Mahidol University
Faculty of Information
and Communication Technology

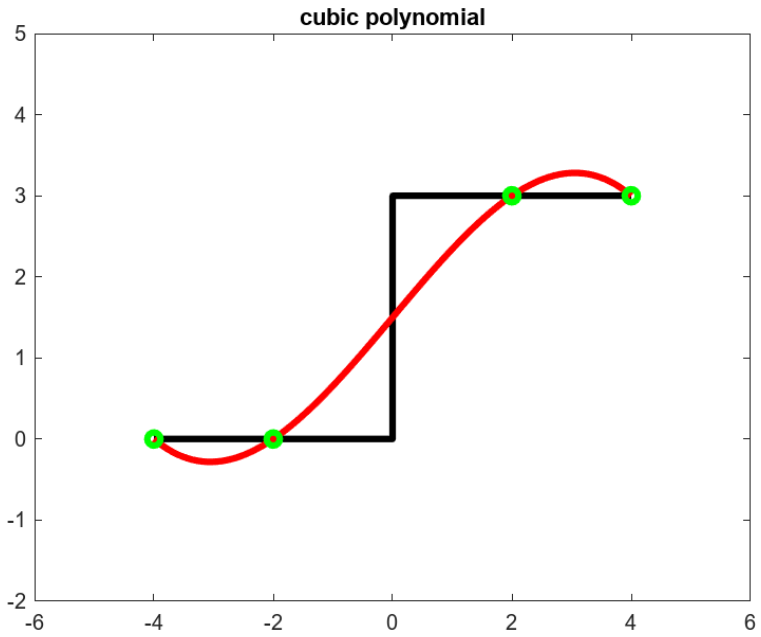
Splines and Piecewise Interpolation

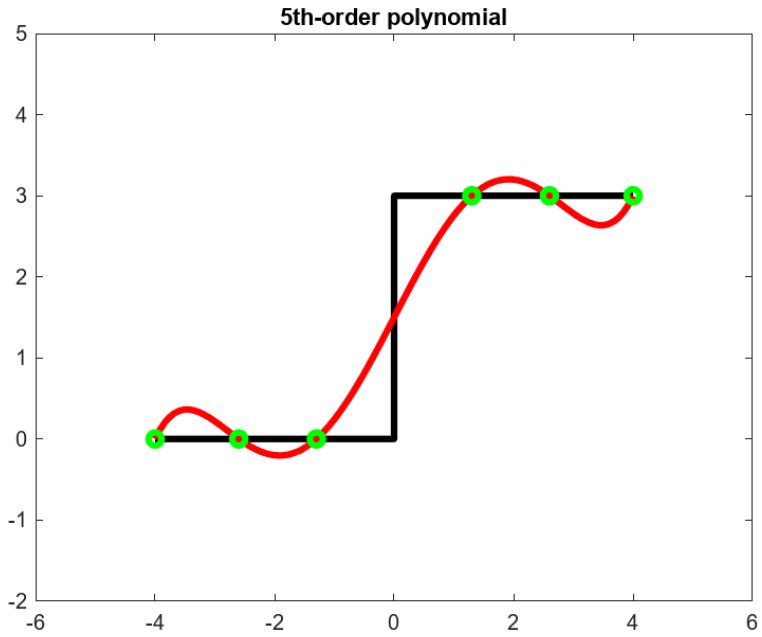
Rawesak Tanawongsuwan, Ph.D.
rawesak.tan@mahidol.ac.th

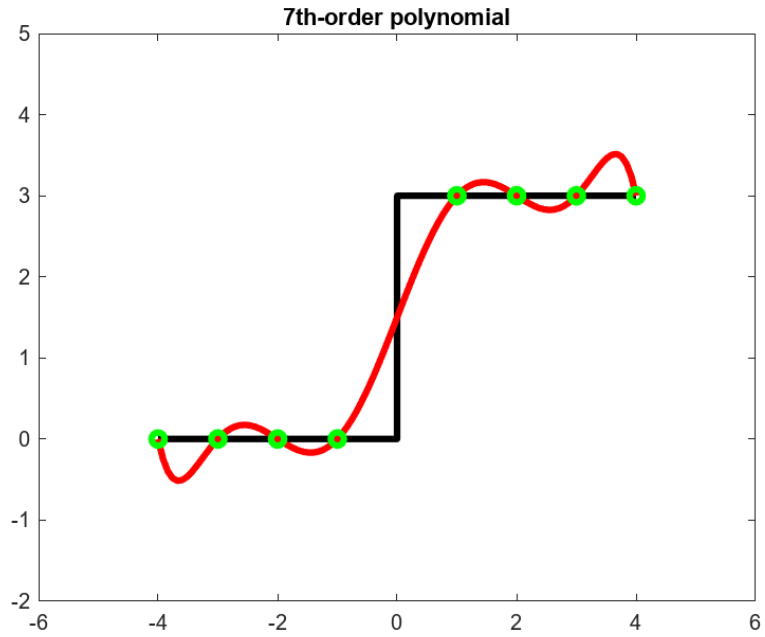
This slide is part of teaching materials for ITCS122 Numerical Methods
Semester 2/2023, Calendar year 2024

Recall on Polynomial Interpolation

- If we interpolate between n data points using $(n-1)^{th}$-order polynomials, the curve would pass through all those points. However, there are cases where these interpolated functions can lead to erroneous results because of roundoff and oscillations.

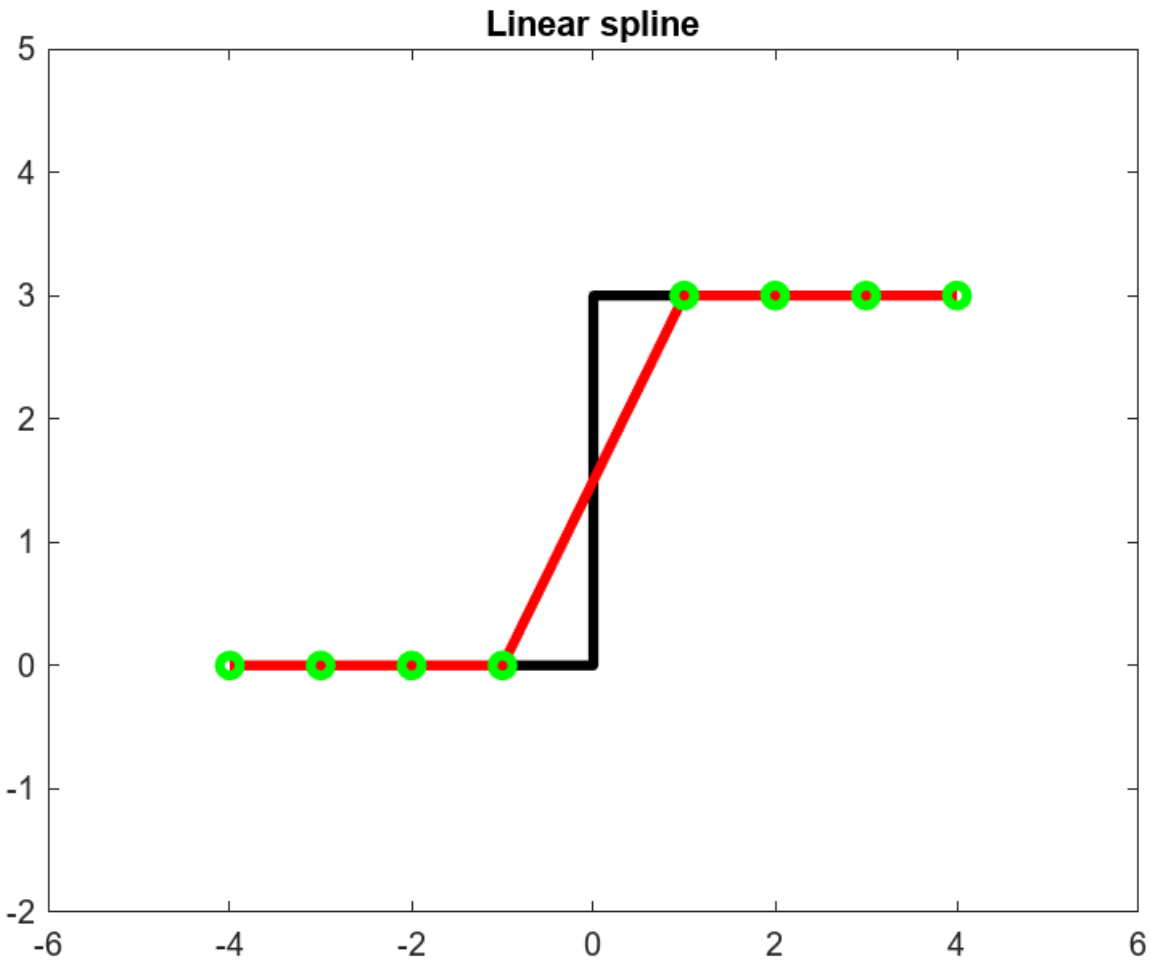






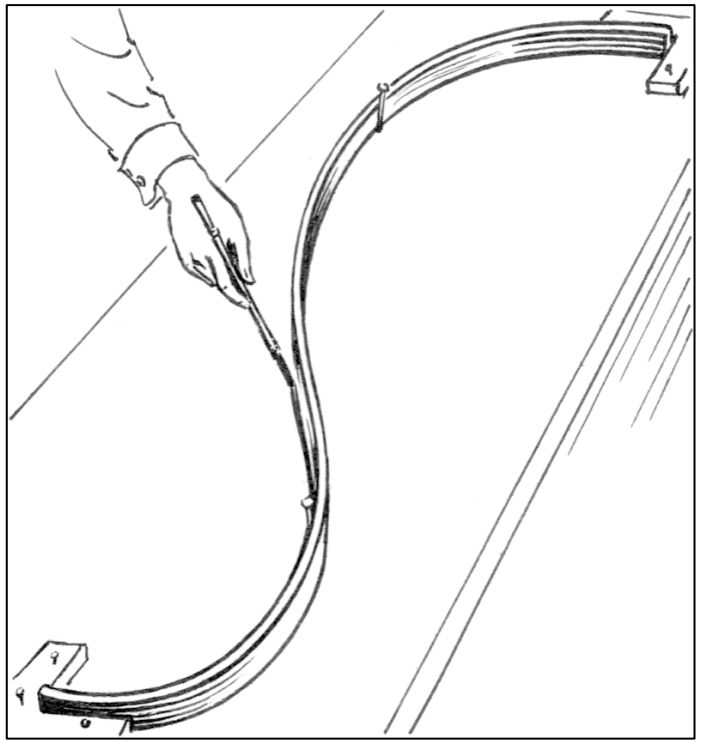
Spline functions

- An alternative approach is to apply lower-order polynomials in a piecewise fashion to subsets of data points.
- Those connecting polynomials are called ***spline functions***.
- The spline usually provides a superior approximation of the behavior of functions that have local, abrupt changes



Drafting technique

- The concept of the spline originated from the drafting technique of using a thin, flexible strip (called a *spline*) to draw smooth curves through a set of points.



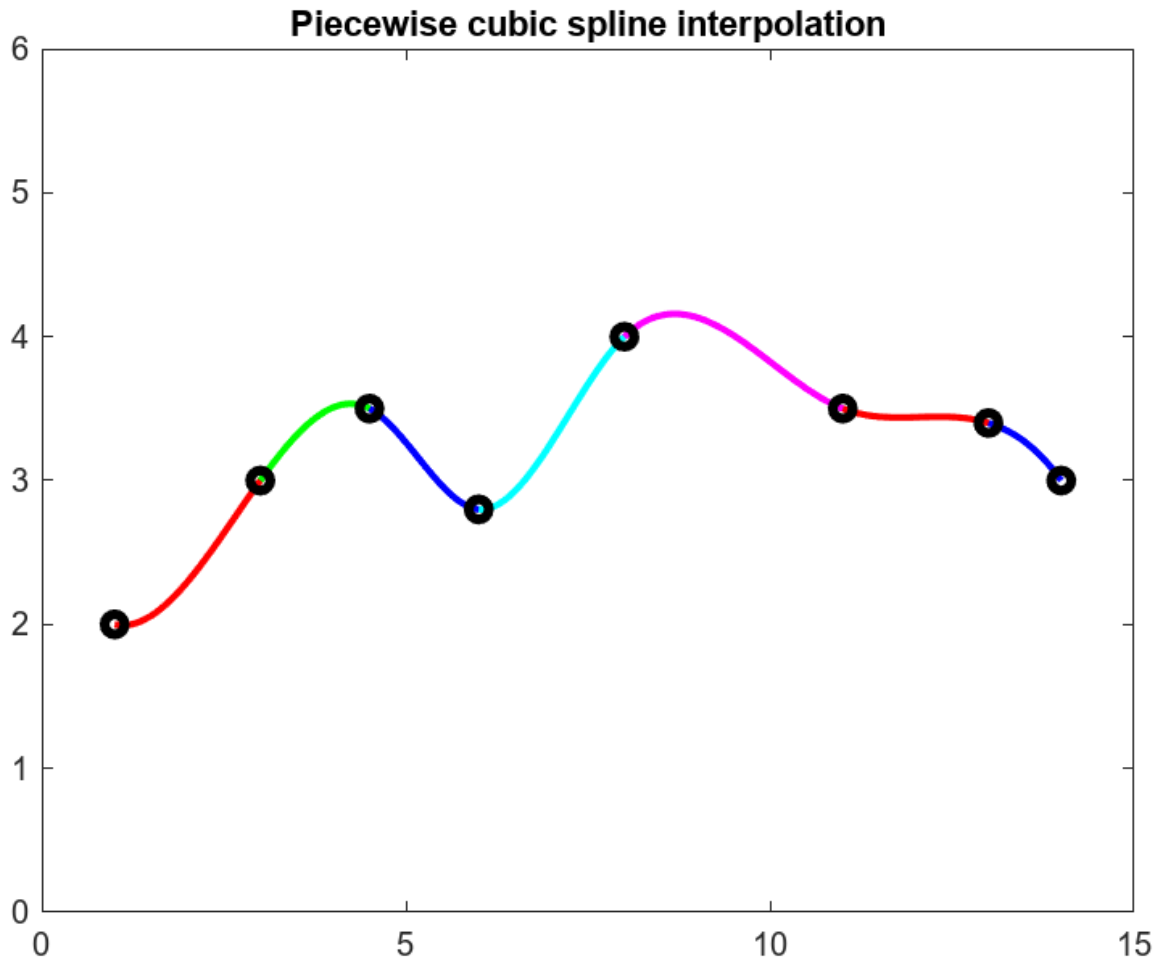
https://en.wikipedia.org/wiki/Flat_spline

https://perryboat.sail2live.com/yacht_design_according_to_perry/2011/11/my-last-blog-entry-on.html

Spline concept

- For n data points $(i = 1,2,3, \dots, n)$, there are $n - 1$ intervals.
- Each interval i has its own spline function, $s_i(x)$

- For example,
  - n data points $(n = 8) \rightarrow 7$ intervals
  - Each interval has its own spline function, $s_1(x), s_2(x), \dots, s_7(x)$



Linear splines

- Each function is the straight line connecting the two points at each end of the interval.

$$s_i(x) = a_i + b_i(x - x_i)$$

where a_i is the intercept : $a_i = f_i$

and b_i is the slope of the straight line : $b_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$

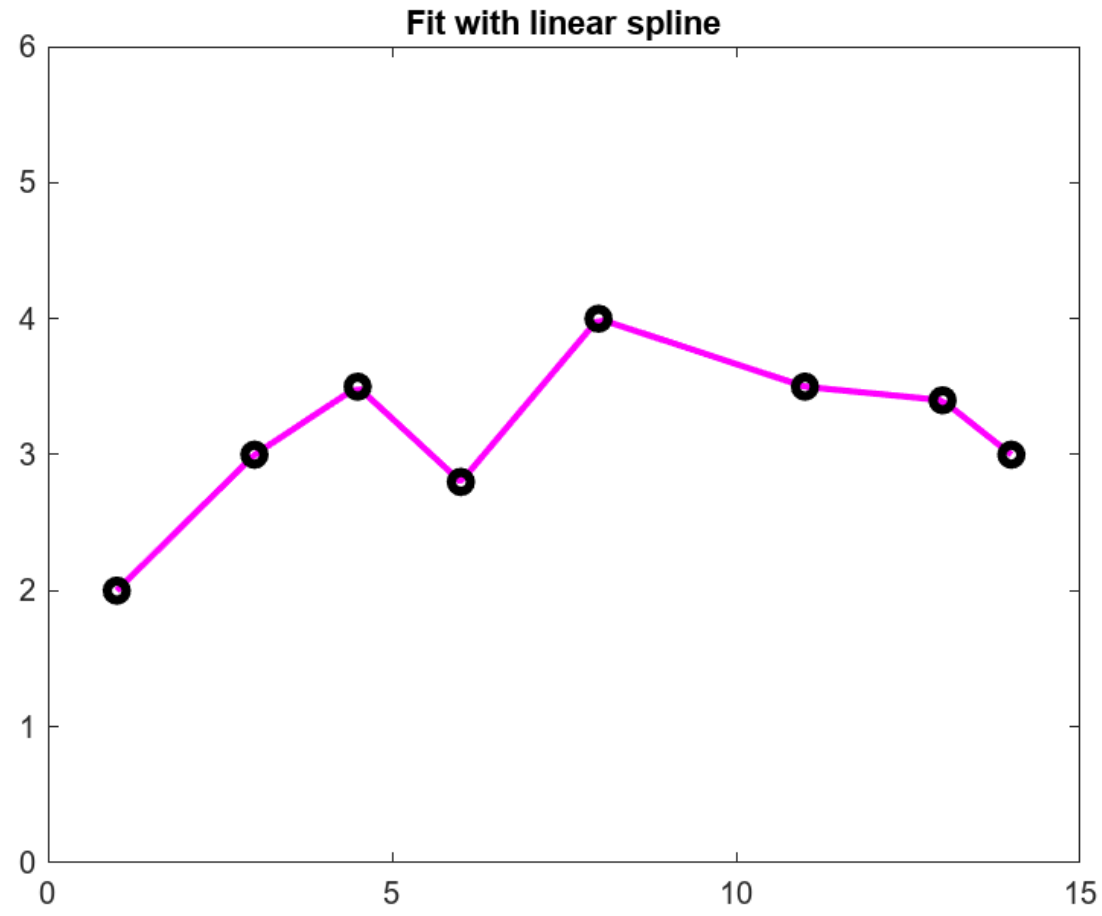
Note that $f_i = f(x_i)$

$$s_i(x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i)$$

- The linear spline is **Newton's first-order polynomial**.

Example of linear spline fitting

- Main disadvantage of linear splines is that they are not smooth.
- At the point (**knot**) where two splines meet, the slope changes abruptly → the first derivative of the function is discontinuous at these knots.
- We can improve by using higher-order polynomial splines that ensure smoothness at the knots by equating derivatives at these points.



Quadratic splines

- Second-order splines find quadratic equations between each pair of points that
  - Go through the points (a ***continuity condition***).
  - Match first derivatives at the interior points.

- Each function is the second-order polynomial connecting the two points at each end of the interval.

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2$$

- For n data points ($i = 1,2, \dots, n$), there are $n - 1$ intervals.
  - There are $\boldsymbol{3(n-1)}$ unknown constants (a, b, c) to evaluate.
  - At least $\boldsymbol{3(n-1)}$ equations or conditions are required to solve for the unknowns.

Solving for $3(n-1)$ variables (1)

1. The function must pass through all the points (a continuity condition), when $x = x_i$.

$$f_i = a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2$$

$$a_i = f_i$$

$$s_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2$$

- Since there are $n-1$ of a_i that we already solved, the number of conditions now is reduced to $2(n-1)$.

Solving for $3(n-1)$ variables (2)

2. The function values of adjacent polynomials must be equals at the knots.

- For knot $i+1$

  $$f_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 = f_{i+1} + b_{i+1}(x_{i+1} - x_{i+1}) + c_{i+1}(x_{i+1} - x_{i+1})^2$$

- Define the width of the i^{th} interval as $h_i = x_{i+1} - x_i$

  $$f_i + b_i h_i + c_i h_i^2 = f_{i+1}$$

- This equation can be written for the nodes, $i = 1, \dots, n-1$. So there will be $n-1$ conditions. There are $2(n-1) - (n-1) = n-1$ remaining conditions.

Solving for $3(n-1)$ variables (3)

3. The first derivatives at the interior nodes must be equal to ensure smoothness.

- Quadratic spline : $s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2$
  Its first derivative : $s_i'(x) = b_i + 2c_i(x - x_i)$

- The equivalence of the derivatives at node $i + 1$

$$s_i'(x_{i+1}) = b_i + 2c_i(x_{i+1} - x_i) = b_{i+1} + 2c_{i+1}(x_{i+1} - x_{i+1}) = s'_{i+1}(x_{i+1})$$
$$b_i + 2c_i h_i = b_{i+1}$$

- For all interior nodes, there are $n - 2$ conditions. There are $\mathrm{n} - 1 - (\mathrm{n} - 2) = 1$ remaining condition.

Solving for $3(n-1)$ variables (4)

4. Assume the second derivative is zero at the first point.

- Quadratic spline : $s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2$
  Its first derivative : $s_i'(x) = b_i + 2c_i(x - x_i)$
  Its second derivative : $s_i''(x) = 2c_i$

- At $x = x_1$, $s_1''(x_1) = 2c_1 = 0 \quad \rightarrow \quad c_1 = 0$

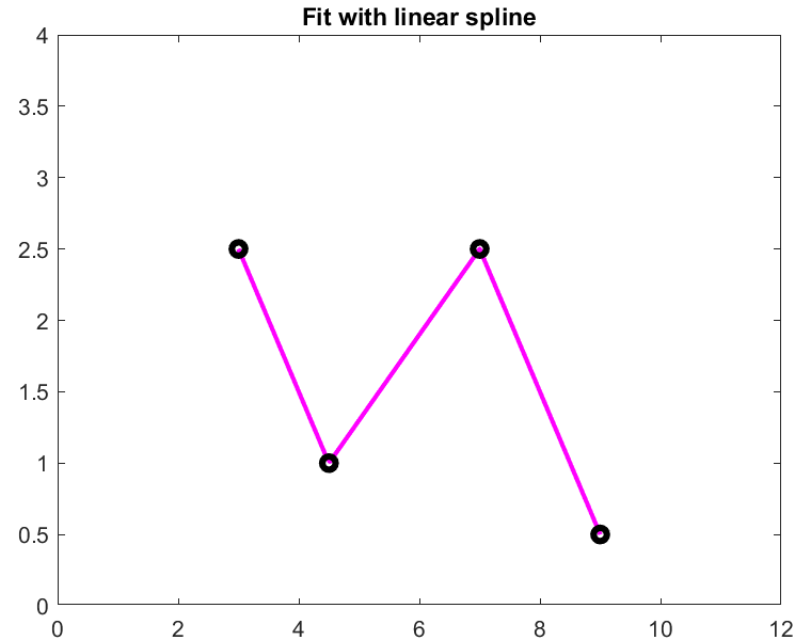
- This means that with this condition, the first two points will be connected by a straight line.

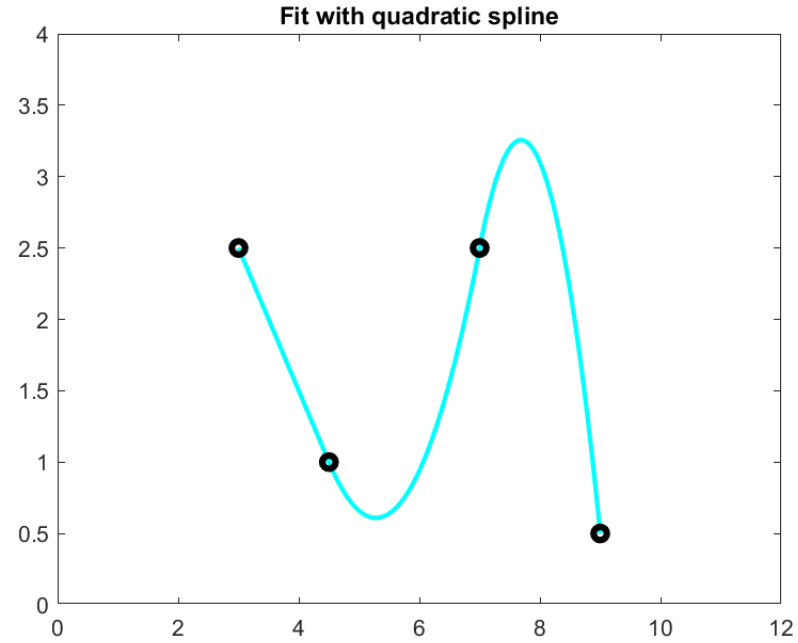
Quadratic spline example

- Fit these 4 points with linear and quadratic splines.

$$(3.0, 2.5) \quad (4.5, 1.0) \quad (7.0, 2.5) \quad (9.0, 0.5)$$

- Two shortcomings of quadratic spline
  - The straight line connecting the first two points
  - The spline for the last interval seems to swing too high





Cubic splines

- Third-order splines find cubic equations between each pair of points that
  - Go through the points (a ***continuity condition***).
  - Match first and second derivatives at the interior points.

- Each function is the third-order polynomial connecting the two points at each end of the interval.

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

- For n data points ($i = 1,2, \ldots, n$), there are $n - 1$ intervals.
  - There are $\boldsymbol{4(n-1)}$ unknown constants (a, b, c, d) to evaluate.
  - At least $\boldsymbol{4(n-1)}$ equations or conditions are required to solve for the unknowns.

Solving for $4(n-1)$ variables (1)

1. The function must pass through all the points (a continuity condition), when $x = x_i$.

$$f_i = a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 + d_i(x_i - x_i)^3$$

$$a_i = f_i$$

$$s_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

- Since there are $n-1$ of a_i that we already solved, the number of conditions now is reduced to $3(n-1)$.

Solving for $4(n-1)$ variables (2)

2. The function values of adjacent polynomials must be equals at the knots.

- For knot $i+1$

$$f_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = f_{i+1} + b_{i+1}(x_{i+1} - x_{i+1}) + c_{i+1}(x_{i+1} - x_{i+1})^2 + d_{i+1}(x_{i+1} - x_{i+1})^3$$

- Define the width of the i^{th} interval as $h_i = x_{i+1} - x_i$

$$f_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_{i+1}$$

- This equation can be written for the nodes, $i = 1, \ldots, n-1$. So there will be $n-1$ conditions. There are $3(n-1) - (n-1) = 2(n-1)$ remaining conditions.

Solving for $4(n-1)$ variables (3)

3. The first derivatives at the interior nodes must be equal to ensure smoothness.

- Quadratic spline : $s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$
  Its first derivative : $s_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$

- The equivalence of the first derivatives at node $i + 1$

$$s_i'(x_{i+1}) = s'_{i+1}(x_{i+1})$$
$$b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1} + 2c_{i+1}(x_{i+1} - x_{i+1}) + 3d_i(x_{i+1} - x_i)^2$$
$$b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1}$$

- For all interior nodes, there are $n - 2$ conditions. There are $2(n-1) - (n-2) = n$ remaining condition.

Solving for $4(n-1)$ variables (4)

4. The second derivatives at the interior nodes must be equal to ensure smoothness.

- Quadratic spline : $s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$
  Its first derivative : $s_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$
  Its second derivative : $s_i''(x) = 2c_i + 6d_i(x - x_i)$

- The equivalence of the second derivatives at node $i + 1$

$$s_i''(x_{i+1}) = s''_{i+1}(x_{i+1})$$
$$2c_i + 6d_i(x_{i+1} - x_i) = 2c_{i+1} + 6d_i(x_{i+1} - x_{i+1})$$
$$c_i + 3d_i h_i = c_{i+1}$$

- For all interior nodes, there are $n - 2$ conditions. There are $n - (n - 2) = 2$ remaining condition.

Solving for $4(n-1)$ variables (5)

5. Assume the second derivative is zero at the first point and the last point → This is called **the natural spline**.

- Quadratic spline : $s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$
  Its first derivative : $s_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$
  Its second derivative : $s_i''(x) = 2c_i + 6d_i(x - x_i)$

- At $x = x_1$, $s_1''(x_1) = 2c_1 + 6d_1(x_1 - x_1) = 0 \quad \rightarrow \quad c_1 = 0$
- At $x = x_n$, $s_{n-1}''(x_n) = 2c_{n-1} + 6d_{n-1}(x_n - x_{n-1}) = 2c_{n-1} + 6d_{n-1}h_{n-1} = 0$

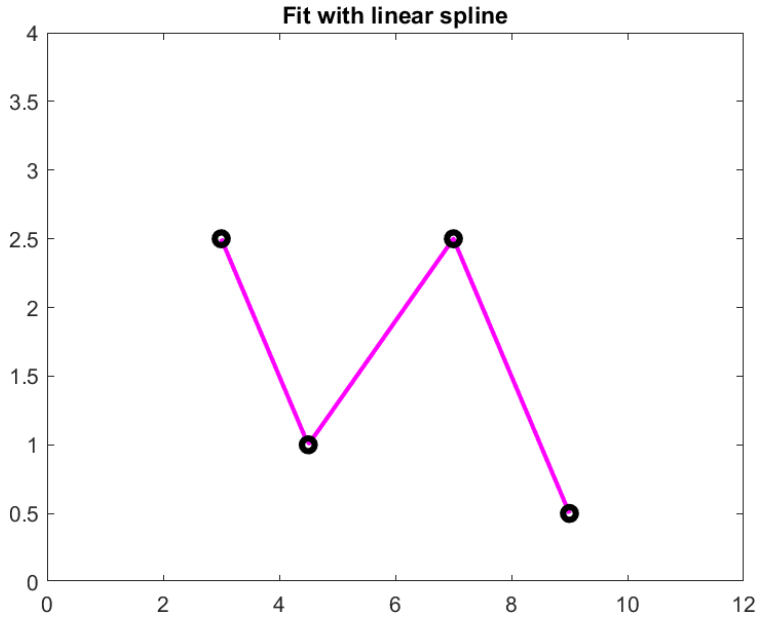
End conditions

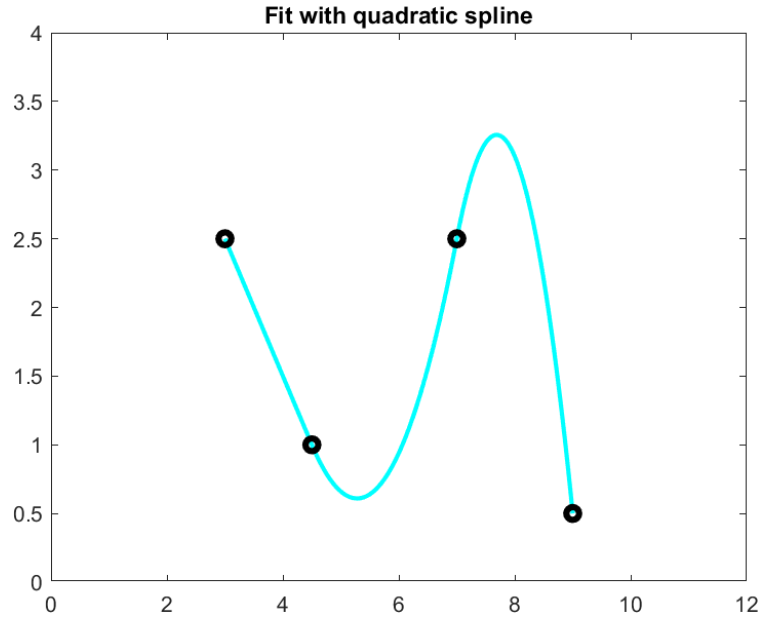
- **Natural end conditions** - assume the second derivative at the end knots are zero.

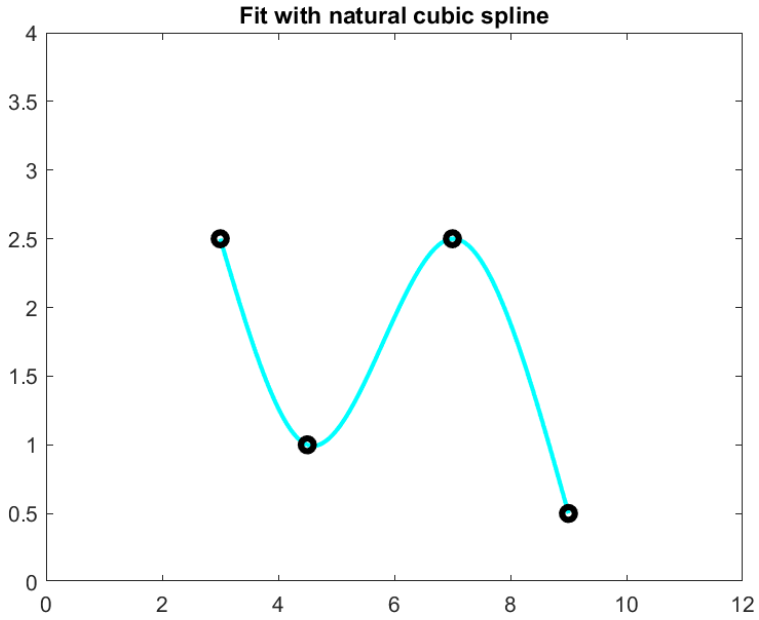
- **Clamped end conditions** - assume the first derivatives at the first and last knots are known.

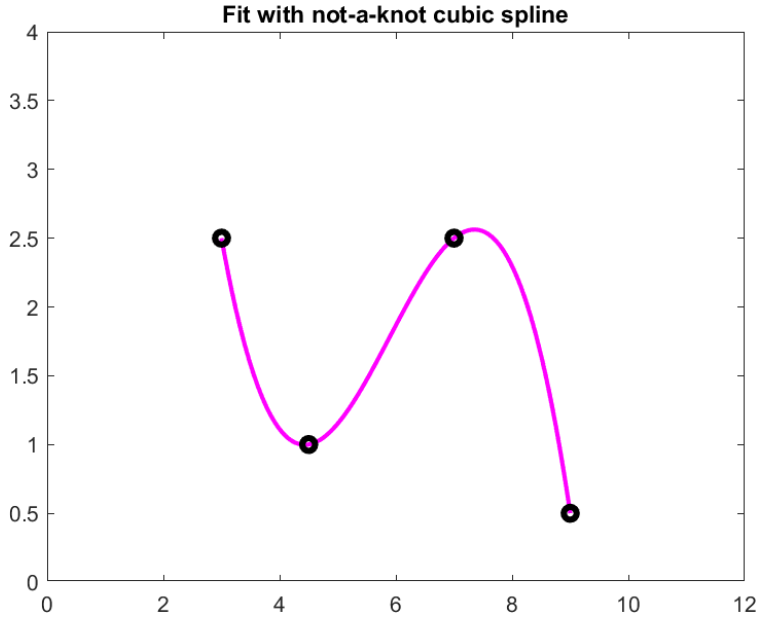
- **“Not-a-knot” end conditions** - force continuity of the *third* derivative at the second and the next-to-last knots (results in the first two intervals having the same spline function and the last two intervals having the same spline function). Since the first internal knots no longer represent the junction of two different cubic functions, they are no longer true knots, thus referred to as the “not-a-knot” condition.
  - There is an additional property in the case of 4 points, the not-a-knot spline is the same as a normal cubic interpolating polynomial.

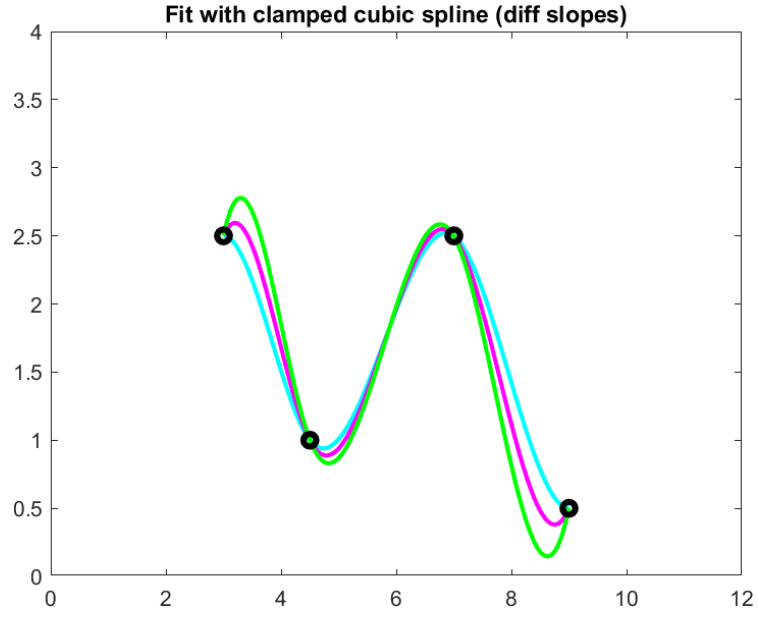
Cubic spline examples



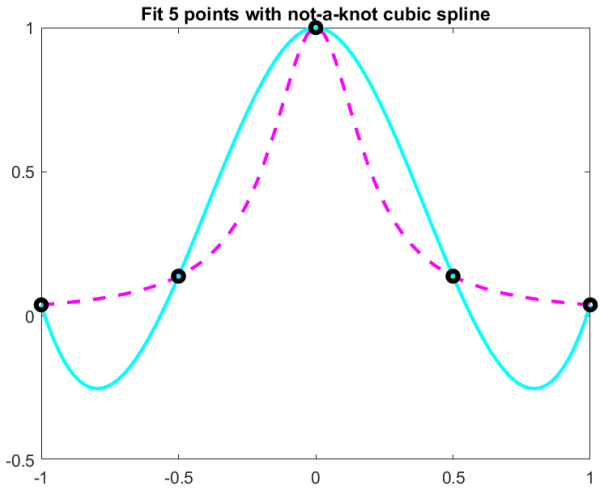


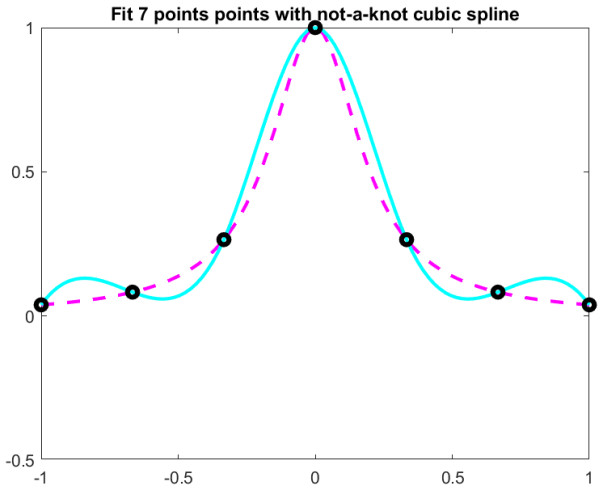


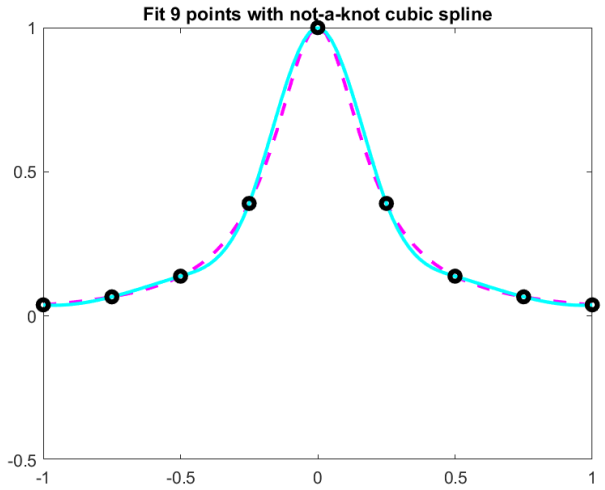


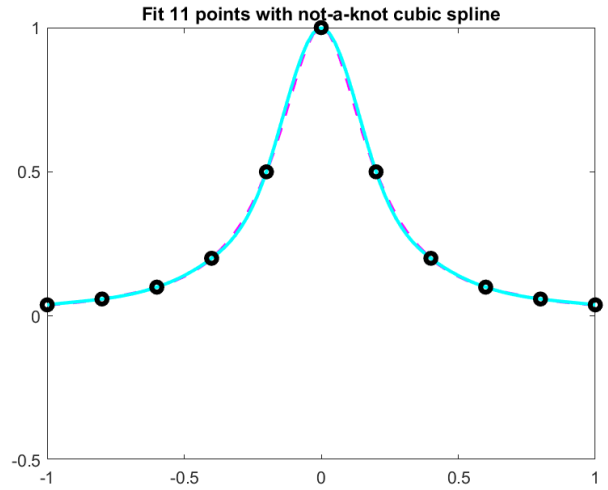


Fit the Runge's function with splines









Piecewise Interpolation in MATLAB

MATLAB has several built-in functions to implement piecewise interpolation. The first is spline:

```
yy = spline(x, y, xx)
```

This performs cubic spline interpolation, generally using not-a-knot conditions. If `y` contains two more values than `x` has entries, then the first and last value in `y` are used as the derivatives at the end points (i.e. clamped)

Not-a-knot Example

Generate data:

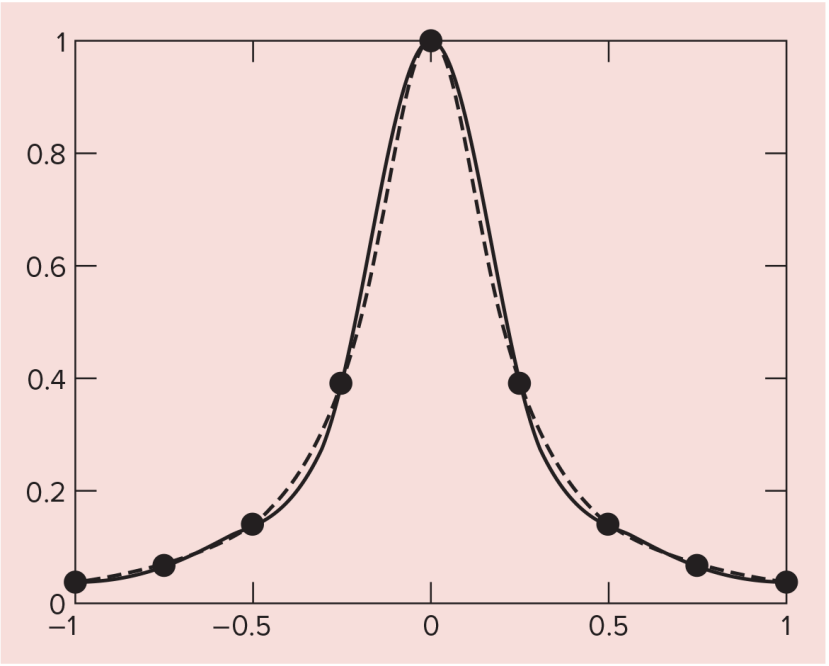
```
x = linspace(-1, 1, 9);
y = 1./(1+25*x.^2);
```

Calculate 100 model points and determine not-a-knot interpolation

```
xx = linspace(-1, 1);
yy = spline(x, y, xx);
```

Calculate actual function values at model points and data points, the 9-point not-a-knot interpolation (solid), and the actual function (dashed),

```
yr = 1./(1+25*xx.^2)
plot(x, y, 'o', xx, yy, '-', xx, yr, '--')
```



Clamped Example

Generate data w/ first derivative information at beginning and end of dependent variable vector:

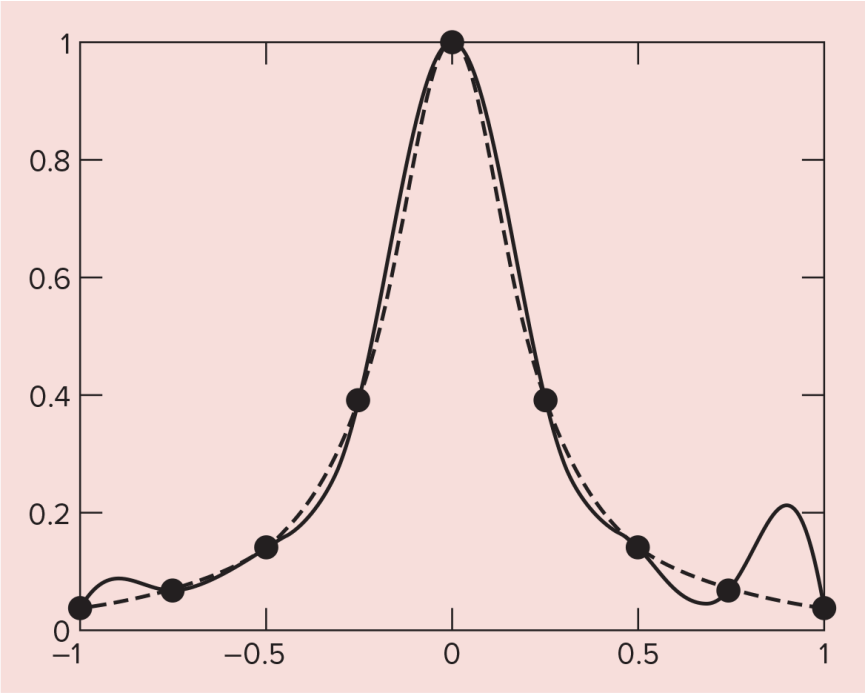
```
x = linspace(-1, 1, 9);
y = 1./(1+25*x.^2);
yc = [1 y -4]
```

Calculate 100 model points and determine not-a-knot interpolation

```
xx = linspace(-1, 1);
yyc = spline(x, yc, xx);
```

Calculate actual function values at model points and data points, the 9-point clamped interpolation (solid), and the actual function (dashed),

```
yr = 1./(1+25*xx.^2)
plot(x, y, 'o', xx, yyc, '-', xx, yr, '--')
```



MATLAB's `interp1` Function

While `spline` can only perform cubic splines, MATLAB's **`interp1`** function can perform several different kinds of interpolation:

```
yi = interp1(x, y, xi, 'method')
```

`x` & **`y`** contain the original data.

`xi` contains the points at which to interpolate.

`'method'` is a string containing the desired method:

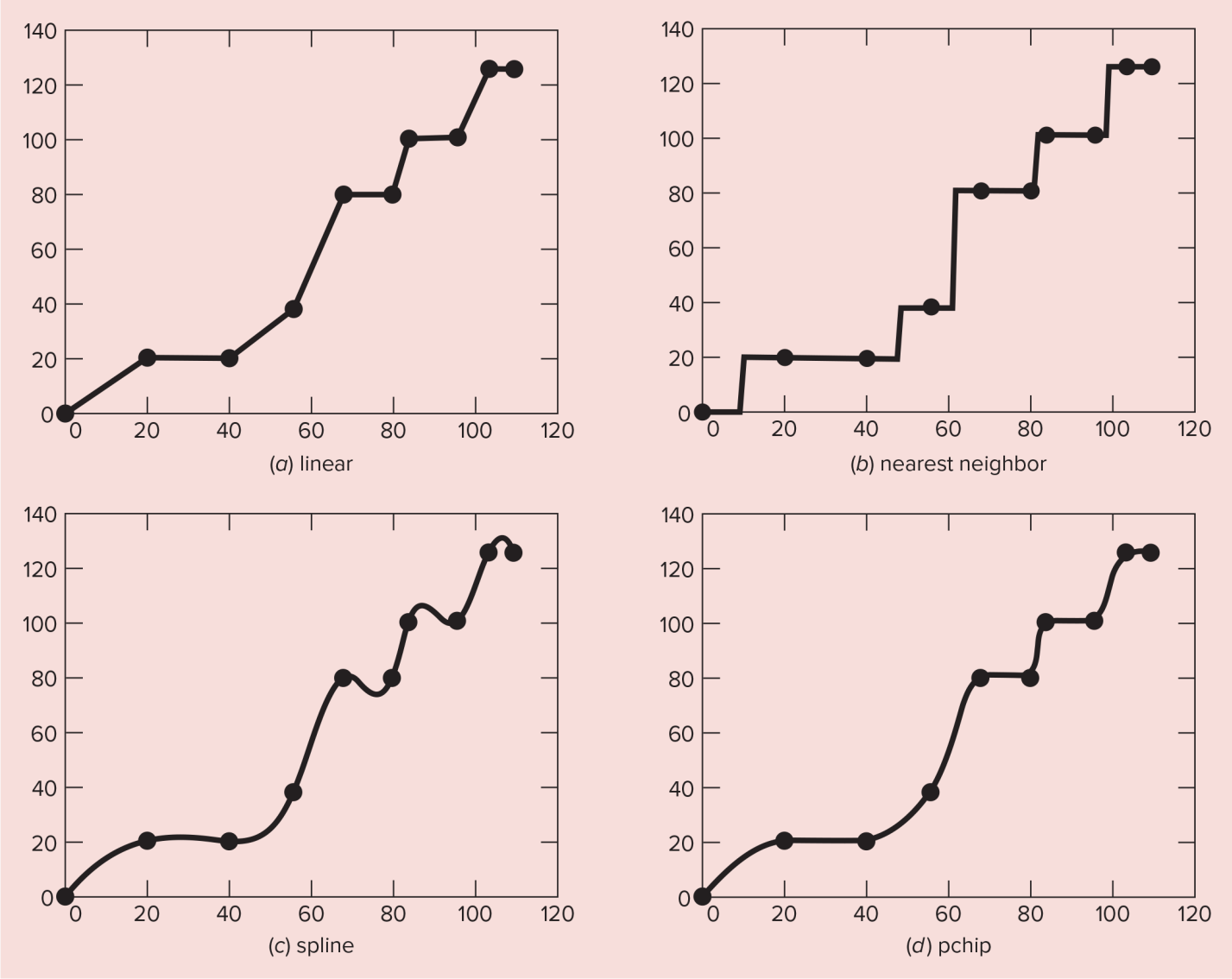
`'nearest'` - nearest neighbor interpolation.

`'linear'` - connects the points with straight lines.

`'spline'` - not-a-knot cubic spline interpolation.

`'pchip'` or **`'cubic'`** - piecewise cubic Hermite interpolation.

Piecewise Polynomial Comparisons

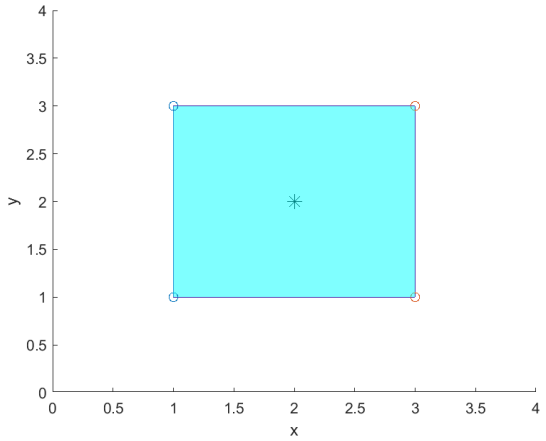


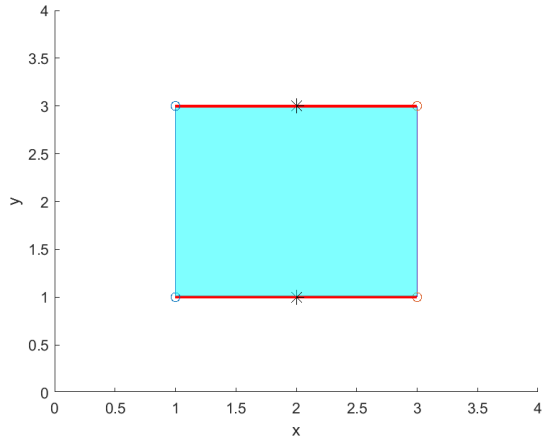
Multidimensional interpolation (Bilinear interpolation)

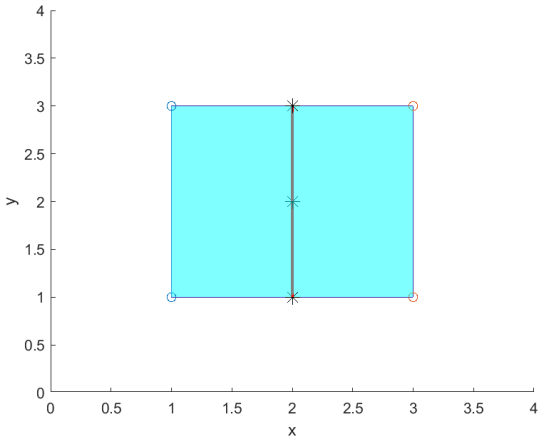
- Extend from one-dimensional interpolation to two-dimensional interpolation

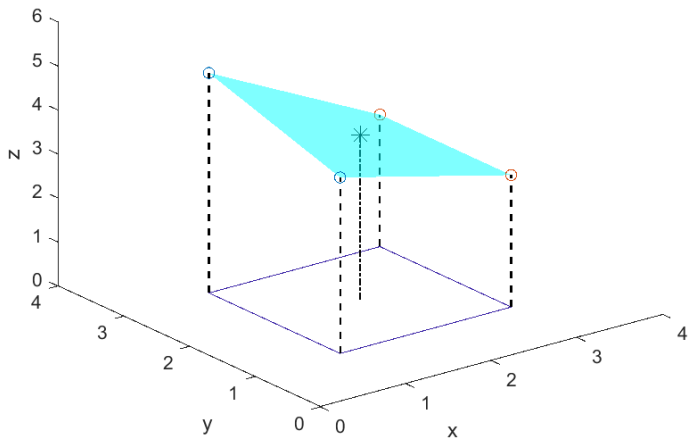
- Suppose we have values at 4 points $f(x_1, y_1), f(x_2, y_1), f(x_1, y_2), f(x_2, y_2)$ and we want to determine intermediate values $z = f(x_i, y_i)$
  - First applying one-dimensional linear interpolation along the x dimension to determine values at x_i.
  - Then applying another linear interpolation along the y dimension to find the final result at x_i, y_i.

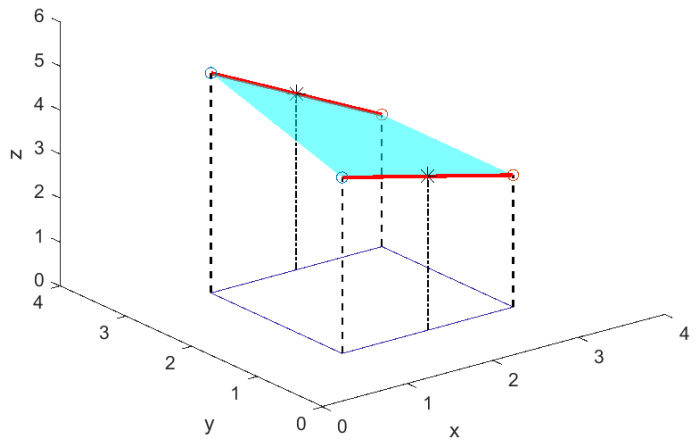
Bilinear interpolation

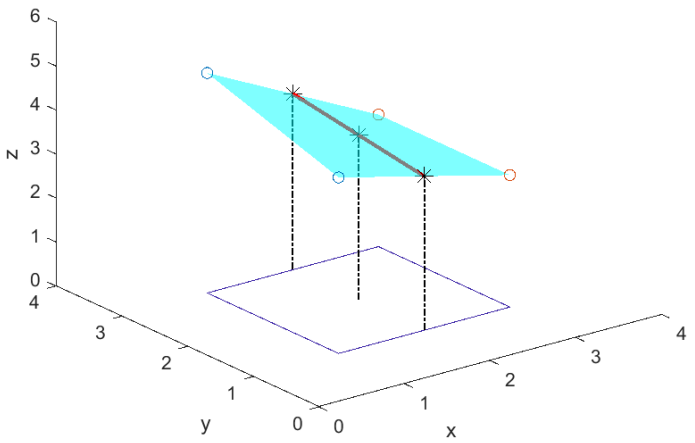












Bilinear interpolation

- First, hold the y value fixed and apply one-dimensional linear interpolation in the x direction. Using the Lagrange form,

$$f(x_i, y_1) = \frac{x_i - x_2}{x_1 - x_2} f(x_1, y_1) + \frac{x_i - x_1}{x_2 - x_1} f(x_2, y_1)$$

$$f(x_i, y_2) = \frac{x_i - x_2}{x_1 - x_2} f(x_1, y_2) + \frac{x_i - x_1}{x_2 - x_1} f(x_2, y_2)$$

- Second, these points then is linearly interpolated along the y dimension

$$f(x_i, y_i) = \frac{y_i - y_2}{y_1 - y_2} f(x_i, y_1) + \frac{y_i - y_1}{y_2 - y_1} f(x_i, y_2)$$

Bilinear interpolation

- A combined equation

$$f(x_i, y_i) = \frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_2}{y_1 - y_2} f(x_1, y_1) + \frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_2}{y_1 - y_2} f(x_2, y_1)$$

$$+ \frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_1}{y_2 - y_1} f(x_1, y_2) + \frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_1}{y_2 - y_1} f(x_2, y_2)$$

Example of bilinear interpolation

- For these 4 points, $P(1,1) = 4,\ P(1,3) = 5,\ P(3,1) = 3,\ P(3,3) = 3,$ use bilinear interpolation to estimate value of $P(2,2)$.

$$f(x_i, y_i) = \frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_2}{y_1 - y_2} f(x_1, y_1) + \frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_2}{y_1 - y_2} f(x_2, y_1)$$

$$+ \frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_1}{y_2 - y_1} f(x_1, y_2) + \frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_1}{y_2 - y_1} f(x_2, y_2)$$

$$f(2,2) = \frac{2-3}{1-3}\frac{2-3}{1-3}(4) + \frac{2-1}{3-1}\frac{2-3}{1-3}(3) + \frac{2-3}{1-3}\frac{2-1}{3-1}(5) + \frac{2-1}{3-1}\frac{2-1}{3-1}(3) = 3.75$$

Multidimensional Interpolation in MATLAB

MATLAB has built-in functions for two- and three-dimensional piecewise interpolation:

```
zi = interp2(x, y, z, xi, yi, 'method')
vi = interp3(x, y, z, v, xi, yi, zi, 'method')
```

`'method'` is again a string containing the desired method: **`'nearest'`**, **`'linear'`**, **`'spline'`**, **`'pchip'`** or **`'cubic'`**

For 2-D interpolation, the inputs must either be vectors or same-size matrices.

For 3-D interpolation, the inputs must either be vectors or same-size 3-D arrays.