Damian Franco

CS 375 – 001

Homework #12:

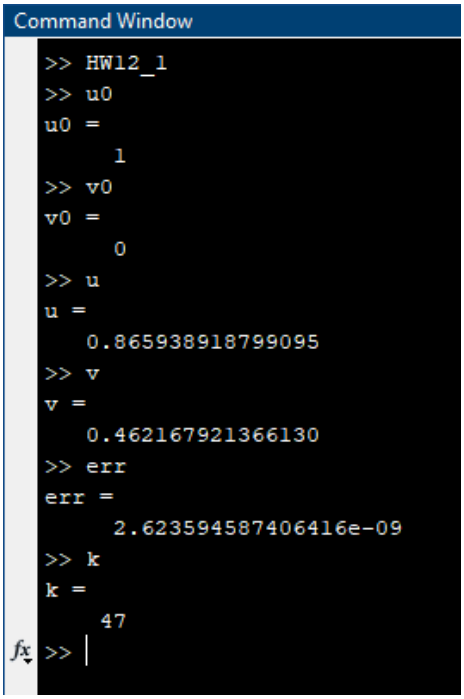
1) For number 1, we were given a nonlinear system to solve for two other solutions using Newton's method. We already are given one solution (u, v) = (*1, 1*) for the nonlinear system. Knowing that we already have the solution, I needed to set some new values for our initial u and v. First, I set up the MATLAB function *nonlinearNewt.m*, which uses Newton's method to estimate the solution. The function is based off the last example in the root4 notes. Next, I found the Jacobian matrix of my system, this is listed in my MATLAB function assigned to the variable *DF*. Lastly, I made some educated guesses on my initial u and v values and I ran the code. The initial guesses are below, along with the outcome of each root found (reported to the 8-digit accuracy). Both MATLAB functions used (*nonlinearNewt.m & HW12_1.m)* and all MATLAB outputs are shown below. Overall, the one thing that intrigued me was the number of k iterations on the first initial guess $[u_0, v_0] = [1,0]$. There were 47 iterations, which seems quite high for the Newton approach. On the second guess $[u_0, v_0] = [0,1]$, there was only 14 iterations and a lower error. Both points found seem very plausible for possible solutions.

   u0 = 1, *v0* = 0:
   - u = 0.86593892
   - v = 0.46216792
   - *err* = 2.623594587406416e-09
   - k = 47

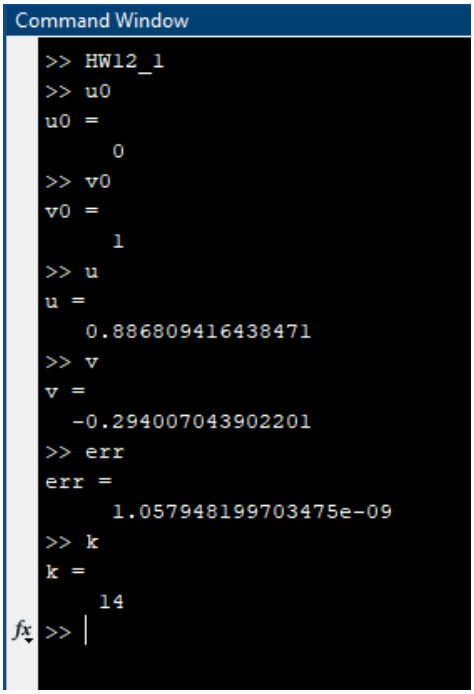
   u0 = 1, *v0* = 2:
   - u = 0.88680942
   - v = -0.29400704
   - *err* = 1.057948199703475e-09
   - k = 14

MALTAB OUTPUT:

u0 = 1, v0 = 0



u0 = 0, v0 = 1



SOURCE CODE:

HW12_1.m

```
% function HW12_1.m
%
% This function sets up all testing for
% question 1 on HW12 and calls the
% MATLAB function nonlinearNewt.m
format compact;
format long
tol = 1e-8;
u0 = 1; % change to initial guess u
v0 = 0; % change to initial guess v
[u, v, err, k] = nonlinearNewt(tol, u0, v0);
```

nonlinearNewt.m

```
% function nonlinearNewt.m
% This takes the newton method approach
% to approximate for #1 on HW12
%
% 6u^3 + uv - 3v^3 - 4 = 0
% u^2 - 18uv^2 + 16v^3 + 1 = 0
%
% function [u, v, err, k] = nonlinearNewt(tol, u0, v0)
function [u, v, err, k] = nonlinearNewt(tol, u0, v0)
err = 1000;
maxk = 1000;
k = 0;
u = u0;
v = v0;
while ((err > tol) && (k < maxk))
    uold = u;
    vold = v;
    F = [6*u^3 + u*v - 3*v^3 - 4;
         u^2 - 18*u*v^2 + 16*v^3 + 1];
    DF = [18*u^2, u - 9*v^2;
          2*u - 18*v^2, 48*v^2 - 36*u*v];
    Z = DF\F;
    u = u - Z(1);
    v = v - Z(2);
    err = sqrt((u - uold)^2 + (v - vold)^2);
    k = k + 1;
end
```

2) For number 2, we were given a three-point Gauss-Legendre formula for approximating integrals. We were also given a non-linear system that we had to solve for under the interval [0,1] and with the degree of at most 5. The question asks us to solve for the system numerically for part (a) and approximate the integral given and compare it with the exact answer.

   a) Part (a), I first solved for each weight with the MATLAB script *NewtonCotesClosedWeights.m* given to us in the notes for quad1. That gave me the three weights *w* needed to solve this equation and it also allowed us to estimate the *x* values. When estimating the values for *x*, I first tried to do it similarly to the first problem on this homework, but I had no luck in getting any answers of significance. I settled on splitting the c values into three equal values which gave me the same as the initial guesses. I moved on with those, but I do believe this is not right. I spent endless hours testing other things and came up empty every time. All MATLAB source code and outputs are show below of my findings.

   b) For part (b), we were given an integral that we needed to approximate using the Gauss-Legendre formula and our results in part (a). First, I set up all my values to approximate this integral. The values I set up were my *x*, *w*, and *h* values as well as some *a* value that have the equation $a + c_n h$ in them. I finally plugged it in to the Gauss-Legendre formula to approximate the integral. I found my approximate to be the value -0.585396052530946 using the Gauss-Legendre formula. I found the exact answer for the integral to be -0.3963576608. My approximate value is actually not too far from the exact answer, which I expected it to be, but as I see over and over again in this class, my expectations sometimes never come true.

Source Code:

HW12_2.m

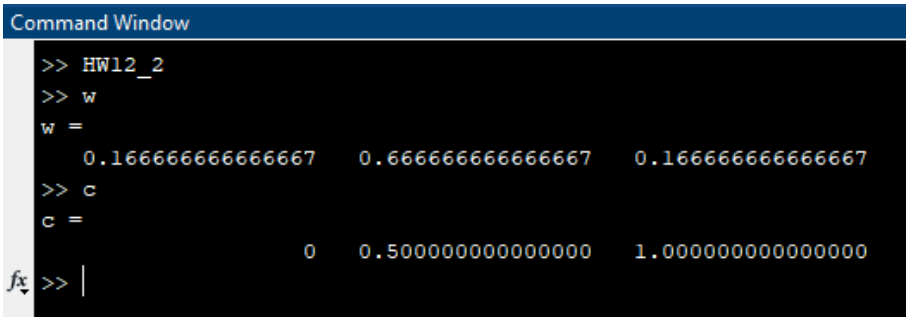
```
% function HW12_2.m
%
% This function sets up all testing for
% question 2 on HW12 and calls the
% MATLAB function nonlinearNewt.m
format compact;
format long;

%Part a
a = 0;
b = 1;
m = 3;
w = NewtonCotesClosedWeights(m);
c = linspace(a,b,m);

% Part b
f = @(x) exp(x)*(cos(5*x));
h = b - a;
w1 = w(1); w2 = w(2); w3 = w(3);
c1 = c(1); c2 = c(2); c3 = c(3);
p1 = a + c1*h; p2 = a + c2*h; p3 = a + c3*h;
approx = h*(w1*f(p1) + w2*f(p2) + w3*f(p3));
```

NewtonCotesClosedWeights.m

```
function w = NewtonCotesClosedWeights(m)
% function w = NewtonCotesClosedWeights(m)
% w is a column m-vector consisting of the weights for the m-point closed
% Newton-Cotes rule. m is an integer that satisfies 2 <= m <= 11. From
% M. Abramowitz and I. Stegun.
switch m,
    case 2,
        w = [1 1]/2;
    case 3,
        w = [1 4 1]/6;
    case 4,
        w = [1 3 3 1]/8;
    case 5,
        w = [7 32 12 32 7]/90;
    case 6,
        w = [19 75 50 50 75 19]/288;
    case 7,
        w = [41 216 27 272 27 216 41]/840;
    case 8,
        w = [751 3577 1323 2989 2989 1323 3577 751]/17280;
    case 9,
        w = [989 5888 -928 10496 -4540 10496 -928 5888 989]/28350;
    case 10,
        w = [2857 15741 1080 19344 5778 5778 19344 1080 15741 2857]/89600;
    case 11,
        w = [16067 106300 -48525 272400 -260550 427368 ...
            -260550 272400 -48525 106300 16067]/598752;
    otherwise,
        error = 'm not between 2 and 11 in NewtonCotesClosedWeights'
        pause
end
```

MATLAB OUTPUT:

Part (a):



Actual answer:

$$-\frac{3e^{\frac{\pi}{4}}}{13\sqrt{2}} - \frac{1}{26} = -0.3963576608$$

My approximate answer: