Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

E.P. de Ingeniería Estadística e Informática

Docente: Ing. Torres Cruz Fred
Presentado por: Quispe Ito Luz Leidy

Metodo de Punto Fijo

1. ¿Qué es el Método de Punto Fijo?

El **método de punto fijo** es un procedimiento numérico utilizado para encontrar una aproximación a la solución real de una ecuación no lineal de la forma:

$$f(x) = 0.$$

En lugar de resolver directamente esta ecuación, se transforma en una expresión equivalente:

$$x = g(x),$$

donde la raíz buscada corresponde a un *punto fijo* de la función $g(x)$. Un punto fijo es un número x^* que cumple:

$$g(x^*) = x^*.$$

El método consiste en elegir un valor inicial x_0 y generar una sucesión mediante la fórmula iterativa:

$$x_{n+1} = g(x_n),$$

hasta que la diferencia entre dos aproximaciones consecutivas sea menor que una tolerancia establecida.

Idea principal

La idea fundamental del método es que, si la función $g(x)$ cumple ciertas condiciones de convergencia, la sucesión generada por:

$$x_{n+1} = g(x_n)$$

se acercará progresivamente al punto fijo x^*, que constituye la solución aproximada de la ecuación original.

Condición de convergencia

Para garantizar que el método converge, se requiere que en un intervalo alrededor de la solución se cumpla:

$$|g'(x)| < 1.$$

Si esta condición no se satisface, la sucesión puede divergir y alejarse de la raíz.

2. Objetivo del programa

El programa desarrollado en Python implementa el método numérico de **punto fijo**, cuyo objetivo es encontrar una solución real de la ecuación:

$$f(x) = 0$$

a través de su transformación en una función equivalente:

$$x = g(x).$$

El método aproxima la raíz mediante la fórmula iterativa:

$$x_{n+1} = g(x_n),$$

hasta que el error entre dos aproximaciones consecutivas sea menor que la tolerancia establecida por el usuario.

2.1 Funcionamiento general

El programa solicita al usuario la siguiente información desde la consola:

- La función de iteración $g(x)$, la cual debe cumplir condiciones de convergencia.
- El valor inicial x_0.
- La tolerancia o error máximo permitido.
- El número máximo de iteraciones.

Luego de recibir estos datos, el programa ejecuta el proceso iterativo del método de punto fijo para encontrar la aproximación de la raíz.

2.2 Procedimiento del algoritmo

El programa sigue los siguientes pasos:

1. Verificación inicial

- Se calcula un primer valor $x_1 = g(x_0)$.
- Se comprueba que la función esté correctamente definida.
- Opcionalmente, se verifica la condición de convergencia:

$$|g'(x)| < 1.$$

Cálculo iterativo

Mientras el error sea mayor que la tolerancia, se realizan las siguientes operaciones:

1. Se calcula el siguiente valor:
$$x_{n+1} = g(x_n).$$
2. Se obtiene el error:
$$E_n = |x_{n+1} - x_n|.$$
3. Se actualiza el valor para la siguiente iteración.
4. Se registran los valores en una tabla que incluye:
$$n,\ x_n,\ g(x_n),\ E_n.$$

El proceso continúa hasta que:

$$E_n < \text{tolerancia}.$$

2.3 Restricciones del programa

Para que el método funcione correctamente deben cumplirse las siguientes condiciones:

- La ecuación debe poder reescribirse como $x = g(x)$.
- La condición de convergencia local debe cumplirse:

$$|g'(x)| < 1.$$

- El valor inicial x_0 debe ser cercano a la raíz.
- La función debe ser válida y compatible con el módulo `math` de Python.

3. Programa en Python

```
import math, re, matplotlib.pyplot as plt

print("=== M TODO DEL PUNTO FIJO ===")

def preparar_funcion(expr):
    expr = expr.lower().strip()
    expr = expr.replace("^", "**").replace("sen", "sin").replace("ln", "
       log").replace("  ", "pi")
    expr = re.sub(r'(\d)([a-zA-Z\(])', r'\1*\2', expr)
    expr = re.sub(r'([a-zA-Z\)])(\d)', r'\1*\2', expr)
    return expr

# --- Ingreso de la funci n original ---
funcion_f = input("Ingrese la funci n f(x): (ejemplo: e^x - 4x)     ")
funcion_f = preparar_funcion(funcion_f)

g_str = None

match = re.match(r"e\*\*x\s*-\s*(\d+)\*?x", funcion_f)
if match:
    a = float(match.group(1))
    g_str = f"log({a}*x)"
    print(f"Transformaci n detectada autom ticamente:  g(x) = ln({a}x)")

# Si no se reconoce, pedir g(x)
if g_str is None:
    print("No se detect  una transformaci n autom tica.")
    g_str = input("Ingrese la transformaci n g(x): ")

# --- Definir funciones ---
def f(x):
    try:
        return eval(funcion_f, {"__builtins__": None}, math.__dict__ | {"x
           ": x})
    except:
        return None

def g(x):
    try:
        return eval(preparar_funcion(g_str), {"__builtins__": None}, math.
           __dict__ | {"x": x})
    except:
        return None

# --- Par metros iniciales ---
x0 = float(input("Ingrese el valor inicial x0 = "))
tol = 1e-10
max_iter = 100

itera = 0
xr = x0
```

```
error = 100
valores = [x0]

print(f"\n{'Itera':<6} {'x_i':<15} {'x_i+1':<15} {'f(x)':<15} {'g(x)':<15} {'Error(%)':<10}")
print("-" * 80)

# --- Iteraciones ---
while itera < max_iter and error > tol:
    x_ant = xr
    xr = g(x_ant)
    if xr is None:
        break
    fx = f(x_ant)
    error = abs((xr - x_ant) / xr) * 100
    itera += 1
    valores.append(xr)
    print(f"{itera:<6} {x_ant:<15.9f} {xr:<15.9f} {fx:<15.9f} {g(xr):<15.9f} {error:<10.3f}")

# --- Resultados ---
print("\n--- RESULTADOS ---")
print(f"xr = {xr:.9f}")
print(f"g(xr) = {g(xr):.9f}")
print(f"error = {error:.9f}%")
print(f"iteraciones = {itera}")

# --- Gr fico ---
xs = [xr - 2 + i*4/200 for i in range(201)]
ys = [g(x) for x in xs]

plt.plot(xs, ys, label=f"g(x)={g_str}")
plt.plot(xs, xs, color='gray', linestyle='--', label="y=x")
plt.scatter(valores, [g(v) for v in valores], color='red', label='Aproximaciones')
plt.title("M todo de Punto Fijo")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```

```
=== MÉTODO DEL PUNTO FIJO ===
Ingrese la función f(x): (ejemplo: e^x - 4x) → (e^x)-4+x
No se detectó una transformación automática.
Ingrese la transformación g(x): ln(4-x)
Ingrese el valor inicial x0 = 1

Itera  x_i            x_i+1          f(x)           g(x)           Error(%)
--------------------------------------------------------------------------------
1      1.000000000    1.098612289    -0.281718172   1.065189144    8.976
2      1.098612289    1.065189144    0.098612289    1.076643007    3.138
3      1.065189144    1.076643007    -0.033423145   1.072732611    1.064
4      1.076643007    1.072732611    0.011453863    1.074069356    0.365
5      1.072732611    1.074069356    -0.003910396   1.073612599    0.124
6      1.074069356    1.073612599    0.001336745    1.073768693    0.043
7      1.073612599    1.073768693    -0.000456757   1.073715352    0.015
8      1.073768693    1.073715352    0.000156094    1.073733580    0.005
9      1.073715352    1.073733580    -0.000053342   1.073727351    0.002
10     1.073733580    1.073727351    0.000018229    1.073729480    0.001
11     1.073727351    1.073729480    -0.000006229   1.073728752    0.000
12     1.073729480    1.073728752    0.000002129    1.073729001    0.000
13     1.073728752    1.073729001    -0.000000727   1.073728916    0.000
14     1.073729001    1.073728916    0.000000249    1.073728945    0.000
15     1.073728916    1.073728945    -0.000000085   1.073728935    0.000
16     1.073728945    1.073728935    0.000000029    1.073728938    0.000
17     1.073728935    1.073728938    -0.000000010   1.073728937    0.000
18     1.073728938    1.073728937    0.000000003    1.073728938    0.000
19     1.073728937    1.073728938    -0.000000001   1.073728938    0.000
20     1.073728938    1.073728938    0.000000000    1.073728938    0.000
21     1.073728938    1.073728938    -0.000000000   1.073728938    0.000
22     1.073728938    1.073728938    0.000000000    1.073728938    0.000
23     1.073728938    1.073728938    -0.000000000   1.073728938    0.000
24     1.073728938    1.073728938    0.000000000    1.073728938    0.000
25     1.073728938    1.073728938    -0.000000000   1.073728938    0.000

--- RESULTADOS ---
xr = 1.073728938
g(xr) = 1.073728938
error = 0.000000000%
iteraciones = 25
```