MÉTODOS NUMÉRICOS

LABORATORIO 2

ALGORITMOS PARA BÚSQUEDA DE RAÍCES REALES

ALEXANDER REYES
Grupo 65

Docente:
HERNÁN ALONSO MANCIPE BOHÓRQUEZ
Ingeniero en Control y Tecnólogo en Electrónica

FUNDACIÓN UNIVERSITARIA INTERNACIONAL DE LA RIOJA - UNIR COLOMBIA
ESCUELA DE INGENIERÍA Y TECNOLOGÍA
PROGRAMA DE INGENIERÍA INFORMÁTICA
COLOMBIA

2024

Índice general

0.1. Introducción

En el ámbito agrícola, una de las plagas más comunes y dañinas es la mosca blanca, un insecto que afecta a una gran variedad de cultivos. Su presencia no solo inhibe el crecimiento de las plantas, sino que también transmite virus que deterioran la calidad de los productos cosechados. Para prevenir la propagación de esta plaga, es fundamental comprender su dinámica de crecimiento y tomar decisiones a tiempo. Este informe tiene como propósito estimar el tiempo necesario para que la población de moscas blancas alcance un determinado umbral crítico, utilizando el método numérico de Newton-Raphson. Esta técnica nos permitirá encontrar el momento preciso en que la población llega a un valor crítico de 110 individuos, con un error relativo aceptable.

0.2. Objetivos

1. **Objetivo general**: Determinar el tiempo necesario para que la población de moscas blancas alcance 110 individuos, utilizando el método de Newton-Raphson para aproximar la raíz de la función que modela su crecimiento.

2. **Objetivos específicos**:

   - Implementar la función de población y su derivada para aplicar el método de Newton-Raphson.
   - Realizar iteraciones del método para calcular el tiempo aproximado en que la población de moscas blancas alcanza el umbral establecido.
   - Consolidar los resultados de las iteraciones en una tabla y graficar la evolución de la población en función del tiempo.
   - Evaluar la efectividad del método utilizado y analizar los resultados obtenidos.

0.3. Algoritmo Desarrollado

El algoritmo desarrollado utiliza el método de Newton-Raphson para aproximar la cantidad de días necesarios para que la población de moscas blancas alcance 110 individuos. Este método es muy útil para encontrar raíces de ecuaciones no lineales, y en nuestro caso lo aplicamos para determinar el tiempo t que se requiere para alcanzar dicha población. A continuación, se describen los pasos del algoritmo:

1. **Definir la función y su derivada**: Se define la función de población $p(t) = (3e^(0{,}68t))/(10t)$y se reescribe la ecuación como $f(t) = p(t) - 110$ para encontrar el valor de t cuando $f(t) = 0$. Además, se deriva esta función para obtener $f'(t)$, que será utilizada en el método de Newton-Raphson.

2. **Inicializar valores**: Se toma un valor inicial para t, llamado x_0, mayor que 1. En este caso, se eligió $x_0 = 2$ como primera aproximación.

3. **Aplicar el método de Newton-Raphson**: Para cada iteración i, se aplica la fórmula: $x_i = x_0 - f(x_0)/f'(x_0)$ Luego, se calcula el error relativo para evaluar la convergencia del método: $Errorrelativo = |(x_i - x_0)/x_i|$

4. **Iterar hasta cumplir condiciones**: Se realizan iteraciones sucesivas hasta obtener un error relativo suficientemente pequeño. En este caso, se realizaron las primeras

tres iteraciones y se almacenaron los resultados.

5. **Mostrar resultados**: Finalmente, los resultados obtenidos se consolidan en una tabla y se graficó la función para visualizar el comportamiento de la población de moscas blancas.

0.3.1. Implementación del Algoritmo en Python

A continuación, se explica el código en Python que se desarrolló para resolver el problema utilizando el método de Newton-Raphson:

1. **Importar las bibliotecas necesarias**: Se importan las bibliotecas necesarias para el cálculo numérico (numpy), manejo de datos en tablas (pandas), graficación (matplotlib) y la interfaz gráfica (tkinter).

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
```

2. **Definir la función de población y su derivada**: Se define la función $f(t)$ que calcula la diferencia entre la población deseada y la estimada, y la función $f_prime(t)$ que es la derivada de $f(t)$.

```python
def f(t):
    return (3 * np.exp(0.68 * t)) / (10 * t) - 110

def f_prime(t):
    return (3 * (0.68 * np.exp(0.68 * t) * t - np.exp(0.68 * t))) / (10 * t ** 2)
```

3. **Inicializar el valor inicial y el proceso de iteración**: Inicializamos las variables necesarias, incluyendo el valor inicial x_0, el error relativo y un contador de iteraciones.

```python
x_0 = 2  # Aproximaci n inicial desde t > 1
error_relativo = 1
iteracion = 0
resultados = []
```

4. **Aplicar el método de Newton-Raphson para cada iteración**: Se ejecutan tres iteraciones del método de Newton-Raphson. En cada iteración, se calcula el nuevo valor de x_i, así como el error relativo, y se guarda la información de la iteración.

```python
while iteracion < 3:
    f_x0 = f(x_0)
    f_prime_x0 = f_prime(x_0)
    x_i = x_0 - f_x0 / f_prime_x0
    error_relativo = abs((x_i - x_0) / x_i)

    resultados.append([iteracion + 1, x_0, f_x0, f_prime_x0, x_i, error_relativo])

    x_0 = x_i
    iteracion += 1
```

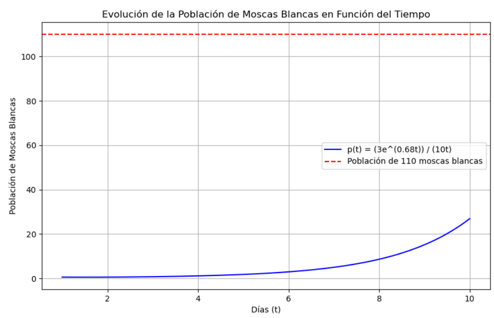
5. **Mostrar los resultados en una ventana emergente**: Se define una función para mostrar los resultados en una ventana emergente utilizando tkinter, mostrando la tabla con las iteraciones y sus resultados.

```python
def mostrar_tabla():
    ventana = tk.Tk()
```

```
    ventana.title("Resultados de Aproximaci n - Newton-Raphson")

    tabla = ttk.Treeview(ventana)
    tabla['columns'] = ['Iteraci n', 'x_0', 'f(x_0)', "f'(x_0)", 'x_i', 'Error relativo']
    tabla.column('#0', width=0, stretch=tk.NO)
    tabla.heading('#0', text='', anchor=tk.CENTER)

    for columna in ['Iteraci n', 'x_0', 'f(x_0)', "f'(x_0)", 'x_i', 'Error relativo']:
        tabla.column(columna, anchor=tk.CENTER, width=120)
        tabla.heading(columna, text=columna, anchor=tk.CENTER)

    for index, row in pd.DataFrame(resultados).iterrows():
        tabla.insert(parent='', index='end', iid=index, values=list(row))

    tabla.pack()
    ventana.mainloop()

mostrar_tabla()
```

6. **Graficar la evolución de la población**: Se define una Finalmente, se grafica la evolución de la población para visualizar cuándo se alcanza la cantidad de 110 moscas blancas.

```
t_values = np.linspace(1, 10, 400)
p_values = (3 * np.exp(0.68 * t_values)) / (10 * t_values)

plt.figure(figsize=(10, 6))
plt.plot(t_values, p_values, label='p(t) = (3e^(0.68t)) / (10t)', color='b')
plt.axhline(y=110, color='r', linestyle='--', label='Poblaci n de 110 moscas blancas')
plt.xlabel('D as (t)')
plt.ylabel('Poblaci n de Moscas Blancas')
plt.title('Evoluci n de la Poblaci n de Moscas Blancas en Funci n del Tiempo')
plt.legend()
plt.grid()
plt.show()
```



0.4. Explicación Paso a Paso de las Ecuaciones y Cálculos

0.4.1. Función de Población y Ecuación Definida

La función que describe el crecimiento de la población de moscas blancas es:
$p(t) = (3e^(0{,}68t))/(10t)$
Queremos determinar el momento en el cual la población alcanza 110 individuos. Para ello, planteamos la ecuación:
$f(t) = (3e^(0{,}68t))/(10t) - 110 = 0$
La derivada de esta función es necesaria para aplicar el método de Newton-Raphson, y se obtiene como:
$f'(t) = (3 * (0{,}68 * e^(0{,}68t) * t - e^(0{,}68t)))/(10t^2)$

0.4.2. Método de Newton-Raphson

El método de Newton-Raphson utiliza la siguiente fórmula para encontrar una mejor aproximación de la raíz de la ecuación $f(t) = 0$:
$x_(i+1) = x_i - f(x_i)/f'(x_i)$
En cada iteración se calculan $f(x_i) y f'(x_i)$, y luego se actualiza el valor de x_i para aproximarse al valor de t que satisface la ecuación.

Los valores calculados durante las primeras tres iteraciones fueron:

- **Iteración 1**: Se tomó $x_0 = 2$.
  - $f(x_0) = -98{,}9528$
  - $f'(x_0) = 14{,}3219$
  - $x_1 = x_0 - f(x_0)/f'(x_0) = 8{,}9119$
  - $Errorrelativo: 0{,}7758$
- **Iteración 2**: Con $x_0 = 8{,}9119$.
  - $f(x_0) = 51{,}9238$
  - $f'(x_0) = 4{,}2765$
  - $x_2 = 7{,}6944$
  - $Errorrelativo: 0{,}1583$
- **Iteración 3**: Con $x_0 = 7{,}6944$.
  - $f(x_0) = 1{,}9894$
  - $f'(x_0) = 3{,}5749$
  - $x_3 = 7{,}1390$
  - $Errorrelativo: 0{,}0777$

0.5. Consolidación de Iteraciones y Resultados en una Tabla

A continuación, se presenta la tabla con los resultados de las tres primeras iteraciones del método de Newton-Raphson:

Iter	x_0	$f(x_0)$	$f'(x_0)$	x_i	Erelat
1	2.0000	-98.9528	14.3219	8.9119	0.7758
2	8.9119	51.9238	4.2765	7.6944	0.1583
3	7.6944	1.9894	3.5749	7.1390	0.0777

0.6. Conclusiones

El método de Newton-Raphson es un método iterativo efectivo para encontrar raíces de ecuaciones no lineales, siempre que se tenga una buena aproximación inicial y que la función y su derivada estén bien definidas. En este caso, a través de tres iteraciones se obtuvo una buena aproximación del valor de t necesario para alcanzar una población de 110 moscas blancas.

A partir de los resultados de las iteraciones, podemos observar que el valor de t converge rápidamente a la solución, reduciendo el error relativo en cada paso. Después de tres iteraciones, se obtuvo un valor aproximado de $t = 7{,}1390$, lo cual indica que en aproximadamente 7 días la población alcanzará los 110 individuos.

Es importante notar que, aunque el método es efectivo, depende de la elección inicial de x_0, y puede no converger si dicha elección no es adecuada o si la función tiene múltiples raíces. En este problema, el valor inicial elegido permitió una convergencia adecuada y rápida al valor buscado.

0.7. Anexos

A continuación se anexa el código del programa

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk

# Definir la funci n y su derivada
def f(t):
    return (3 * np.exp(0.68 * t)) / (10 * t) - 110

def f_prime(t):
    return (3 * (0.68 * np.exp(0.68 * t) * t - np.exp(0.68 * t))) / (10 * t ** 2)

# Inicializar variables para el m todo de Newton-Raphson
x_0 = 2  # Aproximaci n inicial desde t > 1
error_relativo = 1
iteracion = 0

# Crear lista para almacenar los resultados
resultados = []

# Realizar tres iteraciones del m todo de Newton-Raphson
while iteracion < 3:
    f_x0 = f(x_0)
    f_prime_x0 = f_prime(x_0)
    x_i = x_0 - f_x0 / f_prime_x0
    error_relativo = abs((x_i - x_0) / x_i)

    # Agregar resultados a la lista
    resultados.append([iteracion + 1, x_0, f_x0, f_prime_x0, x_i, error_relativo])

    # Actualizar para la siguiente iteraci n
    x_0 = x_i
    iteracion += 1

# Convertir los resultados en un DataFrame
columnas = ['Iteraci n', 'x_0', 'f(x_0)', "f'(x_0)", 'x_i', 'Error relativo']
df_resultados = pd.DataFrame(resultados, columns=columnas)

# Mostrar la tabla de resultados en una ventana emergente
def mostrar_tabla():
    ventana = tk.Tk()
    ventana.title("Resultados de Aproximaci n - Newton-Raphson")

    tabla = ttk.Treeview(ventana)
    tabla['columns'] = columnas
    tabla.column('#0', width=0, stretch=tk.NO)
    tabla.heading('#0', text='', anchor=tk.CENTER)

    for columna in columnas:
        tabla.column(columna, anchor=tk.CENTER, width=120)
        tabla.heading(columna, text=columna,
```

```
                anchor=tk.CENTER)

    for index, row in df_resultados.iterrows():
        tabla.insert(parent='', index='end', iid=index, values=list(row))

    tabla.pack()
    ventana.mainloop()

mostrar_tabla()

# Graficar la funci n para visualizar la soluci n
t_values = np.linspace(1, 10, 400)
p_values = (3 * np.exp(0.68 * t_values)) / (10 * t_values)

plt.figure(figsize=(10, 6))
plt.plot(t_values, p_values, label='p(t) = (3e^(0.68t)) / (10t)', color='b')
plt.axhline(y=110, color='r', linestyle='--', label='Poblaci n de 110 moscas blancas')
plt.xlabel('D as (t)')
plt.ylabel('Poblaci n de Moscas Blancas')
plt.title('Evoluci n de la Poblaci n de Moscas Blancas en Funci n del Tiempo')
plt.legend()
plt.grid()
plt.show()
```