Chebyshev Spectral Methods

- For problems that cannot easily be transformed into periodic-BC problems, spectral methods can still be very attractive but require a nonperiodic set of basis fns. that allow for arbitrary endpoint values.

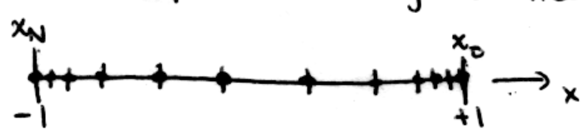
Reference: Trefethen, N: *Spectral Methods in Matlab*, SIAM Press

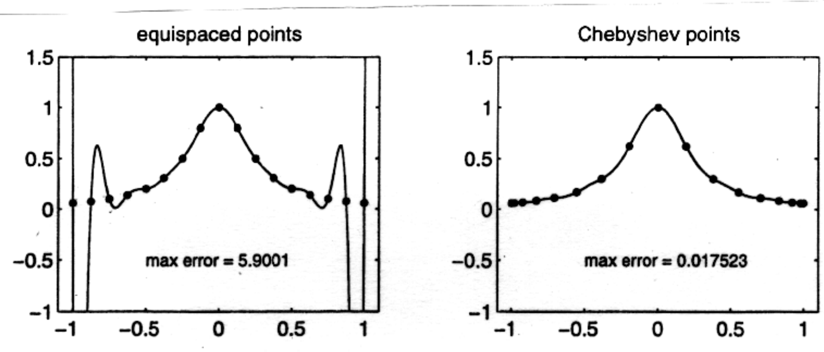
- We might instead try to compute q_x by approximating q as a high-order polynomial in x and finding its derivs at the gridpoints. The obvious choice would be evenly-spaced gridpoints. However, this is susceptible to Runge instability for some smoothly varying $q(x)$'s, in which as the interpolating polynomial attains higher degree N, it may oscillate wildly between gridpoints. A classic example is:

$$q(x) = \frac{1}{1+16x^2} \quad ; -1 < x < 1 \quad \text{interpolated on grid } x_j = \frac{2j}{N},\ j = -\frac{N}{2}, \dots, \frac{N}{2}.$$

- Polynomial interpolation with a higher density of interpolating points near the bdries is much better behaved. An elegant theory we'll later discuss (Fornberg, Ch 3) suggests that nodal spacing $x_j = -1 + c\left(\frac{j}{N}\right)^2$, $j \ll N$ and $x_j = 1 - c\left(\frac{N-j}{N}\right)^2$, $N-j \ll N$ is optimal. One form of efficiently implementable polynomial interpolation with this clustering is Chebyshev interpolation using the Chebyshev points:

$x_j = \cos \frac{j\pi}{N}$, $j = 0, \dots, N$.





Output 9: *Degree N interpolation of $u(x) = 1/(1+16x^2)$ in $N+1$ equispaced and Chebyshev points for $N = 16$. With increasing N, the errors increase exponentially in the equispaced case—the Runge phenomenon—whereas in the Chebyshev case they decrease exponentially.*

This is a bit surprising – why shouldn't equispaced points work best? Essentially this is because closer spacing near the boundary controls the wild oscillations of the highest-order polynomials, which are magnified near domain edges.

Chebyshev interpolation

N'th order polynomial interpolation thru N+1 points (x_j, Q_j) can be expressed

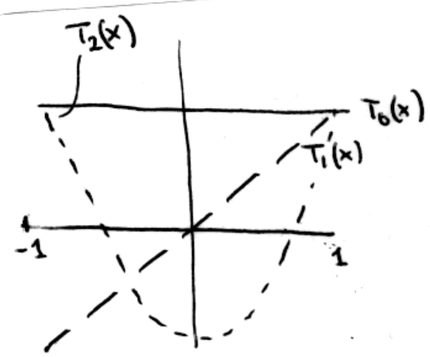
$$Q^N(x) = \sum_{n=0}^{N} \tilde{q}_n T_n(x)$$

where $T_n(x)$ are the Chebyshev polynomials, $x_j = \cos\frac{j\pi}{n}$ are in reverse order with $x_0 = 1$, $x_N = -1$:

$$T_n(x) = \cos\{n \cos^{-1} x\}$$

or, with $x = \cos\theta$

$$T_n(\cos\theta) = \cos(n\theta)$$



Thus

$$T_0(\cos\theta) = 1 \quad \Rightarrow T_0(x) = 1$$

$$T_1(\cos\theta) = \cos\theta \quad \Rightarrow T_1(x) = x$$

$$T_2(\cos\theta) = \cos 2\theta = 2\cos^2\theta - 1 \quad \Rightarrow T_2(x) = 2x^2 - 1$$

etc.

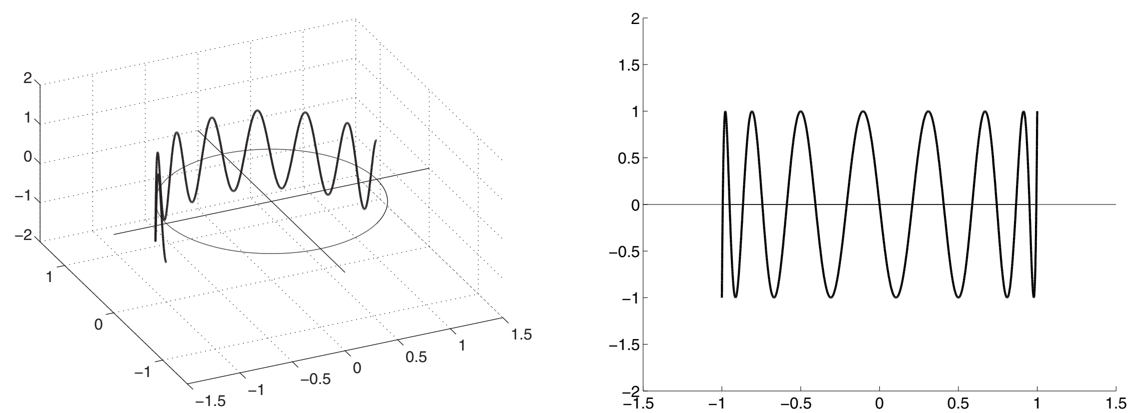


Figure B.2. *The Chebyshev polynomial viewed as a function $C_m(\theta)$ on the unit disk $e^{i\theta}$ and when projected on the x-axis, i.e., as a function of $x = \cos(\theta)$. Shown for $m = 15$.*

The beauty of this is exposed by changing variables: $x = \cos\theta$, so

$$Q^N(x) = q^N(\theta) = \sum_{n=0}^{N} \tilde{q}_n T_n(\cos\theta) = \sum_{n=0}^{N} \tilde{q}_n \cos n\theta \,, \quad 0 \le \theta \le \pi \qquad (T)$$

In particular

$$Q^N(x_j) = Q_j = \sum_{n=0}^{N} \tilde{q}_n \cos n\theta_j \quad , j = 0, .., N$$

- The $\{\tilde{q}_n\}$ can be deduced from $\{Q_j\}$ using even extension to the domain $[0, 2\pi]$ and an appropriate DFT, as we'll see shortly
- We can use this interpolation formula to differentiate $Q^N(x)$ at the $\{x_j\}$:

$$\frac{dQ^N}{dx}(x_j) = \frac{d\theta}{dx} \cdot \frac{dq^N}{d\theta}(\theta_j) = \frac{1}{\sin\theta_j}\left\{\sum_{n=0}^{N}\left[-n\tilde{q}_n \sin n\theta_j\right]\right\}$$

Again the sum is efficiently evaluated by a DFT on $[0, 2\pi]$.

At the boundary points $x_0 = 1$ and $x_N = -1$, $\sin\theta_j = 0$ and this differentiation formula is indeterminate. Instead we note that at $x = 1$ $(\theta = 0)$

$$\frac{dT_n}{dx} = \lim_{\theta\to 0} \frac{d\theta}{dx} \cdot \frac{d}{d\theta} \cos n\theta = \lim_{\theta\to 0} \frac{n \sin n\theta}{\sin\theta} = \lim_{\theta\to 0} \frac{n^2\theta}{\theta} = n^2$$

and similarly at $x = -1$,

$$\frac{dT_n}{dx}(-1) = (-1)^n n^2$$

Thus

$$\frac{dQ^N}{dx}(\pm 1) = \sum_{n=0}^{N} \tilde{q}_n \cdot n^2 \cdot (\pm 1)^n$$

Chebyshev spectral differentiation via FFT

- Given data $v_0, \ldots, v_N$ at Chebyshev points $x_0 = 1, \ldots, x_N = -1$, extend this data to a vector V of length $2N$ with $V_{2N-j} = v_j$, $j = 1, 2, \ldots, N-1$.
- Using the FFT, calculate

$$\hat{V}_k = \frac{\pi}{N} \sum_{j=1}^{2N} e^{-ik\theta_j} V_j, \qquad k = -N+1, \ldots, N.$$

- Define $\hat{W}_k = ik\hat{V}_k$, except $\hat{W}_N = 0$.
- Compute the derivative of the trigonometric interpolant Q on the equispaced grid by the inverse FFT:

$$W_j = \frac{1}{2\pi} \sum_{k=-N+1}^{N} e^{ik\theta_j} \hat{W}_k, \qquad j = 1, \ldots, 2N.$$

- Calculate the derivative of the algebraic polynomial interpolant q on the interior grid points by

$$w_j = -\frac{W_j}{\sqrt{1-x_j^2}}, \qquad j = 1, \ldots, N-1,$$

with the special formulas at the endpoints

$$w_0 = \frac{1}{2\pi} \sum_{n=0}^{N}{}' \, n^2 \hat{v}_n, \qquad w_N = \frac{1}{2\pi} \sum_{n=0}^{N}{}' \, (-1)^{n+1} n^2 \hat{v}_n,$$

where the prime indicates that the terms $n = 0, N$ are multiplied by $\frac{1}{2}$.

Review of DFT – Chebyshev relationships

Chebyshev	**Fourier**
$x = \cos\theta$, $1 \geq x \geq -1$	θ, $0 \leq \theta \leq \pi$ (even extn to $0 \leq \theta \leq 2\pi$)
$T_n(x)$	$\cos n\theta$
Chebyshev points $x_j = \cos\theta_j$, $j=0,\ldots,N$	$\theta_j = j\pi/N$, $j=0,\ldots, 2N-1$
$Q^N(x) = \sum_{n=0}^{N} \hat{q}_n T_n(x)$	$q^N(\theta) = \sum_{n=0}^{N} \hat{q}_n \cos n\theta$
$dQ^N(x)/dx$; can take any value at $x = -1,1$	$-(1/\sin\theta)dq^N/d\theta$; $dq^N/d\theta = 0$ at $\theta = 0, \pi$
Polynomial interpolation/differentiation	Fourier interpolation/differentiation

This algorithm can be viewed as an N+1 x N+1 **derivative matrix** D^N operating on the vector of function values Q_j at the Chebyshev points to give the derivative at those points to spectral accuracy.

With some work, the elements of D^N can be explicitly computed (`Dcheb.m`). Multiplication by D^N is less computationally efficient than using the DFT, but it is conceptually easy, fast enough to use for large enough N to achieve high accuracy for smooth problems, and flexible for setting up the solution of two-point BVPs.

We won't do the algebra, but if $c_k = \begin{cases} 2 & , k=0 \text{ or } N \\ 1 & , 1 \le k \le N-1 \end{cases}$, then:

$$(D^N)_{kj} = \begin{cases} \frac{2N^2+1}{6} & k=j=0 \\ -\frac{2N^2+1}{6} & k=j=N \\ -\frac{x_j}{2(1-x_j^2)} & 1 \le k=j \le N-1 \\ \frac{c_k}{c_j}\frac{(-1)^{k+j}}{x_k - x_j} & k \ne j \end{cases}$$

This is a full, non-normal, singular matrix. The function cheb.m (Trefethen, on class WWW page) computes D^N for any specified N.

Another way to compute the derivative matrix is by remembering that the Chebyshev interpolating function is the unique N'th order polynomial that passes through the given values Q_j at the vector **x** of the Chebyshev points, so the form of the p'th derivative at each of those points x_j can be computed using `fdcoeffF(p,x(i),x)`. This approach is used in RJL's example `BVP_spectral.m`

Application of Chebyshev differentiation

Matlab scripts on Class WWW page, from Trefethen:

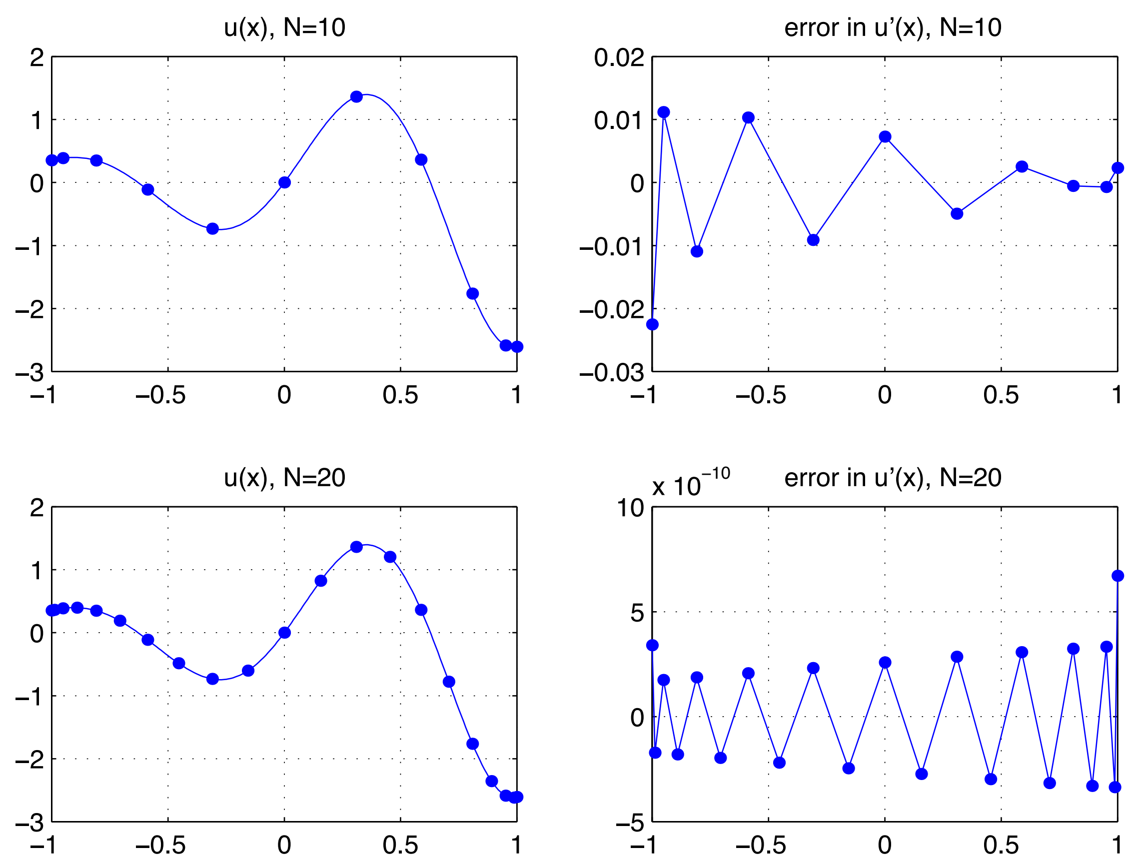
p11.m - Example of Chebyshev-based differentiation, showing its extraordinary accuracy (which is a major attraction)

p13.m - Solution of a 1D BVP with Dirichlet BCs.

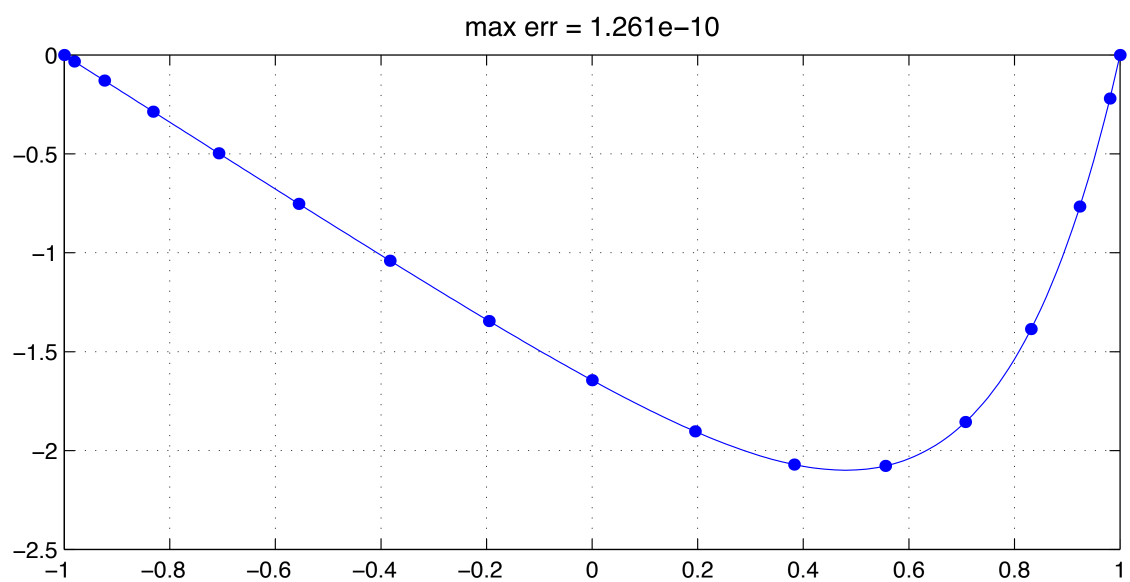
$$q'' = e^{4x}, \quad -1 < x < 1 \qquad (*)$$
$$q(-1) = q(1) = 0.$$

Here we let Q_j, $j=0,..,N$ be the approximate soln at the Chebyshev points $x_j = \cos\frac{j\pi}{N}$. Then $(*)$ is approximated

$$(D^N)^2 \vec{Q} = \vec{f}, \quad f_j = f(x_j) = e^{4x_j}$$



Output of p11.m, showing accuracy of Chebyshev method for differentiation of a smooth function



Output of p13.m, showing Chebyshev solution (N=16) of $u'' = e^{4x}$, $u(-1) = u(1) = 0$.

The BCs are implemented by replacing the rows of $(D^N)^2$ corresponding to the two boundary points $x=1$ (0) and $x=-1$ (N) by the relevant BC. Defining

$$\begin{bmatrix} 1 \quad 0 \quad \cdots \quad 0 \\ \hline \\ (D^N)^2_{kj}, \ 1 \le k \le N-1 \\ \\ \hline 0 \quad \cdots \quad 0 \quad 1 \end{bmatrix} \begin{bmatrix} Q_0 \\ \vdots \\ \\ Q_N \end{bmatrix} = \begin{bmatrix} q(1) \\ \hline f_1 \\ \vdots \\ f_{N-1} \\ \hline q(-1) \end{bmatrix}$$

Letting $\tilde{L}$ be the $(N-1)\times(N-1)$ matrix

$$(\tilde{L})_{kj} = (D^N)^2_{kj}, \quad 1 \le k, j \le N-1$$

we then solve

$$\tilde{L} \begin{bmatrix} Q_1 \\ \vdots \\ Q_{N-1} \end{bmatrix} = \begin{bmatrix} f_1 - [(D^N)^2]_{10}\, q(1) - [(D^N)^2]_{1N}\, q(-1) \\ \vdots \\ f_{N-1} - [(D^N)^2]_{N-1,0}\, q(1) - [(D^N)^2]_{N-1,N}\, q(-1) \end{bmatrix}$$

In this case, $q(1) = Q_0 = 0$, so:
$q(-1) = Q_N = 0$:

$$\tilde{L} \begin{bmatrix} Q_1 \\ \vdots \\ Q_{N-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

We use a standard matrix solve. If N is large this would be inefficient and we would set the problem up using the Chebyshev coefficients $\tilde{q}_n$ instead. Note indices are offset by 1 in script since 0 indices not allowed in Matlab.

p37.m $\qquad q'' = 4x, \quad q'(-1) = q(1) = 0$

In this case, $Q_0 = q(1) = 0$ is known but $Q_N = q(-1)$ is not. We replace the row $k=N$ with the derivative BC, implemented as (row N of derivative matrix D^N) $\cdot \begin{pmatrix} Q_0 \\ \vdots \\ Q_N \end{pmatrix} = q'(-1) = 0$.

to get

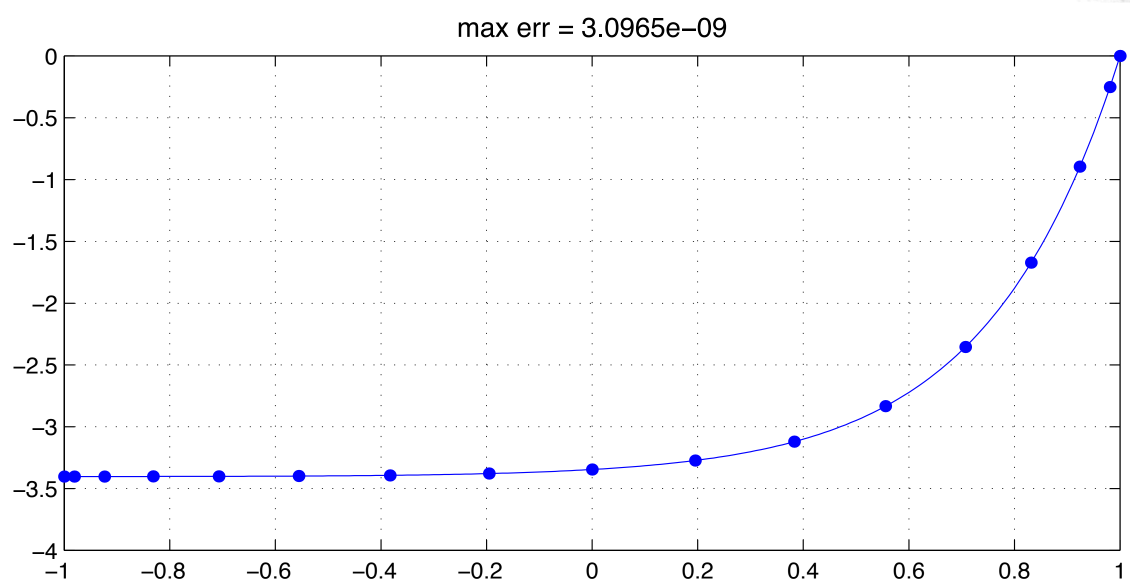
$$\begin{bmatrix} 1 \quad \cdots\cdots \quad 0 \\ \hline \\ \left[(D^N)^2\right]_{kj}, \; 1 \le k \le N-1 \\ \\ \hline \left[D^N\right]_{Nj} \end{bmatrix} \begin{bmatrix} Q_0 \\ \vdots \\ \vdots \\ Q_N \end{bmatrix} = \begin{bmatrix} q(1)=0 \\ \hline f_1 \\ \vdots \\ f_{N-1} \\ \hline q'(-1)=0 \end{bmatrix}$$

Eliminating Q_0, we actually solve the $N \times N$ system

$$\tilde{L} \begin{bmatrix} Q_1 \\ \vdots \\ Q_N \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_{N-1} \\ 0 \end{bmatrix}$$

where

$$\left[\tilde{L}\right]_{kj} = \begin{cases} \left[(D^N)^2\right]_{kj} & 1 \le k \le N-1 \\ \left[D^N\right]_{Nj} & k = N \end{cases}, \quad 1 \le j \le N.$$



Output of p13.m, showing Chebyshev solution ($N=16$) of $u'' = e^{4x}$, $\quad u'(-1) = u(1) = 0$.