Universidad Nacional de San Agustín

Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de Ciencia de la Computación

Calculadora Matricial en Octave

Curso: Algebra Lineal Númerica

Presentado por:
Juan Miguel Ravelo Jove

Arequipa-Perú
2019

Índice general

Índice de figuras

Capítulo 1

Introducción

Palabras clave:

- Calculadora de Matrices.
- Octave.
- GUI.

1.1. Funciones

La calculadora posee las siguientes funciones hechas en *Octave*, validas tambien en *Matlab*:

1. Operaciones Unarias
    - Descomposición de Cholesky
    - Descomposición LU
    - Descomposición PLU
    - Descomposición QR
    - Determinante de una matriz
    - Inversa de una matriz
    - Proceso de ortonormalización de Gram Schmith

- Algoritmo de Leverrier para eigen valores y eigen vectores
- Rango de una matriz

2. Operaciones Binarias

- Matriz de cambio de Base
- Solución de sistema de ecuaciones por:
    - Cholesky
    - LU
    - PLU
    - Eliminacion gaussiana
- Rango de una matriz aumentada
- Suma de matrices
- Multiplicación de matrices

3. Regresion polinomial en una variable por descomposición QR [1]

4. Regresion polinomial en una variable por operaciones con matrices

Capítulo 2

Operaciones Unarias

Consideraremos operaciones con una sola matriz.

2.1. Descomposición de Cholesky

Si la matriz es simetrica y positiva definida, esta admite descomposición $A = G * G^T$. Teniendose las siguientes funciones involucradas para su resolución en Octave.

2.1.1. Función Cholesky

Esta función busca si la *matriz* A es **simetrica** y **positiva definida**

```
function verdad=cholesky(A)
  [fila col]=size(A);
  verdad=true;
  if (fila!=col)
    error(" tiene que ser una matriz cuadratica \n");
  else
    if(simetria(A))
      for i=1:fila
        if (det(A(1:i,1:i))<=0)
          verdad=false;
          break;
```

```
        endif
      endfor
    endif
  endif
endfunction
```

2.1.2. Descomposición de Cholesky

Una vez cumplida la función anterior, se procede a descomponer la matriz **A** en $G * G^T$.

```
%G es triangular inferior
%Gt es triangular superior
function [G,Gt]=descomposicionCholesky(A,b)
  if (cholesky(A))
    [m_row m_col]=size(A);
    G=zeros(m_row,m_col);
    G(1,1)=sqrt(A(1,1));
    for i=2:m_row
      for j=1:i
        if(j==1)
          G(i,1)=A(i,1)/G(1,1);
        elseif (i==j)
          temp=0;
          for k=1:(i-1)
            temp = temp + (G(i,k))^2;
          endfor
          G(i,i)=sqrt(A(i,i)-temp);
        else
          temp=0;
          for k=1:(j-1)
            temp=temp + G(i,k)*G(j,k);
          endfor
          G(i,j)=(A(i,j)-temp)/G(j,j);
```

```
        endif
      endfor
    endfor
    Gt=G';
  else
    error("no tiene descomposicion de Cholesky \n");
  endif
endfunction
```

2.2. Descomposición LU

No toda matriz A tiene descomposición LU, si el elemento pivote en el momento de hacer el escalonamiento inferior es igual a cero, entonces dicha matriz A no tiene descomposición LU.

```
%L es triangular inferior
%U es triangular superior
function [m_L,m_U]=descomposicionLU(A)
    [m_row m_col]=size(A);
    m_A=A;
    m_L=zeros(m_row);
    m_U=m_A;
    %m_I=eye(m_row);
    for j=1:(m_row-1)
        if(m_U(j,j) == 0)
            error("!La matriz A no tiene descomposicion LU! \n");
            break;
        endif
        for i=j+1:m_row
            m=m_U(i,j)/m_U(j,j);
            m_L(i,j)=m;
            for k=1:m_col
                m_U(i,k)=m_U(i,k)-m*m_U(j,k);
```

```
                %m_I(i,k)=m_I(i,k)-m*m_I(j,k);
            endfor
        endfor
    endfor
    I=eye(m_row);
    m_L = m_L + I;
endfunction
```

2.3. Descomposición PLU

A diferencia de la descomposición LU, en PLU; se logra superar la condición del elemento pivote, gracias a la matriz P.

```
function [m_P,m_L,m_U]=descomposicionPLU(A)
  [m_row m_col]=size(A);
  m_A = A;
  m_P = eye(m_row);
  m_L = zeros(m_row);
  m_U = A;
  for j=1:(m_row - 1)
    max_val=m_U(j,j);
    max_file=j;
    for x=(j+1):m_row
      if(max_val<abs(m_U(x,j)))
        max_val=m_U(x,j);
        max_file=x;
      endif
    endfor
    if(j!=max_file)
        m_U=swapRow(m_U,j,max_file);
        m_P=swapRow(m_P,j,max_file);
        m_L=swapRow(m_L,j,max_file);
    endif
```

```
    for i=(j+1):m_row
        m=m_U(i,j)/m_U(j,j);
        m_L(i,j)=m;
        for k=1:m_col
          m_U(i,k)=m_U(i,k)-m*m_U(j,k);
        endfor
    endfor
  endfor
  I=eye(m_row);
  m_L = m_L + I;
endfunction
```

2.4. Descomposición QR

```
%proceso Gram Schmith columna -> QR
% vi= es vector-columna
% siendo A={v1,v2,...,vn} son base de un espacio vectorial V
% Ui=(Wi)/|Wi|
% Wi=vi-combinacionLineal
% combinacionLineal= S<vi,uj>*uj from j=1 to i-1
% Q es el proceso GramSchmith en columnas
% R es una matriz triangular superior
function [Q R]=descomposicionQR(A)
  [f c]=size(A);
  Q=zeros(f,c);
  R=zeros(c,c);
  for i=1:c
    v=A(:,i);
    combinacionLineal=zeros(f,1);
    for j=1:(i-1)
      u=Q(:,j);
      R(j,i)=sum(v.*u);
```

```
      combinacionLineal=combinacionLineal+R(j,i)*u;
    endfor
    Wi=v-combinacionLineal;
    R(i,i)=sqrt(sum(Wi.*Wi));
    Q(:,i)=Wi/R(i,i);
  endfor
endfunction
```

2.5. Determinante de una matriz

La determinante de una matriz A, en esta función se calcula hallando a una matriz equivalente de la forma triangular superior, de modo tal que se multiplique los elementos de la diagonal principal de la matriz equivalente, para obtener la determinante de la matriz A.

```
function det=determinante(A)
    m_matrix=A;
    [m_row m_col]=size(A);
    n=0;
    det=1;
    for j=1:(m_row-1)
      % max_val elemento pivote
      max_val=m_matrix(j,j);
      max_file=j;
      for x=(j+1):m_row
        % hallando el valor maximo para q sea pivote */
        if(max_val<abs(m_matrix(x,j)))
          max_val=abs(m_matrix(x,j));
          max_file=x;
          n=n+1; % es para (-1)^n
        endif
      endfor
      % donde esta la fila con el valor q va ser pivote
```

```
      % se lo lleva para arriba
      if(j!=max_file)
        m_matrix=swapRow(m_matrix,j,max_file);
      endif
      % f1=f1-mf2
      % m21=a21/a11;
      for i=(j+1):m_row
        m=m_matrix(i,j)/m_matrix(j,j);
        for k=1:m_col
          m_matrix(i,k)=m_matrix(i,k) - m*m_matrix(j,k);
        endfor
      endfor
    endfor
    for i=1:m_row
      det=det*m_matrix(i,i);
    endfor
    det=det*(-1)^n;
endfunction
```

2.6. Inversa de una matriz

Para hallar la inversa, se prosigue el método de Gauss-Jordan, añadiendo una matriz identidad a la matriz cuadrada A, y mediante operaciones elementales de fila y fila, llevar a la matriz A a la identidad, de modo que la matriz identidad se convierta en la matriz inversa de A.

```
function inv=inversa(matrix)
  [m_row m_col]=size(matrix);
  if(m_row==m_col)
    filas_no_nulas=rango(matrix);
    if(filas_no_nulas==m_row)
      A=matrix;
      inv=eye(m_row,m_col);
```

```
%escalonamiento inferior 000
for j=1:(m_row-1)
    max_val=A(j,j);
    max_file=j;
    %identificacion del mayor valor en la columna j
    for x=(j+1):m_row
        if(max_val<A(x,j))
            max_val=A(x,j);
            max_file=x;
        endif
    endfor
    %en el caso q el elemento pivote sea cero
    %hallamos el valor mas negativo
    if(max_val==0)
      for x=(j+1):m_row
        if(max_val>A(x,j))
          max_val=A(x,j);
          max_file=x;
        endif
      endfor
    endif
    if(j!=max_file)
        A=swapRow(A,j,max_file);
        inv=swapRow(inv,j,max_file);
    endif
    %Resta de Filas para escalonar
    for i=(j+1):m_row
        m=A(i,j)/A(j,j);
        for k=1:m_col
            A(i,k)=A(i,k) - m*A(j,k);
            inv(i,k)=inv(i,k) - m*inv(j,k);
        endfor
```

```
        endfor
    endfor
    % igualando a 1 la diagonal principal
    for i=1:m_row
        max_val=A(i,i);
        for j=1:m_col
            A(i,j)=A(i,j)/max_val;
            inv(i,j)=inv(i,j)/max_val;
        endfor
    endfor
    %escalonamimento superior 000
    for j=m_row:-1:2
        for i=(j-1):-1:1
            m=A(i,j)/A(j,j);
            for k=1:m_col
                A(i,k)=A(i,k) - m*A(j,k);
                inv(i,k)=inv(i,k) - m*inv(j,k);
            endfor
        endfor
    endfor
  else
    printf("no tiene inversa dado que tiene filas nulas \n");
  endif
 else
  printf("dimensionalmente incorrecto \n");
 endif
endfunction
```

2.7. Proceso de Ortonormalización de Gram Schmith

```
%proceso Gram Schmith filas
% vi= es vector-fila
% siendo B={v1;v2;...;vn} son base de un espacio vectorial V
% Ui=(Wi)/|Wi|
% Wi=vi-combinacionLineal
% combinacionLineal= S<vi,uj>*uj from j=1 to i-1

function G=gramSchmith(B)
  [f c]=size(B);
  G=zeros(f,c);
  for i=1:f
    v=B(i,:);
    combinacionLineal=zeros(1,c);
    for j=1:(i-1)
      u=G(j,:);
      combinacionLineal=combinacionLineal+(sum(v.*u))*u;
    endfor
    Wi=v-combinacionLineal;
    modulo=1/sqrt(sum(Wi.*Wi));
    G(i,:)=modulo*Wi;
  endfor
endfunction
```

2.8. Algoritmo de Leverrier

En este algoritmo, se busca encontrar los coeficientes del polinomio caracteristico de una matriz cuadrada A.

```
%polinomio caracteristico de A
function p=polinomio(A)
```

```
  [n m]=size(A);
  if (n==m)
    p=zeros(1,n); #vector de coef del polinomio
    Mi=I=eye(n);
    for i=1:n
      T=A*Mi; #matriz para sumar su diagonal principal
      p(i)=(-1/i)*trace(T);
      Mi=T+p(i)*I;
    endfor
    p=[1 p]; %vector de coeficientes
  else
    error("no es una matriz cuadrada")
  endif
endfunction
```

Teniendo el polinomio caracteristico se procede a calcular las raices del polinomio, cuyo conjunto solución seria, los autovalores de la matriz A, y teniendo dichos autovalores se calcula los autovectores asociados a cada autovalor.

```
function [V D]=algoritmoLeverrier(A)
    poli=polinomio(A);
    n=length(A);
    raiz=roots(poli);
    B=zeros(n);
    C=-1.*A(2:n,1); %columna de A
    D=diag(raiz); %valores propios en la matriz diagonal
    V=zeros(n);
    S=zeros(n,1); %vector propio
    for i=1:length(D)
        B=A(2:n,2:n)-D(i,i)*eye(n-1); %submatriz de A
        S=[1 (B\C)'];
        % vectores propios normalizados
        % cada vector propio es una columna de V
        V(1:n,i)=S/norm(S);
```

```
    endfor
endfunction
```

2.9. Rango de una matriz

Es el número de filas no nulas una matriz A, una vez llevada a una matriz equivalente, triangular superior por operaciones de fila y fila (escalonamiento).

```
%Rango
%A matriz para escalonar
%contador, cuenta numero de filas no nulas
function contador=rango(A)
  C=A;
  [m_row m_col]=size(A);
  verdad=true; %empieza con q es fila nula
  contador=m_row;
  C=escalonadoInferior(C);
  for i=1:m_row
    for j=1:m_col
      if(C(i,j)!=0); %apenas haye un numero en la fila
        verdad=false;
      endif
    endfor
    if(verdad)
      contador=contador-1;
    endif
    verdad=true;
  endfor
endfunction
```

Capítulo 3

Operaciones Binarias

En este tipo de operaciones se necesitan dos matrices para llevar a cabo la operación de la función correspondiente.

3.1. Matriz de cambio de Base

Es una matriz que te lleva de un sistema de coordenadas (definido por un base) a otro.

```
% Matriz de cambio de base de 1 a 2
% B1={u1,u2,...,un}
% B2={v1,v2,...,vn}
% u1=a1*v1+a2*v2+...+an*vn
% (u1,u2,...,un)=(v1,v2,v3,...,vn)* Mb1b2
% B1 = B2 * Mb1->b2
% M=B2^-1 *B1
function M=cambioBase12(A,B)
  B1=A';
  B2=B';
  M= inv(B2)*B1;
endfunction
```

3.2. Solución de sistema de ecuaciones

En esta sección se pretende solucionar el sistema de ecuaciones de la forma $AX = B$; siendo:

- A una matriz del orden de mxn
- X un vector columna del orden nx1
- B un vector columna del orden mx1

Teniendo las siguientes métodos.

3.2.1. Cholesky

Haciendo uso de la descomposición de Cholesky, una vez teniendo las matrices de la descomposicion ($G * G^T$), para realizar con G una sustitución progresiva, dado que G es una matriz triangular inferior y para G^T una sustitución regresiva, dado que G^T es una matriz triangular superior.

```
% descomposicionCholesky
% AX=b
% (G*G')X=b
% G'X=Y
% GY=b  %sustitucion progresiva
% con Y solucionamos G'X=Y con  sustitucion regresiva
function X=solucionCholesky(A,b)
    [G,Gt]=descomposicionCholesky(A)
    Y=sust_progresiva(G,b); %G*Y=b
    X=sust_regresiva(Gt,Y); %Gt*X=Y
endfunction
```

3.2.2. LU

Haciendo uso de la descomposición de LU, una vez teniendo las matrices de la descomposicion ($L * U$), para realizar con L una sustitución progresiva, dado que L

es una matriz triangular inferior y para U una sustitución regresiva, dado que U es una matriz triangular superior.

```
%descomposicion LU
% Ax=b
% A=LU
% L(Ux)=b
% L(y)=b sust_progresiva
% obtenemos como solucion y
% Ux =y        sust_regresiva
% obtenemos la solucion x
%b vector fila
%L es triangular inferior
%U es triangular superior
function X=descomposicionLU(A,b)
    [m_L,m_U]=descomposicionLU(A);
    y=sust_progresiva(m_L,b);
    X=sust_regresiva(m_U,y);
endfunction
```

3.2.3. PLU

Es similar al caso de LU, con la diferencia de la matriz P se multiplica con la matriz bs^T del sistema $AX = bs^T$.

```
%descomposicion PLU
% A matriz
% bs para la matriz aumentada
% bs vector fila
% PA = LU
% Ax=b
% P*b = y_temp
% L(Ux)=(P*b)
% L(y)=y_temp sust_progresiva
```

```
% Ux=y   sust_regresiva
function x=solucionPLU(A,bs)
  [m_P,m_L,m_U]=descomposicionPLU(A);
  bs=bs'; %vector columna
  %printf("Matriz P*bs \n");
  PB=m_P*bs;
  [fila col]=size(PB);
  y_temp=(PB(:,col))'; %vector fila
  y=sust_progresiva(m_L,y_temp);
  x=sust_regresiva(m_U,y);
endfunction
```

3.2.4. Eliminacion Gaussiana regresivo

La matriz aumentada $A|B$ de $AX = B$ se realiza operaciones elementales de fila y fila , para obtener una matriz equivalente triangular superior para realizar la función de sustitución regresiva.

```
%sustitucion regresiva
%A matrix de nxm
%xs es un vector fila
function rpta=sust_regresiva(A,xs)
  m_matrix=A;
  m_row=length(m_matrix(:,1));
  m_col=length(m_matrix(1,:));
  rpta=zeros(1,m_row);
  for i=m_row:-1:1
    s=0;
    for j=i:m_row
      s=s+m_matrix(i,j)*rpta(1,j);
    endfor
    rpta(1,i)=(xs(1,i)-s)/m_matrix(i,i);
  endfor
endfunction
```

```
%emiliminacion de Gauss regresivo
% r vector solucion
function r=elim_gauss_regresivo(A,xs)
  m_matrix=A;
  columna=xs';
  [v1,v2]=rango_aumentado(A,xs);
  if(v2)
    %matriz aumentada insertando una columna a la matriz A
    m_matrix=insertCol(m_matrix,columna);
    m_matrix=escalonadoInferior(m_matrix)
    %hago un get de la ultima columna
    [n m]=size(m_matrix);
    temp=m_matrix(:,m);
    temp=temp'
    r=sust_regresiva(m_matrix,temp);
  else
    error("no se puede operar \n");
  endif
endfunction
```

3.2.5. Eliminación gaussiana progresiva

La matriz aumentada $A|B$ de $AX = B$ se realiza operaciones elementales de fila y fila , para obtener una matriz equivalente triangular inferior para realizar la función de sustitución progresiva.

```
%sustitucion progresiva
function rpta=sust_progresiva(A,xs)
  m_matrix=A;
  m_row=length(m_matrix(:,1));
  m_col=length(m_matrix(1,:));
  rpta=zeros(1,m_row);
  for i=1:m_row
    s=0;
```

```
    for j=1:i
      s=s+m_matrix(i,j)*rpta(1,j);
    endfor
    rpta(1,i)=(xs(1,i)-s)/m_matrix(i,i);
  endfor
endfunction
```

```
%emiliminacion de Gauss progresivo
% r vector solucion
function r=elim_gauss_progresivo(A,xs)
  m_matrix=A;
  columna=xs';
  [v1,v2]=rango_aumentado(A,xs);
  if(v2)
    m_matrix=insertCol(m_matrix,columna);
    m_matrix=escalonadoSuperior(m_matrix)
    [n m]=size(m_matrix);
    temp=m_matrix(:,m);
    temp=temp'
    r=sust_progresiva(m_matrix,temp);
  else
    error("no se puede operar \n");
  endif
endfunction
```

3.2.6. Rango de una matriz aumentada

Es la comparación del número de filas no nulas de la matriz aumentada $A|B$ de $AX = B$, una vez llevada a una matriz equivalente, triangular superior por operaciones de fila y fila (escalonamiento) y de la matriz A; para determinar si tiene o no solución.

```
%rango aumentado comparado
% M es la matriz base
```

```
% xs vector para la matriz aumentada
% xs vector fila
% t bool
function [v1,v2]=rango_aumentado(M,xs)
  A=M;
  [n m]=size(A);
  xs=xs'; %vector columna
  Ab=insertCol(M,xs);
  i=rango(A);
  j=rango(Ab);
  if(i==j)
    v1=true;
    if(n==i)
      %el sistema tiene solucion unica
      v2=true;
    else
      %tiene infinitas soluciones
      v2=false;
      %printf("tiene infinitas soluciones \n");
    endif
  else
    %no tiene solucion
    v1=false;
    v2=false;
    %printf("no tiene solucion \n");
  endif
endfunction
```

3.2.7. Suma de matrices

```
function s=suma(A,B)
  %primero vemos si son de la misma dimension
  [m n]=size(A);
```

```
  [f c]=size(B);
  if(m==f & n==c)
    s=zeros(m,n);
    for i=1:m
      for j=1:n
        s(i,j)=A(i,j)+B(i,j);
      endfor
    endfor
  else
    error("error de dimension \n");
  endif
endfunction
```

3.2.8. Producto de matrices

```
function r=producto(A,B)
  A_row=length(A(:,1));
  A_col=length(A(1,:));
  B_row=length(B(:,1));
  B_col=length(B(1,:));
  if(A_col==B_row)
    r=zeros(A_row,B_col);
    total=0;
    for i=1:A_row
        for j=1:B_col
            for z=1:A_col
                total = total + A(i,z) * B(z,j);
            end
            r(i,j) =total;
            total=0;
        end
    end
  else
```

```
    error("no se puede multiplicar \n");
  end
endfunction
```

Capítulo 4

Regresion polinomial

En esta sección tenemos:

- X, lista de datos de la variable independiente.
- Y, lista de datos de la variable dependiente.
- n, el grado del polinomio de tipo $y = a_0 + a_1x + ... + a_nx^n$

4.1. Regresion polinomial en una variable por operaciones con matrices

```
% regresion polinomial de grado n
% a0+a1x+...+anx^n=y
% coef=[a0:an]
% A x = b
% X=inv(A'*A)*A'*B
% Xi es vector lineal fila
% Yi es vector lineal fila
function coef=regresionNAt(Xi,Yi,n)
  if (n<1)
    error("no hay regresion");
  elseif (length(Xi)==length(Yi))
```

```
    x0=ones(length(Xi),1);
    xi=Xi';
    A=zeros(length(Xi),n+1);
    A(:,1)=x0;
    for i=1:n
      xi=xi.^i;
      A(:,i+1)=xi;
    endfor
    %A
    B=Yi';
    %B
    coef=inv(A'*A)*A'*B;
  endif
endfunction
```

4.2. Regresion polinomial en una variable por descomposición QR

```
% regresion polinomial de grado n
# por descomposicion QR
% a0+a1x+...+anx^n=y
% coef=[a0:an]
% A x = b
% X=inv(A'*A)*A'*B
% Xi es vector lineal fila
% Yi es vector lineal fila
% n es el tipo de regresion a utilizar
function coef=regresionQR(Xi,Yi,n)
  if (n<1)
    error("no hay regresion");
  elseif (length(Xi)==length(Yi))
    x0=ones(length(Xi),1);
```

```
    xi=Xi';
    A=zeros(length(Xi),n+1);
    A(:,1)=x0;
    for i=1:n
      xi=xi.^i;
      A(:,i+1)=xi;
    endfor
    [Q R]=descomposicionQR(A);
% Q base ortonormal -> Q'==inv(Q)
% R matrix diagonal superior
    B=Yi';
    b=Q'*B;
% R*X=Q'*B
    coef=sust_regresiva(R,b');
  else
    error("las listas Xi e Yi estan mal dimensionadas");
  endif
endfunction
```

Capítulo 5

Calculadora

5.1. Codigo

Handles: Estructura general de una GUI.

Callback: Subrutina que se ejecuta debido a un evento.

Guidata: Consultar o establecer datos de GUI personalizados por el usuario. Los datos de la GUI se almacenan en el identificador de figura **h**. Si h no es un identificador de figura, su figura principal se utilizará para el almacenamiento. Los datos deben ser un solo objeto, lo que significa que generalmente es preferible que sea un contenedor de datos, como una matriz de celdas o una estructura, para que se puedan agregar fácilmente elementos de datos adicionales.

uicontrol: Cree un objeto uicontrol y devuélvale un identificador. Un objeto uicontrol se utiliza para crear controles interactivos simples, como botones, casillas de verificación, controles de edición y lista. Si se omite el padre, se crea un uicontrol para la figura actual. Si no hay una figura disponible, primero se crea una nueva figura. Si se proporciona el padre, se crea un control uicontrol relativo al padre. Cualquier par de valores de propiedad proporcionado anulará los valores predeterminados del objeto uicontrol creado. Las propiedades de los objetos de uicontrol se documentan en las Propiedades de Uicontrol. El tipo de uicontrol creado se especifica mediante la propiedad de estilo. Si no se proporciona ninguna propiedad de estilo, se creará un botón pulsador [2].

Este codigo en *Octave* trabaja de la siguiente manera:

- h.objeto es un uicontrol, el cual es asociado:
  - Texto, de la forma ("style", "text")
  - Entrada de datos, de la forma ("style", "edit")
  - Botones, de la forma ("style", "pushbutton")
- Cada uno de estos objetos, lleva asociado una subrutina, la cual es llamada cuando es apretado un boton, para nuestro caso, esto se realiza de la siguiente forma ("callback", "@update_plot")

Bajo la estructura siguiente:

```
switch (objeto_n)
    case {objeto_1}
        %hace algo
    %...
    case {objeto_n}
        %hace algo
endswitch
```

Mediante la función *get(h.edit_matrix, "string")* obtenemos del objeto *h.salida_matrix* la matriz que inserta el usuario.

Este resultado tiene que ser convertido a un número, de modo que las funciones de los capitulos anteriores, puedan operar, mediante la funcion *str2num()* convierte las palabras en números.

Cada *objeto_n*, el cual es un botón, tiene una función asociada, la cual realiza diferentes acciones, que son las funciones que se ha visto en capitulos anteriores ,según sea el objeto_n.

Los resultados obtenidos son expresados mediante los objetos "*h.salida_matrix*" que son de tipo "text", previa conversion con "*num2str()*" la cual estos objetos de tipo texto, puedan leer el resultado para el usuario [3].

```
close all
clear h

function update_plot (obj, init = false)

  %% gcbo holds the handle of the control
  h = guidata (obj);

  switch (gcbo)
    case {h.edit_matrix_A}
      v = get (h.edit_matrix_A, "string");
      set (h.salida_matrix_R1,"string", v);
    case {h.boton_determinante}
      A = get (h.edit_matrix_A, "string");
      A = str2num(A);
      det=determinante(A);
      det = num2str(det);
      set (h.label_matrix_R1, "string", "Determinante");
      set (h.salida_matrix_R1,"string",det);
    case {h.boton_inversa}
      A = get (h.edit_matrix_A, "string");
      A = str2num(A);
      inv=inversa(A);
      inv = num2str(inv);
      set (h.label_matrix_R1, "string", "Inversa");
      set (h.salida_matrix_R1,"string",inv);
    case {h.boton_gram_Schmith}
      A = get (h.edit_matrix_A, "string");
      A = str2num(A);
```

```
    u=gramSchmith(A);
    u = num2str(u);
    set (h.label_matrix_R1, "string", "Matriz ortonormalizada");
    set (h.salida_matrix_R1,"string",u);
  case {h.boton_leverrier}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    [V,D]=algoritmoLeverrier(A);
    V = num2str(V);
    D = num2str(D);
    set (h.label_matrix_R1, "string", "Eigen Vectores");
    set (h.salida_matrix_R1,"string",V);
    set (h.label_matrix_R2, "string", "Matriz Diagonal");
    set (h.salida_matrix_R2,"string",D);
  case {h.boton_descomposicionCholesky}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    [G,Gt]=descomposicionCholesky(A);
    G = num2str(G);
    Gt = num2str(Gt);
    set (h.label_matrix_R1, "string", "Matrix G");
    set (h.salida_matrix_R1,"string",G);
    set (h.label_matrix_R2, "string", "Matrix Gt");
    set (h.salida_matrix_R2,"string",Gt);
  case {h.boton_solucionCholesky}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    X=solucionCholesky(A,B);
    X=num2str(X);
    set (h.label_matrix_R1, "string", "Vector solucion");
```

```
    set (h.salida_matrix_R1,"string",X);
  case {h.boton_cambio_de_Base}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    M12=cambioBase12(A,B);
    M12 = num2str(M12);
    set (h.label_matrix_R1, "string", "Matriz cambio de base 1 a 2");
    set (h.salida_matrix_R1,"string",M12);
  case {h.boton_descomposicionLU}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    [L,U]=descomposicionLU(A);
    L = num2str(L);
    U = num2str(U);
    set (h.label_matrix_R1, "string", "Matrix L");
    set (h.salida_matrix_R1,"string",L);
    set (h.label_matrix_R2, "string", "Matrix U");
    set (h.salida_matrix_R2,"string",U);
  case {h.boton_solucionLU}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    X=solucionLU(A,B);
    X=num2str(X);
    set (h.label_matrix_R1, "string", "Vector solucion");
    set (h.salida_matrix_R1,"string",X);
  case {h.boton_descomposicionPLU}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
```

```
    [P,L,U]=descomposicionPLU(A);
    P = num2str(P);
    L = num2str(L);
    U = num2str(U);
    set (h.label_matrix_R1, "string", "Matrix P");
    set (h.salida_matrix_R1,"string",P);
    set (h.label_matrix_R2, "string", "Matrix L");
    set (h.salida_matrix_R2,"string",L);
    set (h.label_matrix_R3, "string", "Matrix U");
    set (h.salida_matrix_R3,"string",U);
  case {h.boton_solucionPLU}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    X=solucionPLU(A,B);
    X=num2str(X);
    set (h.label_matrix_R1, "string", "Vector solucion");
    set (h.salida_matrix_R1,"string",X);
  case {h.boton_descomposicionQR}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    [Q,R]=descomposicionQR(A);
    Q = num2str(Q);
    R = num2str(R);
    set (h.label_matrix_R1, "string", "Matrix Q");
    set (h.salida_matrix_R1,"string",Q);
    set (h.label_matrix_R2, "string", "Matrix R");
    set (h.salida_matrix_R2,"string",R);
  case {h.boton_elim_gausiana_progresiva}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
```

```
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    X=elim_gauss_progresivo(A,B);
    X=num2str(X);
    set (h.label_matrix_R1, "string", "Vector solucion");
    set (h.salida_matrix_R1,"string",X);
  case {h.boton_elim_gausiana_regresiva}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    X=elim_gauss_regresivo(A,B);
    X=num2str(X);
    set (h.label_matrix_R1, "string", "Vector solucion");
    set (h.salida_matrix_R1,"string",X);
  case {h.boton_regresion_N}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    n = get (h.edit_n, "string");
    n = str2num(n);
    coef = regresionNat(A,B,n);
    coef = num2str(coef);
    set (h.label_matrix_R1, "string", "Coeficientes :: a0+a1X+...+anX^n");
    set (h.salida_matrix_R1,"string",coef);
  case {h.boton_regresion_QR}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    n = get (h.edit_n, "string");
```

```
    n = str2num(n);
    coef = regresionQR(A,B,n);
    coef = num2str(coef);
    set (h.label_matrix_R1, "string", "Coeficientes :: a0+a1X+...+anX^n");
    set (h.salida_matrix_R1,"string",coef);
  case {h.boton_rango}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    c = rango(A);
    c = num2str(c);
    set (h.label_matrix_R1, "string", "Rango de la Matrix");
    set (h.salida_matrix_R1,"string",c);
  case {h.boton_rango_aumentada}
    A = get (h.edit_matrix_A, "string");
    A = str2num(A);
    B = get (h.edit_matrix_B, "string");
    B = str2num(B);
    [v1,v2] = rango_aumentado(A,B);
    if(v1)
      if(v2)
        set (h.label_matrix_R1, "string", "El sistema tiene:");
        set (h.salida_matrix_R1,"string", "solucion unica");
      else
        set (h.label_matrix_R1, "string", "El sistema tiene:");
        set (h.salida_matrix_R1,"string", "infita soluciones");
      endif
    else
      set (h.label_matrix_R1, "string", "El sistema tiene:");
      set (h.salida_matrix_R1,"string", "no tiene solucion");
    endif
  case {h.boton_suma}
    A = get (h.edit_matrix_A, "string");
```

```
      A = str2num(A);
      B = get (h.edit_matrix_B, "string");
      B = str2num(B);
      X=suma(A,B);
      X=num2str(X);
      set (h.label_matrix_R1, "string", "A+B");
      set (h.salida_matrix_R1,"string",X);
    case {h.boton_producto}
      A = get (h.edit_matrix_A, "string");
      A = str2num(A);
      B = get (h.edit_matrix_B, "string");
      B = str2num(B);
      X=producto(A,B);
      X=num2str(X);
      set (h.label_matrix_R1, "string", "A*B");
      set (h.salida_matrix_R1,"string",X);
    case {h.boton_limpiar}
      set (h.edit_matrix_A,"string", " ");
      set (h.edit_matrix_B,"string", " ");
      set (h.edit_n,"string", " ");
      set (h.salida_matrix_R1,"string", " ");
      set (h.salida_matrix_R2,"string", " ");
      set (h.salida_matrix_R3,"string", " ");
  endswitch

endfunction

%%%% datos de entrada %%%%%%%%%%%%
h.label_matrix_A = uicontrol ("style", "text",
                                "units", "normalized",
                                "string", "Matrix A",
                                "horizontalalignment", "left",
```

```
                              "position", [0.05 0.95 0.4 0.05]);
h.edit_matrix_A = uicontrol ("style", "edit",
                             "units", "normalized",
                             "string", "Please fill me! (edit)",
                             "callback", @update_plot,
                             "position", [0.05 0.9 0.45 0.05]);
h.label_matrix_B = uicontrol ("style", "text",
                              "units", "normalized",
                              "string", "Matrix B",
                              "horizontalalignment", "left",
                              "position", [0.05 0.85 0.4 0.05]);
h.edit_matrix_B = uicontrol ("style", "edit",
                             "units", "normalized",
                             "string", "Please fill me! (edit)",
                             "callback", @update_plot,
                             "position", [0.05 0.8 0.45 0.05]);
h.label_n = uicontrol ("style", "text",
                              "units", "normalized",
                              "string", "regresion polinomial orden n:",
                              "horizontalalignment", "left",
                              "position", [0.05 0.75 0.3 0.05]);
h.edit_n = uicontrol ("style", "edit",
                             "units", "normalized",
                             "string", "-",
                             "callback", @update_plot,
                             "position", [0.36 0.75 0.1 0.047]);
%%%%% datos de salida %%%%%%%%%%%%%%%%%%%%%%%%%
h.label_matrix_R1 = uicontrol ("style", "text",
                              "units", "normalized",
                              "string", "Matrix Rpta",
                              "horizontalalignment", "left",
                              "position", [0.05 0.7 0.4 0.05]);
```

```
h.salida_matrix_R1 = uicontrol ("style", "text",
                               "units", "normalized",
                               "string", "Matrix de salida R1",
                               "callback", @update_plot,
                               "position", [0.05 0.5 0.45 0.2]);
h.label_matrix_R2 = uicontrol ("style", "text",
                                "units", "normalized",
                                "string", "Matrix Rpta",
                                "horizontalalignment", "left",
                                "position", [0.05 0.45 0.4 0.05]);
h.salida_matrix_R2 = uicontrol ("style", "text",
                               "units", "normalized",
                               "string", "Matrix de salida R2",
                               "callback", @update_plot,
                               "position", [0.05 0.25 0.45 0.2]);
h.label_matrix_R3 = uicontrol ("style", "text",
                                "units", "normalized",
                                "string", "Matrix Rpta",
                                "horizontalalignment", "left",
                                "position", [0.05 0.2 0.4 0.05]);
h.salida_matrix_R3 = uicontrol ("style", "text",
                               "units", "normalized",
                               "string", "Matrix de salida R3",
                               "callback", @update_plot,
                               "position", [0.05 0.0 0.45 0.2]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Funciones en Botones :: operador unario :: Matrix A %%%%%%
h.label_unario = uicontrol ("style", "text",
                                "units", "normalized",
                                "string", "Operaciones\nUnarias",
                                "horizontalalignment", "left",
                                "position", [0.51 0.91 0.24 0.09]);
```

```
h.boton_descomposicionCholesky = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "descomposicion\nCholesky",
                                "callback", @update_plot,
                                "position", [0.51 0.8 0.24 0.09]);
h.boton_descomposicionLU = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "descomposicion\nLU",
                                "callback", @update_plot,
                                "position", [0.51 0.7 0.24 0.09]);
h.boton_descomposicionQR = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "descomposicion\nQR",
                                "callback", @update_plot,
                                "position", [0.51 0.6 0.24 0.09]);
h.boton_descomposicionPLU = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "descomposicion\nPLU",
                                "callback", @update_plot,
                                "position", [0.51 0.5 0.24 0.09]);
h.boton_determinante = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "determinante",
                                "callback", @update_plot,
                                "position", [0.51 0.45 0.24 0.045]);
h.boton_inversa = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "inversa",
                                "callback", @update_plot,
                                "position", [0.51 0.4 0.24 0.045]);
h.boton_gram_Schmith = uicontrol ("style", "pushbutton",
                                "units", "normalized",
```

```
                                "string", "Gram Schmith",
                                "callback", @update_plot,
                                "position", [0.51 0.35 0.24 0.045]);
h.boton_leverrier = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Leverrier",
                                "callback", @update_plot,
                                "position", [0.51 0.3 0.24 0.045]);
h.boton_regresion_N = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Regresion\npolinomica",
                                "callback", @update_plot,
                                "position", [0.51 0.2 0.24 0.09]);
h.boton_rango = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Rango de la\nMatriz",
                                "callback", @update_plot,
                                "position", [0.51 0.1 0.24 0.09]);
h.boton_limpiar = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Limpiar",
                                "callback", @update_plot,
                                "position", [0.51 0.02 0.24 0.07]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Funciones en Botones :: operador binario :: Matrix A %%%%%%
h.label_binario = uicontrol ("style", "text",
                                "units", "normalized",
                                "string", "Operaciones\nBinarias",
                                "horizontalalignment", "left",
                                "position", [0.75 0.91 0.24 0.09]);
h.boton_cambio_de_Base = uicontrol ("style", "pushbutton",
                                "units", "normalized",
```

```
                                "string", "cambio de\nBase 1 a 2",
                                "callback", @update_plot,
                                "position", [0.75 0.8 0.24 0.09]);
h.boton_solucionCholesky = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Solucion por\nCholesky",
                                "callback", @update_plot,
                                "position", [0.75 0.7 0.24 0.09]);
h.boton_solucionLU = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Solucion por\nLU",
                                "callback", @update_plot,
                                "position", [0.75 0.6 0.24 0.09]);
h.boton_solucionPLU = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Solucion por\nPLU",
                                "callback", @update_plot,
                                "position", [0.75 0.5 0.24 0.09]);
h.boton_elim_gausiana_progresiva = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Eliminacion de Gauss\nprogresiva",
                                "callback", @update_plot,
                                "position", [0.75 0.4 0.24 0.09]);
h.boton_elim_gausiana_regresiva = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Eliminacion de Gauss\nregresiva",
                                "callback", @update_plot,
                                "position", [0.75 0.3 0.24 0.09]);
h.boton_regresion_QR = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Regresion\nQR",
                                "callback", @update_plot,
```

```
                                "position", [0.75 0.2 0.24 0.09]);
h.boton_rango_aumentada = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "Rango Matriz\nAumentada",
                                "callback", @update_plot,
                                "position", [0.75 0.1 0.24 0.09]);
h.boton_suma = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "+",
                                "callback", @update_plot,
                                "position", [0.75 0.02 0.1 0.07]);
h.boton_producto = uicontrol ("style", "pushbutton",
                                "units", "normalized",
                                "string", "*",
                                "callback", @update_plot,
                                "position", [0.85 0.02 0.1 0.07]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set (gcf, "color", get(0, "defaultuicontrolbackgroundcolor"))
guidata (gcf, h)
update_plot (gcf, true)
```

Capítulo 6

Ejecución

En windows, simplemente hagar doble click sobre el icono de **Octave** y para ubuntu, en la terminal teclear **Octave&** y una vez en el Octave, dirigirnos a la carpeta donde estan nuestros archivos.

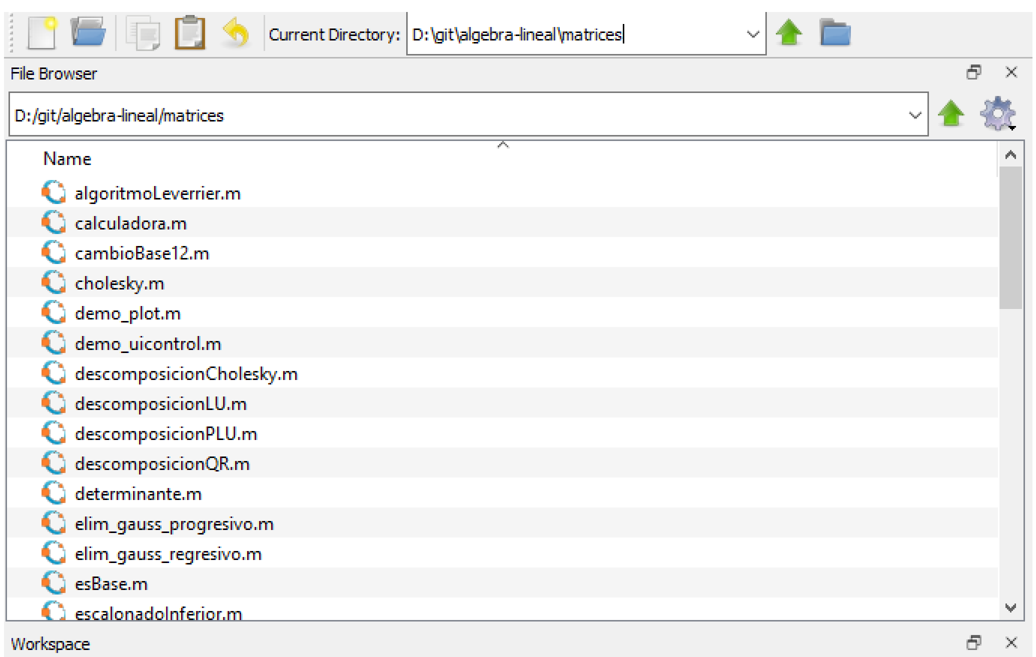

Figura 6.1: Carpeta de archivos

Luego en la ventana de comandos llamamos a **calculadora.m**

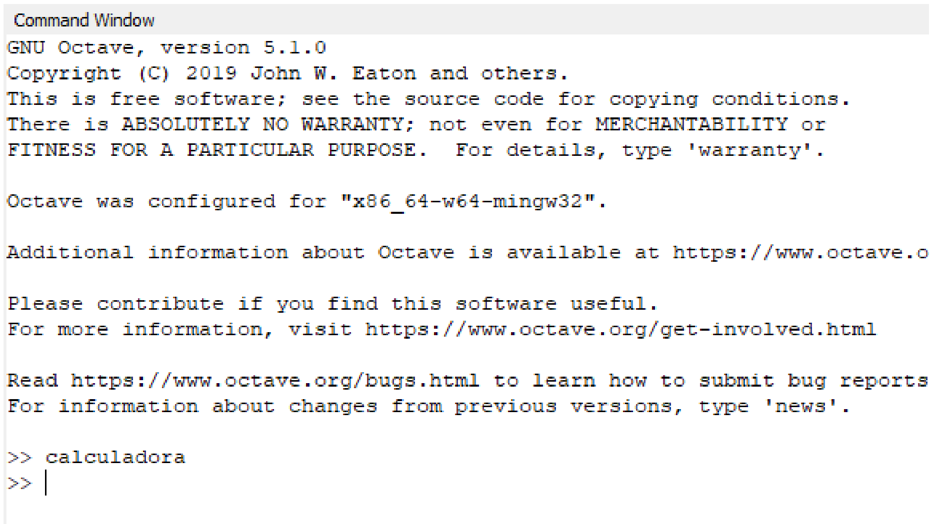

Figura 6.2: Ventana de comandos

El cual nos da la siguiente ventana

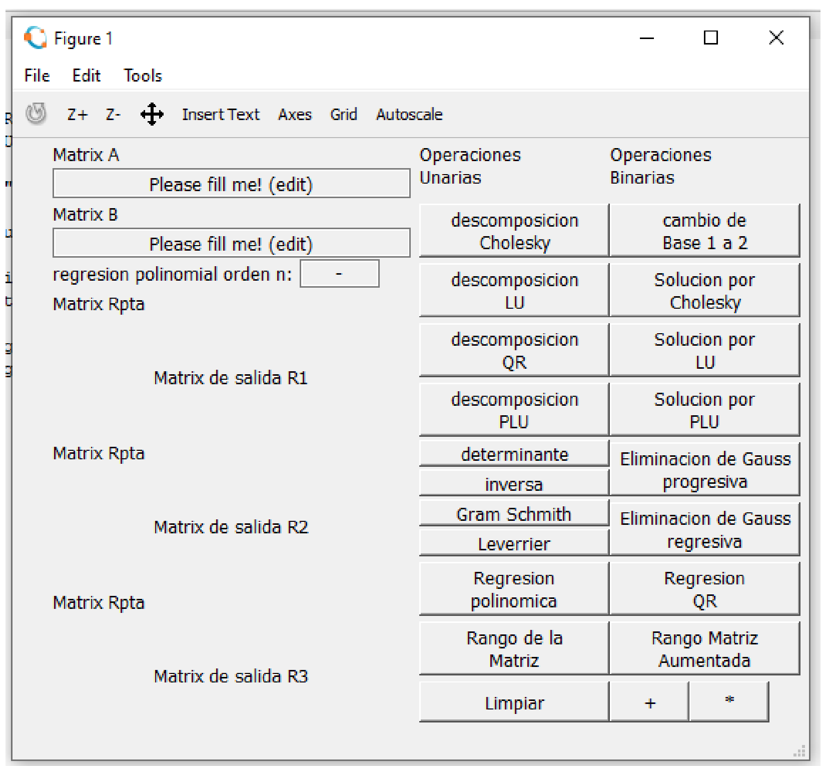

Figura 6.3: Calculadora de matrices

6.1. Ejemplos

Descomposición PLU: Vamos a descomponer la siguiente matriz.

$$\begin{pmatrix} -1 & 2 & -1 & 0 \\ 2 & -1 & 1 & 1 \\ 4 & 1 & -2 & 1 \\ 3 & 2 & -3 & 4 \end{pmatrix}$$

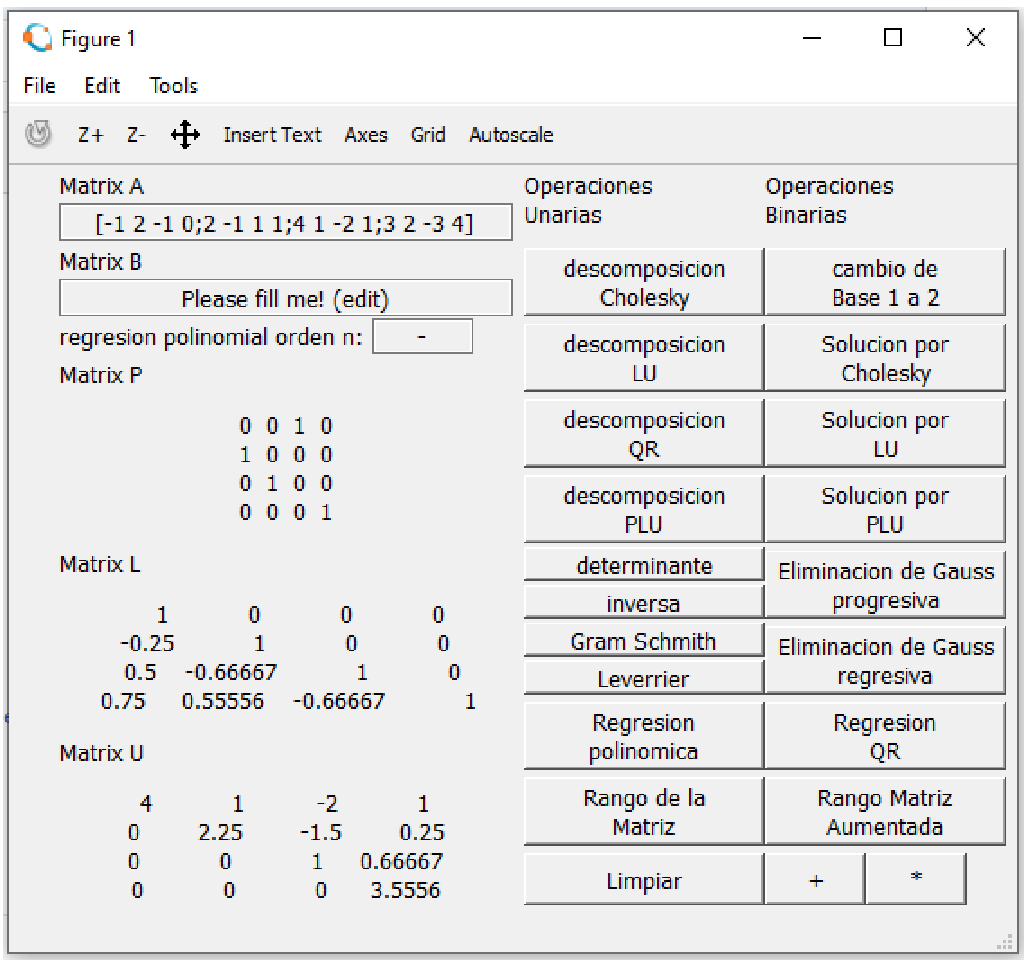

Figura 6.4: Descomposición PLU

Eigen valores y eigen vectores: Tomaremos la siguiente matriz para obtener los autovalores y autovectores.

$$\begin{pmatrix} 3 & -5 \\ 1 & -1 \end{pmatrix}$$

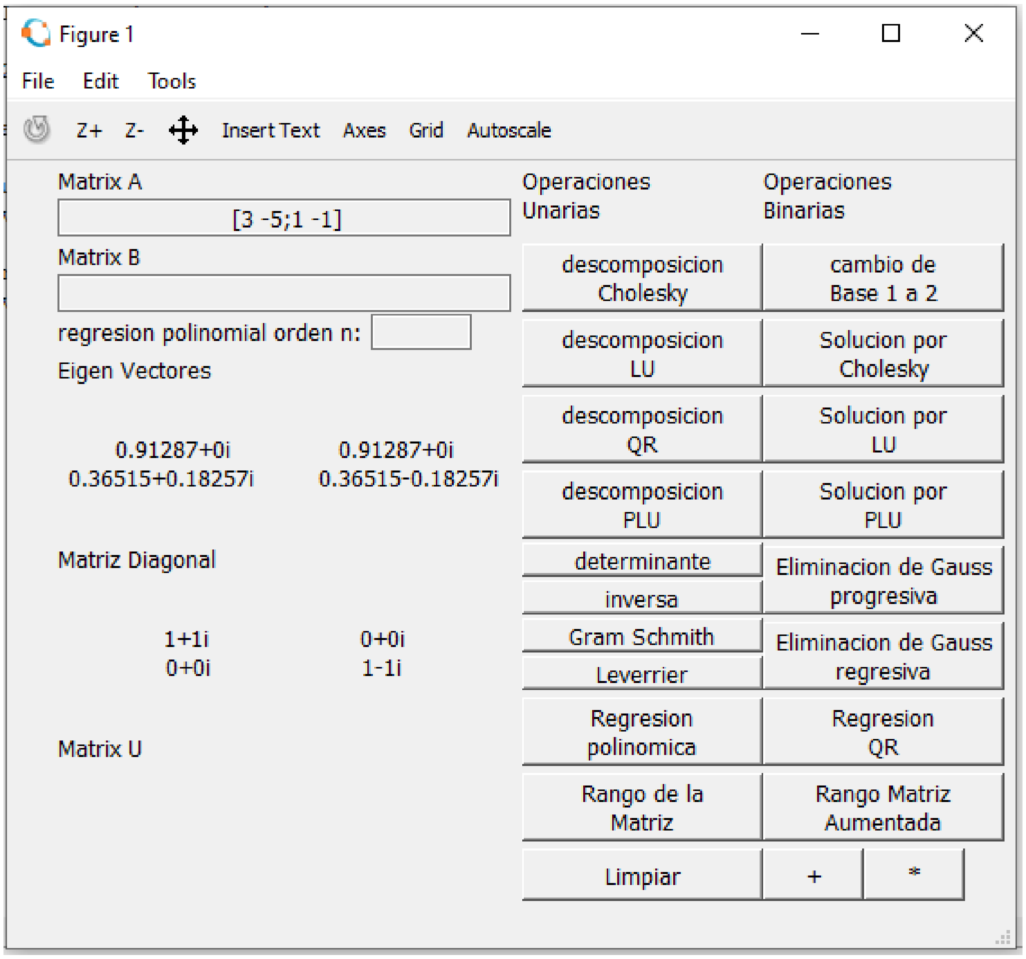

Figura 6.5: Algoritmo de Leverrier

Regresion QR: Tomaremos como ejemplo las siguientes tablas X & Y para hacer la regresion lineal para $n = 1$.

$$X = \begin{pmatrix} 20 & 30 & 30 & 40 & 50 & 60 & 60 & 60 & 70 & 80 \end{pmatrix}$$

$$Y = \begin{pmatrix} 50 & 73 & 69 & 87 & 108 & 128 & 135 & 132 & 148 & 170 \end{pmatrix}$$

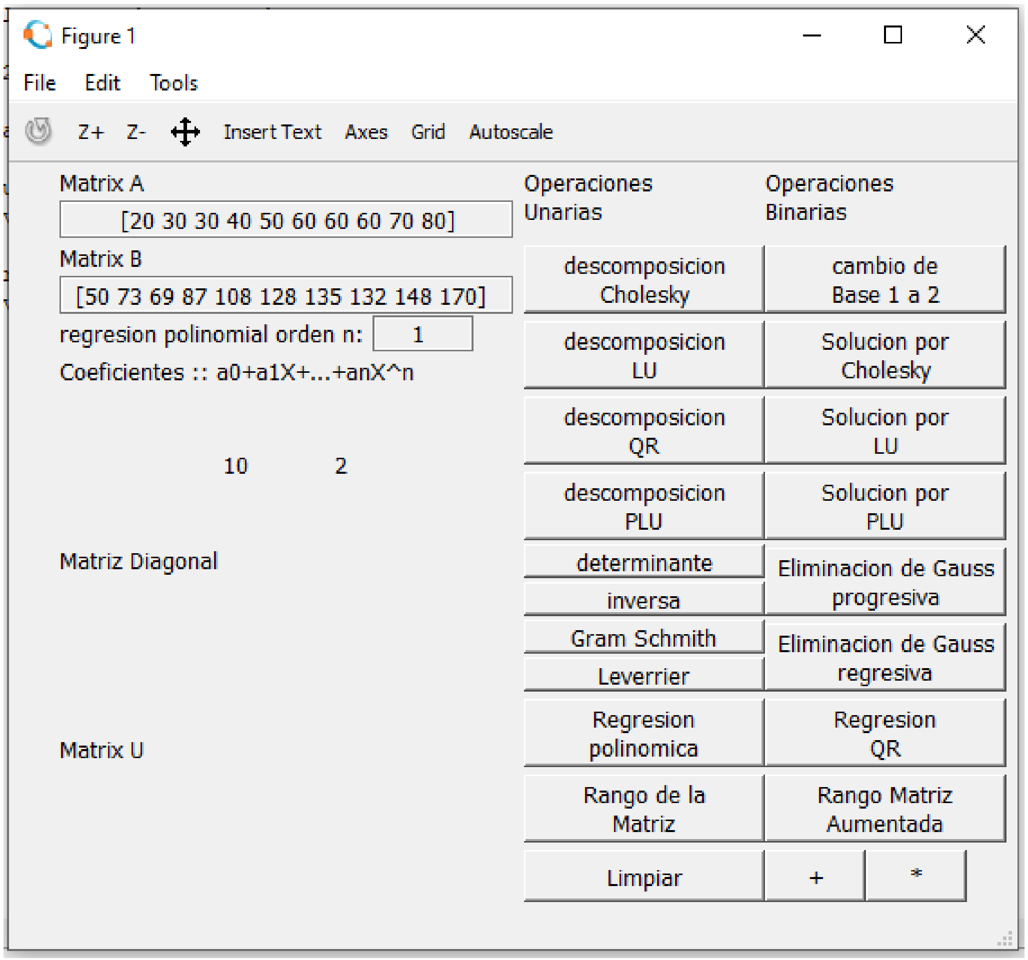

Figura 6.6: Regresion QR

teniendo la siguiente ecuación:

$$Y = 10 + 2X$$

Bibliografía

[1] Bernard Kolman y David R Hill. *Introductory linear algebra: An applied first course*. Prentice-Hall, 2005.

[2] octave. https://octave.org/doc/interpreter/UI-Elements.html#XREFuicontrol, 2019.

[3] octave. https://wiki.octave.org/Uicontrols, 2019.