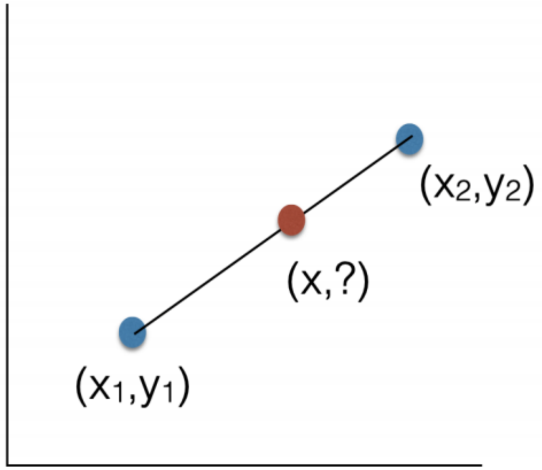
Interpolation

2017135002/최성윤

기본원리(Linear Interpolation)



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

$$y = w_1 y_1 + w_2 y_2$$

구해진 값을 통하여 구한 1차함수를 통하여 지점과 지점 사이의 값을 계산한다

ω_1, ω_2 weigthed average를 통하여 구할수있다

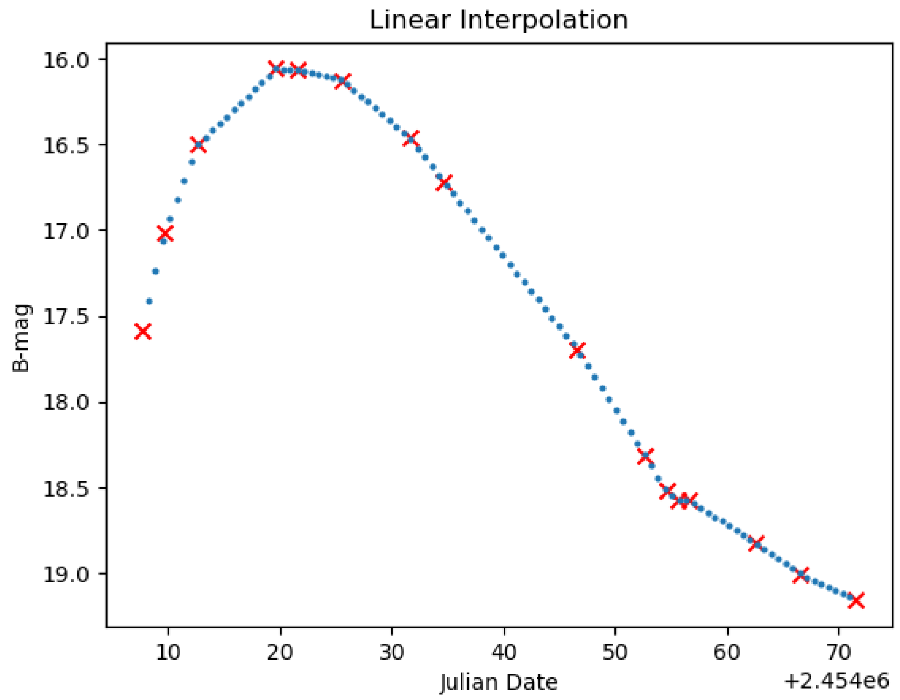
실행

```
j,m=np.loadtxt('sn.dat',usecols=(1,2),skiprows=1,unpack=True)
jd=np.linspace(min(j),max(j),102)
jd=jd[1:101]
bm=[]
mm=0
for a in range(100):
    mm=0
    for b in range(16):
        if j[b]<jd[a]<j[b+1]:
            mm=m[b]+(m[b+1]-m[b])/(j[b+1]-j[b])*(jd[a]-j[b])
            bm.append(mm)
plt.scatter(j,m,s=50,c='r',marker='x')
plt.scatter(jd,bm,s=4)
plt.gca().invert_yaxis()
plt.show()
```

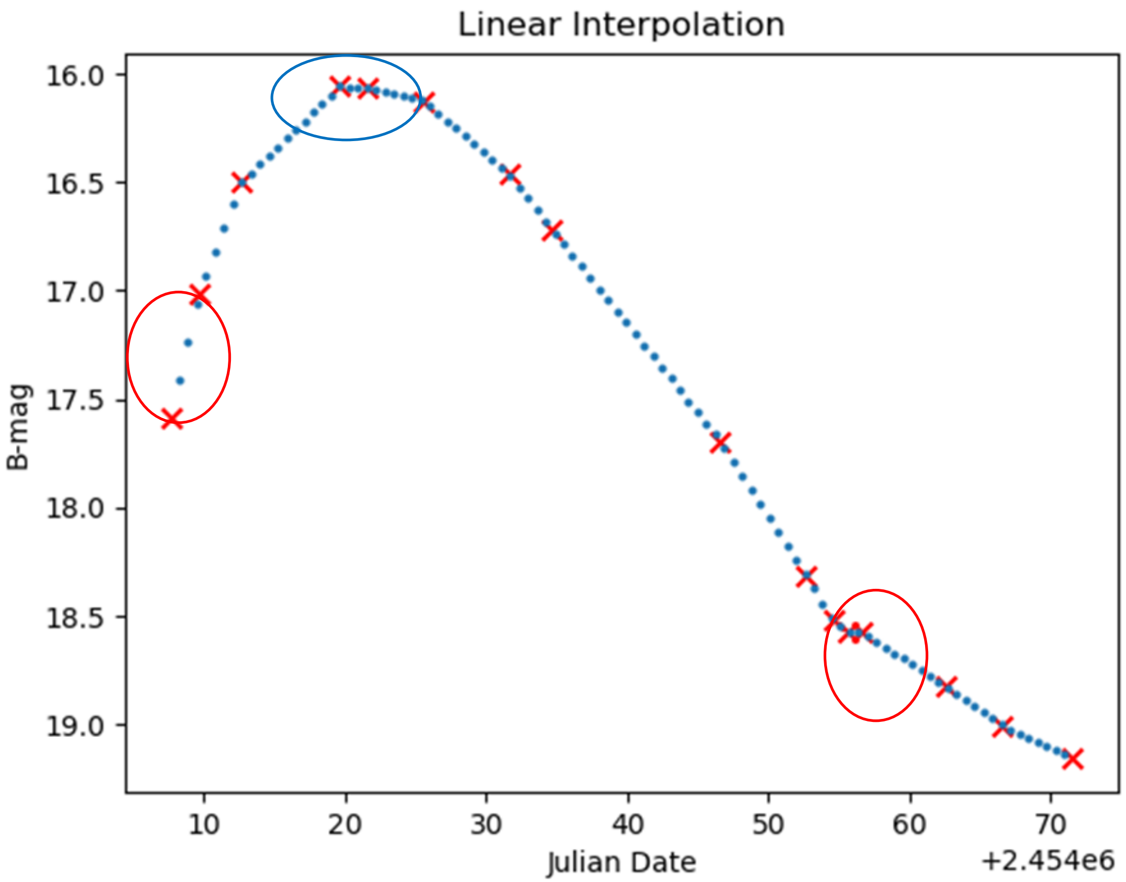
최대값과 최소값 사이를 100개의 균등한 간격으로 나누었다.
(최대값, 최소값은 미포함)

각각 100개의 난수에 대하여
Linear Interpolation 실행

결과



토의사항



1.최대값과 최소값 사이의 균등한 간격을 가지는 값으로 선형 내삽을 하였기 때문에 일부구간은 데이터가 적고 일부 구간은 데이터가 많다.

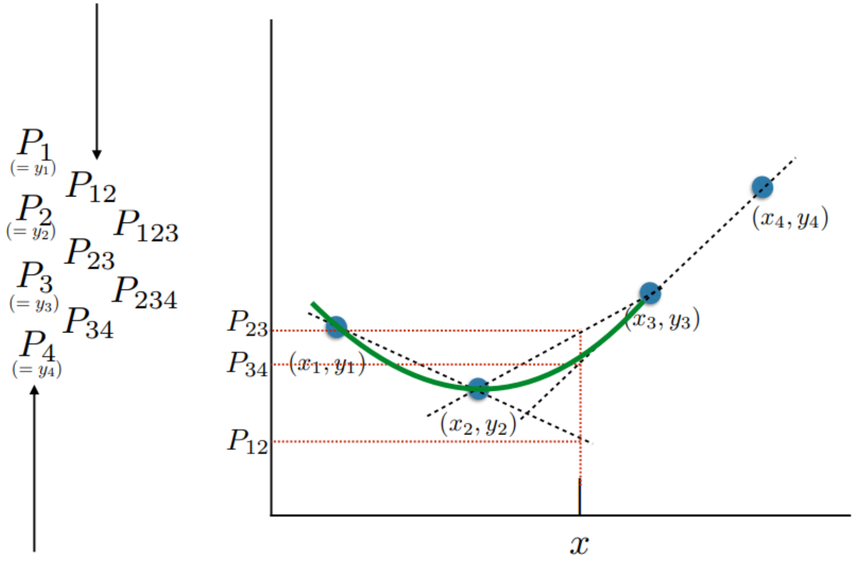
2.데이터들이 선형으로 변화하기때문에 부드럽게 연결하지 못한다

기본원리(Polynomial Interpolation)

1.Neville's Algortihm

2개의 포인트를
이용한 y 값 추정

Neville's Algorithm



1개의 포인트를 이용한 y 값 추정

1차 interpolation -> 2차 interpolation
->……->m차 interpolation 해당 식 반복 수행

$$P_{i(i+1)\dots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\dots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\dots(i+m)}}{x_i - x_{i+m}}$$

m은 주어진 데이터 개수-1
현재 sn data에서는 m=15

기본원리(Polynomial Interpolation)

2. Newton’s Method

N+1개의 데이터가 있을경우 equation 다음과 같이 정의(sn에서는 N=15)

$P_0(x) = a_3$

$P_1(x) = a_2 + (x - x_2)\,P_0(x)$

$P_2(x) = a_1 + (x - x_1)\,P_1(x)$

$P_3(x) = a_0 + (x - x_0)\,P_2(x)$

$$P_0(x) = a_n \qquad P_k(x) = a_{n-k} + (x - x_{n-k})\,P_{k-1}(x), \quad k = 1, 2, \ldots, n$$

$$P_n(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + \cdots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})a_n$$

$$a_0 = y_0 \qquad a_1 = \nabla y_1 \qquad a_2 = \nabla^2 y_2 \qquad \cdots \qquad a_n = \nabla^n y_n$$

Coefficient를 구하기위해서는?

$$\nabla^n y_n = \frac{\nabla^{n-1} y_n - \nabla^{n-1} y_{n-1}}{x_n - x_{n-1}}$$

x_0	y_0				
x_1	y_1	∇y_1			
x_2	y_2	∇y_2	$\nabla^2 y_2$		
x_3	y_3	∇y_3	$\nabla^2 y_3$	$\nabla^3 y_3$	
x_4	y_4	∇y_4	$\nabla^2 y_4$	$\nabla^3 y_4$	$\nabla^4 y_4$

실행

1.Neville's Algortihm(3차 interpolation까지 구현)

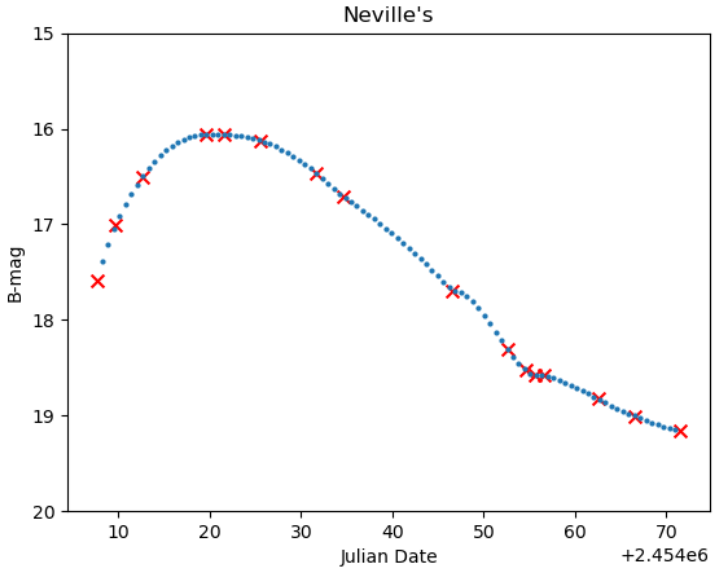
```
NA=[]
for a in range(100):
    for b in range(16):
        #하지만 이 if문은 b가 13일때부터는 리스트의 범위를 초과하여 따로 정해준다
        #b<=12인 경우의 if문 j[b] ~ j[b+3] 을 사용하여 Neville's Algorithm 사용
        if j[b]<jd[a]<j[b+1] and b<=12:
            P12=((jd[a]-j[b+1])*m1[b]+(j[b]-jd[a])*m1[b+1] )/(j[b]-j[b+1])
            P23=((jd[a]-j[b+2])*m1[b+1]+(j[b+1]-jd[a])*m1[b+2] )/(j[b+1]-j[b+2])
            P34=((jd[a]-j[b+3])*m1[b+2]+(j[b+2]-jd[a])*m1[b+3] )/(j[b+2]-j[b+3])
            P123=( (jd[a]-j[b+2])*P12+(j[b]-jd[a])*P23 )/(j[b]-j[b+2])
            P234=( (jd[a]-j[b+3])*P23+(j[b+1]-jd[a])*P34 )/(j[b+1]-j[b+3])
            P1234=( (jd[a]-j[b+3])*P123+(j[b]-jd[a])*P234 )/(j[b]-j[b+3])
            NA.append(P1234)
        # b>12인 경우의 if문 j[b-2] ~ j[b+1] 을 사용하여 Neville's Algorithm 사용
        elif j[b] < jd[a] < j[b + 1] and b > 12:
            P12 = ((jd[a] - j[b - 1]) * m1[b - 2] + (j[b - 2] - jd[a]) * m1[b - 1]) / (j[b - 2] - j[b - 1])
            P23 = ((jd[a] - j[b]) * m1[b - 1] + (j[b - 1] - jd[a]) * m1[b]) / (j[b - 1] - j[b])
            P34 = ((jd[a] - j[b+1]) * m1[b] + (j[b] - jd[a]) * m1[b+1]) / (j[b] - j[b+1])
            P123 = ((jd[a] - j[b]) * P12 + (j[b-2] - jd[a]) * P23) / (j[b-2] - j[b])
            P234 = ((jd[a] - j[b+1]) * P23 + (j[b-1] - jd[a]) * P34) / (j[b-1] - j[b+1])
            P1234 = ((jd[a] - j[b+1]) * P123 + (j[b-2] - jd[a]) * P234) / (j[b-2] - j[b+1])
            NA.append(P1234)
```

j는 데이터 16개
jd는 균등한 간격을 가지는100개의 JD

j[b]<jd[a]<j[b+1] 인경우 interpolation 할 4개 변수는 j[b]~j[b+3]까지 설정

b 12초과시 범위를 벗어나기 때문에 interpolation 할 4개 변수는 j[b-2]~j[b+1]로 설정

결과



실행

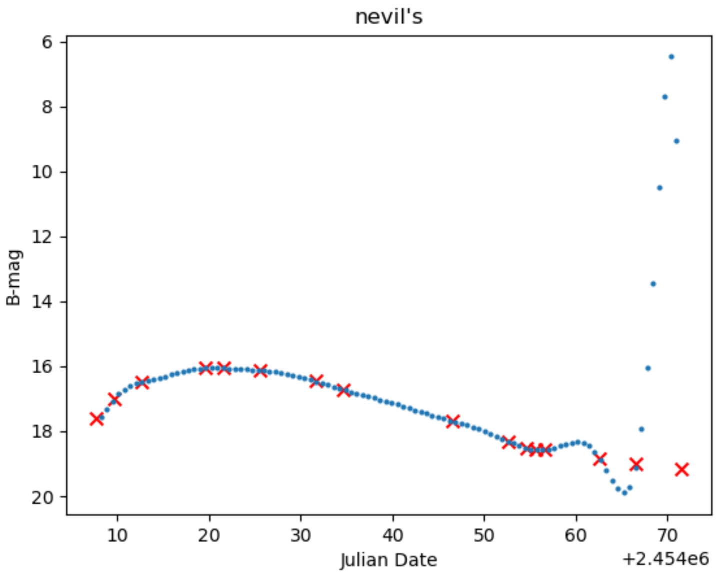
2. Neville's Algortihm(재귀함수를 사용하여 직접 구현)

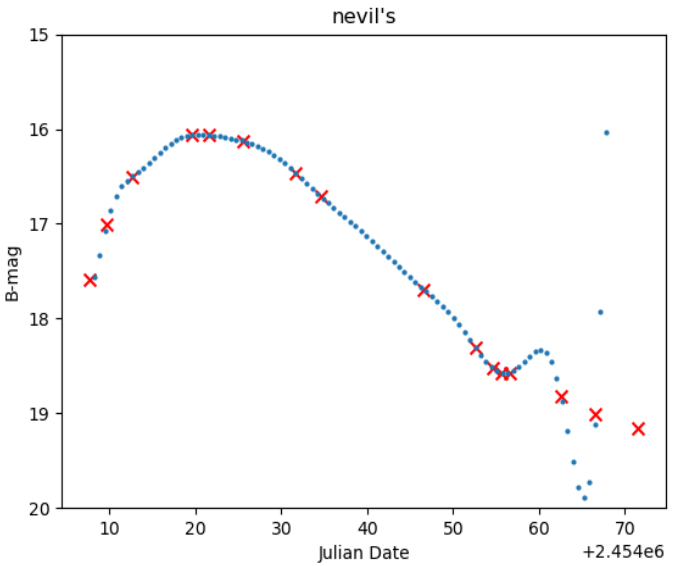
$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

```
def ne(n, m, a):
    global m1
    global j
    if n == m:
        return m1[n - 1]
    else:
        return ((a - j[m - 1]) * ne(n, m - 1, a) + (j[n - 1] - a) * ne(n + 1, m, a)) / (j[n - 1] - j[m - 1])
```

위의 식에서
i가 ne()에서는 n
i+m이 ne()에서는 m
ne()에서 a는 Julian Date

결과





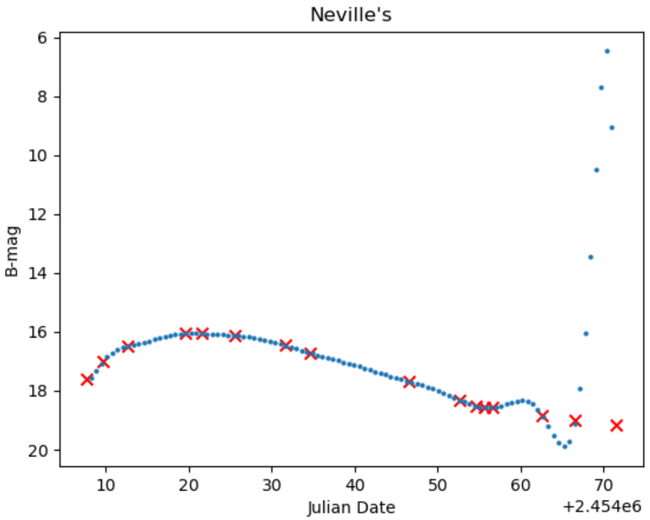
실행

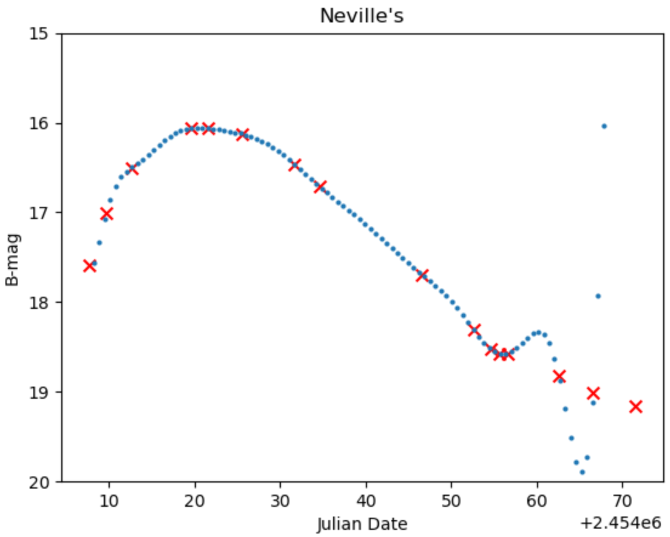
3.Neville's Algortihm(라이브러리 사용)

```
#Neville
def neville(xData,yData,x):
    m=len(xData)
    y=yData.copy()
    for k in range(1,m):
        y[0:m-k]=(xData[k:m]-x)*y[0:m-k]+(x-xData[0:m-k])*y[1:m-k+1]
        y[0:m-k]=y[0:m-k]/(xData[k:m]-xData[0:m - k])
    return y[0]
mm=[]
for i in jd:
    mm.append(neville(j,m1,i))
```

해당 for문 반복마다 k차 interpolation 수행
그후 m-1차 interpolation 한 값 반환

결과





실행

4.Newton’s Method(재귀함수를 사용하여 직접 구현)

```
cof=[]
mm=[]
def co(n,m):
    global j
    global m1
    if n==0:
        return m1[m]
    else:
        return ( co(n-1,m)-co(n-1,n-1) )/ (j[m]-j[n-1])
for i in range(16):
    cof.append(co(i,i))
def nw(k,x):
    global j
    global m1
    global cof
    n=len(j)-1
    if k==0: return cof[n]
    else:
        return cof[n-k]+(x-j[n-k])*nw(k-1,x)
for i in jd:
    mm.append(nw(15,i))
```

Coefficient를 계산하는 재귀함수

$$\nabla^n y_m = (\nabla^{n-1} y_m - \nabla^{n-1} y_{n-1}) \quad / \; x_m - x_{n-1}$$

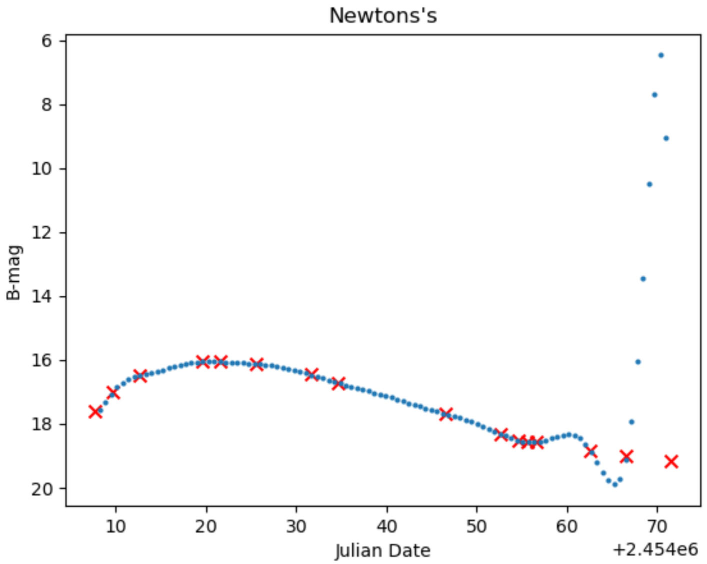
co(i,i) 는 $\nabla^n y_n = a_n$을 의미한다.

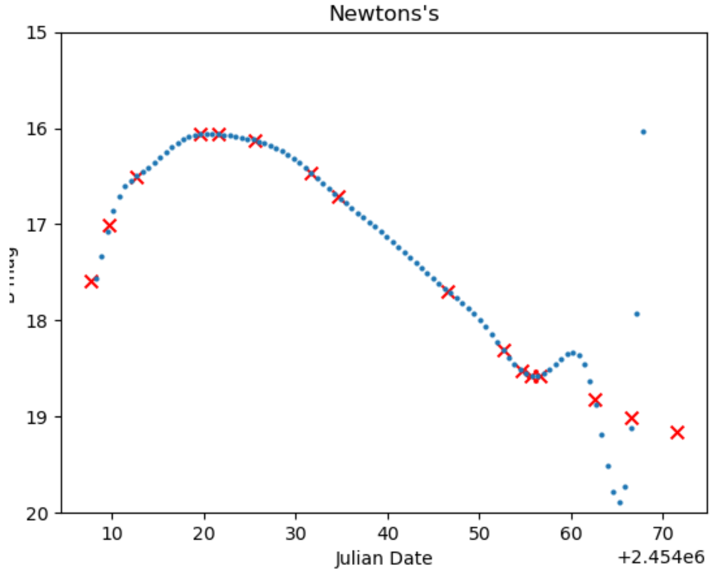
Polynomial을 계산하는 재귀함수

$P_k(x) = a_{n-k} + (x - x_{n-k}) * P_{k-1}(x)$ 을 이용

nw(k,x)에 $P_k(x)$ 반환 (k는 데이터개수-1 , x는 mag값을 계산할 JD)

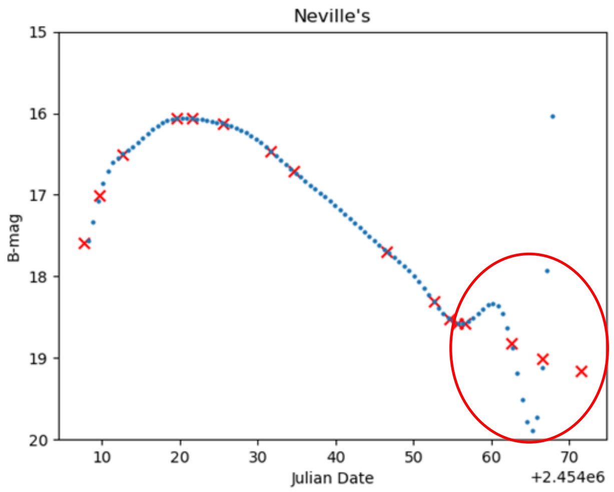
결과

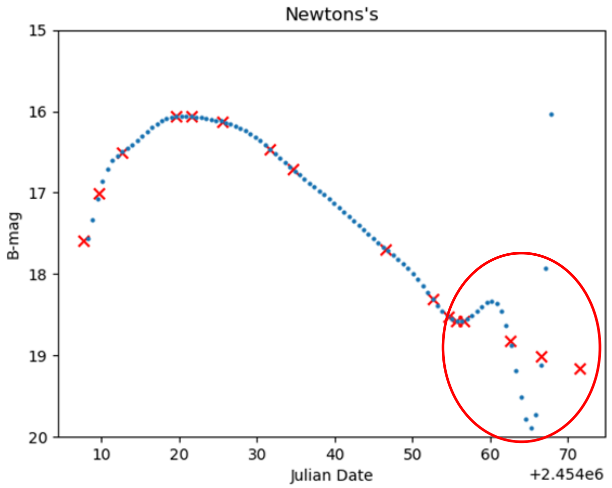




토의사항

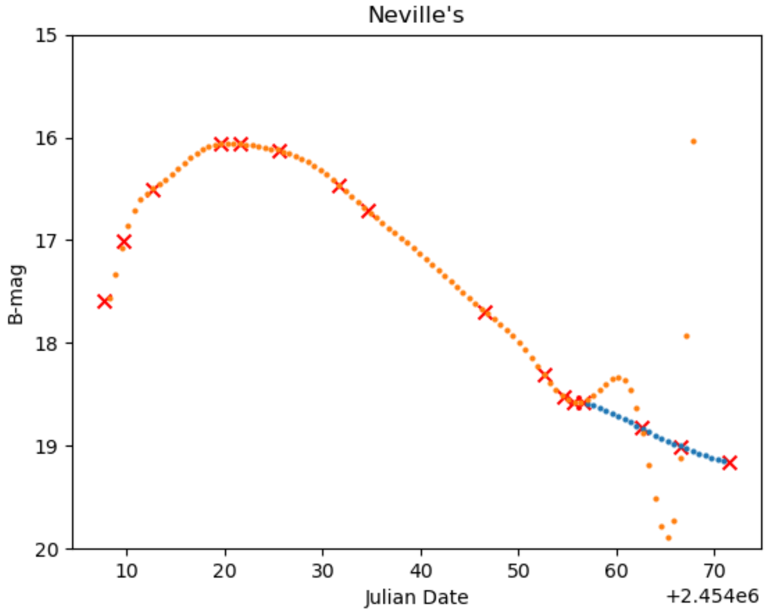
Polynomial Interpolation의 한계





핵버그 발생(값이 갑자기 급변한다.)
->오차가 심하다

오차가 발생하는 구간은
실행2를 사용하여 interpolation



기본원리(Cubic Spline Interpolation)

위의 Interpolation들의 단점을 보완

$$y = Ay_j + By_{j+1} + Cy_j'' + Dy_{j+1}''$$

$$A = \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B = \frac{x - x_j}{x_{j+1} - x_j}$$

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2$$

Second Derivates(y_j'') 구하기

$$\frac{x_j - x_{j-1}}{6} y_{j-1}'' + \frac{x_{j+1} - x_{j-1}}{3} y_j'' + \frac{x_{j+1} - x_j}{6} y_{j+1}'' = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}$$

J는 2~N-1까지 이지만
구해야하는 y''는 N개이다.

$$y_1'' = y_N'' = 0.$$

실행

1. (np.solve를 통한 직접 구현)

```
Y=[y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15]
e=[]
for i in range(14):
    e.append((j[i+1]-j[i])*Y[i]/6+(j[i+2]-j[i])*Y[i+1]/3+(j[i+2]-j[i+1])*Y[i+2]/6-(m[i+2]-m[i+1])/(j[i+2]-j[i+1])+(m[i+1]-m[i])/(j[i+1]-j[i]))
result=solve(e)
for a in range(100):
    mm=0
    for b in range(16):
        if j[b]<jd[a]<j[b+1]:
            e=str('y')+str(b)
            f =str('y') + str(b+1)
            A=(j[b+1]-jd[a])/(j[b+1]-j[b])
            B=(jd[a]-j[b])/(j[b+1]-j[b])
            C=(A**3-A) * ((j[b+1]-j[b])**2) / 6
            D = (B ** 3 - B) * ((j[b + 1] - j[b]) ** 2) / 6
            mm=A*m[b]+B*m[b+1]+C*k[b]+D*k[b+1]
            bm.append(mm)
```

옆의 epquation을 np.solve를 통하여 y1~y14까지의 값을 구한다.
Second Derivates(y_j'') 값 계산
(y0=y15=0)

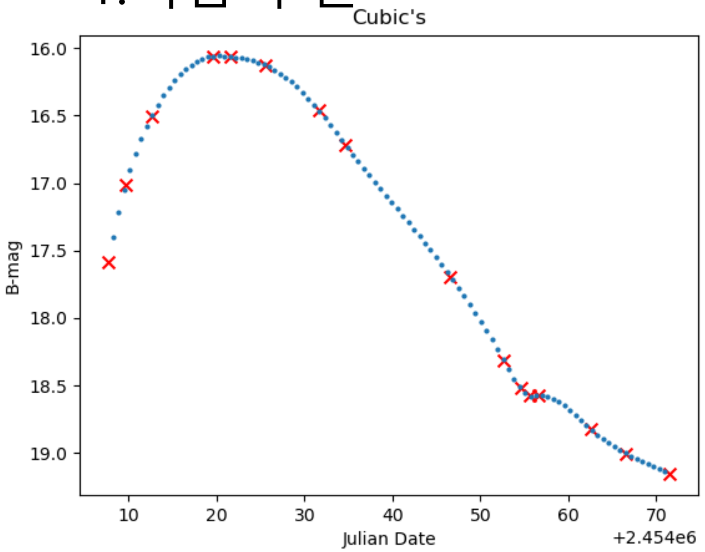
$$y = Ay_j + By_{j+1} + Cy_j'' + Dy_{j+1}''$$

그후 위의 식을 이용하여 과제1와 같은 linear interpolation 수행

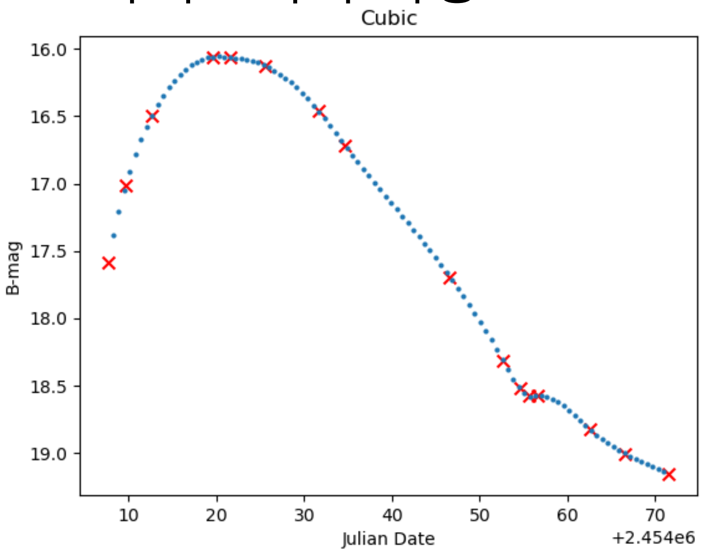
2.라이브러리 사용

```
cb=cs(j,m)
plt.scatter(j,m,s=50,c='r',marker='x')
plt.scatter(jd,cb(jd),s=4)
plt.gca().invert_yaxis()
plt.title('Cubic')
plt.xlabel('Julian Date')
plt.ylabel('B-mag')
plt.show()
```

1.직접 구현

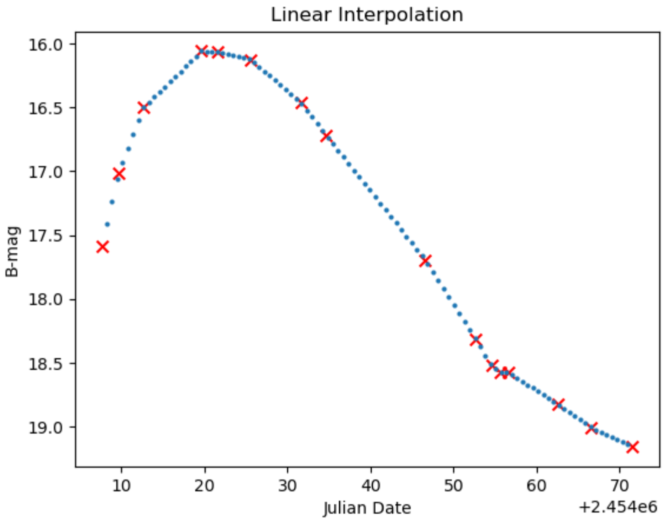


2.라이브러리 사용

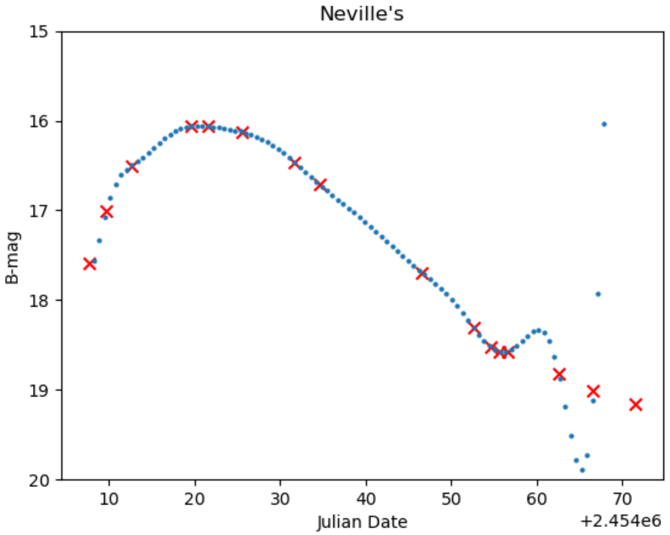


토의사항

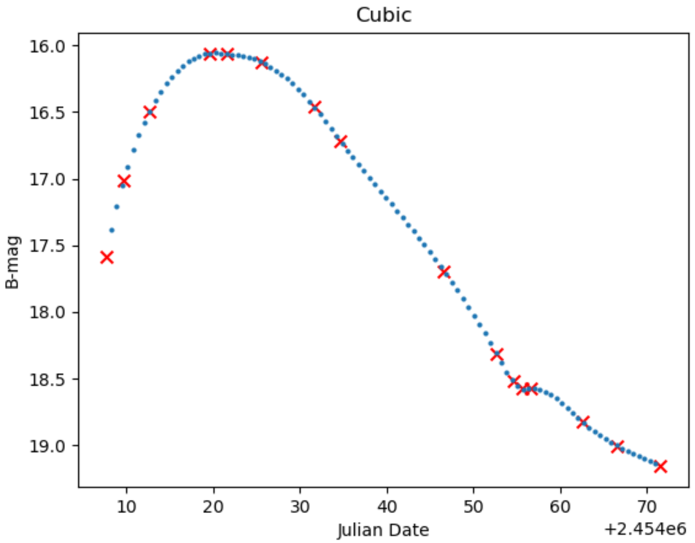
Linear vs Polynomial vs Spline



vs



vs

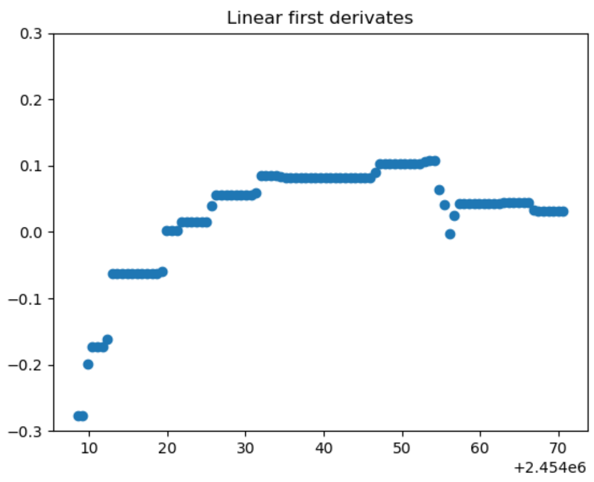


Linear의 경우 값들이 이어지기는 하지만 부드럽게 이어지지 못하고 선형적으로 이어진다

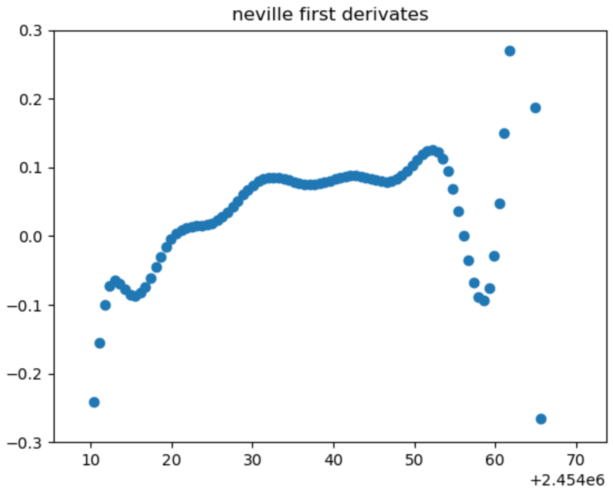
Polynomial의 경우 값들이 어느정도 부드럽게 이어지기는 하지만 값이 급변 하는 구간이 있다.

Cubic spline의 경우 값들이 부드럽게 이어지며 값이 급변하는 구간도 나타나지 않는다

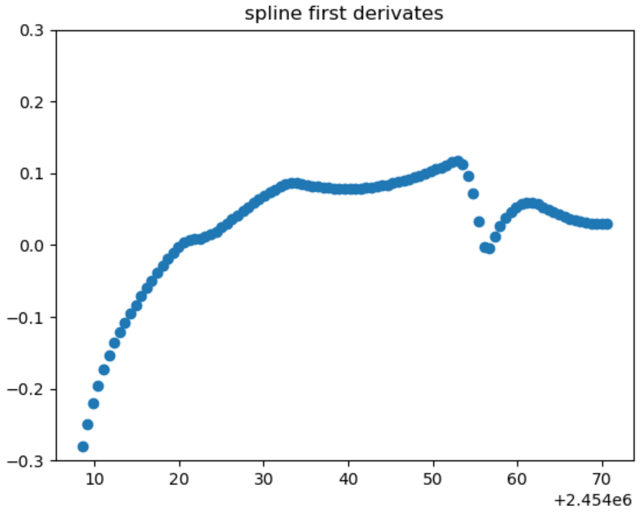
First Derivates 비교



vs



vs

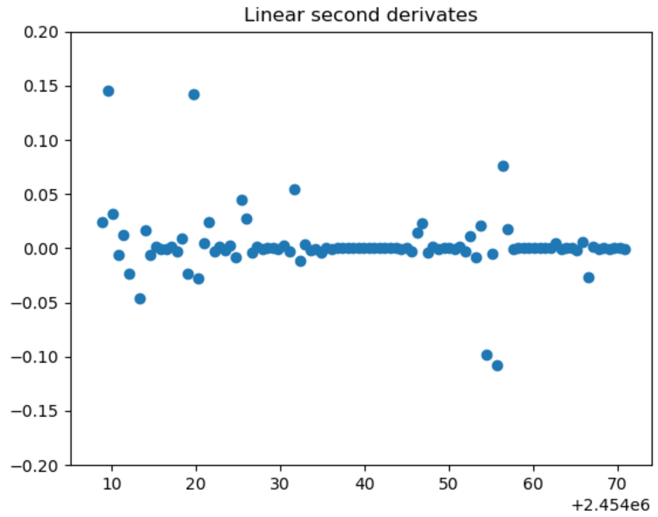


Linear의 경우 값이 이어지지 못하고 중간 중간 급변한다.

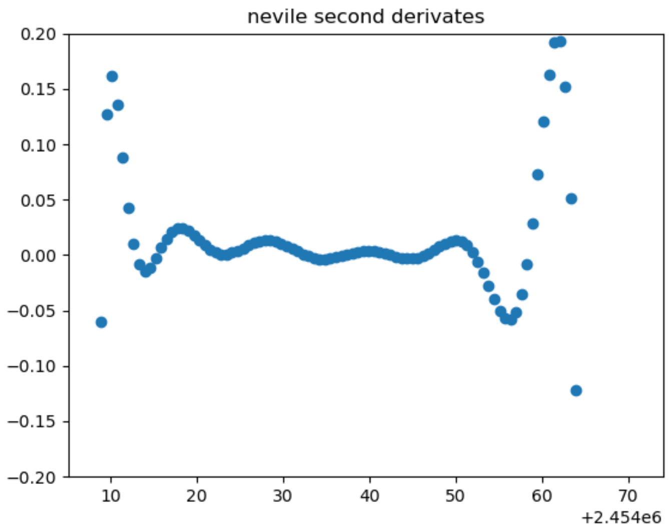
Polynomial의 경우 값들이 어느정도 부드럽게 이어지기는 양 끝에서 급변한다.

Cubic spline의 경우 일부구간에서 값이 급변하지만 대체로 다 부드럽게 이어진다

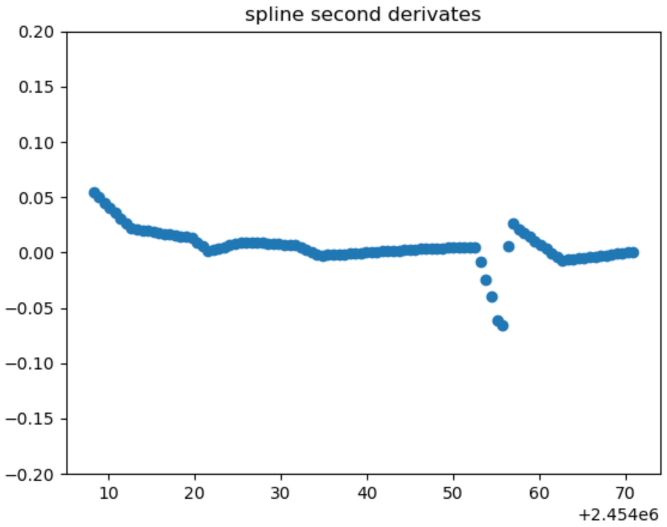
Second Derivates 비교



vs



vs



Linear의 경우 값들이 불연속적으로 값이 바뀐다/

Polynomial의 경우 값들이 어느정도 부드럽게 이어지기는 양 끝에서 급변한다.

Cubic spline의 경우 일부구간에서 값이 급변하지만 대체로 다 부드럽게 이어진다

Cubic Spline Method 승!

Reference

How to plot the derivative of a plot python?, Stackoverflow, 2018/10/23

https://stackoverflow.com/questions/52957623/how-to-plot-the-derivative-of-a-plot-python

Second Derivative in python - scipy/numpy/pandas, Stackoverflow, 2016/10/24

https://stackoverflow.com/questions/40226357/second-derivative-in-python-scipy-numpy-pandas

scipy.interpolate.CubicSpline, Scipy.org

https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html

Maths III - Numerical Methods Matt Probert