SWAN Data Analysis

Session 0: Python Intro

Topics covered:

- Python installation (UNIX and Windows)
- Jupyter Notebook installation
- Demo:
    1. Jupyter Notebook basics
    2. Python basics
    3. Numpy basics
    4. Plotting basics

Prepared by Hrishikesh Shetaonkar

> Feel free to reach out if stuck anywhere:
> hrishikesh036@gmail.com * LinkedIn
>
> **You can find all tutorial sessions on my [Github](https://github.com/hrshe). *Fork it!***

Let's Start

1. Jupyter Notebook basics

Tasks

1. Help ==> Keyboard Shortcuts
2. Cell? ,
    - Command mode <*esc*> and Edit Mode <*return*>

3. Insert cell
    - Above: <*A*>
    - Below: <*B*>
4. Delete cell
    - Enter command mode and <*dd*>
5. Run cells:
    - simple run: <*cmd + return*>
    - run and select next cell: <*shift + return*>
    - run and insert a new cell: <*option + return*>

In [1]:

```
#Insert a new cell below and the delete it
```

In [2]:

```
# Try running this cell in different modes
# Note that the next line of code is irrelevant and is only for demonstration. Don't fret over it.
b'\xff\xfeH\x00e\x00l\x00l\x00o\x00 \x00U\x00n\x00i\x00v\x00e\x00r\x00s\x00e\x00'.decode('utf-16')
```

Out[2]:

```
'Hello Universe'
```

In [3]:

```
#A simple print statement
print("Hey there! I'm using Python Notebook.")
```

```
Hey there! I'm using Python Notebook.
```

In [4]:

```
#Interactive prompt
"Hey there! I'm using Python Notebook."
```

Out[4]:

```
"Hey there! I'm using Python Notebook."
```

In [5]:

```
# run bash commands: Just add '!' as a prefix
! ls ~/RRIProject
```

```
BITS                    Ooty                    gain_correction
GBD                     RRI                     pulsar-analysis-package
IUCAA                   Taipei                  readmeImages
Lulin                   das_dyn_dedisp_lin.f    referances
NCRA                    de-compression          spec
NCU                     de-dispersion           swan-analysis-demo
Old                     desh_mail               ts_integration_flag
```

In [6]:

```
! pwd
```

```
/Users/hrishikesh.s/SWAN/sessions/data_processing/swan-processing-sessions/session-0
```

2. Python basics

Tasks

2.1 import libraries

2.2 Data types

2.3 Type Casting

2.4 Dictionary

2.5 List

2.6 For loop

2.7 If - else

2.8 Input

Disclaimers!

- Python (programming in general) is vast and fast evolving
  - Don't wait to do a course
  - Pick up a project and start implementing
- Language is not important. Focus on the logic
  - Why many languages?
  - Focus on the logic
- Googling is a very important skill!

2.1 Import libraries

In [7]:

```python
import numpy as np
import warnings
import matplotlib.pyplot as plt
```

If you face errors in import, use *pip* to list all packages and check if the package is installed.

If not, you can use *pip* to install the package

In [8]:

```
#! pip3 list
```

In [9]:

```
#imports
#nan, nammean
#typecast to integers
```

2.2 Data Types

In [10]:

```
#data types
int_var = 4
float_var = 4.2
str_var1 = "This is a string"
str_var2 = 'This is also a string'
bool_var = True

print(f"Integer: {int_var}")
print(f"Float: {float_var}")
print(f"String 1: {str_var1}")
print(f"String 2: {str_var2}")
print(f"Boolean: {bool_var}")
```

```
Integer: 4
Float: 4.2
String 1: This is a string
String 2: This is also a string
Boolean: True
```

2.3 Type Casting

In [11]:

```
#Type Casting
print(f"This is now a float: {float(int_var)}")
print(f"This is now a integer: {int(float_var)}")

str_var3 = '347'
print(f"This is a : {type(str_var3)} with value {str_var3}")
str_var3 = int(str_var3)
print(f"String is now converted to: {type(str_var3)} with value {str_var3}")
```

```
This is now a float: 4.0
This is now a integer: 4
This is a : <class 'str'> with value 347
String is now converted to: <class 'int'> with value 347
```

More on implicit and explicit type casting: https://www.stackoftuts.com/python-3/typecasting-in-python/

2.4 Dictionary

In [12]:

```
#dictionaries ==> Is made up of key-value pairs
person = {
          "first_name": "Jane",
          "last_name": "Doe",
          "age":20,
          "graduated": True,
          "cgpa": 9.21
         }
```

In [13]:

```
#get all keys
person.keys()
```

Out[13]:

```
dict_keys(['first_name', 'last_name', 'age', 'graduated', 'cgpa'])
```

In [14]:

```
#get all values
person.values()
```

Out[14]:

```
dict_values(['Jane', 'Doe', 20, True, 9.21])
```

In [15]:

```
#fetch data
print("name of person: " + person.get("first_name"))
print("cgpa of person: " + str(person.get("cgpa")))
```

```
name of person: Jane
cgpa of person: 9.21
```

2.5 List

In [16]:

```
students = ["Alice", "Bob", "John", "Jane", "Pavan", "Yash", "Akhil"]
print(students)
```

```
['Alice', 'Bob', 'John', 'Jane', 'Pavan', 'Yash', 'Akhil']
```

In [17]:

```
students.append("Hrishi")
students
```

Out[17]:

```
['Alice', 'Bob', 'John', 'Jane', 'Pavan', 'Yash', 'Akhil', 'Hrishi']
```

In [18]:

```
students[0]
```

```
Out[18]: 'Alice'
```

```
In [19]: students[:4]
```

```
Out[19]: ['Alice', 'Bob', 'John', 'Jane']
```

```
In [20]: students[4:]
```

```
Out[20]: ['Pavan', 'Yash', 'Akhil', 'Hrishi']
```

```
In [21]: students[2:6]
```

```
Out[21]: ['John', 'Jane', 'Pavan', 'Yash']
```

```
In [22]: students[-3]
         # We generally use arr[-1] to access last element
```

```
Out[22]: 'Yash'
```

2.6 For loop

```
In [23]: for i in range(10):
             print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
In [24]: integers_array = []
         for i in range(10):
             integers_array.append(i**2)

         integers_array
```

```
Out[24]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [25]:

```
integers_array = [i**2 for i in range(15)]
integers_array
```

Out[25]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
```

In [26]:

```
# you can iterate over elements of a list directly without giving the index
for student in students:
    print(student)
```

```
Alice
Bob
John
Jane
Pavan
Yash
Akhil
Hrishi
```

In [27]:

```
students
```

Out[27]:

```
['Alice', 'Bob', 'John', 'Jane', 'Pavan', 'Yash', 'Akhil', 'Hrishi']
```

2.7 If - else

In [28]:

```
condition = True
if condition:
    print("executing true block")
else:
    print("executing false block")
# also works for 0 (false) and non zeros (true)
```

```
executing true block
```

In [29]:

```
float_var = 5.7
output = 0
#simple rounding logic
if float_var - int(float_var) < 0.5:
    output = int(float_var)
else:
    output = int(float_var) + 1

print("rounded: " + str(output))
```

```
rounded: 6
```

2.8 Input

In [30]:
```
input_str = input("Enter a value: ")
print(f"Value entered: {input_str}")
```

```
Enter a value:
Value entered:
```

3. Numpy basics

Tasks

3.1 Create Numpy Arrays

3.2 Accessing elements

3.3 2D numpy arrays

3.4 Numpy methods

3.5 Flagging data with *np.nan*

In [31]:
```
import numpy as np
```

3.1 Create Numpy Arrays

In [32]:
```
# create numpy arrays
list_arr = [1,2,3,4,5]
np.array(list_arr)
```

Out[32]:
```
array([1, 2, 3, 4, 5])
```

In [33]:
```
size = 10
np.zeros(size)
#similarly np.ones
```

Out[33]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [34]:
```
np.arange(10)
```

Out[34]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [35]:

```
# np.linspace(start,end, number of elements)
x_axis = np.linspace(0,10, 5)
x_axis
```

Out[35]:

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

3.2 Accessing elements

In [36]:

```
print(f"{x_axis[2]=}")
print(f"{x_axis[2:]=}")
print(f"{x_axis[2:4]=}")
```

```
x_axis[2]=5.0
x_axis[2:]=array([ 5. ,  7.5, 10. ])
x_axis[2:4]=array([5. , 7.5])
```

3.3 2D numpy arrays

In [37]:

```
# 2D arrays
np.ones((3,5))
```

Out[37]:

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

You can view 2D arrays as array of arrays

The first dim is rows, and second is columns... Everywhere!

Eg. no.ones((rows, cols))

```
                       columns
                 =========================>
            || [[23, 35,  0, 18, 10, 18, 24,  6],
            ||  [ 5, 44, 23, 44, 35, 29, 35, 22],
       rows ||  [41,  9, 41, 21, 23, 40, 41, 25],
            ||  [28, 36, 39, 30, 48, 19, 18, 22],
            \/  [38, 38, 24, 47, 30, 40, 22, 30]])
```

In [38]:

```
np_2d_array = np.arange(20).reshape(4,5)
np_2d_array
```

Out[38]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [39]:

```
# get number of rows and cols
np_2d_array.shape
```

Out[39]:

```
(4, 5)
```

In [40]:

```
print(f"rows: {np_2d_array.shape[0]}\n"
      f"columns: {np_2d_array.shape[1]}")
```

```
rows: 4
columns: 5
```

3.4 Numpy methods

In [41]:

```
np_1d_array = np.random.randint(50, size=20)
np_1d_array
```

Out[41]:

```
array([44, 15, 33, 46,  8, 41, 33, 35, 26,  4, 43, 47, 20, 20, 14, 14, 11,
       23, 24, 15])
```

Lets say we want to find mean of elements in np_1d_array. This can be achieved simply by:

In [42]:

```
np_1d_array.mean()
```

Out[42]:

```
25.8
```

- Similarly we have methods like *sum( ), std( ), max( ), argmax( )*...
- *np.loadtxt( )* and *np.savetxt( )* too some are useful numpy methods. You'll see an example of its usage towards the end of this notebook(section)

We can also take mean along rows and columns for 2D numpy arrays

In [43]:

```
np_2d_array = np.random.randint(50, size=40).reshape(5,-1)
np_2d_array
```

Out[43]:

```
array([[25, 23,  3, 43, 21, 27, 17, 10],
       [22, 37, 27, 14,  9, 30,  2, 23],
       [48, 38, 41, 16, 49, 30, 39,  5],
       [14, 48,  9, 33, 19, 41, 21, 35],
       [ 3, 31, 22, 31, 33,  9, 31, 19]])
```

In [44]:

```
np.mean(np_2d_array, axis=0)
```

Out[44]:

```
array([22.4, 35.4, 20.4, 27.4, 26.2, 27.4, 22. , 18.4])
```

In [45]:

```
np.mean(np_2d_array, axis=1)
```

Out[45]:

```
array([21.125, 20.5  , 33.25 , 27.5  , 22.375])
```

3.5 Flagging data with *np.nan*

While processing dat, you'll come across **missing data**. You might also need to **flag unwanted data** (eg. RFIs) so that this data is not included in your processing. Setting such data to ***np.nan*** helps a lot. Numpy also provides **inbuilt methods** to process data by **ignoring np.nan data**

We'll revisit the np_1d_array. But now, we'll flag data whose value is greater than 35. You can think of this as if you are flagging strong signals.

In [46]:

```
np_1d_array = np.random.randint(50, size=20)
```

In [47]:

```
# Approach 1: Naive

np_1d_array_copy = np_1d_array.copy()
#NaN is a float... So first need to convert from int to float
np_1d_array_copy = np.array(np_1d_array_copy, dtype = "float")
print(np_1d_array_copy)

# From some previous analysis, you know that data at index 5, 7 and 13 is bad and you have flagged that as well
np_1d_array_copy[5] = np.nan
np_1d_array_copy[7] = np.nan
np_1d_array_copy[13] = np.nan
print(np_1d_array_copy)

#Logic to flag strong signals:
for i in range(np_1d_array_copy.shape[0]):
    if np_1d_array_copy[i] > 35:
        np_1d_array_copy[i] = np.nan

print(np_1d_array_copy)


#Logic to calculate
temp_sum = 0
temp_count = 0
for i in range(np_1d_array_copy.shape[0]):
    if not np.isnan(np_1d_array_copy[i]):
        temp_sum = temp_sum + np_1d_array_copy[i]
        temp_count = temp_count +1

print(f"\nMean for unflagged data is: {str(temp_sum/temp_count)}")
```

```
[48.  7. 29.  0. 36.  2.  1.  4.  0. 28. 43.  3. 28. 12. 29.  5. 39. 46.
 49.  9.]
[48.  7. 29.  0. 36. nan  1. nan  0. 28. 43.  3. 28. nan 29.  5. 39. 46.
 49.  9.]
[nan  7. 29.  0. nan nan  1. nan  0. 28. nan  3. 28. nan 29.  5. nan nan
 nan  9.]

Mean for unflagged data is: 12.636363636363637
```

In [48]:

```
# Approach 2: "Pythonic"

np_1d_array = np.array(np_1d_array, dtype = "float")
print(np_1d_array)

np_1d_array[[5,7,13]] = np.nan
print(np_1d_array)

np_1d_array[np_1d_array > 35] = np.nan
print(np_1d_array)

print(f"\nMean for unflagged data is: {str(np.nanmean(np_1d_array))}")
```

```
[48.  7. 29.  0. 36.  2.  1.  4.  0. 28. 43.  3. 28. 12. 29.  5. 39. 46.
 49.  9.]
[48.  7. 29.  0. 36. nan  1. nan  0. 28. 43.  3. 28. nan 29.  5. 39. 46.
 49.  9.]
[nan  7. 29.  0. nan nan  1. nan  0. 28. nan  3. 28. nan 29.  5. nan nan
 nan  9.]

Mean for unflagged data is: 12.636363636363637
```

4. Plotting basics

Tasks

4.1 Scatter plot

4.2 Plotting sine and cosine

In [49]:

```
import matplotlib.pyplot as plt
# This import can be skipped as we have already imported previously once

%matplotlib inline
```

4.1 Scatter Plot

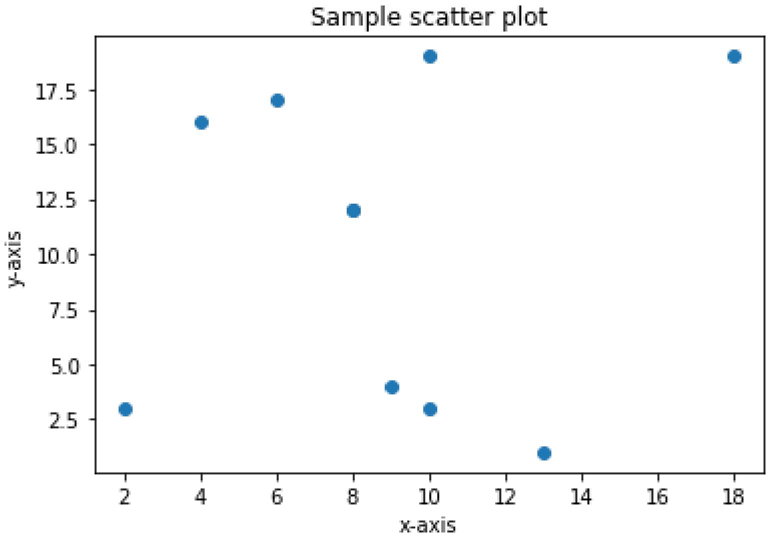
In [50]:

```
x_data = np.random.randint(20, size=10)
y_data = np.random.randint(20, size=10)

print(f"{x_data=}\n"
      f"{y_data=}\n")

plt.scatter(x_data, y_data)
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Sample scatter plot")
#plt.show()
# More info: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html
```

```
x_data=array([ 2, 18,  4,  6,  9, 10,  8, 10, 13,  8])
y_data=array([ 3, 19, 16, 17,  4, 19, 12,  3,  1, 12])
```

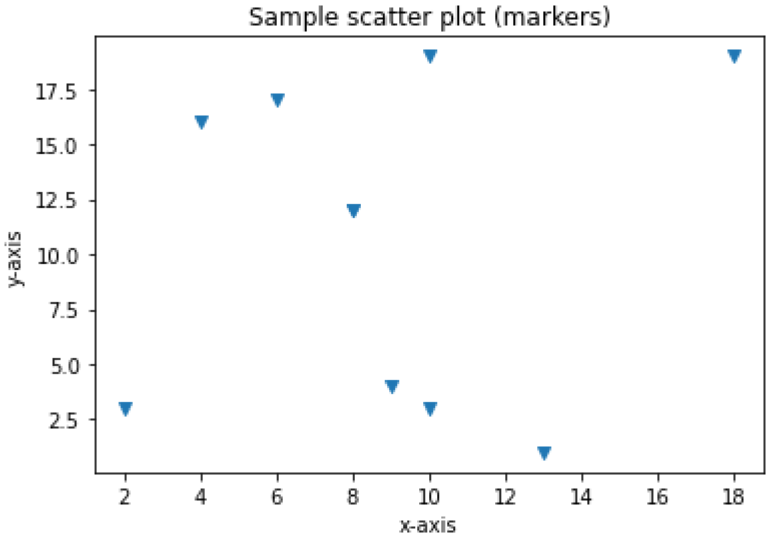
Out[50]:

```
Text(0.5, 1.0, 'Sample scatter plot')
```



```
In [51]: plt.scatter(x_data, y_data, marker='v')
         plt.xlabel("x-axis")
         plt.ylabel("y-axis")
         plt.title("Sample scatter plot (markers)")
         #plt.show()
         # More info: https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers
```

```
Out[51]: Text(0.5, 1.0, 'Sample scatter plot (markers)')
```



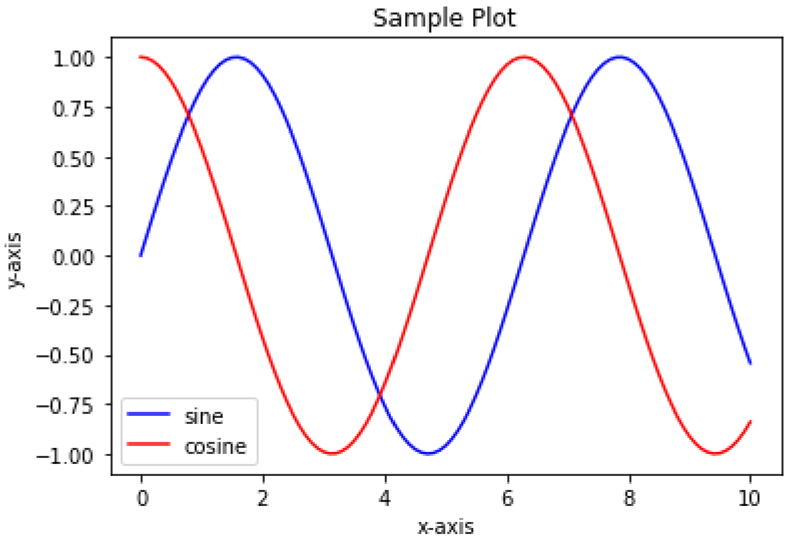
4.2 Plotting sine and cosine

In [52]:

```
x_data = np.linspace(0,10, 101)
y_sin_data = np.sin(x_data)
y_cos_data = np.cos(x_data)
plt.plot(x_data, y_sin_data, 'b')
plt.plot(x_data, y_cos_data, 'r')
plt.xlabel("x-axis")
plt.ylabel("y-axis")

plt.legend(['sine', 'cosine'])

plt.title("Sample Plot")
#plt.show()
# More info: https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers
```

Out[52]:

```
Text(0.5, 1.0, 'Sample Plot')
```



4.3 Plotting image

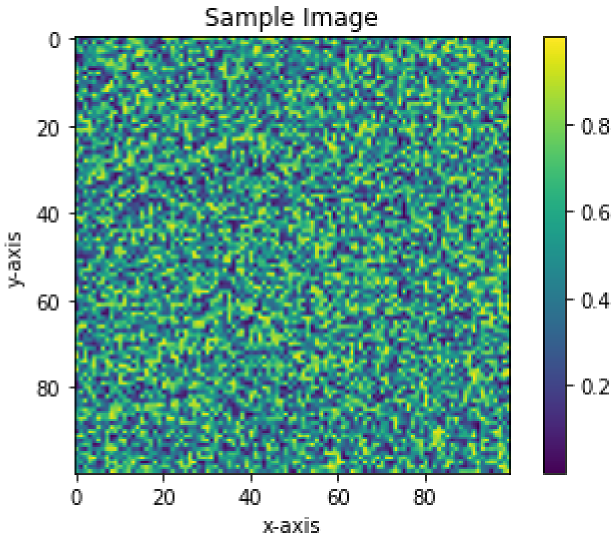
In [54]:

```
image_arr = np.random.random((100,100))

print(image_arr)

plt.imshow(image_arr)
```

```
# add colorbar
plt.colorbar()

# add title
plt.title("Sample Image")

# add axes labels
plt.xlabel("x-axis")
plt.ylabel("y-axis")

# change extent
# change color
# plt.show()

# More info: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
```

```
[[0.71371813 0.11077371 0.11110527 ... 0.44508612 0.30722475 0.73979231]
 [0.54299687 0.12029012 0.82404973 ... 0.06815587 0.5955616  0.54099243]
 [0.67486494 0.67785516 0.14733675 ... 0.91691587 0.64529347 0.32837097]
 ...
 [0.47803411 0.62653464 0.2387872  ... 0.39665164 0.72416419 0.50730771]
 [0.51244054 0.01980691 0.77946267 ... 0.98781156 0.94141419 0.64514688]
 [0.5981331  0.38653903 0.59938006 ... 0.29125298 0.83224894 0.93391248]]
```

```
Out[54]: Text(0, 0.5, 'y-axis')
```



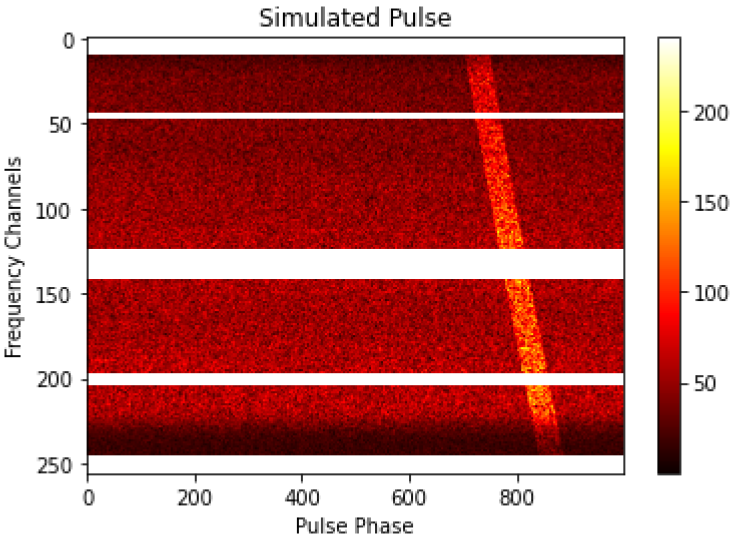
4.4 Exercise

In [55]:

```
dynanmic_spectrum = np.loadtxt("simulated_pulse.image")

plt.imshow(dynanmic_spectrum,aspect="auto", cmap="hot")
plt.title("Simulated Pulse")
plt.xlabel("Pulse Phase")
plt.ylabel("Frequency Channels")
plt.colorbar()
```

Out[55]:

```
<matplotlib.colorbar.Colorbar at 0x1240c8280>
```

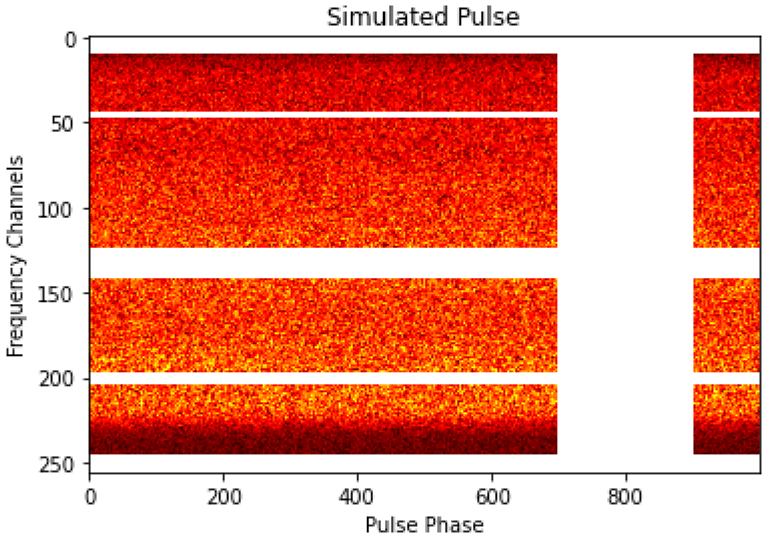


In [56]:

```
# Flag the pulse
dynanmic_spectrum[:, 700:900] = np.nan

plt.imshow(dynanmic_spectrum,aspect="auto", cmap="hot")
plt.imshow(dynanmic_spectrum,aspect="auto", cmap="hot")
plt.title("Simulated Pulse")
plt.xlabel("Pulse Phase")
plt.ylabel("Frequency Channels")
```

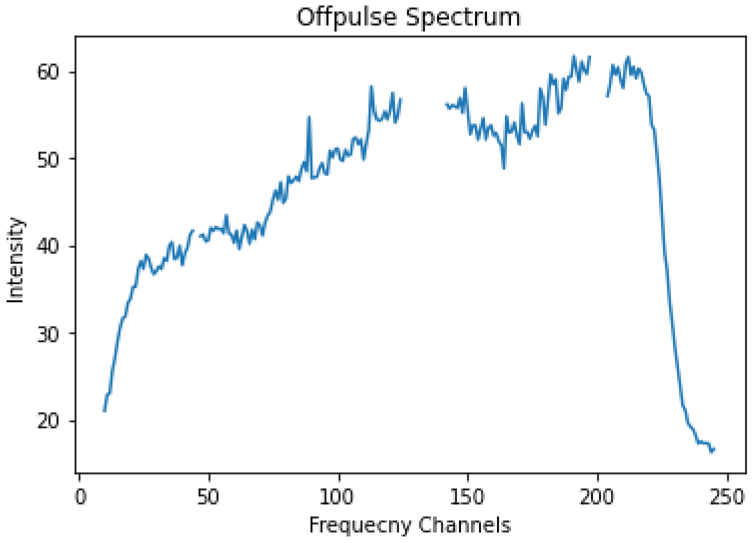
Out[56]:

```
Text(0, 0.5, 'Frequency Channels')
```



In [57]:

```
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=RuntimeWarning)
    offpulse_spectrum = np.nanmean(dynanmic_spectrum, axis=1)

plt.plot(offpulse_spectrum)
plt.title("Offpulse Spectrum")
plt.xlabel("Frequecny Channels")
plt.ylabel("Intensity")

np.savetxt("offpulse_spectrum.dat", offpulse_spectrum)
```


Offpulse Spectrum
Intensity
60
50
40
30
20
0
50
100
150
200
250
Frequecny Channels