Newton Interpolation

1. Incremental Interpolation
   - adding more interpolation points
2. Divided Differences
   - the Newton form of the interpolating polynomial
   - algorithms for Newton interpolation
   - an implementation in Julia
3. Condition of the Interpolation Problem
   - interpolation errors
   - the interpolation error

MCS 471 Lecture 15
Numerical Analysis
Jan Verschelde, 26 September 2022

Newton Interpolation

1. Incremental Interpolation
   - adding more interpolation points

2. Divided Differences
   - the Newton form of the interpolating polynomial
   - algorithms for Newton interpolation
   - an implementation in Julia

3. Condition of the Interpolation Problem
   - interpolation errors
   - the interpolation error

problem statement

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

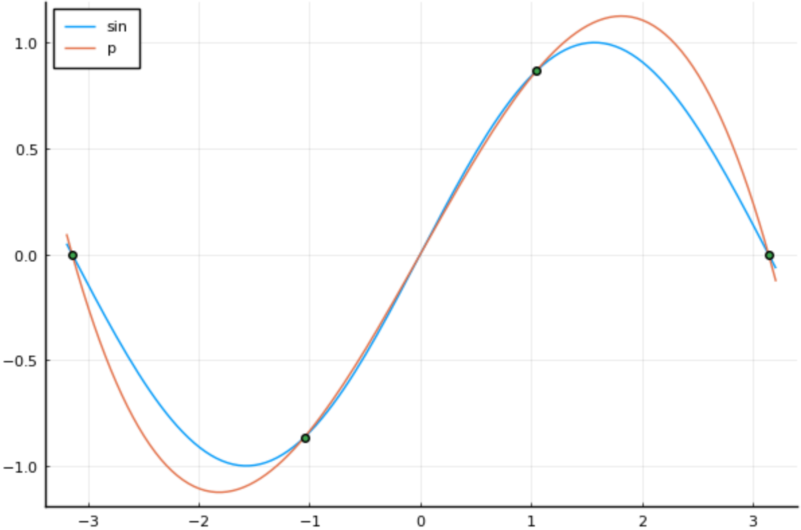
Input: $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $n + 1$ data points,
$x_i \neq x_j$, for all $i \neq j$, distinct values for x.
Output: $p(x)$ a polynomial of degree at most n so that
for all $i = 0, 1, \ldots, n$: $p(x_i) = f_i$.

Two questions:

1. How to efficiently add new interpolation points?
2. How to decide if more points should be added?

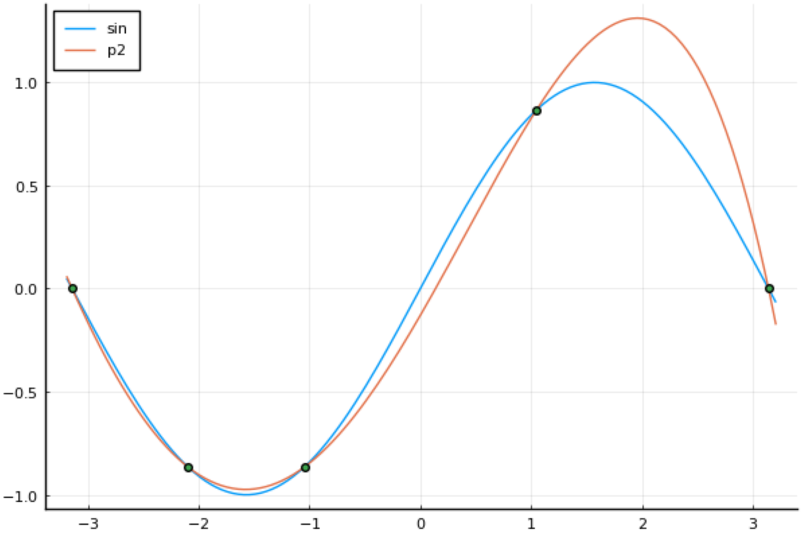
interpolating $\sin(x)$ at four points

at $-\pi$, $-\pi/3$, $\pi/3$, and π



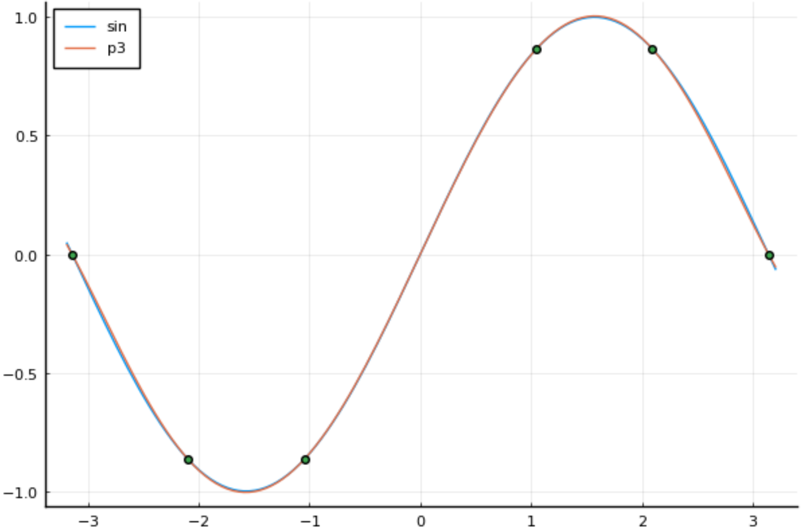
interpolating $\sin(x)$ at five points

at $-\pi$, $-\pi/3$, $\pi/3$, π, and $(-\pi - \pi/3)/2$



interpolating $\sin(x)$ at six points

at $-\pi$, $-\pi/3$, $\pi/3$, π, $(-\pi - \pi/3)/2$ and $(\pi/3 + \pi/3)/2$



Newton Interpolation

the Newton form of the interpolating polynomial

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 Newton form of the interpolating polynomial p is

$$\begin{array}{rcl} p(x) & = & f[x_0] \\ & + & f[x_0, x_1](x - x_0) \\ & + & f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ & \vdots & \\ & + & f[x_0, x_1, x_2, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{array}$$

incremental interpolation

The Newton form of the interpolating polynomial p is

$$\begin{array}{rcl} p(x) & = & f[x_0] \\ & + & f[x_0, x_1](x - x_0) \\ & + & f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ & \vdots & \\ & + & f[x_0, x_1, x_2, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{array}$$

The form allows for incremental interpolation: adding an extra point (x_{n+1}, f_{n+1}) adds and extra term

$$f[x_0, x_1, x_2, \ldots, x_n, x_{n+1}](x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - x_n)$$

to the polynomial which p which interpolates the first $n + 1$ points. The coefficients of p are divided differences.

divided differences

The coefficients of p are divided differences

$$
\begin{aligned}
f[x_i] &= f_i, \quad i = 0, 1, \ldots, n \\
f[x_i, x_j] &= \frac{f_i - f_j}{x_i - x_j}, \quad i \neq j \\
f[x_i, x_{i+1}, \ldots, x_{j-1}, x_j] &= \frac{f[x_i, x_{i+1}, \ldots, x_{j-1}] - f[x_{i+1}, \ldots, x_{j-1}, x_j]}{x_i - x_j}
\end{aligned}
$$

The divided differences $f[x_i, \ldots, x_j]$ can be organized in a triangular table. For example, for $n = 3$:

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

Observe that we may replace f_1 by $f_{0,1}$, f_2 by $f_{0,2}$, and f_3 by $f_{0,3}$.

Newton Interpolation

algorithm to compute divided differences

Input: (x_i, f_i), $i = 0, 1, \ldots, n$.
Output: divided differences $f[x_0]$, $f[x_0, x_1]$, …, $f[x_0, x_1, \ldots, x_n]$.

for $i = 1, 2, \ldots, n$ do
    for $j = 0, 1, \ldots, i-1$ do

$$f_{0,\ldots,j,i} = \frac{f_{0,\ldots,j-1,j} - f_{0,\ldots,j-1,i}}{x_j - x_i}$$

For efficient memory usage, relabel $f_{0,\ldots,j-1,i}$ as $f[i]$, $f_{0,\ldots,j-1,j}$ as $f[j]$, and $f_{0,\ldots,j-1,i}$ as $f[i]$.

The cost of computing divided differences is $O(n^2)$.

evaluating the Newton form

$$
\begin{aligned}
p(x) &= f[x_0] \\
&+ f[x_0, x_1](x - x_0) \\
&+ f[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&\vdots \\
&+ f[x_0, x_1, x_2, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}).
\end{aligned}
$$

Recall the nested Horner scheme to evaluate a polynomial.

$$
\begin{aligned}
p(x) &= f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\
&\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1](x - x_0) + f[x_0] \\
&= ((f[x_0, x_1, x_2, x_3](x - x_2) + f[x_0, x_1, x_2])(x - x_1) \\
&\quad + f[x_0, x_1])(x - x_0) + f[x_0]
\end{aligned}
$$

the cost of evaluation

For four points x_0, x_1, x_2, x_3, the divided differences are the coefficients of the interpolating polynomial of degree three.

$$\begin{aligned}
p(x) &= f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\
&\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \; + \; f[x_0, x_1](x - x_0) \; + \; f[x_0] \\
&= ((f[x_0, x_1, x_2, x_3](x - x_2) + \; f[x_0, x_1, x_2])(x - x_1) \\
&\quad + f[x_0, x_1])(x - x_0) \; + \; f[x_0]
\end{aligned}$$

We count the number of arithmetical operations to evaluate p at x:

- three subtractions, three additions, and
- three multiplications.

Exercise 1: Given the divided differences at $n+1$ points, evaluating the interpolating polynomial takes n subtractions, n additions, and n multiplications. Apply induction to prove this statement.

algorithm to evaluate the Newton form

Input: $(x_i, f_{0,\ldots,i})$, $i = 0, 1, \ldots, n$, x^*.
Output: $p(x^*)$ value of the interpolating polynomial at x^*.

$p(x^*) := f_{0,\ldots,n}$
for $i = n-1, \ldots, 1, 0$ do
    $p(x^*) := p(x^*)(x^* - x_i) + f_{0,\ldots,i}$

Observe: Newton interpolation with divided differences provides a convenient form to evaluate the interpolating polynomial and thus solves both the coefficient and the value problem.

Newton Interpolation

1. Incremental Interpolation
   - adding more interpolation points

2. Divided Differences
   - the Newton form of the interpolating polynomial
   - algorithms for Newton interpolation
   - an implementation in Julia

3. Condition of the Interpolation Problem
   - interpolation errors
   - the interpolation error

divided differences

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

Returns the vector of divided differences for the
Newton form of the interpolating polynomial.

On entry are x and f;
x contains the abscisses, given as a column vector; and
f contains the ordinates, given as a column vector.

On return are the divided differences.
"""
function divdif(x::Array{Float64,1},f::Array{Float64,1})
    n = length(x)
    d = deepcopy(f)
    for i=2:n
        for j=1:i-1
            d[i] = (d[j] - d[i])/(x[j] - x[i])
        end
    end
    return d
end
```

evaluating the Newton form

```
"""
Evaluates the Newton form of the
interpolating polynomial, with abscisses
in x and divided differences in d at xx.
"""
function newtonform(x::Array{Float64,1},
                    d::Array{Float64,1},
                    xx::Float64)
    n = length(d)
    result = d[n]
    for i=n-1:-1:1
        result = result*(xx - x[i]) + d[i]
    end
    return result
end
```

Newton interpolation

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

Implements the interpolation algorithm of Newton

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

ON RETURN :
   d       divided differences, computed from and f;
   p       value of the interpolating polynomial 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
  d, p = newton(x,f,xx)
"""
function newton(x::Array{Float64,1},f::Array{Float64,1},xx::Float64)
    divided = divdif(x,f)
    result = newtonform(x,divided,xx)
    return divided, result
end
```

an exercise

Exercise 2:
Consider the polynomial $p(x) = 2x^2 - 4x + 1$
and interpolation points $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$.
Let $f_0 = p(x_0)$, $f_1 = p(x_1)$, $f_2 = p(x_2)$, $f_3 = p(x_3)$.

1. Compute the table of divided differences.
   What do you observe about the size of the last element?
2. Compute the Newton form of the interpolating polynomial.
   Compare the Newton form with $p(x)$ and
   explain the outcome of your comparison.

Newton Interpolation

1. Incremental Interpolation
   - adding more interpolation points
2. Divided Differences
   - the Newton form of the interpolating polynomial
   - algorithms for Newton interpolation
   - an implementation in Julia
3. Condition of the Interpolation Problem
   - interpolation errors
   - the interpolation error

condition of the interpolation problem

If all $x_i \neq x_j$, for $i \neq j$, then the solution is unique.

What if $x_i \approx x_j$? What if points are close to each other?

Consider the line through (x_0, y_0), (x_1, y_1):

$$\begin{aligned} p(x) &= \left(\frac{x - x_1}{x_0 - x_1} \right) y_0 + \left(\frac{x - x_0}{x_1 - x_0} \right) y_1 \\ &= \left(\frac{1}{x_0 - x_1} \right) ((x - x_1) y_0 - (x - x_0) y_1) \,. \end{aligned}$$

If $x_0 \approx x_1$, then the coefficients of p will be huge.

Small changes in the input (x_0, y_0), (x_1, y_1) will lead to huge changes in the coefficients of p: bad conditioning.

a numerical experiment

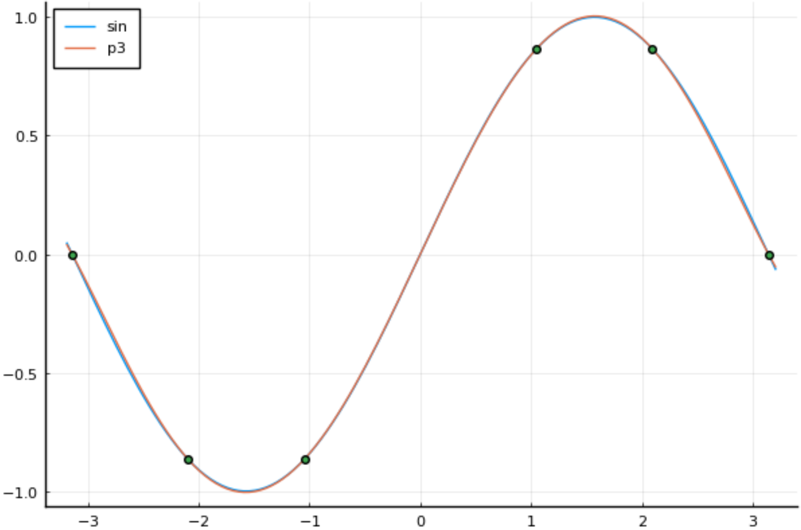
Exercise 3:
Consider equidistant interpolation points $x_i = i/n$, $i = 0, 1, 2, \ldots, n$ in the interval $[0, 1]$ and take $f_i = 1$, $i = 0, 1, 2, \ldots, n$.

Make a table for increasing values of n, for $n = 5, 10, 20, 40$, with the errors in the divided differences.

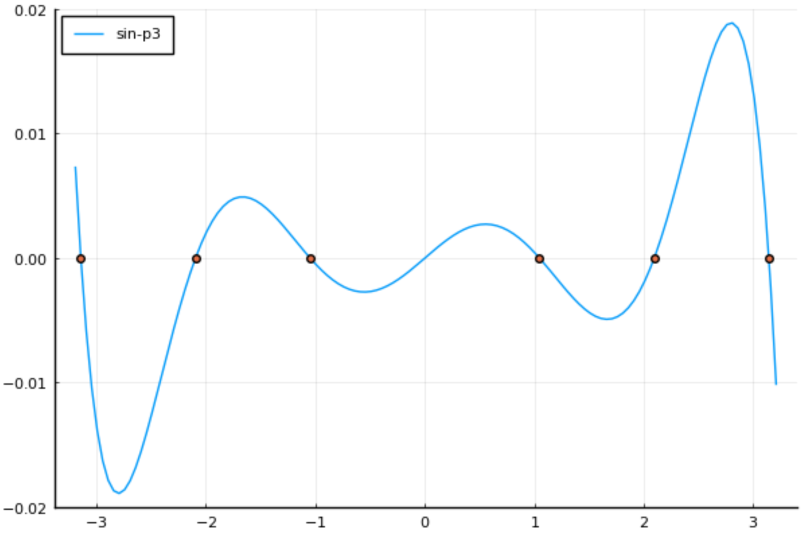
Summarize your observations.

interpolating $\sin(x)$ at six points

at $-\pi$, $-\pi/3$, $\pi/3$, π, $(-\pi - \pi/3)/2$ and $(\pi/3 + \pi/3)/2$



the interpolation error $\sin(x) - p(x)$



deriving the interpolation error

Assume $f(x)$ is continuously differentiable up to the $(n+1)$-th derivative over $[a, b]$. Given are $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $x_i \neq x_j$ for $i \neq j$, and all $x_i \in [a, b]$.
Let $p(x)$ be the interpolating polynomial: $p(x_i) = f_i$, $i = 0, 1, \ldots, n$.

The error function $E(x) = f(x) - p(x)$ has $n+1$ roots as $E(x_i) = f(x_i) - p(x_i) = f_i - f_i = 0$.
Let $\mathcal{E}(x) = K(x - x_0)(x - x_1) \cdots (x - x_n)$ for some constant K.

To derive a value for K, consider
$F(x) = E(x) - \mathcal{E}(x) = f(x) - p(x) - \mathcal{E}(x)$.
If we differentiate the equation $F(x) = 0$ $n+1$ times:

$$F^{(n+1)}(x) = f^{(n+1)}(x) - 0 - K(n+1)! = 0.$$

The above expression has a root $\alpha \in [a, b]$: $K = \dfrac{f^{(n+1)}(\alpha)}{(n+1)!}$.

Newton Interpolation

1. Incremental Interpolation
   - adding more interpolation points

2. Divided Differences
   - the Newton form of the interpolating polynomial
   - algorithms for Newton interpolation
   - an implementation in Julia

3. Condition of the Interpolation Problem
   - interpolation errors
   - the interpolation error

the interpolation error

For $(x_i, f_i = f(x_i))$, $i = 0, 1, \ldots, n$, $x_i \neq x_j$ for $i \neq j$, and $x_i \in [a, b]$, the interpolation error $E(x)$ is

$$E(x) = \frac{f^{(n+1)}(\alpha)}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n), \quad \alpha \in [a, b].$$

Let $M = \max\limits_{x \in [a,b]} \left| f^{(n+1)}(x) \right|$ and for all $x \in [a, b]$: $|x - x_i| \leq (b - a)$,

we have the bound:

$$|E(x)| \leq M \frac{(b-a)^{n+1}}{(n+1)!}.$$

Since $(n+1)!$ grows faster than any polynomial $|E(x)| \rightarrow 0$ as $n \rightarrow \infty$.

the next term rule

The Newton form of the interpolating polynomial

$$\begin{aligned} p(x) &= f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) + \cdots \\ &\quad + f[x_0,x_1,x_2,\ldots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1}) \end{aligned}$$

allows for incremental interpolation.
Take an additional point $\overline{x} \neq x_i$, $i = 0,1,\ldots,n$.

$$\begin{aligned} f(\overline{x}) &= p(\overline{x}) + f[x_0,\ldots,x_n,\overline{x}](\overline{x}-x_0)\cdots(\overline{x}-x_n) \\ f(\overline{x}) - p(\overline{x}) &= \underbrace{f[x_0,\ldots,x_n,\overline{x}]}_{K \text{ in } E(x)}(\overline{x}-x_0)\cdots(\overline{x}-x_n) \end{aligned}$$

Use $f[x_0,\ldots,x_n,\overline{x}] \approx \frac{f^{(n+1)}(\alpha)}{(n+1)!}$
to decide whether to add an extra point $\overline{x}$.

a cubic interpolation for $\sin(x)$

Exercise 4:

Consider $\sin(x)$ over $[0, \pi/2]$.

1. Take four equidistant points in $[0, \pi/2]$ and construct the interpolating polynomial p_3.
2. Make a plot of both p_3 and $\sin(x)$ in different colors over the interval $[0, \pi/2]$.
3. Compute the error with the next term rule.