MAT 480 Programming Project

Diego G. Avalos

Spring 2017

Introduction

The purpose of this project is to utilize three algorithms for minimizing specific scalar functions of two variables $f : \mathbb{R}^2 \to \mathbb{R}$. Each algorithm takes an initial guess of the minimizer and outputs a $K+1$ long sequence $\{x_k\}_{k=0}^{K} \subset \mathbb{R}^2$ which converges[1] to the actual minimizer point $x^* \in \mathbb{R}^2$. We then analyze the behavior of the convergence from the sequences we obtain from Matlab. The functions we study are

$$f_1(x,y) = \frac{1}{4}x^4 - \frac{1}{3}x^3 - 3x^2 + 1 + y^4$$

and

$$f_2(x,y) = -\exp(-x^2 - y^2).$$

The algorithms we will use are the following

1. Newton's method for minimization (NM).
2. The method of steepest descent (SD).
3. Broyden's method (BM).

Moreover, for each method we plot three figures, namely, iteration k vs root approximation (i.e., the approximation of the minimizer), x vs y components at iteration k, and iteration k vs $f(x_k)$.
We will give a brief explanation of our three methods in the next section.

Description of the Methods

Method 1. Newton's Method for minimization is similar to Newton's method for solving systems. Let $x_0 \in \mathbb{R}^n$ be our initial guess of a minimizer of $f : \mathbb{R}^n \to \mathbb{R}$, so to use NM we consider the gradient ∇f, and we denote the Hessian of f by H_f. Hence, if we seek to obtain a $K+1$ long sequence of points (where x_0 is our initial guess), which converge to the actual minimizer x^*, we can find such sequence recursively by the formula

$$x_{k+1} = x_k - H_f^{-1}(x_k)\nabla f(x_k), \quad 0 \le k \le K-1.$$

It is important to know that NM has a descent direction, so our sequence terms converge to x^*.

[1] A finite sequence does not converge, so we are abusing our language here.

Method 2. To use the method of steepest descent on a function $f : \mathbb{R}^n \to \mathbb{R}$ with initial guess $x_0 \in \mathbb{R}^n$, we calculate ∇f. Since we seek to find a $K+1$ long sequence of points that converge to the minimizer x^* (where x_0 is our initial guess), we omit the stopping rule. Furthermore, instead of computing and minimizing at the kth iteration the scalar function $\varphi(t_k) = f(x_k - t_k \nabla f(x_k))$ over $t_k \geq 0$, we use the fixed value $t_k = 0.1$ at each iteration. Hence our SD algorithm becomes the recursive formula

$$x_{k+1} = x_k - 0.1 \nabla f(x_k), \quad 0 \leq k \leq K-1.$$

Note that our version of SD will not necessarily move in orthogonal steps because we modified it.

Method 3. To use Broyden's method on $f : \mathbb{R}^n \to \mathbb{R}$, we seek to solve the system $\nabla f = \mathbf{0}$ with initial guess $x_0 \in \mathbb{R}^n$ and initial $n \times n$ constant matrix D_0. To obtain a $K+1$ long sequence of points that converge to the minimizer x^* (where x_0 is our initial guess), we apply the following algorithm at iteration k, where $0 \leq k \leq K-1$

Step 1. Compute $x_{k+1} = x_k - D_k^{-1} \nabla f(x_k)$.

Step 2. Set $d_k = x_{k+1} - x_k$, and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.

Step 3. Update

$$D_{k+1} = D_k + \frac{(y_k - D_k d_k) d_k^T}{d_k^T d_k}.$$

It is important to note that BM does not necessarily go in a descent direction.

Analysis of Results

Our initial guess points for f_1 and f_2 are $x_0 = (2, 1)$ and $x_0 = (0.2, 0.4)$, respectively. Furthermore, we are going to use $K = 20$ iterations for each function and each method in this project. Our Matlab function `mat480project` takes inputs of the form `(K,x0,nFunction,nMethod)`, where `K` is the number of iterations, `x0` is an appropriate x_0, `nFunction` is 1 for f_1 and 2 for f_2, and `nMethod` is 1 for NM, 2 for SD, and 3 for BM. For each of the two functions, we apply our three methods. This leaves us with the following six inputs

Input 1. NM for f_1: `mat480project(20,[2;1],1,1)` Input 4. NM for f_2: `mat480project(20,[0.2;0.4],2,1)`

Input 2. SD for f_1: `mat480project(20,[2;1],1,2)` Input 5. SD for f_2: `mat480project(20,[0.2;0.4],2,2)`

Input 3. NM for f_1: `mat480project(20,[2;1],1,3)` Input 6. BM for f_2: `mat480project(20,[0.2;0.4],2,3)`

Next, we present and analyze our plots generated by Matlab for each input one by one.

Figure 1: Input 1 Figure 1

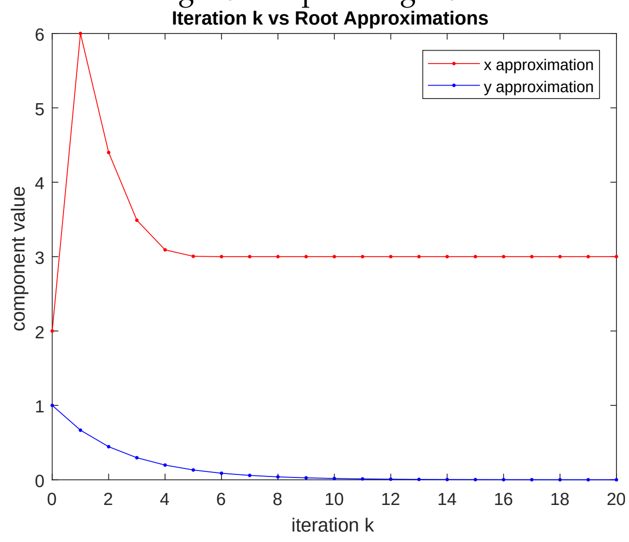


Figure 2: Input 1 Figure 2

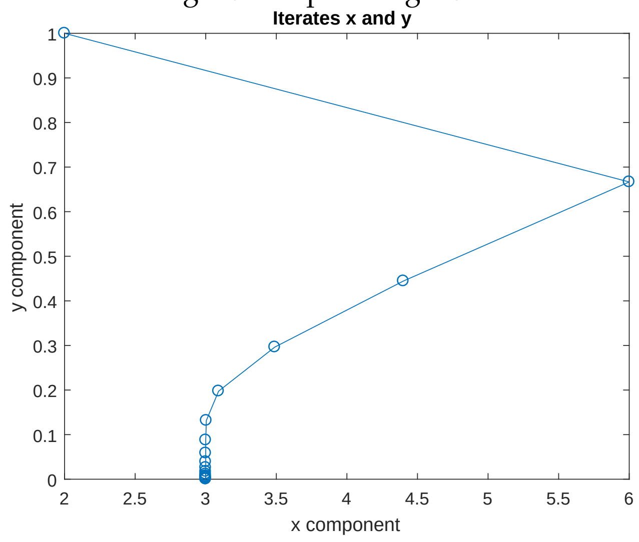


Figure 3: Input 1 Figure 3

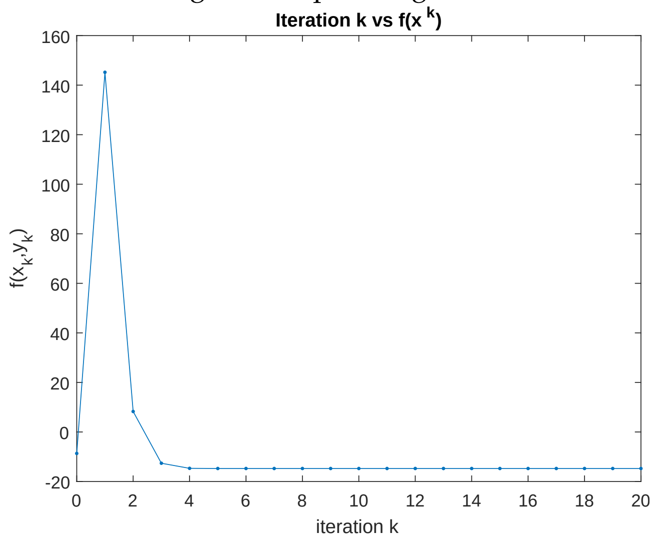


Figure 4: Input 2 Figure 1

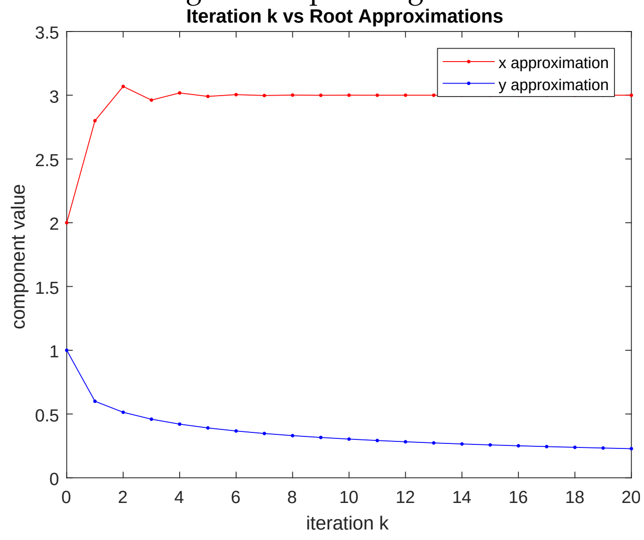


Figure 5: Input 2 Figure 2

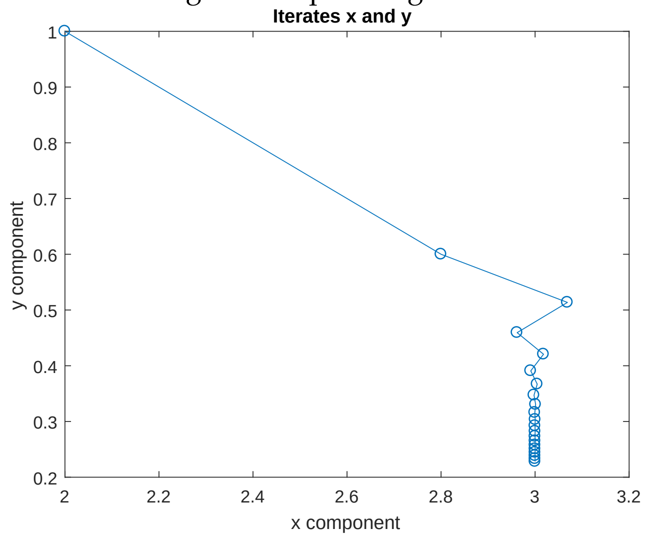


Figure 6: Input 2 Figure 3

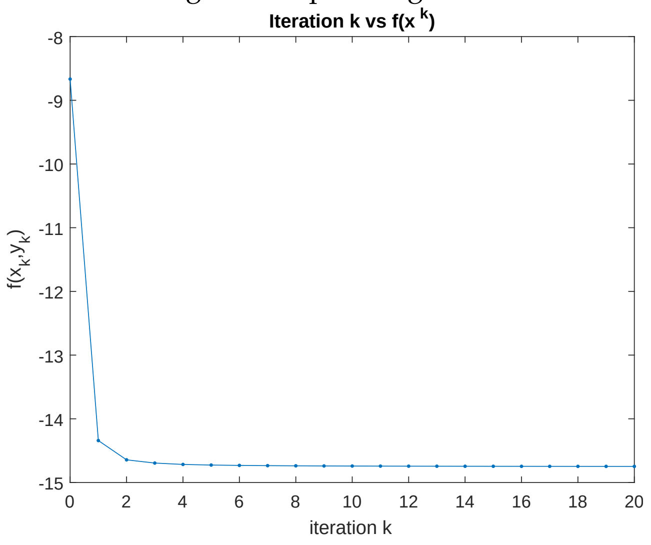


Figure 7: Input 3 Figure 1

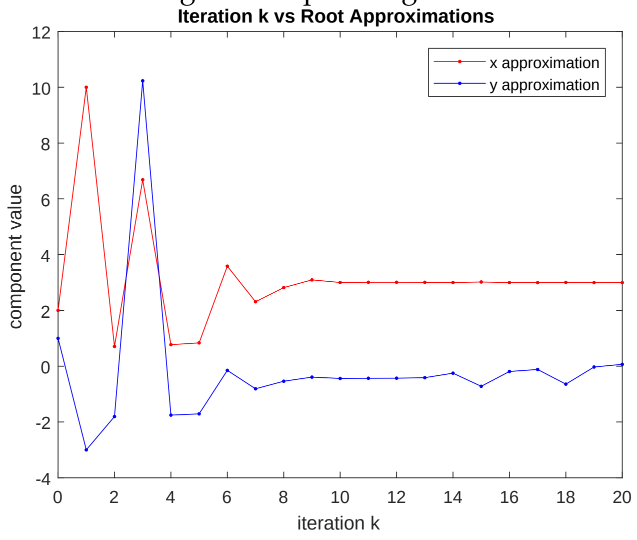


Figure 8: Input 3 Figure 2

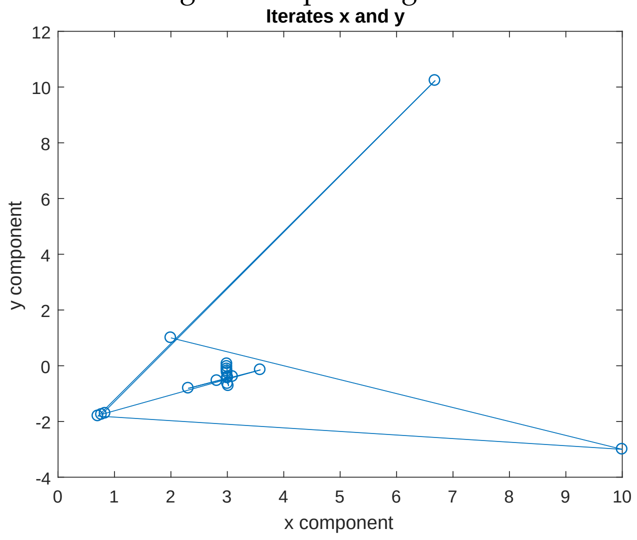


Figure 9: Input 3 Figure 3

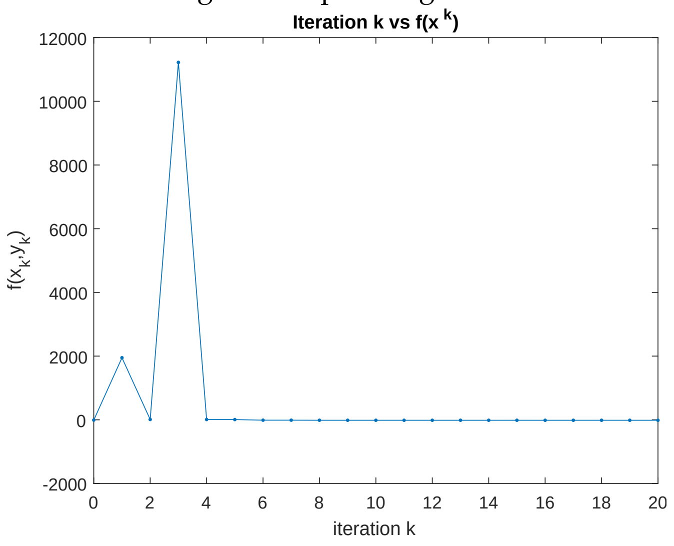


Figure 10: Input 4 Figure 1

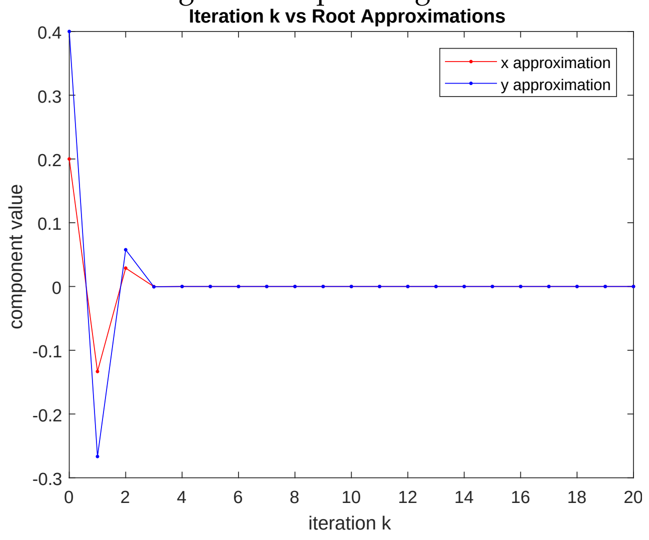


Figure 11: Input 4 Figure 2

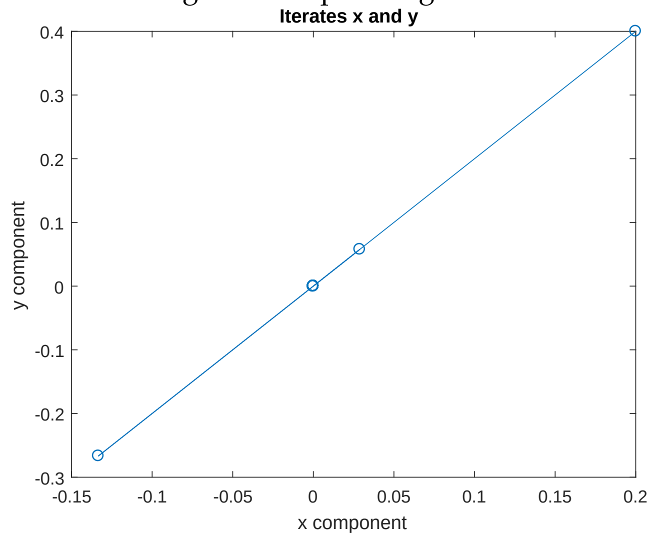


Figure 12: Input 4 Figure 3

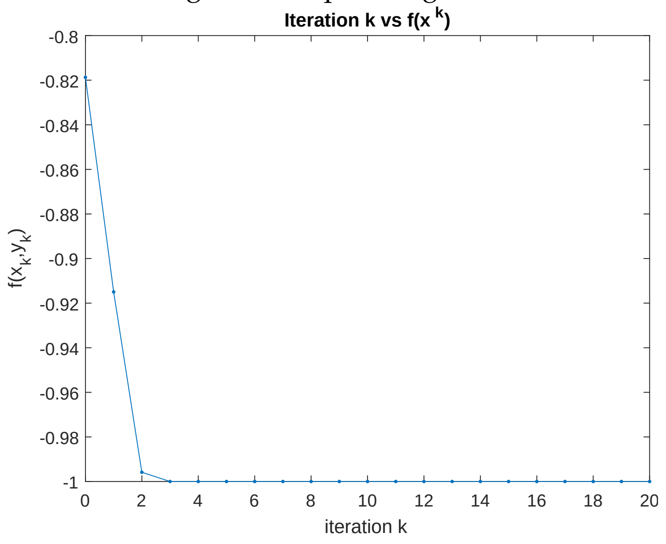


Figure 13: Input 5 Figure 1

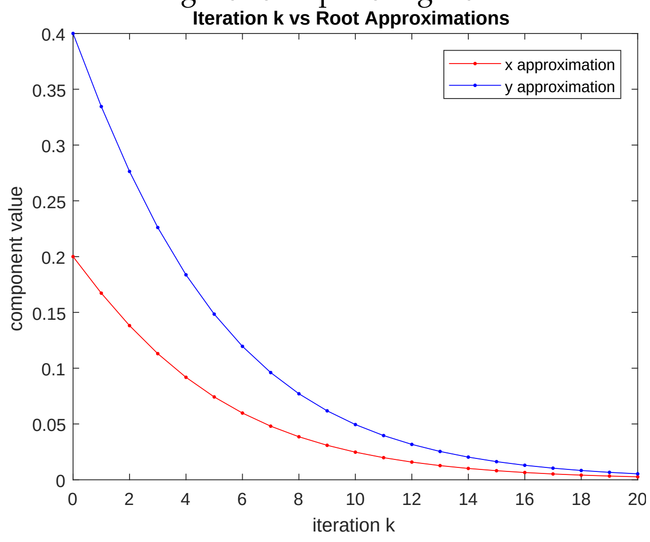


Figure 14: Input 5 Figure 2

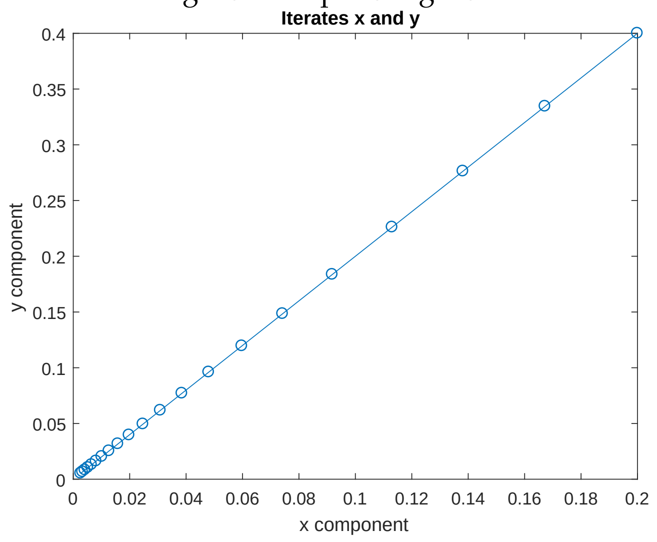


Figure 15: Input 5 Figure 3

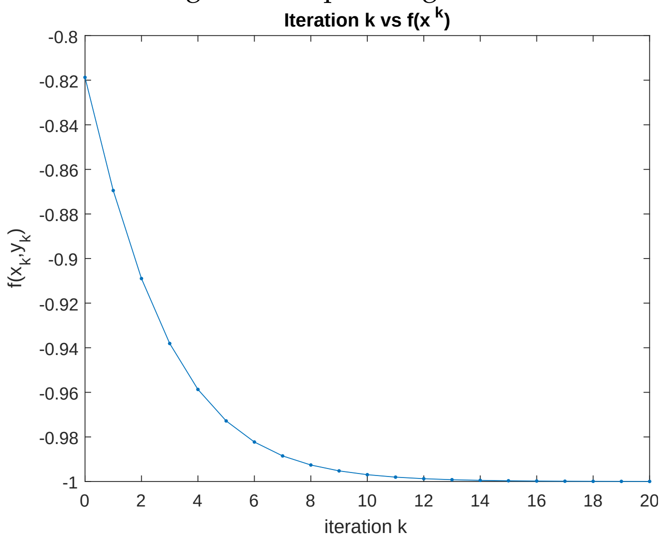


Figure 16: Input 6 Figure 1

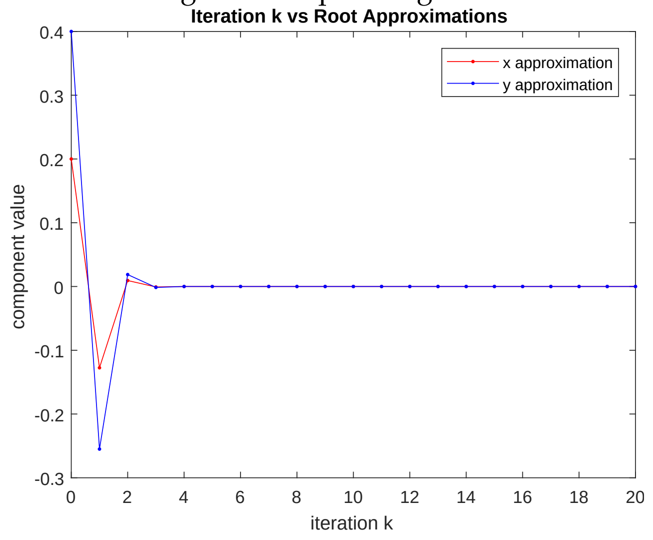


Figure 17: Input 6 Figure 2

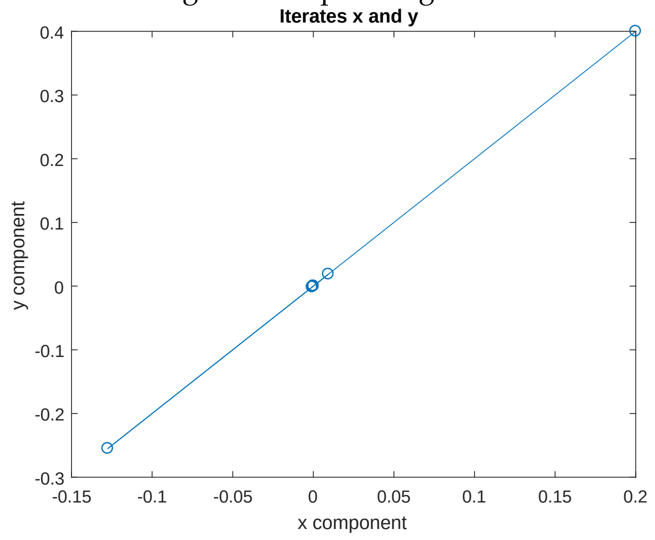
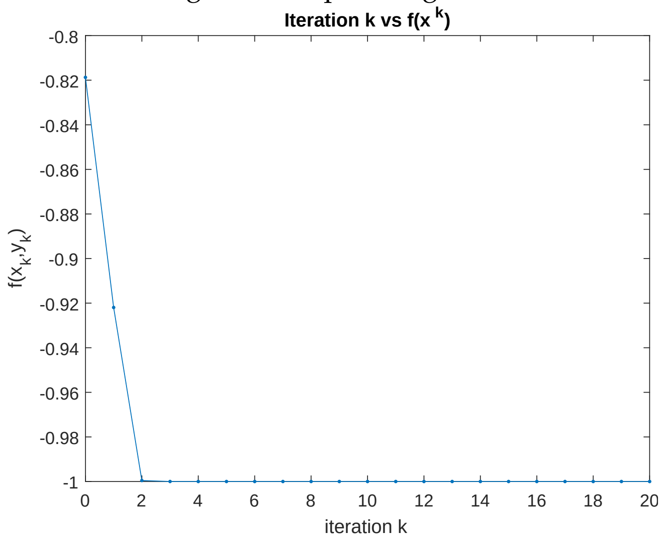


Figure 18: Input 6 Figure 3



We first analyze our results from using our algorithms with f_1.

NM From figure 1, observe that the x values converged quickly to 3 after iteration 3, whereas the y values converged quickly to 0 from the zeroth iteration. From figure 2, it is evident that the vectors approach very quickly to x^* after the 5th iteration. Figure 3 tells us that the f values increase quickly at the first iteration, but then quickly decrease and approach to the minimum value, which we can only tell is between -20 and -10.

SD Figures 4 and 6 suggest that the convergence is more steady, that is, our x and y values in our sequence are closer together without making high jumps as in NM and clearly converge to $(3, 0)$. Figure 6 in particular shows steepest descent in action and it allows us to see a better approximation for $f(x^*)$ than NM (≈ -15). However, figure 5 does not display orthogonal steps; this is because of our constant value t_k in the algorithm.

BM Broyden's method shows a more chaotic convergence, as figure 7 displays multiple jumps in the x and y values, and figure 9 makes a jump in the f values up to over 10000, and quickly goes back to around zero, which does not allow us to make a confident approximation of $f(x^*)$. We can see from figure 8 that the vectors of our sequence go around x^* for a while and eventually get very close to the minimizer.

Secondly, we present the analysis for the methods applied to f_2.

NM From figure 10, it is clear that our x and y values both approach to 0 very rapidly after iteration 2. Also, figure 12 clearly tells us that eventually our minimum value $f(x^*)$ approaches -1. Our approximate vectors in figure 11 converge and all lie on the same line.

SD Our SD figures display a rather aesthetic behavior of convergence. Figures 13 and 15 suggest that the rate of convergence of our x and y values as well as $f(x_n)$ is similar to that of e^{-t}. Furthermore, the vectors in figure 14 approach to the origin from only one direction along a line.

BM Our Broyden's method plots look very similar to NM for this function. Recall that for f_1 BM displayed a more chaotic behavior of convergence. For the current input, BM has a very steady convergence after iteration 2.

We end this section by taking a look at an additional function f_3 defined by

$$f_3(x, y) = \frac{x}{\log x} + \int_0^{2y} \frac{\sin t}{t}\, dt.$$

The first reason I picked this function is because I am obsessed with $\frac{x}{\log x}$ because it represents the approximate number of primes up to x for large x, and the antiderivative of the sinc function $\frac{\sin t}{t}$ does not exist. The second reason is that if I pick `x0=[3;3]`, then x has to approach the local minimizer of $\frac{x}{\log x}$, namely e, and $2y$ has to approach the local minimizer of the integral $\int_0^{2y} \frac{\sin t}{t}\, dt$, namely 2π (so y has to approach π). Therefore, we will study the following additional inputs using f_3

Input 7. NM for f_3: `mat480project(20,[3;3],3,1)`

Input 8. SD for f_3: `mat480project(20,[3;3],3,2)`

Input 9. BM for f_3: `mat480project(20,[3;3],3,3)`

where `nFunction = 3` corresponds to the function f_3. For instance, consider the following plots generated by Matlab.

Figure 19: Input 7 Figure 1

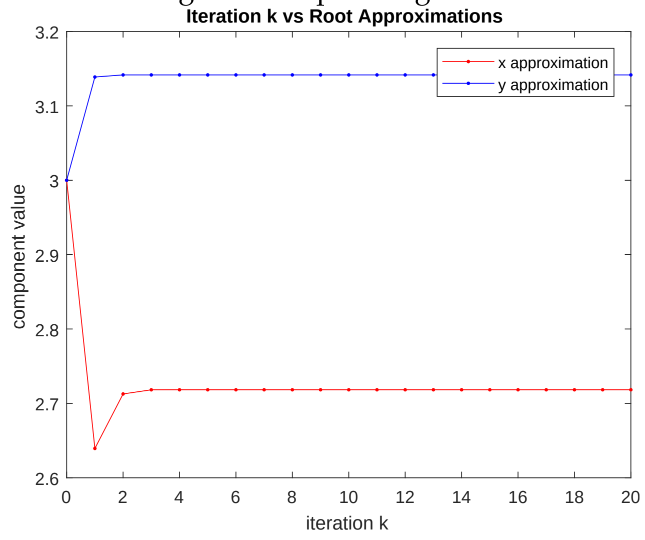


Figure 20: Input 7 Figure 2

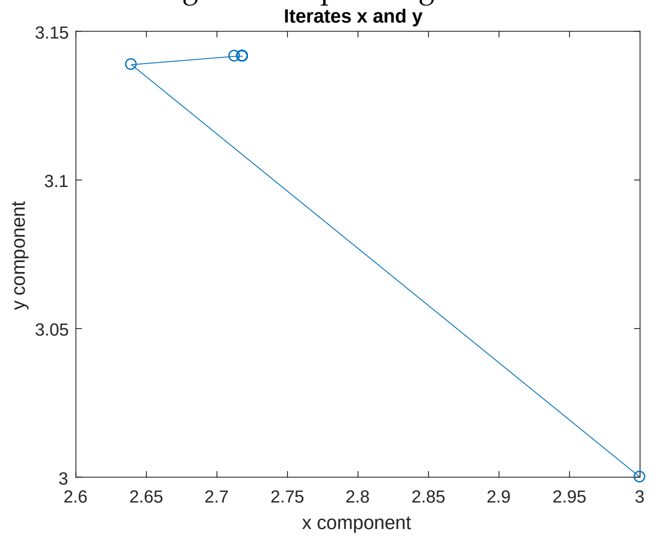


Figure 21: Input 7 Figure 3

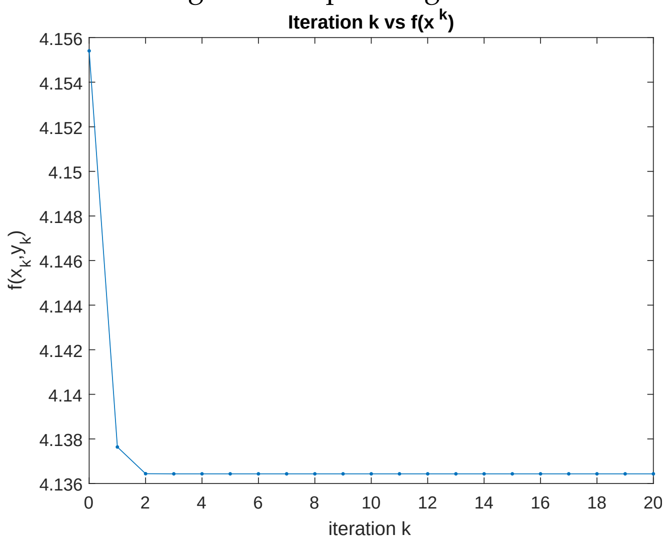


Figure 22: Input 8 Figure 1

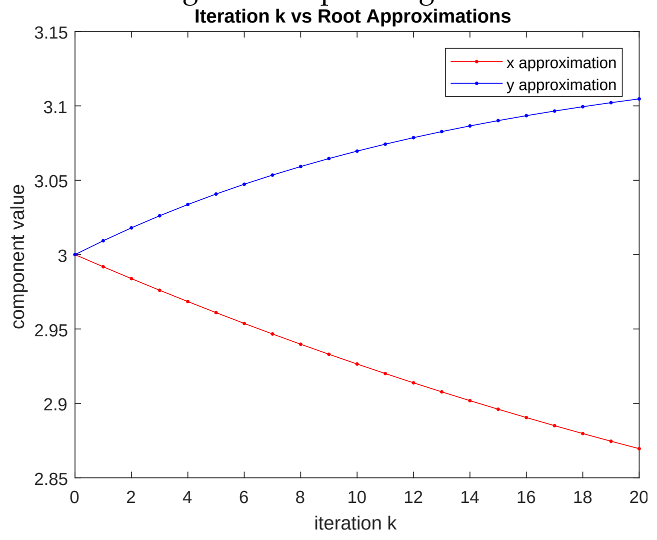


Figure 23: Input 8 Figure 2

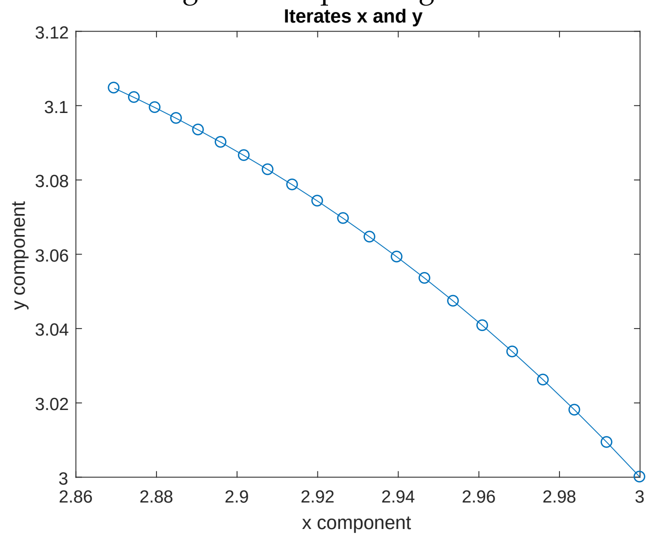


Figure 24: Input 8 Figure 3

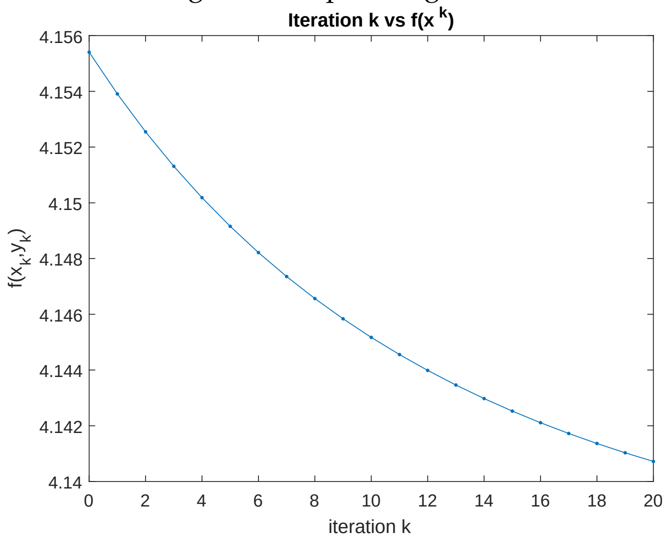


Figure 25: Input 9 Figure 1

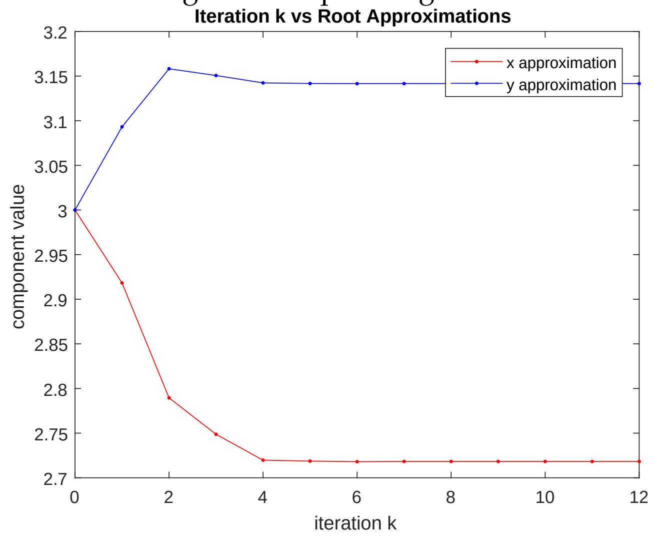


Figure 26: Input 9 Figure 2

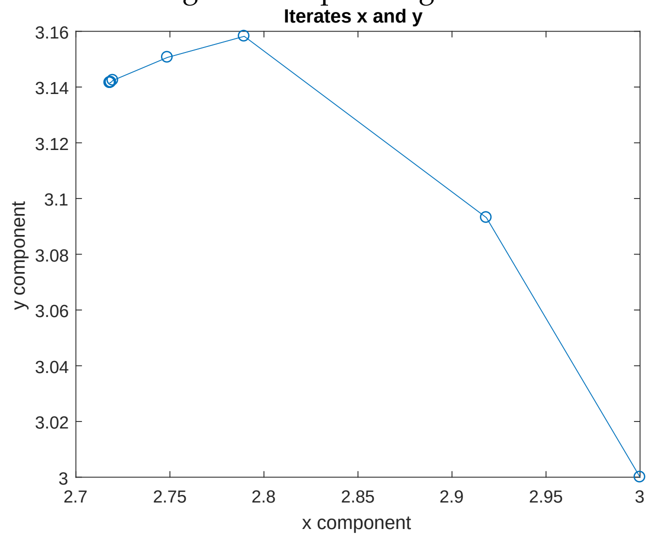
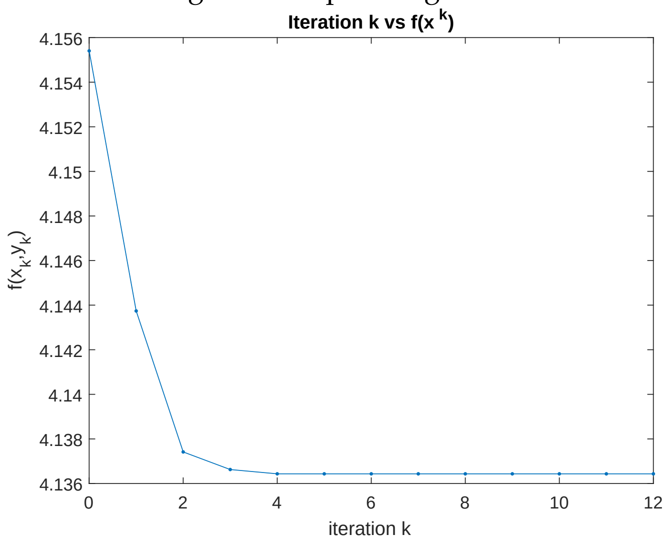


Figure 27: Input 9 Figure 3



We now analyze our results for f_3.

NM After iteration 2, figure 19 and 20 tell us that x and y approach to e and π quickly. From figure 21, we can clearly see that the minimum f value is around 4.136.

SD Steepest descent yet again shows us a clear harmonious pattern of convergence in figures 22 to 24. However, the convergence is slower than NM because in figure 22, x gets above 2.85 and y is barely above 3.1, which are worse approximations than NM (contrast with figure 19). Also, the approximation for $f(x^*)$ is worse than NM, namely 4.141.

BM From figure 25 and 26, BM goes in small jumps but eventually shows a convergence faster than SD. Figure 27 gives the same good approximation as NM. Observe that, even though our inputs for this function is `K=20`, our plots only display 12 iterations. This is because our algorithm eventually gets matrices D_k that are not invertible, so I included `M(:,13)` in my code (see Appendix) to obtain the x and y values calculated at the last iteration. Here is what I got

```
ans = [2.7183;3.1416]
```

which is a number very close to $x^* = (e, \pi)$.

Conclusion

For the functions considered in this paper, the convergence of our sequences in BM are always as good as those obtained from NM. However, BM sometimes displays chaotic looking steps in the xy plane (see figure 8) and at other times it has a precision issue due to its more complicated algorithm and the higher number of operations involved, so we could end up obtaining fewer iterations than intended (see figures 25 and 27). On the other hand, SD has very smooth patterns of convergence (see figures 13 to 15 and 22 to 24). The only issue is that it performs a slower convergence than NM. Therefore, NM wins in clear pattern and convergence speed, whereas SD clearly wins in smooth convergence, so my preferred method is NM.

Appendix

Figure 28: Source Code

```
function  mat480project(K,x0,nFunction,nMethod)
%%%%%%%%%%%%%%%%%%%%mat480project%%%%%%%%%%%%%%%%%%%%%%%
%Inputs
%K = number of iterations
%x0 = initial 2x1 vector
%nFunction = 1 for the polynomial, 2 for the exponential
%nMethod = 1 for NM, 2 for SD, 3 for BM
%Outputs three plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
syms x y
%Choose Function
if nFunction == 1
    f(x,y) = (1/4)*x.^4 - (1/3)*x.^3 - 3*x.^2 + 1 + y.^4;
elseif nFunction == 2
    f(x,y) = -exp(-x.^2-y.^2);
elseif nFunction ==3
    syms t
    f(x,y)=x/log(x)+int(sin(t)/(t),0,2*y);
end

%Hessian, gradient, D0
H(x,y) = [diff(f,x,2), diff(f,x,y); diff(f,x,y), diff(f,y,2)];
G(x,y) = [diff(f,x); diff(f,y)];
D = [1,0;0,1];
M = zeros(2,K+1); %the x & y values are stored here
M(1,1) = x0(1); M(2,1) = x0(2);

%Choose Method
if nMethod == 1 %Use Newton's
    for j = 1:K
        Hinv = inv(H(M(1,j),M(2,j)));
        M(:,j+1) = M(:,j) - Hinv*G(M(1,j),M(2,j));
    end
elseif nMethod == 2 %Use Steepest Descent with tk=.1
     for j = 1:K
         M(:,j+1) = M(:,j) - 0.1*G(M(1,j),M(2,j));
     end
elseif nMethod == 3 %Use Broyden's
    for j = 1:K
        M(:,j+1) = M(:,j) - inv(D)*G(M(1,j),M(2,j));
        d = M(:,j+1) - M(:,j);
        y = double(G(M(1,j+1),M(2,j+1)) - G(M(1,j),M(2,j)));%we use double here because it improves the precision for f2
        D = D + (y - D*d)*d'/(d'*d);
    end
end
M(:,13)
%plots
figure(1)
plot(0:K,M(1,1:(K+1)), '.-r', 0:K, M(2,1:(K+1)),'.-b');
legend('x approximation','y approximation');
xlabel('iteration k');ylabel('component value');
title('Iteration k vs Root Approximations');
figure(2)
plot(M(1,1:(K+1)),M(2,1:(K+1)),'-o');
xlabel('x component');ylabel('y component');
title('Iterates x and y ');
figure(3)
plot(0:K,f(M(1,1:(K+1)),M(2,1:(K+1))),'.-');
xlabel('iteration k');ylabel('f(x_k,y_k)');
title('Iteration k vs f(x^k)');
end
```