Lecture 17: Spectral Methods for BVPs (2)

Today:

- Continue with ***global spectral methods*** for ***boundary value problems (BVPs)***
  - Implementation, implementation, implementation

Where are we up to now?

Last time.

(A) We developed a framework for approximating the solution to a BVP using global spectral methods

(B) We arrived at a linear system to solve for the coefficients

$$\Longrightarrow \begin{bmatrix} (b_0, b_0)_E & (b_1, b_0)_E & \cdots & (b_{n-1}, b_0)_E & (b_n, b_0)_E \\ (b_0, b_1)_E & (b_1, b_1)_E & \cdots & (b_{n-1}, b_1)_E & (b_n, b_1)_E \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ (b_0, b_{n-1})_E & (b_1, b_{n-1})_E & \cdots & (b_{n-1}, b_{n-1})_E & (b_n, b_{n-1})_E \\ (b_0, b_n)_E & (b_1, b_n)_E & \cdots & (b_{n-1}, b_n)_E & (b_n, b_n)_E \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = - \begin{bmatrix} (f, b_0)_s \\ (f, b_1)_s \\ \vdots \\ (f, b_{n-1})_s \\ (f, b_n)_s \end{bmatrix}$$

Remember, this equation system assumes homogeneous Dirichlet BCs!

(C) Can back out our approximate solution via

$$u_a(x) = \sum_{j=0}^{n} c_j b_j(x)$$

Today. Implementing these ***global spectral methods***

Consider the specific example of approximating the BVP solution on the space defined by trigonometric functions

- What does (1) become for this case?
- How do we implement it in code?

Global spectral methods:
Approximating the solution of the 1D Poisson problem with zero Dirichlet BCs

Continue to focus on the 1D Poisson problem.

$$\frac{d^2u}{dx^2} = f, \quad x \in [a,b] \tag{2}$$

Could extend the ensuing derivation to a more general BVP

$$u(a) = \cancel{u_a}^{\,0}, \; u(b) = \cancel{u_b}^{\,0} \tag{3}$$

Could consider other BCs, this just makes the derivation a little cleaner

We need a subspace and associated basis

Let's choose a space we call $\mathcal{T}_0^n$, defined as the collection of all trigonometric functions of degree n or less that satisfy the zero-BCs

A basis for this space is

$$\left\{ \sin\left(\frac{\pi(x-a)}{b-a}\right), \sin\left(\frac{2\pi(x-a)}{b-a}\right), \ldots, \sin\left(\frac{n\pi(x-a)}{b-a}\right) \right\}$$

Notice we don't need the cosine terms because they don't satisfy the BCs!

So what does the matrix in (1) become for this choice of basis?

Figuring out the matrix in (1) for our choice of basis

We have that

$$\left(\sin\left(\frac{j\pi(x-a)}{b-a}\right), \sin\left(\frac{k\pi(x-a)}{b-a}\right)\right)_E = \int_a^b \sin'\left(\frac{j\pi(x-a)}{b-a}\right)\sin'\left(\frac{k\pi(x-a)}{b-a}\right)dx$$

$$= \frac{jk\pi^2}{(b-a)^2}\int_a^b \cos\left(\frac{j\pi(x-a)}{b-a}\right)\cos\left(\frac{k\pi(x-a)}{b-a}\right)$$

$$= \begin{cases} 0 & j \neq k \\ \frac{j^2\pi^2}{2(b-a)} & j = k \end{cases}$$

So that (1) becomes

Note: we switched starting index to 1 to match the indexing for the sine functions. Just convention!

$$-\frac{\pi^2}{2(b-a)}\begin{bmatrix} 1^2 & 0 & \cdots & 0 & 0 \\ 0 & 2^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & (n-1)^2 & 0 \\ 0 & 0 & \cdots & 0 & n^2 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} \left(f, \sin\left(\frac{\pi(x-a)}{b-a}\right)\right)_s \\ \left(f, \sin\left(\frac{2\pi(x-a)}{b-a}\right)\right)_s \\ \vdots \\ \left(f, \sin\left(\frac{(n-1)\pi(x-a)}{b-a}\right)\right)_s \\ \left(f, \sin\left(\frac{n\pi(x-a)}{b-a}\right)\right)_s \end{bmatrix}$$

And in python…

```python
#preamble stuff


#Interval properties
a = 2
b = 4
L = (b-a)

#fine grid for pretty plotting of soln
xx = np.linspace(a,b,1000)

#BCs
alpha = uex( a, a, b )
beta = uex( b, a, b )

#Consider different n values to see how solution changes as we change n
nv = np.array([10, 20, 40, 80])

#initalize
err = np.zeros([len(nv),1])
fig, ax = plt.subplots(len(nv[range(3)]), 1, sharex=True, squeeze=False)

for j in range(len(nv)):
    n = nv[j]

    #solve for coeffs
    c = np.zeros([n,1])

    #matrix is diagonal so can solve for each coeff independently
    for jj in range(n-1):
        #rhs
        #could replace trapz with a better quadrature rule!
        bj = np.trapz( f(xx,a,b) * np.sin( (jj+1)*np.pi*(xx-a)/ (b-a) ), x=xx)

        #normalization
        dj = -2*(b-a)/((jj+1)**2*np.pi**2);
        c[jj] = bj*dj;


    # Now that we have coeffs, express approx soln as linear combo of basis
    # fcns
    u = 0;
    for kk in range(n-1):
        u = u + c[kk] * np.sin( (kk+1)*np.pi*(xx-a)/ (b-a) );

    #get error
    err[j] = LA.norm(u - uex(xx, a,b))/ LA.norm(uex(xx,a,b))

    #Then do some plotting...
```

s/GoogleDrive/My Drive/Me/UIUC/Classes/AE 370/Me/SP2021/Content/Week 10/Lectures/Monday/trig_spectral_demo.py 85:5 LF