科学技術計算

第7回レポート

30114150　KIMHYUNWOO

02.05

課題1

本kadai1.pyでは自作のbasis関数を用いてｋ次元のｊノード数を持つスプライン基底関数のグラフを求める。Basis関数の基本的な仕組みは講義資料の理論と同じで、０次元の場合は各ノードの区間内の値だけ１で他の区間では０になることから初めて、次元が増えるほど１つ下の次元の基底関数を用いて新たな基底関数を構成する形である。引数としては次元の数と適用するノード区間のナンバー、データの番号、データセット、ノード区間セットが与えられる。

```
def basis(j,k,t,Data,sector):
```

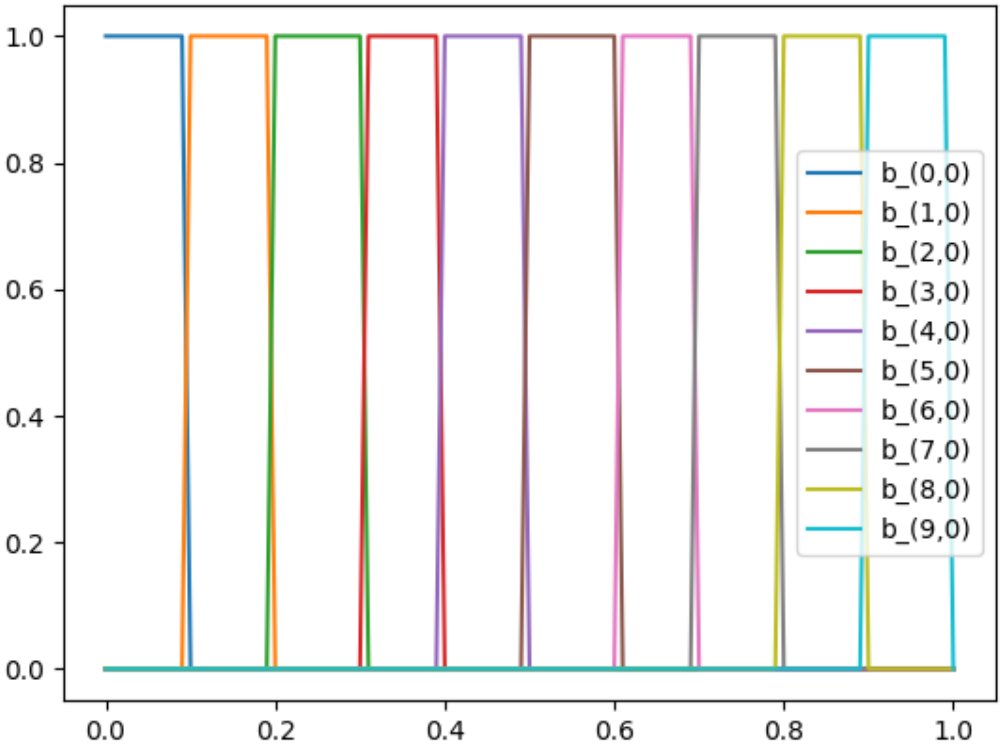
続いて、データセットやノード区間を条件に合わせて設定するためにlinspace関数を用いて区間を設定する。

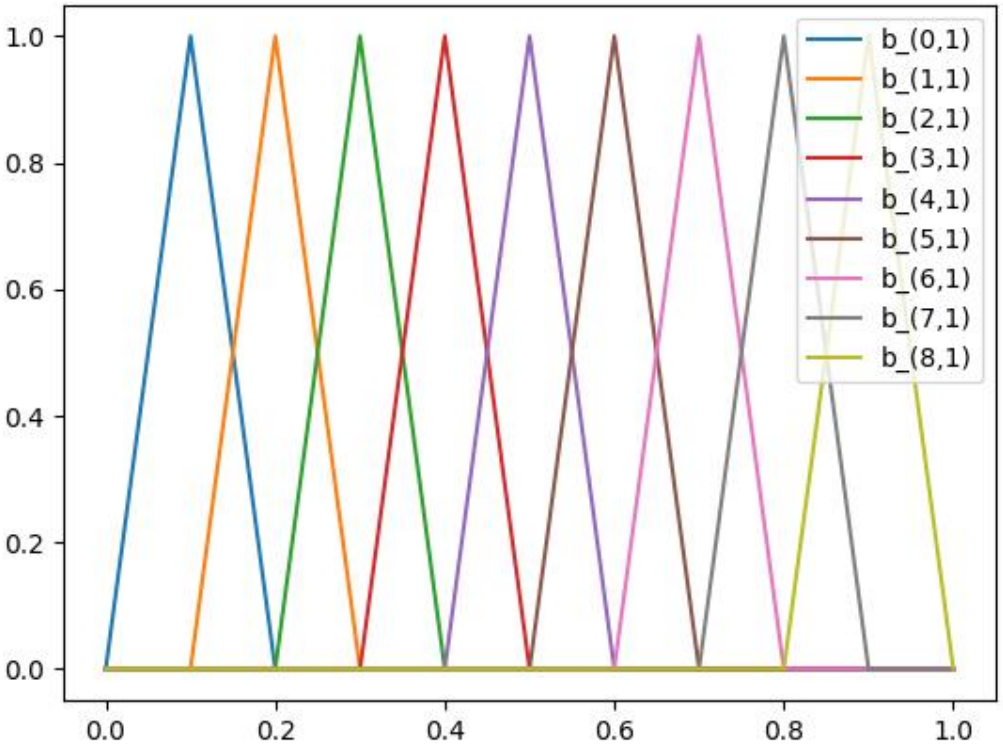
```
Nodesector = np.linspace(LowerBound,UpperBound,NodeNum,endpoint='True')
Data = np.linspace(LowerBound,UpperBound,101,endpoint='True')
DataNum = np.shape(Data)[0]
```

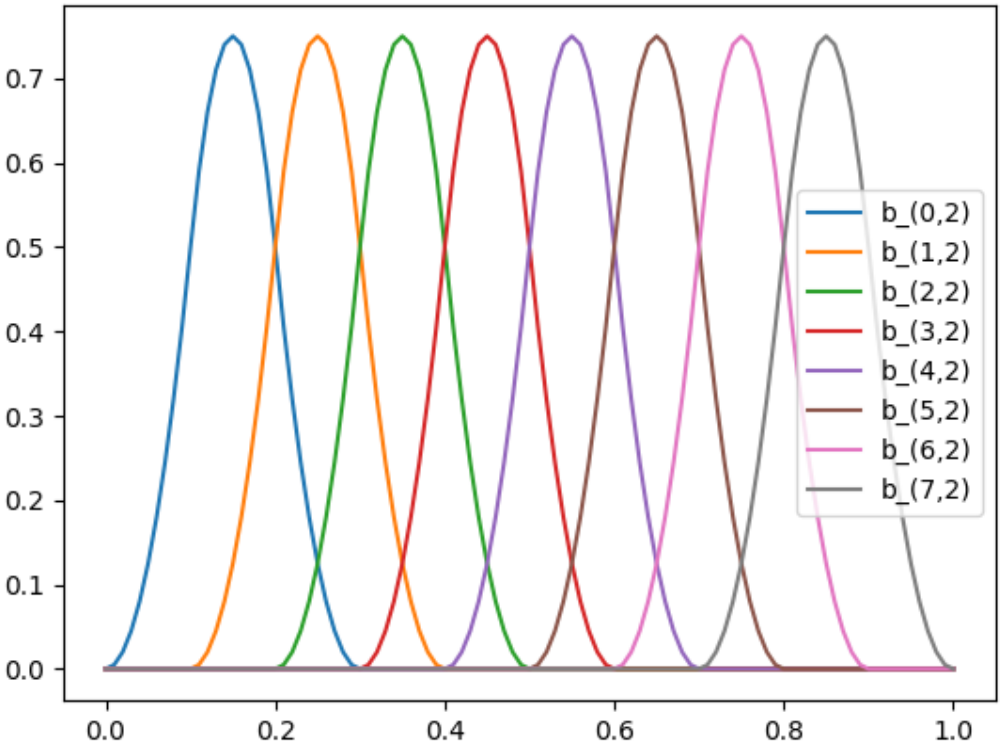
最後は全てのデータ区間に対して値を求めてからグラフに移す過程で、次元をループしながら最大ノード区間数を調整してからデータセットとノード区間をループしながら値を求める。

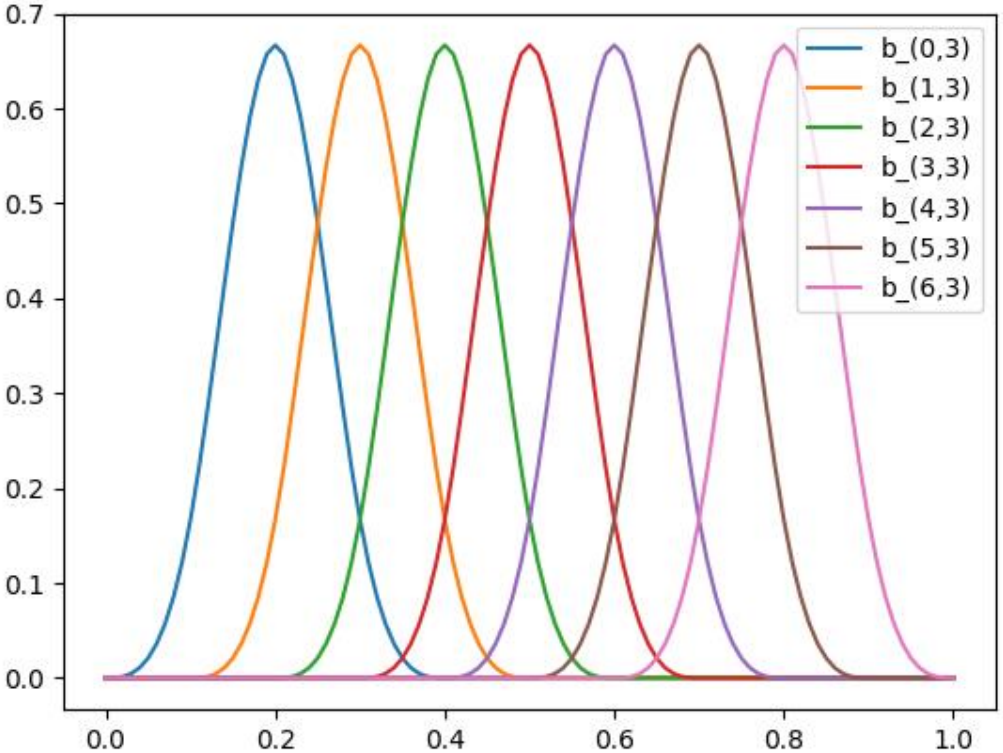
```
for dim in range(Dimension+1):
    j = NodeNum - dim - 2
    thislabel = ''
    for iter_j in range(j+1):
        Y = np.zeros(DataNum)
        for iter_data in range(DataNum):
            Y[iter_data] += basis(iter_j,dim,iter_data,Data,Nodesector)
        thislabel = 'b_('+str(iter_j)+','+str(dim)+')'
        plt.plot(Data,Y,label=thislabel)
        plt.legend()
    plt.savefig(thislabel)
    plt.show()
```

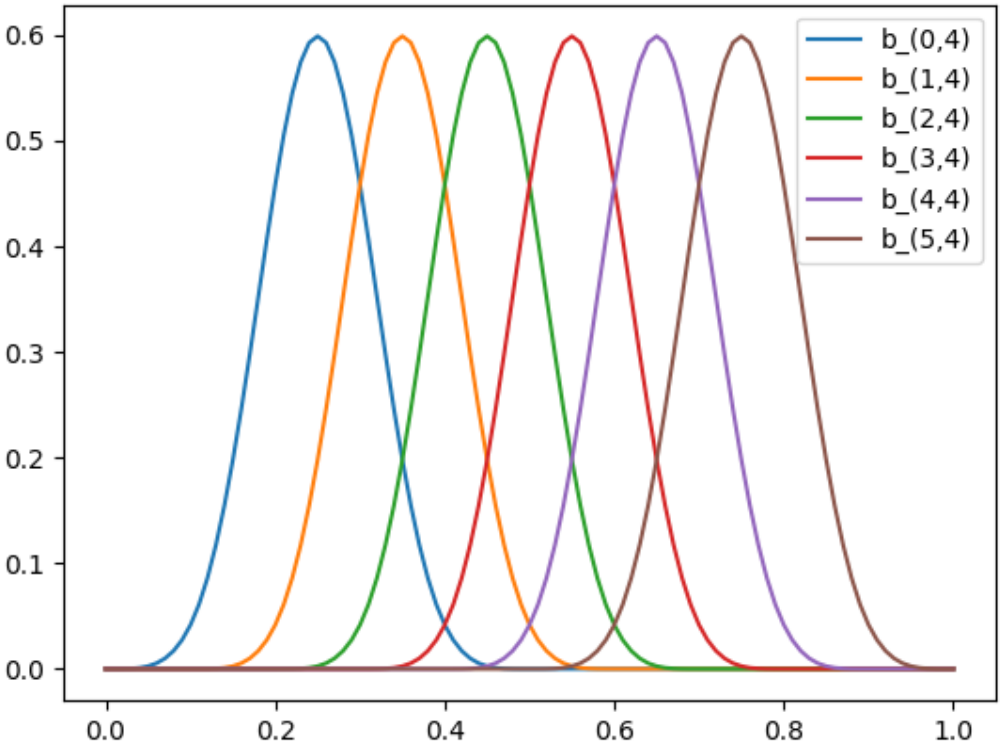
以上の過程から得られた結果は続く５つのグラフであり、講義資料の図と比較してみた所、大きな差が見えないことが分かる。


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
b_(0,0)
b_(1,0)
b_(2,0)
b_(3,0)
b_(4,0)
b_(5,0)
b_(6,0)
b_(7,0)
b_(8,0)
b_(9,0)


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
b_(0,1)
b_(1,1)
b_(2,1)
b_(3,1)
b_(4,1)
b_(5,1)
b_(6,1)
b_(7,1)
b_(8,1)


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0
b_(0,2)
b_(1,2)
b_(2,2)
b_(3,2)
b_(4,2)
b_(5,2)
b_(6,2)
b_(7,2)


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0
b_(0,3)
b_(1,3)
b_(2,3)
b_(3,3)
b_(4,3)
b_(5,3)
b_(6,3)


b_(0,4)
b_(1,4)
b_(2,4)
b_(3,4)
b_(4,4)
b_(5,4)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0

課題2

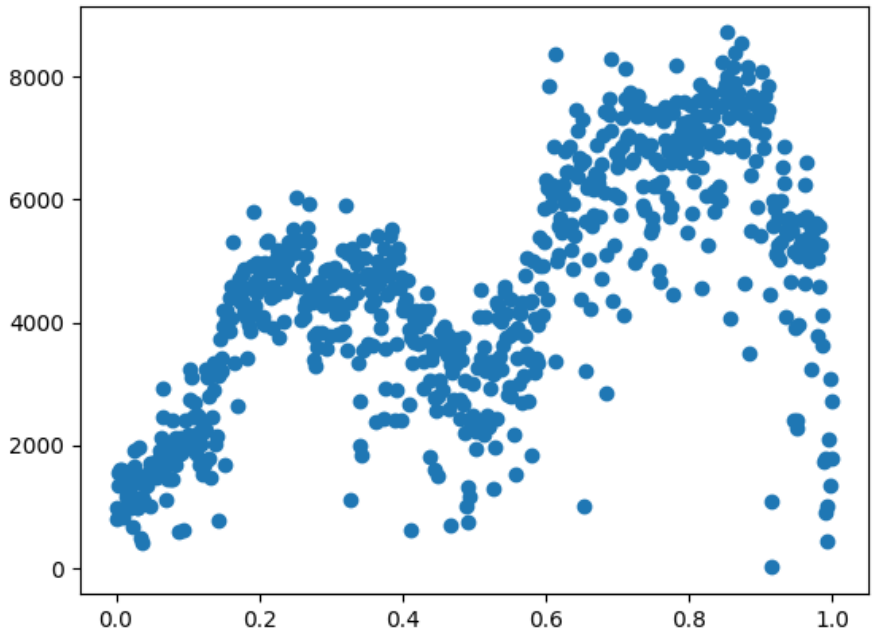
本kadai2.pyでは 4 回の課題で使用したbikesharing.csvファイルを用いて日別利用者の推移を推定する。基本的な仕組みは課題 1 と同じでb-spline基底関数を基本に、各データ別の最適な重さを求めるための最小二乗法で求めた重さからグラフを描くことである。

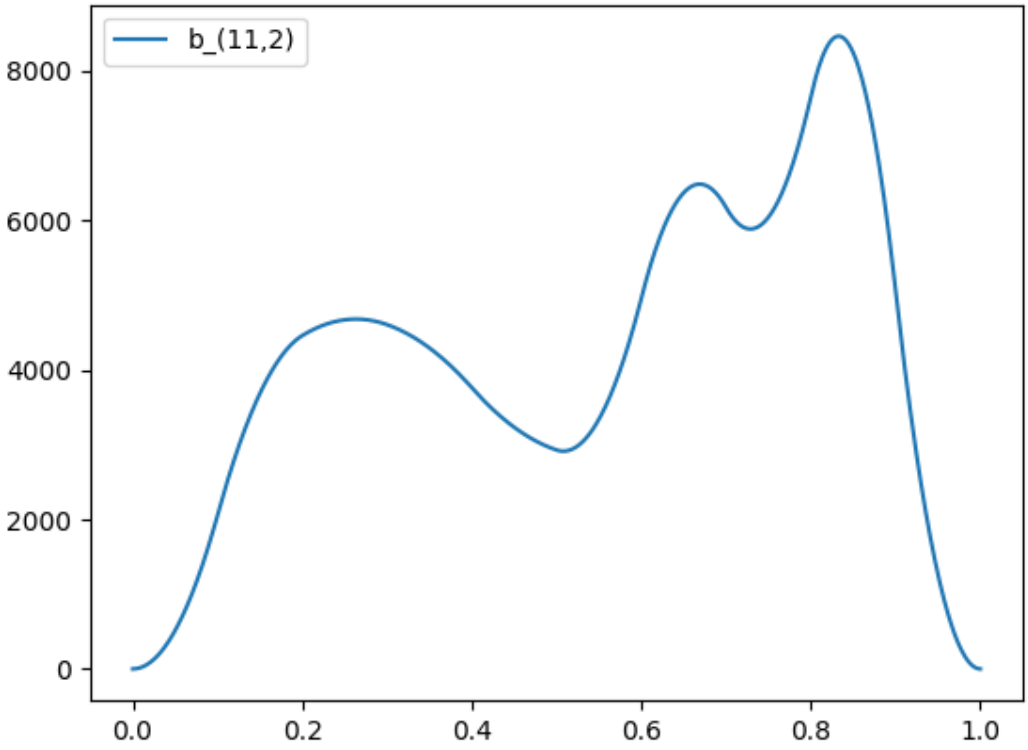
課題 1 と同じく上下限からの区間を求めつつデータの範囲を定める一方、最小二乗法に使うためのY軸データや便宜上x軸データを正規化する過程を追加している。

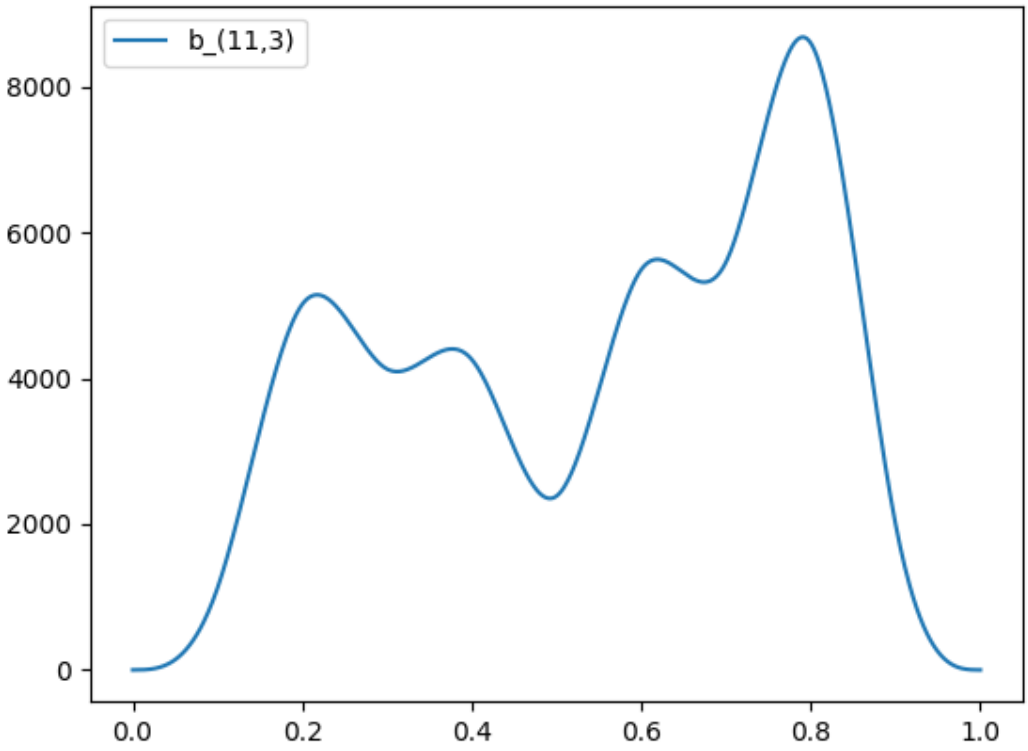
基底関数を求める方法は課題 1 と同じであるが、基底関数を求めてからは以下の式で加重値（最小二乗解）を求める。ここでAは各 x データから求めた基底関数行列であり、Y は元の値である利用者数ベクトルである。また、解を求める式は最小二乗法の表現方$(X^TX)\beta = X^TY$から誘導されたものである。

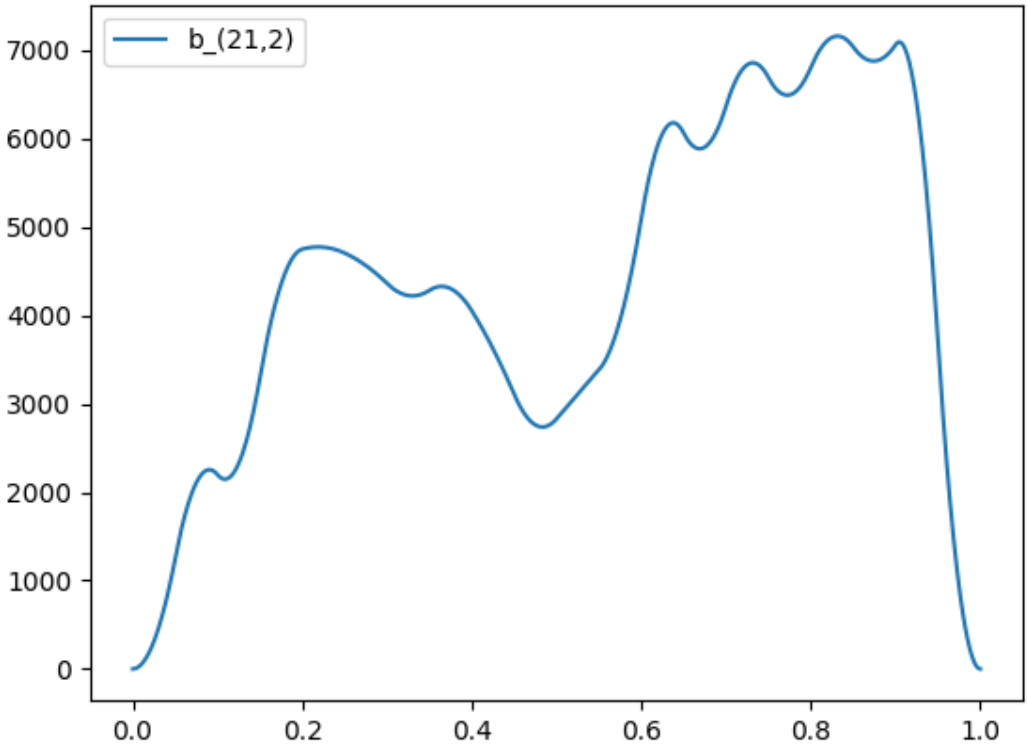
result = np.linalg.pinv(np.transpose(A)@A)@np.transpose(A)@Y

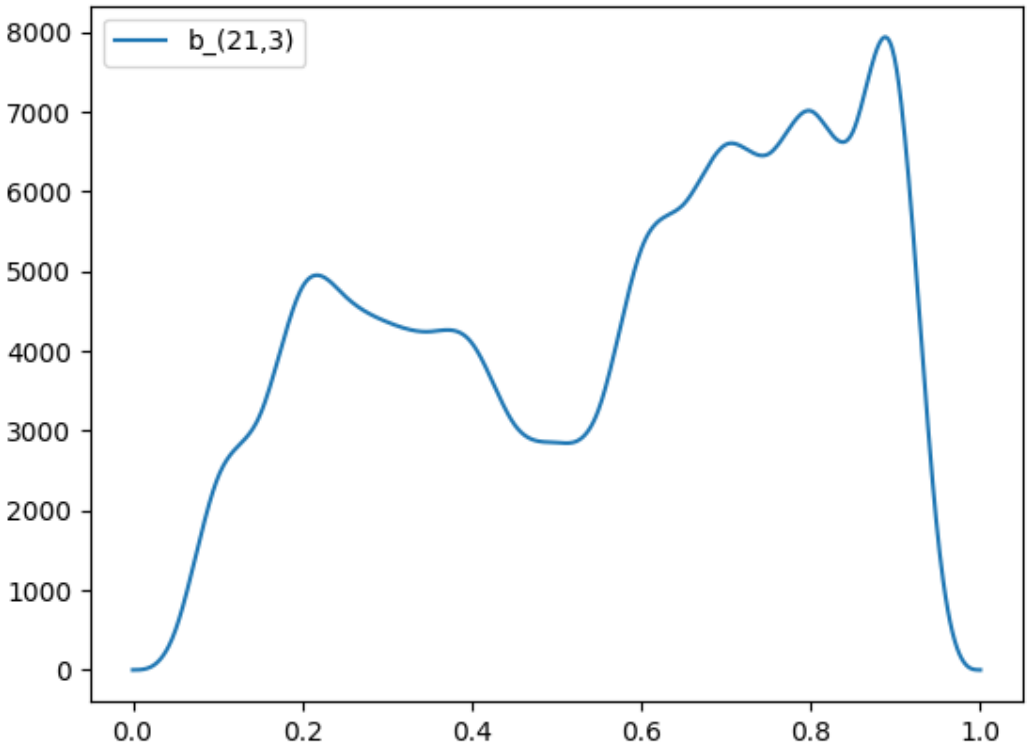
このように求めた加重値をもって基底関数行列との行列掛け算で推定した推移グラフと元のscatter関数でプロットしたグラフとの形態を比較して見た所、区間数と次元が上がるたびに推定グラフが精巧になることが分かる。




b_(11,2)
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0


b_(11,3)
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0


b_(21,2)
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0


b_(21,3)
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

課題3

本kadai3.pyでは課題２と同じくb-spline関数による近似を行う。仕組みは課題２と同じで、今回はｘデータをcosとし、ｙデータをsinとするデータ集合から得られるグラフを推定して、各データのラジアン範囲は０から２πまでのθである。

```
LowerBound = 0
UpperBound = 2*pi
theta = np.arange(LowerBound,UpperBound,pi/50)
```

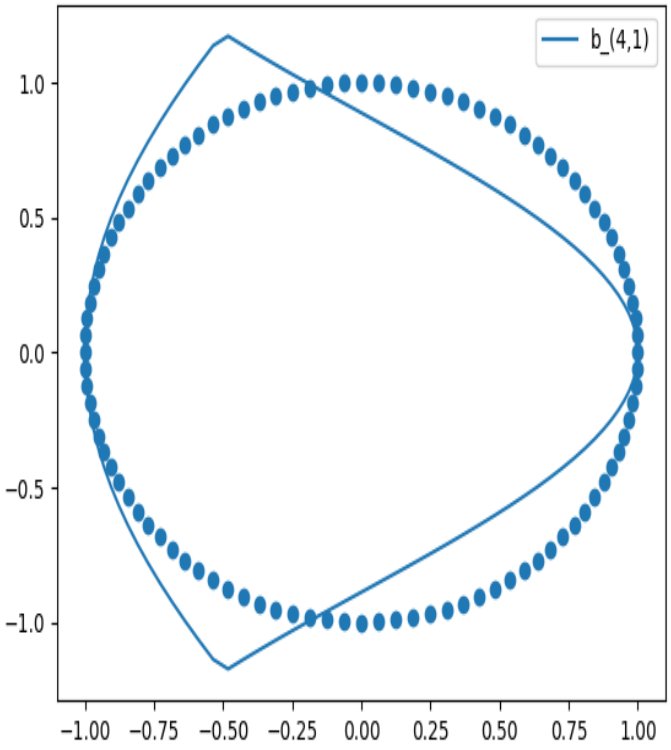
但し、基底関数行列はｘデータであるcosxではなくθであるが、理由としてはｙ軸データであるsinyはcosxによる値ではなくθで定められる値であるため,y値を推定するにはθから基底関数行列を求める必要がある。

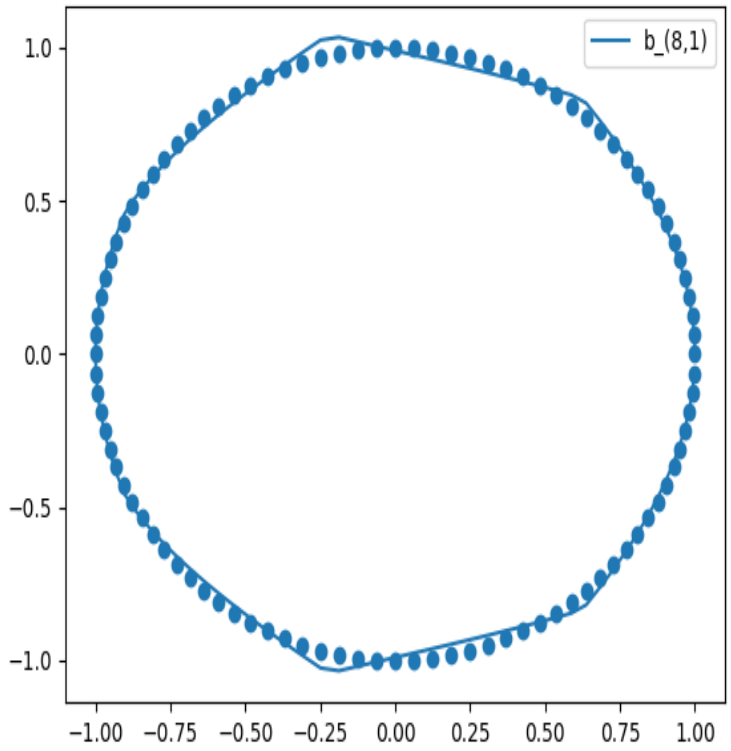
```
A[iter_t, iter_j]= basis(iter_j,dim,iter_t,theta,Nodesector)
```

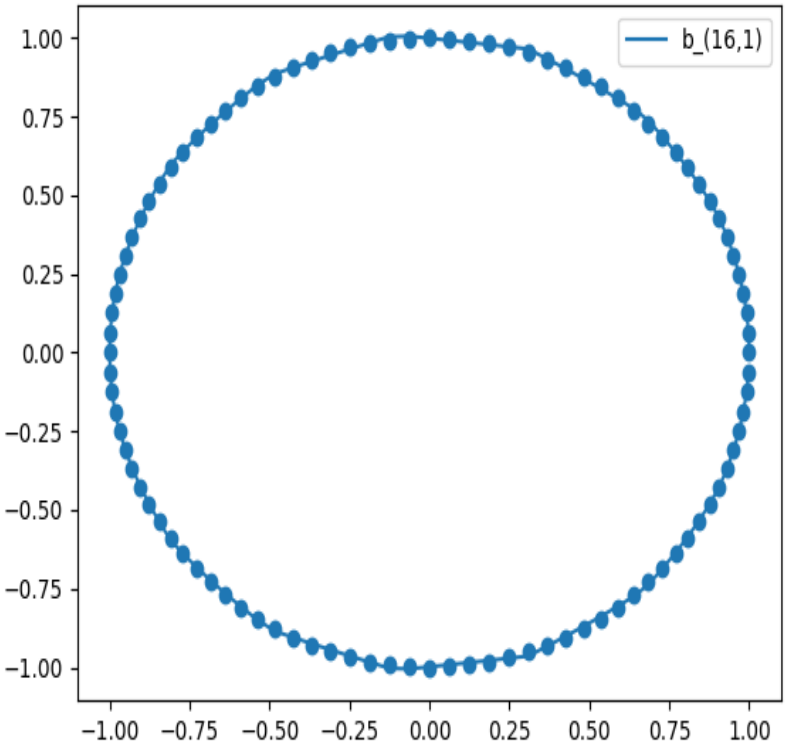
以後の最小二乗法ではYにsinyを代入することで加重値を求め、基底関数行列との行列積でグラフをフロットさせる。

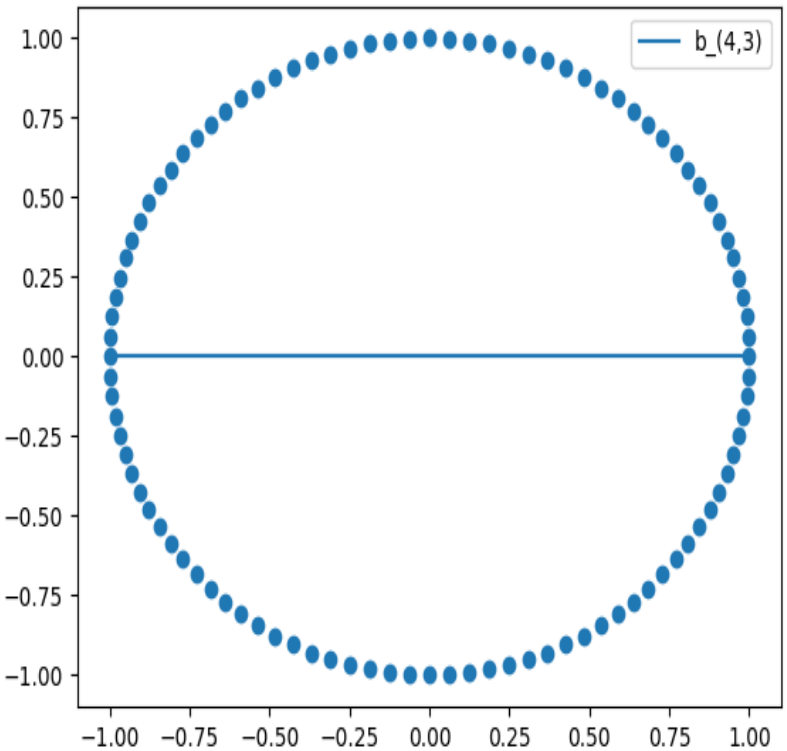
```
SinData = np.linalg.pinv(np.transpose(A)@A)@np.transpose(A)@siny
…
plt.plot(cosx,A@SinData,label=thislabel)
```

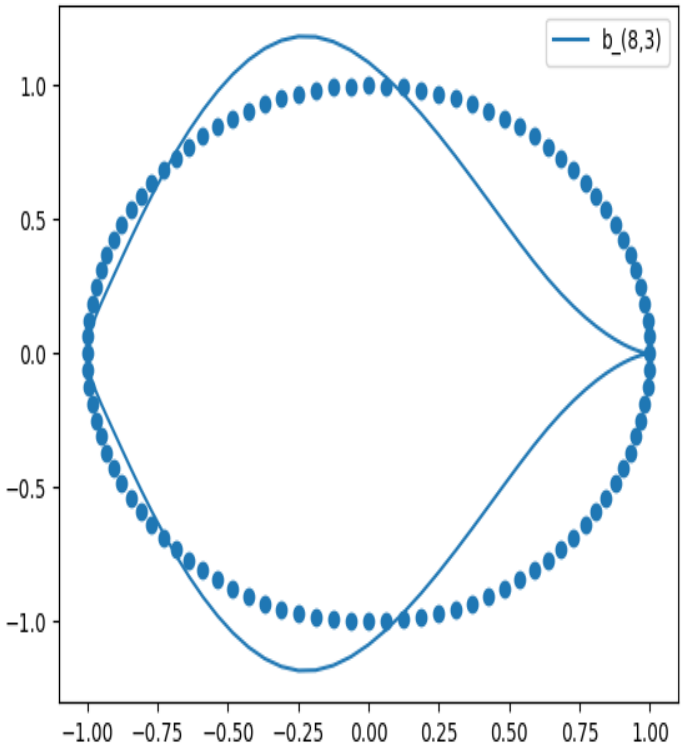
以上の過程から得られたcosx-A@SinDataグラフの結果を元のcosx-sinyグラフと比較して見た所、１次元では８次数、１６次数の場合ほぼ元のグラフに沿っていることが分かるが、３次元では次数が上がっても１次元ほど円に沿っていないことが確認できた。

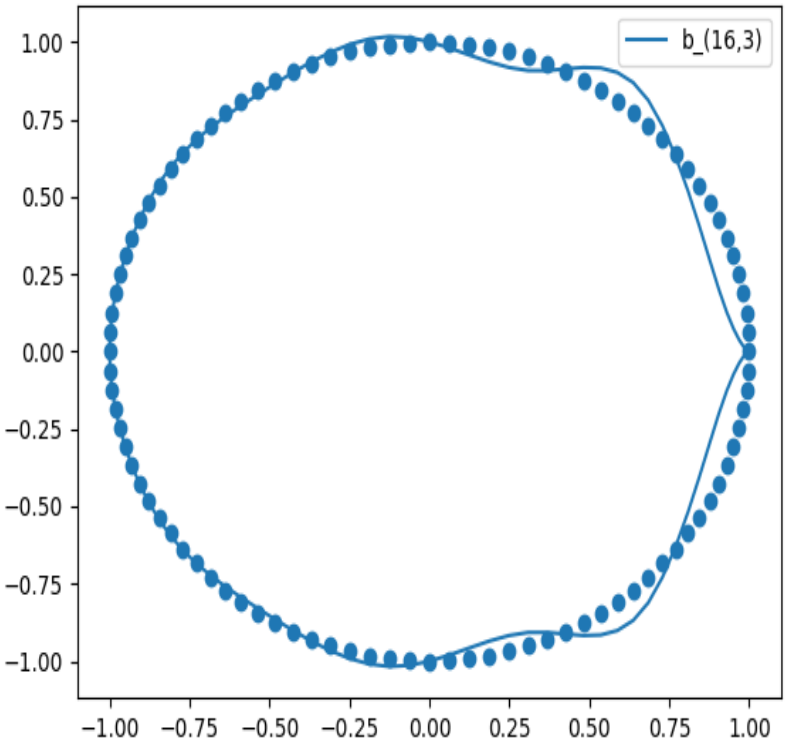

b_(4,1)
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


b_(8,1)
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


b_(16,1)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


b_(4,3)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


b_(8,3)
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


b_(16,3)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

考察と感想

今回はb-spline基底関数とb-spline関数による近似を練習した。５回の講義で練習したスプライン関数の補間と同じくスプラインという概念を使用しているが、ｎ次スプラインがサンプル点での微分値の同化及びサンプル点を必ず通じる性質を持つ反面、b-spline補間の場合基底関数と設定した制御点の積を元のデータで最小二乗法を行うことで近似していることが異なる。一般のｎ次スプラインがデータの変更に影響を受けやすく一部の変更でもグラフの様子が大きく変わる可能性が高いが、b-spline近似の場合基底関数が区間別に分けられているため同じ変更でもグラフの様子が余り変わりない、より自由なグラフの調整ができる長所がある。但し、課題３で確認できたように線形回帰の場合と同じ問題である過適合の問題があるため、適切な次数や区間を設定する必要があると考えられる。