Material suplementario

Dispersión de aditivos en alimentos: un modelo de convección - difusión resuelto por diferencias finitas

Dima, Jimena Berndette; Ferrari, Mariano; Fiedorowicz Kowal, Martina; Mandelman, Ivan.

Se presentan en el material suplementario las funciones programadas en Python para resolver el modelo de convección-difusión propuesto en tres geometrías regulares.

Se presentan además las funciones utilizadas para graficar los perfiles de concentración y la concentración total en cada caso.

Sistemas adimencionales

Lamina plana

$$\begin{aligned} u_s &= u_{rr} && \text{para } r \in (0,1);\ s > 0 \\ u_r &= 0 && \text{para } r = 0;\ s > 0 \\ u_r &= \alpha(1-u) && \text{para } r = 1;\ s > 0 \\ u &= 0 && \text{para } r \in (0,1);\ s = 0 \end{aligned}$$

Concentración total en variable adimensional:

$$u_T(s) = \int_0^1 u(r,s)\, dr.$$

Cilindro alargado

$$\begin{aligned} u_s &= \frac{1}{r}u_r + u_{rr} && \text{para } r \in (0,1);\ s > 0 \\ u_r &= \alpha(1-u) && \text{para } r = 1;\ s > 0 \\ u &= 0 && \text{para } r \in (0,1);\ s = 0 \end{aligned}$$

Concentración total en variable adimensional:

$$u_T(s) = 2\int_0^1 u(r,s) r\, dr.$$

Esfera

$$\begin{aligned} u_s &= \frac{2}{r}u_r + u_{rr} && \text{para } r \in (0,1);\ s > 0 \\ u_r &= \alpha(1-u) && \text{para } r = 1;\ s > 0 \\ u &= 0 && \text{para } r \in (0,1);\ s = 0 \end{aligned}$$

Concentración total en variable adimensional:

$$u_T(s) = 3\int_0^1 u(r,s) r^2\, dr.$$

Funciones que resuelven los sistemas anteriores aproximando u y u_T

```
import numpy as np
import matplotlib.pyplot as plt


# Lamina plana
def calculate_u_cart(alpha, s_max, N_r, N_s):
    """
    Resuelve el sistema correspondiente a la lámina plana por un método de diferencias finitas.
```

```
    Args:
      alpha: parámetro adimensional del modelo
      s_max: valor máximo adoptado para la variable s
      N_r: número de intervalos en la discretización equiespaciada de la variable r
      N_s: número de intervalos en la discretización equiespaciada de la variable s

    Return: una matriz N_r por N_s que aproxima la solución en la discretización utilizada
    """
    r_values = np.linspace(0, 1, N_r) # Discretización de r
    r_values = r_values[1:]
    N_r = N_r - 1
    s_values = np.linspace(0, s_max, N_s) # Discretización de s
    dr = r_values[1] - r_values[0] # Paso de r
    ds = s_values[1] - s_values[0] # Paso de s
    delta = ds / dr ** 2
    A = np.eye(N_r)

    for i in range(1, N_r - 1):
        A[i, i - 1 : i + 2] = [-delta, 1 + 2 * delta, -delta]

    u_old = np.zeros(N_r)
    u_old[-1] = dr * alpha / (1 + dr * alpha)
    u = [u_old.copy()]

    for _ in range(1, N_s):
        u_new = np.linalg.solve(A, u_old)
        u_new[0] = u_new[1]  # Condición de frontera en r = 0
        u_new[-1] = (u_new[-2] + dr * alpha)/(1+dr*alpha)  # Condición de frontera en r = 1
        u.append(u_new.copy())
        u_old = u_new

    return np.array(u).T


# Concentración total en Lamina plana
def calculate_u_cart_T(u):
    """
    Calcula la concentración total u_T para el caso de la lámina plana

    Args:
      u: Matriz de soluciones u(r, s), donde la primera dimensión representa los valores de la variable r.

    Return: Array de valores que aproxian u_T(s) en la discretización utilizada.
    """
    # Asumimos que la primera dimensión de u representa los valores de r
    N_r = u.shape[0]
    r_values = np.linspace(0, 1, N_r)  # Generar r_values basado en la dimensión de u
    # Realizar la integración sobre r para obtener u_T
    return np.trapezoid(u, r_values, axis=0)


# Cilindro alargado
def calculate_u_cyl(alpha, s_max, N_r, N_s):
    """
    Resuelve el sistema correspondiente al cilindro alargado por un método de diferencias finitas.

    Args:
      alpha: parámetro adimensional del modelo
      s_max: valor máximo adoptado para la variable s
      N_r: número de intervalos en la discretización equiespaciada de la variable r
      N_s: número de intervalos en la discretización equiespaciada de la variable s

    Return: una matriz N_r por N_s que aproxima la solución en la discretización utilizada
    """
    r_values = np.linspace(0, 1, N_r)  # Discretización de r
    r_values = r_values[1:]  # Excluir el primer valor para evitar singularidad en r=0
    N_r = N_r - 1  # Ajustar el número de puntos en r
    s_values = np.linspace(0, s_max, N_s)  # Discretización de s
    dr = r_values[1] - r_values[0]  # Paso en r
    ds = s_values[1] - s_values[0]  # Paso en s
    delta = ds / dr ** 2  # Parámetro delta
    gamma = ds / (2 * dr)  # Parámetro gamma
    A = np.eye(N_r)  # Matriz identidad de tamaño N_r

    # Llenar la matriz A con los coeficientes adecuados
    for i in range(1, N_r - 1):
        A[i, i - 1: i + 2] = [gamma / r_values[i] - delta, 1 + 2 * delta, -gamma / r_values[i] - delta]

    # Condición de frontera en r = 0
    A[0, [0, 1, 2]] = [1 + 2 * gamma / r_values[0] - delta, 2 * (delta - gamma / r_values[0]), -delta]

    u_old = np.zeros(N_r)  # Inicializar u_old con ceros
    u_old[-1] = dr * alpha / (1 - dr * alpha)  # Condición de frontera en r = 1
```

```
    u = [u_old.copy()]  # Lista para almacenar las soluciones en cada paso de s

    # Iterar sobre los pasos de s
    for _ in range(1, N_s):
        u_new = np.linalg.solve(A, u_old)  # Resolver el sistema de ecuaciones
        u_new[-1] = (u_new[-2] + dr * alpha) / (1 + dr * alpha)  # Actualizar la condición de frontera en r = 1
        u.append(u_new.copy())  # Almacenar la nueva solución
        u_old = u_new  # Actualizar u_old para el siguiente paso

    return np.array(u).T  # Devolver la matriz de soluciones transpuesta


# Concentración total en Cilindro alargado
def calculate_u_cyl_T(u):
    """
    Calcula la concentración total u_T para el caso del cilindro alargado

    Args:
      u: Matriz de soluciones u(r, s), donde la primera dimensión representa los valores de la variable r.

    Return: Array de valores que aproxian u_T(s) en la discretización utilizada.
    """
    N_r = u.shape[0]  # Número de puntos en r
    r_values = np.linspace(0, 1, N_r + 1)[1:]  # Generar valores de r, excluyendo el primer valor
    r_v = r_values[:, np.newaxis]  # Convertir r_values a una matriz columna
    u_T = 2 * np.trapezoid(u[:-1, :] * r_v[:-1], r_values[:-1], axis=0)  # Calcular u_T
    return u_T


#Esfera
def calculate_u_sph(alpha,s_max,N_r,N_s):
    """
    Resuelve el sistema correspondiente a la esfera por un método de diferencias finitas.

    Args:
      alpha: parámetro adimensional del modelo
      s_max: valor máximo adoptado para la variable s
      N_r: número de intervalos en la discretización equiespaciada de la variable r
      N_s: número de intervalos en la discretización equiespaciada de la variable s

    Return: una matriz N_r por N_s que aproxima la solución en la discretización utilizada
    """
    r_values = np.linspace(0, 1, N_r)
    r_values = r_values[1:]
    N_r=N_r-1
    s_values = np.linspace(0, s_max, N_s)
    dr = r_values[1] - r_values[0]
    ds = s_values[1] - s_values[0]
    delta = ds / dr ** 2
    gamma = ds / dr
    A = np.eye(N_r)
    for i in range(1, N_r - 1):
        A[i, i - 1: i + 2] = [gamma/r_values[i]-delta, 1 + 2 * delta, -gamma/r_values[i]-delta]

    A[0,[0,1,2]] = [1+gamma/r_values[1] - delta, 2*delta-gamma/r_values[1], -delta]

    u_old = np.zeros(N_r)
    u_old[-1] = dr * alpha / (1 - dr * alpha)
    u = [u_old.copy()]

    for _ in range(1, N_s):
        u_new = np.linalg.solve(A, u_old)
        u_new[-1] = (u_new[-2] + dr * alpha)/(1+dr*alpha)
        u.append(u_new.copy())
        u_old = u_new

    return np.array(u).T


# Concentración total en Esfera
def calculate_u_sph_T(u):
    """
    Calcula la concentración total u_T para el caso de la esfera

    Args:
      u: Matriz de soluciones u(r, s), donde la primera dimensión representa los valores de la variable r.

    Return: Array de valores que aproxian u_T(s) en la discretización utilizada.
    """
    N_r = u.shape[0]  # Número de puntos en r
    r_values = np.linspace(0, 1, N_r + 1)[1:]  # Generar valores de r, excluyendo el primer valor
    r_squared = r_values ** 2  # Calcular r al cuadrado
    r_squared = r_squared[:, np.newaxis]  # Convertir r_squared a una matriz columna
```

```
    u_T = 3 * np.trapezoid(u * r_squared, r_values, axis=0)  # Calcular u_T
    return u_T
```

Gráficos de las funciones u y u_T

Lámina Plana

```
# Discretización
s_max = 100
N_r = 500 # Número de puntos en la discretización de r
N_s = 2000 # Número de puntos en la discretización de s
r_values = np.linspace(0, 1, N_r)  # Discretización de r
r_values = r_values[1:]  # Excluir el primer valor para evitar singularidad en r=0
s_values = np.linspace(0, s_max, N_s) # Discretización de s
```

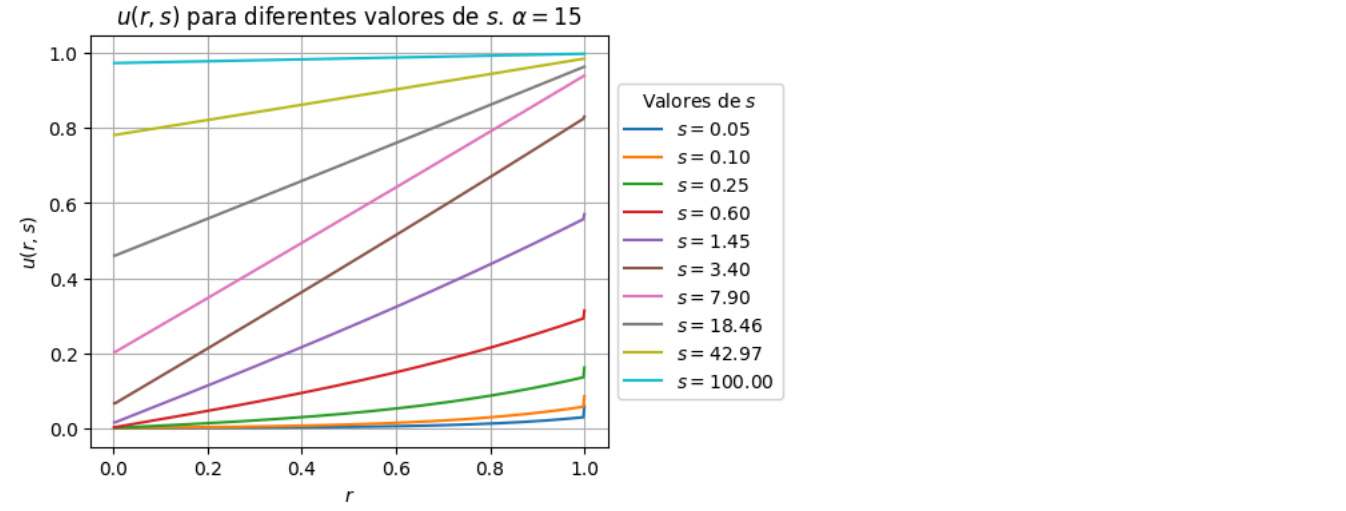
Perfiles de la variación de la concentración en función de r

Gráficos de $u(r, s)$ para valores fijos de s:

$\alpha = 15$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_cart(15, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 10

# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=15$')
plt.grid(True)

# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

```
<matplotlib.legend.Legend at 0x7dc5d0aa8ed0>
```

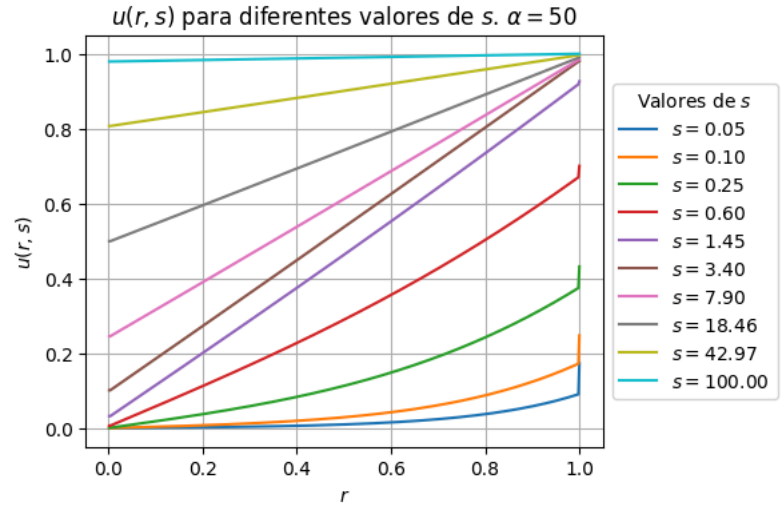


$\alpha = 50$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_cart(50, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 10
```

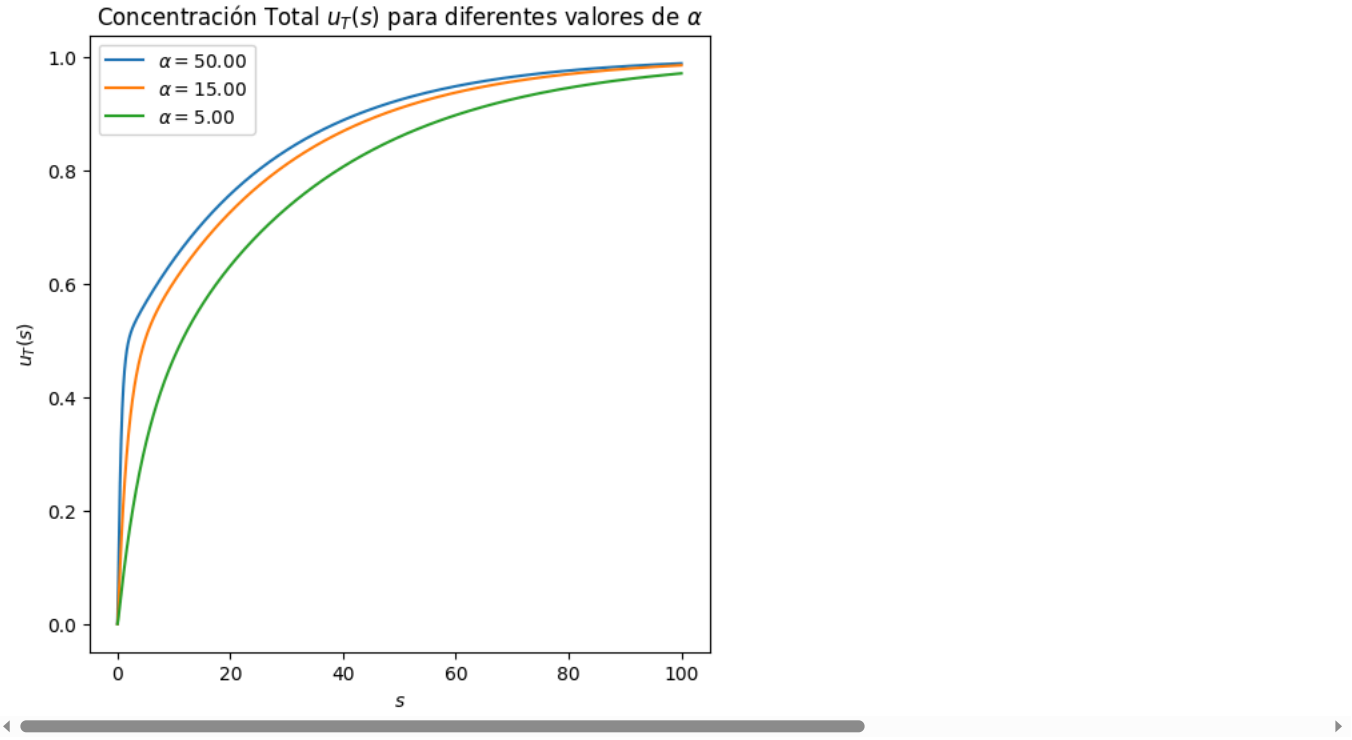
```
# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=50$')
plt.grid(True)

# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

```
<matplotlib.legend.Legend at 0x7dc59dfabc90>
```



Variación de la concentración total en función de la variable s

```
plt.figure(figsize=(6, 6))

alpha_values = [50, 15,5]  # Valores específicos de alpha que deseas graficar

for i, alpha in enumerate(alpha_values):
    u = calculate_u_cart(alpha, s_max, N_r, N_s)
    u_T = calculate_u_cart_T(u)  # Calcular u_T usando la función definida

    # Graficar u_T para cada valor de alpha
    plt.plot(s_values, u_T, label=f'$\\alpha = {alpha:.2f}$')

# Personalizar la gráfica
plt.xlabel('$s$')
plt.ylabel('$u_T(s)$')
plt.title('Concentración Total $u_T(s)$ para diferentes valores de $\\alpha$')
plt.legend()

plt.show()
```



Cilindro alargado

```
# Discretización
s_max = 100
N_r = 500 # Número de puntos en la discretización de r
N_s = 500 # Número de puntos en la discretización de s
r_values = np.linspace(0, 1, N_r)  # Discretización de r
r_values = r_values[1:]  # Excluir el primer valor para evitar singularidad en r=0
s_values = np.linspace(0, s_max, N_s) # Discretización de s
```

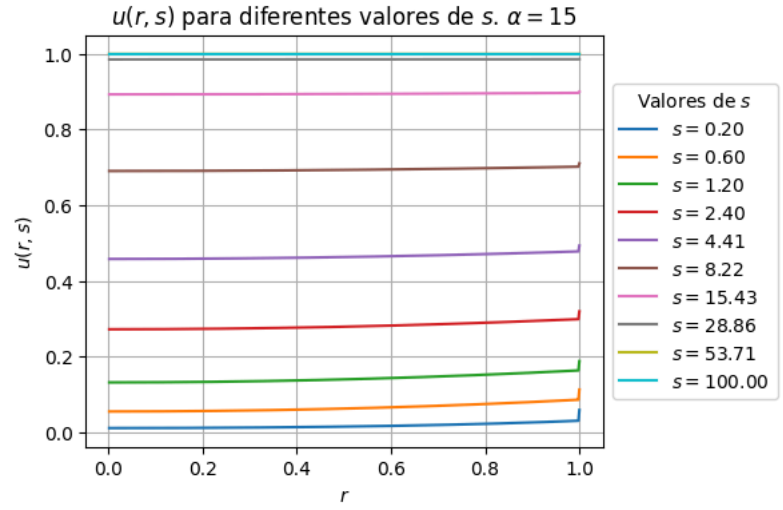
Perfiles de la variación de la concentración en función de r

Gráficos de $u(r, s)$ para valores fijos de s:

$\alpha = 15$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_cyl(15, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 11

# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=15$')
plt.grid(True)

# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

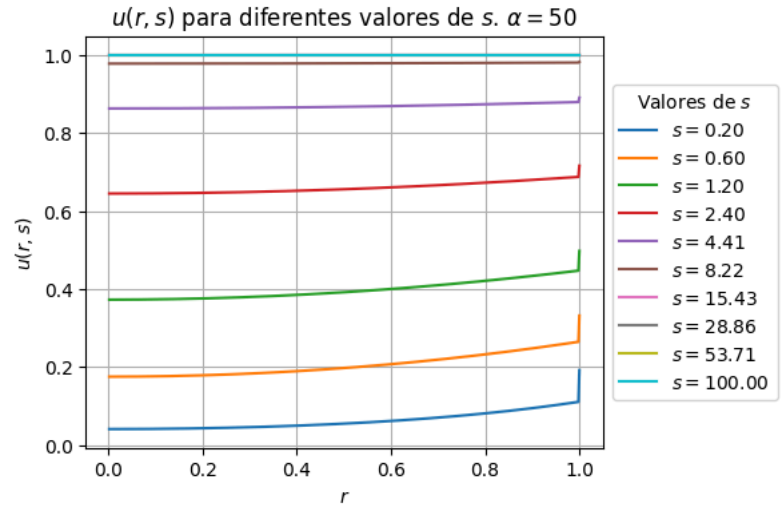
```
<matplotlib.legend.Legend at 0x7dc59defd210>
```



$\alpha = 50$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_cyl(50, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 11

# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=50$')
plt.grid(True)

# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

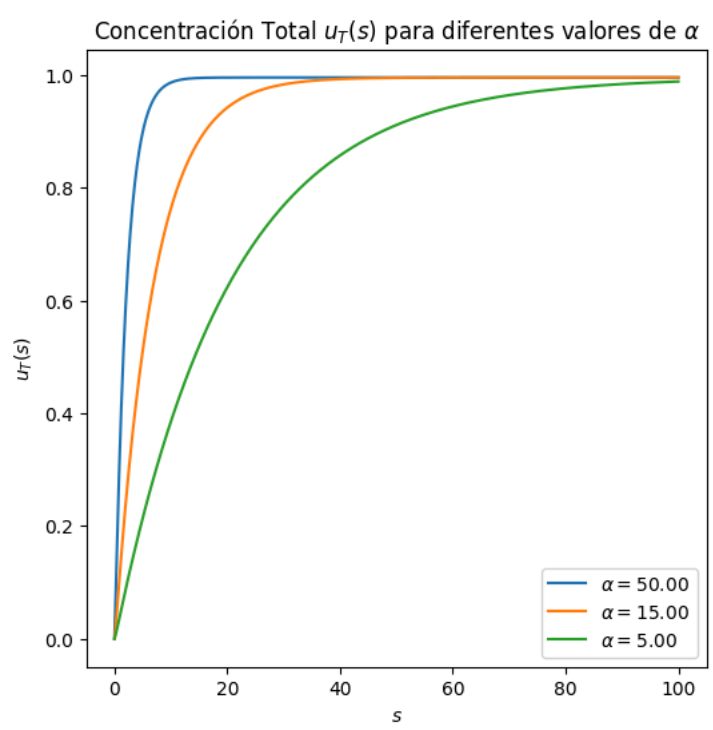
```
<matplotlib.legend.Legend at 0x7dc5b465ee10>
```



Variación de la concentración total en función de la variable s

```
plt.figure(figsize=(6, 6))

alpha_values = [50, 15,5]  # Valores específicos de alpha que deseas graficar

for i, alpha in enumerate(alpha_values):
    u = calculate_u_cyl(alpha, s_max, N_r, N_s)
    u_T = calculate_u_cyl_T(u)  # Calcular u_T usando la función definida
```

```
    # Graficar u_T para cada valor de alpha
    plt.plot(s_values, u_T, label=f'$\\alpha = {alpha:.2f}$')

# Personalizar la gráfica
plt.xlabel('$s$')
plt.ylabel('$u_T(s)$')
plt.title('Concentración Total $u_T(s)$ para diferentes valores de $\\alpha$')
plt.legend()

plt.show()
```



Esfera

```
# Discretización
s_max = 100
N_r = 500 # Número de puntos en la discretización de r
N_s = 500 # Número de puntos en la discretización de s
r_values = np.linspace(0, 1, N_r)  # Discretización de r
r_values = r_values[1:]  # Excluir el primer valor para evitar singularidad en r=0
s_values = np.linspace(0, s_max, N_s) # Discretización de s
```

Perfiles de la variación de la concentración en función de r

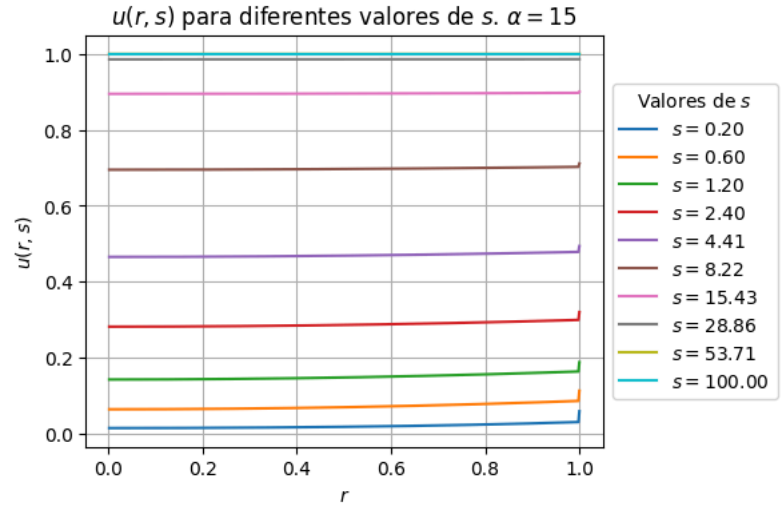
Gráficos de $u(r, s)$ para valores fijos de s:

$\alpha = 15$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_sph(15, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 11

# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=15$')
plt.grid(True)
```

```
# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

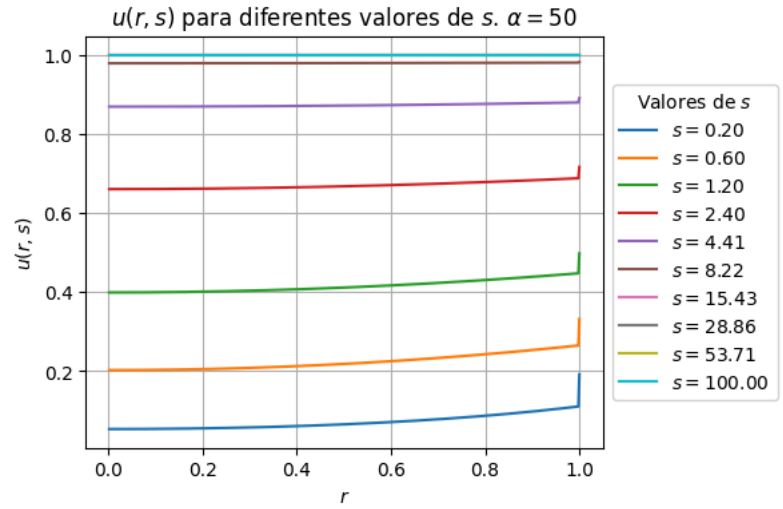
```
<matplotlib.legend.Legend at 0x7dc5b43ffe10>
```



$\alpha = 50$

```
# Calcula u para el valor de alpha y la discretización seleccionada
u = calculate_u_sph(50, s_max, N_r, N_s)

# Elegir el número de valores temporales (s) a graficar
num_s_values = 11

# Generar indices logaritmicamente espaciados
s_indices = np.unique(np.geomspace(1, N_s - 1, num_s_values, dtype=int))

plt.figure(figsize=(5, 4))
for i in s_indices:
    plt.plot(r_values, u[:, i], label=f'$s = {s_values[i]:.2f}$')

# Personalizar la gráfica
plt.xlabel('$r$')
plt.ylabel('$u(r, s)$')
plt.title('$u(r,s)$ para diferentes valores de $s$. $\\alpha=50$')
plt.grid(True)

# Etiquetas fuera del gráfico
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Valores de $s$")
```

```
<matplotlib.legend.Legend at 0x7dc5b46e6e90>
```



Variación de la concentración total en función de la variable s

```
plt.figure(figsize=(6, 6))

alpha_values = [50, 15,5]  # Valores específicos de alpha

for i, alpha in enumerate(alpha_values):
```

```
    u = calculate_u_sph(alpha, s_max, N_r, N_s)
    u_T = calculate_u_sph_T(u)

    # Graficar u_T para cada valor de alpha
    plt.plot(s_values, u_T, label=f'$\\alpha = {alpha:.2f}$')

# Personalizar la gráfica
plt.xlabel('$s$')
plt.ylabel('$u_T(s)$')
plt.title('Concentración Total $u_T(s)$ para diferentes valores de $\\alpha$')
plt.legend()
```