2023-07-09_PracticalRemarks_02_Export

July 11, 2023

1 Exporting figures to PDF

Far from being a complete guide, this file very briefly showcases one example of such an export and the option to render figure elements with LaTeX math formulas.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

%matplotlib inline
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
```

1.1 Example dataset

[2]:
```
data=pd.read_csv("./PET.csv",comment="#")
```

[3]:
```
data
```

[3]:

	experimentIndex	beta	activity	locError	stdError	timeSteps
0	0	0.08832	2.8	12.503711	0.665868	64
1	0	0.19200	2.8	10.163309	0.544522	64
2	0	0.42240	2.8	8.394741	0.672629	64
3	0	0.88320	2.8	8.058029	0.694879	64
4	0	1.92000	2.8	10.534795	0.581206	64
5	1	0.01920	3.5	14.415172	0.606430	64
6	1	0.04224	3.5	11.954669	0.570749	64
7	1	0.08832	3.5	9.795868	0.514492	64
8	1	0.19200	3.5	7.931059	0.467283	64
9	1	0.42240	3.5	6.595897	0.402097	64
10	1	0.88320	3.5	6.370780	0.530820	64
11	1	1.92000	3.5	8.312561	0.696537	64
12	1	4.22400	3.5	12.307082	0.227363	64
13	2	0.01920	4.3	11.098006	0.620813	64
14	2	0.04224	4.3	9.153340	0.622166	64

15	2	0.08832	4.3	7.555648	0.608666	64
16	2	0.19200	4.3	6.218279	0.527185	64
17	2	0.42240	4.3	5.391804	0.432128	64
18	2	0.88320	4.3	5.308332	0.454125	64
19	2	1.92000	4.3	7.081657	0.469018	64
20	2	4.22400	4.3	11.148281	0.193397	64
21	3	0.01920	5.7	8.359967	0.437061	64
22	3	0.04224	5.7	7.073810	0.311978	64
23	3	0.08832	5.7	5.948800	0.194625	64
24	3	0.19200	5.7	4.982759	0.105157	64
25	3	0.42240	5.7	4.388853	0.058897	64
26	3	0.88320	5.7	4.348137	0.051738	64
27	3	1.92000	5.7	5.502549	0.168485	64
28	3	4.22400	5.7	9.781202	0.260838	64
29	4	0.01920	8.3	5.521314	0.200028	64
30	4	0.04224	8.3	4.826293	0.177562	64
31	4	0.08832	8.3	4.270250	0.129589	64
32	4	0.19200	8.3	3.837957	0.088597	64
33	4	0.42240	8.3	3.591359	0.077132	64
34	4	0.88320	8.3	3.644718	0.105985	64
35	4	1.92000	8.3	4.480520	0.143204	64
36	4	4.22400	8.3	8.046639	0.135632	64
37	5	0.01920	8.3	7.274472	0.141576	128
38	5	0.04224	8.3	5.676809	0.169815	128
39	5	0.08832	8.3	4.568200	0.149747	128
40	5	0.19200	8.3	3.762851	0.107292	128
41	5	0.42240	8.3	3.259325	0.080355	128
42	5	0.88320	8.3	3.137426	0.096064	128
43	5	1.92000	8.3	3.592089	0.212666	128
44	5	4.22400	8.3	8.955321	0.290763	128

[4]:
```
data.columns
```

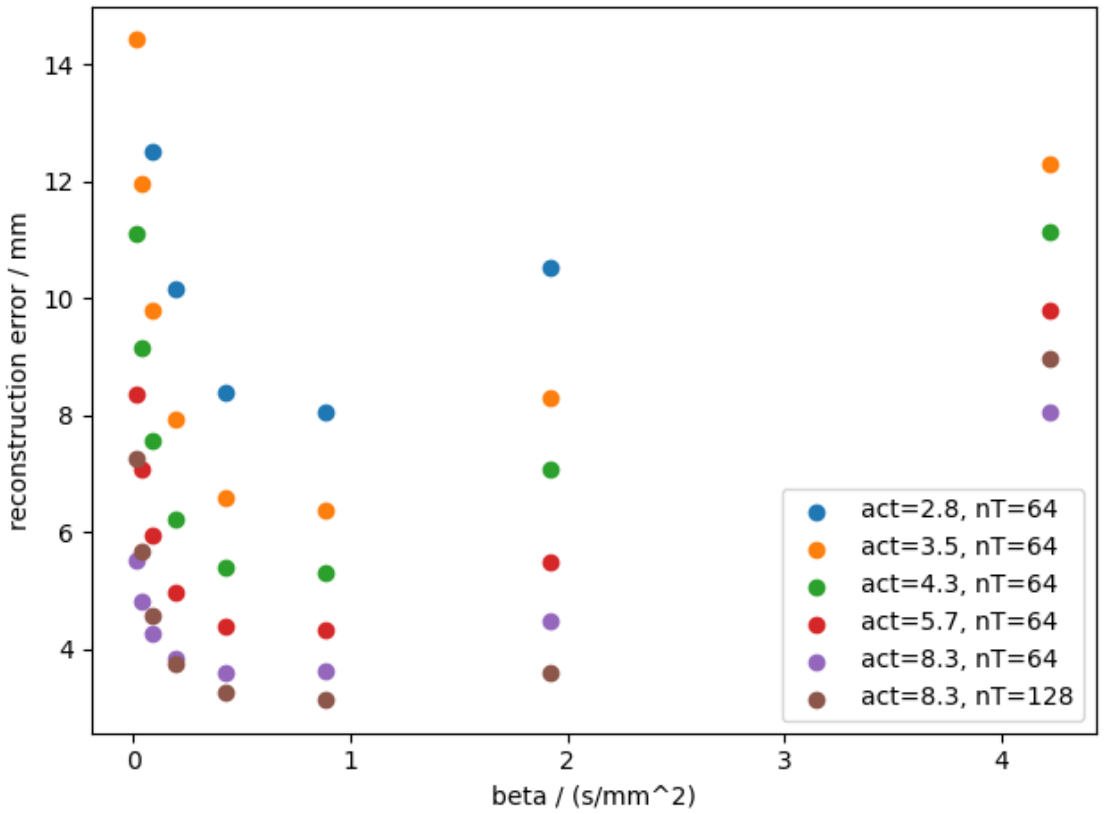
[4]:
```
Index(['experimentIndex', 'beta', 'activity', 'locError', 'stdError',
       'timeSteps'],
      dtype='object')
```

[5]:
```
idxList=np.unique(data["experimentIndex"])
print(idxList)
```

```
[0 1 2 3 4 5]
```

1.2 Naive version of the plot

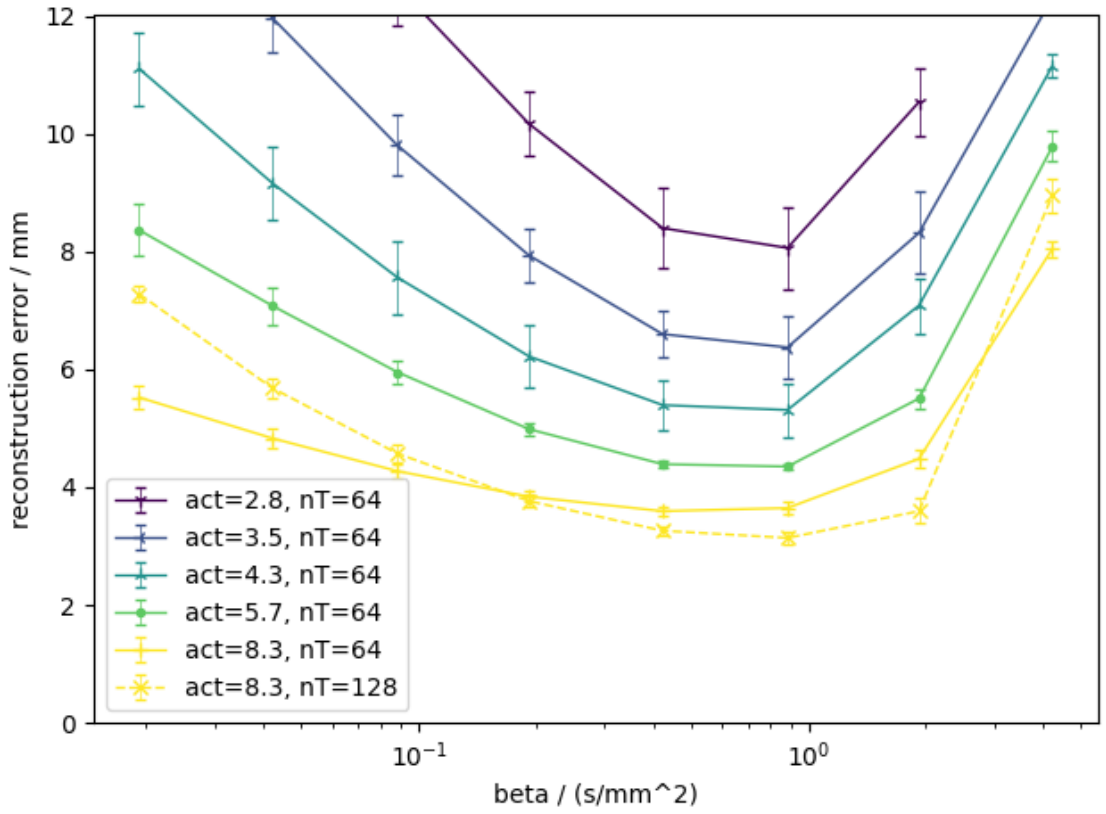
[6]:
```
fig=plt.figure()
ax=fig.add_subplot()

for i,idx in enumerate(idxList):
    datsub=data[data["experimentIndex"]==idx]
    label="act={:}, nT={}".format(datsub["activity"].
 ↪iloc[0],datsub["timeSteps"].iloc[0])
    plt.scatter(datsub["beta"],datsub["locError"],label=label)
plt.ylabel("reconstruction error / mm")
plt.xlabel("beta / (s/mm^2)")
plt.legend()
plt.tight_layout()
plt.show()
```



Throughout the lecture we have discussed plenty of aspects on how this plot can be improved. An example is given below.

1.3 Improved version

[7]:
```
markerList=["1","3","2",".","+","x"]

fig=plt.figure()
ax=fig.add_subplot()
plt.xscale("log")

for i,idx in enumerate(idxList):
    datsub=data[data["experimentIndex"]==idx]
    if i<5:
        col=cm.viridis(i/4)
        ls="solid"
    else:
        col=cm.viridis(1.)
        ls="dashed"
    label="act={:}, nT={}".format(datsub["activity"].
 ↪iloc[0],datsub["timeSteps"].iloc[0])
    plt.errorbar(datsub["beta"],datsub["locError"],yerr=datsub["stdError"],\
            lw=1,elinewidth=0.
 ↪5,capsize=2,marker=markerList[i],label=label,c=col,ls=ls)

plt.ylim([0,12])

plt.ylabel("reconstruction error / mm")
plt.xlabel("beta / (s/mm^2)")
plt.legend()
plt.tight_layout()
plt.show()
```

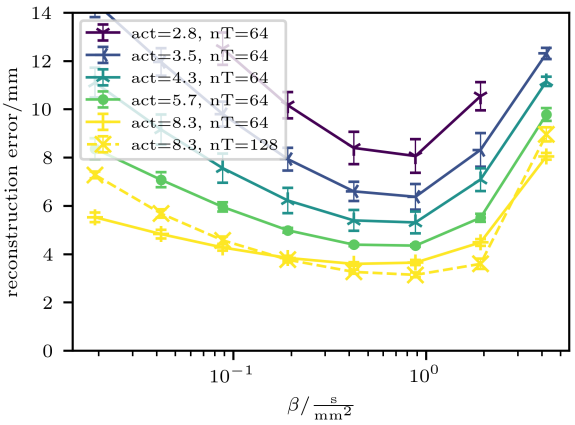


1.4 Exported version

[11]:
```
centm=1/2.54
```

[12]:
```
markerList=["1","3","2",".","+","x"]

plt.rc('text', usetex=True)
plt.rc('font',**{'family':'cmr10','serif':['Computer Modern Roman'],'size':'7'})
plt.rc('axes.formatter', use_mathtext=True)

fig=plt.figure(figsize=(8*centm,6*centm),dpi=600,facecolor="w")
ax=fig.add_subplot()
plt.xscale("log")

for i,idx in enumerate(idxList):
    datsub=data[data["experimentIndex"]==idx]
    if i<5:
        col=cm.viridis(i/4)
        ls="solid"
```

```
    else:
        col=cm.viridis(1.)
        ls="dashed"
    label="act={:}, nT={}".format(datsub["activity"].
 ↪iloc[0],datsub["timeSteps"].iloc[0])
    plt.errorbar(datsub["beta"],datsub["locError"],yerr=datsub["stdError"],\
            lw=1,elinewidth=0.
 ↪5,capsize=2,marker=markerList[i],label=label,c=col,ls=ls)

plt.ylim([0,14])

plt.ylabel(r"reconstruction error$ / \mathrm{mm}$")
plt.xlabel(r"$\beta / \frac{\mathrm{s}}{\mathrm{mm}^2}$")
plt.legend(loc="upper left",prop={'size': 6})
plt.tight_layout()
plt.show()
```



[14]:
```
fig.savefig("PET.pdf",dpi=600,bbox_inches='tight',pad_inches=0.025)
```

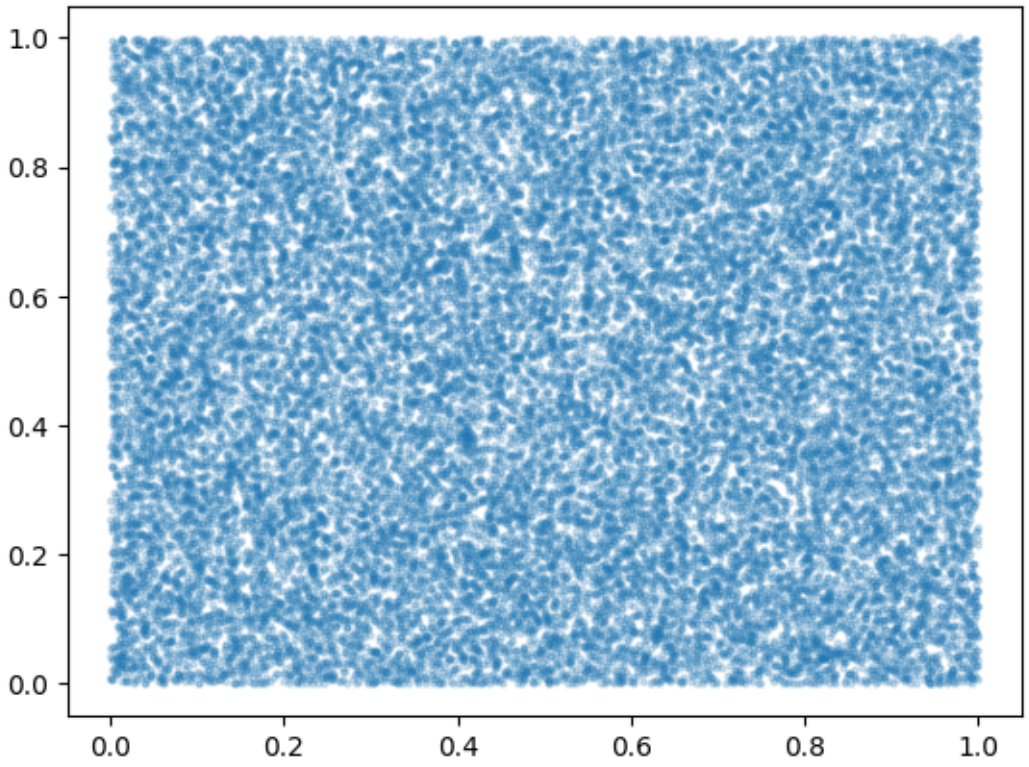
1.5 A small final trick, that could come in handy

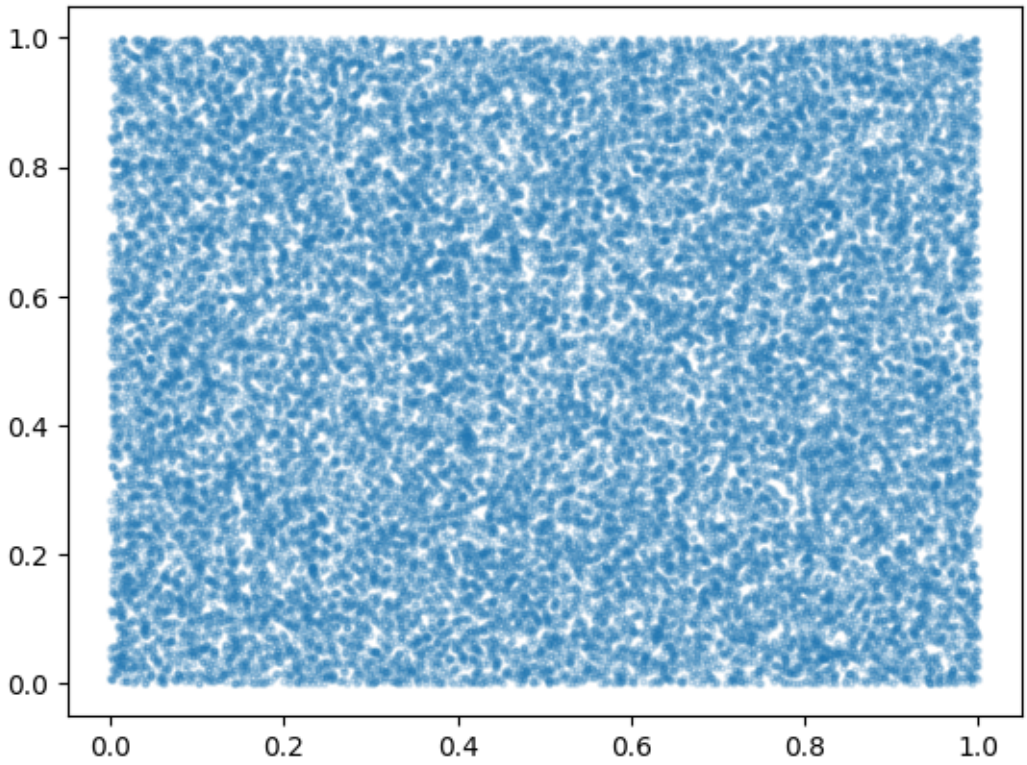
Of course, usually we will try to export figures as vector graphics, such as PDF (for, hopefully, obvious reasons). For some complex renderings this is not possible. In same cases it would be possible but is still not advisable, e.g. for very heavy point clouds. See the example below.

[8]:
```
n=30000
data=np.random.random(size=(n,2))
```

[9]:
```
for rast in [True,False]:
    fig=plt.figure()
```

```
    plt.scatter(data[:,0],data[:,1],s=5,alpha=0.2,rasterized=rast)
    fn="./export_example_"
    if rast:
        fn+="rast"
    else:
        fn+="vec"
    fn+=".pdf"
    fig.savefig(fn)
```




1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: