

Práctica Dirigida 6

Análisis y Modelamiento Numérico I



Alumno:

■ Chowdhury Gomez, Junal Johir

20200092K

13 de noviembre de 2024

Ejercicio 3

Enunciado

Use el método del punto fijo para encontrar las soluciones del sistema:

$$\begin{cases} x^2 + xy - 10 = 0 \\ y + 3xy^2 - 57 = 0 \end{cases}$$

Inicie con los puntos $x = 2,5$ y $y = 3,5$.

Solución

Primero, reformulamos el sistema de ecuaciones para aplicar el método del punto fijo. Despejamos x y y en términos de las otras variables:

1. De la primera ecuación, despejamos x en términos de y :

$$x = \sqrt{10 - xy}$$

2. De la segunda ecuación, despejamos y en términos de x :

$$y = \sqrt{\frac{57 - y}{3x}}$$

Reformulamos estas ecuaciones para obtener funciones de iteración adecuadas:

$$\begin{cases} x_{n+1} = \frac{10}{x_n + y_n} \\ y_{n+1} = \sqrt{\frac{57 - y_n}{3x_n}} \end{cases}$$

Código en python

```
1 import numpy as np
2
3 def punto_fijo(x0, y0, max_iter=1000, tol=1e-6):
4     x, y = x0, y0
5     for i in range(max_iter):
6         x_n1 = 10 / (x + y)
7         y_n1 = np.sqrt((57 - y) / (3 * x))
8         if np.abs(x_n1 - x) < tol and np.abs(y_n1 - y) < tol:
9             break
10        x, y = x_n1, y_n1
11        print(f"Iteration {i + 1}: x = {x:.6f}, y = {y:.6f}")
12    return x, y
13
14 x0 = 2.5
15 y0 = 3.5
16 x, y = punto_fijo(x0, y0)
17 print(f"Solucion encontrada: x = {x}, y = {y}")
```

Salida del código

```
Iteration 1: x = 1.666667, y = 2.670830
Iteration 2: x = 2.305477, y = 3.296336
Iteration 3: x = 1.785136, y = 2.786512
Iteration 4: x = 2.187395, y = 3.181686
Iteration 5: x = 1.862516, y = 2.863788
Iteration 6: x = 2.115818, y = 3.112672
Iteration 7: x = 1.912598, y = 2.913691
... ..
Iteration 57: x = 1.999999, y = 2.999999
Iteration 58: x = 2.000001, y = 3.000001
Iteration 59: x = 2.000000, y = 3.000000
Solución encontrada: x = 1.9999995044972836, y = 2.9999995139522455
```

Conclusión

- Si existe una solución cuando las iteraciones comienzan con los puntos $x = 2.5$ y $y = 3.5$.
- La solución encontrada después de 59 iteraciones es aproximadamente $x = 1,99999950449$ y $y = 2,9999995139$.

Ejercicio 18

Enunciado

Use el método de Newton para calcular la única raíz de la ecuación:

$$x + e^{-Bx^2} \cos(x) = 0$$

con $B = 1, 5, 10, 25, 50$ y puntos iniciales $x_0 = 0, 1, 2, 10$.

Solución:

Primero, definimos la función $f(x)$ y su derivada $f'(x)$:

$$f(x) = x + e^{-Bx^2} \cos(x)$$

La derivada de $f(x)$ con respecto a x es:

$$f'(x) = 1 - e^{-Bx^2} \sin(x) - 2Bxe^{-Bx^2} \cos(x)$$

El método de Newton usa la fórmula iterativa:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Vamos a implementar este método en Python para resolver la ecuación para diferentes valores de B y puntos iniciales x_0 .

Implementación en Python

```
1 import numpy as np
2
3 def f(x, B):
4     return x + np.exp(-B * x**2) * np.cos(x)
5
6 def df(x, B):
7     return 1 - np.exp(-B * x**2) * np.sin(x) - 2 * B * x *
8     np.exp(-B * x**2) * np.cos(x)
9
10 def newton_method(x0, B, tol=1e-6, max_iter=1000):
11     x = x0
12     for _ in range(max_iter):
13         fx = f(x, B)
14         dfx = df(x, B)
15         if dfx == 0:
16             raise ValueError("La derivada es cero. No se puede continuar.")
17         #print(f"x = {x}, f(x) = {fx}, df(x) = {dfx} ")
18         x_new = x - fx / dfx
19
20         if abs(x_new - x) < tol:
21             return x_new
22         x = x_new
23     raise ValueError("El metodo de Newton no convergio.")
24
25 # Valores de B y puntos iniciales
26 Bs = [1, 5, 10, 25, 50]
27 x0s = [0, 1, 2, 10]
28
29 # Calcular y mostrar resultados
30 for B in Bs:
31     for x0 in x0s:
32         try:
33             root = newton_method(x0, B)
34             print(f"Para B = {B}, x0 = {x0}, la raiz es: {root:.6f}")
35         except ValueError as e:
36             print(f"Para B = {B}, x0 = {x0}, hubo un error: {e}")
```

Salida del código

Al ejecutar el código, obtenemos los siguientes resultados para las iteraciones:

```
Para B = 1, x0 = 0, la raiz es: -0.588401777
Para B = 1, x0 = 1, la raiz es: -0.588401777
Para B = 1, x0 = 2, la raiz es: -0.588401777
Para B = 1, x0 = 10, la raiz es: -0.588401777
Para B = 5, x0 = 0, la raiz es: -0.404911548
Para B = 5, x0 = 1, la raiz es: -0.404911548
Para B = 5, x0 = 2, la raiz es: -0.404911548
Para B = 5, x0 = 10, la raiz es: -0.404911548
Para B = 10, x0 = 0, hubo un error: El metodo de Newton no convergio.
Para B = 10, x0 = 1, hubo un error: El metodo de Newton no convergio.
Para B = 10, x0 = 2, hubo un error: El metodo de Newton no convergio.
Para B = 10, x0 = 10, hubo un error: El metodo de Newton no convergio.
Para B = 25, x0 = 0, hubo un error: El metodo de Newton no convergio.
Para B = 25, x0 = 1, hubo un error: El metodo de Newton no convergio.
Para B = 25, x0 = 2, hubo un error: El metodo de Newton no convergio.
Para B = 25, x0 = 10, hubo un error: El metodo de Newton no convergio.
Para B = 50, x0 = 0, hubo un error: El metodo de Newton no convergio.
Para B = 50, x0 = 1, hubo un error: El metodo de Newton no convergio.
Para B = 50, x0 = 2, hubo un error: El metodo de Newton no convergio.
Para B = 50, x0 = 10, hubo un error: El metodo de Newton no convergio.
```

Observación:

- El método Newton, solo converge cuando B toma los valores de 1 y 5, para todos los valores de x_0 .
- Cuando B toma los valores: 10, 25 y 50, y todos los valores de x_0 , el metodo de Newton no converge

Ejercicio 10

Enunciado

Use el método de Newton para aproximar la raíz de la función:

$$f(x) = \cos(x) + \sin^2(50x)$$

e intente aproximar la raíz $\alpha = \frac{\pi}{2}$.

Solución

Primero, definimos la función $f(x)$ y su derivada $f'(x)$:

$$f(x) = \cos(x) + \sin^2(50x)$$

La derivada de $f(x)$ con respecto a x es:

$$f'(x) = -\sin(x) + 100 \sin(50x) \cos(50x)$$

Usando la identidad trigonométrica $\sin(2\theta) = 2 \sin(\theta) \cos(\theta)$, podemos simplificar:

$$f'(x) = -\sin(x) + 50 \sin(100x)$$

El método de Newton usa la fórmula iterativa:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Vamos a implementar este método en Python para aproximar la raíz $\alpha = \frac{\pi}{2}$.

Código en python

```
1 import numpy as np
2 def f(x):
3     return np.cos(x) + np.sin(50 * x)**2
4 def df(x):
5     return -np.sin(x) + 50 * np.sin(100 * x)
6 def newton_method(x0, tol=1e-8, max_iter=1000):
7     x = x0
8     for i in range(max_iter):
9         fx = f(x)
10        dfx = df(x)
11        print(f"x = {x}, f(x) = {fx}, df(x) = {dfx} ")
12        if dfx == 0:
13            raise ValueError("La derivada es cero. No se puede continuar.")
14        x_new = x - fx / dfx
15        if abs(x_new - x) < tol:
16            return x_new
17        x = x_new
18    raise ValueError("El metodo de Newton no convergio.")
19 # Aproximacion inicial
20 x0 = np.pi / 2
21 # Solucion usando el metodo de Newton
22 root = newton_method(x0)
23 print(f"La raiz aproximada es: {root:.6f}")
24 print(f"Error relativo: {abs(root - np.pi / 2):.6e}")
```

Al ejecutar el código, obtenemos la siguiente aproximación para la raíz:

Salida del código

x = 1.5707963267948966, f(x) = 6.123233995736791e-17, df(x) = -0.9999999999999509

La raiz aproximada es: 1.570796

Error relativo: 0.000000e+00

Conclusión

- La raíz aproximada es: 1.570796.
- El método de Newton converge rápidamente a $\frac{\pi}{2}$.

Ejercicio 16

Enunciado

Resolver el sistema de ecuaciones no lineales:

■ a)

$$\begin{cases} x^2 - 2x - y + 0,5 = 0 \\ x^2 + 4y^2 - 4 = 0 \end{cases}$$

■ b)

utilizando el método de Newton y el método de punto fijo a los problemas.

Solución a)

El método de Newton para sistemas de ecuaciones no lineales se expresa como:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)\mathbf{F}(\mathbf{x}_n)$$

donde:

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} x^2 - 2x - y + 0,5 \\ x^2 + 4y^2 - 4 \end{pmatrix}$$

y la matriz Jacobiana es:

$$J(\mathbf{x}) = \begin{pmatrix} 2x - 2 & -1 \\ 2x & 8y \end{pmatrix}$$

Gráfica de las funciones

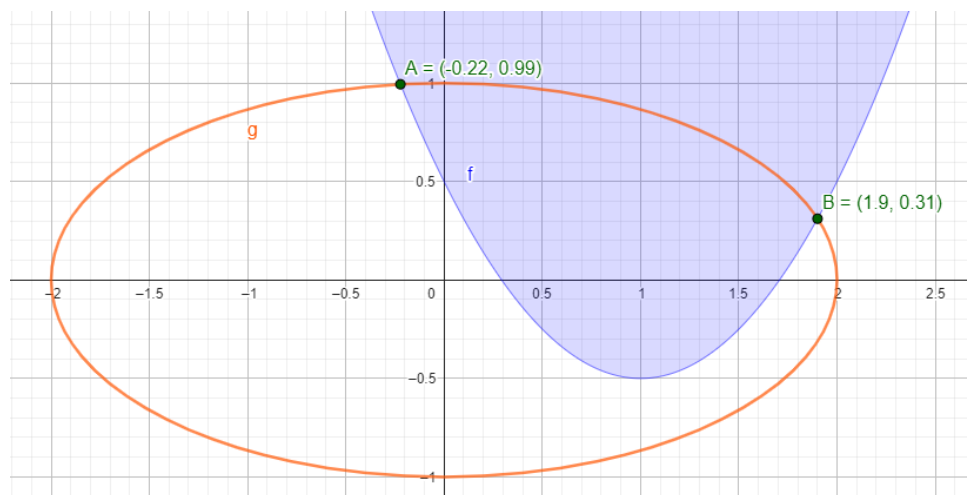


Figura 1: Gráficas de las funciones de a) en Geogebra.

Código en python

```
1 import numpy as np
2 def F(x):
3     return np.array([
4         x[0]**2 - 2*x[0] - x[1] + 0.5,
5         x[0]**2 + 4*x[1]**2 - 4
6     ])
7 def J(x):
8     return np.array([
9         [2*x[0] - 2, -1],
10        [2*x[0], 8*x[1]]
11    ])
12 def metodo_newton_SE(F, J, x0, tol, max_iter=1000):
```

```

13     x_n = np.array(x0, dtype=float)
14     for _ in range(max_iter):
15         Fx_n = F(x_n)
16         Jx_n = J(x_n)
17         print(f"x_n = {x_n}, F(x_n) = {Fx_n}, J(x_n) = {Jx_n}")
18         if np.linalg.det(Jx_n) == 0:
19             print("La matriz Jacobiana es singular")
20             return None
21         delta = np.linalg.solve(Jx_n, -Fx_n)
22         x_n1 = x_n + delta
23         if np.linalg.norm(delta, ord=np.inf) < tol:
24             return x_n1
25         x_n = x_n1
26     print("El metodo no convergio")
27     return None
28 # Parametros
29 x0 = [2, 0.5] # Estimacion inicial
30 tolerancia = 1e-6
31 # Encontrar la solucion
32 solucion = metodo_newton_SE(F, J, x0, tolerancia)
33 print(f"La solucion encontrada es: {solucion}")

```

Salida del código

```

x_n = [2.  0.5], F(x_n) = [0.  1.],
J(x_n) = [[ 2. -1.]
 [ 4.  4.]]
x_n = [1.91666667 0.33333333], F(x_n) = [0.00694444 0.11805556],
J(x_n) = [[ 1.83333333 -1.]
 [ 3.83333333  2.66666667]]
x_n = [1.90100849 0.31157113], F(x_n) = [0.00024518 0.00213955],
J(x_n) = [[ 1.80201699 -1.]
 [ 3.80201699  2.492569  ]]
x_n = [1.90067683 0.31121865], F(x_n) = [1.09998290e-07 6.06962269e-07],
J(x_n) = [[ 1.80135367 -1.]
 [ 3.80135367  2.48974918]]
La solucion encontrada es: [1.90067673 0.31121857]

```

Conclusión

- El método de Newton aplicado a este sistema de ecuaciones no lineales converge a la solución (1,90067673,0,31121857) en 4 iteraciones.

Solución b)

Gráfica de las funciones

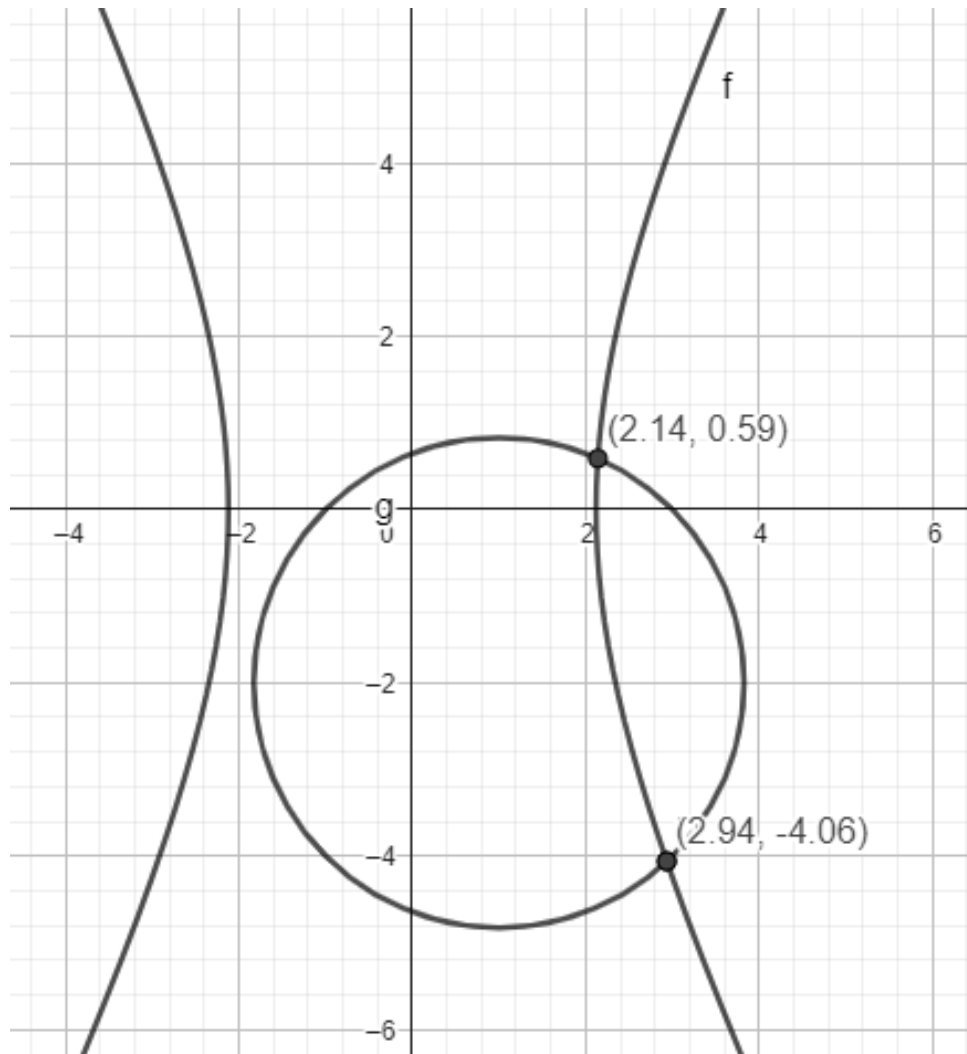


Figura 2: Gráficas de las funciones de b) en Geogebra.

Código en python

```
1 def g1(x, y):
2     return (8*x - 4*(x**2) + (y**2) + 1)/8
3
4 def g2(x, y):
5     return (2*x - (x**2) + 4*y - (y**2) + 3)/4
6
7 def punto_fijo(g_funcs, x0, tol, max_iter=1000):
8     x = np.array(x0, dtype=float)
9     for _ in range(max_iter):
10        x_nuevo = np.array([g(x[0], x[1]) for g in g_funcs])
11        if np.linalg.norm(x_nuevo - x, ord=np.inf) < tol:
12            return x_nuevo
13        x = x_nuevo
14    print("El metodo no convergio.")
15    return None
16
17 # Parametros
18 x0 = [2, 0.5] # Estimacion inicial
19 tolerancia = 1e-6
20
```

```
21 # Definir las funciones g
22 g_funcs = [g1, g2]
23
24 # Encontrar la solucion
25 solucion = punto_fijo(g_funcs, x0, tolerancia)
26 print(f"La solucion encontrada es: {solucion}")
```

Salida del código

```
Iteracion 1: x: [2.  0.5],x_nuevo: [0.15625 1.1875 ]
Iteracion 2: x: [0.15625 1.1875 ],x_nuevo: [0.4453125  1.65698242]
Iteracion 3: x: [0.4453125  1.65698242],x_nuevo: [0.81435973 1.89366518]
Iteracion 4: x: [0.81435973 1.89366518],x_nuevo: [1.05601482 1.98855765]
Iteracion 5: x: [1.05601482 1.98855765],x_nuevo: [1.11772636 1.99918285]
Iteracion 6: x: [1.11772636 1.99918285],x_nuevo: [1.11766176 1.99653496]
Iteracion 7: x: [1.11766176 1.99653496],x_nuevo: [1.11634684 1.99653593]
Iteracion 8: x: [1.11634684 1.99653593],x_nuevo: [1.11650117 1.99661285]
Iteracion 9: x: [1.11650117 1.99661285],x_nuevo: [1.1165216 1.996604 ]
Iteracion 10: x: [1.1165216 1.996604 ],x_nuevo: [1.1165148 1.9966028]
Iteracion 11: x: [1.1165148 1.9966028],x_nuevo: [1.11651499 1.99660319]
La solucion encontrada es: [1.11651499 1.99660319]
```

Conclusión

- El método del punto fijo partiendo desde el punto (2,0,5) aplicado a este sistema de ecuaciones no lineales converge a la solución (1,11651499,1,99660319) en 11 iteraciones.

Ejercicio 18

Enunciado

Use el método de la bisección para encontrar una raíz de la ecuación:

$$f(x) = x^8 - 36x^7 + 546x^6 - 4536x^5 + 22449x^4 - 67284x^3 + 118124x^2 - 109584x + 40320 = 0$$

en el intervalo $[5,5,6,5]$.

Solución

Primero, definimos la función $f(x)$:

$$f(x) = x^8 - 36x^7 + 546x^6 - 4536x^5 + 22449x^4 - 67284x^3 + 118124x^2 - 109584x + 40320$$

Para aplicar el método de la bisección, verificamos que $f(x)$ cambie de signo en el intervalo $[5,5,6,5]$, lo que asegura la existencia de al menos una raíz en este intervalo.

El método de la bisección consiste en dividir el intervalo a la mitad repetidamente y seleccionar el subintervalo donde la función cambia de signo.

Código en python

```
1 def f(x):
2     return x**8 - 36*x**7 + 546*x**6 - 4536*x**5 + 22449*x**4
3     - 67284*x**3 + 118124*x**2 - 109584*x + 40320
4
5 def biseccion(a, b, tol=1e-6, max_iter=100):
6     if f(a) * f(b) >= 0:
7         raise ValueError("La funcion no cambia de signo en el intervalo dado")
8
9     for i in range(max_iter):
10        c = (a + b) / 2.0
11        print(f"Iteracion {i+1}: a = {a}, f(a) = {f(a)}, b = {b},
12              f(b) = {f(b)}, c = {c}, f(c) = {f(c)}")
13        if abs(f(c)) < tol:
14            return c
15        if f(c) * f(a) < 0:
16            b = c
17        else:
18            a = c
19
20    raise ValueError("El metodo de la biseccion no convergio")
21
22 # Intervalo dado
23 a = 5.5
24 b = 6.5
25
26 # Encontrar la raiz usando el metodo de la biseccion
27 try:
28     raiz = biseccion(a, b)
29     print(f"Una raiz aproximada en el intervalo [{a}, {b}] es: {raiz:.6f}")
30 except ValueError as e:
31     print(e)
```

Salida del código

Iteracion 1: a = 5.5, f(a) = -55.37109375, b = 6.5,
f(b) = 121.81640625, c = 6.0, f(c) = 0.0

Una raiz aproximada en el intervalo $[5.5, 6.5]$ es: 6.000000

Conclusión

- Una raíz aproximada en el intervalo $[5.5, 6.5]$ es: 6.0.
- La raíz se encuentra con la primera iteración.

Ejercicio 25

Enunciado

Resolver el siguiente sistema de ecuaciones no lineales:

$$\begin{cases} -x_1(x_1 + 1) + 2x_2 = 18 \\ (x_1 - 1)^2 + (x_2 - 6)^2 = 25 \end{cases}$$

Solución

Parte a: Aproximación Gráfica de las Soluciones

Graficamos las curvas de las dos ecuaciones para encontrar las intersecciones.

Código en python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Definir los rangos para x1 y x2
5 x1 = np.linspace(-5, 5, 400)
6 x2 = np.linspace(0, 15, 400)
7
8 # Crear una malla para evaluar las funciones
9 X1, X2 = np.meshgrid(x1, x2)
10
11 # Definir las funciones
12 F1 = -X1*(X1 + 1) + 2*X2 - 18
13 F2 = (X1 - 1)**2 + (X2 - 6)**2 - 25
14
15 # Graficar las curvas
16 plt.figure(figsize=(10, 8))
17 plt.contour(X1, X2, F1, levels=[0], colors='r', label='F1: -x1(x1+1) + 2x2 = 18')
18 plt.contour(X1, X2, F2, levels=[0], colors='b', label='F2: (x1-1)^2 + (x2-6)^2 = 25')
19 plt.xlabel('x1')
20 plt.ylabel('x2')
21 plt.title('Intersección de las curvas de las ecuaciones')
22 plt.axhline(0, color='black', linewidth=0.5)
23 plt.axvline(0, color='black', linewidth=0.5)
24 plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
25 plt.legend(['F1: -x1(x1+1) + 2x2 = 18', 'F2: (x1-1)^2 + (x2-6)^2 = 25'])
26 plt.show()
```

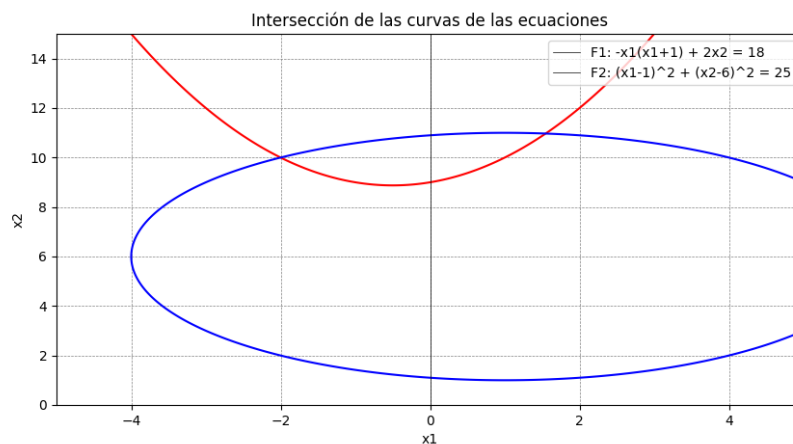


Figura 3: Gráficas de las funciones.

Parte b: Método del punto fijo

Reformulamos las ecuaciones para obtener las funciones de iteración del punto fijo:

$$g_1(x, y) = \frac{18 + x(x + 1)}{2}$$

$$g_2^+(x) = 6 + \sqrt{25 - (x - 1)^2}$$

$$g_2^-(x) = 6 - \sqrt{25 - (x - 1)^2}$$

Código en python

```
1 import numpy as np
2
3 # Definir las funciones de iteracion
4 def g1(x1, x2):
5     return (18 + x1*(x1 + 1))/2
6
7 def g2_plus(x1):
8     return 6 + np.sqrt(25 - (x1 - 1)**2)
9
10 def g2_minus(x1):
11     return 6 - np.sqrt(25 - (x1 - 1)**2)
12
13 def punto_fijo(g1, g2, x0, tol, max_iter=1000):
14     x = np.array(x0, dtype=float)
15     for _ in range(max_iter):
16         x_nuevo = np.array([g1(x[0], x[1]), g2(x[0])])
17         if np.linalg.norm(x_nuevo - x, ord=np.inf) < tol:
18             return x_nuevo
19         x = x_nuevo
20     print("El metodo no convergio.")
21     return None
22
23 # Parametros
24 x0_1 = [2, 11] # Primera estimacion inicial
25 x0_2 = [-2, 10] # Segunda estimacion inicial
26 tolerancia = 1e-5
27
28 # Encontrar la solucion usando g2_plus
29 solucion_1_plus = punto_fijo(g1, g2_plus, x0_1, tolerancia)
30 print(f"Solucion encontrada usando g2_plus: {solucion_1_plus}")
31
32 solucion_2_plus = punto_fijo(g1, g2_plus, x0_2, tolerancia)
33 print(f"Solucion encontrada usando g2_plus: {solucion_2_plus}")
34
35 # Encontrar la solucion usando g2_minus
36 solucion_1_minus = punto_fijo(g1, g2_minus, x0_1, tolerancia)
37 print(f"Solucion encontrada usando g2_minus: {solucion_1_minus}")
38
39 solucion_2_minus = punto_fijo(g1, g2_minus, x0_2, tolerancia)
40 print(f"Solucion encontrada usando g2_minus: {solucion_2_minus}")
```

Salida del código

El metodo no convergio.
Solucion encontrada usando g2_plus: None
El metodo no convergio.
Solucion encontrada usando g2_plus: None
El metodo no convergio.
Solucion encontrada usando g2_minus: None
El metodo no convergio.
Solucion encontrada usando g2_minus: None

Conclusión

- Para las aproximaciones dadas y utilizando el método del punto fijo, las iteraciones no convergen, esto debido a un mal modelamiento o aproximaciones muy lejanas a las soluciones.