HW1 Numerical Methods Fall 2017

Michael Laufer

November 5, 2017

1.1 Elliptic, Parabolic, Hyperbolic PDEs

a.

$$\frac{\partial \phi}{\partial t} + \vec{U} \cdot \nabla \phi = 0$$

There are no second order derivatives, therefore $b^2 - 4ac = 0$. The PDE is Parabolic.

b.

$$\frac{\partial \phi}{\partial t} + \vec{U} \cdot \nabla \phi = \Gamma \nabla^2 \phi$$

We can see that \$ a=c=-Γ\$ where \$ Γ is a positive number. Therefore $b^2 - 4ac < 0$. The PDE is Elliptic.

c.

$$\frac{\partial}{\partial x}\left[\Gamma_1 \frac{\partial}{\partial x} - \Gamma_2 \frac{\partial}{\partial y}\right] + \frac{\partial}{\partial y}\left[\Gamma_1 \frac{\partial}{\partial y} - \Gamma_2 \frac{\partial}{\partial x}\right] = 0$$

Rearranging the equation:

$$\Gamma_1 \frac{\partial^2 \phi}{\partial x^2} - (\Gamma_1 + \Gamma_2) \frac{\partial^2 \phi}{\partial x \partial y} + \Gamma_1 \frac{\partial^2 \phi}{\partial y^2} = 0$$

Γ_1 and Γ_2 can be either positive or negative. Splitting into different cases:

- If $\Gamma_1 > \Gamma_2$: $b^2 - 4ac > 0$ and the PDE is hyperbolic
- If $\Gamma_1 < \Gamma_2$: $b^2 - 4ac < 0$ and the PDE is elliptic
- If $\Gamma_1 = \Gamma_2$: $b^2 - 4ac = 0$ and the PDE is parabolic.

d.

$$\frac{\partial\phi}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\phi}{\partial r}\right) = 0$$

Rearranging the equation:

$$\frac{\partial\phi}{\partial t} - \frac{1}{r}\frac{\partial\phi}{\partial r} - \frac{\partial^{2\phi}}{\partial r^2} = 0$$

$b^2 - 4ac = 0$. The PDE is parabolic.

e.

$$\frac{\partial^2\phi}{\partial t^2} + \frac{\partial\phi}{\partial t} - \nabla\cdot(\Gamma\nabla\phi) = \phi$$

Rearranging yields,

$$\frac{\partial^2\phi}{\partial t^2} + \frac{\partial\phi}{\partial t} - \Gamma\nabla^2\phi - \phi = 0$$

In regards to space, the PDE is clearly elliptic as only a single second order spatial derivative exists. In regards to a time-space point of view, given $\Gamma > 0$, $b^2 - 4ac > 0$. The PDE is hyperbolic.

1.2 Galerkin FE method

The analytical solution is found by integrating twice with respect to ϕ and plugging in the provided boundary conditions, resulting in:

$$\phi(x) = -\cos(x) + \cos(1)x + 1$$

To solve this equation using the Galerking Finite Element method we will follow the same procedure as Example 1.1, and choose the same top-hat linear base functions but with a different right hand side.
Using the same basis function gives us the same interior tradiagonal stiffness values:
if $i = j$, $K_{j,i} = 2/\Delta x$
if $i = j + 1$ or $i = j - 1$, $K_{j,i} = -1/\Delta x$
The interior node load vector values are computed just as in example 1.1. Utilizing integration by parts leads to:

$$F_j = -\int_0^1 \cos x \psi(x) dx = \frac{-1}{\Delta x}\left(\int_{x_{i-1}}^{x_i} \cos x\,(x - x_{i-1})\,dx + \int_{x_i}^{x_{i+1}} \cos x\,(x_{i+1} - x)\,dx\right) =$$

$$= \frac{-1}{\Delta x}\left(2\cos x_i - \cos x_{i-1} - \cos x_{i+1}\right)$$

which is valid for $2 \leq j \leq n - 1$

Regarding boundary conditions, we first notice that the we can infer that $a_1 = 0$ and $a_n = 1$. In order to take these values into account in our system, we will force the values at the boundary nodes, leading to the equation:

$$K_{j,i} a_i = F_j$$

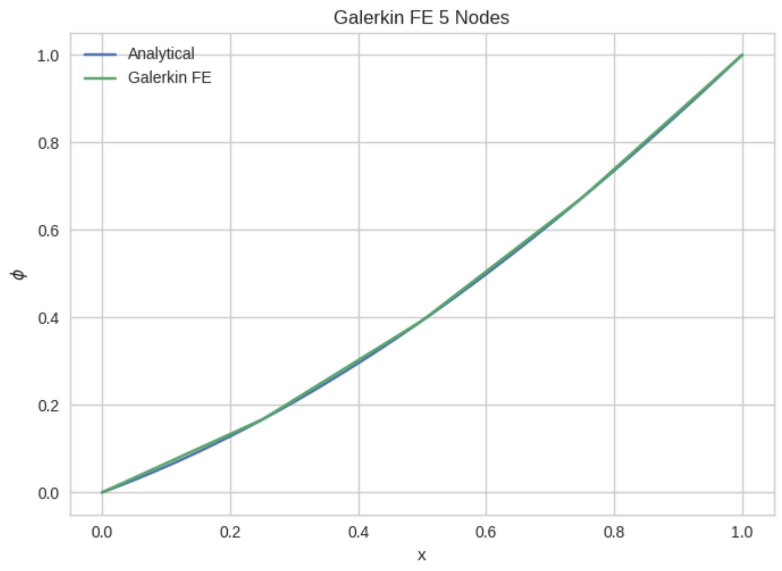
Or more explicitly written:

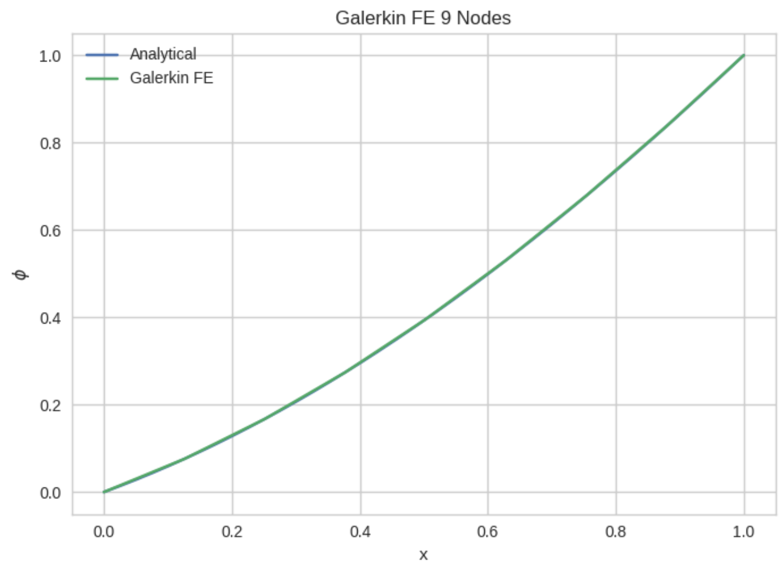
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1/dx & 2/dx & -1/dx & 0 & 0 \\ 0 & -1/dx & 2/dx & -1/dx & 0 \\ 0 & 0 & -1/dx & 2/dx & -1/dx \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 0 \\ F_2 \\ F_3 \\ F_4 \\ 1 \end{bmatrix}$$

Due to fact the matrix is banded (in this case tridiagonal), we can employ the Scipy banded solver, which solves banded matrices much faster and more efficiently than the Numpy linear algebra solver.

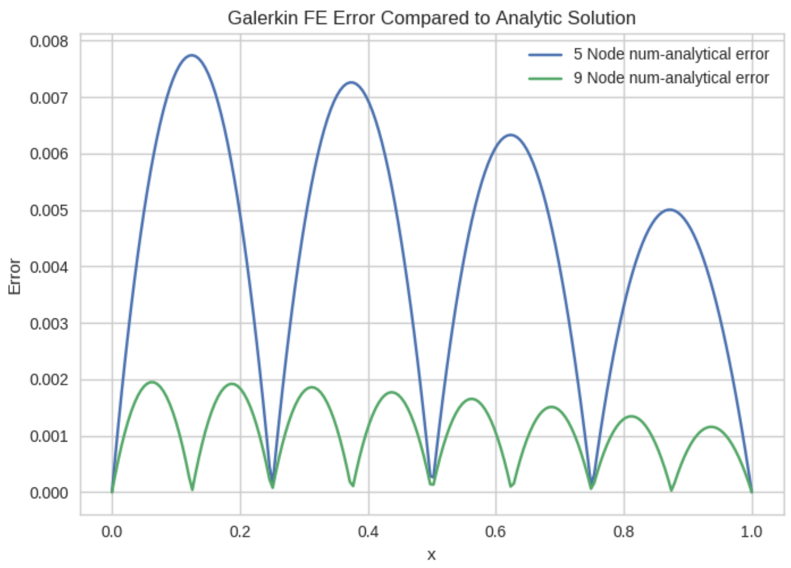
```
from scipy.linalg import solve_banded
def tridiagsolver(K,F)
    ud = np.insert(np.diag(K,1), 0, 0)          # upper diagonal
    d = np.diag(K)                              # main diagonal
    ld = np.insert(np.diag(K,-1), len(F)-1, 0)  # lower diagonal
    ab = np.matrix([ud, d, ld])                 # simplified matrix
    a = solve_banded((1, 1), ab, F)
    return a
```

A plot of the numerical solution plotted against the analytical solution can be seen here for both a 5 node system as well as a 9 node system:





We can see that even for 5 nodes, an accurate solution is obtained. Additionally a plot of the error throughout the domain can be plotted.



We can see that at nodal values we get particularly good accuracy. We can also observe that errors are largest as the point moves away from the nodal points.

The fluxes are computed by differentiating the $\phi(x)$ expansion. The coefficients are constant with regard to the spatial derivative and thus:

$$J_L = \frac{\partial \phi}{\partial x}(0) = a_1 \frac{-1}{dx} + a_2 \frac{1}{dx}$$

$$J_R = \frac{\partial \phi}{\partial x}(1) = a_n \frac{1}{dx} + a_{n-1} \frac{-1}{dx}$$

Lastly the numerical net source in the domain by integration is computed by integrating our initial equaiton from 0 to 1:

$$S = \int_0^1 \frac{\partial^2 \phi}{\partial x^2} = \frac{\partial \phi}{\partial x}(1) - \frac{\partial \phi}{\partial x}(0) = J_R - J_L$$

Numerical results are compared with anylytical results in the following table:

	Analytical	**Galerking FE 5 Nodes**	**Galerkin FE 9 Nodes**
J_L	0.5403	0.66464	0.60272
J_R	1.3817	1.3058	1.3458
S	0.8414	0.64116	0.74308
$I = J_R - J_L - S$	0	-0.202	-0.0983

It is clear that as more nodes are added, the solution better converges to the analytical solution. We can observe that $I \neq 0$ for the numerical cases, suggesting that local conservation is not satisfied.

Appendix: Python Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_banded
import math
import seaborn as sns
sns.set_style("whitegrid")


def tridiagsolver(K,F):
    ud = np.insert(np.diag(K,1), 0, 0)           # upper diagonal
    d = np.diag(K)                               # main diagonal
    ld = np.insert(np.diag(K,-1), len(F)-1, 0)   # lower diagonal
    ab = np.matrix([ud, d, ld])                  # simplified matrix
    a = solve_banded((1, 1), ab, F)
    return a

def psi(j,x, dx):
    if x > (j+1)*dx or x < (j-1)*dx:
        return 0
    elif x < j*dx:
        return (x - (j-1)*dx)/dx
    else:
        return ((j+1)*dx - x)/dx

def galerkin1d(nx):
    x = np.linspace(0,1,nx)
    dx = 1.0/(nx-1)
    K = np.zeros((nx,nx))                   # Stiffness matrix
    for i in range(nx):
        if i == 0:
            K[i,i] = 1
            K[i,i+1] = 0
        elif i == len(K)-1:
            K[i,i] = 1
            K[i,i-1] = 0
        else:
            K[i,i] = 2/dx
            K[i,i-1] = -1/dx
            K[i,i+1] = -1/dx

    F = np.zeros(nx)                            # Load vector
    F [0] = 0
    F[1:-1] = (-1.0/dx)*(2*np.cos(x[1:-1]) - np.cos(x[0:-2]) - np.cos(x[2:]))
    F[-1] = 1

    a = tridiagsolver(K,F)                      # Solve system

    nxplot = 200                                # Recombine phi from basis functions
    plot_x = np.linspace(0,1,nxplot)
    phi_galerkin = np.zeros(nxplot)
    for i in range(len(plot_x)):
        for j in range(len(a)):
            phi_galerkin[i] +=  a[j]*psi(j, plot_x[i], dx)
    return phi_galerkin

if __name__ == "__main__":
    plot_x = np.linspace(0,1,200)
    phi_galerkin5 = galerkin1d(nx=5)
    phi_galerkin9 = galerkin1d(nx=9)
    phi_analy = -np.cos(plot_x) + np.cos(1)*plot_x + 1

    plt.figure(1)
    plt.plot(plot_x, phi_analy, label= "Analytical")
    plt.plot(plot_x, phi_galerkin5, label="Galerkin FE")
    plt.title("Galerkin FE 5 Nodes")
    plt.ylabel("$\phi$")
    plt.xlabel("x")
    plt.legend()

    plt.figure(2)
    plt.plot(plot_x, phi_analy, label= "Analytical")
    plt.plot(plot_x, phi_galerkin9, label="Galerkin FE")
    plt.title("Galerkin FE 9 Nodes")
    plt.ylabel("$\phi$")
    plt.xlabel("x")
    plt.legend()

    plt.figure(3)
    plt.plot(plot_x, np.abs(phi_analy-phi_galerkin5), label= "5 Node num-analytical er
    plt.plot(plot_x, np.abs(phi_analy-phi_galerkin9), label= "9 Node num-analytical er
    plt.title("Galerkin FE Error Compared to Analytic Solution")
    plt.ylabel("Error")
    plt.xlabel("x")
    plt.legend()
    plt.show()
```