UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INFORMÁTICA E INDUSTRIAL

MATERIA: Cálculo Numérico

Taller #1

Profesor: Jorge Omar

Integrantes:

Karina Gonzalez. CI: V-28318100

Nicolas Setién. CI: V-30395284

Luis Sierra. CI: V-28329965

Junio 2022

Ejercicio 1:

Parte 1.a)

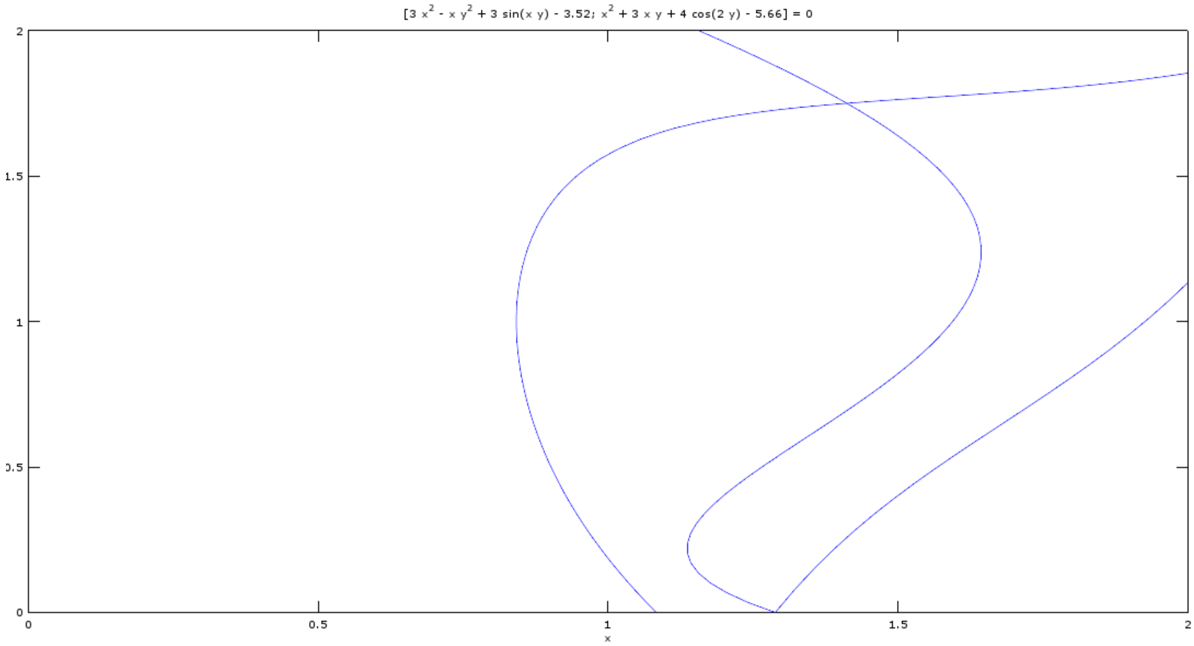
1. Se introduce las ecuaciones dadas a Octave

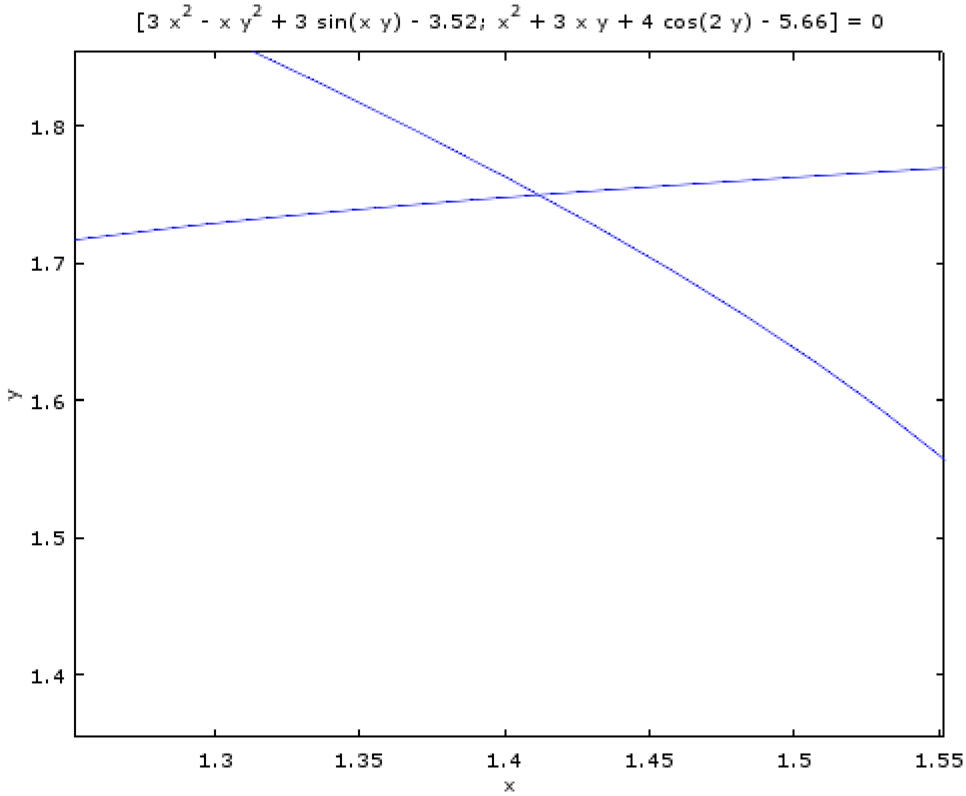
```
>> F=@(x,y)[3*x.^2-x.*y.^2+3*sin(x.*y)-3.52; x.^2+3*x.*y+4*cos(2*y)-5.66]
F =

@(x, y) [3 * x .^ 2 - x .* y .^ 2 + 3 * sin(x .* y) - 3.52; x .^ 2 + 3 * x .* y + 4 * cos(2 * y) - 5.66]
```

2. Se grafica las funciones para obtener el vector que permita la convergencia

```
>> ezplot(F,[0,2])
```





Al acercarse a la raíz, podemos observar valores cercanos que permitan armar vector inicial para garantizar la convergencia:

Vector Inicial: $\mathbf{X^{(0)}=[1.39; 1.76]^T}$

3. Usando el método de Newton nos da los siguientes resultados en **5 iteraciones:**

```
raiz =

   1.41177621728269
   1.75012630679511

niter =   5
```

Donde en cada interacciones se consiguieron los siguientes valores:

Iteración #1

```
jac =

   1.18773034036401  -8.09539403937798
   8.06009999998025   7.12627105199815

f0 =

  -0.107763812153140
  -0.105716821376493

dx =

   0.0220280875075701
  -0.0100798532947601

x =

   1.41202808750757
   1.74992014670524

error =  0.0220280875075701
```

Iteración #2

```
jac =

   1.29722687408407  -8.26080745837121
   8.07391661512114   7.04190277327044

f0 =

  2.02929735286883e-003
  5.81723076622964e-004

dx =

  -2.51814707895470e-004
  2.06110184152980e-004

x =

   1.41177627279967
   1.75012625688939

error =   2.51814707895470e-004
```

Iteración #3

```
jac =

   1.29499851031500  -8.25952220999238
   8.07403131625861   7.04423537190024

f0 =

  4.84074643480881e-007
  9.67300497478618e-008

dx =

  -5.55188553397869e-008
  4.99033476806696e-008

x =

   1.41177621728082
   1.75012630679274

error =   5.55188553397869e-008
```

Iteración #4

```
jac =

   1.29499797239419  -8.25952195576019
   8.07403135491214   7.04423595296433

f0 =

  1.71413994110026e-011
  -3.18154391720782e-011

dx =

  1.87353151293733e-012
  2.36909823913315e-012

x =

   1.41177621728269
   1.75012630679511

error =   2.36909823913315e-012
```

Iteración #5

```
jac =

   1.29499797236754  -8.25952195579127
   8.07403135494766   7.04423595299986

f0 =

  2.22044604925031e-015
 -2.66453525910038e-015

dx =

  8.39790145997156e-017
  2.82001635850755e-016

x =

   1.41177621728269
   1.75012630679511

error =   2.82001635850755e-016
```

Parte 1.b)

Iteraccion 1	jac = 1.18773034036401 -8.09539403937798 8.06009999998025 7.12627105199815	f0 = -0.107763812153140 -0.105716821376493	dx = 0.0220280875075701 -0.0100798532947601	x = 1.41202808750757 1.74992014670524	error = 0.0220280875075701
Iteraccion 2	jac = 1.29722687408407 -8.26080745837121 8.07391661512114 7.04190277327044	f0 = 2.02929735286883e-003 5.81723076622964e-004	dx = -2.51814707895470e-004 2.06110184152980e-004	x = 1.41177627279967 1.75012625688939	error = 2.51814707895470e-004
Iteraccion 3	jac = 1.29499851031500 -8.25952220999238 8.07403131625861 7.04423537190024	f0 = 4.84074643480881e-007 9.67300497478618e-008	x = 1.41177621728082 1.75012630679274	dx = -5.55188553397869e-008 4.99033476806696e-008	error = 5.55188553397869e-008
Iteraccion 4	jac = 1.29499797239419 -8.25952195576019 8.07403135491214 7.04423595296433	f0 = 2.22044604925031e-015 -2.66453525910038e-015	dx = 1.87353151293733e-012 2.36909823913315e-012	x = 1.41177621728269 1.75012630679511	error = 2.36909823913315e-012
Iteraccion 5	jac = 1.29499797236754 -8.25952195579127 8.07403135494766 7.04423595299986	f0 = 1.71413994110026e-011 -3.18154391720782e-011	dx = 8.39790145997156e-017 2.82001635850755e-016	x = 1.41177621728269 1.75012630679511	error = 2.82001635850755e-016

<u>Ejercicio 2:</u>

Parte a:

1. **Evaluando la función para los valores de a desde 4 hasta 6 sumando progresivamente 0.5**

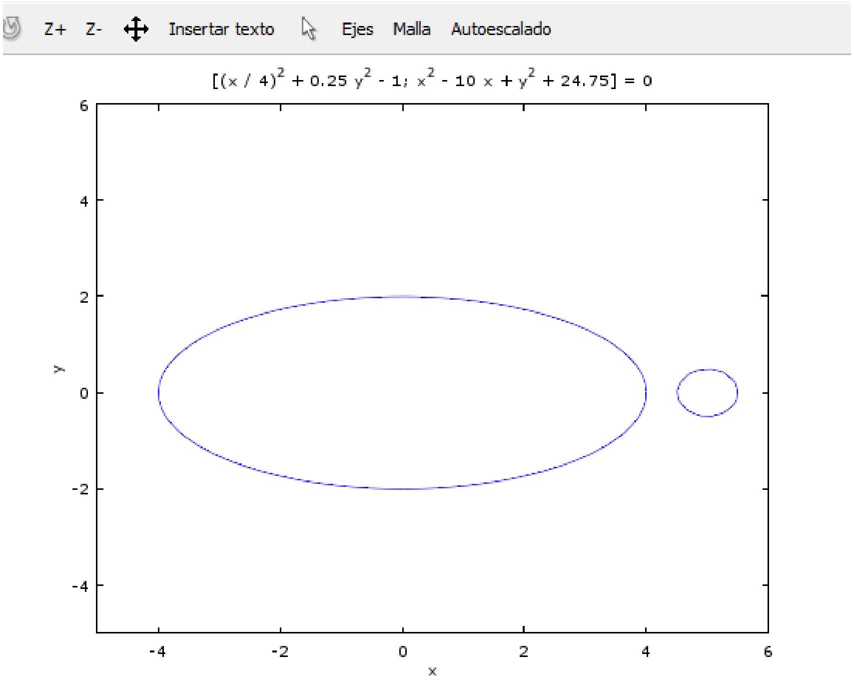
- Se define la función, utilizando a=4

```
>> f=@(x,y)[(x/4).^2+0.25*y.^2-1;x.^2-10*x+y.^2+24.75]
f =

@(x, y) [(x / 4) .^ 2 + 0.25 * y .^ 2 - 1; x .^ 2 - 10 * x + y .^ 2 + 24.75]
```

- Se grafica la función, en este caso tomando un intervalo de [-5,6]

```
>> ezplot(f,[-5,6])
```



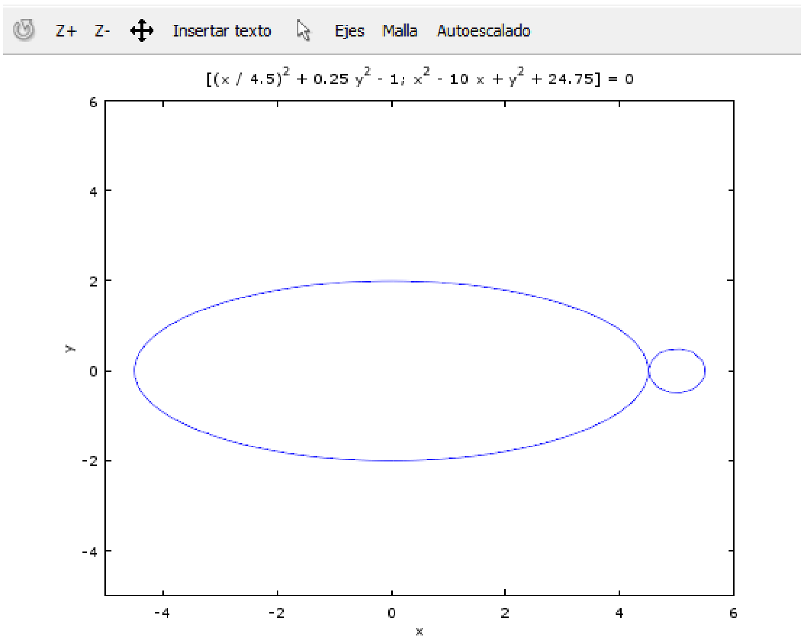
Como en este caso, al observar la gráfica se aprecia que el sistema no posee dos soluciones, se procede a evaluar la función con el siguiente valor de a posible.

- Se define la función, utilizando a=4.5

```
>> f=@(x,y)[(x/4.5).^2+0.25*y.^2-1;x.^2-10*x+y.^2+24.75]
f =

@(x, y) [(x / 4.5) .^ 2 + 0.25 * y .^ 2 - 1; x .^ 2 - 10 * x + y .^ 2 + 24.75]
```

- Se grafica la función, utilizando el mismo intervalo anterior



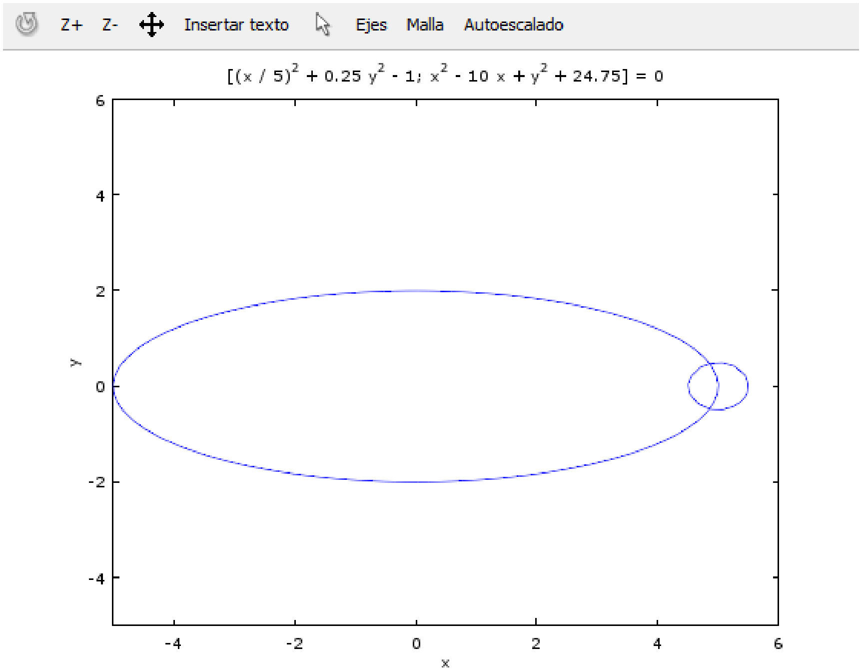
En vista de no obtenerse aún dos soluciones para el sistema, se continúa evaluando la función con otro valor de a.

- Se define la función, utilizando a=5.

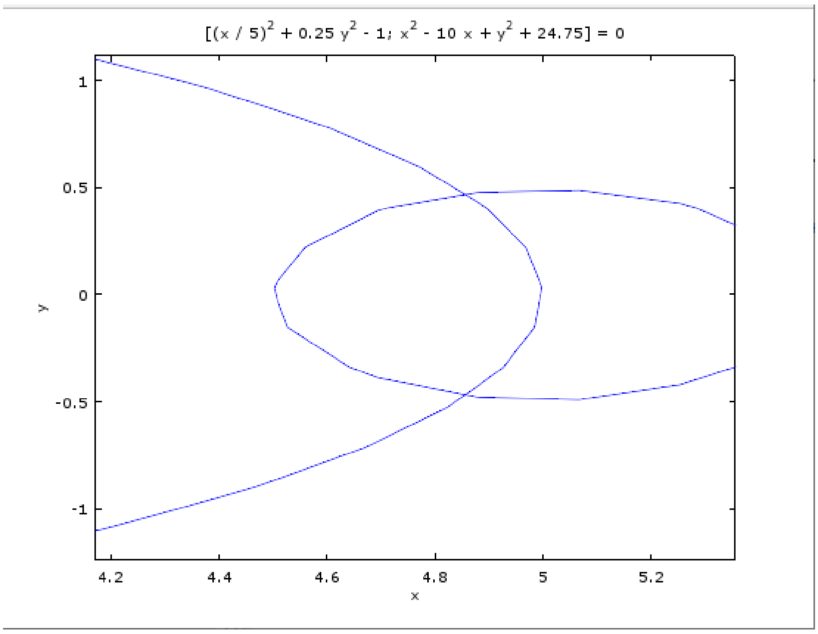
```
>> f=@(x,y)[(x/5).^2+0.25*y.^2-1;x.^2-10*x+y.^2+24.75]
f =

@(x, y) [(x / 5) .^ 2 + 0.25 * y .^ 2 - 1; x .^ 2 - 10 * x + y .^ 2 + 24.75]
```

- Se grafica la función:



Es posible observar que para este valor de a, el sistema posee dos soluciones.



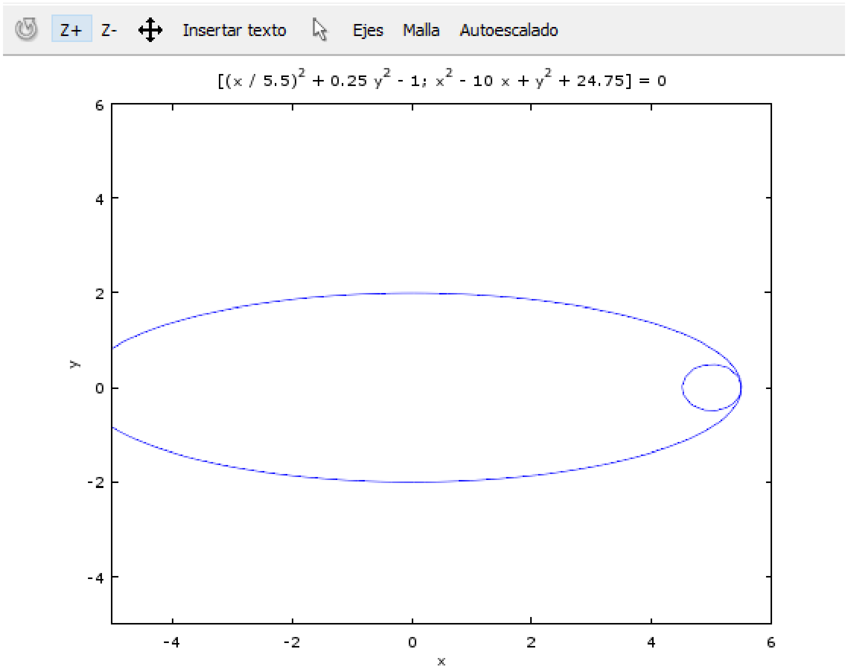
De igual forma, se realizan las iteraciones con los valores de a restantes, a fin de comprobar que efectivamente a=5 es el valor solicitado.

- Se define la función, utilizando a=5.5

```
>> f=@(x,y)[(x/5.5).^2+0.25*y.^2-1;x.^2-10*x+y.^2+24.75]
f =

@(x, y) [(x / 5.5) .^ 2 + 0.25 * y .^ 2 - 1; x .^ 2 - 10 * x + y .^ 2 + 24.75]
```

- Se grafica la función

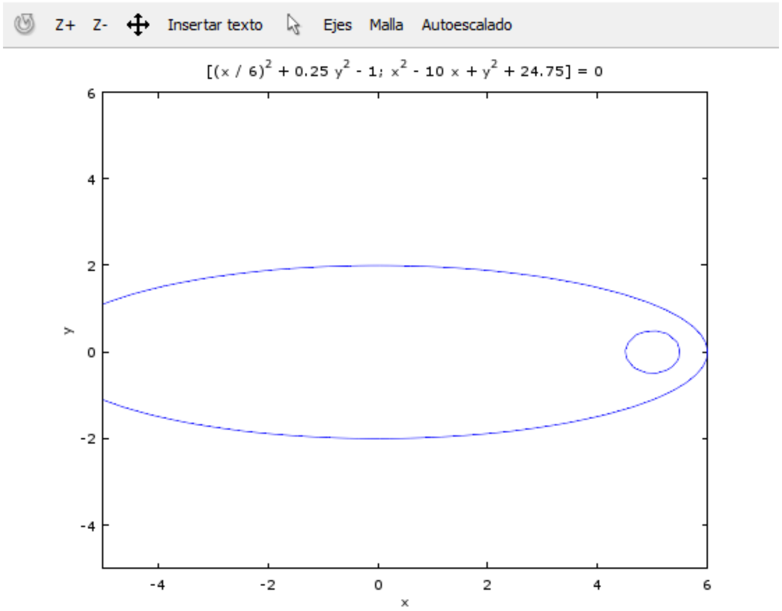


- Se define la función, utilizando a=6.

```
>> f=@(x,y)[(x/6).^2+0.25*y.^2-1;x.^2-10*x+y.^2+24.75]
f =

@(x, y) [(x / 6) .^ 2 + 0.25 * y .^ 2 - 1; x .^ 2 - 10 * x + y .^ 2 + 24.75]
```

- Se realiza la gráfica:

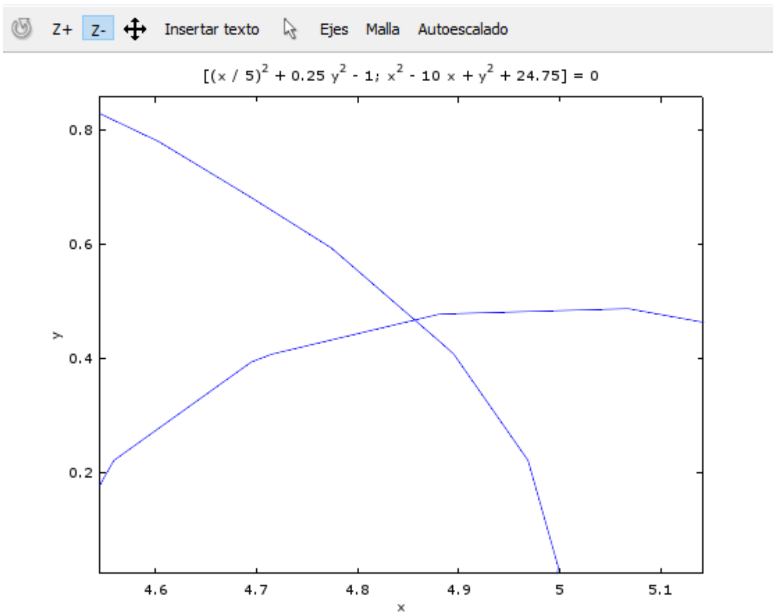


Una vez evaluados y comprobados los resultados, resulta posible determinar que el valor solicitado de a, para cumplir con la condición de que el sistema tenga dos soluciones, es **a=5.**

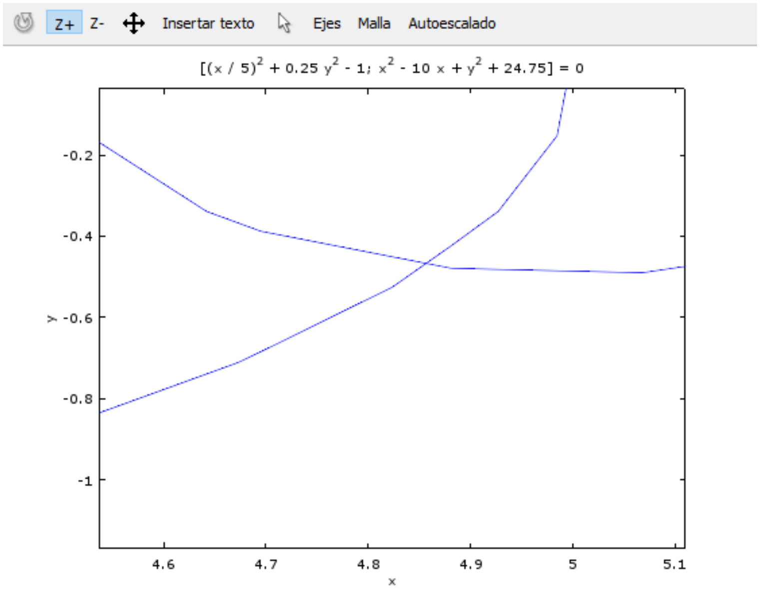
Parte b:

2. **Aproximando las soluciones, utilizando el método de Newton**

- Hallando los vectores iniciales para cada raíz



Vector inicial de la primera raíz: $\mathbf{X^{(0)}=[4.8;0.4]^T}$



Vector inicial de la segunda raíz: $X^{(0)}=[4.8;-0.4]^T$

- Empleando la función newtonRaphson, ingresando como parámetros la función antes definida, los vectores iniciales para cada raíz, y una tolerancia de 0.00000001, que corresponde a 10^-8.
- Se aplicó "format long" para obtener resultados con mayor precisión.

Primera raíz:

```
>> [raiz,niter] = newtonRaphson2(f,[4.8;0.4],0.00000001)
```

Resultado obtenido:

```
raiz =

   4.854816136036558
   0.478457569325275

niter =  5
```

Valores obtenidos por cada iteración:

Primera iteración:

```
jac =

   0.384003999998050   0.200025000000048
  -0.399900000012110   0.800100000013515

f0 =

  -0.038400000000000
  -0.050000000000007

dx =

   0.053514718614319
   0.089239515027258

x =

   4.853514718614319
   0.489239515027258

error =  0.0892395150272575
```

Segunda iteración:

```
jac =

   0.388285177486125   0.244644757512180
  -0.292870562716985   0.978579030075366

f0 =

   0.00210303071825990
   0.0108132407267440 6

dx =

   0.00130070595395584
  -0.0106606640051097 6

x =

   4.854815424568275
   0.478578851022148

error =  0.0106606640051098
```

Tercera iteración:

```
jac =

   0.388389233962894   0.239314425511949
  -0.290269150866607   0.957257702047798

f0 =

  2.87414264614760e-005
  1.16277588965374e-004

dx =

  7.11500465754497e-007
  -1.21253725178743e-004

x =

   4.854816136068741
   0.478457597296969

error =   1.21253725178743e-004
```

<u>*Cuarta iteración:*</u>

```
jac =

   0.388389290884028   0.239253798648420
  -0.290267727862670   0.957015194593680

f0 =

  6.70413391468117e-009
  2.67571991230398e-008

dx =

  -3.21812206580491e-011
  -2.79687725378324e-008

x =

   4.854816136036559
   0.478457569328196

error =   2.79687725378324e-008
```

<u>*Quinta iteración:*</u>

```
jac =

   0.388389290881808   0.239253784664051
  -0.290267727827143   0.957015138638440

f0 =

  6.99440505513849e-013
  2.79598566521599e-012

dx =

  -9.63408616418436e-016
  -2.92186110621441e-012

x =

   4.854816136036558
   0.478457569325275

error =   2.92186110621441e-012
```

<u>Segunda raíz:</u>

```
>> [raiz,niter] = newtonRaphson2(f,[4.8;-0.4],0.00000001)
```

Resultado obtenido:

```
raiz =

   4.854816136036560
  -0.478457569325022

niter =   4
```

Valores obtenidos por cada iteración:

<u>Primera iteración:</u>

```
jac =

   0.384003999998050  -0.199975000000352
  -0.399900000012110  -0.799899999996967

f0 =

  -0.038400000000000
  -0.050000000000007

dx =

   0.053514718613499 8
  -0.089261827696534 7

x =

   4.853514718613500
  -0.489261827696535

error =   0.0892618276965347
```

Segunda iteración:

```
jac =

   0.388285177488346  -0.244605913848339
  -0.292870562752512  -0.978423655411120

f0 =

   0.00210848896215365
   0.01083507370383430

dx =

   0.00130070595479643
   0.01068467137017945

x =

   4.854815424568296
  -0.478577156326355

error =   0.0106846713701795
```

Tercera iteración:

```
jac =

   0.388389233967334  -0.239263578163396
  -0.290269150866607  -0.957054312671346

f0 =

  2.83359044048748e-005
  1.14655500706107e-004

dx =

  7.11500446772516e-007
  1.19584617675594e-004

x =

   4.854816136068743
  -0.478457571708680

error =   1.19584617675594e-004
```

Cuarta iteración:

```
jac =

   0.388389290881808  -0.239203785854514
  -0.290267727933724  -0.956815143382528

f0 =

  5.82679460237046e-010
  2.27137775254960e-009

dx =

  -3.21831474283236e-011
  2.38365738398390e-009

x =

   4.854816136036560
  -0.478457569325022

error =   2.38365738398390e-009
```

Ejercicio 3: inciso (a)

1. Se escribió la función en Octave como una función vectorial:
   `f=@(x,y,z)[x.^2-2*exp(y)-5*z; x.*y+z.^2+sin(y.^2)-1; 2*x+y.^2-z-3]`
2. Se modificó el código de los programas `newtonRaphson2.m` y `jacobian.m` para que admitiesen funciones vectoriales de 3 dimensiones. El cambio se hizo en las líneas 7 y 12 de `jacobian.m`, donde `f0` y `f1` valen:
   `func(x(1),x(2),x(3))`
   Añadiendo el parámetro x(3) para tener tres incógnitas. Esto dio lugar al programa `newtonRaphson3x3.m`
3. Se hizo el comando `[raiz,niter]=newtonRaphson3x3(f,[1,1,1],10.^(-10))`, donde el sistema de ecuaciones dado se sometió al método de Newton para hallar las raíces del mismo, usando como vector inicial a (1,1,1) transpuesto.
4. El programa se ejecutó en 8 iteraciones, como se ilustra a continuación:
   a. Iteración 1:

```
jac =

   2.000099999985849  -5.436835494165848  -5.000000000006111
   0.999999999997669   2.080490323730366   2.000099999994731
   1.999999999999780   2.000099999999172  -0.999999999999890

f0 =

  -9.43656365691809
   1.84147098480790
  -1.00000000000000

dx =

   1.030964098861995
  -0.821655612538239
  -0.581465192913353

x =

   2.030964098861995
   0.178344387461761
   0.418534807086647

error =  1.03096409886200
```

b. Iteración 2:

```
jac =

   4.062028197733270  -2.390593277588060  -5.000000000001670
   0.178344387461449   2.387572210712507   0.837169614172906
   2.000000000004221   0.356788774924510  -1.000000000002110

f0 =

  -0.358332614481580
  -0.430816209248950
   0.675200111176454

dx =

  -1.152529057834209
   0.744837514433040
  -1.364108340201569

x =

   0.878435041027786
   0.923181901894801
  -0.945573533114922

error =  1.36410834020157
```

c. Iteración 3:

```
jac =

   1.756970082054110  -5.034826580976315  -4.999999999997229
   0.923181901892889   2.093796783837476  -1.891047066235885
   2.000000000004221   1.846463803794052  -1.000000000002110

f0 =

   0.464940943037913
   1.457837861097725
   0.554708439156595

dx =

   0.309602215704662
  -0.340689141034518
   0.544842703300152

x =

   1.188037256732448
   0.582492760860282
  -0.400730829814770

error =  0.544842703300152
```

d. Iteración 4:

```
jac =

   2.376174513467610  -3.581171357884827  -5.000000000001670
   0.582492760861086   2.286677032625750  -0.801361659628075
   2.000000000004221   1.165085521721565  -1.000000000002110

f0 =

  -0.165905629846542
   0.185433322809900
   0.116103159734301

dx =

  -0.00707226307358878
  -0.07361892768221953
   0.01618628681983812

x =

   1.180964993658859
   0.508873833178063
  -0.384544542994932

error =  0.0736189276822195
```

e. Iteración 5:

```
jac =

   2.362029987317271  -3.327000056949636  -4.999999999997229
   0.508873833178214   2.164863049538557  -0.768989085990768
   1.999999999999780   1.017847666355820  -0.999999999997669

f0 =

  -0.00943267850002316
   0.00490488174265069
   0.00542710840598559

dx =

  -0.00230962129361424
  -0.00224891471396646
  -0.00148118677468987

x =

   1.178655372365245
   0.506624918464096
  -0.386025729769622

error =  0.00230962129361424
```

f. Iteración 6:

```
jac =

   2.357410744731059  -3.319526324609967  -4.999999999999449
   0.506624918465981   2.158795845437211  -0.771951459538345
   1.999999999999780   1.013349836926913  -1.000000000002110

f0 =

  -3.21541098080580e-006
  1.19468668624556e-005
  5.28250886455695e-006

dx =

  1.31159264735086e-006
  -4.72819180979423e-006
  3.11438175983766e-006

x =

   1.178656683957892
   0.506620190272287
  -0.386022615387862

error =   4.72819180979423e-006
```

g. Iteración 7:

```
jac =

   2.357413367917172  -3.319510629293809  -4.999999999999449
   0.506620190272589   2.158789243318360  -0.771945230775284
   1.999999999995339   1.013340380540129  -1.000000000002110

f0 =

  -9.51296819096115e-010
  1.06464170812615e-010
  4.95175456194374e-010

dx =

  -3.65804453189836e-010
  -7.53000086470254e-011
  -3.12737989599984e-010

x =

   1.178656683592088
   0.506620190196987
  -0.386022615700600

error =   3.65804453189836e-010
```

h. Iteración 8:

```
jac =

   2.357413367184424  -3.319510629049560  -4.999999999999449
   0.506620190197093   2.158789242825421  -0.771945231401450
   1.999999999999780   1.013340380393579  -1.000000000002110

f0 =

  2.42028619368284e-014
  3.77475828372553e-014
  7.10542735760100e-015

dx =

  1.32665912662817e-014
  -1.34405546614470e-014
  2.00187531167878e-014

x =

   1.178656683592101
   0.506620190196973
  -0.386022615700580

error =   2.00187531167878e-014
```

5. Tras tener estas 8 iteraciones, obtenemos una raíz con 10 cifras decimales exactas:

```
raiz =

   1.178656683592101
   0.506620190196973
  -0.386022615700580
```

   Y un error muy pequeño de 2×10^{-14}.

<u>Ejercicio 3: inciso (b)</u>

1. Se modificó el código del programa newtonRaphson3x3.m para que mostrara el error relativo en cada iteración. Este fue calculado multiplicando el error absoluto por 100 y dividiéndolo entre la norma de la raíz obtenida en dicha iteración, de la forma:
   `erel = (error*100)/norm(x,inf)`
   Y luego el valor final de `erel` se devolvió al argumento de salida `erRel`.
2. Se ejecutó el comando `[raiz,niter,erRel]=newtonRaphson3x3(f,[-1,-1,-1],0.01)`, donde el sistema de ecuaciones dado se sometió al método de Newton para hallar las raíces del mismo, usando como vector inicial a (-1,-1,-1) transpuesto.
3. El programa se ejecutó en 6 iteraciones como se muestra a continuación:

a. Iteración 1:

```
jac =

  -1.999899999995947  -0.735795671511497  -4.999999999997229
  -1.000000000002110  -2.080718851669161  -1.999900000004828
   1.999999999999780  -1.999900000000387  -0.999999999997669

f0 =

   5.26424111765712
   1.84147098480790
  -3.00000000000000

dx =

   1.449257335625956
  -0.310167284489561
   0.518818223503594

x =

   0.449257335625956
  -1.310167284489561
  -0.481181776496406

error =  1.44925733562596
erel =  110.616205486354
```

b. Iteración 2:

```
jac =

   0.898614671251963  -0.539576825291377  -5.000000000001670
  -1.310167284489516   0.829445420443831  -0.962263552992670
   1.999999999995339  -2.620234568979640  -1.000000000002110

f0 =

   2.0681911891969476
  -0.3676679393625730
   0.0962347610950673

dx =

  -1.095060634558591
  -0.919754636204389
   0.316086384744355

x =

  -0.645803298932636
  -2.229921920693950
  -0.165095391752051

error =  1.09506063455859
erel =  49.1075774625243
```

c. Iteración 3:

```
jac =

  -1.291506597866920  -0.215084406460786  -5.000000000001670
  -2.229921920693378  -1.792057263078073  -0.330090783504655
   1.999999999999780  -4.459743841405128  -0.999999999997669

f0 =

   1.027465207254976
  -0.499000693546390
   0.846040566278174

dx =

  -0.27005713382418900
   0.00694567344317222
   0.27495037426789826

x =

  -0.915860432756825
  -2.222976247250778
   0.109854982515848

error =  0.274950374267898
erel =  12.3685700469331
```

d. Iteración 4:

```
jac =

  -1.831620865513361  -0.216583512633317  -5.000000000000560
  -2.222976247252539  -1.925138072147448   0.219809965029683
   1.999999999999780  -4.445852494514568  -1.000000000002110

f0 =

  7.29527360725343e-002
  7.41634592846896e-002
  4.75478116537253e-005

dx =

   0.02505306139915991
   0.01016247624326228
   0.00497280025340489

x =

  -0.890807371357665
  -2.212813771007515
   0.114827782769253

error =  0.0250530613991599
erel =  1.13218119515557
```

e. Iteración 5:

```
jac =

  -1.781514742713597  -0.218795759348023  -5.000000000000560
  -2.212813771007571  -1.700232540280400   0.229755565539502
   1.999999999995339  -4.425527542029073  -1.000000000002110

f0 =

  6.14039276765954e-004
  1.28505120644573e-003
  1.02259675918592e-004

dx =

  4.11054393851829e-004
  2.16355684827412e-004
  -3.31195784578675e-005

x =

  -0.890396316963813
  -2.212597415322688
   0.114794663190795

error =   4.11054393851829e-004
erel =  0.0185779116890038
```

f. Iteración 6:

```
jac =

  -1.780692633926995  -0.218843102176569  -5.000000000000560
  -2.212597415323092  -1.695576671968402   0.229689326380900
   1.999999999995339  -4.425094830660115  -1.000000000002110

f0 =

  1.25106116466434e-007
  3.40303768719963e-007
  2.51742191537119e-008

dx =

  1.07730672756005e-007
  5.79690651346996e-008
  -1.58830458005057e-008

x =

  -0.890396209233140
  -2.212597357353623
   0.114794647307749

error =   1.07730672756005e-007
erel =   4.86896869861836e-006
```

4. Con estas 6 iteraciones se obtuvo una raíz:

```
raiz =

  -0.890396209233140
  -2.212597357353623
   0.114794647307749
```

Y el error relativo fue de:

```
erRel =    4.86896869861836e-006
```