Chapter 3
Interpolation and Integration

Numerical Methods for Visual Computing
WS 19/20

Michael Moeller
Vaishnavi Gandikota
Computer Vision
Department of Computer Science
and Electrical Engineering
University of Siegen

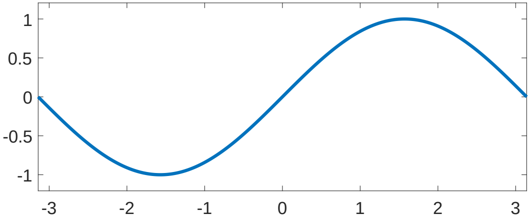
Interpolation

Approximating functions

How does a calculator compute evaluations of complex functions like

$$f(x) = \sin(x),$$

despite only having units that can do basic operations like addition and multiplication?

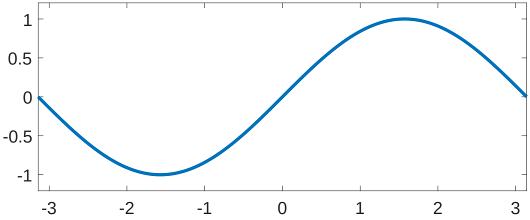


Approximating functions

How does a calculator compute evaluations of complex functions like

$$f(x) = \sin(x),$$

despite only having units that can do basic operations like addition and multiplication?



First possible idea: Taylor expansion to obtain a polynomial

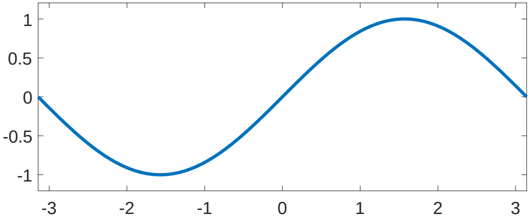
$$\sin(x) \approx \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}?$$

Approximating functions

How does a calculator compute evaluations of complex functions like

$$f(x) = \sin(x),$$

despite only having units that can do basic operations like addition and multiplication?

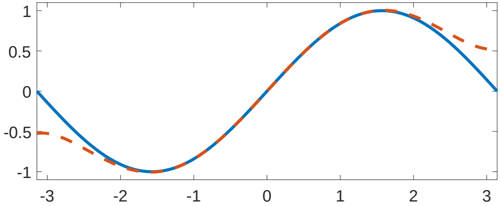


First possible idea: Taylor expansion to obtain a polynomial

$$\sin(x) \approx \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}?$$

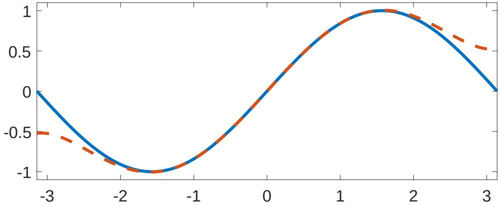
For $n \to \infty$ the approximation converges to the sine function.

Approximating functions



Taylor approximation of $\sin(x)$ for $n = 2$

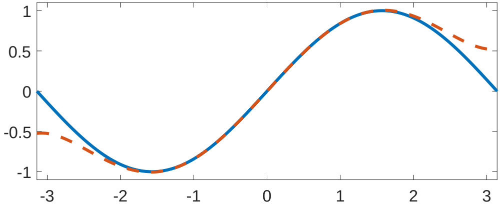
Approximating functions



Taylor approximation of $\sin(x)$ for $n = 2$

Faithful around zero, poor at the boundary.

Approximating functions



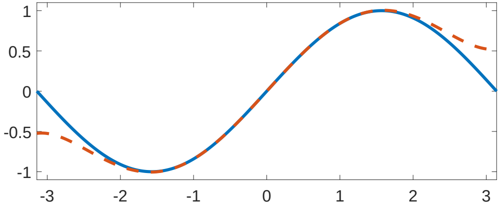
Taylor approximation of $\sin(x)$ for $n = 2$

Faithful around zero, poor at the boundary.

Furthermore, Taylor approximations can be treacherous, consider the approximation around zero of

$$f(x) = \begin{cases} \exp(-1/x^2) & \text{for } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Approximating functions

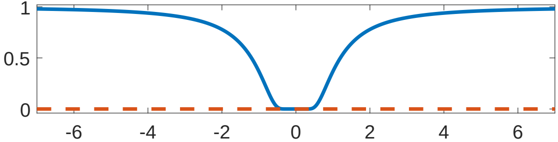


Taylor approximation of $\sin(x)$ for $n = 2$

Faithful around zero, poor at the boundary.

Furthermore, Taylor approximations can be treacherous, consider the approximation around zero of

$$f(x) = \begin{cases} \exp(-1/x^2) & \text{for } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$



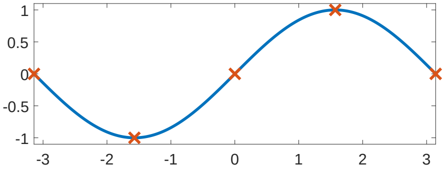
Let us consider a different idea that aims at approximating the function well in the entire interval.

Let us consider a different idea that aims at approximating the function well in the entire interval.

How about sampling the function at a few specific points with high accuracy and then computing a polynomial curve through the sample points?

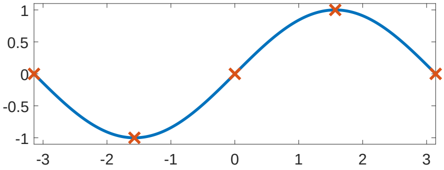
Let us consider a different idea that aims at approximating the function well in the entire interval.

How about sampling the function at a few specific points with high accuracy and then computing a polynomial curve through the sample points?



Let us consider a different idea that aims at approximating the function well in the entire interval.

How about sampling the function at a few specific points with high accuracy and then computing a polynomial curve through the sample points?



We already know one way to solve this: For n sampling points we use a polynomial of degree $n-1$, i.e.,

$$p(x) = \sum_{i=0}^{n-1} a_i x^i,$$

and form a linear system from the equations $p(x_j) = y_j$ for all n many sampled points (x_j, y_j).

Polynomial Interpolation

A different way to immediately state the polynomial without solving a linear system is to consider

$$l_{jn}(x) = \frac{(x - x_1)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)}.$$

Polynomial Interpolation

A different way to immediately state the polynomial without solving a linear system is to consider

$$l_{jn}(x) = \frac{(x - x_1)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)}.$$

Note that $l_{jn}(x_j) = 1$ as the nominator and denominator are equal, and that $l_{jn}(x_i) = 0$ for all $i \neq j$.

Michael Moeller
Vaishnavi Gandikota

Polynomial Interpolation

A different way to immediately state the polynomial without solving a linear system is to consider

$$l_{jn}(x) = \frac{(x - x_1)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)}.$$

Note that $l_{jn}(x_j) = 1$ as the nominator and denominator are equal, and that $l_{jn}(x_i) = 0$ for all $i \neq j$.

This motivates to use

$$p(x) = \sum_{j=1}^{n} l_{jn}(x) y_j$$

as the interpolation polynomial:

- The l_{jn} are polynomials of order $n - 1$ and so is p,
- and obviously $p(x_j) = y_j$ for all j,

which means we found our desired polynomial.

Polynomial Interpolation

There is a lot of theory for efficiently determining interpolation polynomials. The previous form is called **Lagrange form**.

Polynomial Interpolation

There is a lot of theory for efficiently determining interpolation polynomials. The previous form is called **Lagrange form**.

Newton polynomials with divided differences are a different popular form in the literature.

We have too little time to understand the computational advantages of each form in detail. Naturally, they all describe the same polynomial.

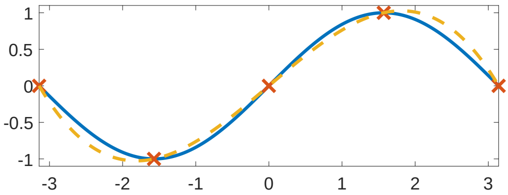
Polynomial Interpolation

There is a lot of theory for efficiently determining interpolation polynomials. The previous form is called **Lagrange form**.

Newton polynomials with divided differences are a different popular form in the literature.

We have too little time to understand the computational advantages of each form in detail. Naturally, they all describe the same polynomial.

Returning to our sine example, we obtain the following result



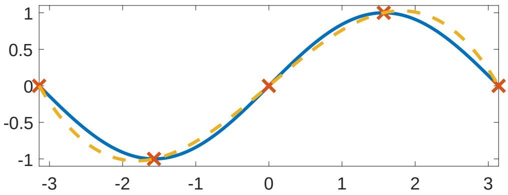
Polynomial Interpolation

There is a lot of theory for efficiently determining interpolation polynomials. The previous form is called **Lagrange form**.

Newton polynomials with divided differences are a different popular form in the literature.

We have too little time to understand the computational advantages of each form in detail. Naturally, they all describe the same polynomial.

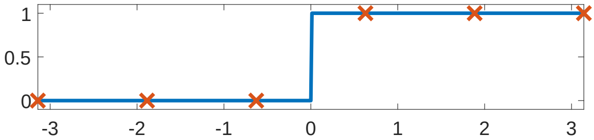
Returning to our sine example, we obtain the following result

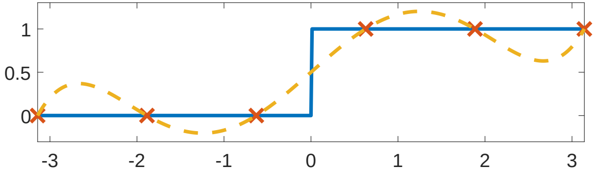


Seems ok - but what if the functions are more complicated?

Polynomial Interpolation

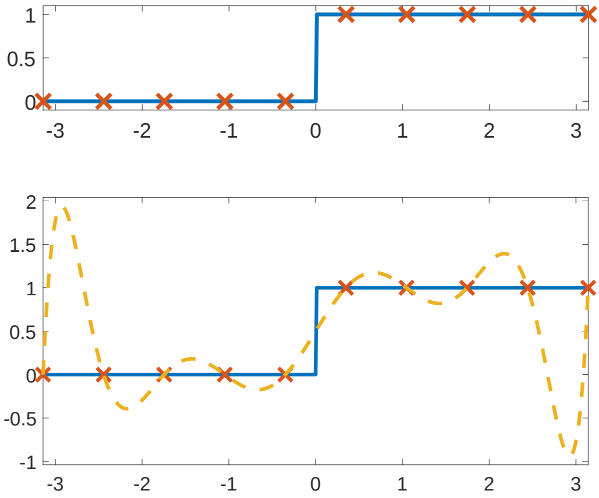
Consider a step function and refine the interpolation





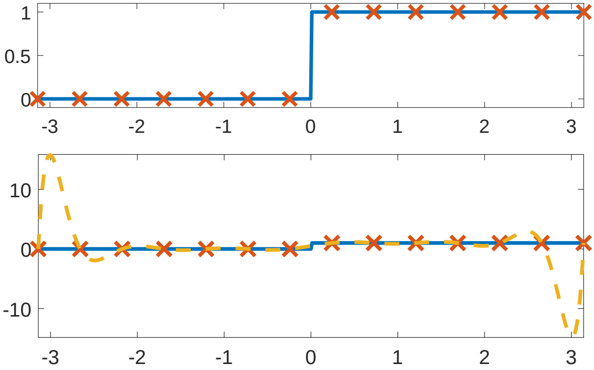
Polynomial Interpolation

Consider a step function and refine the interpolation



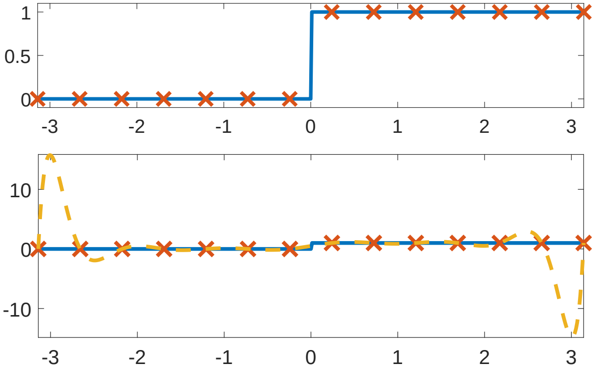
Polynomial Interpolation

Consider a step function and refine the interpolation



Polynomial Interpolation

Consider a step function and refine the interpolation



Bad! Large errors close to the boundary!

Polynomial Interpolation

Reason: If we approximate a function $f : [-1,\ 1] \to \mathbb{R}$ that is n times continuously differentiable by an interpolation polynomial p of degree $n-1$, one can show that

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!} \prod_{i=1}^{n} (x - x_i)$$

for some $\xi \in [-1,\ 1]$, where the x_i are the points with $p(x_i) = f(x_i)$.

Polynomial Interpolation

Reason: If we approximate a function $f : [-1,\ 1] \to \mathbb{R}$ that is n times continuously differentiable by an interpolation polynomial p of degree $n-1$, one can show that

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!} \prod_{i=1}^{n} (x - x_i)$$

for some $\xi \in [-1,\ 1]$, where the x_i are the points with $p(x_i) = f(x_i)$.

For equally spaced x_i the term $|\prod_{i=1}^{n}(x - x_i)|$ is much larger at the boundary of the interval.

Michael Moeller
Vaishnavi Gandikota

Polynomial Interpolation

Reason: If we approximate a function $f : [-1,\ 1] \to \mathbb{R}$ that is n times continuously differentiable by an interpolation polynomial p of degree $n-1$, one can show that

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!} \prod_{i=1}^{n} (x - x_i)$$

for some $\xi \in [-1,\ 1]$, where the x_i are the points with $p(x_i) = f(x_i)$.

For equally spaced x_i the term $|\prod_{i=1}^{n}(x - x_i)|$ is much larger at the boundary of the interval.

Natural idea: Choose the nodes x_i such that

$$\max_{x \in [-1,1]} \left| \prod_{i=1}^{n} (x - x_i) \right|$$

is minimal.

Chebyshev roots

One can show that the problem of choosing the nodes x_i such that

$$\max_{x\in[-1,1]} \left| \prod_{i=1}^{n} (x - x_i) \right|$$

is minimal can be achieved

Chebyshev roots

One can show that the problem of choosing the nodes x_i such that

$$\max_{x \in [-1,1]} \left| \prod_{i=1}^{n} (x - x_i) \right|$$

is minimal can be achieved by picking x_i to be the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \ldots, n.$$

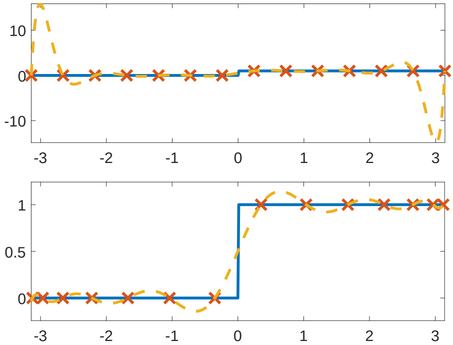
Chebyshev roots

One can show that the problem of choosing the nodes x_i such that

$$\max_{x \in [-1,1]} \left| \prod_{i=1}^{n} (x - x_i) \right|$$

is minimal can be achieved by picking x_i to be the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \dots, n.$$



Systematical analysis

We have just seen an example of using an interpolation polynomial with Chebyshev roots, which tried to minimize the maximal pointwise error bound.

Systematical analysis

We have just seen an example of using an interpolation polynomial with Chebyshev roots, which tried to minimize the maximal pointwise error bound.

This rises the systematic idea to call p^* a **best approximation** of a function f in some set T of candidate functions with respect to the norm $\|\cdot\|$ if

$$\|p^* - f\| \leq \|p - f\| \qquad \forall p \in T.$$

Systematical analysis

We have just seen an example of using an interpolation polynomial with Chebyshev roots, which tried to minimize the maximal pointwise error bound.

This rises the systematic idea to call p^* a **best approximation** of a function f in some set T of candidate functions with respect to the norm $\|\cdot\|$ if

$$\|p^* - f\| \leq \|p - f\| \qquad \forall p \in T.$$

Will will now consider two examples, both of which choose T to be the set of polynomials of degree n, and

- $\|\cdot\|$ being the L^2 norm
- $\|\cdot\|$ being the L^∞ norm

Best L^2-norm approximation

One can show that

$$\langle f, g \rangle := \int_{-1}^{1} f(x) \cdot g(x) \, dx$$

defines a scalar product on the space of continuous functions.

Best L^2-norm approximation

One can show that

$$\langle f, g \rangle := \int_{-1}^{1} f(x) \cdot g(x)\, dx$$

defines a scalar product on the space of continuous functions.

The *induced* norm

$$\|f\|_2 := \sqrt{\langle f, f \rangle}$$

is called the L^2 norm, and is one of the most commonly used norms in visual computing.

Best L^2-norm approximation

One can show that

$$\langle f, g \rangle := \int_{-1}^{1} f(x) \cdot g(x) \, dx$$

defines a scalar product on the space of continuous functions.

The *induced* norm

$$\|f\|_2 := \sqrt{\langle f, f \rangle}$$

is called the L^2 norm, and is one of the most commonly used norms in visual computing.

Question: How can we find the best approximation of a given function f with a polynomial p in the sense of the L^2-norm?

Reminder: Orthogonal systems in linear algebra

We say that two vectors f and g of some vector space V with an inner product $\langle \cdot, \cdot \rangle$ are **orthogonal**, if

$$\langle f, g \rangle = 0.$$

Reminder: Orthogonal systems in linear algebra

We say that two vectors f and g of some vector space V with an inner product $\langle \cdot, \cdot \rangle$ are **orthogonal**, if

$$\langle f, g \rangle = 0.$$

We will show in the exercise:

Best approximation

Let $v_1, \dots, v_n$ be pairwise orthogonal with $\|v_i\| = 1 \ \forall i$. For any $f \in V$ it holds that

$$\sum_{i=1}^{n} \langle v_i, f \rangle v_i$$

is the best approximation of f in $\text{span}(v_1, \dots, v_n)$.

Reminder: Orthogonal systems in linear algebra

We say that two vectors f and g of some vector space V with an inner product $\langle \cdot, \cdot \rangle$ are **orthogonal**, if

$$\langle f, g \rangle = 0.$$

We will show in the exercise:

Best approximation

Let $v_1, \ldots, v_n$ be pairwise orthogonal with $\|v_i\| = 1 \; \forall i$. For any $f \in V$ it holds that

$$\sum_{i=1}^{n} \langle v_i, f \rangle v_i$$

is the best approximation of f in $\text{span}(v_1, \ldots, v_n)$.

How can we use this result for finding a best polynomial fit?

Best polynomial L^2 fit

Surely, $1,\ x,\ x^2,\ \dots,\ x^n$ is a basis of the polynomials of degree n.

Best polynomial L^2 fit

Surely, $1,\ x,\ x^2,\ \ldots,\ x^n$ is a basis of the polynomials of degree n.

Idea: Use the Gram-Schmidt technique to form an orthonormal basis. $\rightarrow$ Board!

Michael Moeller
Vaishnavi Gandikota

Best polynomial L^2 fit

Surely, $1,\ x,\ x^2,\ \ldots,\ x^n$ is a basis of the polynomials of degree n.

Idea: Use the Gram-Schmidt technique to form an orthonormal basis. $\rightarrow$ Board!

$$v_1(x) = \frac{1}{\sqrt{2}}, \quad v_2(x) = \sqrt{\frac{3}{2}}x, \quad v_3(x) = \sqrt{\frac{5}{8}}(3x^2 - 1).$$

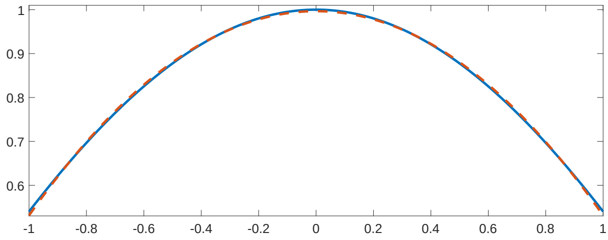
Best polynomial L^2 fit

Surely, $1,\ x,\ x^2,\ \dots,\ x^n$ is a basis of the polynomials of degree n.

Idea: Use the Gram-Schmidt technique to form an orthonormal basis. $\rightarrow$ Board!

$$v_1(x) = \frac{1}{\sqrt{2}}, \quad v_2(x) = \sqrt{\frac{3}{2}}x, \quad v_3(x) = \sqrt{\frac{5}{8}}(3x^2 - 1).$$

Example: Approximating the cosine



$$\cos(x) \approx 0.99656 - 0.46525x^2.$$

Remarks

The fact that

$$\sum_{i=1}^{n} \langle v_i, f \rangle v_i$$

is the best approximation of f in $\mathrm{span}(v_1, \ldots, v_n)$ is of course not limited to polynomials, but holds for any orthonormal system.

Remarks

The fact that

$$\sum_{i=1}^{n} \langle v_i, f \rangle v_i$$

is the best approximation of f in $\text{span}(v_1, \ldots, v_n)$ is of course not limited to polynomials, but holds for any orthonormal system.

Prominent example:

$$\left\{ \frac{1}{2}, \cos(\pi x), \sin(\pi x), \cos(2\pi x), \ldots, \cos(n\,\pi x), \sin(n\,\pi x) \right\}$$

on $[-1, 1]$ with the L^2 norm. In this case the inner products $\langle v_i, f \rangle$ are also known as the **Fourier coefficients**.

Remarks

The fact that

$$\sum_{i=1}^{n} \langle v_i, f \rangle v_i$$

is the best approximation of f in $\text{span}(v_1, \dots, v_n)$ is of course not limited to polynomials, but holds for any orthonormal system.

Prominent example:

$$\left\{ \frac{1}{2}, \cos(\pi x), \sin(\pi x), \cos(2\pi x), \dots, \cos(n\,\pi x), \sin(n\,\pi x) \right\}$$

on $[-1, 1]$ with the L^2 norm. In this case the inner products $\langle v_i, f \rangle$ are also known as the **Fourier coefficients**.

Next question: What if we change the measure of approximation quality?

Best L^∞-norm approximation

Instead of looking for the best L^2 approximation, let us try to find the best polynomial L^∞ approximation, i.e. find a polynomial p of degree n such that

$$\|f - p\|_\infty := \max_{x \in [-1,1]} |f(x) - p(x)|$$

is minimal among all possible polynomials p of degree n.

Best L^∞-norm approximation

Instead of looking for the best L^2 approximation, let us try to find the best polynomial L^∞ approximation, i.e. find a polynomial p of degree n such that

$$\|f - p\|_\infty := \max_{x \in [-1,1]} |f(x) - p(x)|$$

is minimal among all possible polynomials p of degree n.

Difficulty: The L^∞ norm is not induced by a scalar product! We cannot look for an orthogonal basis – there is no notion of orthogonality!

Best L^∞-norm approximation

Math is great! :-) People have proven the following theorem to still determine an algorithm:

Best L^∞-norm approximation

Math is great! :-) People have proven the following theorem to still determine an algorithm:

Equi-oscillation property

A polynomial p of degree n is the best L^∞ approximation to a continuous function $f : [-1,\ 1] \to \mathbb{R}$ if and only if there exists a set of $n+2$ distinct points x_i and $\sigma \in \{-1, 1\}$ such that

$$f(x_i) - p(x_i) = \sigma\,(-1)^i\,\|f - p\|_\infty, \qquad i = 0, \dots, n+1,$$

Michael Moeller
Vaishnavi Gandikota

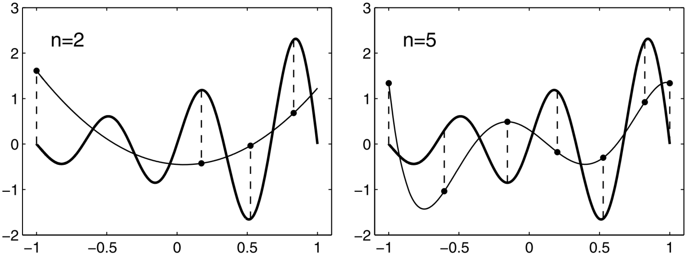
Best L^∞-norm approximation

Math is great! :-) People have proven the following theorem to still determine an algorithm:

Equi-oscillation property

A polynomial p of degree n is the best L^∞ approximation to a continuous function $f : [-1, 1] \to \mathbb{R}$ if and only if there exists a set of $n+2$ distinct points x_i and $\sigma \in \{-1, 1\}$ such that

$$f(x_i) - p(x_i) = \sigma\,(-1)^i\,\|f - p\|_\infty, \qquad i = 0, \dots, n+1,$$



From: Pachon, Trefethen: Barycentric-Remez algorithms for best polynomial approximation in the chebfun system.

Best L^∞-norm approximation

The **equi-oscillation property** allows to develop an algorithm for determining best L^∞ approximations.

Michael Moeller
Vaishnavi Gandikota

Best L^∞-norm approximation

The **equi-oscillation property** allows to develop an algorithm for determining best L^∞ approximations.

First of all note that for given $x_0 < \ldots < x_{n+1}$ there exists a (unique) polynomial p of degree n and a constant D such that

$$f(x_i) - p(x_i) = (-1)^i \ D$$

hold for all $i = 0, \ldots, n+1$.

Best L^∞-norm approximation

The **equi-oscillation property** allows to develop an algorithm for determining best L^∞ approximations.

First of all note that for given $x_0 < \ldots < x_{n+1}$ there exists a (unique) polynomial p of degree n and a constant D such that

$$f(x_i) - p(x_i) = (-1)^i \, D$$

hold for all $i = 0, \ldots, n+1$.

Reason: A polynomial of degree n has $n+1$ free parameters. The constant D is an additional free parameter, i.e., we have $n+2$ linear equations and $n+2$ unknowns. The property $x_0 < \ldots < x_{n+1}$ ensures that the linear system has a unique solution.

Best L^∞-norm approximation

Idea for an algorithm:

1. Sample the interval $[-1, 1]$ uniformly to obtain initial guesses for $x_0, \ldots, x_{n+1}$.

Best L^∞-norm approximation

Idea for an algorithm:

1. Sample the interval $[-1, 1]$ uniformly to obtain initial guesses for $x_0, \ldots, x_{n+1}$.
2. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i\, D \quad \forall i.$$

Best L^∞-norm approximation

Idea for an algorithm:

1. Sample the interval $[-1, 1]$ uniformly to obtain initial guesses for $x_0, \ldots, x_{n+1}$.

2. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i \, D \quad \forall i.$$

3. If $D = \|p - f\|_\infty$, the equioscillation property tells us that we found the best L^∞ approximation.

Best L^∞-norm approximation

Idea for an algorithm:

1. Sample the interval $[-1, 1]$ uniformly to obtain initial guesses for $x_0, \dots, x_{n+1}$.
2. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i D \quad \forall i.$$

3. If $D = \|p - f\|_\infty$, the equioscillation property tells us that we found the best L^∞ approximation.
4. If $D < \|p - f\|_\infty$, slightly move the x_i such that $|p(x_i) - f(x_i)|$ increases, and return to 2.

Best L^∞-norm approximation

Very simplified version of the **Remez-Algorithm**[1]: Pick some starting points $x_0, \ldots, x_{n+1}$, and a step sizes τ, ϵ. Iterate

1. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i D \quad \forall i.$$

[1] For more details see, e.g., Barycentric-Remez algorithms for best polynomial approximation in the chebfun system.

Best L^∞-norm approximation

Very simplified version of the **Remez-Algorithm**[1]: Pick some starting points $x_0, \ldots, x_{n+1}$, and a step sizes τ, ϵ. Iterate

1. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i \, D \quad \forall i.$$

2. For each x_i compute

$$e_i^1 = f(x_i) - p(x_i), \quad e_i^2 = f(x_i + \epsilon) - p(x_i + \epsilon)$$

[1] For more details see, e.g., Barycentric-Remez algorithms for best polynomial approximation in the chebfun system.

Best L^∞-norm approximation

Very simplified version of the **Remez-Algorithm**[1]: Pick some starting points $x_0, \dots, x_{n+1}$, and a step sizes τ, ϵ. Iterate

1. Compute the polynomial p and the constant D that satisfy
$$f(x_i) - p(x_i) = (-1)^i \, D \quad \forall i.$$
2. For each x_i compute
$$e_i^1 = f(x_i) - p(x_i), \quad e_i^2 = f(x_i + \epsilon) - p(x_i + \epsilon)$$
3. If $e_i^1 > 0$ and $e_i^1 > e_i^2$ or if $e_i^1 < 0$ and $e_i^1 < e_i^2$, update
$$x_i \leftarrow \max(x_i - \tau, -1)$$

[1] For more details see, e.g., Barycentric-Remez algorithms for best polynomial approximation in the chebfun system.

Best L^∞-norm approximation

Very simplified version of the **Remez-Algorithm**[1]: Pick some starting points $x_0, \dots, x_{n+1}$, and a step sizes τ, ϵ. Iterate

1. Compute the polynomial p and the constant D that satisfy

$$f(x_i) - p(x_i) = (-1)^i \, D \quad \forall i.$$

2. For each x_i compute

$$e_i^1 = f(x_i) - p(x_i), \quad e_i^2 = f(x_i + \epsilon) - p(x_i + \epsilon)$$

3. If $e_i^1 > 0$ and $e_i^1 > e_i^2$ or if $e_i^1 < 0$ and $e_i^1 < e_i^2$, update

$$x_i \leftarrow \max(x_i - \tau, -1)$$

4. Otherwise update

$$x_i \leftarrow \min(x_i + \tau, 1)$$

---

[1] For more details see, e.g., Barycentric-Remez algorithms for best polynomial approximation in the chebfun system.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.
- The results often improve drastically if one chooses the sample points x_i as the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \dots, n.$$

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.
- The results often improve drastically if one chooses the sample points x_i as the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \dots, n.$$

- More systematically, one can compute the best approximation of f with respect to a given norm.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.
- The results often improve drastically if one chooses the sample points x_i as the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \dots, n.$$

- More systematically, one can compute the best approximation of f with respect to a given norm.
  - For the L^2 norm we compute orthonormal basis and use $\langle v_i, f \rangle$ as the coefficients.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.
- The results often improve drastically if one chooses the sample points x_i as the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \dots, n.$$

- More systematically, one can compute the best approximation of f with respect to a given norm.
  - For the L^2 norm we compute orthonormal basis and use $\langle v_i, f \rangle$ as the coefficients.
  - For the L^∞ norm we can use the equioscillation property to develop efficient algorithms.

Intermediate summary

- We approximate a continuous function $f : [-1, 1] \to \mathbb{R}$ with polynomials.
- A simple (but unreliable) version is to compute the interpolation polynomial for uniformly sampled points $(x_i, f(x_i))$.
- The results often improve drastically if one chooses the sample points x_i as the **Chebyshev roots**

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \ldots, n.$$

- More systematically, one can compute the best approximation of f with respect to a given norm.
  - For the L^2 norm we compute orthonormal basis and use $\langle v_i, f \rangle$ as the coefficients.
  - For the L^∞ norm we can use the equioscillation property to develop efficient algorithms.

Splines

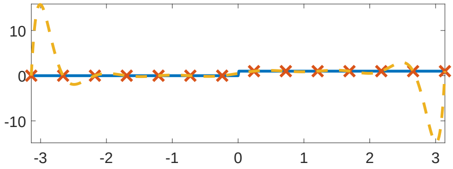
So far, we have mostly considered polynomials to approximate functions and solve interpolation problems.

Michael Moeller
Vaishnavi Gandikota

Splines

So far, we have mostly considered polynomials to approximate functions and solve interpolation problems.

As we have seen in examples like

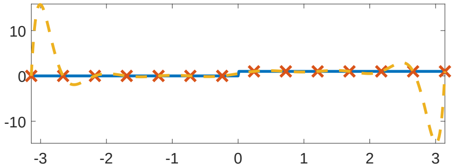


polynomials of high degrees can quickly show oscillatory behavior.

Splines

So far, we have mostly considered polynomials to approximate functions and solve interpolation problems.

As we have seen in examples like



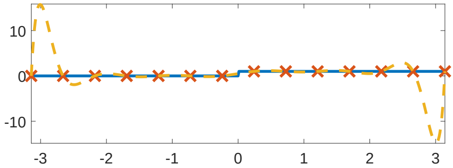
polynomials of high degrees can quickly show oscillatory behavior.

In practice one almost never uses polynomials of high degree (≥ 10) for any interpolation problem.

Splines

So far, we have mostly considered polynomials to approximate functions and solve interpolation problems.

As we have seen in examples like



polynomials of high degrees can quickly show oscillatory behavior.

In practice one almost never uses polynomials of high degree (≥ 10) for any interpolation problem.

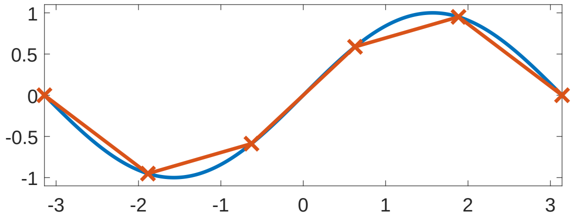
Much smoother version: So called **splines**.

Spline interpolation

While the name **spline** might sound fancy, it is the most natural idea for interpolation: Use a piecewise polynomial function (with polynomials of low degree) for interpolation.

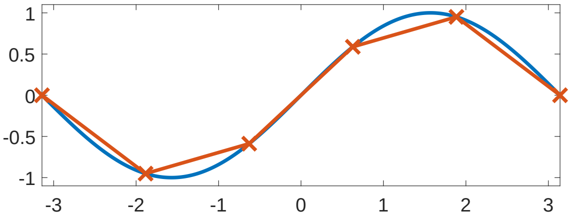
Spline interpolation

While the name **spline** might sound fancy, it is the most natural idea for interpolation: Use a piecewise polynomial function (with polynomials of low degree) for interpolation.



Spline interpolation

While the name **spline** might sound fancy, it is the most natural idea for interpolation: Use a piecewise polynomial function (with polynomials of low degree) for interpolation.



A **linear spline** is nothing but the piecewise linear function

$$s(x) = \frac{x - x_i}{x_{i+1} - x_i} f(x_{i+1}) + \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) f(x_i), \quad \text{for } x \in [x_i, x_{i+1}[.$$

Spline interpolation

The drawback of linear splines is that the resulting approximation s is not differentiable.

Spline interpolation

The drawback of linear splines is that the resulting approximation s is not differentiable.

To have more freedom and impose more regularity one can utilize higher order piecewise polynomial functions.

Michael Moeller
Vaishnavi Gandikota

Spline interpolation

The drawback of linear splines is that the resulting approximation s is not differentiable.

To have more freedom and impose more regularity one can utilize higher order piecewise polynomial functions.

Example: **Cubic splines** are of the form

$$s(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad \text{for } x \in [x_i, x_{i+1}[.$$

Spline interpolation

The drawback of linear splines is that the resulting approximation *s* is not differentiable.

To have more freedom and impose more regularity one can utilize higher order piecewise polynomial functions.

Example: **Cubic splines** are of the form

$$s(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad \text{for } x \in [x_i, x_{i+1}[.$$

For linear splines we had two end-points that fully determined the two coeffients of a linear function. How do we determine the coefficients of cubic splines?

Spline interpolation

Usual choice: So called C^2 **splines**, i.e., a piecewise cubic function that is two times continuously differentiable.

Spline interpolation

Usual choice: So called C^2 **splines**, i.e., a piecewise cubic function that is two times continuously differentiable.

Let $s_{|[x_i,x_{i+1}]}(x) = a_i x^3 + b_i x^2 + c_i x + d_i$. We require

$$s_{|[x_i,x_{i+1}]}(x_i) = y_i \tag{1}$$

$$s_{|[x_i,x_{i+1}]}(x_{i+1}) = y_{i+1} \tag{2}$$

$$s'_{|[x_{i-1},x_i]}(x_i) = s'_{|[x_i,x_{i+1}]}(x_i) \tag{3}$$

$$s''_{|[x_{i-1},x_i]}(x_i) = s''_{|[x_i,x_{i+1}]}(x_i) \tag{4}$$

Spline interpolation

Usual choice: So called C^2 **splines**, i.e., a piecewise cubic function that is two times continuously differentiable.

Let $s_{|[x_i,x_{i+1}]}(x) = a_i x^3 + b_i x^2 + c_i x + d_i$. We require

$$s_{|[x_i,x_{i+1}]}(x_i) = y_i \tag{1}$$
$$s_{|[x_i,x_{i+1}]}(x_{i+1}) = y_{i+1} \tag{2}$$
$$s'_{|[x_{i-1},x_i]}(x_i) = s'_{|[x_i,x_{i+1}]}(x_i) \tag{3}$$
$$s''_{|[x_{i-1},x_i]}(x_i) = s''_{|[x_i,x_{i+1}]}(x_i) \tag{4}$$

We have 4 coefficients for each of the $n-1$ many $s_{|[x_{i-1},x_i]}$, i.e. $4(n-1)$ many unknowns.

Spline interpolation

Usual choice: So called C^2 **splines**, i.e., a piecewise cubic function that is two times continuously differentiable.

Let $s_{|[x_i,x_{i+1}]}(x) = a_i x^3 + b_i x^2 + c_i x + d_i$. We require

$$s_{|[x_i,x_{i+1}]}(x_i) = y_i \tag{1}$$
$$s_{|[x_i,x_{i+1}]}(x_{i+1}) = y_{i+1} \tag{2}$$
$$s'_{|[x_{i-1},x_i]}(x_i) = s'_{|[x_i,x_{i+1}]}(x_i) \tag{3}$$
$$s''_{|[x_{i-1},x_i]}(x_i) = s''_{|[x_i,x_{i+1}]}(x_i) \tag{4}$$

We have 4 coefficients for each of the $n-1$ many $s_{|[x_{i-1},x_i]}$, i.e. $4(n-1)$ many unknowns.

We have $n-1$ many equations from (1),
$n-1$ many equations from (2),
$n-2$ many equations from (3),
$n-2$ many equations from (4),
i.e., $4(n-1)-2$ equations in total.

Michael Moeller
Vaishnavi Gandikota

Spline interpolation

What do we do with the two extra degrees of freedom?

Michael Moeller
Vaishnavi Gandikota

Spline interpolation

What do we do with the two extra degrees of freedom?

Most common choice: **Natural C^2 spline**, i.e., additionally require

$$s'_{|[x_1,x_2]}(x_1) = 0, \qquad \text{(a)}$$

$$s'_{|[x_{n-1},x_n]}(x_n) = 0. \qquad \text{(b)}$$

Michael Moeller
Vaishnavi Gandikota

Spline interpolation

What do we do with the two extra degrees of freedom?

Most common choice: **Natural C^2 spline**, i.e., additionally require

$$s'_{|[x_1,x_2]}(x_1) = 0, \quad \text{(a)}$$
$$s'_{|[x_{n-1},x_n]}(x_n) = 0. \quad \text{(b)}$$

Instructive computation on the board: Set up the linear equations for a natural C^2 spline going through the points

$$(0,0),\ (1,1),\ (2,0).$$

Spline interpolation

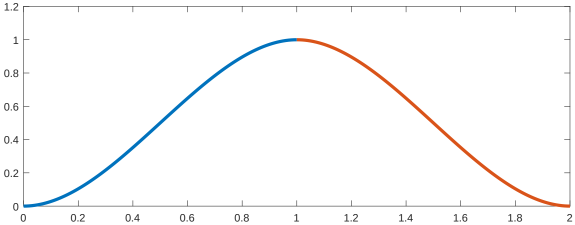
What do we do with the two extra degrees of freedom?

Most common choice: **Natural C^2 spline**, i.e., additionally require

$$s'_{|[x_1,x_2]}(x_1) = 0, \quad \text{(a)}$$
$$s'_{|[x_{n-1},x_n]}(x_n) = 0. \quad \text{(b)}$$

Instructive computation on the board: Set up the linear equations for a natural C^2 spline going through the points

$$(0,0),\ (1,1),\ (2,0).$$



Integration

Integration in 1d

Let us first consider the $1d$ integral

$$\int_a^b f(x)\,dx.$$

Integration in 1d

Let us first consider the $1d$ integral

$$\int_a^b f(x)\,dx.$$

The general strategy is to approximate f by a function that is easy to integrate, e.g., by a spline function and use the integral over the approximate function instead!

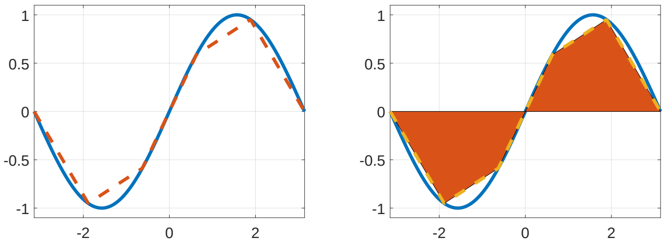
Integration in 1d

Let us first consider the $1d$ integral

$$\int_a^b f(x)\,dx.$$

The general strategy is to approximate f by a function that is easy to integrate, e.g., by a spline function and use the integral over the approximate function instead!

Example: Trapezoidal rule



Trapezoidal Rule

To approximate a function f linearly on an interval $[t_{i-1},\ t_i]$ one uses

$$f^i_{approx}(x) = \frac{t_i - x}{t_i - t_{i-1}} f(t_{i-1}) + \left(1 - \frac{t_i - x}{t_i - t_{i-1}}\right) f(t_i).$$

Trapezoidal Rule

To approximate a function f linearly on an interval $[t_{i-1},\ t_i]$ one uses

$$f^i_{approx}(x) = \frac{t_i - x}{t_i - t_{i-1}} f(t_{i-1}) + \left(1 - \frac{t_i - x}{t_i - t_{i-1}}\right) f(t_i).$$

One finds that

$$\begin{aligned}
&\int_{t_{i-1}}^{t_i} f^i_{approx}(x)\ dx \\
&= \int_{t_{i-1}}^{t_i} f(t_i)\ dx + (f(t_i) - f(t_{i-1})) \int_{t_{i-1}}^{t_i} \frac{t_i - x}{t_i - t_{i-1}}\ dx \\
&= \int_{t_{i-1}}^{t_i} f(t_i)\ dx + (f(t_i) - f(t_{i-1})) \left[\frac{xt_i - 0.5x^2}{t_i - t_{i-1}}\right]_{t_{i-1}}^{t_i} \\
&= (t_i - t_{i-1}) f(t_i) + (f(t_i) - f(t_{i-1})) \frac{0.5t_i^2 - t_i t_{i-1} + 0.5t_i^2}{t_i - t_{i-1}} \\
&= (t_i - t_{i-1}) f(t_i) + \frac{1}{2}(f(t_i) - f(t_{i-1}))(t_i - t_{i-1}) \\
&= \frac{1}{2}(f(t_i) + f(t_{i-1}))(t_i - t_{i-1})
\end{aligned}$$

Trapezoidal Rule

Therefore one approximates

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} \int_{t_{i-1}}^{t_i} f^i_{approx}(x)\,dx$$

for $t_0 = a$ and $t_i = a + h\,i$, $h = \frac{b-a}{n}$.

Trapezoidal Rule

Therefore one approximates

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} \int_{t_{i-1}}^{t_i} f^i_{approx}(x)\,dx$$

for $t_0 = a$ and $t_i = a + h\,i$, $h = \frac{b-a}{n}$.

We find

$$\int_a^b f(x)\,dx \approx h\left(\frac{f(t_0)}{2} + \frac{f(t_n)}{2} + \sum_{i=1}^{n-1} f(t_i)\right)$$

the **trapezoidal rule**.

Higher order approximation

As long as the function f one tries to approximate is smooth (varying slowly) the accuracy of the integral improves as one approximates the function within each interval by a higher order polynomial.

Higher order approximation

As long as the function f one tries to approximate is smooth (varying slowly) the accuracy of the integral improves as one approximates the function within each interval by a higher order polynomial.

If the interval $[c, d]$ is sampled at points

$$x_i = c + h \cdot \frac{i}{n}, \qquad h = d - c, \qquad i = 0, \dots, n$$

and one fits a polynomial of degree n through them,

Higher order approximation

As long as the function f one tries to approximate is smooth (varying slowly) the accuracy of the integral improves as one approximates the function within each interval by a higher order polynomial.

If the interval $[c, d]$ is sampled at points

$$x_i = c + h \cdot \frac{i}{n}, \qquad h = d - c, \qquad i = 0, \ldots, n$$

and one fits a polynomial of degree n through them,one obtains the **Newton-Cotes** formula

$$\int_c^d f(x)\, dx = h \sum_{i=1}^{n} \alpha_i f(x_i),$$

with α_i being the integrals over the **Lagrange polynomials**,

$$\alpha_i = \frac{1}{h} \int_c^d \prod_{i=0, i \neq j}^{n} \frac{x - x_j}{x_i - x_j}\, dx.$$

Higher order approximation

n	name	x_i	α_i
1	trapezoidal rule	0, 1	$\frac{1}{2}, \frac{1}{2}$
2	Simpson rule	$0, \frac{1}{2}, 1$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$
3	$\frac{3}{8}$-rule	$0, \frac{1}{3}, \frac{2}{3}, 1$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$

Higher order approximation

n	name	x_i	α_i
1	trapezoidal rule	$0, 1$	$\frac{1}{2}, \frac{1}{2}$
2	Simpson rule	$0, \frac{1}{2}, 1$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$
3	$\frac{3}{8}$-rule	$0, \frac{1}{3}, \frac{2}{3}, 1$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$

Example approximation by applying the Simpson rule to each interval $[t_{i-1}, t_i]$:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left(f(t_0) + 4f\left(\frac{t_1-t_0}{2}\right) + 2f(t_1) + 4f\left(\frac{t_2-t_1}{2}\right) + \ldots + 2f(t_{n-1}) + 4f\left(\frac{t_2-t_1}{2}\right) + f(t_n)\right)$$

Higher order approximation

The **Newton-Cotes** formulas based on

$$\int_c^d f(x)\, dx = h \sum_{i=1}^{n} \alpha_i f(x_i),$$

is exact for f being a polynomial of order n (at least).

Higher order approximation

The **Newton-Cotes** formulas based on

$$\int_c^d f(x)\,dx = h\sum_{i=1}^{n} \alpha_i f(x_i),$$

is exact for f being a polynomial of order n (at least).

One can also optimize the weights α_i and the points x_i at which the function is evaluated for being exact for a polynomial of maximal order. Surprisingly, one can get exact polynomial integration of order $2n + 1$ with only $n + 1$ evaluations.

Higher order approximation

The **Newton-Cotes** formulas based on

$$\int_c^d f(x)\,dx = h\sum_{i=1}^n \alpha_i f(x_i),$$

is exact for f being a polynomial of order n (at least).

One can also optimize the weights α_i and the points x_i at which the function is evaluated for being exact for a polynomial of maximal order. Surprisingly, one can get exact polynomial integration of order $2n+1$ with only $n+1$ evaluations.

Such formulas are known under the name of **Gauß quadrature**. We will not detail those more sophisticated methods here - in image processing simple piecewise constant or piecewise linear approximations are used most often.

Higher dimensional integration

In imaging, we often have to integrate functions $f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ which correspond to images and are therefore two-dimensional.

Higher dimensional integration

In imaging, we often have to integrate functions $f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ which correspond to images and are therefore two-dimensional.

Note that we can reduce multi-dimensional integrals to multiple one-dimensional integrals, e.g., if $\Omega = [a, b] \times [a, b]$, then

$$\int_\Omega f(x)\ dx = \int_a^b \int_a^b f(x_1, x_2)\ dx_1\ dx_2.$$

Higher dimensional integration

In imaging, we often have to integrate functions $f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ which correspond to images and are therefore two-dimensional.

Note that we can reduce multi-dimensional integrals to multiple one-dimensional integrals, e.g., if $\Omega = [a, b] \times [a, b]$, then

$$\int_\Omega f(x)\ dx = \int_a^b \int_a^b f(x_1, x_2)\ dx_1\ dx_2.$$

Note that the common division of $[a, b]$ into $[a = t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{n-1}, t_n]$ now (at least) requires the evaluation of f at all $(n+1)^2$ points $f(t_i, t_j)$.

Michael Moeller
Vaishnavi Gandikota

Higher dimensional integration

In imaging, we often have to integrate functions $f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ which correspond to images and are therefore two-dimensional.

Note that we can reduce multi-dimensional integrals to multiple one-dimensional integrals, e.g., if $\Omega = [a, b] \times [a, b]$, then

$$\int_\Omega f(x)\ dx = \int_a^b \int_a^b f(x_1, x_2)\ dx_1\ dx_2.$$

Note that the common division of $[a, b]$ into $[a = t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{n-1}, t_n]$ now (at least) requires the evaluation of f at all $(n+1)^2$ points $f(t_i, t_j)$.

Since the complexity grows exponentially with the dimension of the space, other techniques (e.g. Monte Carlo methods) have to be applied for dimensions significantly larger than 2.

Line integrals

Sometimes, one has to evaluate the length of a path:

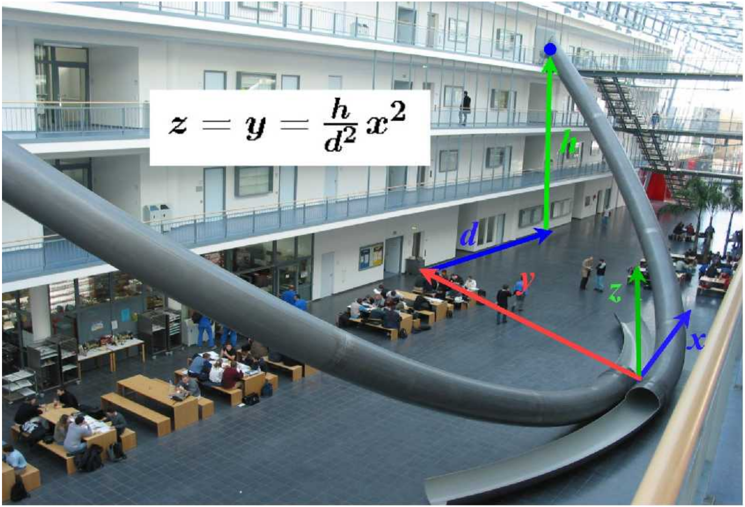
Line integrals

Sometimes, one has to evaluate the length of a path:

From https://en.wikipedia.org/wiki/Technical_University_of_Munich

Michael Moeller
Vaishnavi Gandikota

Line integrals



From https://www.ma.tum.de/Mathematik/Parabelrutsche

Step 1: Parameterize the path!

Line integrals

We can find a parametrization of the curve in 3d:

$$r : [0, d] \to \mathbb{R}^3$$
$$t \mapsto \left[t, \frac{h}{d^2}t^2, \frac{h}{d^2}t^2\right]$$

Line integrals

We can find a parametrization of the curve in 3d:

$$r : [0, d] \to \mathbb{R}^3$$
$$t \mapsto \left[t, \frac{h}{d^2}t^2, \frac{h}{d^2}t^2\right]$$

Line integral

The **line integral** of a function $f : U \subset \mathbb{R}^n \to \mathbb{R}$ along a piecewise smooth curve $C \subset U$ is defined as

$$\int_a^b f(r(t)) \cdot |r'(t)| \, dt$$

where $r : [a, b] \to C$ is an arbitrary bijective parametrization of the curve C such that $r(a)$ and $r(b)$ give the endpoints of C and $a < b$.

Line integrals

We can find a parametrization of the curve in 3d:

$$r : [0, d] \to \mathbb{R}^3$$
$$t \mapsto \left[t, \frac{h}{d^2} t^2, \frac{h}{d^2} t^2 \right]$$

Line integral

The **line integral** of a function $f : U \subset \mathbb{R}^n \to \mathbb{R}$ along a piecewise smooth curve $C \subset U$ is defined as

$$\int_a^b f(r(t)) \cdot |r'(t)| \, dt$$

where $r : [a, b] \to C$ is an arbitrary bijective parametrization of the curve C such that $r(a)$ and $r(b)$ give the endpoints of C and $a < b$.

Let's do this for the above example of deterimining the length, i.e., $f \equiv 1$.

Line integral

We find

$$r'(t) = \left(1, \frac{2h}{d^2}t, \frac{2h}{d^2}t\right)^T$$

Michael Moeller
Vaishnavi Gandikota

Line integral

We find

$$r'(t) = \left(1, \frac{2h}{d^2}t, \frac{2h}{d^2}t\right)^T$$

and therefore

$$\int_0^d \sqrt{1^2 + 2\frac{4h^2}{d^4}t^2}\, dt \qquad \text{(a)}$$

is the length of the slide in Munich.

Line integral

We find

$$r'(t) = \left(1, \frac{2h}{d^2}t, \frac{2h}{d^2}t\right)^T$$

and therefore

$$\int_0^d \sqrt{1^2 + 2\frac{4h^2}{d^4}t^2}\, dt \qquad \text{(a)}$$

is the length of the slide in Munich.

If $|r'(t)|$ is too complicated to compute, we can of course also use a polygonal chain to approximate the length directly.

Line integral

We find

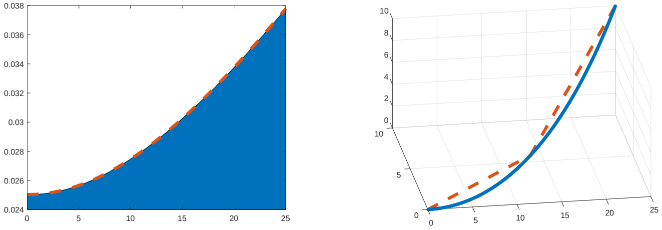
$$r'(t) = \left(1, \frac{2h}{d^2}t, \frac{2h}{d^2}t\right)^T$$

and therefore

$$\int_0^d \sqrt{1^2 + 2\frac{4h^2}{d^4}t^2}\,dt \qquad \text{(a)}$$

is the length of the slide in Munich.

If $|r'(t)|$ is too complicated to compute, we can of course also use a polygonal chain to approximate the length directly.



Using (a) Using polygons

The end

Any questions about anything?