Proyecto Final Analisis Numerico

Juan Jose Moreno; Sebastian Tapias; Samuel Botero

November 2022

1 Introduction

There are some mathematical problems with a high level of complexity, problems that analytically we cannot find a solution, nowadays, due to computers we can perform more complex calculations and implement algorithms to solve these problems, those are the numerical methods. As a final project of numerical analysis we will implement some numerical methods about solutions of non linear equations, solution of systems of linear equations and interpolation, everything in python.

2 User Guide

First, make sure to have Python3 installed in your computer. Then, download the repository and open the project directory using command line in which you will input the following command: **Python ./interface.pyw** which will display the interface of the program. You will see appear a window divided in three in which they have all the methods. You just have to click on the method you want to calculate an a sub-window will appear telling you exactly what to do in order to get your desired result.

3 Methods

3.1 Numerical solutions of non linear equations

• **Bisection:** The algorithm finds the solution of f(x) in an interval [a,b].

```
bisection (f, a, b, tol, max_ite):
  fa = f (a)
  Middle_Point= (a + b) / 2
  fmp = f (Middle_point)
  E = 0;
  iterations= 1
  while E> tol or iterations < max_ite
    if fa * fmp <0:
```

```
    b = mp;
  else:
    a = mp
  p = mp
  pm = (a + b) / 2
  fpm = f (mp)
  E = abs (mp-p)
  iterations = iterations + 1
end while
Return pm, E, iterartions
```

• **False Rule:** The algorithm finds a solution of f(x)=0.

```
false rule (f, a, b, tol, max_ite):
  fa = f (a)
  fb = f (b)
  mp = (fb * a-fa * b) / (fb-fa)
  fmp = f (mp)
  E = 0
  iterations = 1
  while E> tol or iterations <max_ite:
    if fa * fmp <0:
      b = mp
    else:
      a = mp
    p = mp
    mp = (f (b) * a-f (a) * b) / (f (b) -f (a))
    fmp = f (mp)
    E = abs (mp-p)
    iterations = iterations + 1
  end while
  Return mp, E, iterations
```

• **Fixed Point:** The algorithm finds the solution of f(x)=0 by solving the equivalent problem f(x)=x.

```
fixed_point (g, x0, tol, max_ite):
  prev_x = x0
  E = 0
  iterations = 0
  while E > tol or iterations < max_ite:
    act_x = g (ant_x)
    E = abs (act_x - ant_x)
    iterations = iterations + 1
```

```
    prev _x= act_x
  end while
  Return act_x, E, iterations
```

• **Incremental Searches:** The algorithm finds a change of sign in an interval, where there is a solution for f(x)=0.

```
Incremental_searches(f, x0, h, max_ite):
  prev_x = x0
  fprev = f (prev_x)
  act_x = prev_x + h
  fact = f (act_x)
  for i = 1: max_ite:
    if fprev * fact <0
      break
    prev_x = act_x
    fprev=fact
    act_x = pev_x + h
    fact = f (act_x)
  end for
  Return prev_x, act_x
```

• **Multiple Roots:** The algorithm finds a solution of f(x)=0.

```
multiple roots (f, df, d2f, x0, tol, max_ite):
  xant = x0
  fant = f (xant)
  E = 0
  iterations = 0
  while E> tol or iterations <max_ite:
    act_x = prev_x-fant * df (prev_x) / ((df (prev_x)) ^ 2-fant * d2f (prev_x))
    fact = f (act_x)
    E = abs (act_x-prev_x)
    iterations = iterations + 1
    prev_x = act_x
    fprev = fact
  end while
  Return act_x, E, iterations
```

• **Newton:** The algorithm finds a solution of f(x)=0.

```
newton (f, df, x0, tol,  max_ite):
  prev_x = x0
```

```
  fprev = f (prev_x)
  E = 0
  iterations = 0
  while E> tol or cont < max_ite:
    act_x = prev_x-fpev / (df (prev_x))
    fact = f (act_x)
    E = abs (act_x-prev_x)
    iterations = iterations + 1
    prev_x = act_x
    fprev = fact
  end while
  Return act_x, E, iterations
```

3.2 Solution of systems of linear equations

• **Secant** The algorithm finds a solution of f(x)=0.

```
secant (f, x0, x1, tol, max_ite ):
  f0 = f (x0)
  f1 = f (x1)
  E = 0
  iterations = 1
  while E> tol or cont < max_ite:
    act_x = x1-f1 * (x1-x0) / (f1-f0)
    fact = f (act_x)
    E = abs (act_x - x1)
    iterations = iterations + 1
    x0 = x1
    f0 = f1
    x1 = act_x
    f1 = fact
  end while
  Return act_x, E, iterations
```

• **Crout:** The algorithm finds the solution for Ax=b and the LU factorization of A.

```
Crout (A, b):
    n = size (A, 1)
    L = eye (n)
    U = eye (n)
    for i = 1: n-1
        for j = i: n
            L (j, i) = A (j, i) -dot (L (j, 1: i-1), U (1: i-1, i) ')
        end for
```

```
        for j = i + 1: n
            U (i, j) = (A (i, j) -dot (L (i, 1: i-1), U (1: i-1, j) )) / L (i, i)
        end for
    end for
    L (n, n) = A (n, n) -dot (L (n, 1: n-1), U (1: n-1, n) )
    Return L, U, x
```

• **Doolitle:** The algorithm finds the solution for Ax=b and the LU factorization of A.

```
Doolittle (A, b):
    n = size (A, 1)
    L = eye (n)
    U = eye (n)
    for i = 1: n-1
        for j = i: n
            U (i, j) = A (i, j) -dot (L (i, 1: i-1), U (1: i-1, j) )
        end for
        for j = i + 1: n
            L (j, i) = (A (j, i) -dot (L (j, 1: i-1), U (1: i-1, i) )) / U (i, i)
        end for
    end for
    U (n, n) = A (n, n) -dot (L (n, 1: n-1), U (1: n-1, n) )
    Return L, U, x
```

• **Gauss:** The algorithm finds the solution of Ax=b.

```
gauss (A, b):
    n = size (A, 1)
    M = [A b]
    for i = 1: n-1
        for j = i + 1: n
            if M (j, i) ~ = 0
                M (j, i: n + 1) = M(j, i: n + 1)-(M (j, i) / M (i, i)) * M (i, i: n + 1)
            end for
        end for
    end for
    Return x
```

• **Gauss-Seidel:** The algorithm finds the solution of Ax=b.

```
gseidel (A, b, x0, tol, Nmax):
    D = diag (diag (A))
    L = -tril (A) + D
```

```
    U = -triu (A) + D
    T = inv (D-L) * U
    C = inv (D-L) * b
    xant = x0
    E = 1000
    count = 0

    while E> tol && cont <Nmax
        xact = T * xant + C
        E = norm (xant-xact)
        xant = xact
        count = count + 1
    end while

    x = xact
    iter = cont
    err = E
    Return x, iter, err
```

• **Jacobi:** The algorithm finds the solution of Ax=b

```
jacobi (A, b, x0, tol, Nmax):
    D = diag (diag (A))
    L = -tril (A) + D
    U = -triu (A) + D
    T = inv (D) * (L + U)
    C = inv (D) * b
    xant = x0
    E = 1000
    count = 0

    while E> tol && cont <Nmax
        xact = T * xant + C
        E = norm (xant-xact)
        xant = xact
        count = count + 1
    end while
    x = xact
    iter = cont
    err = E
    Return x, iter, err
```

• **LU factorization:** The algorithm finds the solution of Ax=b and the LU factorization of A.

```
lu (A, b):
    n = size (A, 1)
    L = eye (n)
    U = zeros (n)
    M = A
    for i = 1: n-1
        for j = i + 1: n
            if M (j, i) ~ = 0
                L (j, i) = M (j, i) / M (i, i)
                M (j, i: n) = M (j, i: n) - (M (j, i) / M (i, i)) * M (i, i: n)
            end if
        end for
        U (i, i: n) = M (i, i: n)
        U (i + 1, i + 1: n) = M (i + 1, i + 1: n)
    end for
    U (n, n) = M (n, n)
    z = subs ([L b])
    x = subs ([U z])
    Return L, U, x
```

3.3 Interpolation:

• **Divided differences:**

```
The algorithm finds the interpolating polynomial of the given points.
differences (X, Y):
  n = length (X)
  A = zeros (n)
  D (:, 1) = Y ’
  for i = 2: n
    aux0 = D (i-1: n, i-1)
    aux = diff (aux0)
    aux2 = X (i: n) -X (1: n-i + 1)
    D (i: n, i) = aux / aux2 ’
   end for
```

• **Lagrange:**

```
lagrange (X, Y):
  n = length (X)
  L = zeros (n)
  for i = 1: n
    aux0 = setdiff (X, X (i))
```

```
    aux = [1 -aux0 (1)]
      for j = 2: n-1
        aux = conv (aux, [1 -aux0 (j)])
      end for
     L (i,:) = aux / polyval (aux, X (i))
  end for
  coef=Y*l
  Returncoef
```

- **Vandermonde:**

```
vandermonde (X, Y):
  n = length (X)
  A = zeros (n)
  for i = (1: n)
    A (:, i) = (X. ^ (N-i))
    end for

    Return
```

4 Diagrams

4.1 Use Case Sequence

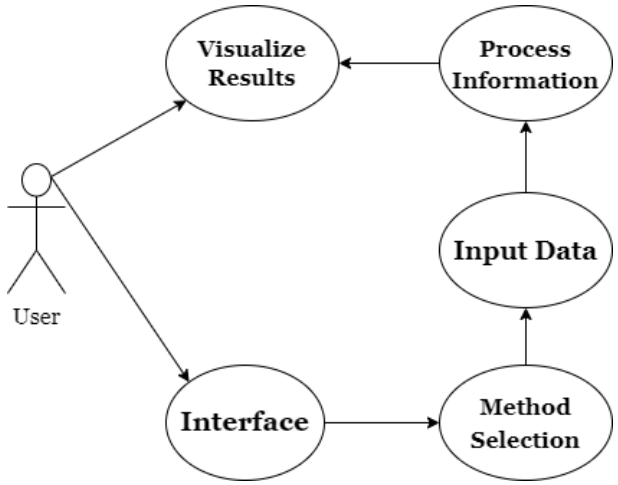

Figure 1: Use Case Diagram

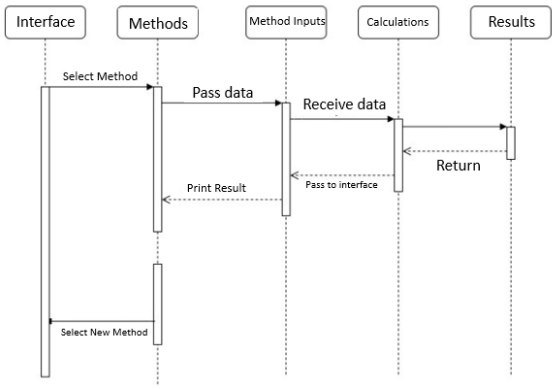

Figure 2: Sequence Diagram

5 Conclusion

This was a very challenging project since as Mathematical Engineers we are not taught about front-end development or even Python for object-oriented programming. We know we could have used a directory called tools in which we could put many functions we repeated over the project to make the code more organized, however as we said, this is not our area of expertise. In the other hand, our mathematical background was very useful to program all the different methods that we used and we believe this is a very useful tool from which we can take a lot of advantage in futures subjects when we encounter with numerical methods.