Interpolation

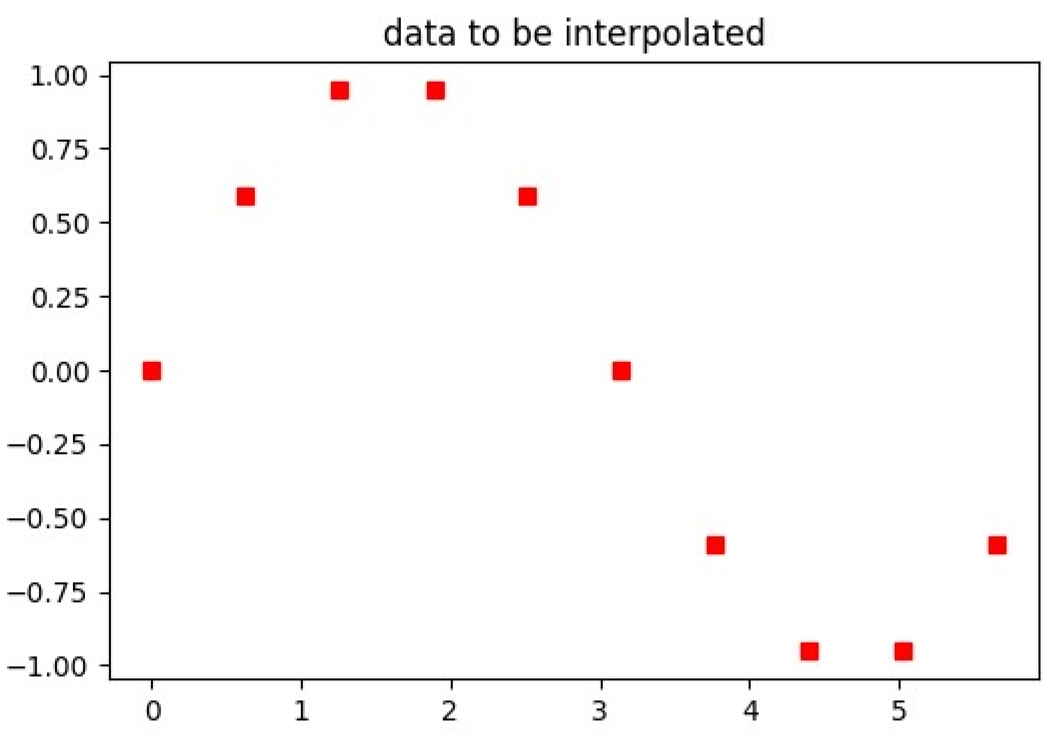
Interpolation is the process of using a set of data values for a *function* to determine the missing values of that function. Some common interpolation techniques are:

1. Nearest neighbour interpolation
2. Linear interpolation
3. Polynomial interpolation
4. Spline interpolation

Consider the following script and plot for the (x,y) data to interpolate.

data.py

```
import matplotlib.pyplot as plt

x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65]
y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

plt.figure(1, figsize=(6.0,4.0), dpi=100)
plt.plot( x, y, "rs" ) # red square
plt.title( 'data to be interpolated' )
plt.savefig( 'data.png' )
```

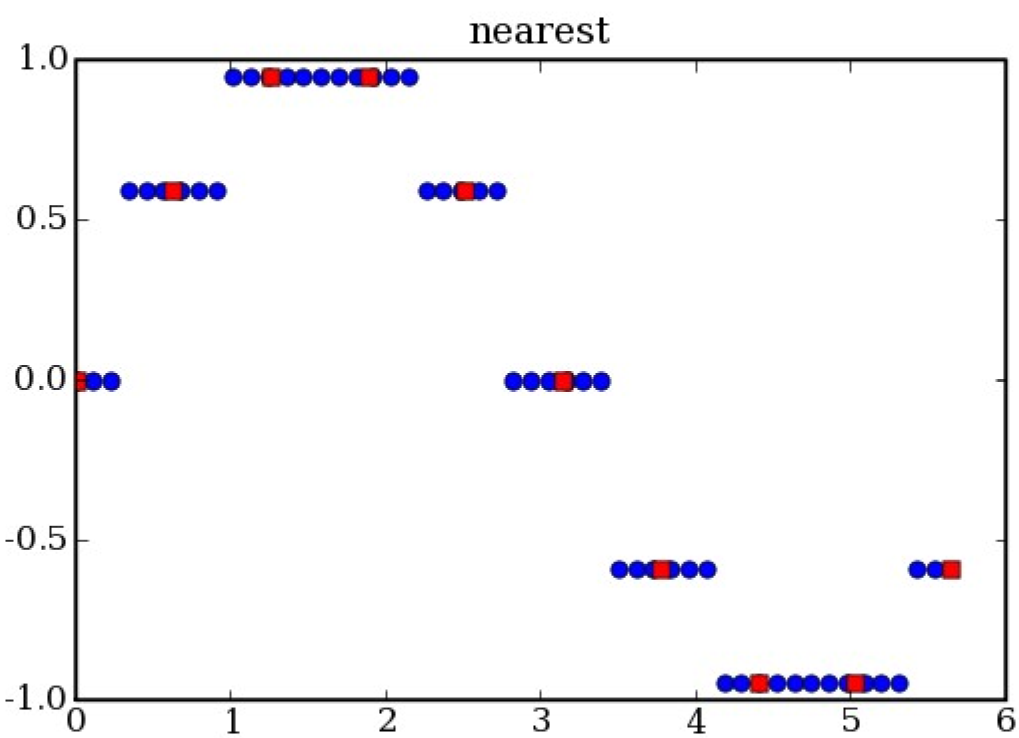


Nearest neighbour interpolation

In nearest neighbour interpolation, the closest data x value to the requested x value is used to select the matching y value for the requested value. A python function that uses the nearest neighbour interpolation technique is:

nearest.py

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

def nearest( v ) :
    index = 0
    diff = abs( nearest.x[index] - v )
    for i in range(1, len(nearest.x)) :
        d =  abs( nearest.x[i] - v )
        if d < diff :
            diff = d
            index = i
    return nearest.y[index]

nearest.x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
nearest.y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = [ nearest(v) for v in x ]
plt.plot( x, y, "bo" ) # blue circle

plt.plot( nearest.x, nearest.y, "rs" ) # red square
plt.title( 'nearest' )
plt.savefig( 'nearest.png' )
```



Nearest with numpy

```
>>> import numpy as np
>>> x = np.array([ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ])
>>> y = np.array([ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59])
>>> # closest to 1.0
>>> dx = np.fabs( x - 1.0 )
```

```
>>> dx
array([ 1.  ,  0.37,  0.26,  0.89,  1.51,  2.14,  2.77,  3.4 ,  4.03,  4.65])
>>> i = dx.argmin()
>>> i
2
>>> y[i]
0.94999999999999996
```

Bisection module (aside)

The bisection module finds the insertion point for a new element in a sorted list, so that the list remains sorted.

```
>>> import bisect
>>> x = [ 3, 7, 8, 10, 11 ]
>>> bisect.bisect_left(x, 1 ) # insert at start
0
>>> bisect.bisect_left(x, 4 )
1
>>> bisect.bisect_left(x, 8 )
2
>>> bisect.bisect_left(x, 9 )
3
>>> bisect.bisect_left(x, 11 )
4
>>> bisect.bisect_left(x, 12 )
5
```

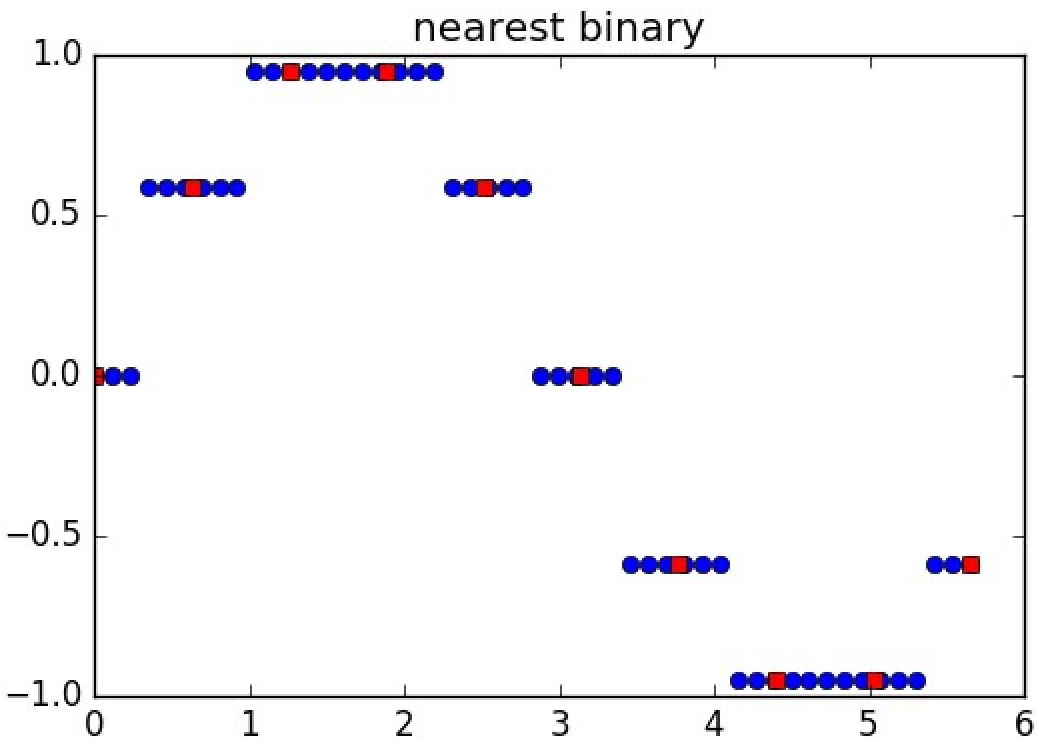
Nearest neighbour interpolation with bisection

Binary search is used to speed up the look up of the nearest x value.

nearest_bin.py

```
import matplotlib.pyplot as plt
import numpy as np
import bisect

def nearest_bin( v ) :
    # bisect_left returns the insertion point that maintains
    # sorted order
    index2 = bisect.bisect_left(nearest_bin.x, v )
    index1 = index2-1
    if index1 < 0 :
        return nearest_bin.y[0]
    if index2 >= len(nearest_bin.x) :
        return nearest_bin.y[ -1 ]
    d1 = abs( nearest_bin.x[index1] - v )
    d2 = abs( nearest_bin.x[index2] - v )
    if d1 < d2 :
        return nearest_bin.y[index1]
    else :
        return nearest_bin.y[index2]

nearest_bin.x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
nearest_bin.y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

plt.figure(1, figsize=(6.0,4.0), dpi=100)
```

```
x = np.linspace( 0.0, 5.65, 50)
y = [ nearest_bin(v) for v in x ]
plt.plot( x, y, "bo" )

plt.plot( nearest_bin.x, nearest_bin.y, "rs" )
plt.title( 'nearest binary' )
plt.savefig( 'nearest_bin.png' )
```

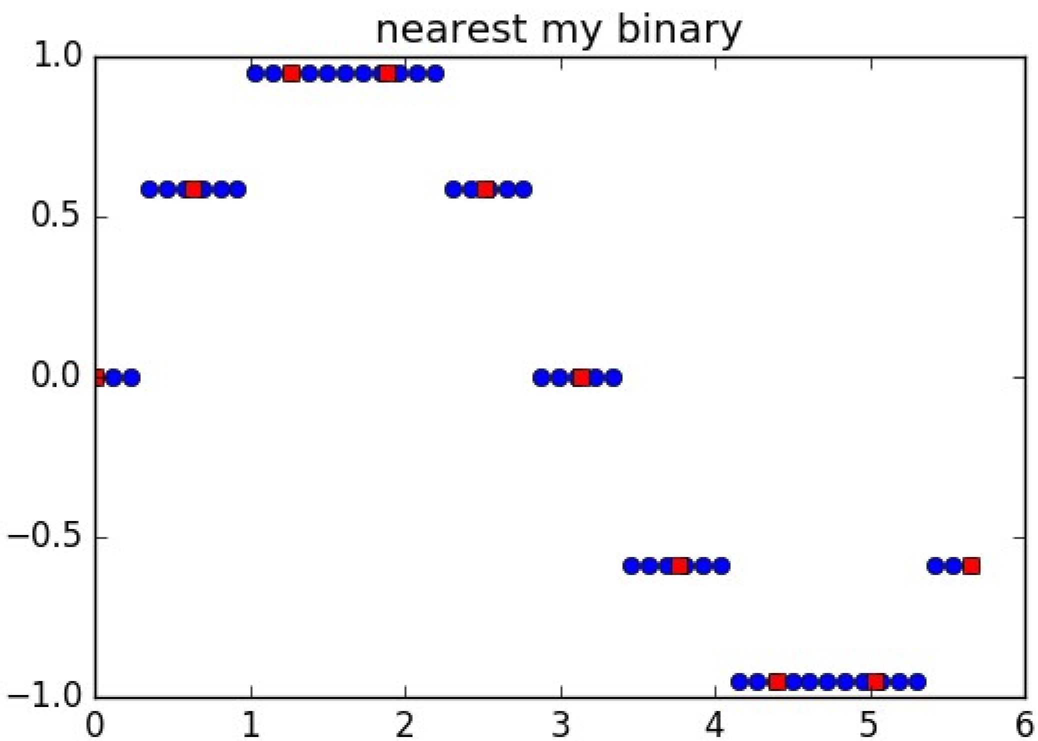


Nearest neighbour interpolation with my bsearch

nearest_mybin.py

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

def bin_search( vec, x, lo, hi ) :
    if lo > hi : return lo
    m = (lo+hi) // 2
    d = vec[m] - x
    if d == 0 :
        return m
    elif d < 0:
        return bin_search(vec, x, m+1, hi)
    else :
        return bin_search(vec, x, lo, m-1)

def bsearch( vec, x ) :
    return bin_search( vec, x, 0, len(vec)-1 )

def nearest_bin( v ) :
    index2 = bsearch(nearest_bin.x, v )
    index1 = index2-1
    if index1 < 0 :
        return nearest_bin.y[0]
```

```
    if index2 >= len(nearest_bin.x) :
        return nearest_bin.y[ -1 ]
    d1 = abs( nearest_bin.x[index1] - v )
    d2 = abs( nearest_bin.x[index2] - v )
    if d1 < d2 :
        return nearest_bin.y[index1]
    else :
        return nearest_bin.y[index2]

nearest_bin.x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
nearest_bin.y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = [ nearest_bin(v) for v in x ]
plt.plot( x, y, "bo" )

plt.plot( nearest_bin.x, nearest_bin.y, "rs" )
plt.title( 'nearest my binary' )
plt.savefig( 'nearest_mybin.png' )
```

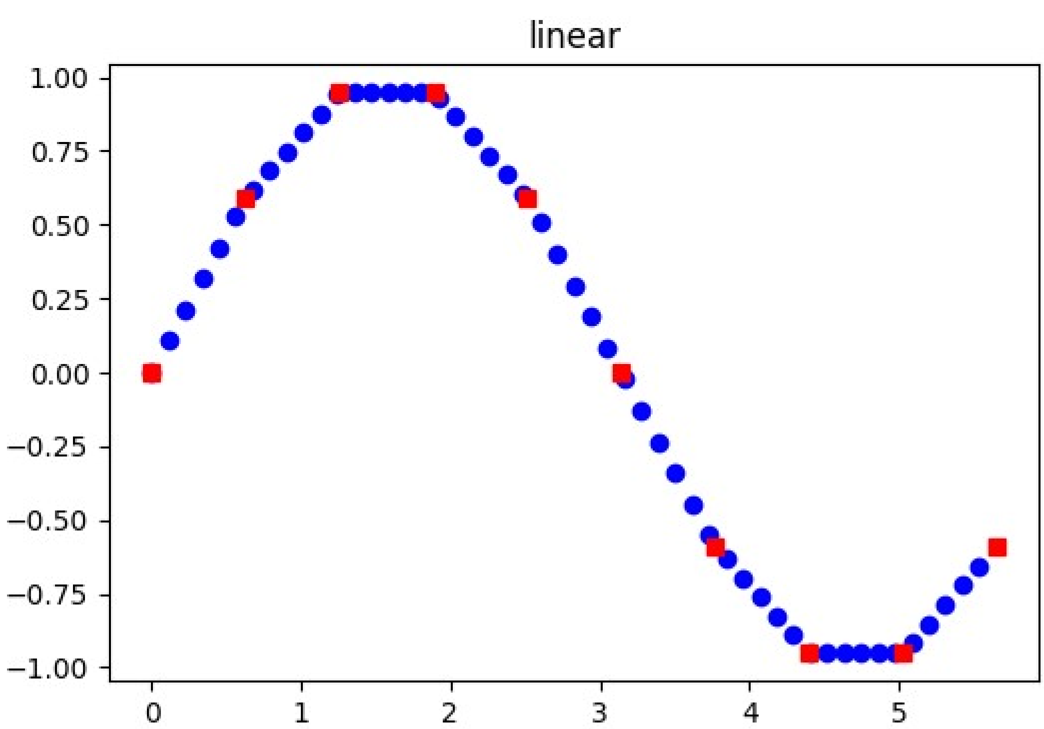


Linear interpolation

A straight line segment is drawn between two adjacent data points. The points along the line are used for interpolation. The intersection of a vertical line drawn through the x-axis and the line segment gives the function value. This value is computed with:

$$y = y_i + \frac{(x - x_i)(y_j - y_i)}{(x_j - x_i)}$$

```
import matplotlib.pyplot as plt
import numpy as np
import bisect

def linear( v ) :
    x = linear.x
    y = linear.y
    # bisect_left returns the insertion point that maintains
    # sorted order
    j = bisect.bisect_left(x, v)
    i = j-1
    if i < 0 :
        return y[0]
    if j >= len(x) :
        return y[ -1 ]
    return y[i] + (v-x[i])*(y[j]-y[i])/(x[j]-x[i])

linear.x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
linear.y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = [ linear(v) for v in x ]
plt.plot( x, y, "bo" )

plt.plot( linear.x, linear.y, "rs" )
plt.title( 'linear' )
plt.savefig( 'linear.png' )
```

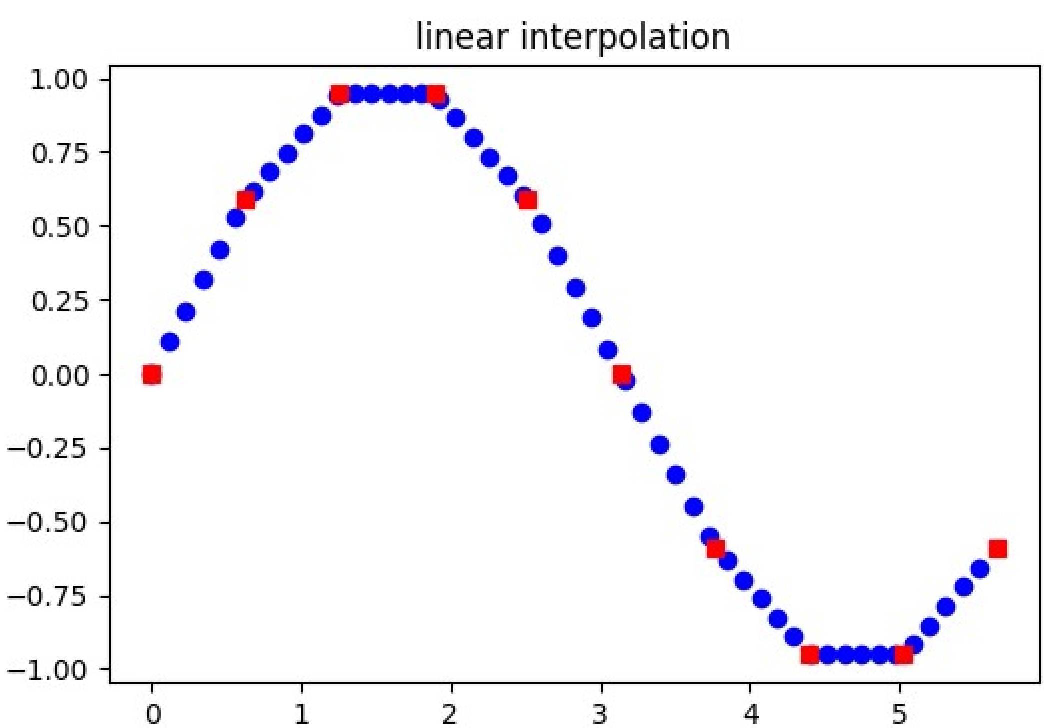


Making a linear interpolation

python 's ability to create function is used to create a function that performs the linear interpolation.

linear_interpolate.py

```
import matplotlib.pyplot as plt
import numpy as np
import bisect

def linear_interpolate(x, y) :
    "return a fn that does linear interpolation of data"
    x = x[:]
    y = y[:]
    def fn( v ) :
        j = bisect.bisect_left(x, v)
        i = j-1
        if i < 0 :
            return y[0]
        if j >= len(x) :
            return y[ -1 ]
        return y[i] + (v-x[i])*(y[j]-y[i])/(x[j]-x[i])
    return fn

data_x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
data_y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

lin = linear_interpolate( data_x, data_y)

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = [ lin(v) for v in x ]
plt.plot( x, y, "bo" )

plt.plot( data_x, data_y, "rs" )
plt.title( 'linear interpolation' )
plt.savefig( 'linear_interpolate.png' )
```

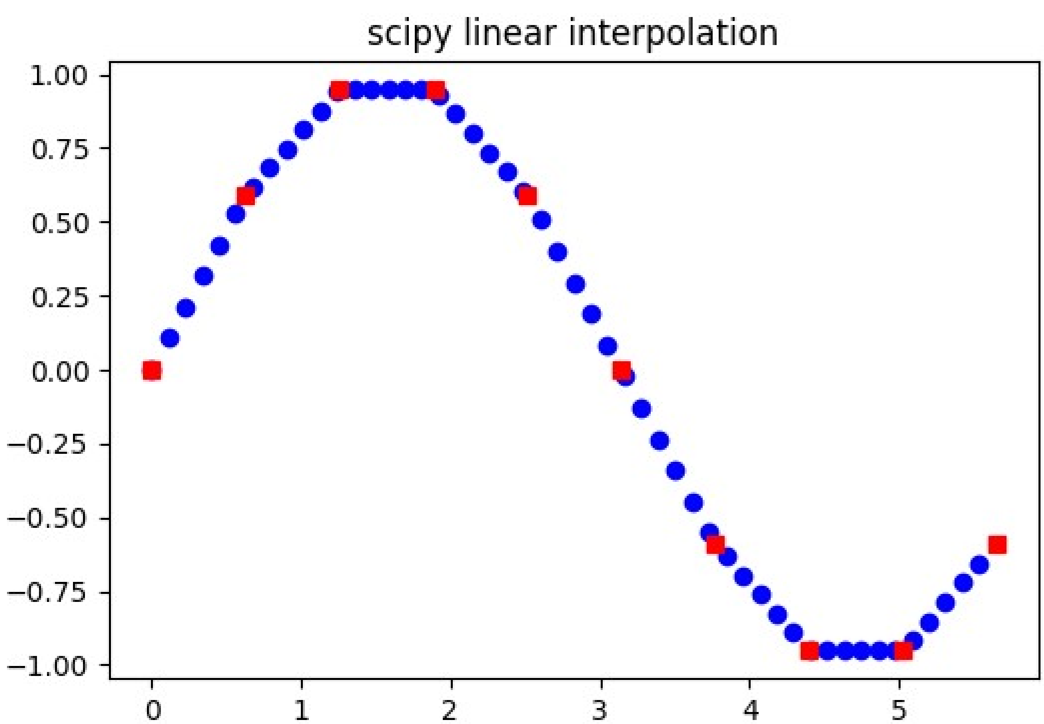


Using scipy for linear interpolation

The `scipy` module is used to create the linear interpolation function.

scipy_linear.py

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate

data_x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
data_y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

lin = scipy.interpolate.interp1d( data_x, data_y)

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = [ lin(v) for v in x ]
plt.plot( x, y, "bo" )

plt.plot( data_x, data_y, "rs" )
plt.title( 'scipy linear interpolation' )
plt.savefig( 'scipy_linear.png' )
```



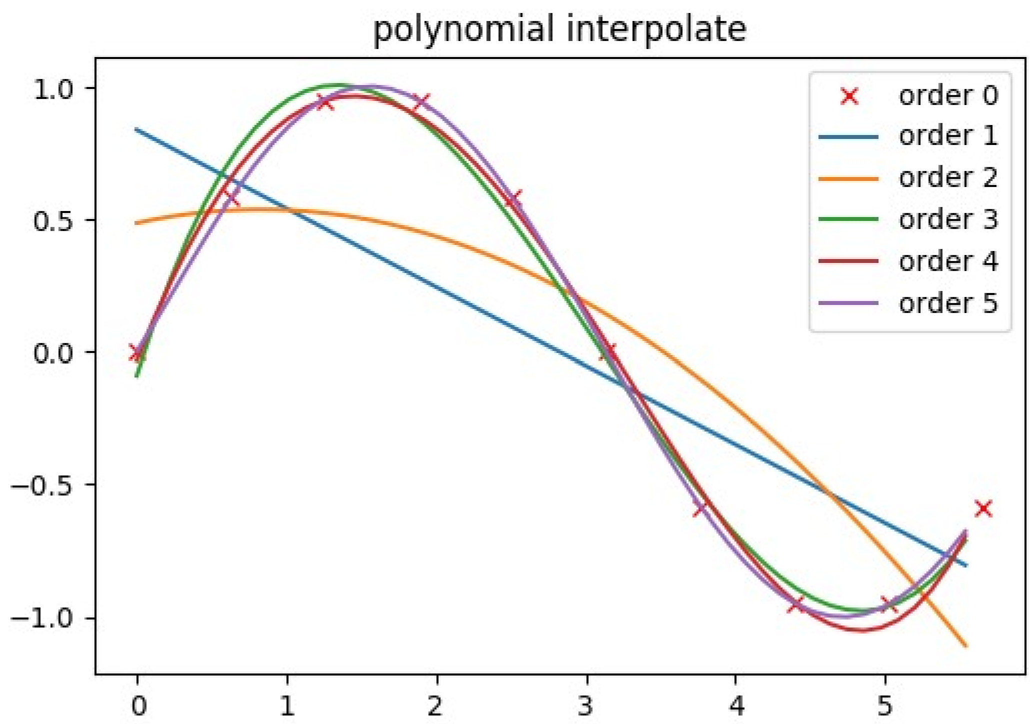
Polynomial interpolation

Fitting a polynomical is used to create an interpolation function. Consider:

poly_interpolate.py

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

```
x = np.array([ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ])
y = np.array([ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59])

px = np.linspace( 0.0, 5.65, 50)
py = []
for order in range(1,6) :
    p = np.polyfit(x,y, order)
    py.append( np.polyval(p,px) )

plt.figure(1, figsize=(6.0,4.0), dpi=100)
plt.plot( x, y, "rx")
for ye in py :
    plt.plot(px,ye)
plt.legend( [ "order %d" % i for i in range(6) ] )
plt.title( 'polynomial interpolate')

plt.savefig( 'poly_interpolate.png' )
```



High degree polynomials can suffer from efficiency and error concerns. In general, a n-1 degree polynomial is required for n data points.

RMS (Root Mean Square)

RMS measures the amount of variation between two signals/functions. It is defined as: $\sqrt{\sum \frac{(a_i - b_i)^2}{N}}$.

A numpy based function is:

```
>>> import numpy as np
>>>
```

```
>>> def rms( f1, f2 ) :
...     d = (f1 - f2)**2
...     return np.sqrt( d.mean() )
...
```

An example of using RMS to measure how well the a cubic polynomial fits a set of data points is:

```
>>> x = np.array([ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ])
>>> y = np.array([ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59])
>>> cubic = np.polyfit(x,y, 3)
>>> cubic
array([ 0.09263115, -0.85929231,  1.80520241, -0.08861615])
>>> fy = np.polyval(cubic,x)
>>> rms( y, fy )
0.072984902548245342
```

A seventh order polynomial produces:

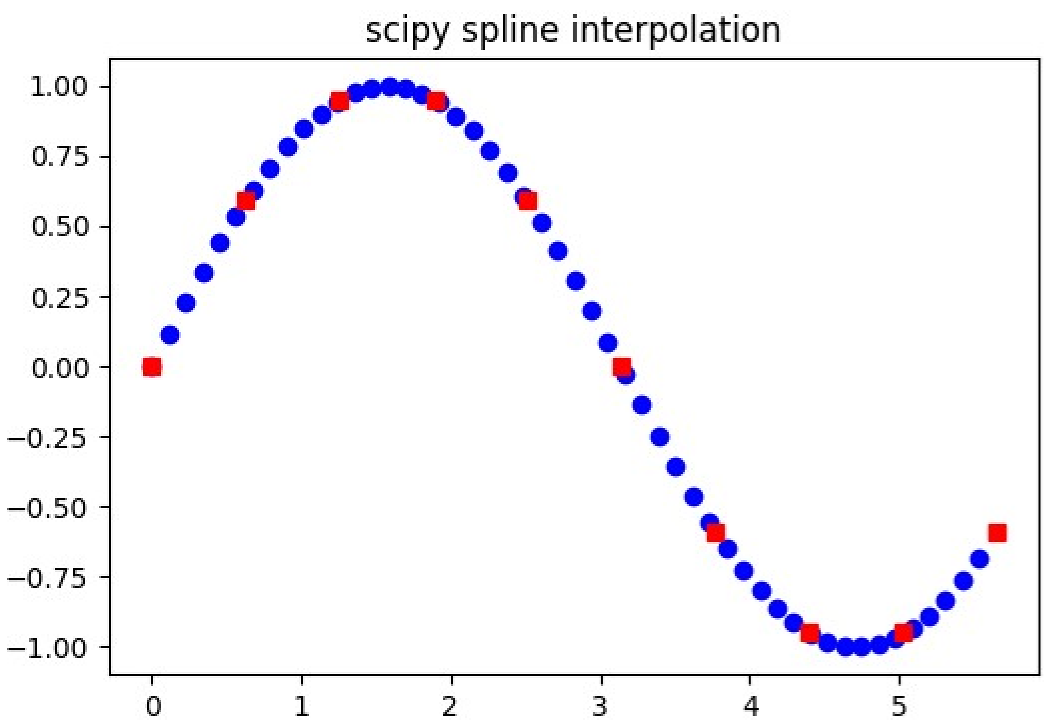
```
>>> seventh = np.polyfit(x,y, 7)
>>> seventh
array([  2.34306155e-04,  -4.96781758e-03,   3.60487155e-02,
        -8.68136906e-02,  -1.57178141e-02,  -1.31208898e-01,
         1.04191847e+00,  -1.32224750e-06])
>>> fy = np.polyval(seventh,x)
>>> rms( y, fy )
0.00031057592923985509
```

Spline interpolation

High degree polynomials can be avoided if the polynomials are fitted over small intervals in the data. A spline interpolation uses such a set of intervals. The `scipy` module provide spline fitting routines.

spline.py

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate

data_x = [ 0.0, 0.63, 1.26, 1.89, 2.51, 3.14, 3.77, 4.40, 5.03, 5.65 ]
data_y = [ 0.0, 0.59, 0.95, 0.95, 0.59, 0.00, -0.59, -0.95, -0.95, -0.59]

spl = scipy.interpolate.splrep(data_x, data_y)

plt.figure(1, figsize=(6.0,4.0), dpi=100)

x = np.linspace( 0.0, 5.65, 50)
y = scipy.interpolate.splev( x, spl)
plt.plot( x, y, "bo" )

plt.plot( data_x, data_y, "rs" )
plt.title( 'scipy spline interpolation' )
plt.savefig( 'spline.png' )
```



RMS For Spline

An example of using RMS to measure how well the spline fits is:

```
>>> import scipy.interpolate
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ModuleNotFoundError: No module named 'scipy'
>>>
>>> spl = scipy.interpolate.splrep(x, y)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'scipy' is not defined
>>> fy = scipy.interpolate.splev( x, spl)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'scipy' is not defined
>>> rms( y, fy )
0.00031057592923985509
```

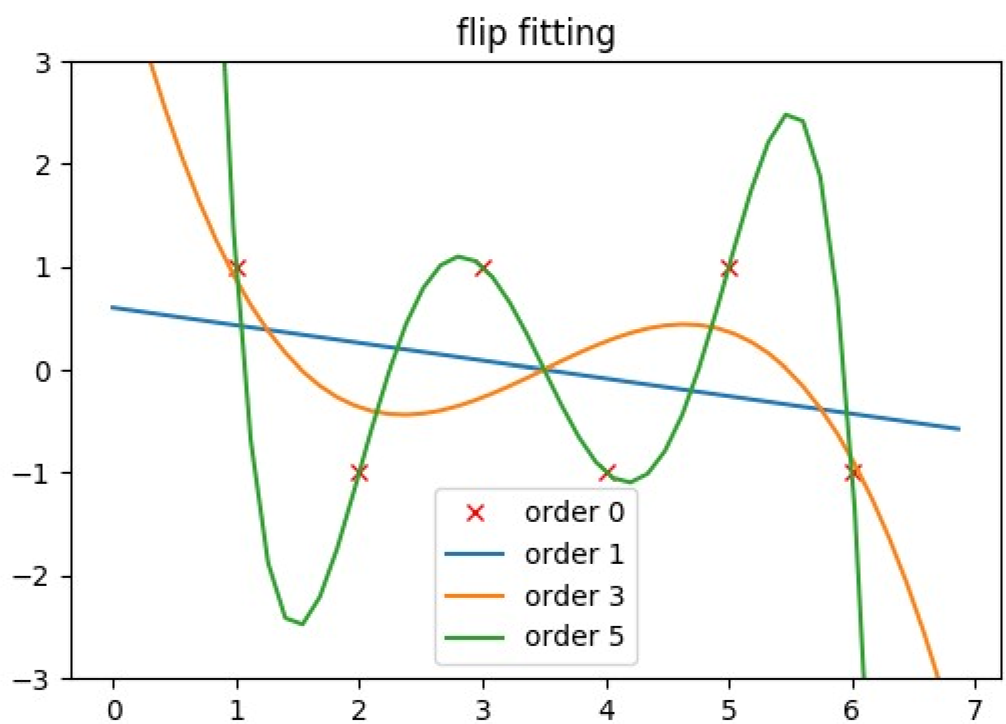
Polynomial interpolation issues

Fitting polynomials to some data can result in wild results.

flip.py

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

def flip(x) :
    if x % 2 == 0 :
        return -1
```

```
    else :
        return 1
x = np.arange( 1, 7, 1 )
y = np.array( [ flip(v) for v in x ] )

px = np.linspace( 0.0, 7.0, 50)
py = []
for order in range(1,6,2) :
    p = np.polyfit(x,y, order)
    py.append( np.polyval(p,px) )

plt.figure(1, figsize=(6.0,4.0), dpi=100)
plt.plot( x, y, "rx")
for ye in py :
    plt.plot(px,ye)
plt.legend( [ "order %d" % i for i in [0] + list(range(1,6,2)) ] )
plt.title( 'flip fitting')
plt.ylim(-3,3)

plt.savefig( 'flip.png' )
```



Flip with spline

A spline is much more stable.

spline_flip.py

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate

def flip(x) :
    if x % 2 == 0 :
        return -1
```

```
    else :
        return 1
x = np.arange( 1, 7, 1 )
y = np.array( [ flip(v) for v in x ] )

spl = scipy.interpolate.splrep(x, y)

px = np.linspace( 0.0, 7.0, 50)
py = scipy.interpolate.splev( px, spl)

plt.plot( px, py, "bo" )
plt.plot( x, y, "rs" )
plt.title( 'spline for flip')
plt.ylim(-3,3)

plt.savefig( 'spline_flip.png' )
```