

INTRODUCTION TO STREAMLIT

There are lots of different ways of creating Python apps that others can use such as Tkinter, PyQt etc. This document focusses on the use of **Streamlit** (streamlit.io) as a convenient way of deploying and sharing your Python apps over the web. The streamlit approach is relatively simple and re-runs the program again every time any change is made to the app. The price for this simplicity is that apps that use a lot of processing can take a long time to refresh, however Streamlit does provide various options for increasing the efficiency, such as caching large datasets the first time they are used (see examples on the **caching** directory) or using forms, which will only run a portion of your code that is associated with a **button** widget. In practice as long as your program is not too compute-intensive this is not likely to be a major problem.

To use streamlit you will need to install the streamlit library (e.g. using **pip install streamlit** although other installers such as **uv** <https://docs.astral.sh/uv/> are becoming very popular) and then accessing the library in your Python program, which is usually done via **import streamlit as st** at the top of your program. To run a program called <filename.py> simply type **streamlit run <filename.py>** on the directory where it is and it will appear in a web browser. Since streamlit runs the program when anything changes you will need a way of storing information between these sessions and you can do this using the **st.session_state** dictionary. There is a **state** subdirectory within the **examples** directory which has some simple examples to get you used to this idea. The more advanced examples all use **st.session_state** to store information between re-runs.

How to Get Started

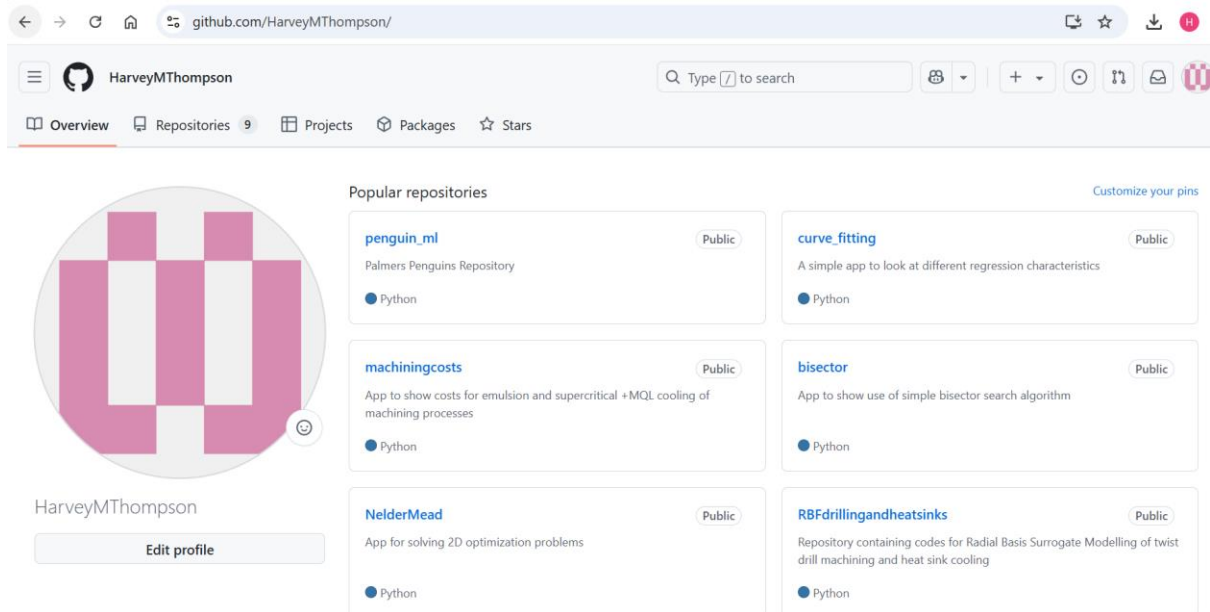
The first thing you will need to do once you have Python installed on your computer is to install the streamlit library and any other libraries that you need. In the examples below you can tell which libraries are being used in a specific program from the list of **import <library names>** you can see. The way to learn is to look at some examples, play around and modify them and use then search the web for solutions to any problems you have. There are a number of examples on the **examples** directory you can look at to get familiar with the ideas and how to use the various widgets. My advice would be to take a look at the following subdirectories of the **examples** directory in the first instance: **appdesign**, **caching**, **dataframes**, **forms**, **slider** and **state**. Also if you have not use any other Python graphics packages (there are lots of brilliant ones) I would stick to using the **matplotlib** library (matplotlib.org) at the beginning. If you are interested in using more interactive graphics there are simple examples in the **Altair** and **Plotly** subdirectories and in the **heatinganddrillingRBF** subdirectory.

You should be able to find what you need online but I also used a great book: **Streamlit for Data Science**, by Tyler Richards from Packt publishing which I found really helpful. I have also included the python programs used in that book on the **StreamlitForDataScience** directory – these are available on the book's **GitHub** repository. You will probably only need to look at these if you are interested in becoming more proficient in Streamlit.

Deploying Your Programs on the Web: Streamlit Community Cloud and GitHub

One of the other nice things about Streamlit is that it enables you to deploy your programs on the web relatively easily (and currently for free) by uploading your programs to **GitHub** (github.com) and accessing them via **Streamlit Community Cloud** (share.streamlit.io). You will need to create accounts on both Streamlit Community Cloud and Github. The first thing you will need to do is to create a repository on Github for your application. You might have to mess around a little bit to get used to how Github works but it should not take too long to be able to do what you need to do. On the **GitHub** directory associated with these notes you will see I have added four repositories I had created on

GitHub for the applications in the **bisector**, **curve-fitting**, **NelderMeadSimplex** and **heatanddrillingRBF** subdirectories of the **examples** directory. These can be found on my Github repository at <https://github.com/HarveyMThompson/>.



In your GitHub repository you will need to include a **requirements.txt** file which tells Streamlit Community Cloud which Python packages it needs to include when building your web app. You can install a useful little utility called **pipreqs** (e.g. via `pip install pipreqs`) which enables you to extract all the packages which are used on your directory by typing '`pipreqs .`'. This will create a requirements.txt file for your directory that you can put onto your GitHub repository. For example on the **curve_fitting** repository the requirements.txt file has:

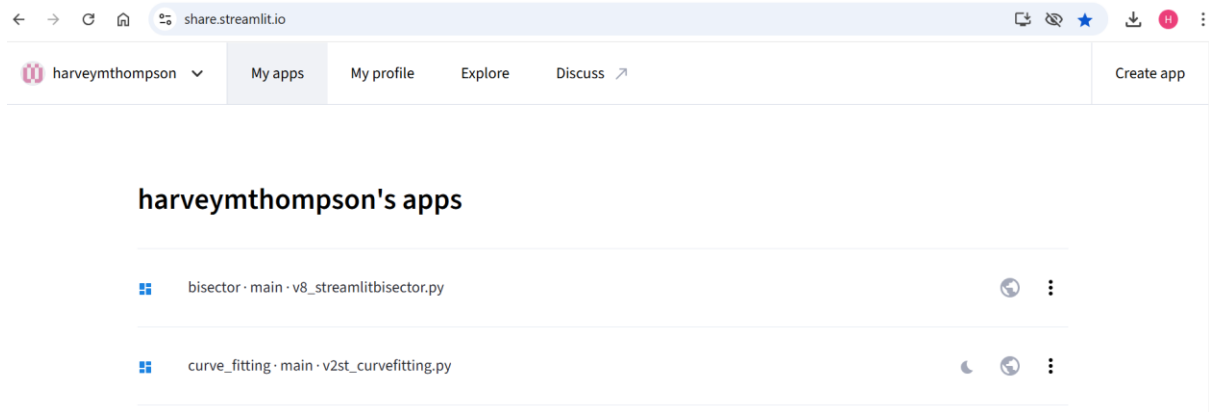
matplotlib

numpy

streamlit

which tells Streamlit Community Cloud that these are the packages it needs to build. One slightly strange feature is that running **pipreqs** on your directory will create a requirements.txt file with version numbers associated with each package. For some reason I found that this creates a problem with Streamlit Community Cloud so I have always just stripped out the versions associated with each package and this seems to work. I am not sure why this is the case! Also, if you want people to use your app (apart from close colleagues it is targeted at) you should also create a **LICENSE** file when you create your repository on GitHub. (Github gives you lots of options that you can create automatically). Without such a license file, strictly speaking, nobody would be able to use the code on your repository. I have not been very good at doing this on my own repositories, however on the **curve_fitting** repository I created an example of a Creative Commons license. Licensing is a big and complex area but licenses such as the MIT, Apache and Creative Commons (CC) ones can give users a lot of freedom to use and modify the codes if they download them from GitHub.

As mentioned above, you will also have to create an account on and use **Streamlit Community Cloud** (share.streamlit.io). This will enable you to create web apps by pointing towards the appropriate main driving program in your GitHub repository of choice. Here is a screenshot of some of my apps on SCC:



To create an app, click on 'Create app' at the top right of the screen and follow the instructions to create your app. It gives you a few options to tailor the names etc. to something that makes more sense to you than the default options. Note that once the app is created you will be able to see it via your computer or phone by tapping on the web link that is created. Sending other people this link will enable them to use the app also. Note that the app will only stay live for a few days so if you want to activate it again so people can use it, just log back into SCC and activate it again by clicking on the app name. For example to activate the bisector app I just need to click on the filename given above. As with GitHub it might take a bit of messing round before you are comfortable with SCC but with a bit of patience you should have your app up and running in no time at all.

In addition to simply looking at the codes on the directories mentioned above, the codes and examples of what you will see when you run them in streamlit are given below.

Happy programming!

Harvey Thompson, School of Mechanical Engineering, University of Leeds, February 2025

CODE EXAMPLES

Altair Directory: has simple examples using the Altair graphics library, which provides much more interactive graphics than matplotlib ('pip install altair', see altair-viz.github.io). Codes read in a number of packages (e.g. numpy, pandas, seaborn) which will need to be installed too.

firstaltair.py

```
import altair as alt

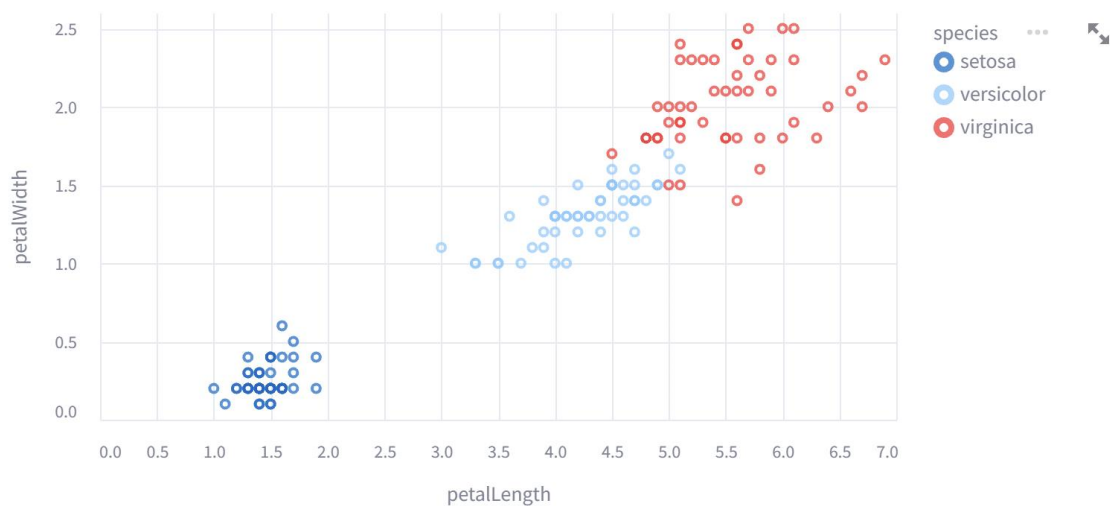
import streamlit as st

from vega_datasets import data

iris = data.iris()

alt_chart = (
    alt.Chart(iris).mark_point().encode(
        x='petalLength',
        y='petalWidth',
        color='species',
    )
    .interactive()
)

st.altair_chart(alt_chart, use_container_width=True)
```



second2dAltair.py

```
import streamlit as st

import pandas as pd

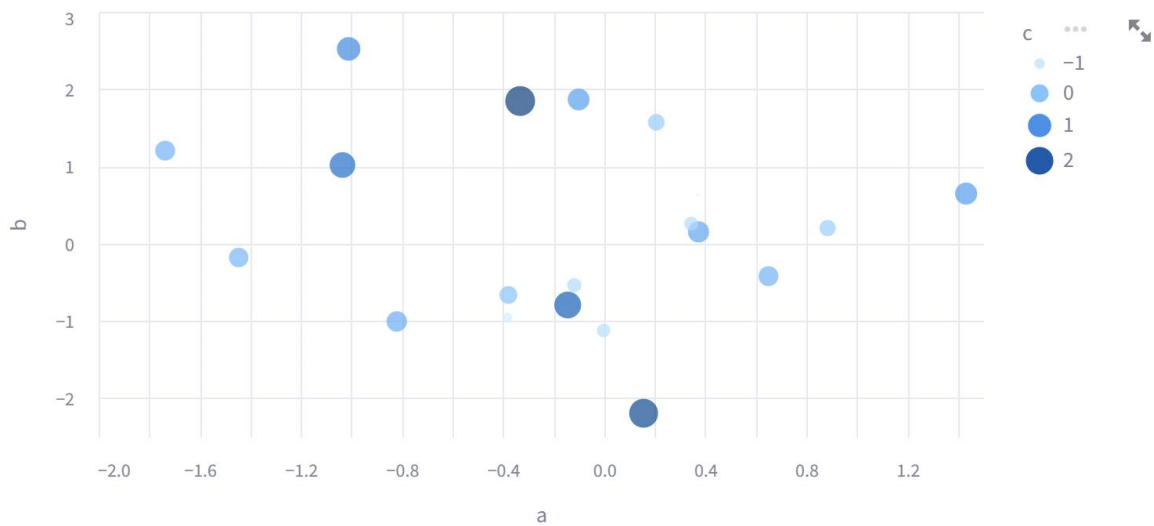
import numpy as np
```

```
import altair as alt
```

```
chart_data = pd.DataFrame(np.random.randn(20, 3), columns=["a", "b", "c"])
```

```
c = (
    alt.Chart(chart_data)
    .mark_circle()
    .encode(x="a", y="b", size="c", color="c", tooltip=["a", "b", "c"])
)
```

```
st.altair_chart(c, use_container_width=True)
```



```
# third2Daltair.py
```

```
import numpy as np
```

```
import pandas as pd
```

```
import altair as alt
```

```
import streamlit as st
```

```
ad_type = ["SEO", "Email", "PPC", "PR", "Direct Mail", "OMB"]
```

```
roi = [68.9, 56.7, 52.4, 48.5, 37.4, 19.9]
```

```
data = {"adtype":ad_type, "roi":roi}
```

```
df = pd.DataFrame(data)
```

```
df
```

```
c = (
```

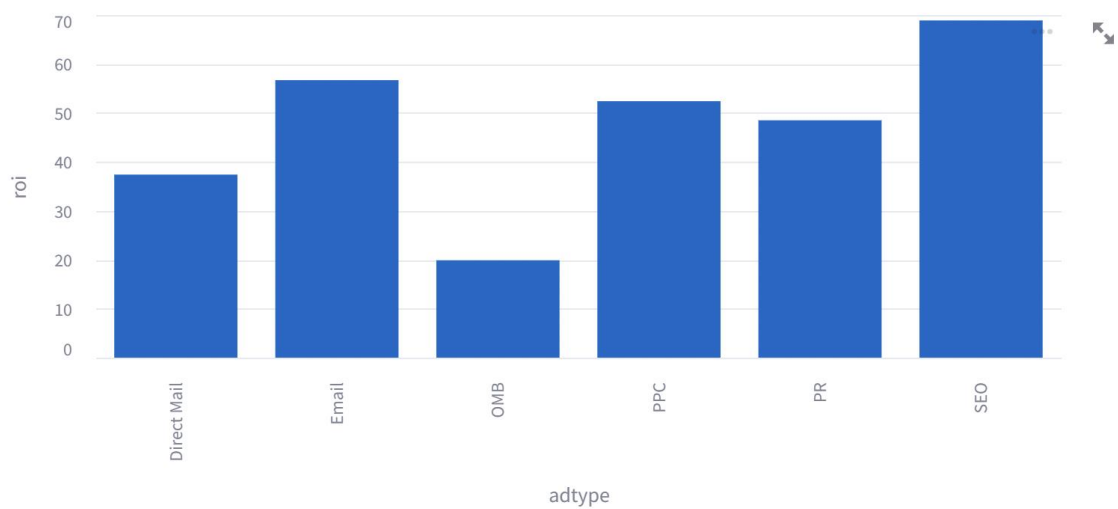
```

alt.Chart(df).mark_bar().encode(
    x = "adtype:O",
    y = "roi:Q",
)
)

st.altair_chart(c, use_container_width=True)

```

	adtype	roi
0	SEO	68.9
1	Email	56.7
2	PPC	52.4
3	PR	48.5
4	Direct Mail	37.4
5	OMB	19.9



fourth2Daltair.py

```

import numpy as np
import pandas as pd
import altair as alt
import streamlit as st

ad_type = ["SEO", "Email", "PPC", "PR", "Direct Mail", "OMB"]
roi = [68.9, 56.7, 52.4, 48.5, 37.4, 19.9]

data = {"adtype":ad_type, "roi":roi}
df = pd.DataFrame(data)

```

```
df
```

```
c = (
```

```
    alt.Chart(df).mark_bar().encode(
```

```
        x = "adtype:O",
```

```
        y = "roi:Q",
```

```
        color= alt.condition(
```

```
            alt.datum.online == "yes",
```

```
            alt.value("orange"),
```

```
            alt.value("steelblue"))
```

```
    ).properties(
```

```
        width=300,
```

```
        height=200
```

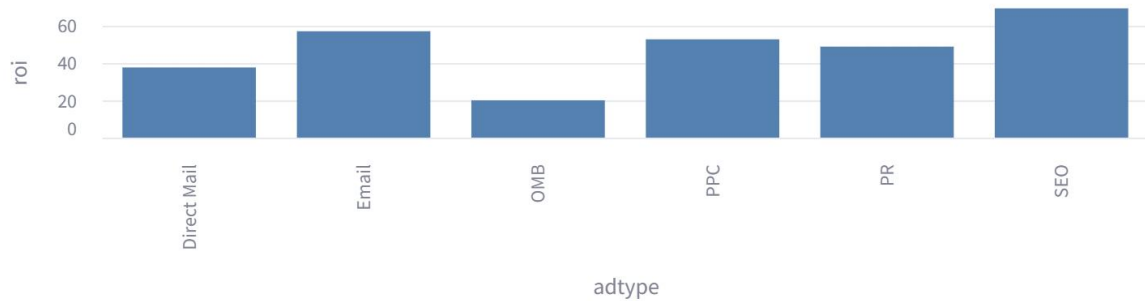
```
    )
```

```
    .interactive()
```

```
)
```

```
st.altair_chart(c, use_container_width=True)
```

	adtype	roi
0	SEO	68.9
1	Email	56.7
2	PPC	52.4
3	PR	48.5
4	Direct Mail	37.4
5	OMB	19.9



```
# fifth2Daltair.py
```

```
import numpy as np
```

```
import pandas as pd
```

```

import altair as alt

import streamlit as st

ad_type = ["SEO", "Email", "PPC", "PR", "Direct Mail", "OMB"]
roi = [68.9, 56.7, 52.4, 48.5, 37.4, 19.9]

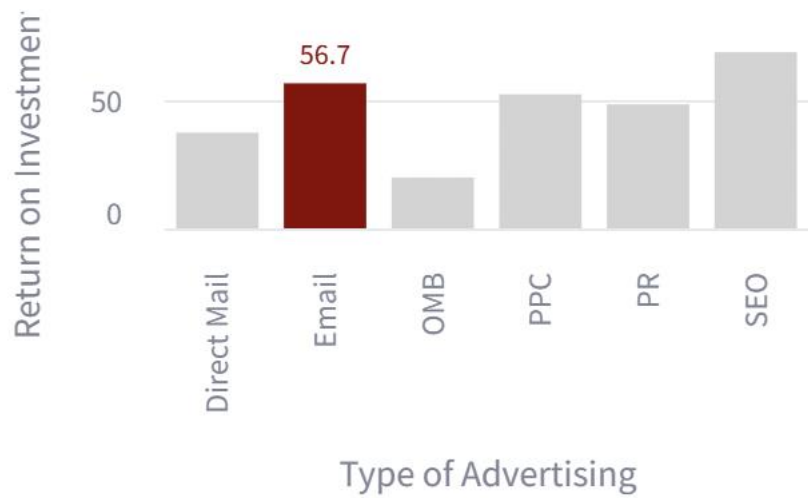
data = {"adtype":ad_type, "roi":roi}
df = pd.DataFrame(data)

roiBars = alt.Chart(df).mark_bar().encode(
    x = alt.X("adtype:O", title="Type of Advertising"),
    y = alt.Y("roi:Q", title="Return on Investment [%]"),
    color= alt.condition(
        alt.datum.adtype == "Email",
        alt.value("darkred"),
        alt.value("lightgrey"))
).properties(
    width=300,
    height=200
)

roiText = roiBars.mark_text(
    align='center',
    baseline='middle',
    dy=-10, # Nudges text up so it doesn't appear on top of the bar
    color="darkred"
).encode(
    text="roi:Q",
    opacity= alt.condition(
        alt.datum.adtype == "Email",
        alt.value(1.0),
        alt.value(0.0))
)

st.altair_chart(roiBars+roiText)

```

```
# sixth2Daltair.py
```

```
import numpy as np
```

```
import pandas as pd
```

```
import altair as alt
```

```
import streamlit as st
```

```
ad_type = ["SEO", "Email", "PPC", "PR", "Direct Mail", "OMB"]
```

```
roi = [68.9, 56.7, 52.4, 48.5, 37.4, 19.9]
```

```
data = {"adtype":ad_type, "roi":roi}
```

```
df = pd.DataFrame(data)
```

```
# define a selector of type "single" (only a single element active at a time)
```

```
# and define what behaviour it should do if nothing is clicked:
```

```
# if "all" is selected, all bars will be dark red by default until you click on one
```

```
# if "none" is selected, all bars will be lightgrey until you click on one
```

```
selector = alt.selection(type="single", empty='none')
```

```
roi_bars = alt.Chart(df).mark_bar().encode(
```

```
    x = alt.X("adtype:O", title="Type of Advertising",
```

```
    sort = alt.EncodingSortField(
```

```
        field="roi",
```

```
        order="descending"
```

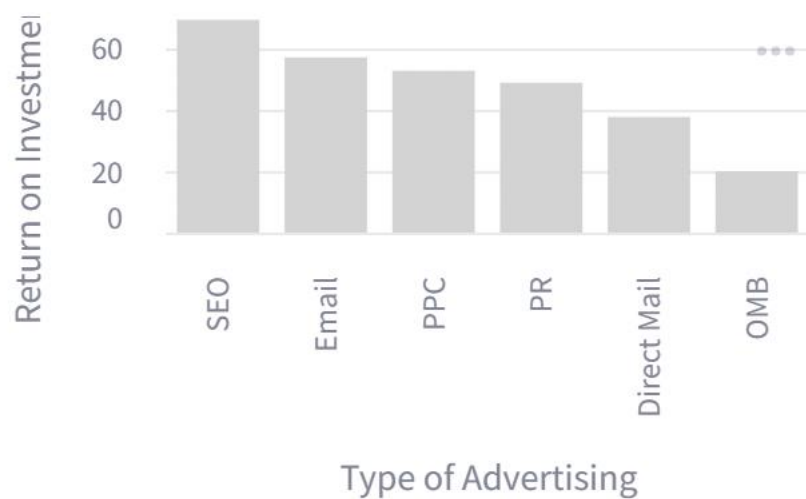
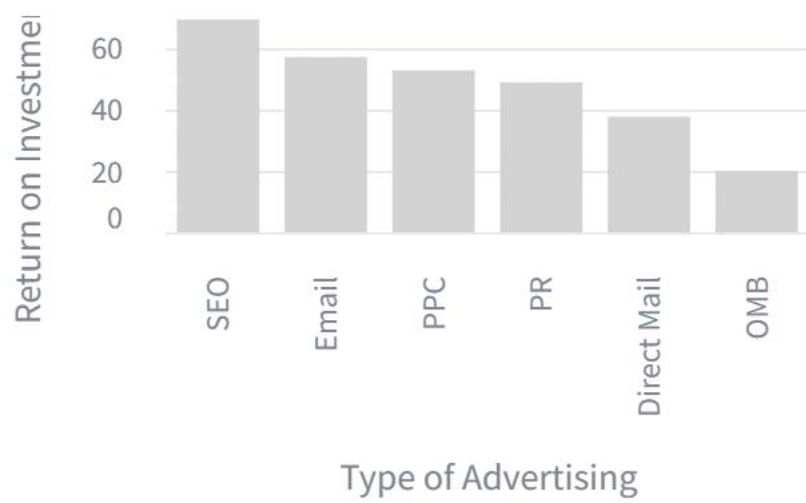
```

    )
  },
  y = alt.Y("roi:Q",
    title="Return on Investment [%]"
  ),
  color= alt.condition(
    selector, # replace the previous condition with the selector
    alt.value("darkred"),
    alt.value("lightgrey"))
).add_selection(
  selector # Add the selector here so that the chart knows to use it
).properties(
  width=300,
  height=200
)

```

```
roiBars
```

```
st.altair_chart(roiBars)
```



```
# HTpenguins2.py
```

```
import altair as alt
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import streamlit as st
```

```
st.title("Palmer's Penguins")
```

```
st.markdown("Use this Streamlit app to make your own scatterplot about penguins!")
```

```
selected_species = st.selectbox(
```

```
    "What species would you like to visualize?", ["Adelie", "Gentoo", "Chinstrap"])
```

```
selected_x_var = st.selectbox(
```

```

    "What do you want the x variable to be?",

    ["bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"],
)

selected_y_var = st.selectbox(

    "What about the y?",

    ["bill_depth_mm", "bill_length_mm", "flipper_length_mm", "body_mass_g"],
)

penguins_df = pd.read_csv("penguins.csv")

penguins_df = penguins_df[penguins_df["species"] == selected_species]

alt_chart = (

    alt.Chart(penguins_df, title="Scatterplot of Palmer's Penguins")

    .mark_circle()

    .encode(

        x=selected_x_var,

        y=selected_y_var,

        color="species",

    )

    .interactive()

)

st.altair_chart(alt_chart, use_container_width=True)

```

Palmer's Penguins

Use this Streamlit app to make your own scatterplot about penguins!

What species would you like to visualize?

Adelie

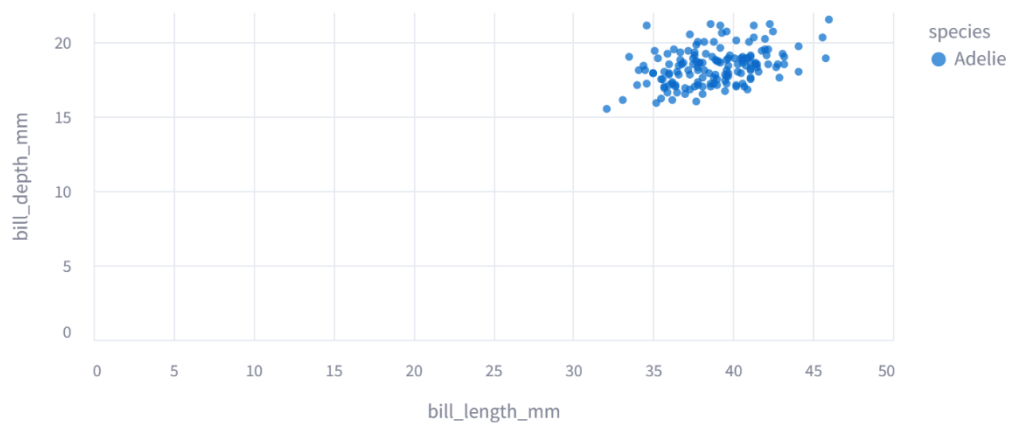
What do you want the x variable to be?

bill_length_mm

What about the y?

bill_depth_mm

Scatterplot of Palmer's Penguins



```
# HTpenguins3_zoomable.py
```

```
import altair as alt
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import streamlit as st
```

```
st.title("Palmer's Penguins")
```

```
st.markdown("Use this Streamlit app to make your own scatterplot about penguins!")
```

```
selected_species = st.selectbox(
```

```
    "What species would you like to visualize?", ["Adelie", "Gentoo", "Chinstrap"])
```

```
selected_x_var = st.selectbox(
```

```
    "What do you want the x variable to be?",
```

```
    ["bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"],
```

```
)
```

```
selected_y_var = st.selectbox(
    "What about the y?",
    ["bill_depth_mm", "bill_length_mm", "flipper_length_mm", "body_mass_g"],
)
```

```
penguins_df = pd.read_csv("penguins.csv")
penguins_df = penguins_df[penguins_df["species"] == selected_species]
```

```
alt_chart = (
    alt.Chart(penguins_df, title=f"Scatterplot of {selected_species} Penguins")
    .mark_circle()
    .encode(
        x=selected_x_var,
        y=selected_y_var,
        color="species",
    )
    .interactive()
)
st.altair_chart(alt_chart, use_container_width=True)
```

Palmer's Penguins

Use this Streamlit app to make your own scatterplot about penguins!

What species would you like to visualize?

Adelie

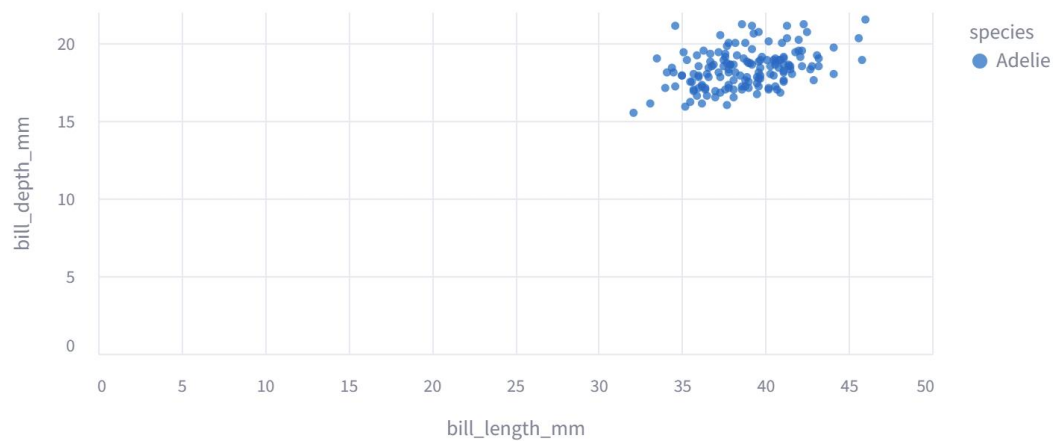
What do you want the x variable to be?

bill_length_mm

What about the y?

bill_depth_mm

Scatterplot of Adelie Penguins



```
# HTpenguins4.py
```

```
import altair as alt
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import streamlit as st
```

```
st.title("Palmer's Penguins")
```

```
st.markdown("Use this Streamlit app to make your own scatterplot about penguins!")
```

```
penguin_file = st.file_uploader("Select your Local Penguins CSV (default provided)")
```

```
if penguin_file is not None:
```

```
    penguins_df = pd.read_csv(penguin_file)
```

```
else:
```

```
    penguins_df = pd.read_csv("penguins.csv")
```

```

selected_x_var = st.selectbox(
    "What do you want the x variable to be?",
    ["bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"],
)

selected_y_var = st.selectbox(
    "What about the y?",
    ["bill_depth_mm", "bill_length_mm", "flipper_length_mm", "body_mass_g"],
)

penguins_df = pd.read_csv("penguins.csv")

alt_chart = (
    alt.Chart(penguins_df, title="Scatterplot of Palmer's Penguins")
    .mark_circle()
    .encode(
        x=selected_x_var,
        y=selected_y_var,
        color="species",
    )
    .interactive()
)

st.altair_chart(alt_chart, use_container_width=True)

```


Palmer's Penguins

Use this Streamlit app to make your own scatterplot about penguins!

Select your Local Penguins CSV (default provided)



Drag and drop file here

Limit 200MB per file

Browse files

What do you want the x variable to be?

bill_length_mm

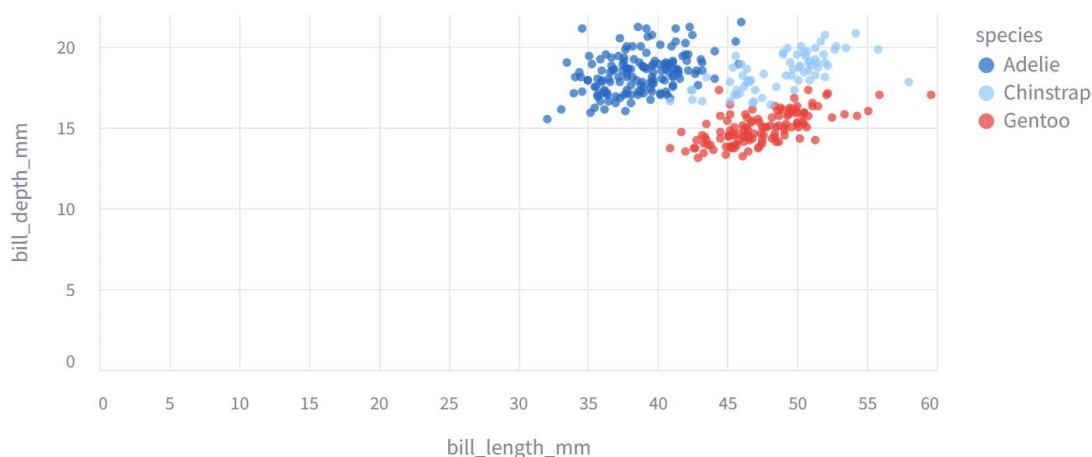


What about the y?

bill_depth_mm



Scatterplot of Palmer's Penguins



```
# HTpenguins5.py
```

```
import altair as alt
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import streamlit as st
```

```
st.title("Palmer's Penguins")
```

```
st.markdown("Use this Streamlit app to make your own scatterplot about penguins!")
```

```
penguin_file = st.file_uploader("Select your Local Penguins CSV (default provided)")
```

```
if penguin_file is not None:
```

```
    penguins_df = pd.read_csv(penguin_file)
```

```
else:
```

```

penguins_df = pd.read_csv('penguins.csv')

selected_x_var = st.selectbox(
    "What do you want the x variable to be?",
    ["bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"],
)

selected_y_var = st.selectbox(
    "What about the y?",
    ["bill_depth_mm", "bill_length_mm", "flipper_length_mm", "body_mass_g"],
)

selected_gender = st.selectbox('What gender do you want to filter for?',
                                ['all penguins', 'male penguins', 'female penguins'])

if selected_gender == 'male penguins':
    penguins_df = penguins_df[penguins_df['sex'] == 'male']
elif selected_gender == 'female penguins':
    penguins_df = penguins_df[penguins_df['sex'] == 'female']
else:
    pass

sns.set_style('darkgrid')
markers = {"Adelie": "X", "Gentoo": "s", "Chinstrap": "o"}

alt_chart = (
    alt.Chart(penguins_df, title="Scatterplot of Palmer's Penguins")
    .mark_circle()
    .encode(
        x=selected_x_var,
        y=selected_y_var,
        color="species",
    )
    .interactive()
)

st.altair_chart(alt_chart, use_container_width=True)

```

Palmer's Penguins

Use this Streamlit app to make your own scatterplot about penguins!

Select your Local Penguins CSV (default provided)



Drag and drop file here

Limit 200MB per file

Browse files

What do you want the x variable to be?

bill_length_mm



What about the y?

bill_depth_mm

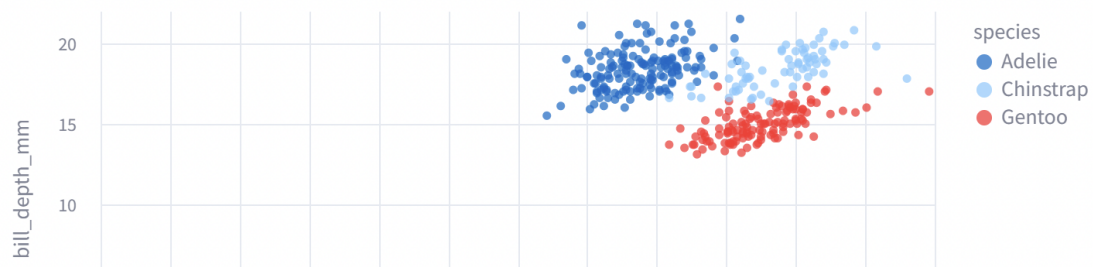


What gender do you want to filter for?

all penguins



Scatterplot of Palmer's Penguins



appdesign directory: simple apps for displaying time series data and widgets such as buttons, input text and number boxes and sliders. Libraries that need to be installed are shown in the import statements at the top of the programs.

animate1.py

```
import streamlit as st
```

```
import pandas as pd
```

```
import numpy as np
```

```
import time
```

```
df = pd.DataFrame(np.random.randn(15, 3), columns=([ "A", "B", "C"])))
```

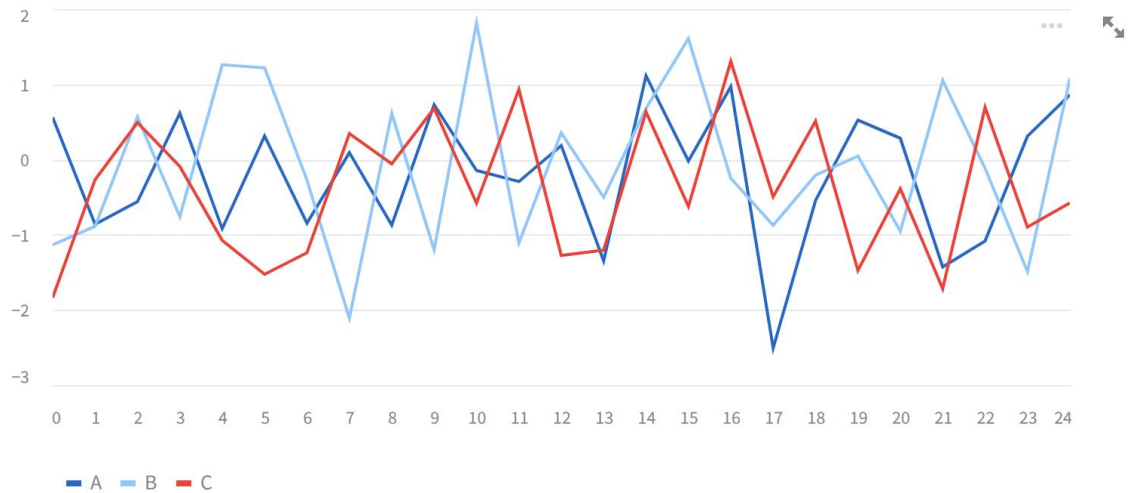
```
my_data_element = st.line_chart(df)
```

```
for tick in range(10):
```

```
    time.sleep(.5)
```

```
add_df = pd.DataFrame(np.random.randn(1, 3), columns=["A", "B", "C"])
my_data_element.add_rows(add_df)
```

```
st.button("Regenerate")
```



Regenerate

```
# button1.py
```

```
"""
```

```
Created on Tue Jun 4 09:59:24 2024
```

```
@author: harveythompson
```

```
"""
```

```
import streamlit as st
```

```
animal_shelter = ['cat', 'dog', 'rabbit', 'bird']
```

```
animal = st.text_input('Type an animal')
```

```
if st.button('Check availability'):
```

```
    have_it = animal.lower() in animal_shelter
```

```
    'We have that animal!' if have_it else 'We don\'t have that animal.'
```

Type an animal

Sheep

Check availability

We don't have that animal.

button2.py

"""

Created on Tue Jun 4 10:01:54 2024

@author: harveythompson

"""

import streamlit as st

if 'clicked' not in st.session_state:

st.session_state.clicked = False

def click_button():

st.session_state.clicked = True

st.button('Click me', on_click=click_button)

if st.session_state.clicked:

The message and nested widget will remain on the page

st.write('Button clicked!')

st.slider('Select a value')

Click me

Button clicked!

Select a value



100

button3.py

"""

Created on Tue Jun 4 10:04:23 2024

@author: harveythompson

```
"""
```

```
import streamlit as st
```

```
if 'button' not in st.session_state:
```

```
    st.session_state.button = False
```

```
def click_button():
```

```
    st.session_state.button = not st.session_state.button
```

```
st.button('Click me', on_click=click_button)
```

```
st.slider('Select a value', disabled=st.session_state.button)
```



Select a value



button4.py

```
"""
```

Created on Tue Jun 4 10:07:40 2024

@author: harveythompson

```
"""
```

```
import streamlit as st
```

```
if 'stage' not in st.session_state:
```

```
    st.session_state.stage = 0
```

```
def set_state(i):
```

```
    st.session_state.stage = i
```

```

if st.session_state.stage == 0:

    st.button('Begin', on_click=set_state, args=[1])

if st.session_state.stage >= 1:

    name = st.text_input('Name', on_change=set_state, args=[2])

if st.session_state.stage >= 2:

    st.write(f'Hello {name}!')

    color = st.selectbox(

        'Pick a Color',

        [None, 'red', 'orange', 'green', 'blue', 'violet'],

        on_change=set_state, args=[3]

    )

    if color is None:

        set_state(2)

if st.session_state.stage >= 3:

    st.write(f':{color}[Thank you!']')

    st.button('Start Over', on_click=set_state, args=[0])

```

Name

Harvey Morpeth Thompson

Hello Harvey Morpeth Thompson!

Pick a Color

red

Thank you!

Start Over

button5.py

"""

Created on Tue Jun 4 10:16:03 2024

@author: harveythompson

"""

import streamlit as st

```

def display_input_row(index):

    left, middle, right = st.columns(3)

    left.text_input('First', key=f'first_{index}')

    middle.text_input('Middle', key=f'middle_{index}')

    right.text_input('Last', key=f'last_{index}')


if 'rows' not in st.session_state:

    st.session_state['rows'] = 0


def increase_rows():

    st.session_state['rows'] += 1


st.button('Add person', on_click=increase_rows)


for i in range(st.session_state['rows']):

    display_input_row(i)


# Show the results
st.subheader('People')

for i in range(st.session_state['rows']):

    st.write(

        f'Person {i+1}:',

        st.session_state[f'first_{i}'],

        st.session_state[f'middle_{i}'],

        st.session_state[f'last_{i}']

    )

```


Add person

First

John

Middle

James

Last

Smith

First

Middle

Last

People

Person 1: John James Smith

Person 2:

button6.py

"""

Created on Wed Jun 5 09:28:57 2024

@author: harveythompson

"""

import streamlit as st

import pandas as pd

import time

def expensive_process(option, add):

with st.spinner('Processing...'):

time.sleep(5)

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}) + add

return (df, add)

cols = st.columns(2)

option = cols[0].selectbox('Select a number', options=['1', '2', '3'])

add = cols[1].number_input('Add a number', min_value=0, max_value=10)

if 'processed' not in st.session_state:

st.session_state.processed = {}

```
# Process and save results
```

```
if st.button('Process'):
```

```
    result = expensive_process(option, add)
```

```
    st.session_state.processed[option] = result
```

```
if option in st.session_state.processed:
```

```
    st.write(f'Option {option} processed with add {add}')
```

```
    st.write(st.session_state.processed[option][0])
```

Select a number

Add a number

Process

Option 1 processed with add 2

	A	B	C
0	3	6	9
1	4	7	10
2	5	8	11

```
# button7.py
```

```
"""
```

```
Created on Wed Jun 5 09:32:38 2024
```

```
@author: harveythompson
```

```
"""
```

```
import streamlit as st
```

```
if st.button('Button 1'):
```

```
    st.write('Button 1 was clicked')
```

```
if st.button('Button 2'):
```

```
    # This will never be executed.
```

```
    st.write('Button 2 was clicked')
```

Button 1

Button 1 was clicked

Button 2

Bisector directory: an applications which allows you to specify and minimise a function of one variable within a given range. Allows you to visualise the solution process. In addition to numpy, streamlit and matplotlib, need to install streamlit_scrollable_textbox library

v8_streamlitbisector.py

Python implementation of the Bisector algorithm for use in Tkinter

import numpy as np

import streamlit as st

import streamlit_scrollable_textbox as stx

import matplotlib.pyplot as plt

For creating the csv file

import sys

import sympy as sp

import time

#####

Functions

#####

define bisector functions

def bisector_calculation(a,b,tol):

L=b-a # domain length

dx=0.25*L # step size reduction parameter

x1=a+dx

x0=a+2*dx

x2=a+3*dx

objective function

```

# obj = variables['Objective']

obj = st.session_state.obj

# store points and functions values

xout = np.linspace(a,b,101)

yout = obj(xout)

outstr=""

converged=0

ncalls=0 # number of call to bisector

xpts = [] # xiteration points

fpts = [] # xiterations function values

ncalls = 0

while (converged==0):

    ncalls=ncalls+1

    fa=obj(a); f1=obj(x1); f0=obj(x0); f2=obj(x2); fb=obj(b);

    outstr+=('iteration {0:5d} of bisector search, domain length = {1:10.5f}\n'.format(ncalls,L))

    outstr+=('a={0:6.3f} f(a)={1:6.3f}\n'.format(a,fa))

    outstr+=('x1={0:6.3f} f(x1)={1:6.3f}\n'.format(x1,f1))

    outstr+=('x0={0:6.3f} f(x0)={1:6.3f}\n'.format(x0,f0))

    outstr+=('x2={0:6.3f} f(x2)={1:6.3f}\n'.format(x2,f2))

    outstr+=('b={0:6.3f} f(b)={1:6.3f}\n'.format(b,fb))

    if ((f2 > f0) and (f0 > f1)):

        b=x0

        x0=x1

    elif ((f2 < f0) and (f0 < f1)):

        a=x0

        x0=x2

    elif ((f1 > f0) and (f2 > f0)):

        a=x1

        b=x2

    else:

        outstr+=('Degenerate search - bisector test conditions not satisfied\n')

        sys.exit()

# update bisector search positions

xpts.append(0.5*(a+b))

fpts.append(obj(0.5*(a+b)))

```

```

L=0.5*L

# check tolerance
if (L < tol):
    converged=1
else:
    dx=0.25*L
    x1=a+dx
    x0=a+2*dx
    x2=a+3*dx

# end while loop

xmin=0.5*(a+b); fmin=obj(xmin)
outstr+='search completed after = {0:5d} xmin={1:10.5f} fmin = {2:10.5f}\n'.format(ncalls,xmin,fmin))
print('finished')

# save dictionary data
st.session_state.Calcs = outstr

st.session_state.xout = xout
st.session_state.yout= yout
st.session_state.xpts= xpts
st.session_state.fpts= fpts
st.session_state.xmin=xmin
st.session_state.fmin=fmin
st.session_state.ncalls = ncalls

return xmin, fmin

# Function to create a plotting function
def create_plot():

    # Get current parameter values
    xmin = st.session_state.xmin
    fmin = st.session_state.fmin

    # Get the current axis
    # plot2 = plt.figure(figsize=[8,8]) # NB plot2 needs to be different from plot1
    # plot2 = plt.figure() # NB plot2 needs to be different from plot1

```

```

plt.figure(figsize=(6,6))

plt.figure()

# Plot the objective function
xout = st.session_state.xout
yout = st.session_state.yout
plt.plot(xout, yout, label='Objective Function')

# Plot out the bisector points - NB always need to use
xpts = st.session_state.xpts
fpts = st.session_state.fpts
plt.scatter(xpts, fpts, label='Bisector Points', color='red', marker='x')

xm = []
fm = []
xm.append(xmin)
fm.append(fmin)

# Plot the minimum point
plt.scatter(xm, fm, label = 'Minimum point', color='blue', marker='o')

# Configure plot details
plt.ylabel('y')
plt.xlabel('x')
plt.title('Bisector Search on '+st.session_state.expression)
plt.legend()

return plt

# Function to create an animated plotting function
def animate_plot():

# Get current parameter values
xmin = st.session_state.xmin
fmin = st.session_state.fmin

# Get the current axis
# plot2 = plt.figure(figsize=[8,8]) # NB plot2 needs to be different from plot1
# plot2 = plt.figure() # NB plot2 needs to be different from plot1
plt.figure(figsize=(6,6))

```

```

plt.figure()

# Plot the objective function
xout = st.session_state.xout
yout = st.session_state.yout
plt.plot(xout, yout, label='Objective Function')

# Plot out the bisector points - NB always need to use
xpts = st.session_state.xpts
fpts = st.session_state.fpts

# create a set of points with st.session_state.npts points in it
# st.session_state.npts increases up to st.session_state.ncalls
npts = st.session_state.npts - 1
plt.scatter(xpts[0:npts], fpts[0:npts], label='Bisector Points', color='red', marker='x')

xm = []
fm = []
xm.append(xmin)
fm.append(fmin)

# Plot the minimum point
plt.scatter(xm, fm, label = 'Minimum point', color='blue', marker='o')

# Configure plot details
plt.ylabel('y')
plt.xlabel('x')
plt.title('Bisector Search on '+st.session_state.expression)
plt.legend()
return plt

#####

# Main program

#####

# st.set_page_config(layout='wide')

st.title("Bisector Algorithm Program")
st.write("This application enables you to experiment with bisector solution of 1-D equations")

```

```

st.write("Parameter settings")

# initialise the objective function
exp = st.text_input("Objective Function",value='7*x**2-20*x+22')

if exp is not None:
    expression = exp
else:
    expression = '7*x**2-20*x+22'
x = sp.symbols('x')

try:
    obj_function = sp.simplify(expression)
except Exception as e:
    st.error(f"An error occured using sympify: {str(e)}")
    st.stop()

try:
    obj = sp.lambdify(x, obj_function, 'numpy')
except Exception as e:
    st.error(f"An error occured using lambdify: {str(e)}")
    st.stop()

st.session_state.obj = obj
st.session_state.expression = expression

cols = st.columns([1,1,1])
minx = cols[0].number_input("Minimum x",-5.0,0.0,-2.0,0.1)
maxx = cols[1].number_input("Maximum x",0.0,5.0,2.0,0.1)
tol = cols[2].number_input("Tolerance",0.0001,0.1,0.01)

cols2 = st.columns([1,1,1])
xmin,fmin = bisector_calculation(minx,maxx,tol)

cols2[0].text("xmin="+"{:.4f}".format(xmin))
cols2[1].text("fmin="+"{:.4f}".format(fmin))

# plot out calculations

```



```

if 'plotgraph' not in st.session_state:

    st.session_state.plotgraph = 1

    if st.button("Plot Graph and Calculations"):

        figplt = create_plot()

        st.pyplot(figplt)

        # st.text(st.session_state.Calcs)

        st.write("Bisector Calculations")

        stx.scrollableTextbox(st.session_state.Calcs,height=200)

    else:

        figplt = create_plot()

        st.pyplot(figplt)

        # st.text(st.session_state.Calcs)

        st.write("Bisector Calculations")

        stx.scrollableTextbox(st.session_state.Calcs,height=200)


# animate calculations

if st.button("Animate Graph"):

    st.session_state.npts = 1

    figplt = animate_plot()

    the_plot = st.pyplot(figplt)

    for i in range(st.session_state.ncalls-1):

        st.session_state.npts += 1

        time.sleep(0.5)

        figplt = animate_plot()

        the_plot.pyplot(figplt)

```

Bisector Algorithm Program

This application enables you to experiment with bisector solution of 1-D equations

Parameter settings

Objective Function

$7x^2 - 20x + 22$

Minimum x

-2.00 - +

Maximum x

2.00 - +

Tolerance

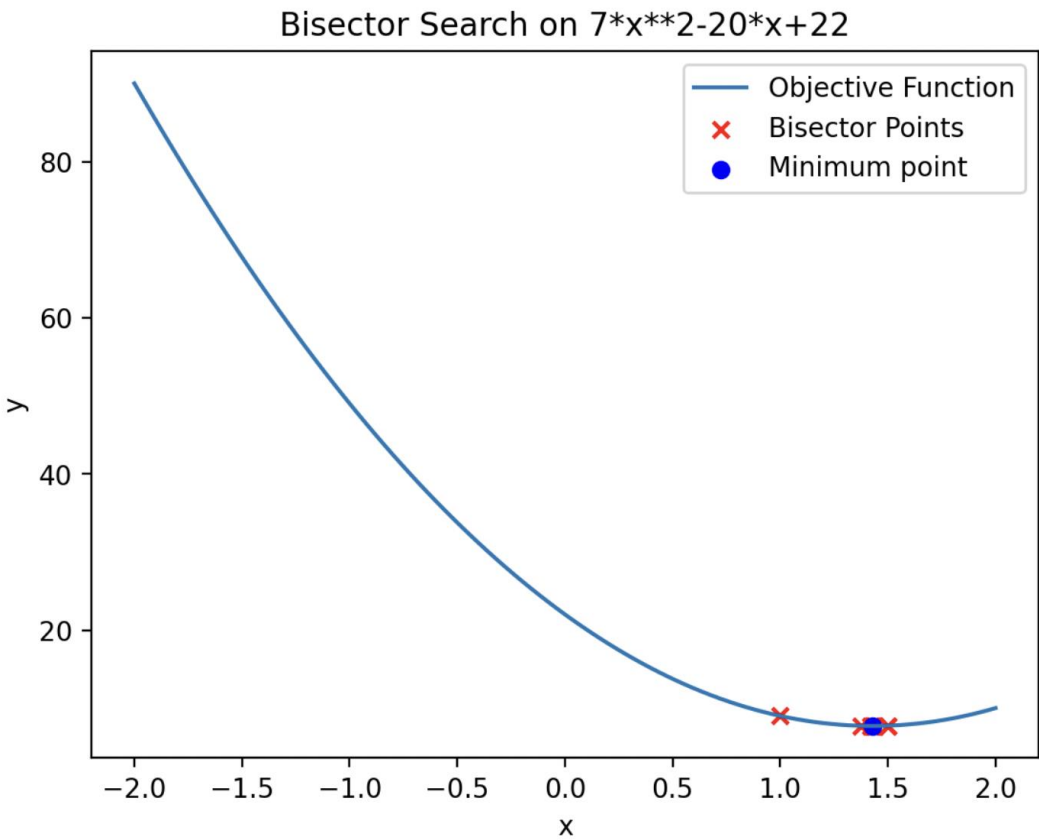
0.01 - +

xmin=1.4297

fmin=7.7143

Plot Graph and Calculations

Animate Graph



Bisector Calculations

```
iteration 1 of bisector search, domain length = 4.00000
a=-2.000 f(a)=90.000
x1=-1.000 f(x1)=49.000
x0= 0.000 f(x0)=22.000
x2= 1.000 f(x2)= 9.000
b= 2.000 f(b)=10.000
iteration 2 of bisector search, domain length = 2.00000
a= 0.000 f(a)=22.000
```

Animate Graph

Caching directory: examples which explore the benefits of using `@st.cache_data` option to cache large datasets rather than having to read them in every time streamlit runs upon an update.

caching1.py

```
import streamlit as st
```

```
import pandas as pd
```

```
import numpy as np
```

```
@st.cache_data(show_spinner='A first caching example') # add the caching decorator
```

```
def load_data(url):
```

```
    df = pd.read_csv(url) # download the data
```

```
    return df
```

```
df = load_data("https://github.com/plotly/datasets/raw/master/uber-rides-data1.csv")
```

```
st.dataframe(df)
```

```
st.button("Rerun")
```

	Date/Time	Lat	Lon
0	2014-04-01 00:11:00	40.769	-73.9549
1	2014-04-01 00:17:00	40.7267	-74.0345
2	2014-04-01 00:21:00	40.7316	-73.9873
3	2014-04-01 00:28:00	40.7588	-73.9776
4	2014-04-01 00:33:00	40.7594	-73.9722
5	2014-04-01 00:33:00	40.7383	-74.0403
6	2014-04-01 00:39:00	40.7223	-73.9887
7	2014-04-01 00:45:00	40.762	-73.979
8	2014-04-01 00:55:00	40.7524	-73.996
9	2014-04-01 01:01:00	40.7575	-73.9846

Rerun

caching2.py

Created on Mon May 20 10:52:49 2024

import streamlit as st

class MyCustomClass:

def __init__(self, initial_score: int):

self.my_score = initial_score

def hash_func(obj: MyCustomClass) -> int:

return obj.my_score # or any other value that uniquely identifies the object

```
@st.cache_data(hash_funcs={MyCustomClass: hash_func})
def multiply_score(obj: MyCustomClass, multiplier: int) -> int:
    return obj.my_score * multiplier
```

```
initial_score = st.number_input("Enter initial score", value=15)
```

```
score = MyCustomClass(initial_score)
```

```
multiplier = 2
```

```
st.write(multiply_score(score, multiplier))
```

Enter initial score

17

- +

34

caching3.py

```
import streamlit as st
```

```
class MyCustomClass:
```

```
    def __init__(self, initial_score: int):
```

```
        self.my_score = initial_score
```

```
@st.cache_data(hash_funcs={"__main__.MyCustomClass": lambda x: hash(x.my_score)})
```

```
def multiply_score(self, multiplier: int) -> int:
```

```
    return self.my_score * multiplier
```

```
initial_score = st.number_input("Enter initial score", value=15)
```

```
score = MyCustomClass(initial_score)
```

```
multiplier = 2
```

```
st.write(score.multiply_score(multiplier))
```

Enter initial score

16

- +

32

caching4.py

```
import streamlit as st
```

```
import pandas as pd

import numpy as np


@st.cache_data # turn off copying behaviour
def load_data(url):

    df = pd.read_csv(url) # download the data

    return df


df = load_data("https://github.com/plotly/datasets/raw/master/uber-rides-data1.csv")

st.dataframe(df)


df.drop(columns=['Lat'], inplace=True) # mutate the dataframe inplace


st.button("Rerun")
```

	Date/Time	Lat	Lon
0	2014-04-01 00:11:00	40.769	-73.9549
1	2014-04-01 00:17:00	40.7267	-74.0345
2	2014-04-01 00:21:00	40.7316	-73.9873
3	2014-04-01 00:28:00	40.7588	-73.9776
4	2014-04-01 00:33:00	40.7594	-73.9722
5	2014-04-01 00:33:00	40.7383	-74.0403
6	2014-04-01 00:39:00	40.7223	-73.9887
7	2014-04-01 00:45:00	40.762	-73.979
8	2014-04-01 00:55:00	40.7524	-73.996
9	2014-04-01 01:01:00	40.7575	-73.9846

Rerun

Callbacks directory: a simple example which shows how to respond to a widget update using a callback function

callack1.py

import streamlit as st

if "attendance" not in st.session_state:

st.session_state.attendance = set()

def take_attendance():

if st.session_state.name in st.session_state.attendance:

```

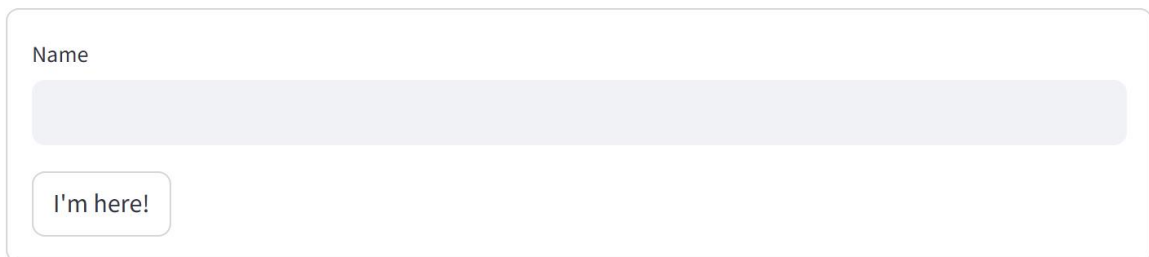
        st.info(f"{st.session_state.name} has already been counted.")
    else:
        st.session_state.attendance.add(st.session_state.name)

```

```

with st.form(key="my_form"):
    st.text_input("Name", key="name")
    st.form_submit_button("I'm here!", on_click=take_attendance)

```



Curve fitting directory: two examples which enable you to explore the effect of changing the order of the polynomial fit to a dataset

v1st_curvefitting.py

```

import numpy as np
import streamlit as st
import matplotlib.pyplot as plt
import time

from mpl_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from matplotlib.figure import Figure

#####

# Functions

#####

# Create function to carry out curve fitting
def fitting(xpts,ypts,x,order):

    p = np.polyfit(xpts,ypts,order)

    p_poly = np.poly1d(p)

    y_fit = p_poly(x)

```



```

    return y_fit

# read data to be used in plotting
def read_data():

    data = dict()

    builddata = open('build.txt','r').readlines()

    xpts = []
    ypts = []
    for line in builddata:
        xpts.append(float(line.split()[0]))
        ypts.append(float(line.split()[1]))
    data['xpts']=xpts
    data['ypts']=ypts

    N = 101
    x = np.linspace(0,2,N)
    data['x']=x

    st.session_state.xpts = xpts
    st.session_state.ypts = ypts
    st.session_state.x = x

    return data

# Function to update the surrogate model plot
def update_curve_plot(data):

    # set data from dictionary
    xpts=st.session_state.xpts; ypts=st.session_state.ypts
    x=st.session_state.x; npoly=st.session_state.order
    orderlabel = ""
    if (npoly == 1):
        orderlabel='first order fit'
    elif (npoly == 2):
        orderlabel='second order fit'

```

```

elif (npoly == 3):
    orderlabel='third order fit'
elif (npoly == 4):
    orderlabel='fourth order fit'
elif (npoly == 5):
    orderlabel='fifth order fit'
elif (npoly == 6):
    orderlabel='sixth order fit'

# Get the current axis

plt.figure()

plt.xlabel('x')

plt.ylabel('Pox')

titlestr='Comparison between experimental data points for a '+orderlabel

plt.title(titlestr)

plt.plot(xpts,ypts,marker='o',c='r')

plt.plot(x,fitting(xpts,ypts,x,npoly))

plt.xlim([0,2])

plt.ylim([1,4])

plt.legend(['experimental data',orderlabel])

return plt

```

```
#####
```

```
# Msin Streamlit commands
```

```
#####
```

```
# Streamlit commands
```

```
st.title("Polynomial Curve Fitting")
```

```
st.write("This application enables you to explore the effect of the order of interpolation")
```

```
# read full ddataset
```

```
data = read_data()
```

```
orders = [1,2,3,4,5,6]
```

```
selected_order = st.selectbox('Select Order of Interpolation',orders)
```

```
if 'order' not in st.session_state:
```

```
    st.session_state.order = 1
```

```
    time.sleep(0.1)
```

```
else:
```

```
st.session_state.order = selected_order
```

```
figplt = update_curve_plot(data)
```

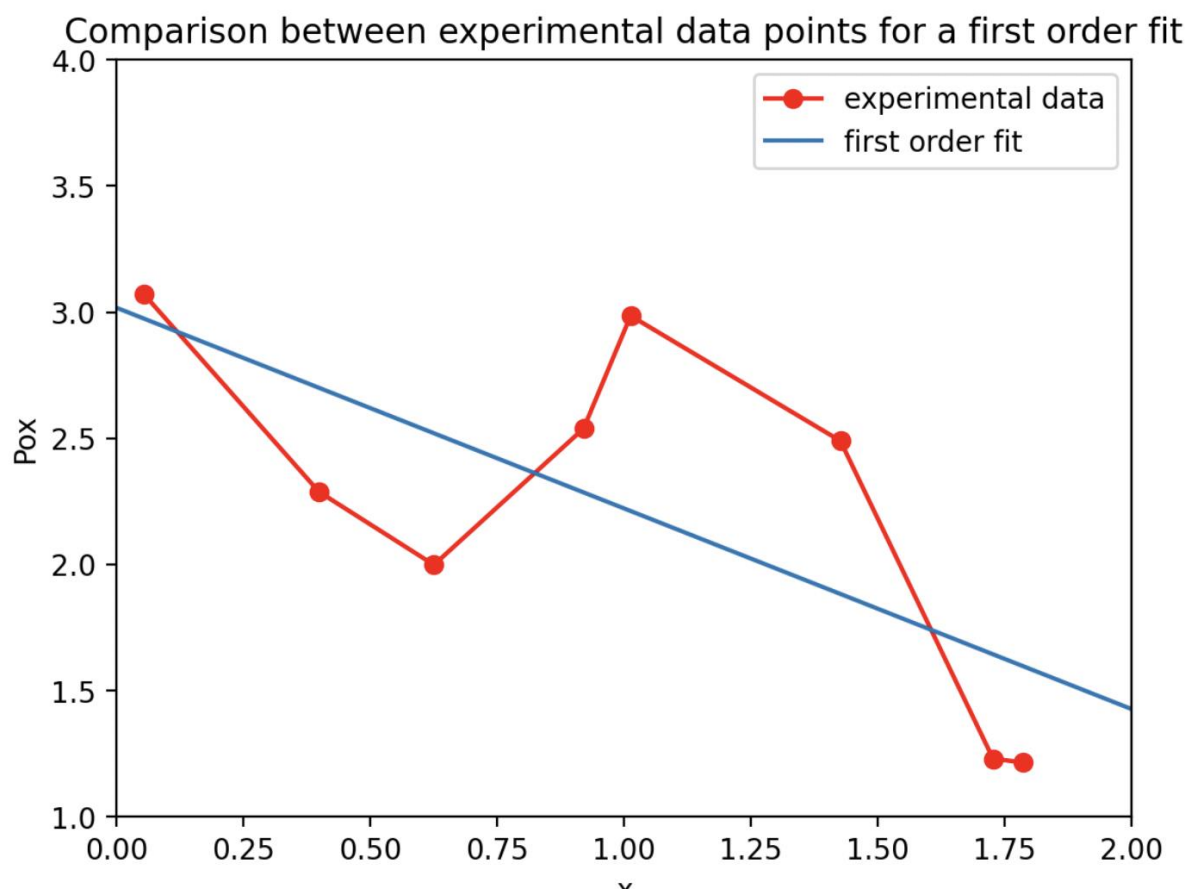
```
st.pyplot(figplt)
```

Polynomial Curve Fitting

This application enables you to explore the effect of the order of interpolation

Select Order of Interpolation

1



```
# v2st_curvefitting.py
```

```
import numpy as np
```

```
import streamlit as st
```

```
import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from matplotlib import cm
```

```

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk

from matplotlib.ticker import LinearLocator, FormatStrFormatter

from matplotlib.figure import Figure

#####

# Functions

#####

# Create function to carry out curve fitting

def fitting(xpts,ypts,x,order):

    p = np.polyfit(xpts,ypts,order)

    p_poly = np.poly1d(p)

    y_fit = p_poly(x)

    return y_fit

# read data to be used in plotting

def read_data():

    data = dict()

    builddata = open('build.txt','r').readlines()

    xpts = []

    ypts = []

    for line in builddata:

        xpts.append(float(line.split()[0]))

        ypts.append(float(line.split()[1]))

    data['xpts']=xpts

    data['ypts']=ypts

    N = 101

    x = np.linspace(0,2,N)

    data['x']=x

    st.session_state.xpts = xpts

    st.session_state.ypts = ypts

    st.session_state.x = x

```

```
return data
```

```
# Function to update the surrogate model plot
```

```
def update_curve_plot(data):
```

```
    # set data from dictionary
```

```
    xpts=st.session_state.xpts; ypts=st.session_state.ypts
```

```
    x=st.session_state.x; npoly=st.session_state.order
```

```
    orderlabel = ''
```

```
    if (npoly == 1):
```

```
        orderlabel='first order fit'
```

```
    elif (npoly == 2):
```

```
        orderlabel='second order fit'
```

```
    elif (npoly == 3):
```

```
        orderlabel='third order fit'
```

```
    elif (npoly == 4):
```

```
        orderlabel='fourth order fit'
```

```
    elif (npoly == 5):
```

```
        orderlabel='fifth order fit'
```

```
    elif (npoly == 6):
```

```
        orderlabel='sixth order fit'
```

```
    # Get the current axis
```

```
    plt.figure()
```

```
    plt.xlabel('x')
```

```
    plt.ylabel('Pox')
```

```
    titlestr='Comparison between experimental data points for a '+orderlabel
```

```
    plt.title(titlestr)
```

```
    plt.plot(xpts,ypts,marker='o',c='r')
```

```
    plt.plot(x,fitting(xpts,ypts,x,npoly))
```

```
    plt.xlim([0,2])
```

```
    plt.ylim([1,4])
```

```
    plt.legend(['experimental data',orderlabel])
```

```
    return plt
```

```
#####
```

```
# Msin Streamlit commands
```

```
#####

# Streamlit commands

st.title("Polynomial Curve Fitting")

st.write("This application enables you to explore the effect of the order of interpolation")

# read full dataset

data = read_data()

orders = [1,2,3,4,5,6]

selected_order = st.selectbox('Select Order of Interpolation',orders)

if 'order' not in st.session_state:

    st.session_state.order = 1

    if st.button("Plot Graph and Calculations"):

        figplt = update_curve_plot(data)

        st.pyplot(figplt)

else:

    st.session_state.order = selected_order

    figplt = update_curve_plot(data)

    st.pyplot(figplt)
```

Polynomial Curve Fitting

This application enables you to explore the effect of the order of interpolation

Select Order of Interpolation

1



Plot Graph and Calculations

Customclasses directory: example which displays and manipulates data specified in a simple class

enumexample.py

from enum import Enum

import streamlit as st

class syntax

class Color(Enum):

RED = 1

GREEN = 2

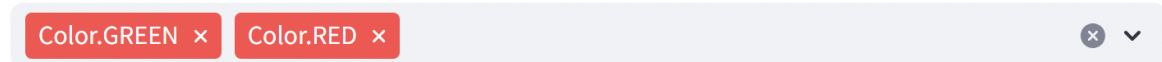
```
BLUE = 3
```

```
selected_colors = set(st.multiselect("Pick colors", options=Color))
```

```
if selected_colors == {Color.RED, Color.GREEN}:
```

```
    st.write("Hooray, you found the color YELLOW!")
```

Pick colors



Hooray, you found the color YELLOW!

Dataframes directory: examples which show how to display and manipulate pandas dataframes in the browser

```
# dataedit1.py
```

```
import streamlit as st
```

```
import pandas as pd
```

```
df = pd.DataFrame(
```

```
[
```

```
    {"command": "st.selectbox", "rating": 4, "is_widget": True},
```

```
    {"command": "st.balloons", "rating": 5, "is_widget": False},
```

```
    {"command": "st.time_input", "rating": 3, "is_widget": True},
```

```
]
```

```
)
```

```
edited_df = st.data_editor(df, num_rows='dynamic') # 📄 An editable dataframe
```

```
favorite_command = edited_df.loc[edited_df["rating"].idxmax()][ "command"]
```

```
st.markdown(f"Your favorite command is **{favorite_command}** 🍷")
```

	⌵ command	⌵ rating	⌵ is_widget
	st.selectbox	4	<input checked="" type="checkbox"/>
	st.balloons	5	<input type="checkbox"/>
	st.time_input	3	<input checked="" type="checkbox"/>

Your favorite command is **st.balloons** 🎈

dataedit2.py

import streamlit as st

import pandas as pd

df = pd.DataFrame(columns=['name','age','color'])

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

config = {

 'name' : st.column_config.TextColumn('Full Name (required)', width='large', required=True),

 'age' : st.column_config.NumberColumn('Age (years)', min_value=0, max_value=122),

 'color' : st.column_config.SelectboxColumn('Favorite Color', options=colors)

}

result = st.data_editor(df, column_config = config, num_rows='dynamic')

if st.button('Get results'):

 st.write(result)

⌵ Full Name (required)	⌵ Age (years)	⌵ Favorite Color

Get results

dataframe1.py


```

import streamlit as st

import pandas as pd

df = pd.DataFrame(
    [
        {"command": "st.selectbox", "rating": 4, "is_widget": True},
        {"command": "st.balloons", "rating": 5, "is_widget": False},
        {"command": "st.time_input", "rating": 3, "is_widget": True},
    ]
)

st.dataframe(df, use_container_width=True)

```

	command	rating	is_widget
0	st.selectbox	4	<input checked="" type="checkbox"/>
1	st.balloons	5	<input type="checkbox"/>
2	st.time_input	3	<input checked="" type="checkbox"/>

dataframe2.py

```

import streamlit as st

import pandas as pd

df = pd.DataFrame(columns=['name', 'age', 'color'])

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

config = {
    'name' : st.column_config.TextColumn('Full Name (required)', width='large', required=True),
    'age' : st.column_config.NumberColumn('Age (years)', min_value=0, max_value=122),
    'color' : st.column_config.SelectboxColumn('Favorite Color', options=colors)
}

result = st.data_editor(df, column_config = config, num_rows='dynamic')

if st.button('Get results'):
    st.write(result)

```

	☰ Full Name (required)	☰ Age (years)	☰ Favorite Color

Get results

Drilling directory: surrogate modelling of the pressure drop in a helical drill. Uses Radial Basis Functions and pre-prepared data from a Leave-One-Out Cross-Validation (LOOCV) exercise stored on the 'data' subdirectory. Uses tabs to display surrogate model calculations, surrogate model and results from LOOCV

```
# v1st_rbf_drilling_leaveoneout.py
```

```
# program to calculate rbf response surface given DoE points and
```

```
# calculating the single and multi-objective optimisation from these responses
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from matplotlib import cm
```

```
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```
import matplotlib.pyplot as plt
```

```
import streamlit as st
```

```
import numpy as np
```

```
from rbffunctions import *
```

```
rbfmodel = 1 # Gaussian weights
```

```
n = 20
```

```
ndv = 2
```

```
x1min_actual = 0.05
```

```
x1max_actual = 0.5
```

```
x2min_actual = 0.0
```

```
x2max_actual = 90.0
```

```
max_pd = 33.56574
```

```
min_pd = 26.7848
```

```
nbetas = 101
```

```
#####
```

```
# reading data into relevant files
```

```
#####
```

```
def read_data():
```

```

st.session_state.Xr = np.loadtxt("data/Xr.txt")
st.session_state.Yr = np.loadtxt("data/Yr.txt")
st.session_state.Zr_rbf = np.loadtxt("data/Zr_rbf.txt")
st.session_state.x_scaled = np.loadtxt("data/x_scaled.txt")
st.session_state.x = np.loadtxt("data/x.txt")
st.session_state.pressure_drop = np.loadtxt("data/pressure_drop.txt")
st.session_state.beta_min = np.loadtxt("data/beta_min.txt")
st.session_state.beta_array = np.loadtxt("data/beta_array.txt")
st.session_state.RMSE_array = np.loadtxt("data/RMSE_array.txt")
lam1 = np.loadtxt("data/lam.txt")
st.session_state.lam = np.transpose(np.asmatrix(lam1))
st.session_state.xopt = np.loadtxt("data/xopt.txt")
st.session_state.yopt = np.loadtxt("data/yopt.txt")
st.session_state.optval = np.loadtxt("data/optval.txt")

```

```
def beta_plot():
```

```

    RMSE_array = st.session_state.RMSE_array
    beta_array = st.session_state.beta_array

    plt.figure()
    plt.xlabel('beta')
    plt.ylabel('RMSE')
    plt.title('Plot out RMSE vs Beta for Leave One Out Cross Validation')
    plt.plot(beta_array, RMSE_array)

    st.pyplot(plt)

```

```
# plot out the RBF surrogate model of pressure drop
```

```
def pd_plot():
```

```

    # Plot out RBF approximation

    plt.figure()

    beta = st.session_state.beta_min

    Xr = st.session_state.Xr
    Yr = st.session_state.Yr
    Zr_rbf = st.session_state.Zr_rbf
    x = st.session_state.x

    pressure_drop = st.session_state.pressure_drop

    xopt = st.session_state.xopt
    yopt = st.session_state.yopt

```

```

optval = st.session_state.optval

plt.suptitle('RBF approximation with beta='+str(beta)+' and n='+str(n),fontsize=10)

#ax = fig.gca(projection='3d')

ax=plt.axes(projection='3d')

surf = ax.plot_surface(Xr, Yr, Zr_rbf, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)

ax.set_zlim(25, 35)

ax.set_xlim(0, x1max_actual)

ax.set_ylim(0, x2max_actual)

ax.zaxis.set_major_locator(LinearLocator(10))

ax.zaxis.set_major_formatter(FormatStrFormatter('%.1f'))

#fig.colorbar(surf, shrink=0.5, aspect=5)


# plot out the scatter points

ax.scatter(x[:,0],x[:,1],pressure_drop, c='r', marker='o',s=4)

ax.scatter(xopt,yopt,optval, c='k', marker='o',s=16) # plot optimum point

ax.set_xlabel('corner radius (mm)')

ax.set_ylabel('orientation angle (degrees)')

ax.set_zlabel('Pressure Drop')

plt.savefig('rbf.jpg')

st.pyplot(plt)


# calculate pressure drop at specified point

def pd_f(x):

    return rbf_point(n,st.session_state.x_scaled,ndv,x,st.session_state.lam,rbfmodel,st.session_state.beta_min)


# calculate surrogate models of thermal resistance and pressure drop

def calc_surrogates(x1,x2):

    xp = np.zeros(ndv)

    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)

    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)

    xp[0] = x1_scaled

    xp[1] = x2_scaled

    pd_val = pd_f(xp)

    return pd_val

```

```

st.title("Drilling Optimisation")

st.write("This application enables you to explore the pressure drop of a twist drill")

st.markdown("---")


checking_password = 0

if (checking_password != 0):

    if 'pwdcheck' not in st.session_state:

        st.session_state['pwdcheck'] = 0

        password_guess = st.text_input('What is the password?')

        if password_guess != st.secrets["password"]:

            st.stop()


# read in data for calculations only at the beginning of session

if 'Xr' not in st.session_state:

    read_data()


tab1, tab2, tab3 = st.tabs(["Surrogate model", "Pressure drop plot", "Leave one out cross validation"])

with tab1:


    # Create the input sliders

    row1 = st.columns([1,1])


    default_value = 0.2

    x1 = row1[0].slider("Corner radius (mm)",x1min_actual,x1max_actual,0.2)

    x2 = row1[1].slider("Design variable x2",x2min_actual,x2max_actual,45.0)

    pd_val = calc_surrogates(x1,x2)

    st.write(f'Pressure drop: {pd_val:.1f} Pa')


with tab2:

    pd_plot()


with tab3:

    beta_plot()


testing = 1

if (testing):

    xp = np.zeros(ndv)

    xp[0] = 0.5

```

```

xp[1] = 0.5

# yp2=rbf_point(n,x_scaled,ndv,xp,lam,rbfmodel,beta)

yp2=pd_f(xp)

```

rbffunctions.py

function to calculate weights for rbfs

def rbfweights(n,x,ndv,f,rbfmodel,beta):

```

import numpy as np

```

```

import sys

```

```

phi = np.zeros(shape=(n,n))

```

```

# phi = np.matrix(phi1)

```

calculate the gram matrix

```

for i in range(0,n):

```

```

    for j in range(0,n):

```

```

        r = 0.0

```

```

        for k in range (0,ndv):

```

```

            r = r + (x[i][k]-x[j][k])**2

```

```

        r = np.sqrt(r)

```

```

        if (rbfmodel == 1):

```

```

            phi[i][j] = np.exp(-beta*(r**2))

```

```

        else:

```

```

            if (rbfmodel == 2):

```

```

                phi[i][j] = np.sqrt(beta**2 + r**2)

```

```

            else:

```

```

                if (rbfmodel == 3):

```

```

                    phi[i][j] = 1/np.sqrt(beta**2 + r**2)

```

```

                else:

```

```

                    print ("invalid rbfmodel value")

```

```

                    sys.exit(0)

```

compute the weights. need to transform f into an array and transpose it

```

farray = np.asarray(f)

```

```

fm = np.asmatrix(farray)

```

```

fmt = fm.transpose()

```

```

lam = np.linalg.solve(phi, fmt)

```

```

return lam

print ("Finished computing rbf weights in rbfweights")

# function to calculate rbf approximation at specified plan design points xa
def rbf(n,x,ndv,xa,na,lam,rbfmodel,beta):

    import numpy as np
    import sys

    phi = np.zeros(shape=(na*na,n))

    # phi = np.zeros(n,n,float)

    # calculate the gram matrix
    for i in range(0,na*na):
        for j in range(0,n):
            r = 0.0
            for k in range (0,ndv):
                r = r + (xa[i][k]-x[j][k])**2
            r = np.sqrt(r)
            if (rbfmodel == 1):
                phi[i][j] = np.exp(-beta*(r**2))
            else:
                if (rbfmodel == 2):
                    phi[i][j] = np.sqrt(beta**2 + r**2)
                else:
                    if (rbfmodel == 3):
                        phi[i][j] = 1/np.sqrt(beta**2 + r**2)
                    else:
                        print ("invalid rbfmodel value")
                        sys.exit(0)

    # compute the rbf response

    # ya = phi*lam

    # phim = np.matrix(phi)

    # lamm = np.matrix(lam)

    ya = np.asmatrix(phi)*lam

    return ya

```

```

print ("Finished computing rbf approximations")

# define function
def yval(x,y):
    yval = x**2 + y**2
    return yval

# define six hump camel function
def sixhumpcamel(x1,x2):
    X1 = 4*x1-2;
    X2 = 2*x2-1;
    fval = (4 - 2.1*(X1**2) + (X1**4)/3)*X1**2 + X1*X2 + (4*(X2**2) - 4)*(X2**2)
    return fval

# generate test dataset from ntest random numbers between 1 and n
def generate_test(ntest,n):
    import random

    testindex = []
    testflag = []

    for i in range(n):
        testflag.append(0)

    num = 0
    for i in range(100):
        if (num < ntest):
            k = random.randint(0,n-1)
            if (testflag[k] == 0):
                testindex.append(k)
                testflag[k] = 1
                num = num + 1
        else:
            break

    return testindex, testflag

```



```
# function to calculate rbf approximation at specified ntest test points x_test
```

```
def rbf_testeval(ntrain,x_train,ndv,x_test,ntest,lam,rbfmodel,beta):
```

```
    import numpy as np
```

```
    import sys
```

```
    phi = np.zeros(shape=(ntest,ntrain))
```

```
    # phi = np.zeros(n,n,float)
```

```
    # calculate the gram matrix
```

```
    for i in range(0,ntest):
```

```
        for j in range(0,ntrain):
```

```
            r = 0.0
```

```
            for k in range (0,ndv):
```

```
                r = r + (x_test[i][k]-x_train[j][k])**2
```

```
            r = np.sqrt(r)
```

```
            if (rbfmodel == 1):
```

```
                phi[i][j] = np.exp(-beta*(r**2))
```

```
            else:
```

```
                if (rbfmodel == 2):
```

```
                    phi[i][j] = np.sqrt(beta**2 + r**2)
```

```
                else:
```

```
                    if (rbfmodel == 3):
```

```
                        phi[i][j] = 1/np.sqrt(beta**2 + r**2)
```

```
                    else:
```

```
                        print ("invalid rbfmodel value")
```

```
                        sys.exit(0)
```

```
    y_test = np.asmatrix(phi)*lam
```

```
    return y_test
```

```
    print ("Finished computing rbf approximations at test data points")
```

```
# function to calculate training and testing data for leave one out validation
```

```
def leaveoneout(i,x,f,n,ndv):
```

```
    import numpy as np
```

```
# create leave one out training and test data sets with test data on index i
```

```
f_train = []
```

```
f_test = []
```

```
x1_train = []
```

```
x1_test = []
```

```
x2_train = []
```

```
x2_test = []
```

```
# create leave one out training and test data sets
```

```
for j in range(0,n):
```

```
    if (j < i):
```

```
        f_train.append(f[j])
```

```
        x1_train.append(x[j][0])
```

```
        x2_train.append(x[j][1])
```

```
    else:
```

```
        if (j == i):
```

```
            f_test.append(f[i])
```

```
            x1_test.append(x[i][0])
```

```
            x2_test.append(x[i][1])
```

```
        else:
```

```
            f_train.append(f[j])
```

```
            x1_train.append(x[j][0])
```

```
            x2_train.append(x[j][1])
```

```
# put training and test data into arrays
```

```
x_test = (np.vstack([x1_test,x2_test])).transpose()
```

```
x_train = (np.vstack([x1_train,x2_train])).transpose()
```

```
return f_train, x_train, f_test, x_test
```

```
# function to calculate a rpf surrogate function value at a specific point xp
```

```
def rbf_point(n,x,ndv,xp,lam,rbfmodel,beta):
```

```
    import numpy as np
```

```
    import sys
```

```
    phi = np.zeros(shape=(1,n))
```

```
    # phi = np.zeros(n,n,float)
```

```

# calculate the gram matrix
for j in range(0,n):

    r = 0.0

    for k in range (0,ndv):

        r = r + (xp[k]-x[j][k])**2

    r = np.sqrt(r)

    if (rbfmodel == 1):

        phi[0][j] = np.exp(-beta*(r**2))

    else:

        if (rbfmodel == 2):

            phi[0][j] = np.sqrt(beta**2 + r**2)

        else:

            if (rbfmodel == 3):

                phi[0][j] = 1/np.sqrt(beta**2 + r**2)

            else:

                print ("invalid rbfmodel value")

                sys.exit(0)

```

```

# compute the rbf response

# ya = phi*lam

# phim = np.matrix(phi)

# lamm = np.matrix(lam)

yp = (np.asmatrix(phi)*lam).item()

return yp

```

```

print ("Finished computing rbf approximations")

```

Drilling Optimisation

This application enables you to explore the pressure drop of a twist drill

Surrogate model Pressure drop plot Leave one out cross validation

Corner radius (mm)

0.20

Design variable x2

45.00

0.05

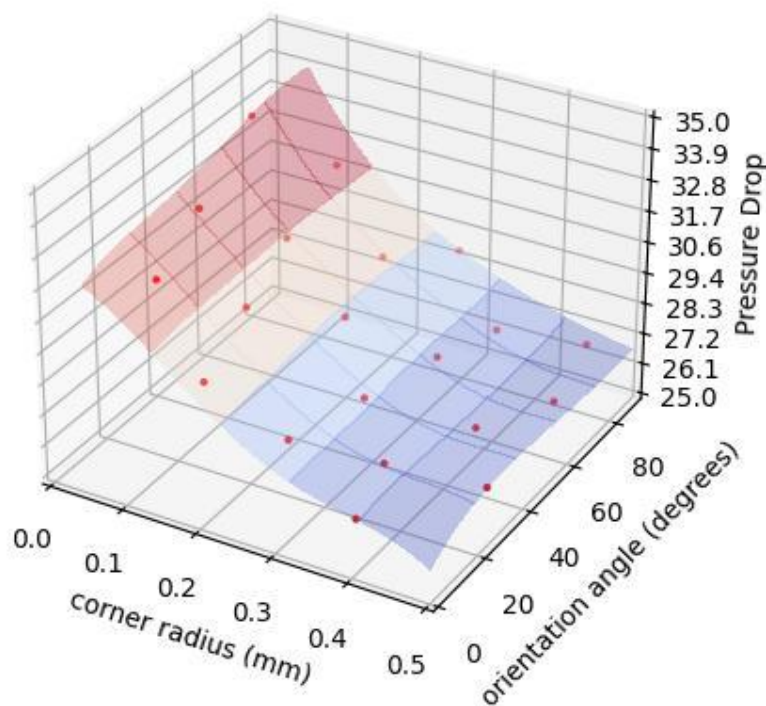
0.50

0.00

90.00

Pressure drop: 29.8 Pa

RBF approximation with $\beta=1.545$ and $n=20$



Executing_input_code directory: simple code which allows to specify and run Python code in a text box

```
# executing_input_code1.py
```

```
import streamlit as st
```

```
import sys
```

```

# Add heading and introductory text

st.title("Program for executing input code")

st.write("this application allows user to input a text box of python code and then run it")

st.markdown("---")


user_code = st.text_area("Input code to be run", value="Hello world",height=200)

# Execute the user's code

try:

    exec(user_code)

except Exception as e:

    st.error(f"An error occurred running the code: {str(e)}")

    st.stop()


""" Example of user_code that could be input into the st.text_area

import numpy as np

for i in range(10):

    print(f"i={i}")

print(np.sqrt(100))

"""

```

Program for executing input code

this application allows user to input a text box of python code and then run it

Input code to be run

```
print('Hello world')
```

Example of user_code that could be input into the st.text_area import numpy as np for i in range(10):

```
print(f"i={i}")
```

```
print(np.sqrt(100))
```

Forms directory: examples showing how to use forms, which are containers that visually groups other elements and widgets together, and contain a Submit button. When the form's Submit button is pressed, all widget values inside the form are processed by Streamlit in a batch.

forms.py

```
import streamlit as st
```

```
import pandas as pd
```

```
import numpy as np
```

```
def get_data():
```

```
    df = pd.DataFrame({
```

```
        "lat": np.random.randn(200) / 50 + 37.76,
```

```
        "lon": np.random.randn(200) / 50 + -122.4,
```

```
        "team": ['A','B']*100
```

```
    })
```

```
    return df
```

```
if st.button('Generate new points'):
```

```
    st.session_state.df = get_data()
```

```
if 'df' not in st.session_state:
```

```

st.session_state.df = get_data()

df = st.session_state.df

with st.form("my_form"):

    header = st.columns([1,2,2])

    header[0].subheader('Color')

    header[1].subheader('Opacity')

    header[2].subheader('Size')


    row1 = st.columns([1,2,2])

    colorA = row1[0].color_picker('Team A', '#0000FF')

    opacityA = row1[1].slider('A opacity', 20, 100, 50, label_visibility='hidden')

    sizeA = row1[2].slider('A size', 50, 200, 100, step=10, label_visibility='hidden')


    row2 = st.columns([1,2,2])

    colorB = row2[0].color_picker('Team B', '#FF0000')

    opacityB = row2[1].slider('B opacity', 20, 100, 50, label_visibility='hidden')

    sizeB = row2[2].slider('B size', 50, 200, 100, step=10, label_visibility='hidden')


    st.form_submit_button('Update map')


alphaA = int(opacityA*255/100)

alphaB = int(opacityB*255/100)


df['color'] = np.where(df.team=='A',colorA+f'{alphaA:02x}',colorB+f'{alphaB:02x}')

df['size'] = np.where(df.team=='A',sizeA, sizeB)


st.map(df, size='size', color='color')

```

Generate new points

Color

Opacity

Size

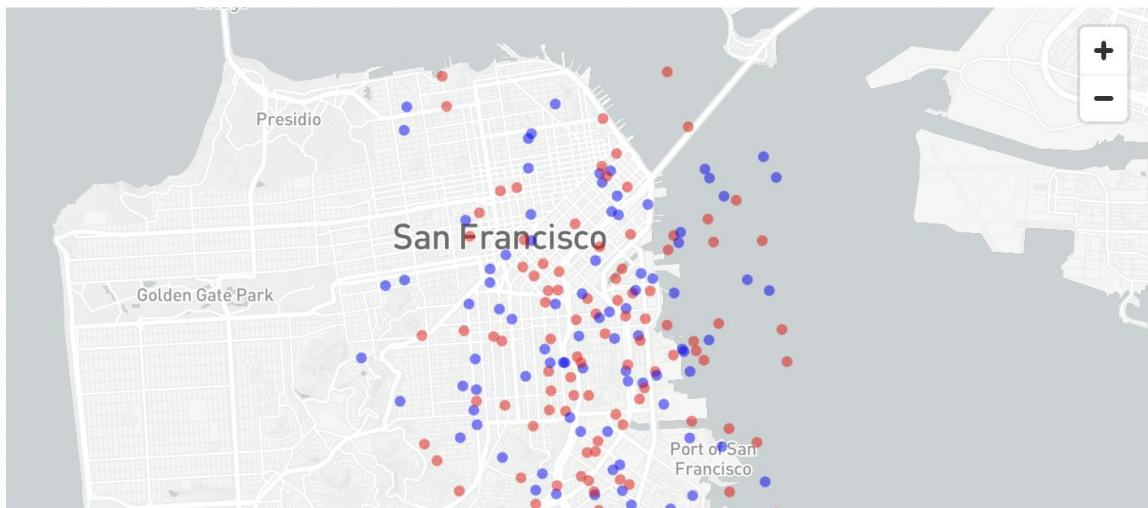
Team A



Team B



Update map



forms2.py

```
import streamlit as st
```

```
with st.form("my_form"):
```

```
    st.write("Inside the form")
```

```
    slider_val = st.slider("Form slider")
```

```
    checkbox_val = st.checkbox("Form checkbox")
```

```
# Every form must have a submit button.
```

```
submitted = st.form_submit_button("Submit")
```

```
if submitted:
```

```
    st.write("slider", slider_val, "checkbox", checkbox_val)
```

```
st.write("Outside the form")
```


Inside the form

Form slider

0 100

☐ Form checkbox

Submit

Outside the form

forms3.py

```
import streamlit as st

form = st.form("my_form")

form.slider("Inside the form")

st.slider("Outside the form")
```

Now add a submit button to the form:

```
form.form_submit_button("Submit")
```

Inside the form

0 100

Submit

Outside the form

0 100

heatinganddrillingRBF directory: a multipage application which displays surrogate modelling of a single objective problem (pressure drop from drilling) or a multi-objective problem (thermal resistance and pressure drop for a heat sink). Uses tabs to display options to calculate surrogate model at a design point or display the surface and curve plots. Two versions created – one which uses the simple but less interactive ‘matplotlib’ graphics library and a second which uses the more interactive ‘Plotly’ library.

matplotlibversion subdirectory

matplotlib/streamlit_app.py

```
import streamlit as st

drilling_page = st.Page("st_rbf_drilling.py", title="Single objective optimisation: drilling")

heatsink_page = st.Page("st_rbf_heatsink.py", title="Double objective optimisation: heat sink")

pg = st.navigation([drilling_page, heatsink_page])

pg.run()
```

matplotlib/st_rbf_heatsink.py

```
from mpl_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

import matplotlib.pyplot as plt

import streamlit as st


import numpy as np


from rbffunctions import *

# note can get rid of all warning flags by using import rbffunctions and
# making an explicit call to the function using 'rbffunctions.etc'


rbfmodel = 1 # Gaussian weights - can automate this
n = 30 # number of DoE points - can automate this
ndv = 2 #

x1min_actual = 0.4
x1max_actual = 1.0
x2min_actual = 0.4
x2max_actual = 1.0


#####

# reading data into relevant files

#####

def read_data():

    # Objective 1 - thermal resistance

    st.session_state.Xr_tr = np.loadtxt("heatsinkdata/Xr_tr.txt")

    st.session_state.Yr_tr = np.loadtxt("heatsinkdata/Yr_tr.txt")

    st.session_state.Zr_tr = np.loadtxt("heatsinkdata/Zr_tr.txt")
```

```

st.session_state.x_scaled_tr = np.loadtxt("heatsinkdata/x_scaled_tr.txt")

lam1 = np.loadtxt("heatsinkdata/lam_tr.txt")

st.session_state.lam_tr = np.transpose(np.asmatrix(lam1))

st.session_state.xopt_tr = np.loadtxt("heatsinkdata/xopt_tr.txt")

st.session_state.yopt_tr = np.loadtxt("heatsinkdata/yopt_tr.txt")

st.session_state.optval_tr = np.loadtxt("heatsinkdata/optval_tr.txt")

st.session_state.beta_tr = np.loadtxt("heatsinkdata/beta_tr.txt")

st.session_state.obj_tr = np.loadtxt("heatsinkdata/obj_tr.txt")


# Objective 2 - pressure difference

st.session_state.Xr_pd = np.loadtxt("heatsinkdata/Xr_pd.txt")

st.session_state.Yr_pd = np.loadtxt("heatsinkdata/Yr_pd.txt")

st.session_state.Zr_pd = np.loadtxt("heatsinkdata/Zr_pd.txt")

st.session_state.x_scaled_pd = np.loadtxt("heatsinkdata/x_scaled_pd.txt")

lam2 = np.loadtxt("heatsinkdata/lam_pd.txt")

st.session_state.lam_pd = np.transpose(np.asmatrix(lam2))

st.session_state.xopt_pd = np.loadtxt("heatsinkdata/xopt_pd.txt")

st.session_state.yopt_pd = np.loadtxt("heatsinkdata/yopt_pd.txt")

st.session_state.optval_pd = np.loadtxt("heatsinkdata/optval_pd.txt")

st.session_state.beta_pd = np.loadtxt("heatsinkdata/beta_pd.txt")

st.session_state.obj_pd = np.loadtxt("heatsinkdata/obj_pd.txt")


# Pareto points

st.session_state.trpareto = np.loadtxt("heatsinkdata/trpareto.txt")

st.session_state.pdpareto = np.loadtxt("heatsinkdata/pdpareto.txt")


# create the actual points

x1min_actual = 0.4

x1max_actual = 1.0

x2min_actual = 0.4

x2max_actual = 1.0

x_actual = np.loadtxt("heatsinkdata/x_scaled_tr.txt")

for i in range(0,n):

    x_actual[i][0] = x1min_actual + st.session_state.x_scaled_tr[i][0]*(x1max_actual-x1min_actual)

    x_actual[i][1] = x2min_actual + st.session_state.x_scaled_tr[i][1]*(x2max_actual-x2min_actual)

st.session_state.x_actual = x_actual


# scale the thermal resistance arrays

```

```

st.session_state.min_tr = 0.1618792

st.session_state.max_tr = 0.2472688

min_tr = st.session_state.min_tr

max_tr = st.session_state.max_tr

st.session_state.Xr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.Xr_tr

st.session_state.Yr = x2min_actual + (x2max_actual-x2min_actual)*st.session_state.Yr_tr

st.session_state.Zr = min_tr + (max_tr-min_tr)*st.session_state.Zr_tr

st.session_state.objactual_tr = min_tr + (max_tr-min_tr)*st.session_state.obj_tr

st.session_state.xoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_tr

st.session_state.yoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_tr

st.session_state.optactual_tr = min_tr + (max_tr-min_tr)*st.session_state.optval_tr


# scale the pressure drop arrays

st.session_state.min_pd = 12553.08

st.session_state.max_pd = 38679.57

min_pd = st.session_state.min_pd

max_pd = st.session_state.max_pd

st.session_state.Zp = min_pd + (max_pd-min_pd)*st.session_state.Zr_pd

st.session_state.objactual_pd = min_pd + (max_pd-min_pd)*st.session_state.obj_pd

st.session_state.xoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_pd

st.session_state.yoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_pd

st.session_state.optactual_pd = min_pd + (max_pd-min_pd)*st.session_state.optval_pd


# calculate dimensionless value of thermal resistance at specified point
def tr_f(x):

    return rbf_point(n,st.session_state.x_scaled_tr,ndv,x,st.session_state.lam_tr,rbfmodel,st.session_state.beta_tr)


# calculate dimensionless value of pressure difference at specified point
def pd_f(x):

    return rbf_point(n,st.session_state.x_scaled_pd,ndv,x,st.session_state.lam_pd,rbfmodel,st.session_state.beta_pd)


# plot thermal resistance RBF surrogate model
def tr_plot():

    beta_tr = st.session_state.beta_tr

    Xr = st.session_state.Xr

    Yr = st.session_state.Yr

    Zr = st.session_state.Zr

```

```

x_actual = st.session_state.x_actual

objactual_tr = st.session_state.objactual_tr

xoptactual_tr = st.session_state.xoptactual_tr

yoptactual_tr = st.session_state.yoptactual_tr

optactual_tr = st.session_state.optactual_tr


# Plot out RBF approximation

plt.figure()

strbeta = f'{beta_tr:.2f}'

plt.suptitle('RBF approximation of thermal resistance with beta='+strbeta+' and n='+str(n),fontsize=10)

#ax = fig.gca(projection='3d')

ax=plt.axes(projection='3d')

surf = ax.plot_surface(Xr, Yr, Zr, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,

                        linewidth=0, antialiased=False)

# ax.set_zlim(-0.1, 1.1)

ax.set_xlim(x1min_actual, x1max_actual)

ax.set_ylim(x2min_actual, x2max_actual)

ax.zaxis.set_major_locator(LinearLocator(10))

ax.zaxis.set_major_formatter(FormatStrFormatter('%0.02f'))

#fig.colorbar(surf, shrink=0.5, aspect=5)


# plot out the scatter points

ax.scatter(x_actual[:,0],x_actual[:,1],objactual_tr, c='r', marker='o',s=4)

ax.scatter(xoptactual_tr,yoptactual_tr,optactual_tr, c='k', marker='o',s=16) # plot optimum point

ax.set_xlabel('x1')

ax.set_ylabel('x2')

ax.set_zlabel('Thermal Resistance')

plt.savefig('tf_rbfheatsink.jpg')

st.pyplot(plt)


# plot pressure drop RBF surrogate model

def pd_plot():

    beta_pd = st.session_state.beta_pd

    Xr = st.session_state.Xr

    Yr = st.session_state.Yr

    Zp = st.session_state.Zp

    x_actual = st.session_state.x_actual

```

```

objactual_pd = st.session_state.objactual_pd
xoptactual_pd = st.session_state.xoptactual_pd
yoptactual_pd = st.session_state.yoptactual_pd
optactual_pd = st.session_state.optactual_pd

# Plot out RBF approximation

plt.figure()

plt.suptitle('RBF approximation of pressure drop with beta='+str(beta_pd)+' and n='+str(n),fontsize=10)

#ax = fig.gca(projection='3d')

ax=plt.axes(projection='3d')

surf = ax.plot_surface(Xr, Yr, Zp, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)

# ax.set_zlim(-0.1, 1.1)

ax.set_xlim(x1min_actual, x1max_actual)

ax.set_ylim(x2min_actual, x2max_actual)

ax.zaxis.set_major_locator(LinearLocator(10))

ax.zaxis.set_major_formatter(FormatStrFormatter('%0f'))

#fig.colorbar(surf, shrink=0.5, aspect=5)

# plot out the scatter points

ax.scatter(x_actual[:,0],x_actual[:,1],objactual_pd, c='r', marker='o',s=4)

ax.scatter(xoptactual_pd,yoptactual_pd,optactual_pd, c='k', marker='o',s=16) # plot optimum point

ax.set_ylabel('x2')

ax.set_xlabel('x1')

ax.set_zlabel('Pressure Drop')

plt.savefig('pd_rbfheatsink.jpg')

st.pyplot(plt)

# Plot Pareto Front

def pareto_plot():

    trpareto = st.session_state.trpareto

    pdpareto = st.session_state.pdpareto

# Plot out pareto set

plt.figure()

# plt.ion()

plt.xlabel('thermal resistance')

```

```

plt.ylabel('pressure drop, Pa')

plt.title('Pareto front for thermal resistance vs pressure drop',fontsize=10)

plt.plot(trpareto,pdpareto)

# plt.show()

plt.savefig('pareto_rbfheatsink.jpg')

st.pyplot(plt)


# calculate surrogate models of thermal resistance and pressure drop
def calc_surrogates(x1,x2):

    xp = np.zeros(ndv)

    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)

    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)

    xp[0] = x1_scaled

    xp[1] = x2_scaled

    min_tr = st.session_state.min_tr

    max_tr = st.session_state.max_tr

    min_pd = st.session_state.min_pd

    max_pd = st.session_state.max_pd

    tr_val = min_tr + (max_tr - min_tr)*tr_f(xp)

    pd_val = min_pd + (max_pd - min_pd)*pd_f(xp)

    return tr_val,pd_val


# Add heading and introductory text
# st.set_page_config(layout='wide')


st.title("Heat Sink Optimisation")

st.write("This application enables you to explore the thermal resistance and pressure drop of a heat sink")

st.markdown("----")


# read in data for calculations only at the beginning of session
if 'Xr_tr' not in st.session_state:

    read_data()


tab1, tab2, tab3, tab4 = st.tabs(["Surrogate model", "Thermal Resistance plot", "Pressure Drop plot", "Pareto Front"])

with tab1:

```

```

# Create the input sliders

row1 = st.columns([1,1])

default_value = 0.6

st.session_state.x1 = default_value

x1 = row1[0].slider("Design variable x1",0.4,1.0,default_value)

st.session_state.x1 = x1


st.session_state.x2 = default_value

x2 = row1[1].slider("Design variable x2",0.4,1.0,default_value)

st.session_state.x2 = x2


# Calculate surrogate models

tr_val,pd_val = calc_surrogates(x1,x2)

col1, col2 = st.columns(2)

with col1:

    st.write(f'Thermal resistance: {tr_val:.3f}')

with col2:

    st.write(f'Pressure drop: {pd_val:.1f} Pa')


with tab2:

    tr_plot()


with tab3:

    pd_plot()


with tab4:

    pareto_plot()


# matplotlib/st_rbf_drilling.py

from mpl_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

import matplotlib.pyplot as plt

import streamlit as st

import numpy as np


from rbffunctions import *

```



```

rbfmodel = 1    # Gaussian weights

n = 20

ndv = 2

x1min_actual = 0.05

x1max_actual = 0.5

x2min_actual = 0.0

x2max_actual = 90.0

max_pd = 33.56574

min_pd = 26.7848

nbetas = 101


#####

# reading data into relevant files

#####

def read_data():

    st.session_state.Xr = np.loadtxt("drillingdata/Xr.txt")

    st.session_state.Yr = np.loadtxt("drillingdata/Yr.txt")

    st.session_state.Zr_rbf = np.loadtxt("drillingdata/Zr_rbf.txt")

    st.session_state.x_scaled = np.loadtxt("drillingdata/x_scaled.txt")

    st.session_state.x = np.loadtxt("drillingdata/x.txt")

    st.session_state.pressure_drop = np.loadtxt("drillingdata/pressure_drop.txt")

    st.session_state.beta_min = np.loadtxt("drillingdata/beta_min.txt")

    st.session_state.beta_array = np.loadtxt("drillingdata/beta_array.txt")

    st.session_state.RMSE_array = np.loadtxt("drillingdata/RMSE_array.txt")

    lam1 = np.loadtxt("drillingdata/lam.txt")

    st.session_state.lam = np.transpose(np.asmatrix(lam1))

    st.session_state.xopt = np.loadtxt("drillingdata/xopt.txt")

    st.session_state.yopt = np.loadtxt("drillingdata/yopt.txt")

    st.session_state.optval = np.loadtxt("drillingdata/optval.txt")


def beta_plot():

    RMSE_array = st.session_state.RMSE_array

    beta_array = st.session_state.beta_array

    plt.figure()

    plt.xlabel('beta')

    plt.ylabel('RMSE')

```

```

plt.title('Plot out RMSE vs Beta for Leave One Out Cross Validation')

plt.plot(beta_array,RMSE_array)

st.pyplot(plt)

# plot out the RBF surrogate model of pressure drop
def pd_plot():

    # Plot out RBF approximation

    plt.figure()

    beta = st.session_state.beta_min

    Xr = st.session_state.Xr

    Yr = st.session_state.Yr

    Zr_rbf = st.session_state.Zr_rbf

    x = st.session_state.x

    pressure_drop = st.session_state.pressure_drop

    xopt = st.session_state.xopt

    yopt = st.session_state.yopt

    optval = st.session_state.optval

    plt.suptitle('RBF approximation with beta='+str(beta)+' and n='+str(n),fontsize=10)

    #ax = fig.gca(projection='3d')

    ax=plt.axes(projection='3d')

    surf = ax.plot_surface(Xr, Yr, Zr_rbf, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,

        linewidth=0, antialiased=False)

    ax.set_zlim(25, 35)

    ax.set_xlim(0, x1max_actual)

    ax.set_ylim(0, x2max_actual)

    ax.zaxis.set_major_locator(LinearLocator(10))

    ax.zaxis.set_major_formatter(FormatStrFormatter('%1f'))

    #fig.colorbar(surf, shrink=0.5, aspect=5)

    # plot out the scatter points

    ax.scatter(x[:,0],x[:,1],pressure_drop, c='r', marker='o',s=4)

    ax.scatter(xopt,yopt,optval, c='k', marker='o',s=16) # plot optimum point

    ax.set_xlabel('corner radius (mm)')

    ax.set_ylabel('orientation angle (degrees)')

    ax.set_zlabel('Pressure Drop')

    plt.savefig('rbf.jpg')

    st.pyplot(plt)

```

```

# calculate pressure drop at specified point
def pd_f(x):
    return rbf_point(n,st.session_state.x_scaled,ndv,x,st.session_state.lam,rbfmodel,st.session_state.beta_min)

# calculate surrogate models of thermal resistance and pressure drop
def calc_surrogates(x1,x2):

    xp = np.zeros(ndv)

    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)
    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)
    xp[0] = x1_scaled
    xp[1] = x2_scaled
    pd_val = pd_f(xp)
    return pd_val

st.title("Drilling Optimisation")

st.write("This application enables you to explore the pressure drop of a twist drill")

st.markdown("---")

checking_password = 0

if (checking_password != 0):
    if 'pwdcheck' not in st.session_state:
        st.session_state['pwdcheck'] = 0
        password_guess = st.text_input('What is the password?')
        if password_guess != st.secrets["password"]:
            st.stop()

# read in data for calculations only at the beginning of session
if 'Xr' not in st.session_state:
    read_data()

tab1, tab2, tab3 = st.tabs(["Surrogate model", "Pressure drop plot", "Leave one out cross validation"])

with tab1:

    # Create the input sliders
    row1 = st.columns([1,1])

```

```

default_value = 0.2

x1 = row1[0].slider("Corner radius (mm)",x1min_actual,x1max_actual,0.2)

x2 = row1[1].slider("Orientation angle (degrees)",x2min_actual,x2max_actual,45.0)

pd_val = calc_surrogates(x1,x2)

st.write(f'Pressure drop: {pd_val:.1f} Pa')

```

with tab2:

```
pd_plot()
```

with tab3:

```
beta_plot()
```

rbffunctions.py

function to calculate weights for rbfs

```
def rbfweights(n,x,ndv,f,rbfmodel,beta):
```

```
    import numpy as np
```

```
    import sys
```

```
    phi = np.zeros(shape=(n,n))
```

```
    # phi = np.matrix(phi1)
```

calculate the gram matrix

```
for i in range(0,n):
```

```
    for j in range(0,n):
```

```
        r = 0.0
```

```
        for k in range (0,ndv):
```

```
            r = r + (x[i][k]-x[j][k])**2
```

```
        r = np.sqrt(r)
```

```
        if (rbfmodel == 1):
```

```
            phi[i][j] = np.exp(-beta*(r**2))
```

```
        else:
```

```
            if (rbfmodel == 2):
```

```
                phi[i][j] = np.sqrt(beta**2 + r**2)
```

```
            else:
```

```
                if (rbfmodel == 3):
```

```
                    phi[i][j] = 1/np.sqrt(beta**2 + r**2)
```

```
            else:
```

```

        print ("invalid rbfmodel value")

        sys.exit(0)

# compute the weights. need to transform f into an array and transpose it
farray = np.asarray(f)
fm = np.asmatrix(farray)
fmt = fm.transpose()
lam = np.linalg.solve(phi, fmt)

return lam

print ("Finished computing rbf weights in rbfweights")

# function to calculate rbf approximation at specified plan design points xa
def rbf(n,x,ndv,xa,na,lam,rbfmodel,beta):

    import numpy as np
    import sys

    phi = np.zeros(shape=(na*na,n))

    # phi = np.zeros(n,n,float)

    # calculate the gram matrix
    for i in range(0,na*na):
        for j in range(0,n):
            r = 0.0

            for k in range (0,ndv):
                r = r + (xa[i][k]-x[j][k])**2

            r = np.sqrt(r)

            if (rbfmodel == 1):
                phi[i][j] = np.exp(-beta*(r**2))
            else:
                if (rbfmodel == 2):
                    phi[i][j] = np.sqrt(beta**2 + r**2)
                else:
                    if (rbfmodel == 3):
                        phi[i][j] = 1/np.sqrt(beta**2 + r**2)
                    else:
                        print ("invalid rbfmodel value")
                        sys.exit(0)

```

```

# compute the rbf response

# ya = phi*lam

# phim = np.matrix(phi)

# lamm = np.matrix(lam)

ya = np.asmatrix(phi)*lam

return ya


print ("Finished computing rbf approximations")


# define function

def yval(x,y):

    yval = x**2 + y**2

    return yval


# define six hump camel function

def sixhumpcamel(x1,x2):

    X1 = 4*x1-2;

    X2 = 2*x2-1;

    fval = (4 - 2.1*(X1**2) + (X1**4)/3)*X1**2 + X1*X2 + (4*(X2**2) - 4)*(X2**2)

    return fval


# generate test dataset from ntest random numbers between 1 and n

def generate_test(ntest,n):

    import random

    testindex = []

    testflag = []

    for i in range(n):

        testflag.append(0)

    num = 0

    for i in range(100):

        if (num < ntest):

            k = random.randint(0,n-1)

            if (testflag[k] == 0):

                testindex.append(k)

                testflag[k] = 1

```

```

        num = num + 1

    else:

        break

return testindex, testflag

# function to calculate rbf approximation at specified ntest test points x_test
def rbf_testeval(ntrain,x_train,ndv,x_test,ntest,lam,rbfmodel,beta):

    import numpy as np
    import sys

    phi = np.zeros(shape=(ntest,ntrain))

    # phi = np.zeros(n,n,float)

    # calculate the gram matrix
    for i in range(0,ntest):

        for j in range(0,ntrain):

            r = 0.0

            for k in range (0,ndv):

                r = r + (x_test[i][k]-x_train[j][k])**2

            r = np.sqrt(r)

            if (rbfmodel == 1):

                phi[i][j] = np.exp(-beta*(r**2))

            else:

                if (rbfmodel == 2):

                    phi[i][j] = np.sqrt(beta**2 + r**2)

                else:

                    if (rbfmodel == 3):

                        phi[i][j] = 1/np.sqrt(beta**2 + r**2)

                    else:

                        print ("invalid rbfmodel value")

                        sys.exit(0)

    y_test = np.asmatrix(phi)*lam

    return y_test

```

```

print ("Finished computing rbf approximations at test data points")

# function to calculate training and testing data for leave one out validation
def leaveoneout(i,x,f,n,ndv)

    import numpy as np

    # create leave one out training and test data sets with test data on index i

    f_train = []
    f_test = []
    x1_train = []
    x1_test = []
    x2_train = []
    x2_test = []

    # create leave one out training and test data sets
    for j in range(0,n):
        if (j < i):
            f_train.append(f[j])
            x1_train.append(x[j][0])
            x2_train.append(x[j][1])
        else:
            if (j == i):
                f_test.append(f[i])
                x1_test.append(x[i][0])
                x2_test.append(x[i][1])
            else:
                f_train.append(f[j])
                x1_train.append(x[j][0])
                x2_train.append(x[j][1])

    # put training and test data into arrays
    x_test = (np.vstack([x1_test,x2_test])).transpose()
    x_train = (np.vstack([x1_train,x2_train])).transpose()

    return f_train, x_train, f_test, x_test

# function to calculate a rpf surrogate function value at a specific point xp
def rbf_point(n,x,ndv,xp,lam,rbfmodel,beta):

```



```

import numpy as np
import sys

phi = np.zeros(shape=(1,n))
# phi = np.zeros(n,n,float)

# calculate the gram matrix
for j in range(0,n):

    r = 0.0

    for k in range (0,ndv):

        r = r + (xp[k]-x[j])[k]**2

    r = np.sqrt(r)

    if (rbfmodel == 1):

        phi[0][j] = np.exp(-beta*(r**2))

    else:

        if (rbfmodel == 2):

            phi[0][j] = np.sqrt(beta**2 + r**2)

        else:

            if (rbfmodel == 3):

                phi[0][j] = 1/np.sqrt(beta**2 + r**2)

            else:

                print ("invalid rbfmodel value")

                sys.exit(0)

# compute the rbf response

# ya = phi*lam

# phim = np.matrix(phi)

# lamm = np.matrix(lam)

yp = (np.asmatrix(phi)*lam).item()

return yp

print ("Finished computing rbf approximations")

```

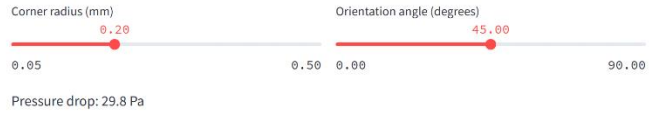
Single objective optimisation: drilling

Double objective optimisation: heat sink

Drilling Optimisation

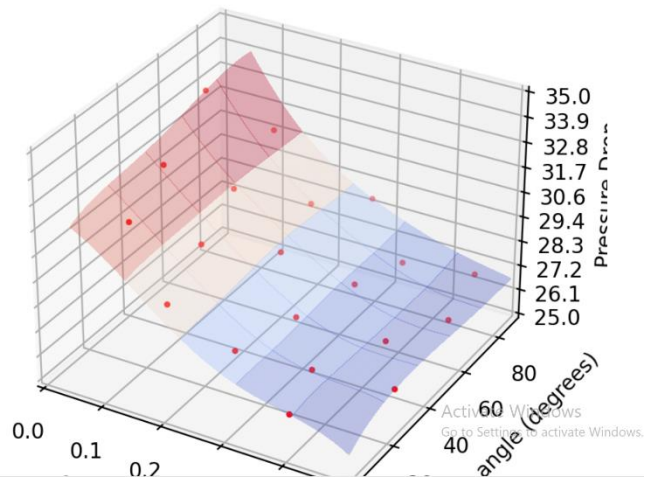
This application enables you to explore the pressure drop of a twist drill

Surrogate model Pressure drop plot Leave one out cross validation



Surrogate model Pressure drop plot Leave one out cross validation

RBF approximation with $\beta=1.545$ and $n=20$

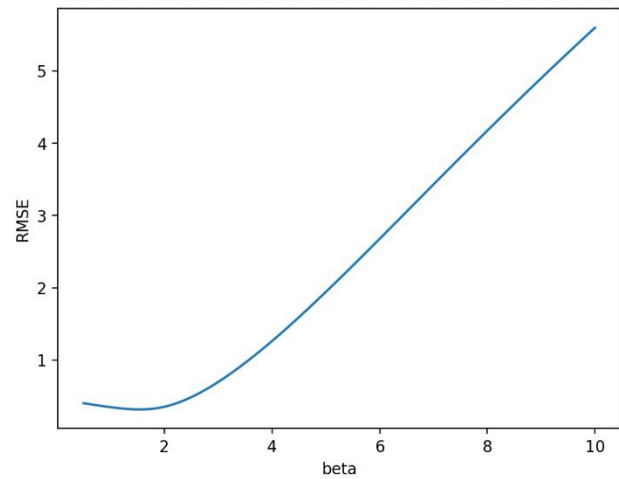


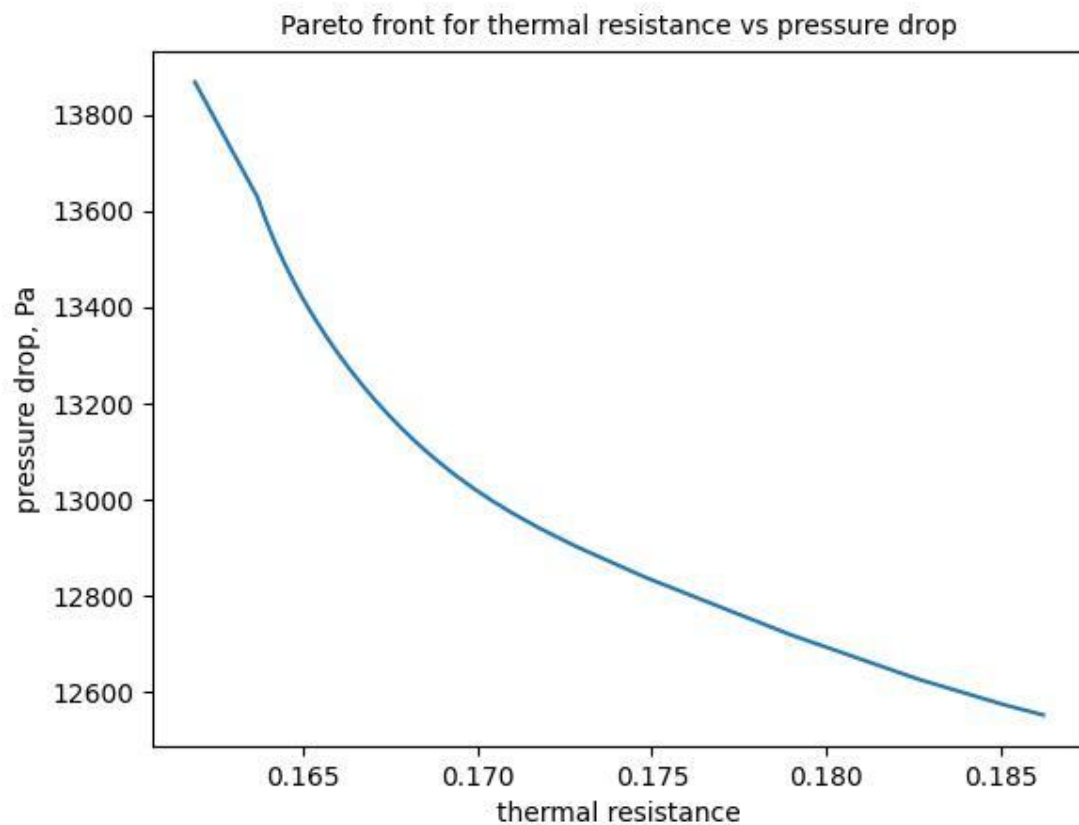
Single objective optimisation: drilling

Double objective optimisation: heat sink

Surrogate model Pressure drop plot Leave one out cross validation

Plot out RMSE vs Beta for Leave One Out Cross Validation





plotlyversion subdirectory

streamlit_app.py

import streamlit as st

drilling_page = st.Page("st_combined_plotly_rbf_drilling.py", title="Single objective optimisation: drilling")

heatsink_page = st.Page("st_plotly_rbf_heatsink.py", title="Double objective optimisation: heat sink")

pg = st.navigation([drilling_page, heatsink_page])

pg.run()

st_combined_plotly_rbf_drilling.py

program to calculate rbf response surface given DoE points and

calculating the single and multi-objective optimisation from these responses

from mpl_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

import matplotlib.pyplot as plt

import streamlit as st

```

import numpy as np

from rbffunctions import *

import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

rbfmodel = 1    # Gaussian weights

n = 20

ndv = 2

x1min_actual = 0.05
x1max_actual = 0.5
x2min_actual = 0.0
x2max_actual = 90.0

max_pd = 33.56574
min_pd = 26.7848

nbetas = 101

#####

# reading data into relevant files

#####

def read_data():

    st.session_state.Xr = np.loadtxt("drillingdata/Xr.txt")

    st.session_state.Yr = np.loadtxt("drillingdata/Yr.txt")

    st.session_state.Zr_rbf = np.loadtxt("drillingdata/Zr_rbf.txt")

    st.session_state.x_scaledd = np.loadtxt("drillingdata/x_scaled.txt")

    st.session_state.x_actuald = np.loadtxt("drillingdata/x.txt")

    st.session_state.pressure_drop = np.loadtxt("drillingdata/pressure_drop.txt")

    st.session_state.beta_mind = np.loadtxt("drillingdata/beta_min.txt")

    st.session_state.beta_array = np.loadtxt("drillingdata/beta_array.txt")

    st.session_state.RMSE_array = np.loadtxt("drillingdata/RMSE_array.txt")

    lam1 = np.loadtxt("drillingdata/lam.txt")

    st.session_state.lamd = np.transpose(np.asmatrix(lam1))

    st.session_state.xopt = np.loadtxt("drillingdata/xopt.txt")

    st.session_state.yopt = np.loadtxt("drillingdata/yopt.txt")

    st.session_state.optval = np.loadtxt("drillingdata/optval.txt")

```

```

st.session_state.xd = st.session_state.x_actuald[:,0]
st.session_state.yd = st.session_state.x_actuald[:,1]
st.session_state.zd = st.session_state.pressure_drop

dfscatterd = pd.DataFrame(
    {'x': st.session_state.xd,
     'y': st.session_state.yd,
     'z': st.session_state.zd,
    })

st.session_state.dfscatterd = dfscatterd

st.session_state.title_text = 'RBF approximation with beta='+str(st.session_state.beta_mind)+' and n='+str(n)

# process the data into a form that can be used with the go.Mesh3d function
xmesh = []
ymesh = []
zmesh = []

for i in range(41):
    for j in range(41):
        xmesh.append(st.session_state.Xr[i,j])
        ymesh.append(st.session_state.Yr[i,j])
        zmesh.append(st.session_state.Zr_rbf[i,j])

st.session_state.xmeshd = xmesh
st.session_state.ymeshd = ymesh
st.session_state.zmeshd = zmesh

st.session_state.dfd = pd.DataFrame(
    {'beta': st.session_state.beta_array,
     'RMSE': st.session_state.RMSE_array,
    })

# end read_data

def beta_plot():

    figbeta = px.line(st.session_state.dfd, x='beta', y='RMSE', markers=True, title='Plot of RMSE vs Beta for Leave One Out Cross Validation')
    st.plotly_chart(figbeta)

```

```

# plot out the RBF surrogate model of pressure drop
def pd_plot():

    xd = st.session_state.xmeshd
    yd = st.session_state.ymeshd
    zd = st.session_state.zmeshd
    trace2 = go.Mesh3d(x=xd,
                       y=yd,
                       z=zd,
                       opacity=0.5,
                       color='rgba(244,22,100,0.6)')

    xscatter = st.session_state.x_actuald[:,0]
    yscatter = st.session_state.x_actuald[:,1]
    zscatter = st.session_state.pressure_drop
    trace3 = go.Scatter3d(x=xscatter, y=yscatter, z=zscatter, mode='markers')
    data2 = [trace2, trace3]
    layout = go.Layout(title=st.session_state.title_text,
                       title_font=dict(size=20,
                                       color='blue',
                                       family='Arial'),
                       title_x=0.25,
                       title_y=0.85)

    fig2 = go.Figure(data=data2, layout=layout)
    fig2.update_scenes(xaxis=dict(title="corner radius (mm)",nticks=6, range=[0.0,0.5]),
                      yaxis=dict(title="angle (degrees)",nticks=10, range=[0.0,90.0]),
                      zaxis=dict(title="Pressure Drop (Pa)"))

    st.plotly_chart(fig2)

# end pd_plot

# calculate pressure drop at specified point
def pd_f(x):
    return rbf_point(n,st.session_state.x_scaledd,ndv,x,st.session_state.lamd,rbfmodel,st.session_state.beta_mind)

# calculate surrogate models of thermal resistance and pressure drop

```

```
def calc_surrogates(x1,x2):
```

```
    xp = np.zeros(ndv)
```

```
    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)
```

```
    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)
```

```
    xp[0] = x1_scaled
```

```
    xp[1] = x2_scaled
```

```
    pd_val = pd_f(xp)
```

```
    return pd_val
```

```
st.title("Drilling Optimisation")
```

```
st.write("This application enables you to explore the pressure drop of a twist drill")
```

```
st.markdown("---")
```

```
checking_password = 0
```

```
if (checking_password != 0):
```

```
    if 'pwdcheck' not in st.session_state:
```

```
        st.session_state['pwdcheck'] = 0
```

```
        password_guess = st.text_input('What is the password?')
```

```
        if password_guess != st.secrets["password"]:
```

```
            st.stop()
```

```
# read in data for calculations only at the beginning of session
```

```
if 'Xr' not in st.session_state:
```

```
    read_data()
```

```
tab1, tab2, tab3 = st.tabs(["Surrogate model", "Pressure drop plot", "Leave one out cross validation"])
```

```
with tab1:
```

```
    # Create the input sliders
```

```
    row1 = st.columns([1,1])
```

```
    default_value = 0.2
```

```
    x1 = row1[0].slider("Corner radius (mm)",x1min_actual,x1max_actual,0.2)
```

```
    x2 = row1[1].slider("Orientation angle (degrees)",x2min_actual,x2max_actual,45.0)
```

```
    pd_val = calc_surrogates(x1,x2)
```

```
    st.write(f'Pressure drop: {pd_val:.1f} Pa')
```

```
with tab2:
```

```
    pd_plot()
```

```
with tab3:
```

```
    beta_plot()
```

```
# program to calculate rbf response surface given DoE points and responses
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from matplotlib import cm
```

```
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```
import matplotlib.pyplot as plt
```

```
import streamlit as st
```

```
import numpy as np
```

```
from rbffunctions import *
```

```
import plotly.express as px
```

```
import plotly.graph_objects as go
```

```
import pandas as pd
```

```
# note can get rid of all warning flags by using import rbffunctions and
```

```
# making an explicit call to the function using 'rbffunctions.etc'
```

```
rbfmodel = 1    # Gaussian weights - can automate this
```

```
n = 30    # number of DoE points - can automate this
```

```
ndv = 2    #
```

```
x1min_actual = 0.4
```

```
x1max_actual = 1.0
```

```
x2min_actual = 0.4
```

```
x2max_actual = 1.0
```

```
#####
```

```
# reading data into relevant files
```

```
#####
```

```
def read_data():
```


Objective 1 - thermal resistance

```
st.session_state.Xr_tr = np.loadtxt("heatsinkdata/Xr_tr.txt")
st.session_state.Yr_tr = np.loadtxt("heatsinkdata/Yr_tr.txt")
st.session_state.Zr_tr = np.loadtxt("heatsinkdata/Zr_tr.txt")
st.session_state.x_scaled_tr = np.loadtxt("heatsinkdata/x_scaled_tr.txt")
lam1 = np.loadtxt("heatsinkdata/lam_tr.txt")
st.session_state.lam_tr = np.transpose(np.asmatrix(lam1))
st.session_state.xopt_tr = np.loadtxt("heatsinkdata/xopt_tr.txt")
st.session_state.yopt_tr = np.loadtxt("heatsinkdata/yopt_tr.txt")
st.session_state.optval_tr = np.loadtxt("heatsinkdata/optval_tr.txt")
st.session_state.beta_tr = np.loadtxt("heatsinkdata/beta_tr.txt")
st.session_state.obj_tr = np.loadtxt("heatsinkdata/obj_tr.txt")
```

Objective 2 - pressure difference

```
st.session_state.Xr_pd = np.loadtxt("heatsinkdata/Xr_pd.txt")
st.session_state.Yr_pd = np.loadtxt("heatsinkdata/Yr_pd.txt")
st.session_state.Zr_pd = np.loadtxt("heatsinkdata/Zr_pd.txt")
st.session_state.x_scaled_pd = np.loadtxt("heatsinkdata/x_scaled_pd.txt")
lam2 = np.loadtxt("heatsinkdata/lam_pd.txt")
st.session_state.lam_pd = np.transpose(np.asmatrix(lam2))
st.session_state.xopt_pd = np.loadtxt("heatsinkdata/xopt_pd.txt")
st.session_state.yopt_pd = np.loadtxt("heatsinkdata/yopt_pd.txt")
st.session_state.optval_pd = np.loadtxt("heatsinkdata/optval_pd.txt")
st.session_state.beta_pd = np.loadtxt("heatsinkdata/beta_pd.txt")
st.session_state.obj_pd = np.loadtxt("heatsinkdata/obj_pd.txt")
```

Pareto points

```
st.session_state.trpareto = np.loadtxt("heatsinkdata/trpareto.txt")
st.session_state.pdpareto = np.loadtxt("heatsinkdata/pdpareto.txt")
st.session_state.df = pd.DataFrame(
    {'Thermal resistance': st.session_state.trpareto,
     'Pressure drop': st.session_state.pdpareto,
    })
```

create the actual points

```
x1min_actual = 0.4
x1max_actual = 1.0
```

```

x2min_actual = 0.4

x2max_actual = 1.0

x_actual = np.loadtxt("heatsinkdata/x_scaled_tr.txt")

for i in range(0,n):

    x_actual[i][0] = x1min_actual + st.session_state.x_scaled_tr[i][0]*(x1max_actual-x1min_actual)

    x_actual[i][1] = x2min_actual + st.session_state.x_scaled_tr[i][1]*(x2max_actual-x2min_actual)

st.session_state.x_actual = x_actual


# scale the thermal resistance arrays

st.session_state.min_tr = 0.1618792

st.session_state.max_tr = 0.2472688

min_tr = st.session_state.min_tr

max_tr = st.session_state.max_tr

st.session_state.Xr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.Xr_tr

st.session_state.Yr = x2min_actual + (x2max_actual-x2min_actual)*st.session_state.Yr_tr

st.session_state.Zr = min_tr + (max_tr-min_tr)*st.session_state.Zr_tr

st.session_state.objactual_tr = min_tr + (max_tr-min_tr)*st.session_state.obj_tr

st.session_state.xoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_tr

st.session_state.yoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_tr

st.session_state.optactual_tr = min_tr + (max_tr-min_tr)*st.session_state.optval_tr


# scale the pressure drop arrays

st.session_state.min_pd = 12553.08

st.session_state.max_pd = 38679.57

min_pd = st.session_state.min_pd

max_pd = st.session_state.max_pd

st.session_state.Zp = min_pd + (max_pd-min_pd)*st.session_state.Zr_pd

st.session_state.objactual_pd = min_pd + (max_pd-min_pd)*st.session_state.obj_pd

st.session_state.xoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_pd

st.session_state.yoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_pd

st.session_state.optactual_pd = min_pd + (max_pd-min_pd)*st.session_state.optval_pd


st.session_state.x = st.session_state.x_actual[:,0]

st.session_state.y = st.session_state.x_actual[:,1]

st.session_state.z = st.session_state.objactual_tr

dfscatter = pd.DataFrame(

    {'x': st.session_state.x,

     'y': st.session_state.y,

```

```

        'z': st.session_state.z,
    })

    st.session_state.dfscatter = dfscatter

    # st.session_state.title_text = 'RBF approximation with beta='+str(st.session_state.beta_min)+' and n='+str(n)

    # process the data into a form that can be used with the go.Mesh3d function

    xmesh = []
    ymesh = []
    zmesh = []

    for i in range(41):
        for j in range(41):
            xmesh.append(st.session_state.Xr[i,j])
            ymesh.append(st.session_state.Yr[i,j])
            zmesh.append(st.session_state.Zr[i,j])

    st.session_state.xmesh = xmesh
    st.session_state.ymesh = ymesh
    st.session_state.zmesh = zmesh

    # calculate dimensionless value of thermal resistance at specified point
    def tr_f(x):
        return rbf_point(n,st.session_state.x_scaled_tr,ndv,x,st.session_state.lam_tr,rbfmodel,st.session_state.beta_tr)

    # calculate dimensionless value of pressure difference at specified point
    def pd_f(x):
        return rbf_point(n,st.session_state.x_scaled_pd,ndv,x,st.session_state.lam_pd,rbfmodel,st.session_state.beta_pd)

    # plot thermal resistance RBF surrogate model
    def tr_plot():

        x = st.session_state.xmesh
        y = st.session_state.ymesh
        z = st.session_state.zmesh

        trace2 = go.Mesh3d(x=x,
                            y=y,
                            z=z,

```

```

        opacity=0.5,
        color='rgba(244,22,100,0.6)')

xscatter = st.session_state.x_actual[:,0]
yscatter = st.session_state.x_actual[:,1]
zscatter = st.session_state.objactual_tr

trace3 = go.Scatter3d(x=xscatter, y=yscatter, z=zscatter, mode='markers')
data2 = [trace2, trace3]
layout = go.Layout(title="Thermal Resistance of Heat Sink",
                    title_font=dict(size=20,
                                    color='blue',
                                    family='Arial'),
                    title_x=0.25,
                    title_y=0.85)

fig2 = go.Figure(data=data2, layout=layout)
fig2.update_scenes(xaxis=dict(title="x1",nticks=7, range=[0.4,1.0]),
                  yaxis=dict(title="x2",nticks=7, range=[0.4,1.0]),
                  zaxis=dict(title="Thermal Resistance",nticks=5, range=[0.1,0.3]))

st.plotly_chart(fig2)

# end tr_plot

# Plot Pareto Front
def pareto_plot():

    figpareto = px.line(st.session_state.df, x='Thermal resistance', y='Pressure drop', markers=True, title='Pareto front for thermal resistance
and pressure drop')

    st.plotly_chart(figpareto)

# calculate surrogate models of thermal resistance and pressure drop
def calc_surrogates(x1,x2):

    xp = np.zeros(ndv)

    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)

    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)

```

```

xp[0] = x1_scaled
xp[1] = x2_scaled

min_tr = st.session_state.min_tr
max_tr = st.session_state.max_tr
min_pd = st.session_state.min_pd
max_pd = st.session_state.max_pd

tr_val = min_tr + (max_tr - min_tr)*tr_f(xp)
pd_val = min_pd + (max_pd - min_pd)*pd_f(xp)

return tr_val,pd_val

```

```

# Add heading and introductory text

# st.set_page_config(layout='wide')

```

```

st.title("Heat Sink Optimisation")

st.write("This application enables you to explore the thermal resistance and pressure drop of a heat sink")

st.markdown("----")

```

```

# read in data for calculations only at the beginning of session

if 'Xr_tr' not in st.session_state:

    read_data()

```

```

tab1, tab2, tab3 = st.tabs(["Surrogate model", "Thermal Resistance plot", "Pareto Front"])

with tab1:

```

```

    # Create the input sliders

    row1 = st.columns([1,1])

    default_value = 0.6

    st.session_state.x1 = default_value

    x1 = row1[0].slider("Design variable x1",0.4,1.0,default_value)

    st.session_state.x1 = x1

```

```

    st.session_state.x2 = default_value

    x2 = row1[1].slider("Design variable x2",0.4,1.0,default_value)

    st.session_state.x2 = x2

```

```

# Calculate surrogate models

```

```
tr_val,pd_val = calc_surrogates(x1,x2)

col1, col2 = st.columns(2)

with col1:

    st.write(f'Thermal resistance: {tr_val:.3f}')

with col2:

    st.write(f'Pressure drop: {pd_val:.1f} Pa')
```

with tab2:

```
tr_plot()
```

with tab3:

```
pareto_plot()
```

Single objective optimisation: drilling

Double objective optimisation: heat sink

Drilling Optimisation

This application enables you to explore the pressure drop of a twist drill

Surrogate model Pressure drop plot Leave one out cross validation

Corner radius (mm) Orientation angle (degrees)

0.05 0.20 0.50 0.00 45.00 90.00

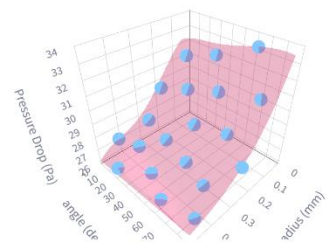
Pressure drop: 29.8 Pa

Drilling Optimisation

This application enables you to explore the pressure drop of a twist drill

Surrogate model Pressure drop plot Leave one out cross validation

RBF approximation with $\beta=1.545$ and $n=20$



Single objective optimisation: drilling

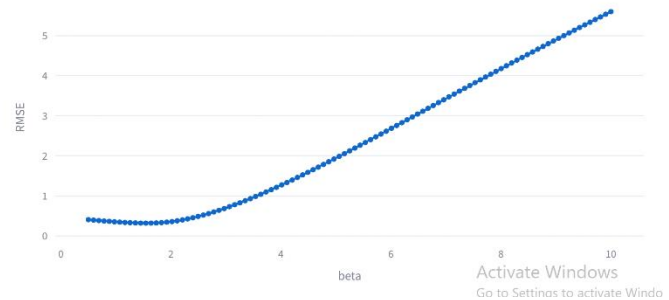
Double objective optimisation: heat sink

Drilling Optimisation

This application enables you to explore the pressure drop of a twist drill

Surrogate model Pressure drop plot Leave one out cross validation

Plot of RMSE vs Beta for Leave One Out Cross Validation



Heatsinks directory: surrogate modelling of the thermal resistance and pressure drop in a heat sink. Uses Radial Basis Functions and pre-prepared data from a Leave-One-Out Cross-Validation (LOOCV) exercise stored on the 'data' subdirectory. Uses tabs to display surrogate model calculations, surrogate models and the pareto front which gives the optimum compromises available between the thermal resistance and pressure drop.

```
# v4st_rbf_heatsink_leaveoneout.py
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from matplotlib import cm
```

```
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```
import matplotlib.pyplot as plt
```

```
import streamlit as st
```

```
import numpy as np
```

```
from rbffunctions import *
```

```
# note can get rid of all warning flags by using import rbffunctions and
```

```
# making an explicit call to the function using 'rbffunctions.etc'
```

```
rbfmodel = 1 # Gaussian weights - can automate this
```

```
n = 30 # number of DoE points - can automate this
```

```
ndv = 2 #
```

```
x1min_actual = 0.4
```

```
x1max_actual = 1.0
```

```
x2min_actual = 0.4
```

```
x2max_actual = 1.0
```

```
#####
```

```

# reading data into relevant files

#####

def read_data():

    # Objective 1 - thermal resistance

    st.session_state.Xr_tr = np.loadtxt("data/Xr_tr.txt")

    st.session_state.Yr_tr = np.loadtxt("data/Yr_tr.txt")

    st.session_state.Zr_tr = np.loadtxt("data/Zr_tr.txt")

    st.session_state.x_scaled_tr = np.loadtxt("data/x_scaled_tr.txt")

    lam1 = np.loadtxt("data/lam_tr.txt")

    st.session_state.lam_tr = np.transpose(np.asmatrix(lam1))

    st.session_state.xopt_tr = np.loadtxt("data/xopt_tr.txt")

    st.session_state.yopt_tr = np.loadtxt("data/yopt_tr.txt")

    st.session_state.optval_tr = np.loadtxt("data/optval_tr.txt")

    st.session_state.beta_tr = np.loadtxt("data/beta_tr.txt")

    st.session_state.obj_tr = np.loadtxt("data/obj_tr.txt")

    # Objective 2 - pressure difference

    st.session_state.Xr_pd = np.loadtxt("data/Xr_pd.txt")

    st.session_state.Yr_pd = np.loadtxt("data/Yr_pd.txt")

    st.session_state.Zr_pd = np.loadtxt("data/Zr_pd.txt")

    st.session_state.x_scaled_pd = np.loadtxt("data/x_scaled_pd.txt")

    lam2 = np.loadtxt("data/lam_pd.txt")

    st.session_state.lam_pd = np.transpose(np.asmatrix(lam2))

    st.session_state.xopt_pd = np.loadtxt("data/xopt_pd.txt")

    st.session_state.yopt_pd = np.loadtxt("data/yopt_pd.txt")

    st.session_state.optval_pd = np.loadtxt("data/optval_pd.txt")

    st.session_state.beta_pd = np.loadtxt("data/beta_pd.txt")

    st.session_state.obj_pd = np.loadtxt("data/obj_pd.txt")

    # Pareto points

    st.session_state.trpareto = np.loadtxt("data/trpareto.txt")

    st.session_state.pdpareto = np.loadtxt("data/pdpareto.txt")

    # create the actual points

    x1min_actual = 0.4

    x1max_actual = 1.0

    x2min_actual = 0.4

```



```

x2max_actual = 1.0

x_actual = np.loadtxt("data/x_scaled_tr.txt")

for i in range(0,n):

    x_actual[i][0] = x1min_actual + st.session_state.x_scaled_tr[i][0]*(x1max_actual-x1min_actual)

    x_actual[i][1] = x2min_actual + st.session_state.x_scaled_tr[i][1]*(x2max_actual-x2min_actual)

st.session_state.x_actual = x_actual


# scale the thermal resistance arrays

st.session_state.min_tr = 0.1618792

st.session_state.max_tr = 0.2472688

min_tr = st.session_state.min_tr

max_tr = st.session_state.max_tr

st.session_state.Xr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.Xr_tr

st.session_state.Yr = x2min_actual + (x2max_actual-x2min_actual)*st.session_state.Yr_tr

st.session_state.Zr = min_tr + (max_tr-min_tr)*st.session_state.Zr_tr

st.session_state.objactual_tr = min_tr + (max_tr-min_tr)*st.session_state.obj_tr

st.session_state.xoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_tr

st.session_state.yoptactual_tr = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_tr

st.session_state.optactual_tr = min_tr + (max_tr-min_tr)*st.session_state.optval_tr


# scale the pressure drop arrays

st.session_state.min_pd = 12553.08

st.session_state.max_pd = 38679.57

min_pd = st.session_state.min_pd

max_pd = st.session_state.max_pd

st.session_state.Zp = min_pd + (max_pd-min_pd)*st.session_state.Zr_pd

st.session_state.objactual_pd = min_pd + (max_pd-min_pd)*st.session_state.obj_pd

st.session_state.xoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.xopt_pd

st.session_state.yoptactual_pd = x1min_actual + (x1max_actual-x1min_actual)*st.session_state.yopt_pd

st.session_state.optactual_pd = min_pd + (max_pd-min_pd)*st.session_state.optval_pd


# calculate dimensionless value of thermal resistance at specified point

def tr_f(x):

    return rbf_point(n,st.session_state.x_scaled_tr,ndv,x,st.session_state.lam_tr,rbfmodel,st.session_state.beta_tr)


# calculate dimensionless value of pressure difference at specified point

def pd_f(x):

    return rbf_point(n,st.session_state.x_scaled_pd,ndv,x,st.session_state.lam_pd,rbfmodel,st.session_state.beta_pd)

```

```
# plot thermal resistance RBF surrogate model
```

```
def tr_plot():
```

```
    beta_tr = st.session_state.beta_tr
```

```
    Xr = st.session_state.Xr
```

```
    Yr = st.session_state.Yr
```

```
    Zr = st.session_state.Zr
```

```
    x_actual = st.session_state.x_actual
```

```
    objactual_tr = st.session_state.objactual_tr
```

```
    xoptactual_tr = st.session_state.xoptactual_tr
```

```
    yoptactual_tr = st.session_state.yoptactual_tr
```

```
    optactual_tr = st.session_state.optactual_tr
```

```
    # Plot out RBF approximation
```

```
    plt.figure()
```

```
    strbeta = f'{beta_tr:.2f}'
```

```
    plt.suptitle('RBF approximation of thermal resistance with beta='+strbeta+' and n='+str(n),fontsize=10)
```

```
    #ax = fig.gca(projection='3d')
```

```
    ax=plt.axes(projection='3d')
```

```
    surf = ax.plot_surface(Xr, Yr, Zr, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,
```

```
        linewidth=0, antialiased=False)
```

```
    # ax.set_zlim(-0.1, 1.1)
```

```
    ax.set_xlim(x1min_actual, x1max_actual)
```

```
    ax.set_ylim(x2min_actual, x2max_actual)
```

```
    ax.zaxis.set_major_locator(LinearLocator(10))
```

```
    ax.zaxis.set_major_formatter(FormatStrFormatter('%02f'))
```

```
    #fig.colorbar(surf, shrink=0.5, aspect=5)
```

```
    # plot out the scatter points
```

```
    ax.scatter(x_actual[:,0],x_actual[:,1],objactual_tr, c='r', marker='o',s=4)
```

```
    ax.scatter(xoptactual_tr,yoptactual_tr,optactual_tr, c='k', marker='o',s=16) # plot optimum point
```

```
    ax.set_xlabel('x1')
```

```
    ax.set_ylabel('x2')
```

```
    ax.set_zlabel('Thermal Resistance')
```

```
    plt.savefig('tf_rbfheatsink.jpg')
```

```
    st.pyplot(plt)
```

```
# plot pressure drop RBF surrogate model
```

```
def pd_plot():
```

```
    beta_pd = st.session_state.beta_pd
```

```
    Xr = st.session_state.Xr
```

```
    Yr = st.session_state.Yr
```

```
    Zp = st.session_state.Zp
```

```
    x_actual = st.session_state.x_actual
```

```
    objactual_pd = st.session_state.objactual_pd
```

```
    xoptactual_pd = st.session_state.xoptactual_pd
```

```
    yoptactual_pd = st.session_state.yoptactual_pd
```

```
    optactual_pd = st.session_state.optactual_pd
```

```
    # Plot out RBF approximation
```

```
    plt.figure()
```

```
    plt.suptitle('RBF approximation of pressure drop with beta='+str(beta_pd)+' and n='+str(n),fontsize=10)
```

```
    #ax = fig.gca(projection='3d')
```

```
    ax=plt.axes(projection='3d')
```

```
    surf = ax.plot_surface(Xr, Yr, Zp, rstride=8, cstride=8, alpha=0.3, cmap=cm.coolwarm,
```

```
        linewidth=0, antialiased=False)
```

```
    # ax.set_zlim(-0.1, 1.1)
```

```
    ax.set_xlim(x1min_actual, x1max_actual)
```

```
    ax.set_ylim(x2min_actual, x2max_actual)
```

```
    ax.zaxis.set_major_locator(LinearLocator(10))
```

```
    ax.zaxis.set_major_formatter(FormatStrFormatter('%0f'))
```

```
    #fig.colorbar(surf, shrink=0.5, aspect=5)
```

```
    # plot out the scatter points
```

```
    ax.scatter(x_actual[:,0],x_actual[:,1],objactual_pd, c='r', marker='o',s=4)
```

```
    ax.scatter(xoptactual_pd,yoptactual_pd,optactual_pd, c='k', marker='o',s=16) # plot optimum point
```

```
    ax.set_ylabel('x2')
```

```
    ax.set_xlabel('x1')
```

```
    ax.set_zlabel('Pressure Drop')
```

```
    plt.savefig('pd_rbfheatsink.jpg')
```

```
    st.pyplot(plt)
```

```
# Plot Pareto Front
```

```
def pareto_plot():
```

```

trpareto = st.session_state.trpareto
pdpareto = st.session_state.pdpareto

# Plot out pareto set

plt.figure()

# plt.ion()

plt.xlabel('thermal resistance')

plt.ylabel('pressure drop, Pa')

plt.title('Pareto front for thermal resistance vs pressure drop',fontsize=10)

plt.plot(trpareto,pdpareto)

# plt.show()

plt.savefig('pareto_rbfheatsink.jpg')

st.pyplot(plt)

# calculate surrogate models of thermal resistance and pressure drop
def calc_surrogates(x1,x2):

    xp = np.zeros(ndv)

    x1_scaled = (x1-x1min_actual)/(x1max_actual-x1min_actual)

    x2_scaled = (x2-x2min_actual)/(x2max_actual-x2min_actual)

    xp[0] = x1_scaled

    xp[1] = x2_scaled

    min_tr = st.session_state.min_tr

    max_tr = st.session_state.max_tr

    min_pd = st.session_state.min_pd

    max_pd = st.session_state.max_pd

    tr_val = min_tr + (max_tr - min_tr)*tr_f(xp)

    pd_val = min_pd + (max_pd - min_pd)*pd_f(xp)

    return tr_val,pd_val

# Add heading and introductory text

# st.set_page_config(layout='wide')

st.title("Heat Sink Optimisation")

st.write("This application enables you to explore the thermal resistance and pressure drop of a heat sink")

st.markdown("---")

```

```

# read in data for calculations only at the beginning of session

if 'Xr_tr' not in st.session_state:

    read_data()

tab1, tab2, tab3, tab4 = st.tabs(["Surrogate model", "Thermal Resistance plot", "Pressure Drop plot", "Pareto Front"])

with tab1:

    # Create the input sliders

    row1 = st.columns([1,1])

    default_value = 0.6

    st.session_state.x1 = default_value

    x1 = row1[0].slider("Design variable x1",0.4,1.0,default_value)

    st.session_state.x1 = x1

    st.session_state.x2 = default_value

    x2 = row1[1].slider("Design variable x2",0.4,1.0,default_value)

    st.session_state.x2 = x2

    # Calculate surrogate models

    tr_val,pd_val = calc_surrogates(x1,x2)

    col1, col2 = st.columns(2)

    with col1:

        st.write(f'Thermal resistance: {tr_val:.3f}')

    with col2:

        st.write(f'Pressure drop: {pd_val:.1f} Pa')

with tab2:

    tr_plot()

with tab3:

    pd_plot()

with tab4:

    pareto_plot()

```

rbffunctions.py

```
def rbfweights(n,x,ndv,f,rbfmodel,beta):
```

```
    import numpy as np
```

```
    import sys
```

```
    phi = np.zeros(shape=(n,n))
```

```
    # phi = np.matrix(phi1)
```

```
    # calculate the gram matrix
```

```
    for i in range(0,n):
```

```
        for j in range(0,n):
```

```
            r = 0.0
```

```
            for k in range (0,ndv):
```

```
                r = r + (x[i][k]-x[j][k])**2
```

```
            r = np.sqrt(r)
```

```
            if (rbfmodel == 1):
```

```
                phi[i][j] = np.exp(-beta*(r**2))
```

```
            else:
```

```
                if (rbfmodel == 2):
```

```
                    phi[i][j] = np.sqrt(beta**2 + r**2)
```

```
                else:
```

```
                    if (rbfmodel == 3):
```

```
                        phi[i][j] = 1/np.sqrt(beta**2 + r**2)
```

```
                    else:
```

```
                        print ("invalid rbfmodel value")
```

```
                        sys.exit(0)
```

```
    # compute the weights. need to transform f into an array and transpose it
```

```
    farray = np.asarray(f)
```

```
    fm = np.asmatrix(farray)
```

```
    fmt = fm.transpose()
```

```
    lam = np.linalg.solve(phi, fmt)
```

```
    return lam
```

```
    print ("Finished computing rbf weights in rbfweights")
```

```
# function to calculate rbf approximation at specified plan design points xa
```

```
def rbf(n,x,ndv,xa,na,lam,rbfmodel,beta):
```

```

import numpy as np
import sys

phi = np.zeros(shape=(na*na,n))
# phi = np.zeros(n,n,float)

# calculate the gram matrix
for i in range(0,na*na):
    for j in range(0,n):
        r = 0.0
        for k in range (0,ndv):
            r = r + (xa[i][k]-x[j][k])**2
        r = np.sqrt(r)
        if (rbfmodel == 1):
            phi[i][j] = np.exp(-beta*(r**2))
        else:
            if (rbfmodel == 2):
                phi[i][j] = np.sqrt(beta**2 + r**2)
            else:
                if (rbfmodel == 3):
                    phi[i][j] = 1/np.sqrt(beta**2 + r**2)
                else:
                    print ("invalid rbfmodel value")
                    sys.exit(0)

# compute the rbf response
# ya = phi*lam
# phim = np.matrix(phi)
# lamm = np.matrix(lam)
ya = np.asmatrix(phi)*lam
return ya

print ("Finished computing rbf approximations")

# define function
def yval(x,y):

```

```
yval = x**2 + y**2
```

```
return yval
```

```
# define six hump camel function
```

```
def sixhumpcamel(x1,x2):
```

```
    X1 = 4*x1-2;
```

```
    X2 = 2*x2-1;
```

```
    fval = (4 - 2.1*(X1**2) + (X1**4)/3)*X1**2 + X1*X2 + (4*(X2**2) - 4)*(X2**2)
```

```
    return fval
```

```
# generate test dataset from ntest random numbers between 1 and n
```

```
def generate_test(ntest,n):
```

```
    import random
```

```
    testindex = []
```

```
    testflag = []
```

```
    for i in range(n):
```

```
        testflag.append(0)
```

```
    num = 0
```

```
    for i in range(100):
```

```
        if (num < ntest):
```

```
            k = random.randint(0,n-1)
```

```
            if (testflag[k] == 0):
```

```
                testindex.append(k)
```

```
                testflag[k] = 1
```

```
                num = num + 1
```

```
        else:
```

```
            break
```

```
    return testindex, testflag
```

```
# function to calculate rbf approximation at specified ntest test points x_test
```

```
def rbf_testeval(ntrain,x_train,ndv,x_test,ntest,lam,rbfmodel,beta):
```

```
    import numpy as np
```

```
    import sys
```



```
phi = np.zeros(shape=(ntest,ntrain))
```

```
# phi = np.zeros(n,n,float)
```

```
# calculate the gram matrix
```

```
for i in range(0,ntest):
```

```
    for j in range(0,ntrain):
```

```
        r = 0.0
```

```
        for k in range (0,ndv):
```

```
            r = r + (x_test[i][k]-x_train[j][k])**2
```

```
        r = np.sqrt(r)
```

```
        if (rbfmodel == 1):
```

```
            phi[i][j] = np.exp(-beta*(r**2))
```

```
        else:
```

```
            if (rbfmodel == 2):
```

```
                phi[i][j] = np.sqrt(beta**2 + r**2)
```

```
            else:
```

```
                if (rbfmodel == 3):
```

```
                    phi[i][j] = 1/np.sqrt(beta**2 + r**2)
```

```
                else:
```

```
                    print ("invalid rbfmodel value")
```

```
                    sys.exit(0)
```

```
y_test = np.asmatrix(phi)*lam
```

```
return y_test
```

```
print ("Finished computing rbf approximations at test data points")
```

```
# function to calculate training and testing data for leave one out validation
```

```
def leaveoneout(i,x,f,n,ndv):
```

```
    import numpy as np
```

```
    # create leave one out training and test data sets with test data on index i
```

```
    f_train = []
```

```
    f_test = []
```

```
    x1_train = []
```

```

x1_test = []
x2_train = []
x2_test = []

# create leave one out training and test data sets
for j in range(0,n):
    if (j < i):
        f_train.append(f[j])
        x1_train.append(x[j][0])
        x2_train.append(x[j][1])
    else:
        if (j == i):
            f_test.append(f[i])
            x1_test.append(x[i][0])
            x2_test.append(x[i][1])
        else:
            f_train.append(f[j])
            x1_train.append(x[j][0])
            x2_train.append(x[j][1])

# put training and test data into arrays
x_test = (np.vstack([x1_test,x2_test])).transpose()
x_train = (np.vstack([x1_train,x2_train])).transpose()

return f_train, x_train, f_test, x_test

```

function to calculate a rpf surrogate function value at a specific point xp

```
def rbf_point(n,x,ndv,xp,lam,rbfmodel,beta):
```

```
import numpy as np
```

```
import sys
```

```
phi = np.zeros(shape=(1,n))
```

```
# phi = np.zeros(n,n,float)
```

```
# calculate the gram matrix
```

```
for j in range(0,n):
```

```
    r = 0.0
```

```
    for k in range (0,ndv):
```

```

        r = r + (xp[k]-x[j])[k]**2
    r = np.sqrt(r)
    if (rbfmodel == 1):
        phi[0][j] = np.exp(-beta*(r**2))
    else:
        if (rbfmodel == 2):
            phi[0][j] = np.sqrt(beta**2 + r**2)
        else:
            if (rbfmodel == 3):
                phi[0][j] = 1/np.sqrt(beta**2 + r**2)
            else:
                print ("invalid rbfmodel value")
                sys.exit(0)

# compute the rbf response
# ya = phi*lam
# phim = np.matrix(phi)
# lamm = np.matrix(lam)
yp = (np.asmatrix(phi)*lam).item()

return yp

print ("Finished computing rbf approximations")

```

Heat Sink Optimisation ⇄

This application enables you to explore the thermal resistance and pressure drop of a heat sink

Surrogate model Thermal Resistance plot Pressure Drop plot Pareto Front

Design variable x1



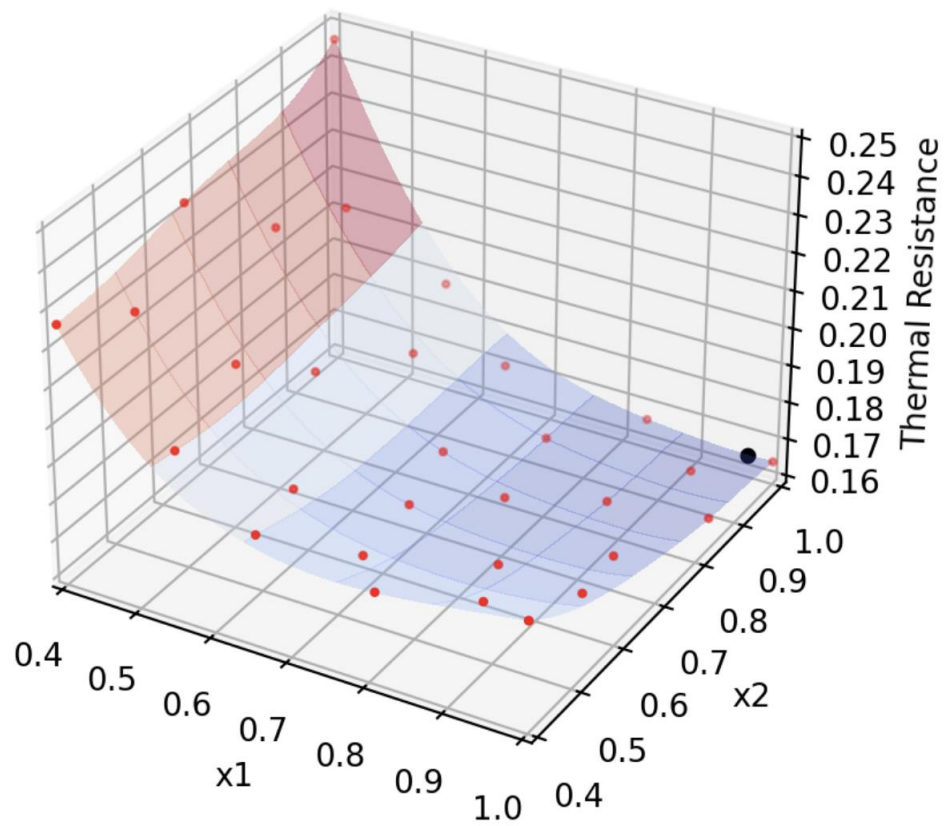
Thermal resistance: 0.181

Design variable x2

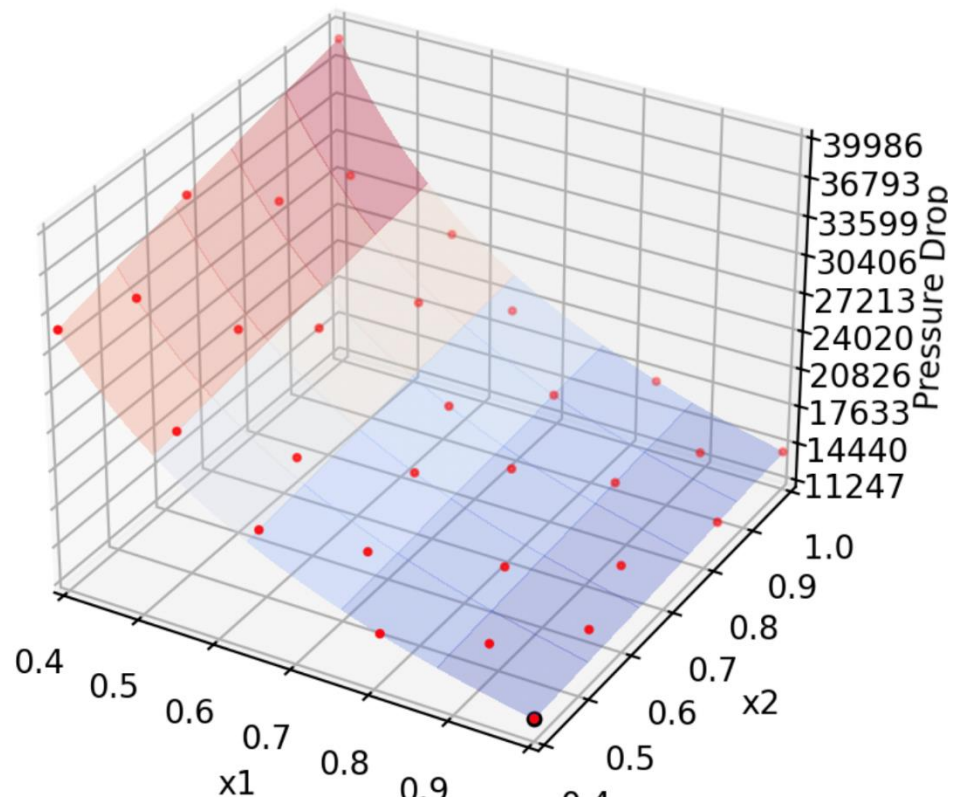


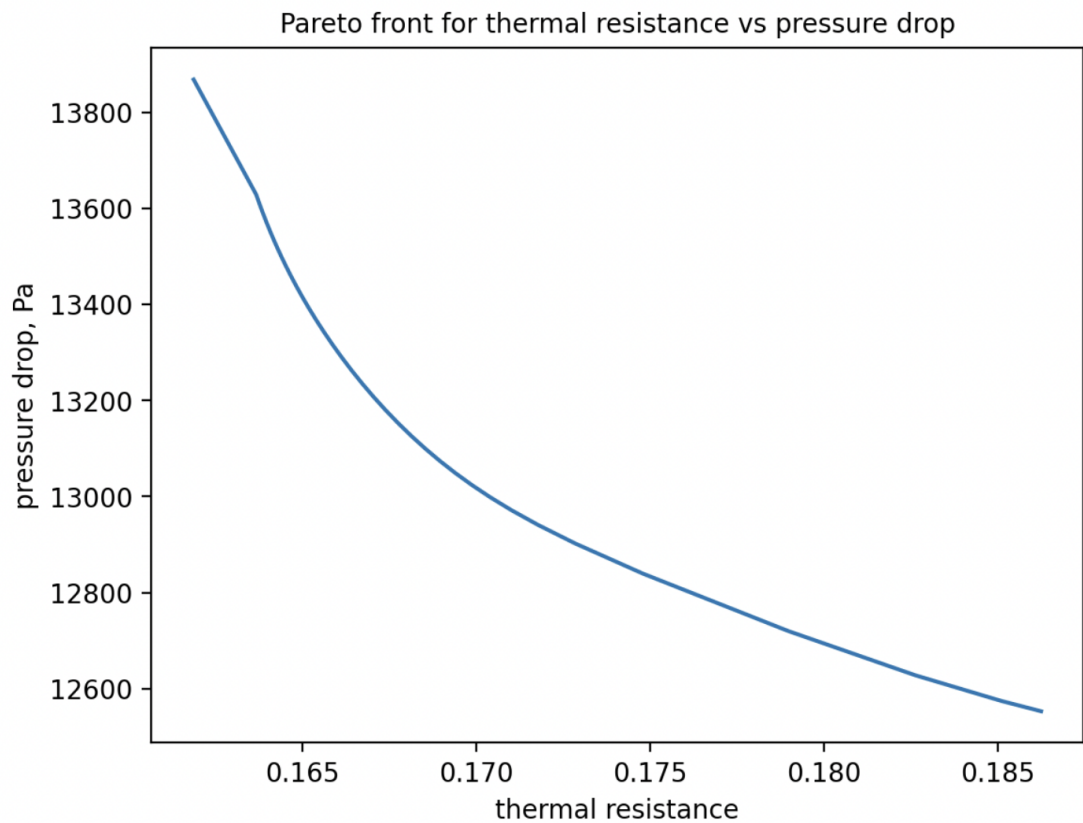
Pressure drop: 22113.5 Pa

RBF approximation of thermal resistance with $\beta=1.64$ and $n=30$



RBF approximation of pressure drop with $\beta=0.88$ and $n=30$





NelderMeadSimplex directory: a multipage application which minimises functions of two variables using the Nelder Mead Simplex algorithm.

Here the main driver program is 'v2st_rosenbrock_2DNMSimplex.py' on the main directory and the other program on the 'pages' subdirectory is 'v2st_sixhump_2DNMSimplex.py'. Both programs use Nelder Mead Simplex functionality in 'NMSimplex.py' on the main directory.

```
# v2st_rosenbrock_2DNMSimplex.py

# Python implementation of the Nelder Mead Simplex algorithm for use in streamlit'

import numpy as np
import streamlit as st
import streamlit_scrollable_textbox as stx
import matplotlib.pyplot as plt
import sys
import sympy as sp
import time

from NMSimplex import *
from scipy import optimize
```

```
#####

# Functions

#####

# objective function
def obj2(x):

    x0 = x[0]

    x1 = x[1]

    obj2 = obj(x0,x1)

    return obj2


# define 2D Nelder-Mead Simplex program
def nelder_mead():

    x_0 = st.session_state.x_0
    y_0 = st.session_state.y_0
    xinitial = [x_0, y_0]

    res = optimize.minimize(obj2,xinitial)

    print('Global minimum value = {0:6.3f} at x = {1:6.3f} {2:6.3f}\n'.format(res.fun,res.x[0],res.x[1]))


# initial NM Simplex setup parameters
ndv=2
x0=np.array([x_0,y_0])
c = st.session_state.c
alpha = st.session_state.alpha
gamma = st.session_state.gamma
beta = st.session_state.beta
rho = st.session_state.rho
tol = st.session_state.tol # convergence tolerance


converged = 0; maxiters = 100;

# store results of simplex calculations
x_simplex = np.zeros((maxiters+1,3,2),dtype='float64')
fmin_simplex = np.zeros(maxiters+1,dtype='float64')
xp = np.zeros((3,2),dtype='float64'); fp = np.zeros(3,dtype='float64'); # arrays for determining next simplex


# create initial NM simplex
xL,xM,xH = create_simplex(x0,c,obj2)
```

```

niter = 0

outstr = ''

outstr+=('Nelder-Mead Simplex Parameters:\n')

outstr+=('-----\n')

outstr+=('x0={0:8.3f} {1:8.3f}, f0={2:8.3f}\n'.format(x0[0],x0[1],obj2(x0)))

outstr+=('c={0:4.2f} alpha={1:4.2f} gamma={2:4.2f} beta={3:4.2f} rho={4:4.2f} tol={5:8.5f} \n'.

        format(c,alpha,gamma,beta,rho,tol))


outstr+=('Initial simplex\n')

outstr+=('xL={0:8.3f} {1:8.3f}, fL={2:8.3e}\n'.format(xL[0],xL[1],obj2(xL)))

outstr+=('xM={0:.3f} {1:8.3f}, fM={2:8.3e}\n'.format(xM[0],xM[1],obj2(xM)))

outstr+=('xH={0:8.3f} {1:8.3f}, fH={2:8.3e}\n'.format(xH[0],xH[1],obj2(xH)))

x_simplex[0][0][:]=xL; x_simplex[0][1][:]=xM; x_simplex[0][2][:]=xH;

fmin_simplex[0]=obj2(xL)


while (converged == 0):


    niter = niter+1

    xCE = 0.5*(xL + xM)


    # Carry out reflection operation

    xR,fR = reflection(xL,xM,xH,alpha,obj2)

    fL=obj2(xL); fM=obj2(xM); fH=obj2(xH);


    if ((fL < fR) and (fR < fM)):


        # update the simplex points

        xH = xM

        xM = xR

        xCE = 0.5*(xL + xM)

        converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

        outstr+=('Reflection carried out for iteration {0:3d}\n'.format(niter))


    elif (fR < fL):


        # carry out expansion

        xE,fE = expansion(xCE,xR,gamma,obj2)

        oldxL = xL; oldxM = xM; oldxH = xH;

```



```

if (fE < fR):

    xL = xE

    xM = oldxL

    xH = oldxM

else:

    xL = xR

    xM = oldxL

    xH = oldxM

xCE = 0.5*(xL + xM)

converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

outstr+=('Expansion carried out for iteration {0:3d}\n'.format(niter))

elif ((fR > fM) and (fR < fH)):

    # carry out outside contraction

    xOC,fOC = outside_contraction(xCE,xR,beta,obj2)

    if (fOC < fR):

        # update simplex points based on three smallest function values

        fp[0]=fL; fp[1]=fM; fp[2]=fOC

        xp = np.zeros((3,2),dtype='float64')

        xp[0,:]=xL; xp[1,:]=xM; xp[2,:]=xOC

        # update simplex with three smallest function values

        ipos = np.argsort(fp)

        xL = xp[ipos[0],:]

        xM = xp[ipos[1],:]

        xH = xp[ipos[2],:]

    else:

        # carry out shrinking operation

        newxM,newxH = shrinking(xL,xM,xH,rho,obj2)

        xM = newxM

        xH = newxH

```

```

xCE = 0.5*(xL + xM)

converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

outstr+=('Outside contraction carried out for iteration {0:3d}\n'.format(niter))

elif (fR > fH):

    # carry out inside contraction

    xIC,fIC = inside_contraction(xCE,xR,beta,obj2)

    if (fIC < fH):

        # update simplex points based on three smallest function values

        fp[0]=fL; fp[1]=fM; fp[2]=fIC

        xp = np.zeros((3,2),dtype='float64')

        xp[0,:]=xL; xp[1,:]=xM; xp[2,:]=xIC

        # update simplex with three smallest function values

        ipos = np.argsort(fp)

        xL = xp[ipos[0],:]

        xM = xp[ipos[1],:]

        xH = xp[ipos[2],:]

    else:

        # carry out shrinking operation

        newxM,newxH = shrinking(xL,xM,xH,rho,obj2)

        xM = newxM

        xH = newxH

    xCE = 0.5*(xL + xM)

    converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

    outstr+=('Inside contraction carried out for iteration {0:3d}\n'.format(niter))

outstr+=('After {0:3d} iterations, simplex is given by:\n'.format(niter))

outstr+=('xL={0:8.3f} {1:8.3f}, fL={2:8.3e}\n'.format(xL[0],xL[1],obj2(xL)))

outstr+=('xM={0:8.3f} {1:8.3f}, fM={2:8.3e}\n'.format(xM[0],xM[1],obj2(xM)))

```

```

    outstr+='xH={0:8.3f} {1:8.3f}, fH={2:8.3e}\n'.format(xH[0],xH[1],obj2(xH))

    x_simplex[niter][0]=xL; x_simplex[niter][1]=xM; x_simplex[niter][2]=xH;
    fmin_simplex[niter]=obj2(xL)

# end while loop

outstr+='NM simplex converged with tol={0:6.3e} after {1:3d} iterations\n'.format(tol,niter))
outstr+='Minimum f={0:8.3e} at x= {1:8.3e} {2:8.3e}\n'.format(obj2(xL),xL[0],xL[1]))

st.session_state.Calcs=outstr

st.session_state.x_simplex=x_simplex
st.session_state.fmin_simplex=fmin_simplex
st.session_state.niter=niter
st.session_state.xMin=x_simplex[niter][0][0]
st.session_state.yMin=x_simplex[niter][0][1]
st.session_state.fMin=fmin_simplex[niter]

print('finished Nelder-Mead')

# Function to create and update the plot for Nelder Mead Simplex
def create_plot():

    # Run the Nelder-Mead Simplex
    nelder_mead()

    x_simplex=st.session_state.x_simplex
    fmin_simplex=st.session_state.fmin_simplex
    niter=st.session_state.niter

    # create initial NM simplex
    x_0 = st.session_state.x_0
    y_0 = st.session_state.y_0
    x0=np.array([x_0,y_0])
    c = st.session_state.c
    xL,xM,xH = create_simplex(x0,c,obj2)

    plt.figure()

    # now plot out solution
    X, Y = np.meshgrid(np.linspace(-2, 2, 101), np.linspace(-1, 3, 101))

```

```
# Z = 100*((Y-X**2)**2)+(1-X)**2
```

```
Z = obj(X,Y)
```

```
# plot out first simplex
```

```
xL[0]=x_simplex[0][0][0]; xL[1]=x_simplex[0][0][1];  
xM[0]=x_simplex[0][1][0]; xM[1]=x_simplex[0][1][1];  
xH[0]=x_simplex[0][2][0]; xH[1]=x_simplex[0][2][1];  
initial_simplex=np.zeros((4,3),dtype='float64')  
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]  
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]  
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]  
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]  
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'r-')
```

```
# plot out second simplex
```

```
xL[0]=x_simplex[1][0][0]; xL[1]=x_simplex[1][0][1];  
xM[0]=x_simplex[1][1][0]; xM[1]=x_simplex[1][1][1];  
xH[0]=x_simplex[1][2][0]; xH[1]=x_simplex[1][2][1];  
initial_simplex=np.zeros((4,3),dtype='float64')  
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]  
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]  
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]  
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]  
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'b-')
```

```
# plot out third simplex
```

```
xL[0]=x_simplex[2][0][0]; xL[1]=x_simplex[2][0][1];  
xM[0]=x_simplex[2][1][0]; xM[1]=x_simplex[2][1][1];  
xH[0]=x_simplex[2][2][0]; xH[1]=x_simplex[2][2][1];  
initial_simplex=np.zeros((4,3),dtype='float64')  
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]  
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]  
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]  
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]  
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'k-')
```

```
# plot out contours
```

```
plt.ylabel('y')
```

```

plt.xlabel('x')

plt.title('Nelder Mead search on '+st.session_state.expression)

# plot1.set_title('f=$100(y-x^2)^2+(1-x)^2$')

CS = plt.contour(X, Y, Z, levels=[0, 2.5, 25, 100])

xout=np.linspace(0,1,2); yout = np.linspace(0,1,2)

for i in range(niter-1):

    xout[0]=x_simplex[i][0][0]; yout[0]=x_simplex[i][0][1];

    xout[1]=x_simplex[i+1][0][0]; yout[1]=x_simplex[i+1][0][1];

    plt.scatter(xout[0],yout[0],c='k',marker='o')

    plt.plot(xout,yout,'k-')

plt.scatter(xout[1],yout[1],c='r',marker='o',)

return plt

```

Function to create plots for Nelder Mead Simplex used in animation

def animate_plot():

Run the Nelder-Mead Simplex

nelder_mead()

x_simplex=st.session_state.x_simplex

fmin_simplex=st.session_state.fmin_simplex

niter=st.session_state.niter

create initial NM simplex

x_0 = st.session_state.x_0

y_0 = st.session_state.y_0

x0=np.array([x_0,y_0])

c = st.session_state.c

xL,xM,xH = create_simplex(x0,c,obj2)

plt.figure()

now plot out solution

X, Y = np.meshgrid(np.linspace(-2, 2, 101), np.linspace(-1, 3, 101))

Z = 100*((Y-X**2)**2)+(1-X)**2

Z = obj(X,Y)

plot out first simplex

```

xL[0]=x_simplex[0][0][0]; xL[1]=x_simplex[0][0][1];
xM[0]=x_simplex[0][1][0]; xM[1]=x_simplex[0][1][1];
xH[0]=x_simplex[0][2][0]; xH[1]=x_simplex[0][2][1];
initial_simplex=np.zeros((4,3),dtype='float64')
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'r-')

# plot out second simplex
xL[0]=x_simplex[1][0][0]; xL[1]=x_simplex[1][0][1];
xM[0]=x_simplex[1][1][0]; xM[1]=x_simplex[1][1][1];
xH[0]=x_simplex[1][2][0]; xH[1]=x_simplex[1][2][1];
initial_simplex=np.zeros((4,3),dtype='float64')
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'b-')

# plot out third simplex
xL[0]=x_simplex[2][0][0]; xL[1]=x_simplex[2][0][1];
xM[0]=x_simplex[2][1][0]; xM[1]=x_simplex[2][1][1];
xH[0]=x_simplex[2][2][0]; xH[1]=x_simplex[2][2][1];
initial_simplex=np.zeros((4,3),dtype='float64')
initial_simplex[0][0]=xL[0]; initial_simplex[0][1]=xL[1]
initial_simplex[1][0]=xM[0]; initial_simplex[1][1]=xM[1]
initial_simplex[2][0]=xH[0]; initial_simplex[2][1]=xH[1]
initial_simplex[3][0]=xL[0]; initial_simplex[3][1]=xL[1]
plt.plot(initial_simplex[:,0],initial_simplex[:,1], 'k-')

# plot out contours
plt.ylabel('y')
plt.xlabel('x')
plt.title('Nelder Mead search on '+st.session_state.expression)
# plot1.set_title('f=$100(y-x^2)^2+(1-x)^2$')
CS = plt.contour(X, Y, Z, levels=[0, 2.5, 25, 100])

```

```

xout=np.linspace(0,1,2); yout = np.linspace(0,1,2)

for i in range(st.session_state.ncalls):

    xout[0]=x_simplex[i][0][0]; yout[0]=x_simplex[i][0][1];

    xout[1]=x_simplex[i+1][0][0]; yout[1]=x_simplex[i+1][0][1];

    plt.scatter(xout[0],yout[0],c='k',marker='o')

    plt.plot(xout,yout,'k-')

if (st.session_state.ncalls == st.session_state.niter - 2):

    plt.scatter(xout[1],yout[1],c='r',marker='o',)

```

```

return plt

```

```

#####

```

```

# Main program

```

```

#####

```

```

# Streamlit commands

```

```

st.title("Nelder-Mead Simplex Algorithm Program")

```

```

st.write("This application enables you to experiment with Nelder-Mead solution of 2-D equations")

```

```

# initialise the objective function

```

```

exp = st.text_input("Objective Function",value='100*(y-x^2)^2 + (1-x)^2')

```

```

if exp is not None:

```

```

    expression = exp

```

```

else:

```

```

    expression = '100*(y-x^2)^2 + (1-x)^2'

```

```

x = sp.symbols('x')

```

```

y = sp.symbols('y')

```

```

try:

```

```

    obj_function = sp.sympify(expression)

```

```

except Exception as e:

```

```

    st.error(f"An error occured using sympify: {str(e)}")

```

```

    st.stop()

```

```

try:

```

```

obj = sp.lambdify([x,y], obj_function, 'numpy')
except Exception as e:
    st.error(f"An error occured using lambdify: {str(e)}")
    st.stop()

st.session_state.obj = obj
st.session_state.expression = expression

st.write("Parameter settings")

# First row
cols = st.columns([1,1,1,1,1])

alpha = cols[0].number_input(chr(945),0.1,5.0,1.0)
st.session_state.alpha = alpha

beta = cols[1].number_input(chr(946),0.1,5.0,0.5,0.1)
st.session_state.beta = beta

gamma = cols[2].number_input(chr(947),0.1,5.0,2.0,0.1)
st.session_state.gamma = gamma

rho = cols[3].number_input(chr(961),0.1,5.0,0.5,0.1)
st.session_state.rho = rho

tol = cols[4].number_input("Tolerance",0.00001,0.01,0.0001)
st.session_state.tol = tol

# Second row
cols2 = st.columns([1,1,1,1,1])

x_0 = cols2[0].number_input("$x_0$",-5.0,5.0,-1.2,0.1)
st.session_state.x_0 = x_0

y_0 = cols2[1].number_input("$y_0$",-5.0,5.0,1.0,0.1)
st.session_state.y_0 = y_0

c = cols2[2].number_input("c",0.1,5.0,2.0,0.1)

```



```

st.session_state.c = c

# Get initial solution of nelder mead simplex calculations
cols3 = st.columns([1,1,1])
nelder_mead()

cols3[0].text("x min="+"{:.4f}".format(st.session_state.xMin))
cols3[1].text("y min="+"{:.4f}".format(st.session_state.yMin))
cols3[2].text("f min="+"{:.4e}".format(st.session_state.fMin))

if 'plotgraph' not in st.session_state:
    st.session_state.plotgraph = 1
    if st.button("Plot Graph and Calculations"):
        figplt = create_plot()
        st.pyplot(figplt)
        st.write('Nelder Mead Calculations')
        stx.scrollableTextbox(st.session_state.Calcs,height=200)
else:
    figplt = create_plot()
    st.pyplot(figplt)
    st.write('Nelder Mead Calculations')
    stx.scrollableTextbox(st.session_state.Calcs,height=200)

# animate calculations
if st.button("Animate Graph"):
    st.session_state.ncalls = 0
    figplt = animate_plot()
    the_plot = st.pyplot(figplt)
    for i in range(st.session_state.niter - 2):
        st.session_state.ncalls += 1
        time.sleep(0.1)
        figplt = animate_plot()
        the_plot.pyplot(figplt)

# pages/v2st_sixhump_2DNMsimplex.py
# Python implementation of the Bisector algorithm for use in streamlit
import numpy as np
import streamlit as st
import streamlit_scrollable_textbox as stx

```

```

import matplotlib.pyplot as plt

import sys

import sympy as sp

import time


from NMsimplex import *

from scipy import optimize


# create and sort initial simplex points

def create_simplex(x0,c,obj):

    xp_ini = np.zeros((3,2),dtype='float64'); fp_ini = np.zeros(3,dtype='float64');

    xp_ini[0][0] = x0[0]; xp_ini[0][1] = x0[1]; fp_ini[0] = obj(xp_ini[0,:]);

    xp_ini[1][0] = xp_ini[0][0] + 0.96593*c; xp_ini[1][1] = xp_ini[0][1] + 0.25882*c;

    fp_ini[1] = obj(xp_ini[1,:])

    xp_ini[2][0] = xp_ini[0][0] + 0.25882*c; xp_ini[2][1] = xp_ini[0][1] + 0.96593*c;

    fp_ini[2] = obj(xp_ini[2,:])

    ipos = np.argsort(fp_ini)

    xL = xp_ini[ipos[0],:]

    xM = xp_ini[ipos[1],:]

    xH = xp_ini[ipos[2],:]


    return xL,xM,xH


# Reflection operation

def reflection(xL,xM,xH,alpha,obj):

    # calculate centroid excluding worst point (biggest function value)

    xCE = 0.5*(xL + xM)

    # calculate reflection point

    xR = (1+alpha)*xCE - alpha*xH

    fR = obj(xR)

    return xR,fR


# end reflection

```

```

# Expansion operation

def expansion(xCE,xR,gamma,obj):

    # calculate expansion point

     $x_E = \gamma \cdot x_R + (1-\gamma) \cdot x_{CE}$ 

     $f_E = \text{obj}(x_E)$ 

    return xE,fE

# end expansion


# Outside Contraction operation

def outside_contraction(xCE,xR,beta,obj):

    # calculate contraction point

     $x_{OC} = x_{CE} + \beta \cdot (x_R - x_{CE})$ 

     $f_{OC} = \text{obj}(x_{OC})$ 

    return xOC,fOC

# end outside contraction


# Inside Contraction operation

def inside_contraction(xCE,xR,beta,obj):

    # calculate contraction point

     $x_{IC} = x_{CE} - \beta \cdot (x_R - x_{CE})$ 

     $f_{IC} = \text{obj}(x_{IC})$ 

    return xIC,fIC

# end inside contraction


# Shrinking operation

def shrinking(xL,xM,xH,rho,obj):

    # calculate shrinkage points about xL

     $newx_M = x_L + \rho \cdot (x_M - x_L)$ 

```

```

newxH = xL + rho*(xH-xL)

return newxM,newxH

# end shrinking

# check convergence
def check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj):

    std = (obj(xL)-obj(xCE))**2 + (obj(xM)-obj(xCE))**2 + (obj(xH)-obj(xCE))**2

    std=np.sqrt(std)/3.0

    converged = 0

    if ((std < tol) or (niter==(maxiters-1))):

        converged = 1

    return converged

# end check_convergence

#####

# Functions

#####

# analytical form of six hump camel function
def sixhump_func(x,y):

    x1 = 4*x -2

    x2 = 2*y - 1

    f = (4 - 2.1*(x1**2) + (x1**4)/3)*x1**2 + x1*x2 + (-4 + 4*(x2**2))*x2**2

    return f

# objective function
def obj2(x):

    x0 = x[0]

    x1 = x[1]

    obj2 = sixhump_func(x0,x1)

    return obj2

# define 2D Nelder-Mead Simplex program
def nelder_mead():

```

```

x_0 = st.session_state.x_0
y_0 = st.session_state.y_0
xinitial = [x_0, y_0]

res = optimize.minimize(obj2,xinitial)
print('Global minimum value = {0:6.3f} at x = {1:6.3f} {2:6.3f}\n'.format(res.fun,res.x[0],res.x[1]))

# initial NM Simplex setup parameters

ndv=2

x0=np.array([x_0,y_0])
c = st.session_state.c

alpha = st.session_state.alpha
gamma = st.session_state.gamma
beta = st.session_state.beta
rho = st.session_state.rho
tol = st.session_state.tol # convergence tolerance

converged = 0; maxiters = 100;

# store results of simplex calculations
x_simplex = np.zeros((maxiters+1,3,2),dtype='float64')
fmin_simplex = np.zeros(maxiters+1,dtype='float64')
xp = np.zeros((3,2),dtype='float64'); fp = np.zeros(3,dtype='float64'); # arrays for determining next simplex

# create initial NM simplex
xL,xM,xH = create_simplex(x0,c,obj2)
niter = 0

outstr = ''
outstr+=('Nelder-Mead Simplex Parameters:\n')
outstr+=('-----\n')
outstr+=('x0={0:8.3f} {1:8.3f}, f0={2:8.3f}\n'.format(x0[0],x0[1],obj2(x0)))
outstr+=('c={0:4.2f} alpha={1:4.2f} gamma={2:4.2f} beta={3:4.2f} rho={4:4.2f} tol={5:8.5f} \n'.
        format(c,alpha,gamma,beta,rho,tol))

outstr+=('Initial simplex\n')
outstr+=('xL={0:8.3f} {1:8.3f}, fL={2:8.3e}\n'.format(xL[0],xL[1],obj2(xL)))
outstr+=('xM={0:8.3f} {1:8.3f}, fM={2:8.3e}\n'.format(xM[0],xM[1],obj2(xM)))
outstr+=('xH={0:8.3f} {1:8.3f}, fH={2:8.3e}\n'.format(xH[0],xH[1],obj2(xH)))

```

```

x_simplex[0][0][:]=xL; x_simplex[0][1][:]=xM; x_simplex[0][2][:]=xH;

fmin_simplex[0]=obj2(xL)

while (converged == 0):

    niter = niter+1

    xCE = 0.5*(xL + xM)

    # Carry out reflection operation

    xR,fR = reflection(xL,xM,xH,alpha,obj2)

    fL=obj2(xL); fM=obj2(xM); fH=obj2(xH);

    if ((fL < fR) and (fR < fM)):

        # update the simplex points

        xH = xM

        xM = xR

        xCE = 0.5*(xL + xM)

        converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

        outstr+=('Reflection carried out for iteration {0:3d}\n'.format(niter))

    elif (fR < fL):

        # carry out expansion

        xE,fE = expansion(xCE,xR,gamma,obj2)

        oldxL = xL; oldxM = xM; oldxH = xH;

        if (fE < fR):

            xL = xE

            xM = oldxL

            xH = oldxM

        else:

            xL = xR

            xM = oldxL

            xH = oldxM

        xCE = 0.5*(xL + xM)

        converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

```

```
outstr+='Expansion carried out for iteration {0:3d}\n'.format(niter))
```

```
elif ((fR > fM) and (fR < fH)):
```

```
# carry out outside contraction
```

```
xOC,fOC = outside_contraction(xCE,xR,beta,obj2)
```

```
if (fOC < fR):
```

```
# update simplex points based on three smallest function values
```

```
fp[0]=fL; fp[1]=fM; fp[2]=fOC
```

```
xp = np.zeros((3,2),dtype='float64')
```

```
xp[0,:]=xL; xp[1,:]=xM; xp[2,:]=xOC
```

```
# update simplex with three smallest function values
```

```
ipos = np.argsort(fp)
```

```
xL = xp[ipos[0],:]
```

```
xM = xp[ipos[1],:]
```

```
xH = xp[ipos[2],:]
```

```
else:
```

```
# carry out shrinking operation
```

```
newxM,newxH = shrinking(xL,xM,xH,rho,obj2)
```

```
xM = newxM
```

```
xH = newxH
```

```
xCE = 0.5*(xL + xM)
```

```
converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)
```

```
outstr+='Outside contraction carried out for iteration {0:3d}\n'.format(niter))
```

```
elif (fR > fH):
```

```
# carry out inside contraction
```

```
xIC,fIC = inside_contraction(xCE,xR,beta,obj2)
```

```
if (fIC < fH):
```

```

# update simplex points based on three smallest function values

fp[0]=fL; fp[1]=fM; fp[2]=fC

xp = np.zeros((3,2),dtype='float64')

xp[0,:]=xL; xp[1,:]=xM; xp[2,:]=xLC


# update simplex with three smallest function values

ipos = np.argsort(fp)

xL = xp[ipos[0],:]

xM = xp[ipos[1],:]

xH = xp[ipos[2],:]


else:


# carry out shrinking operation

newxM,newxH = shrinking(xL,xM,xH,rho,obj2)

xM = newxM

xH = newxH


xCE = 0.5*(xL + xM)

converged = check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj2)

outstr+=('Inside contraction carried out for iteration {0:3d}\n'.format(niter))


outstr+=('After {0:3d} iterations, simplex is given by:\n'.format(niter))

outstr+=('xL={0:8.3f} {1:8.3f}, fL={2:8.3e}\n'.format(xL[0],xL[1],obj2(xL)))

outstr+=('xM={0:8.3f} {1:8.3f}, fM={2:8.3e}\n'.format(xM[0],xM[1],obj2(xM)))

outstr+=('xH={0:8.3f} {1:8.3f}, fH={2:8.3e}\n'.format(xH[0],xH[1],obj2(xH)))

x_simplex[niter][0,:]=xL; x_simplex[niter][1,:]=xM; x_simplex[niter][2,:]=xH;

fmin_simplex[niter]=obj2(xL)


# end while loop


outstr+=('NM simplex converged with tol={0:6.3e} after {1:3d} iterations\n'.format(tol,niter))

outstr+=('Minimum f={0:8.3e} at x= {1:8.3e} {2:8.3e}\n'.format(obj2(xL),xL[0],xL[1]))


st.session_state.Calcs=outstr

st.session_state.x_simplex=x_simplex

st.session_state.fmin_simplex=fmin_simplex

```



```

st.session_state.niter=niter

st.session_state.xMin=x_simplex[niter][0][0]

st.session_state.yMin=x_simplex[niter][0][1]

st.session_state.fMin=fmin_simplex[niter]


print('finished Nelder-Mead')


# Function to create and update the plot for Nelder Mead Simplex
def create_plot():

    # Run the Nelder-Mead Simplex
    nelder_mead()

    x_simplex=st.session_state.x_simplex
    fmin_simplex=st.session_state.fmin_simplex
    niter=st.session_state.niter


    # create initial NM simplex
    x_0 = st.session_state.x_0
    y_0 = st.session_state.y_0
    x0=np.array([x_0,y_0])
    c = st.session_state.c
    xL,xM,xH = create_simplex(x0,c,obj2)


    plt.figure()

    # now plot out solution
    X, Y = np.meshgrid(np.linspace(0, 1, 101), np.linspace(0, 1, 101))
    Z = sixhump_func(X,Y)


    # plot out contours
    plt.ylabel('y')
    plt.xlabel('x')

    plt.title('Nelder Mead search on Six Hump Camel Back Function')

    CS = plt.contour(X, Y, Z, levels=[-1.0, -0.5, 0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0])

    xout=np.linspace(0,1,2); yout = np.linspace(0,1,2)

    for i in range(niter-1):

        xout[0]=x_simplex[i][0][0]; yout[0]=x_simplex[i][0][1];

        xout[1]=x_simplex[i+1][0][0]; yout[1]=x_simplex[i+1][0][1];

        plt.scatter(xout[0],yout[0],c='k',marker='o')

```

```

plt.plot(xout,yout,'k-')

plt.scatter(xout[1],yout[1],c='r',marker='o',)

return plt

# Function to create plots for Nelder Mead Simplex used in animation
def animate_plot():

    # Run the Nelder-Mead Simplex
    nelder_mead()

    x_simplex=st.session_state.x_simplex
    fmin_simplex=st.session_state.fmin_simplex
    niter=st.session_state.niter

    # create initial NM simplex
    x_0 = st.session_state.x_0
    y_0 = st.session_state.y_0
    x0=np.array([x_0,y_0])
    c = st.session_state.c
    xL,xM,xH = create_simplex(x0,c,obj2)

    plt.figure()

    # now plot out solution
    X, Y = np.meshgrid(np.linspace(0, 1, 101), np.linspace(0, 1, 101))
    #  $Z = 100*((Y-X**2)**2)+(1-X)**2$ 
    Z = sixhump_func(X,Y)

    # plot out contours
    plt.ylabel('y')
    plt.xlabel('x')
    plt.title('Nelder Mead search on Six Hump Camel Back Function')
    CS = plt.contour(X, Y, Z, levels=[-1.0, -0.5, 0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
    xout=np.linspace(0,1,2); yout = np.linspace(0,1,2)
    for i in range(st.session_state.ncalls):
        xout[0]=x_simplex[i][0][0]; yout[0]=x_simplex[i][0][1];
        xout[1]=x_simplex[i+1][0][0]; yout[1]=x_simplex[i+1][0][1];
        plt.scatter(xout[0],yout[0],c='k',marker='o')

```

```

plt.plot(xout,yout,'k-')

if (st.session_state.ncalls == st.session_state.niter - 2):
    plt.scatter(xout[1],yout[1],c='r',marker='o',)

return plt

#####

# Main program

#####

# Streamlit commands

st.title("Nelder-Mead Simplex Solution of Six Hump Camel Back Function")

st.write("This application enables you to experiment with Nelder-Mead solution of Six Hump Camel Bank Function")

st.write("Parameter settings")

# First row

cols = st.columns([1,1,1,1,1])

alpha = cols[0].number_input(chr(945),0.1,5.0,1.0)

st.session_state.alpha = alpha

beta = cols[1].number_input(chr(946),0.1,5.0,0.5,0.1)

st.session_state.beta = beta

gamma = cols[2].number_input(chr(947),0.1,5.0,1.0,0.1)

st.session_state.gamma = gamma

rho = cols[3].number_input(chr(961),0.1,5.0,0.5,0.1)

st.session_state.rho = rho

tol = cols[4].number_input("Tolerance",0.00001,0.01,0.0001)

st.session_state.tol = tol

# Second row

cols2 = st.columns([1,1,1,1,1])

```

```

x_0 = cols2[0].number_input("$x_0$",0.0,1.0,0.5,0.1)
st.session_state.x_0 = x_0

y_0 = cols2[1].number_input("$y_0$",0.0,1.0,0.5,0.1)
st.session_state.y_0 = y_0

c = cols2[2].number_input("c",0.1,5.0,0.5,0.1)
st.session_state.c = c

# Get initial solution of nelder mead simplex calculations
cols3 = st.columns([1,1,1])
nelder_mead()

cols3[0].text("x min="+"{:.4f}".format(st.session_state.xMin))
cols3[1].text("y min="+"{:.4f}".format(st.session_state.yMin))
cols3[2].text("f min="+"{:.4e}".format(st.session_state.fMin))

if 'plotgraph' not in st.session_state:
    st.session_state.plotgraph = 1
    if st.button("Plot Graph and Calculations"):
        figplt = create_plot()
        st.pyplot(figplt)
        st.write('Nelder Mead Calculations')
        stx.scrollableTextbox(st.session_state.Calcs,height=200)
else:
    figplt = create_plot()
    st.pyplot(figplt)
    st.write('Nelder Mead Calculations')
    stx.scrollableTextbox(st.session_state.Calcs,height=200)

# animate calculations
if st.button("Animate Graph"):
    st.session_state.ncalls = 0
    figplt = animate_plot()
    the_plot = st.pyplot(figplt)
    for i in range(st.session_state.niter - 2):
        st.session_state.ncalls += 1
        time.sleep(0.1)
        figplt = animate_plot()

```

```
the_plot.pyplot(figplt)
```

NMsimplex.py

```
import numpy as np
```

```
# create and sort initial simplex points
```

```
def create_simplex(x0,c,obj):
```

```
    xp_ini = np.zeros((3,2),dtype='float64'); fp_ini = np.zeros(3,dtype='float64');
```

```
    xp_ini[0][0] = x0[0]; xp_ini[0][1] = x0[1]; fp_ini[0] = obj(xp_ini[0,:]);
```

```
    xp_ini[1][0] = xp_ini[0][0] + 0.96593*c; xp_ini[1][1] = xp_ini[0][1] + 0.25882*c;
```

```
    fp_ini[1] = obj(xp_ini[1,:])
```

```
    xp_ini[2][0] = xp_ini[0][0] + 0.25882*c; xp_ini[2][1] = xp_ini[0][1] + 0.96593*c;
```

```
    fp_ini[2] = obj(xp_ini[2,:])
```

```
    ipos = np.argsort(fp_ini)
```

```
    xL = xp_ini[ipos[0],:]
```

```
    xM = xp_ini[ipos[1],:]
```

```
    xH = xp_ini[ipos[2],:]
```

```
    return xL,xM,xH
```

```
# Reflection operation
```

```
def reflection(xL,xM,xH,alpha,obj):
```

```
    # calculate centroid excluding worst point (biggest function value)
```

```
    xCE = 0.5*(xL + xM)
```

```
    # calculate reflection point
```

```
    xR = (1+alpha)*xCE - alpha*xH
```

```
    fR = obj(xR)
```

```
    return xR,fR
```

```
# end reflection
```

```
# Expansion operation
```

```
def expansion(xCE,xR,gamma,obj):
```

```
    # calculate expansion point
```

```
    xE = gamma*xR + (1-gamma)*xCE
```

```

fE = obj(xE)

return xE,fE

# end expansion

# Outside Contraction operation
def outside_contraction(xCE,xR,beta,obj):

    # calculate contraction point
    xOC = xCE + beta*(xR-xCE)
    fOC = obj(xOC)

    return xOC,fOC

# end outside contraction

# Inside Contraction operation
def inside_contraction(xCE,xR,beta,obj):

    # calculate contraction point
    xIC = xCE - beta*(xR-xCE)
    fIC = obj(xIC)

    return xIC,fIC

# end inside contraction

# Shrinking operation
def shrinking(xL,xM,xH,rho,obj):

    # calculate shrinkage points about xL
    newxM = xL + rho*(xM-xL)
    newxH = xL + rho*(xH-xL)

    return newxM,newxH

# end shrinking

```

```

# check convergence

def check_convergence(xL,xM,xH,xCE,tol,niter,maxiters,obj):

    std = (obj(xL)-obj(xCE))**2 + (obj(xM)-obj(xCE))**2 + (obj(xH)-obj(xCE))**2

    std=np.sqrt(std)/3.0

    converged = 0

    if ((std < tol) or (niter==(maxiters-1))):

        converged = 1

    return converged

# end check_convergence

```

v2st rosenbrock 2DNMSimplex

v2st sixhump 2DNMSimplex

Nelder-Mead Simplex Algorithm Program

This application enables you to experiment with Nelder-Mead solution of 2-D equations

Objective Function

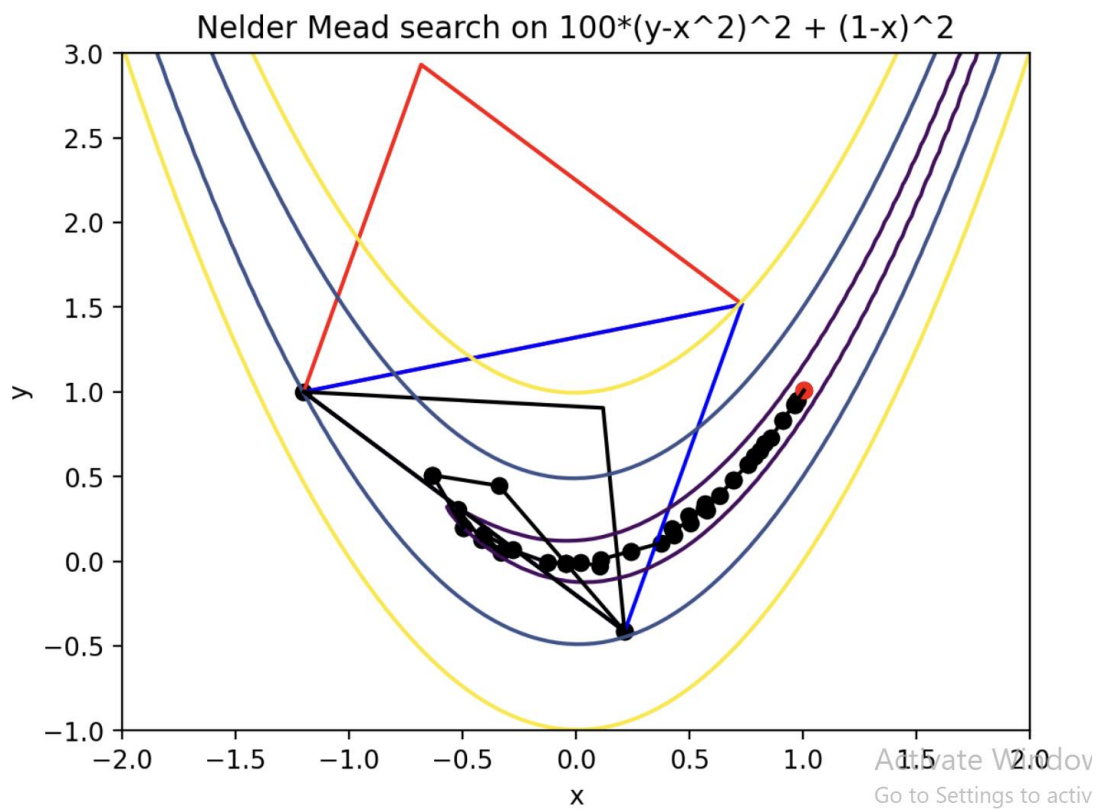
100*(y-x^2)^2 + (1-x)^2

Parameter settings

α	β	γ	ρ	Tolerance
1.00 - +	0.50 - +	2.00 - +	0.50 - +	0.00 - +

x ₀	y ₀	c
-1.20 - +	1.00 - +	2.00 - +

x min=1.0041	y min=1.0084	f min=2.2311e-05
--------------	--------------	------------------



Nelder Mead Calculations

Nelder-Mead Simplex Parameters:

 $x_0 = 0.500 \ 0.500$, $f_0 = 0.000$

$c=0.50$ $\alpha=1.00$ $\gamma=1.00$ $\beta=0.50$ $\rho=0.50$ $\text{tol}=0.00010$

Initial simplex

$x_L = 0.500 \ 0.500$, $f_L = 0.000\text{e}+00$

$x_M = 0.629 \ 0.983$, $f_M = 1.177\text{e}+00$

$x_H = 0.983 \ 0.629$, $f_H = 3.256\text{e}+00$

Animate Graph

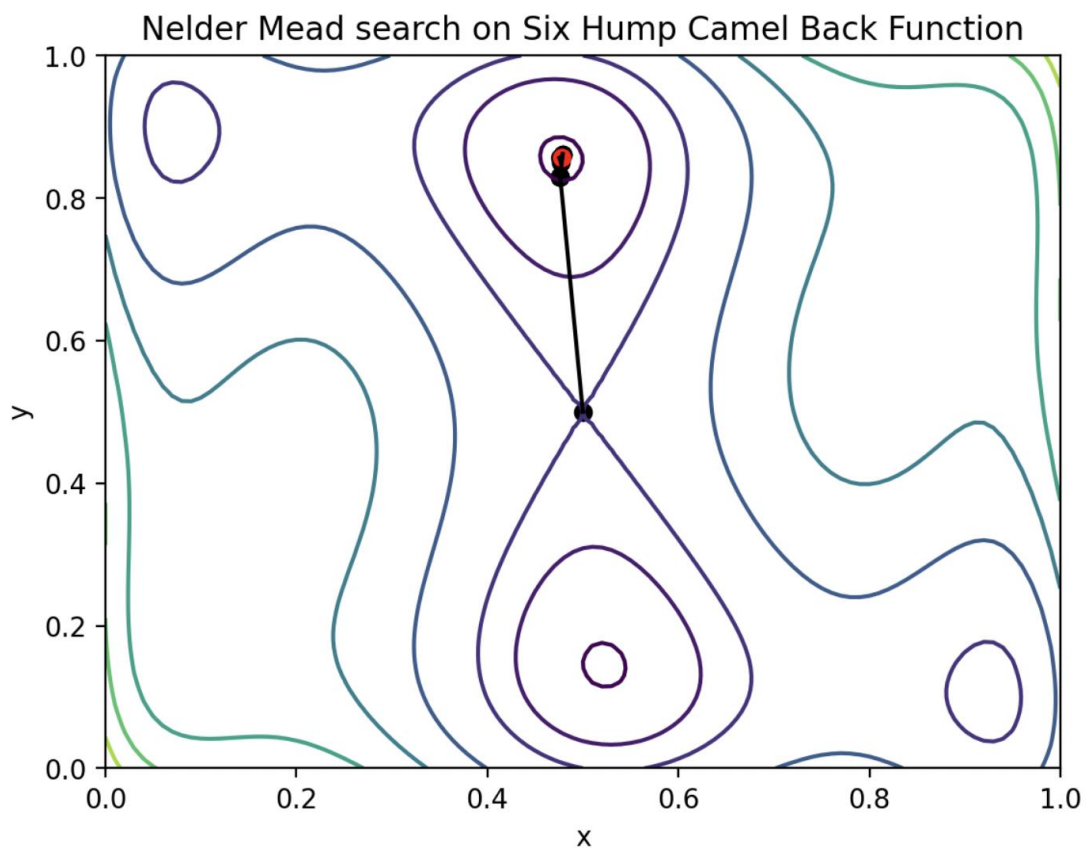
Nelder-Mead Simplex Solution of Six Hump Camel Back Function

This application enables you to experiment with Nelder-Mead solution of Six Hump Camel Bank Function

Parameter settings

α	β	γ	ρ	Tolerance
1.00 - +	0.50 - +	1.00 - +	0.50 - +	0.00 - +
x_0	y_0	c		
0.50 - +	0.50 - +	0.50 - +		

x min=0.4776 y min=0.8576 f min=-1.0316e+00



Plotly directory: examples using the 'plotly' interactive graphics package. This is more interactive than the matplotlib package. These examples require the installation of the 'plotly' package. I found this easier for 3D surface plotting than 'Altair'.

```
# 2dscatterplotly.py
```

```
import streamlit as st
```

```
import pandas as pd
```

```
# x and y given as array_like objects
```

```
import plotly.express as px
```

```
# Need to create a pandas dataframe of data for use in plotly
```

```
x=[0, 1, 2, 3, 4]
```

```
y=[0, 1, 4, 9, 16]
```

```
df = pd.DataFrame(
```

```
    {'x': x,
```

```
    'y': y,
```

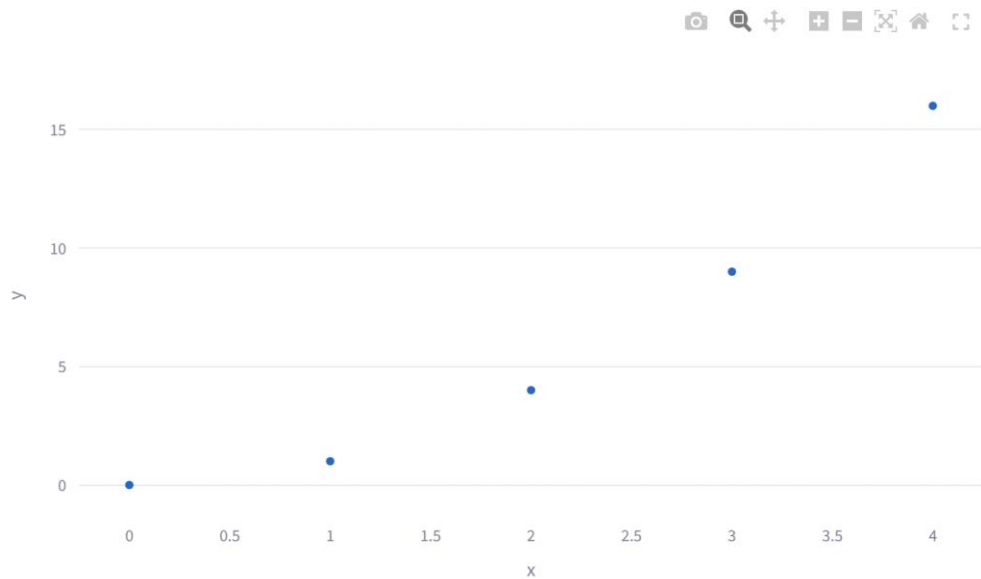
```
)
```

```
fig = px.scatter(df, x='x', y='y')
```

```
fig2 = px.line(df, x='x', y='y')
```

```
st.plotly_chart(fig)
```

```
st.plotly_chart(fig2)
```



3dmeshplotly1.py

```
import plotly.graph_objects as go
```

```
import numpy as np
```

```
import streamlit as st
```

```
# Download data set from plotly repo
```

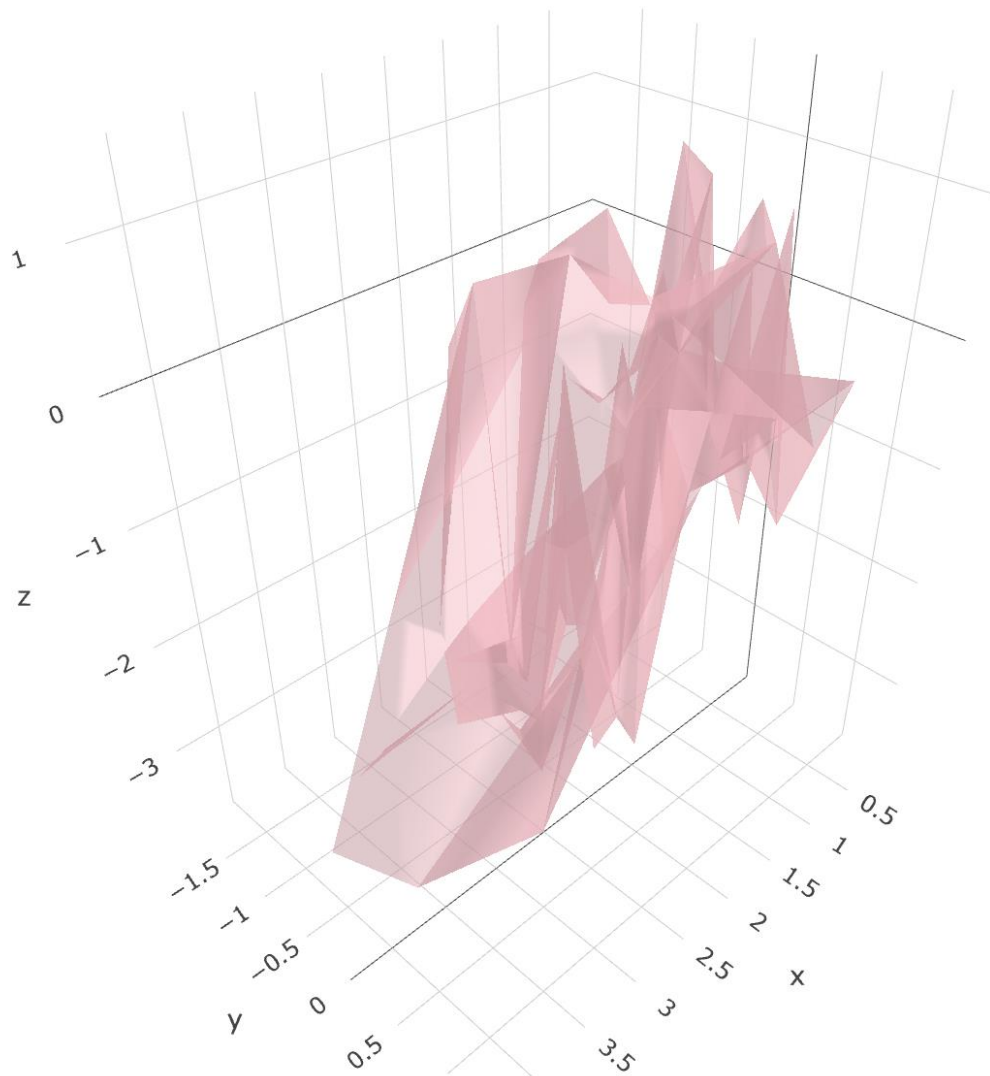
```
pts = np.loadtxt(np.DataSource().open('https://raw.githubusercontent.com/plotly/datasets/master/mesh_dataset.txt'))
```

```
x, y, z = pts.T
```

```
fig = go.Figure(data=[go.Mesh3d(x=x, y=y, z=z, color='lightpink', opacity=0.50)])
```

```
fig.show()
```

```
st.plotly_chart(fig)
```



```
# 3dscatterplotly1.py
```

```
import streamlit as st
```

```
import plotly.graph_objects as go
```

```
import numpy as np
```

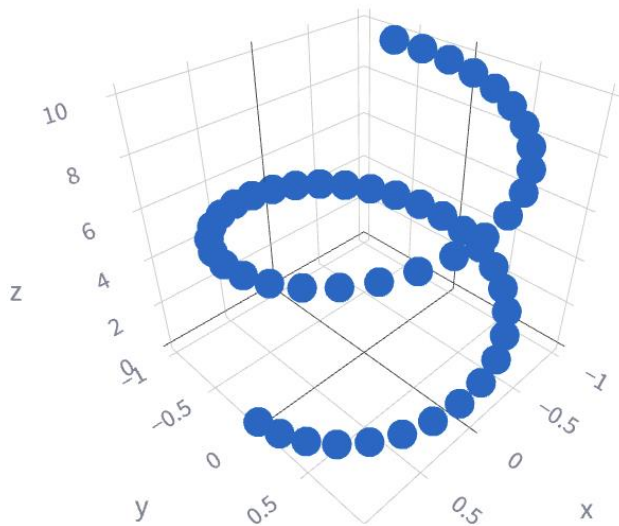
```
# Helix equation
```

```
t = np.linspace(0, 10, 50)
```

```
x, y, z = np.cos(t), np.sin(t), t
```

```
fig = go.Figure(data=[go.Scatter3d(x=x, y=y, z=z,  
                                   mode='markers')])
```

```
st.plotly_chart(fig)
```



3dsurfaceplotly1.py

```
import plotly.graph_objects as go
```

```
import numpy as np
```

```
import streamlit as st
```

```
np.random.seed(1)
```

```
N = 70
```

```
x = (70*np.random.randn(N))
```

```
fig = go.Figure(data=[go.Mesh3d(x=(70*np.random.randn(N)),
```

```
                                y=(55*np.random.randn(N)),
```

```
                                z=(40*np.random.randn(N)),
```

```
                                opacity=0.5,
```

```
                                color='rgba(244,22,100,0.6)'
```

```
                                ))
```

```
fig.update_layout(
```

```
    scene = dict(
```

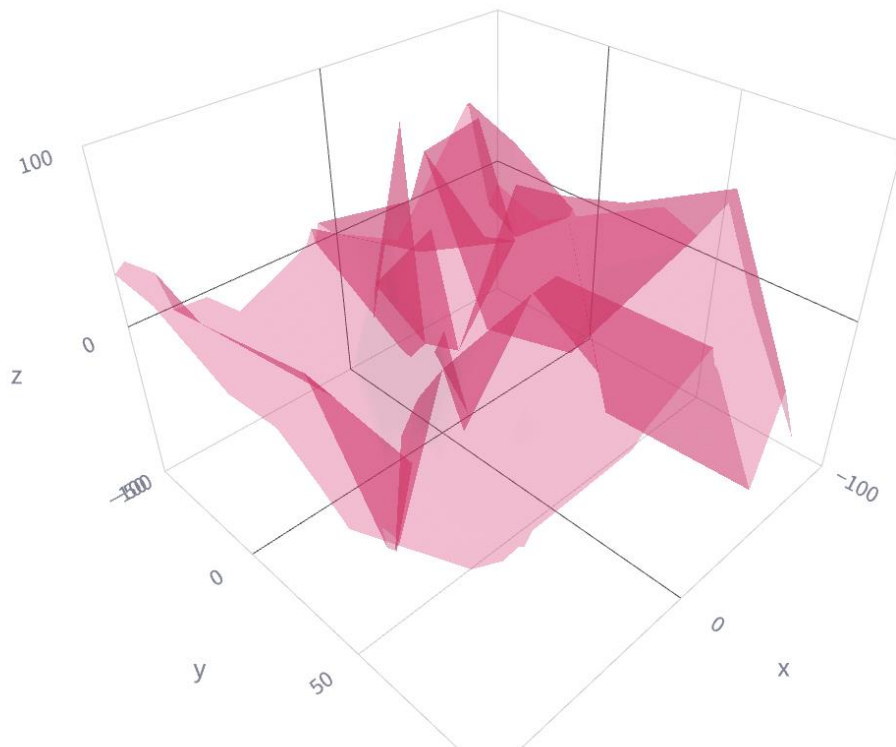
```
        xaxis = dict(nticks=4, range=[-100,100],),
```

```
        yaxis = dict(nticks=4, range=[-50,100],),
```

```
        zaxis = dict(nticks=4, range=[-100,100],),),
```

```
width=700,
margin=dict(r=20, l=10, b=10, t=10))
```

```
st.plotly_chart(fig)
```



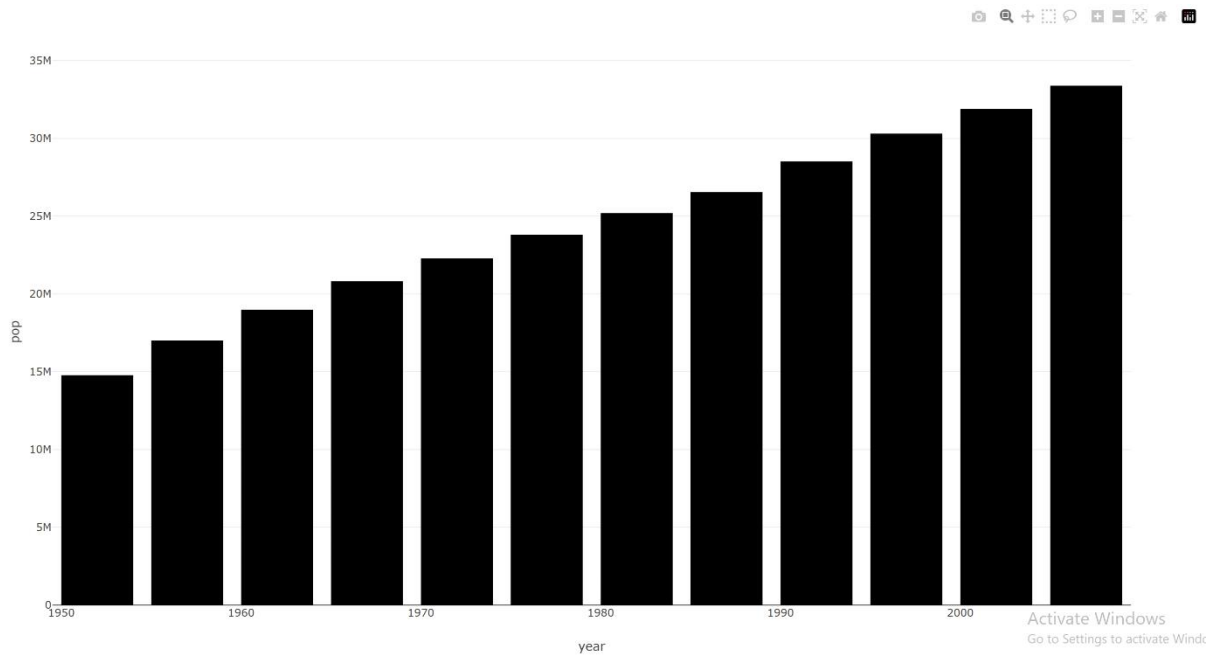
```
# barplotly1.py
```

```
import plotly.express as px
```

```
data_canada = px.data.gapminder().query("country == 'Canada'")
```

```
fig = px.bar(data_canada, x='year', y='pop')
```

```
fig.show()
```



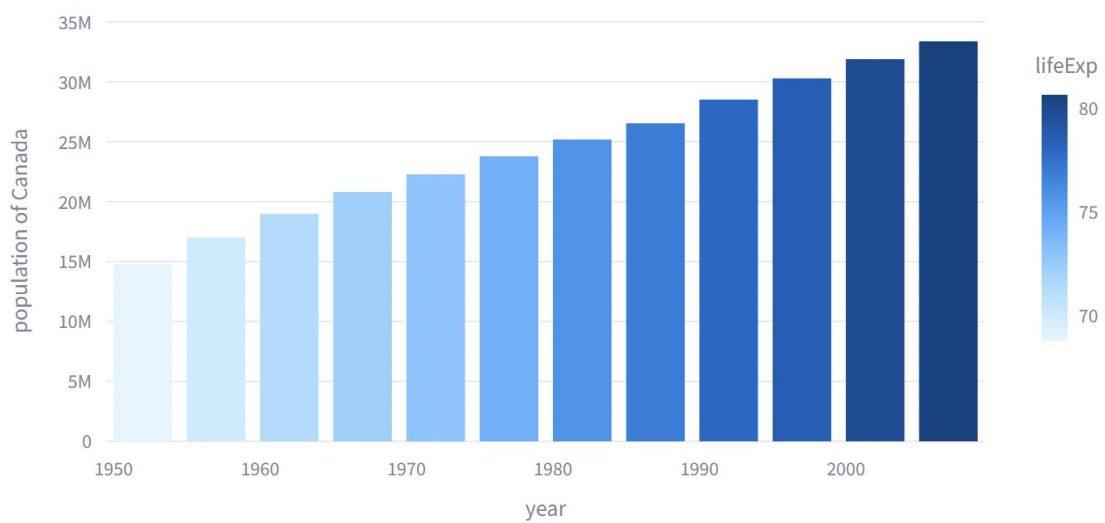
barplotly3.py

```
import plotly.express as px
import streamlit as st

df = px.data.gapminder().query("country == 'Canada'")

fig = px.bar(df, x='year', y='pop',
             hover_data=['lifeExp', 'gdpPercap'], color='lifeExp',
             labels={'pop': 'population of Canada'}, height=400)

st.plotly_chart(fig)
```



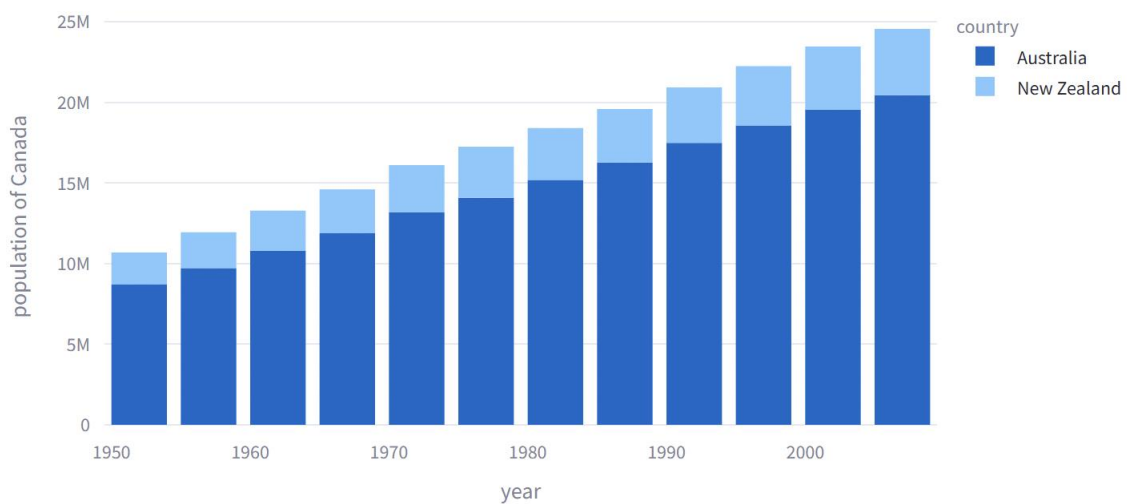
barplotly4.py

```
import plotly.express as px
```

```
import streamlit as st

df = px.data.gapminder().query("continent == 'Oceania'")
fig = px.bar(df, x='year', y='pop',
             hover_data=['lifeExp', 'gdpPercap'], color='country',
             labels={'pop': 'population of Canada'}, height=400)

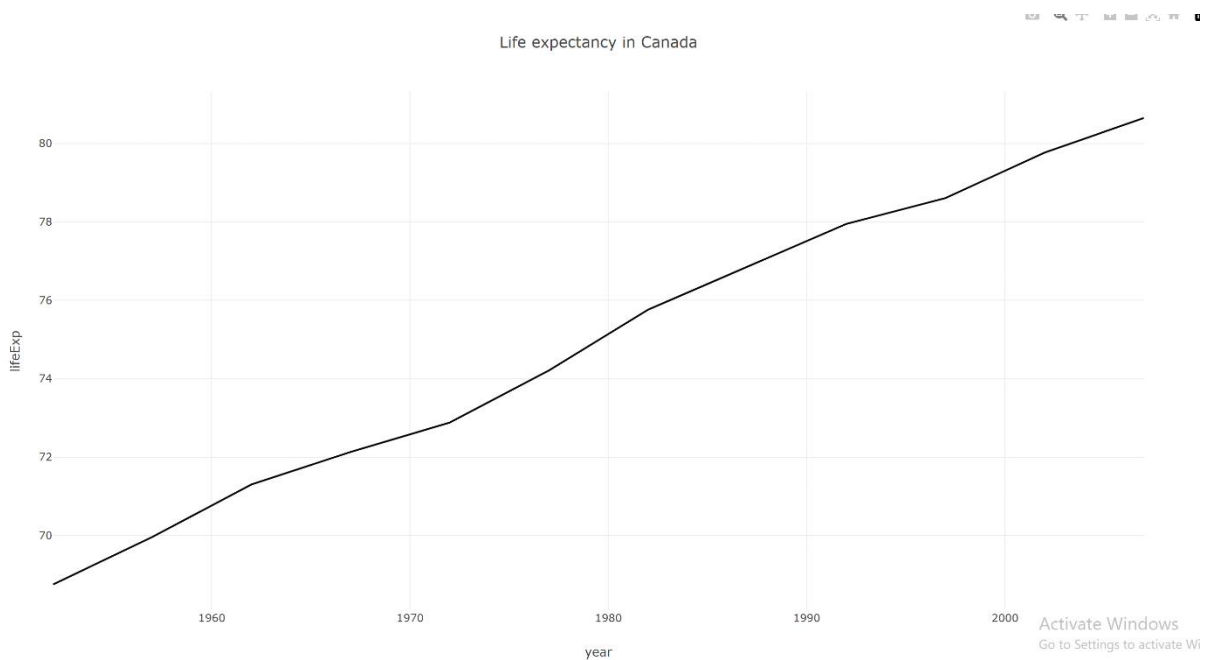
st.plotly_chart(fig)
```



lineplotly1.py

```
import plotly.express as px

df = px.data.gapminder().query("country=='Canada'")
fig = px.line(df, x="year", y="lifeExp", title='Life expectancy in Canada')
fig.show()
```



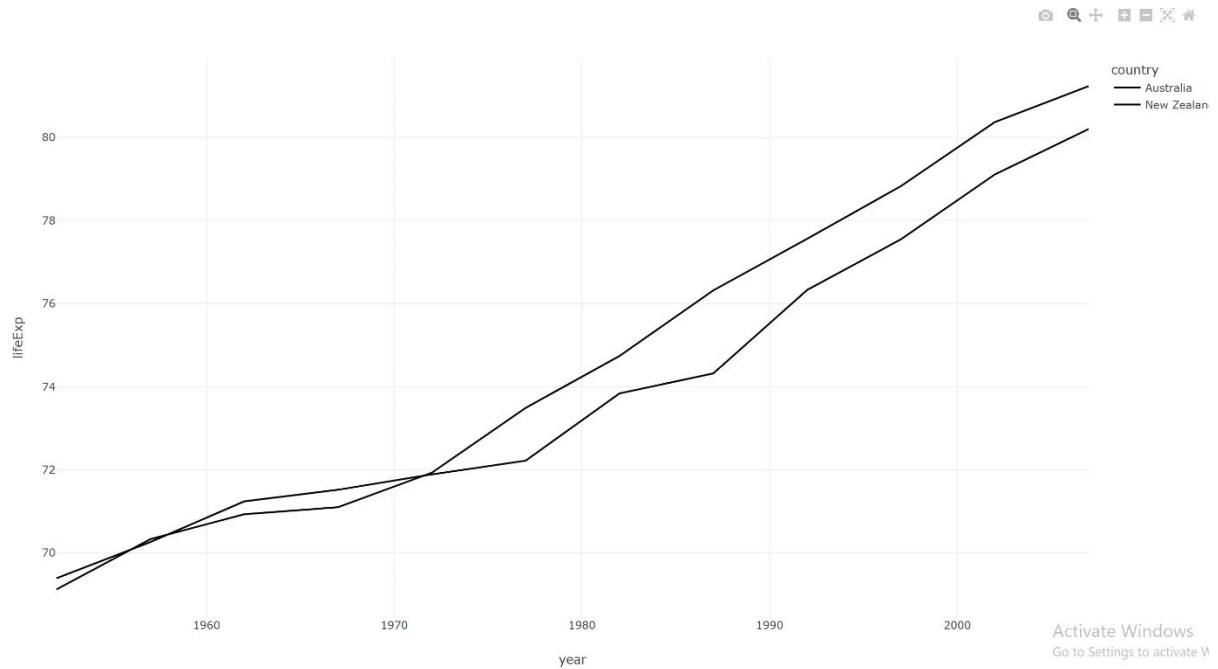
lineplotly2.py

```
import plotly.express as px

df = px.data.gapminder().query("continent=='Oceania'")

fig = px.line(df, x="year", y="lifeExp", color='country')

fig.show()
```



multipleaxes.py

```
import streamlit as st

import numpy as np

import math #needed for definition of pi

import plotly.graph_objects as go
```

```
x = np.arange(1,11)
```

```
y1 = np.exp(x)
```

```
y2 = np.log(x)
```

```
trace1 = go.Scatter(
```

```
    x = x,
```

```
    y = y1,
```

```
    name = 'exp'
```

```
)
```

```
trace2 = go.Scatter(
```

```
    x = x,
```

```
    y = y2,
```

```
    name = 'log',
```

```
    yaxs = 'y2'
```

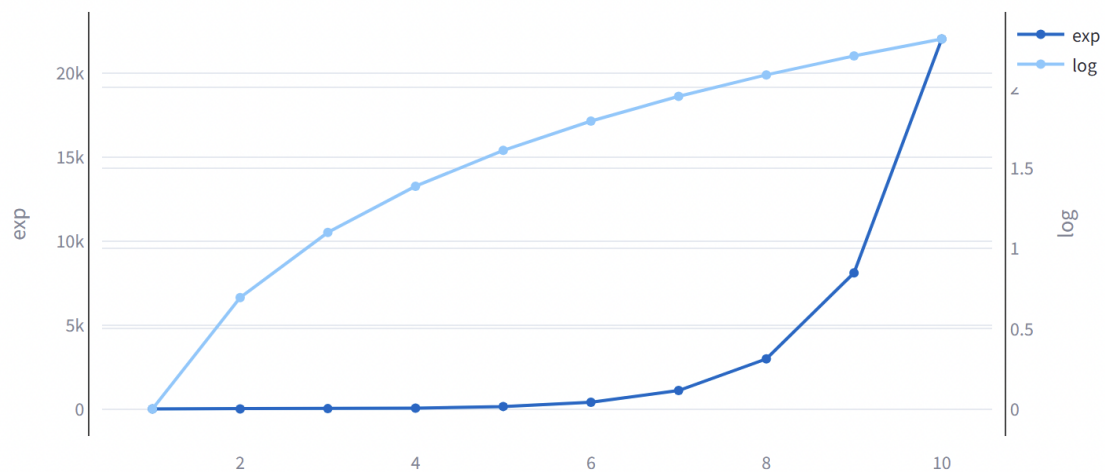


```

)
data = [trace1, trace2]
layout = go.Layout(
    title = 'Double Y Axis Example',
    yaxis = dict(
        title = 'exp', zeroline=True,
        showline = True
    ),
    yaxis2 = dict(
        title = 'log',
        zeroline = True,
        showline = True,
        overlaying = 'y',
        side = 'right'
    )
)
fig = go.Figure(data=data, layout=layout)
st.plotly_chart(fig)

```

Double Y Axis Example



surfplotly2.py

```

import plotly.graph_objects as go
import pandas as pd
import numpy as np
import streamlit as st

```

```
# Read data from a csv
z_data = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/api_docs/mt_bruno_elevation.csv')

z = z_data.values

sh_0, sh_1 = z.shape

x, y = np.linspace(0, 1, sh_0), np.linspace(0, 1, sh_1)

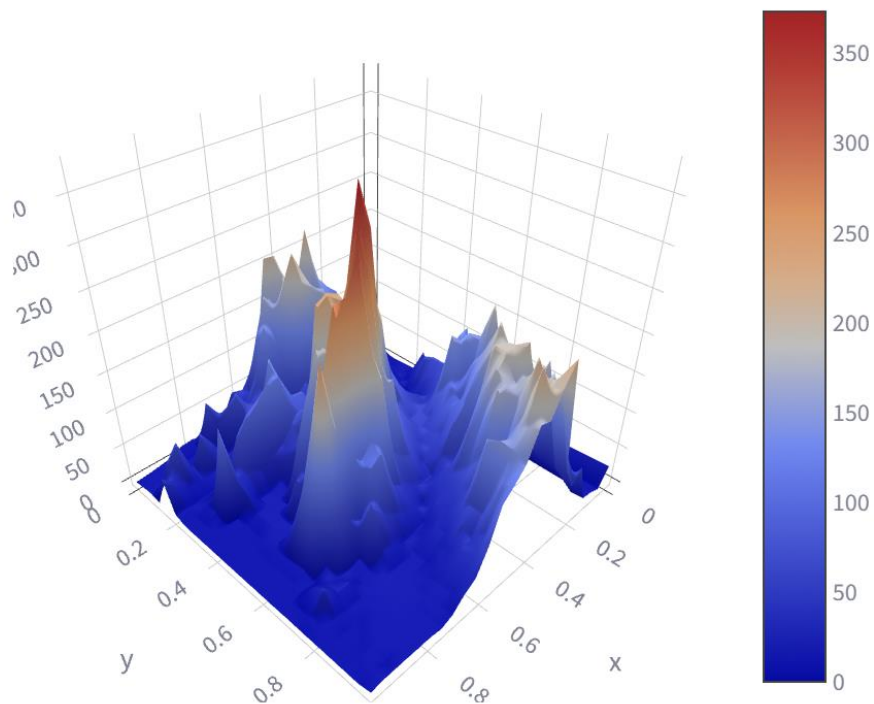
fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])

fig.update_layout(title='Mt Bruno Elevation', autosize=False,
                    width=500, height=500,
                    margin=dict(l=65, r=50, b=65, t=90))

st.plotly_chart(fig)
```



Mt Bruno Elevation



```
# v2sineplot.py

import streamlit as st

import numpy as np

import math #needed for definition of pi

import plotly.graph_objects as go

xpoints = np.arange(0, math.pi*2, 0.05)
```

```

y1 = np.sin(xpoints)
y2 = np.cos(xpoints)
trace0 = go.Scatter(
    x = xpoints,
    y = y1,
    name='Sine'
)
trace1 = go.Scatter(
    x = xpoints,
    y = y2,
    name = 'cos'
)
data = [trace0, trace1]
layout = go.Layout(title = "Sine and cos", xaxis = {'title':'angle'}, yaxis = {'title':'value'})
fig = go.Figure(data = data, layout = layout)
st.plotly_chart(fig)

```

```

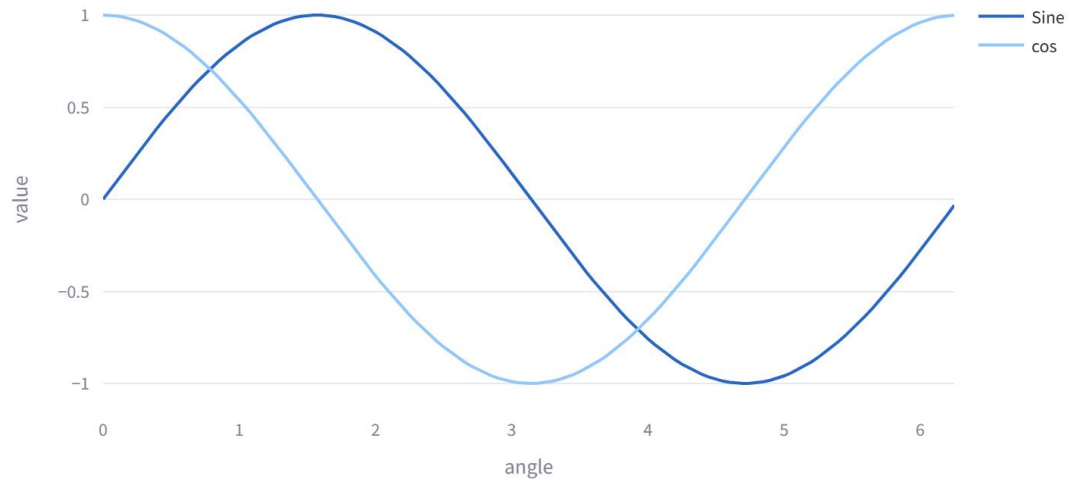
layout2 = go.Layout(
    title = "Sine and cos",
    xaxis = dict(
        title = 'angle',
        showgrid = True,
        zeroline = True,
        showline = True,
        showticklabels = True,
        gridwidth = 1
    ),
    yaxis = dict(
        showgrid = True,
        zeroline = True,
        showline = True,
        gridcolor = '#bdbdbd',
        gridwidth = 2,
        zerolinecolor = '#969696',
        zerolinewidth = 2,
        linecolor = '#636363',
        linewidth = 2,
        title = 'VALUE',

```

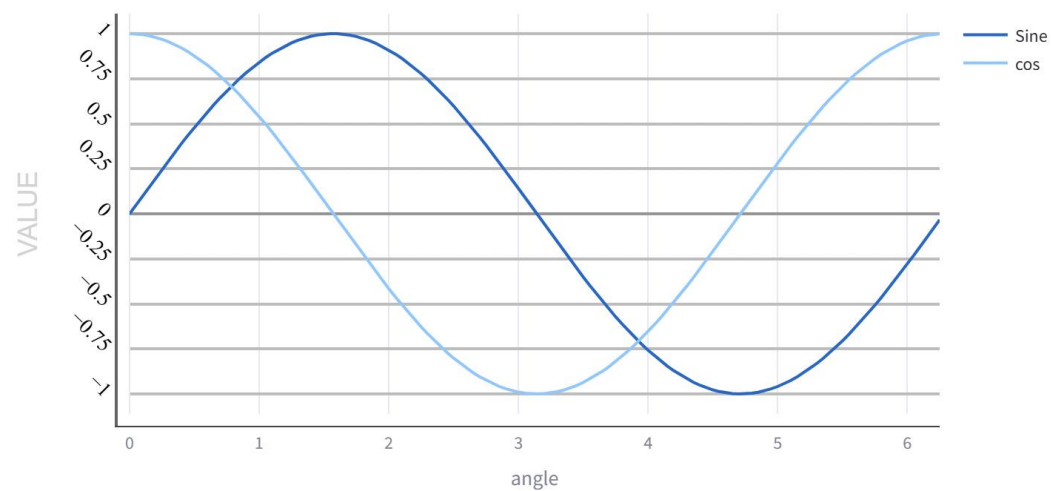
```
titlefont = dict(
    family = 'Arial, sans-serif',
    size = 18,
    color = 'lightgrey'
),
showticklabels = True,
tickangle = 45,
tickfont = dict(
    family = 'Old Standard TT, serif',
    size = 14,
    color = 'black'
),
tickmode = 'linear',
tick0 = 0.0,
dtick = 0.25
)
)

fig2 = go.Figure(data = data, layout = layout2)
st.plotly_chart(fig2)
```

Sine and cos



Sine and cos



Slider directory: simple example of using a slider widget.

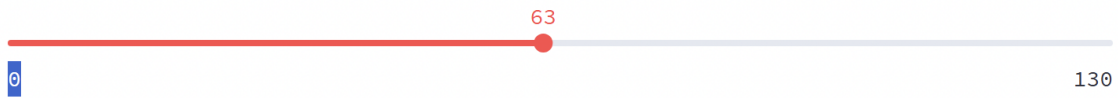
```
# slider.py
```

```
import streamlit as st
```

```
age = st.slider("How old are you?", 0, 130, 25)
```

```
st.write("I'm ", age, "years old")
```

How old are you?



I'm 63 years old

State directory: examples how to use `st.session_state` to store information between refreshes when variables change. It is very important to understand how to use `st.session_state` for applications.

state1.py

```
import streamlit as st

st.title('Counter Example')

count = 0

increment = st.button('Increment')

if increment:
    count += 1

st.write('Count = ', count)
```

Counter Example

Increment

Count = 0

counter_formsandcallbacks.py

```
import streamlit as st

import datetime

st.title('Counter Example')

if 'count' not in st.session_state:

    st.session_state.count = 0

    st.session_state.last_updated = datetime.time(0,0)
```

```
def update_counter():

    st.session_state.count += st.session_state.increment_value

    st.session_state.last_updated = st.session_state.update_time


with st.form(key='my_form'):

    st.time_input(label='Enter the time', value=datetime.datetime.now().time(), key='update_time')

    st.number_input('Enter a value', value=0, step=1, key='increment_value')

    submit = st.form_submit_button(label='Update', on_click=update_counter)


st.write('Current Count = ',st.session_state.count)

st.write('Last Updated = ',st.session_state.last_updated)
```

Counter Example

Enter the time

14:10

Enter a value

0

-

+

Update

Current Count = 0

Last Updated = 00:00:00

```
# counter_state.py

import streamlit as st

st.title('Counter Example')

if 'count' not in st.session_state:

    st.session_state.count = 0


increment = st.button('Increment')

if increment:

    st.session_state.count += 1


st.write('Count = ',st.session_state.count)
```

Counter Example

Increment

Count = 0

```
# counter_state_callback.py
```

```
import streamlit as st
```

```
st.title('Counter Example using Callbacks')
```

```
if 'count' not in st.session_state:
```

```
    st.session_state.count = 0
```

```
def increment_counter():
```

```
    st.session_state.count += 1
```

```
st.button('Increment', on_click=increment_counter)
```

```
st.write('Count = ', st.session_state.count)
```

Counter Example using Callbacks

Increment

Count = 0

```
# counter_state_callback_withargs.py
```

```
import streamlit as st
```

```
st.title('Counter Example using Callbacks with args')
```

```
if 'count' not in st.session_state:
```

```
    st.session_state.count = 0
```

```
increment_value = st.number_input('Enter a value', value=0, step=1)
```

```
def increment_counter(increment_value):
```



```
st.session_state.count += increment_value
```

```
increment = st.button('Increment', on_click=increment_counter,  
                       args=(increment_value, ))
```

```
st.write('Count = ',st.session_state.count)
```

Counter Example using Callbacks with args

Enter a value

0

- +

Increment

Count = 0

```
# counter_state_callback_withkwargs.py
```

```
import streamlit as st
```

```
st.title('Counter Example using Callbacks with kwargs')
```

```
if 'count' not in st.session_state:
```

```
    st.session_state.count = 0
```

```
def increment_counter(increment_value=0):
```

```
    st.session_state.count += increment_value
```

```
def decrement_counter(decrement_value=0):
```

```
    st.session_state.count -= decrement_value
```

```
st.button('Increment', on_click=increment_counter,
```

```
          kwargs=dict(increment_value=5))
```

```
st.button('Decrement', on_click=decrement_counter,
```

```
          kwargs=dict(decrement_value=1))
```

```
st.write('Count = ',st.session_state.count)
```

Counter Example using Callbacks with kwargs

Increment

Decrement

Count = 0

```
# widget_state1.py
```

```
import streamlit as st
```

```
if 'celsius' not in st.session_state:
```

```
    # set the initial default value of the slider widget
```

```
    st.session_state.celsius = 50.0
```

```
st.slider(
```

```
    'Temperature in Celsius',
```

```
    min_value=-100.0,
```

```
    max_value=100.0,
```

```
    key='celsius')
```

```
st.write(st.session_state.celsius)
```

Temperature in Celsius



50.0

Streamlit cheatsheet directory: there is a 'streamlit_cheatsheet.pdf' with these notes. This program implements the simple examples on this sheet in a single program.

```
# streamlit_cheatsheet.py
```

```
import streamlit as st
```

```
import numpy as np
```

```
import time
```

```
st.text('Hello world')
```

```
number = st.slider("Pick a number",0,100)

file = st.file_uploader("Pick a file")

date = st.date_input("Pick a date")

st.markdown('_Markdown_')

st.caption('Balloons. Hundreds of them ...')

st.latex(r''' e^{i\pi}+1=0''')

st.write('Most objects')

st.title('My title')

st.header('My header')

st.subheader('My sub')

st.code('for i in range(8): foo()')

st.json({'foo':'bar','fu':'ba'})

st.metric(label="Temp", value="273 K", delta="1.2 K")
```

```
st.image('./leaveoneout1.png')

col1,col2 = st.columns(2)

col1.write('Column 1')

col2.write('Column 2')

col1, col2, col3 = st.columns([3,1,1])

with col1:

    st.write('This is column 1')
```

```
tab1, tab2 = st.tabs(["Tab1", "Tab 2"])

tab1.write("this is tab 1")

tab2.write("this is tab 2")
```

```
with tab1:

    st.radio('Select one:', [1,2])
```

```
# st.stop()
```

```
with st.form(key='my_form'):

    username = st.text_input('Username')

    password = st.text_input('Password')

    st.form_submit_button('Login')
```

```
# personalize app for users
```

```
#if st.user.email == 'jane@email.com':
```

```

# display_jane_content()

#elif st.user.email == 'adam@foocopr.io':

# display_adam_content()

#else:

# st.write('Please contact us to get access')


# Display interactive widgets

st.button('Hit me')

# st.data_editor('Edit data') # data needs to be a pandas dataframe

st.checkbox('Check me out')

st.radio('Pick one:', ['nose', 'ear'])

st.selectbox('Select',[1,2,3])

st.multiselect('Multiselect',[1,2,3])

st.slider('Slide me',min_value=0, max_value=10)

st.select_slider('Slide to select',options=[1,'2'])

my_text = st.text_input('Enter some text')

print(my_text)

st.number_input('Enter a number')

st.text_area('Area for textual entry')

st.date_input('Date input')

st.time_input('Time entry')

data = 'hello world'

st.download_button('One the dl',data)

st.camera_input("----")

st.color_picker('Pick a color')


# Use widgets' returned values in variables

for i in range(int(st.number_input('Num:'))): print('Hello')


my_slider_val = st.slider('Quinn Mallory',1,88)

st.write(my_slider_val**2)


with st.chat_message("user"):

    st.write("Hello ")

    st.line_chart(np.random.randn(30,3))


# display a chat input widget

st.chat_input("Say something")

```

```

# display code
st.echo()

with st.echo():
    st.write('Code will be executed and printed')

# display progress and status
with st.spinner(text='In progress'):
    time.sleep(10)
    st.success('Done')

bar = st.progress(50)
time.sleep(3)
bar.progress(100)

```

Hello world

Pick a number



Pick a file



Drag and drop file here

Limit 200MB per file

Browse files

Pick a date

2025/02/05

Markdown

Balloons. Hundreds of them ...

$$e^{i\pi} + 1 = 0$$

Most objects

Timezone directory: simple example showing how to use timezone functionality in the pytz package.

trz1.py

```

import streamlit as st

from datetime import datetime

import pytz

```

```
st.write(datetime(2020, 1, 10, 10, 30, tzinfo=pytz.timezone("GMT")))
```

```
2020-01-10 10:30:00+00:00
```