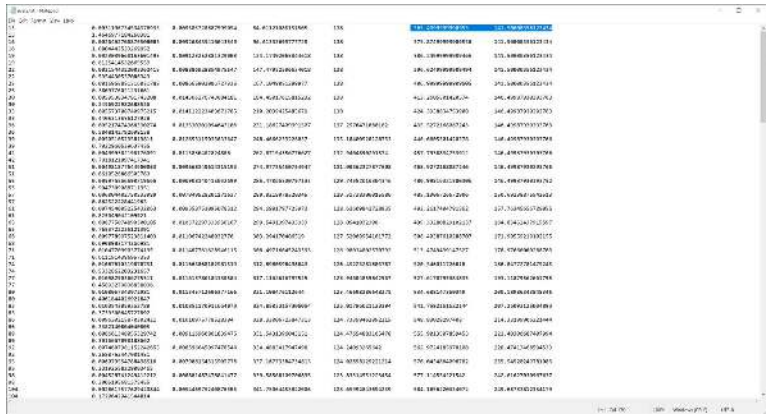
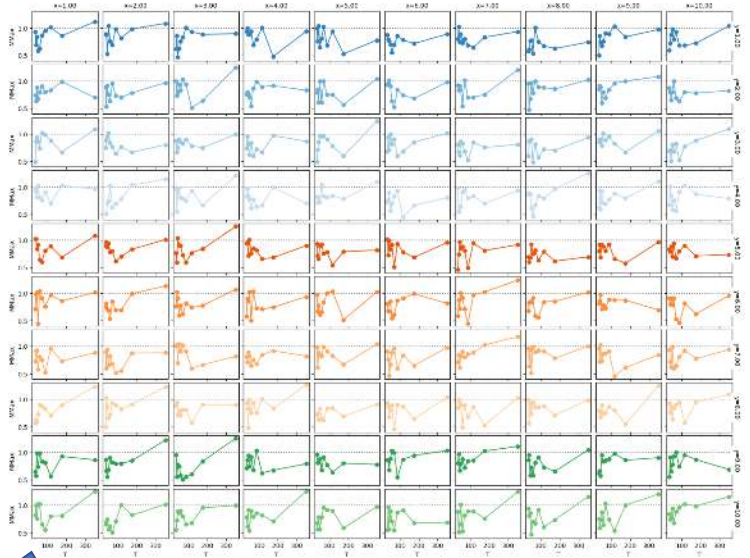
Day-to-day scripting in Simulation

Tore Findsen

Resolvent P/S





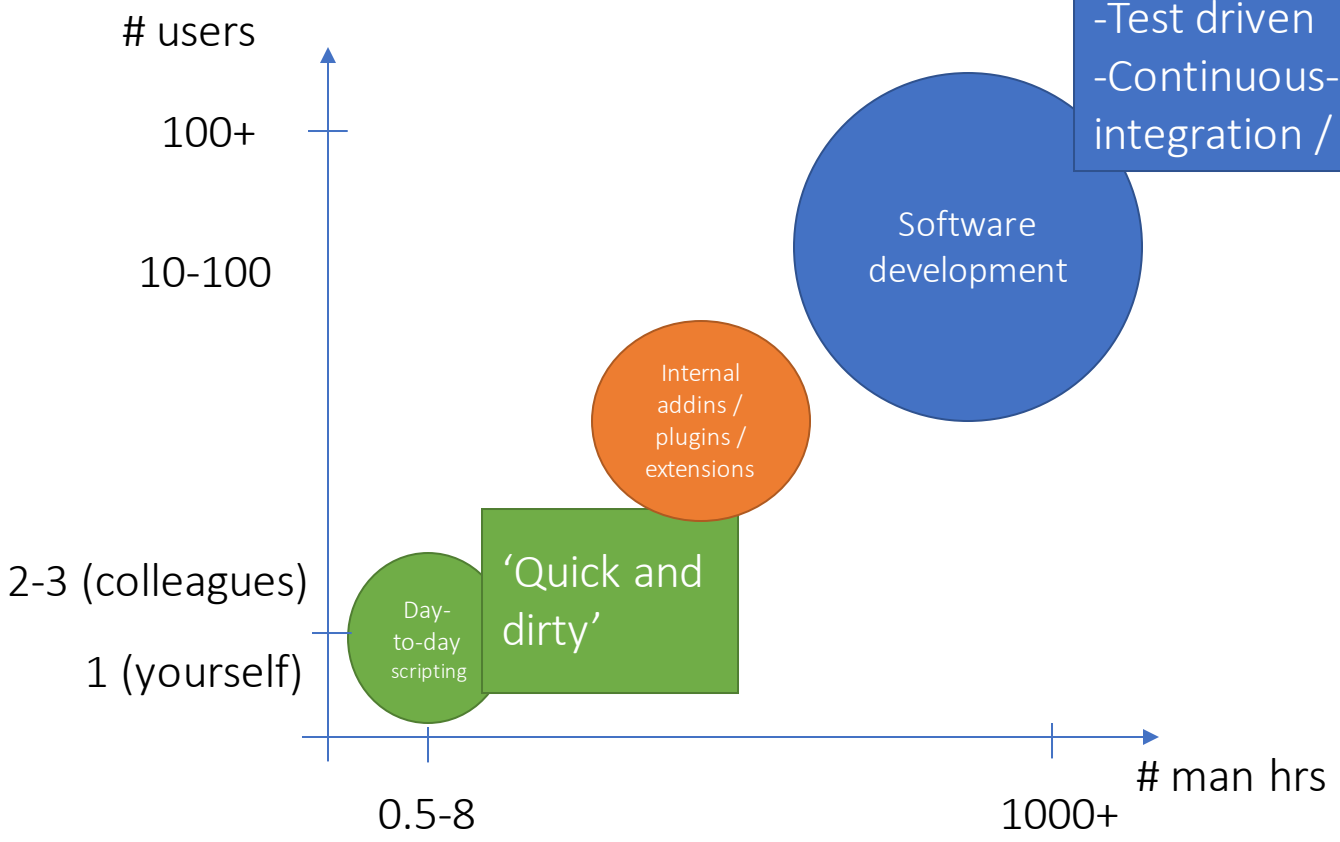
resolvent
Modelling Solutions

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 pfds
  - Case 3: Large scale ML based scraping
- Final thoughts
- Discussion

Definition

What do I mean by ‘day-to-day’ scripting?



Quality and collaborational workflows:
-Version control
-Issue management
-Test driven
-Continuous-integration / delivery

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

resolvent Modelling Solutions

When should scripting be used?

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

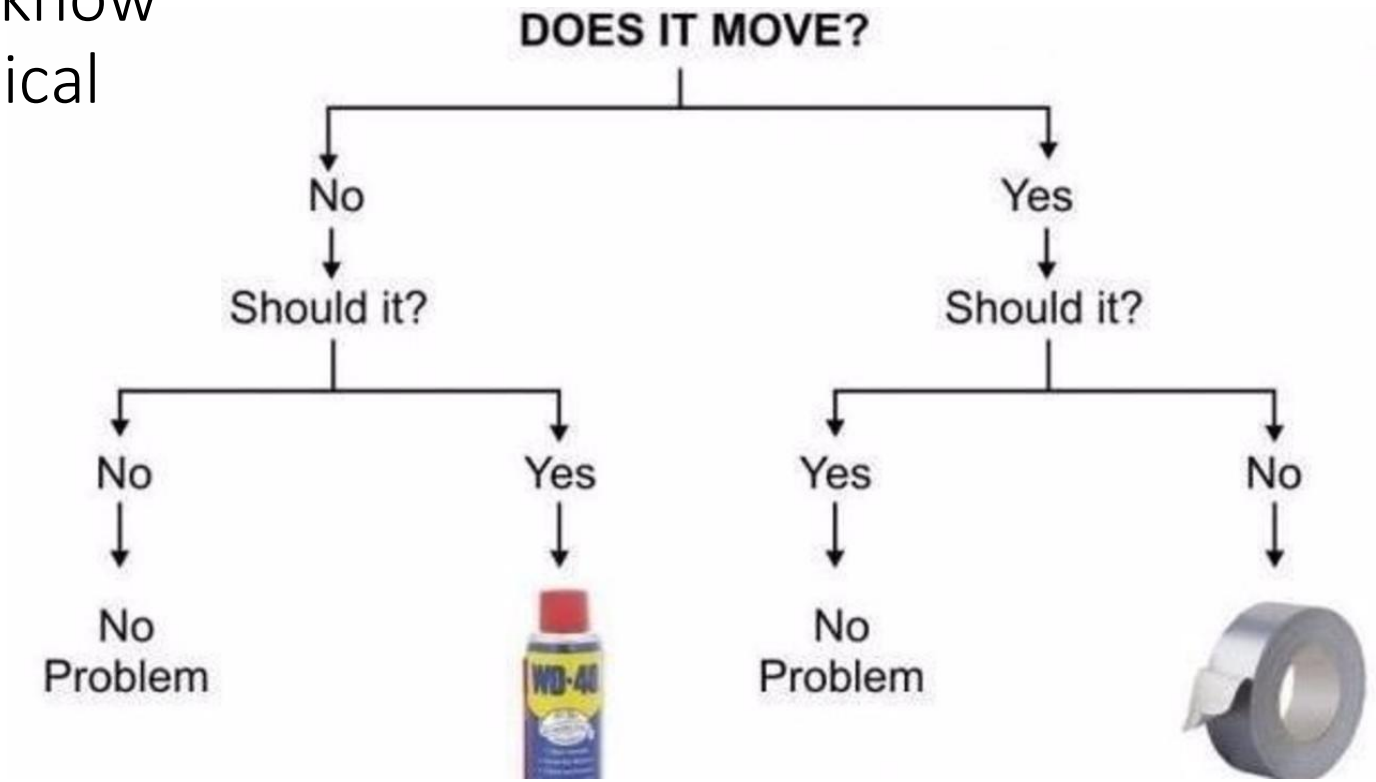
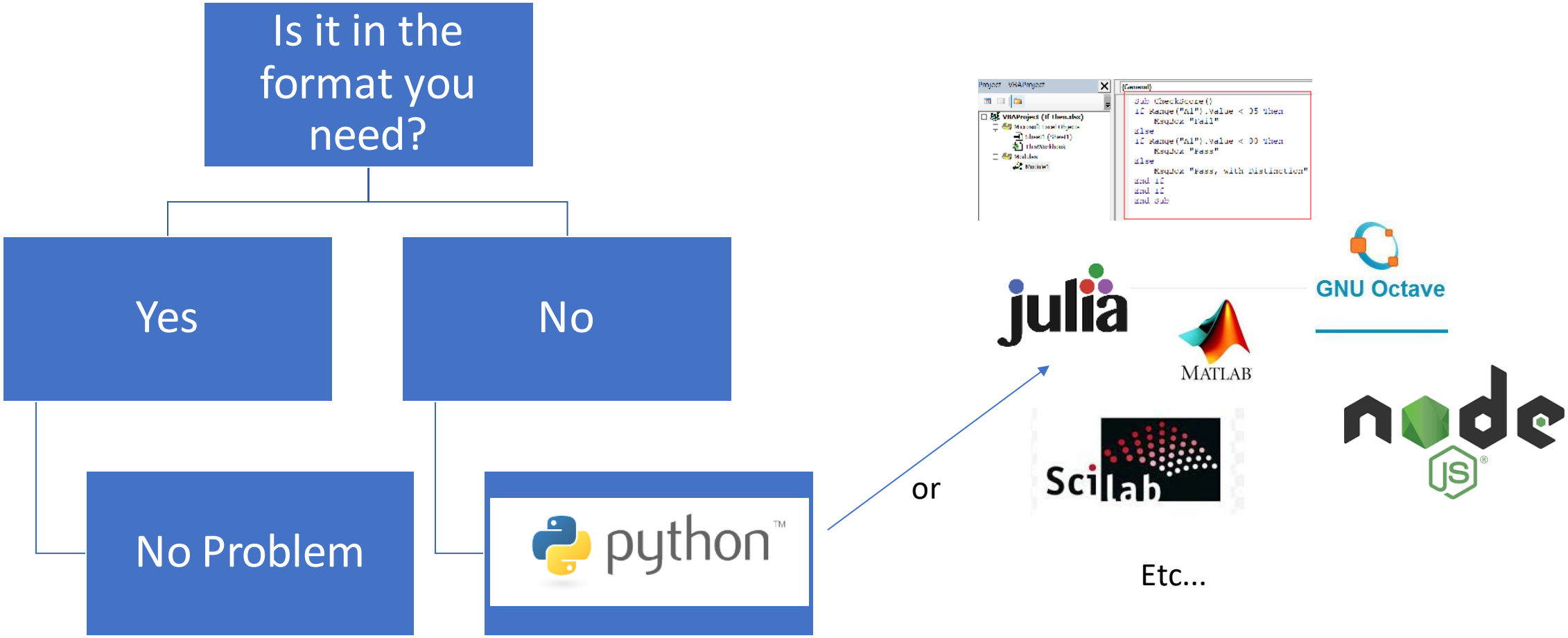


Image courtesy of https://www.flickr.com/photos/dullhunk/7214525854

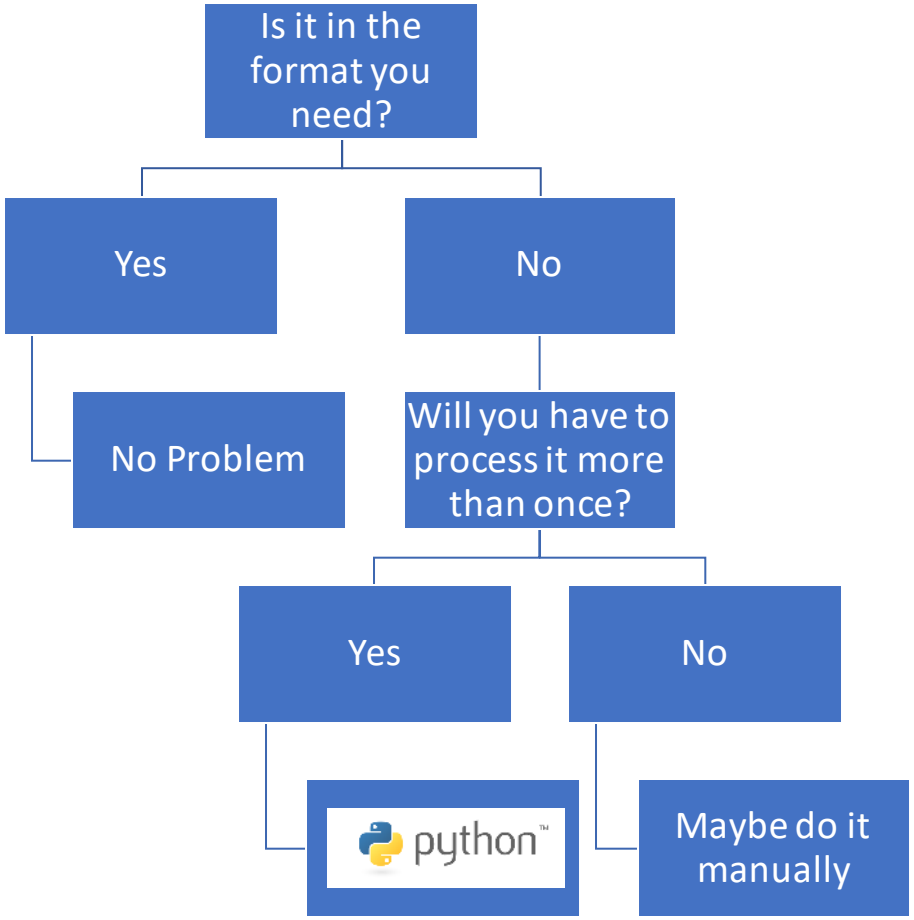
resolvent Modelling Solutions

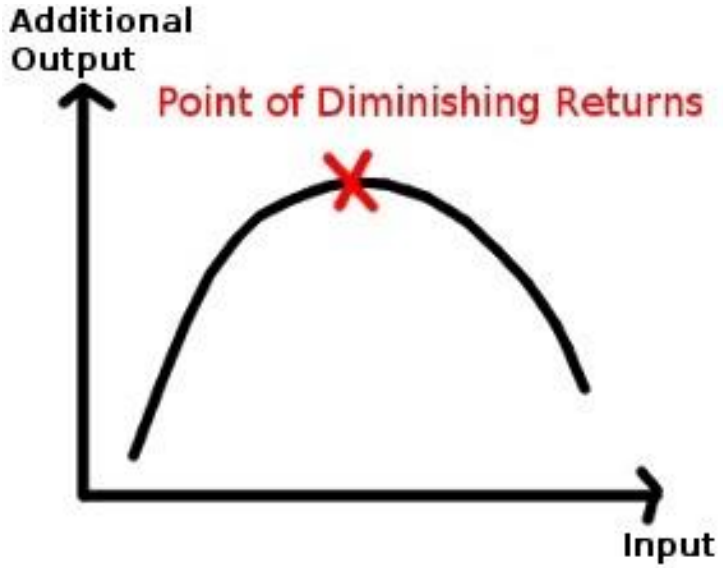
…I think a similar flowchart could be made for data



resolvent
Modelling Solutions

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





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

resolvent Modelling Solutions

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/excecuting 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

Python 3

Java 8

resolvent Modelling Solutions

Workflow

resolvent
Modelling Solutions

A minimal toolbox

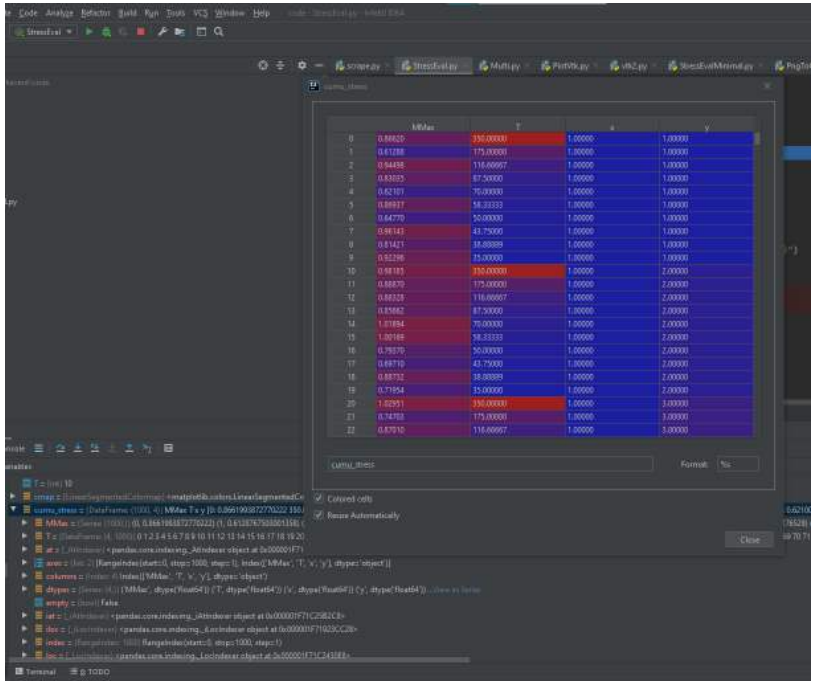
- Python distribution
  - Anaconda is popular
    - Conda allows for nice (and graphical package management)
- Modules
  - Numpy
  - Pandas
  - Matplotlib
- Editor
  - This is likely contentious!
  - I highly recommend Jetbrains 2 free options (Without any affiliation☺)
    - Pycharm community edition
    - 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 !!

resolvent Modelling Solutions

A few words on the editor

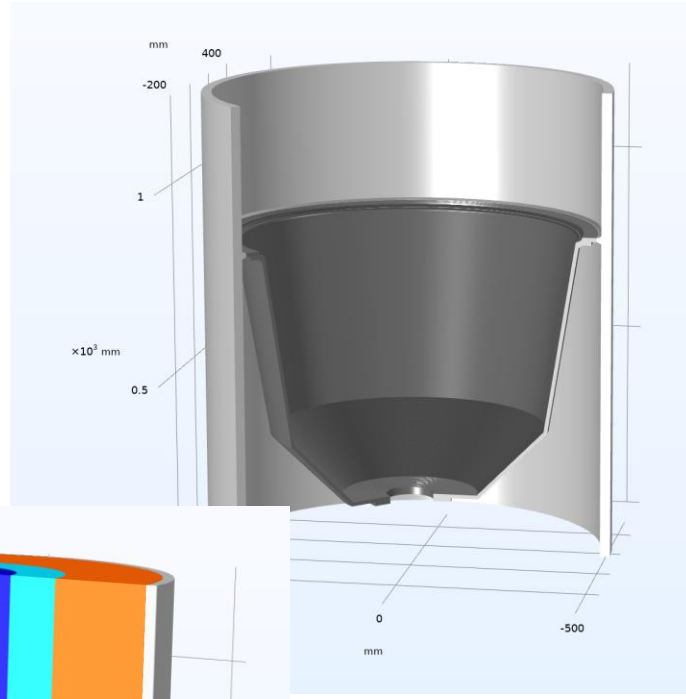
- For the editor code completions, and an interactive debugger is a must! Pandas dataframe inspector is a very nice feature!!
- When you script inside the simulation tool, you usually have to use their environment
  - Although you can sometimes setup your workflow for debugging from outside – a topic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

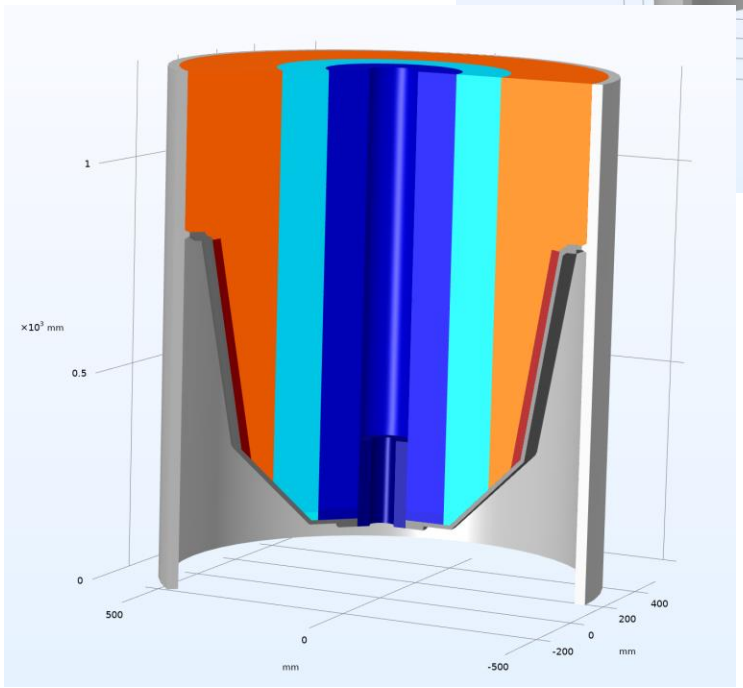


resolvent
Modelling Solutions

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

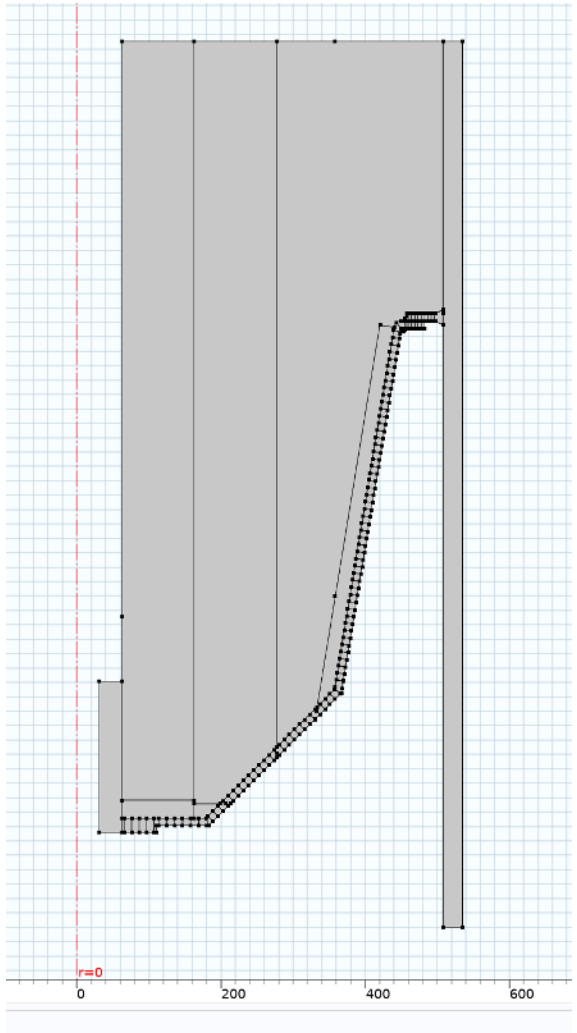


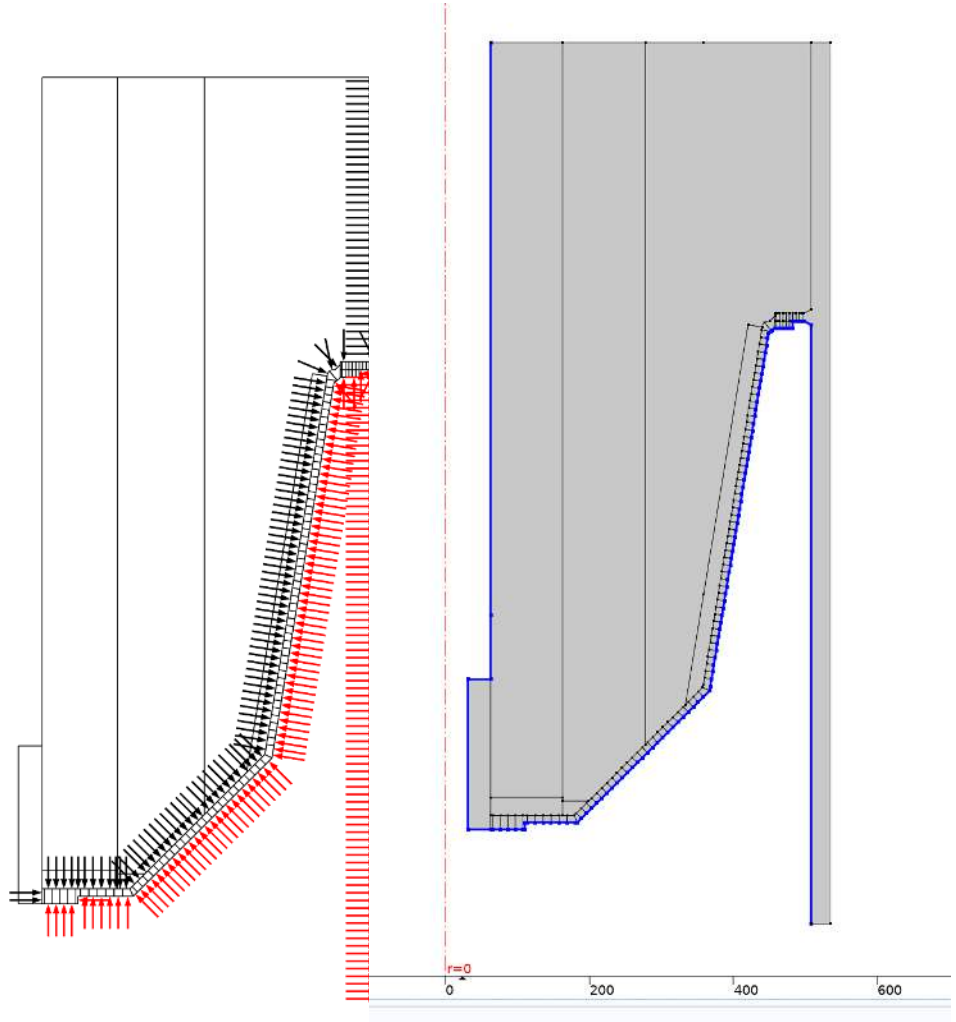


resolvent
Modelling Solutions

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

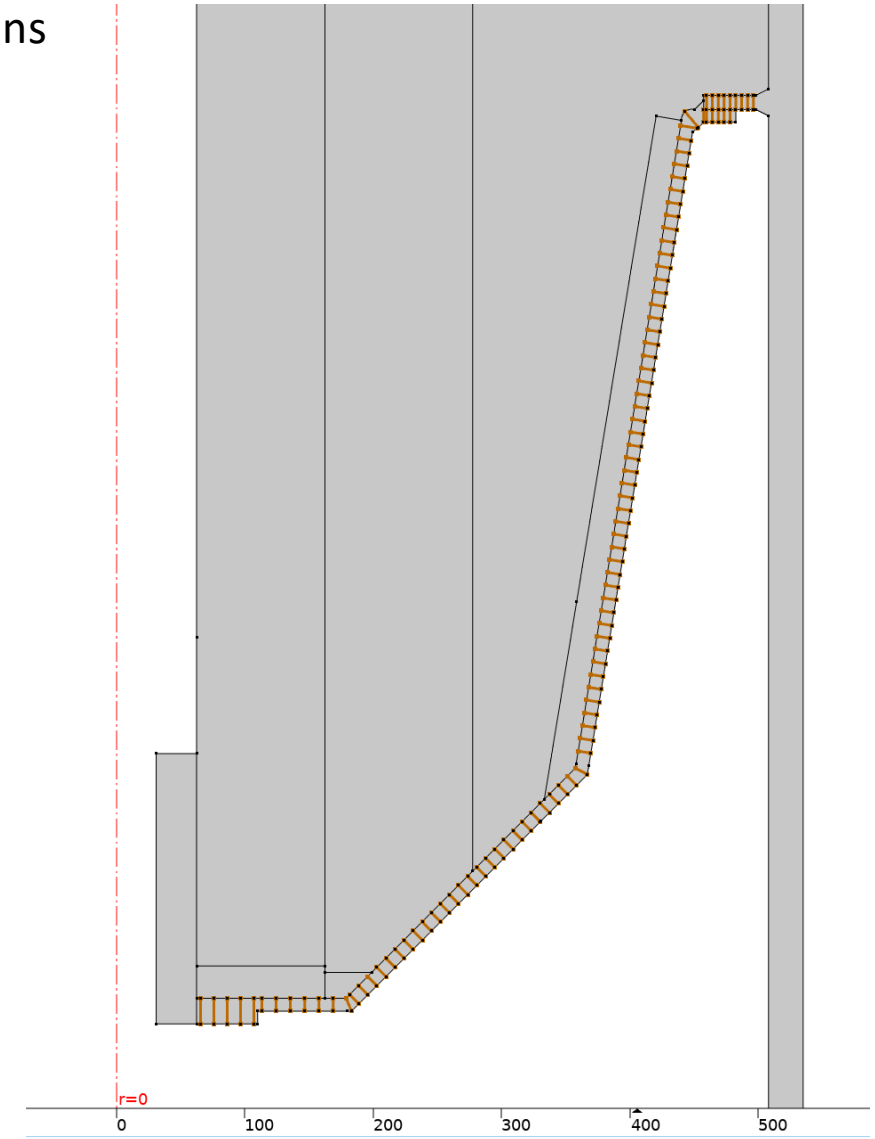




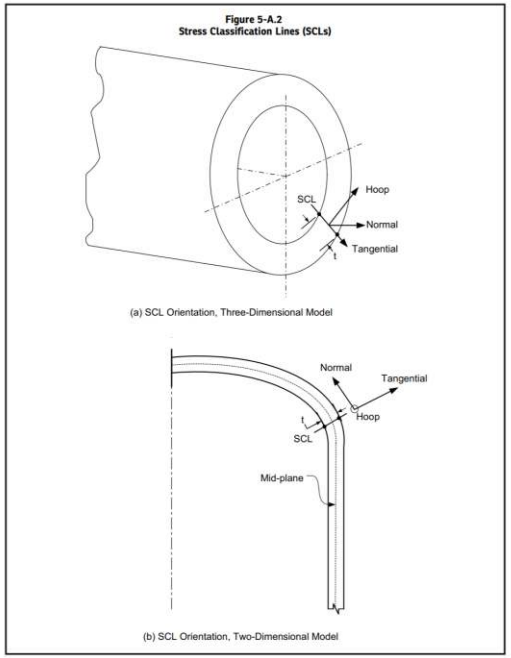
resolvent
Modelling Solutions

Stress evaluation

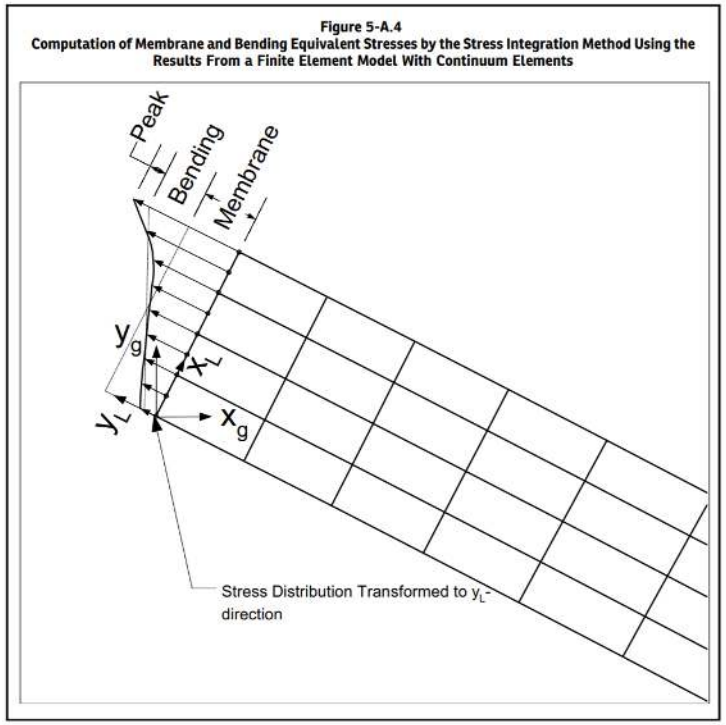
During design iterations the max can move around – thus we must investigate ~100 locations



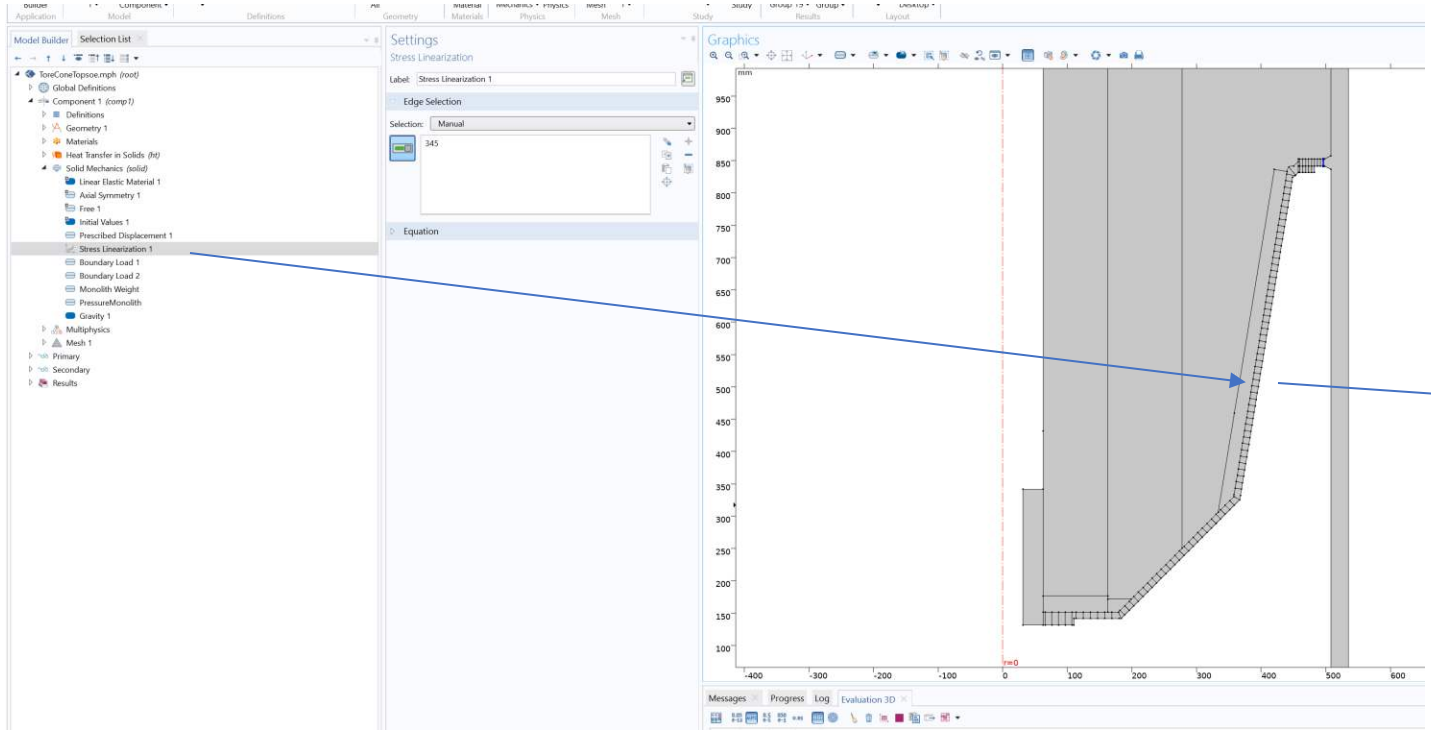
Introduce Stress classification line through critical locations



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

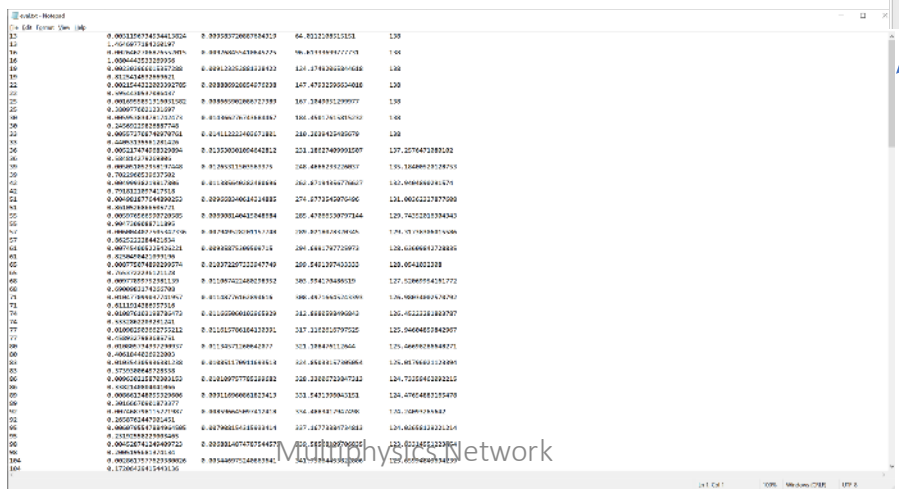


resolvent
Modelling Solutions



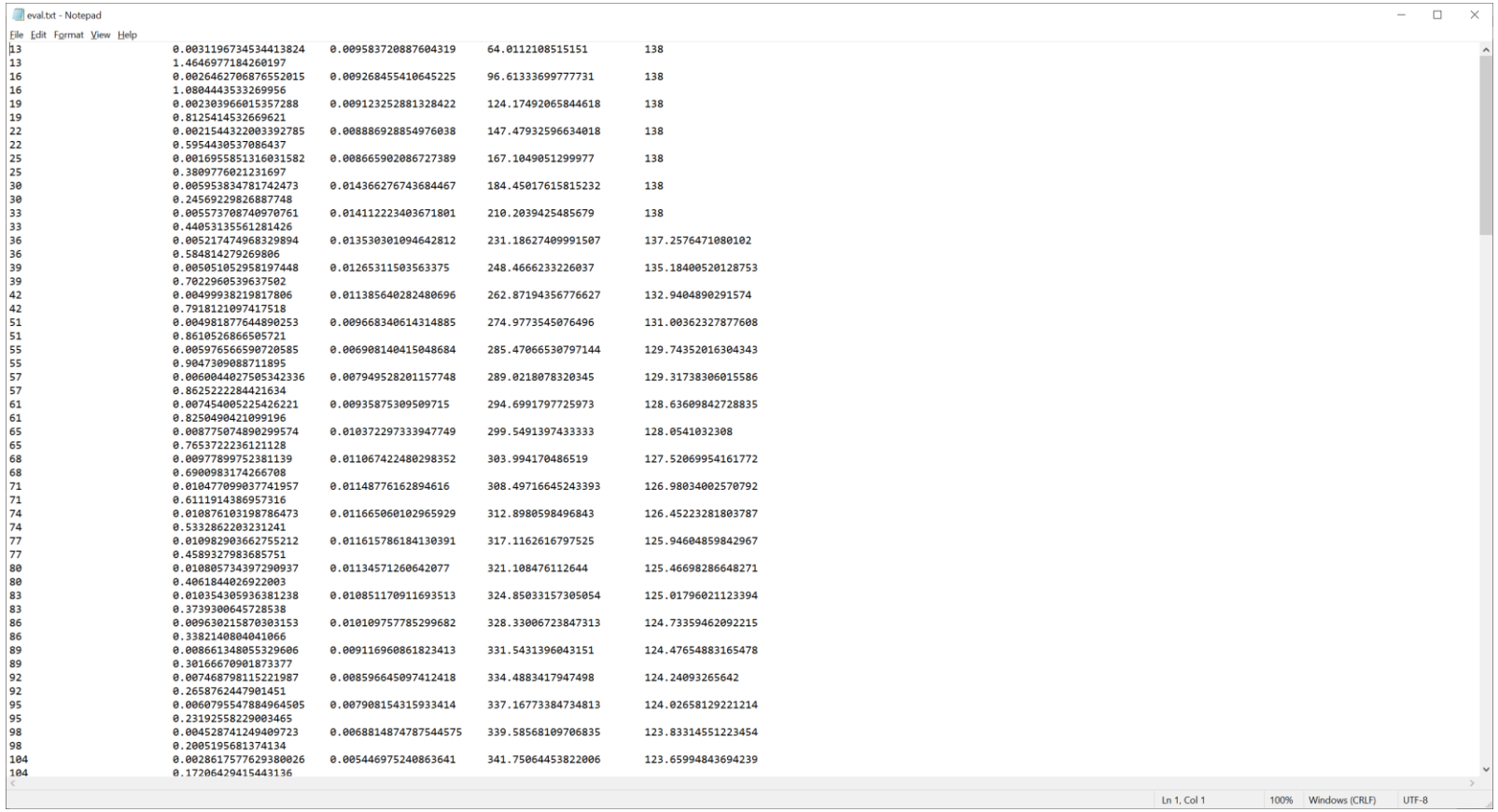
We create a small export script in comsol that traverses the ~100 SCL's and outputs the necessary quantities to a textfile

```
package builder;

import com.comsol.api.*;
import com.comsol.model.*;
import com.comsol.model.physics.*;
import com.comsol.model.application.*;

public class method1 extends ApplicationMethod {

  public void execute() {
    int[] pathNos = model.component("comp1").selection("sel5").entities(1);

    for (int pathNo : pathNos) {
      model.param().set("pathnom", pathNo);

      model.component("comp1").physics("solid").feature("sl1").selection("EdgeSelection").set();
      model.component("comp1").physics("solid").feature("sl1").selection("EdgeSelection").set(pathNo);

      model.sol("sol1").updateSolution();
      model.sol("sol2").updateSolution();


      model.result().export("data1").run();
      model.result().export("data2").run();
    }
    int[] pathNos2 = model.component("comp1").selection("sel6").entities(1);


    for (int pathNo2 : pathNos2) {
      model.param().set("pathnom2", pathNo2);

      model.component("comp1").physics("solid").feature("sl1").selection("EdgeSelection").set();
      model.component("comp1").physics("solid").feature("sl1").selection("EdgeSelection").set(pathNo2);

      model.sol("sol1").updateSolution();
      model.sol("sol2").updateSolution();


      model.result().export("data3").run();
      model.result().export("data4").run();



    }
  }

}
```



M
MB
MBQ
Path
T
S

resolvent
Modelling Solutions

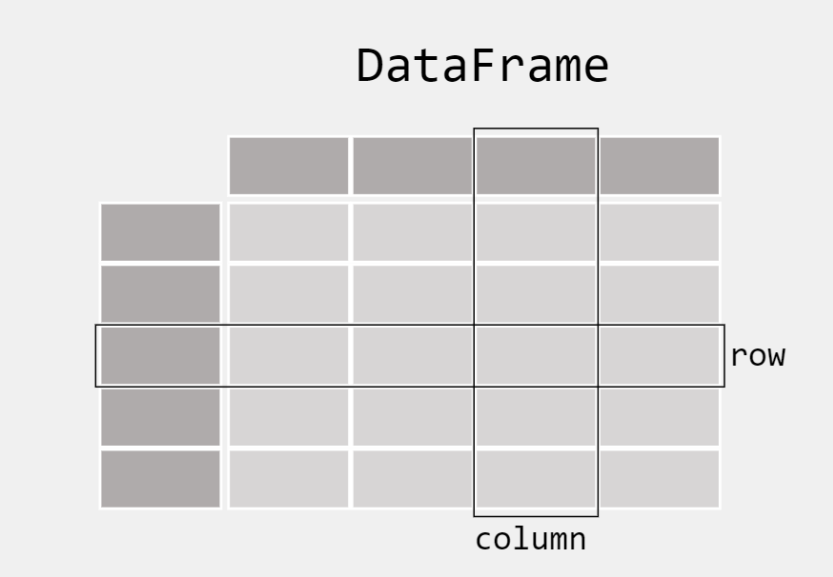


- Now we have a text file
- It's 'jagged' because we're evaluating secondary stresses separately
- It's not very intuitive to look at – it's very hard to make good design decisions from
- Time for some scripting!
- I'm going to show you some code – my main message is how little you need – not the actual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

resolvent Modelling Solutions

…enter pandas

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

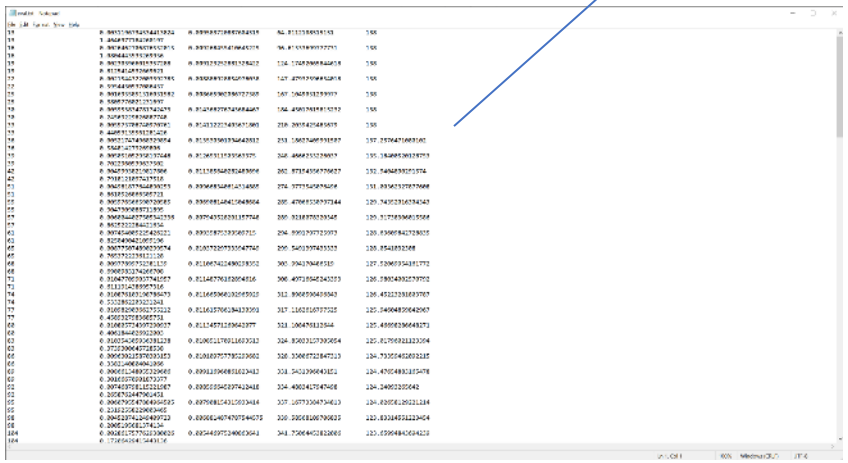


	MMax	T	x	y
0	0.86620	350.00000	1.00000	1.00000
1	0.61288	175.00000	1.00000	1.00000
2	0.94498	116.66667	1.00000	1.00000
3	0.83035	87.50000	1.00000	1.00000
4	0.62101	70.00000	1.00000	1.00000
5	0.86937	58.33333	1.00000	1.00000
6	0.64770	50.00000	1.00000	1.00000
7	0.96143	43.75000	1.00000	1.00000
8	0.81421	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

resolvent Modelling Solutions

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

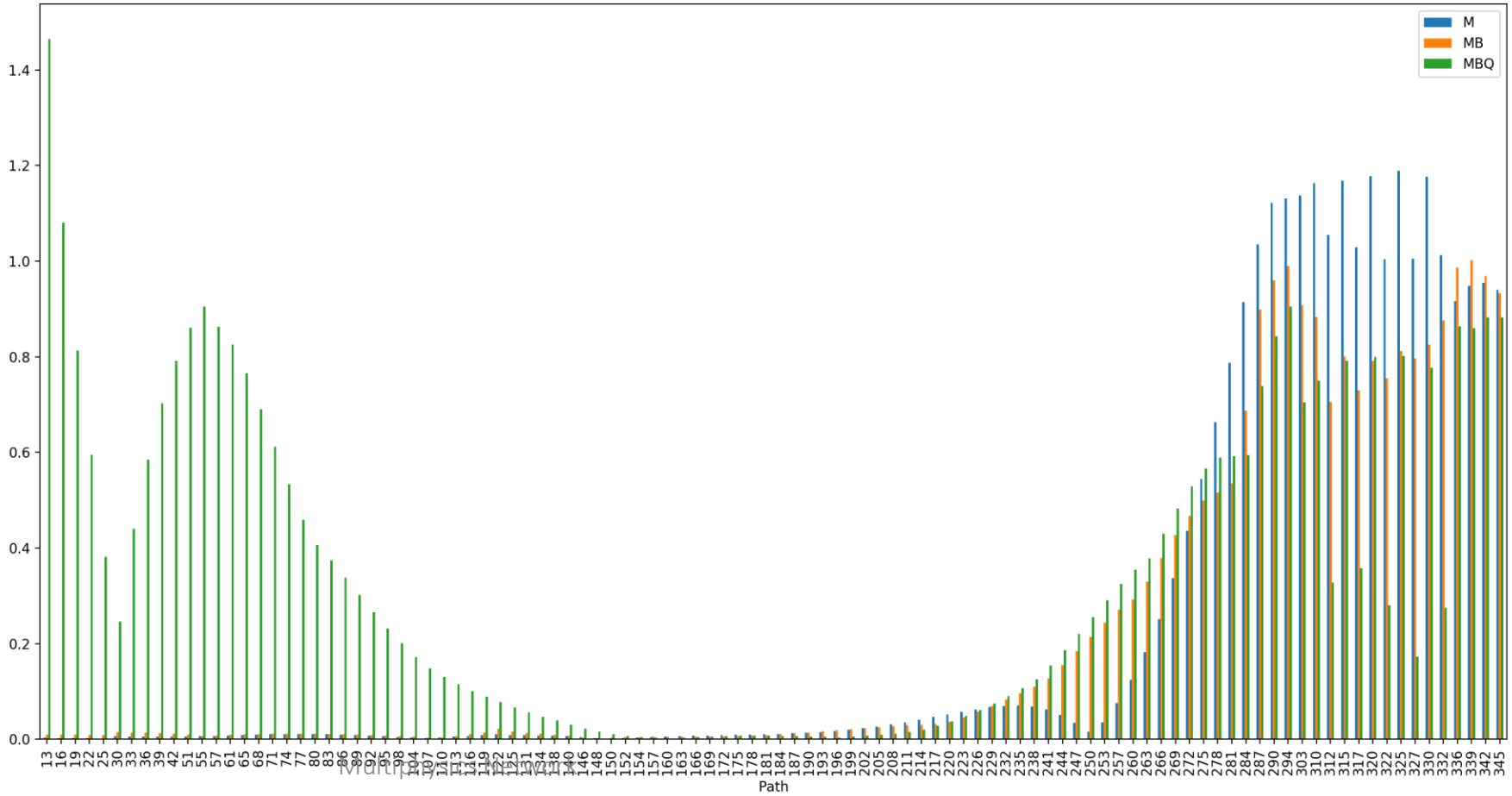
```
import pandas as pd

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'])
```



	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

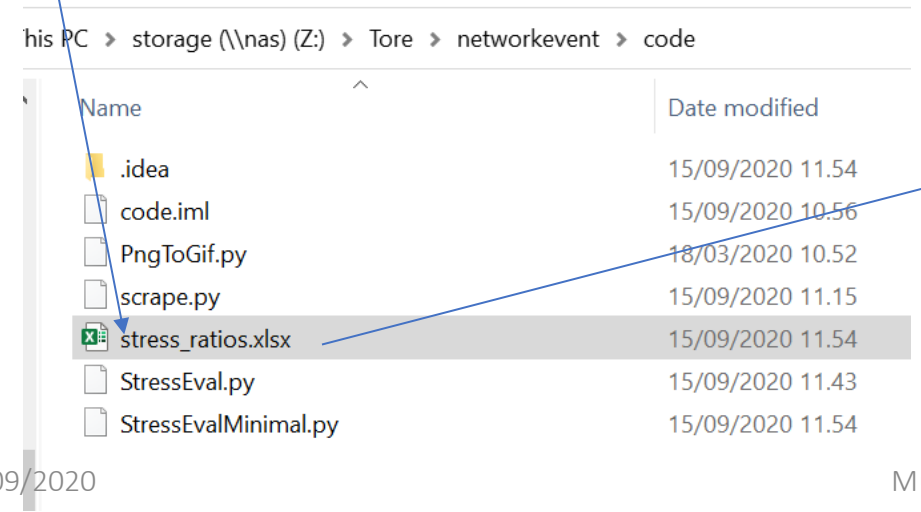
- Plotting it, is 3 extra lines

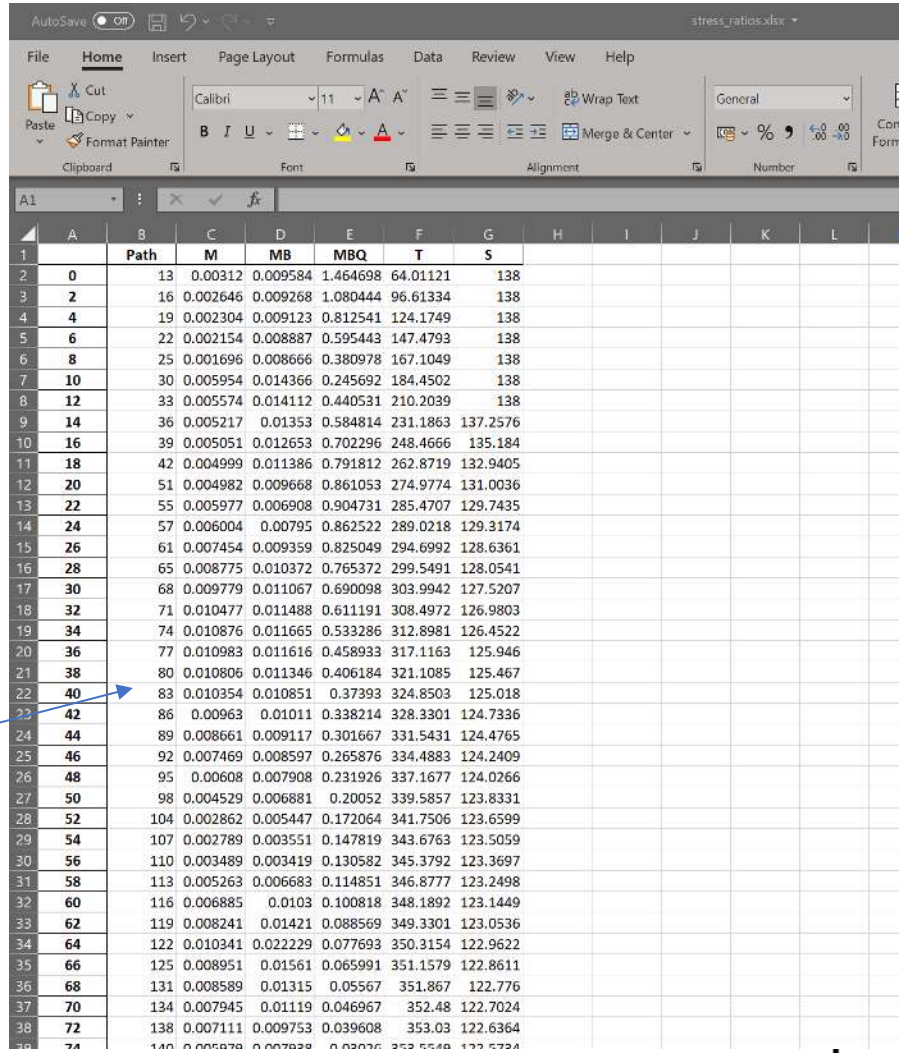
```python
import matplotlib.pyplot as plt
import pandas as pd

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'])

stresses.plot.bar(x='Path',y=['M','MB','MBQ'])

plt.show()
```



1 statement exports the dataframe to Excel

```python
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=48, 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(r"Z:\Tore\networkevent\eval2.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], '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()
```

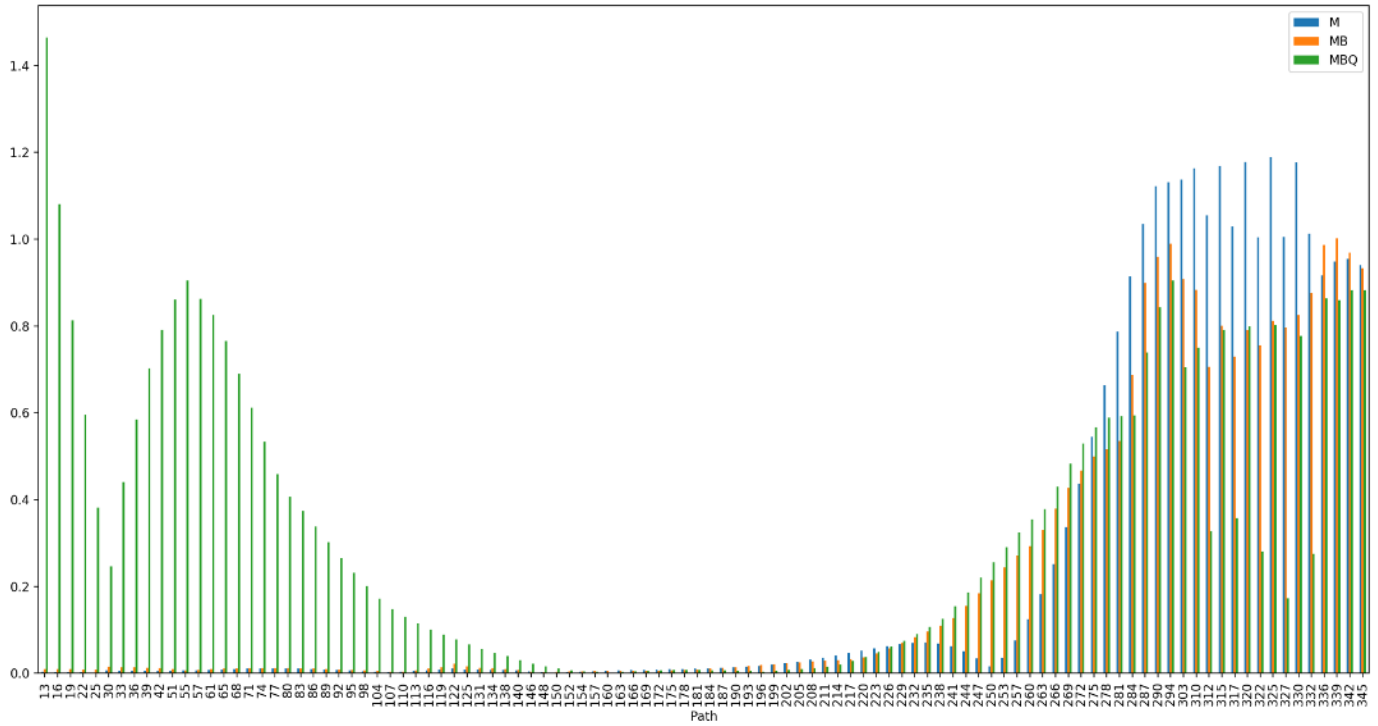


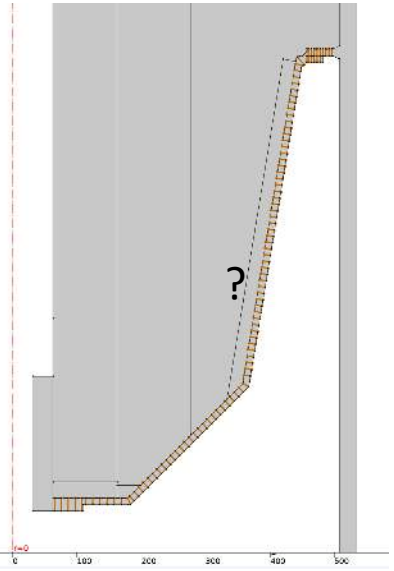


	A	Path	M	MB	MBQ	T	S
1							
2	0	13	0.00312	0.009584	1.464698	64.01121	138
3	2	16	0.002646	0.009268	1.080444	96.61334	138
4	4	19	0.002304	0.009123	0.812541	124.1749	138
5	6	22	0.002154	0.008887	0.595443	147.4793	138
6	8	25	0.001696	0.008666	0.380978	167.1049	138
7	10	30	0.005954	0.014366	0.245692	184.4502	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

resolvent
Modelling Solutions

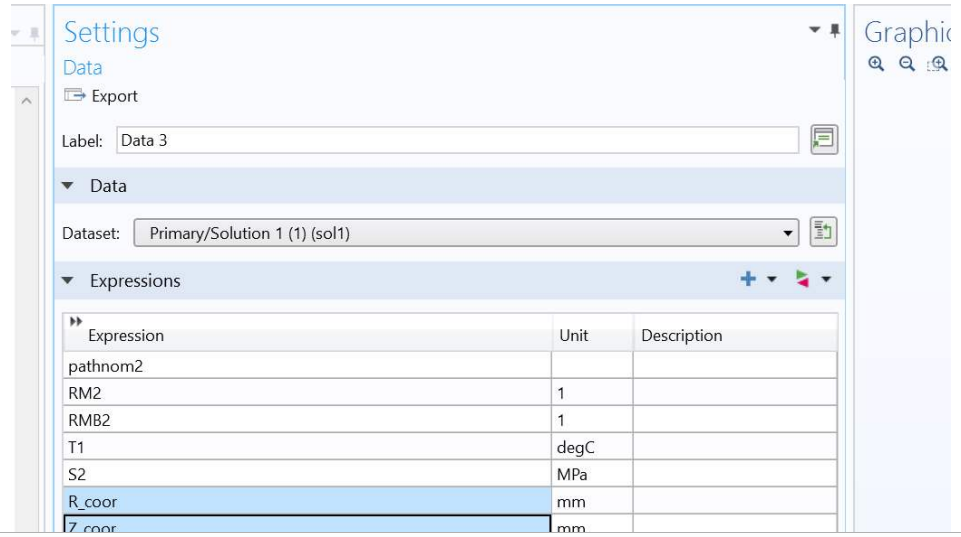
- We can see that ie path #13 is above the allowable
- We then have to go back into Comsol to figure out where it is and how to mitigate it
- This is not efficient...

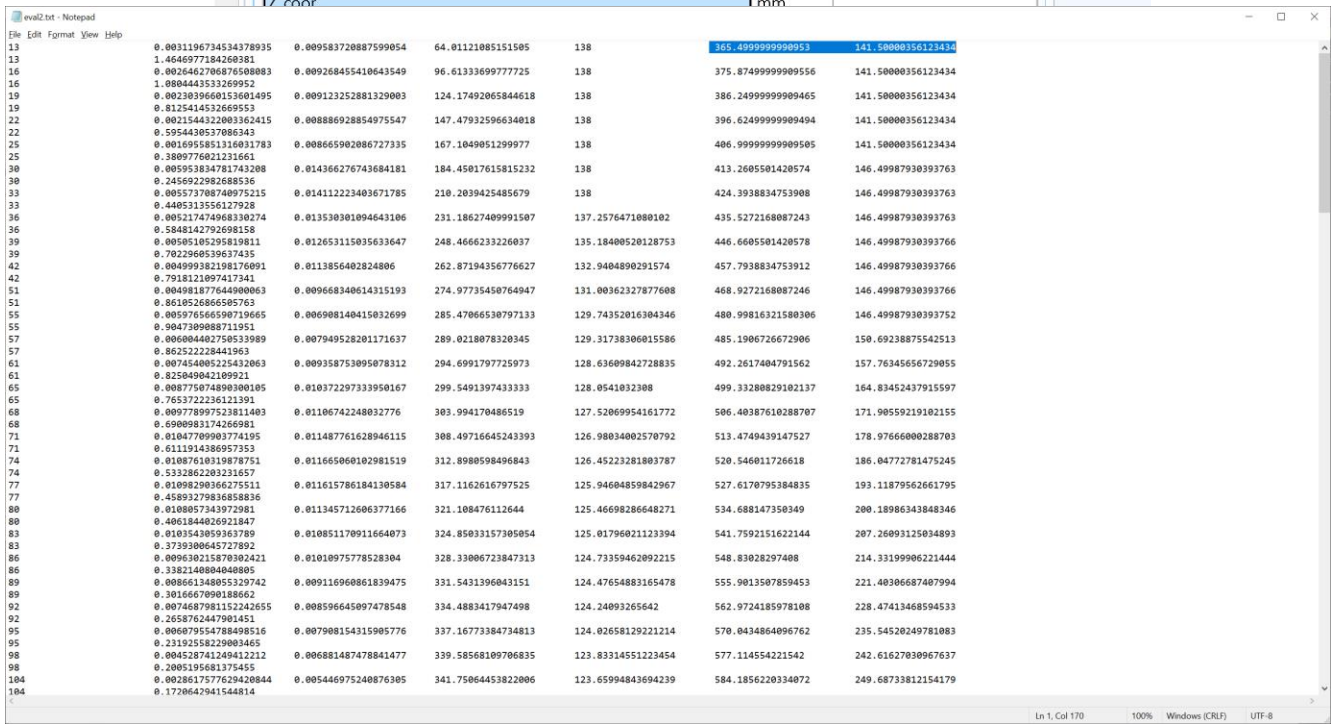




resolvent
Modelling Solutions

- ...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...*

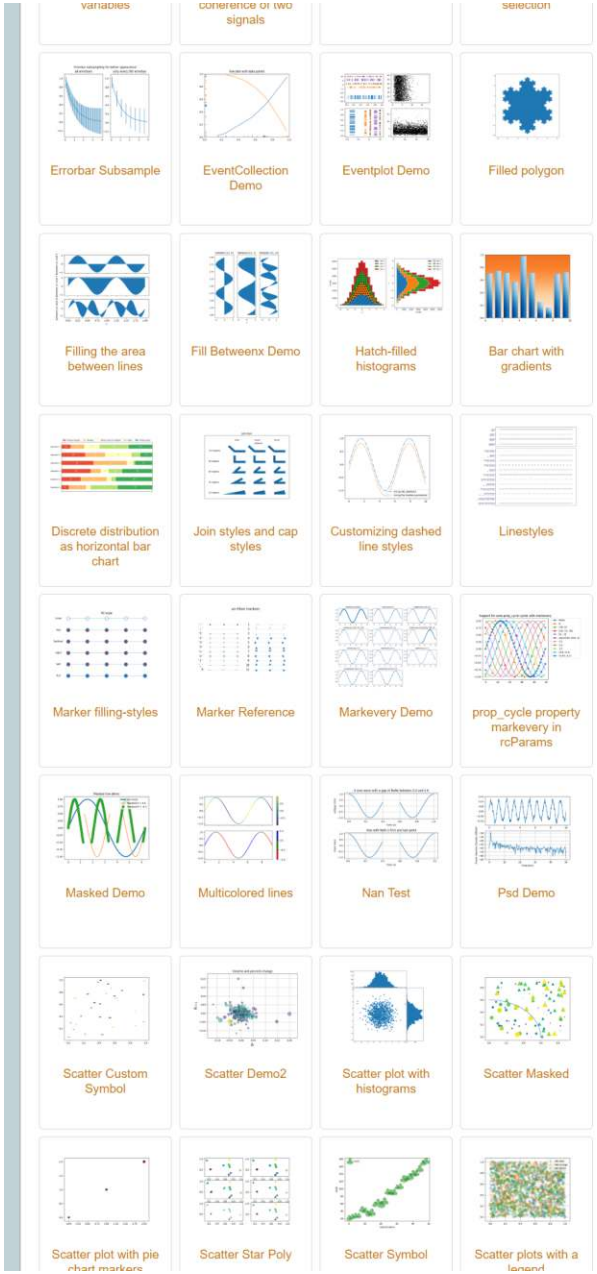




resolvent Modelling Solutions

…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!

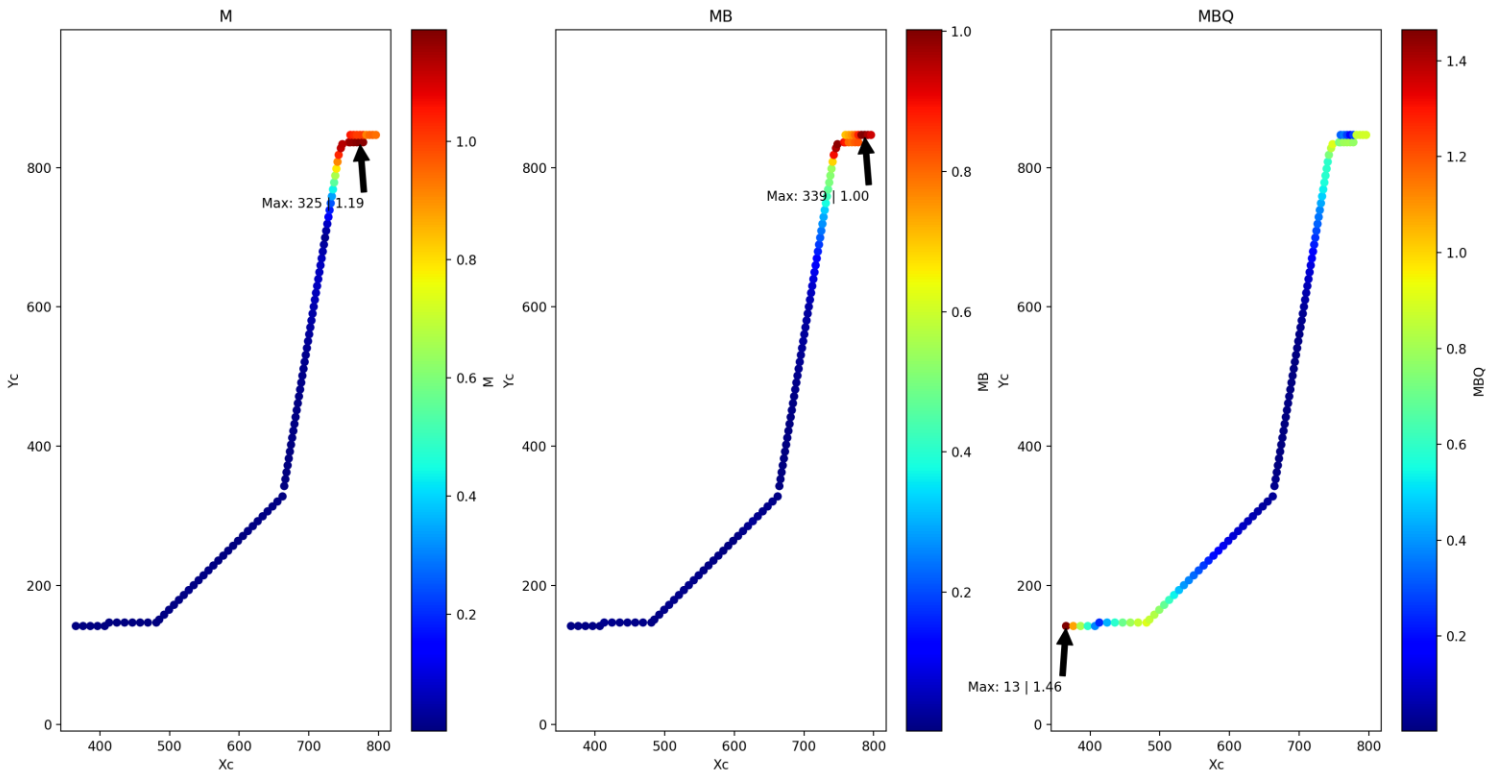


resolvent Modelling Solutions

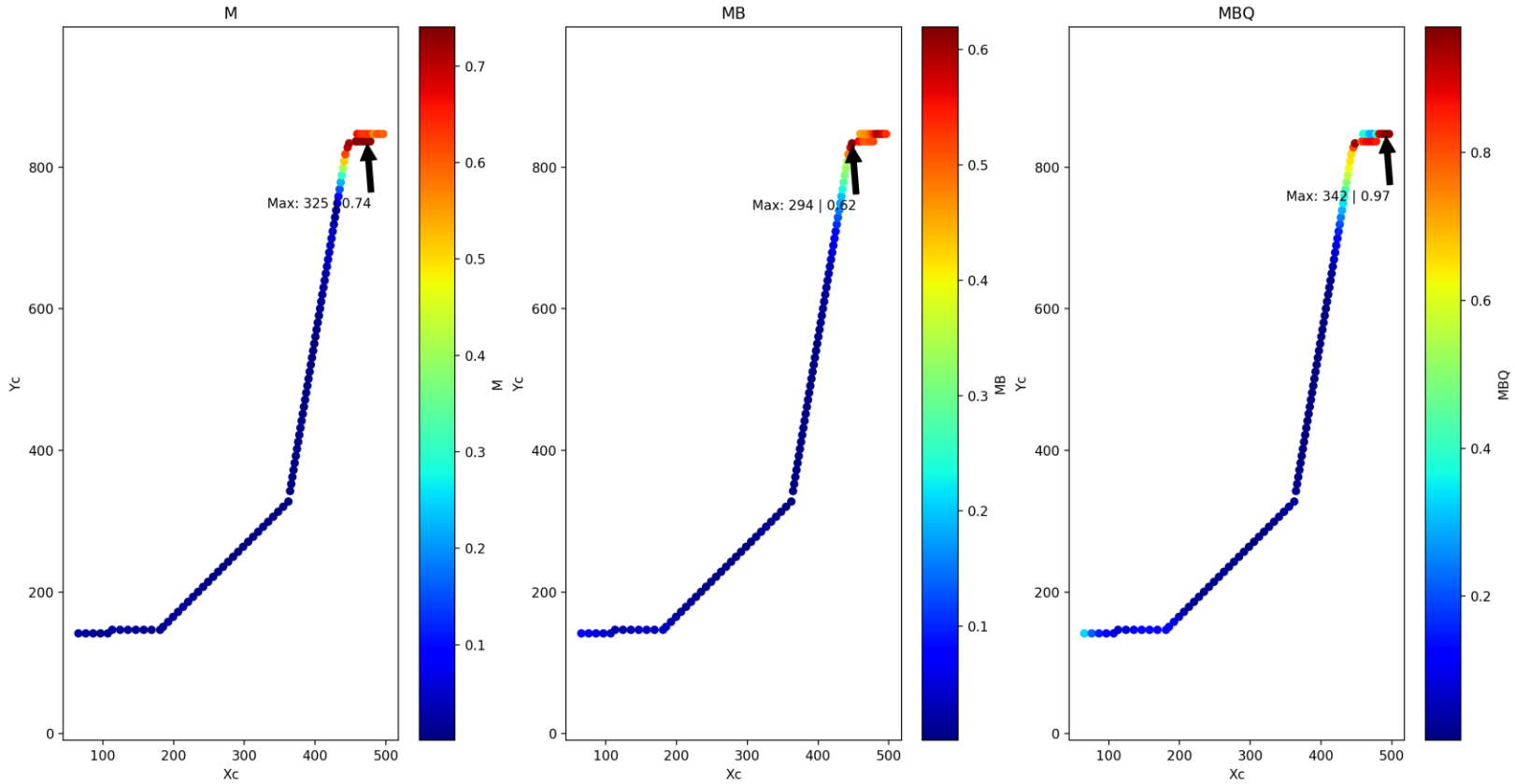
```python
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib import cm

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(r"Z:\Tore\networkevent\eval2.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':

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)

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

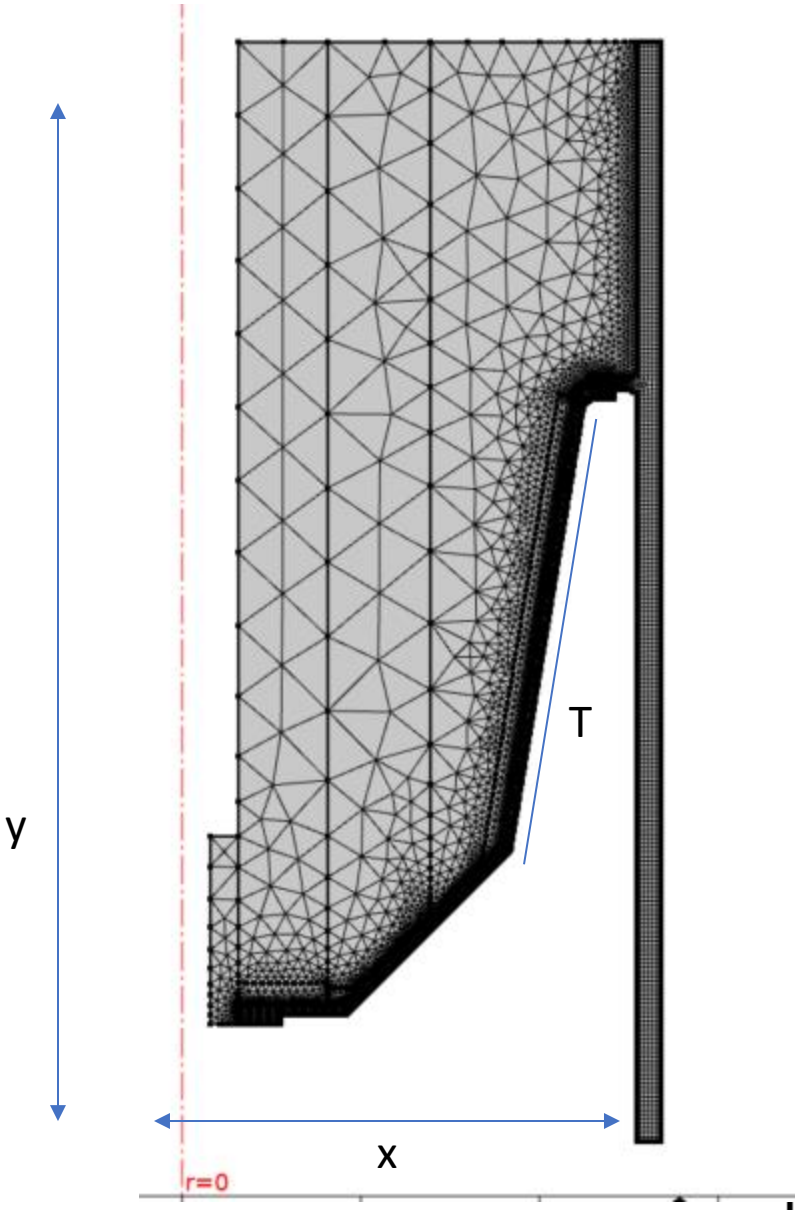


- For 1 plot this might be somewhat tedious – but when you modify your design it starts to pay off
- Reduce radius by 300 mm
- Rerun model and script
- – ok for 1hrs worth of coding...
- For this specific project only 5-10 design iterations where necessary, so in terms of day-to-day scripting we stopped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



resolvent
Modelling Solutions

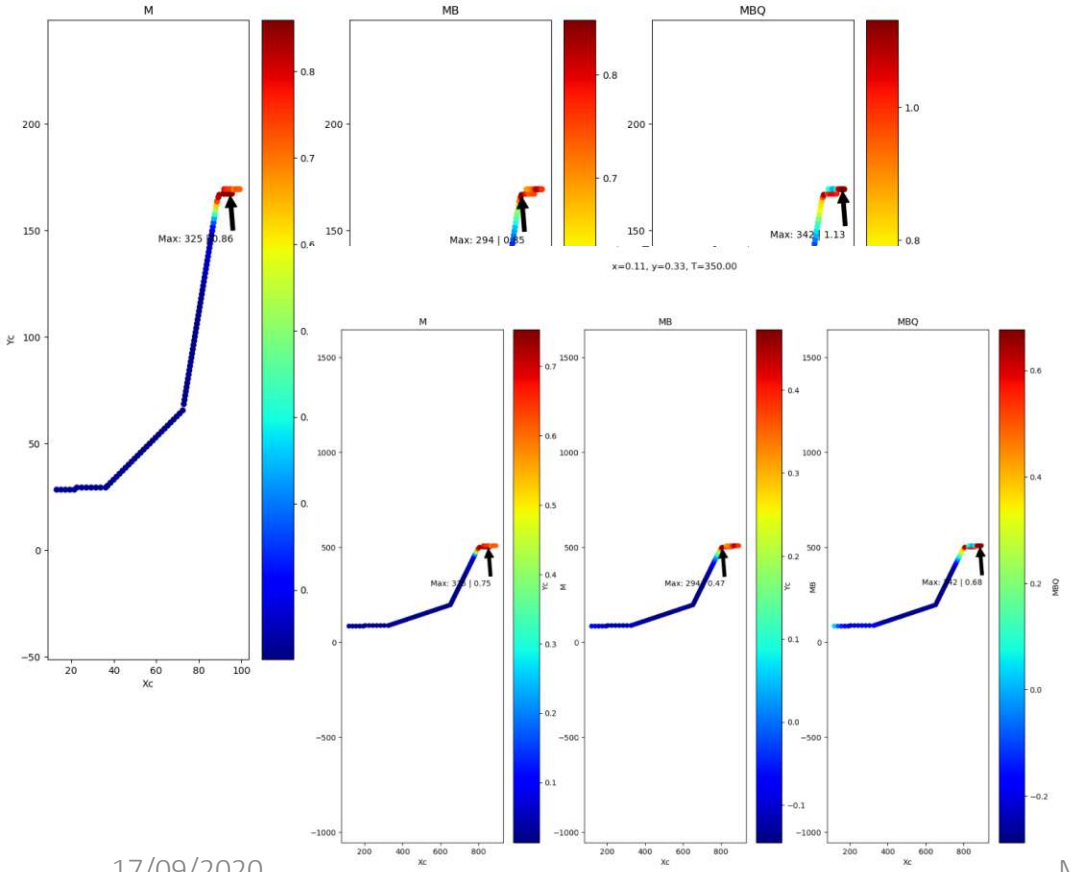
- We've created a model that solves in ~10s
- Let's utilize this to investigate the solution-space
- Parameterize
  - Cone-Height
  - Diameter
  - Convective coeff at underside
    - Corresponds to adding thermal protection
- 10 x 10 x 10 = 1000 designs to evaluate
- This can be accomplished through a nested parametric sweep in Comsol, or some 'execution scripting'
  - Ie. 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 pertube the original data to dreate 1000 'synhetic' datasets



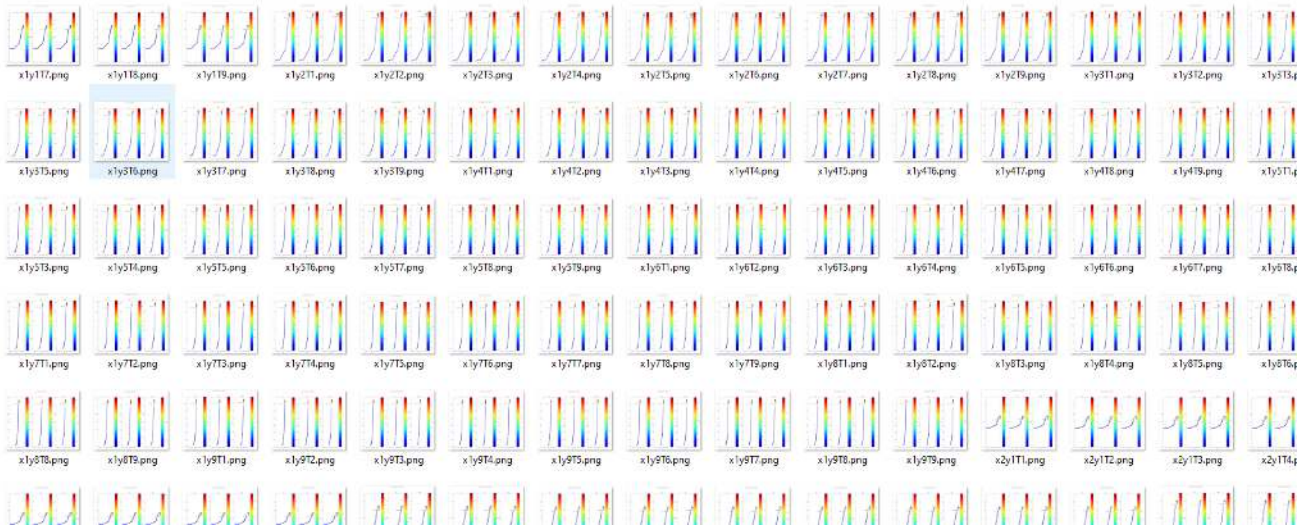
resolvent
Modelling Solutions

```
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

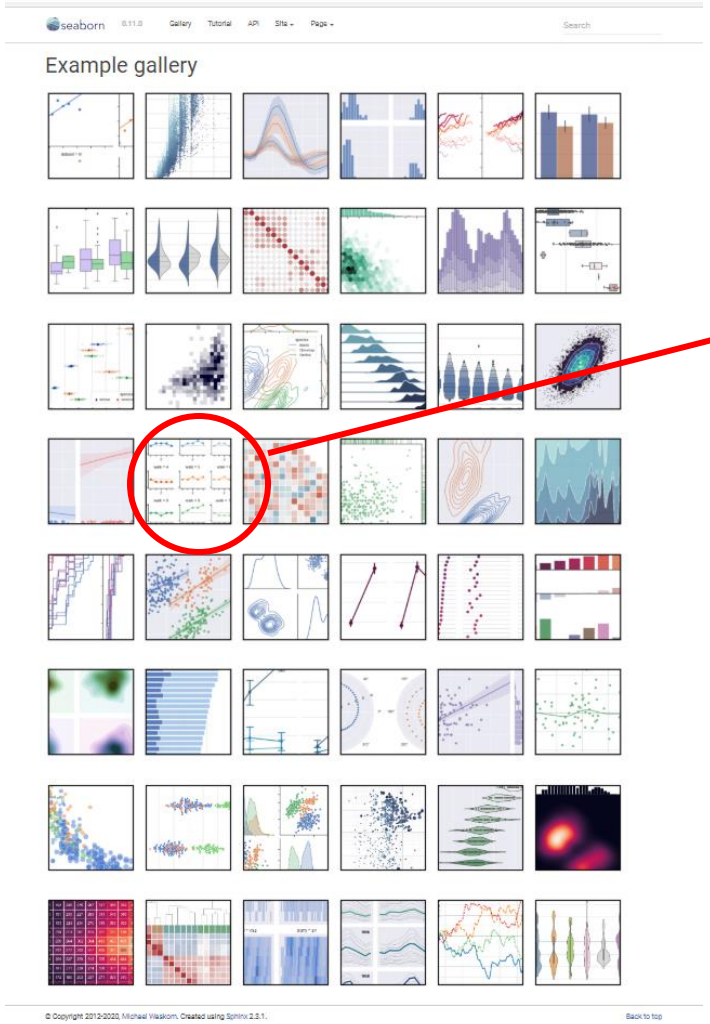


We get 1000 plots – not ideal for making design decisions...

resolvent Modelling Solutions

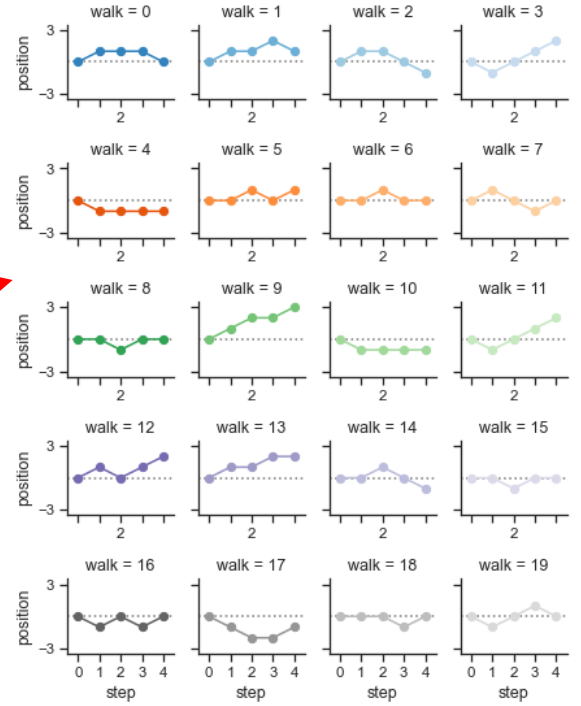
...enter Seaborn

Seaborn is a higher-level plotting library based on Matplotlib



seaborn 0.11.0 Gallery Tutorial API Site Page

Plotting on a large number of facets

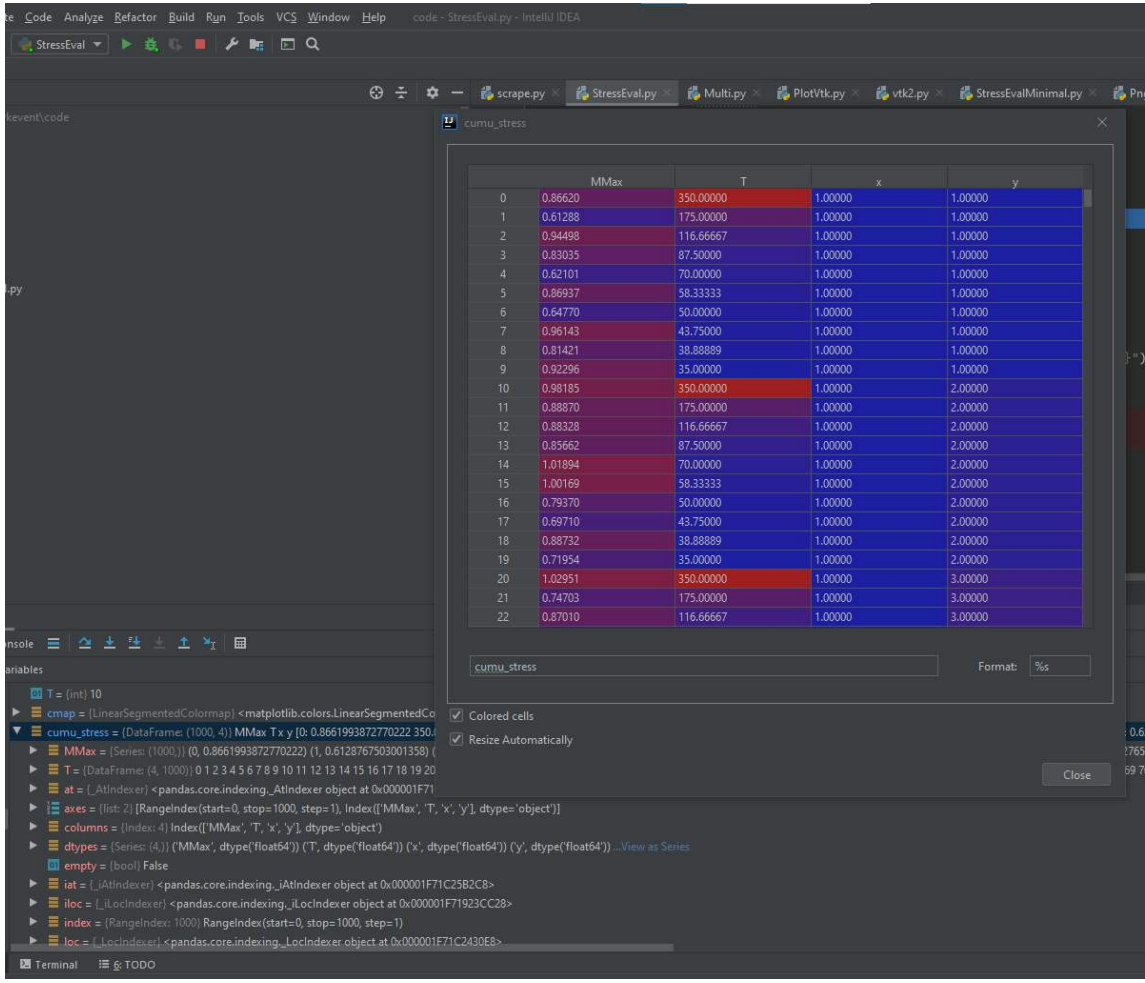


seaborn components used: `set_theme()`, `FacetGrid`

```
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
```

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

- 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)
```


resolvent Modelling Solutions

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

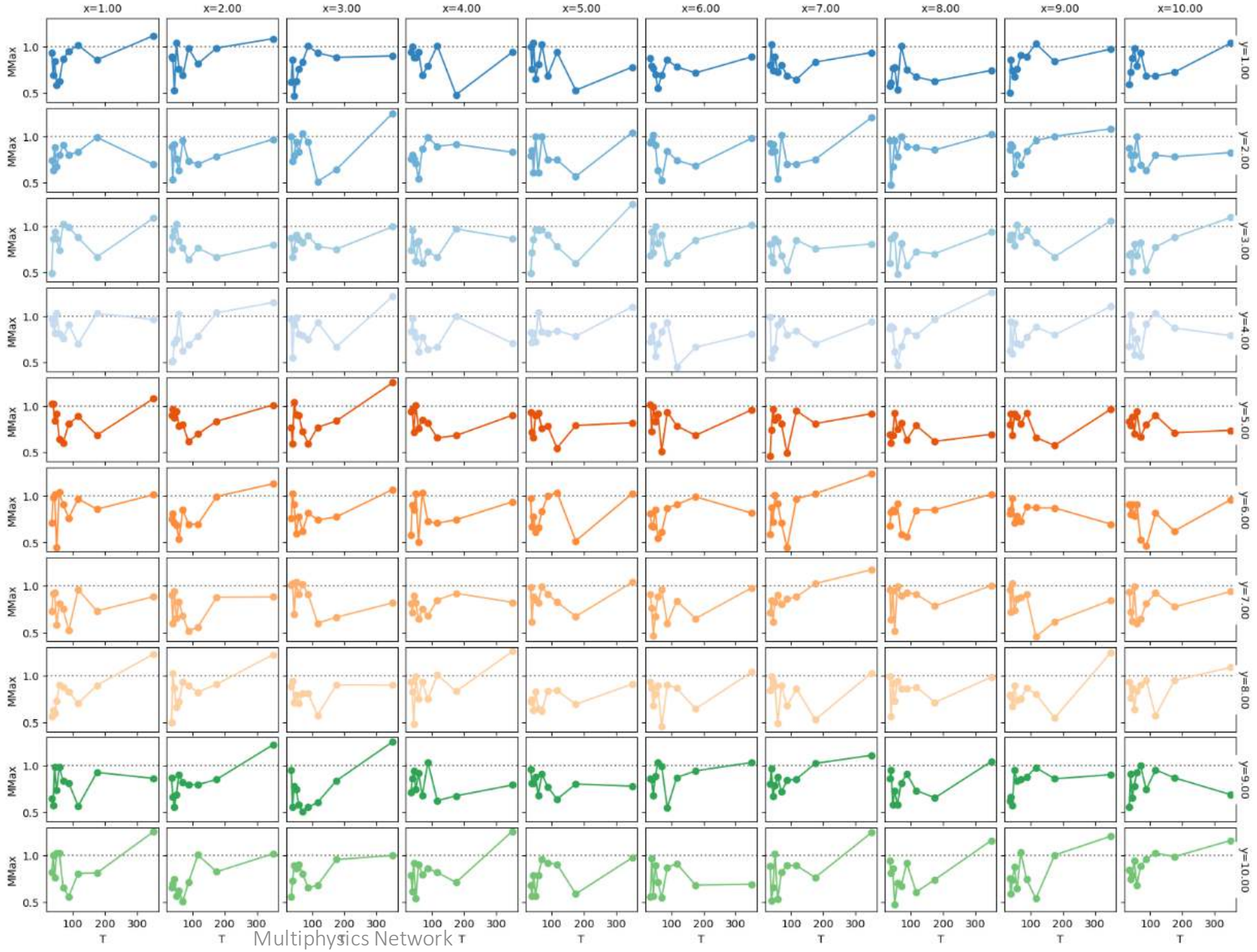
seaborn components used: `set_theme()`, `FacetGrid`

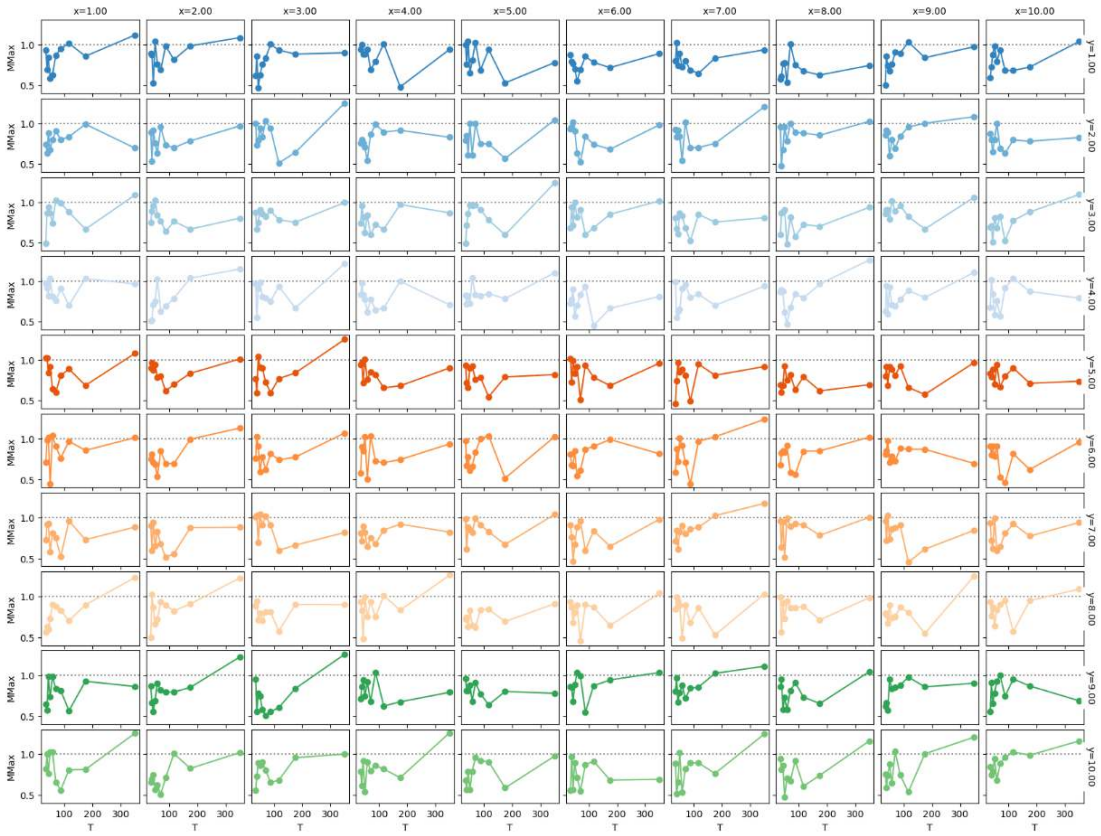
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

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’



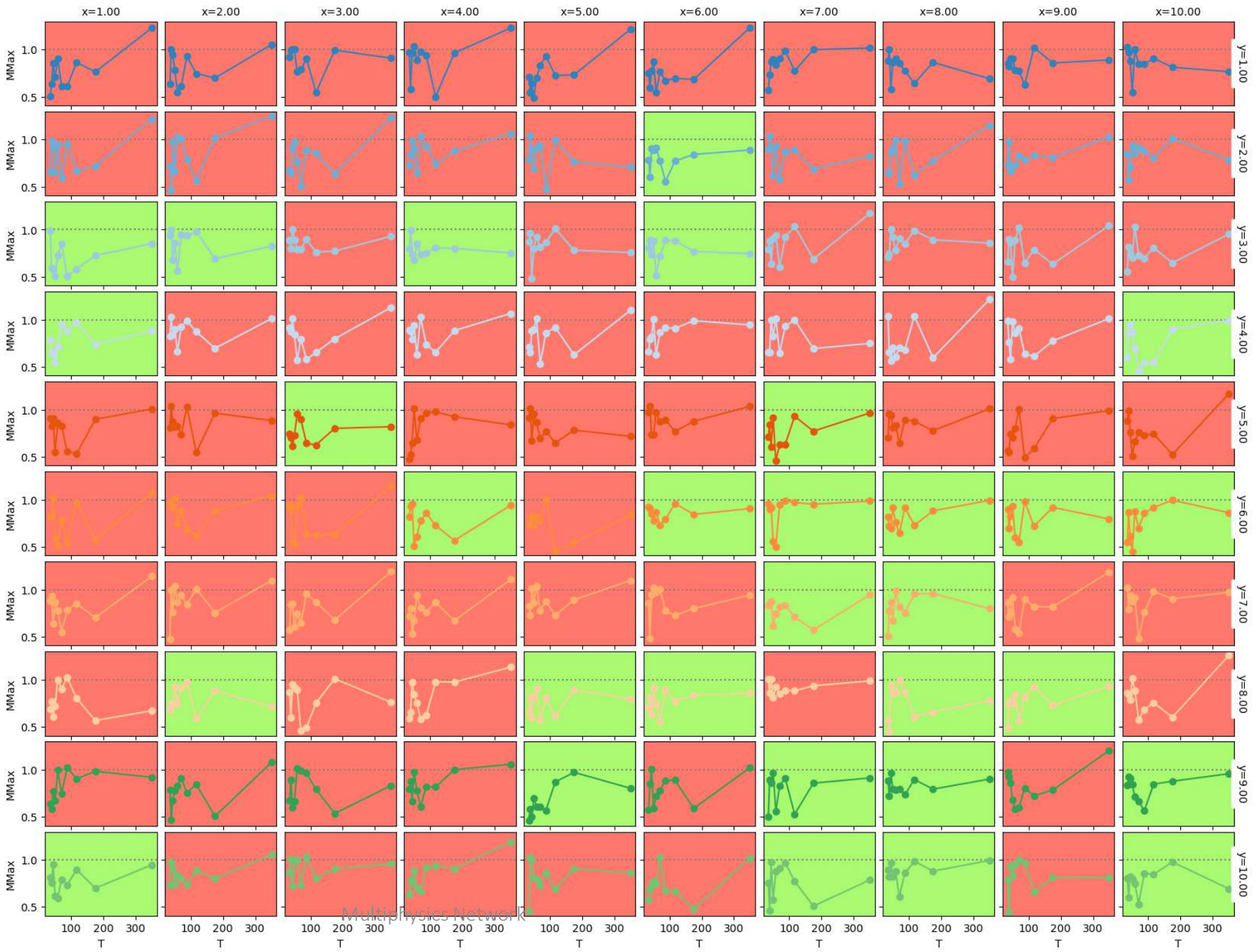


- Still quite dense to look at – can we maybe add some color to distinquish the x,y values that have a design that is ok?
- A few lines to loop over all the plots

```
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
```

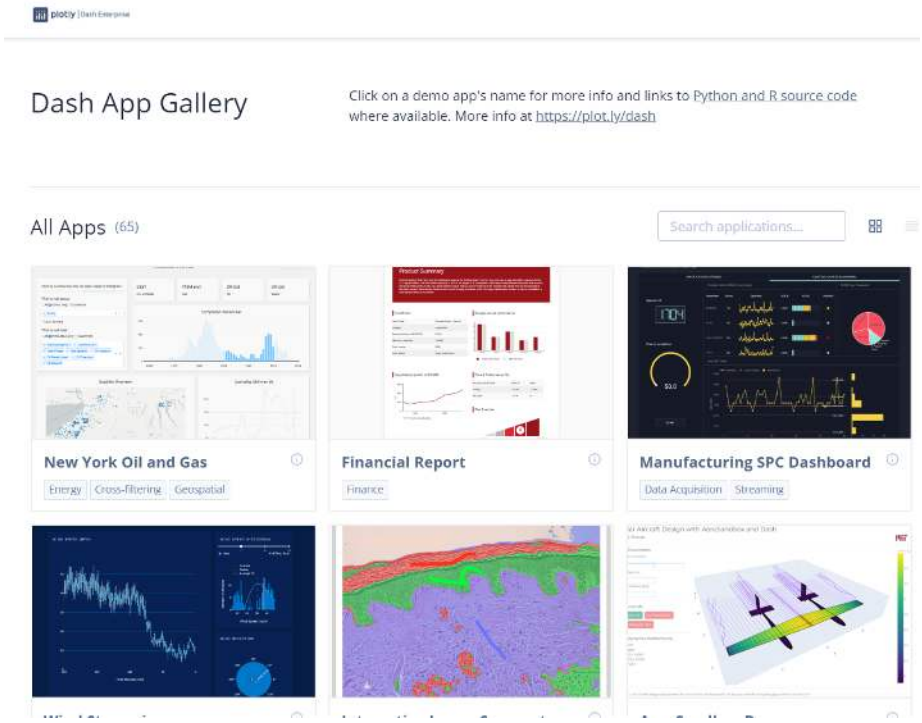

resolvent
Modelling Solutions

- 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...



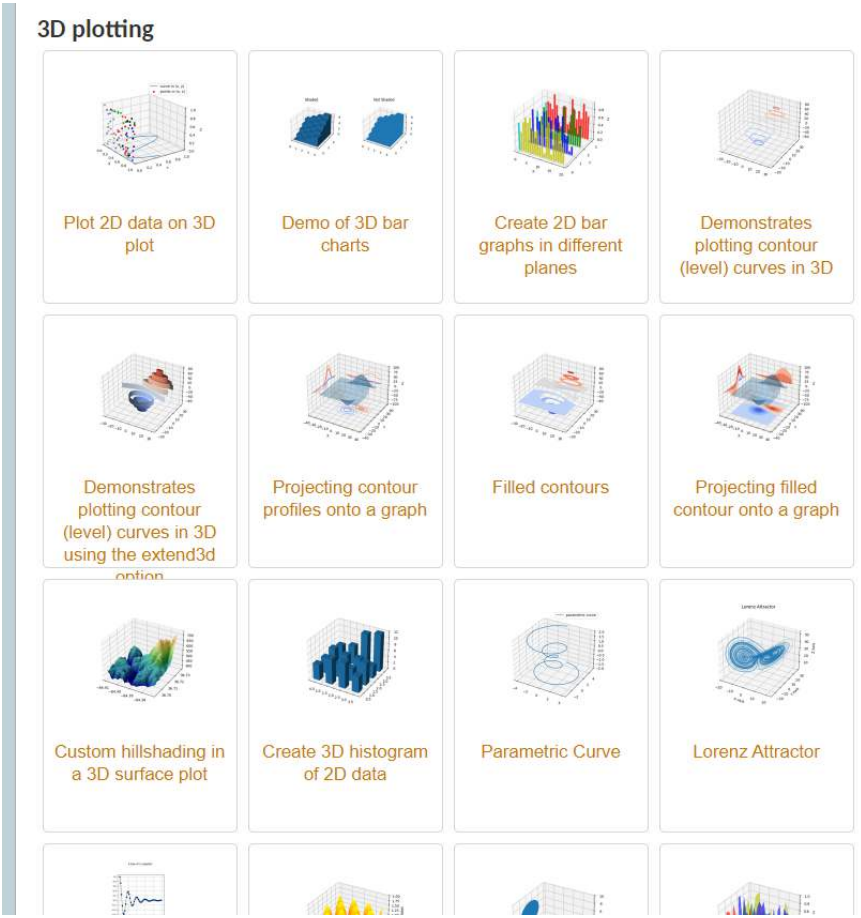
A note on large higher dimensional datasets

- 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
  - Ie. 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...



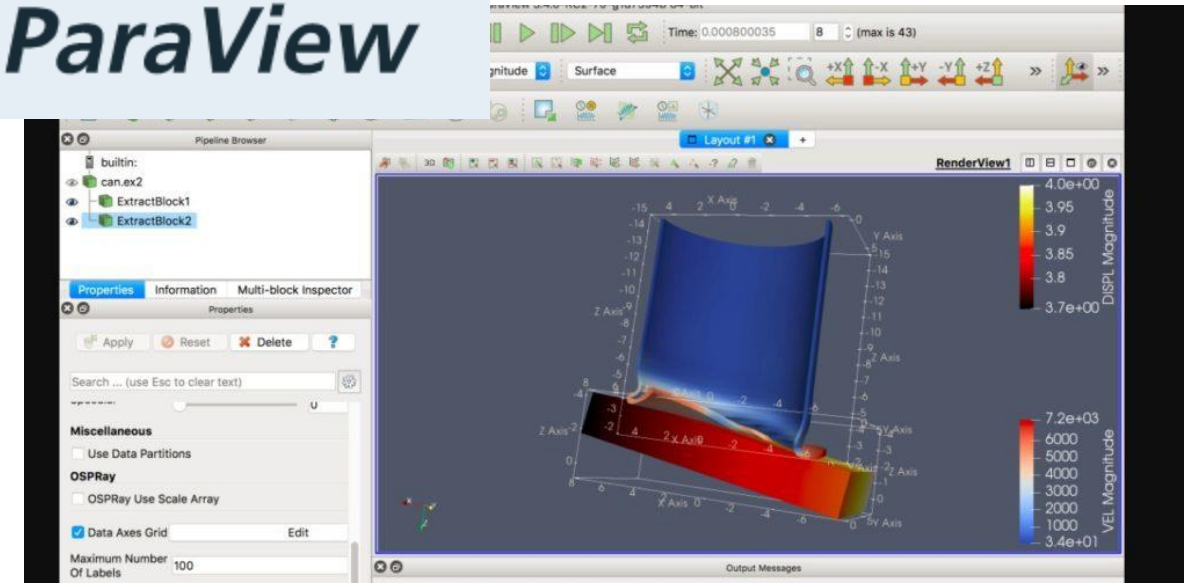
resolvent
Modelling Solutions

- 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

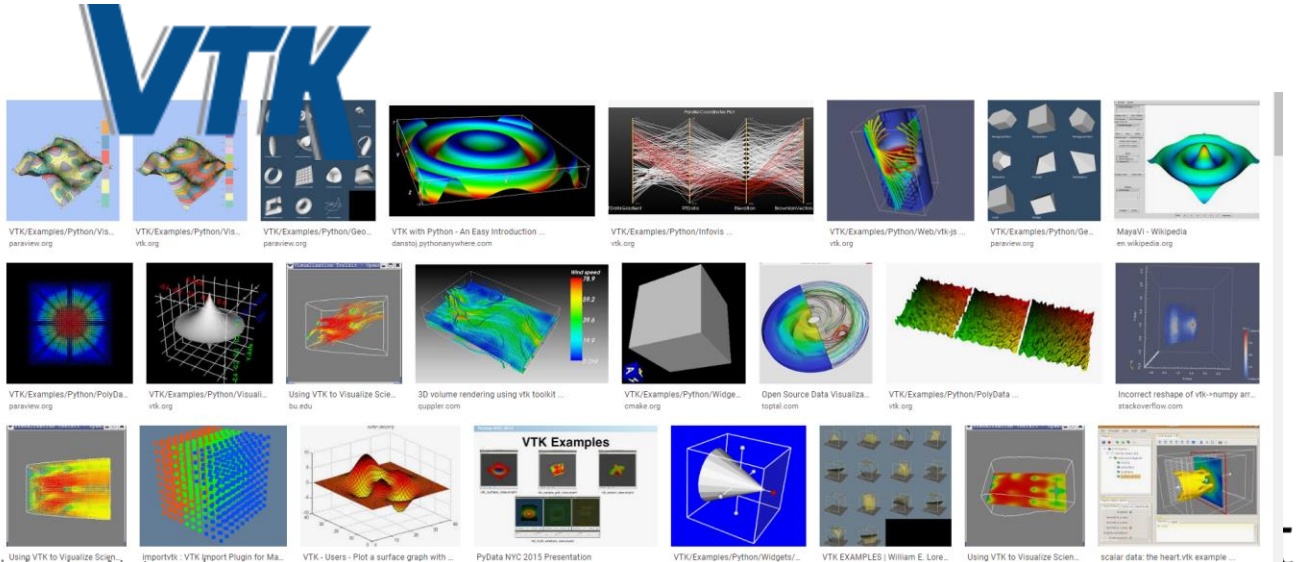


resolvent Modelling Solutions

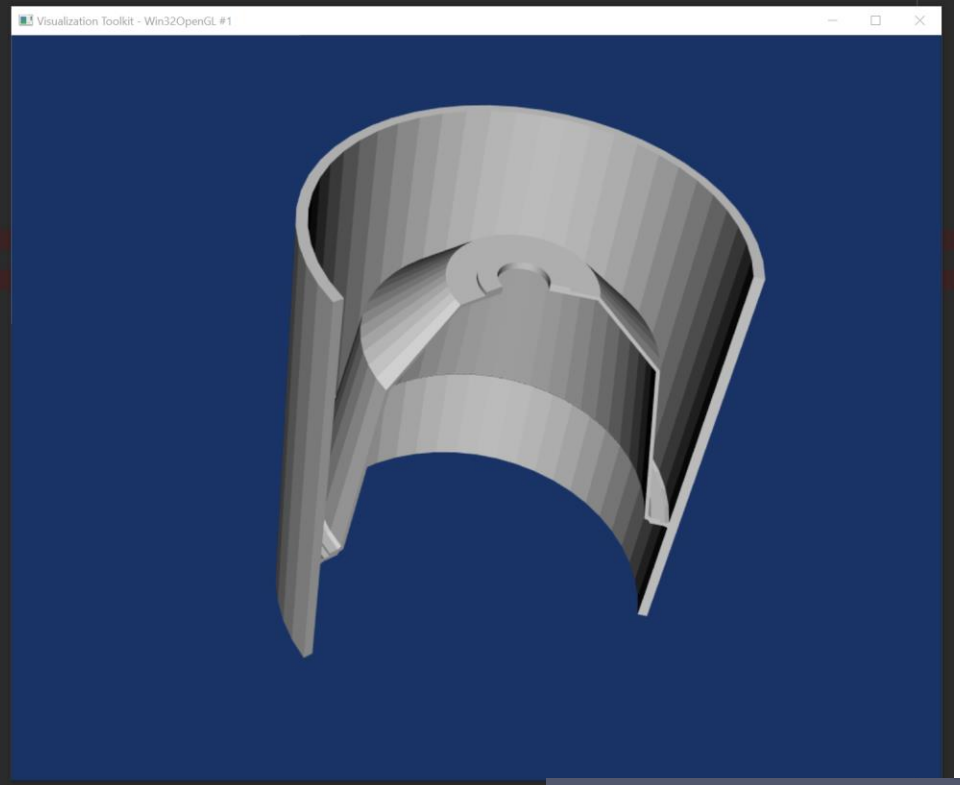
…enter VTK



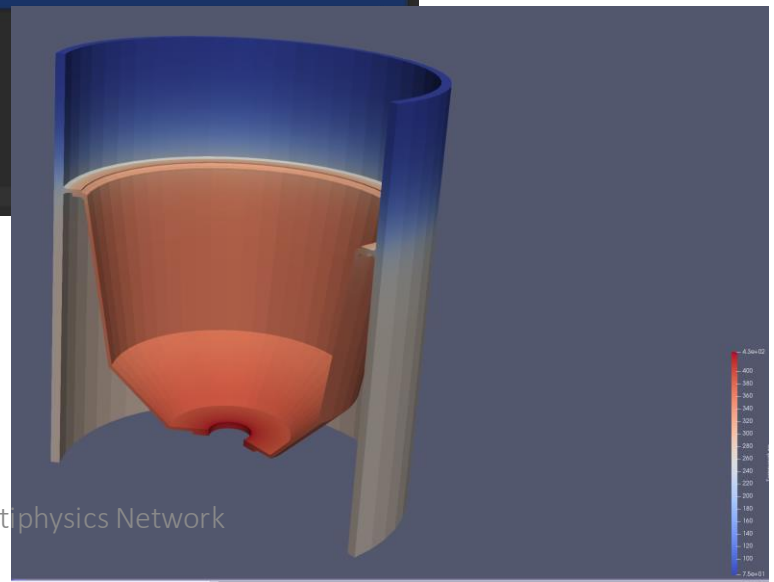
- Maybe best known as the underlying engine for Paraview
- Very powerful 3d plotting engine
- With good, well documented python bindings



```python
import vtk

# The source file
file_name = r"C:\Users\ToreFindsen\Desktop\temp\pressure_shell.vtu"

# Create a custom lut. The lut is used both at the mapper and at the
# scalar_bar
# lut = vtk.vtkLookupTable()
# lut.Build()

# Read the source file.
reader = vtk.vtkXMLUnstructuredGridReader()
reader.SetFileName(file_name)
reader.Update() # Needed because of GetScalarRange
output = reader.GetOutput()
scalar_range = output.GetAttributes(0).GetArray(0).GetRange()

mapper = vtk.vtkDataSetMapper()
if vtk.VTK_MAJOR_VERSION <= 5:
    mapper.SetInput(output)
else:
    mapper.SetInputData(output)
mapper.SetScalarRange(scalar_range)
# mapper.SetLookupTable(lut)

actor = vtk.vtkActor()
actor.SetMapper(mapper)

renderer = vtk.vtkRenderer()
renderer.AddActor(actor)
renderer.SetBackground(0.1, 0.2, 0.4)

render_window = vtk.vtkRenderWindow()
render_window.AddRenderer(renderer)
render_window.SetSize(300, 300)

interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# create the scalar_bar
# scalar_bar = vtk.vtkScalarBarActor()
# scalar_bar.SetOrientationToHorizontal()
# scalar_bar.SetLookupTable(lut)
#
# # create the scalar_bar_widget
# scalar_bar_widget = vtk.vtkScalarBarWidget()
# scalar_bar_widget.SetInteractor(interactor)
# scalar_bar_widget.SetScalarBarActor(scalar_bar)
# scalar_bar_widget.On()

interactor.Initialize()
render_window.Render()
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

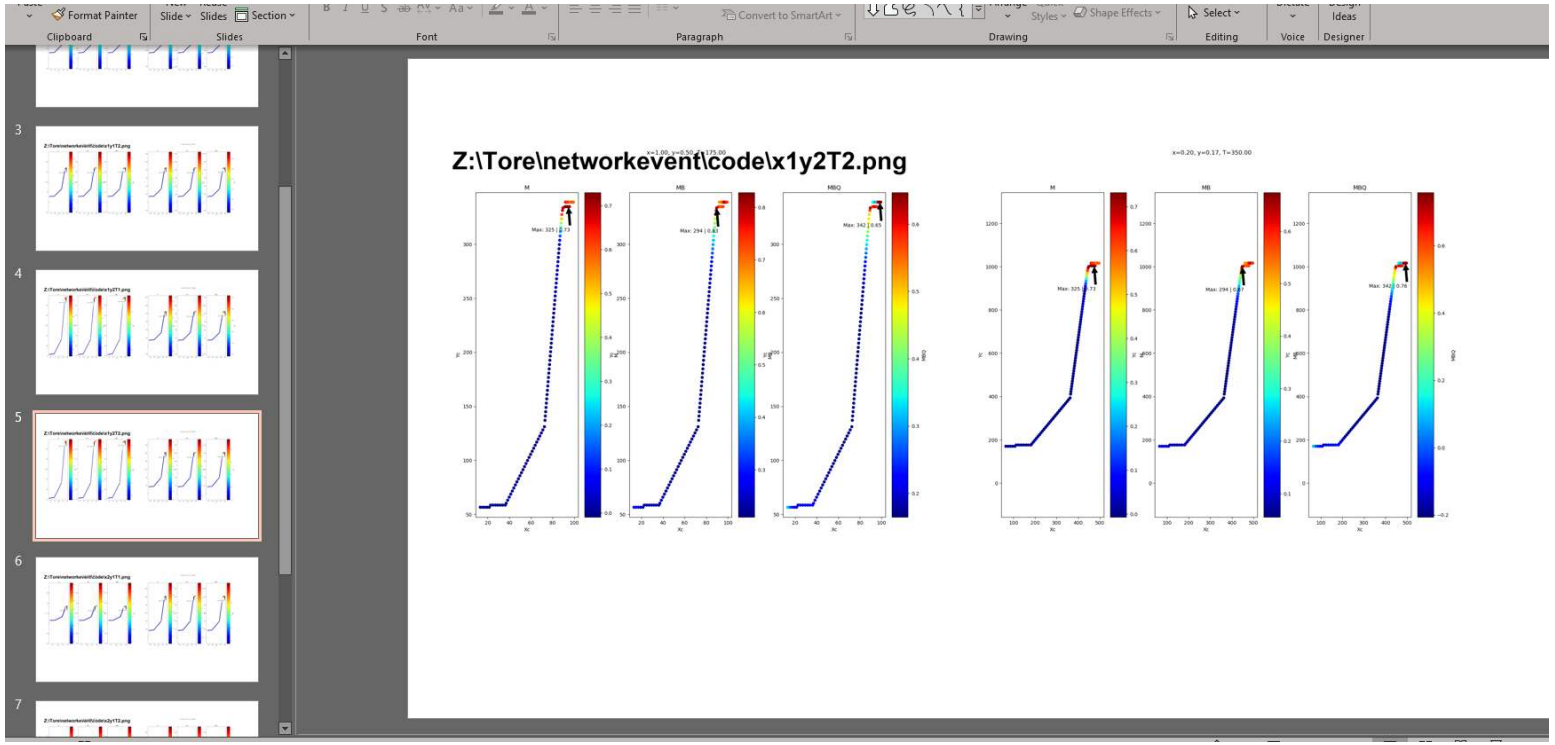
> `glob` — Unix style pathname pattern expansion
>
> **Source code:** Lib/glob.py

> **python-pptx**
>
> Release v0.6.18 (Installation)
>
> build passing
>
> *python-pptx* is a Python library for creating and updating PowerPoint (.pptx) files.

resolvent Modelling Solutions

```python
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(r"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)


prs.save('pics.pptx')
```

- 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

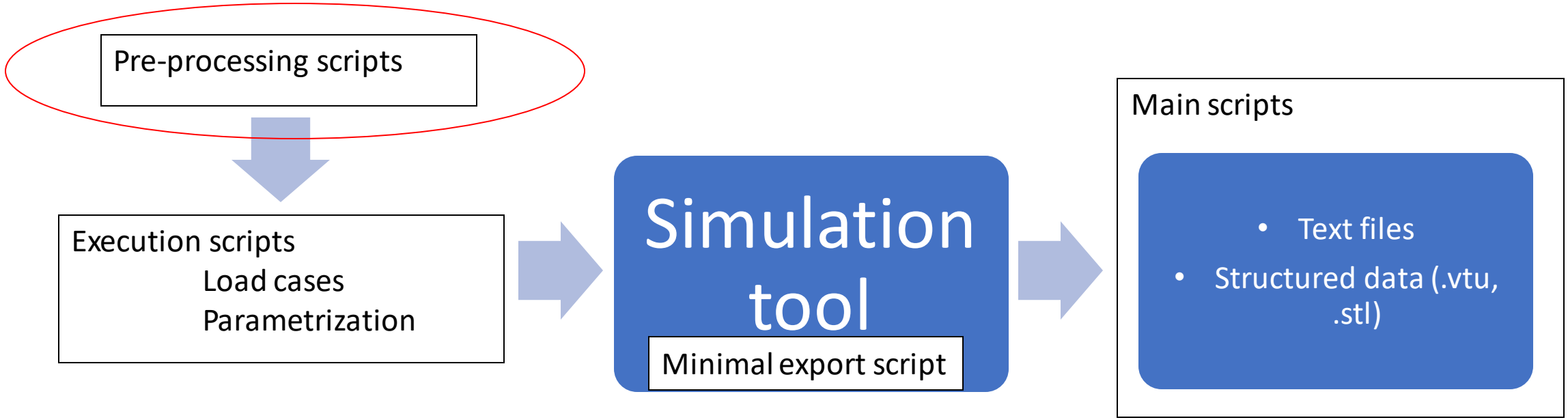


Take away:

- 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...

resolvent Modelling Solutions

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

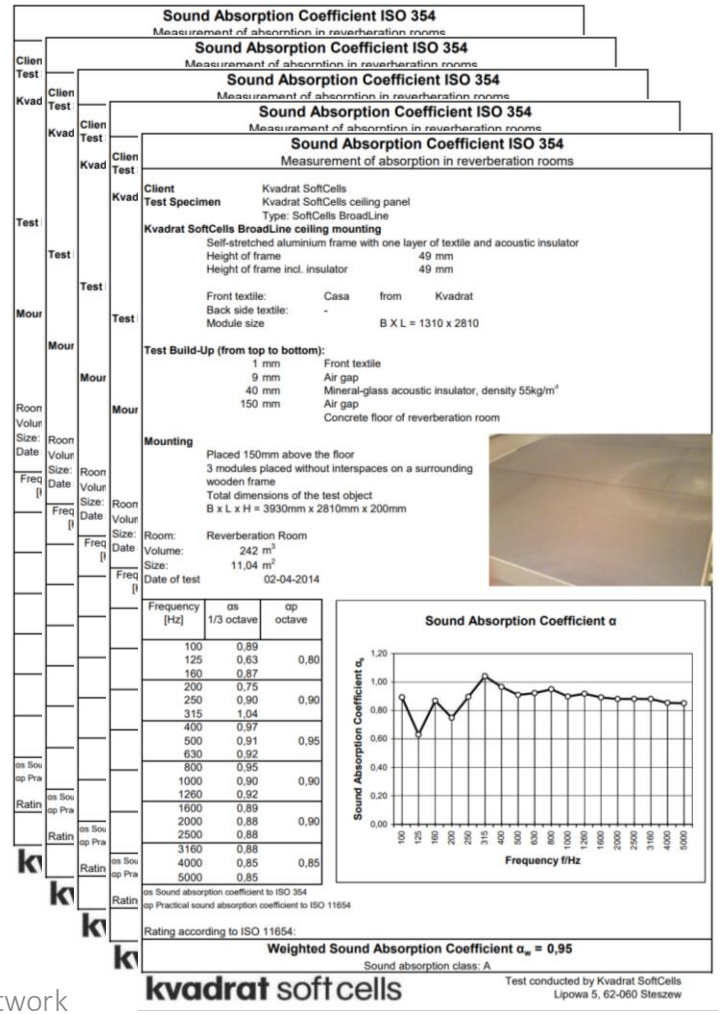


- So far our focus has been on post processing – we can script any aspect of our modelling workflow – a few examples

resolvent Modelling Solutions

Case 2: Small scale pdf data extraction

- We were doing a Comsol App for Kavdrat 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



resolvent Modelling Solutions

…enter PDFQuery

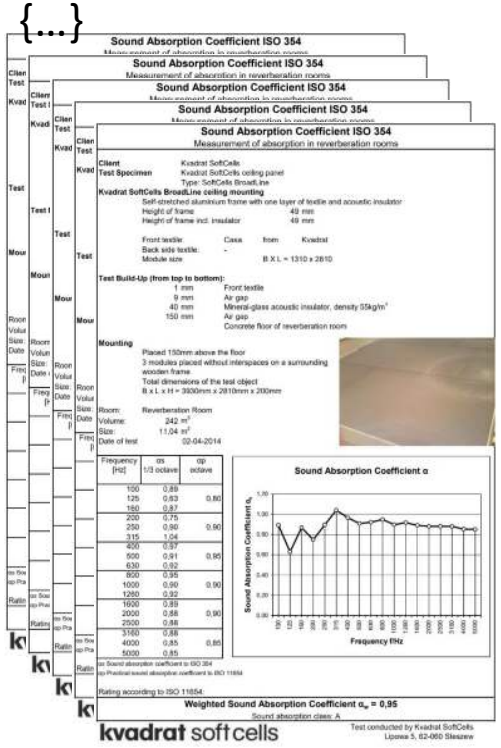
- The pdf is “reduced” to a hierachy that can be searched
- <100 lines of code allows us to traverse this hierarchy and get the data we need

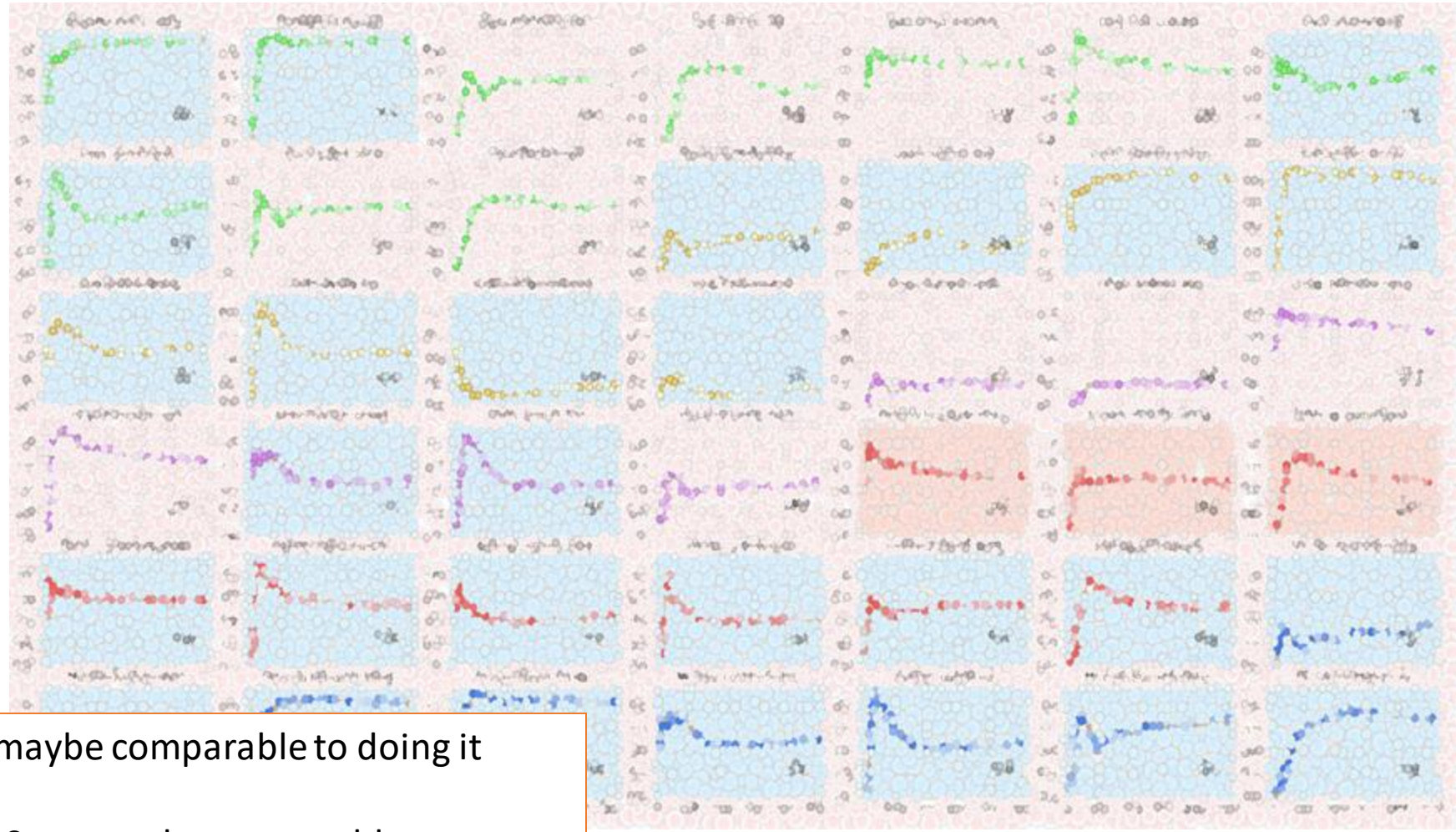
```python
import csv
import os
import re

from pdfquery import pdfquery


def get_by_offset(pdf, word, dir):...


def get_abs_data(pdf, n1, n2):...


def get_buildup(pdf, n1, n2):...


directory_in_str = r"E:\Softcells\From softcell"
directory = os.fsencode(directory_in_str)

out = []
for file in os.listdir(directory):
    filename = os.fsdecode(os.path.join(directory, file))

    pdf = pdfquery.PDFQuery(filename)
    pdf.load()

    room =re.findall(r'\d+', get_by_offset(pdf, "Volume:", [0, 0, 1000, 0]))[0]

    sc_type =re.findall(r"SoftCells (\w+)", pdf.pq('LTTextLineHorizontal:contains("{0}")'.format("Type:")).text(),re.IGNORECASE)[0]

    ft_string=get_by_offset(pdf, "Front textile:", [0, 0, 1000, 0])
    regexes = ["casa",r"[^\s]*asita","ginger","memory","mi casa"]
    combined = "(" + ")|(".join(regexes) + ")"
    front_textile = re.search(combined,ft_string,re.IGNORECASE)[0]

    mount_string=get_by_offset(pdf, "Type:", [-1000, -12, 1000, -12])
    regexes = ["ceiling","wall"]
    combined = "(" + ")|(".join(regexes) + ")"
    mount =str.lower(re.search(combined,mount_string,re.IGNORECASE)[0])

    ms_string = get_by_offset(pdf, "odule size", [0, 0, 1000, 0])
    module_size=[int(s) for s in ms_string.split() if s.isdigit()]

    td_string=get_by_offset(pdf, "Total dimensions", [0, -12, 0, -12])
    td_string_stripped=re.sub("mm","",td_string)
    total_dims = [int(s) for s in td_string_stripped.split() if s.isdigit()]

    test_date = get_by_offset(pdf, "Date", [0, 0, 1000, 0])

    buildup = get_buildup(pdf, "Concrete floor of reverberation room", "Test Build-Up (from top to bottom):")

    alpha_string = get_abs_data(pdf, "Sound absorption coefficient to ISO 354", "octave")
    alpha_matched=alpha_string.split(" ")[:36]
    alpha=(alpha_matched[:int(len(alpha_matched)/2)],alpha_matched[int(len(alpha_matched)/2):])
    out.append([room,sc_type,front_textile,mount,module_size,total_dims,test_date,buildup,alpha])

with open("output.csv", "w", encoding='utf-8') as f:
    writer = csv.writer(f, delimiter='|')
    writer.writerows(list(zip(*out)))
```


resolvent Modelling Solutions

Clients tests

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





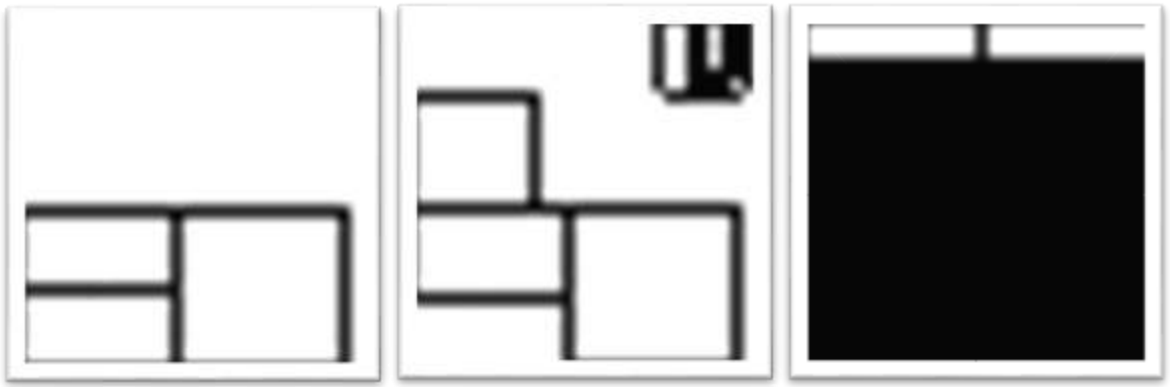
Blurred for IP reasons

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

resolvent
Modelling Solutions

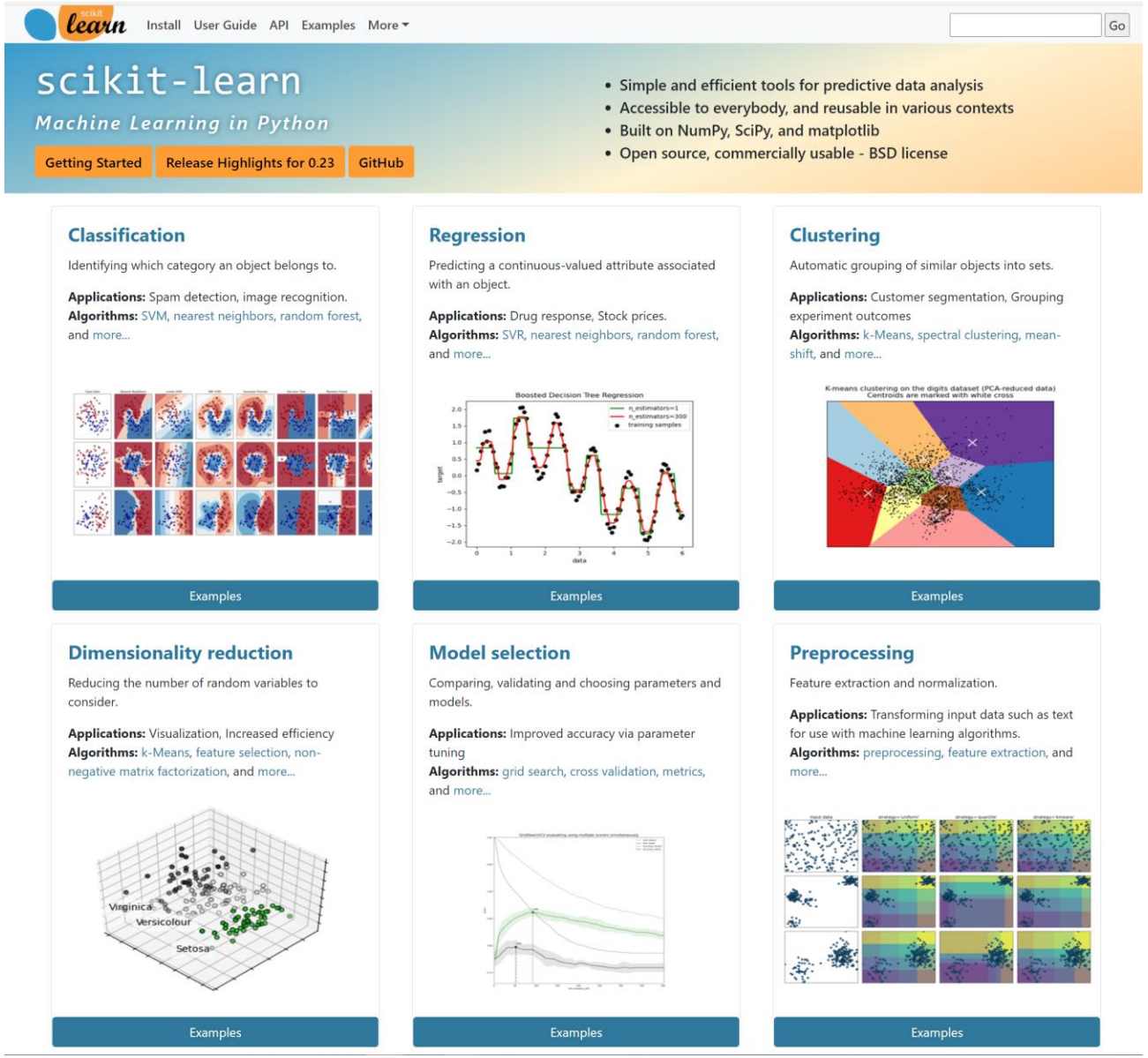
Case 3: Large scale pdf scraping using ML

- 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



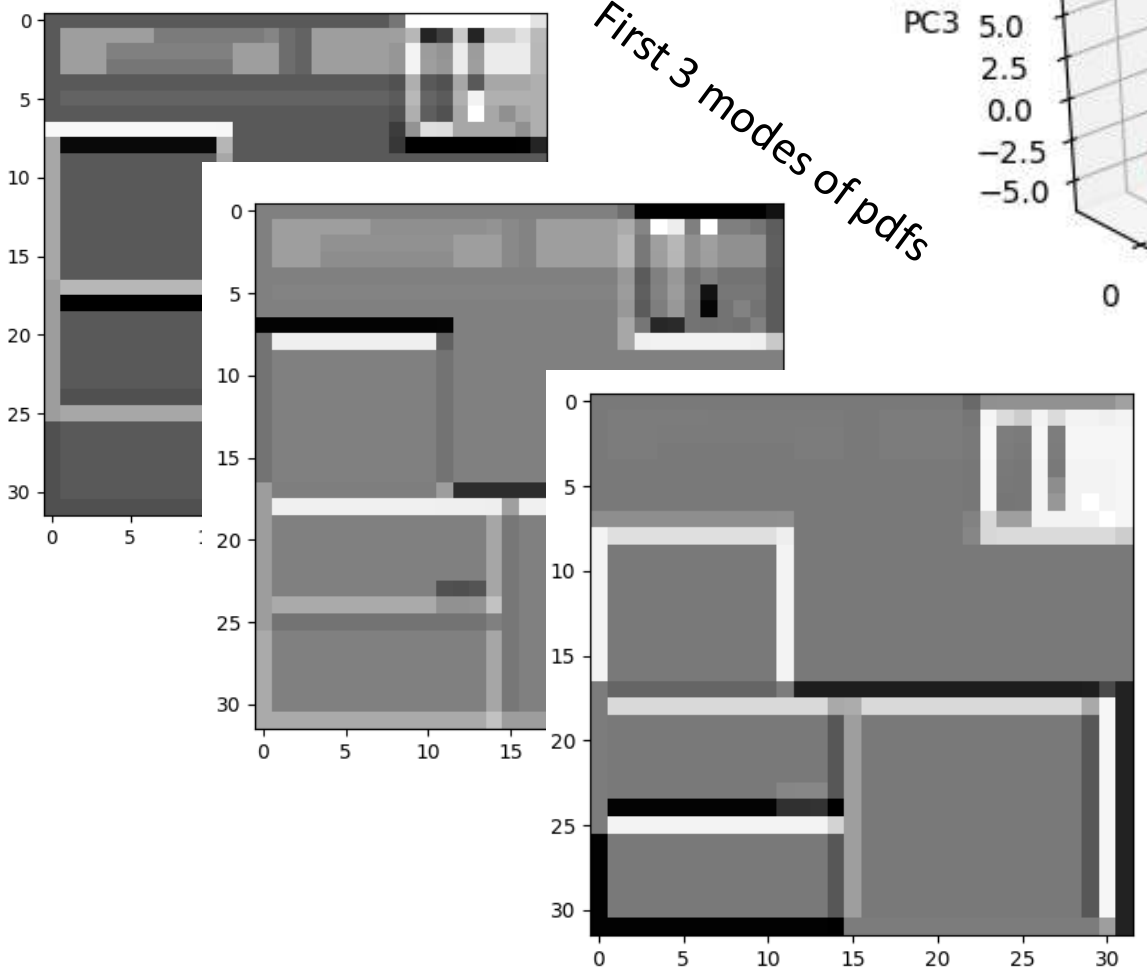
…enter scikit-learn

- Well integrated and documented ML library for Python

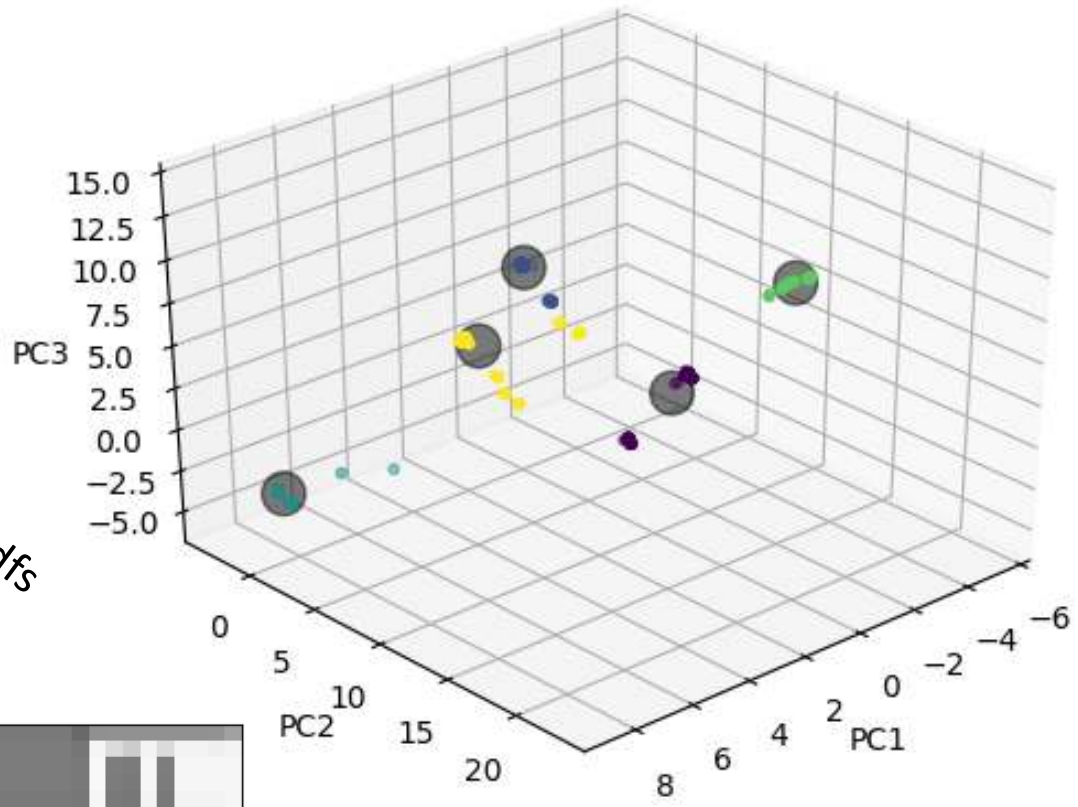


resolvent
Modelling Solutions

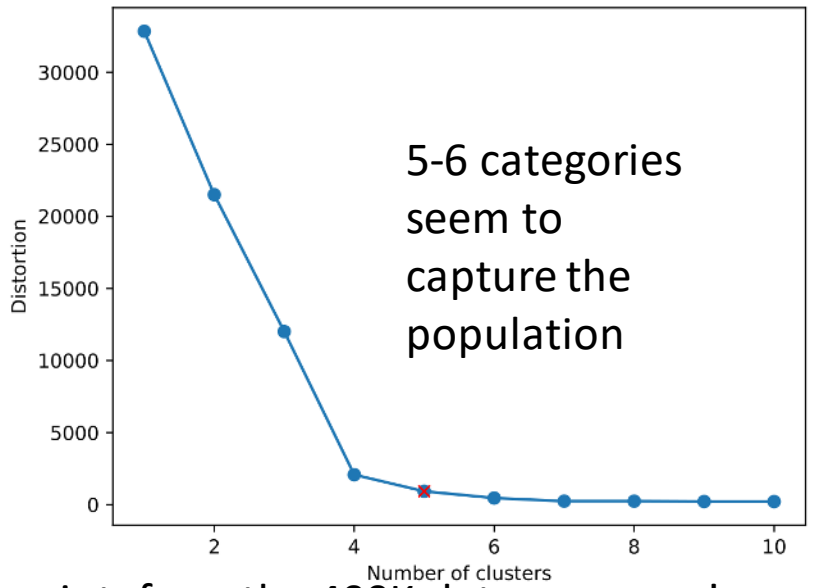
PCA + cluster



First 3 modes of pdfs



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



5-6 categories seem to capture the population

We were able to extract 120K datapoints from the 400K data – success!

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 you workflow?
  - Languages
  - Editors
  - Modules
- Are you facing any impediments?
  - Time constraints
  - Learnings curves
  - IT security
  - License considerations