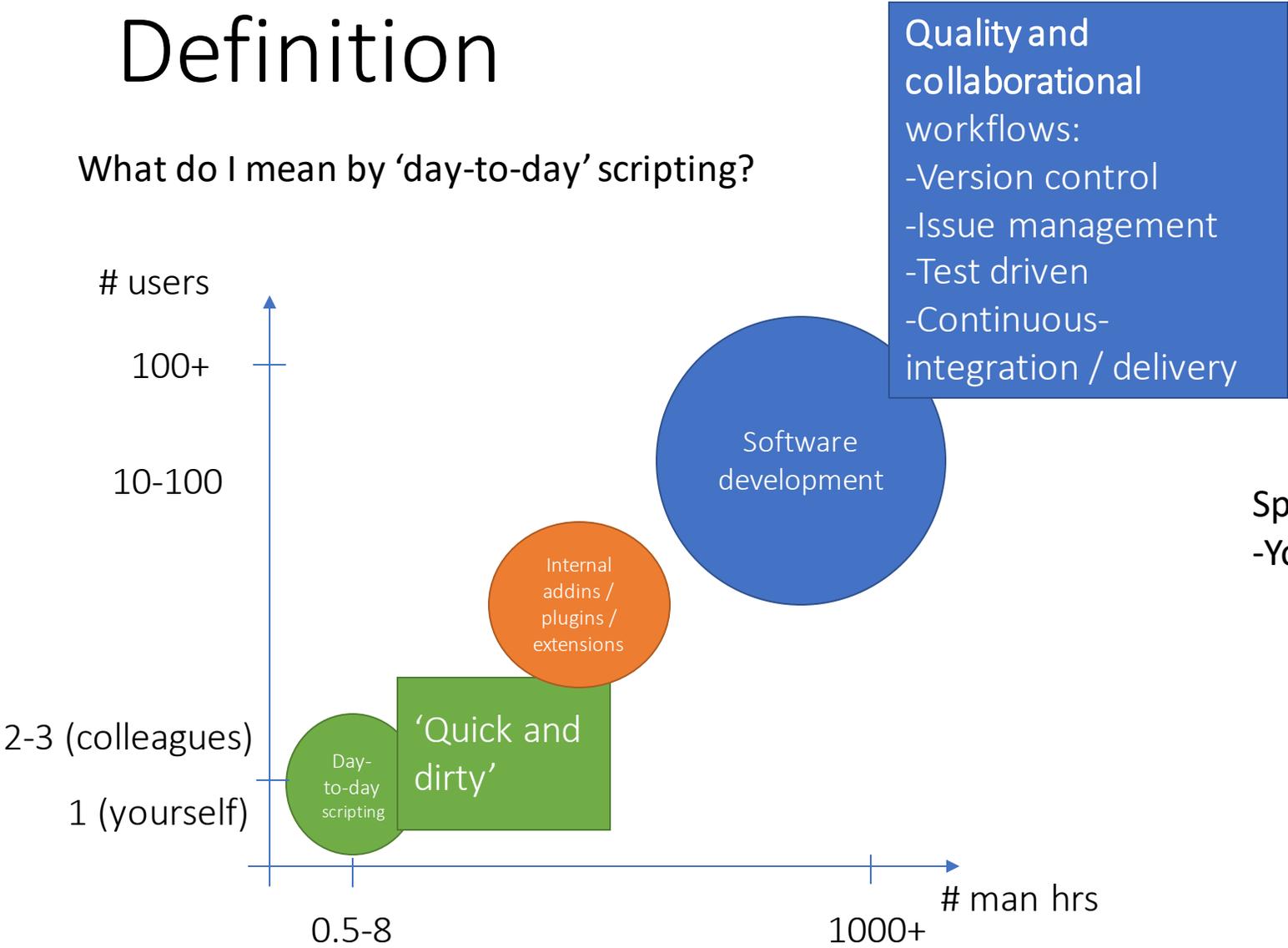


Agenda

- Background
- Case 1: Creating an effective workflow around (after) a simulation model
 - and a detour around pressurized equipment
- Further cases (if time permits)
 - Case 2: Small scale scraping of pfd
 - Case 3: Large scale ML based scraping
- Final thoughts
- Discussion

Definition

What do I mean by 'day-to-day' scripting?



Spaghetti code is ok for 'Quick and dirty'
-You can structure when it works...



When should scripting be used?

- Most engineers probably know this flowchart for mechanical systems...

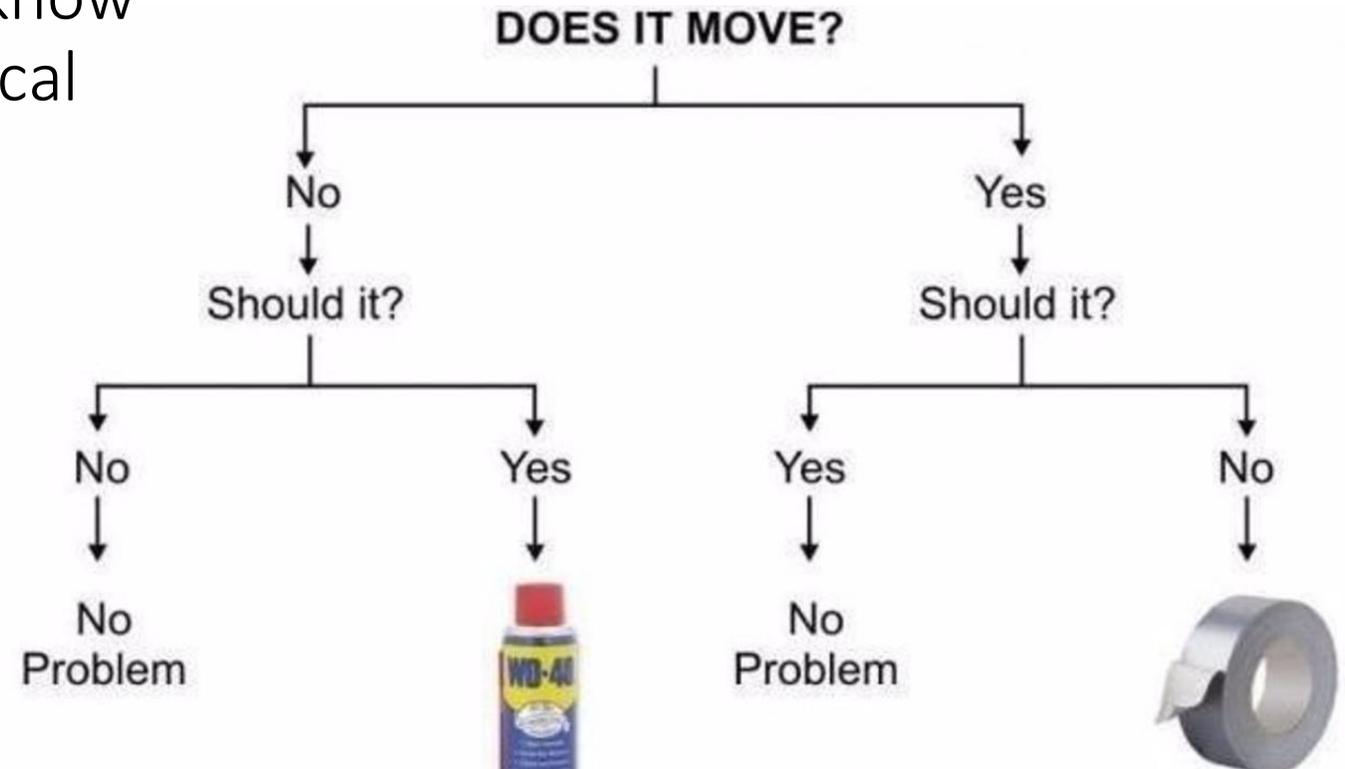
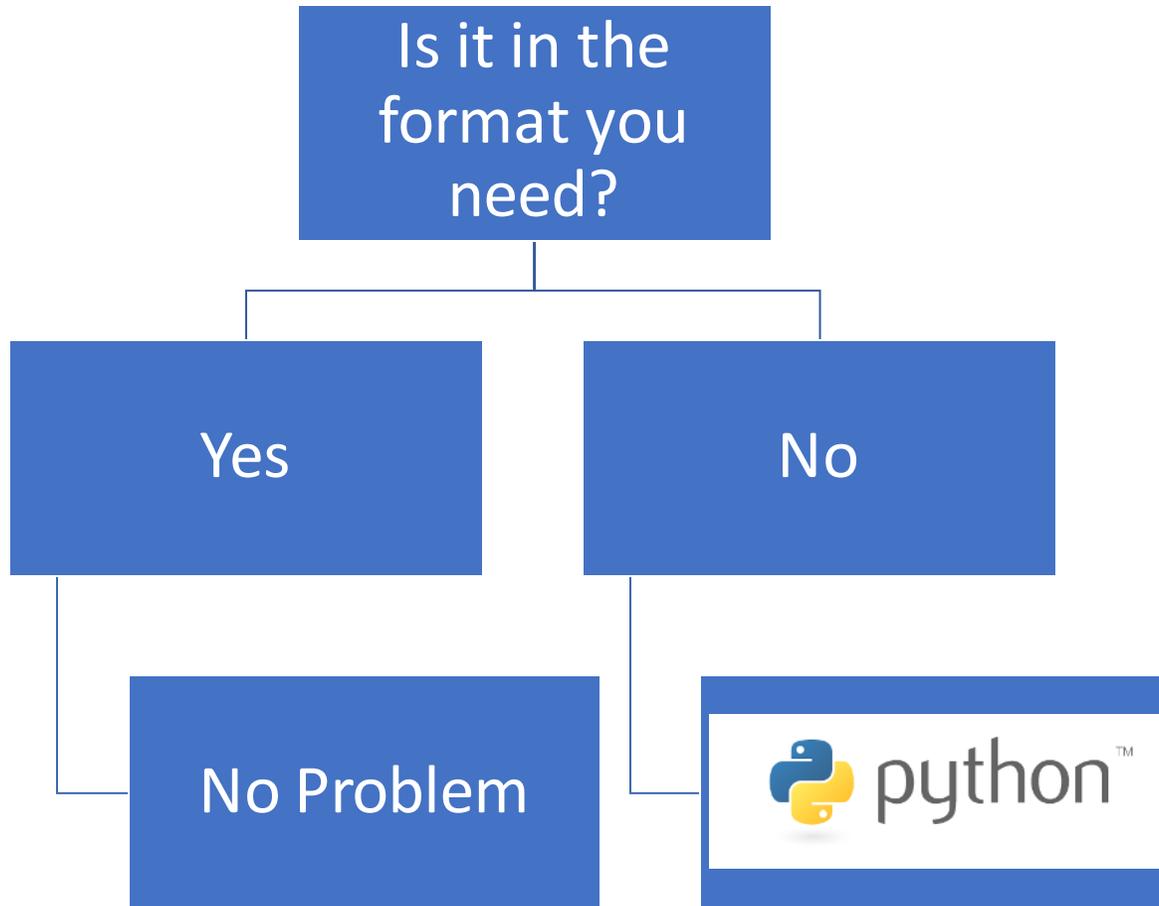
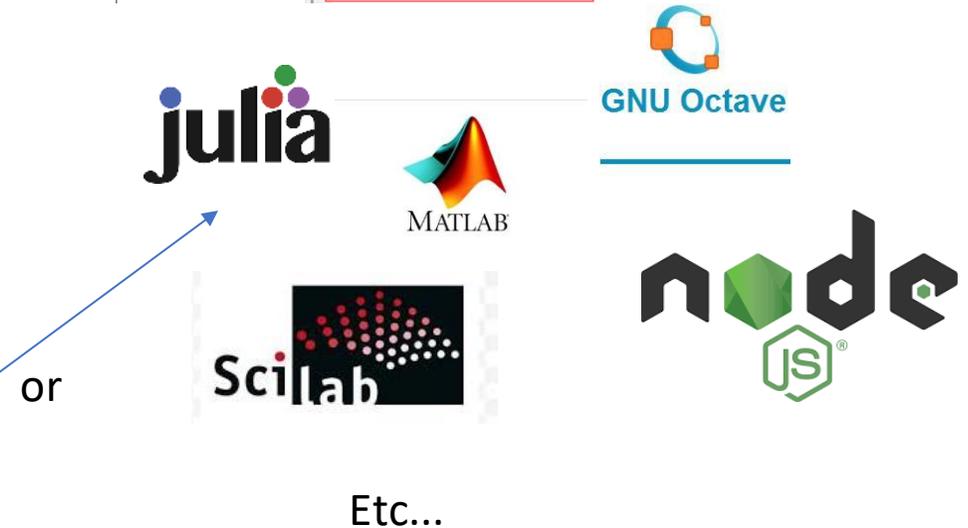


Image courtesy of <https://www.flickr.com/photos/dullhunk/7214525854>

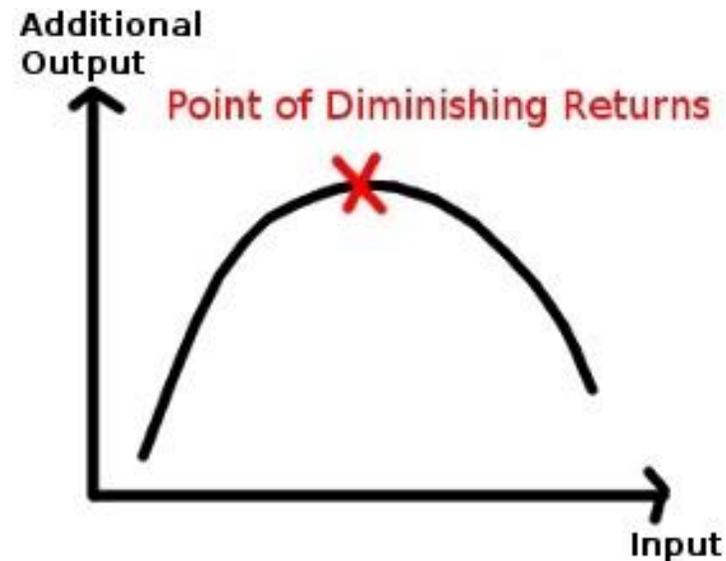
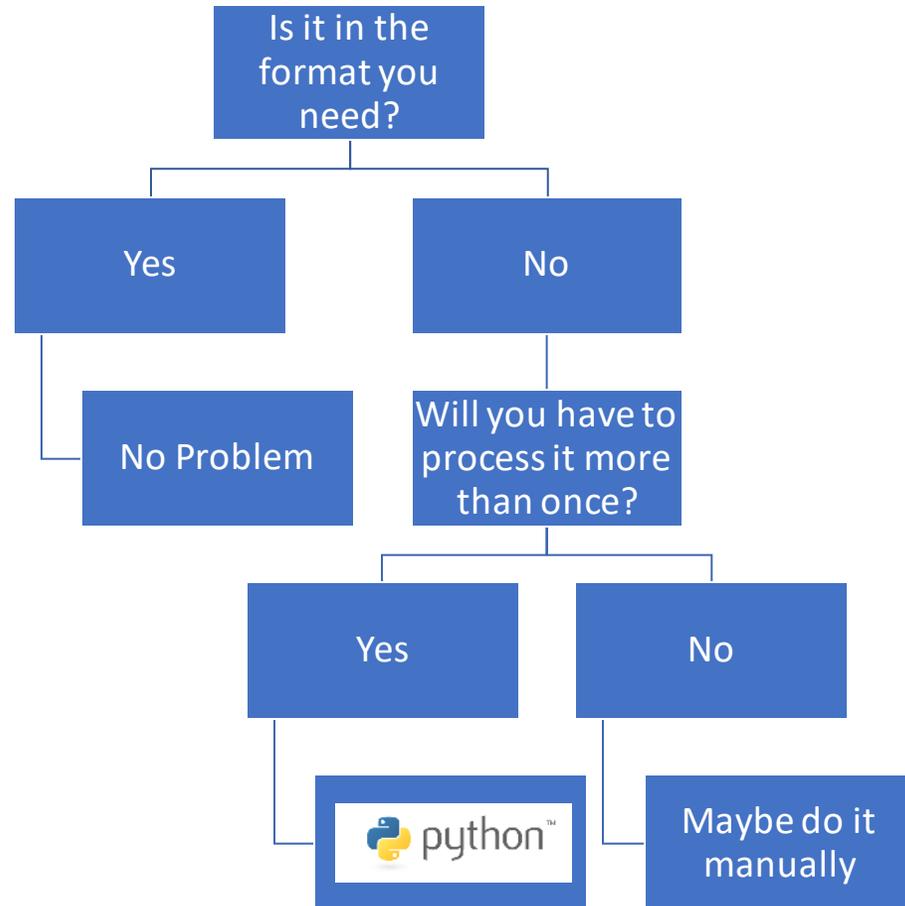
...I think a similar flowchart could be made for data



```
Sub CheckScore()  
If Range("A1").Value < 35 Then  
MsgBox "Fail"  
Else  
If Range("A1").Value < 60 Then  
MsgBox "Pass"  
Else  
MsgBox "Pass, with distinction"  
End If  
End If  
End Sub
```



...it's maybe worth softening it a bit..



Coding/scripting should add value to your project...

A decision

- Should script inside your Simulation tool or outside?
- Most simulation tools have well rounded API's
- But scripting outside them have some advantages:
 - You have direct access to your favourite packages (as we'll see later)
 - Your scripts can be reused across simulation tools
 - You do not take up a license while developing/executing your scripts
 - You can use your favourite editor / IDE
 - You can script across models – this can be trickier from within the tool
- For me it is probably 30/70 in favour of outside – I script inside if:
 - The clients wants the script integrated into the tool as an extension / plugin / model method
 - The benefits of having access to native plotting capabilities in the tool outweighs the advantages of working outside
- In deadline pressed projects is usually a 'hybrid'



Python 2 (IronPython)



Python 2

Altair SimLab™

Python 3



Java 8

Workflow



A minimal toolbox

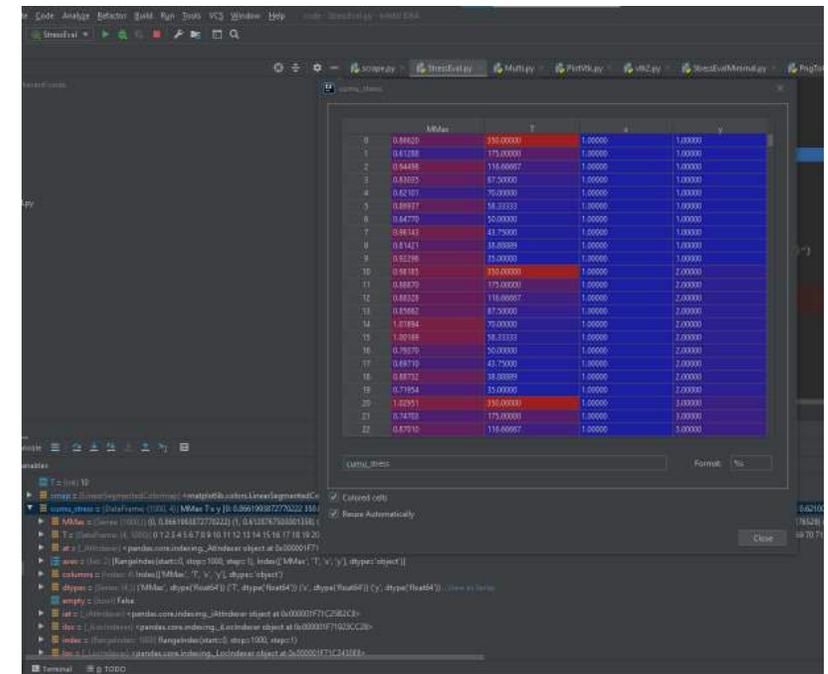
- Python distribution
 - Anaconda is popular
 - Conda allows for nice (and graphical) package management
- Modules
 - Numpy
 - Pandas
 - Matplotlib
- Editor
 - This is likely contentious!
 - I highly recommend JetBrains 2 free options (Without any affiliation 😊)
 - Pycharm community edition
 - IntelliJ community edition with Python Plugin for cross-language development
 - The paid versions have database integration, docker support and javascript – very nice, but not a must
 - Alternatives
 - VS Studio code
 - Spyder (comes with Anaconda)



If you go with the free editors – you have a powerful coding setup for 0 DKK !!

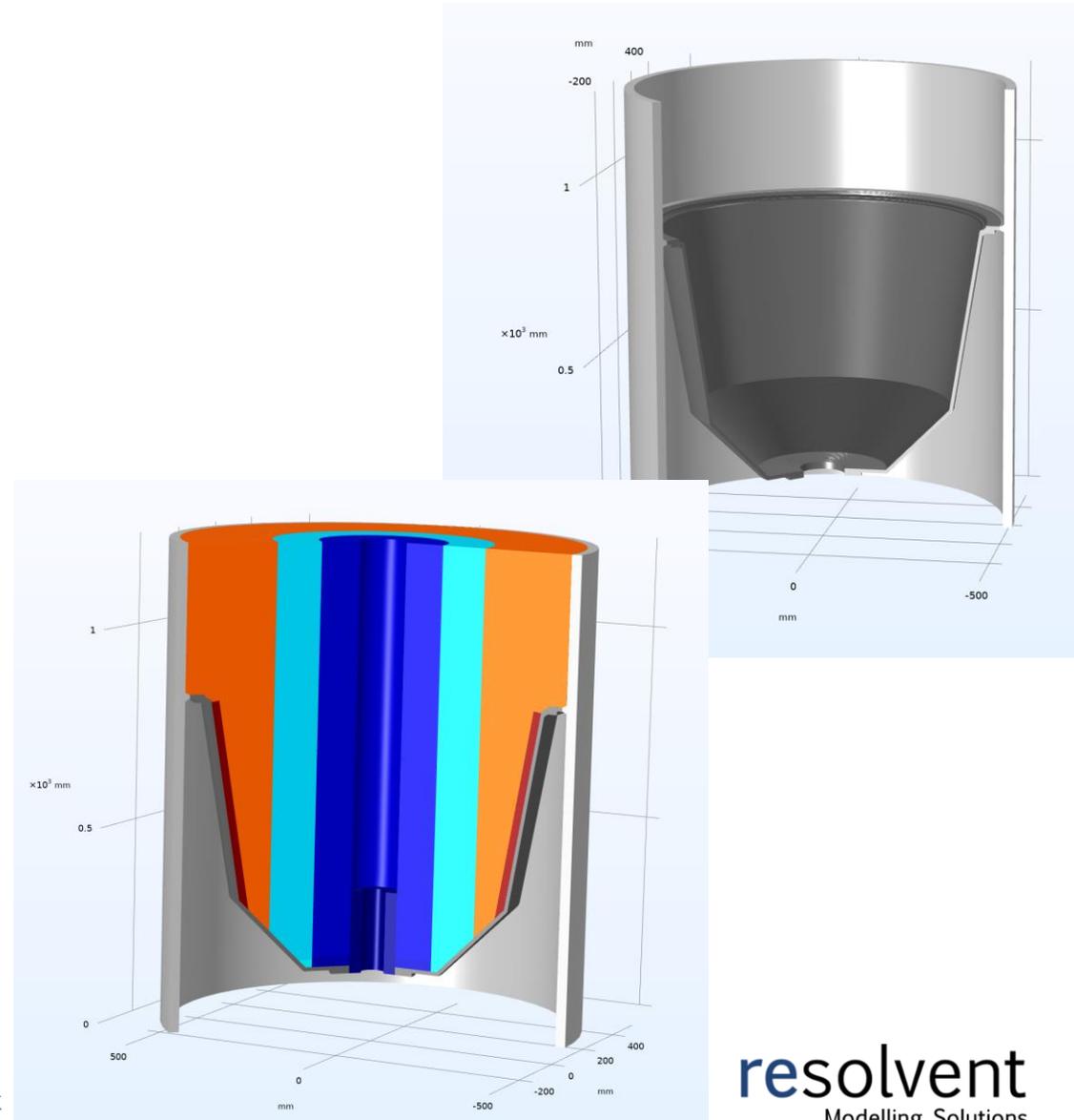
A few words on the editor

- For the editor code completions, and an interactive debugger is a must! Pandas dataframe inspector is a very nice feature!!
- When you script inside the simulation tool, you usually have to use their environment
 - Although you can sometimes setup your workflow for debugging from outside – a topic for another talk 😊



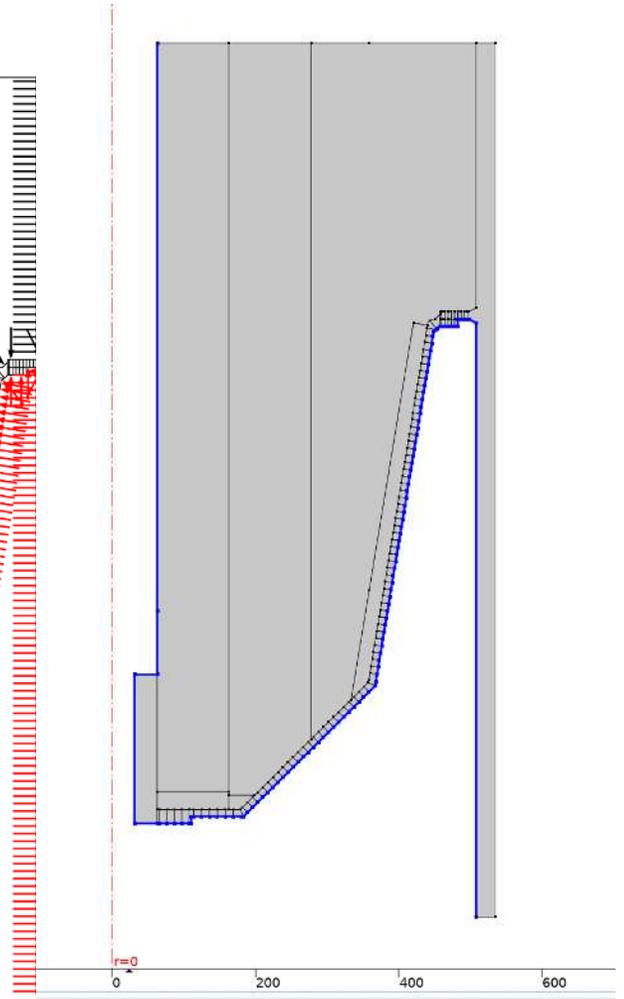
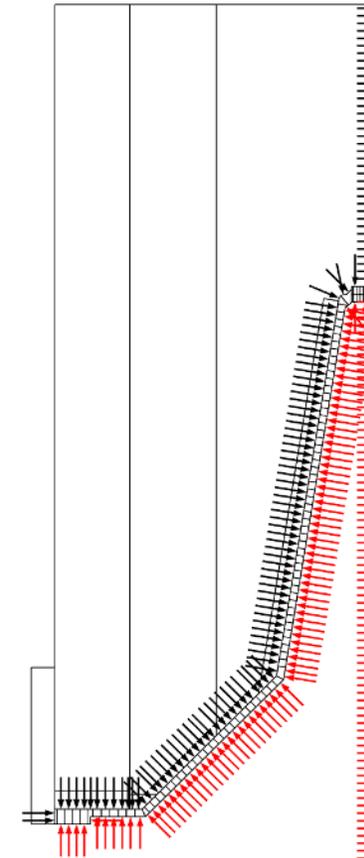
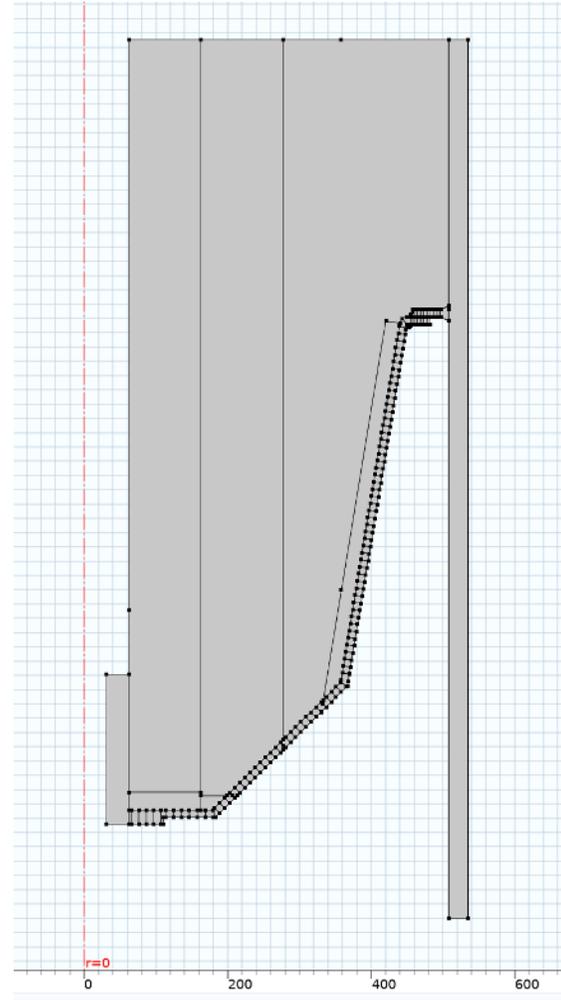
Case: Creating an effective workflow around (after) a simulation model

- We need a detour around analyzing pressurized equipment to provide some context....
- This task was executed last week and illustrates a VERY deadline-induced *quick-and-dirty* approach to scripting
- Background:
 - Our client has a conical component inside a pressure shell
 - I've changed to component for IP reasons
 - We're tasked with modifying the design to be structurally safe according to ASME BPVC
 - We use Comsol Multiphysics and a coupled HeatTransfer + Solid Mechanics Model



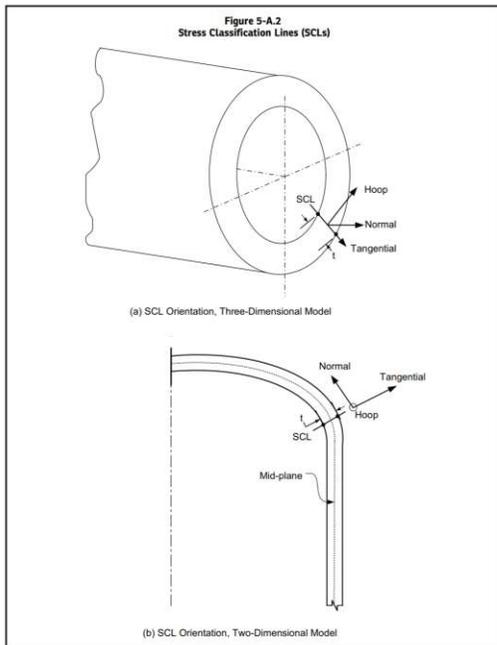
Underlying model

- Axial symmetry is employed for simplicity and speed
- The component is subject to pressure and deadweight loads
- It is protected by refractory and subject to convective/radiative heattransfer from a gas flowing through it

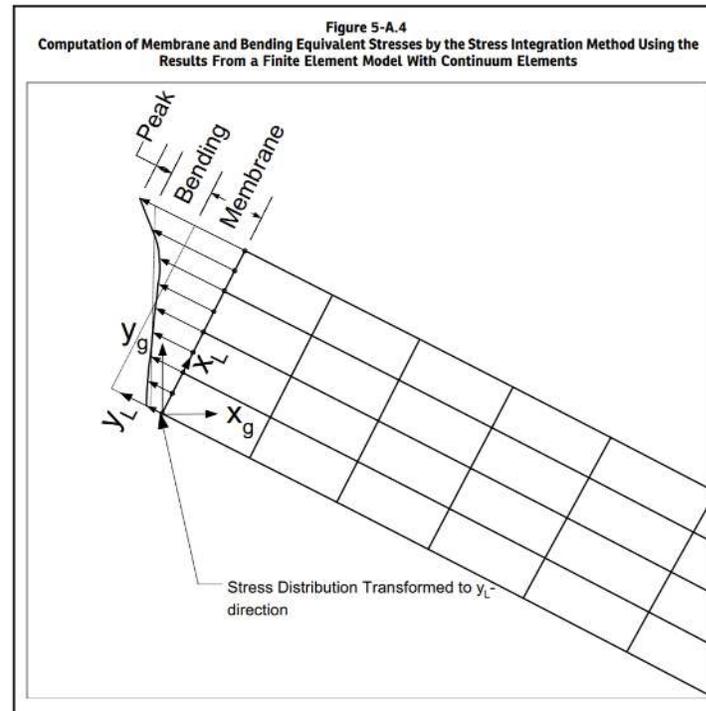


Stress evaluation

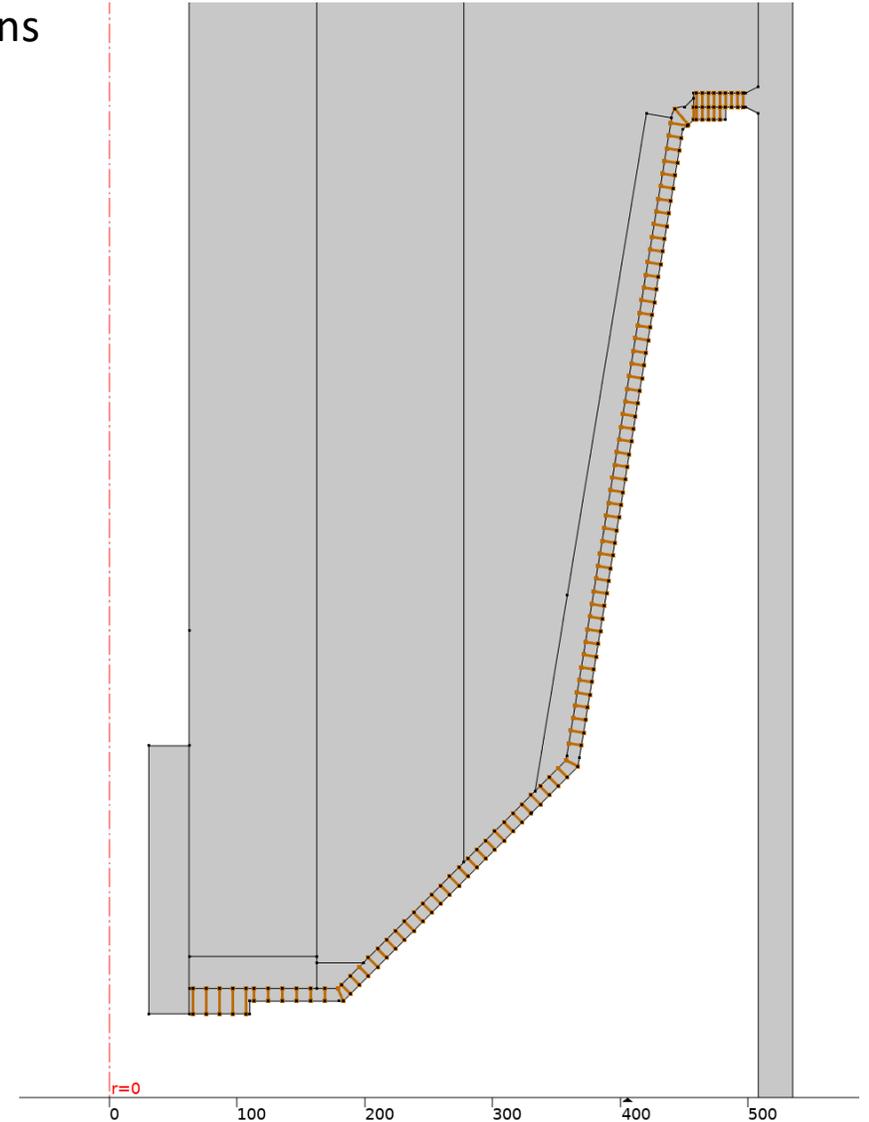
Introduce Stress classification line through critical locations

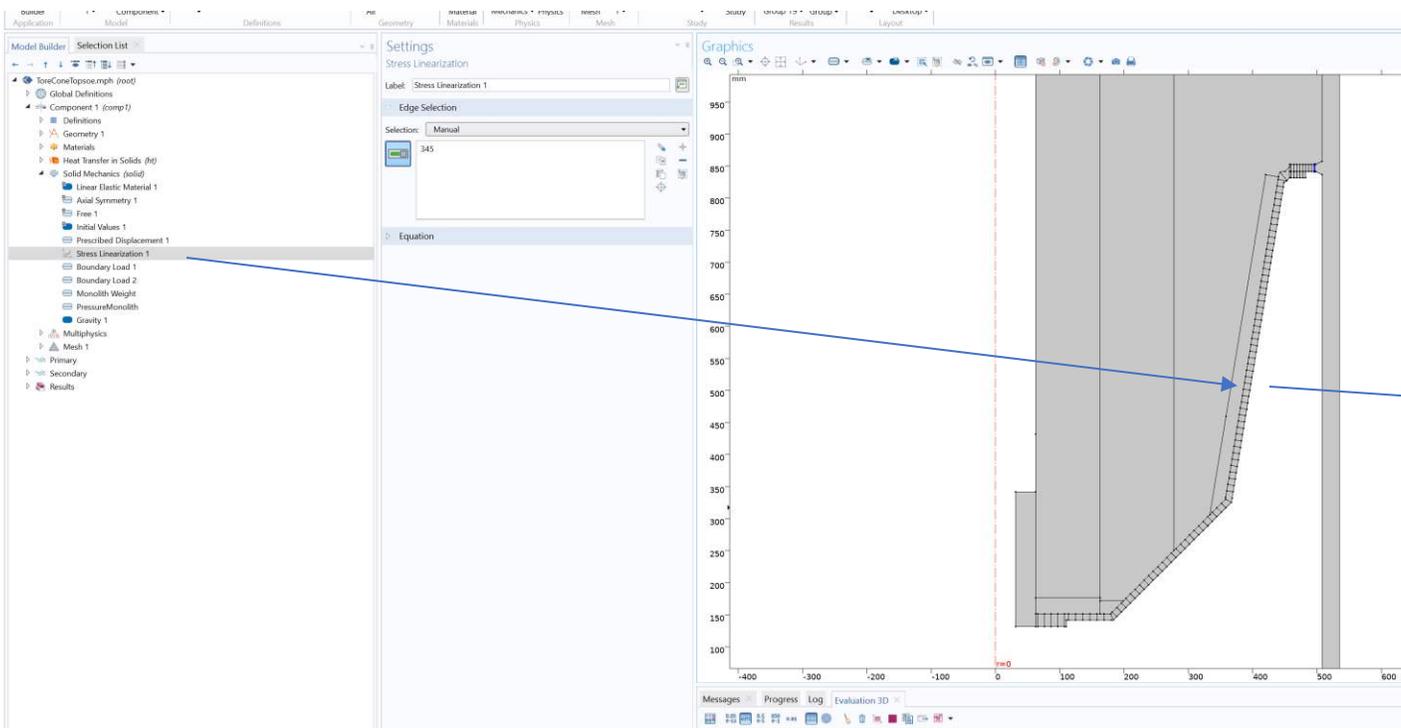


In the SCL's "reduce" the stress to a "Membrane" and a "Bending" contribution
-These can then be evaluated against allowable values



During design iterations the max can move around – thus we must investigate ~100 locations





We create a small export script in comsol that traverses the ~100 SCL's and outputs the necessary quantities to a textfile

```

1 package builder;
2
3 import com.comsol.api.*;
4 import com.comsol.model.*;
5 import com.comsol.model.physics.*;
6 import com.comsol.model.application.*;
7
8 public class method1 extends ApplicationMethod {
9
10 public void execute() {
11     int[] pathNos = model.component("comp1").selection("sol1").entities(1);
12
13     for (int pathNo : pathNos) {
14         model.param().set("pathnom", pathNo);
15
16         model.component("comp1").physics("solid").feature("sl1").selection("edgeSelection").set();
17         model.component("comp1").physics("solid").feature("sl1").selection("edgeSelection").set(pathNo);
18
19         model.sol("sol1").updatesolution();
20         model.sol("sol2").updatesolution();
21
22
23
24         model.result().export("data1").run();
25         model.result().export("data2").run();
26     }
27     int[] pathNos2 = model.component("comp1").selection("selc").entities(1);
28
29     for (int pathNo2 : pathNos2) {
30         model.param().set("pathnom2", pathNo2);
31
32         model.component("comp1").physics("solid").feature("sl1").selection("edgeSelection").set();
33         model.component("comp1").physics("solid").feature("sl1").selection("edgeSelection").set(pathNo2);
34
35         model.sol("sol1").updatesolution();
36         model.sol("sol2").updatesolution();
37
38
39
40         model.result().export("data3").run();
41         model.result().export("data4").run();
42
43
44
45     }
46 }
47
48
49 }

```

ANSYS Workbench Messages Log showing a list of messages:

Msg ID	Msg Text	Msg Type	Msg Category
15	0.000127745134412024	0.0000000000000000	66.41221289113113
16	1.059467181204247	0.0000000000000000	130
17	0.0000000000000000	0.0000000000000000	1.00
18	0.0000442131239359	0.0000000000000000	147.1794202091658
19	0.0000000000000000	0.0000000000000000	148
20	0.0000000000000000	0.0000000000000000	147.4793220500408
21	0.0000000000000000	0.0000000000000000	147.394931209077
22	0.0000000000000000	0.0000000000000000	148.4941241487470
23	0.0000000000000000	0.0000000000000000	148.20121625485076
24	0.0000000000000000	0.0000000000000000	151.18121409999107
25	0.0000000000000000	0.0000000000000000	150.33688912828755
26	0.0000000000000000	0.0000000000000000	142.6049494261426
27	0.0000000000000000	0.0000000000000000	141.00001217877000
28	0.0000000000000000	0.0000000000000000	129.74221201334343
29	0.0000000000000000	0.0000000000000000	129.4785138797044
30	0.0000000000000000	0.0000000000000000	127.2576471093102
31	0.0000000000000000	0.0000000000000000	150.33688912828755
32	0.0000000000000000	0.0000000000000000	142.6049494261426
33	0.0000000000000000	0.0000000000000000	141.00001217877000
34	0.0000000000000000	0.0000000000000000	129.74221201334343
35	0.0000000000000000	0.0000000000000000	129.4785138797044
36	0.0000000000000000	0.0000000000000000	127.2576471093102
37	0.0000000000000000	0.0000000000000000	150.33688912828755
38	0.0000000000000000	0.0000000000000000	142.6049494261426
39	0.0000000000000000	0.0000000000000000	141.00001217877000
40	0.0000000000000000	0.0000000000000000	129.74221201334343
41	0.0000000000000000	0.0000000000000000	129.4785138797044
42	0.0000000000000000	0.0000000000000000	127.2576471093102
43	0.0000000000000000	0.0000000000000000	150.33688912828755
44	0.0000000000000000	0.0000000000000000	142.6049494261426
45	0.0000000000000000	0.0000000000000000	141.00001217877000
46	0.0000000000000000	0.0000000000000000	129.74221201334343
47	0.0000000000000000	0.0000000000000000	129.4785138797044
48	0.0000000000000000	0.0000000000000000	127.2576471093102
49	0.0000000000000000	0.0000000000000000	150.33688912828755
50	0.0000000000000000	0.0000000000000000	142.6049494261426
51	0.0000000000000000	0.0000000000000000	141.00001217877000
52	0.0000000000000000	0.0000000000000000	129.74221201334343
53	0.0000000000000000	0.0000000000000000	129.4785138797044
54	0.0000000000000000	0.0000000000000000	127.2576471093102
55	0.0000000000000000	0.0000000000000000	150.33688912828755
56	0.0000000000000000	0.0000000000000000	142.6049494261426
57	0.0000000000000000	0.0000000000000000	141.00001217877000
58	0.0000000000000000	0.0000000000000000	129.74221201334343
59	0.0000000000000000	0.0000000000000000	129.4785138797044
60	0.0000000000000000	0.0000000000000000	127.2576471093102
61	0.0000000000000000	0.0000000000000000	150.33688912828755
62	0.0000000000000000	0.0000000000000000	142.6049494261426
63	0.0000000000000000	0.0000000000000000	141.00001217877000
64	0.0000000000000000	0.0000000000000000	129.74221201334343
65	0.0000000000000000	0.0000000000000000	129.4785138797044
66	0.0000000000000000	0.0000000000000000	127.2576471093102
67	0.0000000000000000	0.0000000000000000	150.33688912828755
68	0.0000000000000000	0.0000000000000000	142.6049494261426
69	0.0000000000000000	0.0000000000000000	141.00001217877000
70	0.0000000000000000	0.0000000000000000	129.74221201334343
71	0.0000000000000000	0.0000000000000000	129.4785138797044
72	0.0000000000000000	0.0000000000000000	127.2576471093102
73	0.0000000000000000	0.0000000000000000	150.33688912828755
74	0.0000000000000000	0.0000000000000000	142.6049494261426
75	0.0000000000000000	0.0000000000000000	141.00001217877000
76	0.0000000000000000	0.0000000000000000	129.74221201334343
77	0.0000000000000000	0.0000000000000000	129.4785138797044
78	0.0000000000000000	0.0000000000000000	127.2576471093102
79	0.0000000000000000	0.0000000000000000	150.33688912828755
80	0.0000000000000000	0.0000000000000000	142.6049494261426
81	0.0000000000000000	0.0000000000000000	141.00001217877000
82	0.0000000000000000	0.0000000000000000	129.74221201334343
83	0.0000000000000000	0.0000000000000000	129.4785138797044
84	0.0000000000000000	0.0000000000000000	127.2576471093102
85	0.0000000000000000	0.0000000000000000	150.33688912828755
86	0.0000000000000000	0.0000000000000000	142.6049494261426
87	0.0000000000000000	0.0000000000000000	141.00001217877000
88	0.0000000000000000	0.0000000000000000	129.74221201334343
89	0.0000000000000000	0.0000000000000000	129.4785138797044
90	0.0000000000000000	0.0000000000000000	127.2576471093102
91	0.0000000000000000	0.0000000000000000	150.33688912828755
92	0.0000000000000000	0.0000000000000000	142.6049494261426
93	0.0000000000000000	0.0000000000000000	141.00001217877000
94	0.0000000000000000	0.0000000000000000	129.74221201334343
95	0.0000000000000000	0.0000000000000000	129.4785138797044
96	0.0000000000000000	0.0000000000000000	127.2576471093102
97	0.0000000000000000	0.0000000000000000	150.33688912828755
98	0.0000000000000000	0.0000000000000000	142.6049494261426
99	0.0000000000000000	0.0000000000000000	141.00001217877000
100	0.0000000000000000	0.0000000000000000	129.74221201334343

M
MB
MBQ
Path
T
S

```

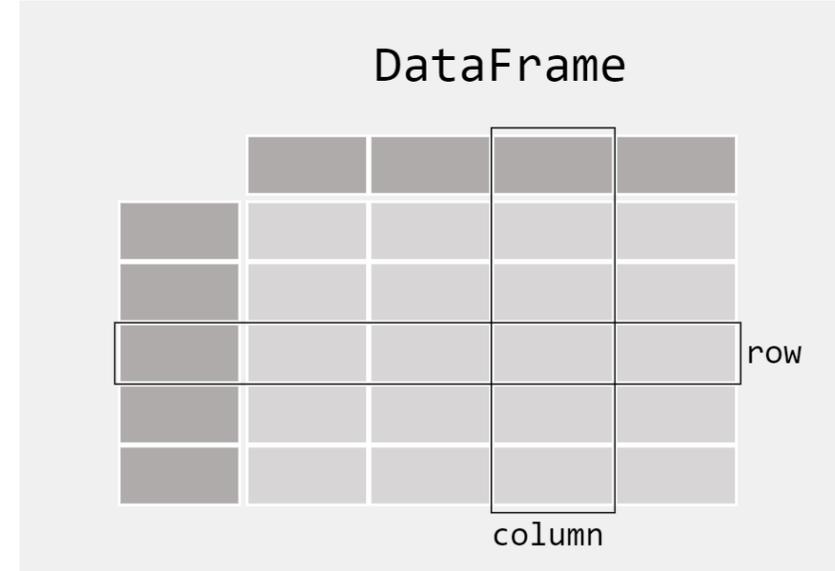
eval.txt - Notepad
File Edit Format View Help
13 0.0031196734534413824 0.009583720887604319 64.0112108515151 138
13 1.4646977184260197
16 0.0026462706876552015 0.009268455410645225 96.61333699777731 138
16 1.0804443533269956
19 0.002303966015357288 0.009123252881328422 124.17492065844618 138
19 0.8125414532669621
22 0.0021544322003392785 0.008886928854976038 147.47932596634018 138
22 0.5954430537086437
25 0.0016955851316031582 0.008665902086727389 167.1049051299977 138
25 0.3809776021231697
30 0.005953834781742473 0.014366276743684467 184.45017615815232 138
30 0.24569229826887748
33 0.005573708740970761 0.014112223403671801 210.2039425485679 138
33 0.44053135561281426
36 0.005217474968329894 0.013530301094642812 231.18627409991507 137.2576471080102
36 0.584814279269806
39 0.005051052958197448 0.01265311503563375 248.4666233226037 135.18400520128753
39 0.7022960539637502
42 0.00499938219817806 0.0113856402824806096 262.87194356776627 132.9404890291574
42 0.7918121097417513
51 0.004981877644890253 0.009668340614314885 274.9773545076496 131.00362327877608
51 0.8610526866505721
55 0.00597656690720585 0.006908140415048684 285.47066530797144 129.74352016304343
55 0.9047309088711895
57 0.0060044027505342336 0.007949528201157748 289.0218078320345 129.31738306015586
57 0.8625222284421634
61 0.007454005225426221 0.00935875309509715 294.6991797725973 128.63609842728835
61 0.8250490421099196
65 0.008775074890299574 0.010372297333947749 299.5491397433333 128.0541032308
65 0.7653722236121128
68 0.00977899752381139 0.011067422480298352 303.994170486519 127.52069954161772
68 0.69090983174266708
71 0.010477099037741957 0.01148776162894616 308.49716645243393 126.98034002570792
71 0.6111914386957316
74 0.010876103198786473 0.011665060102965929 312.8900598496843 126.45223281803787
74 0.532862203231241
77 0.010982903662755212 0.011615786184130391 317.1162616797525 125.94604859842967
77 0.4589327983685751
80 0.010805734397290937 0.01134571260642077 321.108476112644 125.46698286648271
80 0.4061844026922003
83 0.010354305936381238 0.010851170911693513 324.85033157305054 125.01796021123394
83 0.3739300645728538
86 0.009630215870303153 0.010109757785299682 328.33006723847313 124.73359462092215
86 0.3382114080401066
89 0.008661348055329606 0.009116960861823413 331.5431396043151 124.47654883165478
89 0.30166670901873377
92 0.007468798115221987 0.008596645097412418 334.4883417947498 124.24093265642
92 0.2658762447901451
95 0.0060795547884964505 0.007908154315933414 337.16773384734813 124.02658129221214
95 0.23192558229003465
98 0.004528741249409723 0.0068814874787544575 339.58568109706835 123.83314552123454
98 0.2005195681374134
104 0.0028617577629380026 0.005446975240863641 341.75064453822006 123.65994843694239
104 0.17206429415443136
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

- Now we have a text file
- It's 'jagged' because we're evaluating secondary stresses separately
- It's not very intuitive to look at – it's very hard to make good design decisions from
- Time for some scripting!
- I'm going to show you some code – my main message is how little you need – not the actual code
- And gode-golfers can surely shorten it further;-)

...enter pandas

- Pandas stores data in rows and columns in a DataFrame
 - a bit like a spreadsheet on steroids
- Getting our data into a dataframe is a good basis for further exploration and visualization
- This is our first order of business



	MMax	T	x	y
0	0.86620	350.00000	1.00000	1.00000
1	0.61288	175.00000	1.00000	1.00000
2	0.94498	116.66667	1.00000	1.00000
3	0.83095	87.50000	1.00000	1.00000
4	0.62101	70.00000	1.00000	1.00000
5	0.86937	58.33333	1.00000	1.00000
6	0.64770	50.00000	1.00000	1.00000
7	0.96143	43.75000	1.00000	1.00000
8	0.81421	38.88889	1.00000	1.00000
9	0.92296	35.00000	1.00000	1.00000
10	0.98185	350.00000	1.00000	2.00000
11	0.88870	175.00000	1.00000	2.00000
12	0.88328	116.66667	1.00000	2.00000
13	0.85662	87.50000	1.00000	2.00000
14	1.01894	70.00000	1.00000	2.00000
15	1.00169	58.33333	1.00000	2.00000
16	0.79370	50.00000	1.00000	2.00000
17	0.69710	43.75000	1.00000	2.00000
18	0.88732	38.88889	1.00000	2.00000
19	0.71954	35.00000	1.00000	2.00000
20	1.02951	350.00000	1.00000	3.00000
21	0.74703	175.00000	1.00000	3.00000
22	0.87010	116.66667	1.00000	3.00000

5 lines of code converts our jagged text file to a nicely structured dataframe

```

import pandas as pd

df = pd.read_csv(r"Z:\Tore\networkevent\eval.txt", header=None, delim_whitespace=True)

prim = df.iloc[:,2:]
sec = df.iloc[1::2,:]

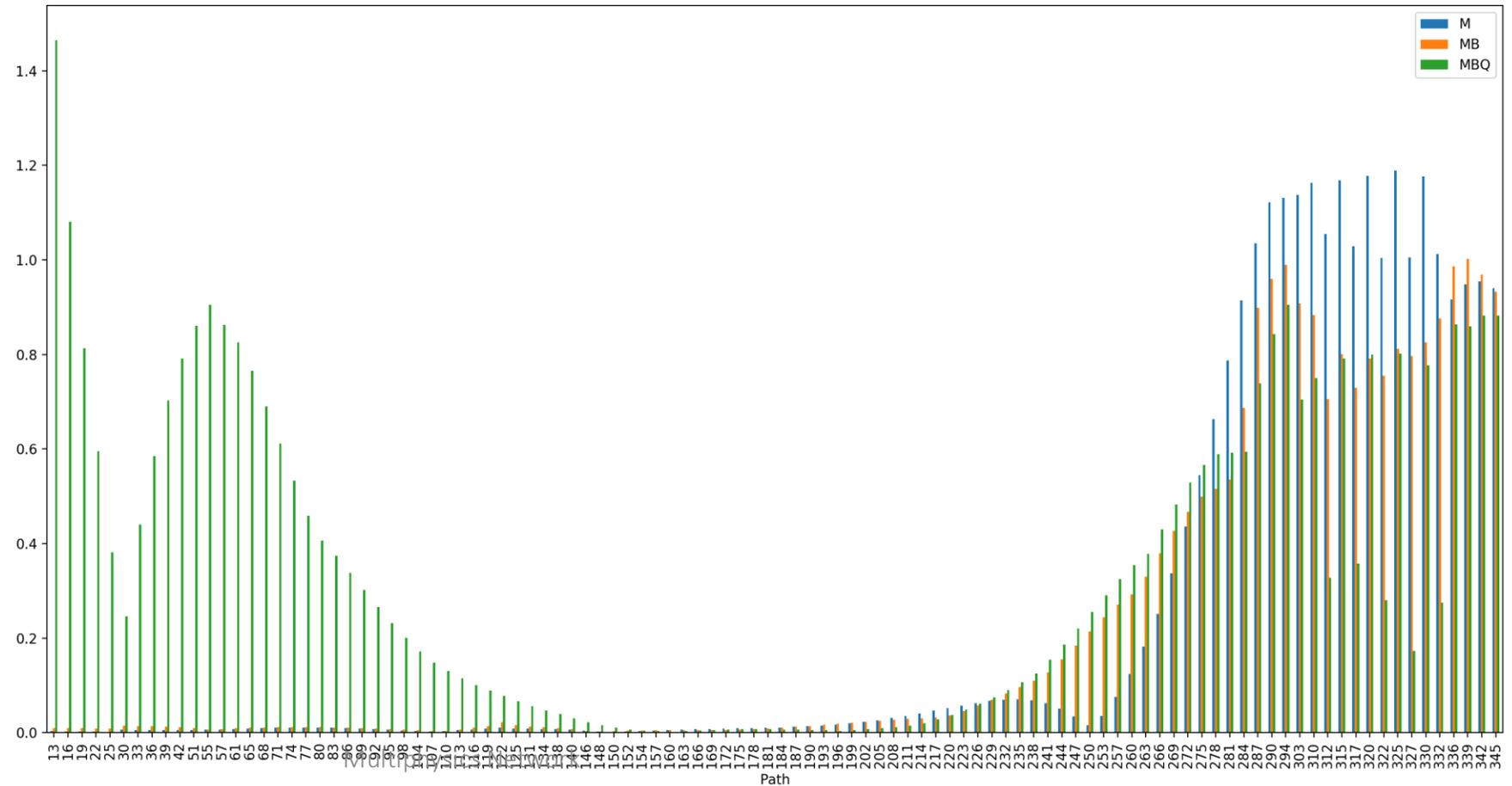
stresses = pd.DataFrame({'Path': prim[0], 'M': prim[1], 'MB': prim[2], 'MBQ': list(sec[1]), 'T': prim[3], 'S': prim[4]}).sort_values(by='Path')
    
```

The image shows a text file with multiple lines of data. Each line represents a network event and contains several columns of values. The number of columns varies from line to line, creating a jagged appearance. The columns appear to contain numerical values, possibly representing network metrics or stress levels.

	Path	M	MB	MBQ	T	S
0	13	0.01951	0.07418	0.32542	418.44493	117.52441
2	16	0.01616	0.06149	0.21772	413.25444	117.93965
4	19	0.01417	0.05266	0.15546	409.02374	118.27810
6	22	0.01331	0.04624	0.11133	405.51847	118.55852
8	25	0.01085	0.03995	0.10156	402.56564	118.79475
10	30	0.02187	0.02981	0.06963	400.60382	118.95169
12	33	0.01843	0.02585	0.08161	397.01608	119.11936
14	36	0.01556	0.02936	0.09893	393.93458	119.24262
16	39	0.01330	0.03211	0.11107	391.21904	119.35124
18	42	0.01148	0.03383	0.11961	388.78684	119.44853
20	51	0.01001	0.03471	0.12498	386.63554	119.53458
22	55	0.01121	0.03577	0.12639	384.83125	119.60675
24	57	0.01344	0.03271	0.10884	384.58777	119.61649
26	61	0.01676	0.02734	0.08839	384.39330	119.62427
28	65	0.01755	0.02238	0.06398	384.37329	119.62507
30	68	0.01690	0.01787	0.04136	384.22964	119.63081
32	71	0.01552	0.01409	0.02421	383.61257	119.65550
34	74	0.01384	0.01112	0.02350	382.73657	119.69054
36	77	0.01212	0.00999	0.02077	381.71110	119.72111

- Plotting it, is 3 extra lines

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3
4 df = pd.read_csv(r"Z:\Tore\networkevent\eval.txt", header=None, delim_whitespace=True)
5
6 prim = df.iloc[:,2, :]
7 sec = df.iloc[1::2, :]
8
9 stresses = pd.DataFrame({'Path': prim[0], 'M': prim[1], 'MB': prim[2], 'MBQ': list(sec[1]), 'T': prim[3], 'S': prim[4]}).sort_values(by=['Path'])
10
11 stresses.plot.bar(x='Path', y=['M', 'MB', 'MBQ'])
12
13 plt.show()
```



1 statement exports the dataframe to Excel

```
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib import cm

cmap = cm.get_cmap('jet')

def scatter_and_label(axis, key, frame):
    frame.plot('Xc', 'Yc', kind='scatter', ax=axis, s=40, linewidth=0,
              c=key, colormap=cmap)

    m = frame[frame[key] == frame[key].max()]
    s = f'Max: {int(m["Path"].values[0])} | {m[key].values[0]:.2f}'
    axis.annotate(s, (m['Xc'], m['Yc']), xycoords='data', xytext=(-75, -50), textcoords='offset points', arrowprops=dict(facecolor='black', shrink=0.05))

    axis.axis('equal')
    axis.set_title(key)

df = pd.read_csv("Z:\Tore\networkevent\eval2.txt", header=None, delimiter=True)

prim = df.iloc[:, 2:]
sec = df.iloc[:, 1:]

stresses = pd.DataFrame({'Path': prim[0], 'M': prim[1], 'MB': prim[2], 'MBQ': list(sec[1]), 'T': prim[3], 'S': prim[4], 'Xc': prim[5], 'Yc': prim[6]}).sort_values(by='Path')

fig, (ax1, ax2, ax3) = plt.subplots(1, 3)

for ax, key in zip((ax1, ax2, ax3), ('M', 'MB', 'MBQ')):
    scatter_and_label(ax, key, stresses)

stresses.to_excel("stress_ratios.xlsx")

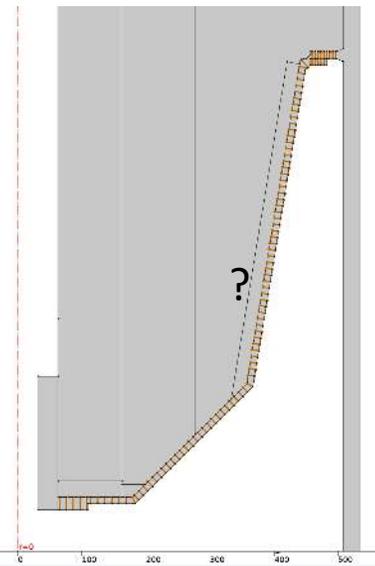
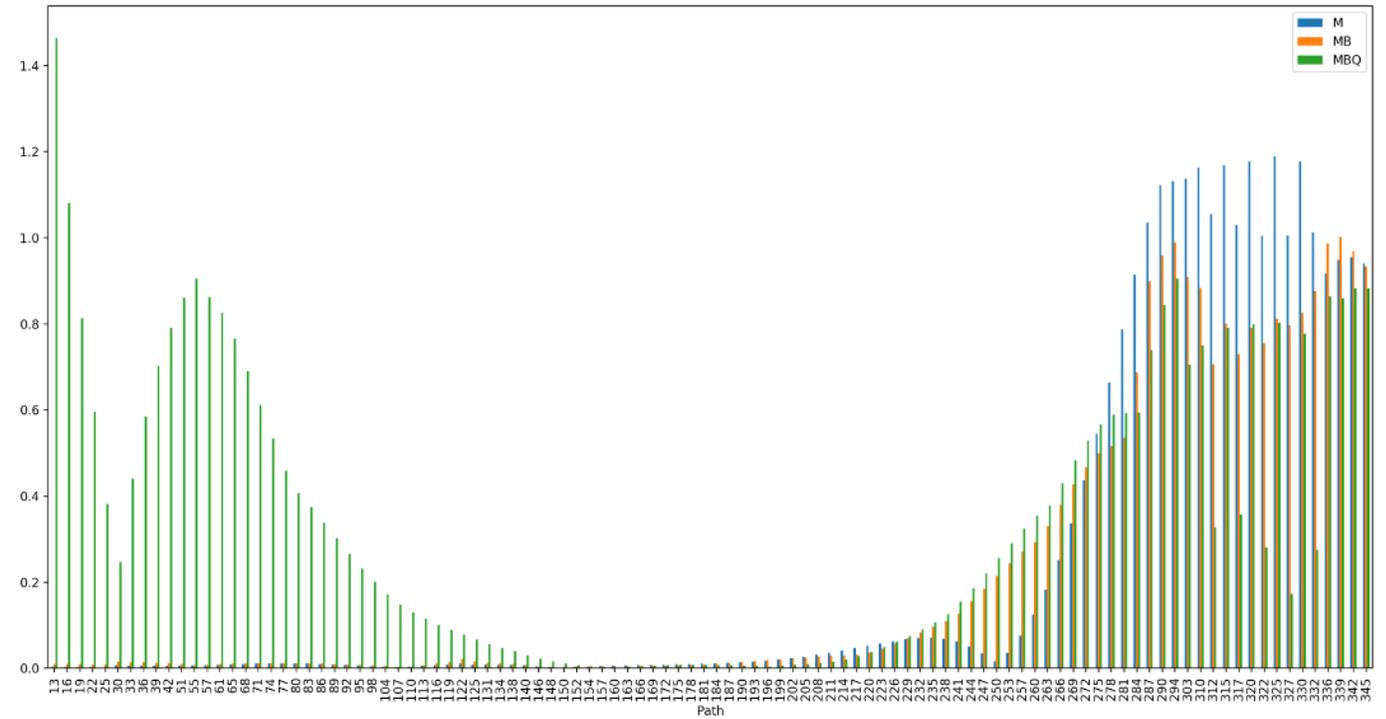
plt.show()
```

	A	B	C	D	E	F	G	H	I	J	K	L
1		Path	M	MB	MBQ	T	S					
2	0	13	0.00312	0.009584	1.464698	64.01121	138					
3	2	16	0.002646	0.009268	1.080444	96.61334	138					
4	4	19	0.002304	0.009123	0.812541	124.1749	138					
5	6	22	0.002154	0.008887	0.595443	147.4793	138					
6	8	25	0.001696	0.008666	0.380978	167.1049	138					
7	10	30	0.005954	0.014366	0.245692	184.4502	138					
8	12	33	0.005574	0.014112	0.440531	210.2039	138					
9	14	36	0.005217	0.01353	0.584814	231.1863	137.2576					
10	16	39	0.005051	0.012653	0.702296	248.4666	135.184					
11	18	42	0.004999	0.011386	0.791812	262.8719	132.9405					
12	20	51	0.004982	0.009668	0.861053	274.9774	131.0036					
13	22	55	0.005977	0.006908	0.904731	285.4707	129.7435					
14	24	57	0.006004	0.00795	0.862522	289.0218	129.3174					
15	26	61	0.007454	0.009359	0.825049	294.6992	128.6361					
16	28	65	0.008775	0.010372	0.765372	299.5491	128.0541					
17	30	68	0.009779	0.011067	0.690098	303.9942	127.5207					
18	32	71	0.010477	0.011488	0.611191	308.4972	126.9803					
19	34	74	0.010876	0.011665	0.533286	312.8981	126.4522					
20	36	77	0.010983	0.011616	0.458933	317.1163	125.946					
21	38	80	0.010806	0.011346	0.406184	321.1085	125.467					
22	40	83	0.010354	0.010851	0.37393	324.8503	125.018					
23	42	86	0.00963	0.01011	0.338214	328.3301	124.7336					
24	44	89	0.008661	0.009117	0.301667	331.5431	124.4765					
25	46	92	0.007469	0.008597	0.265876	334.4883	124.2409					
26	48	95	0.00608	0.007908	0.231926	337.1677	124.0266					
27	50	98	0.004529	0.006881	0.20052	339.5857	123.8331					
28	52	104	0.002862	0.005447	0.172064	341.7506	123.6599					
29	54	107	0.002789	0.003551	0.147819	343.6763	123.5059					
30	56	110	0.003489	0.003419	0.130582	345.3792	123.3697					
31	58	113	0.005263	0.006683	0.114851	346.8777	123.2498					
32	60	116	0.006885	0.0103	0.100818	348.1892	123.1449					
33	62	119	0.008241	0.01421	0.088569	349.3301	123.0536					
34	64	122	0.010341	0.022229	0.077693	350.3154	122.9622					
35	66	125	0.008951	0.01561	0.065991	351.1579	122.8611					
36	68	131	0.008589	0.01315	0.05567	351.867	122.776					
37	70	134	0.007945	0.01119	0.046967	352.48	122.7024					
38	72	138	0.007111	0.009753	0.039608	353.03	122.6364					
39	74	140	0.00599	0.007939	0.03226	353.5549	122.5724					

his PC > storage (\nas) (Z:) > Tore > networkevent > code

Name	Date modified
.idea	15/09/2020 11:54
code.iml	15/09/2020 10:56
PngToGif.py	18/03/2020 10:52
scrape.py	15/09/2020 11:15
stress_ratios.xlsx	15/09/2020 11:54
StressEval.py	15/09/2020 11:43
StressEvalMinimal.py	15/09/2020 11:54

- We can see that ie path #13 is above the allowable
- We then have to go back into Comsol to figure out where it is and how to mitigate it
- This is not efficient...



- ...instead we go back into Comsol and add a location of the individual SCL's to the export script
- *If less time constrained we might have redone our approach and used 100 stress linearizations directly in Comsol –rather than 1 that was reevaluted – but hindsight is always 20/20...*

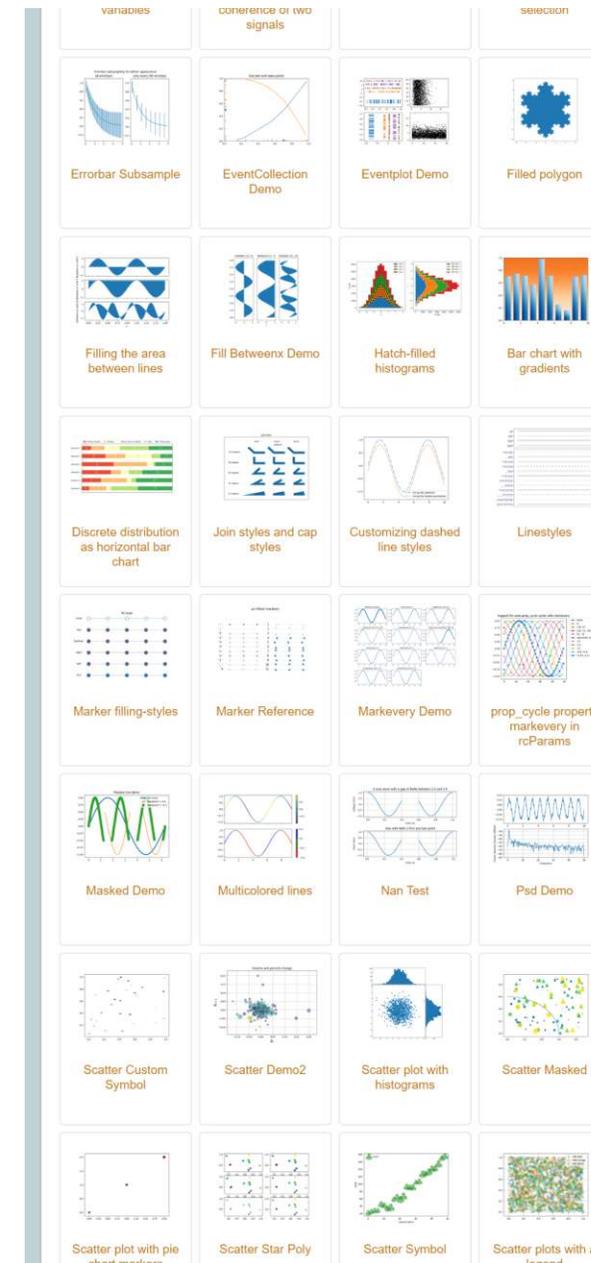
The screenshot shows the Comsol Multiphysics interface. The 'Settings' window is open to the 'Data' tab, where the 'Export' button is visible. The 'Label' is set to 'Data 3' and the 'Dataset' is 'Primary/Solution 1 (1) (sol1)'. Below this, the 'Expressions' section contains a table with the following data:

Expression	Unit	Description
pathnom2		
RM2	1	
RMB2	1	
T1	degC	
S2	MPa	
R_coor	mm	
Z_coor	mm	

Below the table, a large data table is displayed, showing numerical results for various expressions. The table has two columns for expression names and two columns for numerical values. The data is organized in a grid format, with rows corresponding to the expressions listed in the table above.

...enter* Matplotlib

- (*...not quite true, because pandas has been using it under the hood in our previous example, anyways...)
- Matplotlib is a powerful plotting tool for static plots –
- and the plotting engine for many higher level modules
 - Pandas
 - Seaborn
- For interactivity *bokeh*, *plotly* etc are gaining traction – but Matplotlib is still very much alive and kicking!



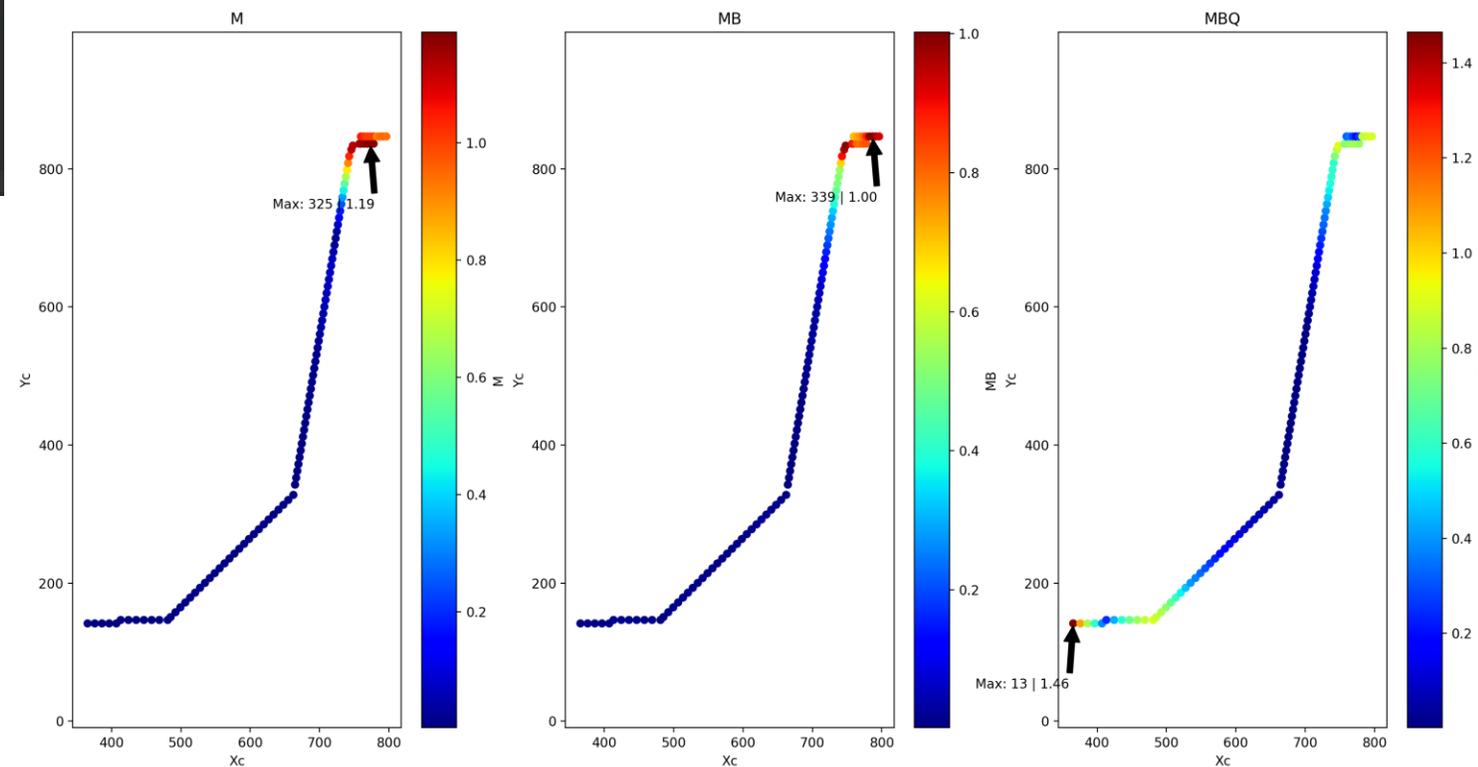
```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 from matplotlib import cm
4
5 cmap = cm.get_cmap('jet')
6
7
8 def scatter_and_label(axis, key, frame):
9     frame.plot('Xc', 'Yc', kind='scatter', ax=axis, s=40, linewidth=0,
10              c=key, colormap=cmap)
11
12     m = frame[frame[key] == frame[key].max()]
13     s=f"Max: {int(m['Path'].values[0])} | {m[key].values[0]:.2f}"
14     axis.annotate(s, (m['Xc'], m['Yc']), xycoords='data', xytext=(-75, -50), textcoords='offset points', arrowprops=dict(facecolor='black', shrink=0.05))
15
16     axis.axis('equal')
17     axis.set_title(key)
18
19 df = pd.read_csv(r"Z:\Tone\networkevent\eval2.txt", header=None, delim_whitespace=True)
20
21 prim = df.iloc[:,2:]
22 sec = df.iloc[:,1:]
23
24 stresses = pd.DataFrame({'Path': prim[0], 'M': prim[1], 'MB': prim[2], 'MBQ': list(sec[1]), 'T':
25
26 fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
27
28 for ax, key in zip((ax1, ax2, ax3), ('M', 'MB', 'MBQ')):
29     scatter_and_label(ax, key, stresses)
30
31 plt.show()

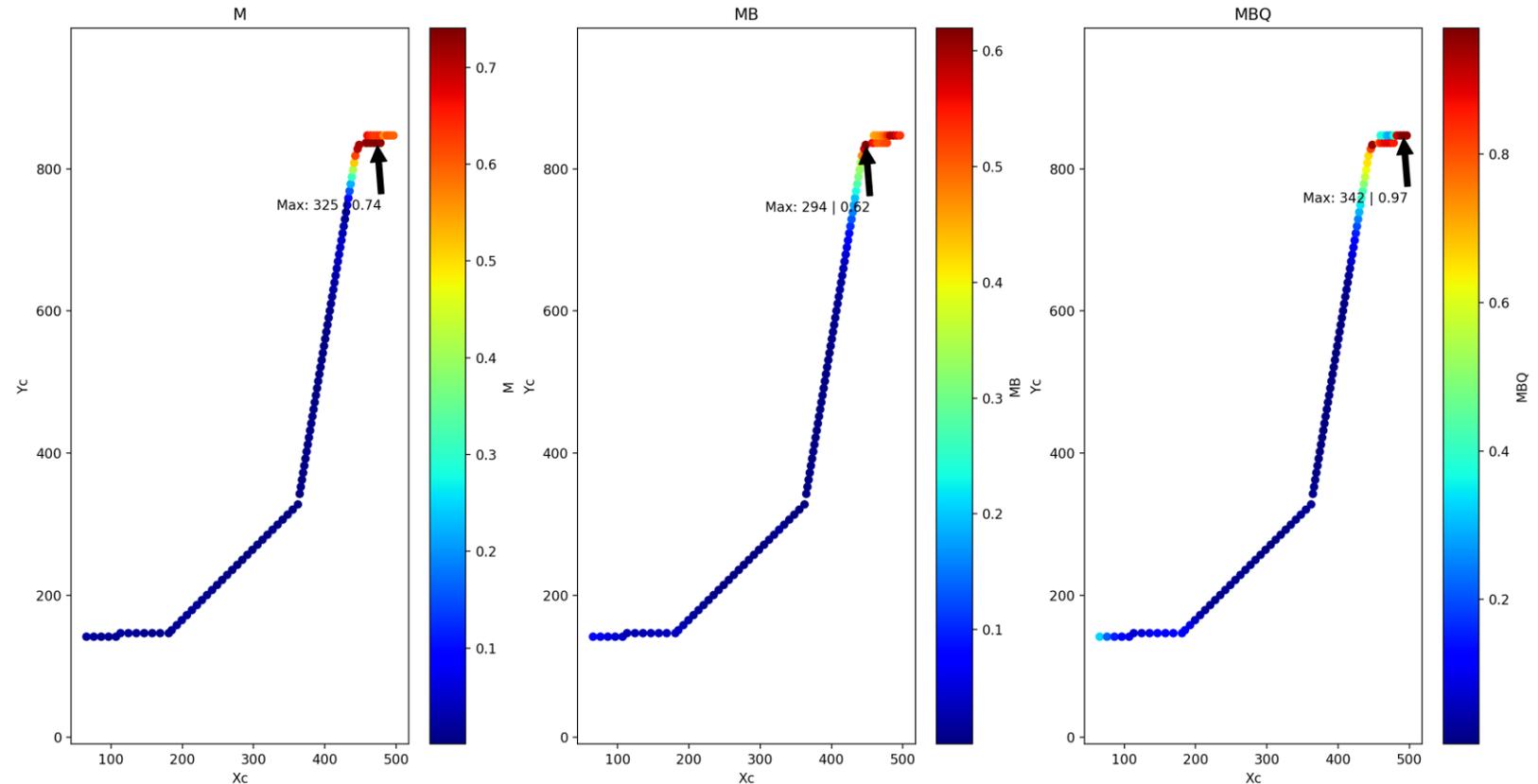
```

Add ~10 lines and we have a plot than we can actually interpret

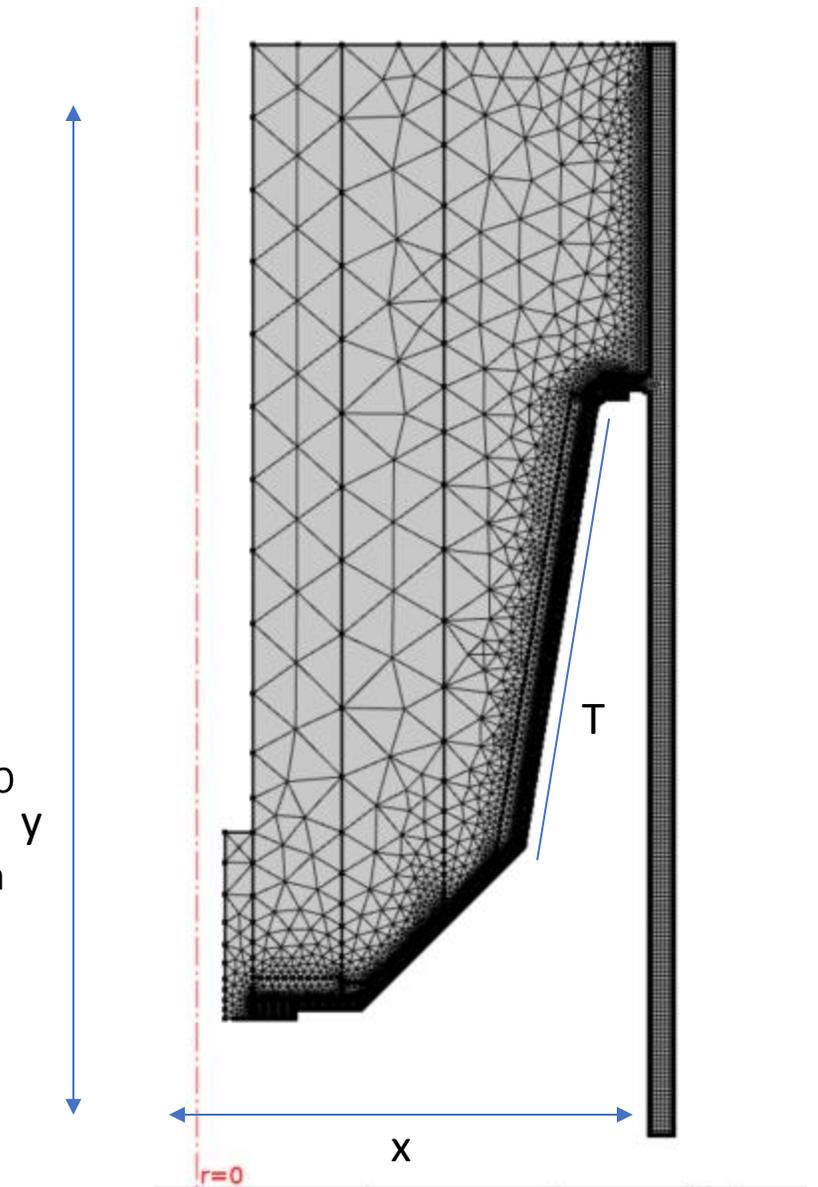
My colleague who used the script found the arrow quite useful for quickly spotting errors – it was a quick visual que to the location of high stress-ratios



- For 1 plot this might be somewhat tedious – but when you modify your design it starts to pay off
- Reduce radius by 300 mm
- Rerun model and script
- – ok for 1hrs worth of coding...
- For this specific project only 5-10 design iterations where necessary, so in terms of day-to-day scripting we stopped here
- But we often face problems, where an acceptable design is harder to find – let's build a bit more on this example...



- We've created a model that solves in ~10s
- Let's utilize this to investigate the solution-space
- Parameterize
 - Cone-Height
 - Diameter
 - Convective coeff at underside
 - Corresponds to adding thermal protection
- $10 \times 10 \times 10 = 1000$ designs to evaluate
- This can be accomplished through a nested parametric sweep in Comsol, or some 'execution scripting'
 - I.e. run the simulation tool in batch-/command model from python
- We get 1000 text-files from Comsol
- For the purpose of this discussion I've 'cheated' and used a small script to perturb the original data to create 1000 'synthetic' datasets



```

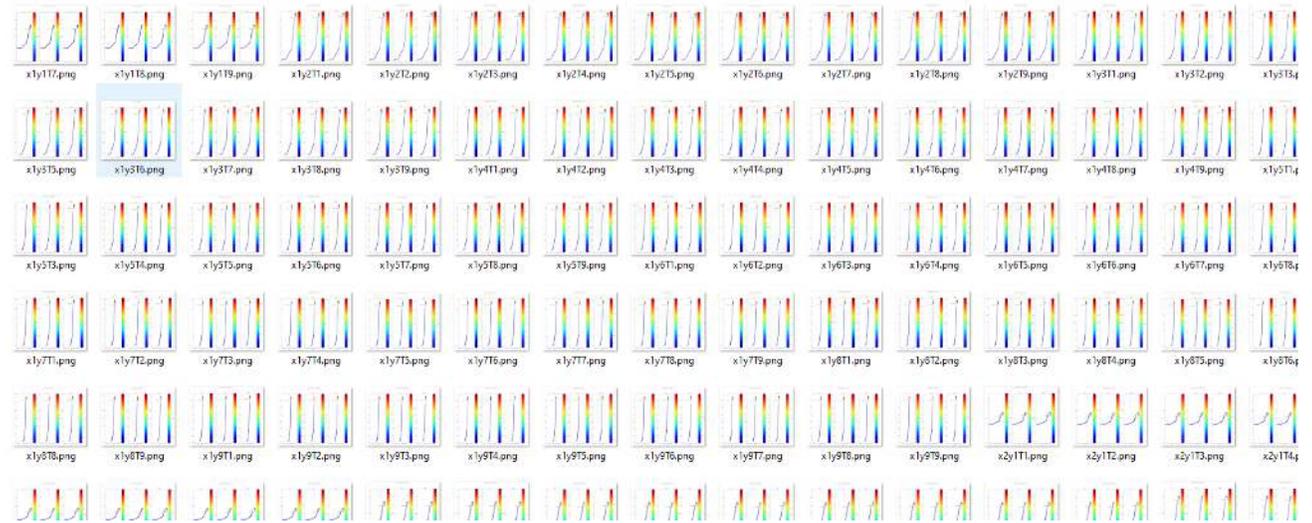
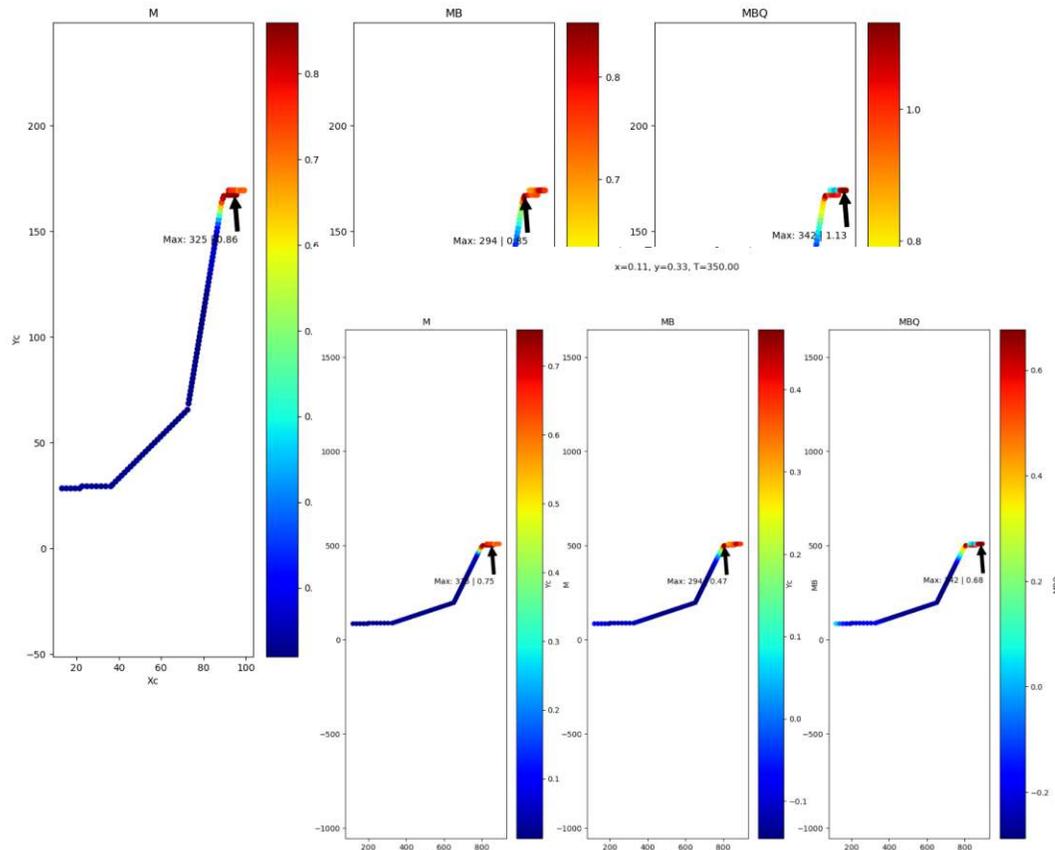
s=stresses.copy()
for x in range(1,r):
    s['Xc']=stresses['Xc']*x*0.2
    for y in range(1,r):
        s['Yc']=stresses['Yc']*y*0.2

        for T in range(1,r):
            s['M']=stresses['M']+random.ur
            s['MB']=stresses['MB']+random.

```

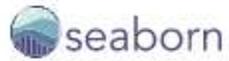
x=1.00, y=1.00, T=350.00

- Wrap our small post-proc script in 3 for-loops
- change the plotting code to write to .png-file instead of showing the plot

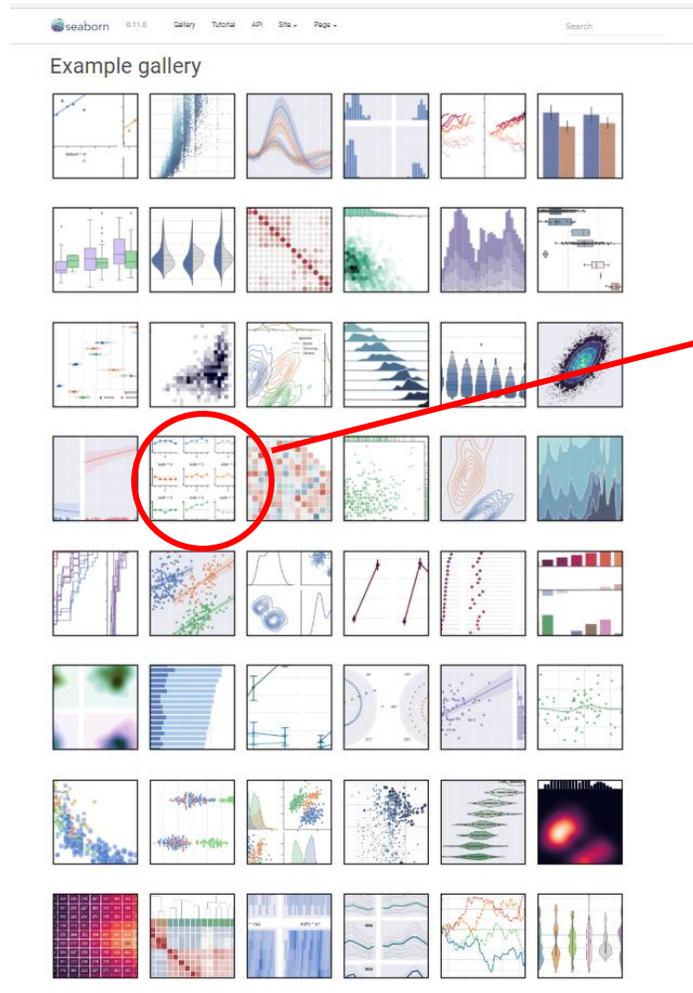


We get 1000 plots – not ideal for making design decisions...

...enter Seaborn



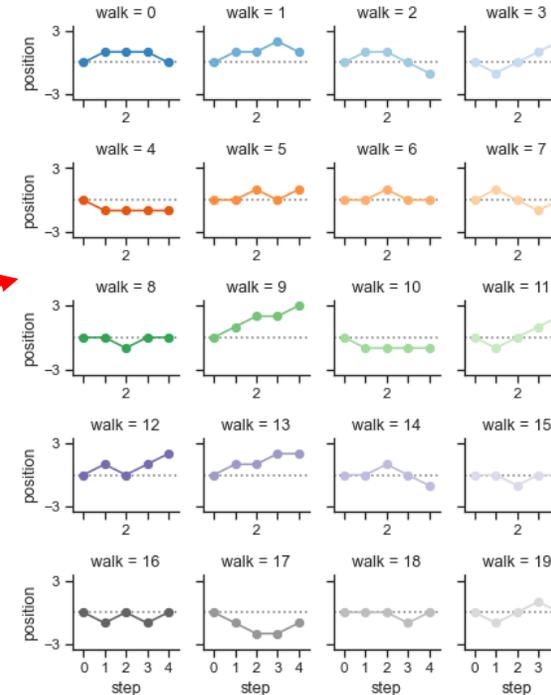
Seaborn is a higher-level plotting library based on Matplotlib



© Copyright 2010-2020, Michael Waskom. Created using Sphinx 2.3.1.

Back to top

Plotting on a large number of facets



Maybe a facet-grid'ed lineplot can shed some light on our data

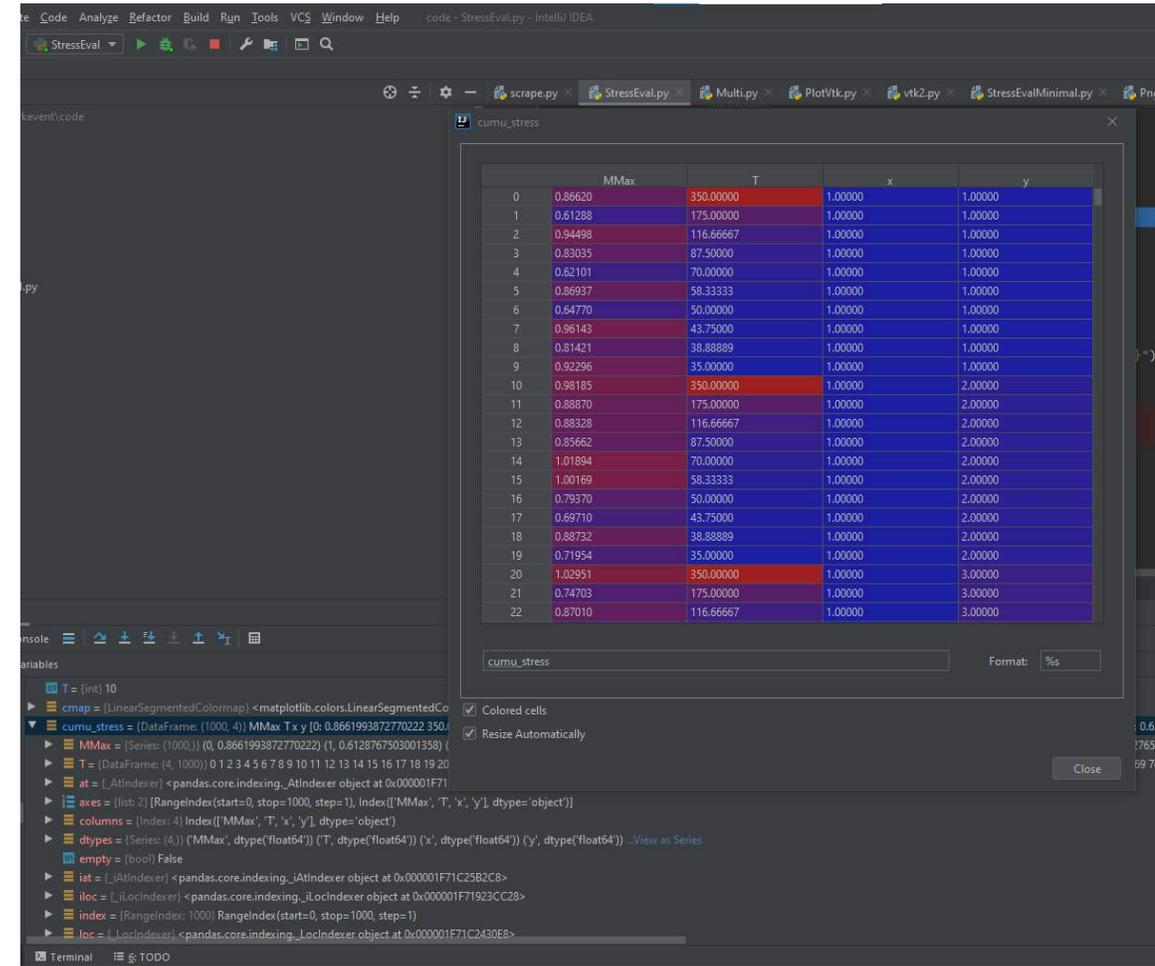
seaborn components used: `set_theme()`, `FacetGrid`

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="ticks")
```

Create a dataset with many short random walks

- We need to a bit of dimensionality-reduction
 - For want of a better idea, we simply take max of M,MB,MBQ across all paths for every design
- Instead of plotting inside our 3-nested for loops we collect the max in a new dataframe in a new 'cumu_stress'
- IntelliJ/Pycharm allows us to inspect this dataframe while we get it right



```
cumu_stress=cumu_stress.append({'x': x, 'y': y, 'T': 350 * 1 / T, 'MMax':s[['M', 'MB', 'MBQ']].max().max(), ignore_index=True})
```

- Now we modify the seaborn example slightly – the essence is seen here

seaborn components used: set_theme(), FacetGrid

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="ticks")

# Create a dataset with many short random walks
rs = np.random.RandomState(4)
pos = rs.randint(-1, 2, (20, 5)).cumsum(axis=1)
pos -= pos[:, 0, np.newaxis]
step = np.tile(range(5), 20)
walk = np.repeat(range(20), 5)
df = pd.DataFrame(np.c_[pos.flat, step, walk],
                  columns=["position", "step", "walk"])

# Initialize a grid of plots with an Axes for each walk
grid = sns.FacetGrid(df, col="walk", hue="walk", palette="tab20c",
                    col_wrap=4, height=1.5)

# Draw a horizontal line to show the starting point
grid.map(plt.axhline, y=0, ls=":", c=".5")

# Draw a line plot to show the trajectory of each random walk
grid.map(plt.plot, "step", "position", marker="o")

# Adjust the tick positions and labels
grid.set_xticks(np.arange(5), yticks=[-3, 3],
               xlim=(-.5, 4.5), ylim=(-3.5, 3.5))

# Adjust the arrangement of the plots
grid.fig.tight_layout(w_pad=1)
```

```
grid = sns.FacetGrid(cumu_stress, row="y", col="x", hue="y", palette="tab20c",
                    margin_titles=True, despine=False)

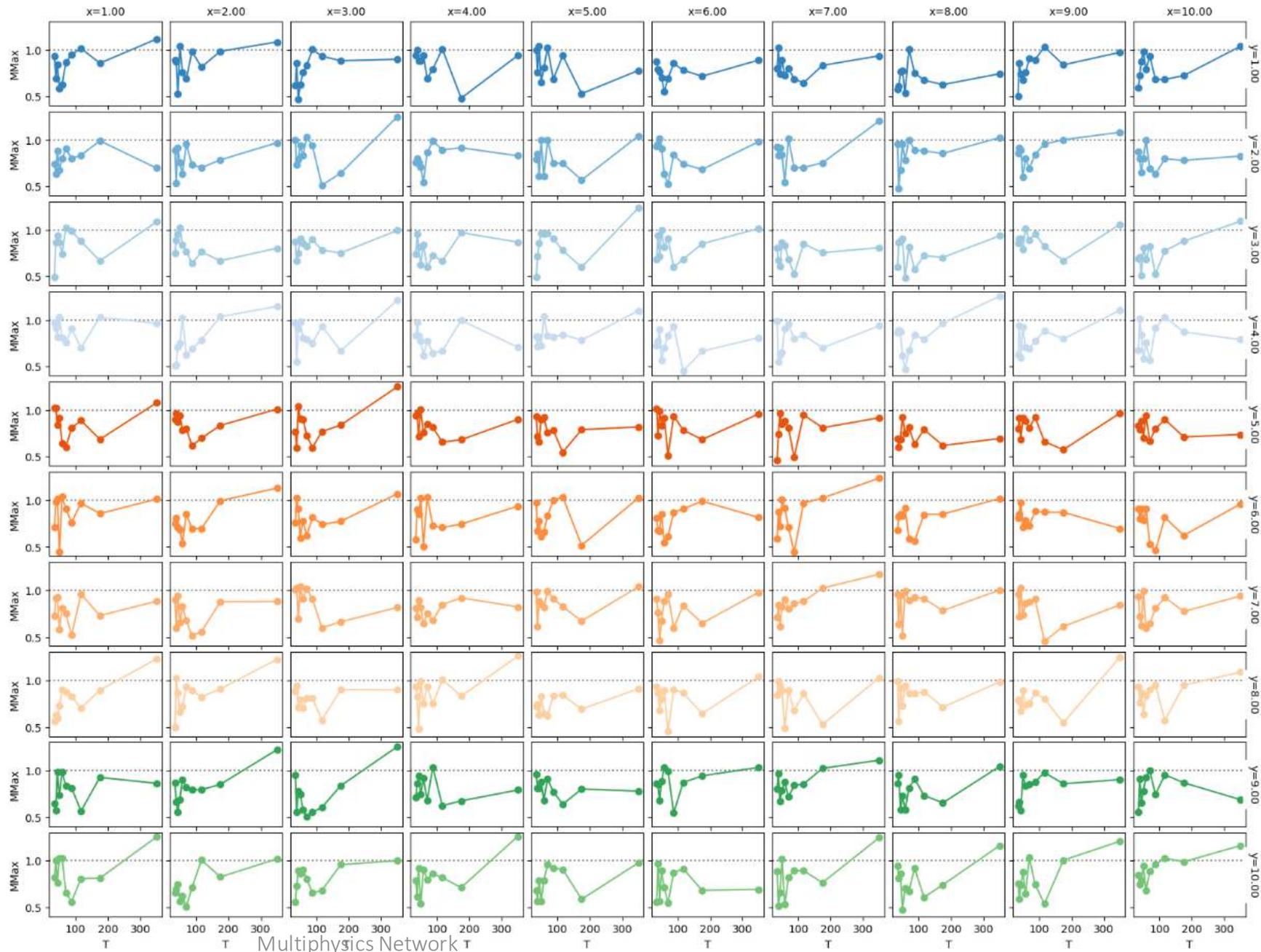
grid.map(plt.axhline, y=1.0, ls=":", c=".5")

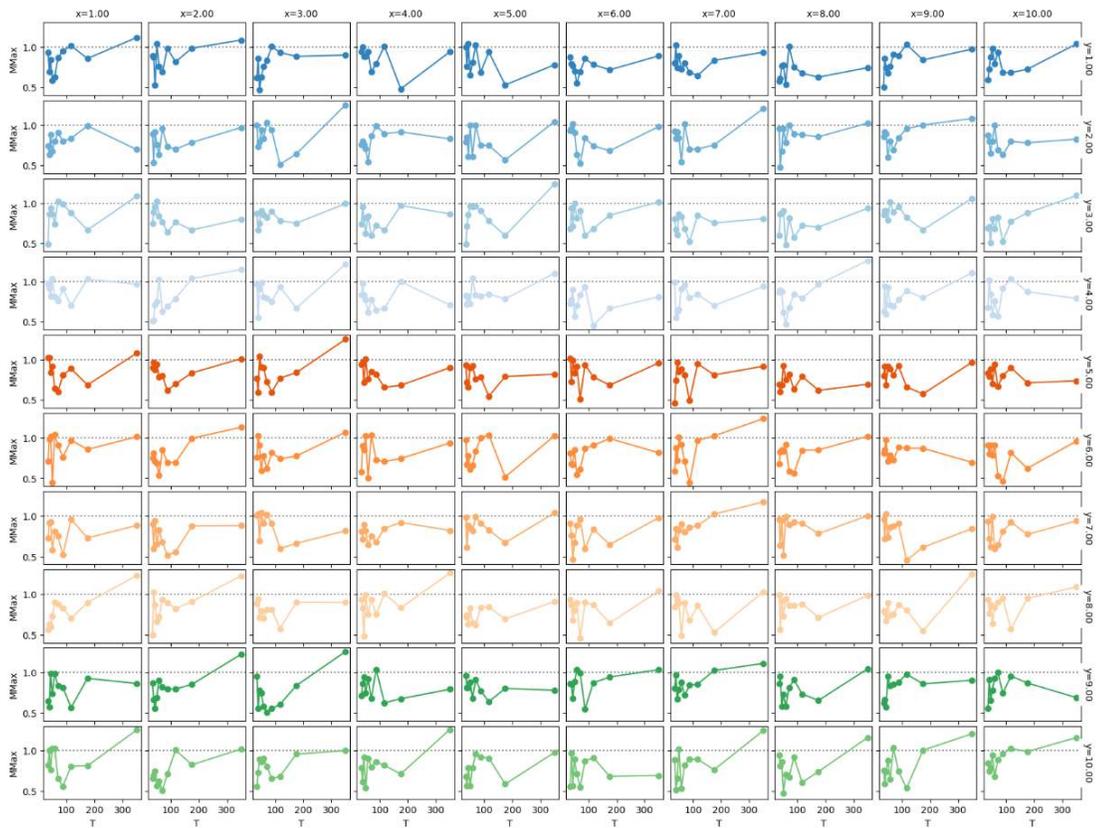
grid.map(plt.plot, "T", "MMax", marker="o")

grid.set_titles(col_template="x={col_name:.2f}", row_template="y={row_name:.2f}")
```

1000 sim runs in one plot

- x-horizontally
- y-vertically
- T- Mmax -on every 'subplot'





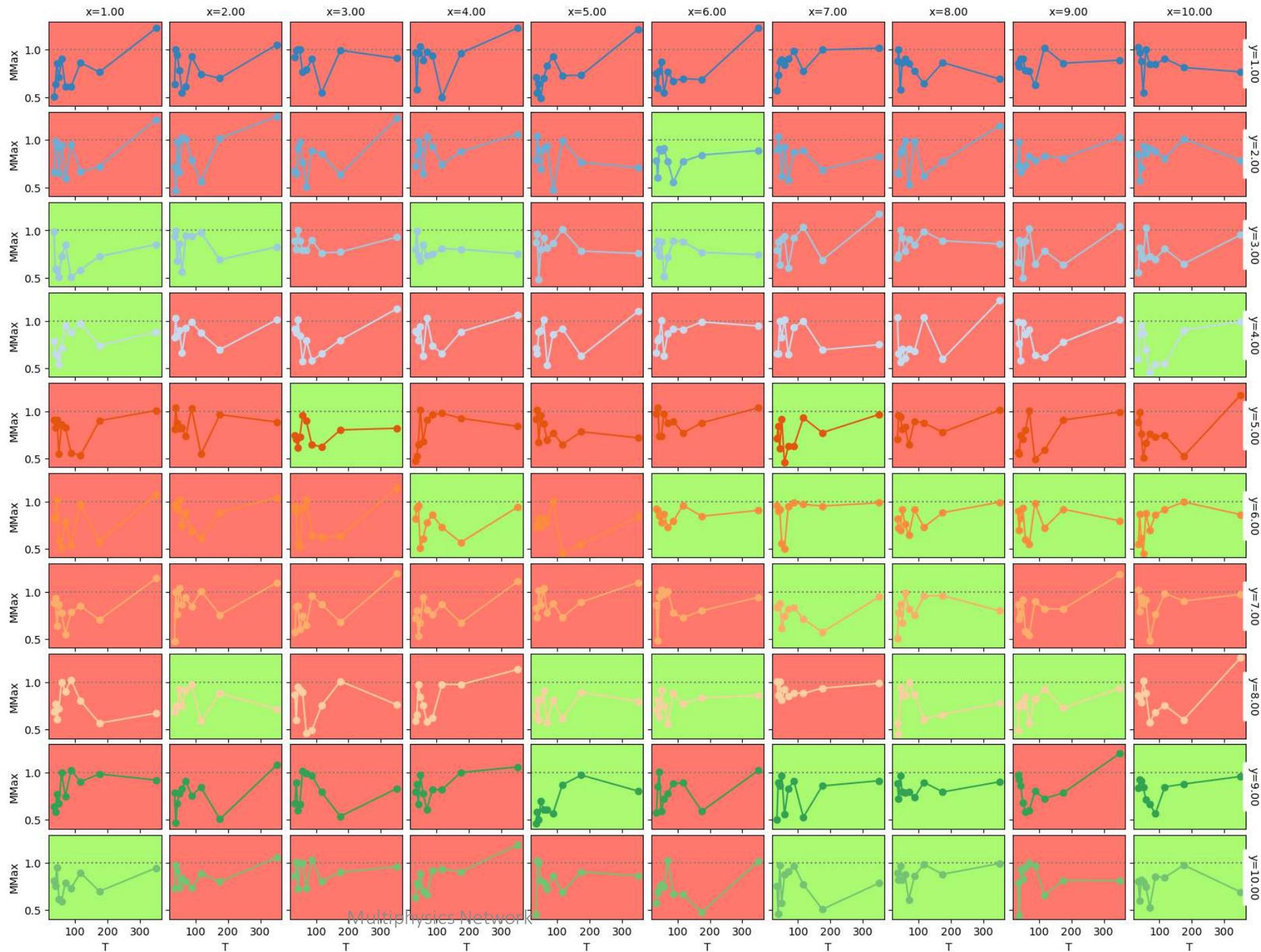
- Still quite dense to look at – can we maybe add some color to distinguish the x,y values that have a design that is ok?
- A few lines to loop over all the plots

```

78
79     for i, ax in enumerate(grid.axes.flat):
80         if ax.get_lines()[1].get_data()[1].max()>=1.0:
81             ax.set_facecolor((1.0, 0.47, 0.42))
82         else:
83             ax.set_facecolor('xkcd:spring green')
84

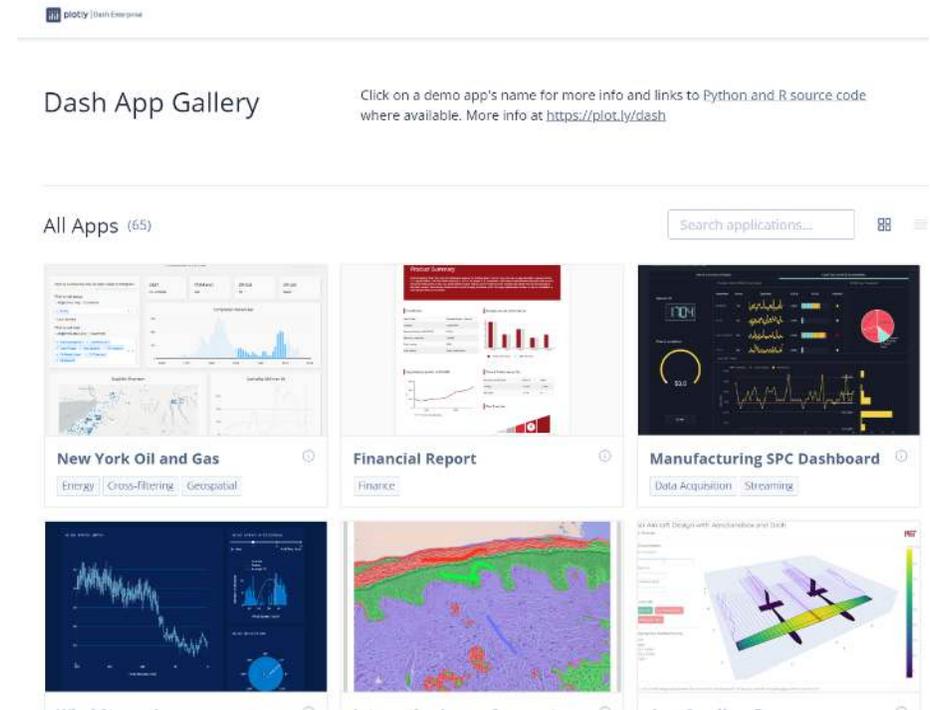
```

- A large diameter design, with a large z seems to perform best
- We could definitely dive deeper into this data
 - A heatmap would probably work better...
- – maybe next time...

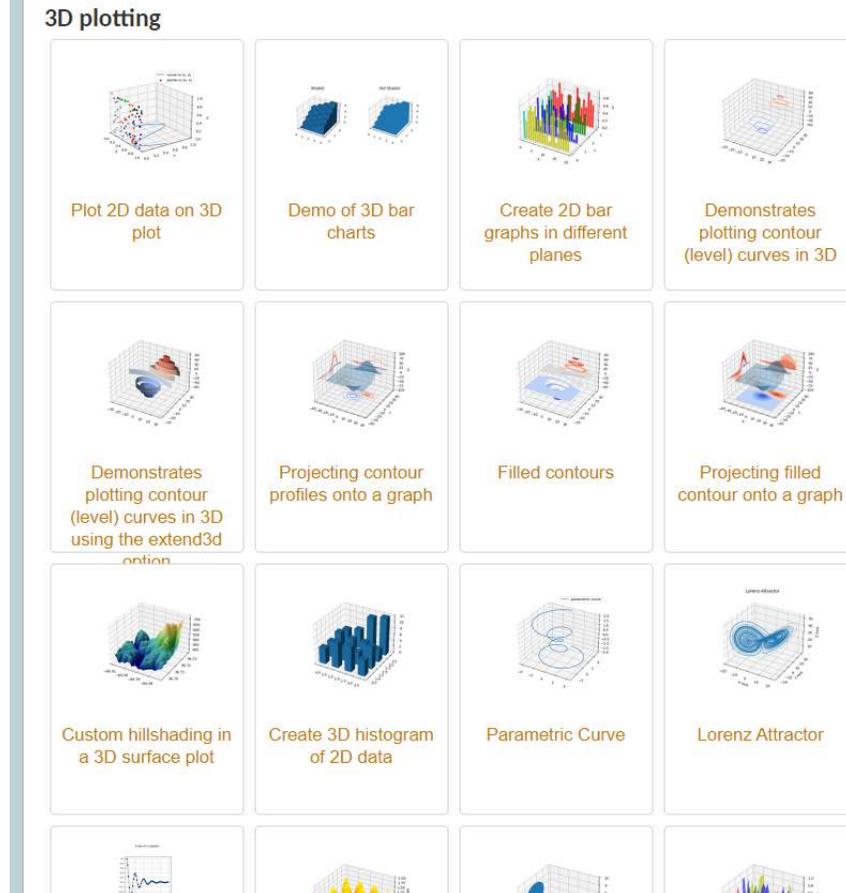


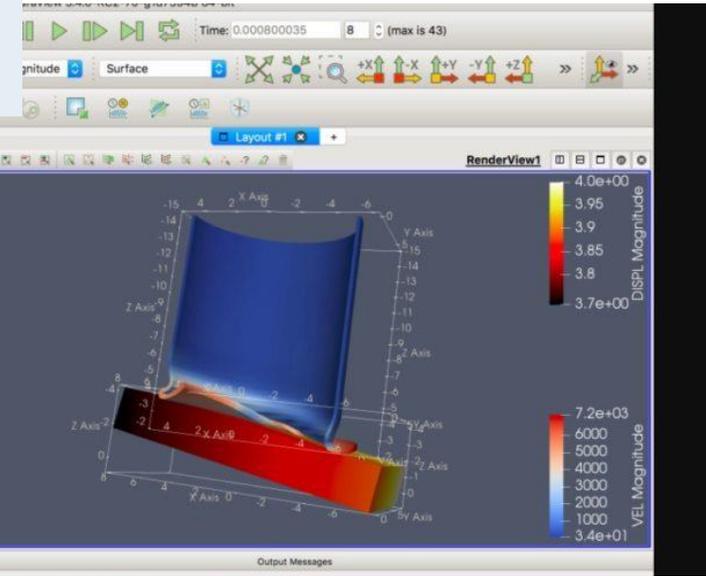
A note on large higher dimensional datasets

- Maybe we would also like to vary the thickness.. – since we could still run the models over a night or 2
- Then we have 4 parameter dimensions
- Reducing this to a 2d plot becomes tricky –
- we could compensate by adding some interactivity
 - I.e. Clicking subplots takes you to a detail view of the design
 - Plotly has a nice module for that called Dash
 - ...again something for another talk...



- What if we want to plot 3D?
 - alá what we see in the post processors in our simulation tools
- Matplotlib has 3d plotting capabilities – but it's not its strongest suit
 - It's mainly x,y,z scatter – x,y bar etc
 - 3d plots will quickly become very slow

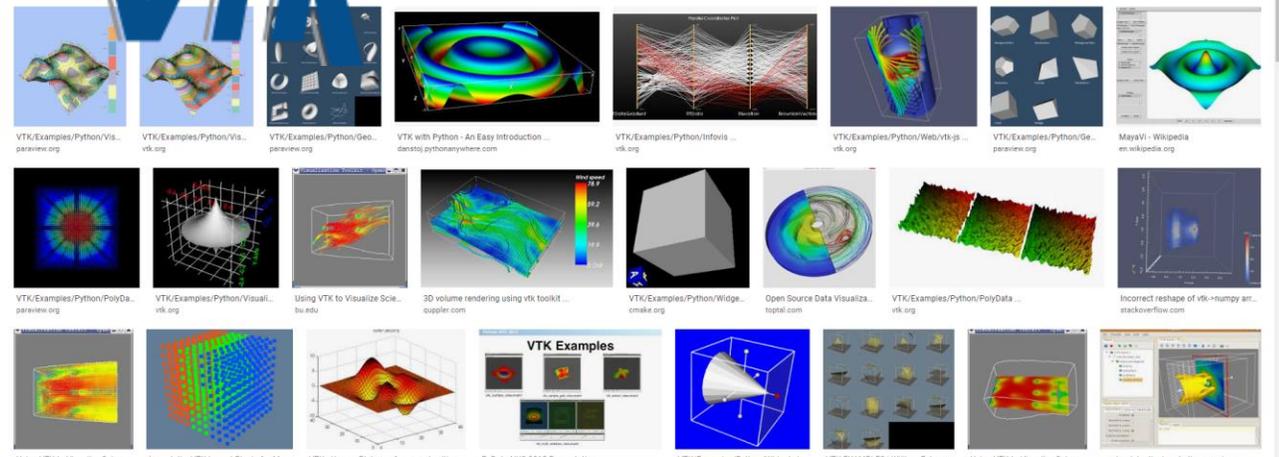




...enter VTK



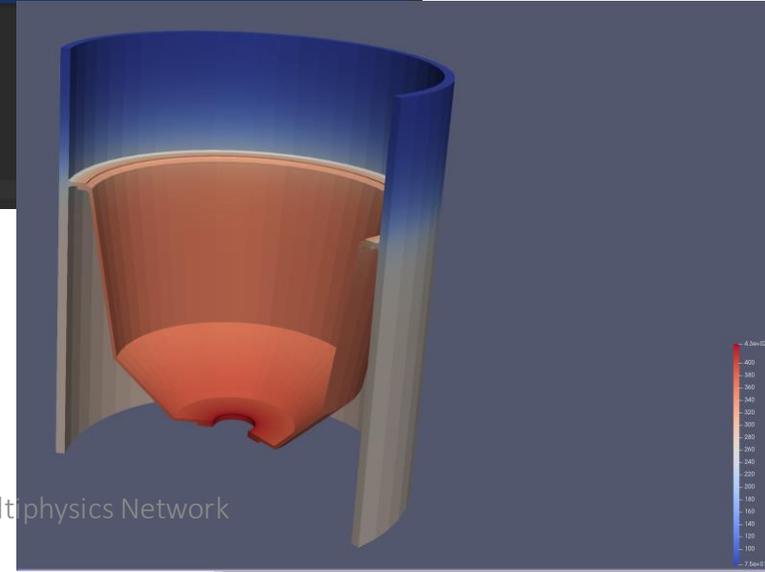
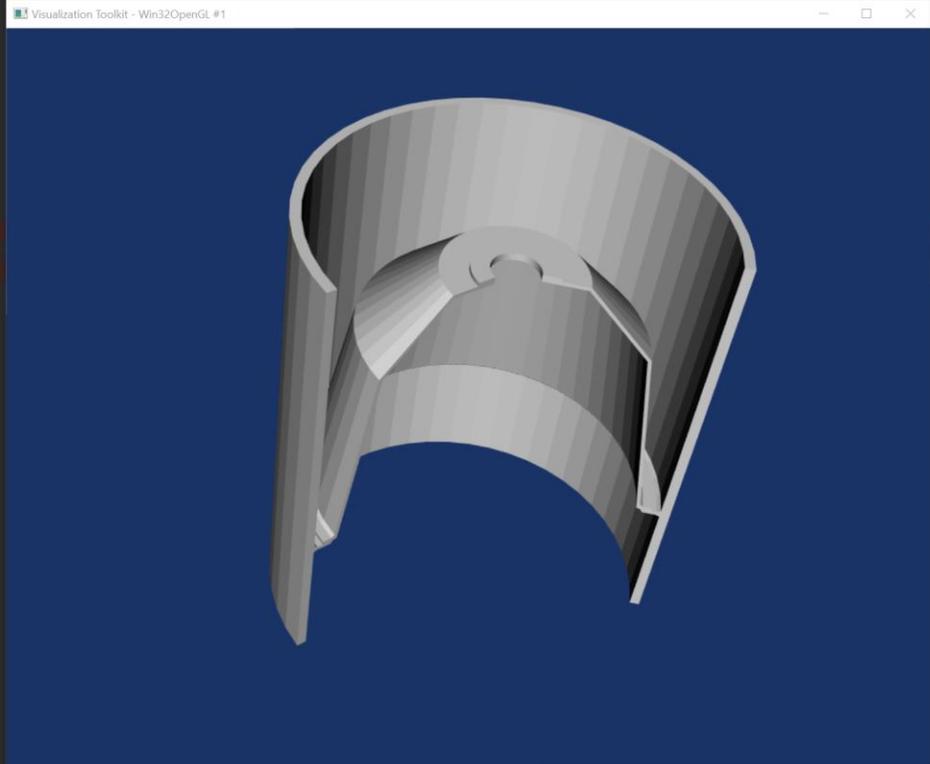
- Maybe best known as the underlying engine for Paraview
- Very powerful 3d plotting engine
- With good, well documented python bindings



```

1 import vtk
2
3 # The source file
4 file_name = r"C:\Users\ToreFindsen\Desktop\temp\pressure_shell.vtu"
5
6 # Create a custom lut. The lut is used both at the mapper and at the
7 # scalar_bar
8 # lut = vtk.vtkLookupTable()
9 # lut.Build()
10
11 # Read the source file.
12 reader = vtk.vtkXMLUnstructuredGridReader()
13 reader.SetFileName(file_name)
14 reader.Update()_# Needed because of GetScalarRange
15 output = reader.GetOutput()
16 scalar_range = output.GetAttributes(0).GetArray(0).GetRange()
17
18 mapper = vtk.vtkDataSetMapper()
19 if vtk.VTK_MAJOR_VERSION <= 5:
20     mapper.SetInput(output)
21 else:
22     mapper.SetInputData(output)
23 mapper.SetScalarRange(scalar_range)
24 # mapper.SetLookupTable(lut)
25
26 actor = vtk.vtkActor()
27 actor.SetMapper(mapper)
28
29 renderer = vtk.vtkRenderer()
30 renderer.AddActor(actor)
31 renderer.SetBackground(0.1, 0.2, 0.4)
32
33 render_window = vtk.vtkRenderWindow()
34 render_window.AddRenderer(renderer)
35 render_window.SetSize(300, 300)
36
37 interactor = vtk.vtkRenderWindowInteractor()
38 interactor.SetRenderWindow(render_window)
39
40 # create the scalar_bar
41 # scalar_bar = vtk.vtkScalarBarActor()
42 # scalar_bar.SetOrientationToHorizontal()
43 # scalar_bar.SetLookupTable(lut)
44 #
45 # # create the scalar_bar_widget
46 # scalar_bar_widget = vtk.vtkScalarBarWidget()
47 # scalar_bar_widget.SetInteractor(interactor)
48 # scalar_bar_widget.SetScalarBarActor(scalar_bar)
49 # scalar_bar_widget.On()
50
51 interactor.Initialize()
52 render_window.Render()
53 interactor.Start()

```



- If we export or geometry, mesh and/or results in something open – ala STL, VTU, NASTRAN
- We can easily read and visualize it in VTK
- ~50 lines of code and you have a fully interactive example!
- Or you can save the plots to files as in the previous example
- And remember – we can do much more than simple visualization – we can aggregate, morph and reduce the data ad nasuem

A more contrived example

- Our project manager would now like us to create a Power Point presentation with all the plots (1000) from our parametric study – placed 2 on each slide
- It's a slightly absurd request – but it allows us to dive into some other nice scripting facilities

glob — Unix style pathname pattern expansion

Source code: [Lib/glob.py](#)



python-pptx

Release v0.6.18 ([Installation](#))

build passing

python-pptx is a Python library for creating and updating PowerPoint (.pptx) files.

```

import glob
import os

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
from pptx import Presentation
from pptx.util import Inches

prs = Presentation("Presentation1.pptx")
blank_slide_layout = prs.slide_layouts[6]

left = Inches(0.5)
top = Inches(1)
height = Inches(5.5)

folder = r"Z:\Tore\networkevent\code"

geoms = glob.glob(folder + "\*.png")
geoms1=geoms[:len(geoms)//2]
geoms2=geoms[len(geoms)//2:]

for idx,geom in enumerate(zip(geoms1,geoms2)):

    label = "{0}".format(geom[0])
    w=2*1088+800
    h=2*688

    new_im = Image.new('RGBA', (w, h))

    new_im.paste(Image.open(geom[0]), (0,0),mask=Image.open(geom[0]))

    draw = ImageDraw.Draw(new_im)
    new_im.paste(Image.open(geom[1]), (1088+400,0),mask=Image.open(geom[1]))

    draw = ImageDraw.Draw(new_im)
    # font = ImageFont.truetype(<font-file>, <font-size>)
    font = ImageFont.truetype(r"arialbd.ttf", 70)
    # draw.text((x, y),"Sample Text", (r,g,b))
    draw.text((0, 0), label, fill=(0, 0, 0), font=font)

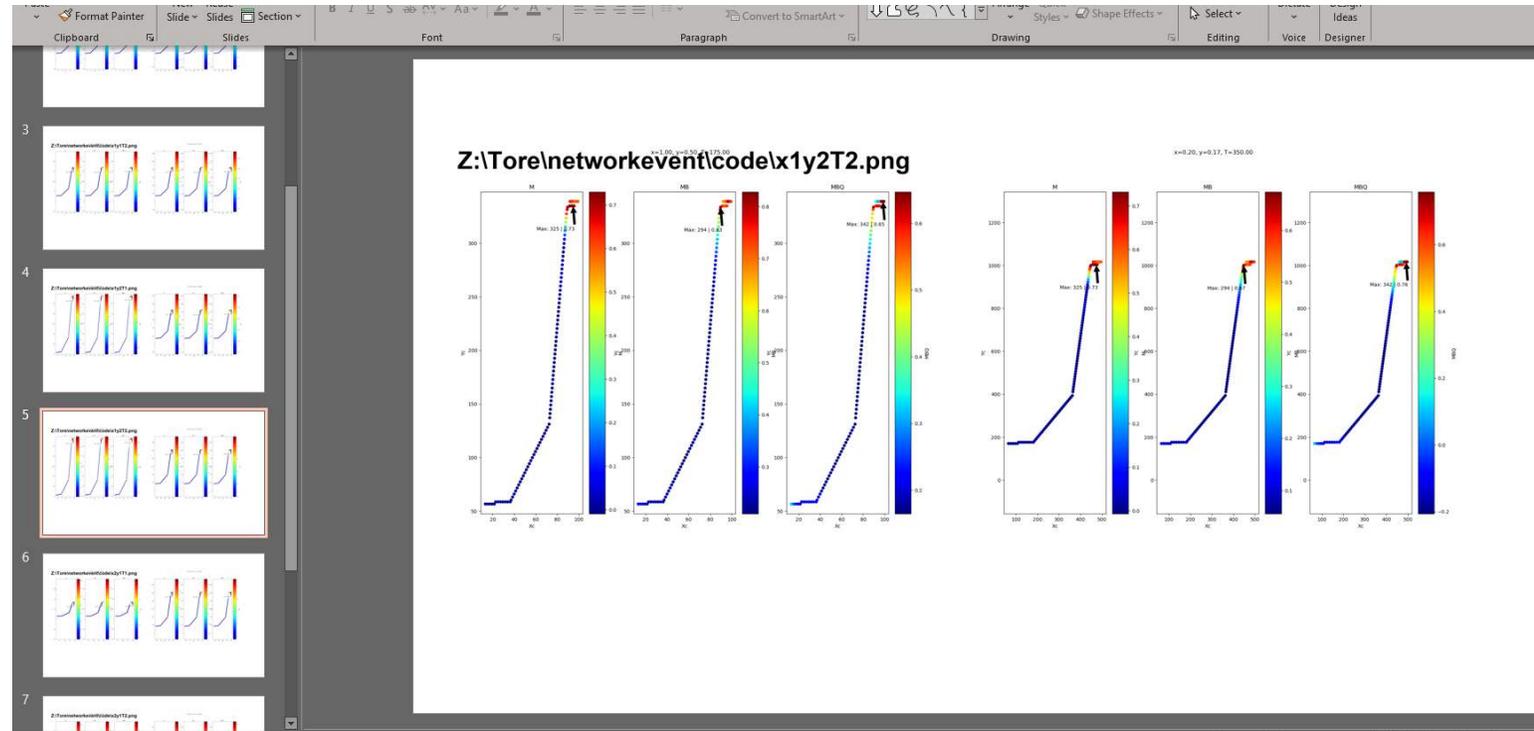
    filename=label.replace('.png','x.png')
    new_im.save(filename)

    slide = prs.slides.add_slide(blank_slide_layout)
    pic = slide.shapes.add_picture(os.path.abspath(filename), left, top, height=height)

```

17/09/2020

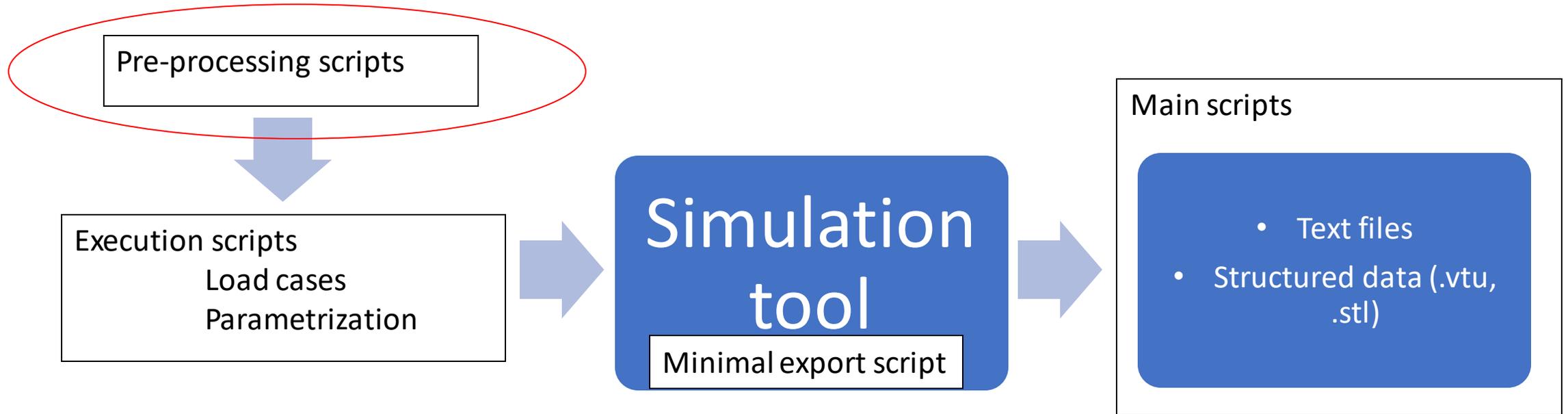
- Read template presentation
- Get all png files in folder and loop over them pairwise
- Construct a 2x1 image and add some text to it
- Insert in presentation
- Save



Take away:

- Doing file-system interaction, batch image processing and Power Point creation is also very accessible
- And we could have made a similar example in Word...

That concludes Case 1 – let's briefly look at some other cases



- So far our focus has been on post processing – we can script any aspect of our modelling workflow – a few examples

Case 2: Small scale pdf data extraction

- We were doing a Comsol App for Kavadrat Soft Cell, to predict acoustic sound absorption of their acoustic panels
- They had 40+ tests – but only as pdfs
- For validation of our model we needed
 - Product name
 - Build up
 - Test setup
 - Measurements data

Sound Absorption Coefficient ISO 354
Measurement of absorption in reverberation rooms

Client Test
Kvadrat SoftCells ceiling panel
Type: SoftCells BroadLine

Test Specimen
Kvadrat SoftCells BroadLine ceiling mounting
Self-stretched aluminium frame with one layer of textile and acoustic insulator
Height of frame 49 mm
Height of frame incl. insulator 49 mm

Front textile: Casa from Kvadrat
Back side textile: -
Module size B X L = 1310 x 2810

Test Build-Up (from top to bottom):
1 mm Front textile
9 mm Air gap
40 mm Mineral-glass acoustic insulator, density 55kg/m³
150 mm Air gap
Concrete floor of reverberation room

Mounting
Placed 150mm above the floor
3 modules placed without interspaces on a surrounding wooden frame
Total dimensions of the test object
B x L x H = 3930mm x 2810mm x 200mm

Room: Reverberation Room
Volume: 242 m³
Size: 11,04 m²
Date of test 02-04-2014

Frequency [Hz]	as 1/3 octave	ap octave
100	0.89	
125	0.63	0.80
160	0.87	
200	0.75	
250	0.90	0.90
315	1.04	
400	0.97	
500	0.91	0.95
630	0.92	
800	0.95	
1000	0.90	0.90
1250	0.92	
1600	0.89	
2000	0.88	0.90
2500	0.88	
3160	0.88	
4000	0.85	0.85
5000	0.85	

Sound Absorption Coefficient α

as Sound absorption coefficient to ISO 354
ap Practical sound absorption coefficient to ISO 11654

Rating according to ISO 11654:
Weighted Sound Absorption Coefficient $\alpha_w = 0,95$
Sound absorption class: A

kvadrat soft cells
Test conducted by Kvadrat SoftCells
Lipowa 5, 62-060 Steszew

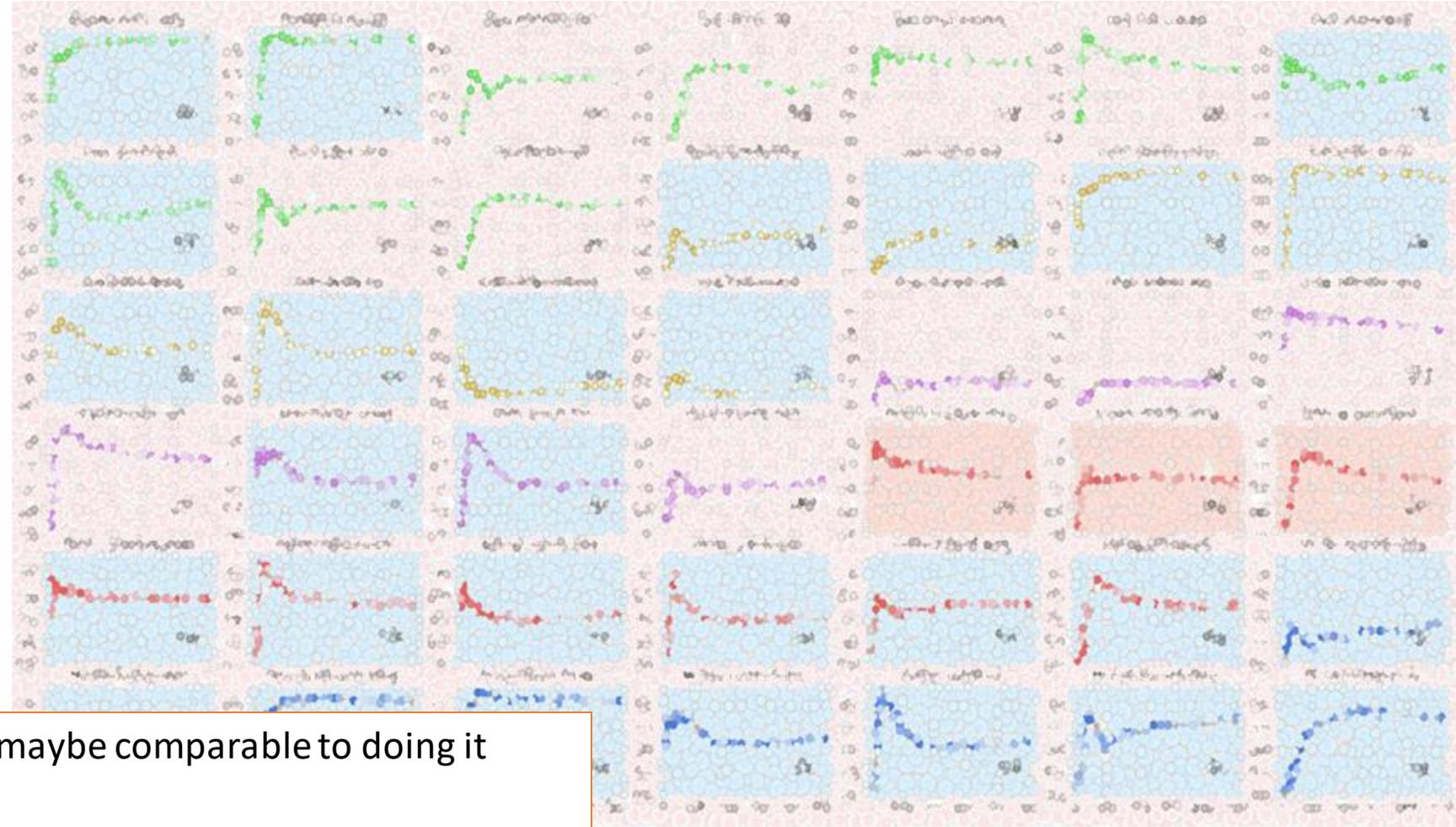
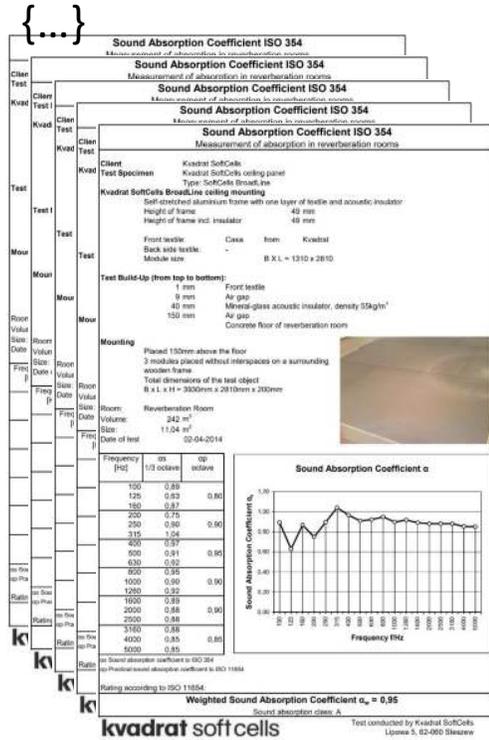
...enter PDFQuery

- The pdf is “reduced” to a hierarchy that can be searched
- <100 lines of code allows us to traverse this hierarchy and get the data we need

```
1 import csv
2 import os
3 import re
4
5 from pdfquery import pdfquery
6
7
8 def get_by_offset(pdf, word, dir):...
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 def get_abs_data(pdf, n1, n2):...
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Clients tests

- Absorption curves sorted by product, build-up and room



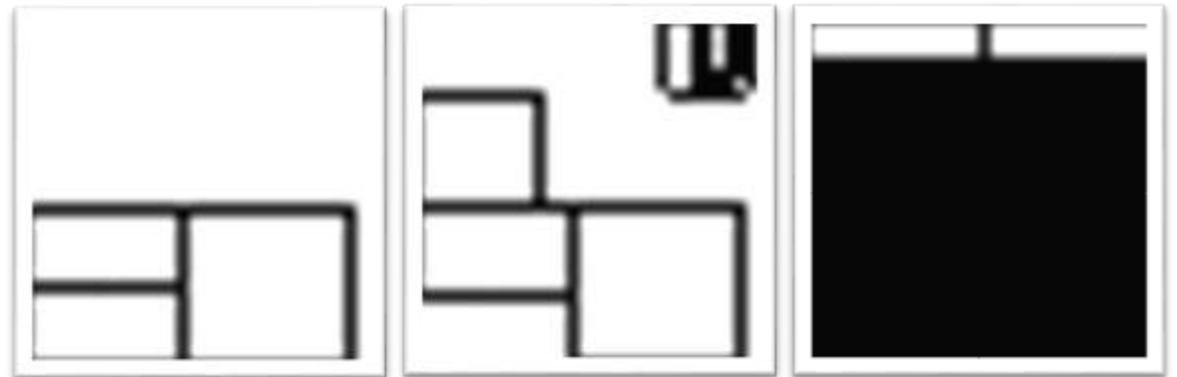
Blurred for IP reasons

For 40 pdfs the speed of scripting is maybe comparable to doing it manually
 But maybe they could find an extra 40 tests – then we could process them in 3 mins
 And more importantly avoiding copy-paste errors

17/09/2020

Case 3: Large scale pdf scraping using ML

- A recent project
- 400K purchase order-pdfs need processing with minimal manual work
 - But the purchases orders are done in different locations around the world, and have very different formats
- Approach
 1. Segment into categories
 1. Remove all text, save picture of background layout as 32x32 png
 2. Do K-means clustering in 32x32=1024 dimensions
 2. Use scraping logic customized per category – ala case 2



...enter scikit-learn

- Well integrated and documented ML library for Python

The screenshot shows the scikit-learn website interface. At the top, there is a navigation bar with the scikit-learn logo and links for 'Install', 'User Guide', 'API', 'Examples', and 'More'. Below this is a header section with the text 'scikit-learn Machine Learning in Python' and a list of features: 'Simple and efficient tools for predictive data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. The main content area is divided into six panels, each representing a different machine learning topic: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing. Each panel includes a brief description, applications, algorithms, and a representative figure or plot. The 'Classification' panel shows a grid of scatter plots. The 'Regression' panel shows a line plot of target vs data. The 'Clustering' panel shows a scatter plot with colored regions and centroids. The 'Dimensionality reduction' panel shows a 3D scatter plot of Iris dataset points. The 'Model selection' panel shows a line plot of cross-validation error. The 'Preprocessing' panel shows a grid of scatter plots with different preprocessing steps.

scikit-learn
Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

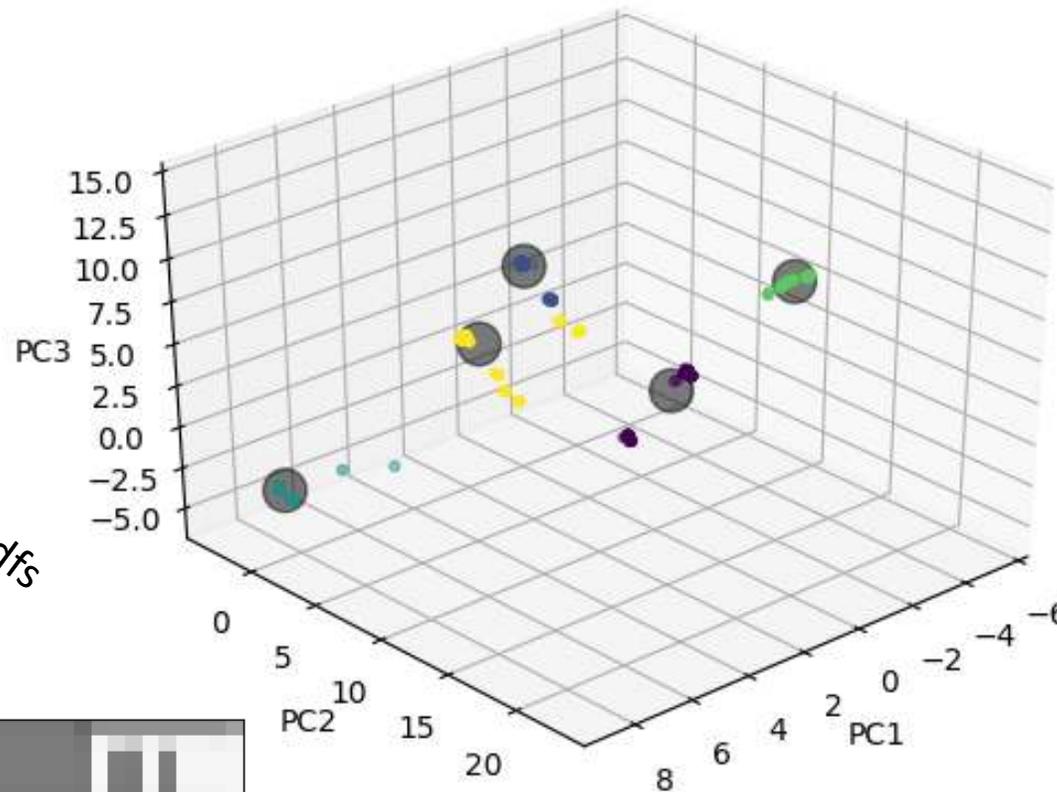
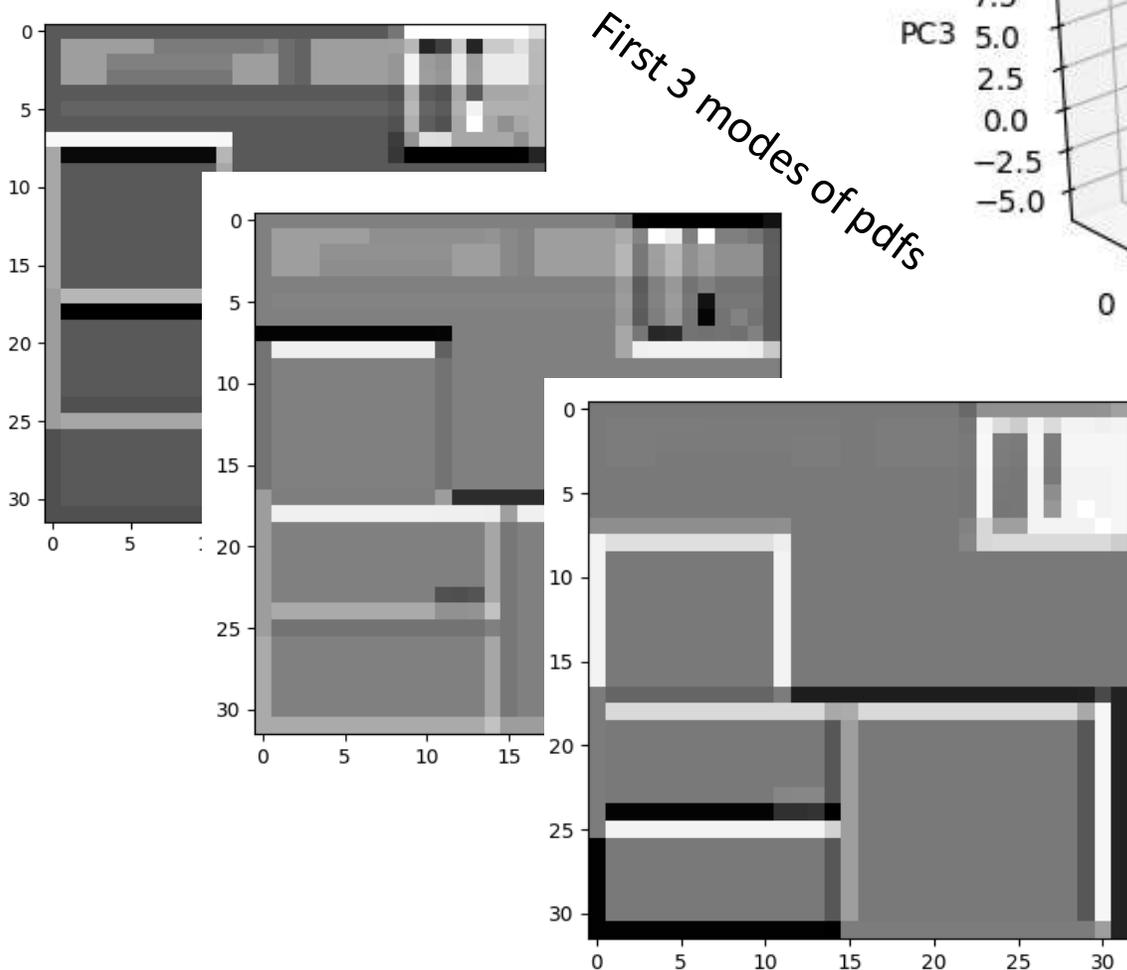
Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

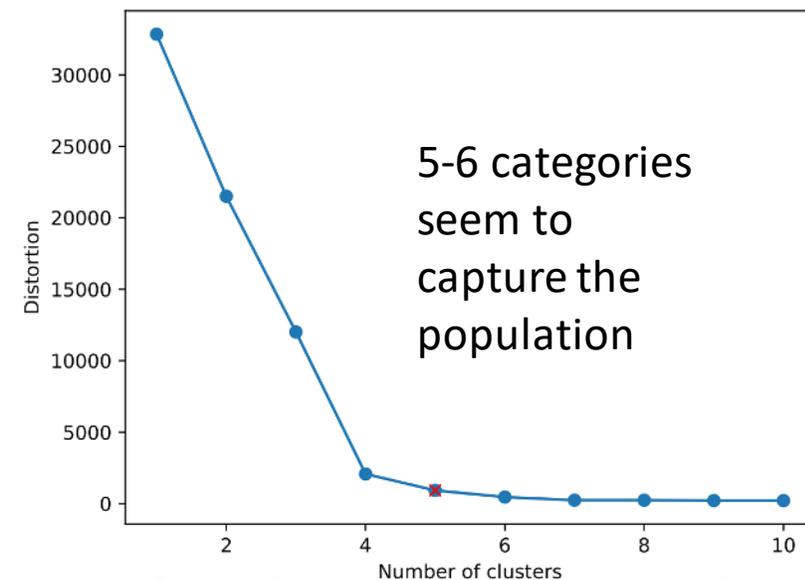
Model selection
Comparing, validating and choosing parameters and models.
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

Preprocessing
Feature extraction and normalization.
Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...

PCA + cluster



Distortion is **defined** as the sum of the squared distances between each observation vector and its cluster centroid...



Considerations

- Don't reinvent the wheel
 - Google and Stackoverflow
- Python2 vs Python3
 - Python 2 is end-of-life – go for 3
- Speed
 - Is python fast enough? It can be
 - Vectorization in numpy
 - JIT compilers (Numba etc)
- Access
 - Some modulus rely on underlying libraries that are not always compiled for Windows
 - Compile them your-self (this can be time consuming...)
 - Run a Linux virtual machine
 - WSL2 / Docker

Final thoughts

- Scripting can enhance simulations on several levels:
 1. It can automatize time-consuming manual workflows
 2. It can reduce manual errors (ie copy-paste errors)
 3. It can yield insights into much larger datasets, than you could process manually
- Pragmatism is crucial when executing simulations under real world deadlines
 1. Choose tools and workflows that you are comfortable with
 2. ...but if you squeeze in just a few lines using a new module / function / etc every time, you gradually build up your arsenal

Discussion

- What tools are you using to speed-up your workflow?
 - Languages
 - Editors
 - Modules
- Are you facing any impediments?
 - Time constraints
 - Learning curves
 - IT security
 - License considerations