Lecture II.: Function Approximation, Integration, Differentiation

presented by:
Laszlo Tetenyi

June 26th 2017

Function Approximation

- Interpolate $f : \mathbb{R}^N \to \mathbb{R}$ because it is analytically not tractable
- Functional equations: $f = Tf$

2 ways to approximate functions:

- Locally at some $x_0 \in \mathbb{R}^N$ - perturbation methods
- Global methods

Notation

Define the approximand $\hat{f}$:

$$\hat{f}(x) = \sum_{j}^{K} c_j \phi_j(x)$$

- Basis function: $\{\phi_j\}^j$
- Coefficients: $\{c_j\}^j$
- Suppose we have L nodes $\{x_l\}^l$ with $y_l = f(x_l)$
- L interpolation condition - minimize $y - \Phi c$

Example: Approximate $exp(z)$ using its power series definition ($exp(z) = \sum_{k=0}^{\infty} \frac{z^k}{k!}$) and monomials of up to order K:

- $\phi_j(x) = x^j$
- $c_j = \frac{1}{j!}$
- $\Phi = \begin{bmatrix} 1 & x_1 & \dots & x_1^K \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_L & \dots & x_L^K \end{bmatrix}$

Some theorems

- Weierstrass Theorem: If $C[a,b]$ is the set of all continuous function on $[a,b]$ then for all $f \in C[a,b]$ and $\epsilon > 0$ there exists a polynomial q for which

$$\sup_{x \in [a,b]} |f(x) - q(x)| < \epsilon$$

- Good motivation for using polynomials, but not very useful
- Equioscillation theorem: Amongst the n-degree polynomials, there is a unique one that minimizes the distance $||f - q_n||_\infty$. Moreover for this q_n^* polynomial there is a sequence of $a = x_0 < x_1 \cdots < x_{n+1} = b$ such that

$$f(x_j) - q_n^*(x_j) = (-1)^j ||f - q_n^*||_\infty$$

- Jackson's theorem:

$$||f - q_n^*||_\infty \leq \frac{(n-k)!}{n!} \left(\frac{\pi(b-a)}{2}\right)^k ||f^{(k)}||_\infty$$

Principles

Interpolations can differ:

- Number of nodes & computation of coefficients
- Node placement
- Basis functions
  - Spectral
  - Finite element methods

Good approximation properties:

- Errors outside of the evaluation nodes should decrease as the number of nodes/ degree of approximation increases
- Easy and fast to compute with good accuracy
- Flexible

Number of nodes

- If we have exactly as many nodes as coefficients - root-finding problem on $y - \Phi c \implies c = \Phi^{-1} y$
- If we have more nodes: Least squares - minimize $y - \Phi c \implies c = (\Phi' \Phi)^{-1} \Phi' y$
- Galerkin: Define the error function $r(x) = f(x) - \hat{f}(x)$ and the interpolation conditions as:

$$\int_a^b r(x)\phi_j(x)dx = 0 \ \forall j$$

with the idea that the errors should be orthogonal to the basis functions.

Node placement

Depends on the application. In general:

- Be aware that evenly spaced nodes ($x_i = a + \frac{i-1}{n-1}(b-a)$) are usually outperformed by Chebyshev nodes ($x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos(\frac{n-i+0.5}{n}\pi)$ or $x_i = \frac{a+b}{2} + \frac{b-a}{2}sec(\frac{\pi}{2n})\cos(\frac{n-i+0.5}{n}\pi)$)
- Place more nodes where the function changes rapidly/has kinks - usually these points are close to the endpoints of the interval - so the approximation has more d.f. around these points

Multidimensional node placement:

- Cartesian product of coordinates: $(x_1^1, x_1^2), (x_1^1, x_2^2) \ldots, (x_1^1, x_{L_2}^2) \ldots (x_{L_1}^1, x_{L_2}^2)$
- Sparse grid

Spectral basis functions

- $\phi_j(x)$ is non-zero apart from finitely many points
- Most of them are polynomials - they are well suited to approximate continuous and differentiable functions
- Monomials: $\phi_j(x) = x^j$ that is, $\hat{f}(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_K x^K$
- Chebyshev polynomials - defined recursively:

$$\begin{aligned} \phi_0(x) &= 1 \\ \phi_1(x) &= x \\ \phi_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x) \end{aligned}$$

- Other orthogonal polynomials: Lagendre, Laguerre, Hermite

Monomials

- The main issue with using monomials is that the basis functions do not form an orthogonal basis of the polynomial space
- If they are not orthogonal, then the matrix:

$$\Phi = \begin{bmatrix} 1 & x_1 & \dots & x_1^K \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_L & \dots & x_L^K \end{bmatrix}$$

  the so-called Vandermonde matrix tends to be ill-conditioned, especially as the degree of approximation increases - difficulty in solving $\min_c y - \Phi c$
- Orthogonality is also a nice property, as thanks to Gram-Schmidt, we know how to obtain basis vectors recursively

Chebyshev polynomials

It can be shown that the Chebyshev-polynomials are not much worse than the best possible polynomial approximation:

- Rivlin Theorem: if q_n^c is the n-th degree Chebyshev approximation of $f \in C^1[-1,1]$ then:

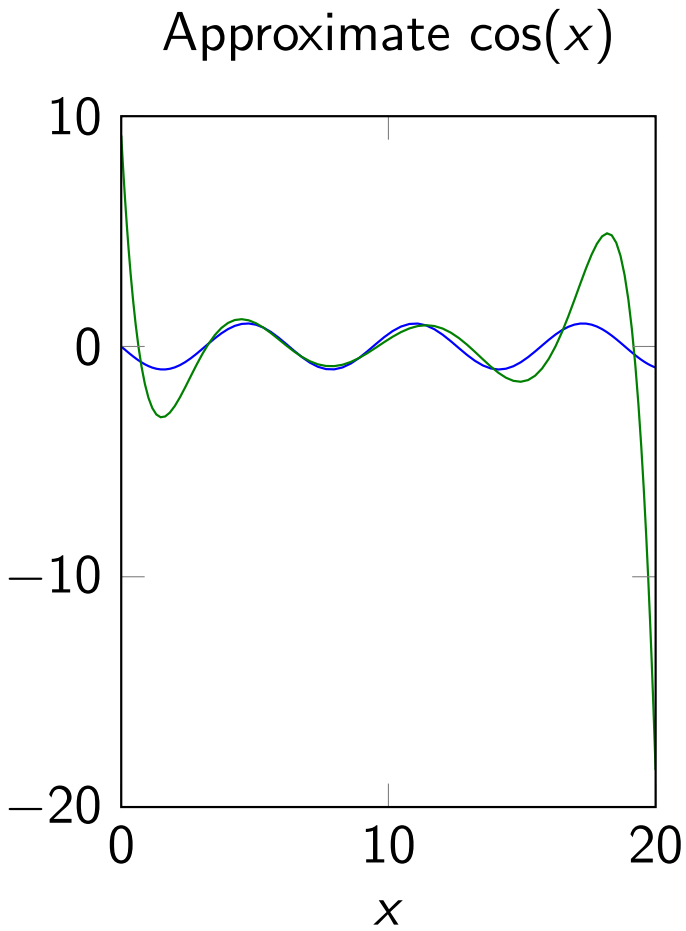
$$||f - q_n^c||_\infty \leq (4 + \frac{4}{\pi^2} \log n)||f - q_n^*||_\infty$$

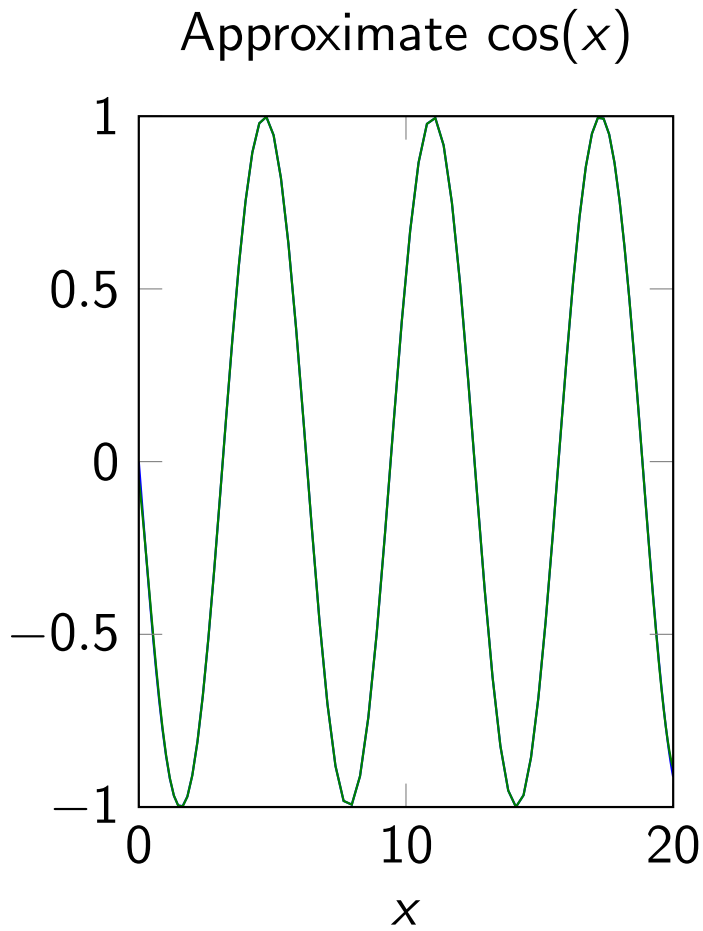
- They form an orthogonal basis with if the dot product in the function space $C^1[-1,1]$ is defined as:

$$< f, g >= \int_{-1}^{1} f(x)g(x)(1-x^2)^{-\frac{1}{2}} dx \ \forall f, g \in C^1[-1,1]$$

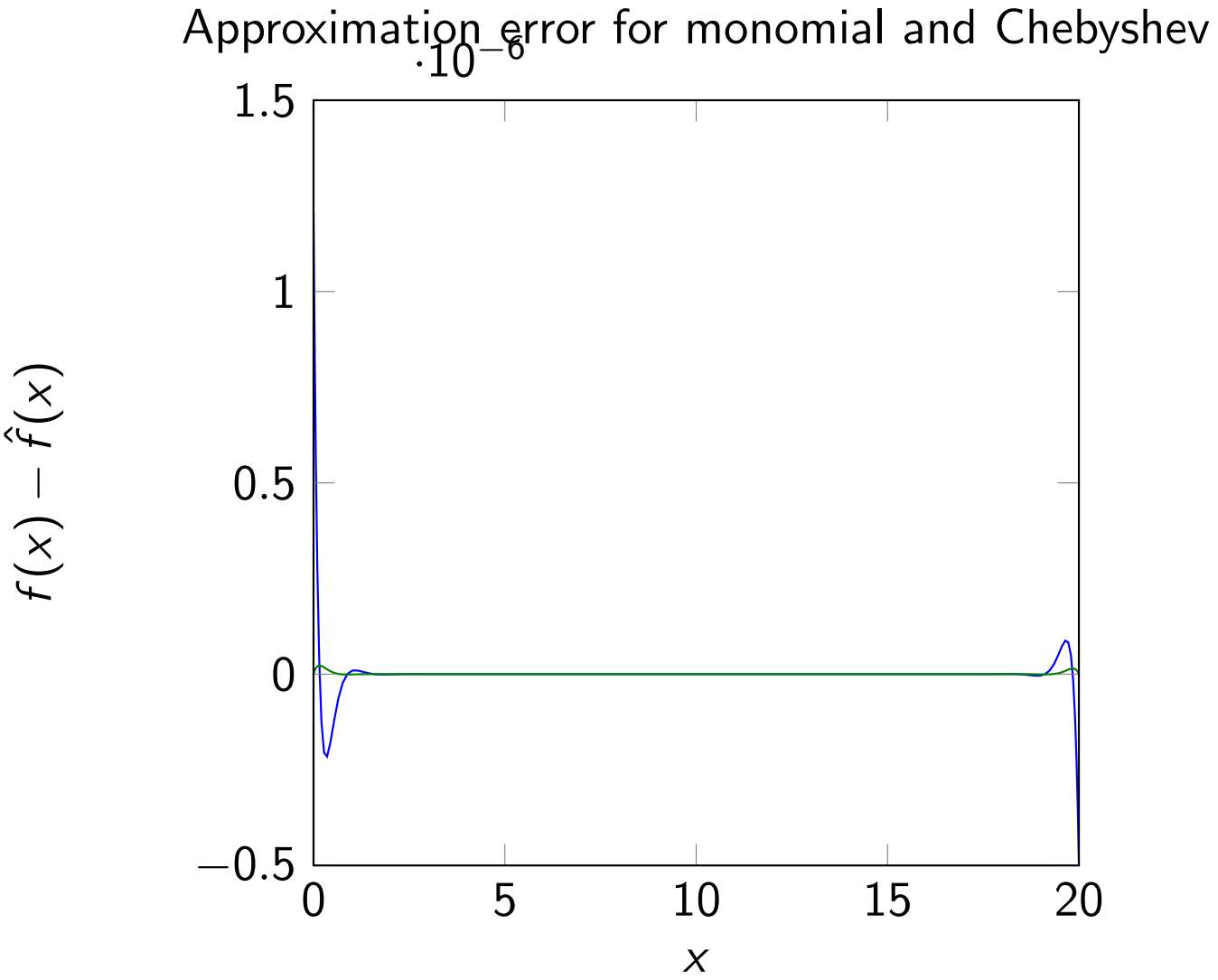
  and so $\Phi'\Phi$ is diagonal.
- They fit well with Chebyshev nodes - the errors of the interpolation are minimized if they are used

Example: Approximate $cos(x)$ - monomial, evenly spaced

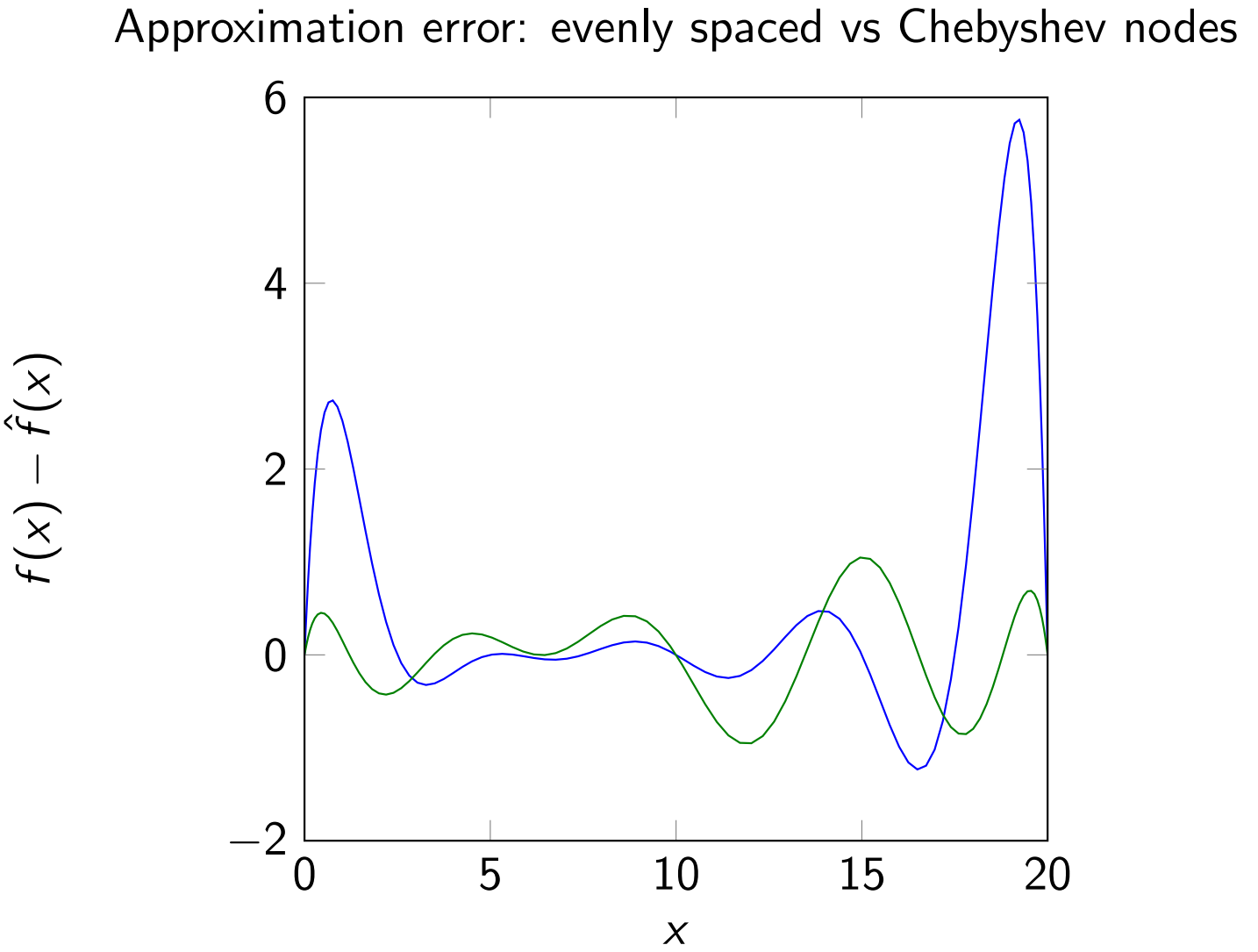




Approximate $cos(x)$ - monomial vs Chebyshev, evenly spaced



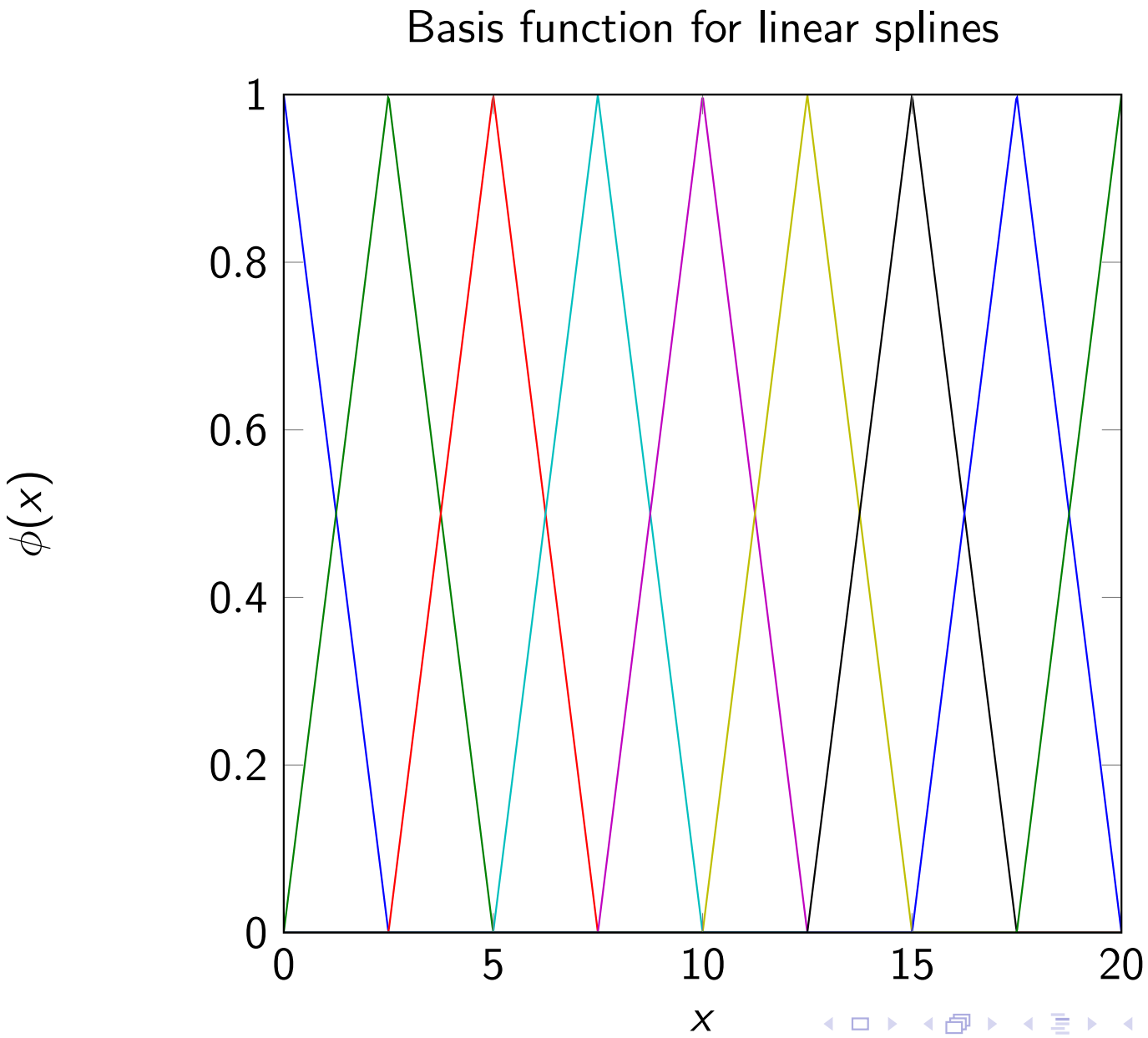
Approximate $cos(x)$ - Chebyshev, evenly spaced vs Chebyshev nodes



Finite element methods - Splines

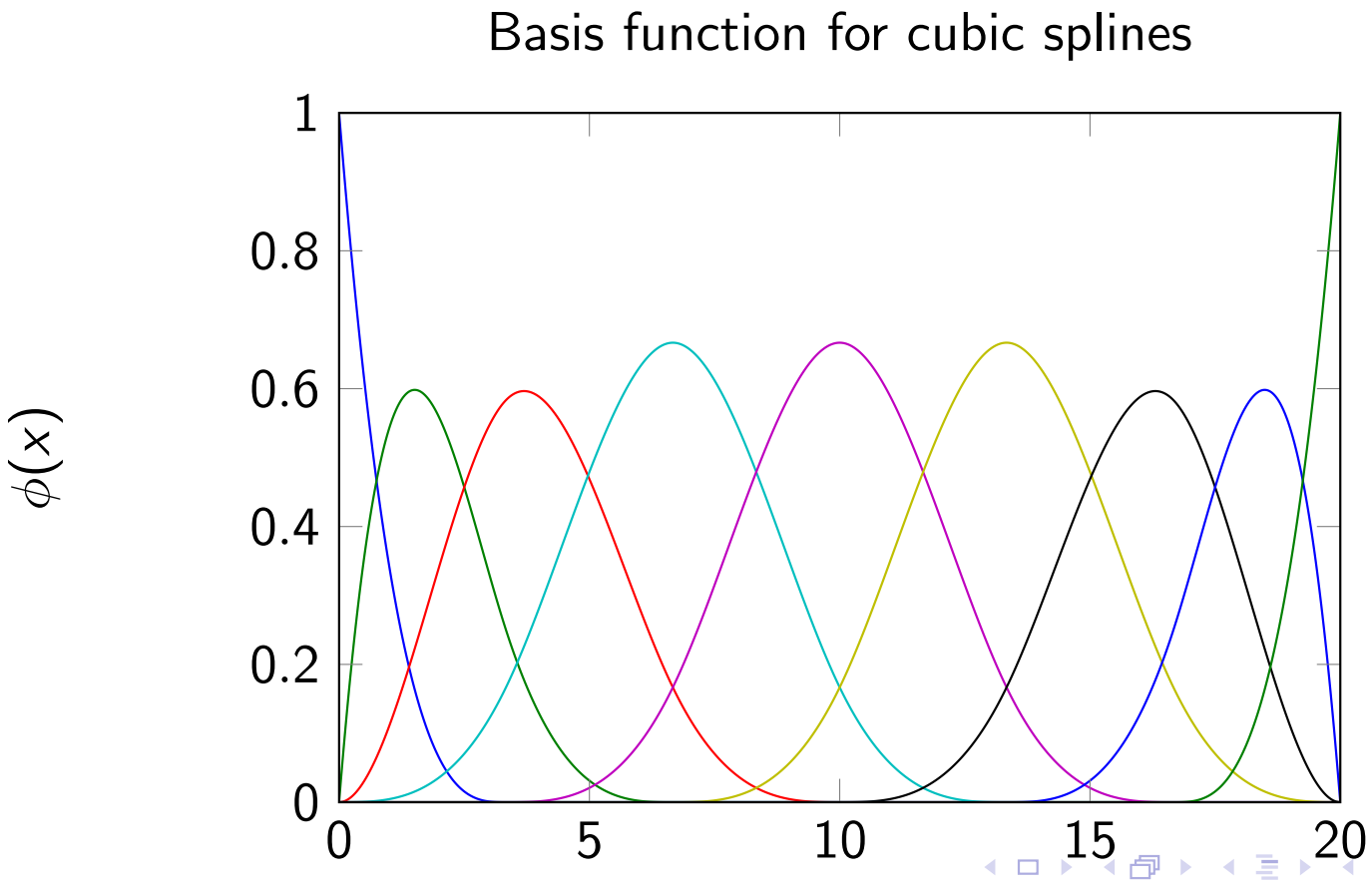
- $\phi_j(x)$ is zero apart from an interval
- A degree n spline consists of n-th degree polynomials, put together in a way to preserve the continuity of derivatives of degree $n-1$
- The points where these polynomials are joined together are called knots
- The default option is to place the nodes at the knots
- Therefore the number of basis functions used now depends on the degree of approximation and the number of knots
- Splines can be superior to spectral methods if the function at hand is not C^∞, has weird shape at known points
- Moreover, as the spline basis matrix Φ is sparse, it can speed up some procedures significantly

Linear spline Basis Functions on [0, 20]

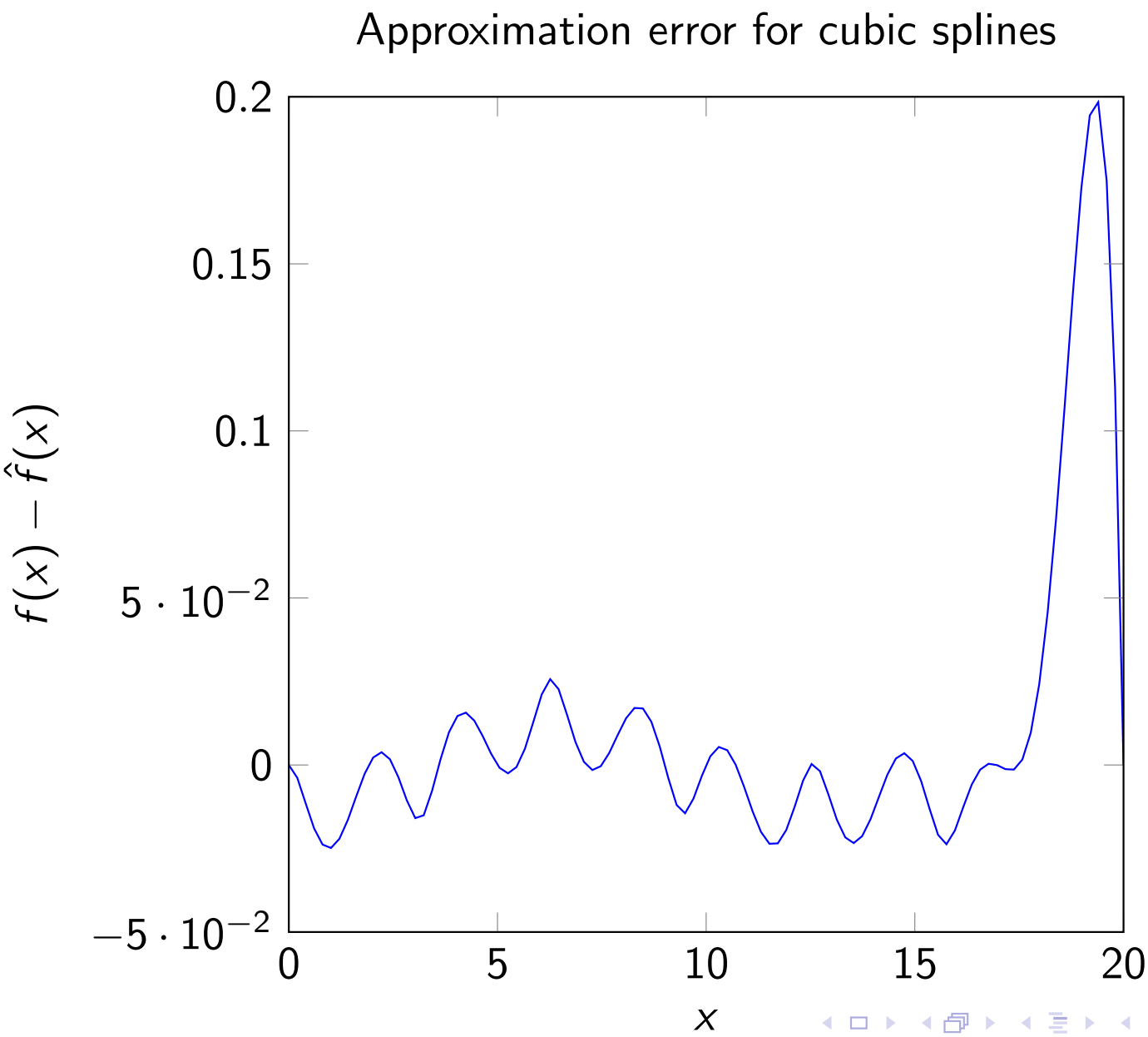


Cubic, 3rd order splines

- Unlike linear splines, they are differentiable everywhere, but do not preserve concavity and monotonicity of f
- If non-differentiability is needed at a point, "stack" knots.
- Schumaker splines preserve both and are differentiable, but the knots are placed by an algorithm



Approximate $cos(x)$ - Cubic splines, smart knots



Multidimensional interpolation

- Tensor products: $x = (x_1, x_2, \dots x_d) \in [-1, 1]^d$
- An n-degree polynomial approximand

$$\hat{f}(x) = \sum_{k_1}^{n} \sum_{k_2}^{n} \cdots \sum_{k_d}^{n} c_{k_1,\dots,k_n} \phi_{k_1}(x_1) \phi_{k_2}(x_2) \dots \phi_{k_d}(x_d)$$

- This yields a simple basis and if the one dimensional basis was orthogonal then the this basis will be orthogonal wrt. to the product norm

Stacking nodes

The tensor product is easy to vectorize. Assume we have a 2 dimensional monomial approximation of degree 2 and 1:

- Take the nodes $x_1 = [0, 1, 2]$ and $x_2 = [0, 1]$.
- Stacking them yields the two dimensional nodes

$$X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} = [i_{n_2} \otimes x_1', x_2' \otimes i_{n_2}]$$

Interpolation matrix

- The univariate basis functions $\phi_1(x) = [1, x, x^2]$ and $\phi_2(x) = [1, x]$ evaluated at the nodes:

$$\Phi_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \quad \Phi_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

- $$\Phi = \Phi_2 \otimes \Phi_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 1 & 1 & 4 \end{bmatrix}$$

- It is more efficient to use the $\otimes$ properties to obtain the coefficients:

$$c = \Phi^{-1} y = (\Phi_2 \otimes \Phi_1)^{-1} y = (\Phi_2^{-1} \otimes \Phi_1^{-1}) y$$

Complete polynomials

- One issue is that the number of elements increase exponentially
- Solution I: restrict the number of polynomials to have a maximum degree of n and drop all other polynomials
- Example above - drop $x_1^2 \cdot x_2$:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 4 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 1 & 1 \end{bmatrix}$$

Smolyak

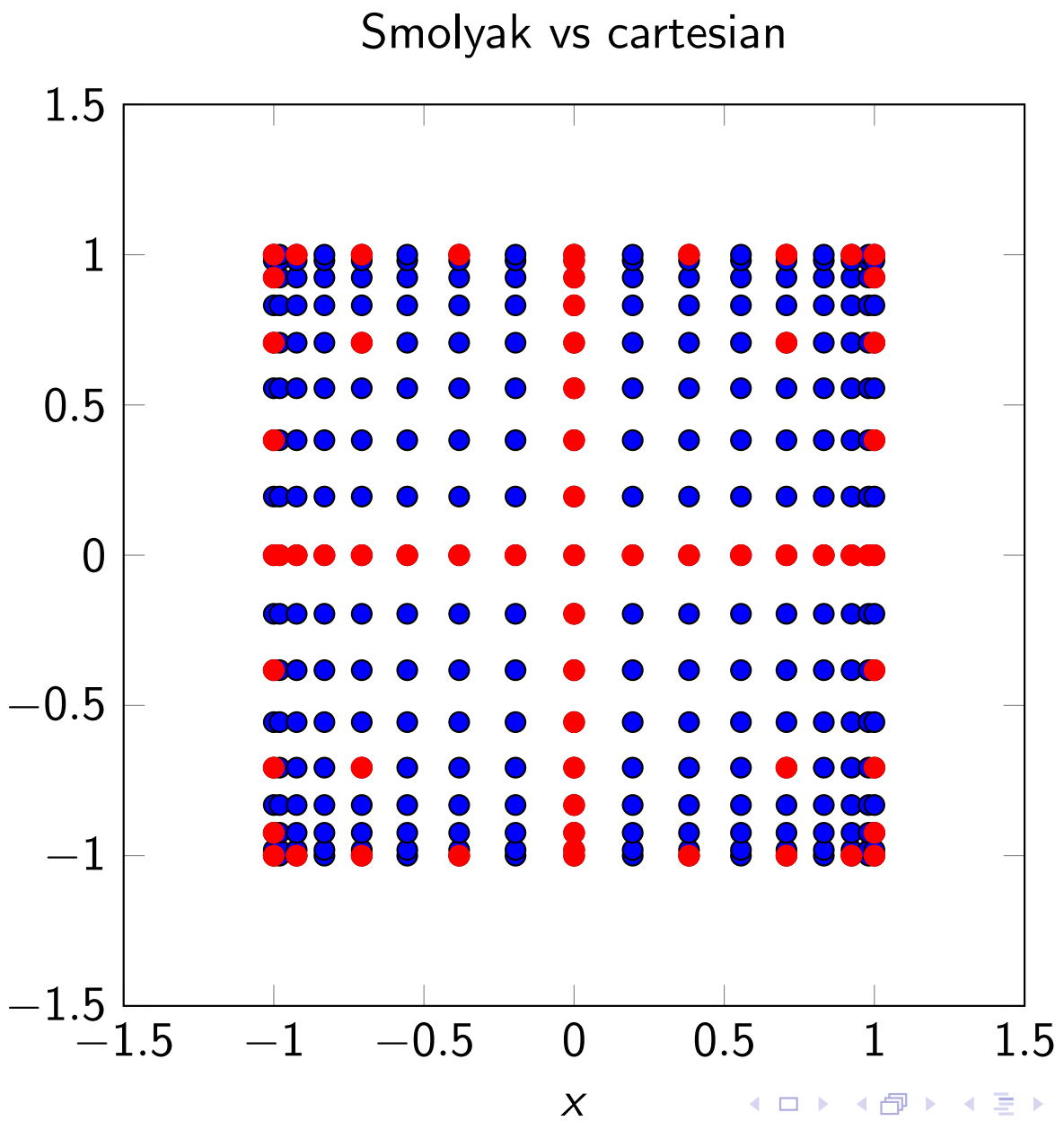
- Instead of taking the tensor product over the nodes, suppose our one dimensional nodes are indexed by i and satisfy:

$$\mathcal{G}^i \subset \mathcal{G}^{i+1}$$

Then for $q > d \in \mathbb{N}$ the sparse grid will be:

$$\mathcal{H} = \bigcup_{q-d+1 \leq \sum_{p=1}^{d} \leq q} (\mathcal{G}^{i_1} \times \mathcal{G}^{i_2} \times \ldots \mathcal{G}^{i_d})$$

Smolyak vs cartesian



Integration

Compute, for any real valued f, "density" or "weight" w over an interval $I \subset \mathbb{R}^N$

$$\int_I f(x)w(x)dx = \sum_{i=0} w_i f(x_i) \tag{1}$$

How to choose w_i and x_i?

- Newton-Cotes - approximate f
- Gaussian quadrature - match the moments
- Monte Carlo - random nodes

Newton-Cotes - trapezoid rule

- Based on approximating the function f directly and calculate the weights accordingly
- Suppose that the nodes are ordered and equidistant: $x_i = a + (i-1)h$ with $h = \frac{b-a}{n-1}$
- Trapezoid rule: linear interpolation

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

and so:

$$\int_a^b f(x)dx = \sum_i^n \int_{x_i}^{x_{i+1}} f(x)dx = \sum_{i=0} w_i f(x_i)$$

with $w_i = \frac{h}{2}$ if $i = 1, n$ and $w_i = h$ for $i = 2, \vdots, n-1$

- if f is smooth then the error is $\mathcal{O}(h^2)$

Newton-Cotes - Simpson's rule

Use quadratic approximation of f.

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{h}{3}(f(x_{2i-1}) + f(x_{2i}) + f(x_{2i+1}))$$

yielding $w_i = \frac{h}{3}$ if $i = 1, n$, $w_i = \frac{4h}{3}$ if i is even and $w_i = \frac{2h}{3}$ if i is odd

- if f is smooth then the error is $\mathcal{O}(h^4)$

Multidimensional extension

Simple and in line with the tensor product - for two dimensions:

- Compute the one dimensional nodes x_i^j and weights w_i^j $\forall j$ dimension
- Then the multidimensional nodes are simply $(x_1^1, x_1^2) \ldots (x_{L_1}^1, x_{L_1}^2)$ and the corresponding weights

$$w_{i,l} = w_i^1 \cdot w_l^2$$

and $\int \int f(x)dx = \sum_i \sum_l w_{i,l} f(x_i^1, x_l^2)$

Gaussian quadrature

- Lets fix the weight function instead of the nodes - the most trivial is Gauss-Legendre: $w(x) = 1$
- Obtain the nodes by solving $2n - 1$ moment matching condition:

$$\int_I x^k w(x) dx = \sum_i^n w_i x_i^k$$

- Idea is that polynomial approximation to f should work well
- The weighting function can be taken to be a density function - well suited to calculate expected values of transformed r.v.
- The only problematic part is solving for x_is - requires a nonlinear solver.
- Seems to be better in approximating the integral than Newton-Cotes if the function is well behaved

Monte Carlo

Main idea - use the SLLN:

- If $X_1, \ldots, X_2$ are iid $\sim X$ then:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(X_i) = \mathbb{E}f(X)$$

- Therefore all we need is obtain a sequence of "random" realizations and use it to approximate the expected value
- It is enough if we have a good random number generator for uniform, as every other r.v. with a "nicely" continuous density can be obtained from it
- Still, generating good uniform random realization is difficult

Differentiation

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

- Sometimes symbolic differentiation is costly, slow or unfeasible
- Finite difference methods
- Automatic differentiation

Finite difference methods

- One-sided:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

- Two-sided:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h} = \frac{f(x_0 + h) - f(x_0 - h)}{(x_0 + h) - (x_0 - h)}$$

- Set h to $\max(|x_0|, 1) \times 6 \times 10^{-6}$ - not too small as then the rounding error is kicking in
- From the Taylor expansion, we get that usually the two-sided is better

Gradient

Simply the above scheme for each coordinate. For two-sided in two dimension:

$$\bigtriangledown f(x_1, x_2) = [\frac{f(x_1 + h, x_2) - f(x_0 - h, x_2)}{(x_1 + h) - (x_1 - h)}, \frac{f(x_1, x_2 + h) - f(x_0, x_2 - h)}{(x_2 + h) - (x_2 - h)}]$$

Automatic differentiation

Relies on the concept of dual numbers:

$$x \to x + \dot{x}d$$

Similar to the extension to of complex numbers but $d^2 = 0$. Turns out that for any differentiable f

$$f(x + \dot{x}d) = f(x) + f'(x)\dot{x}d$$

Then use the chain rule and simple symbolic differentiation to obtain the derivative

Differentiation of an interpolated function

- Sometimes we need the (anti-)derivative of an interpolated function - we might want to maximize it using Newton methods
- It is more efficient to use the derivatives of the basis functions as:

$$f'(x_0) \approx \sum_j^K c_j \frac{\partial \phi_j(x)}{\partial x}\Big|_{x=x_0}$$

and typically $\frac{\partial \phi_j(x)}{\partial x}$ is readily available even symbolically.

Practice Exercises

- Write your own routine for creating the monomial basis matrix in one dimension, that takes nodes, degree and boundaries of the approximation as inputs and returns a conforming matrix
- Use this to approximate Runge's function $\frac{1}{1+25x^2}$ using 20 nodes on $[-1, 1]$. Place the nodes as you wish. Plot the approximation error (relative to a fine grid with at least 100 nodes)
- Create a difference operator for the monomials that takes the degree and boundaries of the approximation as inputs and returns a matrix that if multiplied from the right ($\Phi \cdot D$) returns the derivative matrix
- Use the coefficients you obtained from Part 2) and the difference operator to approximate the derivative of Runge's function and plot the errors
- Approximate the banana function $(1 - x_1)^2 + 100(x_2 - x_1^2)^2$ between $[-3, 3] \times [-3, 3]$ using monomials and 20 nodes in each dimensions.