Cálculo Diferencial con SymPy

1. Introducción a SymPy

SymPy es una biblioteca de Python para matemáticas simbólicas. Permite realizar operaciones matemáticas de forma exacta, en lugar de aproximada como lo hacen otras bibliotecas como NumPy.

2. Conceptos básicos

- **Símbolos**: En SymPy, las variables deben ser definidas como símbolos.

```
from sympy import symbols

x, y, z = symbols('x y z')
```

- **Expresiones**: Una vez definidos los símbolos, podemos crear expresiones matemáticas.

```
expr = x**2 + 2*x + 1
```

3. Derivación

Para derivar una función en SymPy, utilizamos la función `diff`.

- **Derivada simple**:

```
from sympy import diff

expr = x**2 + 2*x + 1
derivada = diff(expr, x)
print(derivada)  # 2*x + 2
```

- **Derivadas de orden superior**:

```
segunda_derivada = diff(expr, x, 2)
print(segunda_derivada)  # 2
```

- **Derivadas parciales**:

Si tienes una función de varias variables, puedes derivarla respecto a una de ellas manteniendo las otras constantes.

```
f = x**2 + y**2
df_dx = diff(f, x)  # 2*x
df_dy = diff(f, y)  # 2*y
```

4. Evaluar derivadas

Para evaluar una derivada en un punto específico, utilizamos el método `subs`.

```
derivada = 2*x + 2
valor_en_3 = derivada.subs(x, 3)
print(valor_en_3)  # 8
```

5. Simplificación

SymPy puede simplificar expresiones para nosotros usando la función `simplify`.

```
from sympy import simplify

expr = (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
simplified_expr = simplify(expr)
print(simplified_expr)  # x - 1
```

Solución de Ecuaciones con SymPy

1. Introducción

SymPy no solo es útil para cálculo diferencial e integral, sino que también es una herramienta poderosa para resolver ecuaciones algebraicas y trascendentales.

2. Conceptos básicos

- **Ecuaciones**: En SymPy, una ecuación se crea utilizando la clase `Eq`.

```
from sympy import Eq, symbols

x = symbols('x')
ecuacion = Eq(x**2 - 5*x + 6, 0)
```

3. Resolviendo ecuaciones

Para resolver ecuaciones, utilizamos la función `solve`.

- **Ecuaciones algebraicas**:

```
from sympy import solve

solucion = solve(ecuacion, x)
print(solucion)  # [2, 3]
```

Esto indica que la ecuación $(x^2 - 5x + 6 = 0)$ tiene soluciones $(x = 2)$ y $(x = 3)$.

- **Ecuaciones trascendentales**:

Supongamos que queremos resolver la ecuación $\sin(x) = \frac{1}{2}$:

```
from sympy import sin

ecuacion_trascendental = Eq(sin(x), 1/2)
solucion_trascendental = solve(ecuacion_trascendental, x)
print(solucion_trascendental)
```

4. Sistemas de ecuaciones

SymPy también puede resolver sistemas de ecuaciones lineales.

Supongamos que queremos resolver el siguiente sistema:

```
x + y = 5
x - y = 1
```

```
y = symbols('y')
sistema = [
    Eq(x + y, 5),
    Eq(x - y, 1)
]

solucion_sistema = solve(sistema, (x, y))
print(solucion_sistema)  # {x: 3, y: 2}
```

5. Ecuaciones no lineales

SymPy también puede intentar resolver ecuaciones no lineales, aunque no siempre garantiza una solución en función de la complejidad de la ecuación.

```
ecuacion_no_lineal = Eq(x**2 + sin(x), 0)
solucion_no_lineal = solve(ecuacion_no_lineal, x)
print(solucion_no_lineal)
```

Estas son algunas de las capacidades de SymPy en cuanto a la resolución de ecuaciones.

Es importante recordar que, aunque SymPy es una herramienta poderosa, no todas las ecuaciones tienen soluciones cerradas o pueden ser resueltas simbólicamente.

En esos casos, se pueden usar métodos numéricos o aproximaciones.

Puntos Críticos con SymPy

1. Definición de Punto Crítico

Un punto crítico de una función `f(x)` ocurre donde su derivada es cero `f'(x) = 0` o está indefinida.

2. Conceptos básicos

- **Símbolos**: En SymPy, las variables deben ser definidas como símbolos.

```
from sympy import symbols

x = symbols('x')
```

- **Expresiones**: Una vez definidos los símbolos, podemos crear expresiones matemáticas.

```
f = x**3 - 3*x**2 + 2*x
```

3. Encontrando la primera derivada

Para encontrar los puntos críticos, primero derivamos la función.

```
from sympy import diff

f_prime = diff(f, x)
```

4. Resolviendo para (f'(x) = 0)

Usamos la función `solve` para encontrar los valores de (x) donde (f'(x) = 0).

```
from sympy import solve

critical_points = solve(f_prime, x)
```

5. Determinando la naturaleza de los puntos críticos

Para determinar si un punto crítico es un máximo, mínimo o punto de inflexión, podemos usar la segunda derivada, `f''(x)` .

- Si `f''(x) > 0`, el punto es un mínimo local.
- Si `f''(x) < 0`, el punto es un máximo local.
- Si `f''(x) = 0`, el test es inconclusivo.

```
f_double_prime = diff(f_prime, x)

for point in critical_points:
    if f_double_prime.subs(x, point) > 0:
        print(f"{point} es un mínimo local.")
    elif f_double_prime.subs(x, point) < 0:
        print(f"{point} es un máximo local.")
    else:
        print(f"{point} es no es concluyente.")
```

Estos pasos te permitirán encontrar y clasificar los puntos críticos de una función utilizando SymPy.

Es importante recordar que los puntos críticos son lugares donde la función puede cambiar de dirección, pero no garantizan que la función tenga un máximo o mínimo absoluto.

Para determinar máximos o mínimos absolutos, es posible que también debas evaluar los valores de la función en los extremos de su dominio o en cualquier punto donde la función esté indefinida.