An Introduction to Spectral Methods in Python

Jonah M. Miller

Perimeter Institute

Tuesday, November 4, 2014

Spectral Methods Vs. Advantages and Disadvantages.

Advantages

- Extremely fast convergence if resolution is Δx, then error ϵ goes as $c^{-\Delta x}$.
- Compare this to finite differences, where the error falls off as $(\Delta x)^c$, where c is the order of the method.
- Mathematically very elegant

Disadvantages

- Performs badly if resolution Δx is larger than the relevant physical length scale. Finite differences methods do much better.
- Good only for smooth solutions. Cannot handle shocks.
- Geometrically inflexible.

Spectral Methods Vs. Advantages and Disadvantages.

Advantages

- Extremely fast convergence if resolution is Δx, then error ϵ goes as $c^{-\Delta x}$.
- Compare this to finite differences, where the error falls off as $(\Delta x)^c$, where c is the order of the method.
- Mathematically very elegant

Disadvantages

- Performs badly if resolution Δx is larger than the relevant physical length scale. Finite differences methods do much better.
- Good only for smooth solutions. Cannot handle shocks.
- Geometrically inflexible.

Note:

There exist hybrid methods (e.g., *discontinuous Galerkin* methods) that combine the strengths of spectral methods with the strengths of shock-capturing methods.

The Method of Lines

The Method of Lines

- Suppose we want to solve the linear advection equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x}$$

The Method of Lines

- Suppose we want to solve the linear advection equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x}$$

- Discretize *space only*

$$\frac{\partial}{\partial t}u_i(t) = c\frac{u_i(t) - u_{i-1}(t)}{\Delta x}$$

The Method of Lines

- Suppose we want to solve the linear advection equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x}$$

- Discretize *space only*

$$\frac{\partial}{\partial t}u_i(t) = c\frac{u_i(t) - u_{i-1}(t)}{\Delta x}$$

- This is a *vector* ODE equation:

$$\frac{\partial}{\partial t}\mathbf{u} = D\mathbf{u},\ \text{for some matrix } D$$

The Method of Lines

- Suppose we want to solve the linear advection equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x}$$

- Discretize *space only*

$$\frac{\partial}{\partial t}u_i(t) = c\frac{u_i(t) - u_{i-1}(t)}{\Delta x}$$

- This is a *vector* ODE equation:

$$\frac{\partial}{\partial t}\mathbf{u} = D\mathbf{u},\ \text{for some matrix } D$$

- Solve using ODE methods discussed earlier such as RK2

What do we Discretize? It's a Matter of Perspective

What do we Discretize? It's a Matter of Perspective

Equation

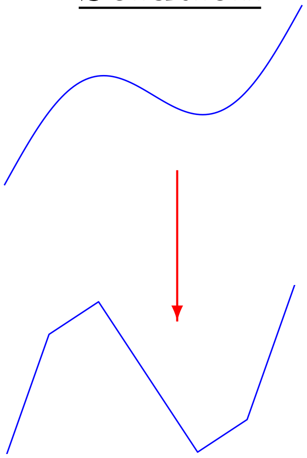
$$\begin{tikzcd} \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \arrow[d] \\ \frac{u_i^{n+1}-u_i^n}{\Delta t} + c\frac{u_i^n-u_{i-1}^n}{\Delta x} = 0 \end{tikzcd}$$

What do we Discretize? It's a Matter of Perspective

Equation

$$\begin{tikzcd} \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \arrow[d] \\ \frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \end{tikzcd}$$

Solution



The Function-Space Picture

$$\text{Consider } \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

The Function-Space Picture

$$\text{Consider } \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

- Represent solution infinite-dimensional as vector in Hilbert space

$$u(t,x) = \sum_{i=0}^{\infty} u_i(t)\phi_i(x)$$

The Function-Space Picture

$$\text{Consider } \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

- Represent solution infinite-dimensional as vector in Hilbert space

$$u(t,x) = \sum_{i=0}^{\infty} u_i(t)\phi_i(x)$$

- Because computer has finite memory, restrict to finite-dimensional subspace:

$$u(t,x) \approx \sum_{i=0}^{N} u_i\phi_i(t,x) \text{ where } N \in \mathbb{N}$$

Finite Differences in the Function Space Picture

- Break total sum into two:

$$u(t,x) = \sum_{i=0}^{N} \sum_{j=0}^{1} u_{ij}(t)\phi_{ij}(x)$$

- Where

$$\begin{cases} \phi_{i0} = m_i x \chi_i(x) \\ \phi_{i1} = b_i \chi_i(x) \end{cases} \quad \text{where } \chi_i(x) = \begin{cases} 1 & \text{if } x \in [x_i, x_i + 1] \\ 0 & \text{otherwise} \end{cases}$$

- Such that $u(t,x)$ between points x_i and x_{i+1} is of the form

$$u = m_i x + b_i$$

Finite Differences in the Function Space Picture

- Break total sum into two:

$$u(t,x) = \sum_{i=0}^{N} \sum_{j=0}^{1} u_{ij}(t)\phi_{ij}(x)$$

- Where

$$\begin{cases} \phi_{i0} = m_i x \chi_i(x) \\ \phi_{i1} = b_i \chi_i(x) \end{cases} \quad \text{where } \chi_i(x) = \begin{cases} 1 & \text{if } x \in [x_i, x_i + 1] \\ 0 & \text{otherwise} \end{cases}$$

- Such that $u(t,x)$ between points x_i and x_{i+1} is of the form

$$u = m_i x + b_i$$

Notice how redundant this is!

Some Better Basis Functions (Orthogonal Functions)

$$\langle \phi_i, \phi_j \rangle = \int_a^b \phi_i(x)\phi_j(x)w(x)dx = \delta_{ij}$$

- A Fourier basis works well for periodic boundary conditions:

$$[a, b] = [-\pi, \pi] \text{ and } w(x) = 1$$

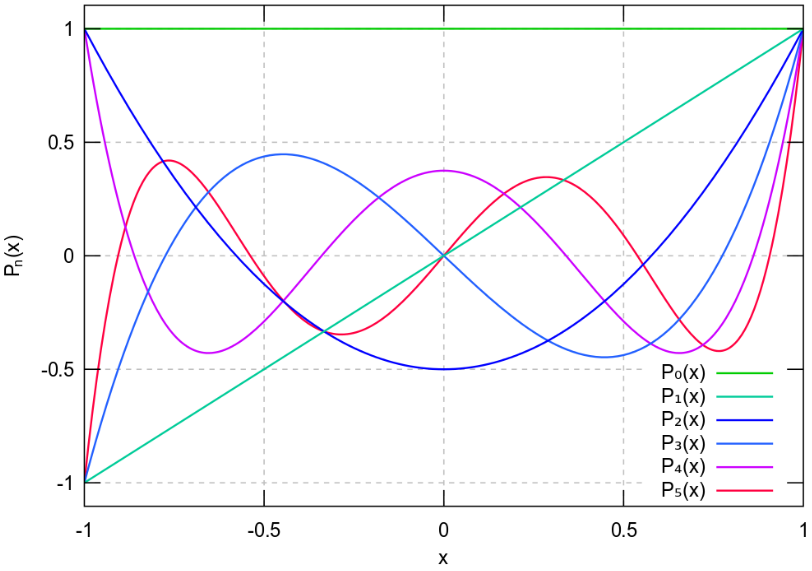
- Chebyshev Polynomials minimize the Gibbs phenomenon and are compatible with all boundary conditions:

$$[a, b] = [-1, 1] \text{ and } w(x) = (1 - x^2)^{\pm 1/2}$$

- Legendre polynomials minimize numerical error when converting a spectral representation to a polynomial interpolation (more later).

$$[a, b] = [-1, 1] \text{ and } w(x) = 1$$

A Legendre Basis



Source: Wikipedia

Precision Quadrature and the Inner Product

Theorem

For all $f \in \mathbb{P}_{2N+\delta}$, there exist $N+1$ positive real numbers x_n in the domain Ω such that:

$$\int_\Omega f(x)w(x)dx = \sum_{n=0}^{N} f(x_n)w_n,$$

where w_n are discrete weights that may be different from the original weight function and where δ is an integer that depends on the precise choice of w_n. (Usually, $\delta \in \{-1, 0, 1\}$.)

Precision Quadrature and the Inner Product

Theorem

For all $f \in \mathbb{P}_{2N+\delta}$, there exist $N+1$ positive real numbers x_n in the domain Ω such that:

$$\int_\Omega f(x)w(x)dx = \sum_{n=0}^{N} f(x_n)w_n,$$

where w_n are discrete weights that may be different from the original weight function and where δ is an integer that depends on the precise choice of w_n. (Usually, $\delta \in \{-1, 0, 1\}$.)

Application

This lets us precisely calculate the inner product between our grid function u and basis elements (or test functions) ϕ_i!

Ensuring "Goodness": The Residual

- A residual $\mathcal{R}$ is a function of the numerical solution that measures how well it solves the original equation. Usually we demand that the residual vanishes.
- E.g., for

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0,$$

In Finite Differences

$$\mathcal{R}_i^n = \frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x}$$

In Galerkin Methods

$$\mathcal{R}_i(t) = \int_a^b \left(\frac{\partial u_h}{\partial t} + c\frac{\partial u_h}{\partial x}\right)\phi_i w(x)dx$$

Recovering A Numerical Scheme (part 1)

- Suppose we want to solve the following equation on the domain $[-1, 1]$:

$$\frac{\partial}{\partial t}u + c\frac{\partial}{\partial x}u(x) = 0$$

Recovering A Numerical Scheme (part 1)

- Suppose we want to solve the following equation on the domain $[-1, 1]$:

$$\frac{\partial}{\partial t}u + c\frac{\partial}{\partial x}u(x) = 0$$

- Then for a given $N \in \mathbb{N}$ we assume our solution is of the form

$$u(t, x) = \sum_{i=0}^{N} u_i \phi_i(x), \text{ where } u_i \in \mathbb{R}$$

Recovering A Numerical Scheme (part 1)

- Suppose we want to solve the following equation on the domain $[-1, 1]$:

$$\frac{\partial}{\partial t}u + c\frac{\partial}{\partial x}u(x) = 0$$

- Then for a given $N \in \mathbb{N}$ we assume our solution is of the form

$$u(t, x) = \sum_{i=0}^{N} u_i \phi_i(x), \text{ where } u_i \in \mathbb{R}$$

- We form a residual and demand that it vanish for all basis functions ϕ_i:

$$\mathcal{R} = \int_{-1}^{1} \left(\frac{\partial u}{\partial t} + c\frac{\partial}{\partial u}x \right) \phi_i(x)dx = 0 \ \forall \ i$$

Recovering A Numerical Scheme (part 2)

- If we plug our ansatz for u into the residual and integrate by parts, we find:

$$\sum_j \mathcal{M}_{ij} \frac{\partial u_j}{\partial t} - c \sum_j \mathcal{S}_{ij} u_j + B_j = 0,$$

where

$$\mathcal{M}_{ij} = \int_{-1}^{1} \phi_i \phi_j dx$$

is the *mass matrix*,

$$\mathcal{S}_{ij} = \int_{-1}^{1} \phi_i \frac{\partial \phi_j}{\partial x} dx$$

is the *stiffness matrix*, and B_j is some vector of boundary terms.

Recovering A Numerical Scheme (part 2)

- If we plug our ansatz for u into the residual and integrate by parts, we find:

$$\sum_j \mathcal{M}_{ij} \frac{\partial u_j}{\partial t} - c \sum_j \mathcal{S}_{ij} u_j + B_j = 0,$$

  where

$$\mathcal{M}_{ij} = \int_{-1}^{1} \phi_i \phi_j dx$$

  is the *mass matrix*,

$$\mathcal{S}_{ij} = \int_{-1}^{1} \phi_i \frac{\partial \phi_j}{\partial x} dx$$

  is the *stiffness matrix*, and B_j is some vector of boundary terms.
- Invert $\mathcal{M}$ and solve for u_i coefficients to get a system of coupled ODEs that can be integrated forward in time using ODE methods.

A Simple Numerical Example

Open up the *linear-advection* IPython notebook for a simple example where we solve the linear advection equation with a spectral method.

Pseudospectral Methods

A Generalization of Spectral Methods

A Generalization of Spectral Methods

- A “wave” description means spectral methods are generically highly non-local.
- This makes solving nonlinear equations (where local values of the solution matter) very complicated.

A Generalization of Spectral Methods

- A “wave” description means spectral methods are generically highly non-local.
- This makes solving nonlinear equations (where local values of the solution matter) very complicated.
- To solve this problem, we represent the function locally (in a similar way to finite differences) but transform to a spectral representation to take derivatives.

Polynomial Interpolation

- Suppose a set of $N+1$ points $\{x_i\}_{i=0}^{N}$ such that $x_{i+1} > x_i$ and a numerical solution $u(x_i)$ defined on those points

Polynomial Interpolation

- Suppose a set of $N+1$ points $\{x_i\}_{i=0}^{N}$ such that $x_{i+1} > x_i$ and a numerical solution $u(x_i)$ defined on those points
- Then an *interpolating polynomial* is the unique polynomial of order N,

$$p(x) = \sum_{i=0}^{N} a_i x^i$$

such that

$$p(x_i) = u(x_i).$$

Nodal and Modal Representations

Nodal

- Represent $u(t,x)$ as a value on a discrete grid: $u_i = u(x_i)$
- Discrete representation is a vector $\mathbf{u} = [u(x_0), u(x_1), ..., u(x_N)]$
- Use polynomial interpolation to extract global solution on $[x_0, x_N]$
- $\mathcal{R}_i = f(u(x_i))$ can be made to be local to x_i

Modal

- Represent $u(t,x)$ as sum over polynomial basis functions: $u(t,x) = \sum c_i(t)\phi_i(x)$
- Discrete representation is a vector $\mathbf{c} = [c_0, c_1, ..., c_N]$
- $\mathcal{R}_i = f(u, \phi_i)$ is highly nonlocal

Combining Nodes and Modes

- Represent the *same* solution *both* nodally and modally.

Combining Nodes and Modes

- Represent the *same* solution *both* nodally and modally.
- The interpolating polynomial $p_t(x)$ is *unique*

Combining Nodes and Modes

- Represent the *same* solution *both* nodally and modally.
- The interpolating polynomial $p_t(x)$ is *unique*
- Therefore,

$$p_t(x) = u(t, x) = \sum_i c_i \phi_i(x)$$

Combining Nodes and Modes

- Represent the *same* solution *both* nodally and modally.
- The interpolating polynomial $p_t(x)$ is *unique*
- Therefore,

$$p_t(x) = u(t, x) = \sum_i c_i \phi_i(x)$$

- Therefore *there exists a transformation between the nodal and modal representations.*

The Pseudospectral Strategy

- Represent our function and impose boundary conditions nodally
- Transform to the modal representation to take derivatives
- Transform back.

Colocation Points

Resources

In no particular order:

- *Numerical Recipes*, by Press, Teukolsky, Vetterling, and Flannery
- *Scientific Computing: An Introductory Survey*, by Heath
- *Introduction to Spectral Methods*, by Grandclement (arXiv:gr-qc/0609020).
- *Spectral Methods: Algorithms, Analysis, and Applications*, by Shen, Tang, and Wang
- *Spectral Methods in MATLAB*, by Trefethen