UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Universidad Autónoma de Querétaro

Facultad de Ingeniería

Métodos de Gauss-Seidel y Newton-Raphson

Análisis numérico

Autor:
David Gómez Torres

22 Septiembre del 2021

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

1.1. Gauss-Seidel

El método de **Gauss-Seidel** es el método iterativo más comúnmente usado. Suponga que se da un sistema de n ecuaciones:

$$[A]X = B$$

Suponga que se limita a un conjunto de ecuaciones de 3×3. Si los elementos de la diagonal no son todos cero, la primera ecuación se puede resolver para x_1, la segunda para x_2 y la tercera para x_3, para obtener:

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \tag{1.1}$$

$$x_2 = \frac{b_2 - a_{21}x_2 - a_{23}x_3}{a_{22}} \tag{1.2}$$

$$x_3 = \frac{b_3 - a_{31}x_2 - a_{32}x_3}{a_{33}} \tag{1.3}$$

Ahora, se puede empezar el proceso de solución al escoger valores iniciales para las x.

La convergencia se verifica usando el criterio:

$$|\varepsilon_{a,i}| = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| 100\,\% < \varepsilon_s \tag{1.4}$$

para todas las i, donde j y j – 1 son las iteraciones actuales y previas, respectivamente.

1.1.1. Criterio de convergencia

El criterio de convergencia se puede desarrollar al recordar que las condiciones suficientes para la convergencia de dos ecuaciones no lineales, $u(x, y)$ y $v(x, y)$, son

$$\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial u}{\partial y}\right| < 1 \tag{1.5}$$

y

$$\left|\frac{\partial v}{\partial x}\right| + \left|\frac{\partial v}{\partial y}\right| < 1 \tag{1.6}$$

Este criterio se aplica también a las ecuaciones lineales que se resuelven con el método de Gauss-Seidel. Entonces, las ecuaciones (1.5) y (1.6) también se reformulan así:

$$|a_{11}| > |a_{12}| \tag{1.7}$$

y

$$|a_{22}| > |a_{21}| \tag{1.8}$$

Esto es, el elemento diagonal debe ser mayor que el elemento fuera de la diagonal para cada renglón.

1.2. Método de Newton-Raphson

Para derivar el método de Newton-Raphson para un sistema de ecuaciones, empezamos con una expansión por serie de Taylor de $f_1(x)$ alrededor del punto x:

$$f_i(x + \Delta x) = f_i(x) + \sum_{f=1}^{n} \frac{\partial f_i}{\partial X_f} \Delta X_f + O(\Delta x^2) \tag{1.9}$$

Quitando los términos del orden Δx^2, podemos reescribir la ecuación (1.9) como

$$f(x + \Delta x) = f(x) + J(x)\Delta x$$

donde J(x) es la matriz Jacobiana (de tamaño $n \times n$) en términos de las derivadas parciales

$$J_{if} = \frac{\partial f_i}{\partial x_f}$$

Ya que puede llegar a ser difícil o impractico la derivación analítica de cada $\partial f_i / \partial X_i$, puede aproximarse como

$$\frac{\partial f_i}{\partial x_f} \approx \frac{f_i(x + e_f h) - f_i(x)}{h} \tag{1.10}$$

donde h es un pequeño incremento de x_f y e_f representa un vector unitario en la dirección de x_f.

CAPÍTULO 2

METODOLOGÍA

2.1. Sistema de concentraciones

Problema 11.9

El sistema de ecuaciones siguiente está diseñado para determinar concentraciones (las c están en g/m^3) en una serie de reactores acoplados como función de la cantidad de masa de entrada a cada uno de ellos (los lados derechos están en g/d),

$$\begin{aligned} 15c_1 - 3c_2 - c_3 &= 3800 \\ -3c_1 + 18c_2 - 6c_3 &= 1200 \\ -4c_1 - c_2 + 12c_3 &= 2350 \end{aligned}$$

Resuelva este problema con el método de Gauss-Seidel para $\varepsilon_s = 5\,\%$.

1. Solución por método de Gauss-Seidel

```
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)
```

2.2. Sistema de ecuaciones

Problema 11.11

Emplee el método de Gauss-Seidel para resolver el sistema siguiente hasta que el error relativo porcentual esté por debajo de $\varepsilon_s = 5\,\%$,

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 27 \\ -3x_1 - 6x_2 + 2x_3 &= -61.5 \\ x_1 + x_2 + 5x_3 &= -21.5 \end{aligned}$$

1. Método de Gauss-Seidel

```
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)
```

2.3. Intersección de dos circunferencias

Problema 23

Determina las coordinadas de dos puntos donde los círculos $(x-2)^2+y^2=4$ y $x^2+(y-3)^2=4$ intersectan. Inicia por estimar las locaciones de los puntos desde un dibujo de los círculos y después usa el método de Newton-Raphson para calcular las coordinadas.

1. Dibujo en Mathematica de las circunferencias del problema

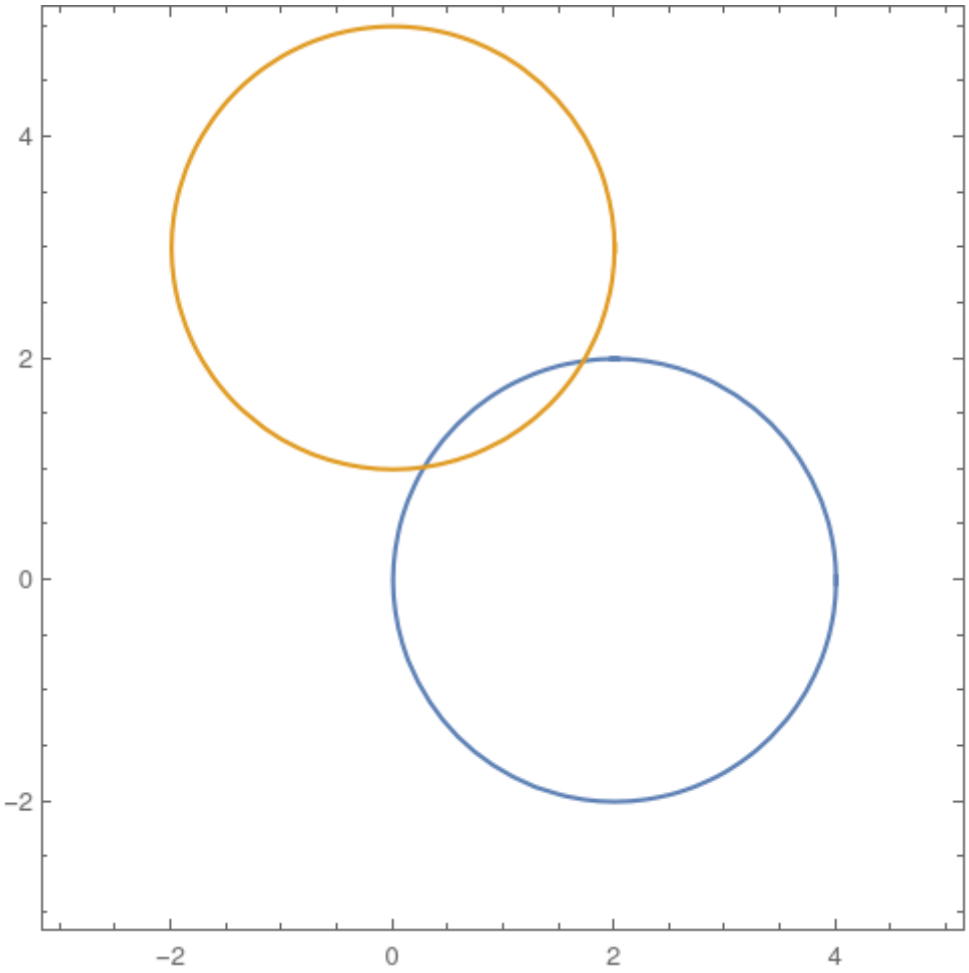

Figura 2.1: Circunferencias del problema

2. Método de Newton-Raphson

```
import numpy as np

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

x=np.array([-1,1])

def f(x): #ingresar las funciones
    #x[0] = x  /   x[1] = y
    f1 = (x[0]-2)**2+x[1]**2-4
    f2 = x[0]**2+(x[1]-3)**2-4
    return np.array([f1,f2])

def df(x): #calcular su derivada, manualmente
    return np.array([[2*x[0]-4,2*x[1]],
                     [2*x[0],2*x[1]-6]])

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,0.001)
```

2.4. Sistema de ecuaciones con funciones trigonométricas

Problema 24

Las ecuaciones

$$\sin x + 3\cos x - 2 = 0$$
$$\cos x - \sin y + 0.2 = 0$$

tienen una solución en la vecindad del punto (1,1). Use el método de Newton-Raphson para refinar la solución.

```
import numpy as np
import math

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

x=np.array([-1,1])

def f(x): #ingresar las funciones
    #x[0] = x  /   x[1] = y
    f1 = math.sin(x[0])+3*math.cos(x[0])-2
    f2 = math.cos(x[0])-math.sin(x[1])+0.2
    return np.array([f1,f2])

def df(x): #calcular su derivada, manualmente
    return np.array([[math.cos(x[0]),-3*math.sin(x[0])],
                     [-math.sin(x[0],-math.cos(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)
```

CAPÍTULO 3

ANEXOS

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

3.1.1. Ejercicio 11.9

```
Error relativo porcentual:   0.848442
Error relativo porcentual:    0.55978
Error relativo porcentual:    0.36933
Soluciones al sistema [320.1876768  227.17673494 321.48475873]
```

Figura 3.1: Método de Gauss-Seidel

3.1.2. Ejercicio 11.11

```
Error relativo porcentual:    4.12721
Error relativo porcentual:    2.41106
Error relativo porcentual:    1.37654
Error relativo porcentual:   0.773832
Error relativo porcentual:   0.432838
Soluciones al sistema [ 0.54874504  7.97331672 -5.98390707]
```

Figura 3.2: Método de Gauss-Seidel Sobrerrelajado

3.1.3. Ejercicio 23

```
Ingrese el numero de ecuaciones: 2
1 | Soluciones: [-0.07142857  0.78571429] | Error relativo:  0.9529760045804524
2 | Soluciones: [0.22197802 0.98131868] | Error relativo:  0.3526308390288958
3 | Soluciones: [0.27730257 1.01820171] | Error relativo:  0.06649183193553311
4 | Soluciones: [0.2794202  1.01961346] | Error relativo:  0.002545069636716528
5 | Soluciones: [0.27942331 1.01961554] | Error relativo:  3.7397006235360158e-06
```

Figura 3.3: Método Newton Raphson

3.1.4. Ejercicio 24

Observar en la figura (3.8) cómo la gráfica de iteración vs error, tiene un pico muy abrupto en la iteración 7. Se entiende que esto sea producto de la naturaleza del sistema, en funciones que oscilan, este efecto se puede producir.

```
12 | Soluciones: [-5.65868647  7.60883275] | Error relativo:  0.3738608789022294
13 | Soluciones: [-5.80183125  8.1232542 ] | Error relativo:  0.5339661512628431
14 | Soluciones: [-4.65138071  9.66530118] | Error relativo:  1.923914068187792
15 | Soluciones: [-4.65828693  9.26978179] | Error relativo:  0.39557967692734747
16 | Soluciones: [-5.04422847  8.88828357] | Error relativo:  0.5426709569036
17 | Soluciones: [-5.05455033  8.85987723] | Error relativo:  0.03022352003517206
```

Figura 3.4: Método Newton Raphson con el jacobiano

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

3.2.1. Ejercicio 11.9

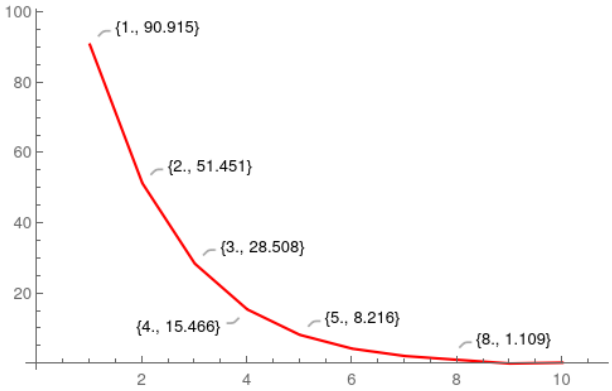

Figura 3.5: Error por iteración

3.2.2. Ejercicio 11.11

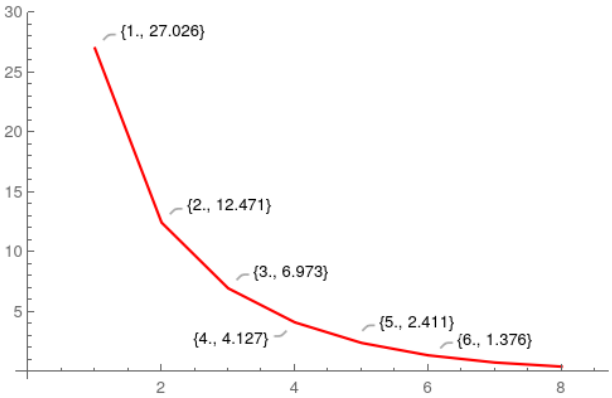

Figura 3.6: Error relativo porcentual por iteración

3.2.3. Ejercicio 23

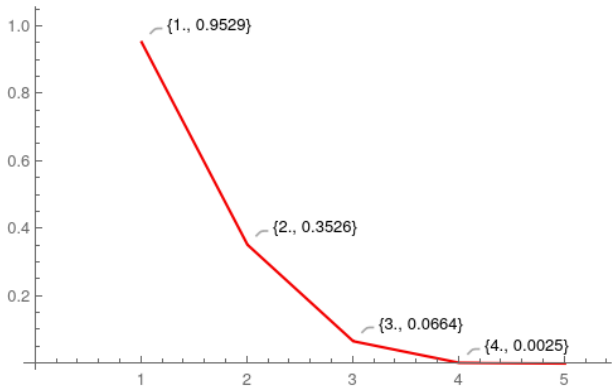

Figura 3.7: Gráfica del error por iteración del método de Newton-Raphson

3.2.4. Ejercicio 24

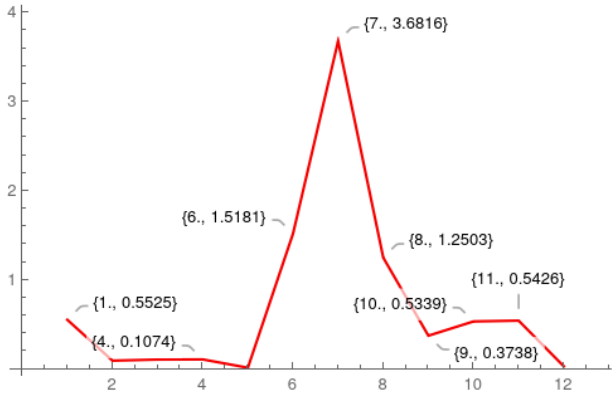

Figura 3.8: Gráfica con errores por iteración

CAPÍTULO 4

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.