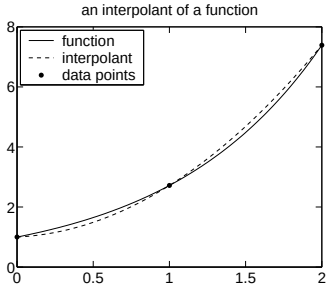
Numerical Interpolation -- Introduction

Given a (possibly complicated) function $f(x)$, it is often desirable to approximate it by a simpler function $g(x)$.
Sometimes, the function $f(x)$ may not be given in explicit but in tabular form, i.e., as a set of values f_i, $i = 0, \cdots, n$, corresponding to function values $f(x_i)$, at points x_i, $i = 0, \cdots, n$.

Definition: We say that $g(x)$ *interpolates* $f(x)$ at points x_i, $i = 0, \cdots, n$, if $g(x_i) = f(x_i)$, $i = 0, \cdots, n$.
Then, g is called the *interpolant* or *interpolating function* of the function f at points x_i, $i = 0, \cdots, n$.
Note that, when $g(x)$ interpolates $f(x)$ at certain points, then $f(x)$ interpolates $g(x)$ at the same points.
We say that $g(x)$ *interpolates* f_i, $i = 0, \cdots, n$, if $g(x_i) = f_i$, $i = 0, \cdots, n$.
Then, g is called the *interpolant* or *interpolating function* of the data f_i, $i = 0, \cdots, n$.

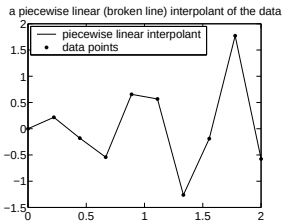


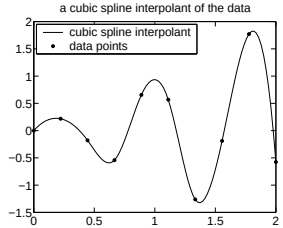
Numerical Interpolation -- Introduction

Examples of practical situations, where approximation of a function or data is needed:

• We are given a complicated function f in explicit form and wish to evaluate it at lots of points. The evaluation of f is costly. We may also wish to differentiate it and evaluate the derivative. The differentiation of f and the evaluation of the derivative are costly too. We may also wish to integrate f. Integrating f may be complicated or impossible to do with standard mathematical formulae. In such cases, it makes sense to have a simpler function g that approximates f ''well enough''. (Clearly, we also need a way to judge how well g approximates f.)

• We are given a set of data (x_i, f_i), $i = 0, \cdots, n$, and wish to plot f versus x. Assume n is not very large, so just joining the data points by straight lines (broken line plot) does not give a smooth and visually pleasing plot. We wish to calculate values of f for lots of intermediate x points (not included in the x_i's).

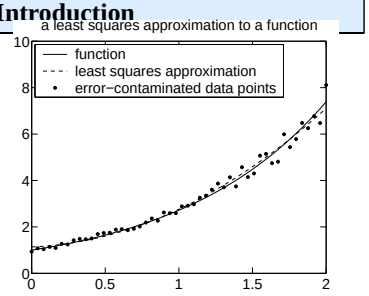


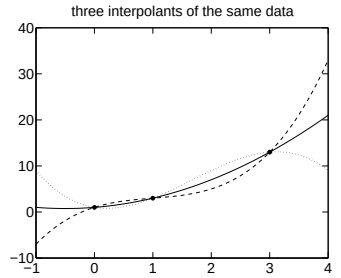


Numerical Interpolation -- Introduction

Notes:

1. Interpolation is not the only type of approximation possible. There are other types of approximation, for example least squares approximation. This type of approximation is often preferred to interpolation, when we are given a large number of error-contaminated data. Least squares approximations are studied in CSC436.
2. Given a set of data (x_i, f_i), $i = 0, \cdots, n$, there is no unique interpolant of the data. We need to impose certain conditions (examples of which are studied later in the course), in order to obtain a unique interpolant for the given data.



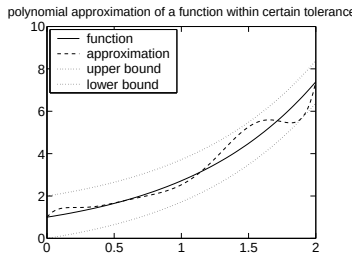


Polynomial Approximation

Polynomials are often chosen to approximate complicated functions. Reasons for choosing polynomials include:
(1) Polynomials are easy to evaluate
(2) Polynomials are easy to differentiate
(3) Polynomials are easy to integrate
(4) Polynomials provide good approximations to arbitrary continuous functions.

Weierstrass Theorem: If $f(x)$ is a continuous function on $[a, b]$, then for every $\varepsilon > 0$ (no matter how small an ε), there exists a polynomial $p_n(x)$ of degree $n = n(\varepsilon)$ (i.e. n depends on ε), such that

$$\max_{x \in [a,b]} |f(x) - p_n(x)| < \varepsilon .$$



Polynomial Approximation

Notes and caveats: The theorem tells us that, given a continuous function $f(x)$, there always exists a polynomial as close to f as we like (we can choose as small an ε as we like). However,

1. The degree n of p_n may be high. The cost of evaluating, differentiating and integrating p_n as well as the instability of any computations involved increase with n.
2. The theorem does not give a method to construct p_n, it only tells us that it exists.
3. The polynomial p_n may not be an interpolant of f, or not an interpolant of f on the points we wish.

For these reasons, the theorem has primarily theoretical value, and does not help in computing an interpolant or other approximation of f.

Evaluating a polynomial -- Horner's rule

We mentioned that polynomials are easy to evaluate. Let's study some relevant algorithms. Suppose we are given the coefficients a_j, $j = 0, \cdots, n$, of $p_n(x)$, i.e.

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

The coefficients are stored in array `a(0:n)`. A reasonable algorithm to compute $p_n(x)$, given x, is the following:

Standard algorithm for evaluating a polynomial

```
p = a(0)
for i = 1:n
    p = p + a(i)*x^i
endfor
```

On exit, `p` is $p_n(x)$.

The standard algorithm is simple, but it is not very efficient. There is a faster alternative, known as Horner's rule or nested multiplication. To understand how Horner's rule works, notice that

$$\begin{aligned} p_n(x) &= a_0 + x(a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}) \\ &= \cdots \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + (a_{n-2} + x(a_{n-1} + x a_n))))) \\ &= ((((a_n x + a_{n-1})x + a_{n-2})x + \cdots) + a_1)x + a_0 \end{aligned}$$

Evaluating a polynomial -- Horner's rule

Horner's rule for evaluating a polynomial

```
p = a(n)
for i = n-1:0 step -1
    p = p*x + a(i)
endfor
```

Notes:

- The standard algorithm uses n additions, n multiplications and n exponentiations.
- Horner's rule uses n additions, n multiplications.
- An extension of Horner's rule algorithm can be used to compute the value of the derivative $p_n'(x) = a_1 + 2a_2 x + 3a_3 x^2 + \cdots + n a_n x^{n-1}$.
- A similar algorithm can be used to compute the value of $p_n(x)$, when p_n is given in Newton's Divided Difference (NDD) form

$$\begin{aligned} p_n(x) = \alpha_0 &+ \alpha_1(x - x_0) + \alpha_2(x - x_0)(x - x_1) + \cdots \\ &+ \alpha_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \end{aligned}$$

  This form will be used later in the course.

Polynomial interpolation with monomial basis

Our goal is to construct a polynomial $p_n(x)$ of degree at most n, that interpolates the data (x_i, f_i), $i = 0, \cdots, n$. Let

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j.$$

Note that we are given $n+1$ data, and we must determine the values of the $n+1$ coefficients a_j (*undetermined coefficients*). In order to determine the coefficients a_j, so that p_n interpolates the given data, we setup the $n+1$ interpolating equations $p_n(x_i) = f_i$, $i = 0, \cdots, n$. The interpolating equations are

$a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = f_0$

$a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = f_1$

...

$a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = f_n.$

The interpolating equations form a linear system $B\mathrm{a} = \mathrm{f}$, where

$$B = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ . & . & . & \cdots & . \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}, \quad \mathrm{a} = \begin{bmatrix} a_0 \\ a_1 \\ . \\ a_n \end{bmatrix}, \quad \mathrm{f} = \begin{bmatrix} f_0 \\ f_1 \\ . \\ f_n \end{bmatrix}.$$

Polynomial interpolation with monomial basis

The matrix B is called a *Vandermonde matrix*. It can be shown that B is nonsingular, iff the points x_i, $i = 0, \cdots, n$, are distinct. In this case, we can uniquely solve the system of the interpolating equations and compute the coefficients a_j, $j = 0, \cdots, n$.

Notes:

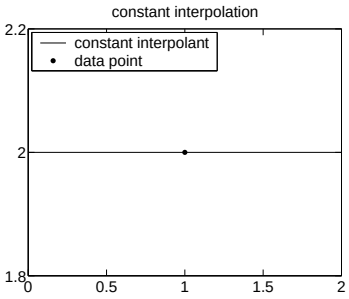
- It may turn out that $a_n = 0$, in which case $p_n(x)$ is of degree less than n. (There is nothing wrong with that.)
- The Vandermonde matrices are known to be ill-conditioned for large n. Therefore, the computation of the coefficients by the above technique is not used in practice, except for cases where n is small.
- Some books or other references present the Vandermonde matrix with the order of the columns reversed, i.e., the nth powers of the x_i's first and the 1's last. Of course, in such a case, the order of the a_j's must also be reversed.

Polynomial interpolation with monomial basis

Example: Consider the case $n = 1$, i.e. the case of constant interpolation. In this case $p_0 = a_0$, i.e. the interpolant is a constant. Assume (x_0, f_0) is given. The interpolating equation is

$p_0(x_0) = f_0 \Rightarrow a_0 = f_0$, and

$p_0 = f_0$. So, in the case of constant interpolation, we are given a point (x_0, f_0), and the interpolant is the line parallel to the x-axis, passing from (x_0, f_0).



Polynomial interpolation with monomial basis

Example: Consider the case $n = 1$, i.e. the case of linear interpolation. In this case $p_1 = a_0 + a_1 x$. Assume (x_i, f_i), $i = 0, 1$ are given. The interpolating equations are

$p_1(x_0) = f_0 \Rightarrow a_0 + a_1 x_0 = f_0$

$p_1(x_1) = f_1 \Rightarrow a_0 + a_1 x_1 = f_1$

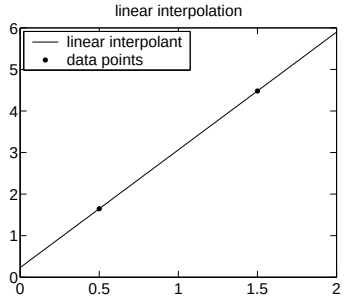
or, in matrix format, $\begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \end{bmatrix}$.

The solution of this system is $a_0 = \dfrac{x_0 f_1 - x_1 f_0}{x_0 - x_1}$, $a_1 = \dfrac{f_0 - f_1}{x_0 - x_1}$. This leads to

$$p_1(x) = \frac{x_0 f_1 - x_1 f_0}{x_0 - x_1} + \frac{f_0 - f_1}{x_0 - x_1}\, x.$$

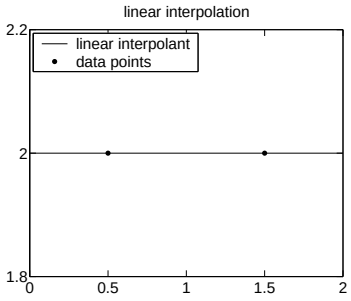
Thus the linear interpolant of the data $(x_0, f_0), (x_1, f_1)$ is the straight line passing from (x_0, f_0) and (x_1, f_1).

Notice that a_1 is given by the familiar formula defining the slope of the straight line passing from (x_0, f_0) and (x_1, f_1).



Also notice that, if $f_0 = f_1$, then p_1 is of degree 0, i.e. a constant.

Polynomial interpolation with monomial basis



Polynomial interpolation with monomial basis

Numerical example: Consider the data $(0, 5)$, $(-1, 7)$, $(2, 13)$. Here, $n = 2$. We construct a polynomial of degree at most 2, interpolating the data.

Let $p_2(x) = a_0 + a_1 x + a_2 x^2$. The interpolating equations are

$p_2(x_0) = f_0 \Rightarrow a_0 + a_1 x_0 + a_2 x_0^2 = f_0 \Rightarrow a_0 + a_1 \cdot 0 + a_2 \cdot 0 = 5$

$p_2(x_1) = f_1 \Rightarrow a_0 + a_1 x_1 + a_2 x_1^2 = f_1 \Rightarrow a_0 + a_1(-1) + a_2 \cdot 1 = 7$

$p_2(x_2) = f_2 \Rightarrow a_0 + a_1 x_2 + a_2 x_2^2 = f_2 \Rightarrow a_0 + a_1 \cdot 2 + a_2 \cdot 4 = 13$

or, in matrix format,

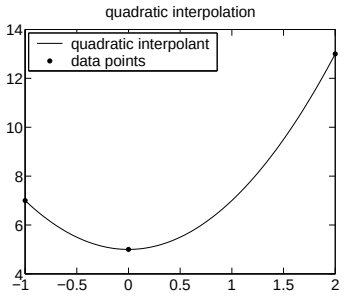
$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 13 \end{bmatrix}.$$

The solution to this system is $a_0 = 5$, $a_1 = 0$, $a_2 = 2$, i.e. $p_2(x) = 5 + 2x^2$. We can easily verify (we must do this to check the correctness of our calculations) that the above $p_2(x)$ satisfies the interpolating equations.

Polynomial interpolation with monomial basis

Example in matlab:

```
> x = linspace(-1, 2, 100)';
> xi = [0 -1 2]';
> yi = [5 7 13]';
> n = length(xi)-1;
> pn = polyfit(xi, yi, n)
pn =
    2.0000   -0.0000    5.0000
> p = polyval(pn, x);
> plot(x, p, 'r-', xi, yi, 'k.');
> legend('quadratic interpolant', 'data points', 2);
> title('quadratic interpolation');
```



Polynomial interpolation with monomial basis

- The function `polyfit(xi, yi, n)` finds the coefficients of a polynomial of degree at most `n` that fits the data in vectors `xi` and `yi`. In general, the function polyfit is used for constructing the least squares approximation of the data. However, when the input argument `n` is set to the number of data minus 1, then the least squares approximation reduces to interpolation and polyfit finds the coefficients of the polynomial of degree at most `n`, interpolating the data. This polynomial is the minimum degree polynomial interpolating the data.
  Note that polyfit returns the coefficients of the polynomial in reverse order, i.e. a_n comes first, and a_0 last. The coefficients are returned in a vector of size `n+1`.
- The function `polyval(pn, x)` evaluates the polynomial given by the coefficients in vector `pn`, at the points given by the vector `x`, and returns the respective values in a vector. We can then plot the vector of values versus `x`.

Polynomial interpolation with monomial basis

- It is worth noting that we normally evaluate the polynomial at points other than the data points (and usually at many more points than the data points). In the above example,
  `xi` is the vector of data points (abscissae),
  `yi` is the vector of data values,
  `x` is the vector of evaluation points (abscissae), and
  `p` is the vector of values of the polynomial at the evaluation points.
  If we do, though, evaluate the polynomial at the data points `xi` used in polyfit to construct the polynomial, we just get back the values of `yi` used in polyfit.
- Why is the coefficient a_1 returned as -0.0000?

