Unconstrained Optimization

Landon Wright

January 30, 2018

1 Program Description

2 Testing Results

3 Matlab Code

3.0.1 Fminun Routine

```
function [xopt, fopt, exitflag] = fminun(obj, gradobj, x0, stoptol, algoflag)

   % get function and gradient at starting point
   [n,~] = size(x0); % get number of variables
   f = obj(x0);
   grad = gradobj(x0);
   x = x0;

   %set starting step length
   alpha = 0.5;

   if (algoflag == 1)     % steepest descent
      s = srchsd(grad)
   end

   % take a step
   xnew = x + alpha*s;
   fnew = obj(xnew);

   xopt = xnew;
   fopt = fnew;
   exitflag = 0;
end

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

3.0.2 Driver

```
function [] = fminunDriv()
%----------------Example Driver program for fminun-----------------
    clear;
```

```
4
5       global nobj ngrad
6       nobj = 0; % counter for objective evaluations
7       ngrad = 0.; % counter for gradient evaluations
8       x0 = [1.; 1.]; % starting point, set to be column vector
9       algoflag = 1; % 1=steepest descent; 2=conjugate gradient; 3=BFGS quasi-Newton
10      stoptol = 1.e-3; % stopping tolerance, all gradient elements must be < stoptol
11
12
13      % ---------- call fminun----------------
14      [xopt, fopt, exitflag] = fminun(@obj, @gradobj, x0, stoptol, algoflag);
15
16      xopt
17      fopt
18
19      nobj
20      ngrad
21  end
22
23   % function to be minimized
24   function [f] = obj(x)
25      global nobj
26      %example function
27      f = 12 + 6*x(1) - 5*x(2) + 4*x(1)^2 -2*x(1)*x(2) + 6*x(2)^2;
28      nobj = nobj +1;
29   end
30
31  % get gradient as a column vector
32   function [grad] = gradobj(x)
33      global ngrad
34      %gradient for function above
35      grad(1,1) = 6 + 8*x(1) - 2*x(2);
36      grad(2,1) = -5 - 2*x(1) + 12*x(2);
37      ngrad = ngrad + 1;
38   end
39
40
```

ME 575
Homework #3 Unconstrained Optimization
Due Feb 7 at 2:50 p.m.

Description

Write an optimization routine in MATLAB (required) that performs unconstrained optimization. Your routine should include the ability to optimize using the methods,

- steepest descent
- conjugate gradient
- BFGS quasi-Newton

Your program should be able to work on both quadratic and non-quadratic functions of *n* variables. To determine how far to step, you may use a line search method (such as the quadratic fit given in the notes), a trust region method, or a combination of these. You will be evaluated both on how theoretically sound your program is and how well it performs.

You should use good programming style, such as selecting appropriate variable names, making the program somewhat modular (employing function routines appropriately) and documenting your code.

You will provide test results for your program on the two functions given here. In addition, you will turn in your function so we can test it on other functions or on different starting points.

Testing

1) Test your program on the following quadratic function of three variables:

$$f = 20 + 3x_1 - 6x_2 + 8x_3 + 6x_1^2 - 2x_1x_2 - x_1x_3 + x_2^2 + 0.5x_3^2$$

The conjugate gradient and quasi-Newton methods should be able to solve this problem in three iterations. Show your data for each iteration (starting point, function value, search direction, step length, number of evaluations of objective) for these two methods starting from the point $\mathbf{x}^T = [10, 10, 10]$. In addition, give data for five steps of steepest descent. At the optimum the absolute value of all elements of the gradient vector should be below 1.e-5. Indicate the total number of objective evaluations and gradient evaluations taken by each method.

2) Test your program on the following non-quadratic function (Rosenbrock's function) of two variables:

$$f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Starting from the point $\mathbf{x}^T = [-1.5, 1]$, show the steps of the methods on a contour plot. Show 20 steps of steepest descent. Continue with the conjugate gradient and quasi-Newton methods until the optimum is reached at $\mathbf{x}^T = [1, 1]$ and the absolute value of all elements of

the gradient vector is below 1.e-3. Indicate the total number of objective evaluations and gradient evaluations taken by each method.

MATLAB Stuff

Your function will be called `fminun`. It will receive and pass back the following arguments:

```
[xopt, fopt, exitflag] = fminun(@obj, @gradobj, x0, stoptol, algoflag);

Inputs:
@obj = function handle for the objective we are minimizing (we will use
obj) The calling statement for obj looks like,

f = obj(x);

@gradobj = function handle for the function that evaluates gradients of
the objective (we will use gradobj, see example) The calling statement for
gradobj looks like,

grad = gradobj(x);

Note that grad is passed back as a column vector.

x0 = starting point (column vector)

stoptol = stopping tolerance. I will set this to 1.e-3, unless this proves
too restrictive. The absolute value of all elements of your gradient
vector should be less than this value at the optimum.

algoflag = 1 for steepest descent, =2 for conjugate gradient, =3 for
quasi-Newton.

Outputs:
xopt = optimal value of x (column vector)

fopt = optimal value of the objective

exitflag = 0 if algorithm terminated successfully; otherwise =1. Your
algorithm should exit (=1) if it has exceeded more than 500 evaluations of
the objective.
```

Attached is an example "driver" routine and example `fminun` routine to get you started. You can copy these from Learning Suite > Content > MATLAB Examples.

Grading

Grading: Your routine will be graded based on two criteria: 1) Soundness of your methodology and implementation as evidenced by your write-up and code (50%), and performance on test functions (50%). The performance score will be based 70% on accuracy (identifying the optimum) and 30% on efficiency (number of objective evaluations plus *n**number of gradient evaluations).

Turn in, in one report through Learning Suite:

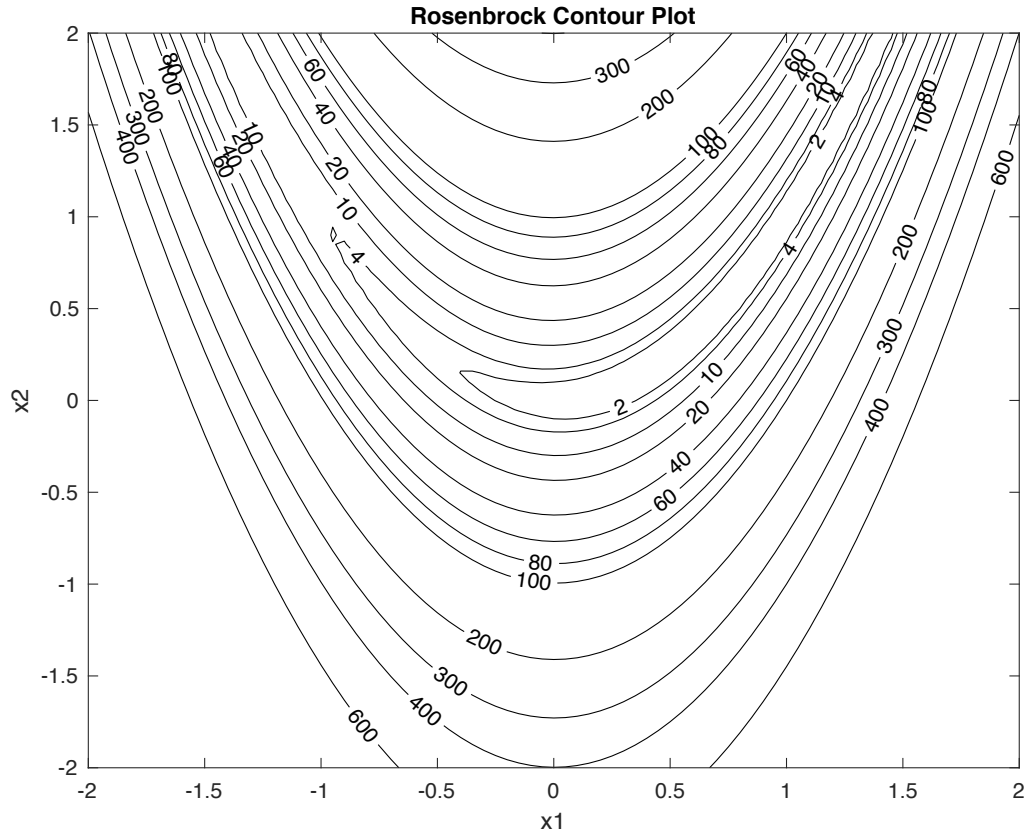
1. A brief written description of your program, including discussion of each of the three methods. This should not be longer than a page (single-spaced). You may include equations if you wish. Discuss how you implemented the methods.
2. The requested results from testing on the two functions.

3. Hardcopy of your MATLAB code.

In addition, email your MATLAB code to Jacob Greenwood (jacobgwood@gmail.com) . Additional information about this will be provided.

Suggestions:

Get started now! Start with a relatively simple routine and work forward, always having a working program. I would suggest you start off with a relatively straight-forward step length approach (such as the quadratic fit given in class) and then you can get more fancy after you have a program working for all three methods, if you want to. A working, straightforward program is much better than a non-working, fancy program.



Rosenbrock's function. The optimum is at $(\mathbf{x}^*)^T = [1, 1]$ where f *= 0. Start at the point, $(\mathbf{x}^0)^T = [-1.5, 1]$

Driver Routine:

```
%----------------Example Driver program for fminun-----------------
    clear;

    global nobj ngrad
    nobj = 0; % counter for objective evaluations
    ngrad = 0.; % counter for gradient evaluations
    x0 = [1.; 1.]; % starting point, set to be column vector
    algoflag = 1; % 1=steepest descent; 2=conjugate gradient; 3=BFGS quasi-Newton
    stoptol = 1.e-3; % stopping tolerance, all gradient elements must be < stoptol


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

    xopt
    fopt

     nobj
    ngrad

     % function to be minimized
     function [f] = obj(x)
        global nobj
        %example function
        f = 12 + 6*x(1) - 5*x(2) + 4*x(1)^2 -2*x(1)*x(2) + 6*x(2)^2;
        nobj = nobj +  1;
     end

    % get gradient as a column vector
     function [grad] = gradobj(x)
        global ngrad
        %gradient for function above
        grad(1,1) = 6 + 8*x(1) - 2*x(2);
        grad(2,1) = -5 - 2*x(1) + 12*x(2);
        ngrad = ngrad + 1;
     end
```

Example fminun function:

```
     function [xopt, fopt, exitflag] = fminun(obj, gradobj, x0, stoptol, algoflag)

        % get function and gradient at starting point
        [n,~] = size(x0); % get number of variables
        f = obj(x0);
        grad = gradobj(x0);
        x = x0;

        %set starting step length
        alpha = 0.5;

        if (algoflag == 1)     % steepest descent
           s = srchsd(grad)
        end

        % take a step
        xnew = x + alpha*s;
        fnew = obj(xnew);

        xopt = xnew;
        fopt = fnew;
        exitflag = 0;
     end

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