1-d Interpolation (Approximation of a Function)

Phys 281 – Class 6

Grant Wilson

Answers to Exercises

```
from matplotlib import pyplot as plt
import numpy as np

#E5.1 - Write a recursive Python function to calculate
#the factorial of any number.
def fact(n):
    if n <= 1:
        return 1
    else:
        return n*fact(n-1)
```

```
#bisection algorithm
#define the function to find the root of
def y(t):
   return t - np.sin(t) - 0.5

#the initial range to search over and
#the convergence criteria
epsilon = 1.e-10
thi = 2.
tlow = 0.
ylow = y(tlow)
yhi = y(thi)

#we don't know when to stop so use
#a while loop
while (abs(thi-tlow) > 2*epsilon):
   tmid = (thi+tlow)/2.
   ymid = y(tmid)
   if(ymid * ylow > 0):
      tlow = tmid
   else:
      thi = tmid

bestT = (thi+tlow)/2.
print "Bisector solution is: ", bestT
```

Bisector Algorithm:
34 iterations to converge
Bisector solution is: 1.49730038905

```
#Newton Raphson algorithm

#define the function to find the root of
def y(t):
   return t - np.sin(t) - 0.5

#define the derivative of the function
def yp(t):
   return 1 - np.cos(t)

#here's our initial guess
tg = 1.2
tnew = 1.3

#the convergence criteria
epsilon = 1.e-10

#and the loop
while(abs(tnew - tg) > epsilon):
   tg = tnew
   tnew = tg - y(tg)/yp(tg)

print "Newton Raphson solution is: ", (tg+tnew)/2.
```

Newton Raphson:
5 iterations to converge
Newton Raphson solution is: 1.4973003891

$$t_{g,\,new} = tg - \frac{y(tg)}{y'(tg)}$$

Using scipy.optimize.brentq()

```
from scipy import optimize
import numpy as np

def f(x,*args):
    return x - np.sin(x) - 0.5

print "Brentq solution is: ", opt.brentq(f,0,2,epsilon)
```

1-d Interpolation (Approximation of a Function)

Phys 281 – Class 6

Grant Wilson

Calculations and Tables

- Calculations result in Tables of numbers

Index	t	x
0	0.2	32.5
1	0.4	28.2
2	0.6	11.9
3	0.8	22.1
4	1.0	48.0

- Interpolation is used to find values between the calculated points.

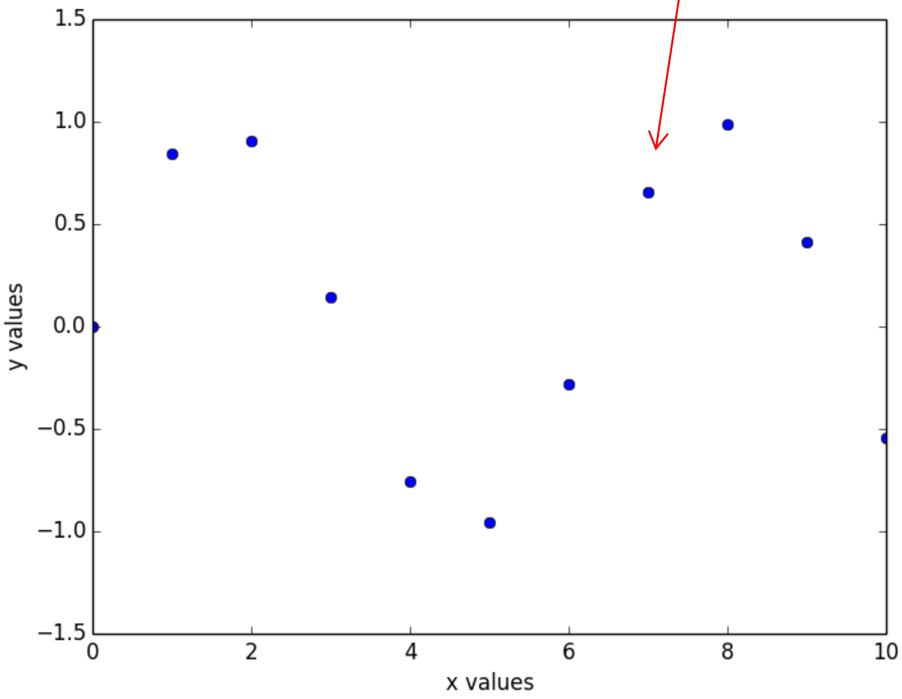
- Throughout this class I will refer to the data points to be interpolated as "support points."

Interpolation Methods

- Polynomial Interpolation
  - Nearest Neighbor
  - Linear
  - Quadratic

- Spline Interpolation

support points



How to think about interpolation

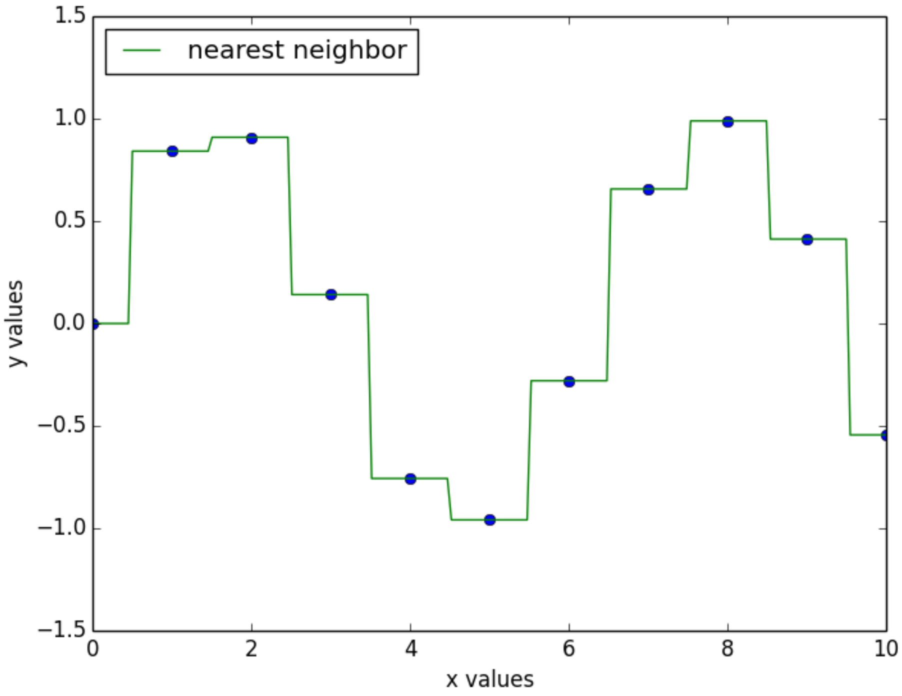
- Interpolation is NOT fitting a function to the data. This is different and we will cover this later in the semester.
- The key rule of an interpolating function:
  - The function must go through all the support points.
- Polynomial functions can be interpolated exactly by a polynomial of the same order.
- Polynomial interpolation is a “local” form of interpolation.
- Spline interpolation is a “global” form of interpolation.

Interpolation Methods

- Polynomial Interpolation
  - Nearest Neighbor

$$y(x) = y_i$$

where y_i is the value in the table corresponding to the x_i closest to x.

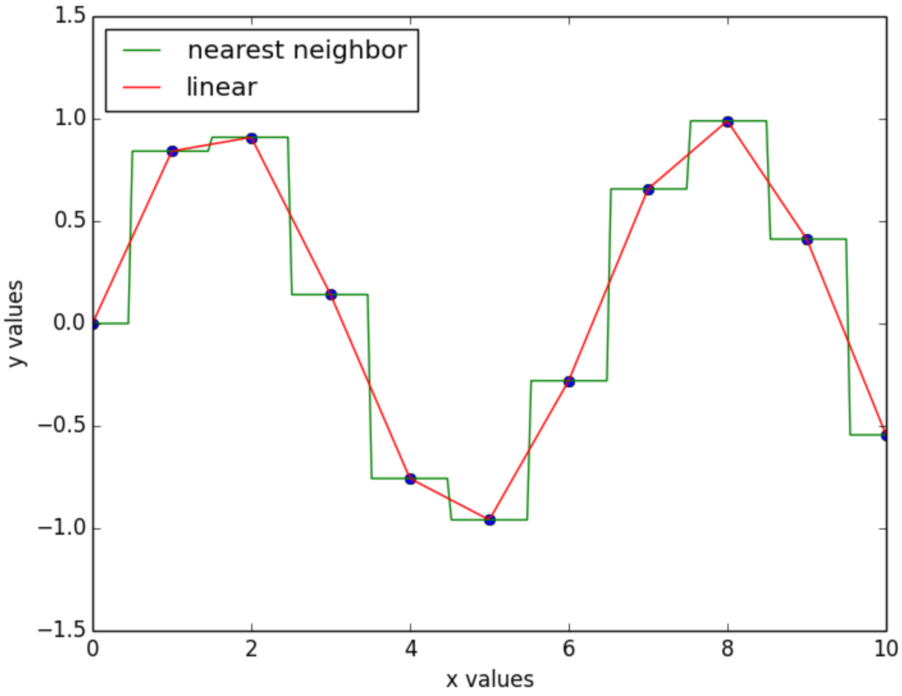


Interpolation Methods

- Polynomial Interpolation
  - Nearest Neighbor
  - Linear

$$y(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$

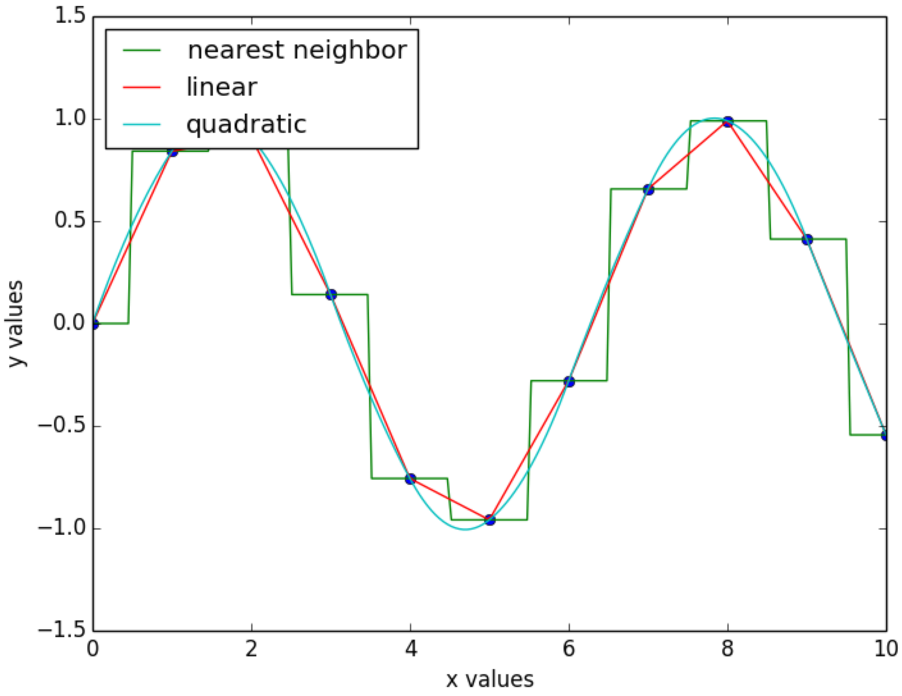
for x between x_i and x_{i+1}



Interpolation Methods

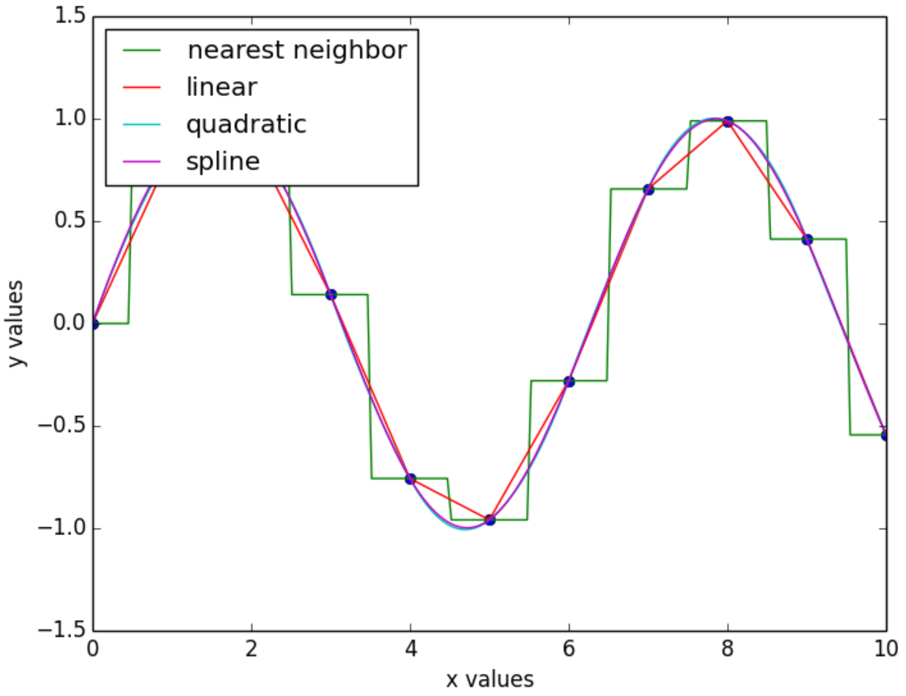
- Polynomial Interpolation
  - Nearest Neighbor
  - Linear
  - Quadratic

$$y(x) = \sum_{i=0}^{2}\left(\sum_{j=0, j\neq i}^{2} \frac{x - x_j}{x_i - x_j}\right) y_i$$



Interpolation Methods

- Polynomial Interpolation
  - Nearest Neighbor
  - Linear
  - Quadratic

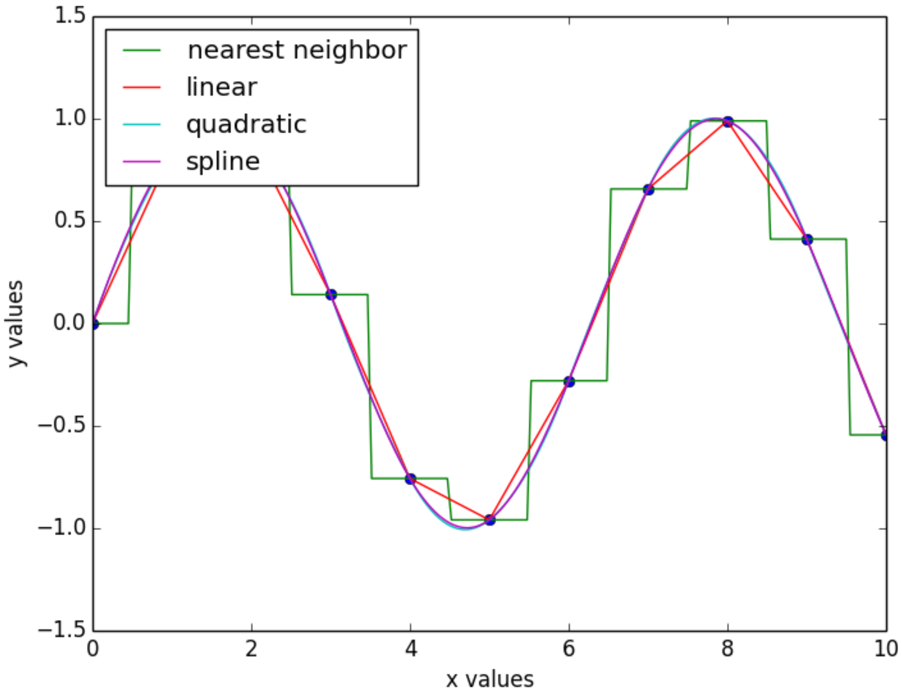


- Spline Interpolation

Interpolation Methods

- Polynomial Interpolation
  - Nearest Neighbor
  - Linear
  - Quadratic

- Spline Interpolation
  - cubic function
  - constraint to match first and second derivatives at each interior support point.



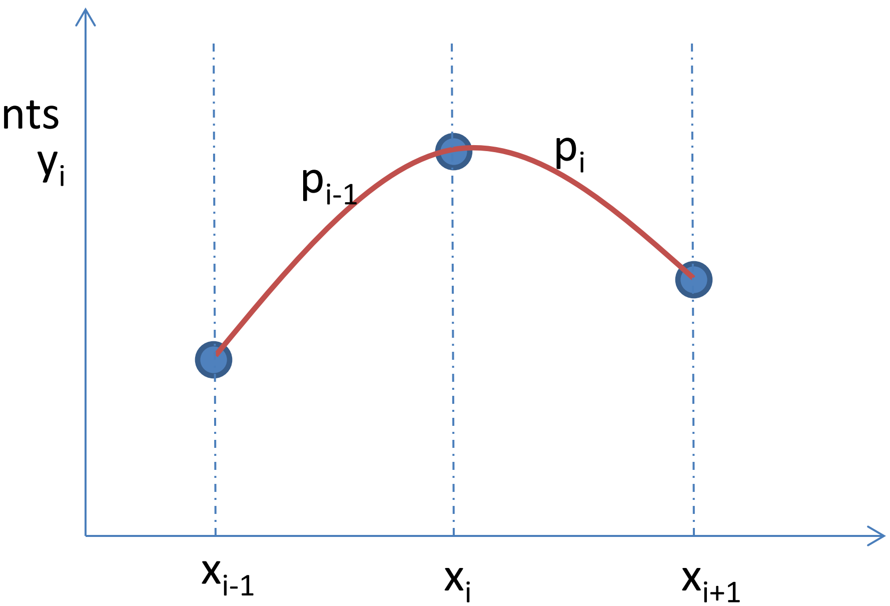
Constructing the Spline

- Spline interpolation comes from elasticity theory.
- The interpolation is constructed by placing the same constraints at all interior support points.

$p_{i-1}(x_i) = p_i(x_i) = y_i$ - continuity at support points

$p'_{i-1}(x_i) = p'_i(x_i)$ - continuity of 1st deriv.

$p''_{i-1}(x_i) = p''_i(x_i)$ - continuity of 2nd deriv.

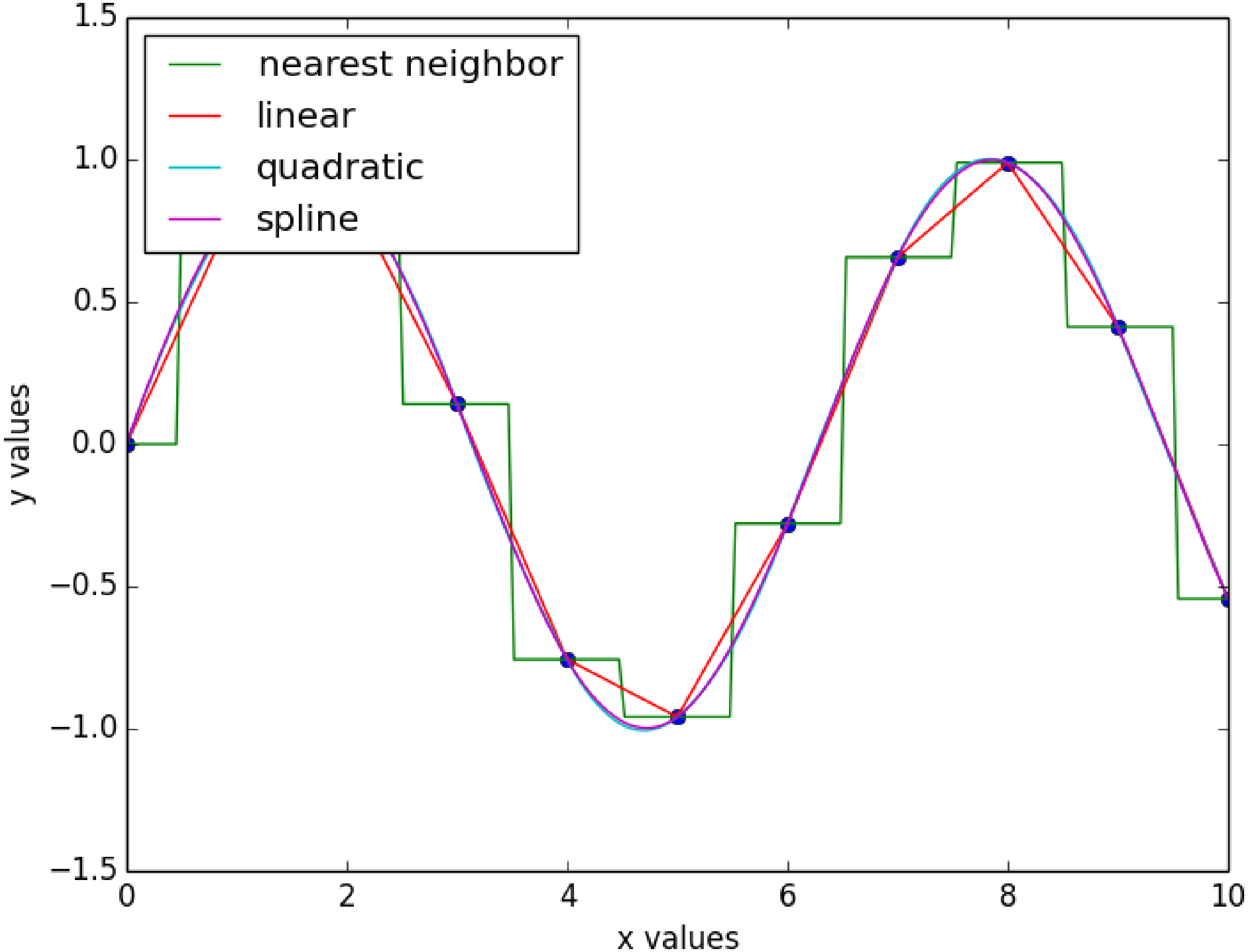


Cubic Spline Interpolation

- Demand that each spline segment is cubic in form:
  - $pi(x) = a + bx + cx^2 + dx^3$
- This leaves us with 4 unknowns {a,b,c,d} for each of the N-1 segments so we have a total of 4(N-1) unknowns.
- The constraints on the interior support points give us a total of 4(N-2) equations.
- Demanding that the spline goes through the first and last points gives us two more. We need to pick the final 2 ourselves.
- The "Natural Spline" conditions are:

$$p''_0(x_0) = 0$$
$$p''_{N-1}(x_{N-1}) = 0$$

Interpolation Methods



Interpolating in Python

- Use the scypy.interpolation library

  from scipy import interpolate

1. Assemble your support points in two arrays
2. Create an array of “x values” where you want to build the interpolation.
3. choose your interpolation method
4. build an interpolation function using interp1d()
5. call the function with your new x-values as the input to generate your new y-values.

Example – linear Interpolation

```
from matplotlib import pyplot as plt
from scipy import interpolate
import numpy as np
plt.ion()

#need some fake data as support points
x_support = np.arange(11)
y_support = np.sin(x_support)

#linearly interpolate (x_support,y_support) in 200
#points across the interval [0,10]
x = np.linspace(0,10,200,endpoint=True)

#build the interpolation function
linear = interpolate.interp1d(x_support,y_support,'linear')

#do the interpolation
y = linear(x)

#plot the original support points and the interpolation
```

Exercise #1

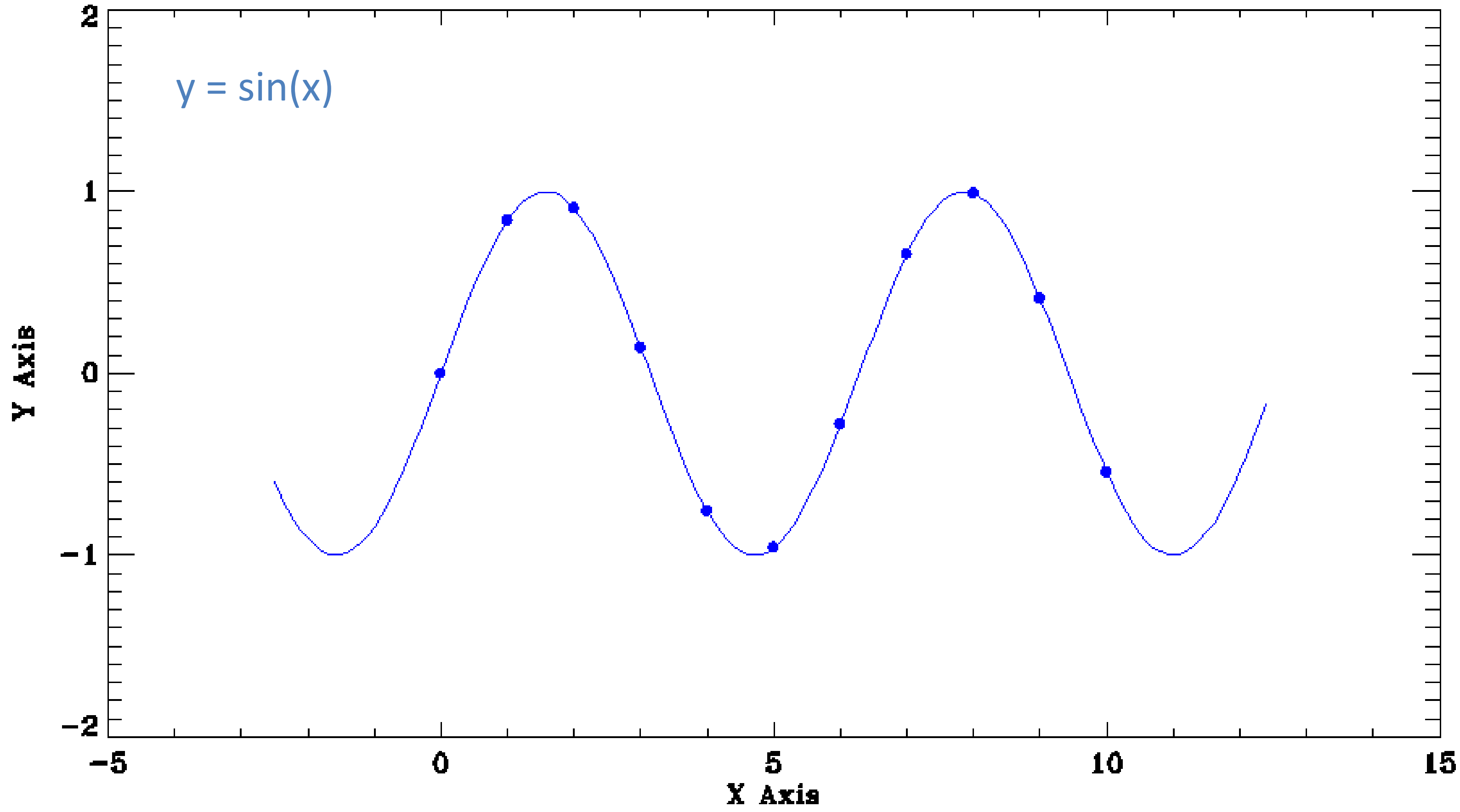
1. download population.csv from the Moodle page

2. make a plot of year vs U.S. population

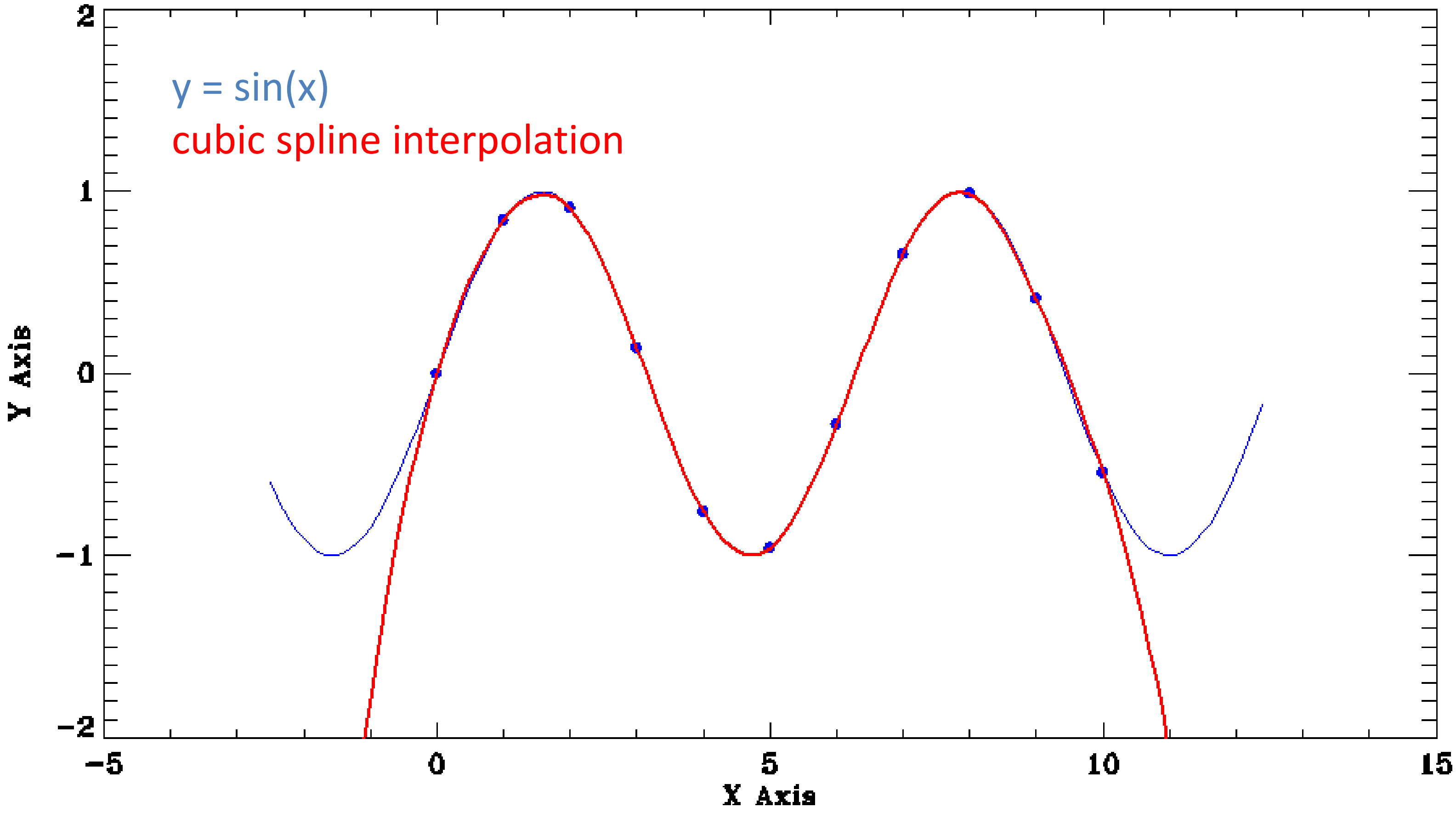
3. estimate the population of the U.S. on the following dates using all four forms of interpolation:
    1. Sept. 4, 1969
    2. Oct. 25, 2005

It’s so easy!
What could possibly go wrong?

- It’s true, interpolation is easy. There are a few things to be wary of:
  - discontinuities in the underlying function
  - poles in the underlying function
  - too few support points to capture the most important features of the data
- Above all, <u>avoid the temptation to extrapolate!</u>



y = sin(x)
2
1
0
−1
−2
Y Axis
−5
0
5
10
15
X Axis



y = sin(x)
cubic spline interpolation
2
1
0
−1
−2
Y Axis
−5
0
5
10
15
X Axis

Exercises

1. Read through the online documentation for scipy.interpolate.interp1d – this is your go-to routine for 1d interpolation.

2. Download the interpolate_me.npz file from the class Moodle page.

3. Build three interpolations for the (x1,y1) data set provided. Each interpolation should have 100 x-values spaced evenly between 1 and 9. Make plots showing:
    1. a linear interpolation,
    2. a quadratic interpolation,
    3. a cubic spline interpolation.

The underlying function used to create the data is

$$y(x) = \sin x / x$$

4. Make a plot showing the fractional error of the interpolation for each of the three cases above. The fractional error is:

$$(y(x)-y_{interp})/y(x)$$