Nicholas Garrett

Professor Dailmari

Math 3316

9/14/2021

Assignment 4

1.

a.

Given $y_n = \int_0^1 \frac{x^n}{x+10}\,dx \;\; for \; n = 0, 1, 2, ...$

$\Rightarrow y_n * 10y_{n-1} = \int_0^1 \frac{x^n + 10x^{n-1}}{x+10}\,dx = y_n = \int_0^1 x^{n-1}dx = \frac{1}{n}$

So, $y_n = \frac{1}{n} - 10y_{n-1}$

Thus, given y_n and solving for y_{n-1},

$y_{n-1} = \frac{1}{10n} - \frac{1}{10}y_n$

b.

...

c.

```
function yn = fixedPointFunc(func, n1, tolerance, maxIterations)
    iterations = 0;
    yn = feval(func, n1)

    while abs(n1-yn)>tolerance && iterations < maxIterations

        iterations = iterations + 1;
        n1 = yn;
        yn=feval(func,n1)
    end
end
```

The n_1 was assigned from the n before 20, so 19.

d.

This approach is stable because as the iterations occur, the return becomes more and more accurate, regardless of initial guess.

2.

a.

To prevent overflow error, you can scale down the input to prevent it from overflowing. This can be done by dividing the inputted values by $max(|x_1|, |x_2|, |x_3|, ..., |x_n|)$, then forming the summation and squares.

b.

Code to solve for l2-norm:

```
function [norm] = l2NormFunc(a)
   %||x|||_2 = sqrt(sum(x_i^2)from 1->n)

   %scale input
   scalingValue = max(abs(a(:)))
   a(:) = a(:)/scalingValue;

   %length of the vector
   n = length(a)

   %variable to hold the sum
   sum = 0;

   %iterate through the terms, adding their squares to the sum
   for i = 1:n
       aiSquared = (a(i))^2;
        sum = sum + aiSquared;
   end

   %finish the algorithm and solve for norm
   norm = sqrt(sum)
   norm = norm * scalingValue
end
```

Given the vector $a = [1, 2, 3, 4]$, this script returned 5.477225575051661, which is $\sqrt{1^2 + 2^2 + 3^2 + 4^2}$

Given the vector $a = [10^{1}23, 4 * 10^{1}24, 9^{1}22, 14^{1}26]$, the script returned 2.583048112093935e+144. The error returned a zero when calculating through

```
error = sqrt(a(1)^2 + a(2)^2 + a(3)^2 + a(4)^2) - norm
```

When running for the array $a = [10^{1}23, 10^{1}24, 9^{1}22, 10^{1}26, 34^{1}23, 23 * 11^{1}24, 29^{1}32, 14^{1}26]$, there was an overflow in the "error" calculating line, where the terms were squared and then added without scaling. The algorithm I wrote returned $norm = 1.087766607871343e + 193$, though the normal method of calculating the norm (demonstrated in the error function, though without subtracting the norm) overflowed.

3.

a.

A Newton's method-based formula that could approximate x=ln(a) would be:

$a_{n+1} = a_n - a_n * ln(a_n)$

b.

A MATLAB script demonstrating the formula from 3.a:

```
% func: function
% dfunc: derrivatie of function
% guess: the initial guess
% tolerance: the error tolerance
% maxRuns: the maximum number of runs allowed before declaring failure
function [] = newtonsMethod(func, dfunc, guess, tolerance, maxRuns)
    iteration = 0;

    fprintf('iteration\ta_k\t\tf(a_k)\t\t\t|a_k - a_k-1|\n');

    xk = guess;
    xk1 = guess;
    error = 1;
    while (error > tolerance) && (iteration<maxRuns)
        iteration = iteration+1;

        fprintf('%d\t%d\t\t%d\t%e.\n', iteration, xk, func(xk), error);
        xk = xk1 - func(xk1)/dfunc(xk1);

        error = abs(xk - xk1);

        xk1 = xk;
    end

    xk

end
```

.

4.

This statement is true.

Given that $A \in R^{nxn}$ is nonsingular (has an inverse) and $Ax = \lambda x$, prove λ^{-1} is an eigenvalue of A^{-1}.

$Ax = \lambda x \Rightarrow A^{-1}Ax = A^{-1}\lambda x \Rightarrow x = A^{-1}\lambda x \Rightarrow \frac{1}{\lambda}x = A^{-1}x$

So, $\frac{1}{\lambda}x = A^{-1}x$, therefore λ^{-1} is an eigenvalue of A^{-1}

5.

a.

Psudo-code for LU decomposition of an upper Hessenberg matrix:

Flop count: n^3

```
function [] = LU_Solver(A,b)
    %size of A
    n = length(A)


    for i = 2:n
        for j = 1:i-1
            % perform row reduction
                differenceMultiply = (A(i,j)/A(j,j))
                A(i,:) = A(i,:) - differenceMultiply*A(j,:)
        end
    end

    % set L to the edited matrix A, this is not actually necessary
    L(:,:) = A(:,:)

    %get transopose of L to get U
    %U = L'
    for i = 1:n
        for j = 1:n
            U(i,j) = L(j,i);
        end
    end
    U
end
```

.

b.

.

To solve a linear system, using an LU decomposition algorithm, it should take $n * n * ((n + 2) + 2)$ for the gaussian elimination, then n^2 for solving reverse replacement to find x from the triangular matrix.

So, it wold be $n * n * ((n + 2) + 2) + n^2$

$= n^3 + 5n^2$, which should be the number of operations performed