MECH 579
Multidisciplinary Design Optimization
Siva Nadarajah

20/20

Project 2: Constrained Optimization

Dritan Harizaj
260426327

October 15, 2013

Table of Contents

Introduction

The purpose of this project is to minimize the Rosenbrock function using the Sequential Quadratic Programming algorithm. Both a Newton and Quasi-Newton (BFGS) approach was implemented to solve 2 different equality constraints:

$$\hat{c}_1(x) = 1 - x - y = 0$$

$$\hat{c}_2(x) = 1 - x^2 - y^2 = 0$$

Only one constraint at a time was used, yielding a total of 4 different algorithm-constraint combinations. A modified Armijo condition involving the merit function $\varphi(x_k;\mu)$ was used to control the step length in a modified backtracking approach. In all cases, the variable to be minimized was $\nabla \pounds_x(x,y)$, i.e. the derivative of the Lagrangian with respect to the vector X = [x y], evaluated at (x,y).

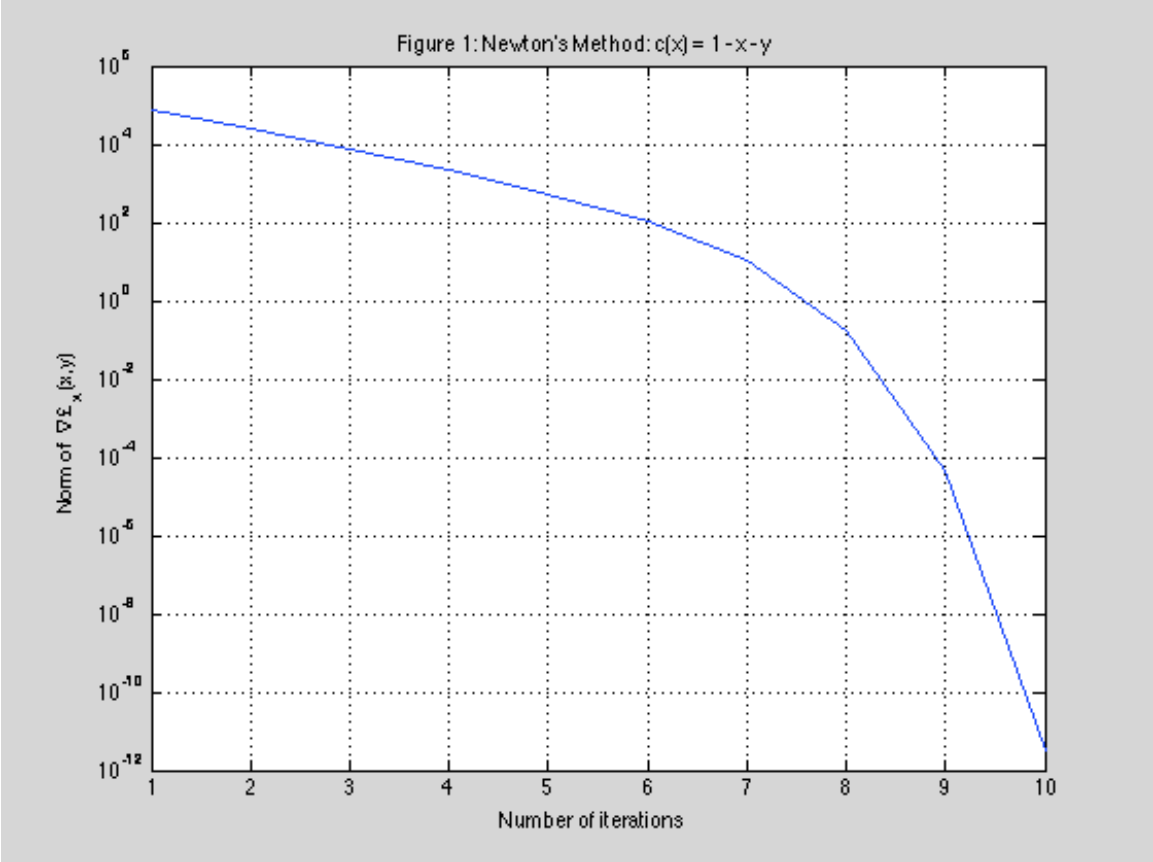
The starting point in all 4 cases was set to [x , y] = [6 , 5] and $\lambda = 1$ (unless otherwise indicated), while the tolerance was kept at $\varepsilon = 10^{-8}$. The value of τ that controls the backtrack reduction (i.e. $\alpha_k = \alpha_k \cdot \tau$) was set to 0.9, while $\eta = 10^{-4}$ and $\mu = 10$.

First constraint: $c_1(x) = 1 - x - y = 0$

Newton's Method

Iterations = 10
Final value [x , y] = [0.6188 , 0.3812]
λ at convergence = 0.3407

$c_1(x) = 1 - 0.6188 - 0.3812 = 0$



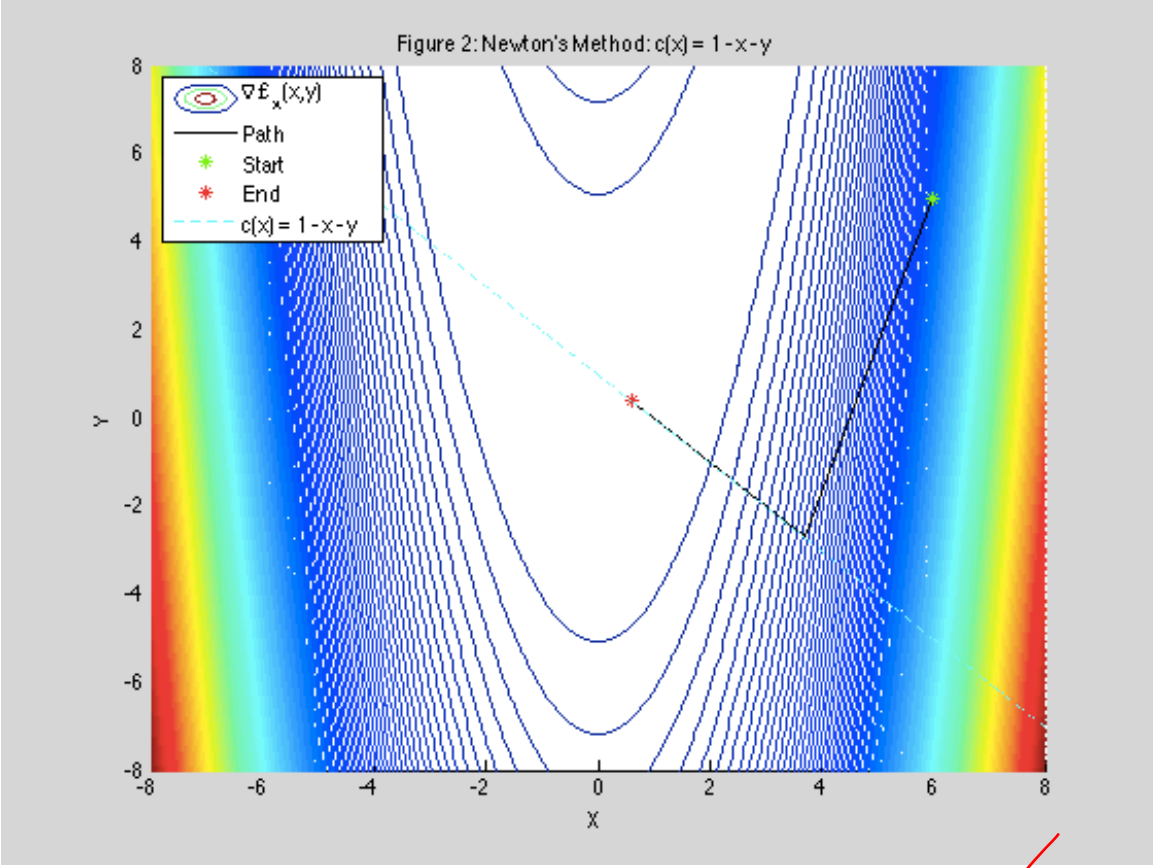
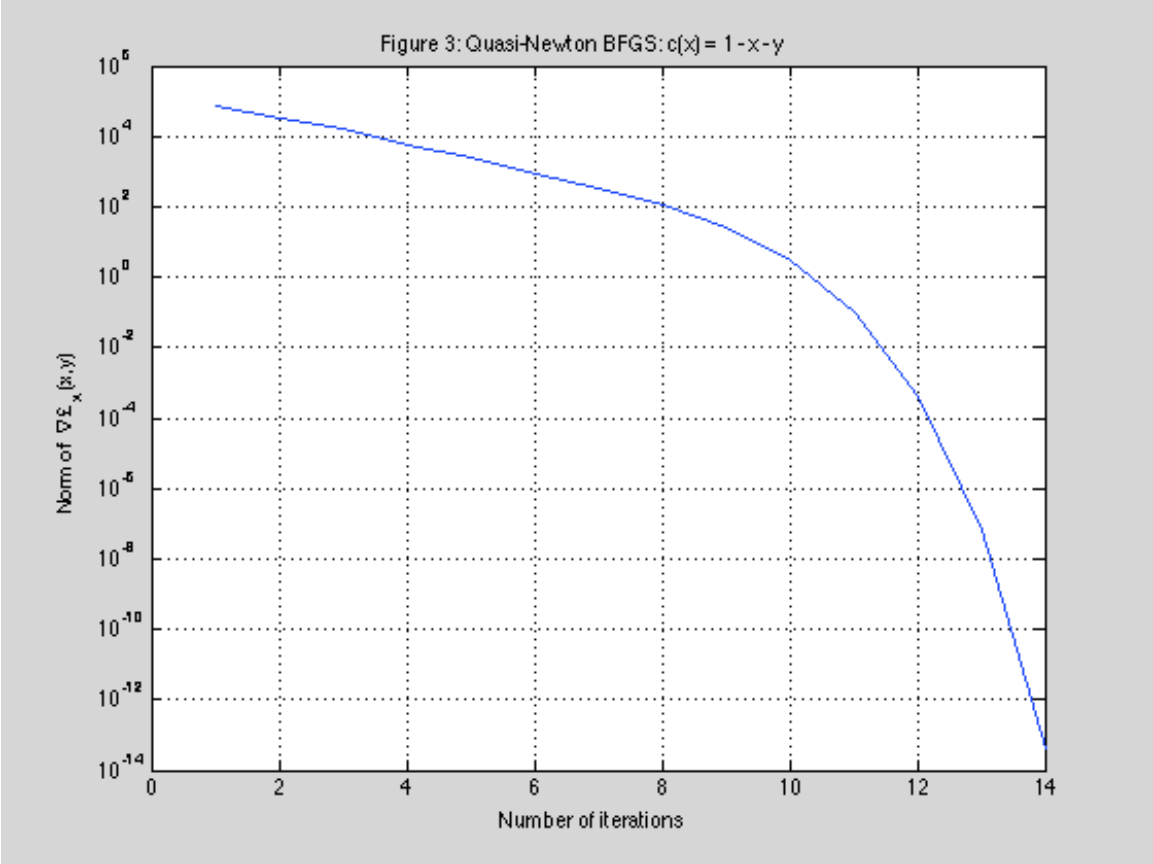


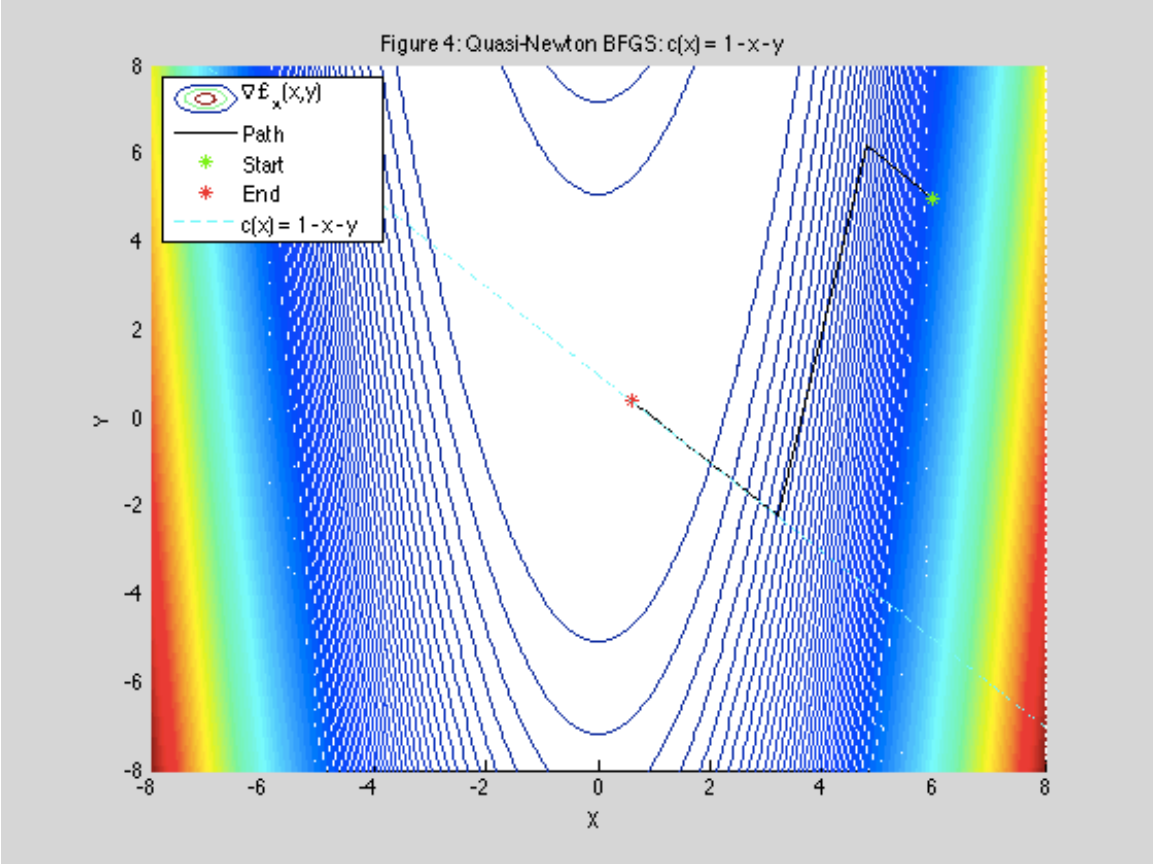
Figure 2: Newton's Method: c(x) = 1 - x - y
∇£x(x,y)
Path
Start
End
c(x) = 1 - x - y
8
6
4
2
0
-2
-4
-6
-8
Y
-8
-6
-4
-2
0
2
4
6
8
X

Quasi-Newton BFGS

Iterations = 14
Final value [x , y] = [0.6188 , 0.3812]
λ at convergence = 0.3407

$c_1(x) = 1 - 0.6188 - 0.3812 = 0$



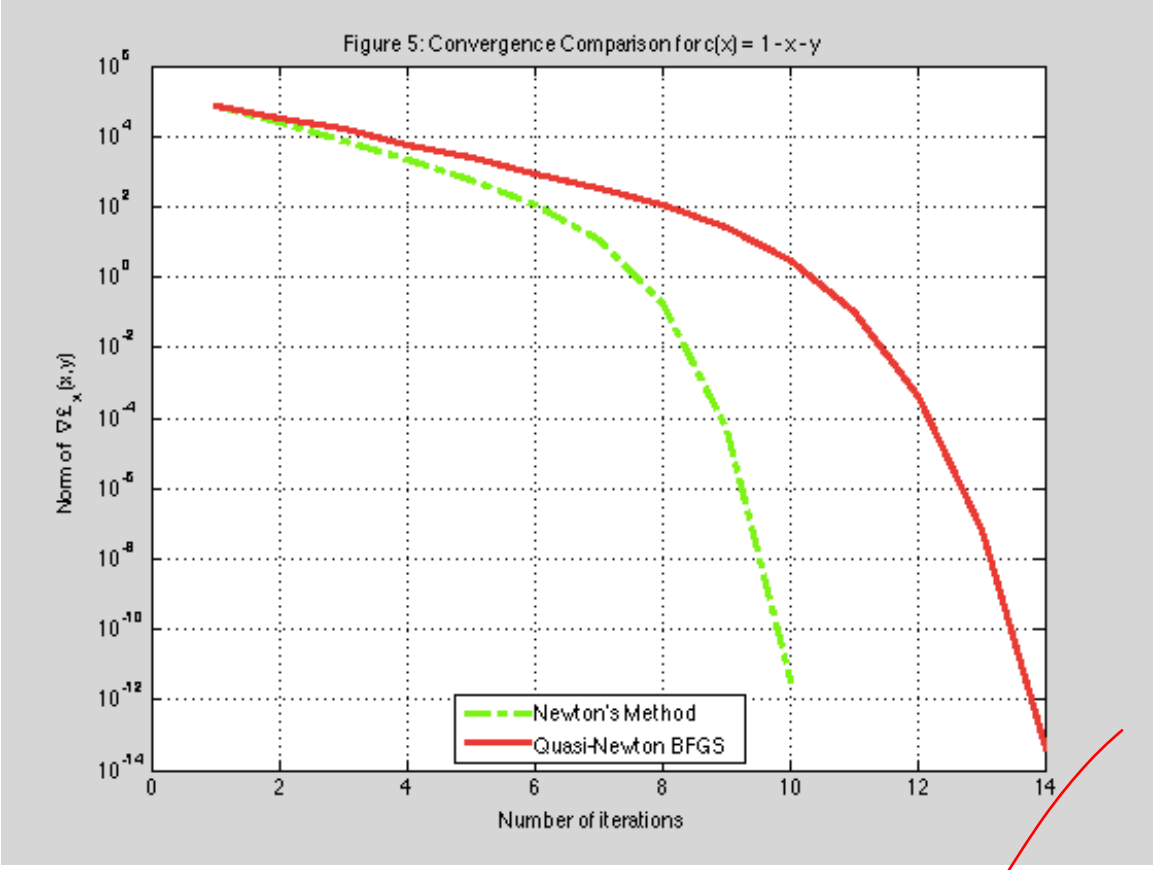


As mentioned during office hours Tuesday October 15th, initially the Hessian was used to initialize the pseudo-hessian approximation matrix B. Choosing the identity matrix caused the algorithm to diverge.

However, this error was determined to be due to a bug in the code related to backtracking, and the problem was subsequently fixed. Specifically, the problem was in the maximum number of iterations of α, i.e. how many ($\alpha_k = \alpha_k \cdot \tau$) iterations were allowed before stopping the backtracking loop: by reducing this number from 1000 to 100, α was not allowed to be reduced too much and the algorithm converged normally.

Notice that the initial search direction is different from Newton's method, which uses the Hessian to determine the optimal p_k.

Comparison



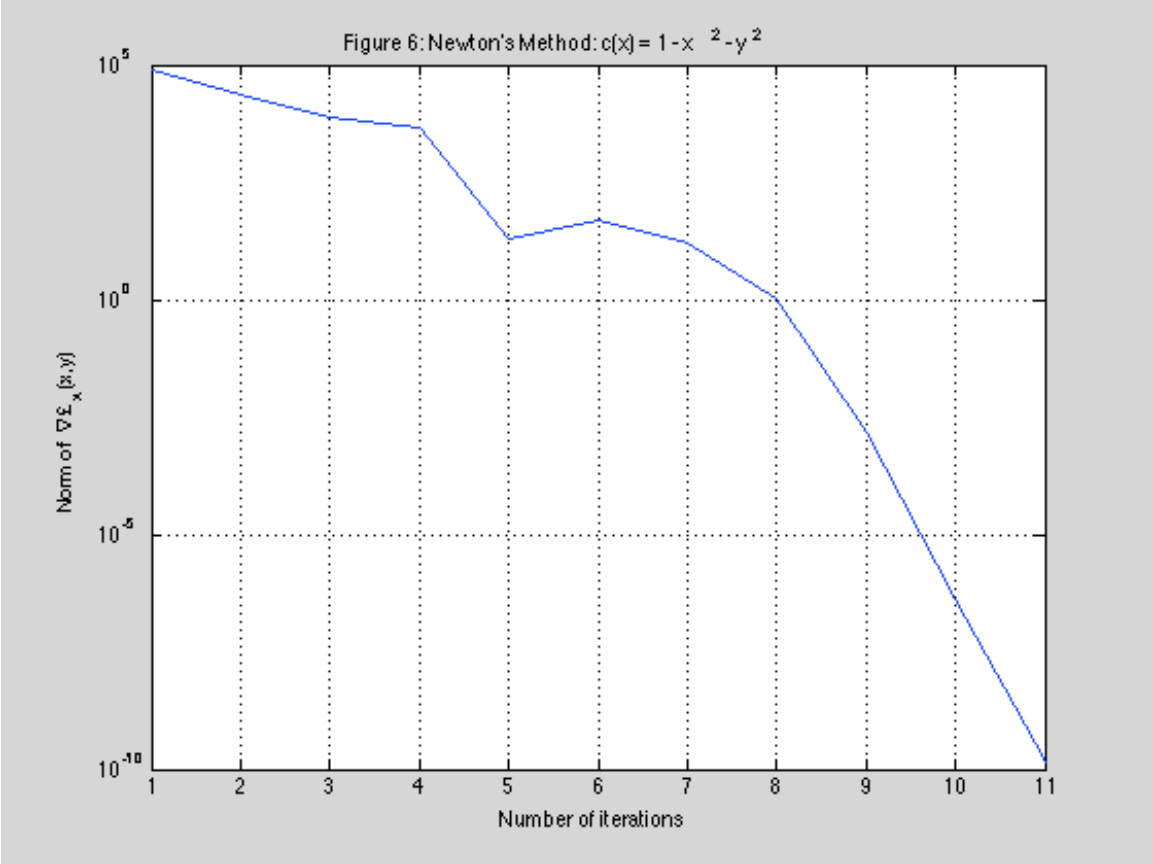
Both methods converge quite rapidly, although Newton's method takes only 10 iterations while BFGS takes 14. They display the same convergence shape.

Second Constraint: $\hat{c}_2(x) = 1 - x^2 - y^2 = 0$

Newton's Method

Iterations = 10
Final value [x , y] = [0.7864 , 0.6177]
λ at convergence = 0.1215

$c_2(x) = 1 - 0.7864^2 - 0.3812^2 = 0$



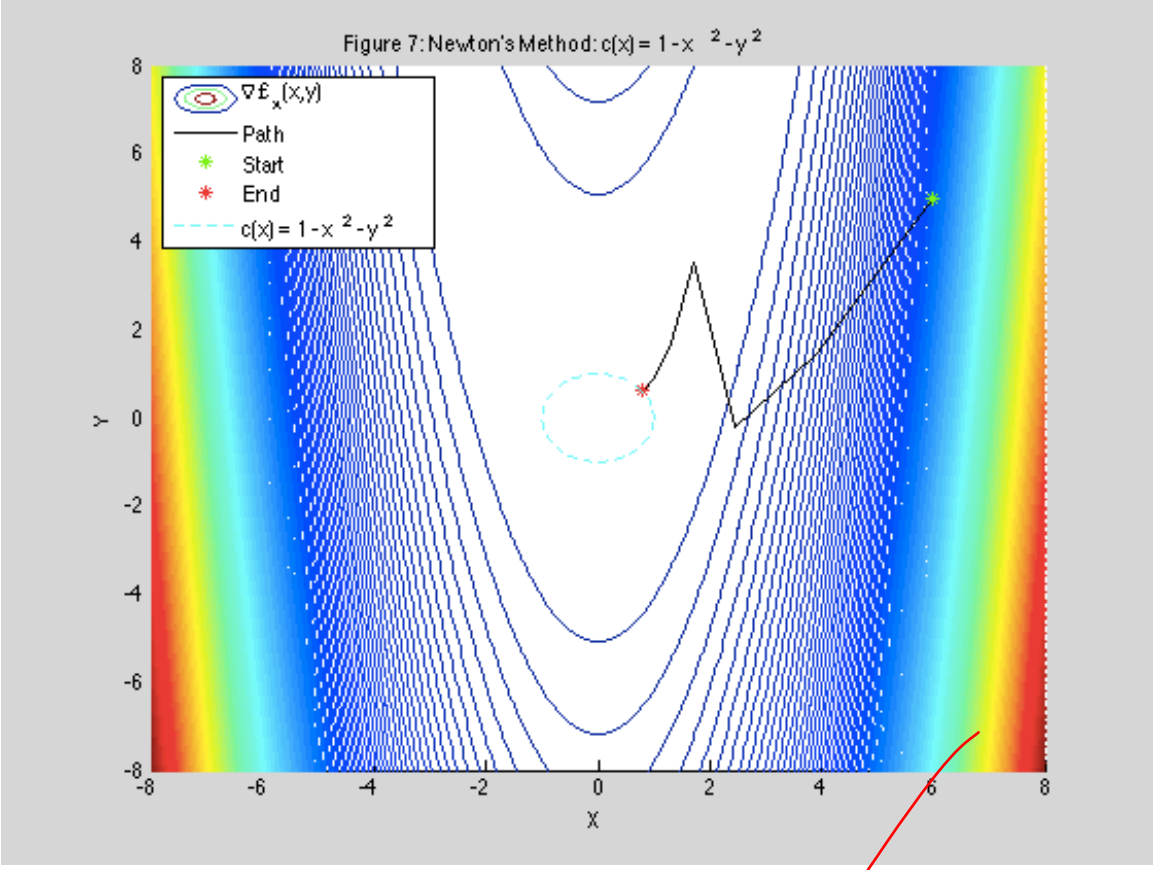
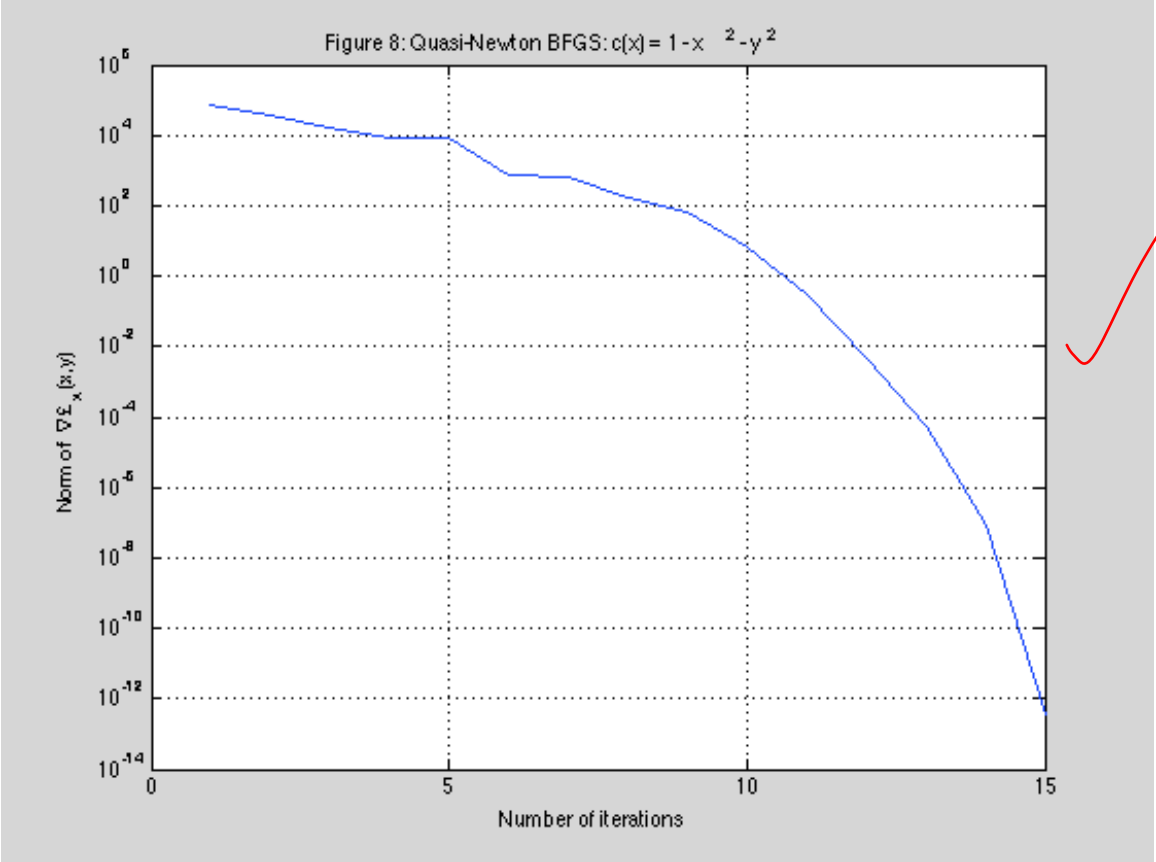


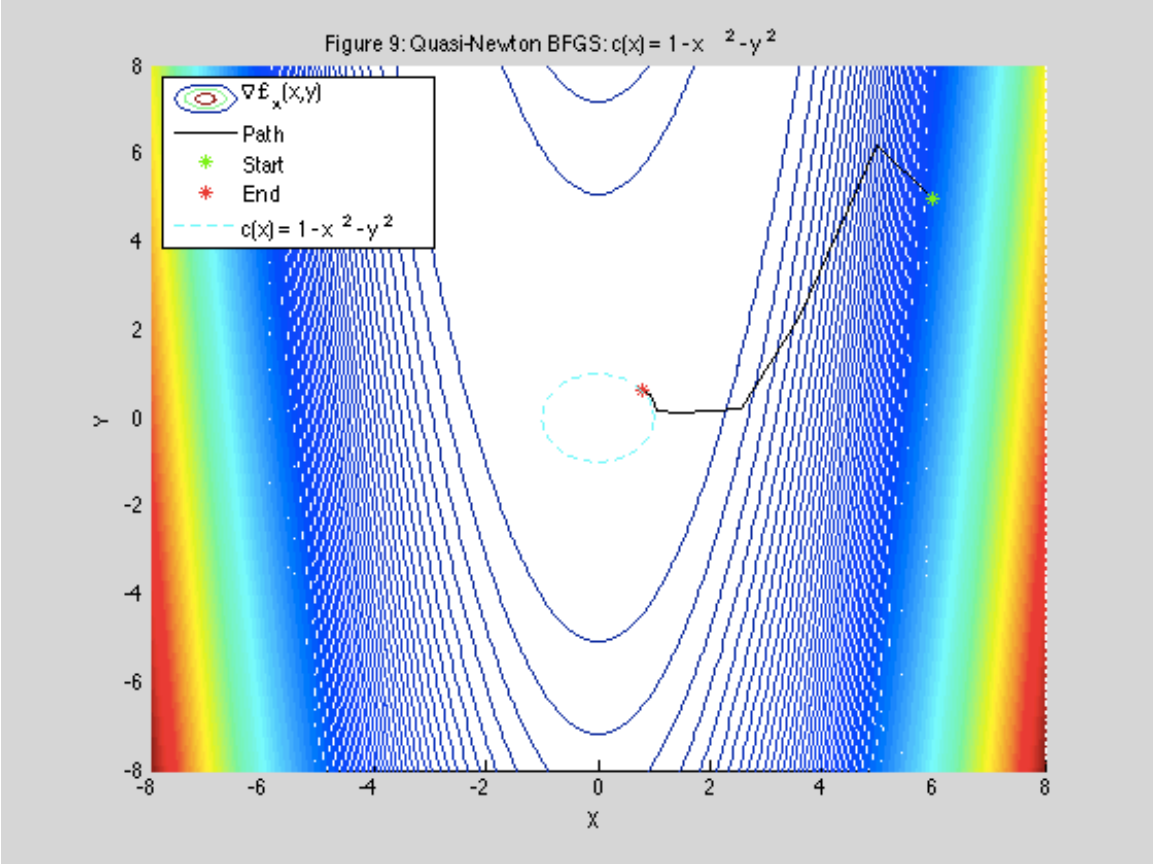
Figure 7: Newton's Method: c(x) = 1 - x ^2 - y ^2
∇£x(x,y)
Path
Start
End
c(x) = 1 - x ^2 - y ^2
8
6
4
2
0
-2
-4
-6
-8
Y
-8
-6
-4
-2
0
2
4
6
8
X

Quasi-Newton BFGS

Iterations = 15
Final value [x , y] = [0.7864 , 0.6177]
λ at convergence = 0.1215

$c_2(x) = 1 - 0.7864^2 - 0.3812^2 = 0$



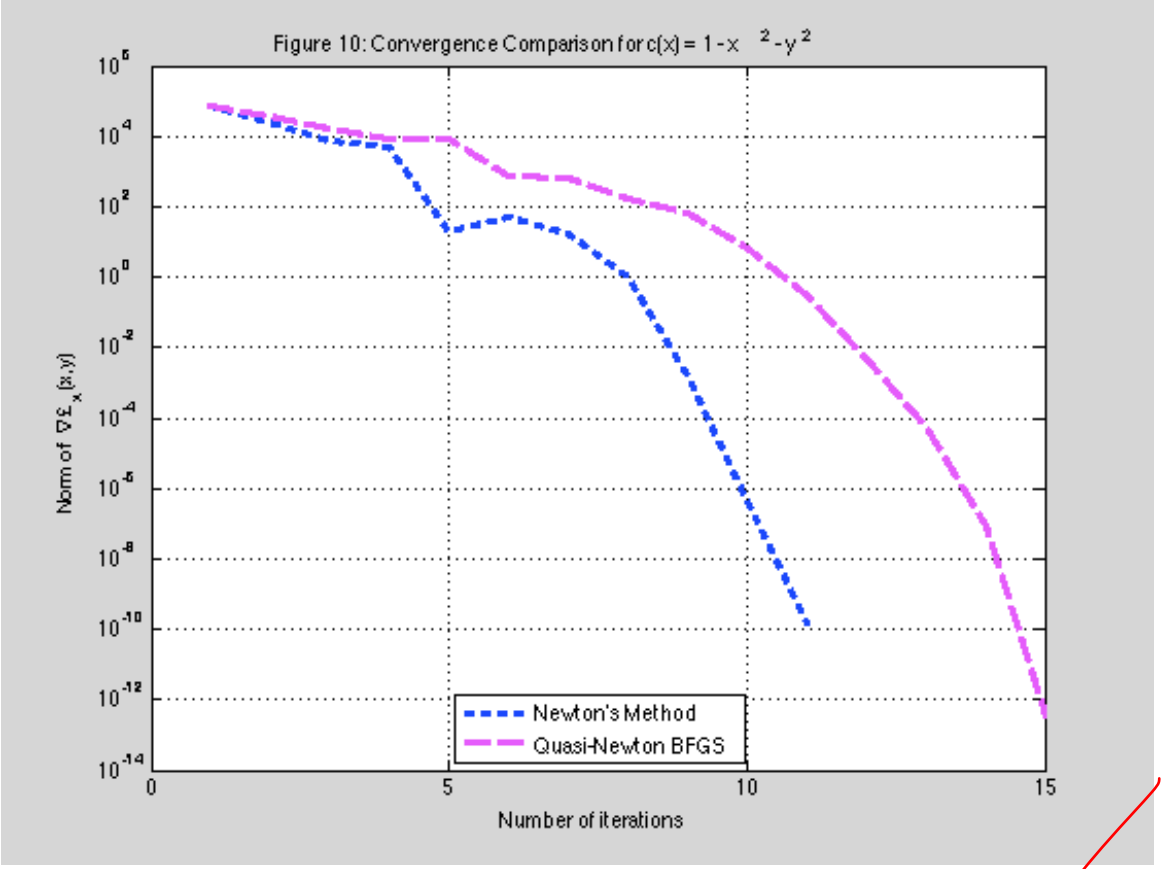


As mentioned during office hours Tuesday October 15th, initially the Hessian was used to initialize the pseudo-hessian approximation matrix B. Choosing the identity matrix caused the algorithm to diverge.

However, this error was determined to be due to a bug in the code related to backtracking, and the problem was subsequently fixed. Specifically, the problem was in the maximum number of iterations of α, i.e. how many ($\alpha_k = \alpha_k \cdot \tau$) iterations were allowed before stopping the backtracking loop: by reducing this number from 1000 to 100, α was not allowed to be reduced too much and the algorithm converged normally.

Notice that the initial search direction is different from Newton's method, which uses the Hessian to determine the optimal p_k.

Comparison



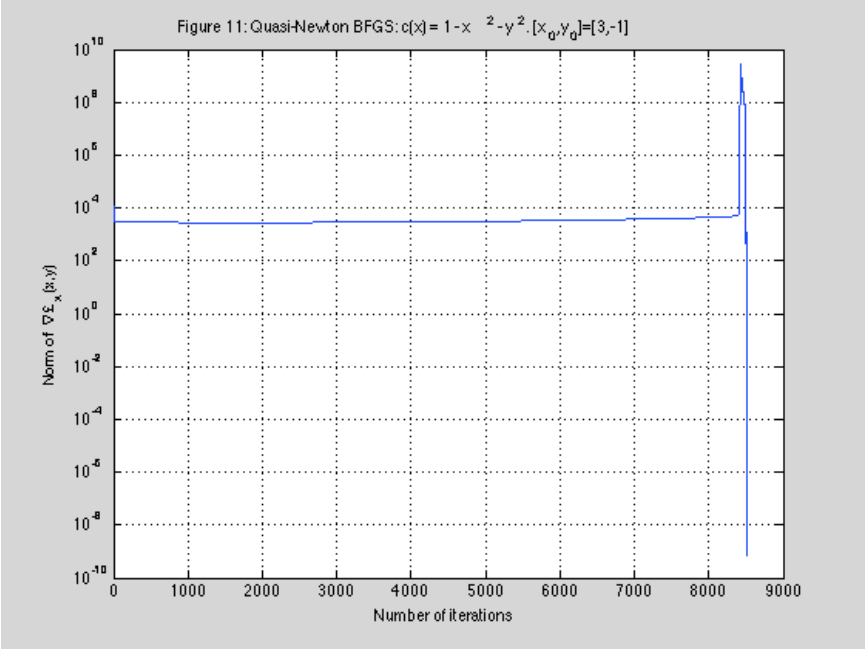
Again, both methods converge at a similar rate, with Newton's method requiring 11 iterations, while the BFGS required 15. They display the same convergence shape.

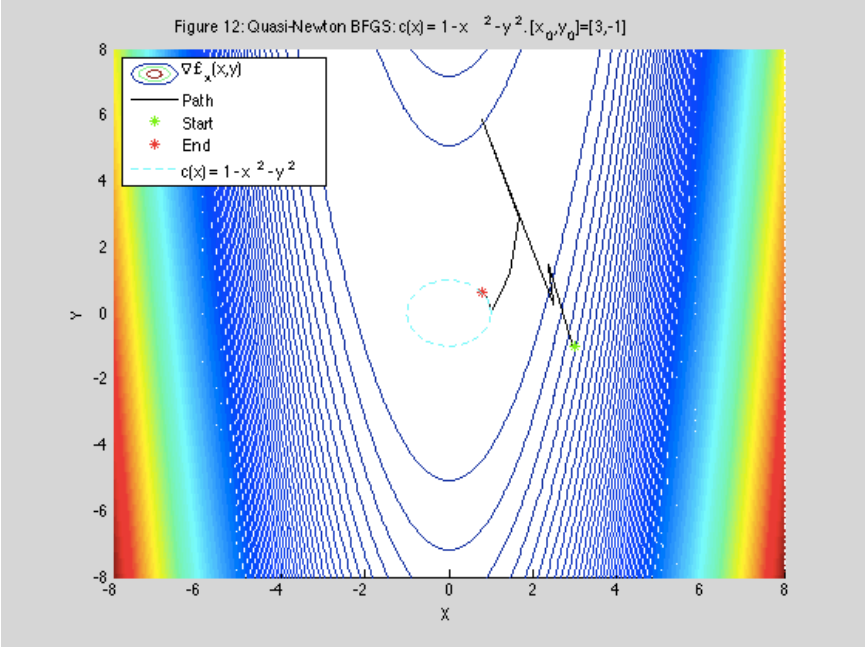
Parameter Variation

In this section, some parameters will be varied to determine their effect on the solution. In all cases, the Quasi-Newton BFGS method with the second constraint was used.

Starting Point

Using $[x_0, y_0] = [3 , -1]$ instead of $[6 , 5]$
Iterations: 8522 instead of 15

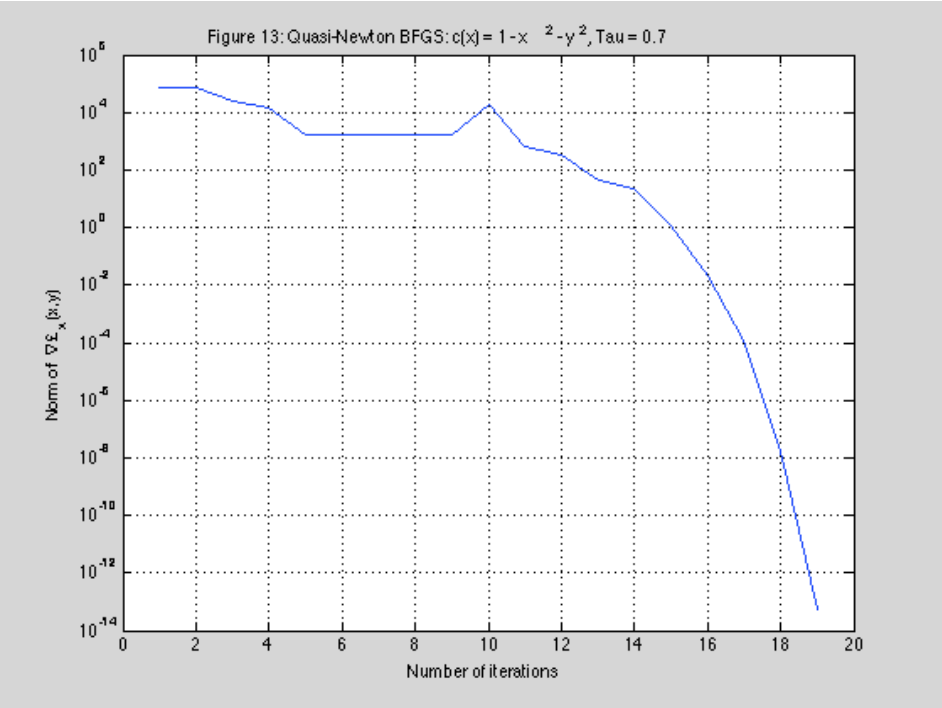


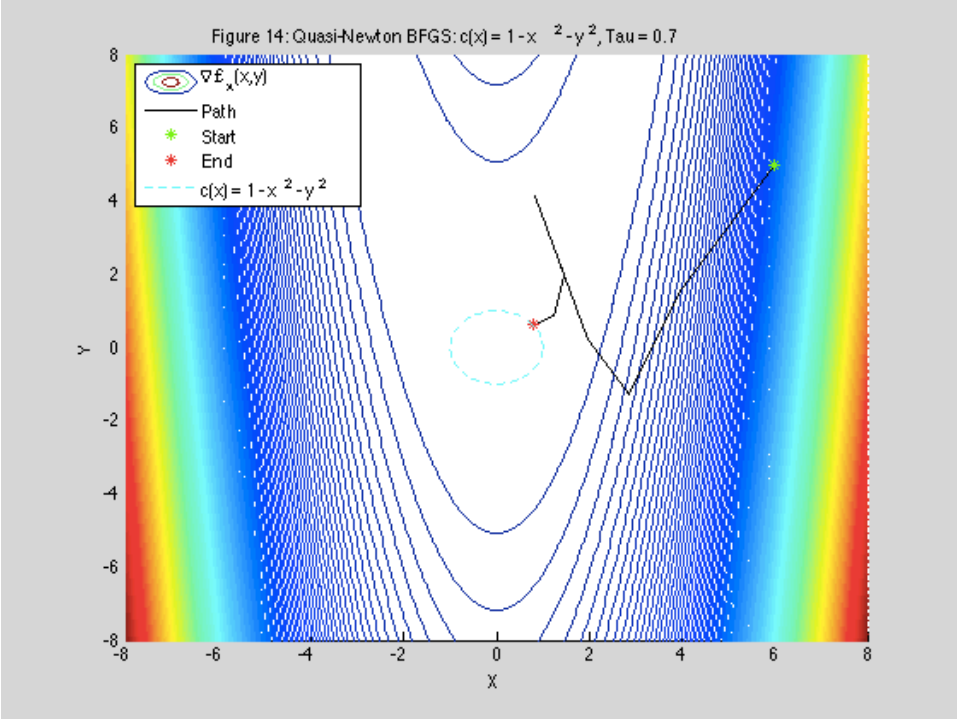


Backtrack reduction

$\tau = 0.7$ instead of 0.9

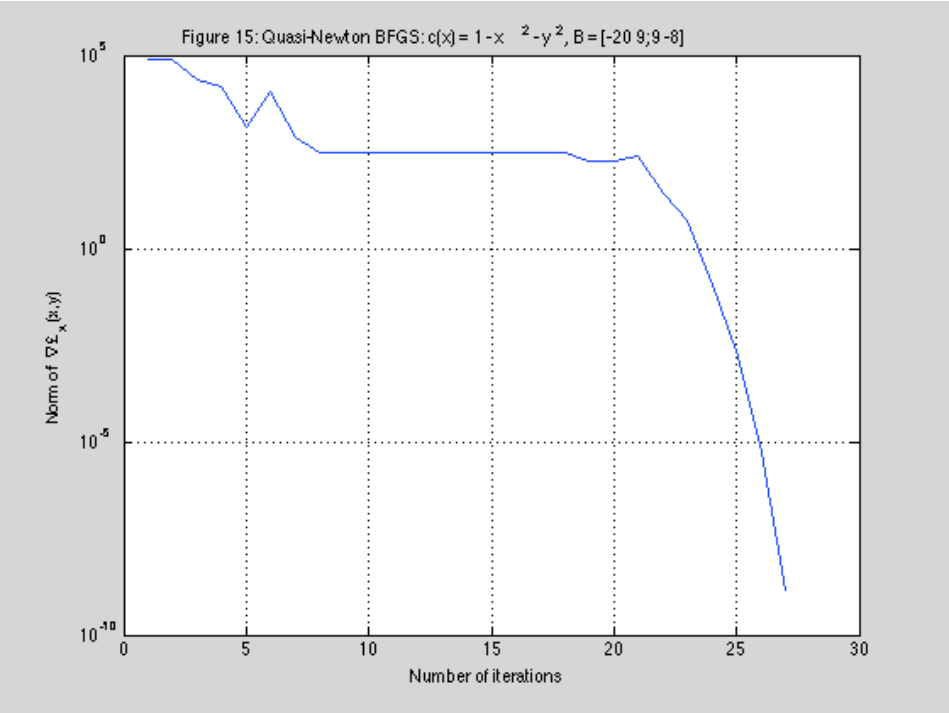
Iterations: 19 instead of 15

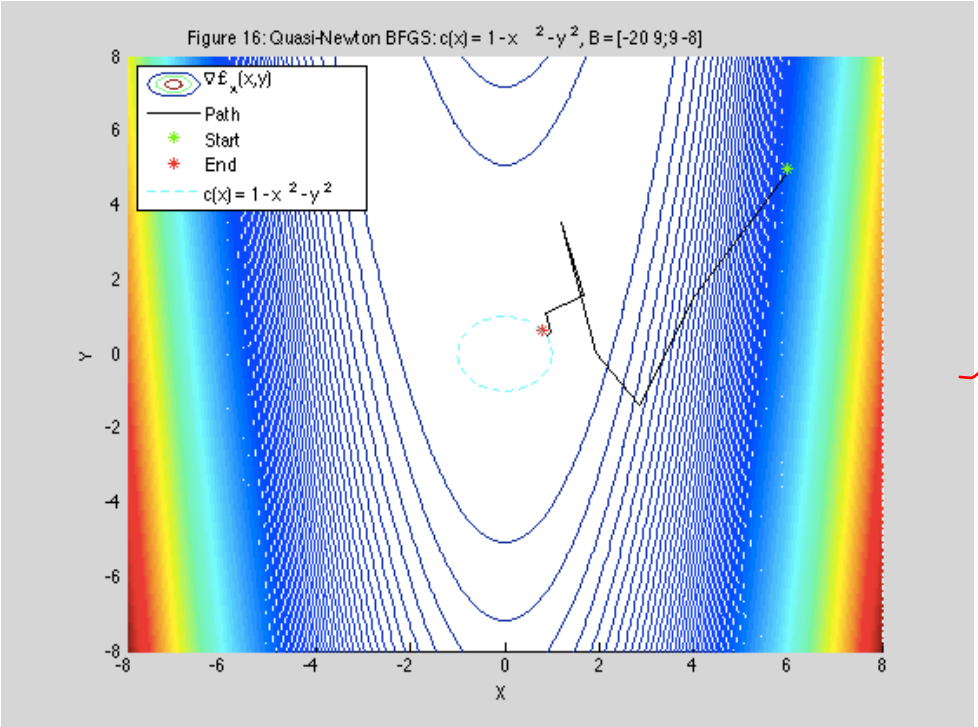




Hessian Approximation Matrix B

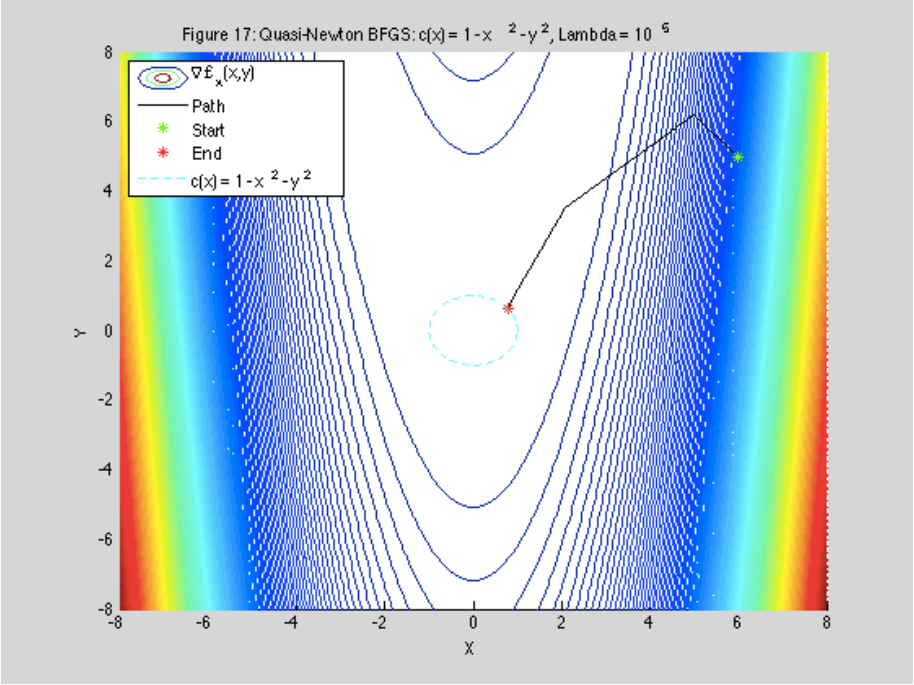
Using $\begin{bmatrix} 20 & 9 \\ 9 & -8 \end{bmatrix}$ instead of $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Iterations: 27 instead of 15



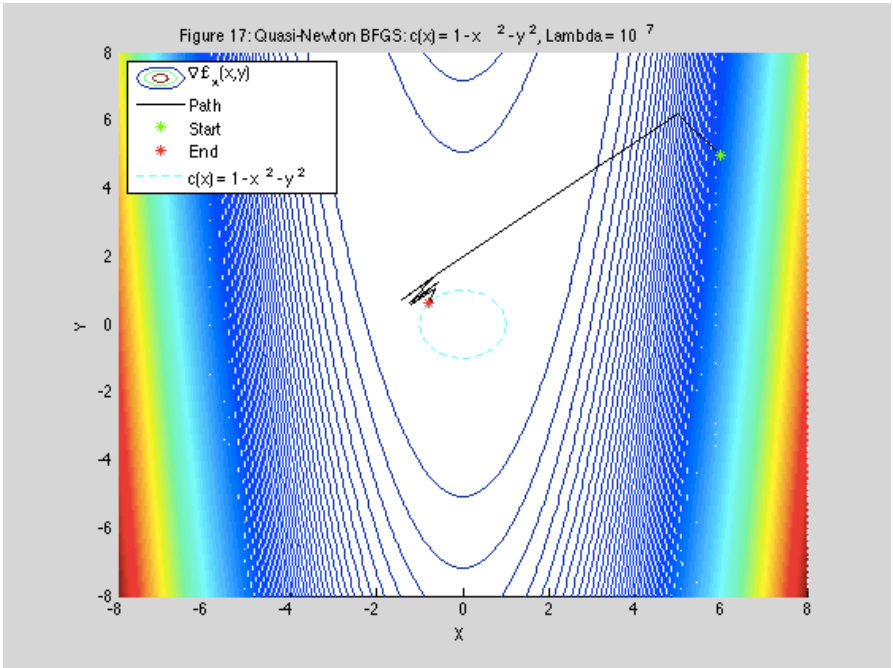


Eigenvalue λ

Using $\lambda = 10^5$ instead of 1
Iterations: 13 instead of 15



Using $\lambda = 10^7$ instead of 1
Iterations: 262 instead of 15



It can be seen that the chosen initial point plays a large role in the convergence of the algorithm. Generally speaking, the closer the initial point is to the solution, the more quickly the BFGS method converges. After some trial and error, it was also determined that choosing an initial point below the x-axis (i.e. negative value of y) caused the number of iterations required for convergence to dramatically increase.

As for the value of the backtracking step reduction (i.e. the value of τ that is used for $\alpha_k = \alpha_k \cdot \tau$), 0.9 was the ideal number, whereas anything less slowed down the convergence. This is to be expected, since the closer this number is to 1, the larger the step size after several reductions. Lowering it to 0.7 caused the number of iterations to convergence to increase from 15 to 19.

Another very important parameter was the initial value of the Hessian approximation matrix B. This matrix determines the initial search direction and initial value of λ. Although the Hessian itself is the ideal initial value (as used in Newton's method), the identity matrix is a reasonable first choice. Using $\begin{bmatrix} 20 & 9 \\ 9 & -8 \end{bmatrix}$ caused the number of iterations to nearly double.

The initial value of λ can have a significant impact on the final solution. For λ less than 10^5, either positive or negative, the number of iterations is relatively constant and the solution is in the first quadrant. For $\lambda > 10^7$, not only does the number of iterations dramatically increase, but the solution ends up being in the second quadrant.

Note that the value of μ was not changed using the formula in the notes because it often caused the algorithms to diverge. Keeping it at a constant positive value allowed the algorithms to converge properly, although changing it from 10 to, say, 10000 had no effect on rate of convergence.

Conclusion

All 4 algorithm-constraint combinations converge in the same order of magnitude, with the Newton methods converging slightly faster than the BFGS. In addition, choosing tau too small, the initial point below the x-axis, or a bad initial guess for the hessian approximation B can significantly reduce the convergence rate for the BFGS method.

Appendix – Matlab Code

For the sake of convenience, only the code for the Quasi-Newton BFGS method with the second constraint is included here.

```
function bfgs_c2()
clearvars
clc

THEX = 0;
THEY = 0;
CONV = 0;

tol = 10^(-8);
limit = 10000;

nu = 10^(-4);
tau = 0.9;
rho = 0.5;
mu = 10;

x = 3;
y = -1;
x_initial = x;
y_initial = y;
lambda = 1;
lambda_old = 1;
B_old = eye(2);
B_new = eye(2);

p_k = [1;1];
l_k = 1;

k = 0;

while(1)

    k = k + 1;
    if (k == limit)
        break
    end

    epsilon = norm(grad_f(x,y)-lambda*grad_c(x,y));

    CONV(k) = epsilon;
    THEX(k) = x;
    THEY(k) = y;

    if ( epsilon < tol )
        break
    end

    % initiate
```

```
    F = [grad_f(x,y) - lambda * grad_c(x,y) ; c(x,y)];
    F_prime = [B_new, -grad_c(x,y) ; grad_c(x,y)', 0];
    P_K = inv(F_prime)*(-F);
    p_k = [P_K(1);P_K(2)];
    l_k = P_K(3);

    % backtrack
    alpha = 1;
    j = 0;
    x_new = x;
    y_new = y;
    while(1)
        j = j+1;
        if (j == 100)
            break
        end
        x_new = x_new + alpha * P_K(1);
        y_new = y_new + alpha * P_K(2);
        phi1 = f(x_new,y_new) + mu*abs(c(x_new,y_new));
        phi0 = f(x,y) + mu * abs(c(x,y));
        D = -(p_k'*grad_f(x,y))-mu*abs(c(x,y));
        RHS = phi0 + nu*alpha*D;
        if (phi1 <= RHS)
            break
        end
        alpha = alpha * tau;
    end

    % update
    x_old = x;
    y_old = y;
    lambda_old = lambda;
    x = x_old + alpha * P_K(1);
    y = y_old + alpha * P_K(2);
    lambda = lambda_old + alpha*P_K(3);

    % bfgs
    B_old = B_new;
    s = [x - x_old; y - y_old];
    temp = (grad_f(x,y)-lambda*grad_c(x,y)) - (grad_f(x_old,y_old)-
lambda*grad_c(x_old,y_old));
    if (s'*temp >= 0.2 * s'*B_old*s)
        theta = 1;
    else
        theta = (0.8*s'*B_old*s)/(s'*B_old*s-s'*temp);
    end
    r = theta * temp + (1-theta)*B_old*s;
    B_new = B_old - (B_old*s*s'*B_old)/(s'*B_old*s) + (r*r')/(s'*r);


end
x
y
lambda
k
```

```
semilogy(CONV)
xlabel('Number of iterations')
ylabel('Norm of \nablaf_x(x,y)')
title('Figure 8: Quasi-Newton BFGS: c(x) = 1 - x^2 - y^2')
grid on
size = 8;
a = linspace(-size,size);
b = linspace(-size,size);
[A,B] = meshgrid(a,b);
C = (1-A).^2+100*((B-(A.^2)).^2);
levels = 100:10:10;
figure
hold on
contour(A,B,C,200)
plot(THEX,THEY,'Black')
plot(x_initial,y_initial,'g*')
plot(x,y,'r*')
[C,D]=circle1;
plot(C,D,'c--')
legend('\nablaf_x(x,y)','Path','Start','End','c(x) = 1 - x^2 -
y^2','Location','NorthWest')
xlabel('X')
ylabel('Y')
title('Figure 9: Quasi-Newton BFGS: c(x) = 1 - x^2 - y^2')

dlmwrite('bfgs_c2.txt',CONV);

end


function f1 = f(x,y)
    f1 = (x - 1)^2 + 100*(- x^2 + y)^2;
end

function grad_f1 = grad_f(x,y)
    grad_f1 = [2*x - 400*x*(- x^2 + y) - 2 ; - 200*x^2 + 200*y];
end

function hess1 = hess(x,y)
    hess1 = [ 1200*x^2 - 400*y + 2, -400*x; -400*x, 200];
end

function c1 = c(x,y)
    c1 = 1 - x^2 - y^2;
end

function grad_c1 = grad_c(x,y)
    grad_c1 = [-2*x ; -2*y];
end

function [C,D] = circle1()
    th = 0:pi/50:2*pi;
    C = cos(th);
    D = sin(th);
end
```