Exercise15

December 16, 2022

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

1 Exercise 15

1.1 a)

Implementing method to generate response matrix A.

[2]:
```python
eps = 0.23


def A(n):
    temp = np.array(n*[1-2*eps])
    temp[0] = 1-eps
    temp[-1] = 1-eps

    return np.diag((n-1)*[eps],1) + np.diag(temp) + np.diag((n-1)*[eps],-1)

A(5)
```

[2]:
```
array([[0.77, 0.23, 0.  , 0.  , 0.  ],
       [0.23, 0.54, 0.23, 0.  , 0.  ],
       [0.  , 0.23, 0.54, 0.23, 0.  ],
       [0.  , 0.  , 0.23, 0.54, 0.23],
       [0.  , 0.  , 0.  , 0.23, 0.77]])
```

Die Matrix A beschreibt die Fluktuationen der Bins einer Messreihe in die anderen Bins. Also eine Art 'Verschmieren' der Bins in die benachbarten Bins.

1.2 b)

[3]:
```python
f = np.array([193, 485, 664, 763, 804, 805, 779, 736, 684, 626, 566, 508, 452,␣
 ↪400, 351, 308, 268, 233, 202, 173])

g = A(20)@f.T

print('g = ', g)
```

```
g =  [260.16 459.01 645.6  749.66 794.8  798.79 775.09 733.93 682.62 625.54
 566.46 508.46 452.92 400.69 352.38 308.69 269.15 233.92 202.46 179.67]
```

[4]:
```python
rng = np.random.default_rng(69*69)
g_measured = np.zeros(20)

for i in range(len(g)):
    g_measured[i] = rng.poisson(g[i])

print('g_measured = ', g_measured)
```

```
g_measured =  [236. 434. 613. 752. 834. 772. 813. 698. 674. 639. 551. 503. 432.
380.
 338. 311. 244. 240. 209. 176.]
```

1.3 c)

Die Gleichung $g = Af$ wird zu $c = Db$ mit $b = U^{-1}f$ und $c = U^{-1}g$ und ermöglicht uns, f einfach über Eigenwertmultiplikation zu bestimmen.

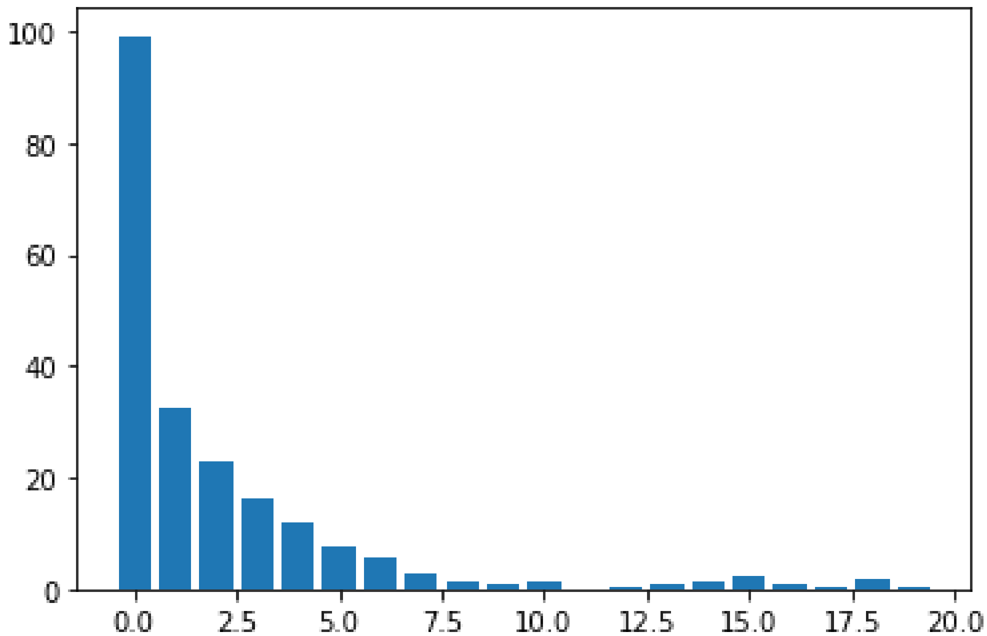
[5]:
```python
A_20 = A(20)
D_values, U = np.linalg.eig(A_20) # Eigenwerte & -vektoren

#U = np.around(U, 2)
idx = D_values.argsort()[::-1]
D_values = D_values[idx]
U = U[:,idx]

D = np.diag(D_values)

D_inv = np.linalg.inv(D)
U_inv = np.linalg.inv(U)


A_test = np.around(U@D@U_inv,6)
#A_test
```

1.4 d)

[6]:
```python
c = U_inv@g_measured
b = D_inv@c
#Varianz von Poissonverteilung ist einfach nur N, also einfach Diagonalmatrix␣
 ↪von g_measured
V = np.diag(g_measured)

B = D_inv@U_inv

cov = B@V@B.T
```

```
b_norm = np.zeros(20)

for i in range(len(b_norm)):
    b_norm[i] = b[i]/np.sqrt(cov[i][i])

i = np.linspace(0,19,20)
fig, ax = plt.subplots()
ax.bar(i,np.abs(b_norm)) # Betrag geplottet, weil schoener
```

[6]: <BarContainer object of 20 artists>

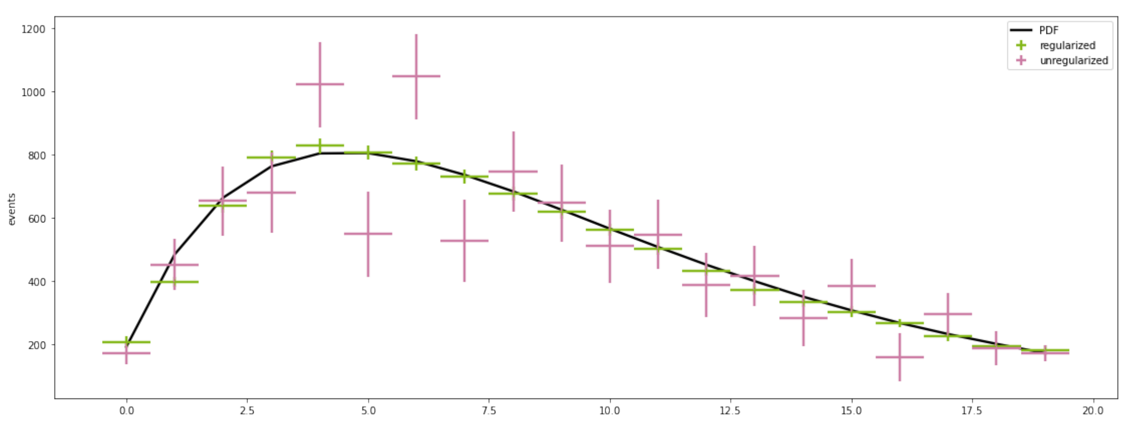


Koeffizienten kleiner als 1 werden als Oszillationen angenommen, die keine Informationen beinhalten. Daher wird der cutoff-Index auf i=8 gesetzt.

1.5 e)

[7]:
```
cut = 10
b_cut = np.append(b[:cut], np.zeros(20-cut))

f_unreg = U@b
f_reg = U@b_cut



errors = np.sqrt(np.diag(U@cov@U.T))
```

```python
# Covarianz Matrix der regularisierten Lösung: obere Blockmatrix wie alte cov, alles andere 0
cov_reg = cov[:cut,:cut]
cov_reg = np.block([
    [cov_reg, np.zeros((cut,20-cut))],
    [np.zeros((20-cut, 20))]
])

#D_reg = np.diag(np.concatenate((np.diag(D)[:cut], np.zeros(20-cut))))
#cov_reg = (np.linalg.inv(D_reg)@U_inv@V@(np.linalg.inv(D_reg)@U_inv).T) # geht nicht weil man D nicht invertieren kann

errors_reg = np.sqrt(np.diag(U@cov_reg@U.T))


fig, ax = plt.subplots(figsize=(19,7))
ax.plot(i,f, label='PDF', c='black', linewidth=2.4)
#ax.errorbar(i,f, xerr=0.5, yerr = 0, fmt='+', label ='PDF', c='black', elinewidth=2.4)
ax.errorbar(i,f_reg, xerr=0.5, yerr = errors_reg, fmt='+', label ='regularized', c='#84b819', elinewidth=2.4)
ax.errorbar(i,f_unreg, xerr = 0.5, yerr = errors, fmt='+', label='unregularized', c='#cc7ca4', elinewidth=2.4)
ax.set_xlabel('x')
ax.set_ylabel('events')
leg = ax.legend()
```



Ohne Regularisierung oszilliert die Lösung stark um die eigentliche Verteilung, mit Regularisierung wird dies unterbunden. Des Weiteren sind die Standardabweichungen der regularisierten Lösung deutlich kleiner.