Tarea 14: Ecuaciones diferenciales parciales
Métodos Numéricos

Rafael Alejandro García Ramírez

1ro de diciembre de 2023

1. Introducción

Al igual que la resolución para ecuaciones diferenciales ordinarias, resulta de vital importancia la descripción de sistemas mediante una ecuación diferencial que represente en su totalidad (al menos de forma muy aproximada) el comportamiento. Anteriormente habíamos visto el caso donde solamente se tenía en cuenta una ecuación diferencial donde se tiene solamente una variable independiente, sin embargo es bastante común encontrar sistemas donde existan dos o más variables independientes.

De aquí que surge la necesidad de buscar soluciones que nos permitan describir y resolver este tipo de sistemas; por otro lado, la necesidad de buscar una solución numérica a estos sistemas surge de la dificultad que muchas veces presenta resolver analíticamente una ecuación diferencial parcial, lo cuál nos genera mucho mayor interés y necesidad la búsqueda de métodos para la resolución de estos sistemas.

2. Pseudocódigos

2.1. θ-Método

Algorithm 1 Método Theta para la Ecuación de Calor

procedure θ_METODO(N, iteraciones, θ, a, b, t, c, f)
    $h \leftarrow b - a/N$
    $\lambda \leftarrow ct/Nh^2$

    $A, B, C, temp1, temp2 \leftarrow$ generar_matriz(N)

    $A \leftarrow$ Matriz de calor
    $B \leftarrow$ Matriz Identidad

    for $i \leftarrow 0$ **to** $N - 1$ **do**
        $C[0][i] \leftarrow f(a + hi)$
    end for

    temp1 $= \theta \cdot \lambda \cdot A$
    temp1 $\leftarrow$ temp1 $+ B$

    temp2 $= (\theta - 1) \cdot \lambda \cdot A$
    temp2 $\leftarrow$ temp2 $+ B$

    for $j \leftarrow 1$ **to** $N - 1$ **do**
        $C[j] \leftarrow$ gauss_seidel($N, temp1,$ multiplicar_matriz_vector($N, temp2, C[j-1]$))
    end for
    Devolver C
end procedure

3. Resultados

Problema 1

Supongamos que tenemos una ecuación de la forma

$$\frac{\partial^n y(x)}{\partial x^n} = f\left(x, y, \frac{\partial y}{\partial x}, \frac{\partial^2 y}{\partial x^2}, \ldots, \frac{\partial^{n-1} y}{\partial x^{n-1}}\right)$$

Con las condiciones de frontera

$$y(x_0) = y_0 \ , \quad y(x_f) = y_f$$

Comenzaremos reduciendo el orden de la ecuación diferencial creando un sistema de ecuaciones de la forma

$$\frac{\partial y}{\partial x} = z \ , \quad \frac{\partial z}{\partial x} = f(x, y, z)$$

Este sistema de ecuaciones comporta una de las condiciones iniciales pero genera uno nuevo desconocido de la forma

$$y(x_0) = y_0 \ , \quad z(x_0) = \vartheta$$

Donde ϑ es una condición desconocida. Tomaremos esta condición inicial para una suposición inicial $\vartheta = \vartheta_0$; realizando la integración para la resolución con este ϑ_0 obtendremos un valor diferente a y_f que buscaremos en nuestra condición de frontera, por lo que nuestro problema ahora se reduce a encontrar la raíz de

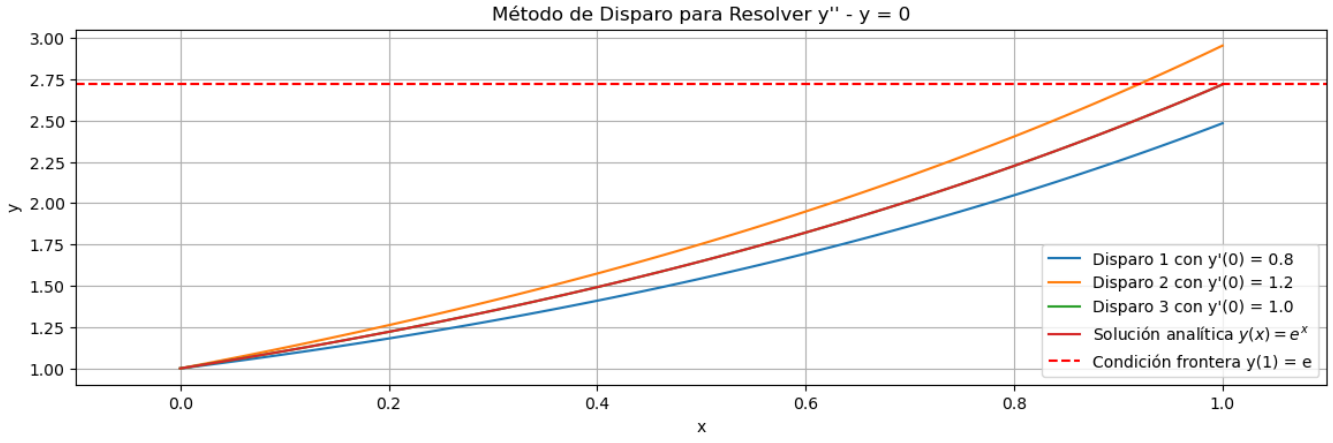
$$F(\vartheta) = y(x_f) - \vartheta_i$$

Para alguna suposición i-ésima de nuestra condición inicial. Así, si para nuestro ϑ_0 se genera una condición de frontera mayor a la original solo bastará con realizar al menos dos suposiciones más: una que nos llevará a un valor de frontera mayor, con el que podremos determinar así un valor específico con el que nuestro sistema cumpla con las condiciones de frontera originales. Es debido a esto se le llama "*Shooting Method*" pues se realiza un disparo de prueba para evaluar donde cae la solución y a partir de ahí realizar correcciones en los disparos hasta dar con el valor de frontera final.

Como ejemplo consideremos el problema

$$y'' - y = 0 \ , \quad y(0) = 1 \ , y'(1) = e$$

Cuya solución analítica es $y(x) = e^x$. Realizando tres disparos con un $\vartheta_0 = 0.8$ obtenemos la siguiente gráfica



Donde la solución para el tercer disparo queda empalmado con la solución analítica.

Problema 2

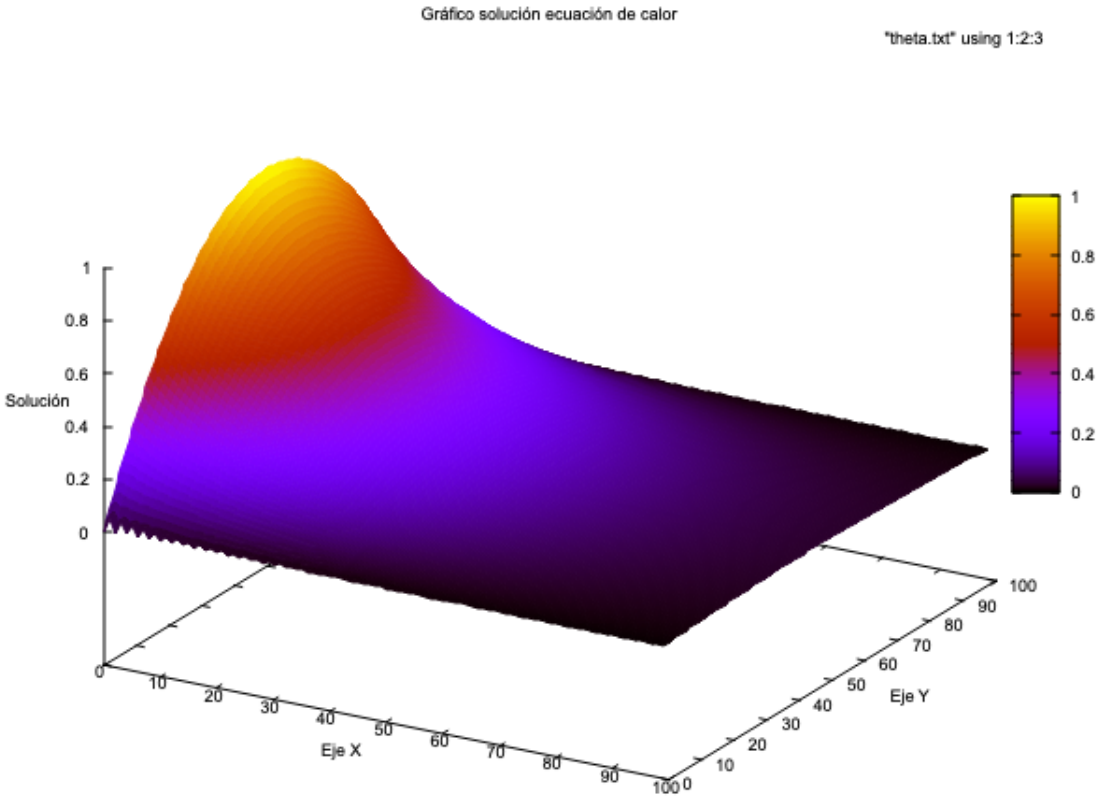
Para el siguiente problema se utilizó el siguiente código

```
double **theta_metodo_calor(int N, int iteraciones, double theta, double a, double b, double t, double
    c, double (*f)(double)){
    double h = (b-a)/N;
    double lambda = (c*(t/N))/(h*h);
    double **A = generar_matriz(N);
    double **B = generar_matriz(N);
    double **C = generar_matriz(N);
    double **temp1 = generar_matriz(N);
    double **temp2 = generar_matriz(N);
```

```
    // generamos la matriz de calor
    for (int i = 0; i < N; i++) {
        A[i][i] = 2.0;  // Diagonal principal
        B[i][i] = 1.0;  // Diagonal principal
        if (i < N - 1) {
            A[i][i + 1] = -1.0;  // Diagonal superior
            A[i + 1][i] = -1.0;  // Diagonal inferior
        }
    }
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (i != j && i != j - 1 && i != j + 1) A[i][j] = 0.0;

    for (int i = 0; i < N; i++) C[0][i] = f(a + h*i);
    for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) temp1[i][j] = theta*lambda*A[i][j];
    for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) temp1[i][j] += B[i][j];
    double dummy = theta - 1;
    for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) temp2[i][j] = dummy*lambda*A[i][j];
    for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) temp2[i][j] += B[i][j];
    double tolerancia = 1e-5;
    for (int j = 1; j < N; j++)
        C[j] = gauss_seidel_(N, temp1, multiplicar_matriz_vector_(N, temp2, C[j - 1]), tolerancia,
    iteraciones);
    // liberamos memoria
    liberar_matriz(N,A);
    liberar_matriz(N, B);
    liberar_matriz(N, temp1);
    liberar_matriz(N, temp2);
    return C;
}
```

Con las condiciones de frontera del sistema que se nos propone, y graficando con `gnuplot` los datos tenemos la siguiente gráfica



Problema 3

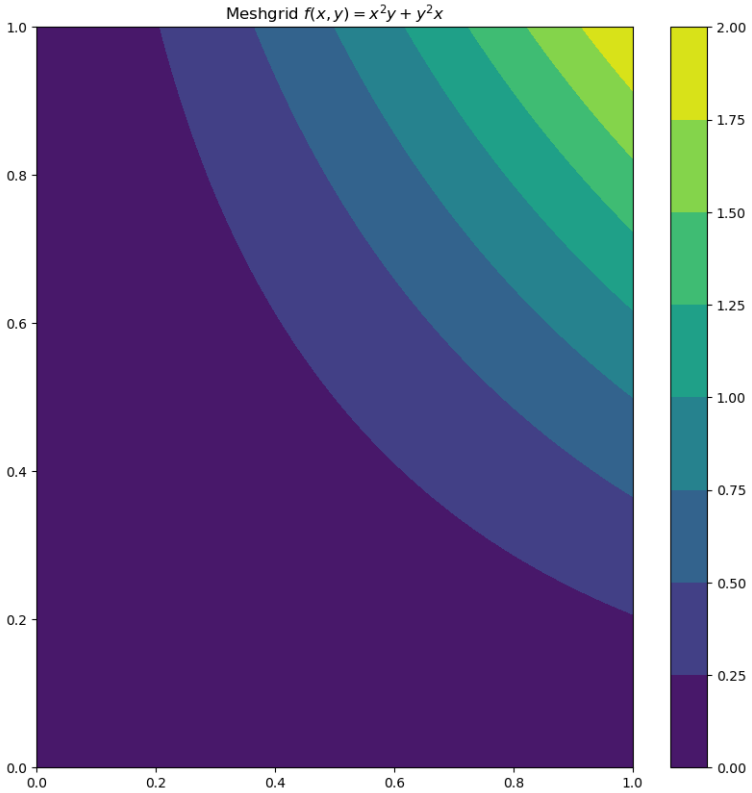
Sea $u = u(x, y)$ tal que

$$u(x, y) = x^2 y + y^2 x \tag{1}$$

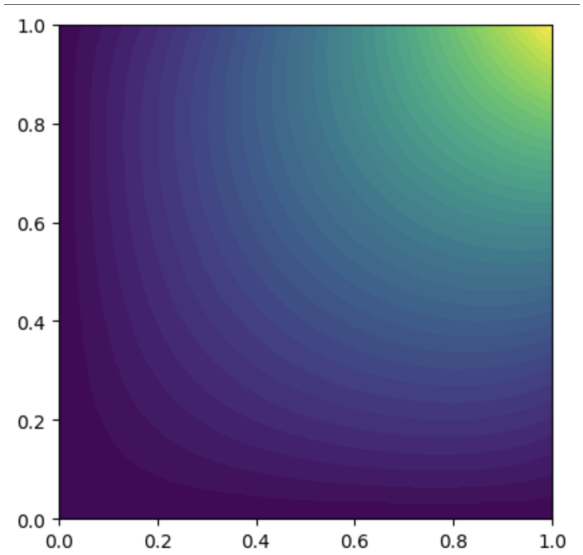
a) Aplicando el operador nabla a $u(x, y)$ tenemos

$$f(x, y) = -\nabla^2 u(x, y) = -\nabla^2 (x^2 y + y^2 x) = -2(x + y)$$

b) Evaluando la función (1) sobre el cuadro unitario tenemos el siguiente `meshgrid`



c) Resolviendo mediante el método de elemento finito tenemos



Esto modificando el código tal que

```
mesh = UnitSquareMesh(20, 20)
V = FunctionSpace(mesh, 'P', 1)

# Define boundary condition
u_D = Expression('x[0]*x[0]*x[1] + x[1]*x[1]*x[0]', degree=2)

def boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u_D, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression('2*x[0] + 2*x[1]', degree=2)
```

```
a = dot(grad(u), grad(v))*dx
L = f*v*dx

# Compute solution
u = Function(V)
solve(a == L, u, bc)

# Save the solution to a file
vtkfile = File('poisson/solution.pvd')
vtkfile << u

# Plot the solution
plot(u)
plt.show()
```

Uno puede notar las partes del código que fueron modificadas, en esencia solo es necesario modificar las lineas correspondientes a 5 y 15 para ajustarlo a nuestro problema.

d) Para calcular las normas utilizamos las funciones incorporadas en la paquetería, tal que

```
# Compute errors in L2 and L_infinity norms
error_L2 = errornorm(u_D, u, 'L2')

# The maximum error at a single vertex
vertex_values_u_D = u_D.compute_vertex_values(mesh)
vertex_values_u = u.compute_vertex_values(mesh)
error_max = np.max(np.abs(vertex_values_u_D - vertex_values_u))

print('Error in L2 norm:', error_L2)
print('Error in L_infinity norm:', error_max)
```

Teniendo como resultados:

```
Error in L2 norm: 0.1663932501131333
Error in L_infinity norm: 0.303808852016775
```

4. Conclusiones

Podemos notar que los métodos para la resolución de ecuaciones diferenciales parciales son bastantes variados en el sentido que proponen soluciones de forma bastantes creativas y diferentes. Asimismo, podemos que en la práctica, existen diferentes formas de enfrentar el problema, siendo que existen ya paqueterías hechas para su cálculo, esto también denota la todavía gran diversidad de soluciones que existen, tales como los métodos de intento iterativos, los métodos analíticos y aquellos donde se requiere una construcción entorno al problema original.

5. Referencias

1. Asmar, N. H. (2016). Partial differential equations with Fourier series and boundary value problems. Courier Dover Publications.
2. Kong, Q., Siauw, T., & Bayen, A. (2020). Python programming and numerical methods: A guide for engineers and scientists. Academic Press.
3. Richard. L. Burden y J. Douglas Faires, Análisis Numérico, 7a Edición, Editorial Thomson Learning, 2002.
4. Samuel S M Wong, Computational Methods in Physics and Enginering, Ed. World Scientific, 3rd Edition, 1997.
5. Teukolsky, S. A., Flannery, B. P., Press, W. H., & Vetterling, W. T. (1992). Numerical recipes in C. SMR, 693(1), 59-70.
6. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, 2007