# 1 Method of Mean Weighted Residuals

**Lab Objective:** *We introduce the method of mean weighted residuals (MWR) and use it to derive a pseudospectral method. This method will then be used to solve several boundary value problems.*

Consider a linear differential equation

$$Lu = f,$$

defined on the interval $[-1, 1]$, together with associated boundary conditions. We will approximate the solution $u(x)$ by a linear combination of $N + 1$ basis functions $\phi_i$, so that

$$u(x) \approx u_N(x) = \sum_{i=0}^{N} a_i \phi_i(x).$$

To determine appropriate constants $a_i$, we then minimize the residual function

$$R(x, u_N) = Lu_N - f.$$

Note that $R(x, u) = Lu - f = 0$ for the true solution $u(x)$.

This general strategy is often called the method of mean weighted residuals (MWR method). The MWR method is a general framework that describes many other, more specific methods. These more specific methods come from differing approaches to minimizing the residual $R(x, u_N)$, and the choice of basis functions $\phi_i$.

## The Pseudospectral Method

The pseudospectral or collocation method is obtained from the MWR method by forcing the residual function $R(x, u_N)$ to equal zero at $N + 1$ points in $[-1, 1]$, called collocation points. When done correctly, the pseudospectral method gives high accuracy and converges rapidly.

We will let the basis functions $\phi_i$ be the Chebyshev polynomials,

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

and the collocation points will be the Gauss-Lobatto points, $x_i = \cos(\pi i/N)$, $i = 0, \ldots, N$. The appropriate solution $u_N$ may be represented with two equivalent forms. First, $u_N$ can be described with the first $N+1$ coefficients $\{a_i\}_{i=0}^N$ of its expansion in the Chebyshev polynomials. Since $u_N$ is a polynomial of order $N$, it may be uniquely described by its values at the collocation points, that is, the unknown values $\{u_N(x_i)\}_{i=0}^N$.

These equivalent forms satisfy

$$MA = F \tag{1.1}$$

and

$$LU = F \tag{1.2}$$

where

$$\begin{aligned} U_i &= u(x_i), \\ A_i &= a_i, \\ F_i &= f(x_i), \\ L_{ij} &= (LC_j(x))|_{x=x_i}, \\ M_{ij} &= (L\phi_j(x))|_{x=x_i}. \end{aligned}$$

The functions $C_j$ above are the cardinal functions, defined to be the polynomials of least degree satisfying

$$C_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Thus, $u_N$ can also be expanded in the basis of the cardinal functions:

$$u_N(x) = \sum_{j=0}^{N} u_N(x_j) C_j(x).$$

When $L = d/dx$, the matrix corresponding to equation (1.2) is given by

$$L_{ij} = \frac{dC_j}{dx}(x_i) = \begin{cases} (1+2N^2)/6 & i = j = 0, \\ -(1+2N^2)/6 & i = j = N, \\ -x_j/[2(1-x_j^2)] & i = j,\ 0 < j < N, \\ (-1)^{i+j}\alpha_i/[\alpha_j(x_i - x_j)] & i \neq j. \end{cases}$$

where $\alpha_0 = \alpha_N = 2$, and $\alpha_j = 1$ otherwise.

This matrix is often called the differentiation matrix ($D$), and can be used to piece together the matrix $L$ for more complicated differential operators. A stable, vectorized function to build the differentiation matrix is given below.

```
import numpy as np

def cheb(N):
    x =  np.cos((np.pi/N)*np.linspace(0,N,N+1))
    x.shape = (N+1,1)
    lin = np.linspace(0,N,N+1)
```

```
    lin.shape = (N+1,1)

    c = np.ones((N+1,1))
    c[0], c[-1] = 2., 2.
    c = c*(-1.)**lin
    X = x*np.ones(N+1) # broadcast along 2nd dimension (columns)

    dX = X - X.T

    D = (c*(1./c).T)/(dX + np.eye(N+1))
    D  = D - np.diag(np.sum(D.T,axis=0))
    x.shape = (N+1,)
    # Here we return the differentiation matrix and the Chebyshev points,
    # numbered from x_0 = 1 to x_N = -1
    return D, x
```

## Using the Differentiation Matrix

**Problem 1.** Use the differentiation matrix to numerically approximate the derivative of $u(x) = e^x \cos(6x)$ on a grid of $N$ Chebychev points where $N = 6, 8$, and 10. (Use the linear system $DU \approx U'$.) Then use barycentric interpolation (`scipy.interpolate.barycentric_interpolate`) to approximate $u'$ on a grid of 100 evenly spaced points.

Graphically compare your approximation to the exact derivative. Note that this convergence would not be occurring if the collocation points were equally spaced.

To approximate $u''(x)$ on the grid $\{x_i\}$, we use

$$U'' \approx D^2 U.$$

The BVP

$$\begin{aligned} u'' &= f(x), \quad x \in [-1, 1], \\ u(-1) &= 0, \quad u(1) = 0, \end{aligned}$$

can be discretized by the linear system

$$D^2 U = F, \tag{1.3}$$

where $F = [f(x_0), \ldots, f(x_N)]^T$. Since we have Dirichlet boundary conditions of 0, we can satisfy the boundary condition by forcing $U[0] = U[N] = 0$. This is done by replacing the first and last equations in (1.3) by the boundary conditions.

```
#The following code will force U[0] = U[N] = 0
D, x = cheb(N)    #for some N
D2 = np.dot(D, D)
D2[0,:], D2[-1,:] = 0, 0
D2[0,0], D2[-1,-1] = 1, 1
F[0], F[-1] = 0, 0
```

**Problem 2.** Use the pseudospectral method to solve the boundary value problem

$$u'' = e^{2x}, \quad x \in (-1, 1),$$
$$u(-1) = 0, \quad u(1) = 0.$$

Use $N = 8$ in the `cheb(N)` method and use barycentric interpolation to approximate $u$ on 100 evenly spaced points. Compare your numerical solution with the exact solution,

$$u(x) = \frac{-\cosh(2) - \sinh(2)x + e^{2x}}{4}.$$

**Problem 3.** Use the pseudospectral method to solve the boundary value problem

$$u'' + u' = e^{3x}, \quad x \in (-1, 1),$$
$$u(-1) = 2, \quad u(1) = -1.$$

Use $N = 8$ in the `cheb(N)` method and use barycentric interpolation to approximate $u$ on 100 evenly spaced points.

The previous exercise involved setting up and solving a linear system

$$AU = F,$$

where $F$ is a vector whose entries are $e^{3x}$ evaluated at the collocation points $x_j$, and $U$ represents the approximation to the solution $u$ at those points. However, whenever the ODE is nonlinear, the discretization becomes a nonlinear system of equations that must be solved using Newton's method. The next exercise contains a BVP whose ODE is nonlinear, with the additional complexity that the domain of the problem is not $[-1, 1]$.

**Problem 4.** Use the pseudospectral method to solve the boundary value problem

$$u'' = \lambda \sinh(\lambda u), \quad x \in (0, 1),$$
$$u(0) = 0, \quad u(1) = 1$$

for several values of $\lambda$: $\lambda = 4, 8, 12$. Begin by transforming this BVP onto the domain $-1 < x < 1$. Use $N = 20$ in the `cheb(N)` method and use barycentric interpolation to approximate $u$ on 100 evenly spaced points.

Below is sample code for implementing Newton's Method

```
from scipy.optimize import root

N = 20
D, x = cheb(20)

def F(U):
    out = None  #Set up the equation you want the root of.
```

```
    #Make sure to set the boundaries correctly

    return out  #Newtons Method will update U until the output is all 0's.

guess = None    #Make your guess, same size as the cheb(N) output
solution = root(F, guess).x
```

## Minimizing the Area of a Surface of Revolution

A surface of revolution that minimizes its area is an example of a larger class of surfaces called minimal surfaces. A famous example of a minimal surface is a soap bubble. Soap bubbles minimize their surface area while containing a fixed volume of air. This behavior extends to merged bubbles, and a soap film whose boundary is a wire frame. Minimal surfaces have applications in molecular engineering and material science, and general relativity, where they describe the apparent horizon of a black hole.

Consider a function $y(x)$ defined on $[-1, 1]$ satisfying $y(-1) = a$, $y(1) = b$. The area of the surface obtained by revolving the graph of $y(x)$ about the $x$-axis is given by

$$T[y(x)] = \int_{-1}^{1} 2\pi y(x)\sqrt{1 + (y'(x))^2}\, dx.$$

To find the function $y(x)$ whose surface of revolution minimizes surface area, we must minimize the functional $T[y]$. This is a classical problem from a branch of mathematics called the calculus of variations. Standard derivatives allow us to find the minimum values of functions defined on $\mathbb{R}^n$, and where they occur. The calculus of variations allows us to find the minimum values of functions whose input are other functions.

From the calculus of variations we know that a necessary condition for $y(x)$ to minimize $T[y]$ is that the Euler-Lagrange equation must be satisfied:

$$L_y - \frac{d}{dx}L_{y'} = 0,$$

where $L(x, y, y') = 2\pi y\sqrt{1 + (y')^2}$. Simplifying the Euler-Lagrange equation for our problem results in the ODE

$$yy'' - (y')^2 - 1 = 0.$$

Discretizing this ODE using the pseudospectral method results in the (nonlinear) system of equations

$$Y \cdot (D^2Y) - (DY) \cdot (DY) = I,$$

where $I$ is a vector of ones.

**Problem 5.** Find the function $y(x)$ that satisfies $y(-1) = 1$, $y(1) = 7$, and whose surface of revolution (about the $x$-axis) minimizes surface area. Compute the surface area, and plot the surface. Use $N = 50$ in the `cheb(N)` method and use barycentric interpolation to approximate $u$ on 100 evenly spaced points.

Below is sample code for creating the 3D wireframe figure.

```
from mpl_toolkits.mplot3d import Axes3D
```

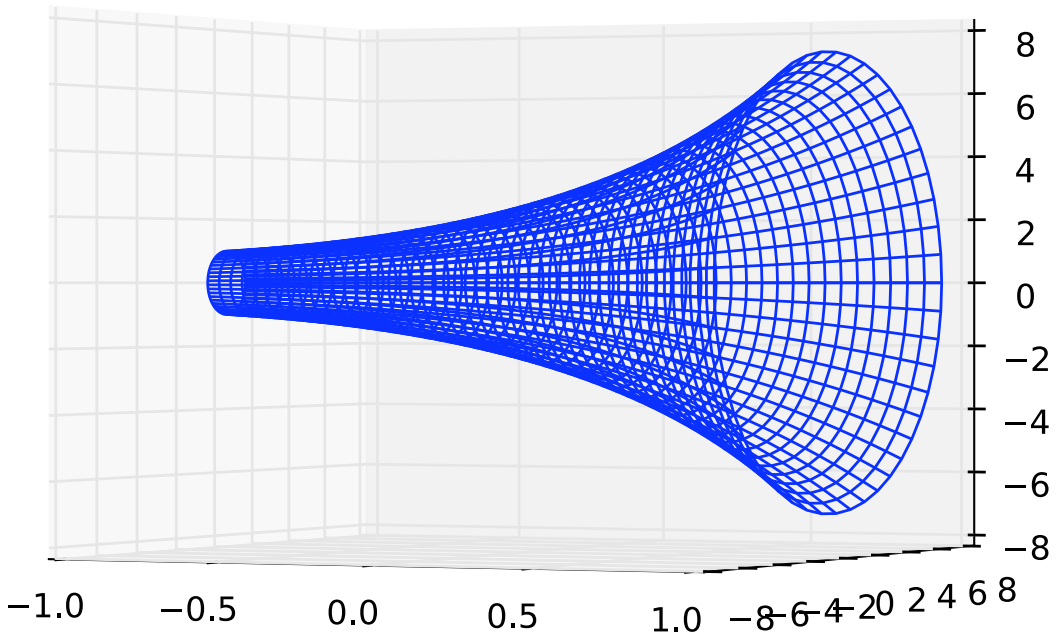


Figure 1.1: The minimal surface corresponding to Problem 5.

```
barycentric = None  #This is the output of barycentric_interpolate() on ↩
    100 points

lin = np.linspace(-1, 1, 100)
theta = np.linspace(0,2*np.pi,401)
X, T = np.meshgrid(lin, theta)
Y, Z = barycentric*np.cos(T), barycentric*np.sin(T)

fig = plt.figure()
ax = fig.gca(projection="3d")
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```