)*(a(1)-a(2)));
    astar = num/den;
```

```
end

function [s,N] = getQNdirection(grad,N,gamma,delta_x)
    % BFGS update is used
    tgamma = transpose(gamma);
    tdelta_x = transpose(delta_x);
    first = 1 + (tgamma*N*gamma)/(tdelta_x*gamma);
    second = (delta_x*tdelta_x)/(tdelta_x*gamma);
    third = (delta_x*tgamma*N+N*gamma*tdelta_x)/(tdelta_x*gamma);
    N = N + first*second - third;

    mag = sqrt(grad'*grad);
    s = -N*(grad/mag);
end

function [a,fn] = takeBestThreePoints(a,fn)
    [~,min_index] = min(fn);
    a = a([min_index-1 min_index min_index+1]);
    fn = fn([min_index-1 min_index min_index+1]);
end

 % get steepest descent search direction as a column vector
function [s] = getSdDirection(grad)
    mag = sqrt(grad'*grad);
    s = -grad/mag;
end
```

5.2 fminunDriver

```
%---Example Driver program for fminun---%
%---Author: Ryan Day---%
    clear;

    global nobj ngrad
    nobj = 0; % counter for objective evaluations
    ngrad = 0.; % counter for gradient evaluations

    algoflag = 2; % 1=steepest descent; 2=BFGS quasi-Newton
    problem = 2;
    stoptol = 1.e-3; % stopping tolerance, all gradient elements must be < stoptol

    x0_1_starting_points = [[10;10;10] [2;7;9] [100;100;100] [15;9;30]];
    x0_1 = x0_1_starting_points(:,1); % starting points, set to be column vector
    x0_2_starting_points = [[-1.5; 1] [-3; 1.8]];
    x0_2 = x0_2_starting_points(:,1);

    if problem == 1
        % quadratic function
        x0 = x0_1;
        obj = @obj1;
        gradobj = @gradobj1;
    elseif problem == 2
        % rosenbrock function
        x0 = x0_2;
```

```
        obj = @obj2;
        gradobj = @gradobj2;
    end
    % ----------- call fminun------------------
    [xopt, fopt, exitflag] = fminun(obj, gradobj, x0, stoptol, algoflag);

    xopt
    fopt
    nobj
    ngrad

     % Quadratic function to be minimized
     function [f] = obj1(x)
        global nobj
        f = 20+3*x(1)-6*x(2)+8*x(3)+6*x(1)^2-(2*x(1)*x(2))-(x(1)*x(3))+x(2)^2+0.5*x(3)^2;
        nobj = nobj +1;
     end

    % get gradient as a column vector
     function [grad] = gradobj1(x)
        global ngrad
        %gradient for function 1
        grad(1,1) = 12*x(1) - 2*x(2) - x(3) + 3;
        grad(2,1) = 2*x(2) - 2*x(1) - 6;
        grad(3,1) = x(3) - x(1) + 8;
        ngrad = ngrad + 1;
     end

     % Rosenbrock function to be minimized
     function [f] = obj2(x)
        global nobj
        f = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
        nobj = nobj +1;
     end

     % get gradient as a column vector
     function [grad] = gradobj2(x)
        global ngrad
        %gradient for function 1
        grad(1,1) = 2*x(1) - 400*x(1)*(- x(1)^2 + x(2)) - 2;
        grad(2,1) = - 200*x(1)^2 + 200*x(2);
        ngrad = ngrad + 1;
     end
```

5.3 Rosenbrock Contour plot

```
function [] = ContourPlotRosenbrock(coordinates)
    clf

    %Rosenbrock code from https://www.mathworks.com/matlabcentral/mlc-
        downloads/downloads/submissions/23972/versions/22/previews/chebfun/
        examples/opt/html/Rosenbrock.html
    f = @(x,y) (1-x).^2 + 100*(y-x.^2).^2;
    x = linspace(-1.5,1.5); y = linspace(-1,3);
    [xx,yy] = meshgrid(x,y); ff = f(xx,yy);
    levels = 10:10:300;
    LW = 'linewidth'; FS = 'fontsize'; MS = 'markersize';
    figure, contour(x,y,ff,levels,LW,1.2), colorbar
    axis([-1.5 1.5 -0.5 1.5]), axis square, hold on
    hold on;
    coordinates
    plot(coordinates(1,:),coordinates(2,:),'r');
end
```