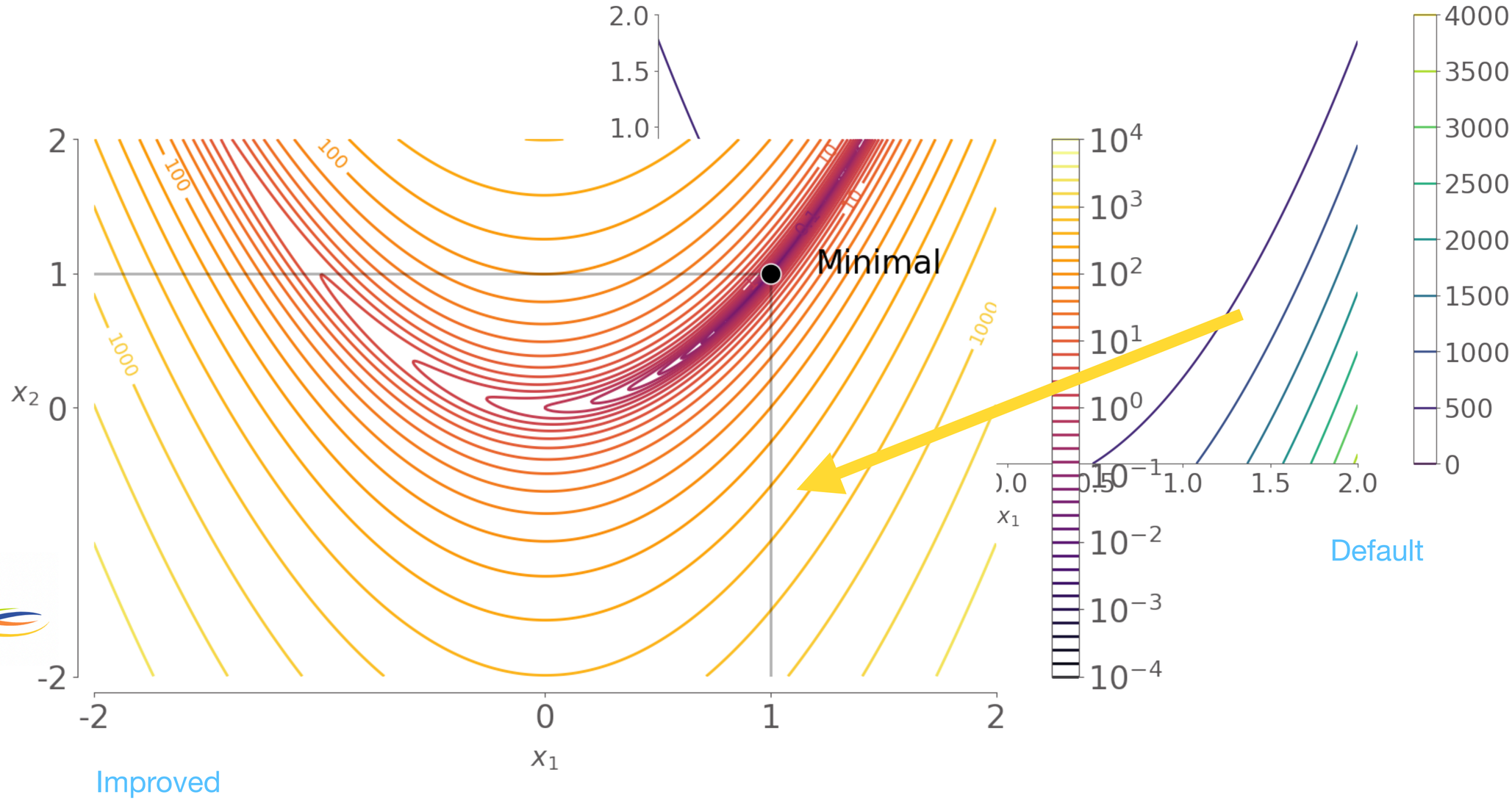
Generating nice figures for your research

Sicheng He

PostDoc Associate

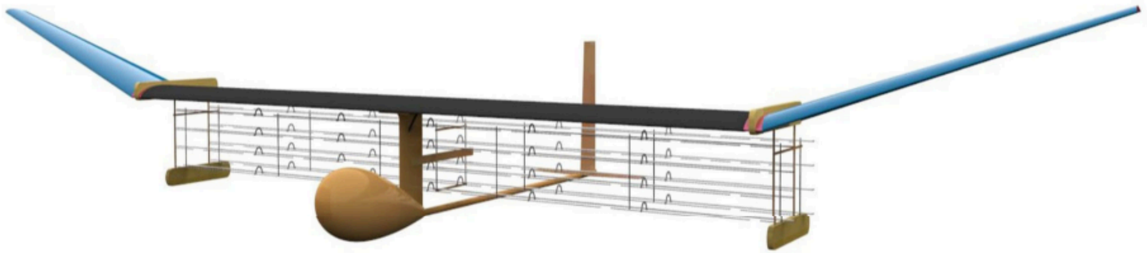
Knowledge and
Expertise Sharing Seminar
Oct 14, 2022
LAE, MIT



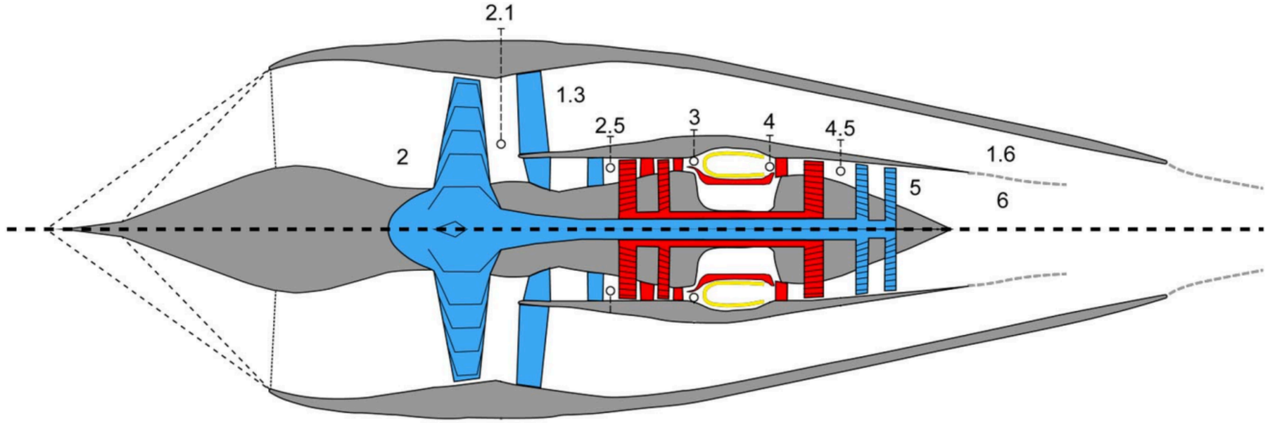
Why does it matter?

- It reduces effort for people to understand your work.
- It forces you to think clearly and do better research.

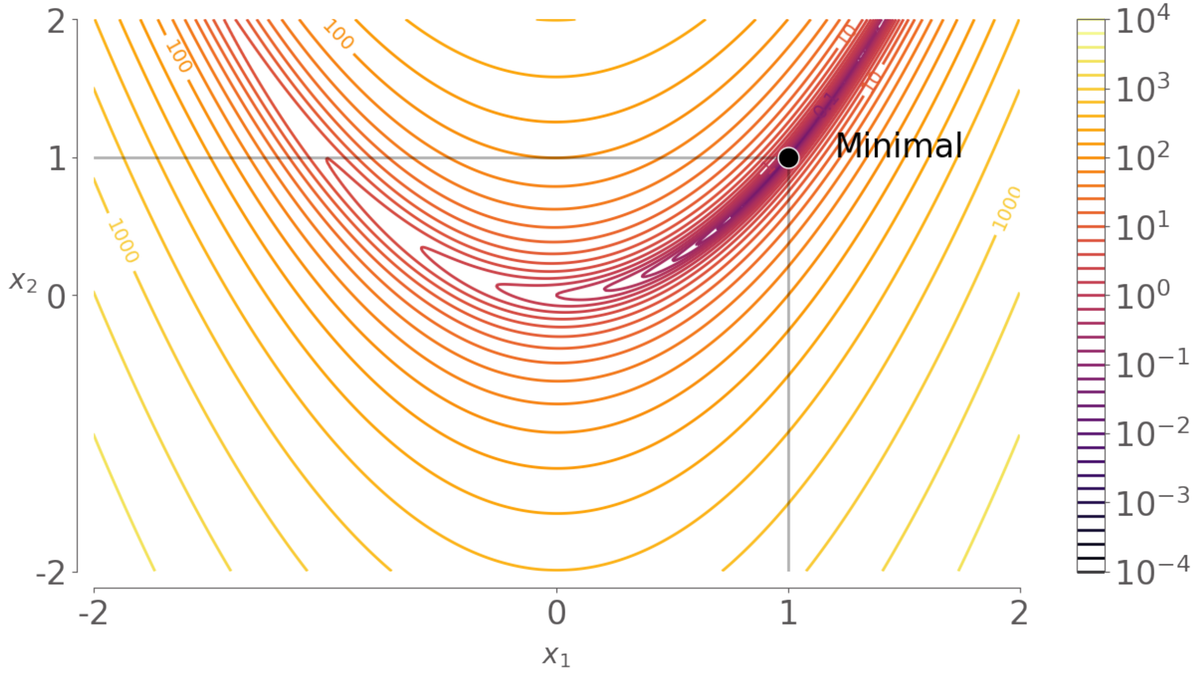
Different types of figures …

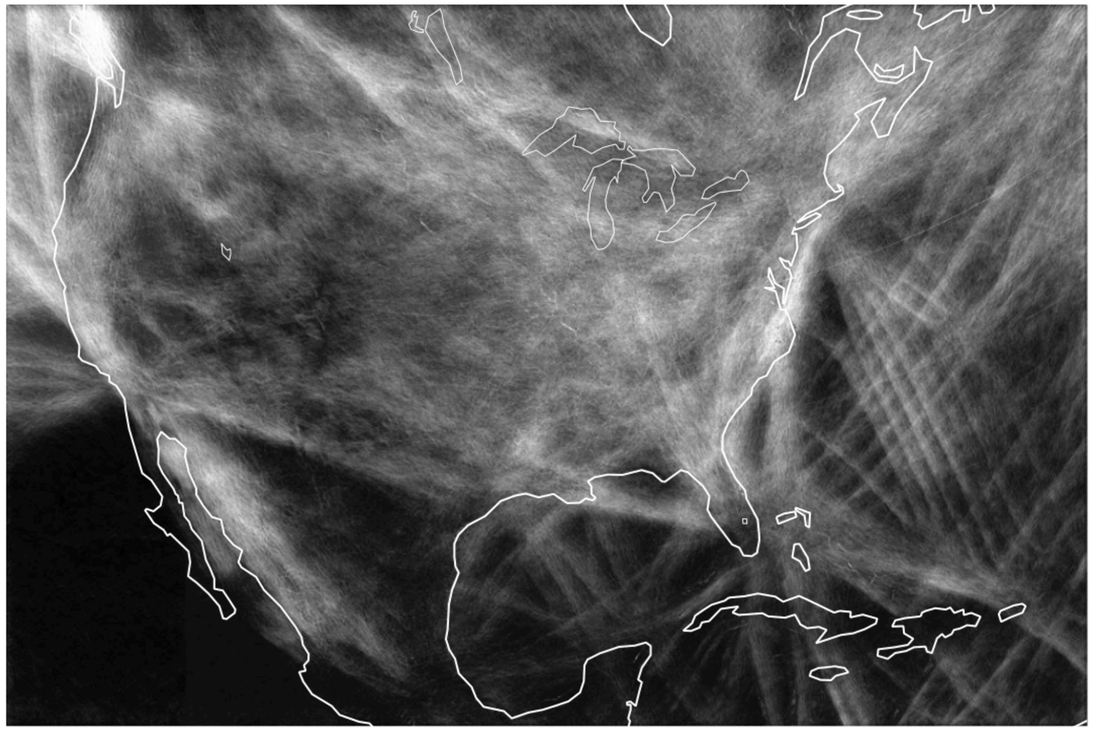


Xu et al. Nature 2018



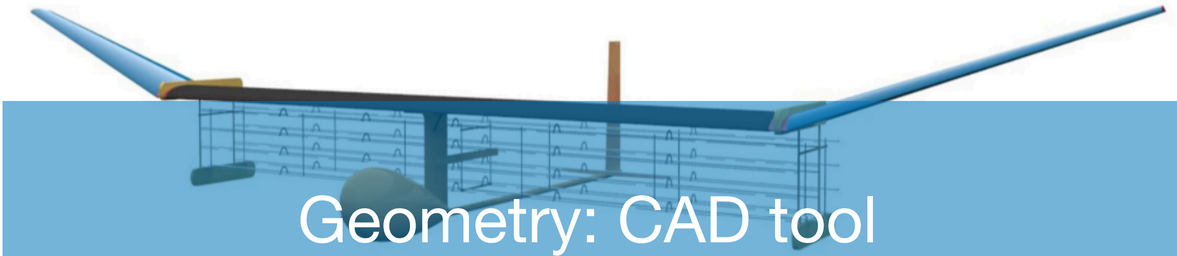
Voet et al. Scitech 2021





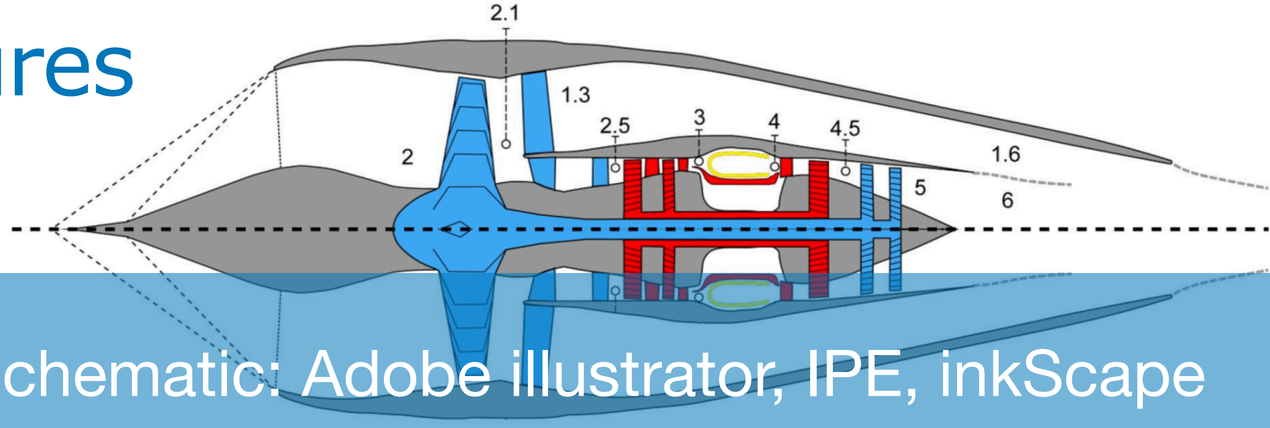
Meijer et al. Environmental Research Letters 2022

Different tools for different figures



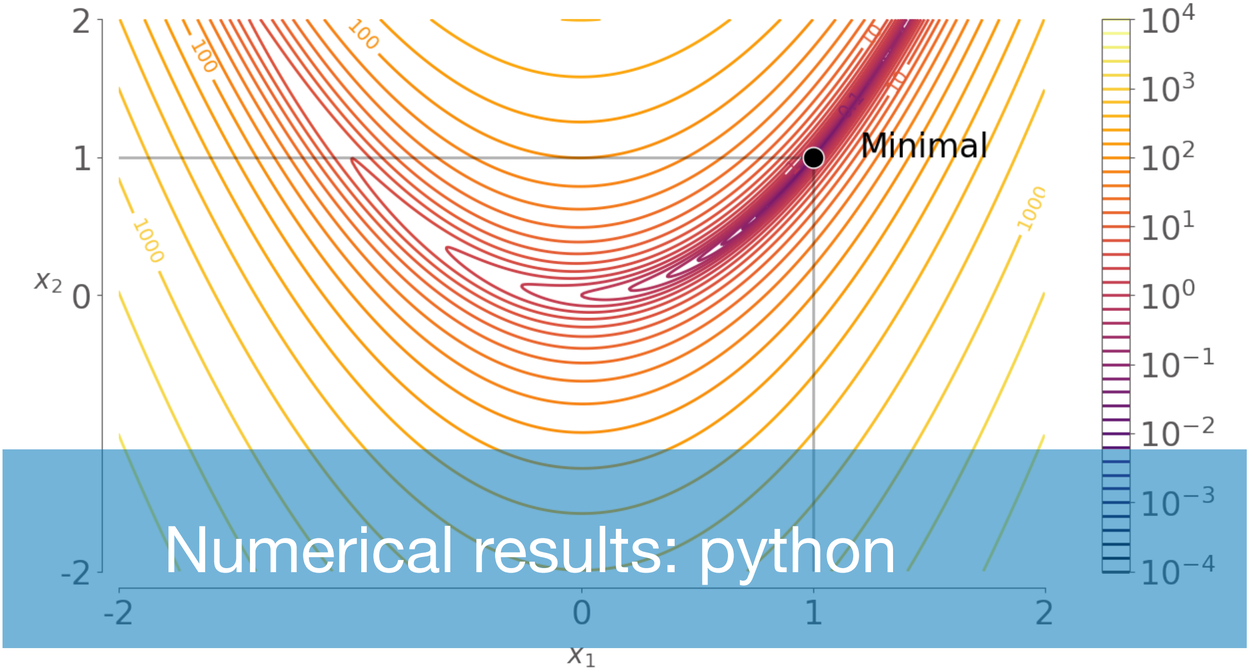
Geometry: CAD tool

Xu et al. Nature 2018

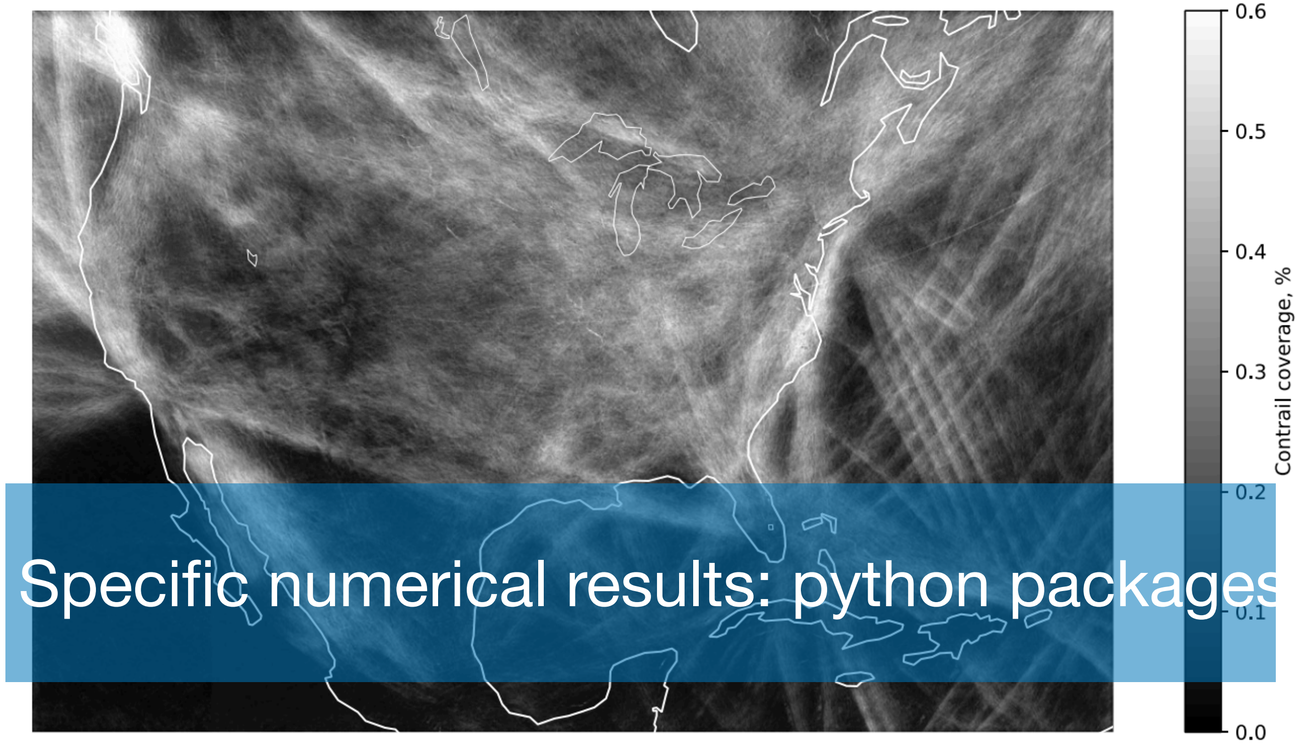


Schematic: Adobe illustrator, IPE, inkScape

Voet et al. Scitech 2021



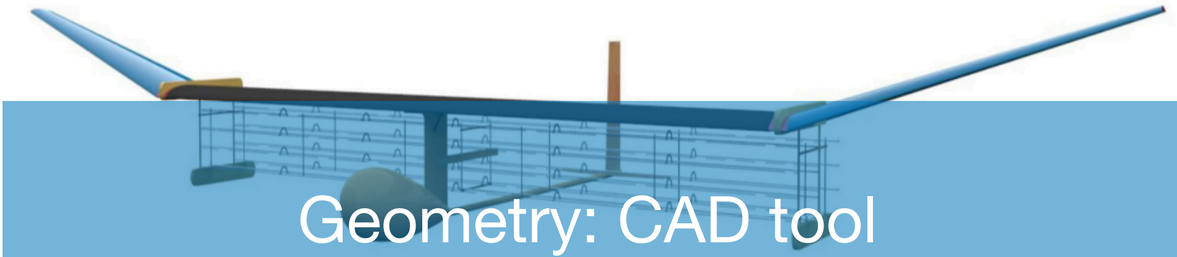
Numerical results: python



Specific numerical results: python packages

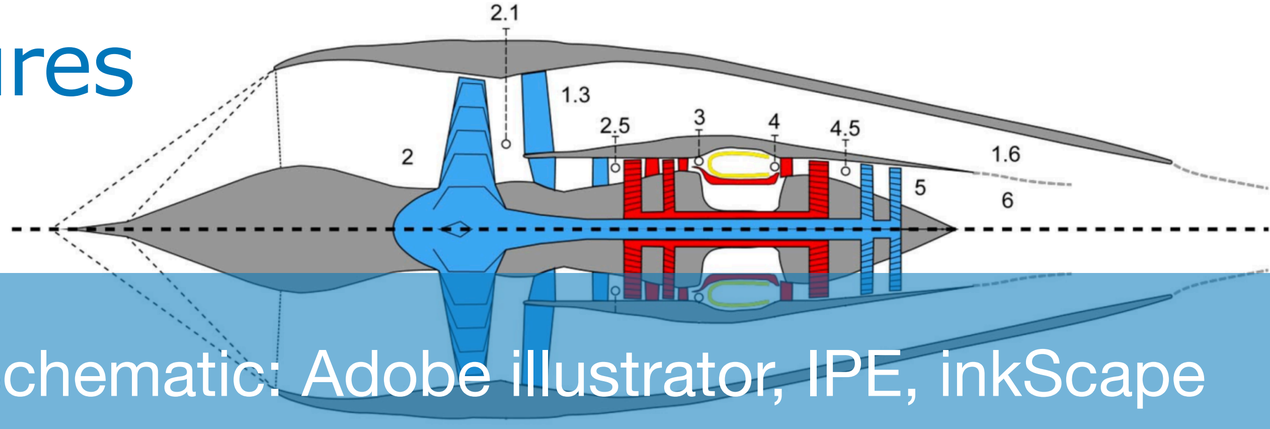
Meijer et al. Environmental Research Letters 2022

Different tools for different figures



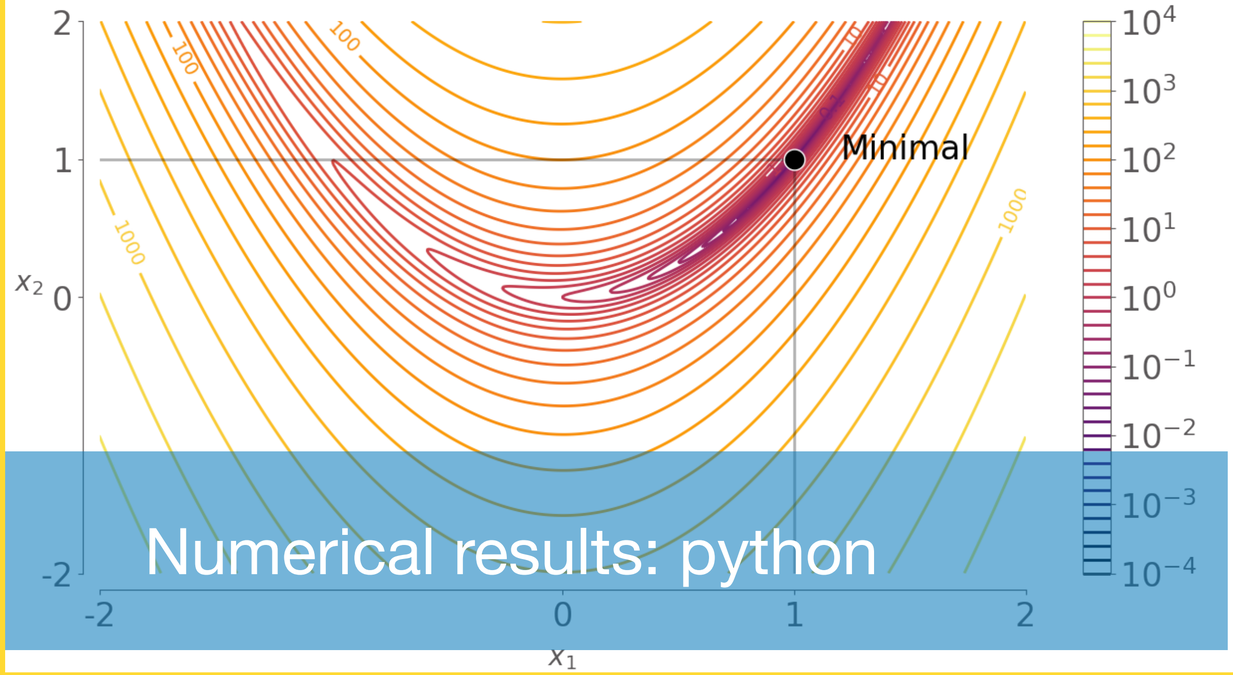
Geometry: CAD tool

Xu et al. Nature 2018

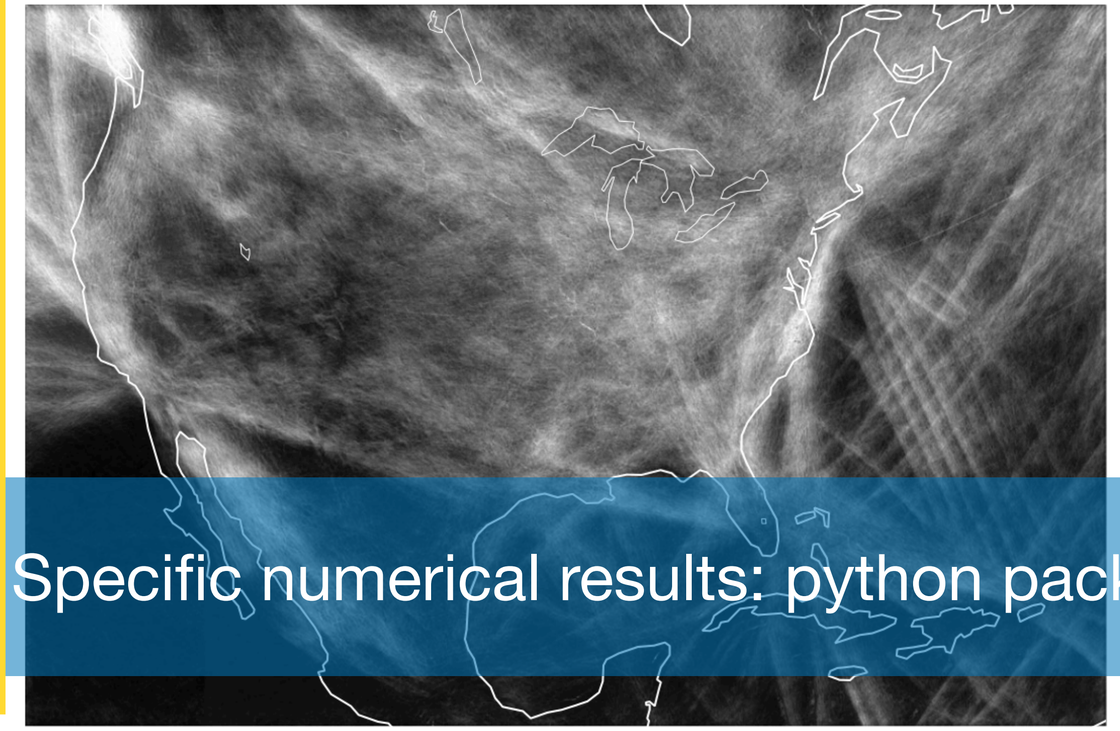


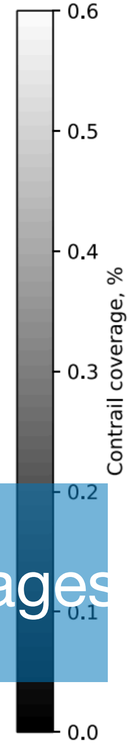
Schematic: Adobe illustrator, IPE, inkScape

Voet et al. Scitech 2021



Numerical results: python





Specific numerical results: python packages

Meijer et al. Environmental Research Letters 2022

Focus of today

A promise ...

You will learn how to generate publication quality curve and contour plot using python and matplotlib.

Outline

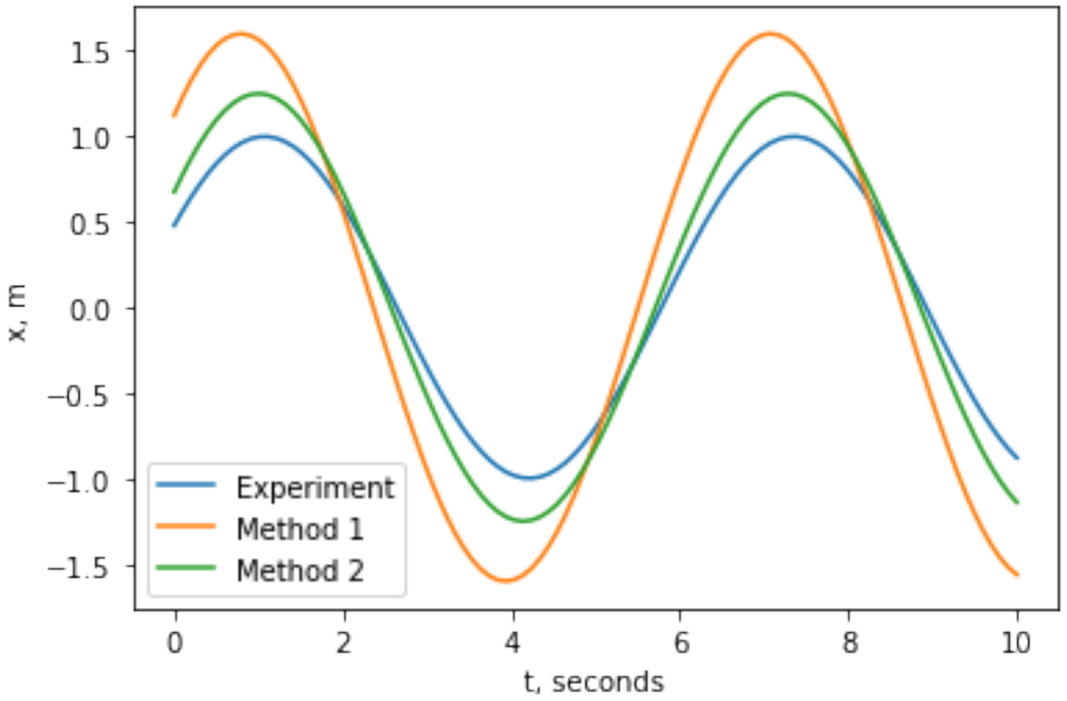
- Module 1: What and how?
- Module 2: Try it yourself
- Module 3: Conclusion

Module 1: What and how?

Define nice plots ...

- Accurately convey n pieces of critical information (usually $1 \leq n \leq 3$).
- High information / noise ratio.
- Good style.

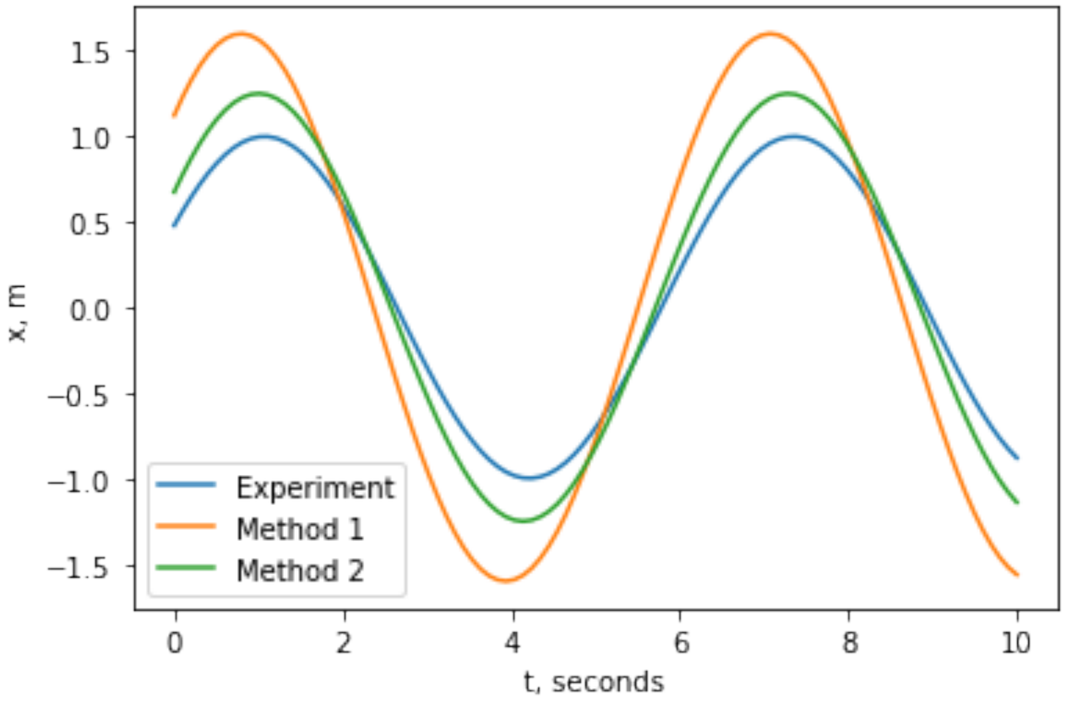
Case study 1: Curve plot



Message:
Method 2 outperforms Method 1 because it is closer to the experimental data.

```
# Initialize the figure
fig, ax = plt.subplots()

# Lines and legends
ax.plot(x, yref, label = "Experiment")
ax.plot(x, y1, label = "Method 1")
ax.plot(x, y2, label = "Method 2")

# x, y labels
ax.set_ylabel('x, m')
ax.set_xlabel('t, seconds')

# plot
ax.legend()
plt.show()
```

What are the issues?



Message:
Method 2 outperforms Method 1 because it is closer to the experimental data.

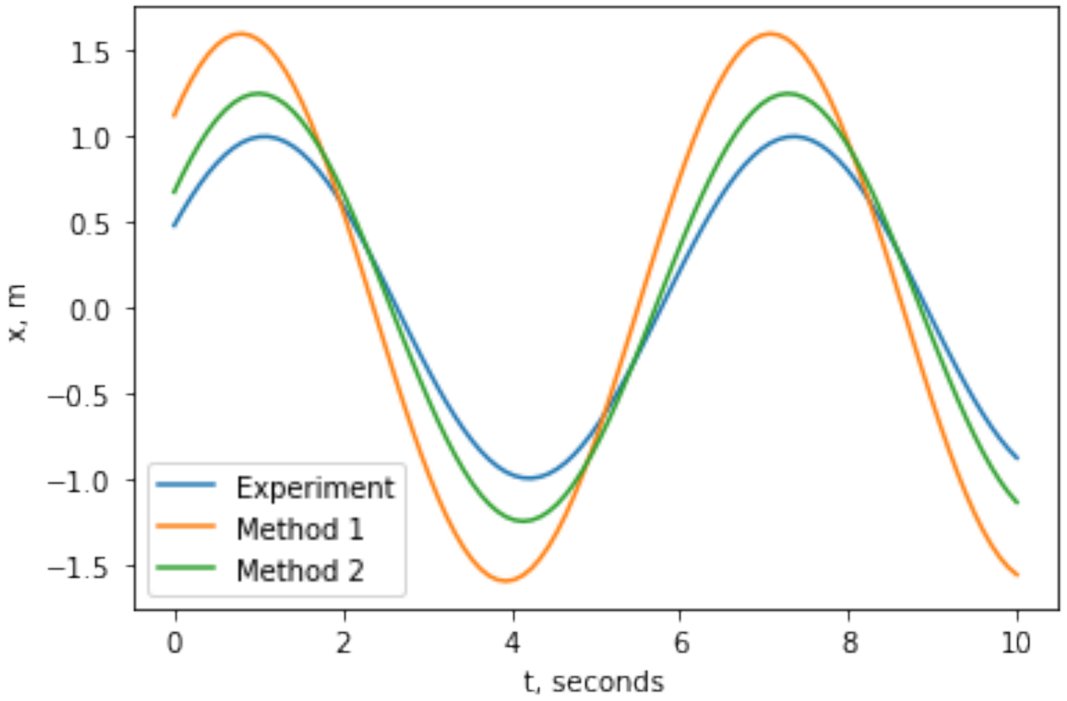
```
# Initialize the figure
fig, ax = plt.subplots()

# Lines and legends
ax.plot(x, yref, label = "Experiment")
ax.plot(x, y1, label = "Method 1")
ax.plot(x, y2, label = "Method 2")

# x, y labels
ax.set_ylabel('x, m')
ax.set_xlabel('t, seconds')

# plot
ax.legend()
plt.show()
```

An algorithm

Algorithm 1.1 Figure improving algorithm

Input Message set, original figure
for message $\in$ message set **do**
    if message unclear **then**
        Fix message.
    end if
end for
set = {labels, legends, boxline, words, lines, ticks, colorbar, . . .}
for feature $\in$ set **do**
    if feature unclear **then**
        Fix feature.
    end if
end for
Return Improved figure

What are the issues?



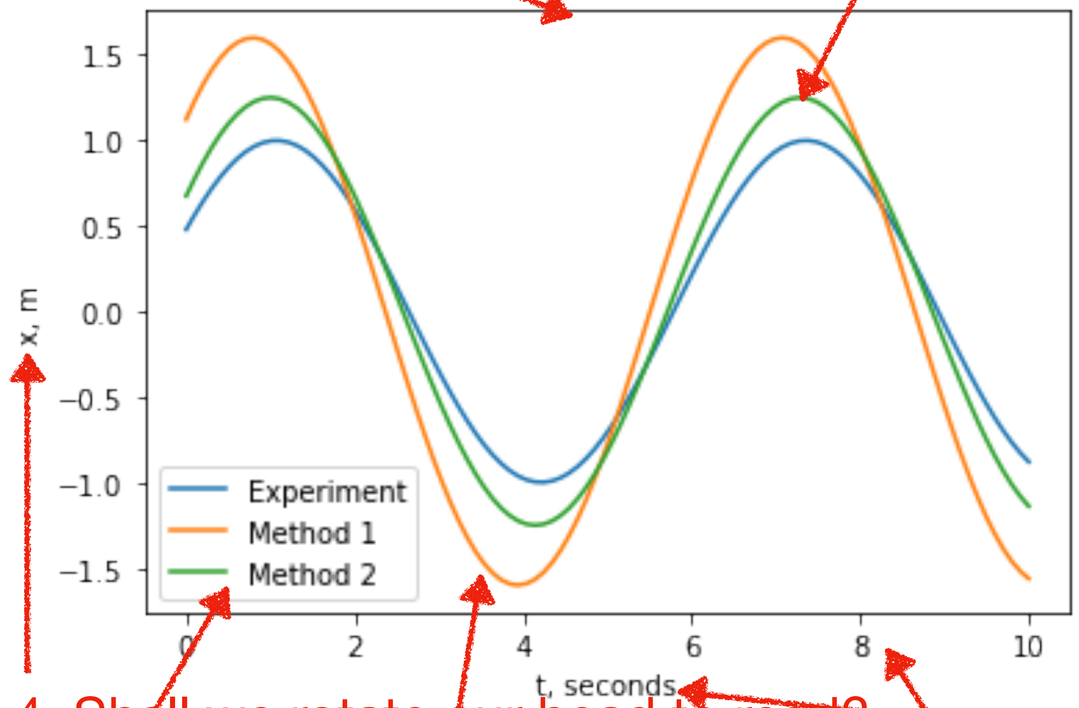
Algorithm 1.1 Figure improving algorithm

Input Message set, original figure
for message ∈ message set **do**
  if message unclear **then**
    Fix message.
  end if
end for
set = {labels, legends, boxline, words, lines, ticks, colorbar, . . .}
for feature ∈ set **do**
  if feature unclear **then**
    Fix feature.
  end if
end for
Return Improved figure

What are the issues?

1. OK, 1 is better than 2, but by how much?

7. Do we need to box it? Noise!



Algorithm 1.1 Figure improving algorithm

Input Message set, original figure
for message ∈ message set **do**
  if message unclear **then**
    Fix message.
  end if
end for
set = {labels, legends, boxline, words, lines, ticks, colorbar, . . .}
for feature ∈ set **do**
  if feature unclear **then**
    Fix feature.
  end if
end for
Return Improved figure

4. Shall we rotate our head to read?

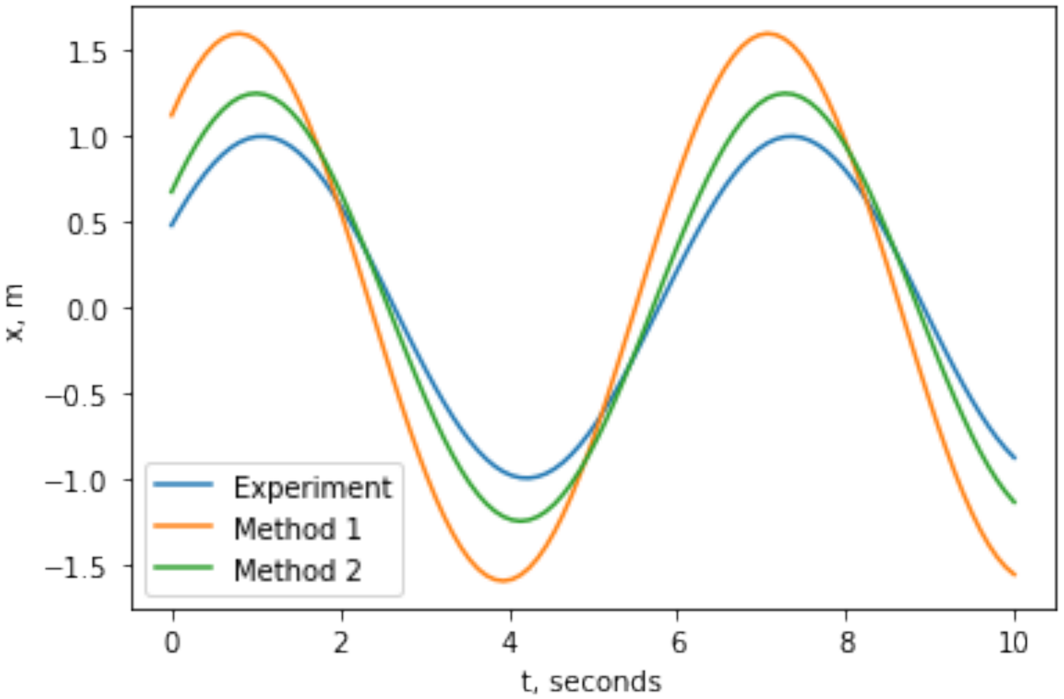
6. Math symbol, use Latex!

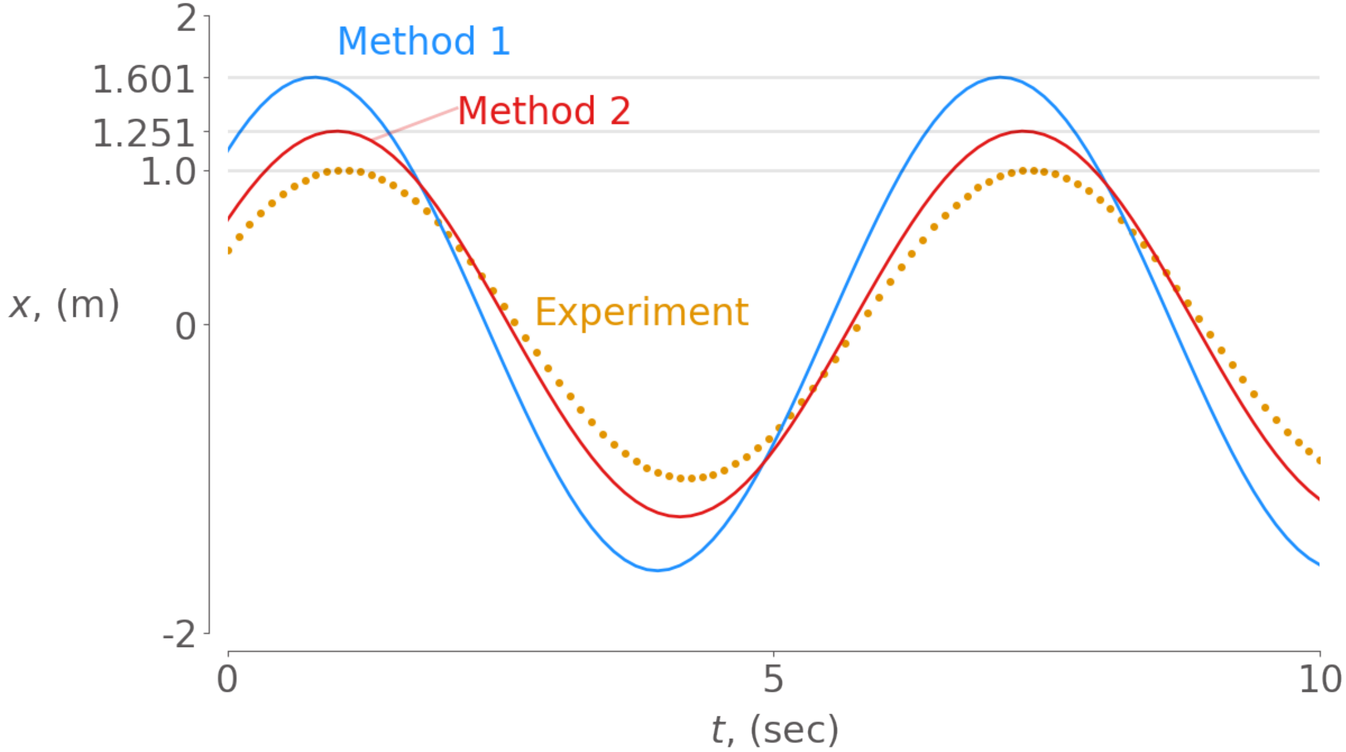
2. Did we do experiments for the whole curve?

3. Constantly switch back and forth from the legends to the curves?

5. Do we need so many ticks points? Noise!

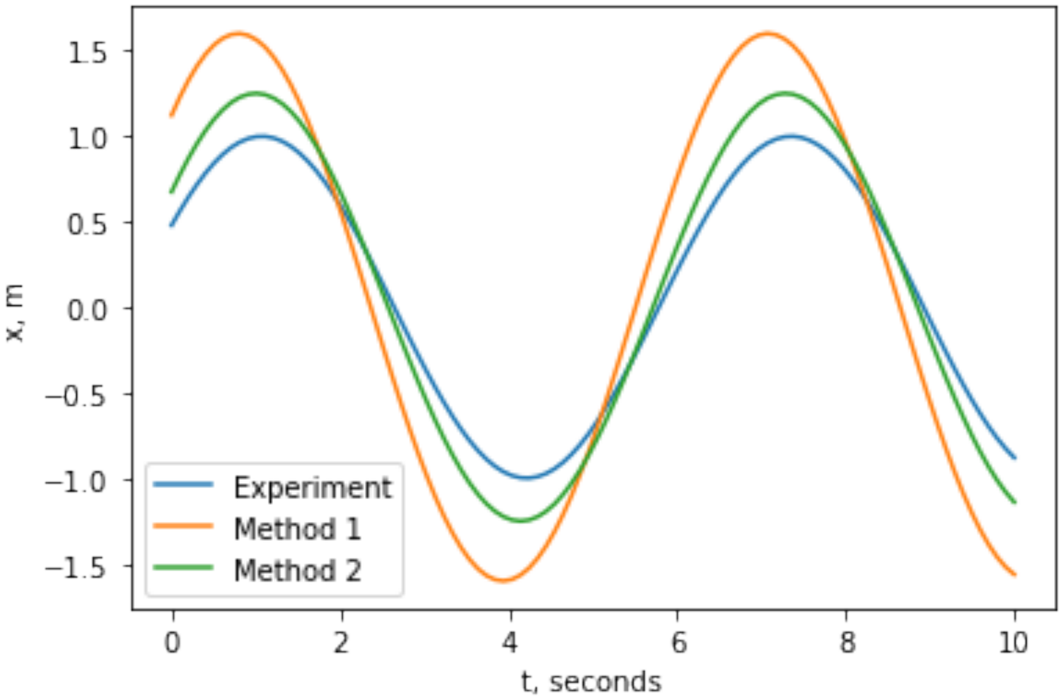
Improved figure





1. OK, 1 is better than 2, but by how much?
2. Did we do experiments for the whole curve?
3. Constantly switch back and forth
   from the legends to the curves?
4. Shall we rotate our head to read?
5. Do we need so many ticks points? Noise!
6. Math symbol, use Latex!
7. Do we need to box it? Noise!

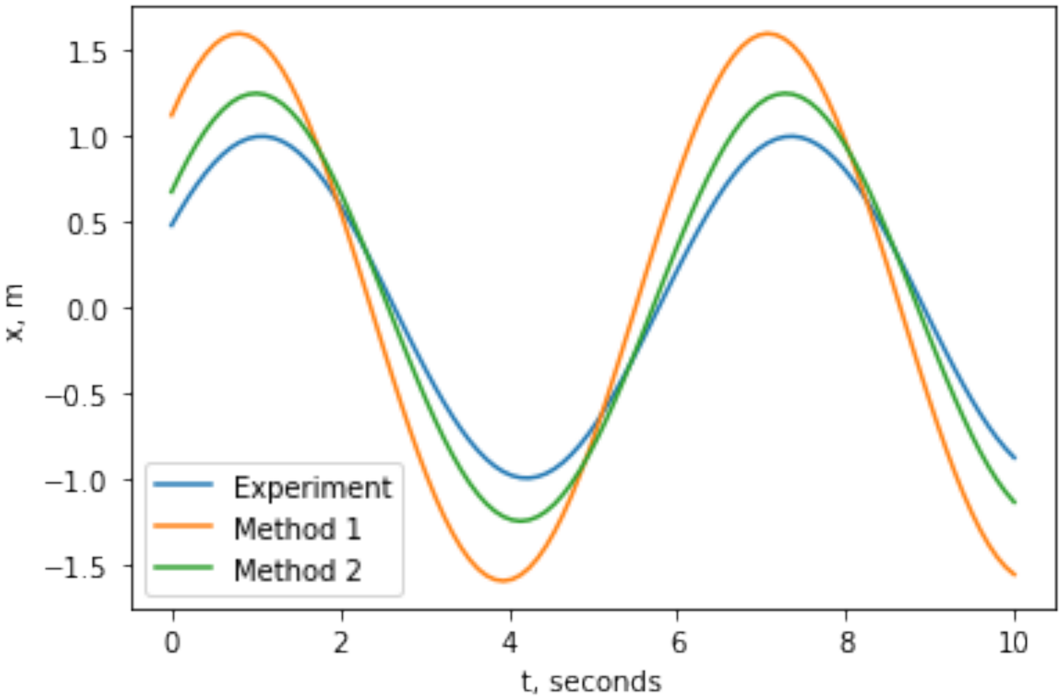
How to do it?



1. OK, 1 is better than 2, but by how much?
2. Did we do experiments for the whole curve?
3. Constantly switch back and forth
   from the legends to the curves?
4. Shall we rotate our head to read?
5. Do we need so many ticks points? Noise!
6. Math symbol, use Latex!
7. Do we need to box it? Noise!

```
1  import niceplots
2
3  fig, ax = plt.subplots()
4
5  colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
6  print("colors", colors)
7
8  # Applying niceplots standard
9  # Fix (I7) and in general makes the figure looks better
10 niceplots.setRCParams()
11 niceplots.All()
12 # If not using niceplots
13 # ax.spines['top'].set_visible(False)
14 # ax.spines['right'].set_visible(False)
15
16 # Fix (I2)
17 ax.plot(x, yref, 'o')
18 ax.plot(x, y1)
19 ax.plot(x, y2)
20
21 # Fix (I4, I6)
22 ax.set_ylabel(r'$x$, (m)', rotation = 0)
23 ax.yaxis.set_label_coords(-.15, .5)
24 ax.set_xlabel(r'$t$, (sec)')
25
```

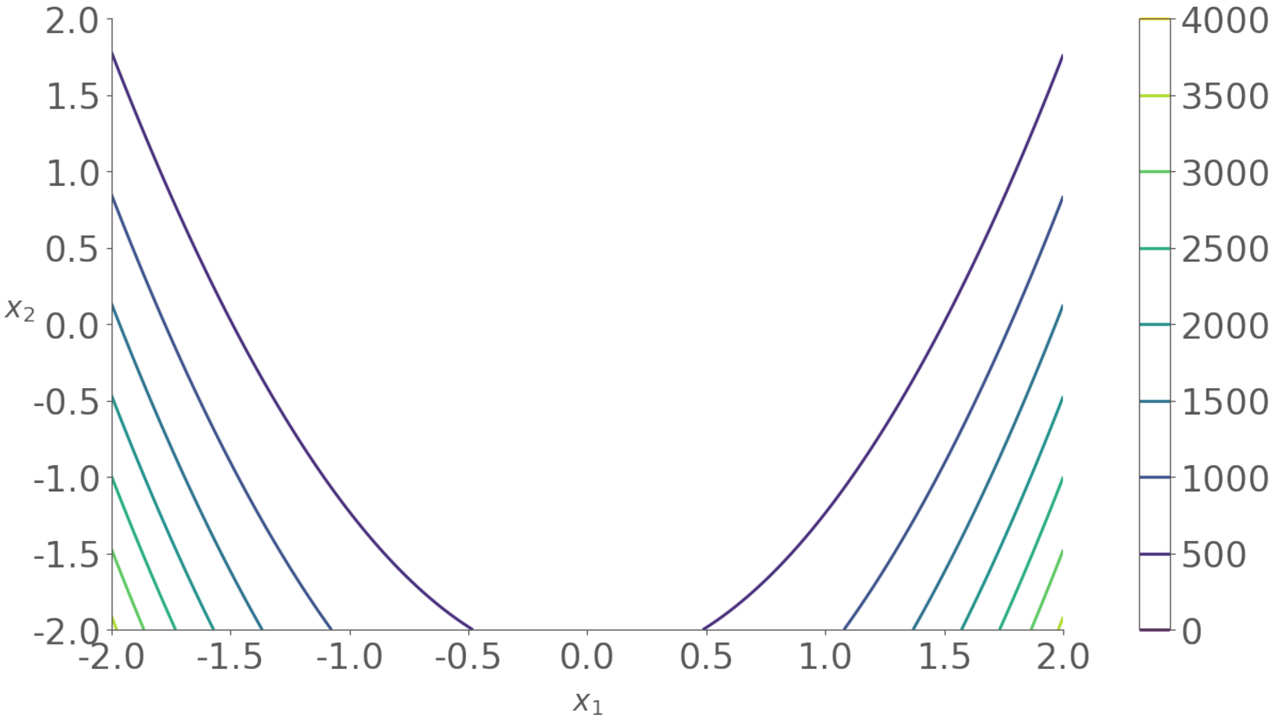
How to do it?



1. OK, 1 is better than 2, but by how much?
2. Did we do experiments for the whole curve?
3. Constantly switch back and forth
   from the legends to the curves?
4. Shall we rotate our head to read?
5. Do we need so many ticks points? Noise!
6. Math symbol, use Latex!
7. Do we need to box it? Noise!

```
26  # Fix (I3)
27  ax.text(2.8, 0, "Experiment", color = colors[0])
28  ax.text(1, 1.75, "Method 1", color = colors[1])
29  ax.text(2.1, 1.3, "Method 2", color = colors[2])
30  ax.plot([x[13], 2.1], [y2[13], 1.4], '-', color = colors[2], alpha = 0.3)
31
32  # Fix (I1)
33  ind_max_ref = np.argmax(yref)
34  ind_max_1 = np.argmax(y1)
35  ind_max_2 = np.argmax(y2)
36  ax.plot([0, 10], [yref[ind_max_ref], yref[ind_max_ref]], color = 'k', alpha =
37  ax.plot([0, 10], [y1[ind_max_1], y1[ind_max_1]], color = 'k', alpha = 0.1)
38  ax.plot([0, 10], [y2[ind_max_2], y2[ind_max_2]], color = 'k', alpha = 0.1)
39
40  # Fix (I5)
41  ax.set_xticks([0, 5, 10], [0, 5, 10])
42  ax.set_yticks([-2, 0, yref[ind_max_ref], y2[ind_max_2], y1[ind_max_1], 2],
43  [-2, 0, float('%.3g' % yref[ind_max_ref]), float('%.4g' % y2[ind_max_2]),
44  float('%.4g' % y1[ind_max_1]), 2])
45
46  # Fix (I8)
47  # plt.savefig('../curve.pdf',bbox_inches='tight')
```

Case study 2: Contour plot

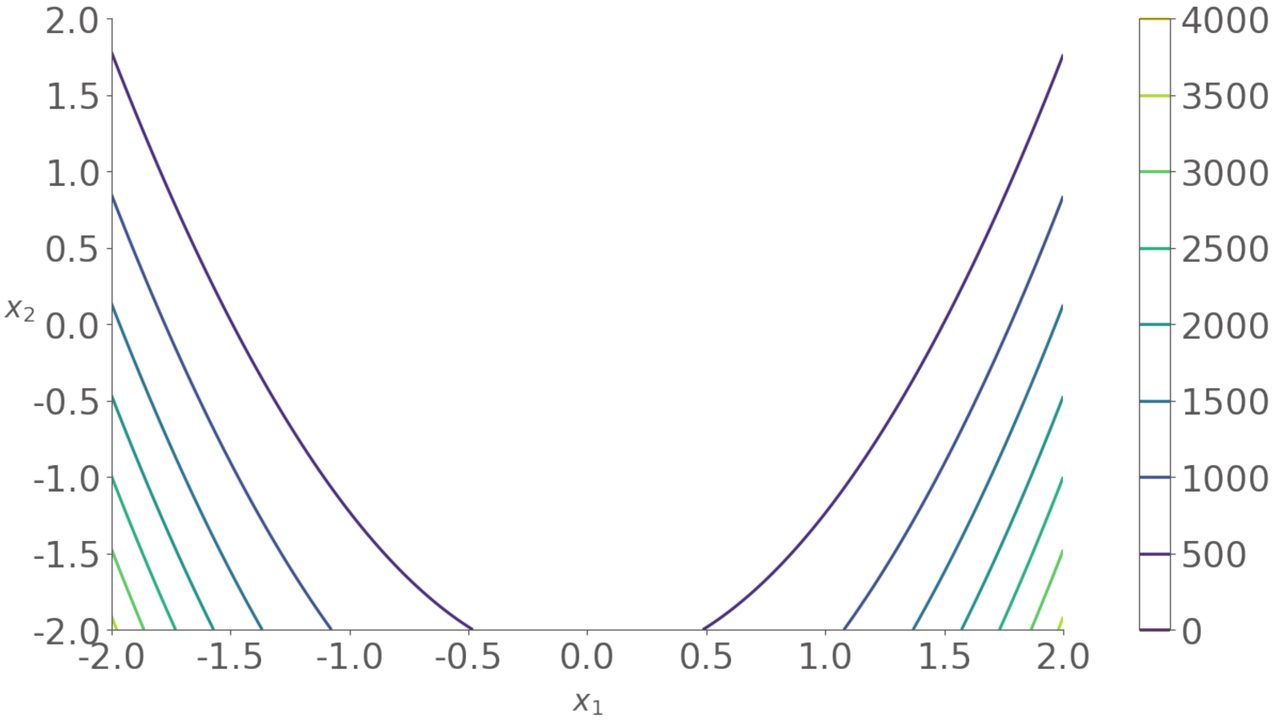


Background:
Rosenblock plot widely used in optimization algorithm performance test. It contains multi-scaled data.

Message:
?

```python
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import ticker, cm

# Create the frame
fig, ax = plt.subplots()
X_arr, Y_arr = np.meshgrid(x_arr, y_arr)

# Plot
CP = ax.contour(X_arr, Y_arr, F_arr.T)

# Set axis
ax.set_xlabel(r'$x_1$', fontsize=20)
ax.set_ylabel(r'$x_2$', fontsize=20, rotation=0)

fig.colorbar(CP)

plt.show()
```

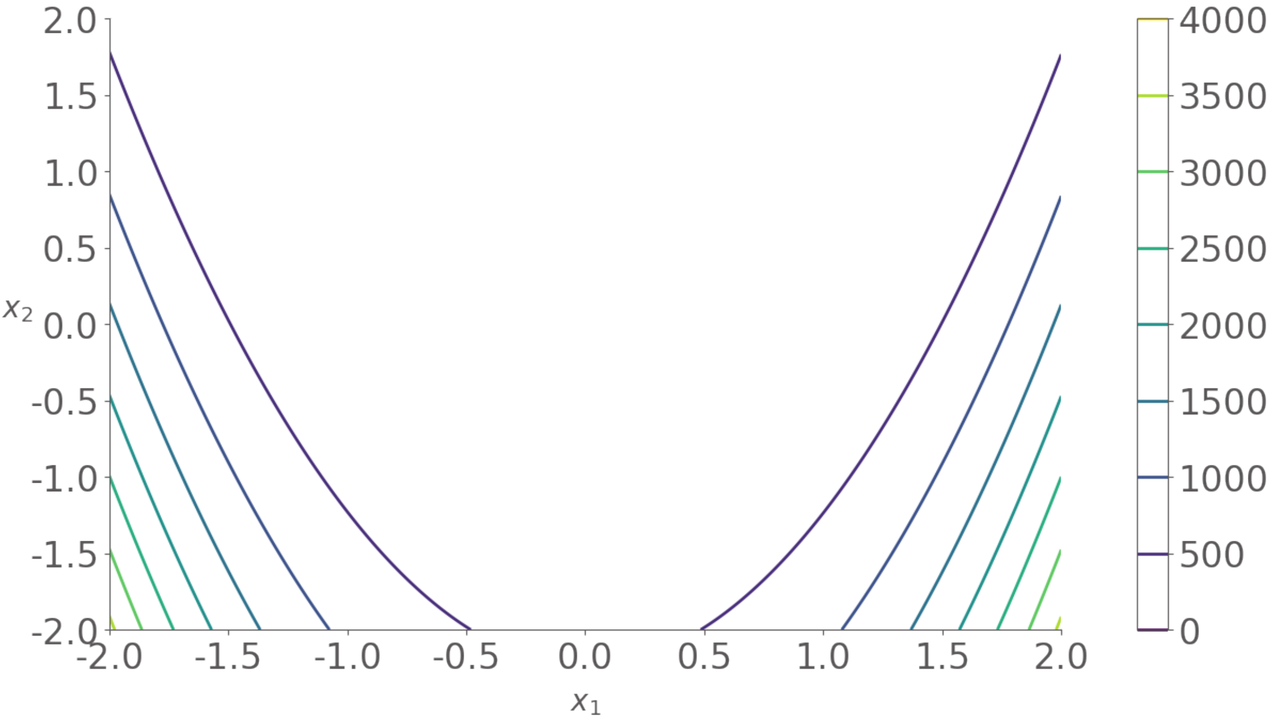
What is the main message?



Background:
Rosenblock plot widely used in optimization algorithm performance test. It contains multi-scaled data.

Message:
The optimal solution is at (1, 1). And the trend of the contour (in log).

What are the issues?

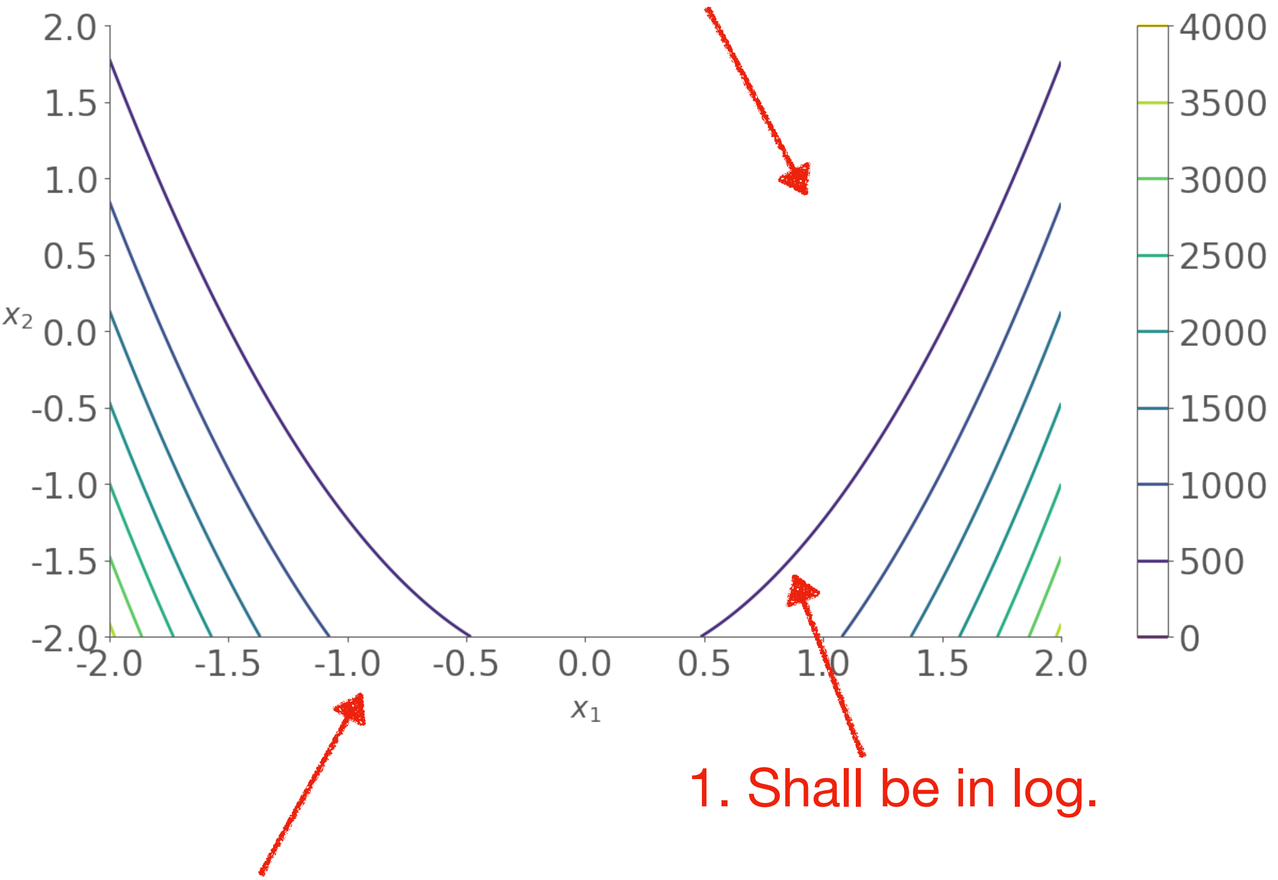


Algorithm 1.1 Figure improving algorithm

Input Message set, original figure
for message ∈ message set **do**
  if message unclear **then**
    Fix message.
  end if
end for
set = {labels, legends, boxline, words, lines, ticks, colorbar, . . .}
for feature ∈ set **do**
  if feature unclear **then**
    Fix feature.
  end if
end for
Return Improved figure

What are the issues?

3. Missing optimal solution.



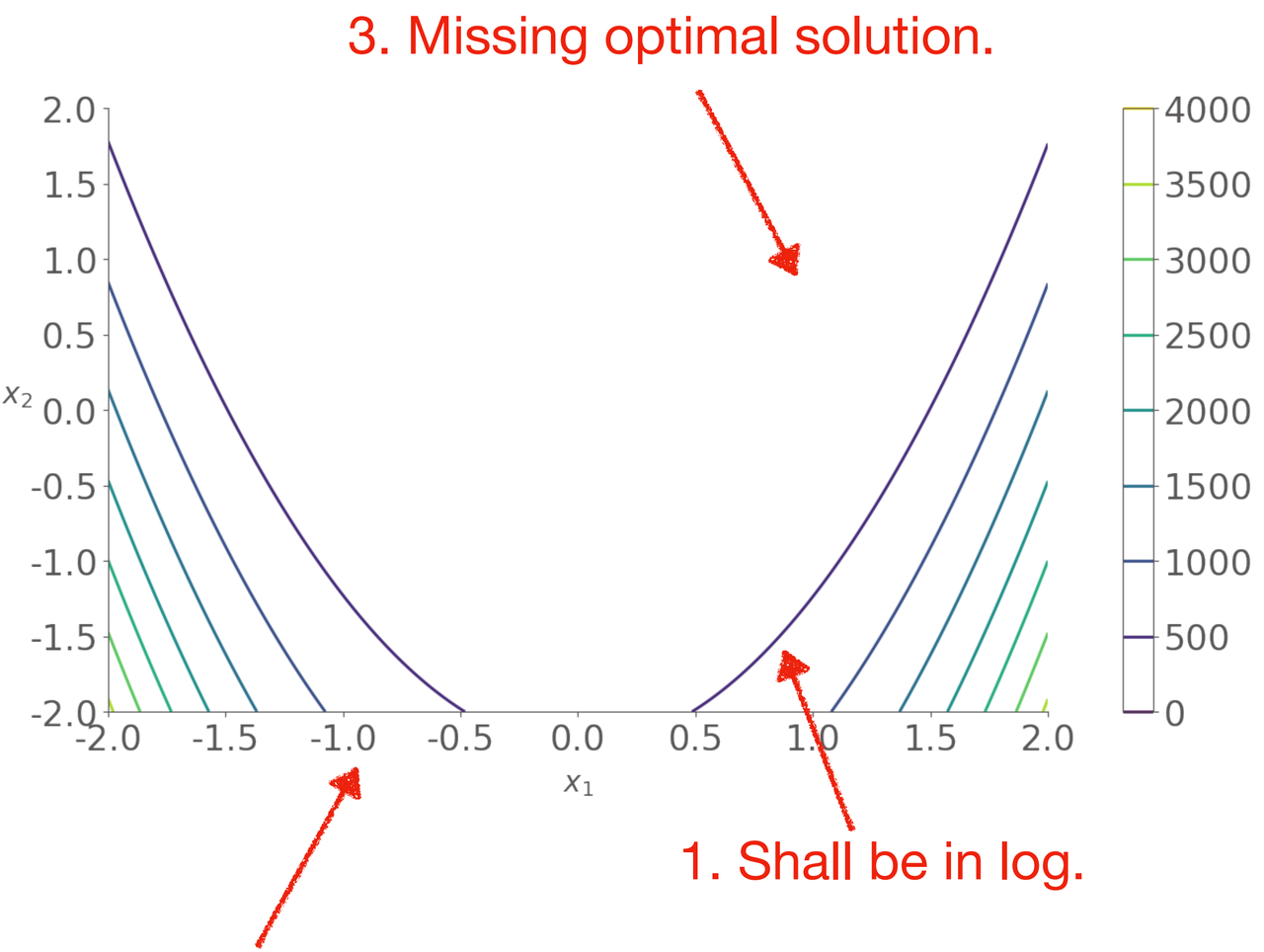
1. Shall be in log.

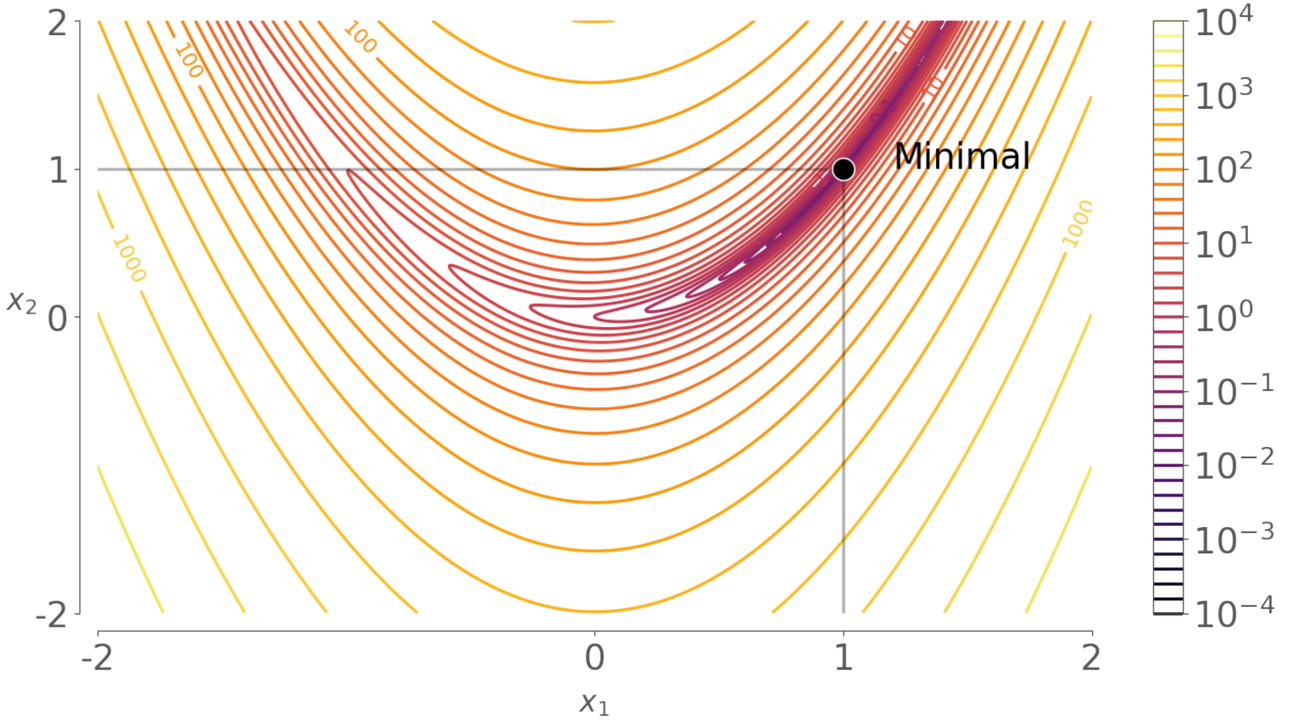
2. To many ticks.

Algorithm 1.1 Figure improving algorithm

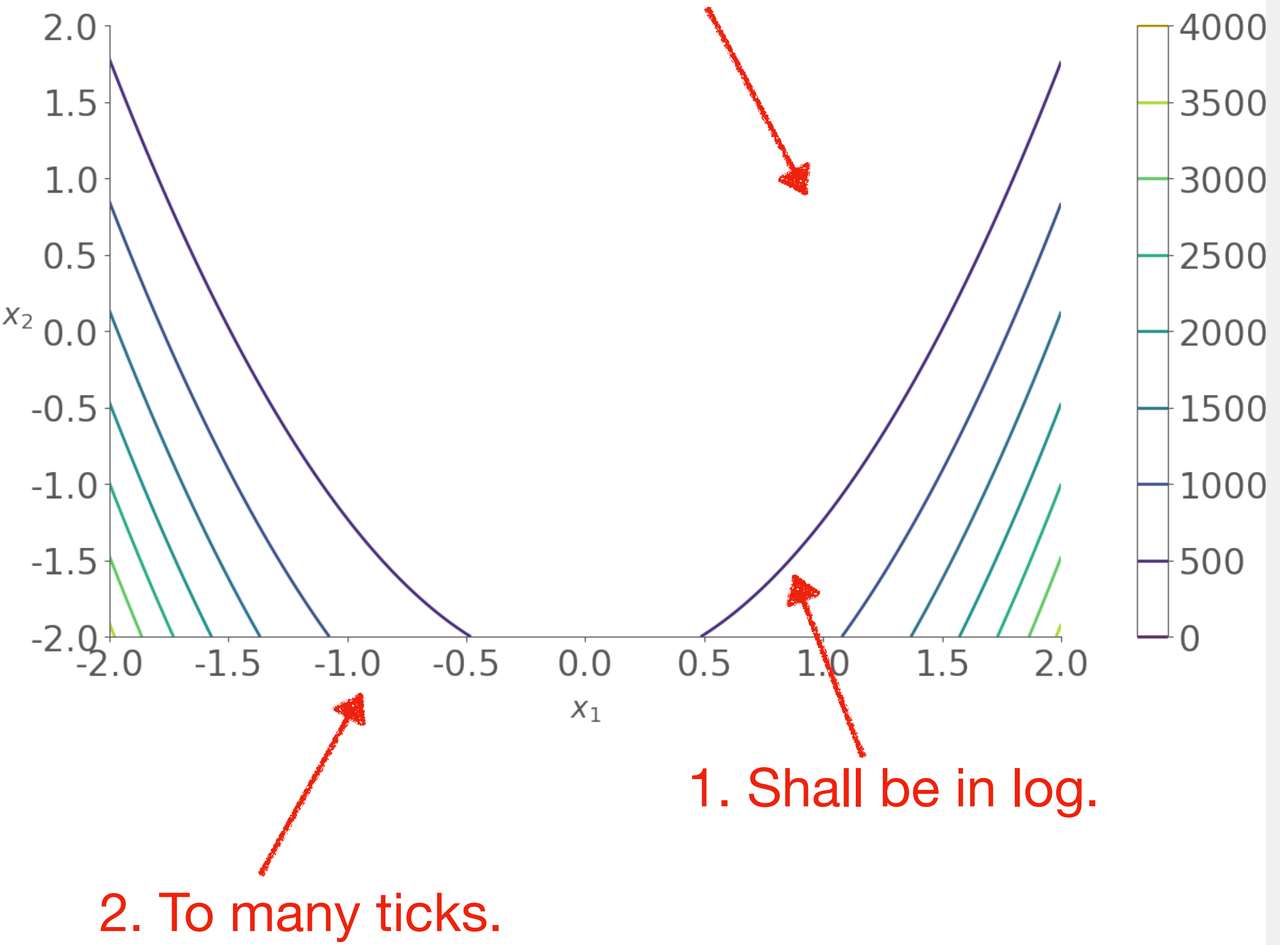
Input Message set, original figure
for message ∈ message set **do**
  if message unclear **then**
    Fix message.
  end if
end for
set = {labels, legends, boxline, words, lines, ticks, colorbar, . . .}
for feature ∈ set **do**
  if feature unclear **then**
    Fix feature.
  end if
end for
Return Improved figure

Improved figure



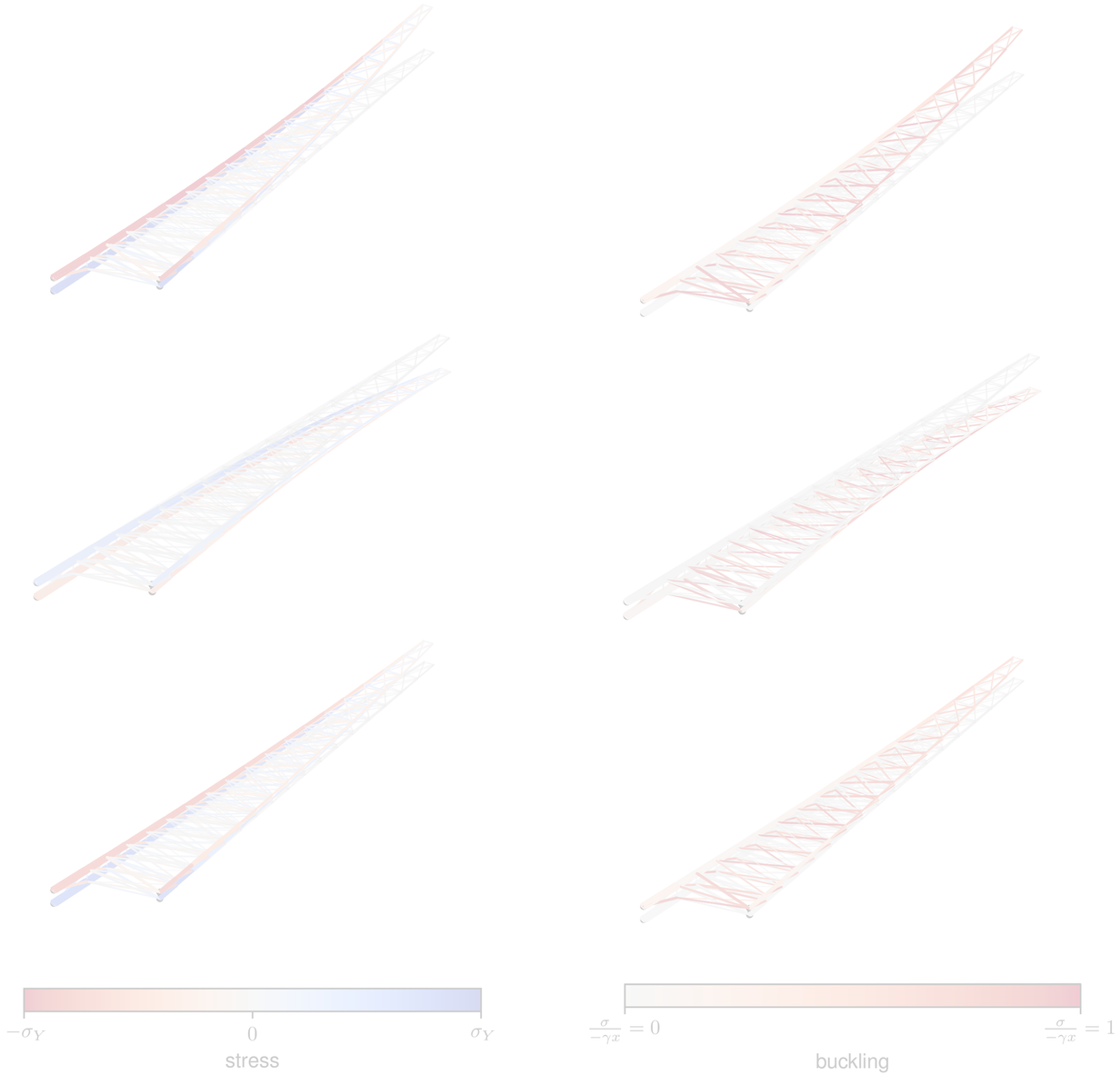


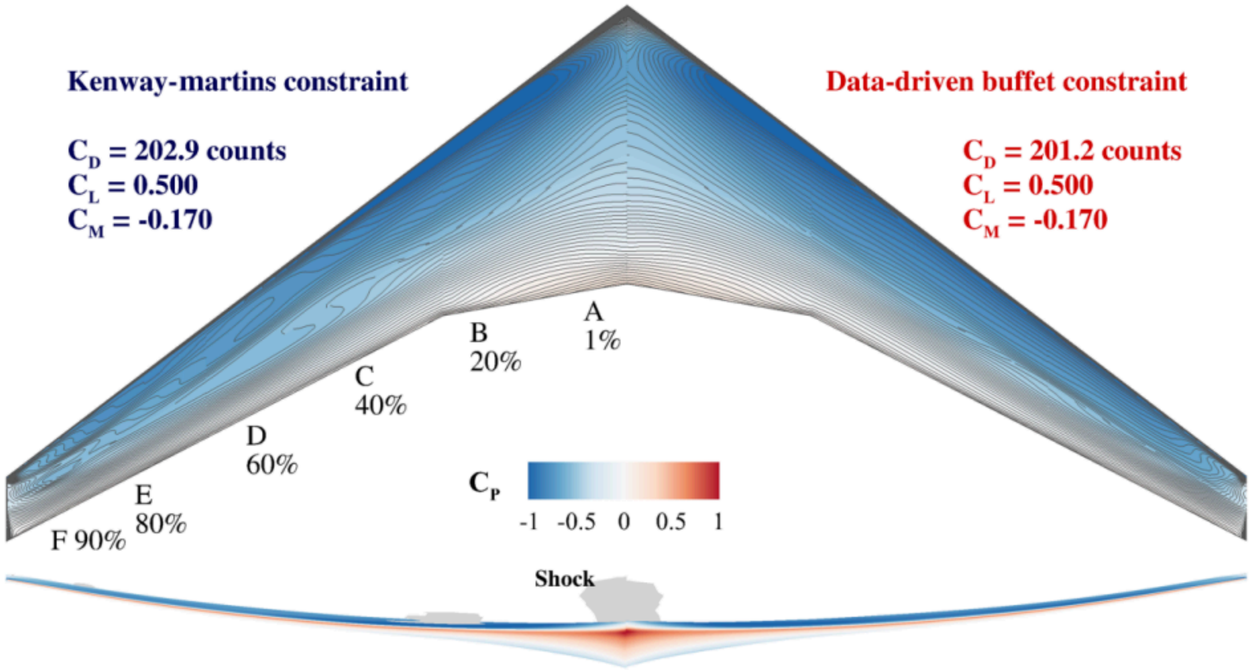
Case study 2: Contour plot

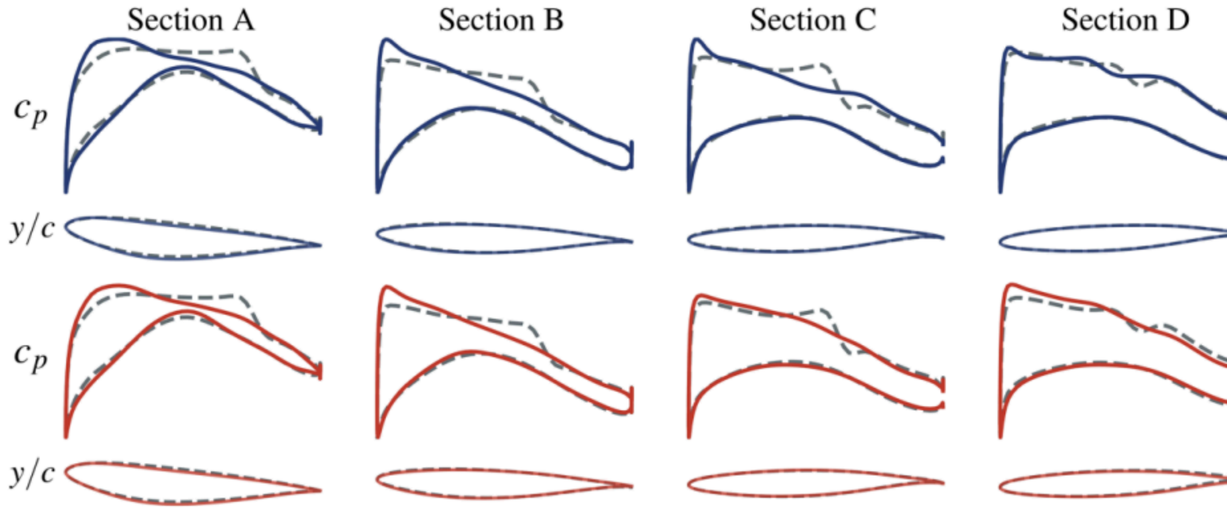



```python
import matplotlib
from matplotlib import ticker, cm
import niceplots


# Create the frame
fig, ax = plt.subplots()
X_arr, Y_arr = np.meshgrid(x_arr, y_arr)

# Load niceplots
niceplots.setRCParams()
niceplots.All()

# Fix (I1)
# Define the contour levels
levels = pow(10, np.linspace(-4, 4, 41))
# The default colormap actually is okay.
CP = ax.contour(X_arr, Y_arr, F_arr.T, levels, locator=plt.LogLocator(), cmap="inferno")

# Add the label
ax.clabel(CP, levels = [10**-3, 10**-2, 10**-1, 1.0, 10.0, 100.0, 1000.0],  # label every tenth level
                inline=True, fmt = "%1.4g", fontsize=14)

# Set axis
ax.set_xlabel(r'$x_1$', fontsize=20)
ax.set_ylabel(r'$x_2$', fontsize=20, rotation=0)

# Format the colorbar
def fmt(x, pos):
    a, b = '{:.2e}'.format(x).split('e')
    b = int(b)
    return r'$10^{{{}}}$'.format(b)

fig.colorbar(CP, format=ticker.FuncFormatter(fmt))

# Redefine the axis ticks (I2)
ax.set_xticks([-2, 0, 1, 2], [-2, 0, 1, 2])
ax.set_yticks([-2, 0, 1, 2], [-2, 0, 1, 2])

# Auxcillary lines (I3)
ax.plot([-2, 1], [1, 1], 'k', alpha = 0.3)
ax.plot([1, 1], [-2, 1], 'k', alpha = 0.3)
ax.plot([1], [1], "ko", markersize = 15)

ax.text(1.2, 1, "Minimal", color = 'k')

plt.show()
```

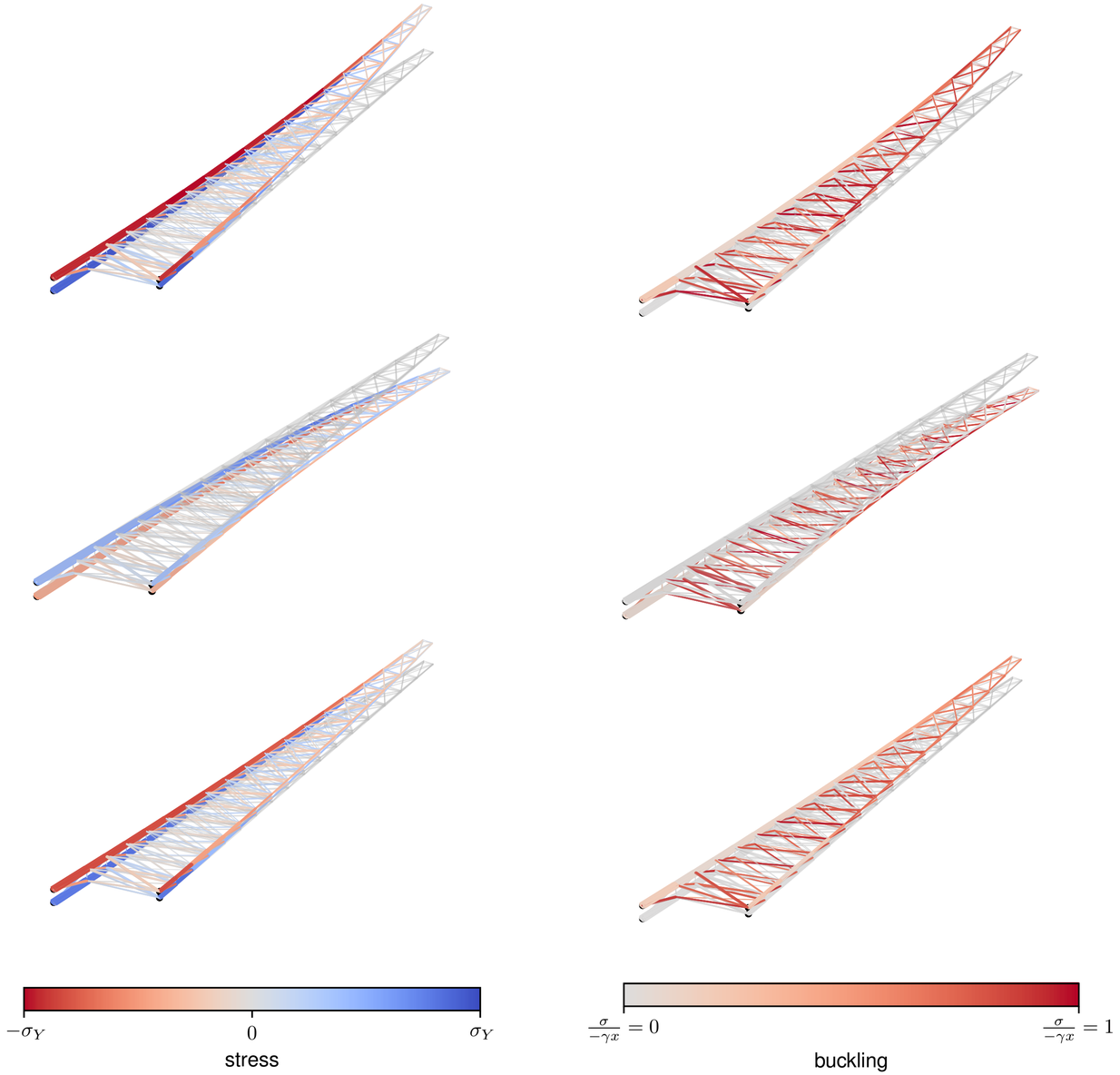
Sequential color scheme for sequential data, divergent color scheme for signed data

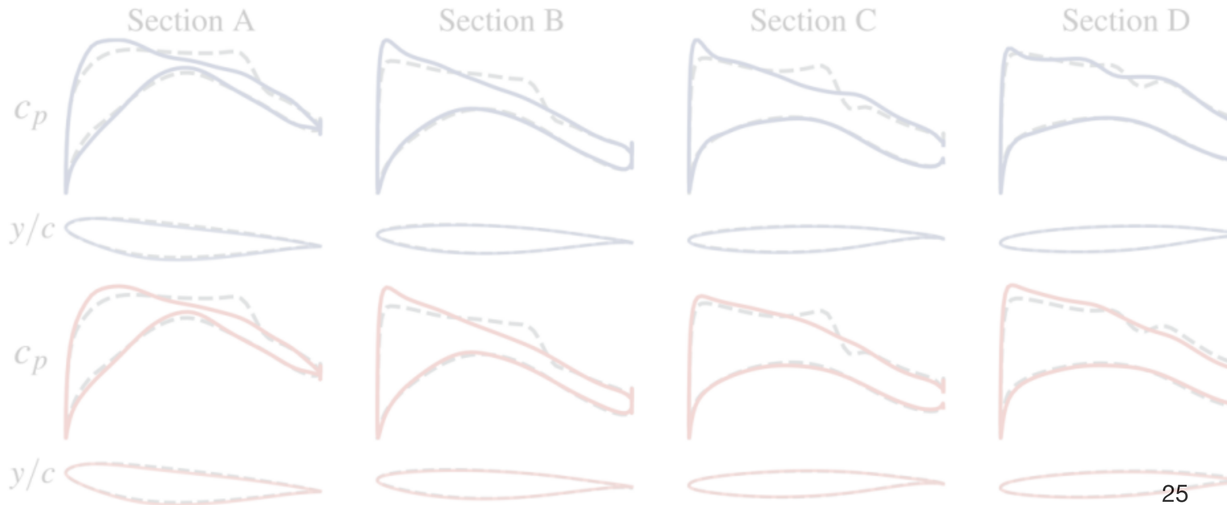




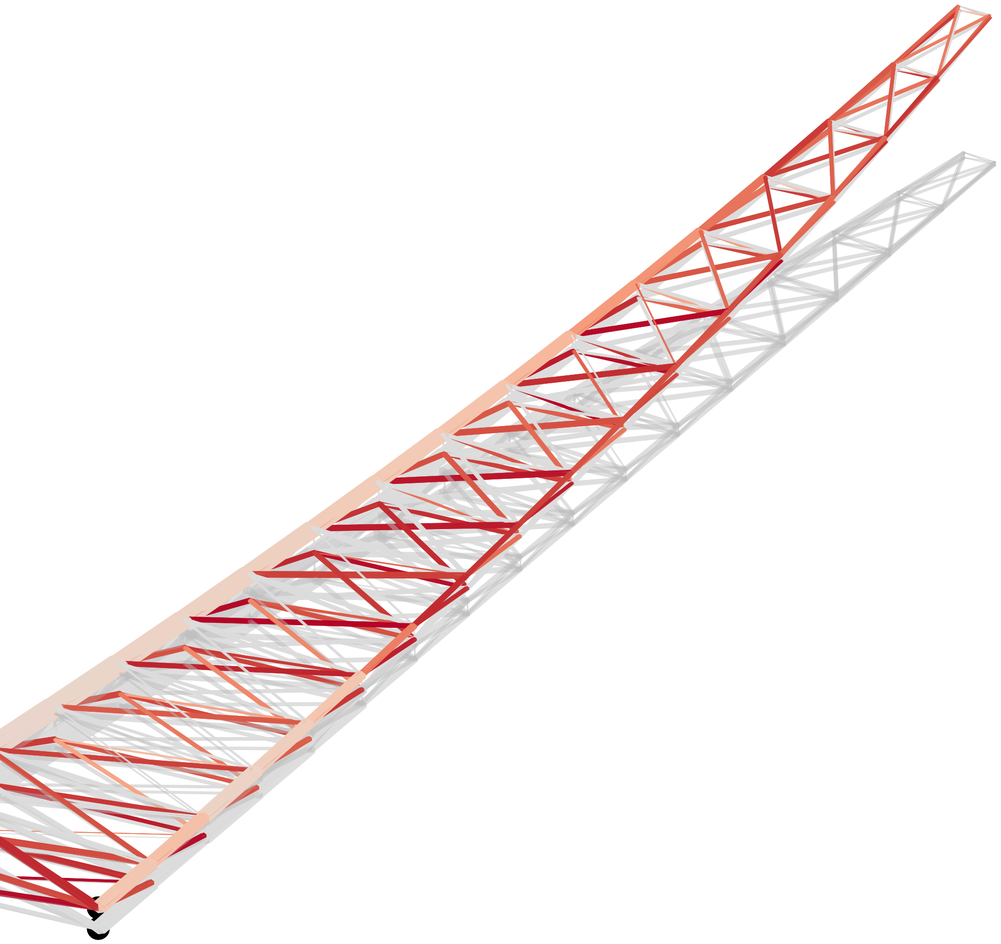


Sequential color scheme for sequential data, divergent color scheme for signed data





Always use a vector format (e.g., .pdf, .svg)



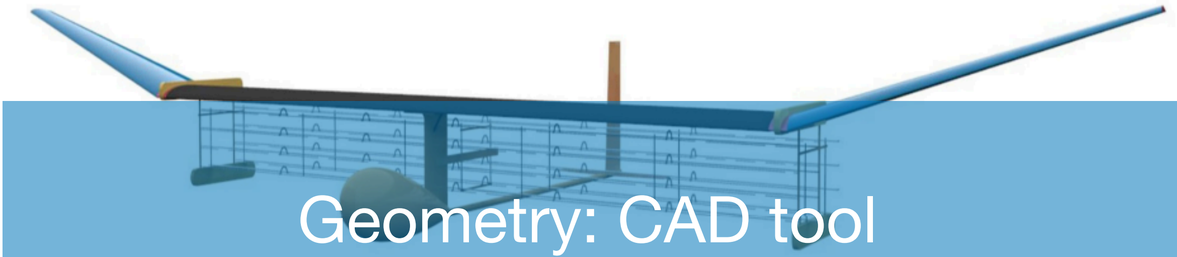
PDF

PNG

Module 2: Try it yourself

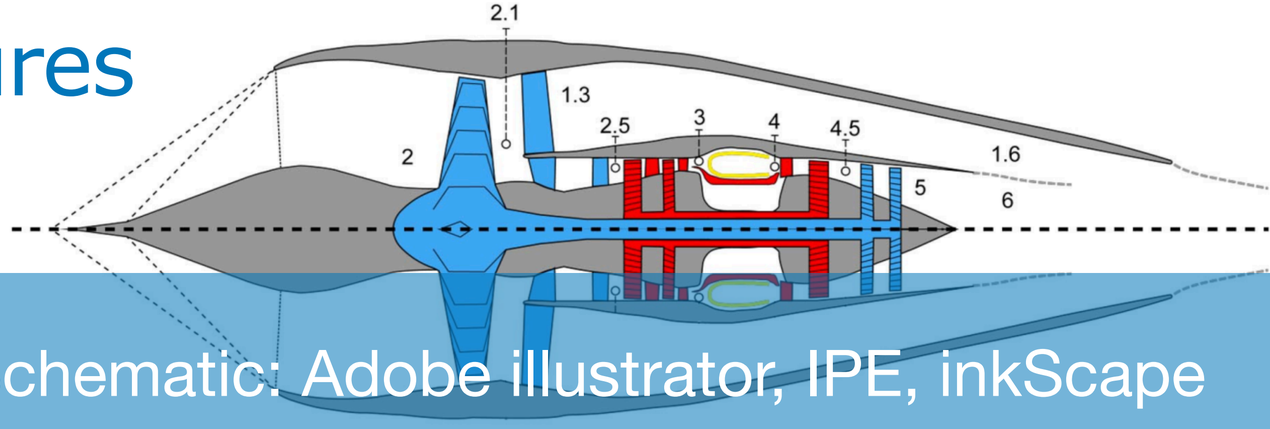
Module 3: Conclusion

Different tools for different figures



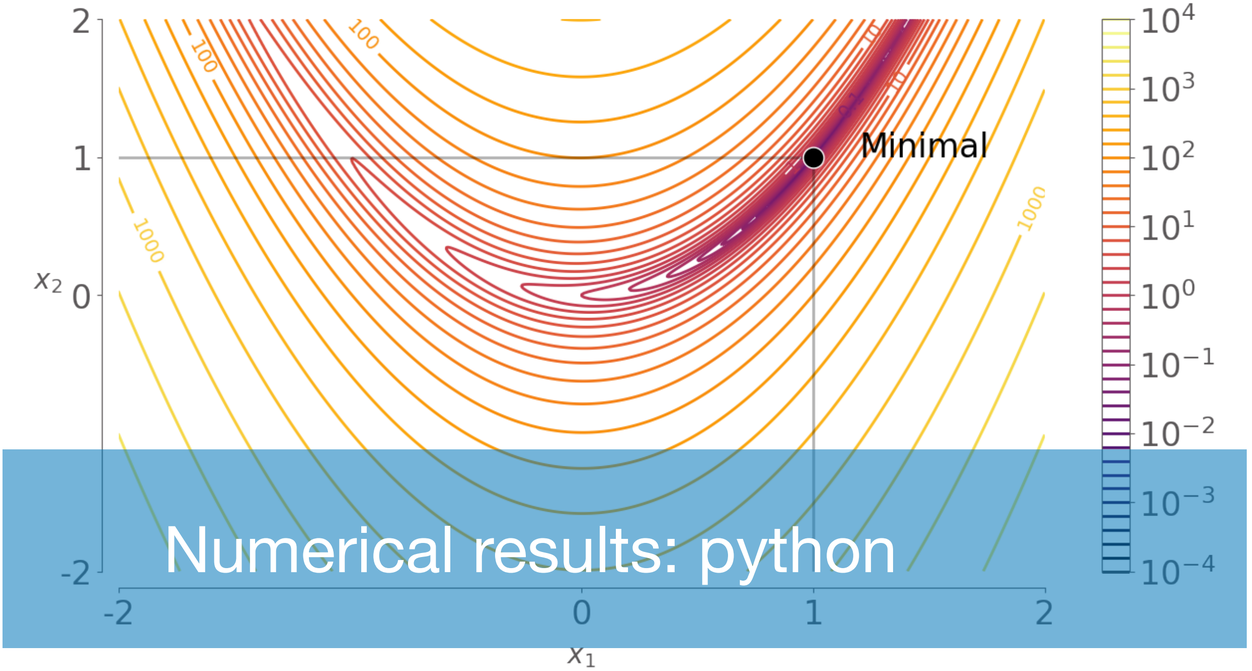
Geometry: CAD tool

Xu et al. Nature 2018

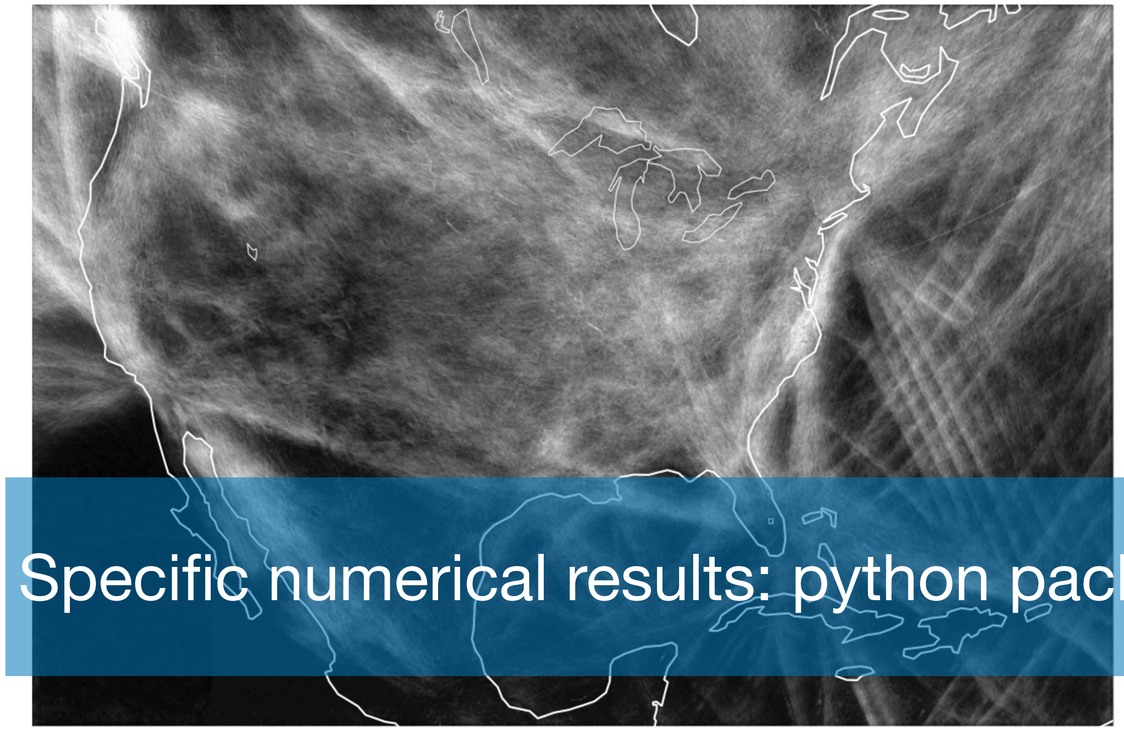


Schematic: Adobe illustrator, IPE, inkScape

Voet et al. Scitech 2021



Numerical results: python



Specific numerical results: python packages

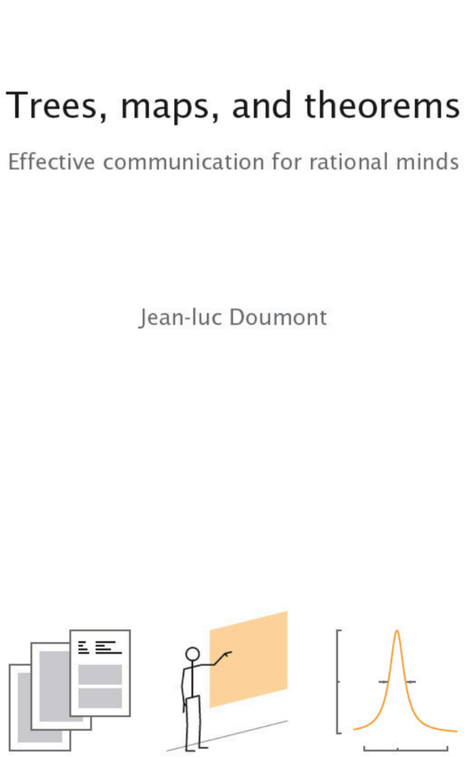
Meijer et al. Environmental Research Letters 2022

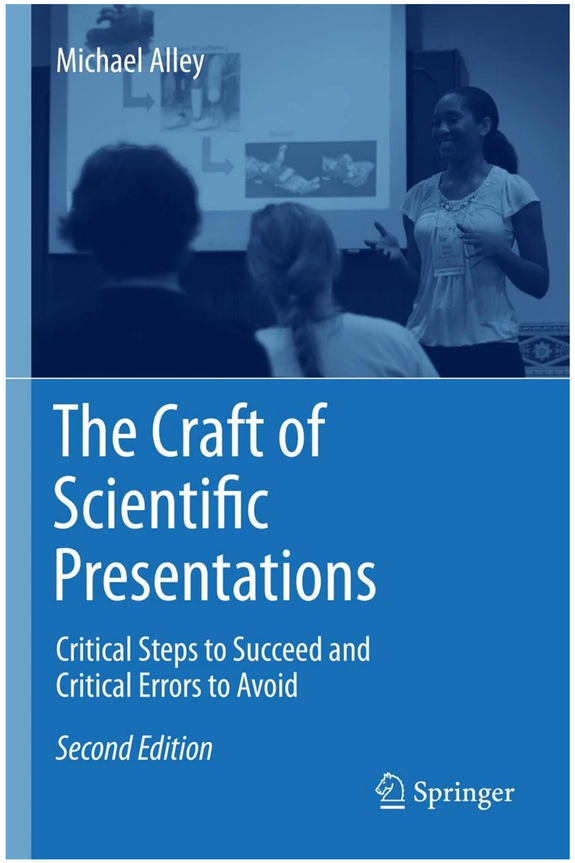
Suggestions

- Do it once and do it well: Create a figure repo for various types of figures, (e.g., curve, contour, etc.)
- Do not reinvent wheels: Start from some existing figures (including your repo) and improvise.

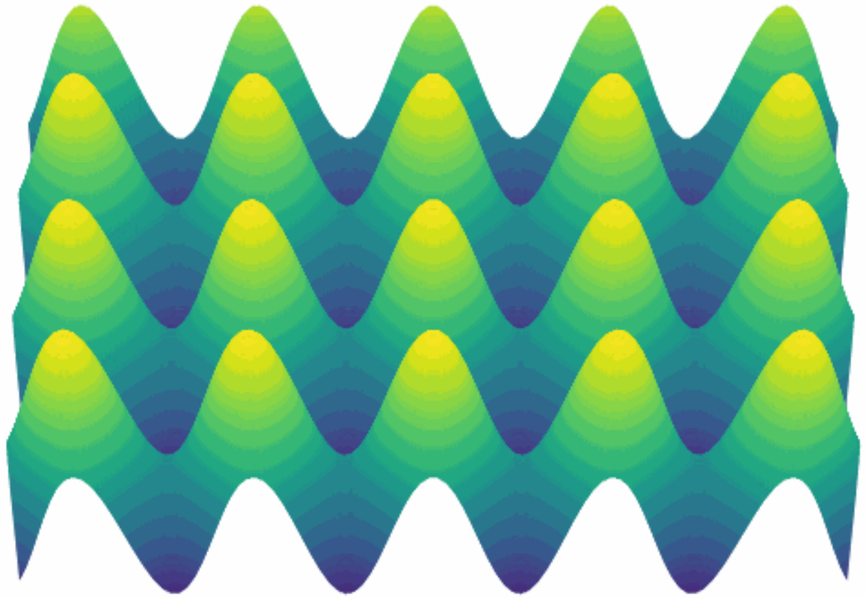
Recommended readings

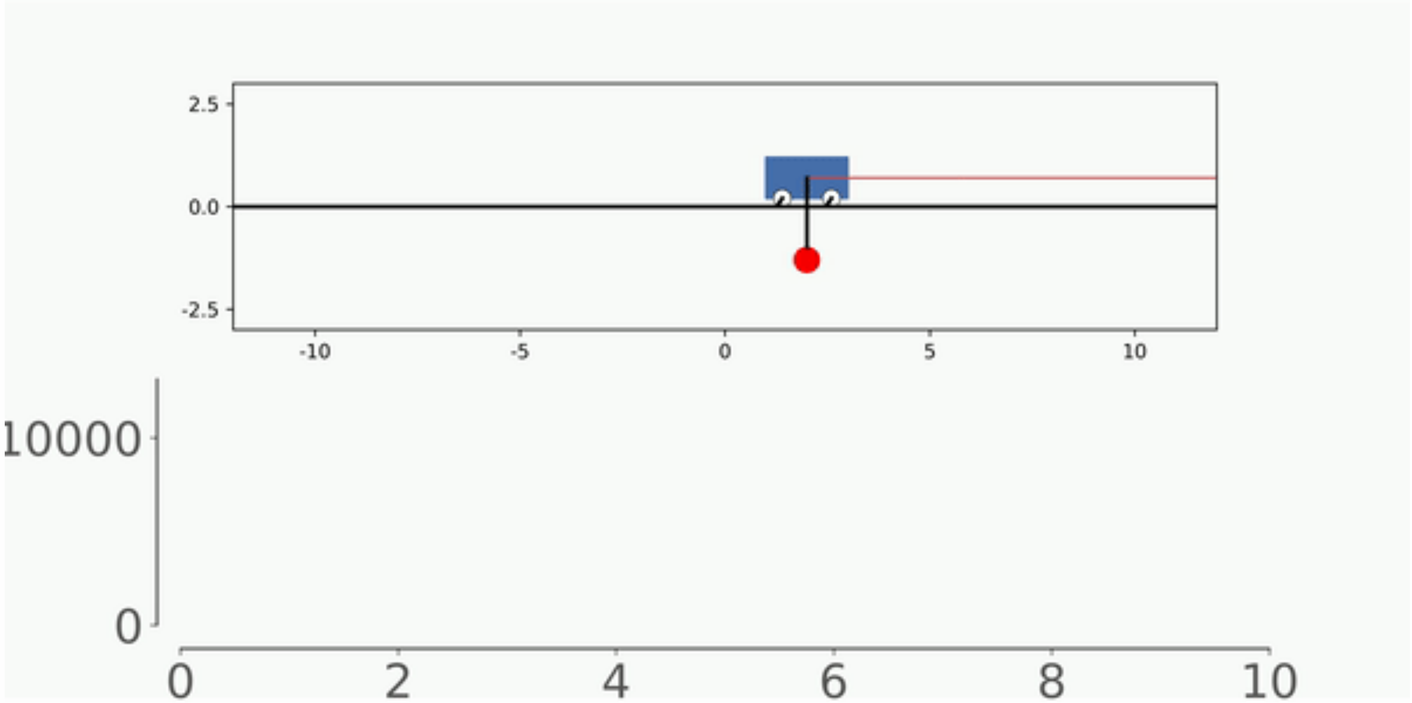
- J. Dumont, *Trees, maps, and theorems*
- M. Alley, *The Craft of Scientific Presentations: Critical Steps to Succeed and Critical Errors to Avoid*



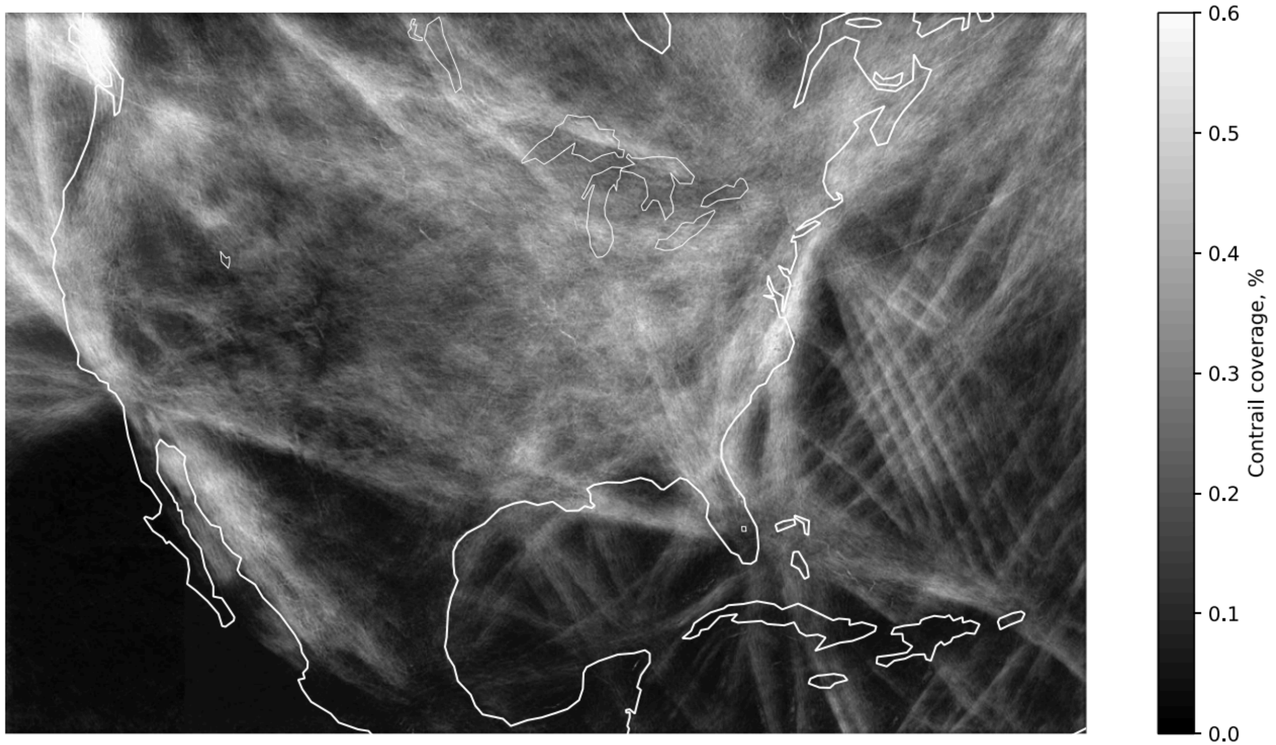


matplotlib can be used to generate video, if possible put some video in your presentation





Put the main message in the title, and waste no space in the slide



Put the main message in the title, and waste no space in the slide

Put the main message in the title, and waste no space in the slide

A picture is worth a thousand words!
Go forth and plot more!