Programa: sisEcuNoLin (sistema de ecuaciones no lineales)

por: Ricardo Santana

A. Introducción

El código proporcionado la funcion SENL que implementa el método de Newton-Raphson para resolver un sistema de ecuaciones no lineales, utilizando como herramienta la función jacobiano.

B. SENL

Entrada:

- Vector inicial $x0 \in \mathbb{R}^{n\times 1}$
- Vector funciones (definidas en sympy) $F \in \mathbb{R}^{n\times 1}$
- máximo número de iteraciones $itera \in \mathbb{Z}$
- tolerancia $tol \in \mathbb{R}$

Salida:

- Vector $x0 \in \mathbb{R}^{n\times 1}$, resultado del sistema **F = 0**
- String 'numero de ecuaciones y variables diferentes' en casos de que el número de ecuaciones e incognitas no coincidan.

C. Definición de la función jacobiano(F, var)

La función jacobiano calcula el jacobiano de un sistema de ecuaciones representado simbólicamente. Toma dos argumentos: F, que es una lista de funciones que representan el sistema de ecuaciones, y var, que es una lista de variables simbólicas del sistema.

Dentro de la función, se crea una matriz J de tamaño (len(F), len(var)) utilizando la función zeros de sympy. Luego, se utiliza un bucle for anidado para iterar sobre las funciones fi en F y las variables s en var. En cada iteración, se utiliza la función diff de sympy para calcular la derivada parcial de fi con respecto a s y se asigna el resultado a la posición correspondiente en la matriz J.

Finalmente, la función devuelve la matriz J como resultado.

D. Programa

1. Se convierte el vector x0 en un array de numpy utilizando la función array de numpy

1. Se convierte el vector x0 en un array de numpy utilizando la función array de numpy.
2. Se verifica si la longitud de x0 es igual a la longitud de F. Si no son iguales, se devuelve un mensaje de error indicando que el número de ecuaciones y variables es diferente y que el sistema no se puede resolver.
3. Se crea una cadena de caracteres letras que representa las variables simbólicas del sistema. La cadena se construye concatenando las letras 'x' seguidas de un número que va desde 1 hasta la longitud de x0.
4. Se utiliza la función symbols de sympy para crear una lista de variables simbólicas a partir de la cadena letras.
5. Se crea una matriz F utilizando la función Matrix de sympy y la lista de funciones F.
6. Se utiliza la función lambdify de sympy para convertir las funciones simbólicas en funciones numéricas que pueden ser evaluadas por numpy. Se crea una función F0 que toma como argumentos las variables simbólicas y devuelve el valor numérico de F.
7. Se llama a la función jacobiano para calcular el jacobiano del sistema de ecuaciones y se asigna el resultado a la variable J.
8. Se utiliza la función lambdify para convertir el jacobiano simbólico en una función numérica J0 que puede ser evaluada por numpy.
9. Se inicializa una variable k con el valor 1 para contar el número de iteraciones.
10. Se inicia un bucle while que se ejecuta mientras k sea menor o igual a itera.
11. Dentro del bucle, se utiliza un bloque try-except para manejar sistemas con diferentes números de variables. En el bloque try, se evalúan las funciones F0 y J0 utilizando los valores actuales de x0. En el bloque except, se evalúan las funciones F0 y J0 con los valores correspondientes de x0.
12. Se utiliza la función solve de numpy para resolver el sistema de ecuaciones lineales J_x * y = -F_x, donde J_x es el jacobiano evaluado en x0 y F_x es el sistema de ecuaciones evaluado en x0. El resultado y es un vector que representa la corrección a x0 en cada iteración.
13. Se actualiza x0 sumándole y.
14. Se verifica si la norma del vector y es menor que la tolerancia tol. Si es así, se devuelve x0 como resultado.
15. Se incrementa k en 1.
16. Si el bucle while termina sin alcanzar la condición de convergencia, se imprime un mensaje indicando que se ha excedido el número máximo de iteraciones.
17. Finalmente, se devuelve x0 como resultado.

D. Código

```
import numpy as np
import sympy as sp

def jacobiano(F, var):
    J = sp.zeros(len(F),len(var))
    for i, fi in enumerate(F):
        for j, s in enumerate(var):
            J[i,j] = sp.diff(fi, s)
    return J

def SENL(x0, F, itera, tol):
    x0 = np.array(x0)

    if len(x0) != len(F):
        return 'numero de ecuaciones y variables diferentes, el sistema no se puede resolv

    letras = ''
    for i in range(len(x0)):
        letras += f'x{i+1} '
    var = sp.symbols(letras)

    F = sp.matrices.Matrix(F)

    F0 = sp.lambdify([i for i in var], F, 'numpy')

    J = jacobiano(F, var)

    J0 = sp.lambdify([i for i in var], J, 'numpy')

    k = 1
    while k <= itera:

        try: #sistema con tres variables
            F_x = F0(x0[0], x0[1], x0[2]) #agregar x[n] donde n es el numero de elementos
            J_x = J0(x0[0], x0[1], x0[2])
        except: #sistema con dos variables
            F_x = F0(x0[0], x0[1]) #agregar x[n] donde n es el numero de elementos de x0
            J_x = J0(x0[0], x0[1])

        y = np.linalg.solve(J_x, -F_x).reshape(1,len(x0))[0]
        x0 = x0 + y
        #print(x0)
        if np.linalg.norm(y) < tol:
            return x0
        k += 1
    print('Número máximo de iteraciones excedido')
```