From Finite Differences to Spectral Accuracy

Pavni: Acharya, last week you mentioned *spectral methods*. I've heard they're very accurate for smooth problems. But… how are they different from finite differences or finite elements?

Acharya: A good question, Pavni. Let's start from what you already know. In **finite difference** methods, we approximate derivatives *locally* using Taylor expansions — yes?

Pavni: Yes. We replace $u'(x)$ by something like $\frac{u(x+h)-u(x-h)}{2h}$, and the error goes as $O(h^2)$ if the function is smooth.

Acharya: Excellent. So, each approximation uses only *neighboring points*. Finite elements do something similar — piecewise approximations over small elements. But tell me — if your function is perfectly smooth, say $u(x) = e^x$, do you really need to treat each small interval separately?

Pavni: Hmm… maybe not. If the function is smooth everywhere, I could use one large polynomial to approximate it globally.

Acharya: Precisely. That is the spirit of *spectral methods.* Instead of stitching local approximations, we represent the whole function as a sum of *global basis functions*:

$$u_N(x) = \sum_{k=0}^{N} a_k \, \phi_k(x).$$

Pavni: So the ϕ_k's are like the building blocks?

Acharya: Exactly. For periodic problems, we might take $\phi_k(x) = e^{ikx}$ — the Fourier basis. For problems on $[-1, 1]$, we prefer *orthogonal polynomials* such as Chebyshev or Legendre polynomials.

Pavni: Then a_k are like coefficients of a series expansion?

Acharya: Yes — determined by enforcing the PDE or by matching the function values at specific points, called *collocation points.*

Pavni: But Acharya, won't a global polynomial oscillate a lot, like in the Runge phenomenon?

Acharya: Ah, you've been observant! The Runge phenomenon indeed haunts equispaced interpolation. Let's recall: $f(x) = \frac{1}{1+25x^2}$. Interpolate it with a degree-10 polynomial at equally spaced points — what happens?

Pavni: The polynomial wiggles wildly near the boundaries.

Acharya: Exactly. But there's a cure: use **Chebyshev points**, clustered near -1 and 1:

$$x_j = \cos\left(\frac{j\pi}{N}\right), \quad j = 0, 1, \ldots, N.$$

This choice minimizes the interpolation error and stabilizes the approximation.

Pavni: So spacing the points unevenly makes the polynomial behave better?

Acharya: Very much so. Think of it as giving more attention to the troublesome boundary regions.

Pavni: You said spectral methods are *very accurate.* How accurate are we talking?

Acharya: If the function is smooth, the error decays **exponentially** with the number of modes N. Compare that with finite differences, where the error decays algebraically — like $O(N^{-2})$ or $O(N^{-4})$. This exponential behavior is called **spectral convergence**.

Pavni: That's amazing! So doubling the number of points can reduce the error by orders of magnitude.

Acharya: Exactly. But remember — only when the solution is smooth. If the solution has discontinuities or sharp corners, spectral methods lose their magic.

Pavni: I see. So the basis choice matters: Fourier for periodic, Chebyshev for nonperiodic.

Acharya: Well summarized. Let's take an example to make this concrete.

Example: 1D Poisson Equation

$$u''(x) = \sin(\pi x), \qquad u(-1) = u(1) = 0.$$

Acharya: We'll approximate $u(x)$ at the Chebyshev–Gauss–Lobatto points x_j. Then we build the *differentiation matrix* D, whose entries satisfy

$$u'(x_j) \approx \sum_{k=0}^{N} D_{jk}\, u(x_k),$$

and hence $D^2 u \approx u''(x_j)$.

Pavni: So instead of computing derivatives analytically, we use matrix multiplication?

Acharya: Exactly. The discrete derivative is a *linear transformation.* Once we have D^2, we simply solve the linear system $D^2 u = f$, adjusting the first and last rows for boundary conditions.

Pavni: That seems surprisingly compact.

Acharya: Indeed. The entire solver can be written in less than 10 lines of MATLAB or Python. That's the power of the *spectral collocation method.*

Pavni: How does it compare with finite differences?

Acharya: For the same $N = 20$, finite difference might give an error of 10^{-2}, while the spectral method can reach 10^{-10}. Both use the same number of grid points!

Pavni: That's… spectacular. Now I understand why they're called *spectral* methods.

Acharya: (smiles) A fitting pun, Pavni. The term actually comes from *spectrum* — the set of basis functions, much like frequencies in Fourier analysis.

Pavni: Are there any downsides?

Acharya: Yes. Spectral methods can struggle with non-smooth problems, complex geometries, or localized features. But for smooth flows — such as those in polymer extrusion or viscoelastic jets — they shine brilliantly.

Pavni: That connects beautifully to your research!

Acharya: Exactly. In the next lecture, we'll derive the **Chebyshev differentiation matrix** and use it to solve PDEs — perhaps even the heat equation.

Pavni: Wonderful. So today's takeaways are: 1. Spectral methods use *global* basis functions.
2. Chebyshev points stabilize interpolation.
3. For smooth problems, errors decay *exponentially*.
4. Derivatives are represented by *differentiation matrices*.

Acharya: Perfect summary, Pavni. Now, try plotting the interpolation of e^x using Chebyshev points and observe the error. You'll *see* spectral convergence with your own eyes.

References

- L. N. Trefethen, *Spectral Methods in MATLAB* (SIAM).
- J. P. Boyd, *Chebyshev and Fourier Spectral Methods*.
- Canuto et al., *Spectral Methods: Fundamentals in Single Domains*.

Demos (Optional)

> **Note**
>
> These sections are optional in-class or homework demos. They're collapsed by default.

Demo 1 — Runge phenomenon vs. Chebyshev nodes (Python)

```
#| label: fig-runge
#| fig-cap: "Runge phenomenon at equispaced points vs. stability at Chebyshev points."
import numpy as np, matplotlib.pyplot as plt
```

```
f = lambda x: 1/(1 + 25*x**2)
xx = np.linspace(-1, 1, 1000)
plt.figure()
plt.plot(xx, f(xx), lw=2, label="f(x)")

for N in [5, 10, 15]:
    # Equispaced
    x_eq = np.linspace(-1, 1, N+1)
    p_eq = np.poly1d(np.polyfit(x_eq, f(x_eq), N))
    plt.plot(xx, p_eq(xx), linestyle="--", label=f"Equispaced N={{N}}")

for N in [5, 10, 15]:
    # Chebyshev
    j = np.arange(N+1)
    x_ch = np.cos(np.pi * j / N)
    p_ch = np.poly1d(np.polyfit(x_ch, f(x_ch), N))
    plt.plot(xx, p_ch(xx), label=f"Chebyshev N={{N}}")
plt.legend(); plt.title("Runge vs. Chebyshev"); plt.xlabel("x"); plt.ylabel("y")
plt.show()
```

Demo 2 — Exponential convergence for e^x (Python)

```
#| label: fig-exp-conv
#| fig-cap: "Interpolation error for e^x at Chebyshev nodes shows near-exponential deca
import numpy as np, matplotlib.pyplot as plt

f = lambda x: np.exp(x)
xx = np.linspace(-1,1,2000)
fxx = f(xx)

Ns = range(4, 40)
errs = []
for N in Ns:
    j = np.arange(N+1)
    x = np.cos(np.pi * j / N)
    p = np.poly1d(np.polyfit(x, f(x), N))
    errs.append(np.max(np.abs(p(xx) - fxx)))

plt.figure()
plt.semilogy(list(Ns), errs, marker='o')
plt.xlabel("N"); plt.ylabel("max error"); plt.title("Spectral-like convergence for e^x
plt.grid(True, which="both"); plt.show()
```

Demo 3 — Chebyshev-collocation Poisson solver (MATLAB)

```
% Chebyshev differentiation matrix (Trefethen's cheb.m)
function [D,x] = cheb(N)
    if N==0, D=0; x=1; return, end
    x = cos(pi*(0:N)/N)'; c = [2; ones(N-1,1); 2] .* (-1).^(0:N)';
    X = repmat(x,1,N+1); dX = X - X';
    D = (c*(1./c)')./(dX + (eye(N+1))); % off-diagonal
    D = D - diag(sum(D,2));             % diagonal
end

% Poisson problem u'' = f with u(-1)=u(1)=0
N = 20; [D,x] = cheb(N); D2 = D^2;
f = sin(pi*x);
% Enforce Dirichlet BCs
D2(1,:) = 0; D2(1,1) = 1; f(1) = 0;
D2(end,:) = 0; D2(end,end) = 1; f(end) = 0;
u = D2\f;
plot(x,u,'-o'); xlabel('x'); ylabel('u'); title('Chebyshev collocation solution')
```