A Python Implementation of Chebyshev Functions

Chris Swierczewski [1]
cswiercz@amath.washington.edu

[1]University of Washington
Department of Applied Mathematics

27 May 2010

A Quick Note

To the applied mathematicians: I know Nick Trefethen gave several talks on Chebfuns this year. Don't worry! I hope to discuss some things he didn't talk about:

- why his methods works so well,
- how to actually compute Chebfuns,
- how to actually compute integrals, derivatives, global roots.

Two Key Theorems

Weierstrass Approximation Theorem on $\mathbb{R}$ (1885)

Suppose $f \in C^0[a,b]$. Then $\forall \epsilon > 0, \exists p$ polynomial such that $||f - p||_\infty < \epsilon$ on $[a,b]$.

(Interesting corrolary: $\mathbb{R}[x]$ is dense in $C^0[a,b]$. $\mathbb{Q}[x]$ is dense in $\mathbb{R}[x]$. Therefore $|C^0[-1,1]| = |\mathbb{R}|$.)

Restriction to Degree n Polynomials: $\mathcal{P}_n$

Given $f \in C^0[a,b]$ find $p^* \in \mathcal{P}_n$ such that

$$||f - p^*||_\infty \leq ||f - p||_\infty \quad \text{for all } p \in \mathcal{P}_n$$

Fact: p^* exists, is unique, and goes by the name "best", "uniform", "Chebyshev", or "minimax" approximation to f.

Objective

Main Goal

Given $f : [-1,1] \to \mathbb{R}$, find a polynomial of lowest degree that approximates f to within machine epsilon. That is, find

$$p_{\text{goal}} = \min_{p, \deg(p)} ||f - p||_\infty < \epsilon_{\text{mach}}$$

Trefethen's method: look at polynomials with a particular structure that make derivatives, integrals, and roots easy to compute.
Use these tools:

- Structure #1: Lagrange interpolants
- Structure #2: Chebyshev polynomial expansions

1. Introduction
   - Polynomial Approximations
   - The Problem

2. Chebyshev Polynomials
   - Lagrange Interpolation
   - The Barycentric Formula
   - Chebyshev Polynomials

3. Computational Results in Python
   - The Basic Algorithm
   - Differentiation, Integration, and Root Finding
   - Future Work

Adding Some Structure

Trefethen restricts to the space of Lagrange interpolating polynomials. Here we will show close they get to the minimax polynomial approximation.

Largrange Interpolating Approximations

Let $L_n(f)$ be a Lagrange interpolant of f on n nodes. Then,

$$||f - L_n(f)||_\infty \leq (1 + ||L_n||_\infty)||f - p^*||_\infty$$

where p^* is the minimax polynomial approximation of degree n.

$||L_n||$ depends on the choice of interpolating points:

- Uniform Distribution: $||L_n||_\infty = O\left(\frac{2^{n+1}}{n \log n}\right)$
- Chebyshev Distribution: $||L_n||_\infty = O\left(\log(n+1)\right)$

More Reasons to Use Lagrange Interpolation

Lagrange Interpolants over the Chebyshev points have good convergence properties:

Bounded Variation

If $\partial^k f : [-1, 1] \to \mathbb{R}$ has bounded variation for some $k \geq 1$ then

$$||f - L_n(f)|| = O(n^{-k})$$

Analyticity

If f is analytic in a neigborhood of $[-1, 1]$ then

$$||f - L_n(f)|| = O(C^n)$$

for some $C < 1$.

Conclusions About Accuracy

Theorem

Chebyshev interpolants are *near-best* or *spectrally accurate*.

Spectral accuracy has to do with how well an approximating function converges in a spectral domain. (i.e. Fourier) The theory of Sobolev spaces gives a rich and precise description of why this works. I would love to discuss this, but I won't.

The point: Lagrange interpolants over the Chebyshev points (Chebyshev interpolants) are good polynomial approximations.

- For example: they defeat the Gibbs phenomenon.

Practical Uses of Lagrange Interpolation

Barycentric Formula

Let (x_j, f_j) be a collection of $N+1$ sample points of f. Then the Lagrange interpolant over these points can be written as

$$p(x) = \sum_{j=0}^{N} \frac{w_j}{x - x_j} f_j \Bigg/ \sum_{j=0}^{N} \frac{w_j}{x - x_j}$$

with $w_j = (-1)^j$. (Divide by two for $j = 0, N$.)

Benefits:

- fast ($O(N)$ evaluation operation count),
- numerically stable,
- $O(N)$ "updating" method for grid refinements.

Why Restriction to $[-1, 1]$?

Summary so far: we've seen the benefits of using Lagrange interpolation over the Chebyshev points and how the Barycentric formula allows $O(N)$ evaluation as well as $O(N)$ method of adding Chebyshev points.
Trefethen exploits another structure: consider only $f : [-1, 1] \to \mathbb{R}$

Chebyshev Polynomials

The Chebyshev polynomials $T_n : [-1, 1] \to \mathbb{R}$ are defined:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

Chebyshev Polynomial Expansions

Theorem (Trefethen)

If $f : [-1, 1] \to \mathbb{R}$ is Lipschitz continuous then the Chebyshev expansion

$$g(x) = \sum_{n=0}^{\infty} a_n T_n(x)$$

converges absolutely and uniformly to f. Additionally, the Chebyshev coefficients are given by

$$a_n = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_n(x)}{\sqrt{1 - x^2}} dx$$

with the special case that for $n = 0$ the constant changes to $1/\pi$.

Relating Chebyshev Polynomials to Chebyshev Interpolants

Chebyshev Polynomial Representations

For $x \in [-1, 1]$ define $\theta = \arccos(x) \in [0, 2\pi]$ and $z = e^{i\theta}$. Then:

$$T_n(x) = \cos n\theta = \frac{1}{2}\left(z^n + z^{-n}\right)$$

For Chebyshev interpolants, this is a finite sum:

Expansion Representation

Let p be a Chebyshev interpolant determined by $N+1$ grid points (x_i, f_i) over the Chebyshev points $x_i = \cos(\pi i/N)$. Then

$$p(x) = \sum_{n=0}^{N} a_n T_n(x) = \sum_{n=0}^{N} a_n \cos(n\theta) = \frac{1}{2}\sum_{n=0}^{N} a_n(z^n + z^{-n})$$

Definition of Chebfun

Definition

A **chebfun** is a minimal degree Barycentric Lagrange interpolant of a function $f : [-1, 1] \to \mathbb{R}$ over the Chebyshev points that uses its Chebyshev polynomial expansion for fast computations.

- "Spectrally optimal approximate"
- Next section: Given $f \in Lip[-1, 1]$, compute its Chebfun, p.

Why Python?

- Python is a popular, powerful, and easy to use programming language. Moreover, it's open-source and free!
    - Numpy/Scipy
    - Sage
    - Clawpack
    - Brian
- Support open-source! (`http://www.opensource.org`)

Generating Chebyshev Functions

- Init: begin with $N = 4$ Chebyshev interpolating points, $\{x_i\}$ and compute $f_i = f(x_i)$.
- Loop: Compute the discrete cosine transform of the $\{f_i\}$, $\{\hat{f}_i\}$, and divide each term by the number of interpolating points, N. Set $a_i = \hat{f}_i / N$.
- **If:**

$$|a_N|, |a_{N-1}| < 2 * \epsilon_{\text{mach}} * \max_i |a_i|$$

  then truncate the sequence $\{a_i\}$ down to the term with largest index M exceeding this bound. The remaining M terms gives you the "optimal Chebyshev interpolant size.
- **Else:** Loop with $2N$.

Demo

Integration

Chebyshev Polynomial Integration

$$\int_{-1}^{1} T_n(x)dx = \begin{cases} 0 & \text{if } n \text{ odd} \\ \frac{2}{1-n^2} & \text{if } n \text{ even} \end{cases}$$

Integral of a chebfun on $[-1, 1]$:

Clenshaw-Curtis Quadrature

Given a chebfun $p(x) = \sum_{n=0}^{N} a_n T_n(x)$,

$$\int_{-1}^{1} p(x)dx = \sum_{n \text{ even}}^{N} \frac{2}{1-n^2}$$

Differentiation

Chebyshev Polynomial Derivative

If $p(x) = \sum_{n=0}^{N} a_n T_n(x)$ then $p'(x) = \sum_{n=0}^{N-1} b_n T_n(x)$ with

$$b_{n-1} = b_{n+1} + 2na_n, \quad b_N = b_{N+1} = 0, \quad b_0 = b_2/2 + a_1.$$

- Backsolve to find b_n and inverse cosine transform to find (x_i, f_i) pairs.
- $O(N)$ computation of function derivative.

Root Finding

Recall that Chebyshev coefficients $\{a_n\}$ are the coefficients of the Laurent series

$$q(z) = \sum_{n=0}^{N} a_n \left(z^n + z^{-n}\right).$$

- The roots, z_i of q, are the roots of $z^N q(z)$; a polynomial of degree $2N$ in z.
- Find the roots of $z^N q(z)$ using your favorite polynomial root-finding algorithm. (Much easier than for a general function.)
- Roots come in pairs $\{z_n, z_n^{-1}\}$. Project back down to the real axis with $\{z_n, z_n^{-1}\} \mapsto \frac{1}{2}(z_n + z_n^{-1}) = x_n$.
- Toss out any "roots" $x_n \notin [-1, 1]$.

Demo

Future Work

The real power of Chebyshev functions is in creating differential operators and solving differential equations: given $f \in L^2[-1,1]$ solve

$$Lu = f.$$

- Reduces to a dense $(N \times N)$ matrix solve.
- Distinction from finite difference methods: spectral accuracy is preserved! This means $O(\epsilon_{\text{mach}})$ accurate solutions, u.
- You can implement this! (See next slide.)

Future Work

The obvious issue: don't want to compete with Trefethen et. al.'s work. (Unless I develop an awesomer algorithm!) Possible solutions:

1. parallel projects in Python/Sage and Matlab,
2. a core C/FORTRAN library with separate Python and Matlab interfaces. (A la LAPACK.)

Current Python implementation is an excellent platform for new Python programmers. Learn Python and Mercurial! Get involved!

http://code.google.com/p/pychebfun

Thank You