Curve Fitting: Interpolation

CH EN 2450
Numerical Methods
Spring 2022

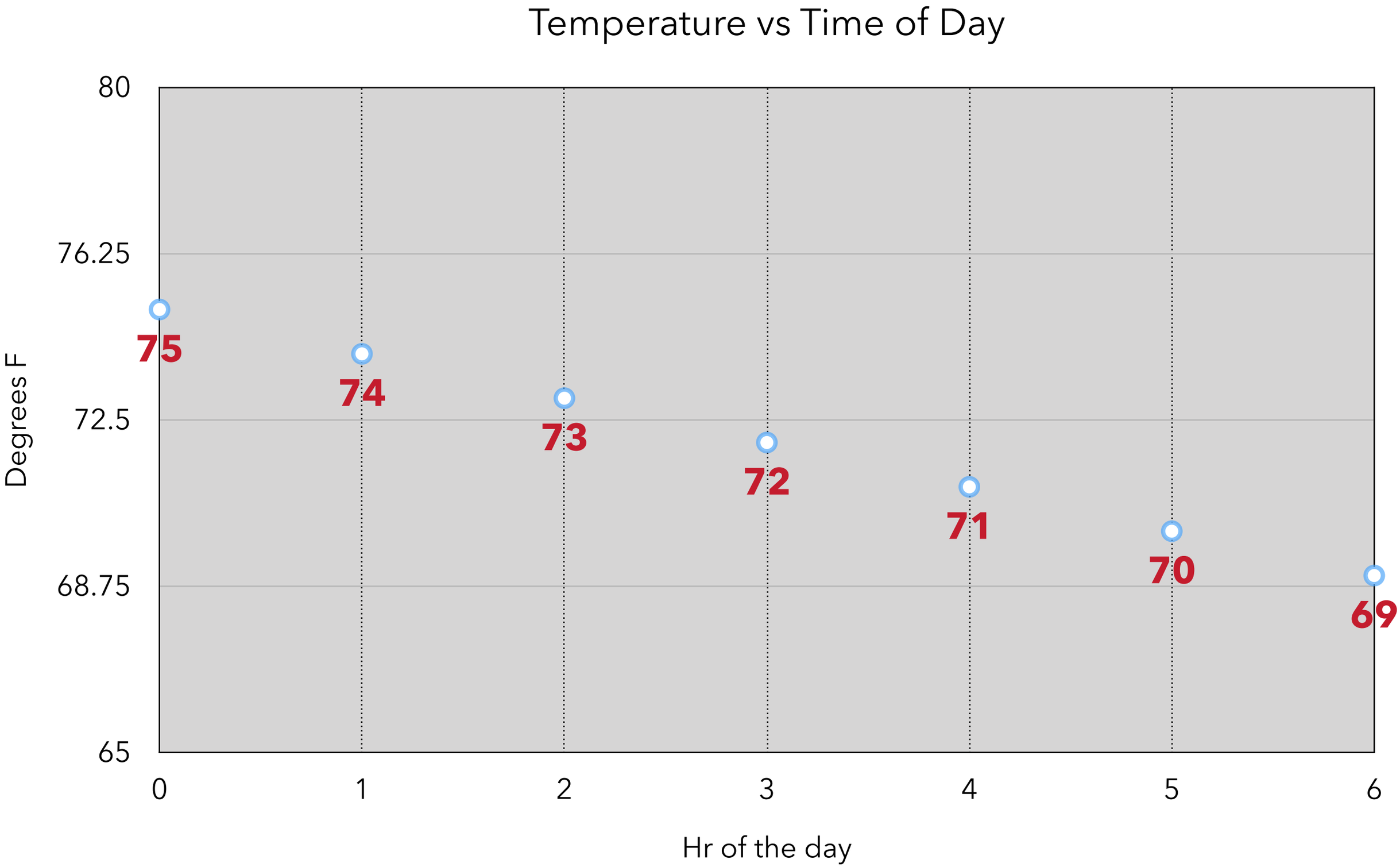
Prof. Tony Saad

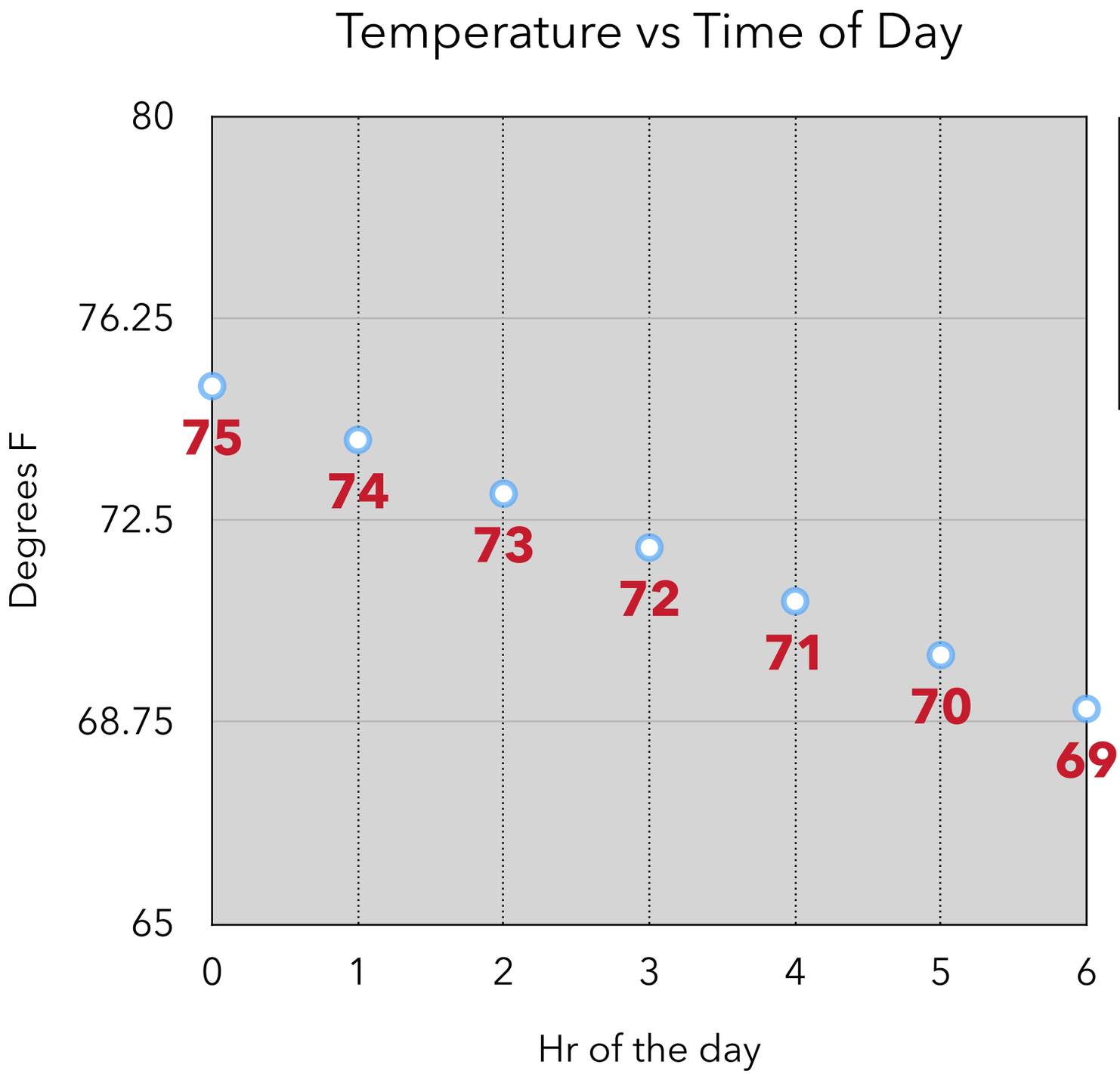
Department of Chemical Engineering
University of Utah

Learning Objectives

At the end of this chapter you should be able to:

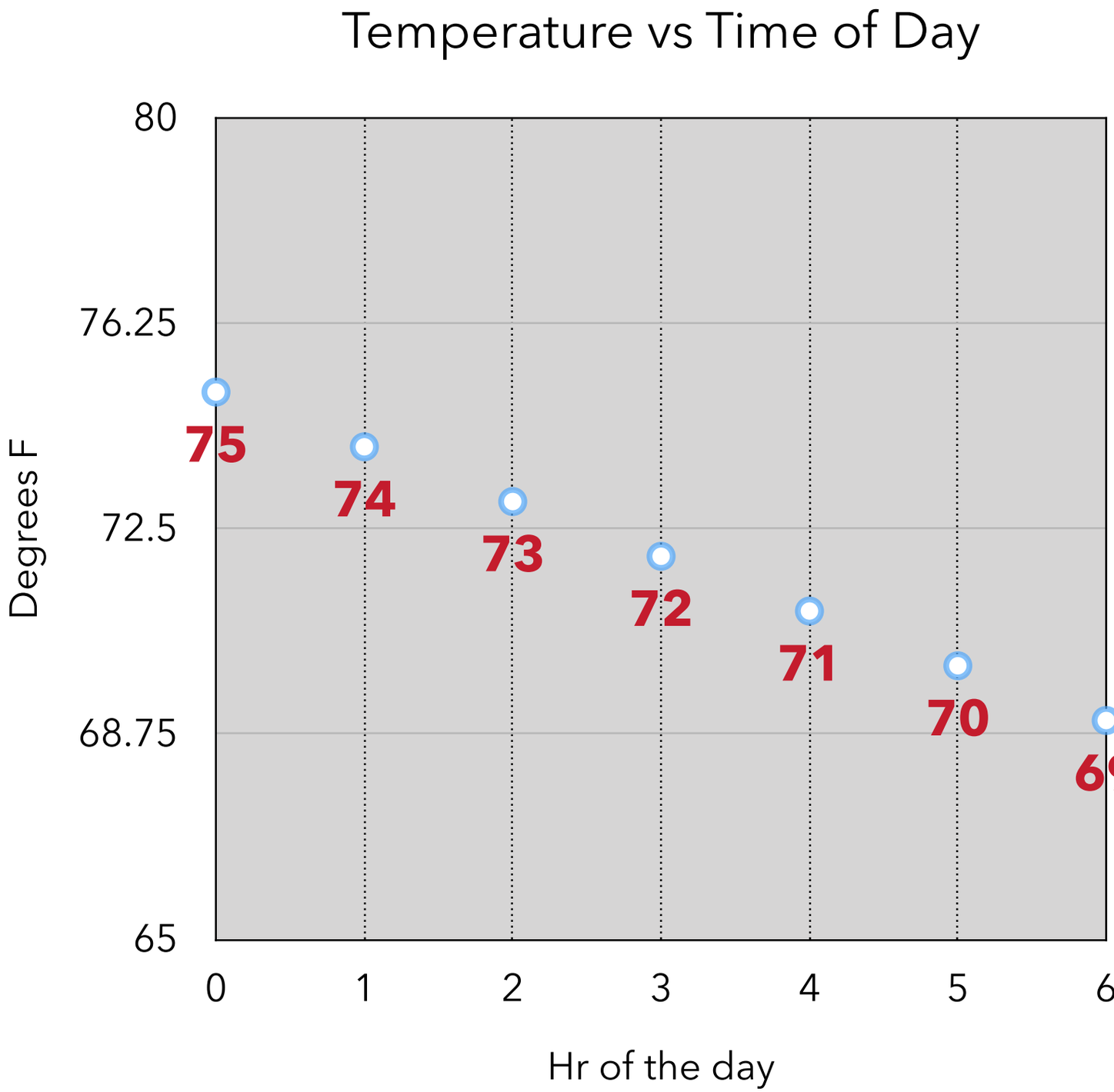
- Define what it means to interpolate data
- Identify the order of the polynomial that interpolates n-data points
- Derive the system of equations that one needs to solve to build a polynomial interpolant
- List the benefits and limitations of first order interpolates
- List the benefits and limitations of high-order polynomial interpolants
- Calculate by hand an interpolated value using a linear interpolant
- Identify which points are best used with a linear and quadratic interpolant
- List the conditions for cubic spline interpolants - only the conditions - and show how many unknowns there are and what the constraints are and be able to list all the constraints
- Use built-in Python functions to perform polynomial and cubic spline interpolation





Activity:

What is the temperature at 5:30 AM?

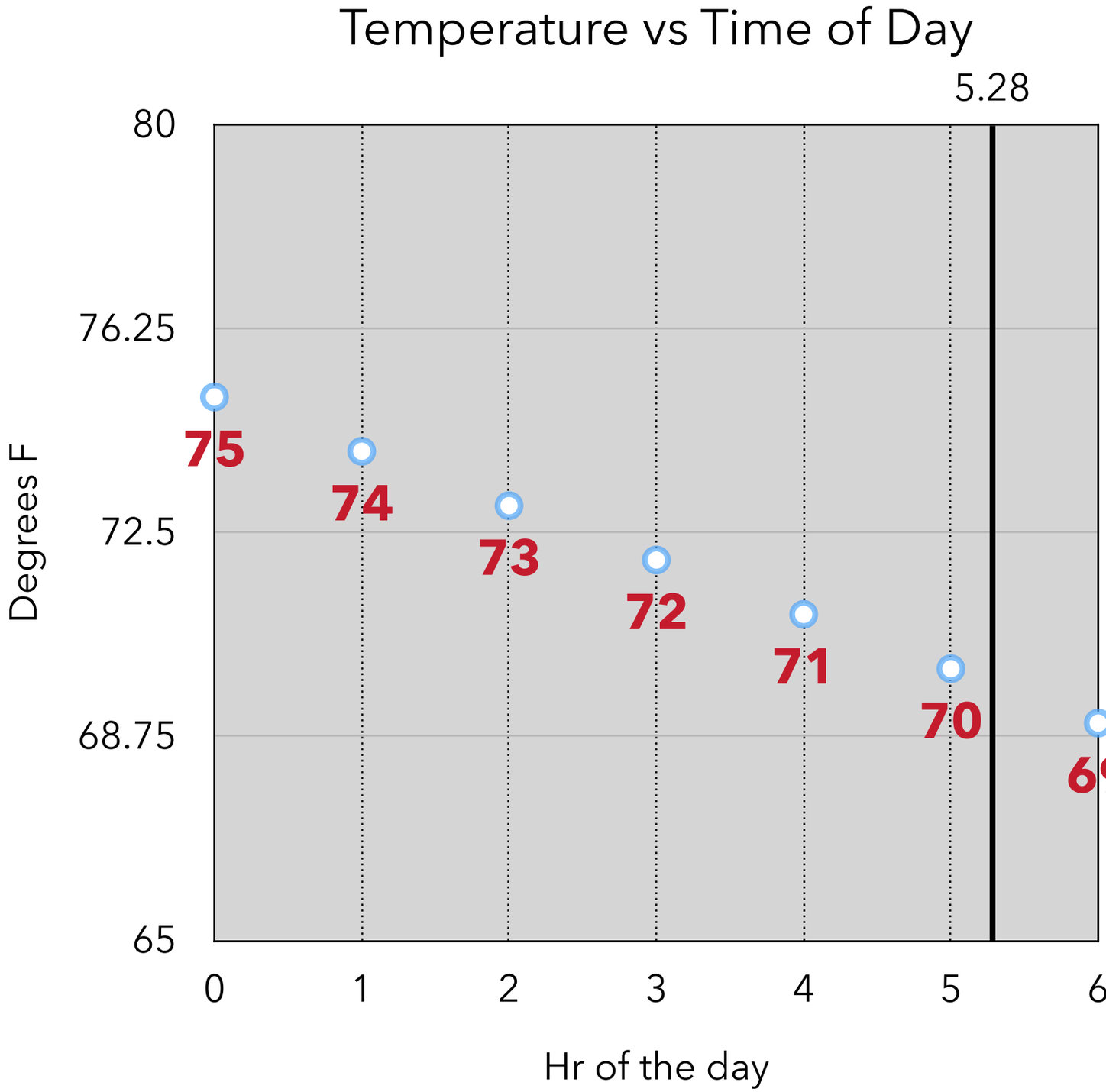


Activity:

What is the temperature at 5:30 AM?

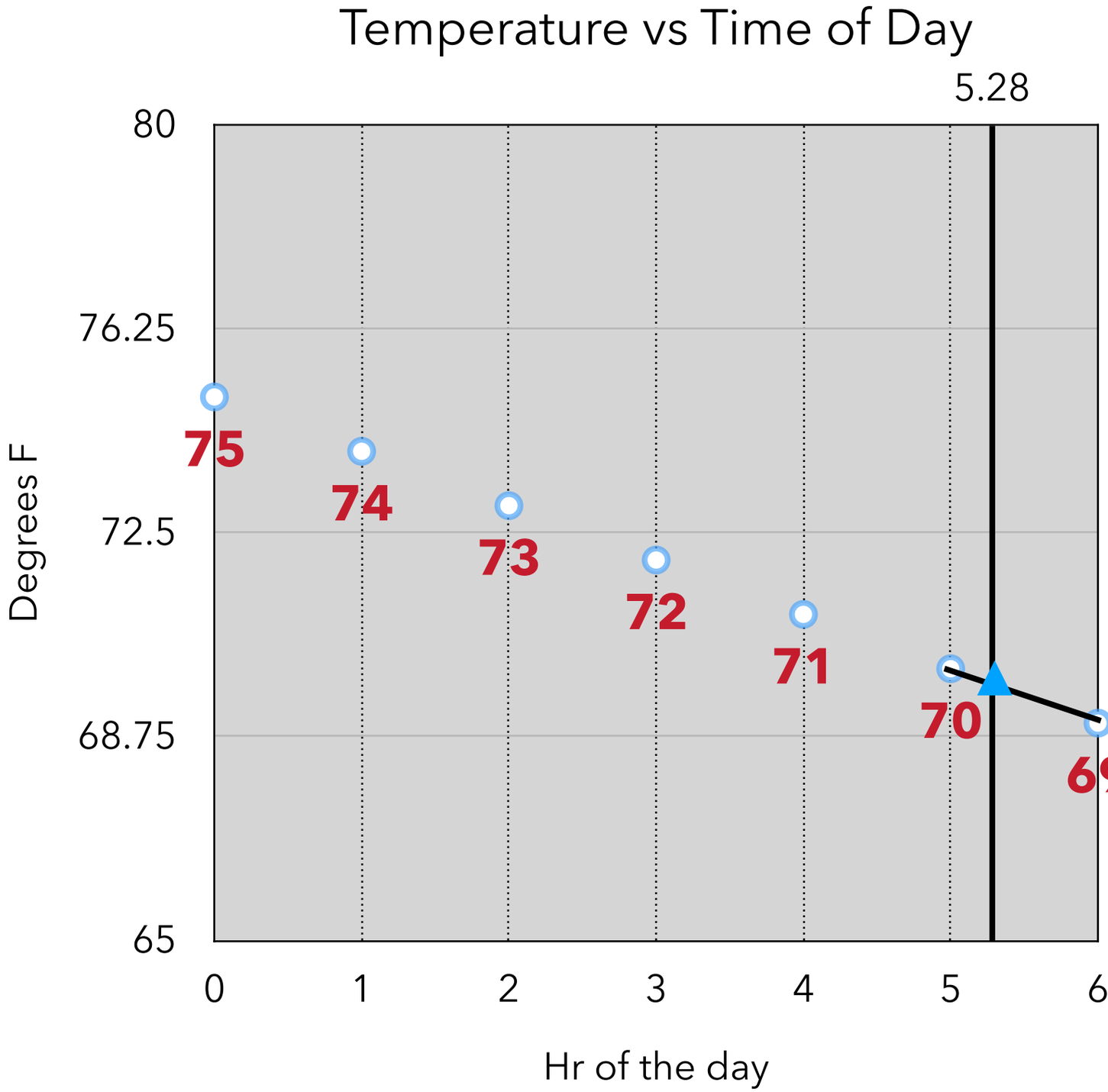
At 5:30 AM, the temperature is likely to be the average between 70 and 69:

T(5:30AM) = 69.5 F



Activity:

What is the temperature at 5:17 AM? (That's at hr 5.28). You only need to describe how you would find that temperature.



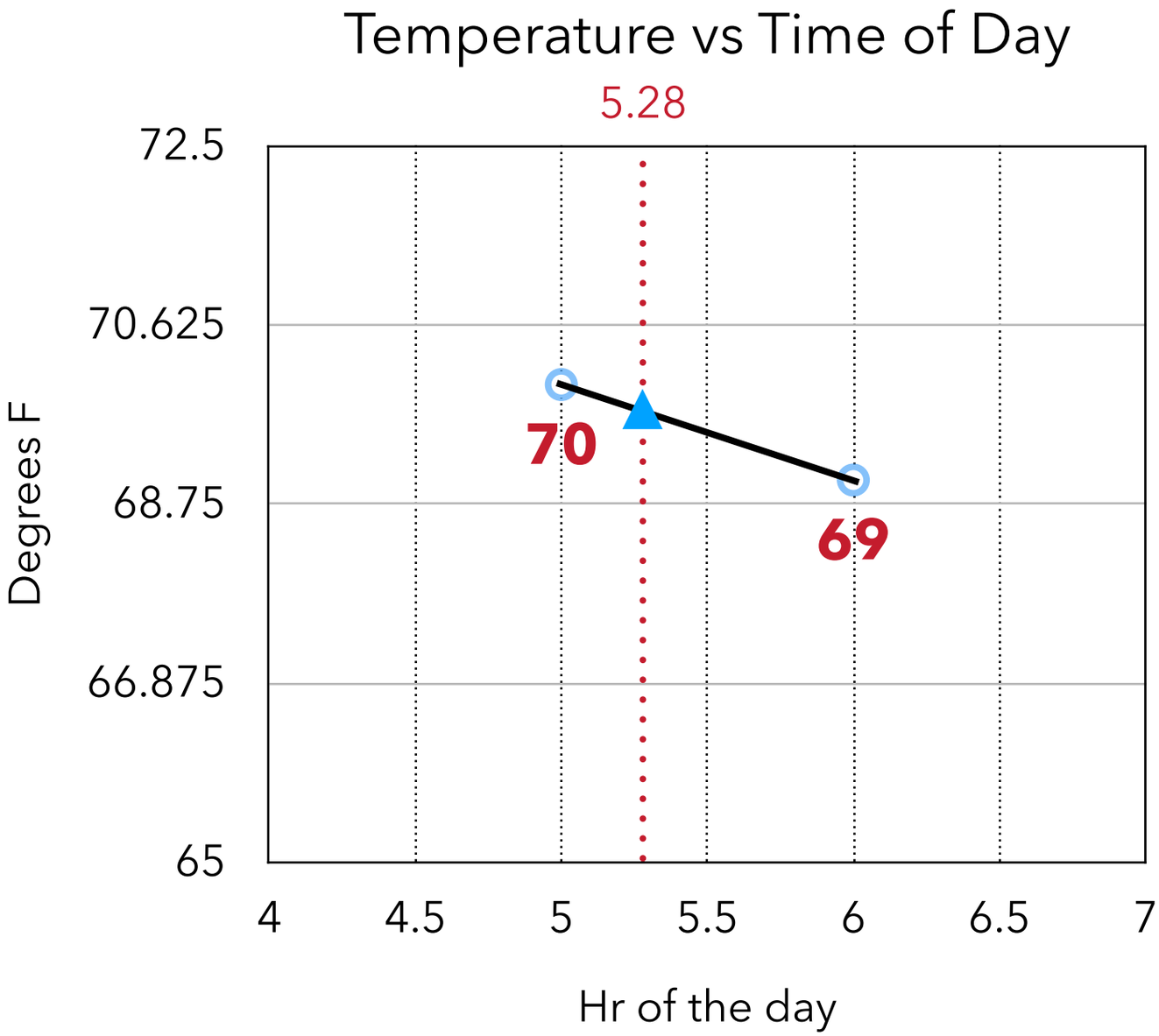
Activity:

What is the temperature at 5:17 AM? (That's at hr 5.28). You only need to describe how you would find that temperature.

Draw a straight line between 70 and 69 and intersect it with the vertical line at 5.28

Activity:

Go ahead and do that



We need an equation for the line connecting the points (5,70) and (6,69).

Equation for a straight line is: $T(x) = mx + b$ **T is temperature and x is hr of day**

Slope is: $m = \dfrac{69 - 70}{6 - 5} = -1$

To find the intercept, evaluate the equation of the line at one of the known points, $T(5) = 70$ or $T(6) = 69$

$$T(5) = 70 = m \times 5 + b = -5 + b \implies b = 75$$

Finally, the temperature at 5:17 AM (x = 5.28 hr) is:

$$T(5.28) = -5.28 + 75 = 69.72F$$

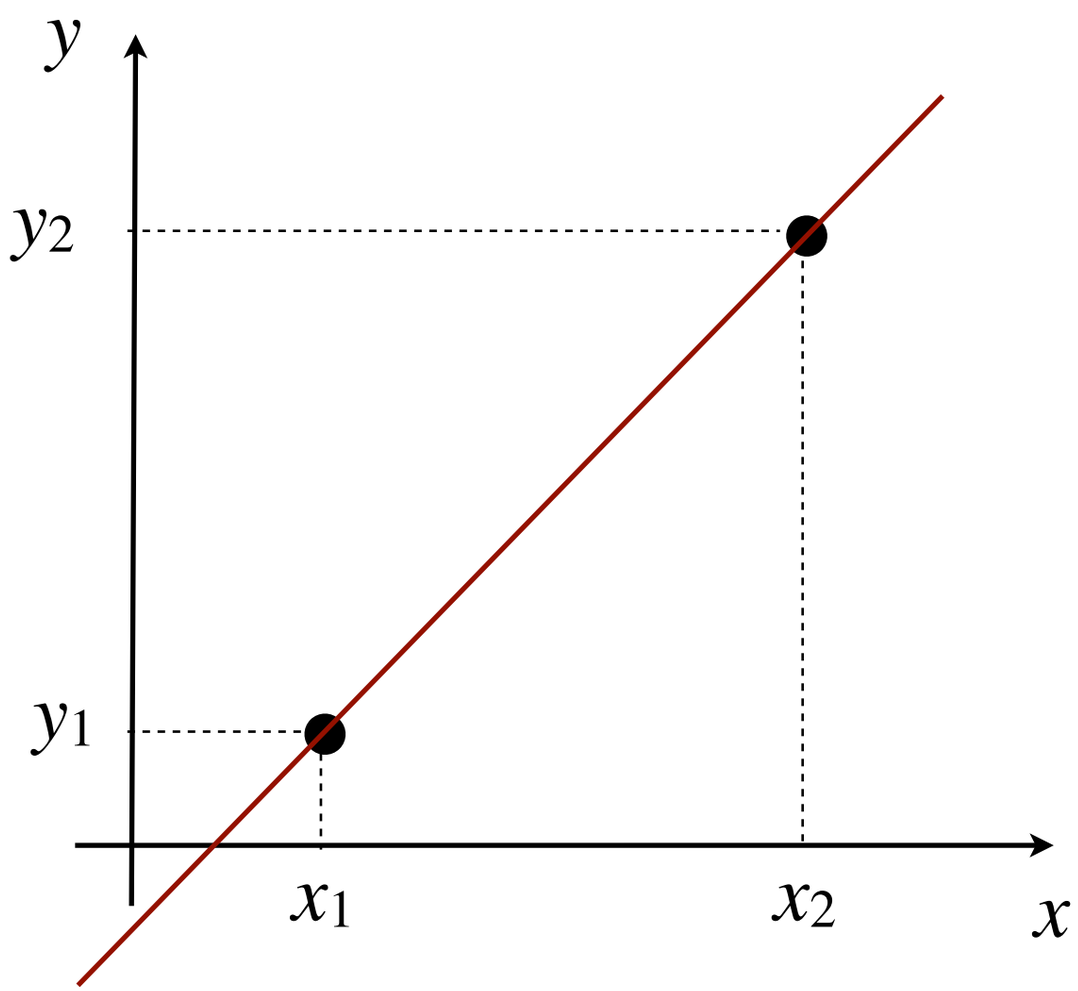
You just did linear interpolation!

Let us Generalize

Linear Interpolation

given: $(x_1, y_1) \quad (x_2, y_2)$

find: $y(x) = mx + b$



Activity:

Given two points (x_1, y_1) and (x_2, y_2), find the general formula for a straight line (linear interpolant) between these two points.

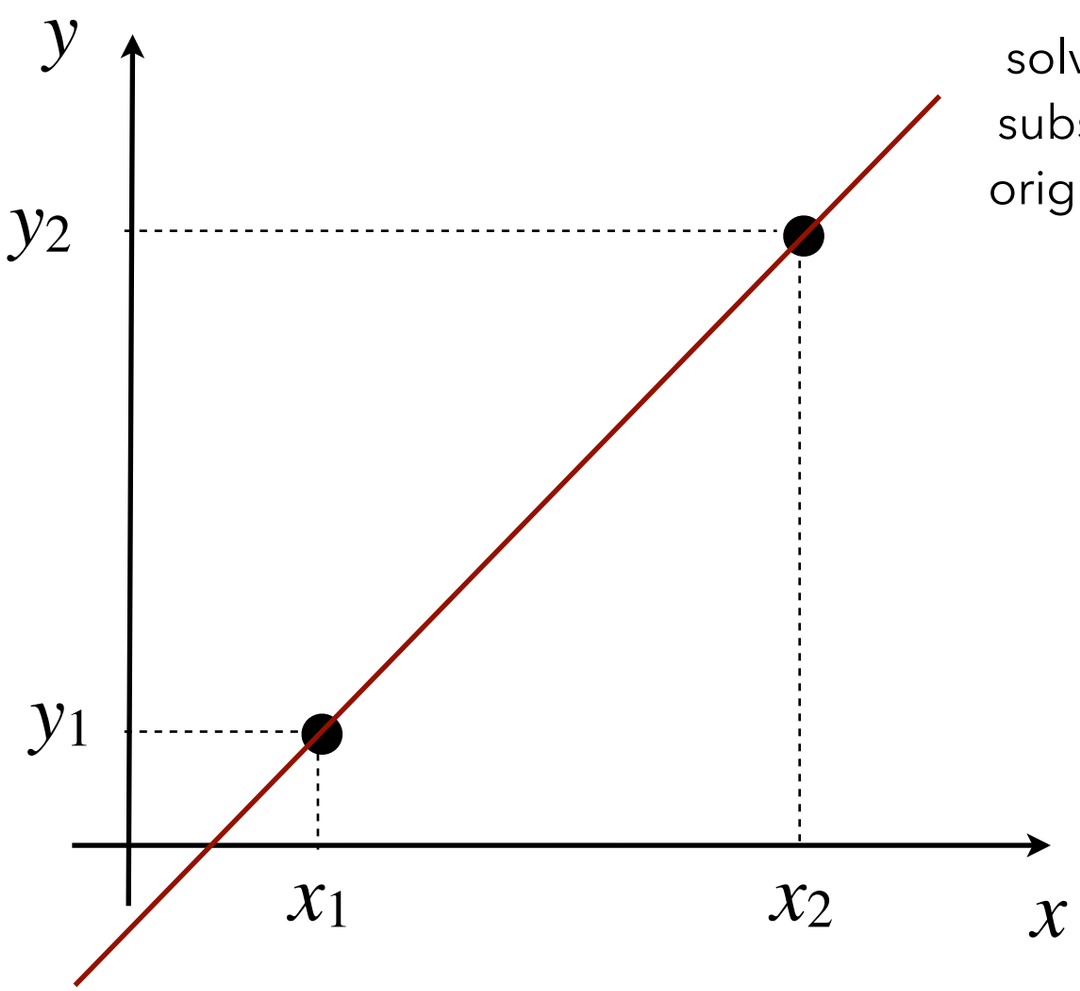
Linear Interpolation

given: (x_1, y_1) (x_2, y_2)

find: $y(x) = mx + b$

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$



solve for m, b and substitute into the original equation...

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

Advantages:

- Easy to use (homework, exams)

Disadvantages:

- Not very accurate for nonlinear functions

Motivation & Concept

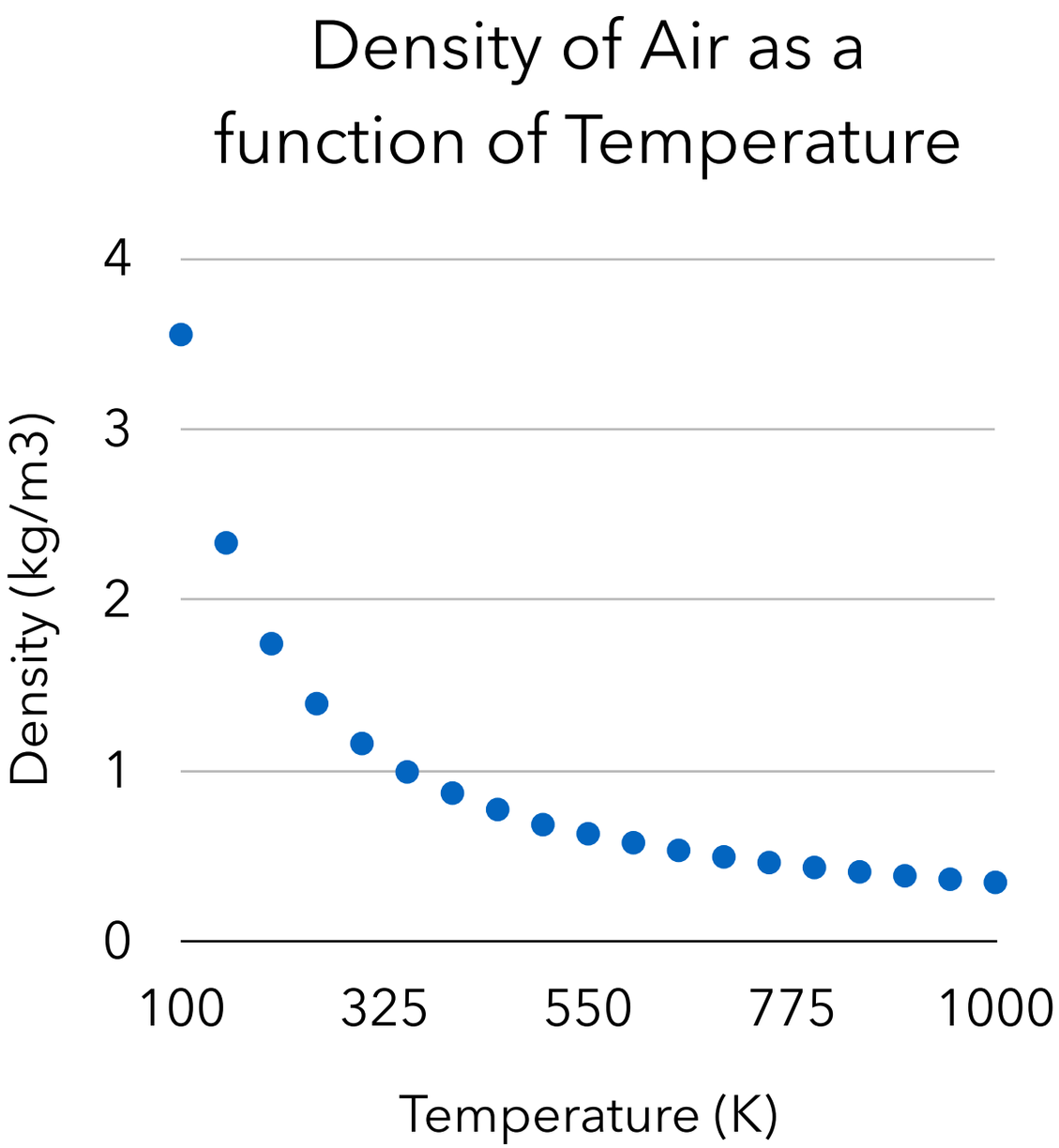
- Motivation:
  - Often we have discrete data (tabulated, from experiments, etc) that we need to interpolate.
- Concept:
  - Choose a polynomial function to fit to the data (connect the dots)
  - Solve for the coefficients of the polynomial
  - Evaluate the polynomial wherever you want (interpolation)

Properties of air at atmospheric pressure

T K	ρ kg/m³	λ W/(m K)	μ N s/m²
100	3.5562	0.0093	7.110e-06
150	2.3364	0.0138	1.034e-05
200	1.7458	0.0181	1.325e-05
250	1.3947	0.0223	1.596e-05
300	1.1614	0.0263	1.846e-05
350	0.9950	0.0300	2.082e-05
400	0.8711	0.0338	2.301e-05
450	0.7750	0.0373	2.507e-05
500	0.6864	0.0407	2.701e-05
550	0.6329	0.0439	2.884e-05
600	0.5804	0.0469	3.058e-05
650	0.5356	0.0497	3.225e-05
700	0.4975	0.0524	3.388e-05
750	0.4643	0.0549	3.546e-05
800	0.4354	0.0573	3.698e-05
850	0.4097	0.0596	3.843e-05
900	0.3868	0.0620	3.981e-05
950	0.3666	0.0643	4.113e-05
1000	0.3482	0.0667	4.244e-05

Incropera & DeWitt, *Fundamentals of Heat and Mass Transfer*, 4th ed.

Motivation & Concept



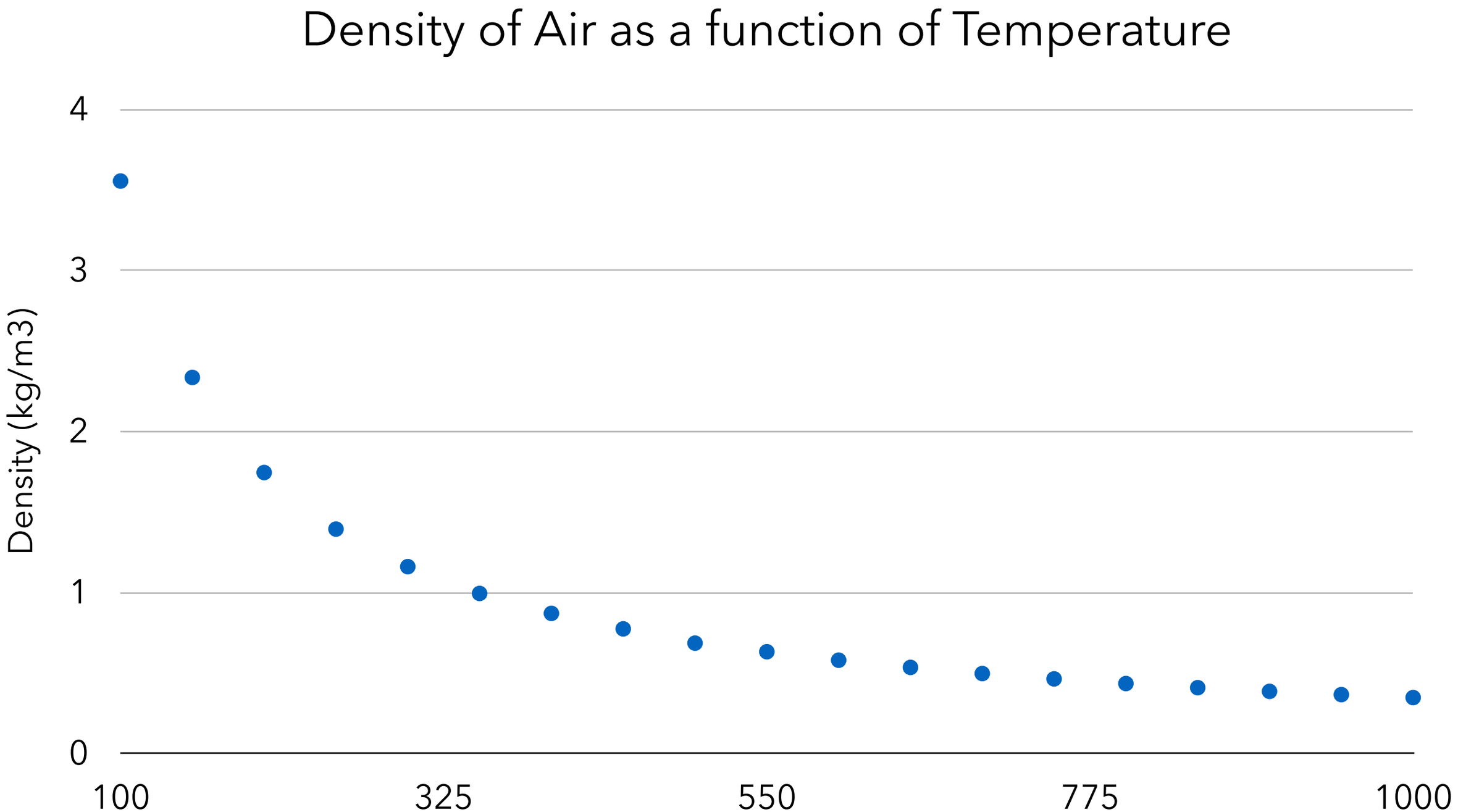
Properties of air at atmospheric pressure

T K	ρ kg/m³	λ W/(m K)	μ N s/m²
100	3.5562	0.0093	7.110e-06
150	2.3364	0.0138	1.034e-05
200	1.7458	0.0181	1.325e-05
250	1.3947	0.0223	1.596e-05
300	1.1614	0.0263	1.846e-05
350	0.9950	0.0300	2.082e-05
400	0.8711	0.0338	2.301e-05
450	0.7750	0.0373	2.507e-05
500	0.6864	0.0407	2.701e-05
550	0.6329	0.0439	2.884e-05
600	0.5804	0.0469	3.058e-05
650	0.5356	0.0497	3.225e-05
700	0.4975	0.0524	3.388e-05
750	0.4643	0.0549	3.546e-05
800	0.4354	0.0573	3.698e-05
850	0.4097	0.0596	3.843e-05
900	0.3868	0.0620	3.981e-05
950	0.3666	0.0643	4.113e-05
1000	0.3482	0.0667	4.244e-05

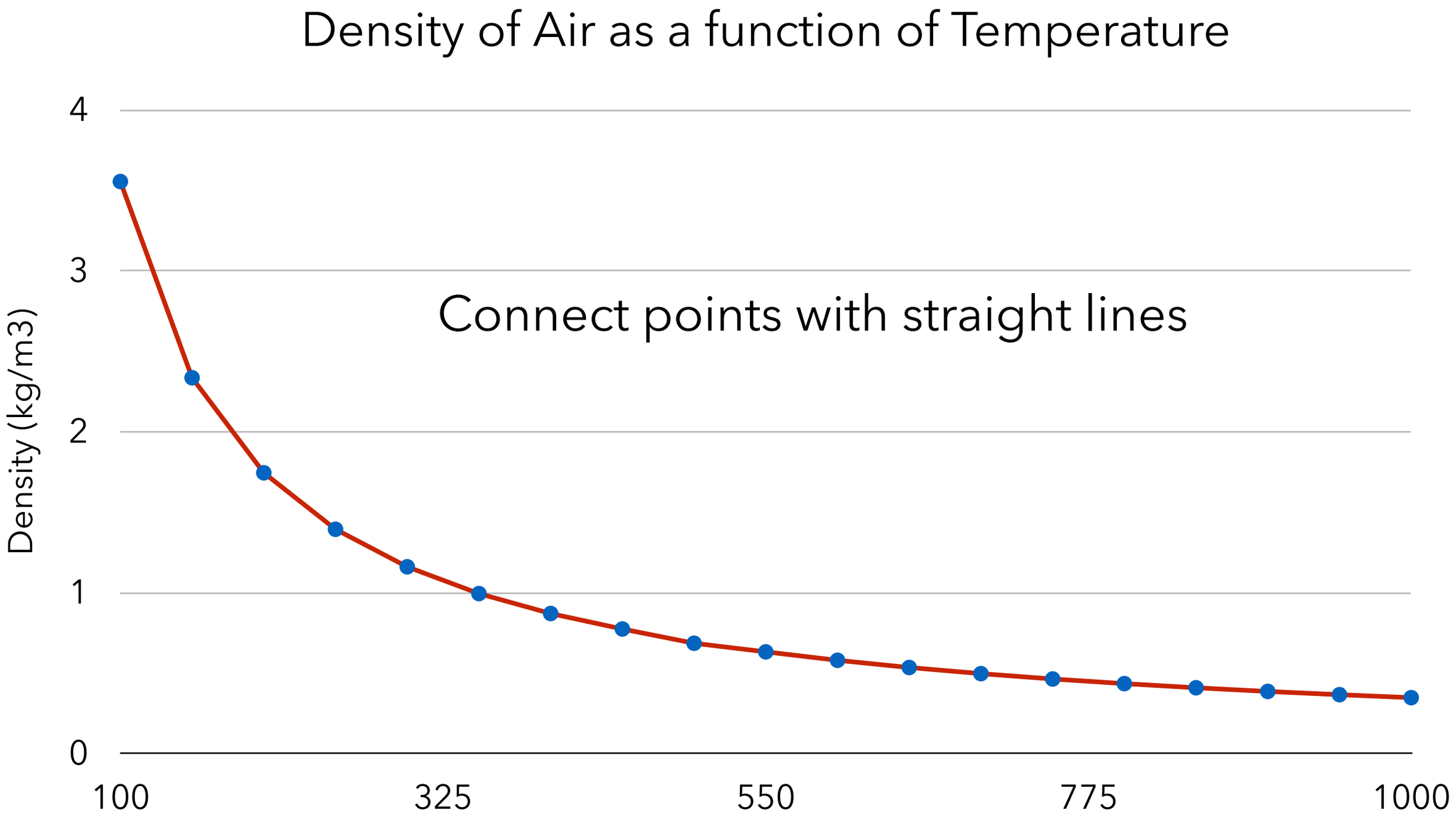
Incropera & DeWitt, *Fundamentals of Heat and Mass Transfer*, 4th ed.

Linear Interpolation

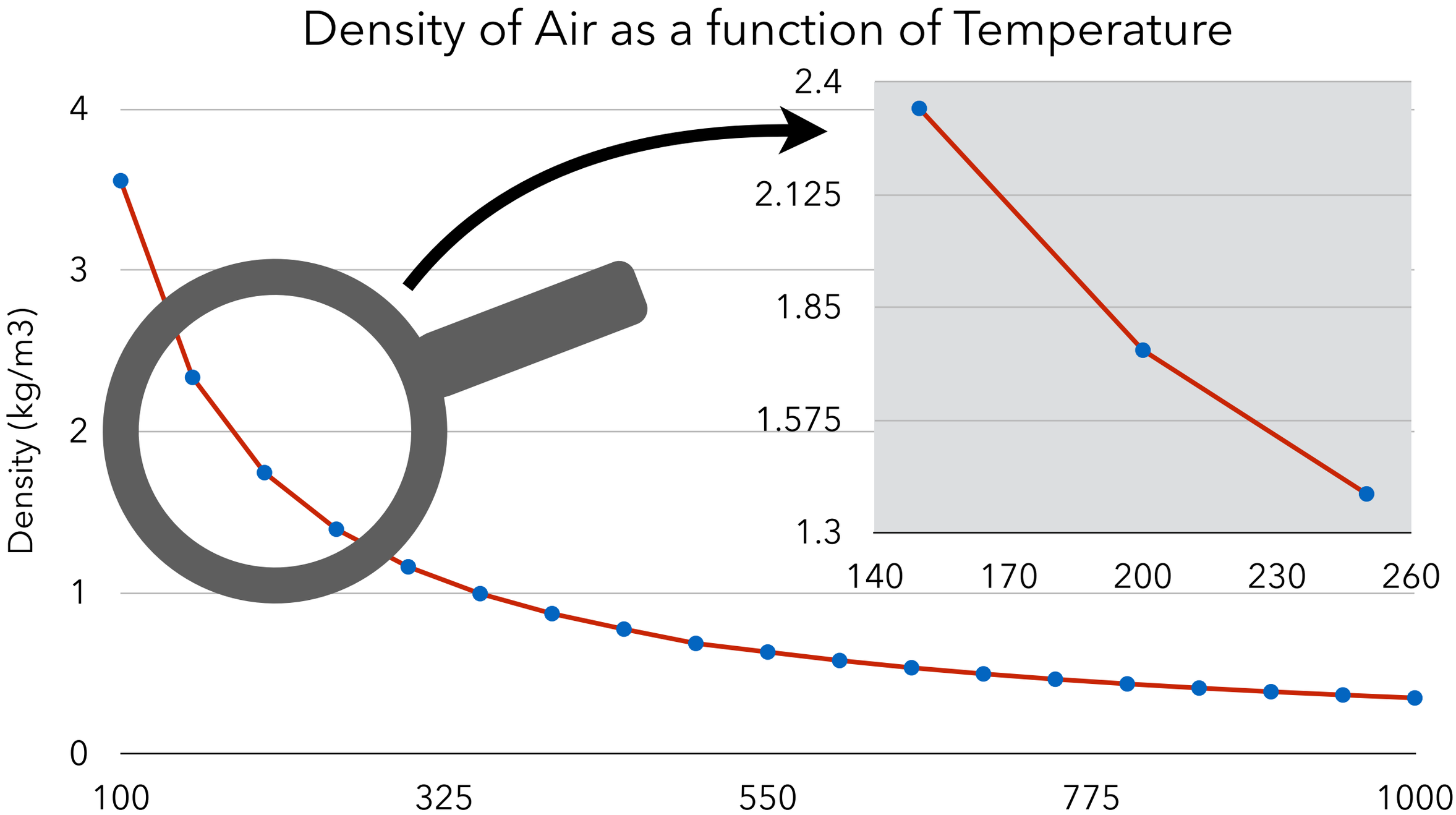
Linear Interpolation



Linear Interpolation



Linear Interpolation



Linear Interpolation

In Python:

```
ye = numpy.interp(xe,xi,yi)
```

- `xi` - (known) independent variable entries (vector)
- `yi` - (known) dependent variable entries (vector)
- `xe` - (known) value(s) where you want to interpolate
- `ye` - interpolated value(s) at `xe`

Coding Activity:
Download the notebook **Interp-activity-1-with-gaps.ipynb** from the Interpolation module on Canvas.

Linear Interpolation

```
ye = numpy.interp(xe,xi,yi)
```

- `xi` - (known) independent variable entries (vector)
- `yi` - (known) dependent variable entries (vector)
- `xe` - (known) value(s) where you want to interpolate
- `ye` - interpolated value(s) at `xe`

Achtung! The abscissae (xi) need to be sorted in increasing order!

- `np.argsort()` - Returns the indices that would sort an array. Useful when you need to sort multiple related arrays
- `np.sort()` - sorts array
- `np.flipud()` - flips array upside down

Higher Order

Polynomial Interpolants

Higher Order Interpolants

2 points
(x_1, y_1)
(x_2, y_2)

straight line $y(x) = a_0 + a_1 x$

$$y_1 = a_0 + a_1 x_1$$
$$y_2 = a_0 + a_1 x_2$$

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

3 points
(x_1, y_1)
(x_2, y_2)
(x_3, y_3)

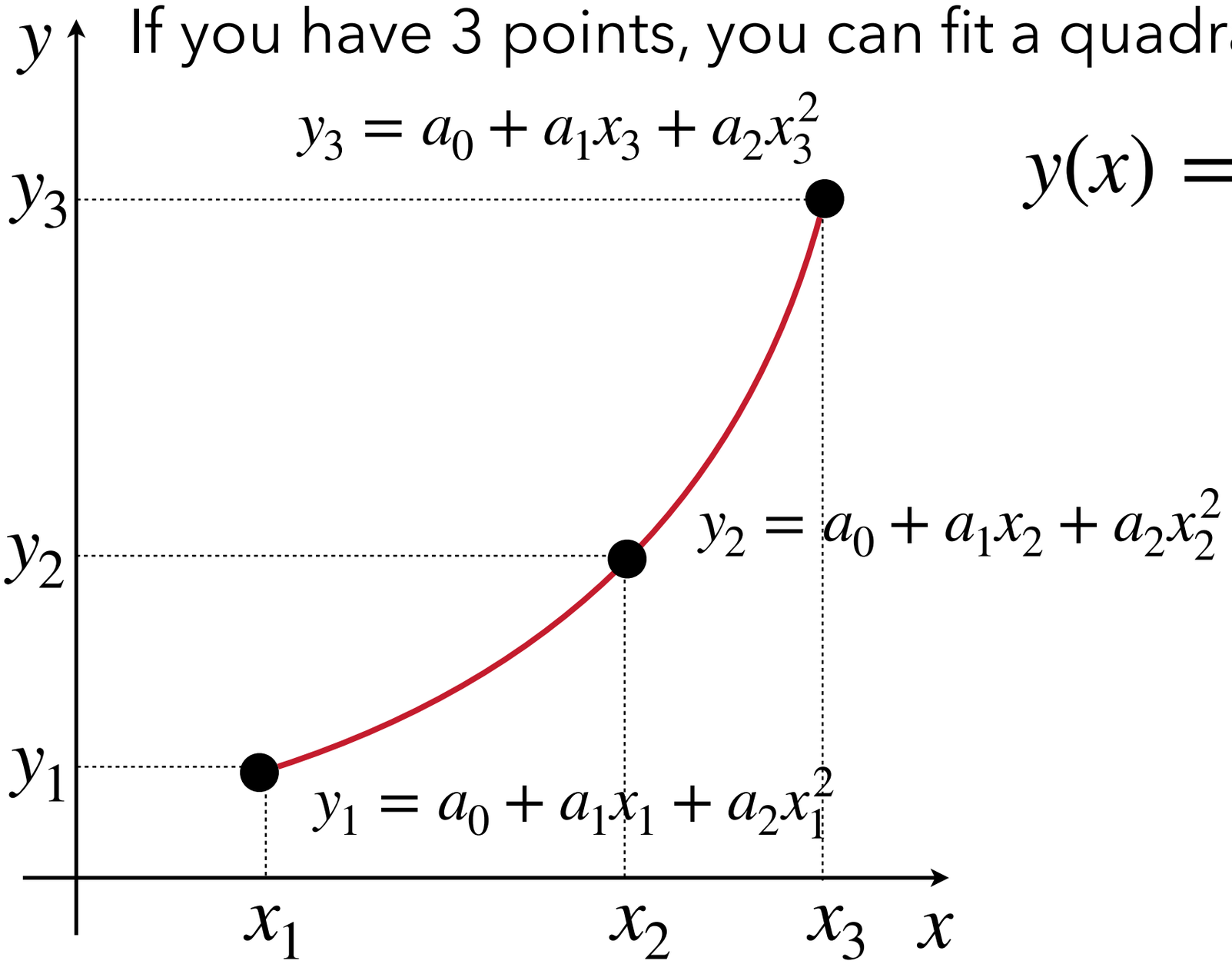
quadratic $y(x) = a_0 + a_1 x + a_2 x^2$

Activity:

Find the system of equations that we need to solve in order to find a_0, a_1, and a_2.

Higher Order Interpolants

If you have 3 points, you can fit a quadratic (parabola)



$$y(x) = a_0 + a_1 x + a_2 x^2$$

Higher Order Interpolants

$$y_1 = a_0 + a_1 x_1 + a_2 x_1^2$$

$$y_2 = a_0 + a_1 x_2 + a_2 x_2^2$$

$$y_3 = a_0 + a_1 x_3 + a_2 x_3^2$$

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Higher Order Interpolants

# data points	fit	equation
2 points	straight line	$a_0 + a_1 x$
3 points	quadratic	$a_0 + a_1 x + a_2 x^2$
4 points	cubic	$a_0 + a_1 x + a_2 x^2 + a_3 x^3$
$n+1$ points	n-th order	$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$

Polynomial Interpolation

Fit polynomial through part of/entire dataset:

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

$a_0, a_1, \ldots, a_n$ are the unknowns

We have $n+1$ unknowns

We need $n+1$ equations

We need $n+1$ points

Polynomial Interpolation

For $n+1$ data points,
we can fit a polynomial of order n,
at most

Remember: An interpolating polynomial, by definition, MUST go through each and every point of the data used to build the polynomial.

Polynomial Interpolation

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

Given n+1 data points, we can fit an n^{th}-degree polynomial.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^n \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{pmatrix}$$

Given: (x_i, y_i), solve for a_i

Two steps:

1. Obtain polynomial coefficients by solving the system of linear equations
2. Evaluate the polynomial at the desired location(s) (x_i)

Pythonification

```
from numpy import polyfit, polyval
p  = polyfit(xi,yi,deg)
ye = polyval(p,xe)
```

- `p` - vector of coefficients of the polynomial, $a_n, a_{n-1}, \ldots, a_0$
- `xi` - vector of independent variable entries (known input values)
- `yi` - vector of dependent variable entries (known input values)
- `deg` - is the degree of the polynomial. Degree n Requires $n+1$ points (input)
- `polyfit(xi,yi,deg)` - returns the coefficients of the polynomial fit
- `ye = polyval(p,xe)`: evaluates polynomial at point(s) `xe`
- For $n+1$ points, if you choose a degree less than n, then regression will be performed (more later)

Pythonification

```
from numpy import polyfit, polyval
p  = polyfit(xi,yi,deg)
ye = polyval(p,xe)
```

- `p` - vector of coefficients of the polynomial, $a_n, a_{n-1}, \ldots, a_0$
- `xi` - vector of independent variable entries (known input values)
- `yi` - vector of dependent variable entries (known input values)
- `deg` - is the degree of the polynomial. Degree n Requires $n+1$ points (input)
- `polyfit(xi,yi,deg)` - returns the coefficients of the polynomial fit
- `ye = polyval(p,xe)`: evaluates polynomial at point(s) `xe`
- For $n+1$ points, if you choose a degree less than n, then regression will be performed (more later)

Coding Activity:

Perform polynomial interpolation on the Density-vs-Temperature dataset in the notebook **interp-activity-1-with-gaps.ipynb** from the Interpolation module on Canvas.

Best Practices for Polynomial Interpolants

- Polynomial Interpolants can be oscillatory, especially high order ones
- Apply a low-order polynomial interpolant (2nd or 3rd locally)
- Unfortunately, it is not straightforward in Python to apply local `polyfit`
- Instead, resort to splines (discussed next)

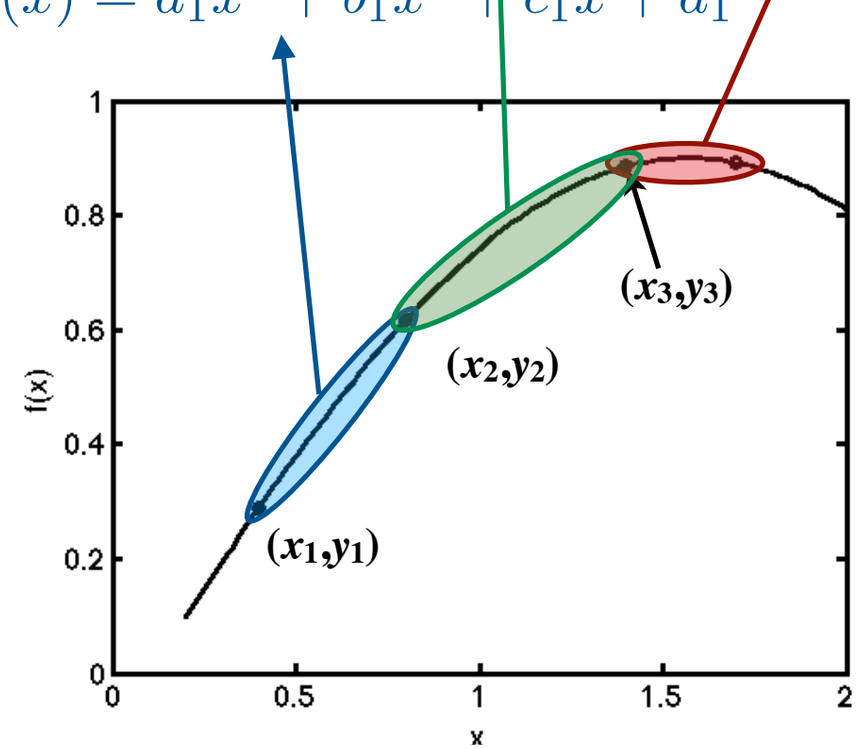
Cubic Spline Interpolation

Concept: use cubic polynomial and "hook" them together over a wide range of data...

$$f_3(x) = a_3x^3 + b_3x^2 + c_3x + d_3$$

$$f_2(x) = a_2x^3 + b_2x^2 + c_2x + d_2$$

$$f_1(x) = a_1x^3 + b_1x^2 + c_1x + d_1$$

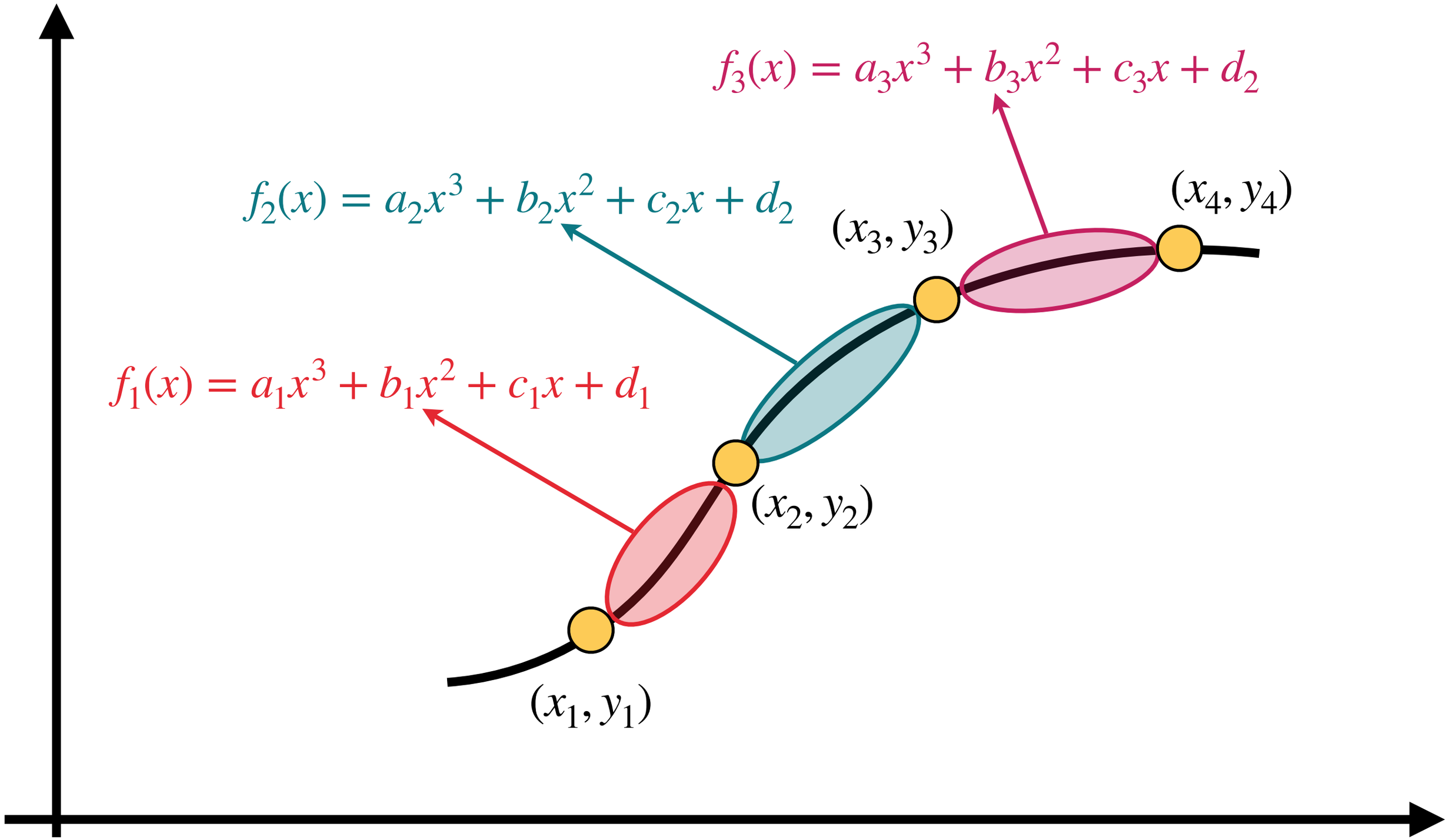


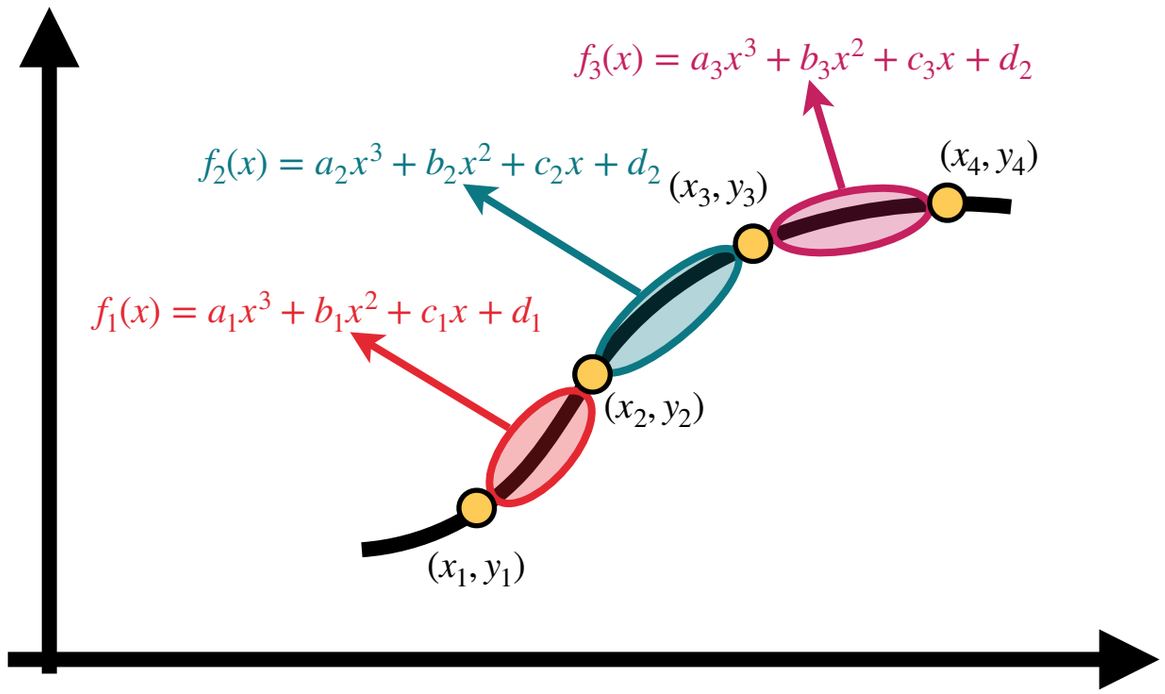
For n+1 points, we form n splines.

We must specify 4 variables per spline ⇒ we need **4*n* equations**.

1. At each data point, the values of adjacent splines must be the same. This applies to all interior points (where two functions meet) ⇒ 2(n-1) constraints.
2. At each point, the *first* derivatives of adjacent splines must be equal (applies to all interior points) ⇒ (n-1) constraints.
3. At each point, the *second* derivative of adjacent splines must be equal (applies to all interior points) ⇒ (n-1) constraints.
4. The first and last splines must pass through the first and last points, respectively ⇒ 2 constraints.
5. The curvature (d^2f / dx^2) must be specified at the end points ⇒ 2 constraints.
   - $d^2f / dx^2 = 0$ ⇒ "natural spline"

4*n* constraints





$$f_1(x_1) = y_1$$
$$f_1(x_2) = y_2$$
$$f_2(x_2) = y_2$$
$$f_2(x_3) = y_3$$
$$f_3(x_3) = y_3$$
$$f_3(x_4) = y_4$$

$$f_1'(x_2) = f_2'(x_2)$$

$$f_2'(x_3) = f_3'(x_3)$$

$$f_1''(x_2) = f_2''(x_2)$$

$$f_2''(x_3) = f_3''(x_3)$$

- natural spline

$$f_1''(x_1) = 0$$
$$f_3''(x_4) = 0$$

Cubic Spline Interpolation

Advantages:

- Provides a "smooth" interpolant.
- Usually more accurate than linear interpolation.
- Doesn't usually get "wiggly" like higher-order polynomial interpolation can.

Disadvantages:

- Requires a bit more work than linear interpolation to implement.

Cubic Spline Interpolation

Python Implementation:

```
from scipy.interpolate import CubicSpline
cs = CubicSpline(xi,yi)
ye = cs(xe)
```

- `xi` - independent variable entries (vector)
- `yi` - dependent variable entries (vector)
- `xe` - value(s) where you want to interpolate
- `ye` - interpolated value(s) at `xe`

Cubic Spline Interpolation

Python Implementation:

```
from scipy.interpolate import CubicSpline
cs = CubicSpline(xi,yi)
ye = cs(xe)
```

- `xi` - independent variable entries (vector)
- `yi` - dependent variable entries (vector)
- `xe` - value(s) where you want to interpolate
- `ye` - interpolated value(s) at `xe`

Coding Activity:
Perform cubic-spline interpolation on the Density-vs-Temperature dataset in the notebook **interp-activity-1-with-gaps.ipynb** from the Interpolation module on Canvas.

2-D Linear Interpolation

Example: Specific volume (m^3/kg) of water at various temperatures and pressures.

p (bar) \ T (C)	75.0	100.0	150.0	200.0	250.0	300.0	350.0	400.0	450.0	500.0
0.1	16.000	17.200	19.500	21.800	24.200	26.500	28.700	30.100	33.300	35.700
0.5	**0.001**	3.410	3.890	4.350	4.830	5.290	5.750	6.210	6.670	7.140
1.0	**0.001**	1.690	1.940	2.170	2.400	2.640	2.870	3.110	3.330	3.570
5.0	**0.001**	**0.001**	**0.001**	0.425	0.474	0.522	0.571	0.617	0.664	0.711
10.0	**0.001**	**0.001**	**0.001**	0.206	0.233	0.258	0.282	0.307	0.330	0.353

2-D Linear Interpolation

Example: Specific volume (m^3/kg) of water at various temperatures and pressures.

175

	T (C)									
p (bar)	75.0	100.0	150.0	200.0	250.0	300.0	350.0	400.0	450.0	500.0
0.1	16.000	17.200	19.500	21.800	24.200	26.500	28.700	30.100	33.300	35.700
0.5	**0.001**	3.410	3.890	4.350	4.830	5.290	5.750	6.210	6.670	7.140
1.0	**0.001**	1.690	1.940	2.170	2.400	2.640	2.870	3.110	3.330	3.570
5.0	**0.001**	**0.001**	**0.001**	0.425	0.474	0.522	0.571	0.617	0.664	0.711
10.0	**0.001**	**0.001**	**0.001**	0.206	0.233	0.258	0.282	0.307	0.330	0.353

0.75

2-D Linear Interpolation

Example: Specific volume (m^3/kg) of water at various temperatures and pressures.

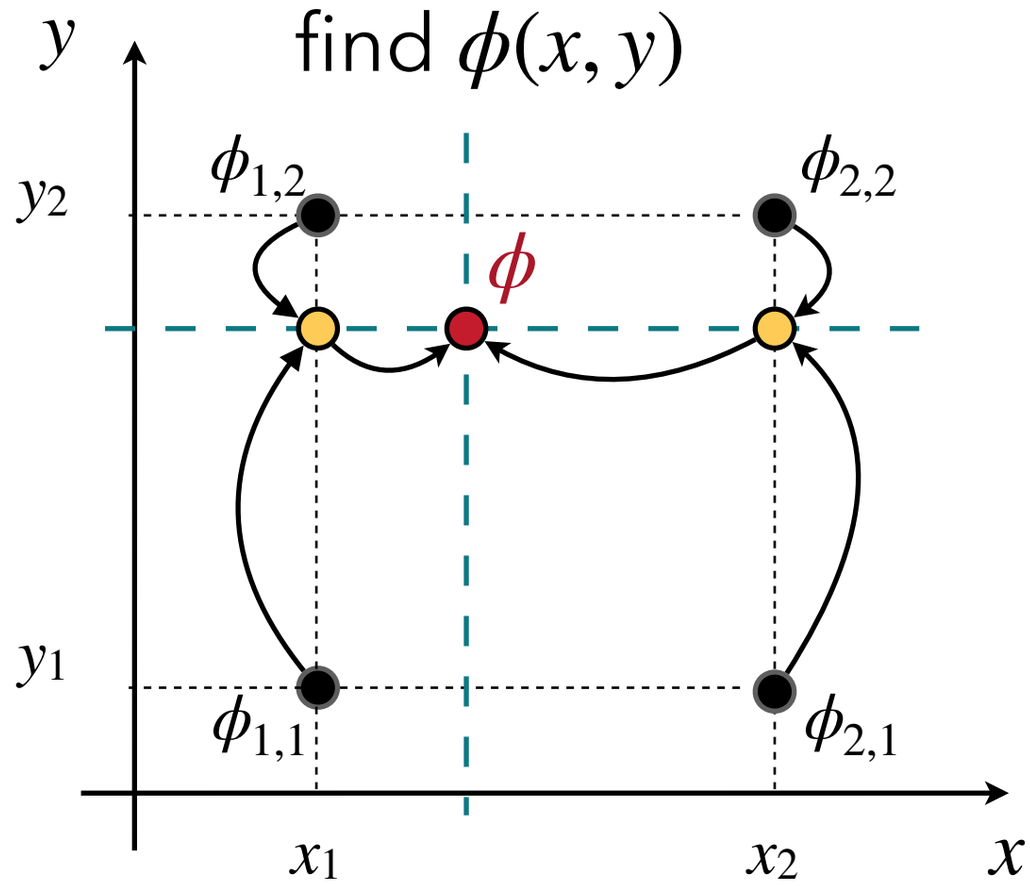
175

	T (C)									
p (bar)	75.0	100.0	150.0	200.0	250.0	300.0	350.0	400.0	450.0	500.0
0.1	16.000	17.200	19.500	21.800	24.200	26.500	28.700	30.100	33.300	35.700
0.5	**0.001**	3.410	3.890	4.350	4.830	5.290	5.750	6.210	6.670	7.140
1.0 (0.75)	**0.001**	1.690	1.940	2.170	2.400	2.640	2.870	3.110	3.330	3.570
5.0	**0.001**	**0.001**	**0.001**	0.425	0.474	0.522	0.571	0.617	0.664	0.711
10.0	**0.001**	**0.001**	**0.001**	0.206	0.233	0.258	0.282	0.307	0.330	0.353

2-D Linear Interpolation

Bilinear Interpolation
If you have "structured" (tabular) data, you can do two 1-D interpolations:
1. Interpolate in one direction
2. Interpolate in second direction.

Use this for simple homework assignments, in-class exams, etc.



Interpolate in y first

175

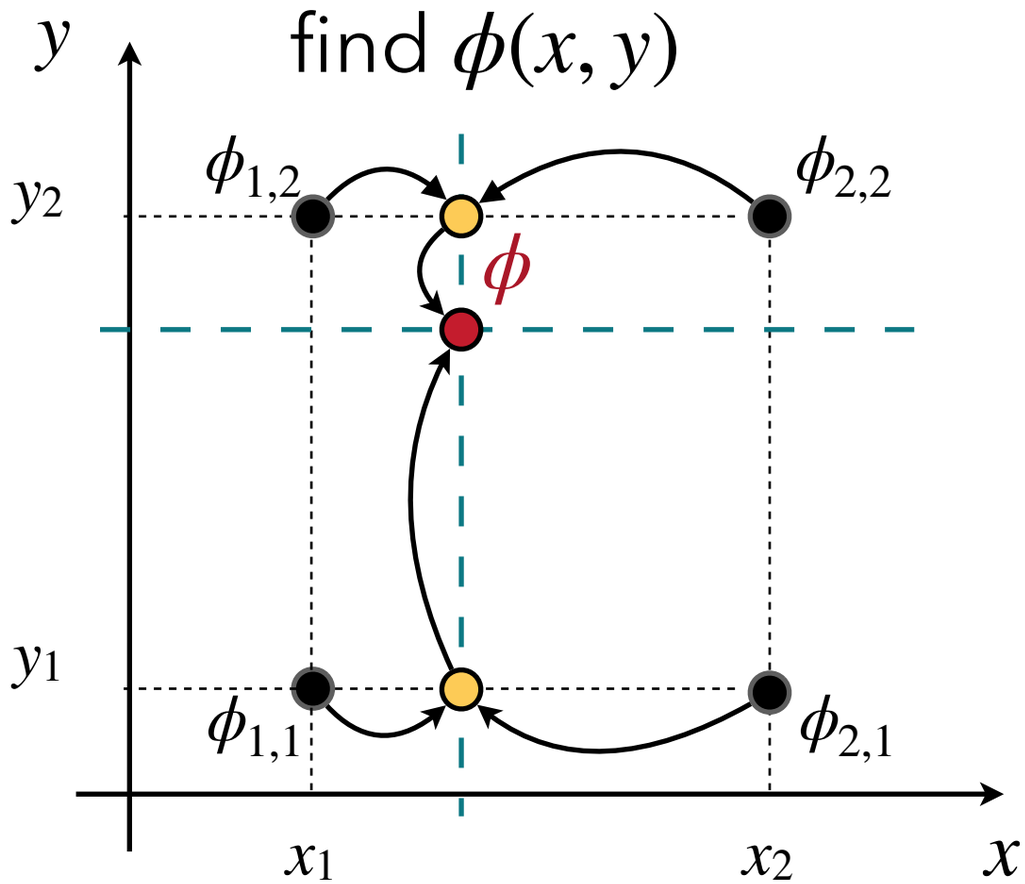
p (bar)	75.0	100.0	150.0	200.0	250.0
0.1	16.000	17.200	19.500	21.800	24.200
0.5	**0.001**	3.410	3.890	4.350	4.830
1.0	**0.001**	1.690	1.940	2.170	2.400
5.0	**0.001**	**0.001**	**0.001**	0.425	0.474
10.0	**0.001**	**0.001**	**0.001**	0.206	0.233

0.75

2-D Linear Interpolation

Bilinear Interpolation
If you have "structured" (tabular) data:
1. Interpolate in one direction (two 1-D interpolations)
2. Interpolate in second direction.

Use this for simple homework assignments, in-class exams, etc.



Interpolate in x first

Either way, you get this:

$$\phi(x,y) \approx \phi_{1,1}\frac{(x_2-x)(y_2-y)}{(x_2-x_1)(y_2-y_1)} + \phi_{2,1}\frac{(x-x_1)(y_2-y)}{(x_2-x_1)(y_2-y_1)} + \phi_{1,2}\frac{(x_2-x)(y-y_1)}{(x_2-x_1)(y_2-y_1)} + \phi_{2,2}\frac{(x-x_1)(y-y_1)}{(x_2-x_1)(y_2-y_1)}$$

p	250.0
0	24.200
0	4.830
1	2.400
5	0.474
10	0.233

Pythonification

```
from scipy.interpolate import interp2d
f2d = interp2d(xi,yi,zi,method)
ze = f2d(xe,ye)
```

- **xi** - independent variable entries (vector)
- **yi** - independent variable entries (vector)
- **zi** - dependent variable entires (2d vector)
- **method** = ‘linear’, ‘cubic’, ‘quintic’
- **xe, ye** - value(s) where you want to interpolate
- **ze** - interpolated value(s) at **xe** and **ye**

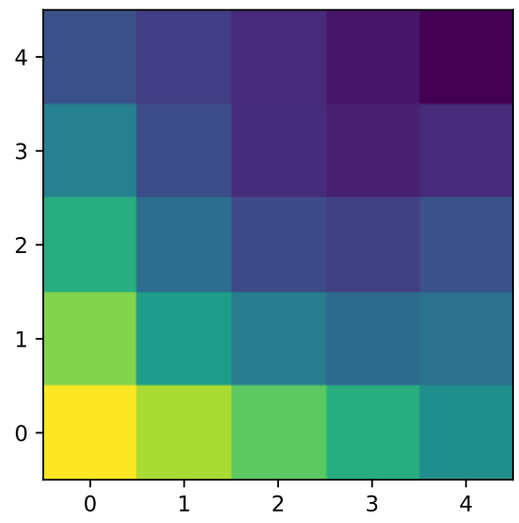
2-D Interpolation Example

Temperatures are measured at various points on a heated plate (Table P18.30). Estimate the temperature at **(a)** $x = 4$, $y = 3.2$, and **(b)** $x = 4.3$, $y = 2.7$.

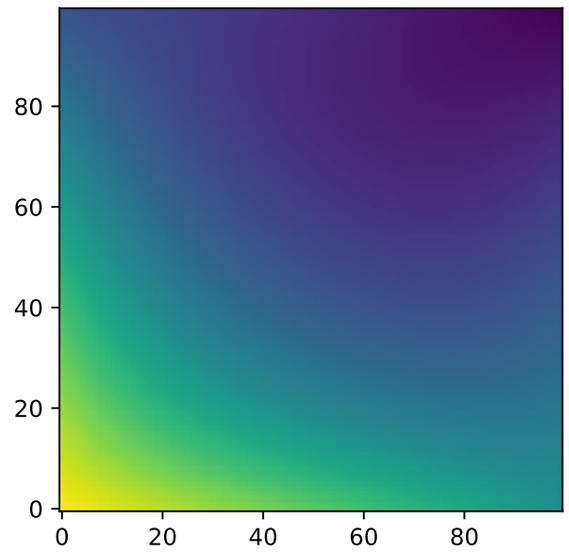
TABLE P18.30 Temperature (°C) at various points on a square heated plate.

	$x = 0$	$x = 2$	$x = 4$	$x = 6$	$x = 8$
$y = 0$	100.00	90.00	80.00	70.00	60.00
$y = 2$	85.00	64.49	53.50	48.15	50.00
$y = 4$	70.00	48.90	38.43	35.03	40.00
$y = 6$	55.00	38.78	30.39	27.07	30.00
$y = 8$	40.00	35.00	30.00	25.00	20.00

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
x = [0.0,2.0,4.0,6.0,8.0]
y = [0.0,2.0,4.0,6.0,8.0]
T = [[100,90,80,70,60],
     [85,64.49,53.50,48.15,50.0],
    [70,48.9,38.43,35.03,40.0],
    [55.00,38.78,30.39,27.07,30.0],
    [40.00,35,30,25,20]]
f = interpolate.interp2d(x, y, T, kind='cubic')
plt.imshow(f(x,y),origin='lower')
plt.show()
xnew = np.linspace(0,8,100)
ynew = np.linspace(0,8,100)
fnew = f(xnew,ynew)
plt.imshow(f(xnew,ynew),origin='lower')
```



Original data



Interpolated data