Unconstrained Optimization

Landon Wright

February 3, 2018

1 Program Description

2 Testing Results

2.1 Function 1 Results

Table 1: Steepest descent progression

	Start-value	Value	Step-direction	Step-len	Function-calls
1	(10.000000, 10.000000, 10.000000)	154.343825	(-0.994269, 0.064146, -0.085528)	7.818505	7.000000
2	(2.226306, 10.501529, 9.331295)	38.883122	(0.033560, -0.572300, -0.819357)	12.526136	13.000000
3	(2.646680, 3.332821, -0.932087)	1.547244	(-0.976543, 0.155690, -0.148744)	2.512183	17.000000
4	(0.193426, 3.723944, -1.305758)	-12.992294	(0.123689, -0.159847, -0.979362)	4.380826	22.000000
5	(0.735284, 3.023680, -5.596172)	-18.666244	(-0.981909, 0.122884, -0.144067)	0.979809	29.000000
6	(-0.226799, 3.144083, -5.737330)	-20.913770	(0.104303, -0.283997, -0.953135)	1.721006	33.000000
7	(-0.047292, 2.655322, -7.377682)	-21.804129	(-0.980760, 0.129642, -0.145955)	0.388142	39.000000
8	(-0.427967, 2.705642, -7.434333)	-22.157079	(0.108735, -0.258162, -0.959963)	0.681980	46.000000
9	(-0.353812, 2.529581, -8.089008)	-22.296993	(-0.980997, 0.128234, -0.145603)	0.153865	51.000000
10	(-0.504753, 2.549312, -8.111412)	-22.352459	(0.107833, -0.263520, -0.958608)	0.270350	56.000000
11	(-0.475600, 2.478069, -8.370572)	-22.374447	(-0.980948, 0.128524, -0.145677)	0.060996	63.000000
12	(-0.535434, 2.485908, -8.379457)	-22.383164	(0.108020, -0.262414, -0.958890)	0.107174	68.000000
13	(-0.523857, 2.457784, -8.482226)	-22.386619	(-0.980958, 0.128464, -0.145662)	0.024181	74.000000
14	(-0.547577, 2.460891, -8.485748)	-22.387989	(0.107981, -0.262642, -0.958832)	0.042487	81.000000
15	(-0.542990, 2.449732, -8.526485)	-22.388532	(-0.980956, 0.128477, -0.145665)	0.009586	90.000000
16	(-0.552393, 2.450963, -8.527882)	-22.388747	(0.107989, -0.262595, -0.958844)	0.016843	96.000000
17	(-0.550574, 2.446541, -8.544031)	-22.388833	(-0.980957, 0.128474, -0.145665)	0.003800	104.000000
18	(-0.554302, 2.447029, -8.544585)	-22.388867	(0.107988, -0.262605, -0.958842)	0.006677	113.000000
19	(-0.553581, 2.445275, -8.550987)	-22.388880	(-0.980957, 0.128475, -0.145665)	0.001506	120.000000
20	(-0.555059, 2.445469, -8.551206)	-22.388885	(0.107988, -0.262603, -0.958842)	0.002647	127.000000
21	(-0.554773, 2.444774, -8.553744)	-22.388888	(-0.980957, 0.128475, -0.145665)	0.000597	136.000000
22	(-0.555359, 2.444851, -8.553831)	-22.388888	(0.107988, -0.262603, -0.958842)	0.001049	146.000000
23	(-0.555245, 2.444575, -8.554838)	-22.388889	(-0.980957, 0.128475, -0.145665)	0.000237	154.000000
24	(-0.555477, 2.444605, -8.554872)	-22.388889	(0.107988, -0.262603, -0.958842)	0.000416	163.000000
25	(-0.555433, 2.444496, -8.555271)	-22.388889	(-0.980957, 0.128475, -0.145664)	0.000094	174.000000
26	(-0.555525, 2.444508, -8.555285)	-22.388889	(0.107989, -0.262603, -0.958842)	0.000165	182.000000
27	(-0.555507, 2.444465, -8.555443)	-22.388889	(-0.980957, 0.128474, -0.145666)	0.000037	192.000000
28	(-0.555543, 2.444470, -8.555448)	-22.388889	(0.107983, -0.262603, -0.958843)	0.000065	203.000000
29	(-0.555536, 2.444453, -8.555511)	-22.388889	(-0.980957, 0.128476, -0.145662)	0.000015	212.000000
30	(-0.555551, 2.444454, -8.555513)	-22.388889	(0.107974, -0.262601, -0.958844)	0.000026	221.000000
31	(-0.555548, 2.444448, -8.555538)	-22.388889	(-0.980959, 0.128480, -0.145645)	0.000006	232.000000
32	(-0.555554, 2.444448, -8.555539)	-22.388889	(0.108111, -0.262619, -0.958824)	0.000010	244.000000
33	(-0.555553, 2.444446, -8.555549)	-22.388889	(-0.980944, 0.128443, -0.145780)	0.000002	254.000000

Table 2: Conjugate gradient progression

	Start-value	Value	Step-direction	Step-len	Function-calls
1	(10.000000, 10.000000, 10.000000)	154.343825	(-0.994269, 0.064146, -0.085528)	7.818505	7.000000
2	(2.226306, 10.501529, 9.331295)	-14.399913	(-0.159337, -0.549094, -0.820431)	18.658898	14.000000
3	(-0.746744, 0.256038, -5.977048)	-22.388889	(0.056441, 0.646046, -0.761209)	3.387385	19.000000

Table 3: Quasi-Newton progression

	Start-value	Value	Step-direction	Step-len	Function-calls
1	(10.000000, 10.000000, 10.000000)	154.343825	(-0.994269, 0.064146, -0.085528)	7.818505	7.000000
2	(2.226306, 10.501529, 9.331295)	-14.399913	(-0.159337, -0.549094, -0.820431)	18.658898	14.000000
3	(-0.746744, 0.256038, -5.977048)	-22.388889	(0.056441, 0.646046, -0.761209)	3.387385	19.000000

Table 4: Respective number of objective and gradient evaluations required to obtain minimum with tolerance of $1e^{-5}$ on the gradient

Method	Objective Evaluations	Gradient Evaluations
Steepest descent	254	34
Conjugate Gradient	19	4
Quasi-Newton	19	4

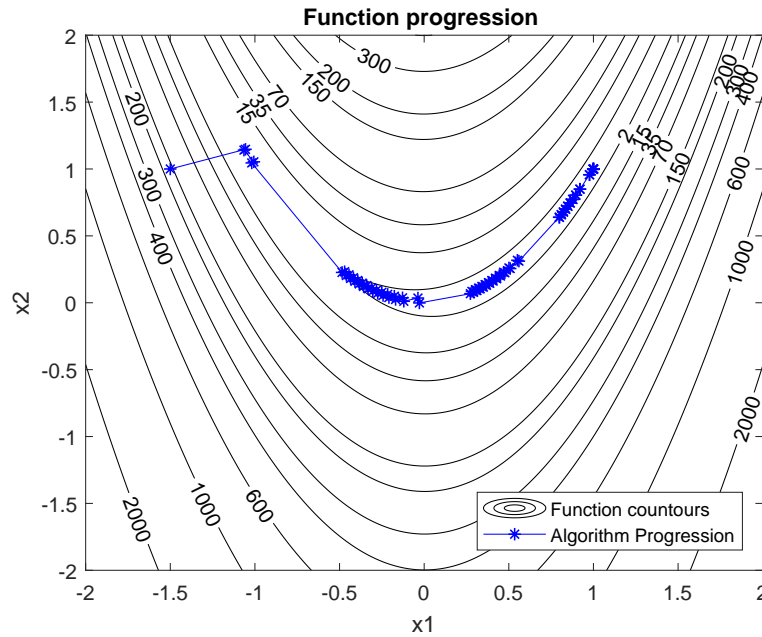## 2.2   Rosenbrock Function Results

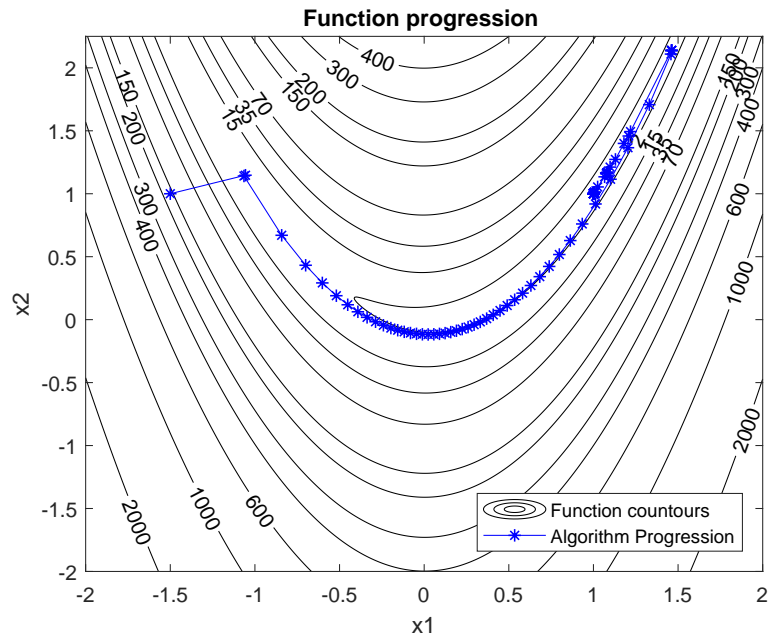Figure 1: Progression of steepest descent algorithm

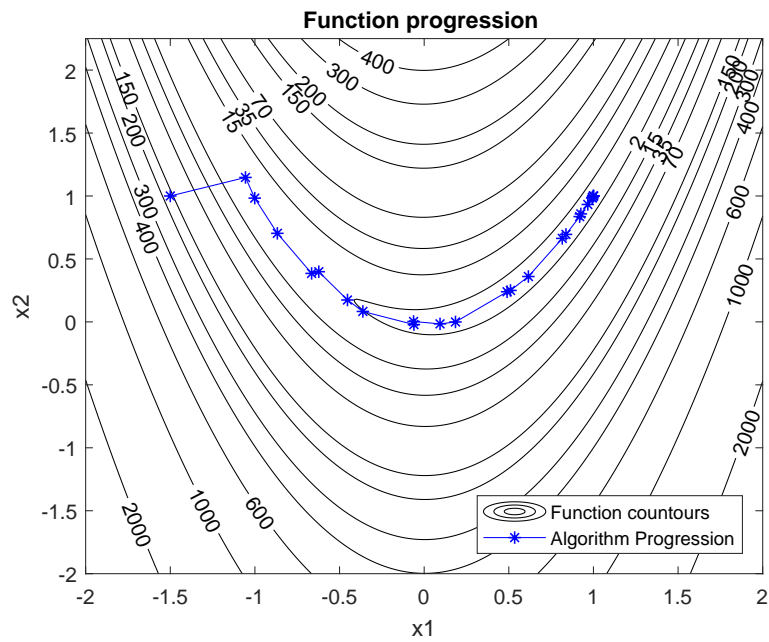Figure 2: Progression of conjugate gradient algorithm

Figure 3: Progression of quasi-Newton algorithm

Table 5: Respective number of objective and gradient evaluations required to obtain minimum with tolerance of $1e^{-3}$ on the gradient

Method	Objective Evaluations	Gradient Evaluations
Steepest descent	931	122
Conjugate Gradient	540	77
Quasi-Newton	172	28

3 Matlab Code

3.1 Fminun Routine

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

    % get function and gradient at starting point
    global nobj;
    [n,~] = size(x0); % get number of variables
    f = obj(x0); % get the value of the function at x0
    grad = gradobj(x0);
    x = x0;
    fOld = inf;
    xOld = zeros(n, 1);
    %set starting step length
    alpha = 0.0005;
    incrementCounter = 0;
    gradOld = ones(n,1);
    sOld = zeros(n,1);
    N = eye(n);
    saveMat = table;


    while (any(abs(grad(:)) > stoptol))
      incrementCounter = incrementCounter + 1
      if (nobj > 500)
        xopt = nan;
        fopt = nan;
        exitflag = 1
        return
      end

      if (algoflag == 1)      % steepest descent
        s = srchsd(grad);
        % find the proper alpha level
        % function [alphaPrime] = aPrime(obj, gradobj, s, f, x)
        alphaPrime = aPrime(obj, gradobj, s, f, x);

      end

      if (algoflag == 2)
        % use conjugate gradient method
        % check that it's not the first round / we wont divide by 0
        if (gradOld' * gradOld == 0)
          sMessy = -grad;
        else
```

```matlab
43          % calculate the beta term
44          betaCorrection = (grad' * grad) / (gradOld' * gradOld);
45          sMessy = -grad + betaCorrection * sOld;
46          sOld = sMessy;
47          % normalize the s vector
48          s = sMessy / norm(sMessy);
49          alphaPrime = aPrime(obj, gradobj, s, f, x);
50        end
51
52      end
53
54      if (algoflag == 3)
55        % use quasi-Newton method
56        gammaK = grad - gradOld;
57        deltaX = x - xOld;
58        if (incrementCounter > 1 & deltaX' * gammaK > 0)
59          t1 = 1 + ((gammaK' * N * gammaK) / (deltaX' * gammaK));
60          t2 = (deltaX * deltaX') / (deltaX' * gammaK);
61          t3 = (deltaX * gammaK' * N + N * gammaK * deltaX') / (deltaX' * gammaK);
62          N = N + t1 * t2 - t3
63        end
64        s = -N * grad;
65        s = s / norm(s);
66        alphaPrime = aPrime(obj, gradobj, s, f, x);
67
68      end
69
70
71      % take a step
72      xnew = x + alphaPrime*s;
73      fnew = obj(xnew);
74      gradOld = grad;
75      grad = gradobj(xnew);
76      fOld = f;
77      xOld = x;
78      f = fnew;
79      x = xnew;
80      newRow = {xOld', fnew, s', alphaPrime, nobj};
81      saveMat = [saveMat; newRow];
82
83
84    end
85    grad
86    xopt = xnew;
87    fopt = fnew;
88    exitflag = 0;
89    % saveMat.Properties.VariableNames = {'Starting_Point', 'Function_Value', ...
90    %      'Search_Direction', 'Step_Length', 'Number_of_Objective_Evaluations'};
91
92    toSave = table2array(saveMat);
93    % fout = fopen(sprintf('output%d.csv', algoflag),'w');
94    % fprintf(fout, '%s, %s, %s, %s, %s, %s, %s, %s, %s\r\n'...
95    % , , 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');
96    % fprintf(fout, '%8.6f, %8.6f, %8.6f, %8.6f, %8.6f, %8.6f, %8.6f, %8.6f, %8.6f   \r\n'...
```

```
97    % , toSave');
98    % fclose(fout);
99
100   end
101
102   % get steepest descent search direction as a column vector
103   function [s] = srchsd(grad)
104     mag = sqrt(grad'*grad);
105     s = -grad/mag;
106   end
```

3.2   Alpha* line search

```
1    function [alphaPrime] = aPrime(obj, gradobj, s, f, x)
2      minVal = f;
3      lastStepVal = f;
4      alphas = [0, f];
5      testStep = 2.1;
6      xTest = x;
7      incrementer = 2;
8      iterTestStep = testStep;
9      while (minVal >= lastStepVal)
10       % calculate at guessed testStep
11       xTest = x + iterTestStep * s;
12       fTest = obj(xTest);
13       if(fTest >= f & incrementer < 3)
14         % disp("Step size to big, recalculating");
15         minVal = f;
16         lastStepVal = f;
17         iterTestStep = iterTestStep / 10;
18         alphas = [0, f];
19         incrementer = 2;
20         continue;
21       end
22       % add it to the stored list
23       alphas(incrementer,1) = iterTestStep;
24       alphas(incrementer,2) = fTest;
25       % increment things for the next loop
26       minVal = min(minVal, fTest);
27       lastStepVal = fTest;
28       incrementer = incrementer + 1;
29       alphaOpt = iterTestStep;
30       iterTestStep = iterTestStep * 2;
31     end
32     % take the half step between the last two steps
33     iterTestStep = (alphas(end, 1) + alphas(end-1, 1)) / 2;
34     xTest = x + iterTestStep * s;
35     fTest = obj(xTest);
36     % store the values of the intermediate step
37     alphas(end + 1, :) = alphas(end, :);
38     alphas(end - 1, :) = [iterTestStep, fTest];
39     % find the index of the minimum function value
40     [minVal, minIdx] = min(alphas(:,2));
41     % get the three alpha and function values
```

```matlab
42      alpha2 = alphas(minIdx, 1);
43      f2 = alphas(minIdx, 2);
44      alpha1 = alphas(minIdx - 1, 1);
45      f1 = alphas(minIdx - 1, 2);
46      alpha3 = alphas(minIdx + 1, 1);
47      f3 = alphas(minIdx + 1, 2);
48      [alpha1, alpha2, alpha3];
49      % calculate the optimum alpha value
50      deltaAlpha = alpha2 - alpha1;
51      alphaPrime = (f1 * (alpha2^2 - alpha3^2) + f2 * (alpha3^2 - alpha1^2) ...
52      + f3 * (alpha1^2 - alpha2^2)) / (2 * (f1 * ...
53      (alpha2 - alpha3) + f2 * (alpha3 - alpha1) + ...
54      f3 * (alpha1 - alpha2)));
55  end
```

3.3 Driver

```matlab
1   function [] = fminunDriv()
2      %---------------Example Driver program for fminun------------------
3      clear;
4
5      global nobj ngrad
6      nobj = 0; % counter for objective evaluations
7      ngrad = 0.; % counter for gradient evaluations
8      x = [1.; 1.]; % starting point, set to be column vector
9      x1 = [10; 10; 10]; % starting point for function 1
10     xRosen = [-1.5; 1];
11     algoflag = 3; % 1=steepest descent; 2=conjugate gradient; 3=BFGS quasi-Newton
12     stoptol = 1.e-5; % stopping tolerance, all gradient elements must be < stoptol
13
14
15     % ---------- call fminun----------------
16     [xopt, fopt, exitflag] = fminun(@obj1, @gradobj1, x1, stoptol, algoflag);
17
18     xopt
19     fopt
20
21     nobj
22     ngrad
23  end
24
25  % function to be minimized
26  function [f] = obj(x)
27     global nobj
28     %example function
29     %min of 9.21739 as [-0.673913, 0.304348]T
30     f = 12 + 6.*x(1) - 5.*x(2) + 4.*x(1).^2 -2.*x(1).*x(2) + 6.*x(2).^2;
31     nobj = nobj +1;
32  end
33
34  % get gradient as a column vector
35  function [grad] = gradobj(x)
36     global ngrad
37     %gradient for function above
```

```matlab
38     grad(1,1) = 6 + 8.*x(1) - 2.*x(2);
39     grad(2,1) = -5 - 2.*x(1) + 12.*x(2);
40     ngrad = ngrad + 1;
41   end
42
43   % function 1 to be optimized on the homework
44   function [f] = obj1(x)
45     global nobj
46     f = 20 + 3 .* x(1) - 6 .* x(2) + 8 .* x(3) + 6 .* x(1).^2 - 2 .* x(1) .* x(2) ...
47       - x(1) .* x(3) + x(2).^2 + 0.5 .* x(3).^2;
48     nobj = nobj + 1;
49   end
50
51   % gradient of function 1
52   function [grad] = gradobj1(x)
53     global ngrad
54     grad(1,1) = 3 + 12 .* x(1) - 2 .* x(2) - x(3);
55     grad(2,1) = -6 - 2 .* x(1) + 2 .* x(2);
56     grad(3,1) = 8 - x(1) + x(3);
57     ngrad = ngrad + 1;
58   end
59
60
61   % function 1 to be optimized on the homework
62   function [f] = objRosen(x)
63     global nobj
64     f = 100 .* (x(2) - x(1).^2).^2 + (1-x(1)).^2;
65     nobj = nobj + 1;
66   end
67
68   % gradient of function 1
69   function [grad] = gradobjRosen(x)
70     global ngrad
71     grad(1,1) = 2 .* (200 .* x(1).^3 - 200 .* x(1) .* x(2) + x(1) - 1);
72     grad(2,1) = 200 .* (x(2) - x(1).^2);
73     ngrad = ngrad + 1;
74   end
```

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 $x^T = [10, 10, 10]$. 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 $x^T = [-1.5, 1]$, 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 $x^T = [1, 1]$ 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:

```matlab
%---------------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:

```matlab
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
```

12

4