Interpolation in $D > 2$ and Extrapolation

Marcin Chrzaszcz, Danny van Dyk
mchrzasz@cern.ch, danny.van.dyk@gmail.com

University of Zurich[UZH]

Numerical Methods,
26. September, 2016

Plan for today

- When does interpolation fail?
  Marcin showed you how to interpolate in $D = 1$. What are the pitfalls? And can we circumvent them?

- How to interpolate a function of $D > 1$ variables?
  Is there are sensible generalization of interpolation to multivariate functions? If yes, let's apply what we learned before to $D = 2$.

- What happens when I use my interpolation beyond the domain of data?
  Why would I do that? When can I do that? How well does it work, and how can I improve it?

Does interpolation always work?

Weierstrass Approximation Theorem

Suppose f is a continuous real-valued function defined on the real interval $[a, b]$. For every $\varepsilon > 0$, there exists a polynomial $P(x)$ such that for all x in $[a, b]$, we have $R[P] \equiv \max_{a \leq x \leq b} |f(x) - P(x)| < \varepsilon$, $[\ldots]$

Using interpolation, we can construct a polynomial $P_n(x)$ of degree n that approximates f up to an approximation error of R_n:

$$R_n \equiv \max_{a \leq x \leq b} |f(x) - P_n(x)| \ .$$

Conclusion Weierstrass's theorem says that there is *at least one* polynomial for each choice of the residual error ε. It does not say that either

- P_n is *in* the set of polynomial for a given ε, or
- $R_n \to 0$ if $n \to \infty$!

Increasing n might be harmful!

When Interpolation fails

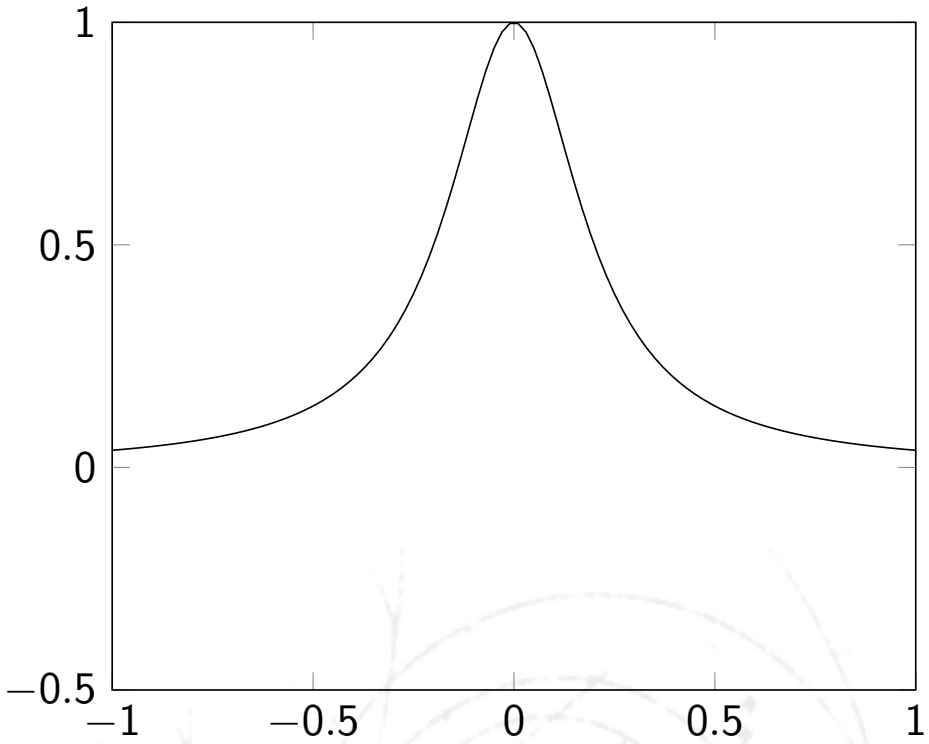
So far, you have been confronted with examples in which interpolation works nicely. Let us now discuss an example, which behaves pathologically.

Consider the classical example by Runge:

$$f(x) = \left[1 + 25x^2\right]^{-1}$$

Let us plot

- the true function $f(x)$



When Interpolation fails

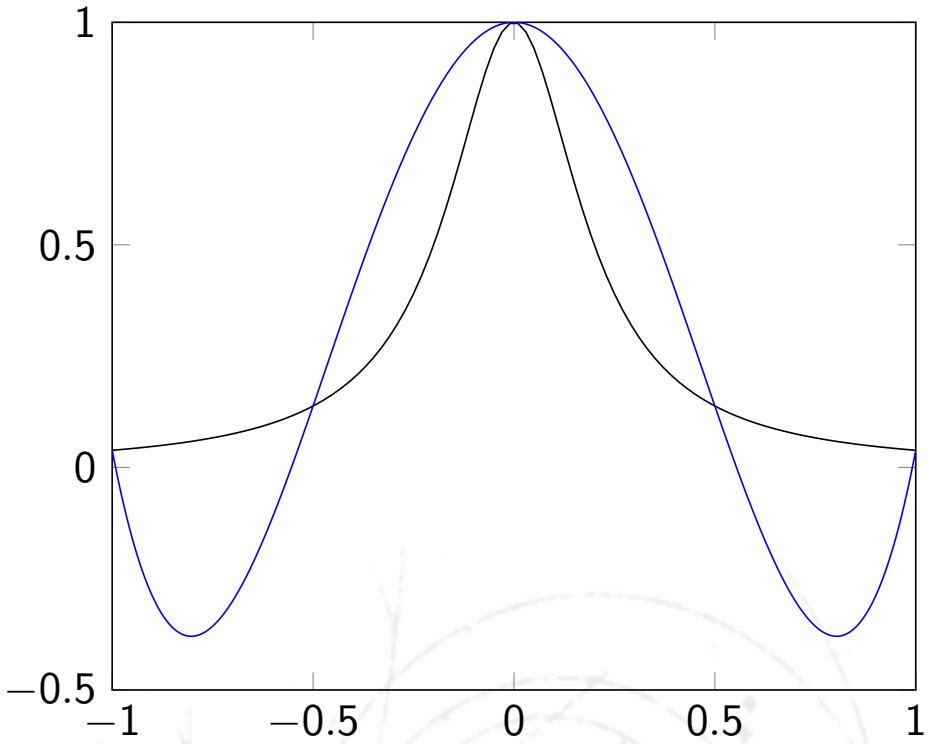
So far, you have been confronted with examples in which interpolation works nicely. Let us now discuss an example, which behaves pathologically.

Consider the classical example by Runge:

$$f(x) = \left[1 + 25x^2\right]^{-1}$$

Let us plot

- the true function $f(x)$
- the interpolating polynomial to degree 4



for equidistant interpolation points on $[-1, +1]$.

When Interpolation fails

So far, you have been confronted with examples in which interpolation works nicely. Let us now discuss an example, which behaves pathologically.

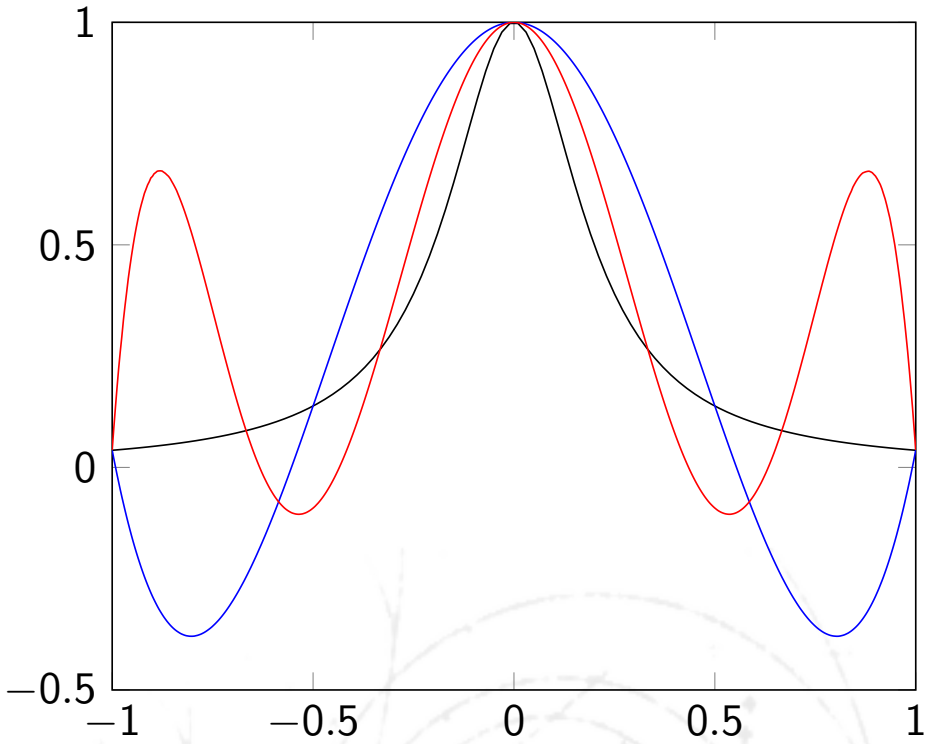
Consider the classical example by Runge:

$$f(x) = \left[1 + 25x^2\right]^{-1}$$

Let us plot

- the true function $f(x)$
- the interpolating polynomial to degree 4
- the interpolating polynomial to degree 6

for equidistant interpolation points on $[-1, +1]$.



When Interpolation fails

So far, you have been confronted with examples in which interpolation works nicely. Let us now discuss an example, which behaves pathologically.

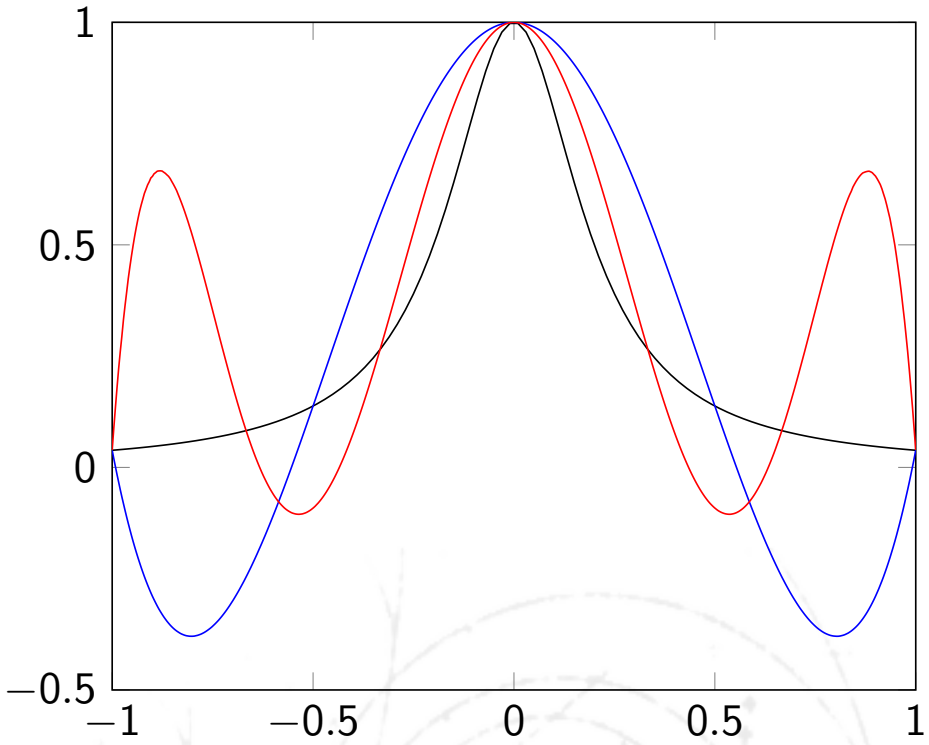
Consider the classical example by Runge:

$$f(x) = \left[1 + 25x^2\right]^{-1}$$

Let us plot

- the true function $f(x)$
- the interpolating polynomial to degree 4
- the interpolating polynomial to degree 6

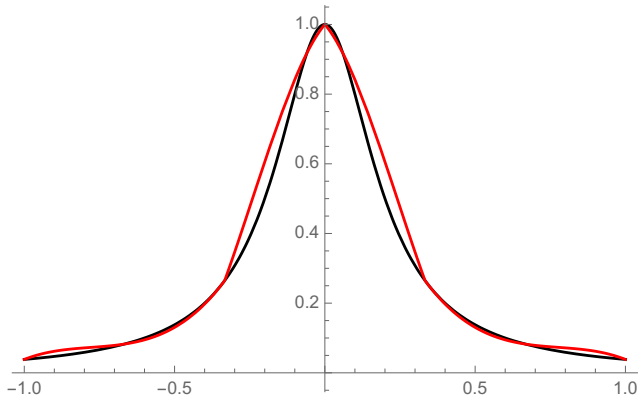
for equidistant interpolation points on $[-1, +1]$.



$R_n \to \infty$ as $n \to \infty$

Pieceswise linear/cubic/... interpolations

This pathological behaviour can be avoided by piecewise interpolation with a small value of the polynomial degree n. Most popular are linear ($n = 1$) or cubic ($n = 3$) piecewise interpolations ("splines"), which do not suffer from the problem that was just described.



The concrete approaches have been discussed last lecture by Marcin.

Interpolation in more than $D = 1$ dimensions

In lecture 2 we discussed various ways to interpolate a univariate function $f(x)$. A very nice fact for polynomial interpolations in $D = 1$ dimension is that the interpolating polynomial is *unique*: For two polynomials $g(x)$ and $h(x)$ of identical degree n one has

$$f_i = g(x_i) = h(x_i) \quad \Rightarrow \quad g(x) \equiv h(x)\,,$$

when considering exactly $\rho_n(D = 1) = n + 1$ interpolation points x_i, for $i = 1, \ldots, n + 1$.

We will now investigate if this still holds for $D > 1$ dimensions.

Interpolation in $D = 2$ dimensions

We will use a bivariate polynomial of degree n:

$$P_n(x, y) = \sum_{i,j}^{i+j \leq n} a_{i,j} x^i y^j \,.$$

This polynomial has exactly $\rho_n(D = 2) = \binom{n+2}{n}$ coefficients, which need to be computed from the interpolation points.
If we evaluate exactly $\rho_n(D = 2)$ points, the system of linear equations has exactly one or zero solutions!

Examples

$$P_1(x, y) = a_{0,0} + a_{1,0} x + a_{0,1} y \,, \qquad \binom{1+2}{1} = 3 \,,$$

$$P_2(x, y) = a_{0,0} + a_{1,0} x + a_{0,1} y \,, \qquad \binom{2+2}{2} = 6 \,,$$
$$+ a_{1,1} xy + a_{2,0} x^2 + a_{0,2} y^2$$

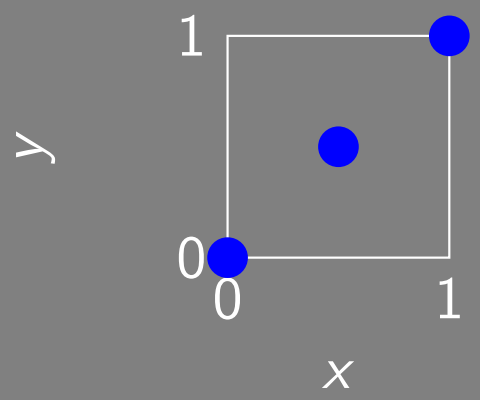
Arrangement of points in $D = 2$

The question is now: How do we choose the $\rho_n(D = 2)$ points that shall be interpolated?

In $D = 1$, this was easy: all points were on the x axis. In $D = 2$ we need to populate a plane, and this gives us more freedom.

How to arrange the interpolation points in the plane? Let's consider two examples for $D = 2$ dimensions for a $n = 1$ polynomial:

Example #1



The interpolation equation reads:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{0,1} \end{bmatrix} = \begin{bmatrix} f(0,0) \\ f(\frac{1}{2},\frac{1}{2}) \\ f(1,1) \end{bmatrix}$$

The Vandermonde matrix is singular, and no polynomial exists that interpolates the blue points.

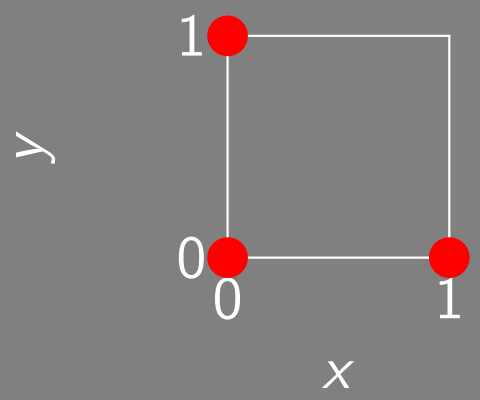
Arrangement of points in $D = 2$

The question is now: How do we choose the $\rho_n(D = 2)$ points that shall be interpolated?

In $D = 1$, this was easy: all points were on the x axis. In $D = 2$ we need to populate a plane, and this gives us more freedom.

How to arrange the interpolation points in the plane? Let's consider two examples for $D = 2$ dimensions for a $n = 1$ polynomial:

Example #2



The interpolation equation reads:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{0,1} \end{bmatrix} = \begin{bmatrix} f(0,0) \\ f(\frac{1}{2},\frac{1}{2}) \\ f(1,1) \end{bmatrix}$$

The Vandermonde matrix is regular, and exactly one polynomial exists that interpolates the red points.

Divide and conquer $D = 2$

We had seen that increasing the degree n does not necessarily reduce the approximation error R_n, even in $D = 1$ dimensions.

In $D = 2$ this problem can become even more serious.

It is therefore a good idea to evaluate the function f on a regular grid in the (x, y) plane. Subequently, one can interpolate within each cell of the grid. This is the $D = 2$ analog to interpolation with splines:

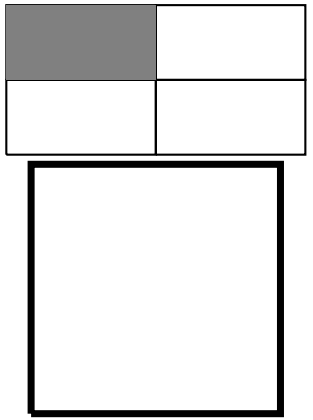
- linear splines $\rightarrow$ bilinear splines
- cubic splines $\rightarrow$ bicubic splines

Algorithm for Bilinear Interpolation

1. Create a rectilinear grid in the (x, y) plane

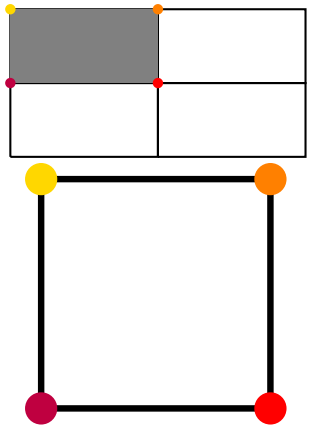
Algorithm for Bilinear Interpolation

1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square



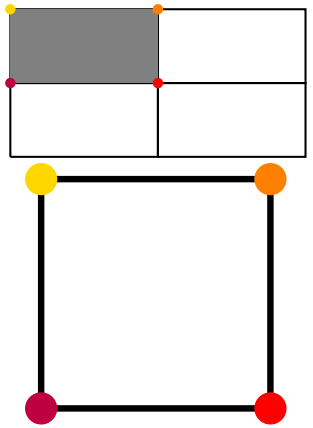
Algorithm for Bilinear Interpolation

1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square
3. Evaluate the function f on the four corners of the (mapped) unit square $Q_{0,0}$, $Q_{1,0}$, $Q_{1,1}$, $Q_{0,1}$.

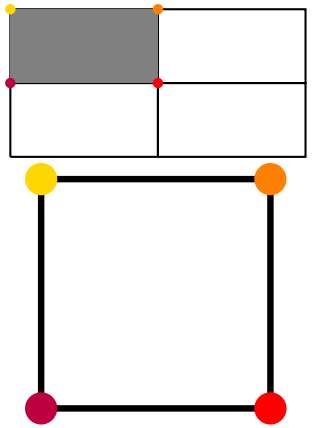


Algorithm for Bilinear Interpolation

1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square
3. Evaluate the function f on the four corners of the (mapped) unit square $Q_{0,0}$, $Q_{1,0}$, $Q_{1,1}$, $Q_{0,1}$.
4. Make an ansatz:
   $P(x, y) = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{1,1}xy$.



Algorithm for Bilinear Interpolation

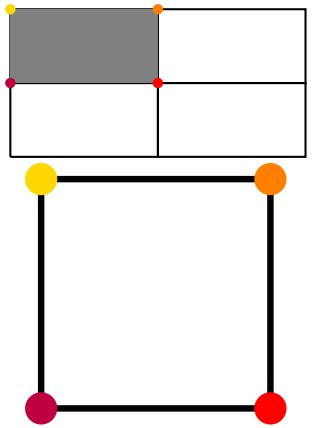


1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square
3. Evaluate the function f on the four corners of the (mapped) unit square $Q_{0,0}$, $Q_{1,0}$, $Q_{1,1}$, $Q_{0,1}$.
4. Make an ansatz:
   $P(x, y) = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{1,1}xy$.
5. Solve the interpolation equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{0,1} \\ a_{1,1} \end{bmatrix} = \begin{bmatrix} f(Q_{0,0}) \\ f(Q_{1,0}) \\ f(Q_{0,1}) \\ f(Q_{1,1}) \end{bmatrix}$$

We only need to invert the Vandermonde matrix once!

Algorithm for Bilinear Interpolation

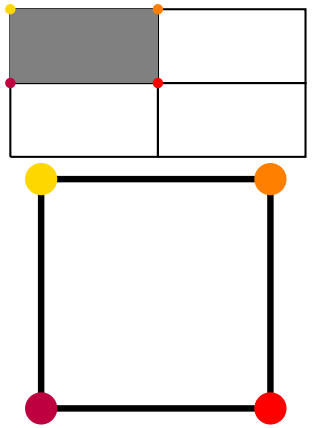


1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square
3. Evaluate the function f on the four corners of the (mapped) unit square $Q_{0,0}$, $Q_{1,0}$, $Q_{1,1}$, $Q_{0,1}$.
4. Make an ansatz:
   $P(x, y) = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{1,1}xy$.
5. Solve the interpolation equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{0,1} \\ a_{1,1} \end{bmatrix} = \begin{bmatrix} f(Q_{0,0}) \\ f(Q_{1,0}) \\ f(Q_{0,1}) \\ f(Q_{1,1}) \end{bmatrix}$$

   We only need to invert the Vandermonde matrix once!
6. Map the result back from the unit square back to the grid piece in (x, y) plane.

Algorithm for Bilinear Interpolation



1. Create a rectilinear grid in the (x, y) plane
2. For each rectangle, map the rectangle to the unit square
3. Evaluate the function f on the four corners of the (mapped) unit square $Q_{0,0}$, $Q_{1,0}$, $Q_{1,1}$, $Q_{0,1}$.
4. Make an ansatz:
   $P(x, y) = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{1,1}xy$.
5. Solve the interpolation equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{0,1} \\ a_{1,1} \end{bmatrix} = \begin{bmatrix} f(Q_{0,0}) \\ f(Q_{1,0}) \\ f(Q_{0,1}) \\ f(Q_{1,1}) \end{bmatrix}$$

   We only need to invert the Vandermonde matrix once!
6. Map the result back from the unit square back to the grid piece in (x, y) plane.
7. Jump back to #2.

Interpolation in D dimensions

The situation becomes even more complicated if you go to $D > 2$ dimensions:

$D = 3$ You can generalize to with either trilinear or tricubic splines, on a rectilinear $3D$ grid. The Vandermonde matrix on the unit cube is now $\binom{n+3}{n}^2$.

arbitrary D There is no polynomial interpolation for D dimensions.

Extrapolation Basics

In many numerical applications a common class of problems arises: Concerning a function $f(x)$ that is expensive to evaluate, we are interested in the value $f(x_0)$. However, at x_0 the function $f(x)$ is numerically instable, or the problem maybe even ill-posed.

Sometime there is an environment around x_0, $x \approx x_0 + h$, in which we can evaluate f. Usually, one now discusses $f(h) \equiv f(x_0 + h)$, or rather the limit $\lim_{h \to 0} f(h)$. [Note: in all generality we can map problems with limits to either $h \to \infty$ or $h \to$ finite value onto limits $h \to 0$.]

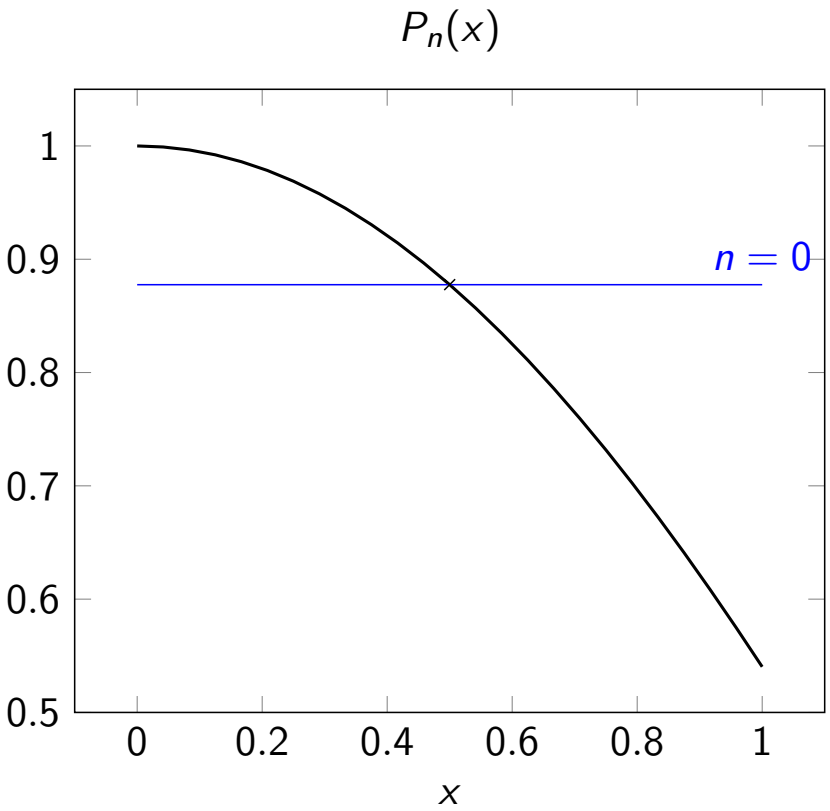
Interpolation, as discussed previously, can not directly help, since $h = 0$ is not part of the domain of data points. Instead, one can take an interpolation at finite $h > 0$, $f_{\text{int}}(h)$, and simply approximate $f(h = 0) \approx f_{\text{int}}(h = 0)$. This step of using the interpolation of f outside the domain of data points is called *extrapolation*.

To extrapolate an arbitrary function might work very well, but also might fail spectacularly. In this part of the course, we will briefly discuss examples of both cases, and what mathematical requirements make extrapolations work.

Working example

Consider the function $\cos(x)$. Our task at hand is to estimate $\cos(0)$ through evaluations of $\cos(h_k)$, with $h_k > 0$ but $h_k \to 0$ with increasing k.

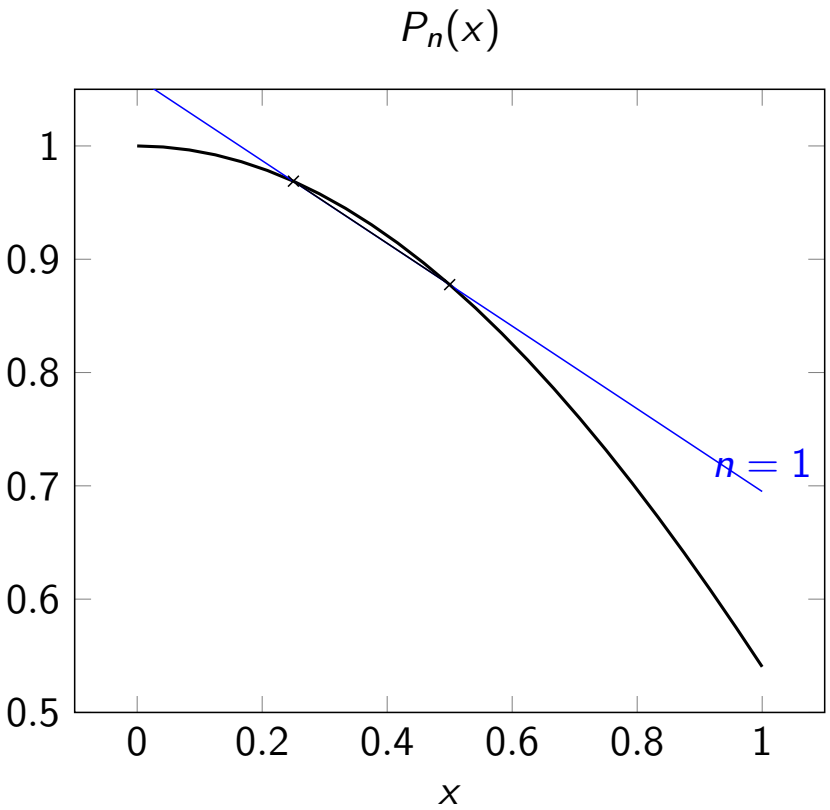
Let $h_k = 2^{-k}$, and construct interpolating polynomials P_n that interpolate $\cos(x)$ in $\{h_1, \ldots, h_{n+1}\}$.



Working example

Consider the function $\cos(x)$. Our task at hand is to estimate $\cos(0)$ through evaluations of $\cos(h_k)$, with $h_k > 0$ but $h_k \to 0$ with increasing k.

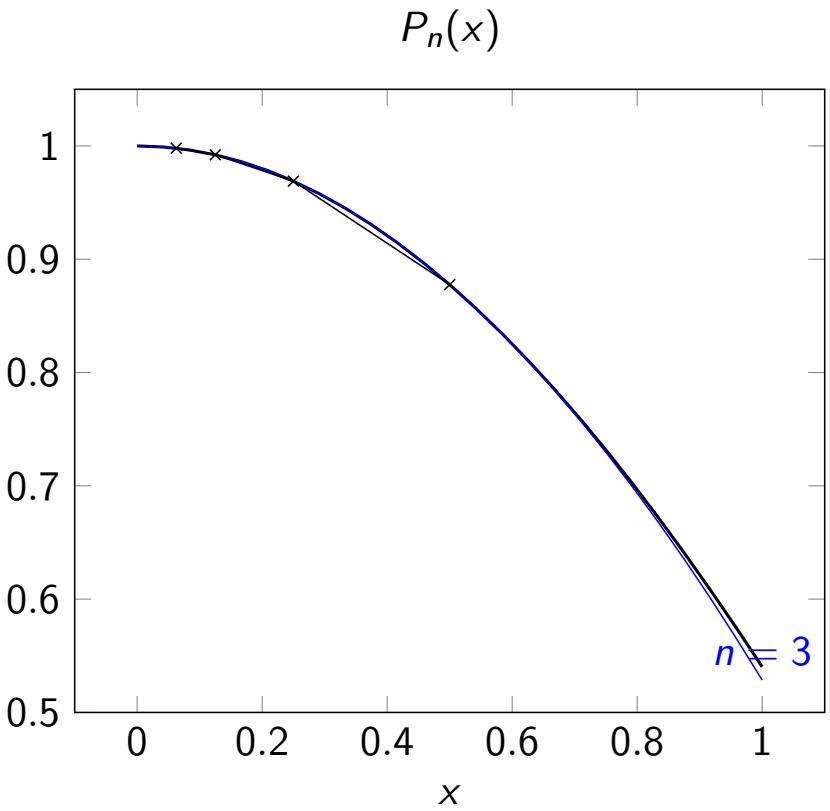
Let $h_k = 2^{-k}$, and construct interpolating polynomials P_n that interpolate $\cos(x)$ in $\{h_1, \ldots, h_{n+1}\}$.



Working example

Consider the function $\cos(x)$. Our task at hand is to estimate $\cos(0)$ through evaluations of $\cos(h_k)$, with $h_k > 0$ but $h_k \to 0$ with increasing k.

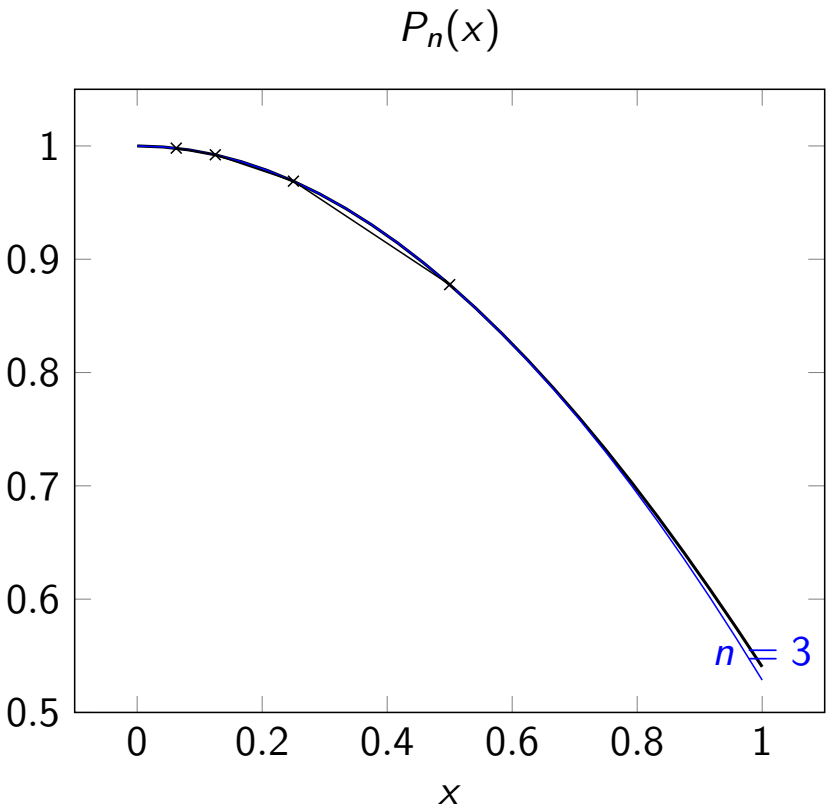
Let $h_k = 2^{-k}$, and construct interpolating polynomials P_n that interpolate $\cos(x)$ in $\{h_1, \ldots, h_{n+1}\}$.

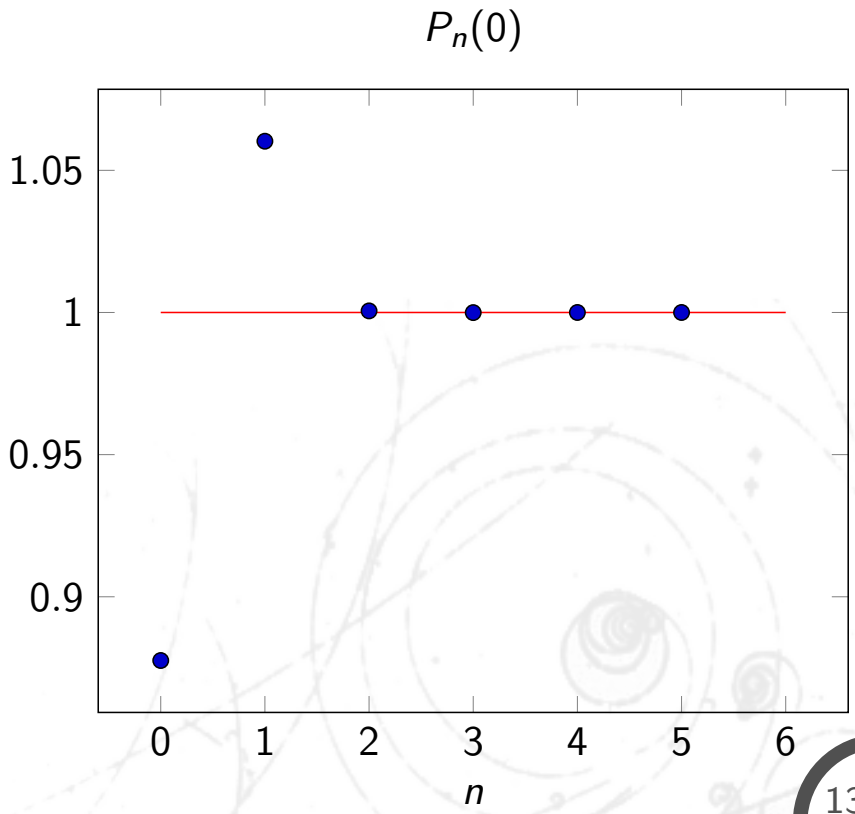


Working example

Consider the function $\cos(x)$. Our task at hand is to estimate $\cos(0)$ through evaluations of $\cos(h_k)$, with $h_k > 0$ but $h_k \to 0$ with increasing k.

Let $h_k = 2^{-k}$, and construct interpolating polynomials P_n that interpolate $\cos(x)$ in $\{h_1, \ldots, h_{n+1}\}$.



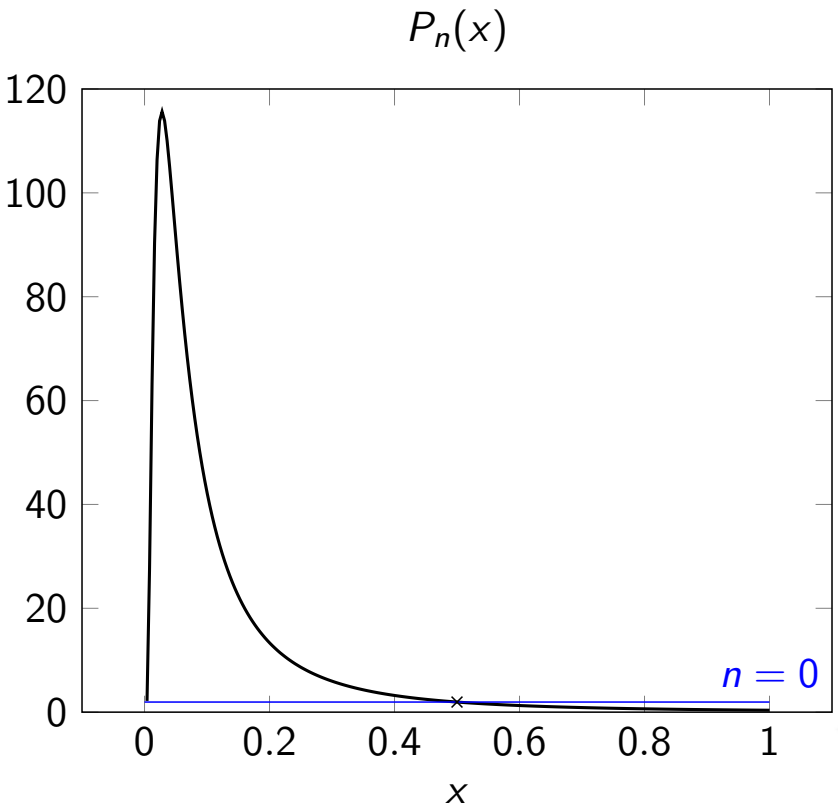


Pathological example

Consider the example:

$$f(x) = \exp(-x^{-1/2})/x^4\,, \quad \text{with} \quad \lim_{x\to 0} f(x) = 0$$

We use the same procedure as before:

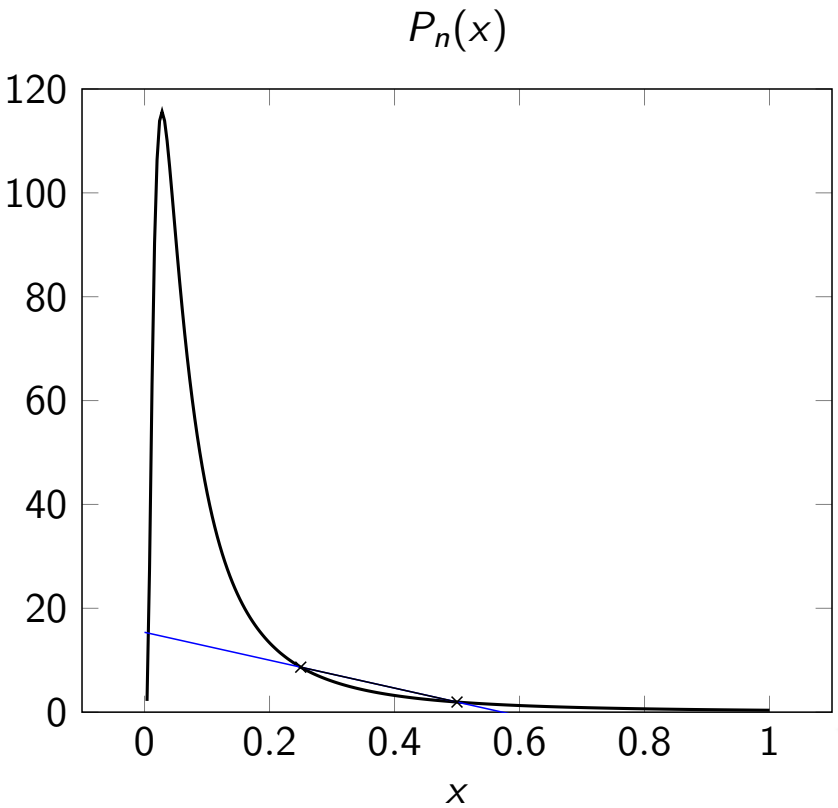


Pathological example

Consider the example:

$$f(x) = \exp(-x^{-1/2})/x^4\,, \quad \text{with} \quad \lim_{x\to 0} f(x) = 0$$

We use the same procedure as before:

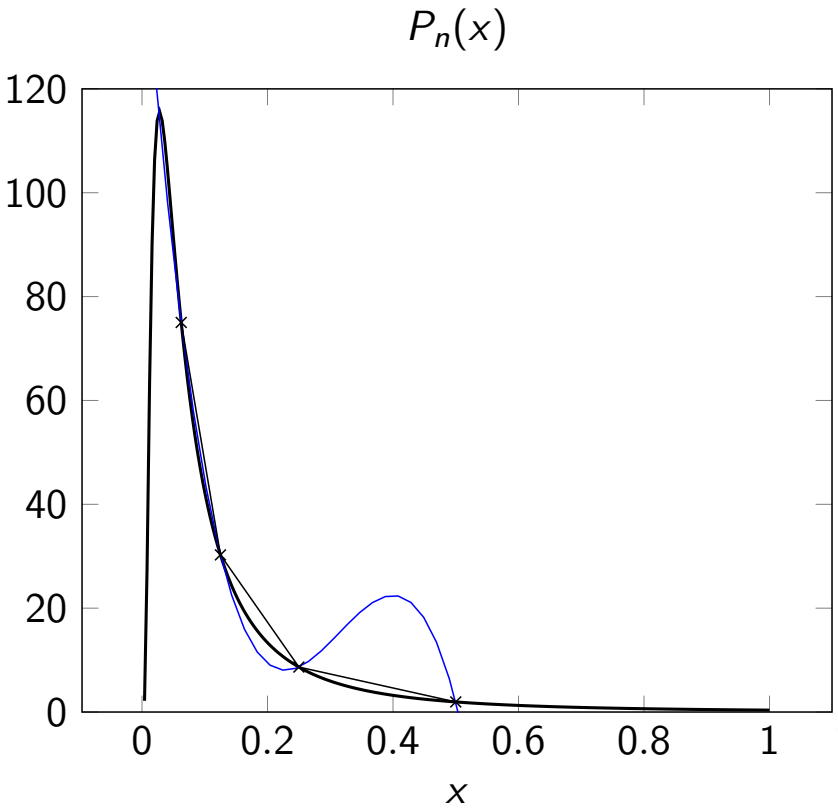


Pathological example

Consider the example:

$$f(x) = \exp(-x^{-1/2})/x^4\,, \quad \text{with} \quad \lim_{x\to 0} f(x) = 0$$

We use the same procedure as before:

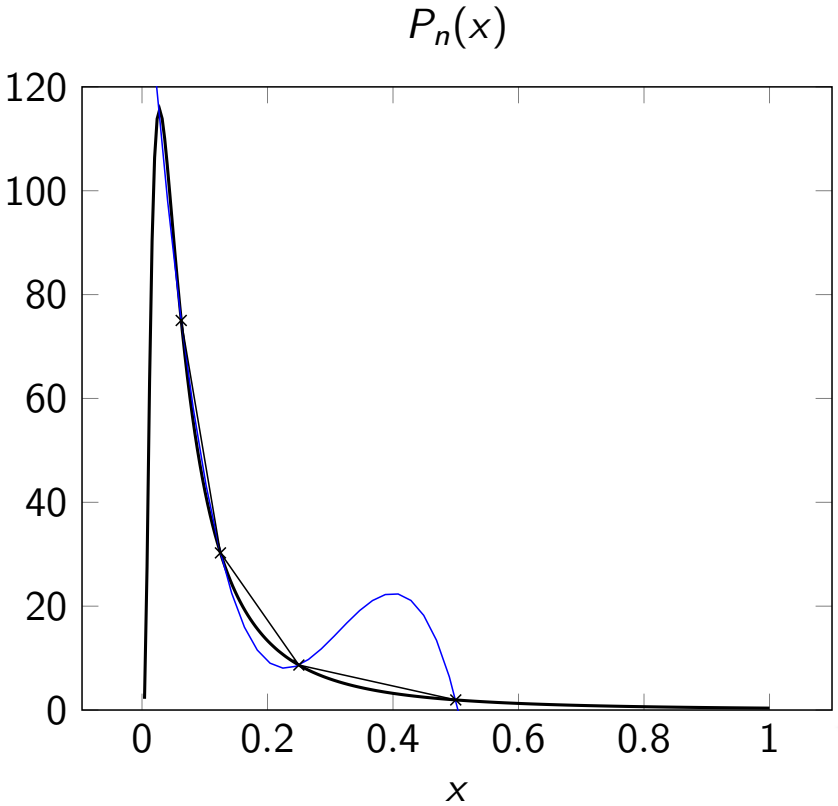


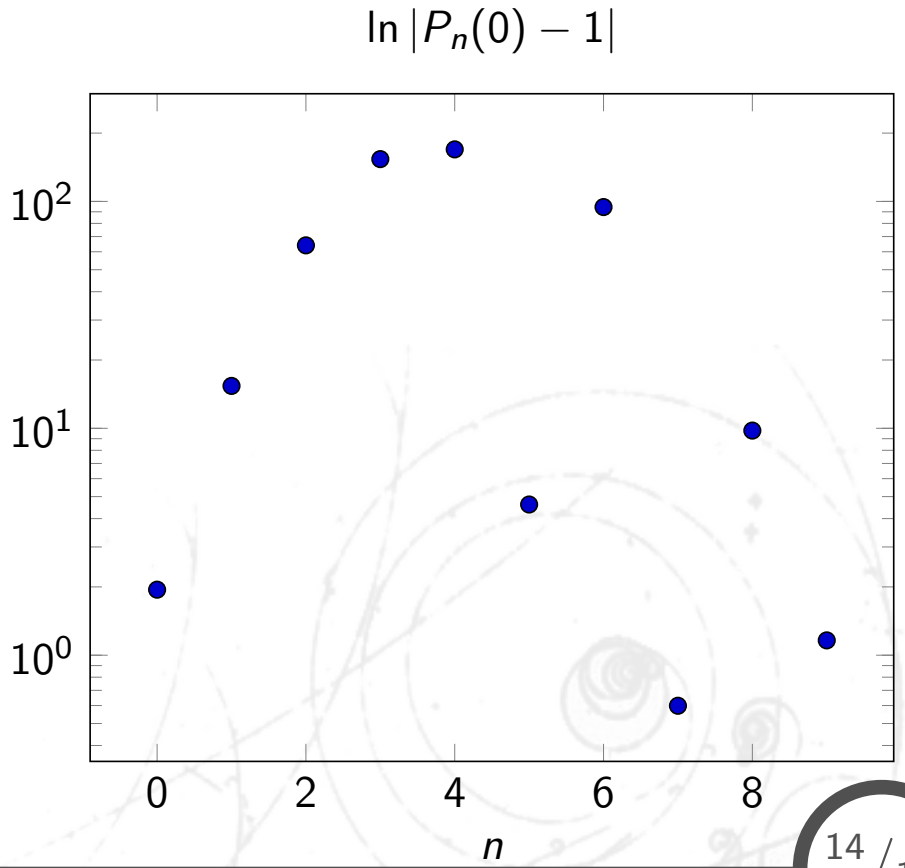
Pathological example

Consider the example:

$$f(x) = \exp(-x^{-1/2})/x^4\,, \quad \text{with} \quad \lim_{x\to 0} f(x) = 0$$

We use the same procedure as before:





Foundations: Necessary prerequisite for extrapolation

In order for the extrapolation to work, the function $f(x)$ needs to fulfill a necessary prerequisite:

- existence of an asymptotic expansion

Asymptotic Expansion

A function $f(x)$ can be asymptotically expanded in a sequence $\phi_n(x)$ around $x \simeq x_0$,

$$f(x) = \sum_n^N a_n \phi_n(x)\,,$$

if

$$f(x) - \sum_n^{N-1} a_n \phi_n(x) = o(\phi_{N-1}(x))\,.$$

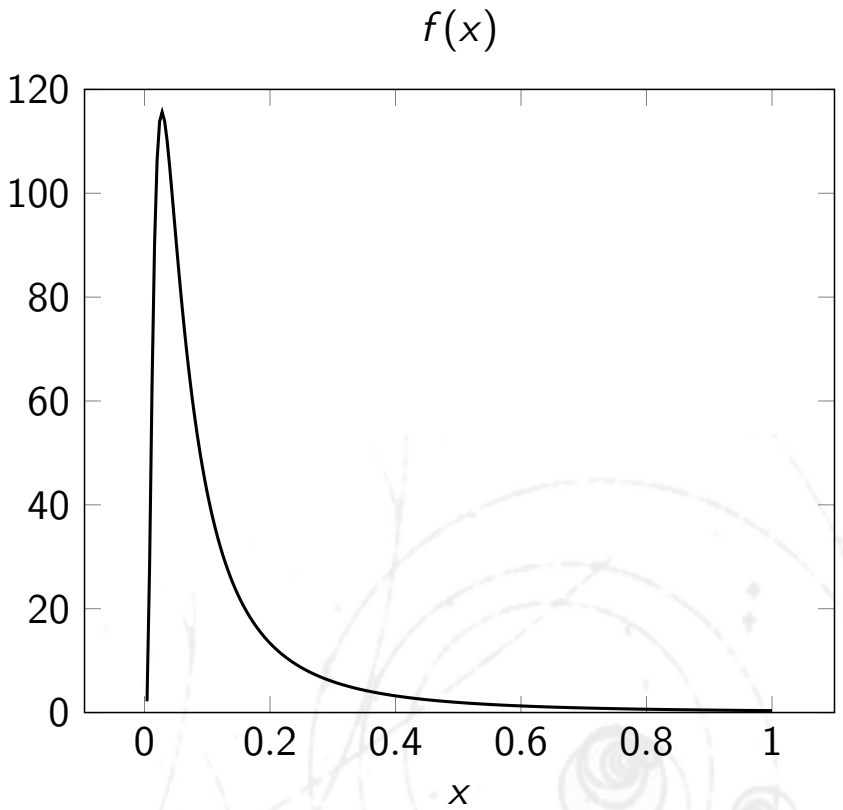
Example: The Taylor expansion is an asymptotic expansion with $\phi_n = x^n$.

Practical Issues

The pathological example you saw earlier *does have* an asymptotic expansion! However, for $n \leq 10$ the extrapolation did not yet show convergence.

The problem here are

- the choice of the starting point,
- the existence of a bump between the starting point $h_1 = 1/2$ and the point of interest ($h = 0$).



Speeding up extrapolations

Let's assume you have carried out an extrapolation to some point x_0 for several different values of the degree n:

$$P_n(x) : \text{ interpolating polynomial} \qquad p_n \equiv P_n(x_0)\,.$$

The p_n are a sequence that (hopefully) converges to the value you seek.
For definiteness, let's use the previous example of $f(x) = \cos(x)$. The first few elements of the sequence p_n were:

$$p_0 = 0.877583 \qquad p_1 = 1.06024 \qquad p_2 = 1.00056$$
$$p_3 = 0.99996 \qquad p_4 = 1.00000$$

Assuming that any sequence p_n is convergent, it would be nice to have a way to accelerate the sequence without any additional (potentially costly!) evaluations of the function $f(x)$!

Delta-Square rule by Aitkens

Define a new sequence q_n as follows:

$$q_n \equiv p_n - \frac{[\Delta(p)_n]^2}{\Delta^2(p)_n} ,$$

where

$$\Delta(p)_k = p_{k+1} - p_k , \qquad \Delta^2(p)_k = \Delta(p)_{k+1} - \Delta(p)_k .$$

This could be written simpler, but in order to avoid numerical instabilities (floating point numbers!) the version written here is preferable.

[This method has further applications which we will revisit next lecture.]

Accelerating our example

n	p_n	q_{n-2}
0	0.87758	—
1	1.06024	—
2	1.00056	0.999954
3	0.99996	0.999998
4	1.00000	...

- compare p_n with q_{n-2} to keep things fair
  i.e.: p_2 vs q_0, p_3 vs q_1 etc.

Accelerating our example

n	p_n	q_{n-2}
0	0.87758	—
1	1.06024	—
2	1.00056	0.999954
3	0.99996	0.999998
4	1.00000	...

- compare p_n with q_{n-2} to keep things fair
  i.e.: p_2 vs q_0, p_3 vs q_1 etc.
- $q_0(p_0, p_1, p_2)$ is competitive with p_3!

Accelerating our example

n	p_n	q_{n-2}
0	0.87758	—
1	1.06024	—
2	1.00056	0.999954
3	0.99996	0.999998
4	1.00000	...

- compare p_n with q_{n-2} to keep things fair
  i.e.: p_2 vs q_0, p_3 vs q_1 etc.
- $q_0(p_0, p_1, p_2)$ is competitive with p_3!
- $q_1(p_1, p_2, p_3)$ is competitive with p_4!

Backup