



## 5. INTERPOLATION

### 5.1.1 Remember: what we saw during the last lesson

**Fitting data** to a linear model yields an over-determined system  $Ax \approx b$

**The normal equations** describe the best (least square residuals) fit:  $A^T Ax = A^T b$

#### Procedure

1. formulate your **model**: identify parameter, measurement and fitting coefficients
2. insert the measurements into the model to obtain an **overdetermined system**
3. solve the **normal equations**, monitor the residuals, plot and analyze the result

**Tradeoff** between accuracy (small residuals) and robustness (uncertainty fitting param.)

### 5.1.2 Overview: what you will learn today

**Interpolate** between discrete data points

**Polynomials** such as linear, cubic Hermite and splines

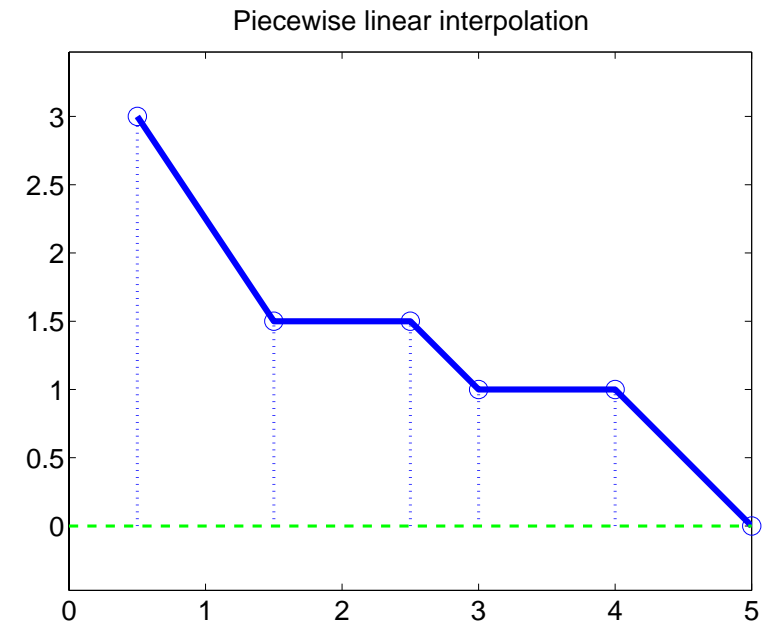
## 5.2 Interpolation using polynomials (NAM 3.1, H 7.3)

### Linear functions

$$P_1(x) = y_i + \underbrace{\frac{y_{i+1} - y_i}{x_{i+1} - x_i}}_{\text{slope}} (x - x_i) = c_1 + c_2 x$$

Built into the Matlab `plot()` command:

```
>> xx=[0.5 1.5 2.5 3 4 5]'; yy=[3 1.5 1.5 1 1 0]';  
>> stem(xx,yy,':'), axis equal, hold on,  
>> plot([0 5],[0 0],'g--'),  
>>  
>> % --- Piecewise linear  
>> plot(xx,yy), plot([0 5],[0 0],'g--'),  
>> title ('Piecewise linear interpolation');
```



## Monomial factors (naive form)

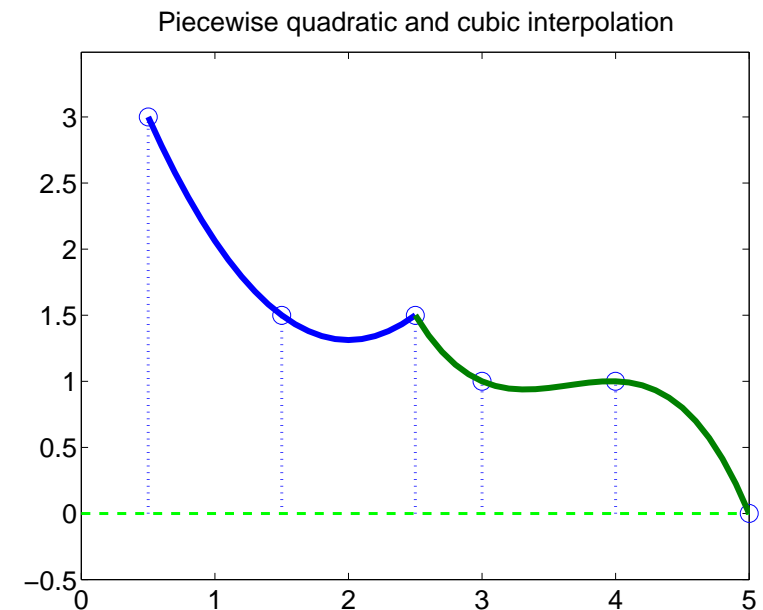
$$P_{n-1}(x) = c_1 + c_2x + c_3x^2 + \dots + c_nx^{n-1}$$

Cheap to evaluate, but expensive to get  $c_i$  from data  $\{(x_i, y_i)\}$ :

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

The Vandermonde matrix is easy to program, but ill conditioned:

```
>> % --- first part: quadratic
>> x=xx(1:3); y=yy(1:3);
>> A=[ones(3,1) x x.^2]; c=A\y
c = 4.3125 -3.0000 0.7500
k = 33.0069
>> X2=xx(1):0.1:xx(3); P2=c(1)+c(2)*X2+c(3)*X2.^2;
>>
>> % --- Second part: cubic
>> x=xx(3:6); y=yy(3:6);
>> A=[ones(4,1) x x.^2 x.^3]; c=A\y
c = 23.0000 -18.4333 5.1000 -0.4667
k = 1.2695e+04
>> X3=xx(3):0.1:xx(6)
>> P3=c(1)+c(2)*X3+c(3)*X3.^2+c(4)*X3.^3;
>> plot(X2,P2,X3,P3)
```



## Newton interpolation

$$P_{n-1}(x) = c_1 + c_2(x-x_1) + c_3(x-x_1)(x-x_2) + c_4(x-x_1)(x-x_2)(x-x_3) + \dots$$

Much cheaper to calculate  $c_i$  with forward-substitution from  $n$  data points  $\{(x_i, y_i)\}$ :

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & x_2 - x_1 & 0 & \dots & 0 \\ 1 & x_3 - x_1 & (x_3 - x_2)(x_3 - x_1) & 0 & \vdots \\ 1 & \vdots & \vdots & \dots & 0 \\ 1 & x_n - x_1 & (x_n - x_2)(x_n - x_1) & \dots & \prod_{k < n} (x_n - x_k) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

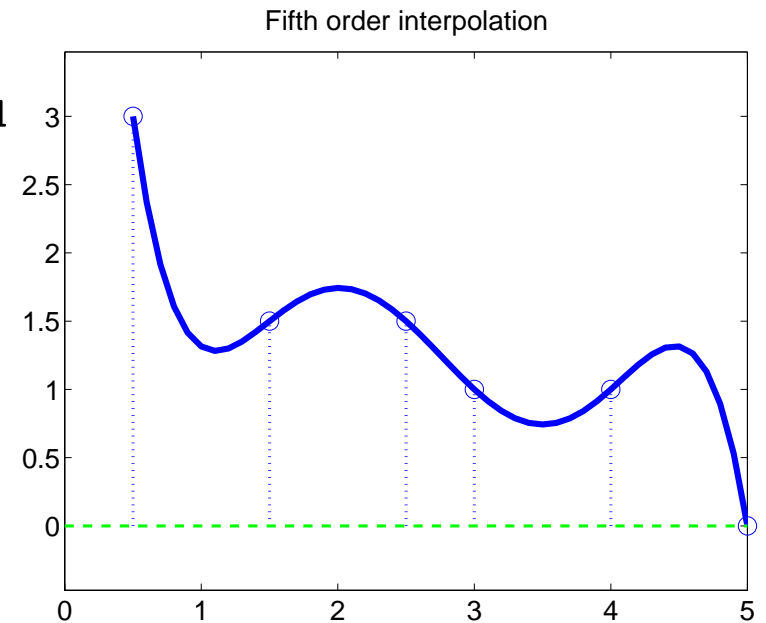
Cheap to evaluate using Horner's algorithm:

$$P(x) = c_1 + (x - x_1)[c_2 + (x - x_2)[c_3 + (x - x_3)[\dots (c_{n-1} + c_n(x - x_n)) \dots ]]]$$

```

>> ak=ones(6,1); A=ak;
>> for col=2:6, ak=ak.*(xx-xx(col-1)); A=[A ak]; end
>> c=A\yy, k=cond(A), A=A
c = 3.00 -1.50 0.75 -0.5667 0.3143 -0.1333
k = 264.9897
A = 1.00 0 0 0 0 0
    1.00 1.00 0 0 0 0
    1.00 2.00 2.00 0 0 0
    1.00 2.50 3.75 1.875 0 0
    1.00 3.50 8.75 13.125 13.125 0
    1.00 4.50 15.75 39.375 78.750 78.75
>> X=xx(1):0.1:xx(6); P=0; % Horner's algorithm
>> for k=6:-1:1, P=P.*(X-xx(k))+c(k); end
>> plot(X,P)

```



Large Runge oscillations (as they are called) appear **between** the data points: they are generally associated with the high order of the polynomials and make high order interpolation methods impractical.

# Lagrange interpolation

$$P_{n-1}(x) = y_1 L_1(x) + y_2 L_2(x) + \dots + y_n L_n(x), \quad L_{n-1}(x) = \frac{\prod_{k \neq j} (x - x_k)}{\prod_{k \neq j} (x_j - x_k)}$$

The coefficients are known (the matrix now is identity and the coefficients are simply the values  $y_i$ ); the evaluation of the polynomial, however, is tedious

$$P_2(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

## Peer Teaching (2 × 1 minutes to think, explain to your neighbour and vote)

**Polynomial interpolation.** Which method has the largest Runge oscillations?

↖ naive                      ↗ Newton                      ↘ Lagrange                      ↙ same

Are the coefficients obtained from each method related? How?

← no                      ↑ indirectly                      → they are the same

## Catastrophic cancellation

An offset  $x_i = z_{\text{offset}} + z_i$  does not affect the Newton and Lagrange interpolations; it does however result in dramatically large coefficients when using the naive interpolation with monomials, leading to catastrophic cancellations that show up as spurious, hacked curves that cannot be plot of reasonably low order polynomials.

## 5.3 Piecewise polynomial interpolation (NAM 3.3, H 7.4)

**Idea:** connect neighbouring data points with a polynomial and guarantee continuity of the function or derivatives from one cell ( $h_i = x_{i+1} - x_i$ ,  $\Delta y_i = y_{i+1} - y_i$ ) to the next

$$f(x) = \sum_i P(x_i + t \cdot h_i), \quad 0 \leq t < 1$$

### Piecewise linear

Require continuity with  $P(x_i + h_i) = y_i + \Delta y_i = y_{i+1} = P(x_{i+1})$  so that

$$P(x_i + t \cdot h_i) = y_i + t \Delta y_i$$

### Piecewise cubic Hermite

Using the continuity of the interpolating function and the first derivative

$$\begin{aligned} P(x_i) &= y_i, & P'(x_i) &= k_i \\ P(x_{i+1}) &= y_{i+1}, & P'(x_{i+1}) &= k_{i+1} \end{aligned}$$

it is possible to calculate the explicit expression for the polynomial and its derivatives

$$\begin{aligned} P(x_i + t \cdot h_i) &= y_i + t \Delta y_i + t(1-t)g_i + t^2(1-t)c_i \\ P'(x_i + t \cdot h_i) &= (\Delta y_i + (1-2t)g_i + (2t-3t^2)c_i) / h_i \\ P''(x_i + t \cdot h_i) &= (-2g_i + (2-6t)c_i) / h_i^2 \end{aligned}$$

where  $g_i = h_i k_i - \Delta y_i$  and  $c_i = 2 \Delta y_i - h_i(k_i + k_{i+1})$ .

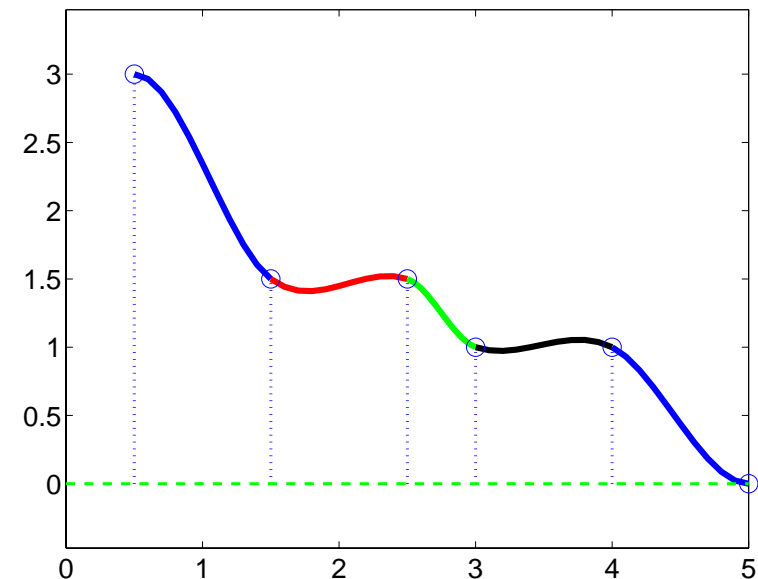
Sometimes the slope of the interpolating function is unknown on the data points  $k_i$ ; it can then be approximated using a centered difference

$$k_i = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}, \quad i = 2, 3, \dots, n - 1$$

specifying only the values at the boundary, e.g.  $k_1 = k_n = 0$ .

```
>> k=[0; (yy(3:6)-yy(1:4))./(xx(3:6)-xx(1:4)); 0];
>> h=diff(xx); dy=diff(yy);
>> g=h.*k(1:5)-dy; c=2*dy-h.*(k(1:5)+k(2:6));
>> t=0:0.1:1;
>> for i=1:5
>>     xt=xx(i)+t*h(i);
>>     yt=yy(i)+t*dy(i)+t.*(1-t)*g(i)
>>         +t.^2.*(1-t)*c(i);
>>     plot(xt,yt); hold on
>> end
```

Cubic Hermite pseudo-spline interpolation



**Peer Teaching** (2 × 1 minutes to think, explain to your neighbour and vote)

**Piecewise cubic Hermite pseudo-splines.** How does the second derivative of the interpolated function look like?

↖ does not exist

↗ discontinuous

↘ equal to zero

↙ continuous

## Piecewise cubic splines

In general, splines are piecewise polynomials of degree  $k$  that are continuously differentiable  $k - 1$  times. Cubic splines have continuous second derivatives

$$\begin{aligned} P''(x_i + 0 \cdot h_i) &= \frac{1}{h_i^2} \left( -2(h_i k_i - \Delta y_i) + 2(2\Delta y_i - h_i(k_i + k_{i+1})) \right) \\ &= \frac{2}{h_i} \left( 3\frac{\Delta y_i}{h_i} - 2k_i - k_{i+1} \right) \end{aligned}$$

$$P''(x_{i-1} + 1 \cdot h_i) = \frac{2}{h_{i-1}} \left( -3\frac{\Delta y_{i-1}}{h_{i-1}} + k_{i-1} + 2k_i \right)$$

which yields a relation between neighbouring derivatives  $(k_{i-1}, k_i, k_{i+1})$

$$h_i k_{i-1} + 2(h_i + h_{i-1})k_i + h_{i-1}k_{i+1} = 3 \left( \frac{h_{i-1}}{h_i} \Delta y_i + \frac{h_i}{h_{i-1}} \Delta y_{i-1} \right)$$

Unless the slopes  $(k_1, k_n)$  are known explicitly on the boundaries, the second derivative (curvature) is sometimes set to zero in so called **natural splines**:

$$\begin{aligned} 2h_1 k_1 + h_1 k_2 &= 3\Delta y_1 \\ h_{n-1} k_{n-1} + 2h_{n-1} k_n &= 3\Delta y_{n-1} \end{aligned}$$

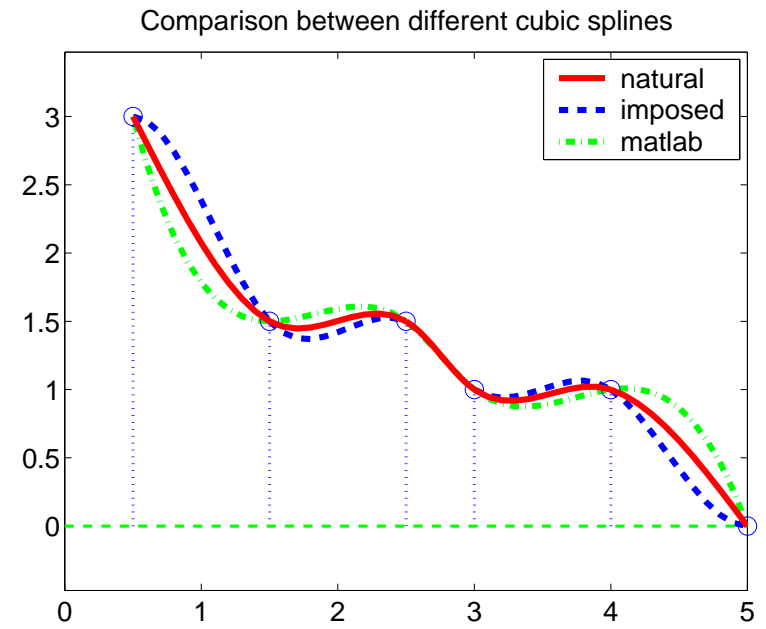
The tridiagonal linear system is solved with  $\mathcal{O}(n)$  operations using `tridia`.

$$\begin{pmatrix} 2h_1 & h_1 & 0 & \dots & \dots & \dots & 0 \\ h_2 & 2(h_2 + h_1) & h_1 & 0 & \ddots & \ddots & 0 \\ 0 & h_3 & 2(h_3 + h_2) & h_2 & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & h_{n-2} & 0 \\ \vdots & 0 & \ddots & \ddots & h_n & 2(h_n + h_{n-1}) & h_{n-1} \\ 0 & \dots & \dots & 0 & \dots & h_n & 2h_n \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

```

>> h=diff(xx); dy=diff(yy);
>> dm=2*(h(2:5)+h(1:4)); dia=[2*h(1); dm; 2*h(5)];
>> sub=[h(1); h(1:4)]; sup=[h(2:5); h(5)];
>> bm=3*(h(1:4).*dy(2:5)./h(2:5) +h(2:5).*dy(1:4)./h(1:4));
>> b=[3*dy(1); bm; 3*dy(5)];
>> % --- Uncomment to impose edge slope to zero
>> %     dia(1)=1; sup(1)=0; b(1)=0;
>> %     dia(6)=1; sub(5)=0; b(6)=0;
>> k=tridia(dia,sup,sub,b)
k = -1.9701 -0.5598 -0.5812 -0.6966 -0.3291 -1.3355
>> g=h.*k(1:5)-dy; c=2*dy-h.*(k(1:5)+k(2:6));
>> t=0:0.1:1;
>> for i=1:5
>>     xt=xx(i)+t*h(i);
>>     yt=yy(i)+t*dy(i)+t.*(1-t)*g(i)+t.^2.*(1-t)*c(i);
>>     plot(xt,yt,'r'); hold on
>> end
>>
>> % --- Matlab's own splines
>> X=xx(1):0.1:xx(6);Pm=spline(xx,yy,X);
>> plot(X,Pm,'-.')

```



Note the similarity between natural and pseudo-splines; what about the 2nd derivative?

## 5.4 Application: multiplication of polynomials (NAM 3.6)

Calculate the product  $P_c(x) = P_a(x) \cdot P_b(x)$  of two polynomials

$$P_a(x) = a_1 + a_2x + a_3x^2 + a_4x^3 + \dots + a_{n+1}x^n$$

$$P_b(x) = b_1 + b_2x + b_3x^2 + \dots + b_{m+1}x^m$$

leading to a polynomial of degree  $n + m$  with  $n + m + 1$  coefficients. Choose correspondingly many values and evaluate the product using Horner's algorithm

```
>> % polymult, multiplication Pc(x)=Pa(x)*Pb(x)
>> clear, clf
>> n=5; a=[2 5 -3 -4 1 3];
>> m=4; b=[7 -3 1 3 -2];
>> N=n+m+1; x=0.2*(-n:m)'; % 0.2 is arbitrary
>> pa=0; for i=n+1:-1:1, pa=pa.*x+a(i); end
>> pb=0; for i=m+1:-1:1, pb=pb.*x+b(i); end
>> pc=pa.*pb;
>> stem(x,pa), hold on
>> stem(x,pb,':'), plot(x,pc,'*', x,pc)
>> title('Pa, Pb and Pc(x)'),
>> axis([-1 1 -25 30])
>> Ak=ones(N,1); A=Ak; %interpolation matrix
>> for kol=2:N, Ak=Ak.*x; A=[A Ak]; end
>> c=A\pc
c = 14.0000 29.0000 -34.0000 -8.0000
    27.0000 -5.0000 -14.0000 14.0000
     7.0000 -6.0000
```

