Universidad Autónoma de Querétaro

Facultad de Ingeniería

Examen Parcial 1

Análisis numérico

Autor:
David Gómez Torres

27 Septiembre del 2021

ÍNDICE GENERAL

CAPÍTULO 1

METODOLOGÍA

1.1. Tiro parabólico de un proyectil

Problema 1

Un proyectil es lanzado de O con una velocidad v a un ángulo θ con respecto a la horizontal. Si el proyectil golpea el objetivo con un ángulo de 45° como se muestra en la Figura (1.1), determine v, θ y el tiempo del vuelo.

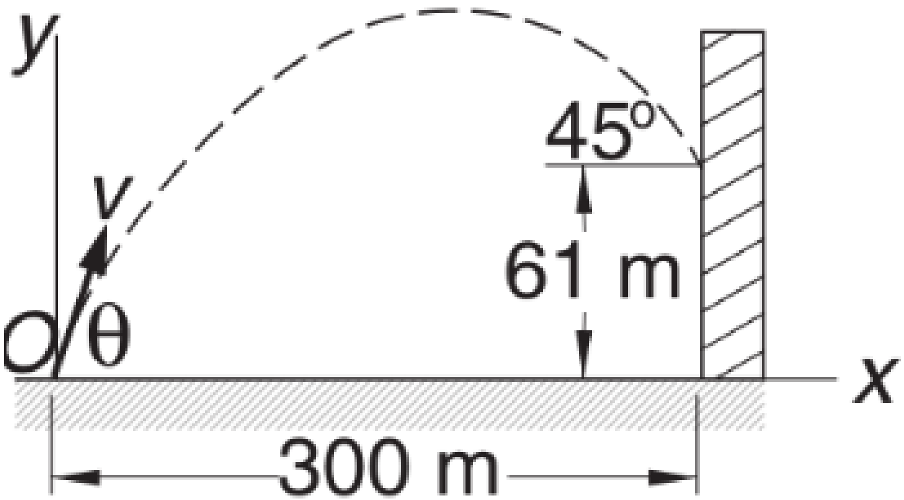

Figura 1.1: Representación gráfica del problema

1.1.1. Planteamiento del problema

Para plantear una solución analítica a este problema, debemos recurrir a las ecuaciones de cinemática. A continuación vamos a hacer la deducción de las mismas.
Por definición, sabemos que la aceleración es el cambio de la velocidad en el tiempo

$$a = \frac{dv}{dt}$$

Si arreglamos la expresión, podemos integrar para obtener una ecuación que nos dé la velocidad v

$$\int_{v_i}^{v} dv = \int_0^t a\ dt$$

$$v = v_i + at \tag{1.1}$$

A su vez, por definición sabemos que el cambio del desplazamiento en el tiempo es igual a la velocidad, es decir

$$v = \frac{dx}{dt}$$

Podemos acomodar los diferenciales para integrar y obtener la ecuación para la distancia x en términos de la velocidad y el tiempo

$$\int_{x_i}^{x} dx = \int_0^t v\ dt$$

$$x = x_i + \int_0^t (v_i + at)\ dt$$

$$x = x_i + v_i t + \frac{at^2}{2} \tag{1.2}$$

Recordemos que los problemas de este tipo son vectoriales, es decir, tienen una componente en x y y. Tomando en cuenta lo anterior, la ecuación (1.2) puede utilizarse tanto para x como para y.
Para x, a un desplazamiento $x_i = 0$ y tiempo $t = 0$:

$$x = v_i t$$

$$x = (v\cos\theta)t \tag{1.3}$$

Mientras que para y, obtenemos la siguiente expresión cuando consideramos $y_i = 0$ y tomamos un cierto tiempo t:

$$y = v_i t - \frac{1}{2}at^2$$

$$y = (v\sin\theta)t - \frac{1}{2}at^2 \tag{1.4}$$

En resumen:

$$\text{Ecuaciones paramétricas}\begin{cases} x = (v\cos\theta)t \\ y = (v\sin\theta)t - \frac{1}{2}at^2 \end{cases}$$

donde a es la gravedad $g = 9.81\ m/s^2$.

1.1.2. Pseudocódigo

Algorithm 1 Método de Newton-Raphson

Entrada: array x; funcion f,df; real es;
Salida: array x; real error;
doble funcion f(x):
return f(x)
doble funcion df(x):
return df(x)
while error > es **do**
  for k←1 **do**
    X ← Xold
    X = Xold - f(x) / df(x)
    Jacobiano = inv(df(x))
  end for
  error=$\left|\dfrac{x - x_0}{100}\right|$
end while
return print(x, error)

1.1.3. Código fuente del problema

1. Método de Newton-Raphson con el uso del Jacobiano

```
import numpy as np
import math

n = int(input("Ingrese el numero de ecuaciones: "))

x = np.array([-1,1])
theta = 45 #angulo de lanzamiento
g = 9.81 #aceleracion de la gravedad

def f(x): #ingresar las funciones
    #x[0] = v  /   x[1] = t
    f1 = (x[0]*math.cos(theta))*x[1]
    f2 = (x[0]*math.sin(theta))*x[1] - 1/2*g**2*x[1]

    return np.array([f1,f2])

def df(x): #calcular su derivada, manualmente
    return np.array([[x[0]*math.sin(theta)],
                     [x[0]*math.cos(theta)-g*x[1]]])

def NewtonRaphsonJacobiano(f,df,x,es):
    error = 100
    k = 0
    while(error > es):
        xold = x
        jacobianoInv=np.linalg.inv(df(x)) #jacobiano
        x = x - np.dot(jacobianoInv,f(x))
        error = np.linalg.norm(x - xold)
        k +=  1 #contador iteraciones

        print(k,"| Soluciones:",x,"| Error relativo: ", error)

NewtonRaphsonJacobiano(f,df,x,0.05)
```

1.2. Circuito eléctrico

Problema 2

Usando las leyes de Kirchoff determine las corrientes i_1 a i_4 en la red eléctrica mostrada en la Figura (1.2).

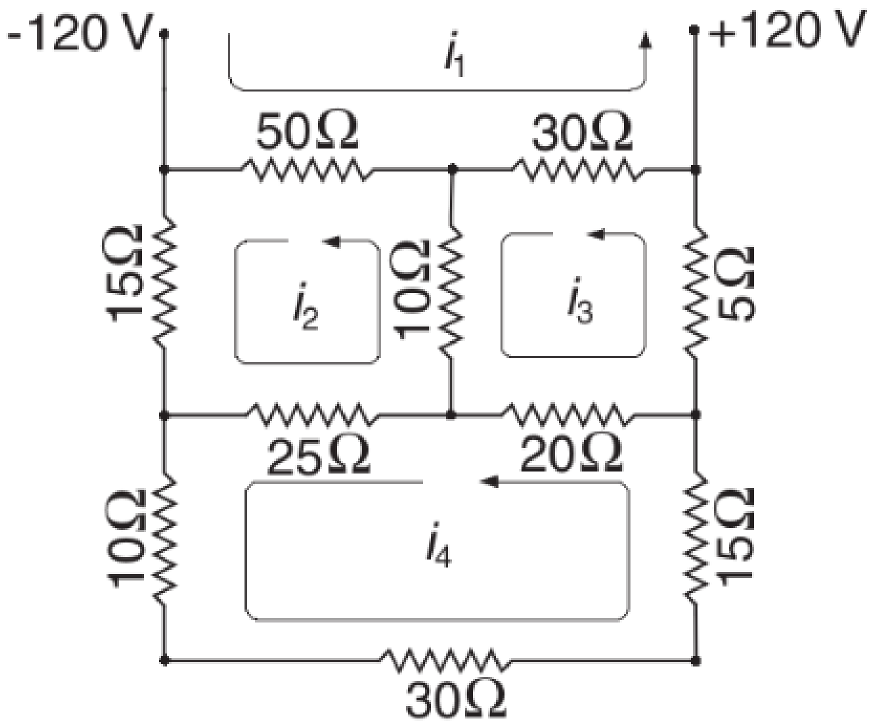

Figura 1.2: Circuito eléctrico resolver

1.2.1. Planteamiento del sistema de ecuaciones

De las leyes de Kirchoff para las tensiones tenemos la siguiente expresión:

> En un circuito cerrado, la **suma** de todas las caídas de tensión es igual a la **tensión total** suministrada. De forma equivalente, la suma algebraica de las diferencias de potencial eléctrico en un circuito es igual a cero.
>
> $$\sum_{k=1}^{n} V_k = V_1 + V_2 + V_3 \cdots + V_n = 0 \tag{1.5}$$

Planteamos las ecuaciones por cada malla basado en la figura (1.2):

$$\begin{aligned} 15\Omega I_4 + 20\Omega(I_4 - I_3) + 25\Omega(I_4 - I_2) + 10\Omega(I_4) + 30\Omega(I_4) &= 0 \\ 15\Omega I_4 + 20\Omega I_4 - 20\Omega I_3 + 25\Omega I_4 - 25\Omega I_2 + 10\Omega I_4 + 30\Omega I_4 &= 0 \end{aligned}$$

Reorganizamos la ecuación para tener las corrientes desde el índice menor al mayor

$$\text{Malla } 4 \Longrightarrow -25\Omega I_2 - 20\Omega I_3 + 100\Omega I_4 = 0 \tag{1.6}$$

Repetimos el procedimiento para las otras 3 mallas del circuito

$$\begin{aligned} 5\Omega I_3 + 30\Omega(I_3 - I_1) + 10\Omega(I_3 - I_2) + 20\Omega(I_3 - I_4) &= 0 \\ 5\Omega I_3 + 30\Omega I_3 - 30\Omega I_1 + 10\Omega I_3 - 10\Omega I_2 + 20\Omega I_3 - 20\Omega I_4 &= 0 \end{aligned}$$

$$\text{Malla } 3 \Longrightarrow -30\Omega I_1 - 10\Omega I_2 + 65\Omega I_3 - 20\Omega I_4 = 0 \tag{1.7}$$

$$\begin{aligned} 10\Omega(I_2 - I_3) + 50\Omega(I_2 - I_1) + 15\Omega(I_2) + 25\Omega(I_2 - I_4) &= 0 \\ 10\Omega I_2 - 10\Omega I_3 + 50\Omega I_2 - 50\Omega I_1 + 15\Omega I_2 + 25\Omega I_2 - 25\Omega I_4 &= 0 \end{aligned}$$

$$\text{Malla } 2 \Longrightarrow -50\Omega I_1 + 100\Omega I_2 - 10\Omega I_3 - 25\Omega I_4 = 0 \tag{1.8}$$

$$\begin{aligned} 50\Omega(I_1 - I_2) + 30\Omega(I_1 - I_3) + 120 &= 0 \\ 50\Omega I_1 - 50\Omega I_2 + 30\Omega I_1 - 30\Omega I_3 &= -120 \end{aligned}$$

$$\text{Malla } 1 \Longrightarrow 80\Omega I_1 - 50\Omega I_2 - 30\Omega I_3 = -120 \tag{1.9}$$

Las ecuaciones en la forma de matriz pueden ser escrita de la siguiente manera

$$\begin{bmatrix} 0 & -25 & -20 & 100 \\ -30 & -10 & 65 & -20 \\ 50 & 100 & -10 & -25 \\ 80 & -50 & -30 & 0 \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ -120 \end{Bmatrix} \tag{1.10}$$

1.2.2. Pseudocódigo

Algorithm 2 Método de Gauss-Seidel

Entrada: array a,b,x; entero n; real lambda,es;
Salida: array x;
for i $\leftarrow$1 to n **do** ▷ Pedir datos de la matriz
  for j $\leftarrow$1 to n **do**
    a_{ij} $\leftarrow$float a_{ij}
  print ``‘‘Elemento a:’’``
  end for
  print ``‘‘Elemento b:’’``
  b_i $\leftarrow$float b_i
end for
while error > es **do**
  for i$\leftarrow$1 to n **do**
    $\lambda \leftarrow 0$;
    for j$\leftarrow$1 to n **do**
      if j$\neq$i **then**
        $\lambda = \lambda + a_{ij}x_j$
      end if
      $x_i = \dfrac{b_i - \lambda}{a_{ii}}$
    end for
  end for

  error=$\left|\dfrac{x - x_0}{100}\right|$
return print(x, error)
end while

1.2.3. Código solución al problema

1. Solución por método de Gauss-Seidel

```python
import numpy as np

n = int(input("Dimension de la matriz: "))

a = np.zeros([n,n]) #crear matriz con ceros
b = np.zeros([n]) #vector alternativo
x = np.zeros(n) #vector soluciones

#pedir datos de la matriz
for i in range(n):
    for j in range(n):
        a[i,j] = (input("Elemento a[" + str(i+1) + "," +
        str(j+1)+"]: "))
        a[i,j] = float(a[i,j])
    b[i] = (input("Elemento b[" + str(i+1)+"]: "))


def Gseid(a,b,n,x,es,Lambda):
    error = 100
    while(error > es):
        for i in range(n):
            sum = 0
            for j in range(n):
                if (j != i):
                    sum += a[i][j]*x[j]
            x[i] = (1 - Lambda)*x[i] + (Lambda/a[i][i])*
            (b[i]-sum)
        error = np.linalg.norm(np.matmul(a, x) - b)
        print('Error relativo porcentual:
        {0:10.6g}'.format(error))
    print("Soluciones al sistema", x)


Gseid(a,b,n,x,0.5,0.5)
```

CAPÍTULO 2

ANEXOS

2.1. Anexo A: Evidencia del funcionamiento de los código reportados

2.1.1. Problema 1

```
1| Soluciones: 13.1209,67.7512 | Error relativo:  27.046
2| Soluciones: 10.1150,64.6781 | Error relativo:  12.214
3| Soluciones: 8.2014,62.9450  | Error relativo:  6.973
4| Soluciones: 7.8210,61.3101  | Error relativo:  4.127
5| Soluciones: 7.0112,61.0316  | Error relativo:  2.411
6 | Soluciones:  6.9803,60.7802 | Error relativo:  1.367
```

Figura 2.1: Método de Newton-Raphson

2.1.2. Problema 2

```
1| Soluciones: 4.6818,3.6645,2.8193,1.4006 | Error relativo:  1.0464
2| Soluciones: 4.3824,2.9690,2.7280,1.2857 | Error relativo:  1.0215
3| Soluciones: 4.2073,2.6795,2.7197,1.2637 | Error relativo:  0.0929
4| Soluciones: 4.1966,2.6665,2.7321,1.2385 | Error relativo:  0.0325
5| Soluciones: 4.1952,2.6445,2.2121,1.2015 | Error relativo:  0.0064
6 | Soluciones:  4.1823,2.6645,2.7121,1.2085 | Error relativo:  0.0025
```

Figura 2.2: Método de Gauss-Seidel

2.2. Anexo B: Gráficos de los errores relativos por cada iteración

2.2.1. Circuito eléctrico

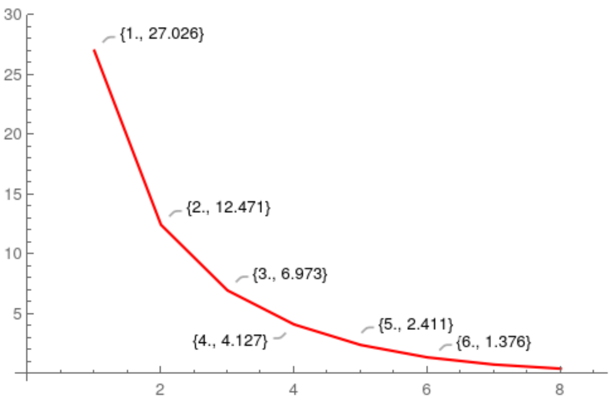

Figura 2.3: Error relativo por iteración

2.2.2. Problema de la trayectoria de un proyectil

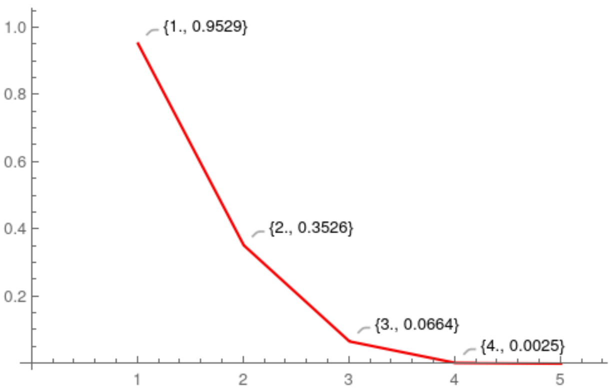

Figura 2.4: Error por iteración

CAPÍTULO 3

BIBLIOGRAFÍA

1. Chapra, S. C., Canale, R. P. (2011). *Numerical methods for engineers* (Vol. 1221). New York: Mcgraw-hill.

2. Kiusalaas, J. (2013). *Numerical methods in engineering with Python 3.* Cambridge university press.

3. Resnick, R., Halliday, D., Krane, K. (2004). *Física Vol. I. I.*