Assignment07

November 11, 2018

- Name : Joonyoung-Choi
- Student ID: 20112096
- Description: Polynomial fitting
- github: https://github.com/mydream757/Computer_Vision

1. Import liabraries

- import needed libraries.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

2. Define funtions

- this makes data following this fomula.

$f(x) = |x| \cdot \sin(x)$

```
In [2]: def fun(x):
            # f = np.sin(x) * (1 / (1 + np.exp(-x)))
            f = np.abs(x) * np.sin(x)
            return f
```

- this make y-data of polynomial function.

```
In [3]: def polyFun(x,coEff,k):
            temp = np.ones(x.shape)
            for i in range(k):
                if i==0:
                    temp *= coEff[k-1]
                temp = temp + coEff[k-i-1]*(x**i)
            return temp
```

3. Generate noisy and clean data

- generate data using function.

```
In [4]: #the number and standard deviation of data
        num     = 1001
        std     = 5
        #generate data
        n       = np.random.rand(num)
        nn      = n - np.mean(n)
        x       = np.linspace(-10,10,num)
        #y1: clean data
        y1      = fun(x)
        #y2: noisy data
        y2      = y1 + nn * std
```

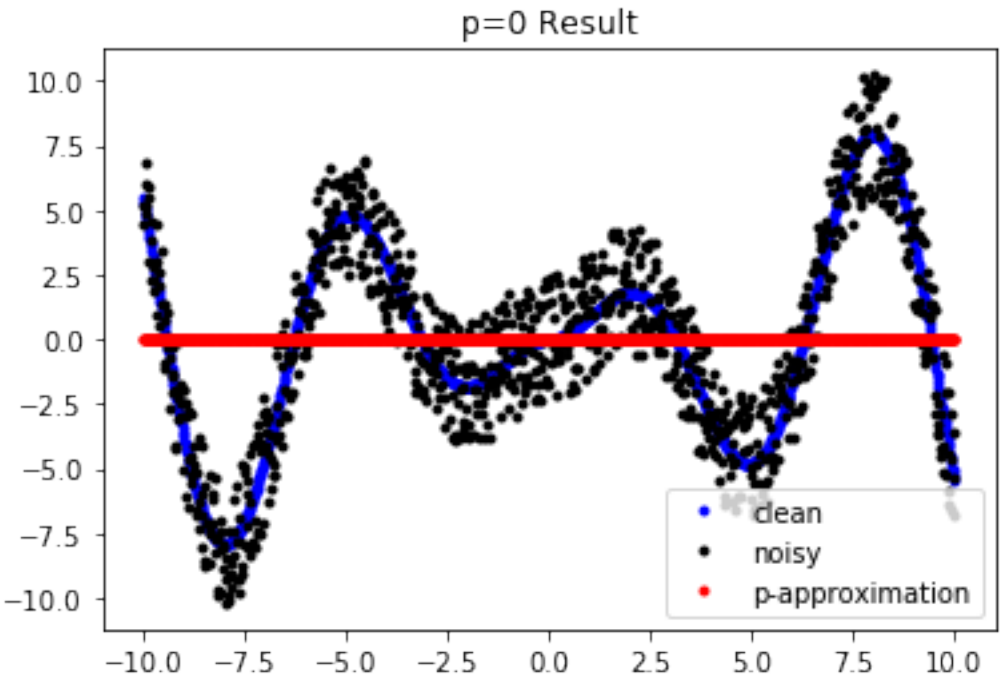
- set the max p and declare a container of sum of errors

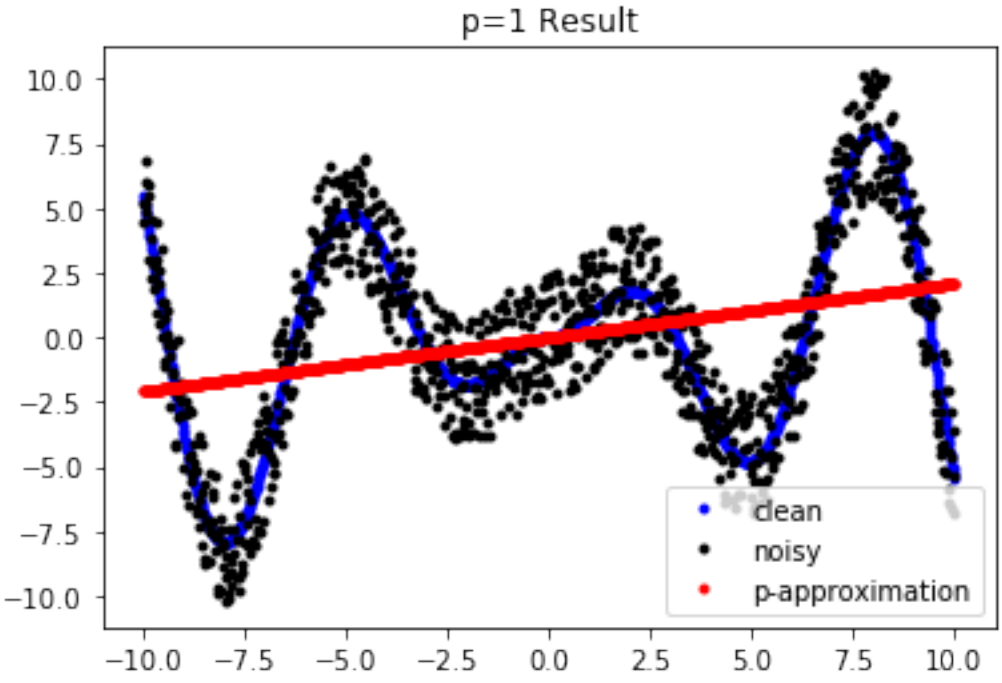
```
In [5]: p = 10
        result = []
```

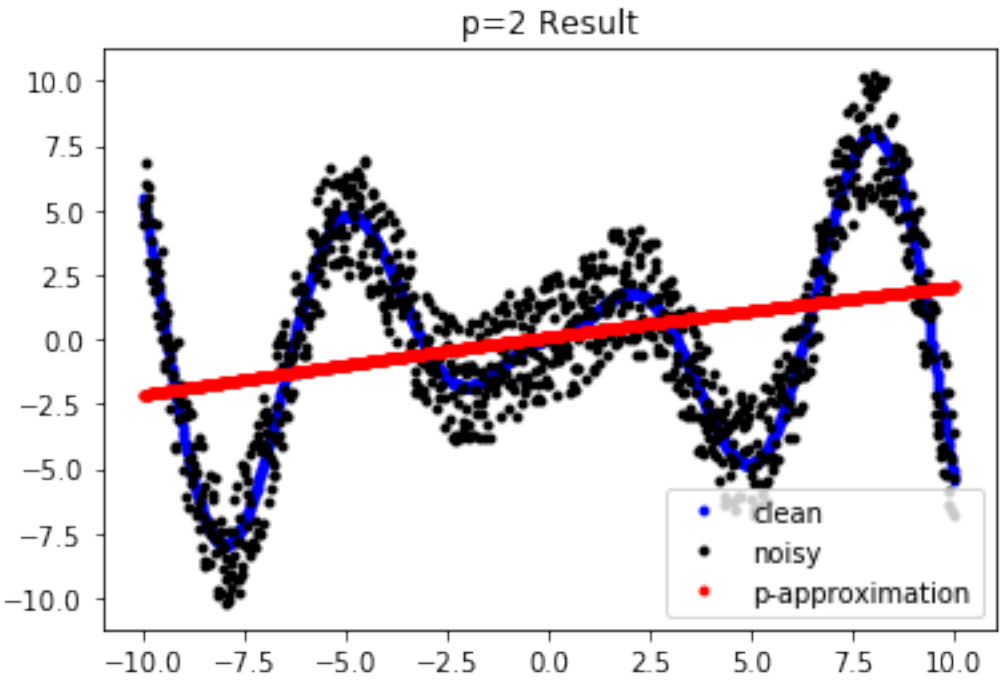
4. Compute coefficients per 'p' and Plot result

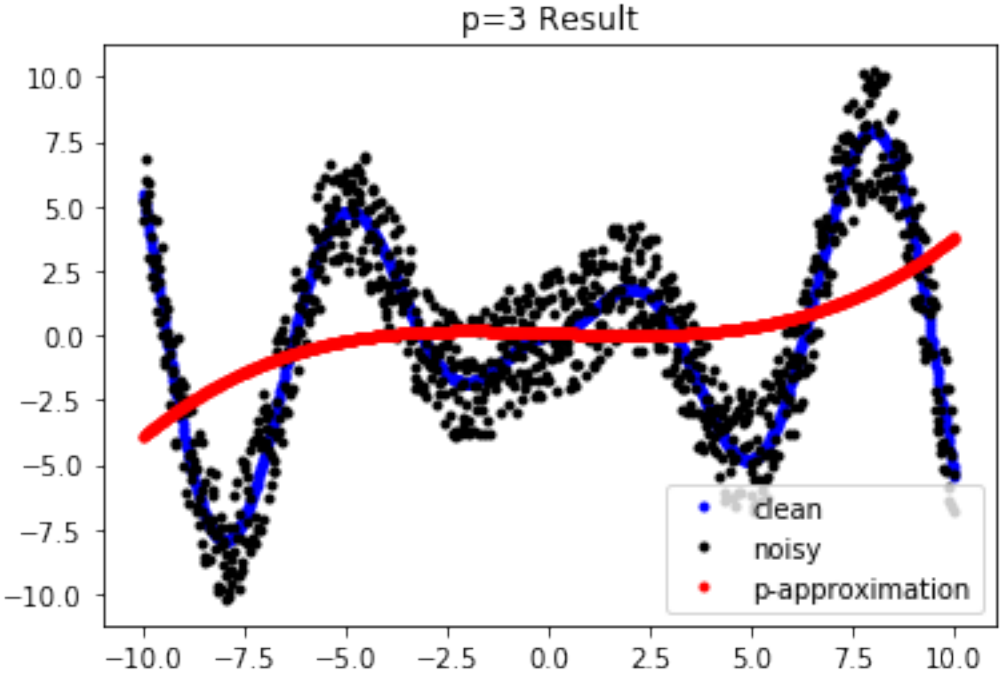
- loop p=0 to p=30, compute coefficients.
- can compute coefficients easily using the function 'numpy.polyfit(x,y,p)'.

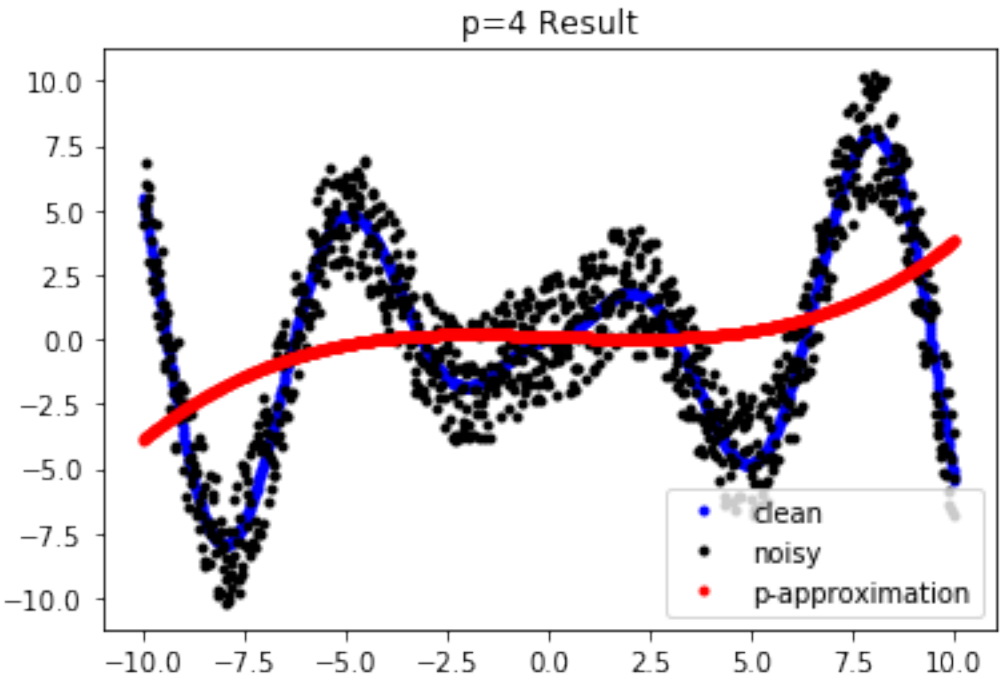
```
In [6]: for i in range(p):
            coEff = np.polyfit(x,y2,i)
            #y3: polynomial approximation data
            y3 = polyFun(x,coEff,i+1)

            plt.figure()
            plt.title("p=%d Result"%i)
            plt.plot(x, y1,'b.',label='clean')
            plt.plot(x, y2,'k.',label='noisy')
            plt.plot(x, y3,'r.',label='p-approximation')
            plt.legend(loc='lower right')
            plt.show()
            result.append(np.sum((y3-y2)**2))
```

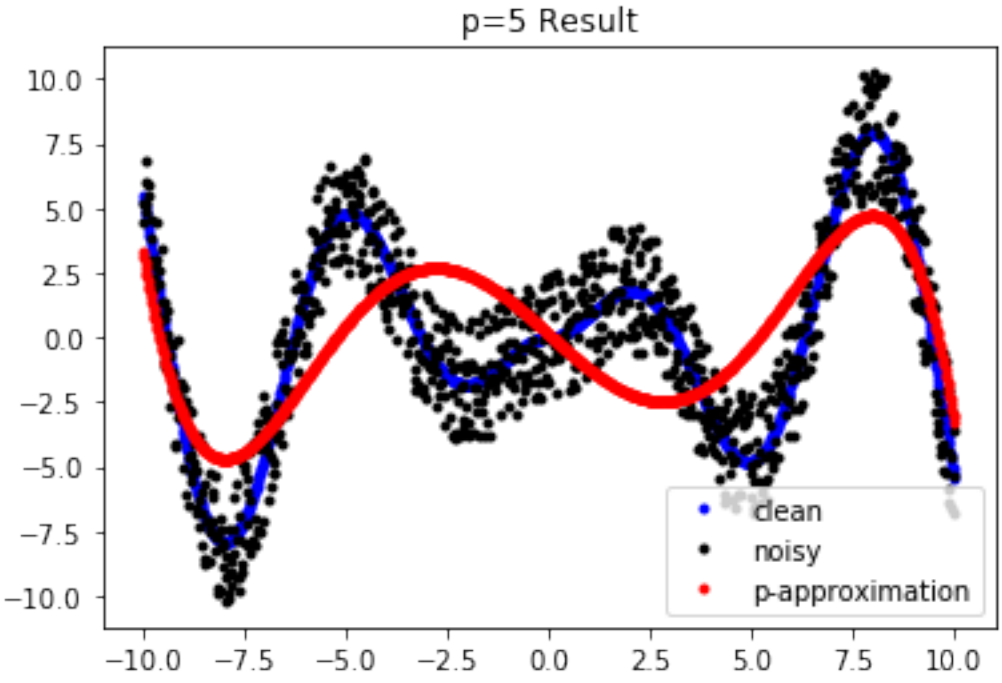

p=0 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation

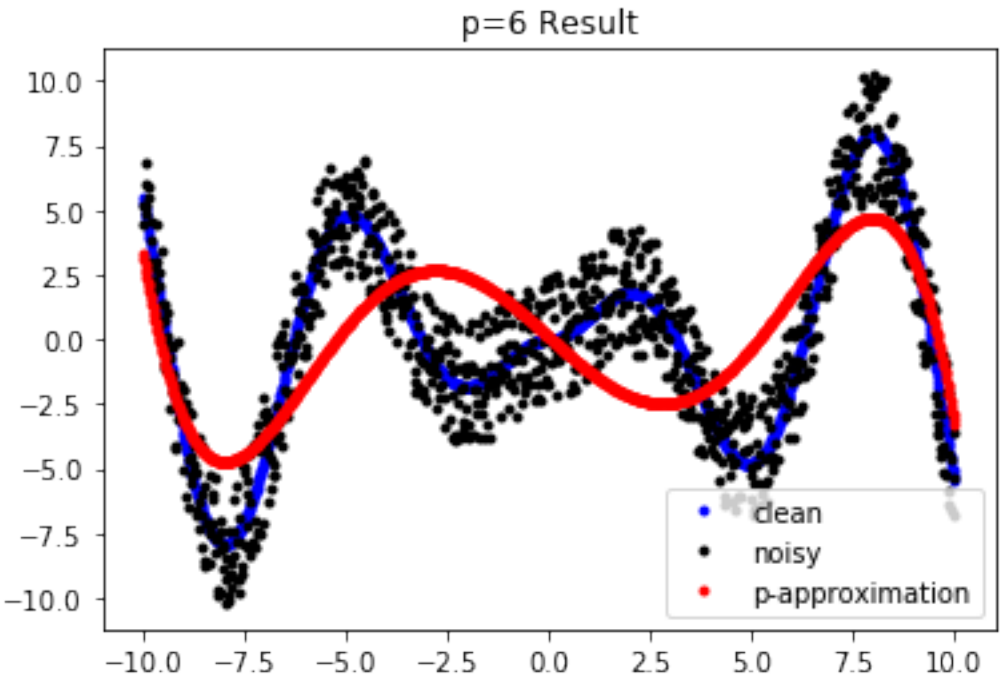

p=1 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation

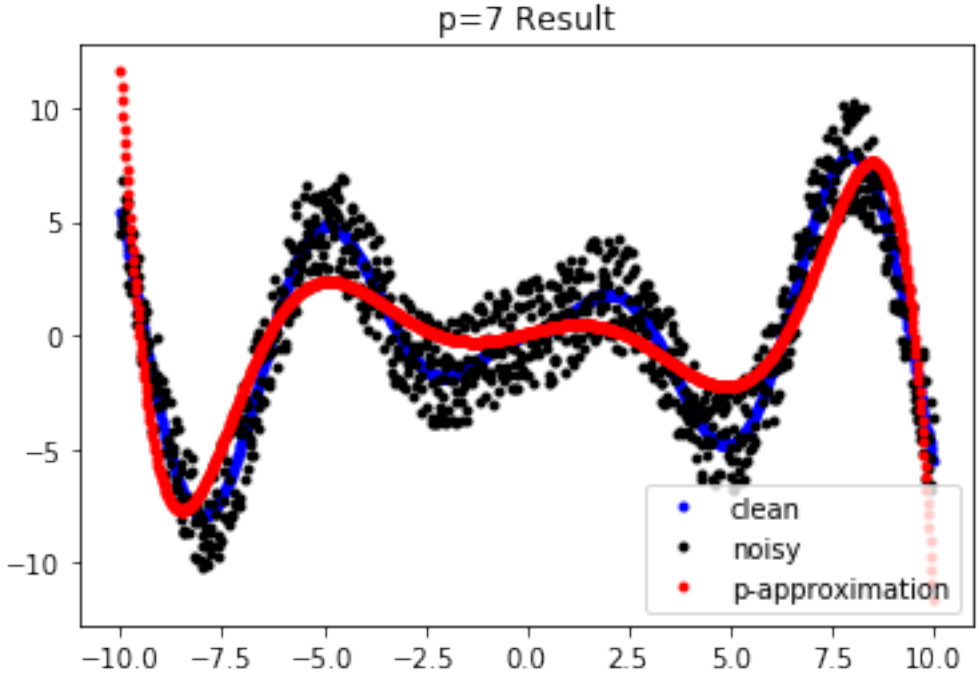

p=2 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation

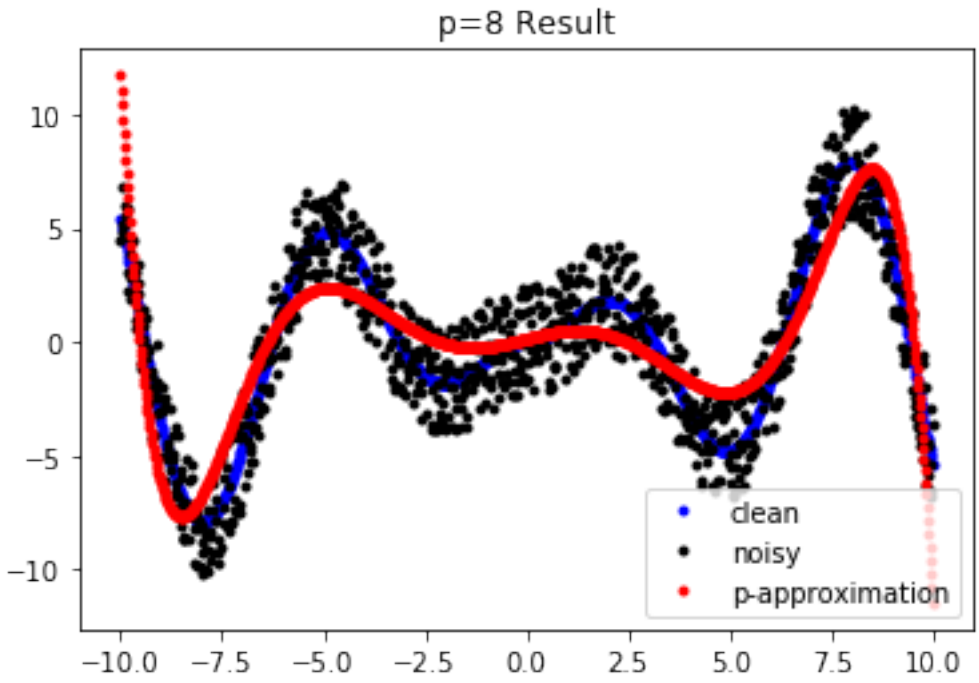

p=3 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation

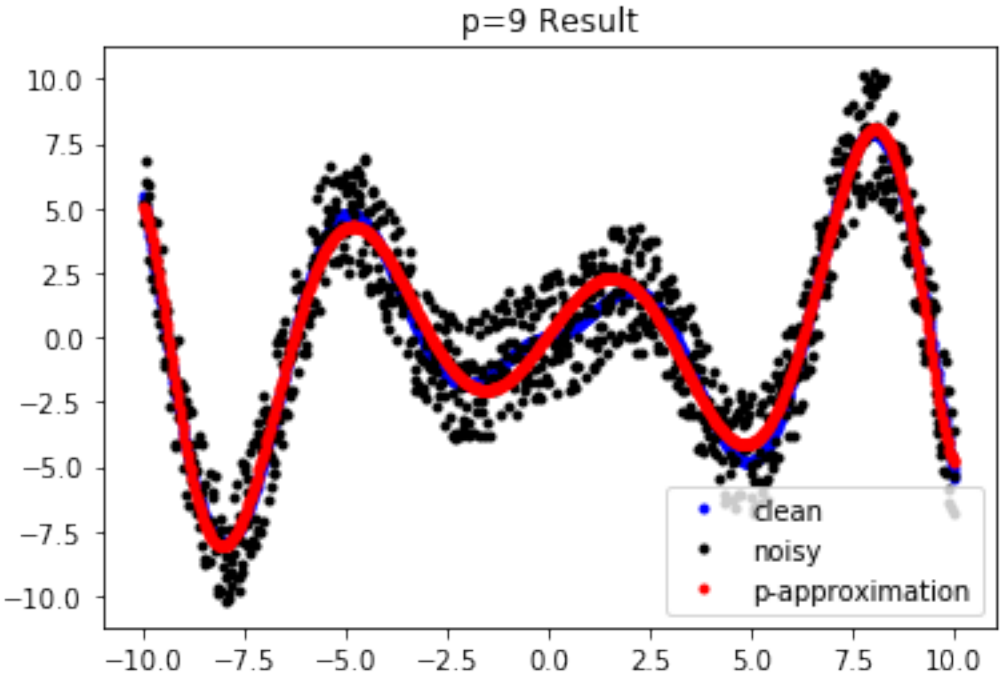

p=4 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation


p=5 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation


p=6 Result
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation

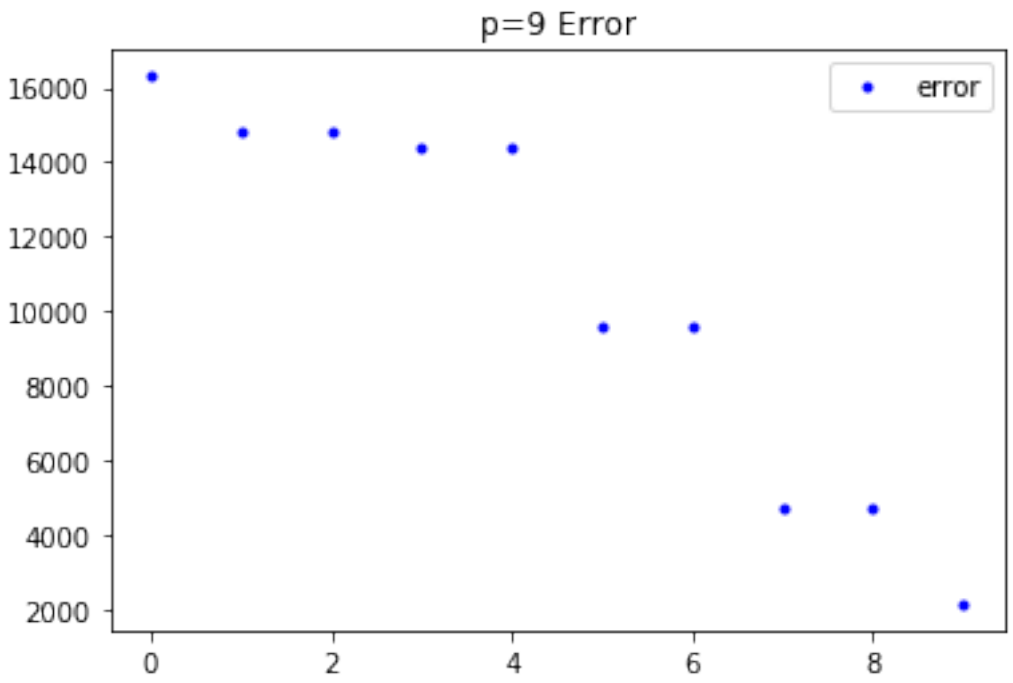

p=7 Result
10
5
0
−5
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
clean
noisy
p-approximation





- compute 'p=i'th error and plot the result

```
In [8]: plt.figure()
        xs=[]
        for k in range(p):
            xs.append(k)
        plt.title("p=%d Error"%i)
        plt.plot(xs,result,'b.',label='error')
        plt.legend(loc='upper right')
        plt.show()
```



- we can know that if 'p' gets bigger then sum gets lower.