Function Approximation

Quantitative Macroeconomics

Raül Santaeulàlia-Llopis

MOVE-UAB and Barcelona GSE

Fall 2018

1. Introduction

2. Local Methods: Taylor Expansion

3. Global Methods
   - Discrete Methods
   - Continuous Methods
     - Spectral Methods: Polynomial Interpolation
     - Finite Element Methods: Piecewise Polynomial Splines

Typical Function Approximation Problem

1. **Interpolation** problem: approximate an analytically intractable real-valued f with a computationally tractable $\widetilde{f}$, given limited information about f.

2. **Functional Equation** problems such as:

   - solve for f in a *functional-fixed* point problem

$$Tf = f$$

   - or solve for f in

$$D(f) = 0$$

Local Methods: Taylor Expansion

- **Taylor's Theorem**: Let $f : [a, b] \to \Re$ be a $n+1$ times continuously differentiable function on (a, b), let $\overline{x}$ be a point in (a, b). Then

$$\begin{aligned} f(\overline{x}+h) &= f(\overline{x}) + f^1(\overline{x})\frac{h^1}{1!} + f^2(\overline{x})\frac{h^2}{2!} + \ldots \\ &\quad + f^n(\overline{x})\frac{h^n}{n!} + f^{n+1}(\zeta)\frac{h^{n+1}}{(n+1)!}, \quad \zeta \in (\overline{x}, \overline{x}+h) \end{aligned}$$

- Where f^i is the *i-th* derivative of f evaluated at the point $\overline{x}$.

- The last term is evaluated at an unknown ζ. When we neglect the last term, we say the above formula approximates f at $\overline{x}$ and the approximation error is of order $n+1$. [1]

[1] The error is $\propto h^{n+1}$ with constant of proportionality $C = f^{n+1}(\zeta)\frac{1}{(n+1)!}$.

Some Taylor expansions:

- Exponential Function:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \ldots \quad \forall x$$

- Infinite Geometric Series:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \quad for |x| < 1$$

- Trigonometric Functions:

$$\sin x = \sum_{n=0}^{\infty} \frac{-1^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} - \frac{x^5}{5!} \ldots \quad \forall x$$

- One definition that we need: A function $f : \Omega \in \mathcal{C} \rightarrow \mathcal{C}$ on the complex plane $\mathcal{C}$ is analytic on Ω iff for every a in Ω, there is an r and a sequence c_k such that $f(z) = \sum_{k=0}^{\infty} c_k \, (z-a)^k$ whenever $||z-a|| < r$. A singularity of f is any point a such that f is analytic on $\Omega - a$ but not on Ω.

- **Theorem 6.1.2** (Judd, 1998). Let f be analytic at $x \in C$. If f or any derivative of f has a singularity at $z \in C$, then the radius of convergence in the complex plane of the Taylor series based at $\overline{x}$, $\sum_{n=0}^{\infty} \frac{f^n(\overline{x})}{n!}(x-\overline{x})^n$, is bounded above by $||\overline{x} - z||$.

This means that,

- Taylor series at $\overline{x}$ cannot reliably approximate $f(x)$ at any point farther away from $\overline{x}$ than any singular point of f (it is the distance that matters not the direction).

- This is important for economic applications, since utility and production functions satisfy an Inada condition, a singularity at some point.

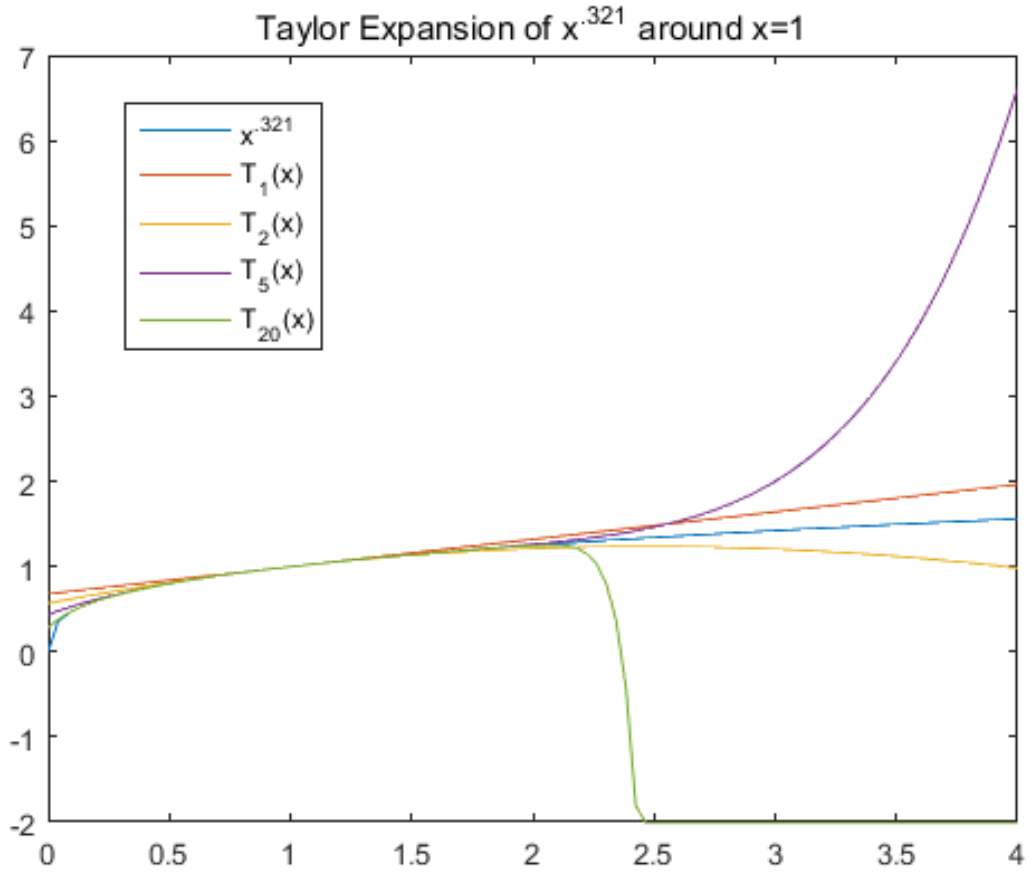
- Example: $f(x) = x^{\alpha}$, for $\alpha \in (0, 1)$, has a singularity at $x = 0$. Hence, if we Taylor approximate f at $\overline{x} = 1$, the approximation error of this approximation increases sharply for $x > 1$ — because the radius of convergence for the Taylor series around one is, in this case, only unity (its distance from the singularity at 0).

[Homework:]

FA.1 Approximate $f(x) = x^{.321}$ with a Taylor series around $\overline{x} = 1$. Compare your approximation over the domain (0,4). Compare when you use up to $1, 2, 5$ and 20 order approximations.

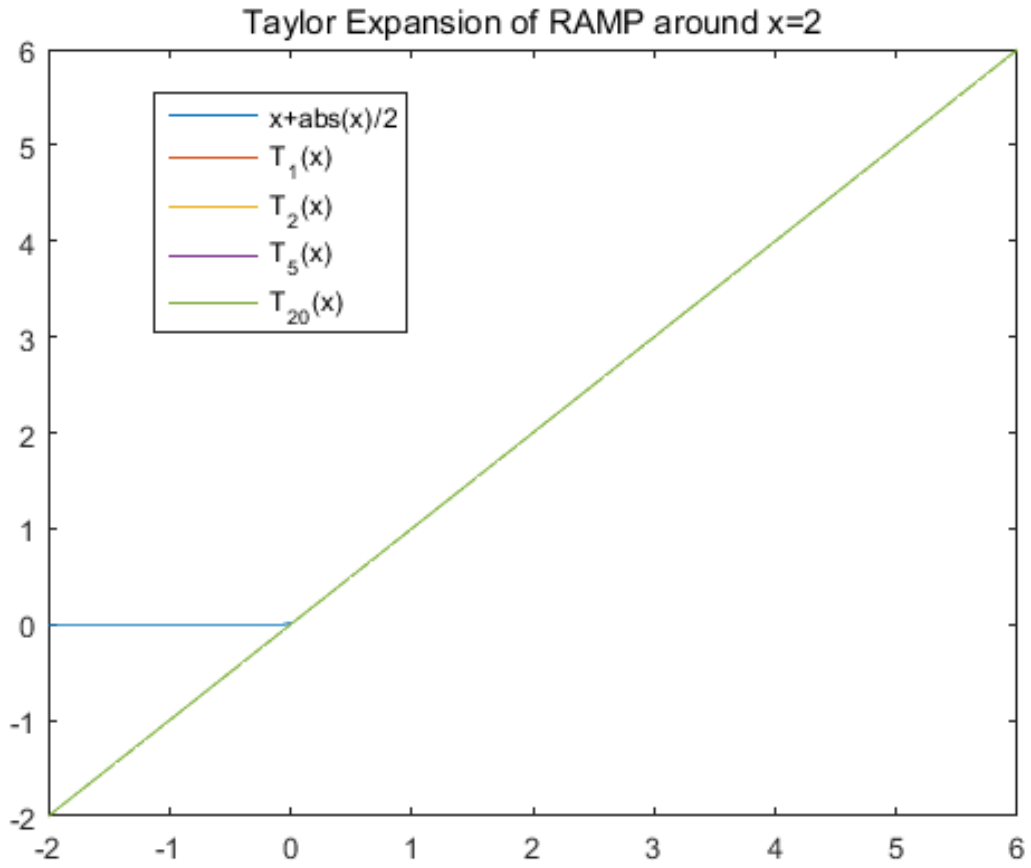
FA.2 Approximate the *ramp function* $f(x) = \frac{x+|x|}{2}$ with a Taylor series around $\overline{x} = 2$. Compare your approximation over the domain (-2,6). Compare when you use up to $1, 2, 5$ and 20 order approximations.

Solution to HWK FA.1



[All credit for this figure goes to Shangdi Hou]

Solution to HWK FA.2



[All credit for this figure goes to Shangdi Hou]

- **Pros:**
  - We can use Taylor for many business cycle models with stand-in households and relatively small shocks.
  - We have many packages (Uhlig, Christiano, Sims, Klein... and a long etc.) that do so very fast. Some packages have the advantage that *log*- or linearize the set of equilibrium conditions (around the steady state) by themselves (Dynare) – we do not need to it by hand any more.

- **Cons:**
  - Cannot deal with big movements in the state space: the economy has to stay at or very close to a steady state.
  - Cannot deal with binding borrowing constraints (kinks) [e.g., as standard in incomplete markets economies] because of the necessary assumption of differentiability.
  - Cannot deal with default or other discontinuities.
  - The list goes on and on...

An alternative formulation of local methods is perturbation (and associated prunning). See Judd 1998 or the recent chapter by Fernandez-Villaverde, Rubio and Schorfheide in the handbook of macro.

Local methods are the usual tool to solve business cycle models (both RBCs and NKs).

Global Methods

1. **Discretization**
2. **Continuous Methods**
   - **Spectral Methods: Polynomial Interpolation**
     - Monomials
     - Orthogonal polynomials: Chebyshev and others.
   - **Finite Element Methods: Piecewise Polynomial Splines**
     - Spline Interpolation: Linear, Quadratic, Cubic Splines and Shape-Preserving Schumaker Splines.
     - Basis Splines (B-Splines).

Discrete Methods

- Discretization implies that we take a continuous state space (domain) and replace it with a discrete one.
- We are in discrete methods: our original continuous function becomes a mapping from a discrete point set into a discrete point set [e.g. a pdf becomes a histogram].

- **Pros:**
  - Very easy and as robust as it gets.
  - Can deal with binding inequality constraints, corner solutions and discontinuities.
  - Increasing the density of the discrete set we can get an arbitrarily good approximation.

- **Cons:**
  - Turtle speed—not good for long-D races. If appropriately combined with known properties of the function we want to approximate (monotonicity, concavity...), we may speed the algorithm but it will still go slow – we will see this in much detail when we approximate the value function of the neoclassical growth model with VFI.
  - Increasing the density of the discrete set in order to get a good approximation increases substantially memory consumption.
  - More than one dimension?. State space increases exponentially. Use of tensor products—one may use some procedure that disciplines the choice of the points on the state space to look at (e.g. Smolyak algorithm).

Continuous Methods

In continuous methods the interpolant $\widetilde{f}(x)$ is assumed continuous over x. To capture this continuous function we use the following interpolation scheme (consistent with the projection methods algorithm):

- FIRST, choose a family of functions $\mathcal{F}$ from which $\widetilde{f}$ can be drawn. For practical we choose the $\mathcal{F}$ that collects functions that can be written as a linear combination of a set of n known linearly independent basis functions ψ_j, $j = 1, ..., n$,

$$\widetilde{f}(x) = \sum_{j=1}^{n} \theta_j \psi_j(x) \tag{1}$$

  with n-basis coefficients θ_j to be determined.

Remark. What are the basis functions?

- A basis function is an element of a particular basis for a function space. Every function in the fucntion space can be represented as a linear combination of basis functions, just as every vector in a vector space can be represented as a linear combination of basis vectors.
- Remember that a vector $\mathbf{x} \in \Re^n$ can be represented as a linear combination of n linearly independent vectors [a]

$$\mathcal{B} := \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\} \text{ of } \Re^n$$

- A basis is a set of vectors that, in a linear combination, can represent every vector in a given vector space or free module, and such that no element of the set can be represented as a linear combination of the others. In other words, a basis is a linearly independent spanning set.
- This way, we say $\mathcal{B}$ builds a base of the vector space $\Re^n$.
- If the members of $\mathcal{B}$ are mutually **orthogonal** (i.e., $\mathbf{v}_i'\mathbf{v}_j = 0$ for $i \neq j$) and normal (i.e., $\mathbf{v}_i'\mathbf{v}_i = 1$) the base is called an **orthonormal** base.

[a] Here, note that like $\Re^n$, the set of continuous functions $\mathcal{C}[a, b]$ is a vector space.

Remark. Interpolation Vs. Approximation

A basic interpolation problem implies that for given data

$$(x_1, y_1), (x_2, y_2), ..., (x_m, y_m) \text{ with } x_1 < x_2 < ... < x_m,$$

we determine the function $f : \Re \to \Re$ such that $\widetilde{f}(x_i) = y_i,\ i = 1, ..., m$. Additional data might be prescribed, such as slope of $\widetilde{f}$ at given points. Additional constraints might be imposed, such as smoothness, monotonicity, or convexity of $\widetilde{f}$.

That is, there is a difference beteween interpolating a function (which by definition fits given data points exactly) and approximating a function (which does not necessarily do so). For example, it might be inappropriate to interpolate data points subject to significant errors being perhaps preferable to smooth noisy data, for example by least squares approximation.

Because it goes through some specific chosen points, interpolation then is closely related to collocation as weighted residual method.

- SECOND, specify the properties of f that one wishes $\widetilde{f}$ to replicate.

  The simplest and most common conditions imposed are that $\widetilde{f}$ *interpolate* or match the value of the original function at selected *interpolation nodes* $x_1, x_2, ..., x_i, ..., x_n$,

$$\widetilde{f}(x_i) = f(x_i)$$

  Given n interpolation nodes (that we know) and n basis functions (that we know), we can compute the n basis coefficients (that we DO NOT know) by solving the **interpolation conditions**:

$$\widetilde{f}(x_i) = \sum_{j=1}^{n} \theta_j \psi_j(x_i) = f(x_i)$$

  See that this is a system of linear equations in θ_j.

- THIRD, we are not limited to the use of point values, but we may also be able to use **first (second or higher order) derivatives (or antiderivatives)** at specified points.

  For example, we can find $\widetilde{f}$ that replicates f at value nodes $x_1, x_2, ..., x_{n_1}$, and its first derivatives at nodes $x_1, x_2, ..., x_{n_2}$.

  Then, we would have to find $\theta_1, \theta_2, ..., \theta_n$, with $n = n_1 + n_2$, that solve

$$\sum_{j=1}^{n} \theta_j \psi_j(x_i) = f(x_i) \quad \forall i = 1, ..., n_1$$

$$\sum_{j=1}^{n} \theta_j \psi_j'(x_h) = f'(x_h) \quad \forall h = 1, ..., n_2$$

Remark. The interpolation problem is linear, but our functional equation problems in economics are (usually) NOT linear.

Note that the interpolation scheme assumes that we know the function that we are interpolating or at least that some exact points (x_i, y_i) of that function (interpolation nodes). That is, the interpolation chooses the coefficients θ_j that make equal to zero the distance between the interpolant and a number m of specific known points (x_i, y_i) (i.e., the interpolation nodes):

$$\sum_{j=1}^{n} \theta_j \psi(x_i) = y_i$$

This is precisely a system of linear equations $A\theta = y$ with unknown parameters $\theta = [\theta_1, ..., \theta_n]$ and where the entries of the $m \times n$ matrix A are given by $a_{ij} = \psi_j(x_i)$.

However, the functional equations in which we are interested,

$$\text{either } Tf = f, \text{ i.e., } T \sum_j \theta_j \psi(x_i) = \sum_j \theta_j \psi(x_i)$$

$$\text{or } D(f) = 0, \text{ i.e., } D\left(\sum_j \theta_j \psi(x_i)\right) = 0,$$

are (usually) NOT linear problems in θ_j. That is, we will require nonlinear solvers.

How do we choose interpolation nodes and basis functions?

Some desirable criteria:

- $\widetilde{f}$ should approach (arbitrary accurate approximation) to f by increasing the number of nodes and basis functions as well as the number of basis (i.e., degree) once a polynomial basis is chosen.
- θ_j should be quickly and accurately computable — diagonal, near-diagonal or orthogonal interpolation matrices are best.
- $\widetilde{f}$ should be easy to work with, that is, basis functions should have the property of being easily evaluated, differentiated and integrated.

Interpolation Nodes

(a) Evenly-spaced interpolation nodes

- n evenly spaced interpolation nodes yield,

$$x_i = a + \frac{i-1}{n-1}(b-a), \quad \forall i = 1, 2, ..., n$$

If there are only two interpolation nodes, then this results in the linear interpolation above.

However, there are smooth functions for which polynomial $\widetilde{f}$ with evenly spaced nodes rapidly deteriorate: a classic example is the *Runge's function* $f(x) = \frac{1}{1+25x^2}$ where the approximation error rises rapidly with the number of nodes.

Interpolation Nodes

(b) Chebyshev interpolation nodes

- Numerical experience suggests polynomial $\widetilde{f}$ over a bounded interval $[a, b]$ should use Chebyshev nodes.

- The Chebyshev nodes are the j roots of the Chebyshev polynomials $\psi_j(x)$ (discussed later) which are actually defined on the $[-1, 1]$ interval. These j roots are:

$$x_j = \cos\left(\frac{2j-1}{2n}\pi\right), \quad \forall j = 1, 2, ..., n$$

- These nodes are not evenly spaced. They are more closely spaced near the endpoints of the interpolation interval and less so near the center—this avoids more easily instabilities at the endpoints.

- We can extend the grid $[-1, 1]$ to any bounded interval $[a, b]$ by linearly transforming the nodes:

$$z_j = \frac{a+b}{2} + \frac{b-a}{2} x_j$$

That is,

$$z_j = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2j-1}{2n}\pi\right), \quad \forall j = 1, 2, ..., n \tag{2}$$

To see this:

- Draw the semicircle on $[a, b]$ centered at the midpoint $\frac{a+b}{2}$
- Select n points and split the semicircle into $n-1$ arcs of equal length. This yields the following figure:

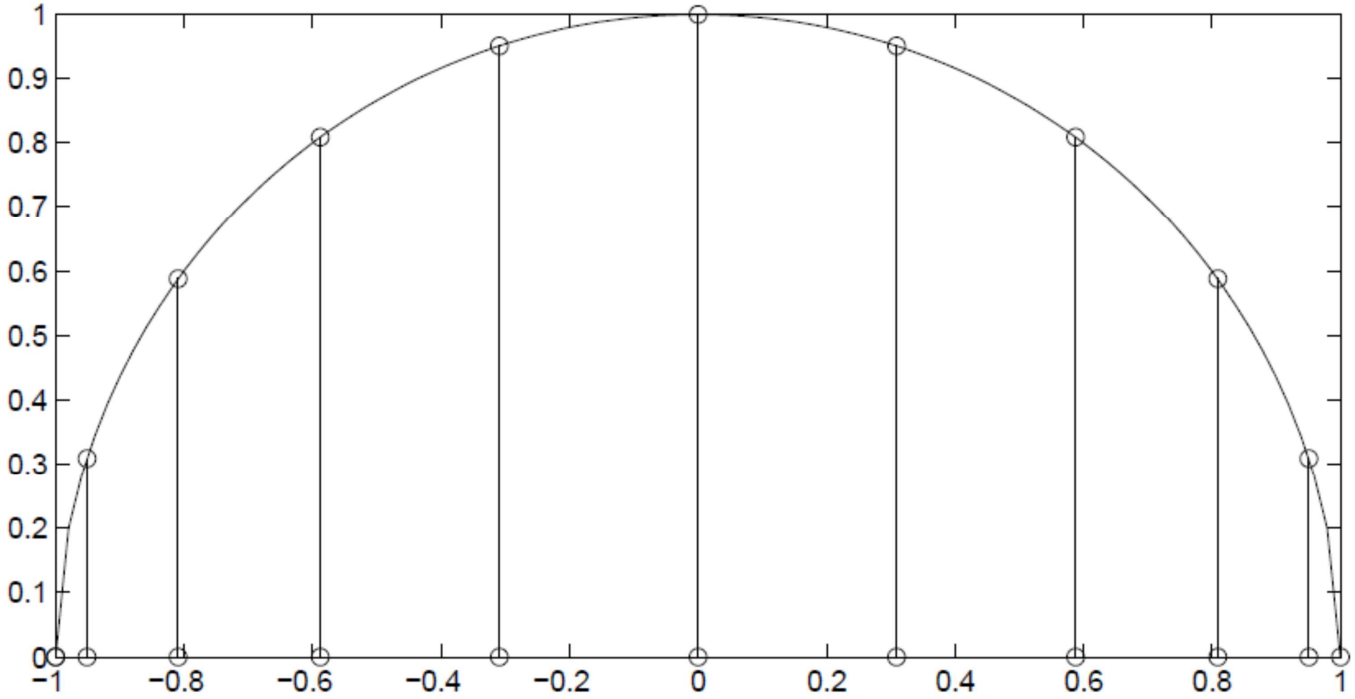


Figure: Chebyshev Nodes

- What equation (2) does is project the arcs in the figure above onto the x-axis.

Two reasons to use Chebyshev nodes:

- **Rivlin's Theorem**: the approximation error of a nth-degree polynomial $\widetilde{f}$ that uses Chebyshev nodes is lower than $(2\pi \log(n) + 2)$ times the lowest error attainable with any other polynomial $\widetilde{f}$ of the same degree that uses alternative nodes.

- **Jackson's Theorem**: the approximation error of a nth-degree polynomial that uses Chebyshev nodes is bounded. Moreover, the error bound goes to zero as n arises. Thus, in contrast with polynomials $\widetilde{f}$ built from evenly spaced nodes, we can achieve any desired degree of accuracy with polynomials $\widetilde{f}$ interpolated at a sufficiently large number of Chebyshev nodes.

  Polynomials $\widetilde{f}$ with Chebyshev nodes exhibit *equioscillant* errors.

Interpolation Basis

We show two types of interpolation methods depending on the choice of the basis functions:

1. **Spectral Methods:** Uses basis functions that are nonzero over the entire domain of f, except possibly at a finite number of points. The most common spectral method are **Polynomial Interpolation**.

2. **Finite Element Methods:** Uses basis functions that are nonzero over subintervals of the approximated domain. The most common finite element method are **Piecewise Polynomial Splines**.

Spectral Methods: Polynomial Interpolation

- **Spectral methods** use basis functions that are nonzero over the entire domain of f, except possibly at a finite number of points.
- The most common spectral method is **polynomial interpolation**.
- We go over several polynomial interpolations:
  - Monomials
  - Orthogonal polynomials: Chebyshev and others.

- **Weirtrass Theorem:** any continuous real-valued function f defined on a bounded interval $[a, b]$ of the real line can be approximated to any degree of accuracy using a polynomial.

  That is, for any $\epsilon > 0$, there is a polynomial $\widetilde{f}$ such that

$$||f - \widetilde{f}||_\infty \equiv sup_{x \in [a,b]} |f(x) - \widetilde{f}(x)| < \epsilon$$

  This theorem motivates the use of polynomials to approximate continuous functions, but it does not tell us which polynomial we should use.

Monomials

- The *monomials* x^i, $i = 1, 2, ...$ build a base $\mathcal{B}$ for the space of functions $\mathcal{C}[a, b]$ and every member of $\mathcal{C}[a, b]$ can be represented by,[2]

$$\widetilde{f}(x) = \sum_{j=1}^{\infty} \theta_j \, x^{j-1}$$

For that reason it is common to use a linear combination of the first n members of this base to approximate continuous functions $\mathcal{C}[a, b]$,

$$f(x) \approx \widetilde{f}(x) = \theta_1 + \theta_2 \, x + \theta_3 \, x^2 + ... + \theta_n \, x^{n-1}$$

[2] Note that f we only have two interpolation nodes ($n = 2$) then we are simply doing linear interpolation. It is simple and preserves concavity and monotonicity. Suppose we want to approximate f at a given point $\overline{x}$ with the property $\overline{x} \in (a, b)$. Linear interpolation uses the point

$$\widetilde{f}(\overline{x}) := f(a) + \frac{f(b) - f(a)}{b - a}(\overline{x} - a) \tag{3}$$

Thus, $f(x)$ is approximated by the line through $(a, f(a))$ and $(b, f(b))$ and the formula given in (3) is a special case of (1) with $n = 2$, $\psi_j(x) = x^{j-1}$, $\theta_1 = \frac{b\, f(a) - a\, f(b)}{b-a}$ and $\theta_2 = \frac{f(b)-f(a)}{b-a}$.

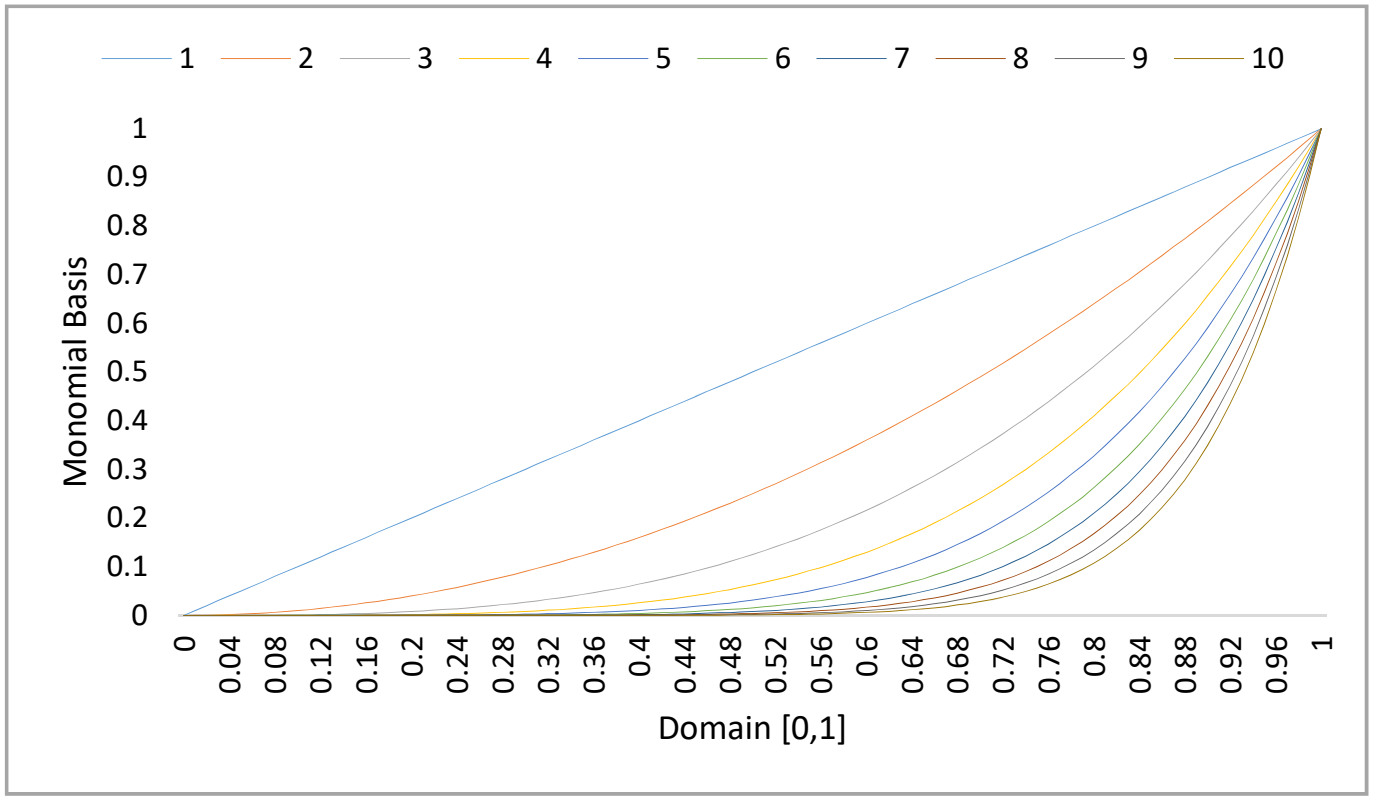


Figure: Monomial Basis (polynomials) from degree 1 to 10

The monomial basis become more similar to each other as we increase the degree of the basis. That is, additional information is marginally less valuable.

Problems of *monomial basis*:

- They pose linear systems of linear equations that are notoriously ill-conditioned (*Vandermonde matrices*):

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & . & . & . & x_1^{n-1} \\ 1 & x_2 & x_2^2 & x_2^3 & . & . & . & x_2^{n-1} \\ 1 & x_3 & x_3^2 & x_3^3 & . & . & . & x_3^{n-1} \\ . & . & . & . & . & & & . \\ . & . & . & . & & . & & . \\ . & . & . & . & & & . & . \\ 1 & x_n & x_n^2 & x_n^3 & & & & x_n^{n-1} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ . \\ . \\ . \\ \theta_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_n \end{bmatrix} \tag{4}$$

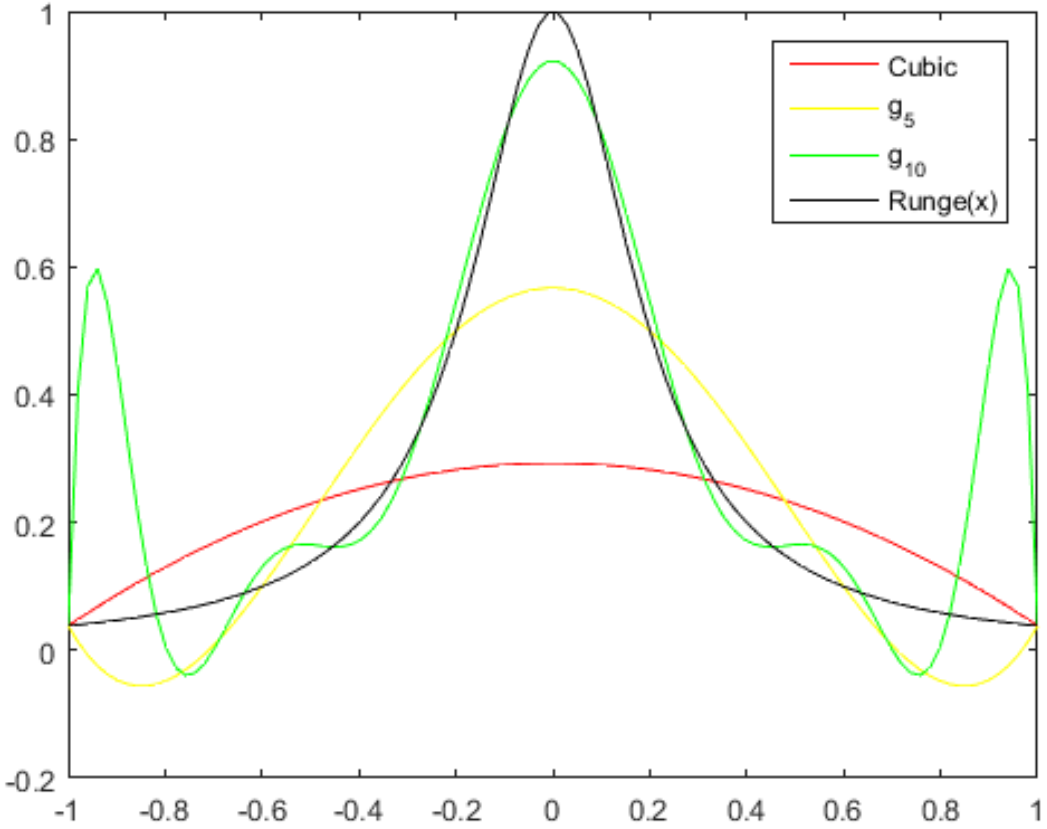
- It is hard to invert these matrices. The efforts to compute the basis coefficients of the monomial basis often fail because of rounding error.
- Attempts to computer more accurate approximations by raising the number of interpolation nodes are often futile.

Why all these problems? Because *monomial basis* are not orthogonal. That is, there is nearly linear dependence among the x^i (multicollinearity) and for large i, x^i and x^{i+1} may be very difficult to distinguish—i.e., adding x^{i+1} to x^i is not informative, much less informative for higher i.

Bases that consists **orthogonal polynomials** circumvent this problem.

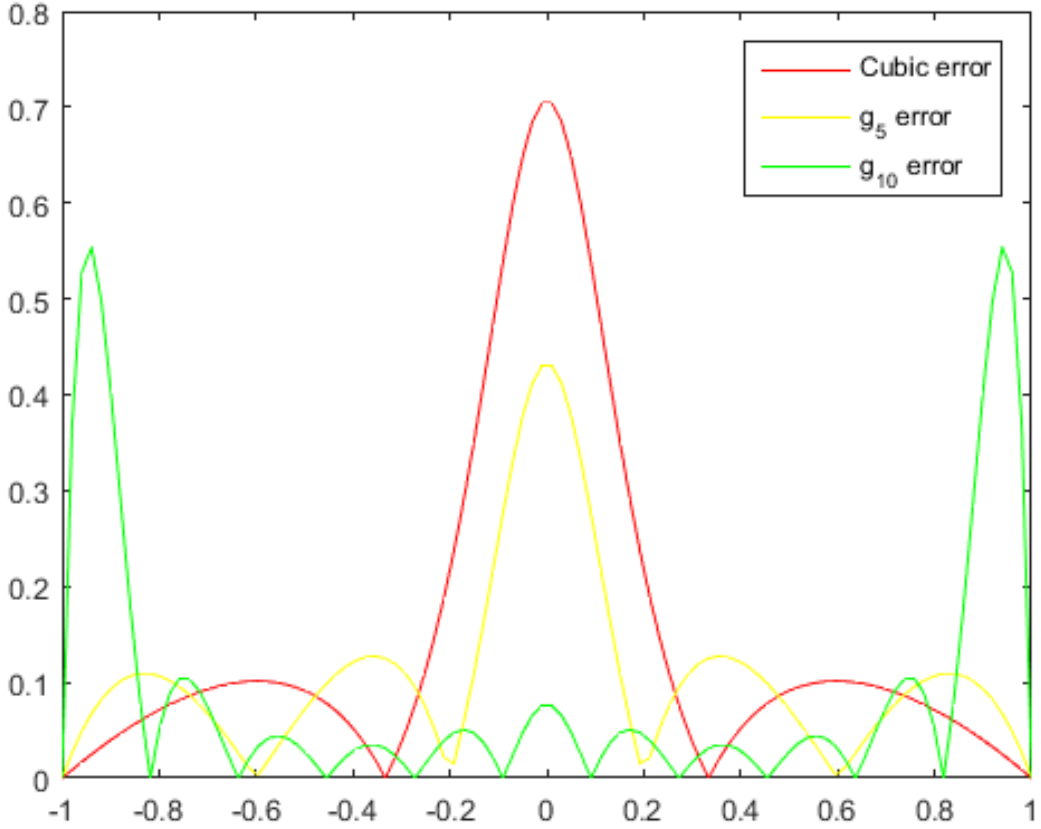
Before discussing orthogonal polynomials let's see how monomials perform with evenly-spaced interpolation nodes for the Runge function.

Runge Function Approximation: Evenly-spaced nodes & monomials



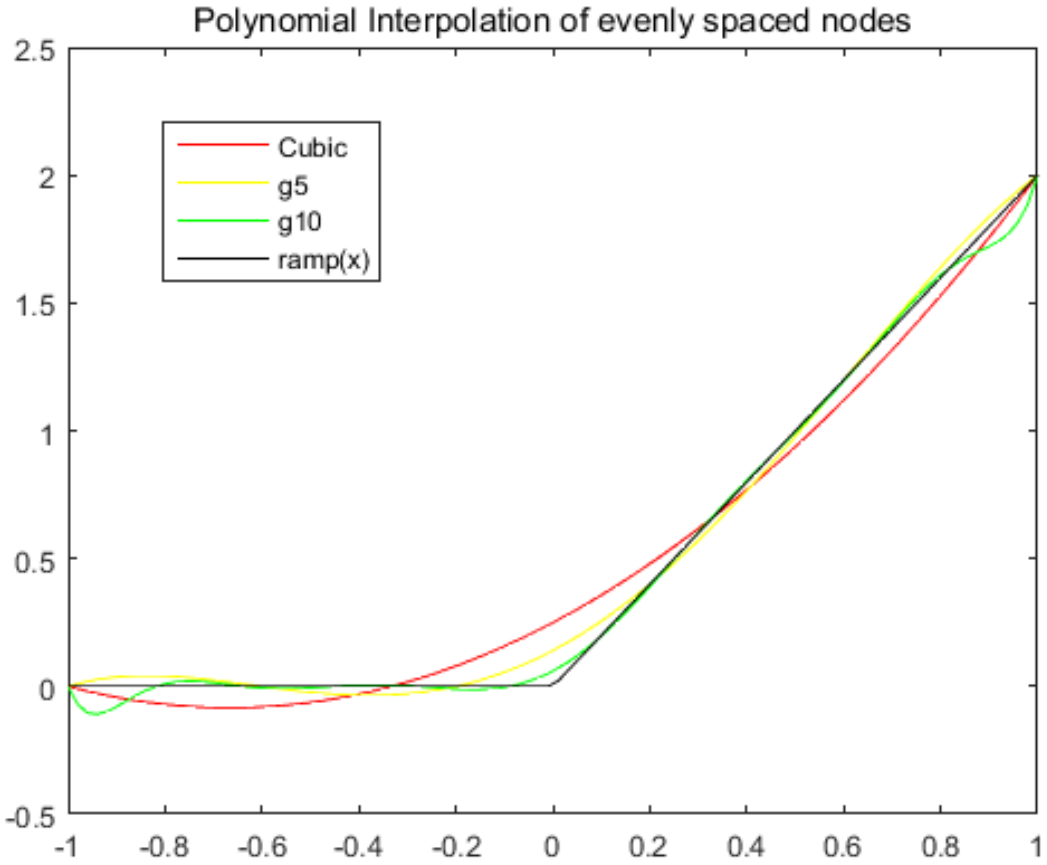
[All credit for this figure goes to Shangdi Hou]

Runge Function Approximation Errors: Evenly-spaced nodes & monomials



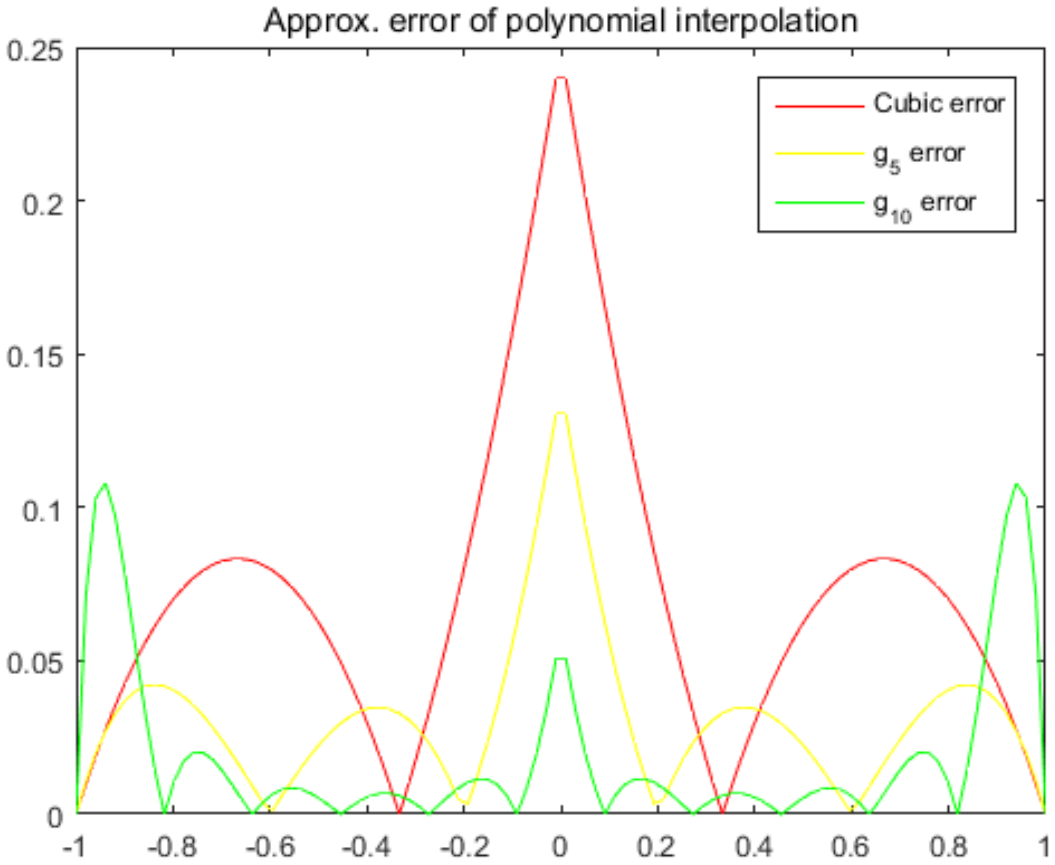
[All credit for this figure goes to Shangdi Hou]

Ramp Function Approximation: Evenly-spaced nodes & monomials



[All credit for this figure goes to Shangdi Hou]

Ramp Function Approximation Errors: Evenly-spaced nodes & monomials



[All credit for this figure goes to Shangdi Hou]

Orthogonal Polynomials

- Inner products can be defined on space of polynomials on interval $[a, b]$ by taking,

$$\langle \psi_j, \psi_k \rangle = \int_a^b \psi_j(x)\psi_k(x)w(x)dx$$

where $w(x)$ is a nonnegative weight function $\int_a^b w(x)dx < \infty$.

- Two polynomials ψ_j and ψ_k are **orthogonal** if $\langle \psi_j, \psi_k \rangle = 0$

- Set of polynomials is **orthonormal** if

$$\langle \psi_j, \psi_k \rangle = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

- Orthogonality is natural for least squares estimation and also useful for generating Gaussian quadrature.

Chebyshev Polynomials

The popularity of Chebyshev polynomials is easily explained if we consider some of their advantages:

1. First, numerous simple closed-form expressions for the Chebyshev polynomials are available. Thus, the researcher can easily move from one representation to another according to her convenience.
2. Second, the change between the coefficients of a Chebyshev expansion of a function and the values of the function at the Chebyshev nodes is quickly performed by the cosine transform.
3. Third, Chebyshev polynomials are more robust than their alternatives for interpolation.
4. Fourth, Chebyshev polynomials are smooth and bounded between $[-1, 1]$.
5. Finally, several theorems bound the errors for Chebyshev polynomials interpolations.

- Chebyshev polynomials (of the first kind) are trigonometric polynomials $\psi_j(x) : [-1, 1] \to [-1, 1]$ defined by

$$\psi_j(x) = \cos(n \ \arccos x)$$

and are orthogonal with respect to the weight function $\left(1 - x^2\right)^{-0.5}$

- While Chebyshev polynomials are defined over the interval $x \in [-1, 1]$, we can easily convert this interval to $[a, b]$ by linearly transforming the data with nodes:

$$z_j = \frac{a+b}{2} + \frac{b-a}{2} x_j$$

- **Recursive Property**: We can compute a Chebyshev polynomial of order $j+1$, if we know the values of the Chebyshev polynomials of order j and $j-1$. This property of the Chebyshev family (shared by Lagerre, Legendre and Hermite polynomials) helps economizing on computational time.

- The recursive scheme is

$$\psi_{j+1}(x) = 2x\psi_j(x) - \psi_{j-1}(x) \text{ for } j > 1$$

- The first four Chebyshev polynomials are,

$$\begin{aligned}
\psi_0(x) &= \cos(0 \arccos x) = cos(0) = 1 \\
\psi_1(x) &= \cos(1 \arccos x) = x \\
\psi_2(x) &= 2x\psi_1(x) - \psi_0(x) = 2x^2 - 1 \\
\psi_3(x) &= 3x\psi_2(x) - \psi_1(x) = 4x^3 - 3x
\end{aligned}$$

- These basis are often written as $T_j(x)$ because of variants of the name Chebyshev as Tchebycheff.

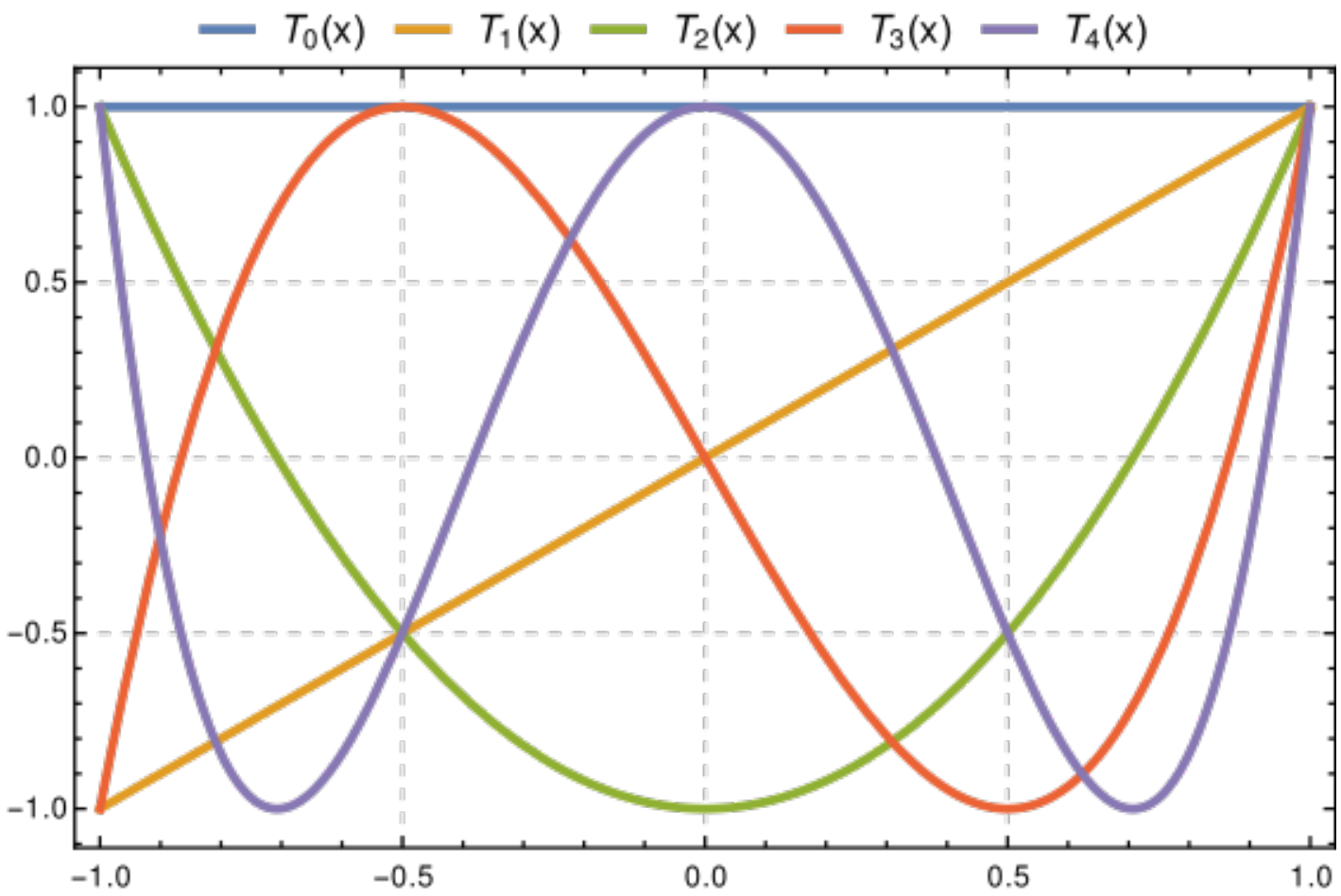


Figure: Chebyshev Basis (polynomials) 0, 1, 2, 3 and 4

[Figure straight from Wiki]

Recall that the Chebyshev nodes are the roots of the Chebyshev polynomials $\psi_j(x)$. Specifically, the Chebyshev polynomial $\psi_j(x)$ has j roots in $[-1, 1]$ given by $x_j = cos\left(\frac{2j-1}{2n}\pi\right)$. Also, note that $\psi_j(x)$ has $j+1$ distinct extrema with value of either -1 or 1.

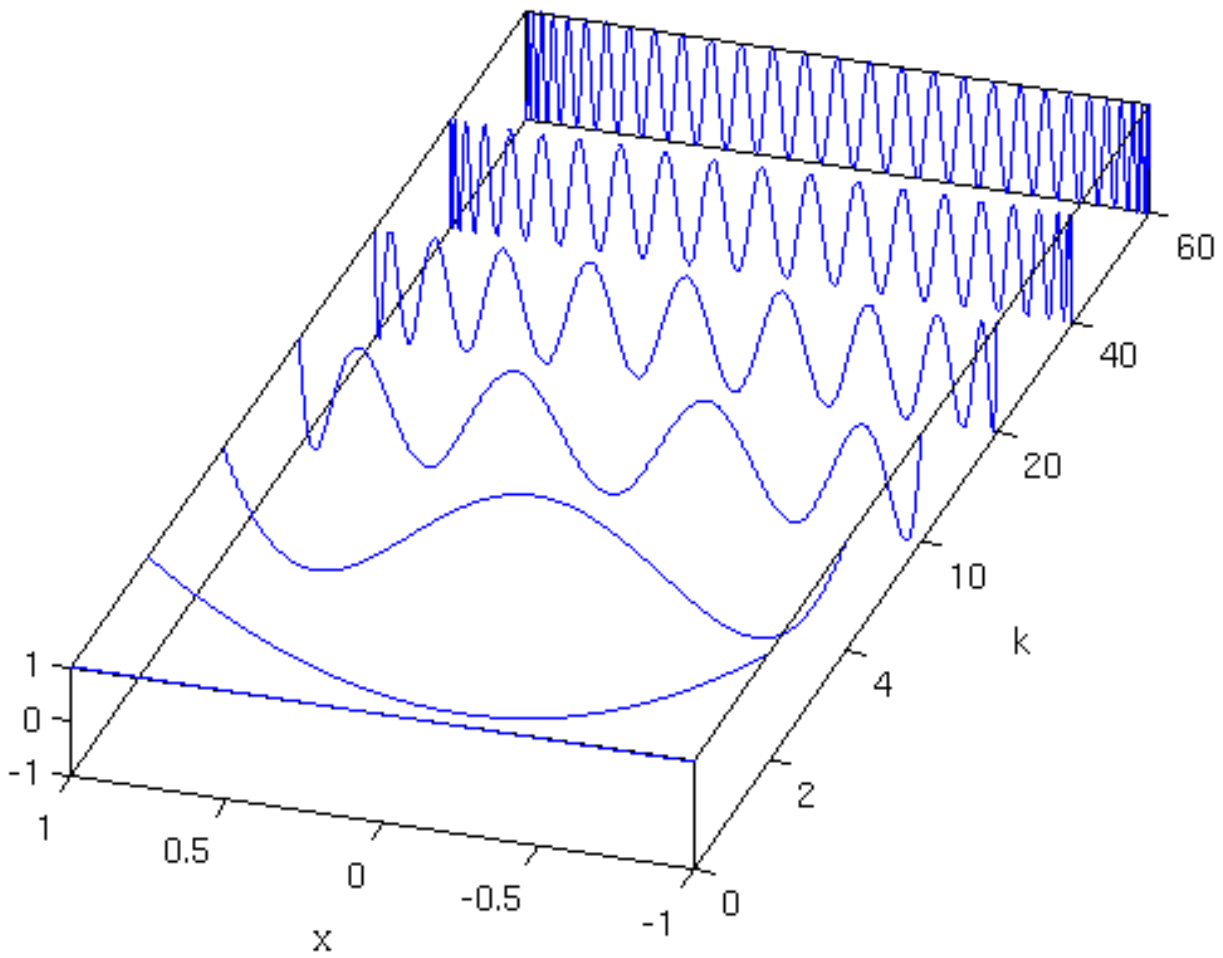


Figure: Chebyshev Basis (polynomials) 0, 2, 4, 10, 20, 40 and 60

[Figure straight from Wiki]

- **Theorem 8.2.3** (Heer and Mauβner (2005)). If $f \in C^k[-1,1]$ has a Chebyshev representation $f(x) = \sum_{j=1}^{\infty} \theta_j \psi_j(x)$, then there is a constant c such that
$$|\theta_j| \leq \frac{c}{j^k}, \;\; j \geq 1$$

- Coefficients $\{\theta_j\}$ of Chebyshev polynomials are strictly decreasing in the order of polynomials and we can confidently ignore higher order basis.

Some remarks:

- Chebyshev basis polynomials with Chebyshev interpolation nodes yield an extremely well-conditioned interpolation equation that can be accurately and efficiently solved, even with high-degree approximants (as opposed to monomials with high degree).[3]
- One can show that the interpolation nodes that minimize the error of the Chebyshev interpolation are the zeros of the Chebyshev polynomials themselves.

[3] The Chebyshev interpolation matrix is orthogonal with Euclidean norm condition number $2^{\cdot 5}$, regardless of the degree of interpolation, very close to the absolute minimum of 1. This means that Chebyshev basis coefficients, θ_j, can be computed quickly and accurately, regardless of the degree of interpolation.

Chebyshev Regression Algorithm

Goal: Approximate $f(x)$ on $[a, b]$. We will use m interpolation nodes (info) to construct an $n+1 \leq m$ polynomial approximation.

That is, the problem is to find the θ_j weights $\{\theta_j\}_{j=0}^{n}$ such that:

$$\sum_{j=0}^{n} \theta_j \psi_j(z_k) = y_k, \quad k = 0, ..., m \geq n+1$$

If $n+1 = m$ we are in collocation, if $n+1 < m$ we are going ot use least squares:

Chebyshev Regression (Algorithm 6.2 in Judd 1998)

Step 1 Compute the $m \geq n+1$ Chebyshev interpolation nodes in $[-1,1]$:

$$z_k = -cos\left(\frac{2k-1}{2m}\pi\right), \quad k = 1, ..., m$$

Step 2 Adjust the nodes to the $[a,b]$ interval:

$$x_k = (z_k + 1)\left(\frac{b-a}{2}\right) + a, \quad k = 1, ..., m$$

Step 3 Evaluate f at the approximation nodes: $w_k = f(x_k), \quad k = 1, ..., m$.

Step 4 Compute Chebyshev coefficients, θ_i, associated with Chebyshev basis $i = 0, ..., n$:

$$\theta_i = \frac{\sum_{k=1}^{m} w_k \psi_i(z_k)}{\sum_{k=1}^{m} \psi_i(z_k)\psi_i(z_k)}$$

to arrive at the approximation for $f(x)$, $x \in [a,b]$:

$$\widetilde{f}(x) = \sum_{i=0}^{n} \theta_i \psi_i\left(2\frac{x-a}{b-a} - 1\right)$$

To see how STEP 4 arises in the Chebyshev regression algorithm note that we can multiply both sides of the aproximand evaluated at the interpolation nodes, i.e., $y_k = \sum_{j=0}^{n} \theta_j \psi_j(z_k)$, by a given basis i, $\psi_i(z_k)$:

$$\sum_{k=1}^{m} \psi_i(z_k) y_k = \sum_{k=1}^{m} \psi_i(z_k) \sum_{j=0}^{n} \theta_j \psi_j(z_k)$$

$$\sum_{k=1}^{m} \psi_i(z_k) y_k = \sum_{j=0}^{n} \theta_j \sum_{k=1}^{m} \psi_i(z_k) \psi_j(z_k)$$

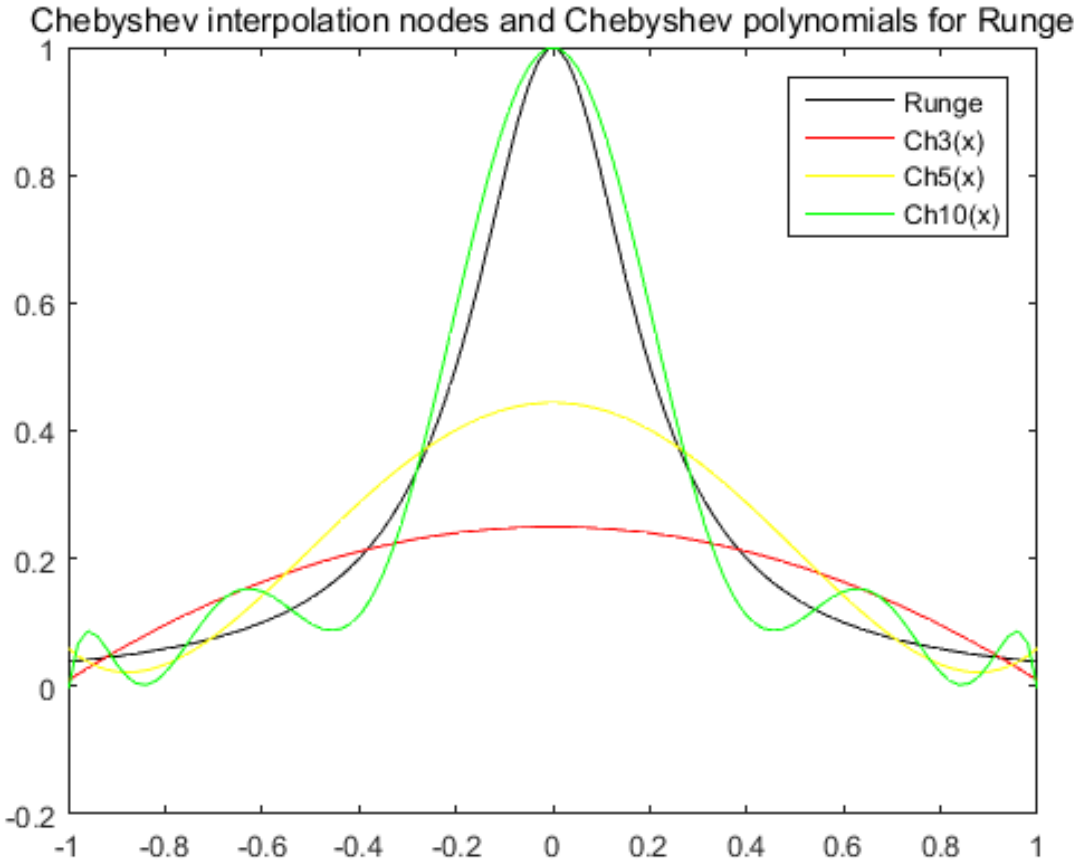
From the discrete orthogonality the terms in $i \neq j$ are zero, so we only keep those cases in which $i = j$ for the basis. This means that we can change subindexes j into i:

$$\sum_{k=1}^{m} \psi_i(z_k) y_k = \theta_i \sum_{k=1}^{m} \psi_i(z_k) \psi_i(z_k)$$

and finally we an isolate θ_i:

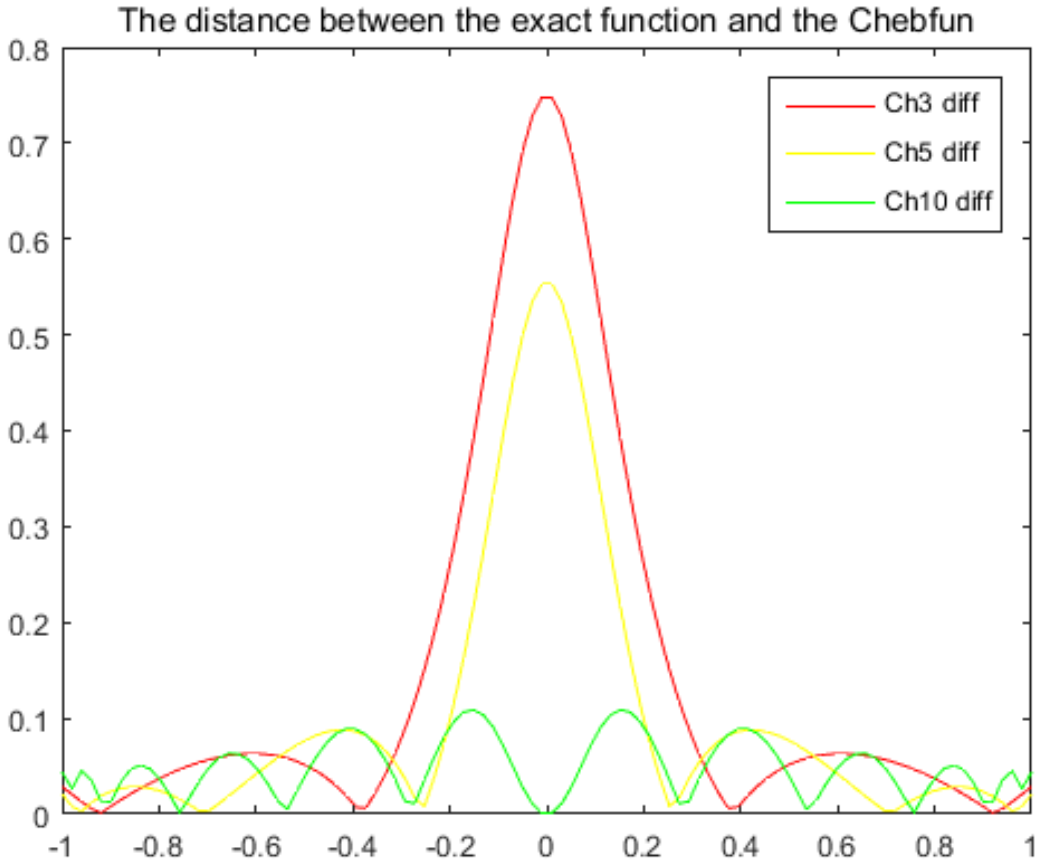
$$\theta_i = \frac{\sum_{k=1}^{m} \psi_i(z_k) y_k}{\sum_{k=1}^{m} \psi_i(z_k) \psi_i(z_k)}$$

Runge Function Approximation: Cheby nodes & Cheby polys



[All credit for this figure goes to Shangdi Hou]

Runge Function Approximation Errors: Cheby nodes & Cheby polys



[All credit for this figure goes to Shangdi Hou]

Polynomials $\widetilde{f}$ with Chebyshev nodes exhibit equioscillant errors.

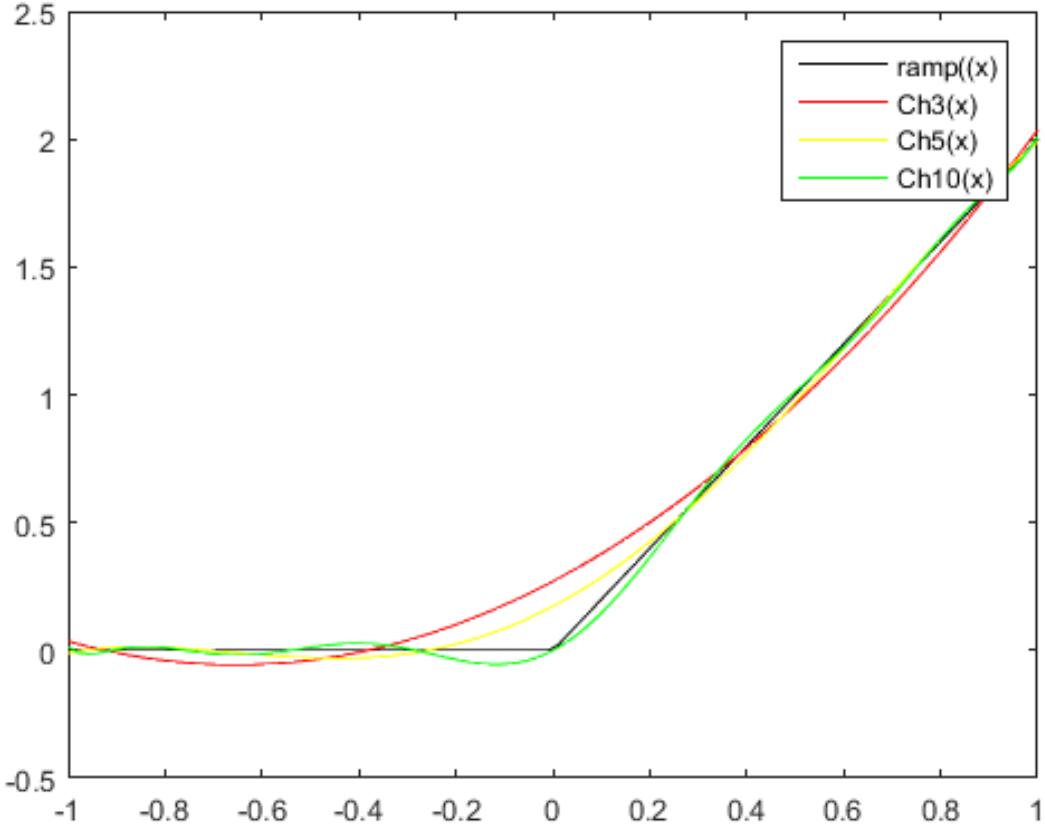
Equioscillation property: Successive extrema of ψ_j are equal in magnitude and alternate in sign, which distributes error uniformly when approximating arbitrary continuous functions.

In other words, for smooth functions polynomial interpolants at equally spaced points **might not** converge (as it is the case of the Runge's function). Instead, polynomial interpolants at Chebyshev points **do** converge.

Smooth functions can be approximated quite accurately by Chebyshev polynomials. However, this is not the case for functions that show some singularity in terms of differentiability. For example, a borrowing constraint might be source of trouble.

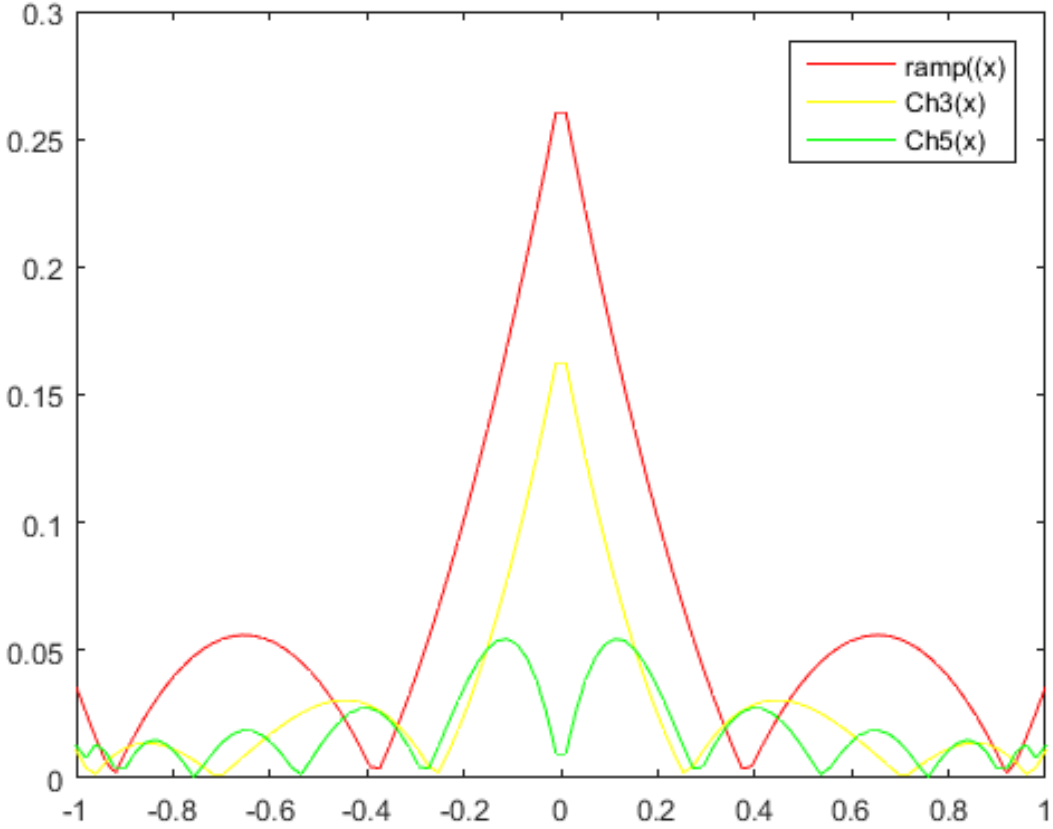
To see this consider the approximation of the ramp function, $\frac{x+|x|}{2}$.

Ramp Function Approximation: Cheby nodes & Cheby polys



[All credit for this figure goes to Shangdi Hou]

Ramp Function Approximation Errors: Cheby nodes & Cheby polys



[All credit for this figure goes to Shangdi Hou]

- **[Homework:]** Approximate $e^{\frac{1}{x}}$, the runge function and the *ramp function* with monomials and Chebyshev polynomials. Do it separately for evenly-space nodes and Chebyshev nodes. What happens when you increase the polynomial degree? Discuss all your results.

Based on its performance Boyd (2000) writes the following criteria to choose among polynomials (see also FVRRS chapter):

1. When in doubt, use Chebyshev polynomials (unless the solution is spatially periodic, in which case an ordinary Fourier series is better).
2. Unless you are sure another set of basis functions is better, use Chebyshev polynomials.
3. Unless you are really, really sure another set of basis functions is better, use Chebyshev polynomials.

We have mostly focused on Chebyshev polynomials because of its nice properties. But there are other orthogonal polynomials out there. Some of these are:

- Laguerre polynomials
- Legendre polynomials
- Hermite polynomials

Laguerre Polynomials

- Defintion:

$$\psi_n(x) = \frac{e^x}{n!}\frac{d^n}{dx^n}\left(x^n e^{-x}\right)$$

- Domain: $[0, \infty)$

- Recursive definition:

$$\begin{aligned}
\psi_0(x) &= 1 \\
\psi_1(x) &= 1 - x \\
\psi_2(x) &= \frac{1}{2}(x^2 - 4x + 2) \\
\psi_{i+1}(x) &= \frac{1}{1+i}\left[(2\,i + 1 - x)\,\psi_i(x) - i\,\psi_{i-1}(x)\right]
\end{aligned}$$

[Figure: Laguerre Basis]

Legendre Polynomials

- Defintion:

$$\psi_n(x) = \frac{1^n}{2^n n!} \frac{d^n}{dx^n} \left(x^2 - 1\right)^n$$

- Domain: $[-1, 1]$

- Recursive definition:

$$\begin{aligned}
\psi_0(x) &= 1 \\
\psi_1(x) &= x \\
\psi_2(x) &= \frac{1}{2}(3x^2 - 1) \\
\psi_{i+1}(x) &= \frac{1}{1+i}[(2\,i+1)x\psi_i(x) - i\,\psi_{i-1}(x)]
\end{aligned}$$

[Figure: Legendre Basis]

'Probabilists' Hermite Polynomials

- Defintion:

$$\psi_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2} \right)$$

- Domain: $(-\infty, \infty)$

- Recursive definition:

$$\begin{aligned} \psi_0(x) &= 1 \\ \psi_1(x) &= x \\ \psi_1(x) &= x^2 - 1 \\ \psi_{i+1}(x) &= x\, \psi_i(x) - \psi_i'(x) \end{aligned}$$

[Figure: Hermite Basis]

Finite Element Methods: Piecewise Polynomial Splines

- **Finite element methods**: we divide the domain into a finite number of disjoint subintervals and approximate f in each ith subinterval. The points at which the polynomial pieces are connected are called knots[4] (or breakpoints), $x_1 = a, ..., x_i, x_{i+1}, ..., x_p = b$.

- The most common finite element method are **piecewise polynomial splines**: a chain of several polynomials (of, usually, lower degree than what is necessary in spectral methods).

- We go over several splines,
  - Spline Interpolation:
    1. Linear, Quadratic and Cubic Splines
    2. Shape-Preserving Schumaker Splines
  - Basis Splines (B-Splines)

[4] If the knots are equidistantly distributed in $[a, b]$ we say the spline is *uniform*, otherwise we say it is *non-uniform*.

Why Splines?

- For example, to overcome the problem of the *Runge's Function* that implies very poor spectral polynomial approximation around the tails (as we have described earlier).
- We can decrease the interpolation error by increasing the number of polynomial pieces which are used to construct the spline instead of increasing the degree of the polynomials used. Here, note that more knots implies more precision, but also more computation.
- An educated allocation of knots on the domain can also improve precision.

Spline Interpolation

- An order-k spline, a function $s_p^k(x)$, consists of a series of order-k polynomial[5] spliced together in $p-1$ segments (that is, p knots) so as to preserve continuity of derivatives of order $k-2$ or less.

- In this context,
  - a piecewise linear interpolant is an order-2 spline —continuity at the knots is preserved for (i) the function value, that is, $s_p^1(x) \in \mathcal{C}^0$.
  - a piecewise quadratic interpolant is an order-3 spline —continuity at the knots is preserved for (i) the function value and (ii) its first derivative, that is, $s_p^2(x) \in \mathcal{C}^1$.
  - a piecewise cubic interpolant is an order-4 spline —continuity at the knots can be preserved for (i) the function value and (ii) its first and (iii) second derivative, that is, $s_p^3(x) \in \mathcal{C}^2$.

[5] That is, $k-1$ degree polynomials

Getting the Coefficients

- The whole deal with the spline interpolation is deriving the polynomial coefficients for each subinterval (we do not know them!).

- **Total # of coefficients:** If the spline is of order k, there are k parameters (associated to polynomials of degree $k-1$) to be computed per subinterval. If there are p knots then there are $p-1$ subintervals. Hence, there are a total of $k(p-1)$ coefficient parameters to be computed.

- **Total # of constraints:** We get the coefficients for each polynomial [one polynomial per subinterval, $\widetilde{f}_i$] making sure they satisfy the constraints imposed by the p-knots and some continuity and smoothness conditions:

  - The interpolation conditions give us, $f(x_i) = \widetilde{f}_i(x_i)$ at each of the p knots. These are p conditions.

  - By continuity, we know that $\widetilde{f}_i(x_{i+1}) = \widetilde{f}_{i+i}(x_{i+1})$ at each of the $p-2$ interior knots. These are $p-2$ conditions.

  - In addition, an order-k spline must be continuous and have continuous derivatives up to order $k-2$ for each of the $p-2$ interior knots. This imposes $(k-2)(p-2)$ more conditions.

  - Then, we have $p + (k-1)(p-2) = k(p-1) + 2 - k$ restrictions, while $k(p-1)$ unknown coefficients. We need to add $2-k$ restrictions to identify all the coefficients.

Linear Spline Interpolation

- We have a linear polynomial,

$$s_i(x) = a_i + b_i\, x$$

  per subinterval $[x_i, x_{i+1}]$

- If we have p knots, then we have $(p-1)$ vectors of a_i, b_i coefficients.

- This implies linear splines have a total number of $2(p-1)$ unknown coefficients.

- We can identify our coefficients using:

  - **The interpolating conditions** at the knots give us p equations:

$$f(x_i) = \widetilde{f}(x_i) = a_i + b_i\, x_i, \quad i = 1, ..., p$$

  - **Continuity** gives $(p-2)$ conditions more, $\widetilde{f}_i(x_i) = \widetilde{f}_{i+1}(x_i)$:

$$a_i + b_i\, x_i = a_{i+1} + b_{i+1}\, x_i, \quad i = 2, ..., p-1$$

  - We cannot further exploit smoothness properties at the knots because smoothness vanishes with linear interpolation. But, do we need to?

- Our constraints imply $p + (p-2) = 2(p-1)$ equations. Since we have $2(p-1)$ unknown coefficients, the system is perfectly identified.

- We are done. Time to solve the linear system for our coefficients!.

- A little bit of algebra yields that for any $x \in [x_i, x_{i+1}]$,

$$\widetilde{f}(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) = a_i + b_i\, x$$

where $a_i = f(x_i) + \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i} x_i$ and $b_i = \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$.

- **Pros:**
  - Preserves monotonicity and concavity of f.
  - We can exploit information about f *clustering points* more closely together in areas of *high curvature* or areas where we know a *kink* exists in order to increase our accuracy.
  - We may be able to capture binding inequality constraints very well.

- **Cons:**
  - The approximated function is not differentiable at the knots (they become kinks).
  - $\widetilde{f}'' = 0$ where it exists (it does not exist at the knots).

Cubic Spline Interpolation

- We have a cubic polynomial,

$$s_i(x) = a_i + b_i\ x + c_i\ x^2 + d_i\ x^3$$

  per subinterval $[x_i, x_{i+1}]$

- If we have p knots, then we have $(p-1)$ vectors of a_i, b_i, c_i, d_i coefficients.

- This implies cubic splines have a total number of $4(p-1)$ unknown coefficients.

- We can identify our coefficients using:

  - **The interpolating conditions** at the knots give us p equations:

$$f(x_i) = \widetilde{f}(x_i) = a_i + b_i\ x_i + c_i\ x_i^2 + d_i\ x_i^3, \quad i = 1, ..., p$$

  - **Continuity** gives $(p-2)$ conditions more, $\widetilde{f}_i(x_i) = \widetilde{f}_{i+1}(x_i)$:

$$a_i + b_i\ x_i + c_i\ x_i^2 + d_i\ x_i^3 = a_{i+1} + b_{i+1}\ x_i + c_{i+1}\ x_i^2 + d_{i+1}\ x_i^3, \quad i = 2, ..., p-1$$

  - **Differentiability** gives $(p-2)$ more, $\widetilde{f}_i'(x_i) = \widetilde{f}_{i+1}'(x_i)$:

$$b_i + 2c_i\ x_i + 3d_i\ x_i^2 = b_{i+1} + 2c_{i+1}\ x_i + 3d_{i+1}\ x_i^2, \quad i = 2, ..., p-1$$

  - **Twice Differentiability** gives $(p-2)$ more, $\widetilde{f}_i''(x_i) = \widetilde{f}_{i+1}''(x_i)$:

$$2c_i + 6d_i\ x_i = 2c_{i+1} + 6d_{i+1}\ x_i, \quad i = 2, ..., p-1$$

- This implies we have $p + 3(p-2) = 4(p-1) - 2$ equations while $4(p-1)$ unknown coefficients.

- We need to set two additional constraints to identify the 2 coefficients left. These 2 constraints can be chosen in several ways:
  - *Natural Splines* impose $s'(x_1) = s'(x_p) = 0$, where x_1 is the first knot and x_p the last one. This minimizes the total curvature of the spline.
  - *Hermite splines* impose $s'(x_1) = f'(x_1)$ and $s'(x_p) = f'(x_p)$ if the derivatives of the actual function are known.
  - *Secant Hermite splines* solve the lack of derivative data by imposing,

$$s'(x_1) = \frac{s(x_2) - s(x_1)}{x_2 - x_1}$$
$$s'(x_p) = \frac{s(x_p) - s(x_{p-1})}{x_p - x_{p-1}}$$

- We now have $4(p-1)$ conditions for $4(p-1)$ unknowns. We are done. Time to solve the linear system.

- **Pros:**
  - Evaluation is cheap (solve for and store coefficients just once, not at each interpolation) and it converges rapidly.
  - For cubic interpolation, error depends only on the 4-th order derivative of f. Higher-order derivative does not affect significantly cubic spline performance. Hence, can approximate very well functions that are not C^∞ with cubic spline interpolation.
  - $\widetilde{f}$ is C^2.

- **Cons:**
  - It does not generally preserve monotonicity or concavity, wich can be a problem, say, for value functions, or for binding inequality constraints (see Examples below).
  - This is important. If our interest is in individuals marginally binding the constraint we might miss a lot of optimal behavior (artificially deviating from it). This can be also very problematic for discrete choice models (e.g., become a entrepreneur or not, migrate or not).

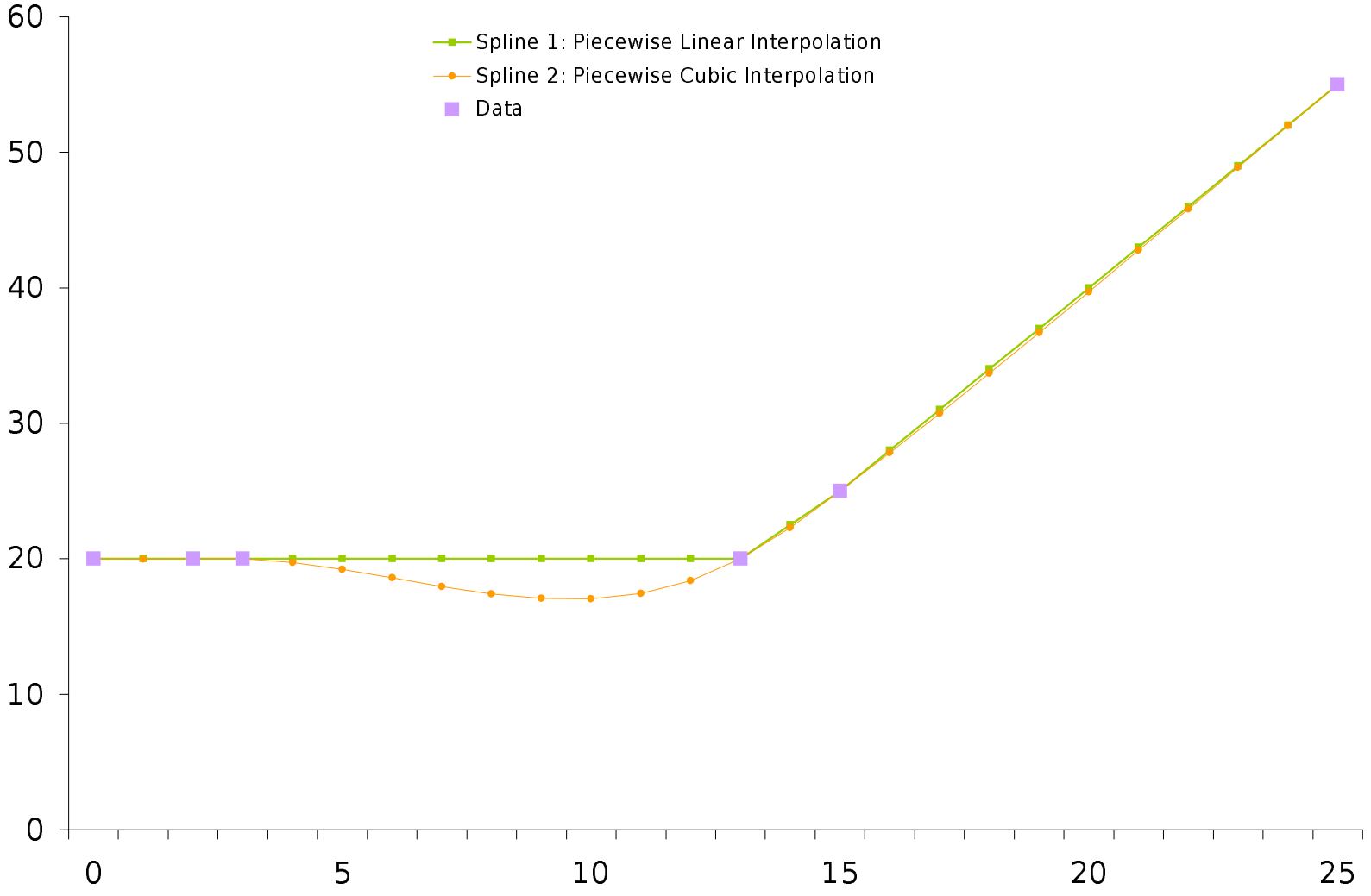


Figure: [Example 1] Splines: Piecewise Linear and Cubic Interpolation

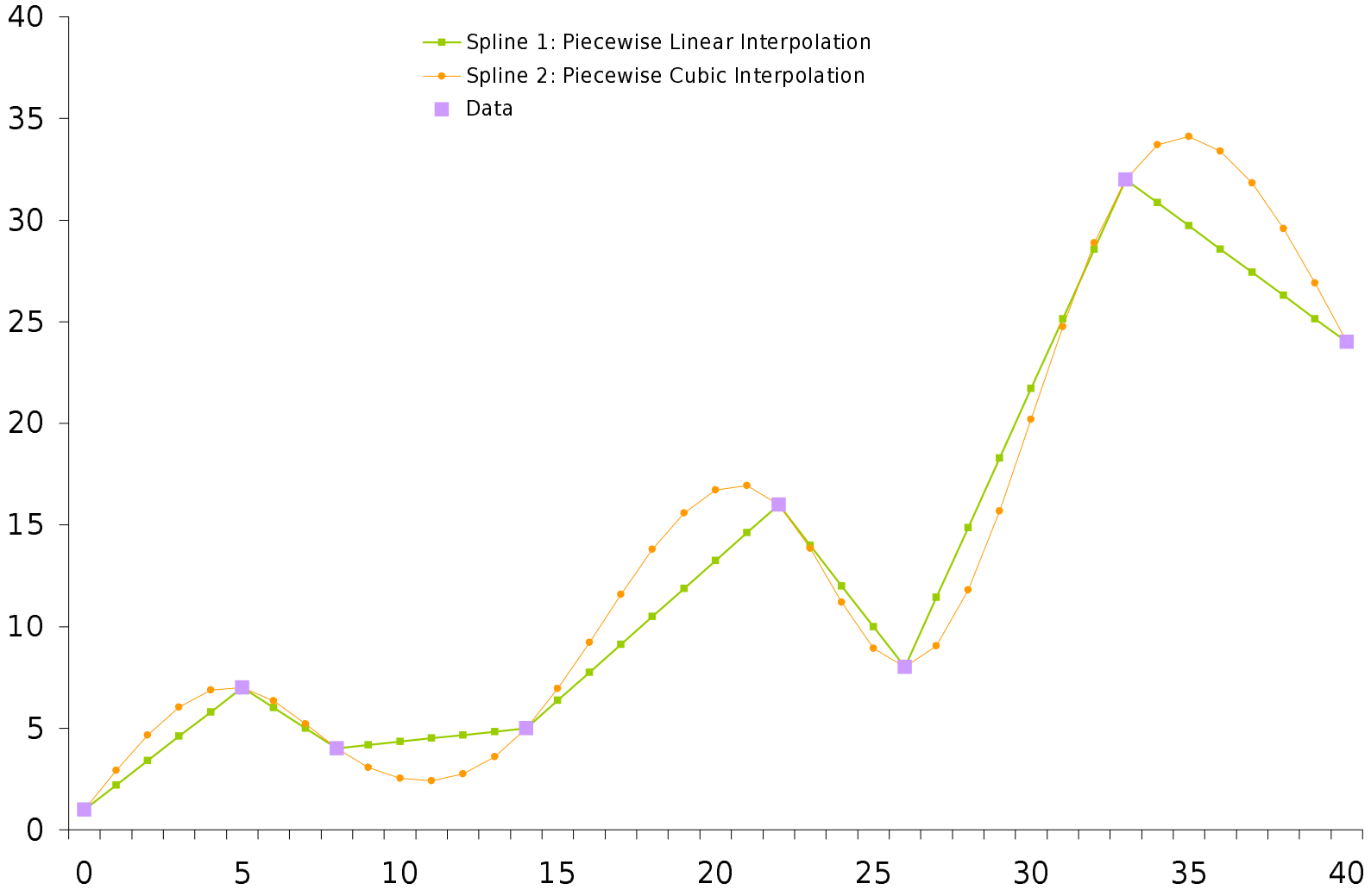


Figure: [Example 2] Splines: Piecewise Linear and Cubic Interpolation

Schumaker Splines

- So far: linear interpolation preserves shape (monotonicity and concavity), but not differentiability. Cubic splines preserves differentiability, but not shape.

- **Schumaker quadratic splines** preserve both shape and differentiability (See Schumaker (1983), SIAM Journal on Numerical Analysis).

- By shape we mean that in those intervals where the data is monotone increasing or decreasing, the spline $s(x)$ should have the same property. Similarly for convexity or concavity.

- Previous shape-preserving methods constructed $s(x)$ as a $\mathcal{C}^1$ quadratic spline with knots at the data points $x_1, ..., x_p$ and with one *ad-hoc selected* additional knot in each subinterval (x_i, x_{i+1}) with $i = 1, ...n$.

- Schumaker (1983) shows us when it is necessary to add knots to a subinterval and where they can be placed by means of an algorithm.

- We will see two cases:
  - Schumaker splines with Hermite data.
  - Schumaker splines with Lagrange data.

With Hermite Data – data provides both value and derivative of the function at each node.

- Suppose we have a subinterval $[x_i, x_{i+1}]$ and the data on the function values $f(.)$ at the knots, $\{y_i, y_{i+1}\}$, and derivatives f' at the knots, $\{d_i, d_{i+1}\}$.

- **Problem 1.** Let $x_i < x_{i+1}$, and suppose $y_i, y_{i+1}, d_i, d_{i+1}$ are given real numbers. We want to construct a function $s \in C^1[x_i, x_{i+1}]$ such that

$$s(x_j) = y_j, s'(x_j) = d_j, \quad j = i, i+1 \tag{5}$$

The following lemma shows that in certain cases **Problem 1** can be solved by a quadratic polynomial.

- **Lemma 2.2** (Schumaker (1983)): If and only if

$$\frac{d_i + d_{i+1}}{2} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \tag{6}$$

then there is a quadratic polynomial solving **Problem 1**. In particular,

$$s(x) = y_i + d_i(x - x_i) + \frac{(d_{i+1} - d_i)(x - x_i)^2}{2(x_{i+1} - x_i)}. \tag{7}$$

When condition (6) fails, it is not possible to solve **Problem 1** using a quadratic polynomial.

Let's analyse the shape-preserving properties of the quadratic polynomial in (7):

- **Lemma 2.4** (Schumaker (1983)): Suppose $d_i d_{i+1} \geq 0$ and that condition (5) holds. Then the quadratic polynomial (7) that solves **Problem 1** is monotone on $I = [x_i, x_{i+1}]$. Moreover, if $d_i < d_{i+1}$ then the polynomial is convex on I, concave otherwise.

The monotonicity assertion follows from the fact that the polynomial $s'(x) = d_i + \frac{(d_{i+1}-d_i)(x-x_i)}{(x_{i+1}-x_i)}$ is linear on I, hence $s'(x)$ has the same sign as d_i and d_{i+1} throughout I.

The convexity (concavity) assertion follows from the fact that $s''(x) = \frac{(d_{i+1}-d_i)}{(x_{i+1}-x_i)}$.

In general, **Lemma 2.2** does not apply because condition (6) is not satisfied.

However, Schumaker (1983)—and others—show that we can always solve **Problem 1** using a quadratic spline with one additional knot. The strategy is to add a knot ξ to the interval (x_i, x_{i+1}) to solve for condition (5) with a quadratic spline within the subinterval.

- **Lemma 2.3** (Schumaker (1983)): For every $\xi \in (x_i, x_{i+1})$, there exists a unique quadratic spline $s(x)$ with an additional knot ξ solving **Problem 1**. In particular, we can write

$$s(x) = \{ \begin{array}{ll} a_1 + b_1(x - x_i) + c_1(x - x_i)^2, & x \in (x_i, \xi) \\ a_2 + b_2(x - \xi) + c_2(x - \xi)^2, & x \in [\xi, x_{i+1}) \end{array} \tag{8}$$

with

$$\begin{aligned} a_1 &= y_i, \quad b_1 = d_i, \quad c_1 = \frac{\overline{d} - d_i}{2\alpha} \\ a_2 &= a_1 + b1\ \alpha + c_1\ \alpha^2, \quad b_2 = \overline{d}, \quad c_2 = \frac{d_{i+1} - \overline{d}}{2\beta} \\ \overline{d} &= s'(\xi) = \frac{2(y_{i+1} - y_i) - (\alpha d_i + \beta d_{i+1})}{x_{i+1} - x_i} \end{aligned}$$

where $\alpha = \xi - x_i$ and $\beta = x_{i+1} - \xi$.

- The shape-preserving properties of this quadratic spline $s(x)$ in (8):
  - monotone increasing vs. monotone decreasing vs. inflextion point
  - convex vs. concave

  depend on the choice of the additinal knot ξ.

- That is, an educated choice of ξ will make the spline $s(x)$ satisfy the properties we desire.

For monotonicity:

- **Lemma 2.5** Suppose that $d_i d_{i+1} \geq 0$. Then the spline $s(x)$ in (8) is monotone if and only if $d_1 \overline{d} \geq 0$. This condition can also be written as

$$\begin{aligned} 2(y_{i+1} - y_i) &\geq d_i(\xi - x_i) + d_{i+1}(x_{i+1} - \xi) \text{ if } d_i, d_{i+1} \geq 0 \\ 2(y_{i+1} - y_i) &\leq d_i(\xi - x_i) + d_{i+1}(x_{i+1} - \xi) \text{ if } d_i, d_{i+1} \leq 0 \end{aligned}$$

Since s' is piecewise linear, $s'(x)$ has one sign throughout I if and only if d_i, d_{i+1} and $\overline{d}$ all have the same sign.

For convexity:

- **Lemma 2.6** Suppose that $d_i < d_{i+1}$, then the spline $s(x)$ in (8) is convex on $I = [x_i, x_{i+1}]$ if and only if
$$d_i \leq \overline{d} \leq d_{i+1}. \tag{9}$$
Similarly, suppose that $d_i > d_{i+1}$,the spline $s(x)$ in (8) is concave on $I = [x_i, x_{i+1}]$ if and only if
$$d_i \geq \overline{d} \geq d_{i+1}. \tag{10}$$

This is obvious since $s''(x) = \frac{\overline{d}-d_i}{\xi - x_i}$ if $x_i \leq x < \xi$ and $s''(x) = \frac{d_{i+1}-\overline{d}}{x_{i+1}-\xi}$ if $\xi \leq x < x_{i+1}$.

Although **Lemma 2.3** shows that we can solve **Problem 1** with a quadratic spline by adding one knot placed arbitrarily in the interval I, it is NOT possible to satisfy conditions (9) and (10) for arbitrary knot locations.

The following **Lemma 2.7** shows which knot locations lead to convex or concave splines.

- **Lemma 2.7**: Let $\delta = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$. Then
  - If $(d_{i+1}-\delta)(d_i-\delta) \geq 0$ implies that $s(x)$ must have an inflection point in the interval I.
  - If $(d_{i+1}-\delta)(d_i-\delta) < 0$ and $|d_{i+1}-\delta| < |d_i-\delta|$ then for all ξ satisfying:

$$x_i < \xi < \overline{\xi} \quad \text{with} \quad \overline{\xi} = x_i + \frac{2(x_{i+1}-x_i)(d_{i+1}-\delta)}{d_{i+1}-d_i} \tag{11}$$

    the spline $s(x)$ in (8) is convex (concave) on I if $d_i < (>) d_{i+1}$. If $d_i d_{i+1} \geq 0$ then $s(x)$ is also monotone.
  - If $(d_{i+1}-\delta)(d_i-\delta) < 0$ and $|d_{i+1}-\delta| > |d_{i+1}-\delta|$, then for all ξ satisfying

$$\underline{\xi} < \xi < x_{i+1} \quad \text{with} \quad \underline{\xi} = x_{i+1} + \frac{2(x_{i+1}-x_i)(d_i-\delta)}{d_{i+1}-d_i} \tag{12}$$

    the spline $s(x)$ in (8) is convex (concave) on I if $d_i < (>) d_{i+1}$. If $d_i d_{i+1} \geq 0$ then $s(x)$ is also monotone.

Schumaker algorithm to choose the additional knot ξ:

- Step 0. Check if **Lemma 2.2** applies. If yes, STOP.
- Step 1. Compute $\delta = \frac{d_{i+1}-d_i}{x_{i+1}-x_i}$.
- Step 2. If $(d_{i+1}-\delta)(d_i-\delta) \geq 0$ set $\xi = .5(x_{i+1}+x_i)$, STOP.
- Step 3. If $(d_{i+1}-\delta)(d_i-\delta) \leq 0$ compute $\overline{\xi}$ using (11), set $\xi = .5(x_i+\overline{\xi})$ and STOP.
- Step 4. If $(d_{i+1}-\delta)(d_i-\delta) \leq 0$ compute $\underline{\xi}$ using (12), set $\xi = .5(x_{i+1}+\underline{\xi})$ and STOP.

Once the ξ is chosen, we compute the spline.

- **[Multiple Intervals]** Schumaker splines with multiple intervals. Given Hermite data for n nodes, apply the process above to find z_i for each interval i.

- Apply the interpolation lemma(s) to each subinterval to construct the spline.

With Lagrange Data – data has only the value of the function at each node.

- We can construct estimates of the function derivative at the nodes. Beware: quality of the approximation will be affected.

- To estimate the slopes $\{d_1, ..., d_n\}$ we use the following: for any node i

$$
\begin{aligned}
L_i &= \left((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2\right)^{\cdot} 5, \quad i = 1, ..., n-1 \\
\delta_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \quad i = 1, ..., n-1 \\
d_1 &= \frac{3\delta_1 - d_2}{2}, d_n = \frac{3\delta_n - d_{n-1}}{2} \\
d_i &= \frac{L_{i-1}\delta_{i-1} + L_i\delta_i}{L_{i-1} + L_i} \; ; \quad if\, \delta_{i-1}\delta_i > 0 \quad i = 1, ..., n-1 \\
d_i &= 0 \; ; \quad if\, \delta_{i-1}\delta_i > 0 \quad i = 1, ..., n-1
\end{aligned}
$$

- With the estimates of the derivatives, proceed as would with Hermite data.

[Homework] Compare time performance and accuracy of Schumaker Splines and Cubic Hermite Splines.

Basis Splines (B-Splines)

- Taking Stock: We have seen so far the case of **spline interpolation** in which we have
  - Data points are interpolated.
  - No convex hull property.
  - Non-local support
  - A previous working example: We achieved $\mathcal{C}^2$ through cubic splines.
- Our goal now is to:
  - Give up interpolation data.
  - Get convex hull property.
  - Local support
  - Current example: $\mathcal{C}^2$ cubic curves with local support.
- How? **B-Splines**. Build basis by designing 'hump' functions.
- Carl de Boor introduced B-splines in 1972 describing a recursion formula for evaluating them.

Theorem: every spline function of a given degree, smoothness, and domain paritition, can be represented as a linear combination of B-splines of that same degree and smoothness, and over that partition (de Boor 1978).

- An order-k spline is characterized by $n = k(p-1)$ free parameters. It should then not be surprising that this spline can be written as a linear combination of n basis functions.

- The B-splines form a basis for splines: basis functions that are nonzero over subintervals of the approximated domain.

B^0-spline

- Order-1 splines implement step function interpolation and are spanned by the B^0-splines. The typical B^0-spline is

$$B_i^0(x) = \begin{cases} 1, & x_i \leq x < x_{i+1} \\ 0, & x < x_i \text{ or } x_{i+1} \leq x \end{cases} \tag{13}$$

  for $i = 1, ..., p$.

- Note that the B_i^0 are right-continuous step functions.

[Figure 10] B^0-spline: step basis functions.

B^1-spline

- Order-2 (linear) splines implement tent function (piecewise linear) interpolation and are spanned by the B^1-splines. The typical B^1-spline is

$$B_i^1(x) = \begin{cases} \frac{x-x_i}{x_{i+1}-x_i}, & x_i \leq x < x_{i+1} \\ \frac{x_{i+2}-x}{x_{i+2}-x_{i+1}}, & x_{i+1} \leq x \leq x_{i+2} \\ 0, & x \leq x_i \text{ or } x \geq x_{i+2} \end{cases} \tag{14}$$

for $i = 1, ..., p$.

- Note that the B_i^1 is the tent function with peak at x_{i_+1}. and zero below x_i and above x_{i+1}.

- Both B^0- and B^1-splines form cardinal bases for interpolation at the x_i's.

[Figure 11] B^1-spline: linear-spline (tent) basis functions.

Higher-order B-splines

- Higher-order B-splines are defined by the recursive relation (de Boor (1978)):

$$\mathrm{B}_i^k(x) = \frac{x - x_i}{x_{i+k} - x_i}\mathrm{B}_i^{k-1}(x) + \frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}}\mathrm{B}_{i+1}^{k-1}(x)$$

  These B-spline families can be used to construct arbitrary splines.

- The formula is simple and stable: A convex combination of two lower-order B-splines gives the value of the next one. The recursion is stable because it combines positive terms. There is no subtraction, no danger of introducing catastrophic cancelation from a finite-precision calculation of the difference of two nearly equal numbers.

[Example: $\mathcal{C}^2$ cubic curves with local support]

[Figure 12] B^2-spline: Cubic-Spline Basis Functions.

- Check `http://ibiblio.org/e-notes/Splines/Intro.htm` for a nice set of several interactive spline java applets.

  i) Bezier splines (2D curves).

  ii) B-splines: Uniform, Cubic, Non-Uniform (2D curves).

  iii) **[Multidimensional]** Tensor product spline surfaces (3D Surfaces)