Section 1

Interpolation

In experiments and simulations, we record values of a function at finite number of points. Normally we need values of the functions at some other points. This is achieved by interpolation.

Also, interpolation is mother of many numerical algorithms: integration, differentiation, ODE solver, etc. This observation will become evident when we discuss these schemes in future.

Lagrange Interpolation

First we discuss linear interpolation. Consider a function f(x) and two points on it: (x_0,y_0) and (x_1,y_1) as shown in Fig. 1. Note that $f(x_0) = y_0$ and $f(x_1) = y_1$.

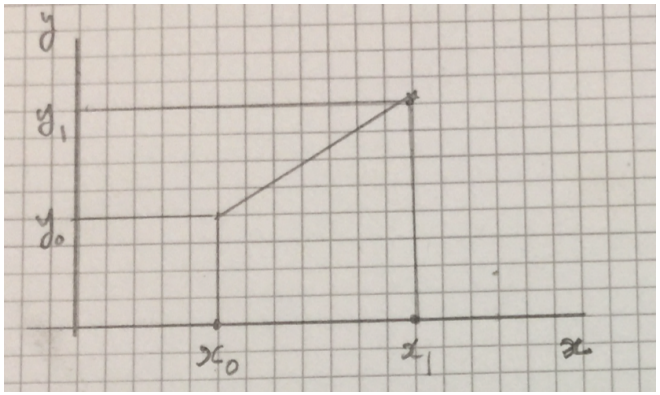


Figure 1: Linear interpolation

Clearly, the interpolating function passing through the two points is

$$f(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1$$

L_0 L_1

The first term is called L_0, while the second term L_1. Note $L_0(x_0) = 1 \quad L_0(x_1) = 0 \quad L_1(x_1) = 1 \quad L_1(x_0) = 0$

Now imagine *n* points: $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$. A function going through these points is

$$P(x) = \sum_j L_j(x) y_j \qquad (1)$$

where

$$L_j(x) = \prod_{i, i \neq j} \frac{(x - x_i)}{(x_j - x_i)} \qquad (2)$$

Note that $L_j(x_k) = \delta_{jk}$

The interpolation function is a nth order polynomial.

Example: Imagine $f(x) = \frac{1}{x}$. Using the data points *x* = (3,4), (3,4,5), (2,3,4), and (2,3,4,5), estimate *f*(3.5).

Solution: We employ formula (1) to construct the polynomials that pass through the points. The polynomials are plotted in Fig. 2 along with f(x)=1/x. We also compute f(3.5) using all the polynomials and list them in Table 1 along with the error = 1/3.5- P(3.5).

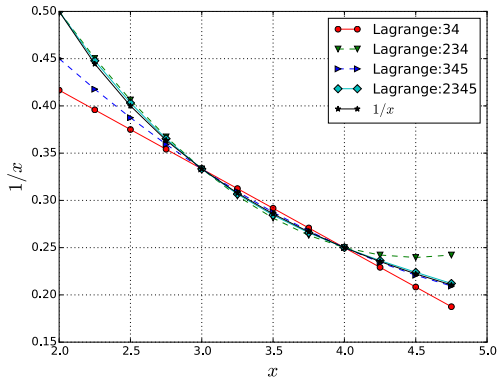


Fig. 2

Table 1

Points	P(x)	Error=1/x-P(x)
(3,4)	0.291667	-0.005952
(2,3,4)	0.28125	0.004464
(3,4,5)	0.2875	-0.001786
(2,3,4,5)	0.284375	0.001339

We also compute the interpolated values linearly-spaced points $x = [2.00,2.25,...,4.75]$ for polynomials constructed using points (3,4), (2,3,4), (3,4,5), (2,3,4,5). These values are shown in Fig. 2.

It is important to estimate errors in numerical schemes. In the following discussion we estimate error for Lagrange interpolation. The proof invokes Rolle's theorem, which is stated first. We also state how the error in Taylor's theorem estimated because the error estimation of Lagrange interpolation follows similar lines of arguments.

Rolle's theorem:Consider a function f(x) and two points x=a,b at which the function takes values f(a) and f(b) respectively. Then, according to the mean value theorem, $\exists c \in (a,b)$ such that

$$f'(c) = \frac{f(b)-f(a)}{b-a}$$

Taylor's Theorem

If $f(x)$ is differentiable $(n+1)$ times in $[a,b]$. Let $x_0 \in [a,b]$, then for every $x \in [a,b]$, $\exists \zeta(x)$ between x_0 and x such that

$$f(x) = P_n(x) + R_n(x)$$

$$P_n(x,x_0) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{n-1}(x_0)}{(n-1)!}(x-x_0)^{n-1} \quad (3)$$

$$R_n(x) = \frac{1}{n!} f^n(\zeta(x))(x-x_0)^n \quad (4)$$

Proof:

Consider a function

$$g(t) = [f(x) - P_n(x,t)] - \left(\frac{x-t}{x-x_0}\right)^n [f(x) - P_n(x,x_0)]$$

where $P_n(x,t)$ is the expansion given by Eq. (1), but around t. Let us visualize the points on the line:

Figure 3

Note that $P_n(x,x)=f(x)$. Therefore, $g(x)= g(x_0)=0$. Applying Rolle's theorem, we deduce that $\exists\zeta$ such that $g'(\zeta)=0$. Therefore,

$$0 = -P_n'(x,\zeta)] + n\frac{(x-\zeta)^{n-1}}{(x-x_0)^n}[f(x)-P_n(x,x_0)]$$

By taking derivative of Eq. (1) wrt x_0 , we obtain

$$[\frac{d}{dt}P_n(x,t)]_{t=\zeta} = \frac{f^n(\zeta)}{(n-1)!}(x-\zeta)^{n-1},$$

we obtain

$$R_n(x) = f(x) - P_n(x,x_0) = \frac{1}{n!}f^n(\zeta(x))(x-x_0)^n,$$

which is same as Eq. (4). This is how error is estimated in Taylor's theorem.

Example: Estimate e^1 as an expansion around $x = x_0 = 0$. Using Eq. (4) we obtain

$P_3(1) = e^{x=1} = 1 + x + \frac{x^2}{2!} = 1 + 1 + \frac{1}{2}$, and the error is

$$R_3(x=1) = \frac{1}{3!}f^3(\zeta(x))x^3 = \frac{1}{6}\exp(\zeta),$$

Since $\zeta \in [0,1]$, the error is bounded by $1/6 < R < e/6$, or $0.166667 < R < 0.453047$. Therefore $2.666667 < e < 2.953047$. Note that the the approximate value of e = 2.718281828 lies within the aforementioned error band.

For $x = -1$, the actual value is $1/e$ = 0.3678794412, and the estimate $P_3(x)$ = 1/2. The bound on the error is (-1/(6e),-1/6) = (-0.061313, -0.166667), and the bound on e is (0.333333, 0.438687).

Errors in Lagrange Interpolation:

The proof is in the similar lines as that for Taylor's theorem. First we define

$$g(t) = f(t) - P(t) - \big[f(x) - P(x)\big]\prod_i \frac{(t-x_i)}{(x-x_i)},$$

where x_i are the given data points, and $P(t)$ is the extrapolating n-th order polynomial. It is easy to see that $g(t=x_i)=0$ and $g(t=x)=0$. Hence $g(t)$=0 at n+1 points. Therefore, according to Rolle's theorem, $g'(t)$=0 at n points. Continuing this argument, we conclude that $g^{(2)}(t)$=0 at n-1 points, ..., and $g^{(n)}(t)$=0 at one point. We denote this point by ζ. Setting $g^{(n)}(\zeta)=0$, we obtain

$$f^{(n)}(\zeta) - P^{(n)}(\zeta) - \big[f(x) - P(x)\big]\frac{n!}{\prod(x-x_i)} = 0.$$

Since $P_n(x)$ is a (n-1)th order polynomial, $P^{(n)}(\zeta)=0$. Hence the error

$$E = f(x) - P(x) = \frac{f^{(n)}(\zeta)}{n!}\prod(x-x_i). \quad (5)$$

For the example $f(x) = 1/x$, the error is

$$E = \left| \zeta^{-(n+1)} \prod (x - x_i) \right|.$$

Therefore, for the interval [3,4], the maximum value of the error occurs for $\zeta = 3$, and the minimum for $\zeta = 4$. These values are listed in the following Table 2.

Table 2

Points	P(x)	Error	max-error	min-error
(3,4)	0.291667	-0.005952	0.03125	0.002
(2,3,4)	0.28125	0.004464	0.023438	0.001465
(3,4,5)	0.2875	-0.001786	0.00463	0.0006
(2,3,4,5)	0.284375	0.001339	0.017578	0.00018

We work out the interpolated values at the linearly-spaced 12 points $x = [2.0,2.25,...,4.75]$ for the Lagrange polynomials discussed earlier. We computer the errors for the interpolated values by computing its difference with the real value 1/*x*. The error for the four cases are exhibited using blue, red, green, and yellow curves in Fig. 4. The interpolation with 4 points yields the best interpolation.

The errors for the end points are more than the intermediate points due to the product term. The above formula indicates that the data points should be chosen carefully.

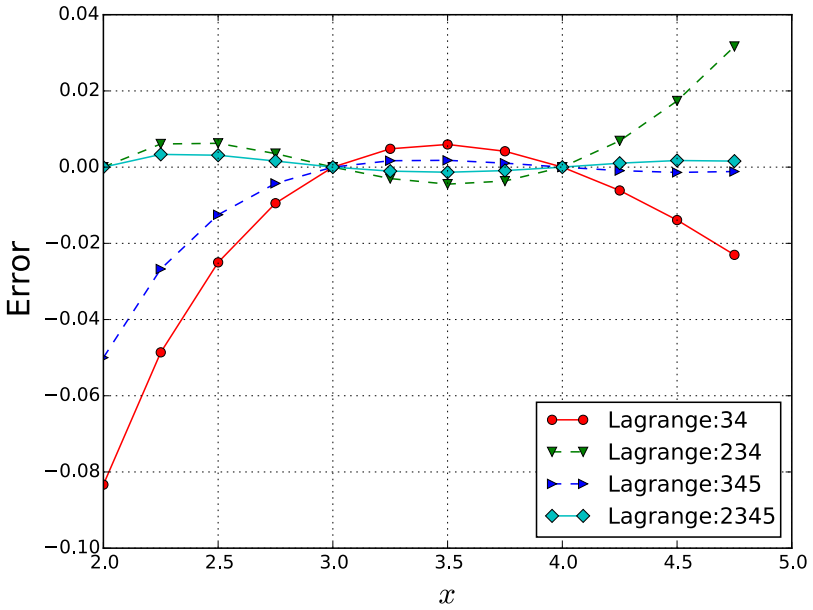


Figure 4

We compute the interpolated values using python functions interp1d and spline. Interestingly they are as accurate as the Lagrange's polynomial computed using (2,3,4,5). This is because spline is cubic order function, and the Lagrange interpolation using 4 points yields cubic order polynomial.

Lagrange Interpolation in 2D

Assume Cartesian 2D mesh with points as (xi,yi). We want to compute f(x,y) using interpolation.

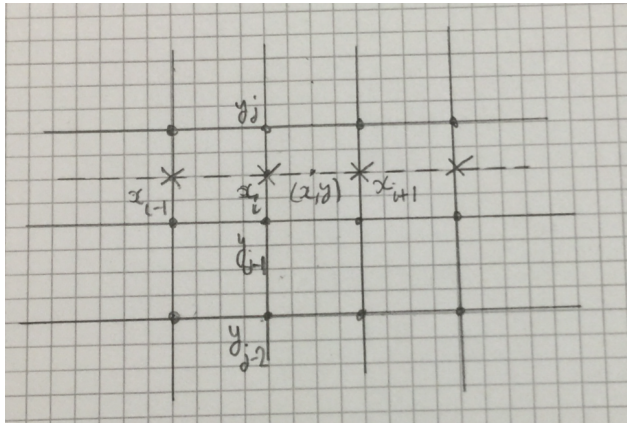


Fig. 5: 2D Mesh (x,y)

We estimate f(x,y) using x_i interpolation first.

$$P(x,y) = \sum_i \prod_{i',i' \neq i} \frac{(x - x_{i'})}{(x_i - x_{i'})} f(x_i, y)$$

Now we proceed to estimate f(x_i,y) as

$$f(x_i,y) = \sum_j \prod_{j',j' \neq j} \frac{(y - y_{j'})}{(y_j - y_{j'})} f(x_i, y_j),$$

substitution of which in P(x,y) yields

$$P(x,y) = \sum_i \prod_{i',i' \neq i} \frac{(x - x_{i'})}{(x_i - x_{i'})} f(x_i, y)$$

$$= \sum_i \sum_j \prod_{i',i' \neq i} \frac{(x - x_{i'})}{(x_i - x_{i'})} \prod_{j',j' \neq j} \frac{(y - y_{j'})}{(y_j - y_{j'})} f(x_i, y_j)$$

Hence

$$P(x,y) = \sum_i \sum_j L_{i,j}(x,y) y_{i,j}, \quad (6)$$

where

$$L_{i,j} = \prod_{i',i' \neq i} \prod_{j',j' \neq j} \frac{(x - x_{i'})}{(x_i - x_{i'})} \frac{(y - y_{j'})}{(y_j - y_{j'})} \quad (7)$$

It can be easily generalized to higher dimensions.

Lagrangian interpolation is useful when the number of points is small. The computational complexity increases for larger number of points. Also, higher-order polynomials tend to exhibit oscillations, which may be spurious (not related to real f(x)). For such cases, we employ piece-wise interpolation. However, we need to make sure that the functions are smooth at the intersections. This is achieved by spline, which is topic of the next subsection.

Spline

Here we discuss cubic spline, which is the solution of the beam equation:

$$EI \frac{d^4}{dx^4} f(x) = F(x), \quad (1)$$

where *E* is the Young's modulus of the material, *I* is the moment of inertia of a cross section, and *F* is the applied force. In

the spline scheme, the force is assumed to be active at points x_i (called *nodes*) with i=0:(n-1), i.e.,

$$EI\frac{d^4}{dx^4}f(x) = F_i\delta(x - x_i)$$

Hence, $f'''(x)$ would be discontinuous at the nodes.

---

i -1 i i+1

Fig 6

We Demand:

1. The curve is piecewise linear in each interval.

2. The curve passes through each of the given points (x_i, y_i).

3. The first and second derivatives are continuous at each of the given points

An example of the function is shown in Fig. 7.

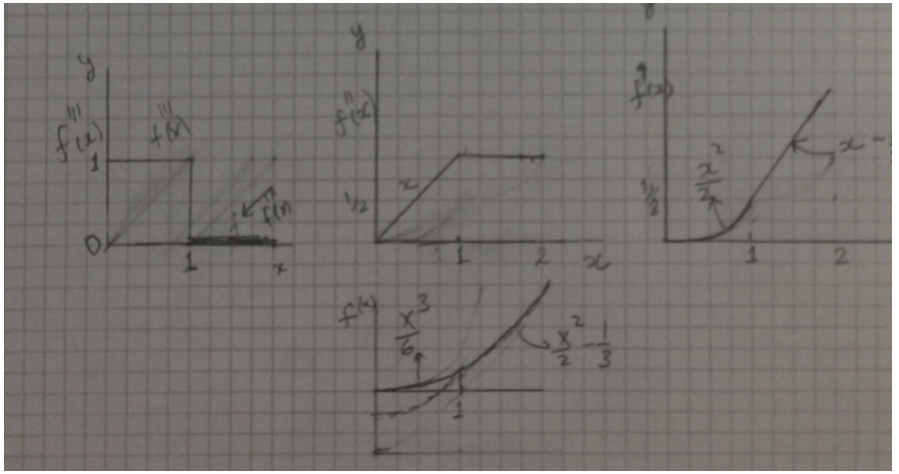


Fig. 7

Suppose we are given $f''(x_i)$ at each point, then using Lagrange's linear interpolation formula, we obtain

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

whose integration yields $f_i(x)$ in the interval (x_i:x_{i-1}) as

$$f_i(x) = f''(x_i)\frac{(x_{i+1} - x)^3}{6h_i} + f''(x_{i+1})\frac{(x - x_i)^3}{6h_i}$$

$$+ \left[\frac{y_i}{h_i} - \frac{h_i}{6}f''(x_i)\right](x_{i+1} - x)$$

$$+ \left[\frac{y_{i+1}}{h_i} - \frac{h_i}{6}f''(x_{i+1})\right](x - x_i),$$

where $h_i = x_{i+1} - x_i$. The constants of integration are determined using the conditions: $f_i(x_i) = y_i$, and $f_i(x_{i+1}) = y_{i+1}$. In the previous interval $(x_{i-1} : x_i)$, the function is

$$f_{i-1}(x) = f''(x_{i-1})\frac{(x_i - x)^3}{6h_{i-1}} + f''(x_i)\frac{(x - x_{i-1})^3}{6h_{i-1}}$$

$$+ \left[\frac{y_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{6}f''(x_{i-1})\right](x_i - x)$$

$$+ \left[\frac{y_i}{h_{i+1}} - \frac{h_{i-1}}{6}f''(x_i)\right](x - x_{i-1}).$$

We impose an additional condition that the first derivative at the nodes is continuous at both sides. Applying this condition to the node at $x = x_i$, i.e., $f_i'(x_i) = f_{i-1}'(x_i)$, we obtain

$$-f''(x_i)\frac{h_i}{2} - \left[\frac{y_i}{h_i} - \frac{h_i}{6}f''(x_i)\right] + \left[\frac{y_{i+1}}{h_i} - \frac{h_i}{6}f''(x_{i+1})\right]$$

$$= -f''(x_i)\frac{h_{i-1}}{2} - \left[\frac{y_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{6}f''(x_{i-1})\right]$$

$$+ \left[\frac{y_i}{h_{i-1}} - \frac{h_{i-1}}{6}f''(x_i)\right],$$

which yields

$$\frac{h_{i-1}}{6}f''(x_{i-1}) + \frac{1}{3}(h_i + h_{i-1})f''(x_i) + \frac{h_i}{6}f''(x_{i+1})$$

$$= \frac{y_{i+1}}{h_i} - y_i\left(\frac{1}{h_i} + \frac{1}{h_{i-1}}\right) + \frac{y_{i-1}}{h_{i-1}}. \quad (8)$$

We obtain (*n*-2) equations for nodes *i*=1:(*n*-2). However we have *n* unknowns *f"*(x_i). To solve this problem, we use one of the following boundary conditions:

1. **Periodic Spline**: We assume that the data is periodic with y_0 identified with y_{n-1}, or $y_0 = y_{n-1}$. Hence, we have (n-1) points, and (n-1) matching conditions in Eq. (8). These conditions are suffice to determine all the (n-1) $f''(x_i)$.

2. **Parabolic Run-out**: We assume that *f*' are constant on both end intervals, hence *f* is quadratic here. Therefore,

$f''(x_0) = f''(x_1)$ and $f''(x_{n-1}) = f''(x_{n-2})$

3. **Free End**: We assume that *f*'=0 at both the ends, or $f''(x_0) = f''(x_{n-1}) = 0$.

4. **Cantilever End**: A intermediate condition between the cases 2 and 3, i.e.,

$f''(x_0) = \lambda f''(x_1)$ and $f''(x_{n-1}) = \lambda f''(x_{n-2})$,

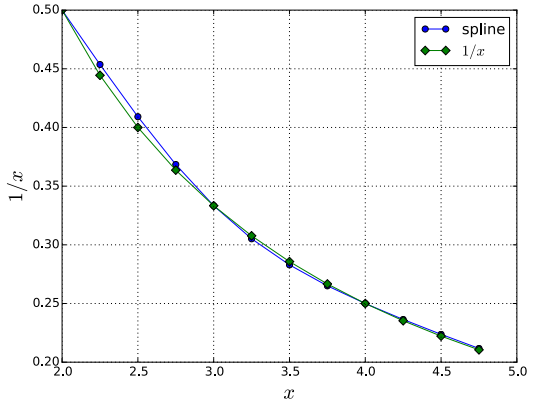
with $0 \le \lambda \le 1$.

We work out the spline interpolation for the example discussed earlier with four points (2,3,4,5). Let us use the Free End boundary condition for this example. Note that $h_i = 1$. Therefore $f_0'' = f_3'' = 0$, and

$$\frac{2}{3}f_1'' + \frac{1}{6}f_2'' = y_2 - 2y_1 + y_0 = \frac{1}{4} - \frac{2}{3} + \frac{1}{2}$$

$$\frac{1}{6}f_1'' + \frac{2}{3}f_2'' = y_3 - 2y_2 + y_1 = \frac{1}{5} - \frac{2}{4} + \frac{1}{3}$$

Using the solution $f''_{1,2}$, we construct $f_i(x)$ discussed earlier. The combine plot of $f_i(x)$ is shown below. The actual curve 1/*x* is shown as the green curve. The spline interpolated function fits well with 1/*x*.



For more points, we need to solve the matrix equation. Fortunately, the matrix is tridiagonal. We will discuss how to solve such matrices in later chapters.

Python functions for interpolation

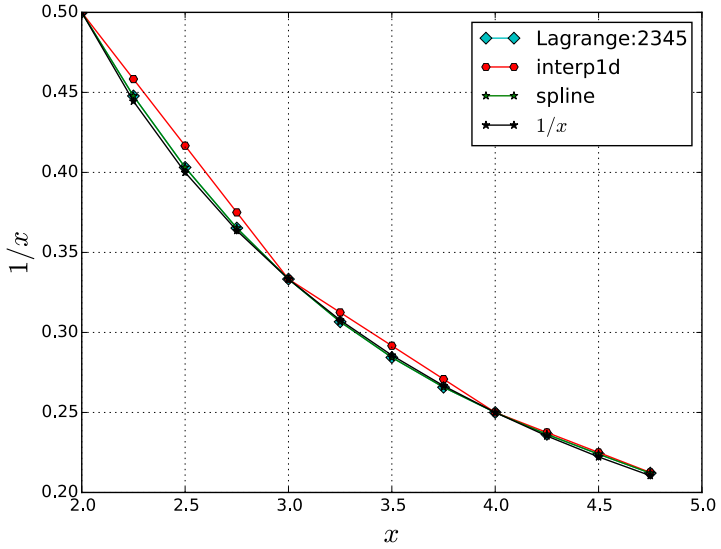
Python offers interpolation functions interp1d and spline function splrev, whose usage is show below. These functions are part of scipy module.

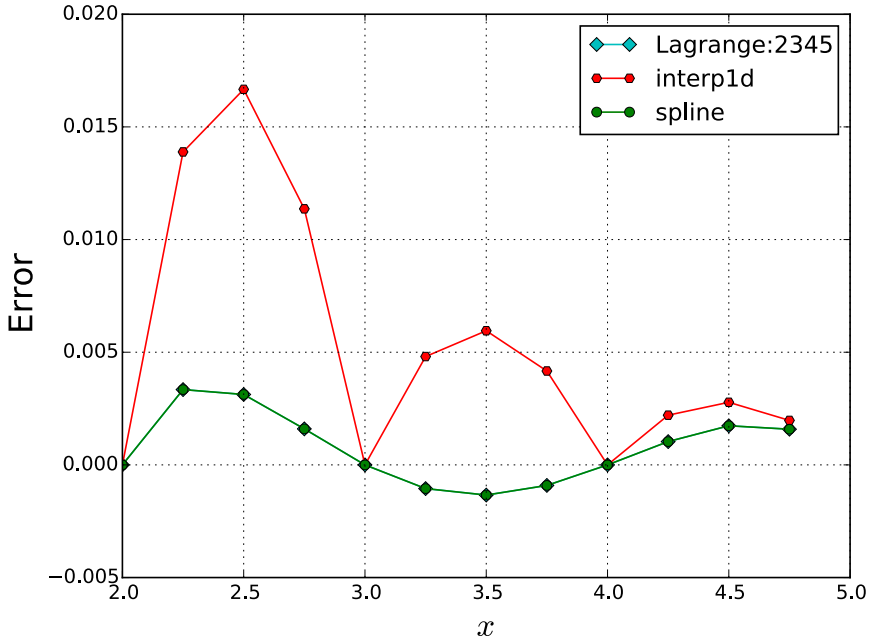
Using Python function interpolate.interp1d

```
xarray = np.array([2,3,4,5])
yarray = np.array([1/2.0,1/3.0,1/4.0,1/5.0])

# python defined function
# interpolate 1d
# interp1d yields a function yp.

from scipy import interpolate

x = np.arange(2,5,0.25)
yp = interpolate.interp1d(xarray,yarray)

yinterp1d = []
for x_p in x:
      yinterp1d.append(yp(x_p))
yinterp1d  = np.array(yinterp1d)
```

Using Python function interpolate.splrep

```
# Using spline, output in tck
# Using tck, splev
tck = interpolate.splrep(xarray,yarray)

yspline = []
for x_p in x:
      yspline.append(interpolate.splev(x_p,tck))
yspline  = np.array(yspline)

print("Using spline: ", interpolate.splev(3.5,tck))
```


Lagrange:2345
interp1d
spline
$1/x$
$1/x$
x
0.50
0.45
0.40
0.35
0.30
0.25
0.20
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Lagrange:2345
interp1d
spline
Error
x
0.020
0.015
0.010
0.005
0.000
−0.005
2.0
2.5
3.0
3.5
4.0
4.5
5.0