CSC 2305 Numerical Methods for Optimization Problems (Winter 2016)
Assignment # 6

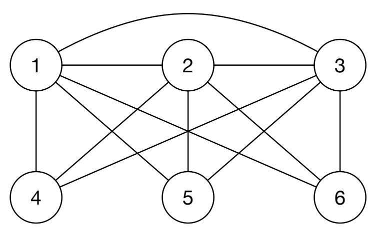
1002119049 Hao Wang (UTORid: wangh110)

April 8, 2016

1 Problem 1

1.1 Solution

We generate the following graph based on the Jacobian matrix has the structure. To find a combination of perturbation vectors p to obtain the minimum number of evaluations of f.



By node coloring algorithm, we could generate:

$$p = \epsilon e_1,\ p = \epsilon e_2,\ p = \epsilon e_3,\ p = \epsilon(e_4 + e_5 + e_6).$$

The in this way, the number of evaluations of f will be minimized to 4. If we have smaller number of evaluations, like 3, which results in two of the e_1, e_2, e_3 and $e_4+e_5+e_6$ will be evaluated at the same time. However, in this way, we can not obtain the correct Jacobian, since any two of them will overlap in a row.

2 Problem 2

2.1 Plots

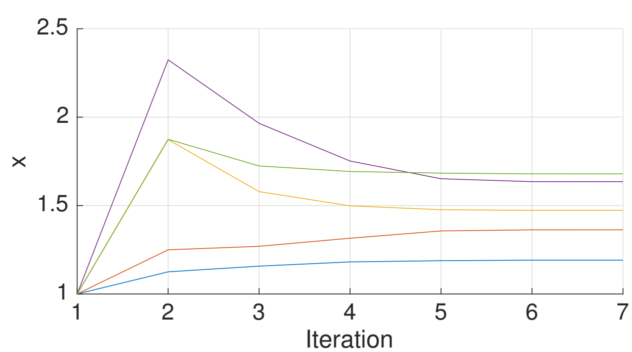


Figure 1: Convergence of x by Newton Method.

2.2 Comments

For the Newton Method, when arriving at the convergence, the $x = [1.1911, 1.3626, 1.4728, 1.6350, 1.6791]'$, $f(x) = 0.0788$ and $\lambda = [0.0388, 0.0167, 0.0003]'$. To check it for optimality, let's calculate the $\nabla f(x^*)$ and $\nabla^2 f(x^*)$:

$$\nabla f(x^*) = 1.0e - 07 \times [0.0507, -0.0165, -0.7556, 0.4640, 0.0676, -0.1958, 0.0316, 0.0570]' \approx \mathbf{0}$$

And $\nabla^2 f(x^*)$ is:

```
 4.0000   -2.0000         0         0   -0.0003   -1.0000         0   -1.6791
-2.0000    3.9224   -2.0000         0         0   -2.7252   -1.0000         0
      0   -2.0000    2.0061   -0.3157         0   -6.5076    2.9456         0
      0         0   -0.3157    0.3390   -0.0233         0   -1.0000         0
-0.0003         0         0   -0.0233    0.0233         0         0   -1.1911
-1.0000   -2.7252   -6.5076         0         0         0         0         0
      0   -1.0000    2.9456   -1.0000         0         0         0         0
-1.6791         0         0         0   -1.1911         0         0         0
```

By executing MATLAB code as below, we could obtain the eigenvalue of $\nabla^2 f(x^*)$:

```
-7.1535   -1.5883   -1.1798    0.6031    0.9700    3.7575    6.4761    8.4056
```

As we can see, the matrix of $\nabla^2 f(x^*)$ is not positive semidefinite. Therefore, this critical point does not meet the second-order optimality.

Let's pick nearby feasible points x, and keep $\lambda = [0.0388, 0.0167, 0.0003]'$. By changing x_1 and x_2, and calculating x_3, x_4 and x_5, we have the following output (the code is presented in the function `sampling` in source code list):

```
    x1        x2         x3         x4         x5        fx
 -------------------------------------------------------------
 0.6911    0.8626    1.6877    2.8143    2.8939    2.4164
 0.7911    0.9626    1.6540    2.6016    2.5281    1.3573
 0.8911    1.0626    1.6163    2.3782    2.2444    0.6852
 0.9911    1.1626    1.5740    2.1435    2.0180    0.3041
 1.0911    1.2626    1.5265    1.8962    1.8330    0.1261
|1.1911    1.3626    1.4728    1.6350    1.6791    0.0788| <-- critical point
 1.2911    1.4626    1.4115    1.3582    1.5491    0.1181
 1.3911    1.5626    1.3407    1.0633    1.4377    0.2572
 1.4911    1.6626    1.2572    0.7465    1.3413    0.6281
 1.5911    1.7626    1.1560    0.4022    1.2570    1.6037
 1.6911    1.8626    1.0267    0.0199    1.1827    4.0607
```

2.3 Output

2.3.1 Newton Method

Output of Newton Method:

x1	x2	x3	x4	x5	lambda1	lambda2	lambda3
1.1252	1.2496	1.8727	2.3242	1.8748	-0.7482	0	-0.1252
1.1573	1.2694	1.5787	1.9649	1.7240	-0.1724	0.0661	0.1300
1.1812	1.3152	1.4983	1.7518	1.6925	-0.0019	-0.0778	0.0583
1.1880	1.3564	1.4765	1.6515	1.6834	0.0362	0.0017	0.0022
1.1911	1.3625	1.4729	1.6352	1.6792	0.0388	0.0165	0.0002
1.1911	1.3626	1.4728	1.6350	1.6791	0.0388	0.0167	0.0003

2.3.2 By MATLAB `fmincon`

The convergence of x:

x1	x2	x3	x4	x5	fx
1.1911	1.3626	1.4728	1.6350	1.6791	0.0788

The iteration output of `fmincon`:

Iter	F-count	f(x)	Feasibility	First-order optimality	Norm of step
0	6	1.000000e+00	6.607e+00	2.564e-01	
1	15	6.255168e-01	5.782e+00	2.323e+00	8.604e-01
2	23	2.944478e+00	2.520e+00	3.238e+00	1.884e+00
3	30	2.409655e+00	1.199e+00	4.269e+00	8.752e-01
4	39	1.496926e+00	1.693e+00	4.238e+00	4.636e-01
5	45	4.295142e-01	1.145e+00	1.693e+00	1.383e+00
6	53	2.322135e-01	1.259e+00	1.902e+00	5.178e-01
7	59	1.189454e-01	2.805e-01	2.378e+00	6.067e-01
8	65	7.439318e-02	1.821e-01	1.900e-01	1.289e-01
9	71	8.028673e-02	1.903e-02	8.334e-01	1.239e-01
10	81	7.978084e-02	1.879e-02	3.941e-01	2.502e-03
...					
41	361	7.877681e-02	4.336e-07	1.159e-01	5.399e-08
42	369	7.877681e-02	4.323e-07	1.066e-01	3.779e-07
43	379	7.877682e-02	1.247e-07	5.268e-02	1.794e-07
44	392	7.877683e-02	6.978e-08	3.030e-02	4.532e-08
45	399	7.877683e-02	6.769e-08	3.053e-02	9.064e-08
46	413	7.877682e-02	6.967e-08	1.089e-01	4.532e-08
47	430	7.877682e-02	6.959e-08	1.635e-01	5.665e-09

2.4 Source code list

```
function a6_2
    TOL = 1e-6;

    %% fmincon
```

```
    x0 = zeros(5, 1);
    options = optimoptions('fmincon','Display','iter');
    [x, fval] = fmincon(@(x) objFunc(x), x0, ...
        [],[],[],[],[],[], @(x) cons(x), options);

    disp(x);
    disp(fval);

    %% newton
    x0 = ones(8, 1);

    [xiter, output] = newton(@lagrangian, x0, TOL);
    fig = draw(xiter);
    fig.PaperUnits = 'inches';
    fig.PaperPosition = [0 0 6 3];
    print(fig, '-depsc', '-r0', sprintf('../figs/a6-2-1'));

    dlmwrite(sprintf('../output/a6-2-1.txt'), ...
        output, 'delimiter','\t', 'precision', '%6.6g');
end
```

```
function f = objFunc( x )
    f = (x(1)-1)^2 + (x(1)-x(2))^2 + (x(2)-x(3))^2 ...
    + (x(3)-x(4))^4 + (x(4)-x(5))^4;
end
```

```
function [ c, ceq ] = cons( x )
    a = [3*sqrt(2)+2; 2*sqrt(2)-2; 2];

    y = [x(1)+x(2)^2+x(3)^3; ...
        x(2)-x(3)^2+x(4); ...
        x(1)*x(5)];

    ceq = norm(y - a);
    c = [];
end
```

```
function [ y, gy, hy ] = lagrangian( x )
    t = num2cell(x);
    [x1, x2, x3, x4, x5] = t{1:5};
    [lambda1, lambda2, lambda3] = t{6:8};


    y = (x1 - x2)^2 + (x2 - x3)^2 ...
```

```
        + (x3 - x4)^4 + (x4 - x5)^4 ...
        - lambda1*(x2^2 + x3^3 + ...
        x1 - 3514294284039789/562949953421312) ...
        - lambda2*(- x3^2 + x2 + x4 ...
        - 1865452045155277/2251799813685248) ...
        + (x1 - 1)^2 - lambda3*(x1*x5 - 2);

    gy = [4*x1 - lambda1 - 2*x2 - lambda3*x5 - 2;
        4*x2 - 2*x1 - lambda2 - 2*x3 - 2*lambda1*x2;
        2*x3 - 2*x2 + 4*(x3 - x4)^3 + 2*lambda2*x3 - 3*lambda1*x3^2;
        4*(x4 - x5)^3 - 4*(x3 - x4)^3 - lambda2;
        - 4*(x4 - x5)^3 - lambda3*x1;
        - x2^2 - x3^3 - x1 + 3514294284039789/562949953421312;
        x3^2 - x2 - x4 + 1865452045155277/2251799813685248;
        2 - x1*x5];

    hy = [4, -2, 0, 0, -lambda3, -1, 0, -x5;
        -2, 4 - 2*lambda1, -2, 0, 0, -2*x2, -1, 0;
        0, -2, 2*lambda2 + 12*(x3 - x4)^2 - 6*lambda1*x3 + 2, ...
        -12*(x3 - x4)^2, 0, -3*x3^2, 2*x3,   0;
        0, 0,  -12*(x3 - x4)^2, 12*(x3 - x4)^2 ...
         + 12*(x4 - x5)^2, -12*(x4 - x5)^2, 0,   -1,   0;
        -lambda3,  0, 0, -12*(x4 - x5)^2, ...
         12*(x4 - x5)^2,  0,    0, -x1;
        -1, -2*x2, -3*x3^2, 0, 0,       0,    0,   0;
        0, -1, 2*x3, -1, 0,       0,    0,   0;
        -x5, 0, 0, 0, -x1,       0,    0,   0];

end
```

```
function [ xiter, outputlist ] = newton( func, x, tol )
    [~, gx, hx] = feval(func, x);
    gxnorm = norm(gx,2);
    it = 1;
    xiter{it} = x;
    outputlist(it, :) = [it, 0];

    while( gxnorm > tol )
        p = -inv(hx) * gx;
        alphak = backtracking(func, p, x, 0.5, 0.1);
        x = x + alphak * p;

        it = it + 1;
        xiter{it} = x;
        disp([x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8)]);
        diff = norm(xiter{it} - xiter{it-1});
        outputlist(it, :) = [it, diff];
```

```

        [~, gx, hx] = feval(func, x);
        gxnorm = norm(gx,2);
    end
    disp(gx);
    disp(hx);
end
```

```
function alphak = backtracking( f, d, x, rho, c )

    alphak = 1;
    [fk, gk] = feval(f, x);
    xx = x;
    x = x + alphak * d;
    fk1 = feval(f, x);

    while fk1 > fk + c * alphak * (gk' * d),
      alphak = alphak * rho;
      x = xx + alphak * d;
      fk1 = feval(f, x);
    end
end
```

```
function [] = sampling()
    x0 = [1.1911, 1.3626, 1.4728, 1.6350, 1.6791];

    for i = -5:5
        x1 = x0(1)+i*0.1;
        x2 = x0(2)+i*0.1;
        x3 = (3*sqrt(2)+2-x1-(x2)^2)^(1/3);
        x4 = (2*sqrt(2)-2-x2+(x3)^2);
        x5 = 2/x1;
        x = [x1,x2,x3,x4,x5];
        fx = objFunc(x);
        disp([x1,x2,x3,x4,x5,fx]);
    end
end
```

3 Problem 3

3.1 Comments

(a)

Problem	**is Considered**	**Example**
Bound Constrained Optimization	Yes	Golden search
Combinatorial Optimization	No	
Complementarity Problems	No	
Constrained Optimization	Yes	Lagrange multiplier
Continuous Optimization	Yes	Line search
Derivative-Free Optimization	Yes	Gradient approximation
Discrete Optimization	No	
Global Optimization	No	
Integer Linear Programming	No	
Linear Programming	No	
Mixed Integer Nonlinear Programming (MINLP)	No	
Mathematical Programs with Equilibrium Constraints (MPEC)	No	
Multi-Objective Optimization	No	
Network Optimization	No	
Non-differentiable Optimization	No	
Nonlinear Programming	Yes	Conjugate gradient
Nonlinear Equations	Yes	Conjugate gradient
Nonlinear Least-Squares Problems	Yes	Conjugate gradient
Optimization Under Uncertainty	No	
Quadratically Constrained Quadratic Programming (QCQP)	No	
Quadratic Programming (QP)	Yes	Quadratic section
Semidefinite Programming (SDP)	No	
Semiinfinite Programming (SIP)	No	
Stochastic Linear Programming	No	
Second Order Cone Programming (SOCP)	No	
Stochastic Programming	No	
Traveling Salesman Problem (TSP)	No	
Unconstrained Optimization	Yes	Line search

(b) We need to input n as float, X as initial point, fx as the function, `MY_GRAD` for gradient and `MY_HESS` for hessian.