Contents

Cheatsheet: Librerías para Integrales Indefinidas en Análisis Económico

SYMPY - Álgebra Simbólica

`sp.symbols()`

¿Qué hace? Crea variables simbólicas para manipulación algebraica

Sintaxis:

```
variable = sp.symbols('nombre', restricciones)
```

Parámetros: - `'nombre'`: String con el nombre de la variable - `positive=True`: Variable estrictamente positiva (> 0) - `nonnegative=True`: Variable no negativa (0) - `real=True`: Variable de números reales - `integer=True`: Variable entera

Input: Nombre y propiedades
Output: Objeto Symbol de SymPy

Casos de uso: - Definir variable de cantidad: `q = sp.symbols('q', positive=True)` - Definir constante desconocida: `F = sp.symbols('F')` - Múltiples variables: `q1, q2 = sp.symbols('q1 q2', real=True)`

¿Cuándo usar? SIEMPRE al inicio, antes de definir cualquier función

Ejemplo:

```
q = sp.symbols('q', positive=True)  # Cantidad de producción
F = sp.symbols('F')                  # Costo fijo (sin restricción)
```

`sp.integrate()`

¿Qué hace? Calcula la integral indefinida (antiderivada) de una función

Sintaxis:

`resultado = sp.integrate(función, variable)`

Parámetros: - `función`: Expresión simbólica a integrar - `variable`: Variable de integración (ej: q, x, t)

Input: Expresión matemática y variable
Output: Expresión simbólica de la antiderivada (SIN constante +C automática)

Casos de uso: - Recuperar costo total: `C = sp.integrate(CMg, q) + F` - Recuperar ingreso total: `I = sp.integrate(IMg, q)` - Calcular área bajo curva

¿Cuándo usar? Cuando tienes una función marginal y necesitas la función total

IMPORTANTE: SymPy NO agrega automáticamente la constante de integración, debes hacerlo manualmente con `+ F`

Ejemplo:

```
CMg = 20 + 2*q
C = sp.integrate(CMg, q) + F  # ← Agregar +F manualmente
# Resultado: C = q² + 20q + F
```

`sp.diff()`

¿Qué hace? Calcula la derivada de una función

Sintaxis:

`derivada = sp.diff(función, variable, orden)`

Parámetros: - `función`: Expresión a derivar - `variable`: Variable respecto a la cual derivar - `orden` (opcional): Orden de derivación (1 por defecto)

Input: Expresión matemática y variable
Output: Expresión simbólica de la derivada

Casos de uso: - Encontrar función marginal: `CMg = sp.diff(C, q)` - Encontrar puntos críticos: `dB = sp.diff(B, q)` - Verificar concavidad: `d2B = sp.diff(B, q, 2)` (segunda derivada)

¿Cuándo usar? Para encontrar máximos/mínimos o verificar integrales

Ejemplo:

```
C = q**2 + 20*q + 100
CMg = sp.diff(C, q)  # → 2q + 20

# Segunda derivada
d2C = sp.diff(C, q, 2)  # → 2
```

`sp.solve()`

¿Qué hace? Resuelve ecuaciones algebraicas

Sintaxis:

```
soluciones = sp.solve(ecuación, variable)
```

Parámetros: - `ecuación`: Ecuación a resolver (usar `sp.Eq()`) o expresión igualada a cero - `variable`: Variable a despejar

Input: Ecuación y variable incógnita
Output: Lista de soluciones (¡SIEMPRE es una lista!)

Casos de uso: - Determinar constante: `F_sol = sp.solve(sp.Eq(C.subs(q, 0), 100), F)[0]` - Encontrar equilibrio: `q_opt = sp.solve(sp.Eq(IMg, CMg), q)[0]` - Puntos críticos: `q_min = sp.solve(sp.Eq(derivada, 0), q)[0]`

¿Cuándo usar? Para encontrar valores específicos de variables

CRÍTICO: Usar `[0]` para extraer la primera solución de la lista

Ejemplo:

```
# Con sp.Eq() (recomendado)
ecuacion = sp.Eq(q**2 + 20*q + F, 100)
F_sol = sp.solve(ecuacion, F)[0]  # → -q² - 20q + 100

# Sin sp.Eq() (expresión = 0)
soluciones = sp.solve(q**2 - 4, q)  # → [-2, 2]
```

`sp.Eq()`

¿Qué hace? Crea una ecuación de igualdad

Sintaxis:

```
ecuacion = sp.Eq(lado_izquierdo, lado_derecho)
```

Parámetros: - `lado_izquierdo`: Expresión del lado izquierdo - `lado_derecho`: Expresión del lado derecho

Input: Dos expresiones
Output: Objeto Equality

Casos de uso: - Condiciones iniciales: `sp.Eq(C.subs(q, 0), 100)` → C(0) = 100 - Equilibrios: `sp.Eq(IMg, CMg)` → IMg = CMg - Puntos críticos: `sp.Eq(derivada, 0)` → f'(x) = 0

¿Cuándo usar? Cuando necesitas resolver ecuaciones con `sp.solve()`

Ejemplo:

```
# Encontrar costo fijo cuando C(0) = 100
condicion = sp.Eq(C.subs(q, 0), 100)
F_sol = sp.solve(condicion, F)[0]
```

`sp.simplify()`

¿Qué hace? Simplifica expresiones algebraicas complejas

Sintaxis:

`expresion_simple = sp.simplify(expresión_compleja)`

Parámetros: - `expresión_compleja`: Expresión a simplificar

Input: Expresión simbólica
Output: Expresión simplificada

Casos de uso: - Simplificar costo medio: `CMed = sp.simplify(C / q)` - Reducir beneficios: `B = sp.simplify(I - C)` - Limpiar resultados: `resultado = sp.simplify(expresion_larga)`

¿Cuándo usar? Después de operaciones que generan expresiones complicadas

¿Por qué usar? Facilita la interpretación y reduce errores numéricos

Ejemplo:

```
C = q**2 + 20*q + 100
CMed = C / q  # → (q² + 20q + 100)/q
CMed = sp.simplify(CMed)  # → q + 20 + 100/q
```

`.subs()`

¿Qué hace? Sustituye variables por valores o expresiones

Sintaxis:

```
resultado = expresion.subs(variable, valor)
# O múltiples sustituciones:
resultado = expresion.subs({var1: val1, var2: val2})
```

Parámetros: - `variable`: Variable a sustituir - `valor`: Valor o expresión de reemplazo

Input: Variable y su valor
Output: Nueva expresión con la sustitución

Casos de uso: - Evaluar en punto: `C.subs(q, 10)` → C(10) - Aplicar constante: `C.subs(F, 100)` → Reemplaza F por 100 - Múltiples: `B.subs({q: 5, F: 100})`

¿Cuándo usar? Para evaluar funciones o aplicar valores conocidos

Ejemplo:

```
C = q**2 + 20*q + F
# Evaluar en q=5
C.subs(q, 5)  # → 25 + 100 + F = 125 + F

# Aplicar F=100
C_final = C.subs(F, 100)  # → q² + 20q + 100
```

`sp.lambdify()`

¿Qué hace? Convierte expresión simbólica en función numérica de Python

Sintaxis:

```
funcion_numerica = sp.lambdify(variable, expresion_simbolica)
```

Parámetros: - `variable`: Variable independiente (ej: q, x) - `expresion_simbolica`: Expresión de SymPy a convertir

Input: Variable y expresión simbólica
Output: Función Python que acepta arrays de NumPy

Casos de uso: - Preparar para graficar: `C_num = sp.lambdify(q, C)` - Evaluación masiva: `valores = C_num(array_de_q)` - Optimización numérica

¿Cuándo usar? SIEMPRE antes de graficar o hacer cálculos numéricos masivos

¿Por qué usar? Las funciones numéricas son 1000x más rápidas que las simbólicas

Ejemplo:

```
C = q**2 + 20*q + 100

# Convertir a función numérica
C_num = sp.lambdify(q, C)

# Evaluar en muchos puntos
q_valores = np.linspace(0, 10, 100)
C_valores = C_num(q_valores)  # ← Array de 100 valores
```

NUMPY - Computación Numérica

`np.linspace()`

¿Qué hace? Crea un array de números equiespaciados

Sintaxis:

`array = np.linspace(inicio, fin, cantidad_puntos)`

Parámetros: - `inicio`: Primer valor del rango - `fin`: Último valor del rango (incluido) - `cantidad_puntos`: Número de valores a generar

Input: Rango y cantidad de puntos
Output: Array de NumPy con valores equiespaciados

Casos de uso: - Crear dominio para gráficos: `q_grid = np.linspace(0.1, 15, 300)` - Generar datos de prueba - Simular rangos continuos

¿Cuándo usar? Para crear el eje X antes de graficar

TIP: Empieza desde 0.1 (no 0) si vas a dividir por la variable

Ejemplo:

```
# 300 puntos entre 0.1 y 15
q_grid = np.linspace(0.1, 15, 300)
# [0.1, 0.15, 0.2, ..., 14.95, 15.0]

# Luego evaluar función
C_valores = C_num(q_grid)  # 300 valores de C(q)
```

`np.array()`

¿Qué hace? Convierte lista o tupla en array de NumPy

Sintaxis:

`array = np.array(lista_o_tupla)`

Input: Lista, tupla o estructura iterable
Output: Array de NumPy

Casos de uso: - Datos manuales: `costos = np.array([100, 120, 140])` - Convertir resultados: `soluciones = np.array(lista_soluciones)`

¿Cuándo usar? Cuando tienes datos en listas y necesitas operaciones vectorizadas

Ejemplo:

```
# Crear array desde lista
cantidades = np.array([1, 2, 3, 4, 5])
precios = np.array([10, 9.5, 9, 8.5, 8])

# Operaciones vectorizadas
ingresos = cantidades * precios  # [10, 19, 27, 34, 40]
```

MATPLOTLIB - Visualización

`plt.subplots()`

¿Qué hace? Crea figura y ejes para gráficos

Sintaxis:

`fig, ax = plt.subplots(filas, columnas, figsize=(ancho, alto))`

Parámetros: - `filas, columnas` (opcional): Layout de subgráficos - `figsize`: Tupla (ancho, alto) en pulgadas

Input: Dimensiones del gráfico
Output: Objeto Figure y Axes

Casos de uso: - Gráfico simple: `fig, ax = plt.subplots(figsize=(10, 6))` - Múltiples gráficos: `fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))`

¿Cuándo usar? Al inicio de cada visualización

¿Por qué? Permite control total sobre el gráfico (método recomendado)

Ejemplo:

```
# Crear figura de 10x6 pulgadas
fig, ax = plt.subplots(figsize=(10, 6))

# Ahora usar ax para graficar
ax.plot(x, y)
ax.set_title("Mi Gráfico")
```

`ax.plot()`

¿Qué hace? Dibuja líneas o puntos conectados

Sintaxis:

`ax.plot(x, y, label='etiqueta', linewidth=grosor, color='color')`

Parámetros: - `x`: Array de valores eje X - `y`: Array de valores eje Y - `label`: Texto para la leyenda - `linewidth`: Grosor de línea (default: 1.5) - `color`: Color ('blue', 'red', 'green', '#FF5733', etc.)

Input: Coordenadas X e Y
Output: Objeto Line2D (se dibuja automáticamente)

Casos de uso: - Graficar función: `ax.plot(q_grid, C_vals, label='C(q)')` - Múltiples líneas: Llamar `plot()` varias veces - Curvas: `ax.plot(x, y, linestyle='--')`

¿Cuándo usar? Para visualizar funciones continuas

Ejemplo:

```
ax.plot(q_grid, C_vals, label='Costo Total', linewidth=2, color='blue')
ax.plot(q_grid, I_vals, label='Ingreso Total', linewidth=2, color='green')
```

`ax.scatter()`

¿Qué hace? Dibuja puntos individuales

Sintaxis:

```
ax.scatter(x, y, c='color', s=tamaño, marker='forma', zorder=nivel)
```

Parámetros: - `x, y`: Coordenadas (pueden ser listas o valores únicos) - `c`: Color del punto - `s`: Tamaño del punto (default: 20) - `marker`: Forma ('o', '*','^', 's', etc.) - `zorder`: Orden de dibujo (mayor = encima)

Input: Coordenadas de puntos
Output: Colección de puntos

Casos de uso: - Marcar óptimos: `ax.scatter([q_opt], [B_opt], c='red', s=150, marker='*')` - Datos discretos: `ax.scatter(cantidades, precios)` - Resaltar puntos clave

¿Cuándo usar? Para marcar puntos importantes (máximos, mínimos, equilibrios)

TIP: Usar `zorder=5` para que aparezca sobre las líneas

Ejemplo:

```
# Marcar punto óptimo con estrella grande roja
ax.scatter([q_opt], [B_opt], c='red', s=150, marker='*',
           zorder=5, label=f'Máximo: q*={q_opt}')
```

`ax.axhline() / ax.axvline()`

¿Qué hacen? Dibujan líneas horizontales/verticales de referencia

Sintaxis:

```
ax.axhline(y, color='color', linewidth=grosor, linestyle='estilo', alpha=transparencia)
ax.axvline(x, color='color', linewidth=grosor, linestyle='estilo', alpha=transparencia)
```

Parámetros: - `y / x`: Posición de la línea - `linestyle`: '–' (guiones), ':' (puntos), '-' (sólida) - `alpha`: Transparencia (0=invisible, 1=opaco)

Casos de uso: - Eje cero: `ax.axhline(0, color='black', linewidth=0.5)` - Marcar óptimo: `ax.axvline(q_opt, linestyle='--', alpha=0.5)` - Niveles de referencia

¿Cuándo usar? Para mostrar valores clave o ejes de referencia

Ejemplo:

```
# Línea horizontal en el beneficio máximo
ax.axhline(B_opt, color='green', linestyle='--', alpha=0.5)

# Línea vertical en la cantidad óptima
ax.axvline(q_opt, color='red', linestyle='--', alpha=0.5)
```

`ax.grid()`

¿Qué hace? Activa/desactiva la grilla del gráfico

Sintaxis:

`ax.grid(True, alpha=transparencia, linestyle='estilo', linewidth=grosor)`

Parámetros: - `True/False`: Activar/desactivar grilla - `alpha`: Transparencia (0.3 es común) - `linestyle`: Estilo de línea - `linewidth`: Grosor de líneas

Casos de uso: - Grilla estándar: `ax.grid(True, alpha=0.3)` - Sin grilla: `ax.grid(False)`

¿Cuándo usar? SIEMPRE en gráficos económicos (facilita lectura de valores)

Ejemplo:

```
ax.grid(True, alpha=0.3, linestyle=':', linewidth=0.5)
```

Configuración de Ejes y Títulos

`ax.set_xlabel()` / `ax.set_ylabel()`

```
ax.set_xlabel('Cantidad (q)', fontsize=11)
ax.set_ylabel('Costo ($)', fontsize=11)
```

`ax.set_title()`

```
ax.set_title('Funciones de Costo', fontsize=14, fontweight='bold')
```

`ax.set_xlim()` / `ax.set_ylim()`

```
ax.set_xlim(0, 15)  # Rango eje X
ax.set_ylim(0, 250) # Rango eje Y
```

`ax.legend()`

```
ax.legend(fontsize=10, loc='upper right')
# loc: 'upper/lower/center' + 'left/right/center'
```

FLUJO COMPLETO TÍPICO

```
# 1. IMPORTAR
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# 2. DEFINIR VARIABLES SIMBÓLICAS
q = sp.symbols('q', positive=True)
F = sp.symbols('F')
```

```python
# 3. FUNCIÓN MARGINAL (del problema)
CMg = 20 + 2*q

# 4. INTEGRAR
C = sp.integrate(CMg, q) + F

# 5. CONSTANTE
F_sol = sp.solve(sp.Eq(C.subs(q, 0), 100), F)[0]
C = C.subs(F, F_sol)

# 6. FUNCIONES DERIVADAS
CMed = sp.simplify(C / q)

# 7. PUNTOS CRÍTICOS
dCMed = sp.diff(CMed, q)
q_min = sp.solve(sp.Eq(dCMed, 0), q)[0]
CMed_min = CMed.subs(q, q_min)

# 8. VISUALIZAR
q_grid = np.linspace(0.1, 15, 300)
CMed_num = sp.lambdify(q, CMed)
CMed_vals = CMed_num(q_grid)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(q_grid, CMed_vals, label='$C_{med}(q)$', linewidth=2)
ax.scatter([q_min], [CMed_min], c='red', s=150, marker='*')
ax.axhline(0, color='black', linewidth=0.5)
ax.grid(True, alpha=0.3)
ax.set_xlabel('Cantidad (q)')
ax.set_ylabel('Costo Medio ($)')
ax.set_title('Costo Medio y su Punto Mínimo')
ax.legend()
plt.tight_layout()
plt.show()
```

ERRORES COMUNES

Error 1: Olvidar [0] en solve()

```python
#  MAL
q_opt = sp.solve(sp.Eq(IMg, CMg), q)  # Retorna lista

#  BIEN
q_opt = sp.solve(sp.Eq(IMg, CMg), q)[0]  # Extrae primer valor
```

Error 2: No agregar constante al integrar

```python
#  MAL
C = sp.integrate(CMg, q)  # Falta +F

#  BIEN
C = sp.integrate(CMg, q) + F
```

Error 3: División por cero en gráficos

```
#  MAL
q_grid = np.linspace(0, 100, 300)  # Empieza en 0

#  BIEN
q_grid = np.linspace(0.1, 100, 300)  # Empieza en 0.1
```

Error 4: No convertir a función numérica

```
#  MAL (muy lento)
C_vals = [C.subs(q, val) for val in q_grid]

#  BIEN (1000x más rápido)
C_num = sp.lambdify(q, C)
C_vals = C_num(q_grid)
```

TABLA RESUMEN

Operación	Librería	Función	Uso Típico
Crear variable	SymPy	`sp.symbols()`	`q = sp.symbols('q', positive=True)`
Integrar	SymPy	`sp.integrate()`	`C = sp.integrate(CMg, q) + F`
Derivar	SymPy	`sp.diff()`	`CMg = sp.diff(C, q)`
Resolver ecuación	SymPy	`sp.solve()`	`q_opt = sp.solve(sp.Eq(IMg, CMg), q)[0]`
Simplificar	SymPy	`sp.simplify()`	`CMed = sp.simplify(C / q)`
Sustituir	SymPy	`.subs()`	`C.subs(F, 100)`
A función numérica	SymPy	`sp.lambdify()`	`C_num = sp.lambdify(q, C)`
Crear dominio	NumPy	`np.linspace()`	`q_grid = np.linspace(0.1, 15, 300)`
Crear figura	Matplotlib	`plt.subplots()`	`fig, ax = plt.subplots(figsize=(10,6))`
Graficar línea	Matplotlib	`ax.plot()`	`ax.plot(x, y, label='Función')`
Marcar punto	Matplotlib	`ax.scatter()`	`ax.scatter([x0], [y0], c='red')`
Línea referencia	Matplotlib	`ax.axhline/vline()`	`ax.axhline(0, color='black')`

TIPS FINALES

1. **Siempre simplifica** después de dividir o restar expresiones
2. **Verifica tus integrales** derivando el resultado
3. **Usa [0]** después de `sp.solve()` para extraer el valor

4. **Empieza en 0.1** (no 0) cuando grafiques funciones con q en el denominador
5. **Convierte a numérico** con `lambdify()` antes de graficar
6. **Agrega grillas** con `ax.grid(True, alpha=0.3)` para facilitar lectura
7. **Marca puntos clave** con `scatter()` y líneas de referencia
8. **Interpreta económicamente** cada resultado matemático

Este cheatsheet es tu referencia rápida para el examen. Imprímelo o tenlo en una pestaña separada mientras practicas.