Communications and Information Engineering Program
Linear and Nonlinear Programming
MATH 404 - Fall 2020

Report 3

Salma Elbess
201601152

Introduction

In this report, four 1D minimization algorithms were implemented and tested in the main script the algorithms are fibonacci method, golden ration method, quadratic interpolation method and cubic interpolation methods. In addition, two benchmark problems were solved using implemented unconstrained nonlinear optimization algorithms. The algorithms are Fletcher-Reeves CG method, Marquardt method and Quasi-Newton Method. The three methods got the same parameters for each problem such as tolerance, function, hessian function, .. etc. For getting optimal step size, Fletcher-Reeves uses a closed form equation, but Quasi-Newton and Marquardt use the fibonacci method for 1D minimization with n = 11. The tolerance passed to the first and second problems are 0.01 and 0.0001 respectively. Rank 1 update is used in the Quasi-Newton method.

Results

Rosenbrock's parabolic valley

Criteria	Fletcher-Reeves CG Method	Marquardt Method	Quasi-Newton Method
Number of iterations	71405	81	20
The optimal solution	[1,1]	[1.0000, 1.0000]	[1.0007, 1.00015]
The optimal value	4.189e-10	4.8950e-08	2.2839e-08
CPU time in sec	0.1370	9.9577e-04	9.9577e-04

Powell's quartic function

Criteria	Fletcher-Reeves CG Method	Marquardt Method	Quasi-Newton Method
Number of iterations	108144	25	22
The optimal solution	[0.0040,	[0.0159,	[-0.0163,

	-3.975e-04, 0.0022, 0.0022]	**-0.0016, 0.0079, 0.0079]**	**0.0016, -0.0047, -0.0047]**
The optimal value	**6.9947e-09**	**1.3198e-07**	**1.8841e-07**
CPU time in sec	**0.144**	**0.001**	**9.9577e-04**

Discussion

In both problems, Fletcher- Reeves method had the maximum number of iterations but with minimum function value. However, all of them reached values that will be considered as 0. The three algorithms are considered to produce the same optimal point after some acceptable rounding. However, Fletcher- Reeves got the best precision. In addition, Fletcher- Reeves is the slowest compared to the two other algorithms.
CPU time varies in each run as it is calculated using processor time not CPU cycles.

References

Rao, S. (2009). *Engineering optimization*.

Source code

Main file

```
%%1D minimization
clc; clear;
f = @(lambda) lambda^5 - 5*(lambda^3) - 20*lambda + 5;
diff_f = @(lambda) 5*lambda^4 - 15*(lambda^2) - 20;
A = 0;
t = 0.5;
eps = 0.0001;
lambda_star_quad = quad_1d_min(f,A,t,eps);
eps1 = 0.05;
eps2 = 0.05;
lambda_star_cub = cub_1d_min(f,diff_f,A,t,eps1,eps2);
[fib_a,fib_b] = fibonacci_1d_minimization(f,6,0,1);
[golden_a,golden_b] = golden_ration_1d_minimization(f,6,0,1);
%% Rosenbrock
problem_number = 1;
f = @(x) 100*((x(2) - x(1)^2)^2) +(1-x(1))^2;
grad_f = @(x) [-400*(x(2) - x(1)^2)*x(1) - 2*(1-x(1)) ; 200*(x(2)-x(1)^2)];
hessian_f = @(x) [-400*(x(2) - x(1)^2)+ 800*x(1)^2,-400*x(1);-400*x(1),200];
```

```
eps = 10^-2; start = [-1.2;1];
%Quasi-Newton Method.
[quasi_i_1,quasi_sol_1,quasi_val_1,quasi_time_1] =
quasi_newton(f,grad_f,start,eps,problem_number);
%Fletcher-Reeves CG Method.
[fr_i_1,fr_sol_1,fr_val_1,fr_time_1] = FR(f,grad_f,hessian_f,start);
%Marquardt Method.
[marq_i_1, marq_sol_1,marq_val_1,marq_time_1] =
marq(f,grad_f,hessian_f,start,eps,problem_number);
%% powell
problem_number = 2;
f = @(x) (x(1) + 10*x(2))^2 + 5*(x(3)-x(4))^2 +(x(2)-2*x(3))^4 + 10*(x(1)-x(4))^4;
grad_f = @(x) [2*(x(1)+10*x(2)) + 40*(x(1)-x(4))^3 ;
20*(x(1)+10*x(2)) + 4*(x(2)-2*x(3))^3;
10*(x(3)-x(4)) - 8*(x(2)-2*x(3))^3;
-10*(x(3)-x(4)) - 40*(x(1)-x(4))^3];
hessian_f = @(x)[2+120*(x(1)-x(4))^2,20,0,-120*(x(1)-x(4))^2;
    20,200+12*(x(2)-2*x(3))^2,-24*(x(2)-2*x(3))^2,0;
    0,-24*(x(2)-2*x(3))^2,10 + 48*(x(2)-2*x(3))^2,-10;
    -120*(x(1)-x(4))^2, 0, -10, 10+120*(x(1)-x(4))^2];
eps = 10^-4; start = [3;-1;0;1];
%Quasi-Newton Method.
[quasi_i,quasi_sol,quasi_val,quasi_time] = quasi_newton(f,grad_f,start,eps,problem_number);
%Fletcher-Reeves CG Method.
[fr_i,fr_sol,fr_val,fr_time] = FR(f,grad_f,hessian_f,start);
%Marquardt Method.
[marq_i, marq_sol,marq_val,marq_time] = marq(f,grad_f,hessian_f,start,eps,problem_number);
```

Fibonacci 1D minimization Method

```
function[a,b] = fibonacci_1d_minimization(f,n,a,b)
N = n+1;
%get fibonacci sequence
fib_seq = zeros(1,n+1);
fib_seq(1) = 1; fib_seq(2) = 1;
for i = 3:length(fib_seq)
    fib_seq(i) = fib_seq(i-1) + fib_seq(i-2);
end

for i = N:-1 :3
L2 = fib_seq(i-2)*(b-a)/fib_seq(i);
L1 = b-a;
if L2 > L1/2
    x1 = b-L2;
```

```
    x2 = a+L2;
else
    x2 = b-L2;
    x1 = a+L2;
end
f1 = double(f(x1));
f2 = double(f(x2));
if f1 > f2
        a = x1;

else
        b = x2;

end
end
end
```

Golden Ratio 1D minimization method

```
function[a,b] = golden_ration_1d_minimization(f,n,a,b)
N = n+1;
for i = N:-1 :3
L2 = 0.382*(b-a);
L1 = b-a;
if L2 > L1/2
    x1 = b-L2;
    x2 = a+L2;
else
    x2 = b-L2;
    x1 = a+L2;
end
f1 = double(f(x1));
f2 = double(f(x2));
if f1 > f2
        a = x1;
elseif f1 <f2
        b = x2;
elseif f1==f2
    a = x1;
    b=x2;
end
end
end
```

Quadratic interpolation method

```
function lambda_star = quad_1d_min(f,A,t,eps)
fa= f(A);
```

```
f1 = f(t);
if f1 > fa
    fc = f1;
    fb = f(t/2);
    t = t/2;
elseif f1<fa
    while(true)
        fb = f1;
        f2 = f(2*t);
        if f2 > f1
            fc = f2;
            break;
        else
            f1 = f2;
            t = 2*t;
        end
    end
end
B = t;
C = 2*t;
if A == 0
    lambda_star = (4*fb - 3*fa - fc)*t/(4*fb - 2*fc - 2*fa);
else
    lambda_star = fb*(C^2 - A^2) + fa*(B^2 - C^2) + fc*(A^2 - B^2)/2*(fa*(B-C) +fb*(C-A) + fc*(A-B));
end
syms a b c
[sola,solb, solc] = solve(fa == a+b*A + c*(A^2),fb == a+b*B + c*(B^2),fc == a+b*C + c*(C^2),[a b c]);
h = double(sola)+double(solb)*lambda_star + double(solc) *(lambda_star^2);
f_lambda = f(lambda_star);
i = 1;
%%refitting
if abs((h - f_lambda)/f_lambda) > eps
    while abs((h - f_lambda)/f_lambda) > eps
        i = i+1;
    if lambda_star > B
        if f_lambda < fb %neglect old A
            A = B;
            B = lambda_star;
        else %neglect old C
            C = lambda_star;
        end
    else
       if f_lambda < fb %neglect old C
           C = B;
            B = lambda_star;
        else %neglect old A
            A = lambda_star;
```

```
        end
    end
    fb = f(B); fc = f(C); fa = f(A);
    syms a b c
[sola,solb, solc] = solve(fa == a+b*A + c*(A^2),fb == a+b*B + c*(B^2),fc == a+b*C + c*(C^2),[a b c]);
lambda_star = -1*double(solb)/(2*double(solc));
    h = double(sola)+double(solb)*lambda_star + double(solc) *(lambda_star^2);
    f_lambda = f(lambda_star);
    n = abs((h - f_lambda)/f_lambda);
    end
end
end
```

Cubic interpolation method

```
function lambda_star = cub_1d_min(f,diff_f,A,t,eps1,eps2)
diff_t = diff_f(t);
while diff_t < 0
    t = t*2;
    diff_t = diff_f(t);
end
B = t;
i = 0; % number of iterations
while true
    i = i+1;
    fa = f(A); fb = f(B); diff_fa = diff_f(A); diff_fb = diff_f(B);
    Z = (3*(fa- fb)/(B-A)) +diff_fa +diff_fb;
    Q = sqrt(Z^2 - (diff_fa*diff_fb));
    lambda_star1 = A + ((diff_fa+Z+Q)*(B-A)/(diff_fa+diff_fb + 2*Z));
    lambda_star2 = A + ((diff_fa+Z-Q)*(B-A)/(diff_fa+diff_fb + 2*Z));
    if lambda_star1 <= B && lambda_star1 >= A
        lambda_star = lambda_star1;
    else
        lambda_star = lambda_star2;
    end
    b = (diff_fa*(B^2)+diff_fb*(A^2) + 2*A*B*Z)/((A-B)^2);
    c = -1*(Z*(A+B) + B*diff_fa + A*diff_fb)/((A-B)^2);
    d = (2*Z + diff_fa +diff_fb)/(3*(A-B)^2);
    a = fa - b*A - c*(A^2) - d*(A^3);
    h = a + b*lambda_star + c*lambda_star^2 + d*lambda_star^3;
    f_lambda = f(lambda_star);
    diff_f_lambda = diff_f(lambda_star);
    if abs(diff_f_lambda) <= eps2 && abs((h-f_lambda)/f_lambda) <= eps1
        break;
    end
    if diff_f_lambda > 0
        B = lambda_star;
    else
```

```
        A = lambda_star;
    end
end
end
```

Quasi-Newton Method

```
function [i, x_old,f_old,imp_time] = quasi_newton(f,grad_f,x_new,eps,problem_number)
%intial values
B = eye(length(x_new));
i = 0;
imp_start = now;
%start algorithm
while (true)
    i = i+1;

    %update old values
    x_old = x_new;
    f_old = f(x_old);
    grad_old = grad_f(x_old);

    %stopping criteria
    if norm(grad_old) < eps
        break;
    end

    %get S
    S =  - B\grad_old;
    %get lambda
    if problem_number ==1
        f_lambda = @(lambda) 100*(((x_old(2)+lambda*S(2)) - (x_old(1)+lambda*S(1))^2)^2)
+(1-(x_old(1)+lambda*S(1)))^2;
    else
        f_lambda = @(lambda) ((x_old(1)+lambda*S(1)) + 10*(x_old(2)+lambda*S(2)))^2 +
5*((x_old(3)+lambda*S(3))-(x_old(4)+lambda*S(4)))^2
+((x_old(2)+lambda*S(2))-2*(x_old(3)+lambda*S(3)))^4 +
10*((x_old(1)+lambda*S(1))-(x_old(4)+lambda*S(4)))^4;
    end
    [lower_limit,upper_limit] = fibonacci_1d_minimization(f_lambda,11,0,1);
    lambda_star = (upper_limit+lower_limit)/2;

    %get new values
    x_new = x_old + lambda_star.*(S);
    grad_new = grad_f(x_new);

    %update B
    y = grad_new - grad_old;
    g=x_new - x_old;
```

```
    B = B + (y-B*g)*(y-B*g)'/((y-B*g)'*g);
end
%end of algorithm
imp_end= now;
imp_time = (imp_end - imp_start)*86400;
end
```

Fletcher-Reeves CG Method

```
function [i, x_new,f_new,imp_time] = FR(f,grad_f,hessian_f,x_old)
%initial values
imp_start = now;
grad_old = grad_f(x_old);
S = - grad_old/norm(grad_old);
i = 0;

%start algorithm
while true
    i = i+1;

    %get lambda
    A = hessian_f(x_old);
    lambda_star = norm(grad_old/norm(grad_old))^2 /(S'*A*S);

    %update x
    x_new = x_old + lambda_star.*S;
    grad_new = grad_f(x_new);

    %stoping criteria
    if norm(grad_new) < 1e-3
        break;
    end

    %update s
    beta = ((norm(grad_new))^2)/ ((norm(grad_old))^2);
    S = -1*grad_new + beta*S;
    S = S/norm(S);

    %change inital values
    x_old = x_new;
    grad_old = grad_new;

end
%end of algoritm

%get final values
f_new = f(x_new);
imp_end= now;
```

```
imp_time = (imp_end - imp_start)*86400;
end
```

Marquardt Method

```
function [i, x_old,f_old,imp_time] = marq(f,grad_f,hessian_f,x_new,eps,problem_number)
%initial values
imp_start = now;
alpha1 = 10^4; c1 = 0.5; c2 = 2;
i = 0;
%start algorithm
while (true)
    i = i+1;
    %update old values
    x_old = x_new; f_old = f(x_old);
    grad = grad_f(x_old); f_hess = hessian_f(x_old);
    %stoping criteria
    if norm(grad) < eps
        break;
    end

    %get S
    I = eye(size(f_hess));
    S = -1*(f_hess+ alpha1*I)^-1 * grad;

    %get lambda
    if problem_number ==1
        f_lambda = @(lambda) 100*(((x_old(2)+lambda*S(2)) - (x_old(1)+lambda*S(1))^2)^2)
+(1-(x_old(1)+lambda*S(1)))^2;
    else
        f_lambda = @(lambda) ((x_old(1)+lambda*S(1)) + 10*(x_old(2)+lambda*S(2)))^2 +
5*((x_old(3)+lambda*S(3))-(x_old(4)+lambda*S(4)))^2
+((x_old(2)+lambda*S(2))-2*(x_old(3)+lambda*S(3)))^4 +
10*((x_old(1)+lambda*S(1))-(x_old(4)+lambda*S(4)))^4;
    end

    [lower_limit,upper_limit] = fibonacci_1d_minimization(f_lambda,11,0,1);
    lambda_star = (upper_limit+lower_limit)/2;

    %update new values
    x_new = x_old + lambda_star*S;
    f_new = f(x_new);
    if f_new < f_old
        alpha1 = alpha1*c1;
    else
        alpha1 = c2*alpha1;
    end
end
```

```
imp_end= now;
imp_time = (imp_end - imp_start)*86400;
end
```