Resolución de Ecuaciones Diferenciales Parciales mediante Series de Fourier en Julia

1 Introducción

En esta memoria se describe la resolución de una ecuación diferencial parcial (EDP) utilizando series de Fourier, tanto bajo condiciones de frontera de Dirichlet como de Neumann. Se presentará una explicación teórica, junto con el código implementado en Julia y la interpretación de los resultados obtenidos.

2 Fundamentos Teóricos

Las ecuaciones diferenciales parciales (EDP) son ecuaciones que involucran derivadas parciales de funciones de varias variables. En este caso, consideramos la EDP del tipo:

$$u_t = bu_{xx},$$

donde $u(x,t)$ es la función desconocida que depende de la posición x y del tiempo t, y b es una constante.

Para resolver esta EDP, utilizamos la técnica de separación de variables, suponiendo que la solución puede escribirse como el producto de dos funciones independientes:

$$u(x,t) = X(x)T(t).$$

Al sustituir esta forma en la EDP y separando las variables, obtenemos dos ecuaciones ordinarias:

$$\frac{T'(t)}{bT(t)} = \frac{X''(x)}{X(x)} = -\lambda,$$

donde λ es una constante de separación. Esto lleva a las siguientes ecuaciones:

$$T'(t) + b\lambda T(t) = 0,$$
$$X''(x) + \lambda X(x) = 0.$$

Estas ecuaciones se resuelven dependiendo de las condiciones de frontera.

2.1 Condiciones de Frontera de Dirichlet

Para las condiciones de Dirichlet, $u(0,t) = u(L,t) = 0$, las soluciones para $X(x)$ son:

$$X_n(x) = \sin\left(\frac{n\pi x}{L}\right),$$

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2,$$

donde n es un entero positivo.

2.2 Condiciones de Frontera de Neumann

Para las condiciones de Neumann, $u_x(0,t) = u_x(L,t) = 0$, las soluciones para $X(x)$ son:

$$X_n(x) = \cos\left(\frac{(2n-1)\pi x}{2L}\right),$$

$$\lambda_n = \left(\frac{(2n-1)\pi}{2L}\right)^2,$$

donde n es un entero positivo.

3 Descripción del Código en Julia

El siguiente código implementa la solución de la EDP usando las series de Fourier, permitiendo tanto condiciones de frontera de Dirichlet como de Neumann.

3.1 Función `Xn_expression`

Esta función determina la expresión $X_n(x)$ en función del tipo de condición de frontera (Dirichlet o Neumann).

```
using SymPy

function Xn_expression(x, n, L, lambda_val,
   boundary_type)
    if boundary_type == "dirichlet"
        if simplify(lambda_val - ((n *   ) / L) ^ 2)
           == 0
            return sin(n *    * x / L)
        else
            return nothing
        end
    elseif boundary_type == "neumann"
```

```
        if simplify(lambda_val - (((2 * n - 1) *   ) /
            (2 * L)) ^ 2) == 0
            return cos((2 * n - 1) *    * x / (2 * L))
        else
            return nothing
        end
    else
        return nothing
    end
end
```

3.2 Función `Tm_expression`

Esta función calcula la parte temporal de la solución $T_m(t)$.

```
function Tm_expression(n, t, lambda_val, b)
    return exp(b * lambda_val * t)
end
```

3.3 Función `calculate_km`

Esta función calcula el coeficiente k_m utilizando la función $f(x)$.

```
function calculate_km(f, Xm, L, b)
    x = symbols("x")
    numerator = integrate(f * Xm, (x, 0, L))
    denominator = integrate(Xm ^ 2, (x, 0, L))
    km = numerator / denominator
    return km / b
end
```

3.4 Función `calculate_km1`

Esta función calcula el coeficiente k_{m1} utilizando la condición inicial $h(x)$.

```
function calculate_km1(h, Xm, L, b)
    x = symbols("x")
    numerator = integrate(h * Xm, (x, 0, L))
    denominator = integrate(Xm ^ 2, (x, 0, L))
    km1 = numerator / denominator
    return km1 / -b
end
```

3.5 Función `U_expression`

Esta función combina las expresiones espaciales y temporales para obtener la solución $U(x,t)$.

```
function U_expression(x, t, n, L, lambda_val, f, h, b,
    km, km1, boundary_type)
    Xn = Xn_expression(x, n, L, lambda_val,
       boundary_type)
    Tm = Tm_expression(n, t, lambda_val, b)
    if Xn !== nothing && Tm !== nothing
        return Xn * (km1 * Tm + km)
    else
        return nothing
    end
end
```

4 Ejemplo de Uso

```
# Ejemplo de uso:
@vars n x t
L = 1
lambda_val_dirichlet = ((n *   ) / L) ^ 2
lambda_val_neumann = (((2 * n - 1) *   ) / (2 * L)) ^
   2
b = -1

# Ejemplo de funci n f(x)
f = 0 # Puedes definir tu propia funci n f(x) aqu

# Ejemplo de condici n inicial h(x)
h = 1 # Puedes definir tu propia funci n h(x) aqu

# Boundary type
boundary_type = "neumann"  # Cambia a "dirichlet" para
    condiciones de frontera de Dirichlet

# Calculando km
if boundary_type == "dirichlet"
    Xm = sin(n *    * x / L)
    lambda_val = lambda_val_dirichlet
elseif boundary_type == "neumann"
    Xm = cos((2 * n - 1) *    * x / (2 * L))
    lambda_val = lambda_val_neumann
else
```

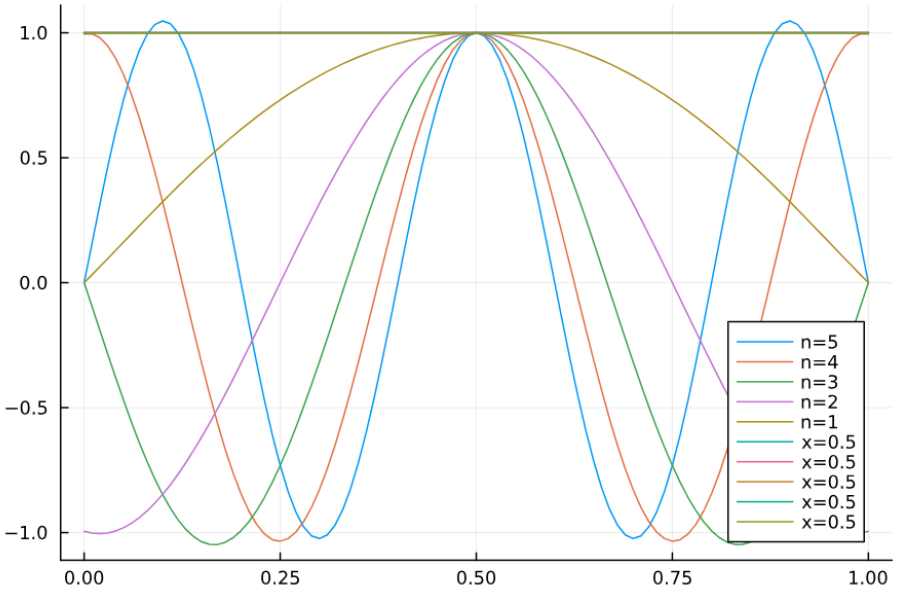
```
    error("Tipo de frontera no v lido")
end

km = calculate_km(f, Xm, L, b)
println("Valor de km:", km)

# Calculando km1
km1 = calculate_km1(h, Xm, L, b)
println("Valor de km1:", km1)

# Expresi n de U(x, t)
U = U_expression(x, t, n, L, lambda_val, f, h, b, km,
    km1, boundary_type)
println("U(x, t) =", U)
```

5 Gráfica de la Solución

Para graficar la solución, utilizamos el siguiente código:

```
using Plots
gr()

x_vals = 0:0.01:L
t_vals = 0:0.01:1

n_vals = 1:5

plot(x_vals, [U(x, 0.1, 5) for x in x_vals], label="n
    =5")
plot!(x_vals, [U(x, 0.1, 4) for x in x_vals], label="n
    =4")
plot!(x_vals, [U(x, 0.1, 3) for x in x_vals], label="n
    =3")
plot!(x_vals, [U(x, 0.1, 2) for x in x_vals], label="n
    =2")
plot!(x_vals, [U(x, 0.1, 1) for x in x_vals], label="n
    =1")

plot!(t_vals, [U(0.5, t, 5) for t in t_vals], label="x
    =0.5")
plot!(t_vals, [U(0.5, t, 4) for t in t_vals], label="x
    =0.5")
plot!(t_vals, [U(0.5, t, 3) for t in t_vals], label="x
    =0.5")
```

```
plot!(t_vals, [U(0.5, t, 2) for t in t_vals], label="x
    =0.5")
plot!(t_vals, [U(0.5, t, 1) for t in t_vals], label="x
    =0.5")

plot!(legend=:bottomright)
```



6 Interpretación de los Resultados

El gráfico muestra las soluciones $U(x,t)$ para diferentes valores de n en función de x y t. En la primera parte del gráfico, se observa la dependencia espacial para un tiempo fijo ($t = 0.1$). Las curvas representan las soluciones para $n = 1$ a $n = 5$, donde se puede ver cómo varían las funciones $X_n(x)$ según las condiciones de Neumann.

En la segunda parte del gráfico, se observa la evolución temporal de la solución en un punto fijo ($x = 0.5$). Cada curva representa la solución para diferentes valores de n, mostrando cómo las soluciones $T_m(t)$ evolucionan con el tiempo.