Polynomial Interpolation

1. Interpolating Polynomials
   - the interpolation problem
   - numerical condition

2. Lagrange Interpolation
   - a basis of Lagrange polynomials
   - Lagrange polynomials in Julia

3. Neville Interpolation
   - the value problem
   - Neville's algorithm
   - a Julia function

MCS 471 Lecture 14
Numerical Analysis
Jan Verschelde, 23 September 2022

Polynomial Interpolation

the interpolation problem

Often we have data collected from some difficult function $f(x)$.
With interpolation we can represent the data by a polynomial.

Input: $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $n + 1$ data points,
$x_i \neq x_j$, for all $i \neq j$, distinct values for x.
Output: $p(x)$ a polynomial of degree at most n so that
for all $i = 0, 1, \ldots, n$: $p(x_i) = f_i$.

The polynomial p *interpolates* the function $f(x)$
at the interpolation points x_i, $i = 0, 1, \ldots, n$.

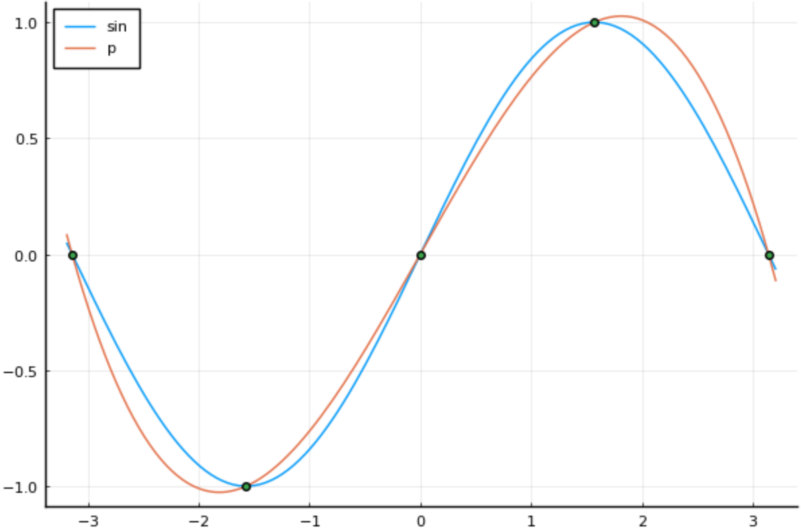
We say that p is the *interpolating polynomial* for the function $f(x)$ at x_i.

Two questions:

1. Is there a unique solution to the interpolation problem?
2. How to efficiently compute the interpolating polynomial?

interpolating $\sin(x)$ at five points

at $-\pi$, $-\pi/2$, 0, $\pi/2$, and π



the coefficient problem

The coefficient problem asks to compute the coefficients of p:

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_2 x^2 + c_1 x + c_0$$

so that $p(x_i) = f_i$ for $i = 0, 1, \ldots, n$.

Observe that we have

- $n + 1$ data points (x_i, f_i) given on input; and
- $n + 1$ coefficients c_i, $i = 0, 1, \ldots, n$ to compute.

Theorem (uniqueness condition on the solution)

If all interpolation points are mutually distinct: $x_i \neq x_j$, for all $i \neq j$, then the polynomial interpolation problem has a unique solution.

We prove this by setting up the interpolation conditions.

the interpolation conditions

For $p(x) = c_n x^n + \cdots + c_2 x^2 + c_1 x + c_0$
the conditions $p(x_i) = f_i$, for $i = 0, 1, \ldots, n$
lead to a linear system of $n + 1$ equations:

$$\left\{ \begin{array}{rcl} c_n x_0^n + c_{n-1} x_0^{n-1} + \cdots + c_2 x_0^2 + c_1 x_0 + c_0 & = & f_0 \\ c_n x_1^n + c_{n-1} x_1^{n-1} + \cdots + c_2 x_1^2 + c_1 x_1 + c_0 & = & f_1 \\ c_n x_2^n + c_{n-1} x_2^{n-1} + \cdots + c_2 x_2^2 + c_1 x_2 + c_0 & = & f_2 \\ & \vdots & \\ c_n x_{n-1}^n + c_{n-1} x_{n-1}^{n-1} + \cdots + c_2 x_{n-1}^2 + c_1 x_{n-1} + c_0 & = & f_{n-1} \\ c_n x_n^n + c_{n-1} x_n^{n-1} + \cdots + c_2 x_n^2 + c_1 x_n + c_0 & = & f_n \end{array} \right.$$

The unknowns are the coefficients c_i of the polynomial p.

the linear system in matrix notation

$$
\left\{
\begin{array}{rcl}
c_n x_0^n + c_{n-1} x_0^{n-1} + \cdots + c_2 x_0^2 + c_1 x_0 + c_0 & = & f_0 \\
c_n x_1^n + c_{n-1} x_1^{n-1} + \cdots + c_2 x_1^2 + c_1 x_1 + c_0 & = & f_1 \\
c_n x_2^n + c_{n-1} x_2^{n-1} + \cdots + c_2 x_2^2 + c_1 x_2 + c_0 & = & f_2 \\
& \vdots & \\
c_n x_{n-1}^n + c_{n-1} x_{n-1}^{n-1} + \cdots + c_2 x_{n-1}^2 + c_1 x_{n-1} + c_0 & = & f_{n-1} \\
c_n x_n^n + c_{n-1} x_n^{n-1} + \cdots + c_2 x_n^2 + c_1 x_n + c_0 & = & f_n
\end{array}
\right.
$$

$$
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} & x_2^n \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} & x_{n-1}^n \\
1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n
\end{bmatrix}
\begin{bmatrix}
c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n
\end{bmatrix}
=
\begin{bmatrix}
f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n
\end{bmatrix}
$$

the Vandermonde matrix

The linear system has a unique solution $\Leftrightarrow$ the determinant of

$$V(x_0, x_1, x_2, \ldots, x_{n-1}, x_n) = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{bmatrix}$$

is different from zero.

Definition

The matrix $V(x_0, x_1, x_2, \ldots, x_{n-1}, x_n)$ is *the Vandermonde matrix* for the points x_0, x_1, x_2, ..., x_{n-1}, x_n.

there is a unique solution

Let $V = V(x_0, x_1, x_2, \ldots, x_{n-1}, x_n)$, the determinant of V is

$$\det(V) = \prod_{i=0}^{n} \prod_{j=i+1}^{n} (x_i - x_j) \,.$$

Example for $n = 3$:

$$\det(V) = (x_0 - x_1)(x_0 - x_2)(x_0 - x_3)(x_1 - x_2)(x_1 - x_3)(x_2 - x_3).$$

Observe $\deg(\det(V)) = n(n+1)/2$ and $\det(V) \neq 0$ if $x_i \neq x_j$ for $i \neq j$.

Theorem (uniqueness condition on the solution)

The solution to the interpolation problem is unique if $x_i \neq x_j$ for $i \neq j$.

Polynomial Interpolation

1. Interpolating Polynomials
   - the interpolation problem
   - numerical condition

2. Lagrange Interpolation
   - a basis of Lagrange polynomials
   - Lagrange polynomials in Julia

3. Neville Interpolation
   - the value problem
   - Neville's algorithm
   - a Julia function

the numerical condition of the interpolation problem

Input: $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $n + 1$ data points,
$x_i \neq x_j$, for all $i \neq j$, distinct values for x.
Output: $p(x)$ a polynomial of degree at most n so that
for all $i = 0, 1, \ldots, n$: $p(x_i) = f_i$.

The requirement $x_i \neq x_j$, for all $i \neq j$, guarantees a unique solution, similarly as $\det(A) \neq 0$ guarantees a unique $\mathbf{x}$ for $A\mathbf{x} = \mathbf{b}$.

What if $x_i \approx x_j$, for some $j \neq i$?

Consider the lines interpolating through x_1 and x_2:

The line at the left is less sensitive to changes in x_1 and x_2 than the line at the right.

a numerical experiment

```
using Printf
Base.show(io::IO, f::Float64) = @printf(io, "%.3e", f)
using LinearAlgebra

dim = 5
pdx = 1.0/dim
pts = [pdx*k for k = 1:dim]
vdm = zeros(dim, dim)
for i=1:dim
    vdm[i, 1] = 1.0
    vdm[i, 2] = pts[i]
    for j=3:dim
        vdm[i, j] = vdm[i, j-1]*pts[i]
    end
end
show(stdout, "text/plain", vdm); println("")
println("the condition number : ", cond(vdm))
```

prints a 5-by-5 Vandermonde matrix and

```
the condition number : 2.300e+03
```

an exercise

Exercise 1:
Use the instructions on the previous slide to compute the condition number of the Vandermonde matrix for equidistant points in $[0,1]$, for dimensions 5, 10, 20, 40.

Summarize your observations in a well written sentence.

If we solve the interpolation problem as a linear algebra problem, then what this implies for the condition of the interpolation problem?

In particular, with 64-bit floats, what is the largest problem you could still solve an achieve an accuracy of 8 decimal places?

Polynomial Interpolation

Lagrange interpolation

Given are $n+1$ interpolation points (x_i, f_i), $i = 0, 1, \ldots, n$, where for all $i \neq j$: $x_i \neq x_j$. The Lagrange interpolating polynomial has the form

$$p(x) = \ell_0(x) f_0 + \ell_1(x) f_1 + \cdots + \ell_n(x) f_n,$$

where

$$\ell_i(x_j) = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j. \end{cases}$$

In this form, we have that $p(x_i) = f_i$, $i = 0, 1, \ldots, n$.

Definition

For $n+1$ mutually distinct points x_i, $i = 0, 1, \ldots, n$,

the i-th Lagrange polynomial is $\ell_i(x) = \displaystyle\prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)$.

the Lagrange interpolating polynomial

The i-th Lagrange polynomial $\ell_i(x)$ is $\ell_i(x) = \displaystyle\prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)$.

Example: $n = 3$, $i = 1$: $\ell_1(x_0) = 0$, $\ell_1(x_1) = 1$, $\ell_1(x_2) = 0$, $\ell_1(x_3) = 0$.

$$\ell_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)}.$$

The solution of the interpolation problem is unique as well, so the form for $\ell_1(x)$ is unique. The Lagrange interpolating polynomial is

$$p(x) = \sum_{i=0}^{n} \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) f_i.$$

This is convenient if only the f_i's change while the x_i's stays the same.

an exercise

Exercise 2:

Show that

$$\sum_{i=0}^{n} \ell_i(x) = 1.$$

Consider the interpolation data $(x_i, f_i = 1)$, $i = 0, 1, \ldots, n$
and remember that the interpolation problem has a unique solution.

Polynomial Interpolation

1. Interpolating Polynomials
   - the interpolation problem
   - numerical condition

2. Lagrange Interpolation
   - a basis of Lagrange polynomials
   - Lagrange polynomials in Julia

3. Neville Interpolation
   - the value problem
   - Neville's algorithm
   - a Julia function

the function lagrange

```
using Polynomials

"""
    lagrange(pts::Array{Float64,1},idx::Int64)

Returns the Lagrange polynomial for the points in pts
with index idx.

REQUIRED:

    All points in pts must be distinct.
    The index idx is between 1 and length(pts).
"""
```

definition of the function

```
function lagrange(pts::Array{Float64,1},idx::Int64)
    result = 1.0
    for i=1:idx-1
        result = result*Polynomial([-pts[i], 1.0])/
                                          (-pts[i]+pts[idx])
    end
    for i=idx+1:length(pts)
        result = result*Polynomial([-pts[i], 1.0])/
                                          (-pts[i]+pts[idx])
    end
    return result
end
```

the verification

```
"""
Verifies for a random collection of three points
that the value of the Lagrange polynomials are 0/1 and
that the sum of the Lagrange polynomials equals one.
"""
function main()
    pts = rand(3)
    sumLagrange = 0.0
    for i=1:3
        Li = lagrange(pts, i)
        println("Lagrange polynomial L", i, ":")
        println(Li)
        for j=1:3
            print("L", i, "(pts[", j, "]) = ")
            println(Li(pts[j]))
        end
        sumLagrange = sumLagrange + Li
    end
    println("Sum of Lagrange polynomials :")
    println(sumLagrange)
end

main()
```

running the program

```
$ julia lagrange.jl
Lagrange polynomial L1:
-4.2658 + 47.3187*x - 43.7541*x^2
L1(pts[1]) = 1.000000000000047
L1(pts[2]) = 4.991441920529706e-15
L1(pts[3]) = -2.9648825579806665e-17
Lagrange polynomial L2:
4.02443 - 44.7556*x + 42.4314*x^2
L2(pts[1]) = -5.309712702581818e-15
L2(pts[2]) = 0.9999999999999946
L2(pts[3]) = 8.552095321848118e-17
Lagrange polynomial L3:
1.24138 - 2.56304*x + 1.32271*x^2
L3(pts[1]) = 5.565465122993589e-17
L3(pts[2]) = 1.2695472919160173e-17
L3(pts[3]) = 1.0000000000000002
Sum of Lagrange polynomials :
1.0 + 4.88498e-15*x - 5.77316e-15*x^2
$
```

accuracy for growing dimensions

At first sight, the last polynomial from the previous slide

```
1.0 + 4.88498e-15*x - 5.77316e-15*x^2
```

does not look as `1.0` ...

Exercise 3:
Use the instructions on the previous slide to compute the sum of the Lagrange polynomials for equidistant points in $[0, 1]$, for dimensions 5, 10, 20, 40.

What do you observe about the errors in the sum of the Lagrange polynomials for growing dimensions?

Polynomial Interpolation

the value problem

Input: $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $n + 1$ data points,
$x_i \neq x_j$, for all $i \neq j$, distinct values for x,
x^* is the value for some x.
Output: $p(x^*)$ a the value of the interpolating polynomial at x^*.

Theorem

Let $p_i = f_i$, for $i = 1, 2, \ldots, n$ and

$$p_{0,1,\ldots,n} = \left(\frac{x^* - x_n}{x_0 - x_n} \right) p_{0,\ldots,n-1} + \left(\frac{x^* - x_0}{x_n - x_0} \right) p_{1,\ldots,n}$$

then $p_{0,1,\ldots,n} = p(x^*)$.

Neville interpolation

Let $p_i = f_i$, for $i = 0, 1, \ldots, n$ and

$$p_{0,1,\ldots,n}(x) = \left(\frac{x - x_n}{x_0 - x_n} \right) p_{0,\ldots,n-1}(x) + \left(\frac{x - x_0}{x_n - x_0} \right) p_{1,\ldots,n}(x)$$

then $p_{0,1,\ldots,n}(x_i) = f_i$, $i = 0, 1, \ldots, n$.

- $p_{0,1,\ldots,n}(x_0) = \left(\dfrac{x_0 - x_n}{x_0 - x_n} \right) p_{0,\ldots,n-1} + 0 = f_0$, analoguous for x_n.
- For x_k, for $k > 1$ and $k < n$:

$$\begin{aligned} p_{0,1,\ldots,n}(x_k) &= \left(\frac{x_k - x_n}{x_0 - x_n} \right) \underbrace{p_{0,\ldots,n-1}(x_k)}_{f_k} + \left(\frac{x_k - x_0}{x_n - x_0} \right) \underbrace{p_{1,\ldots,n}(x_k)}_{f_k} \\ &= \frac{x_k - x_n - (x_k - x_0)}{x_0 - x_n} f_k \\ &= f_k \end{aligned}$$

Polynomial Interpolation

1. Interpolating Polynomials
   - the interpolation problem
   - numerical condition
2. Lagrange Interpolation
   - a basis of Lagrange polynomials
   - Lagrange polynomials in Julia
3. Neville Interpolation
   - the value problem
   - Neville's algorithm
   - a Julia function

derivation of Neville's algorithm

For interpolation data (x_0, f_0), (x_1, f_1), ..., (x_n, f_n) and a value x^*:

- $p_i = f_i$, for $i = 0, 1, \ldots, n$,
- $p_{i,\ldots,j} = p(x^*)$ is the value of the interpolating polynomial at x^* through (x_i, f_i), (x_{i+1}, f_{i+1}), ..., (x_j, f_j).

The values $p_{i,\ldots,j}$ can be organized in a triangular table.
For example, for $n = 3$:

$$\begin{array}{ccccc} x_0 & f_0 & & & \\ x_1 & f_1 & p_{0,1} & & \\ x_2 & f_2 & p_{1,2} & p_{0,1,2} & \\ x_3 & f_3 & p_{2,3} & p_{1,2,3} & p_{0,1,2,3} \end{array}$$

Observe that we may replace f_0 by $p_{0,1}$, f_1 by $p_{1,2}$, and f_2 by $p_{2,3}$.

Neville's algorithm

For interpolation data (x_0, f_0), (x_1, f_1), ..., (x_n, f_n) and a value x^*:

- $p_i = f_i$, for $i = 0, 1, \ldots, n$,
- $p_{i,\ldots,j} = p(x^*)$ is the value of the interpolating polynomial at x^* through (x_i, f_i), (x_{i+1}, f_{i+1}), ..., (x_j, f_j).

for $i = 1, 2, \ldots, n$ do
  for $j = 1, 2, \ldots, i$ do

$$p_{i-j,\ldots,i} = \frac{(x^* - x_i)p_{i-j,\ldots,i-1} - (x^* - x_{i-j})p_{i-j+1,\ldots,i}}{x_{i-j} - x_i}$$

For efficient memory usage, relabel $p_{i-j,\ldots,i}$ as $p[i-j]$, $p_{i-j,\ldots,i-1}$ as $p[i-j]$ and $p_{i-j+1,\ldots,i}$ as $p[i-j+1]$.

The cost of Neville's algorithm is $O(n^2)$.

Polynomial Interpolation

a Julia function

```
"""
    neville(x::Array{Float64,1},f::Array{Float64,1},xx::Float64)

Implements the interpolation algorithm of Neville.

ON ENTRY :
    x       are the abscisses, given as a column vector
    f       are the ordinates, given as a column vector
    xx      is the point where to evaluate the interpolating
            polynomial through (x[i],f[i])

ON RETURN :
    p       is the last row of Neville's table where p[1] is
            the value of the interpolator at xx

EXAMPLE :
    x = [32.0, 22.2, 41.6, 10.1, 50.5];
    f = [0.52992, 0.37784, 0.66393, 0.17537, 0.63608];
    xx = 27.5;
    p = neville(x,f,xx)
"""
function neville(x::Array{Float64,1},f::Array{Float64,1},xx::Float64)
```

Neville's algorithm

```
n = length(x)
p = f
dx = [0.0 for i=1:n]
for i=1:n
    dx[i] = xx - x[i]
end
for i=2:n
    for j=2:i
        p[i-j+1] = (dx[i]*p[i-j+1] - dx[i-j+1]*p[i-j+2])/
                                  (x[i-j+1] - x[i])
    end
end
return p
```

the cost of interpolation

Exercise 4:
Compare the cost of Neville interpolation
(which is $O(n^2)$ for n points)
with the cost of Lagrange interpolation.

In particular, what is the cost to

1. construct the polynomial interpolating through n points with Lagrange polynomials; and then
2. to compute the value at some x^* of the interpolating polynomial?

Express this cost as a function of n.

uniqueness of interpolation

Exercise 5:

Consider $p = x^2 + 4x - 1$.

1. Apply Neville interpolation to compute the value at $x = 0$, using the points $(-2, p(-2))$, $(-1, p(-1))$, and $(+1, p(+1))$.
2. Explain why the value you obtain must equal $p(0)$.