Assignment 4 100750401

Mathew.Kasbarian

March 2021

1 Question 1A

Function NR
Input;
float Function(linear system of equations)
float DofF (Jacobian matrix of partial derivatives)
float a_0(intial point close to solution)
int Max(max iterations)
float eps_x(error tolerance)
float eps_f(residual tolerance)

Output;
float a(approx solution)
float error(error of solution)
float residual(residual of solution)

Initializations:
x <− x_0

for $a = 1, kMax$ **do**
  $b <-$ f(x)
  $c <-$ mat($\frac{df^i}{dx^c}$)
  solve $c * dx = -r$
  $x <-$ x + dx
  $error <- dx_2$
  $residual <- r_2$
  if residual < eps_f AND error < eps_x **then**
    STOP
  end if
end for
Return: x, error, residual
end

2 Question 1B

refer to the *test_NR_template.py* file in the repository to see the changes made to the functions F(x) and DF(x), along with the initialization of the variables used to test it.

3 Question 1C

If we compare the sequence of residuals for the Newton-Raphson iteration as well as the Newton iteration they are only slightly different in terms of their rate of convergence. The Newton-Raphson is a way to quickly find a good approximation for the root of a real-valued function $f(xn+1) = x_n(\acute{f}(x_n)/f(x_n))$. The Newton-Raphson is also a system of equations and matrices that deals with linear systems such as the Jacobian opposed to the regular Newton method which is only using 1 equation and its derivative.

4 Question 2a

Function lowerProduct
Input: matrix A is lower triangular and matrix B is lower triangular
Output: matrix C is a product of matrix A and of matrix B
Initializations:
x $<-$ size of matrix A (columns)
C $<-$ Matrix of size (x,x)

for $a = 0, x$ **do**
  for $b = 0, x$ **do**
    for $d = 0, x$ **do**
      if A[a,d] != 0 and B[d,b] != 0 **then**
        C[a,b] $<-$ C[a,b] + A[a,d] * B[d,c]
      end if
    end for
  end for
end for
Return: C
end

5 Question 2b

Function lowerProduct
Input: matrix A and B are both lower triangular matrix
Output: matrix C is a product of matrix A and matrix B
Initializations:

```
n <− size of matrix A
C <− Matrix of size (n,n)
for i = 0, n do
    for j = 0, n do
        for k = 0, n do
            print('C[', i, j, ']', C[i,j], '+=', A[i,k], '*', B[k,j])
            C[i,j] += A[i,k] * B[k,j]
        end for
    end for
end for
Return: C
end
```

BONUS

If we compute the flops of the pseudo code you will eventually come to the conclusion that the glop count will be $2i^2n - 2jin + 4in - 2jn + 2n$ meaning it is 6 times fewer flop counts than regular matrix-matrix multiplication.