full_presentation

February 15, 2022

0.1 MAP solutions for Cosmic Ray data and US Presidential Memes data

The cosmic ray analysis is presented first and is followed by the us presidential memes data.

[1]:
```
#Importing necessary modules
import numpy as np
import pandas as pd
import nifty7 as ift
import matplotlib.pyplot as plt
from requests_html import HTMLSession
import time
from datetime import datetime
from IPython.display import display, Image

datetimeorigin = datetime(2010, 1, 1, 0, 0)
datetime_format = "%m/%d/%y %H:%M"
```

0.1.1 Electron flux signal at AMS-02

Flux data is being measured to determine effects introduced by the heliosphere on incoming primary cosmic rays.

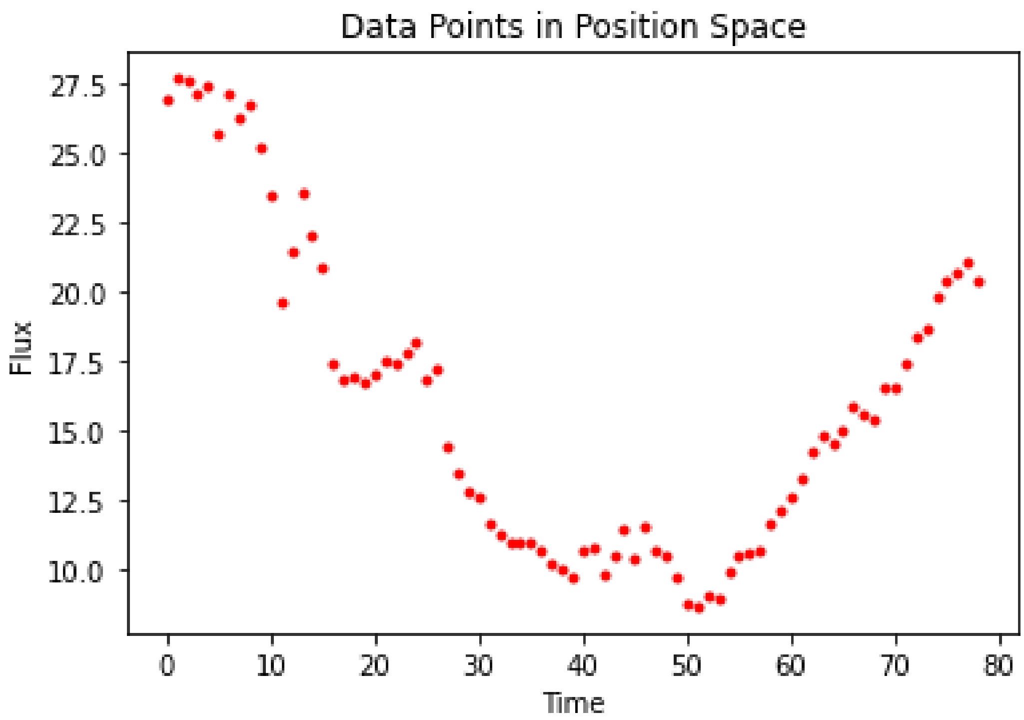
The specific dataset focuses on leptons (Electron and positrons).

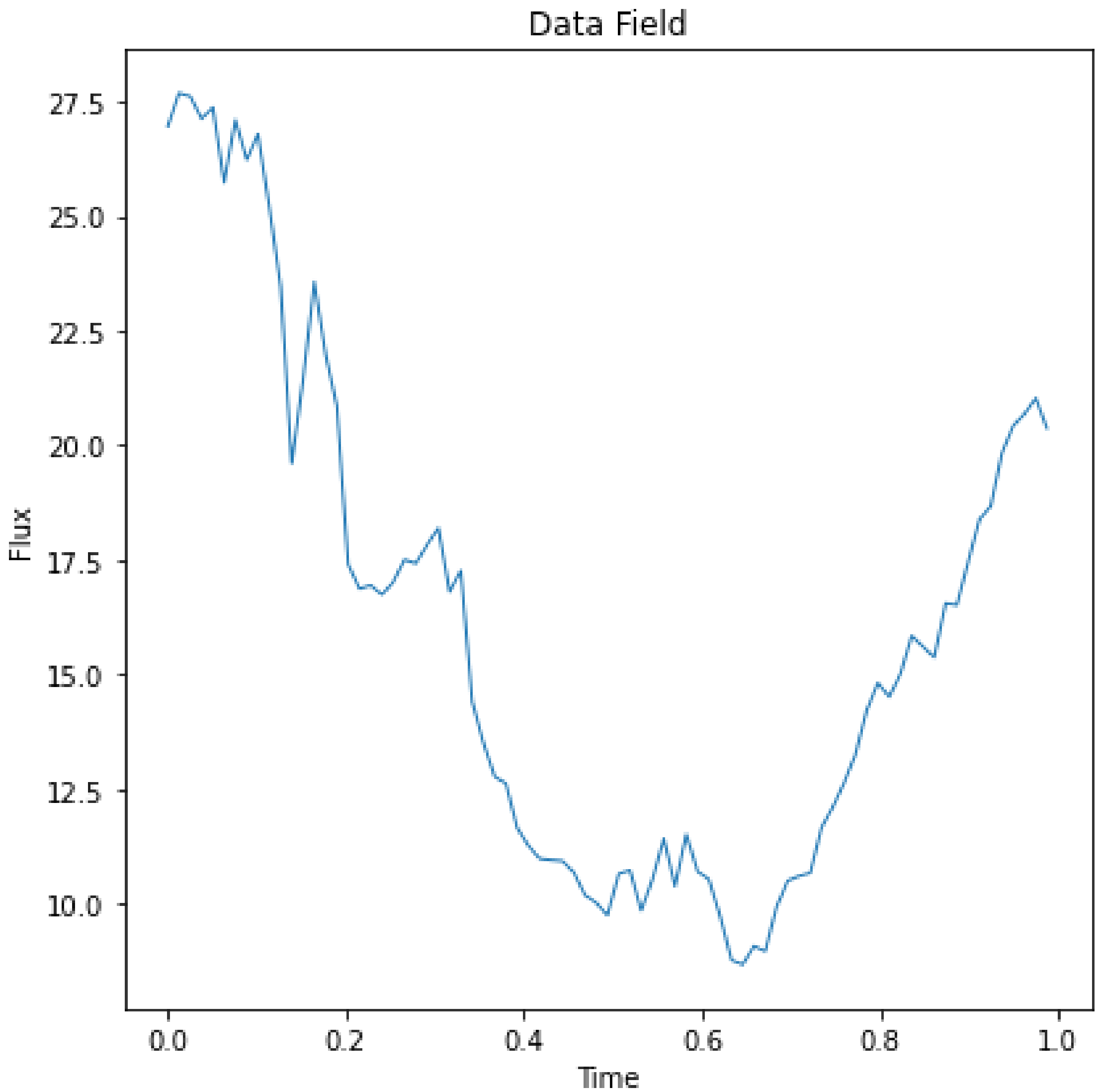
Measurements were made approximately once every 28 days based on solar rotation from May 2011 to May 2017.

Setting up data to be used

[2]:
```
Position_data = 'data_pos.xml'
Power_data = 'data_pow.xml'

data_file = pd.read_xml(Position_data)
input_data = np.array(data_file['flux'][1:])

position_space = ift.RGSpace(len(input_data))
data_field = ift.makeField(position_space, input_data)


plt.plot(input_data, 'r.', linewidth = 2)
```

```
plt.title("Data Points in Position Space")
plt.xlabel("Time")
plt.ylabel("Flux")
plt.show()

ift.single_plot(data_field, title="Data Field", xlabel="Time", ylabel="Flux")
```





Defining arguments used for ift.SimpleCorrelatedField()

[3]:
```
args = {
        'offset_mean': 17,
        'offset_std': (4, 2),

        # Amplitude of field fluctuations
        'fluctuations': (3., 0.8),  # 1.0, 1e-2

        # Exponent of power law power spectrum component
        'loglogavgslope': (-3., 1),  # -6.0, 1

        # Amplitude of integrated Wiener process power spectrum component
```

```python
        'flexibility': (2, 1.),  # 1.0, 0.5

        # How ragged the integrated Wiener process component is
        'asperity': (0.5, 0.4)  # 0.1, 0.5
}

correlated_field = ift.SimpleCorrelatedField(position_space, **args)
power_spectrum = correlated_field.power_spectrum
```

Introducing response operator

[4]:
```python
R = ift.GeometryRemover(position_space)
lamb = R(correlated_field)
data_space = R.target
data = ift.Field.from_raw(data_space,input_data)
```

Here white noise is considered

[5]:
```python
noise = 10
N = ift.ScalingOperator(data_space, noise)

likelihood_energy = ift.GaussianEnergy(mean = data, inverse_covariance=N.
 ↪inverse)(lamb)
```

Setting up minimizer

[6]:
```python
ic_newton = ift.DeltaEnergyController(
        name='Newton', iteration_limit=100, tol_rel_deltaE=1e-8)
minimizer = ift.NewtonCG(ic_newton)
```

Compute MAP solution by minimizing the information Hamiltonian

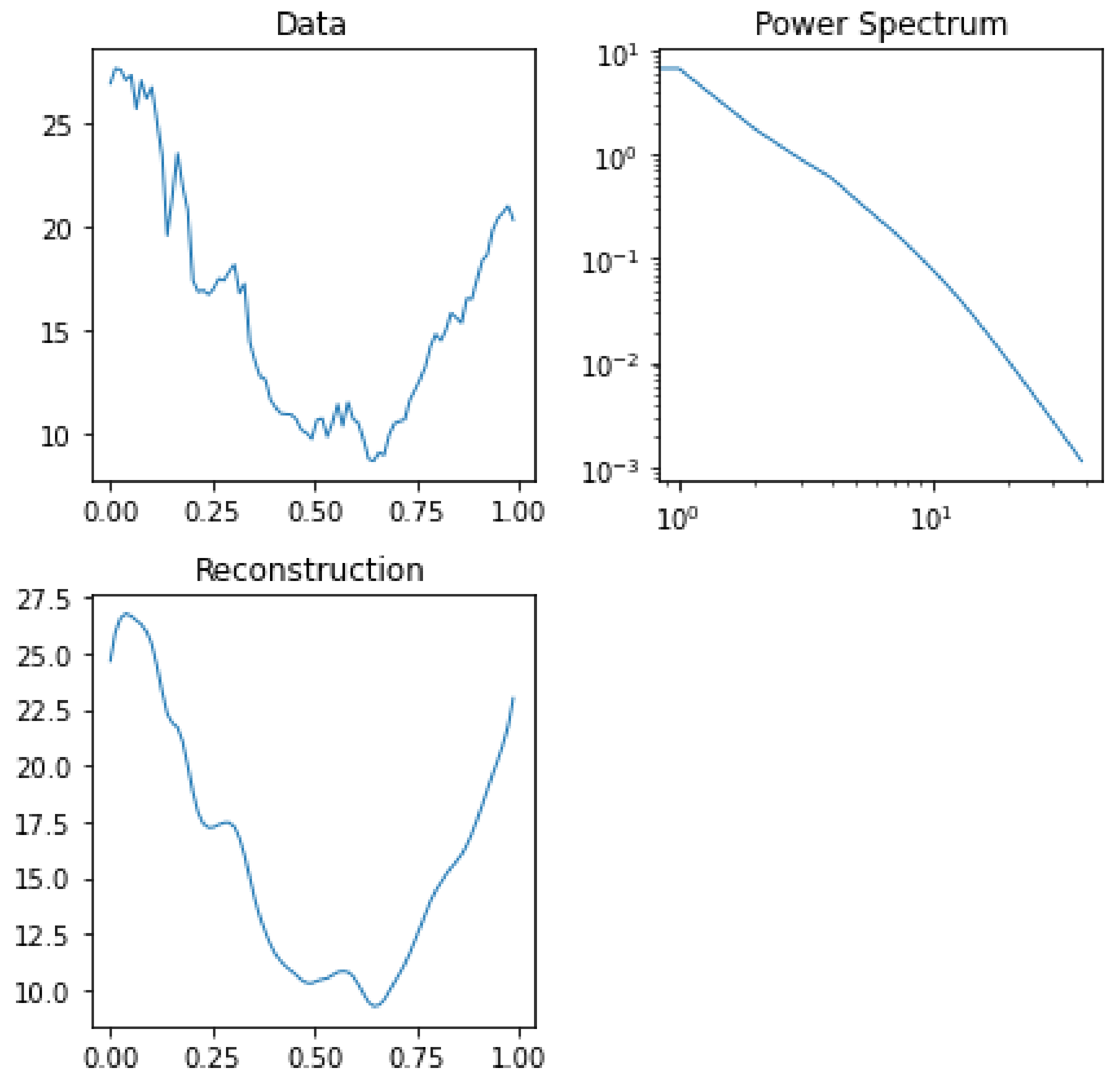
[7]:
```python
initial_position = ift.from_random(correlated_field.domain)
H = ift.StandardHamiltonian(likelihood_energy)
H = ift.EnergyAdapter(initial_position, H, want_metric=True)
H, convergence = minimizer(H)
```

```
Newton: Iteration #0 energy=2.747548E+02 reldiff=1.000000E+00 clvl=0
Iteration limit reached. Assuming convergence
Newton: Iteration #1 energy=3.682532E+01 reldiff=8.659702E-01 clvl=0
Newton: Iteration #2 energy=1.653002E+01 reldiff=5.511235E-01 clvl=0
Newton: Iteration #3 energy=1.196135E+01 reldiff=2.763862E-01 clvl=0
Newton: Iteration #4 energy=8.842832E+00 reldiff=2.607164E-01 clvl=0
Newton: Iteration #5 energy=8.763029E+00 reldiff=9.024530E-03 clvl=0
Newton: Iteration #6 energy=8.226836E+00 reldiff=6.118813E-02 clvl=0
Newton: Iteration #7 energy=8.216813E+00 reldiff=1.218334E-03 clvl=0
Newton: Iteration #8 energy=8.186065E+00 reldiff=3.742025E-03 clvl=0
Newton: Iteration #9 energy=8.167277E+00 reldiff=2.295123E-03 clvl=0
Newton: Iteration #10 energy=8.163253E+00 reldiff=4.926823E-04 clvl=0
Newton: Iteration #11 energy=8.158846E+00 reldiff=5.398557E-04 clvl=0
```

```
Newton: Iteration #12 energy=8.158773E+00 reldiff=8.962722E-06 clvl=0
Newton: Iteration #13 energy=8.157643E+00 reldiff=1.385503E-04 clvl=0
Newton: Iteration #14 energy=8.156965E+00 reldiff=8.314756E-05 clvl=0
Newton: Iteration #15 energy=8.156951E+00 reldiff=1.637931E-06 clvl=0
Newton: Iteration #16 energy=8.156731E+00 reldiff=2.698438E-05 clvl=0
Newton: Iteration #17 energy=8.156717E+00 reldiff=1.723868E-06 clvl=0
Newton: Iteration #18 energy=8.156686E+00 reldiff=3.776217E-06 clvl=0
Newton: Iteration #19 energy=8.156615E+00 reldiff=8.739425E-06 clvl=0
Newton: Iteration #20 energy=8.156614E+00 reldiff=1.120520E-07 clvl=0
Newton: Iteration #21 energy=8.156602E+00 reldiff=1.435585E-06 clvl=0
Newton: Iteration #22 energy=8.156601E+00 reldiff=1.380384E-07 clvl=0
Newton: Iteration #23 energy=8.156598E+00 reldiff=3.829187E-07 clvl=0
Newton: Iteration #24 energy=8.156598E+00 reldiff=6.435045E-09 clvl=1
```

Plotting

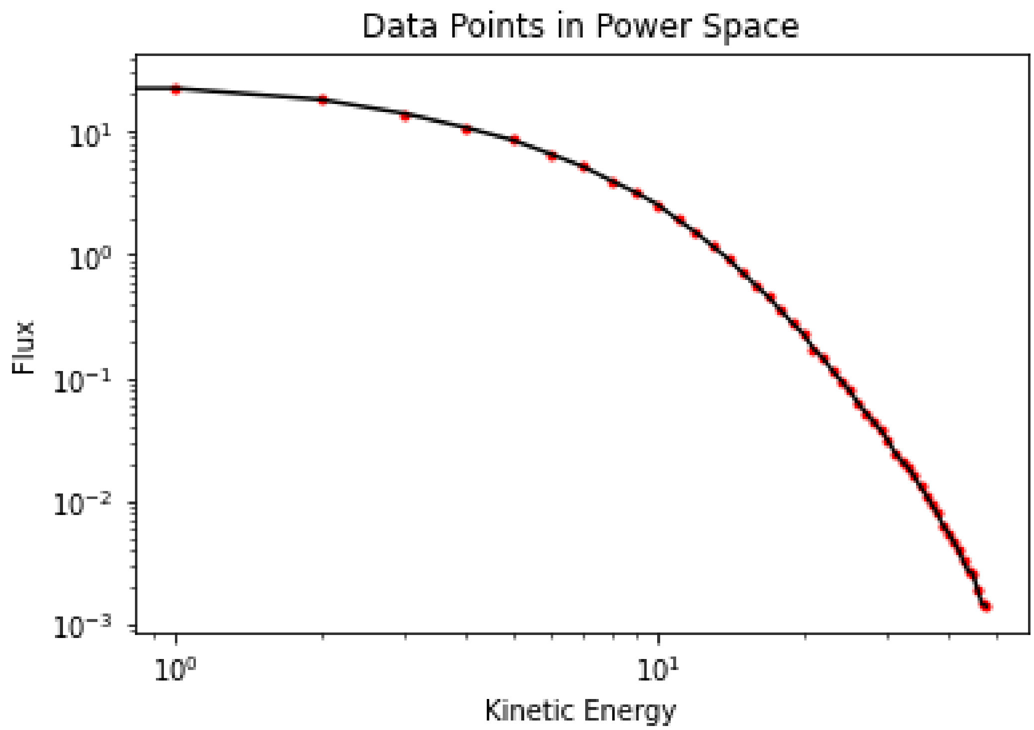
[8]:
```
reconst = correlated_field(H.position)
pspec = power_spectrum.force(H.position)
filename = "Testing.png"
plot = ift.Plot()
plot.add(R.adjoint(data), title='Data')
plot.add(pspec, title='Power Spectrum')
plot.add(reconst, title='Reconstruction')
plot.output()
```

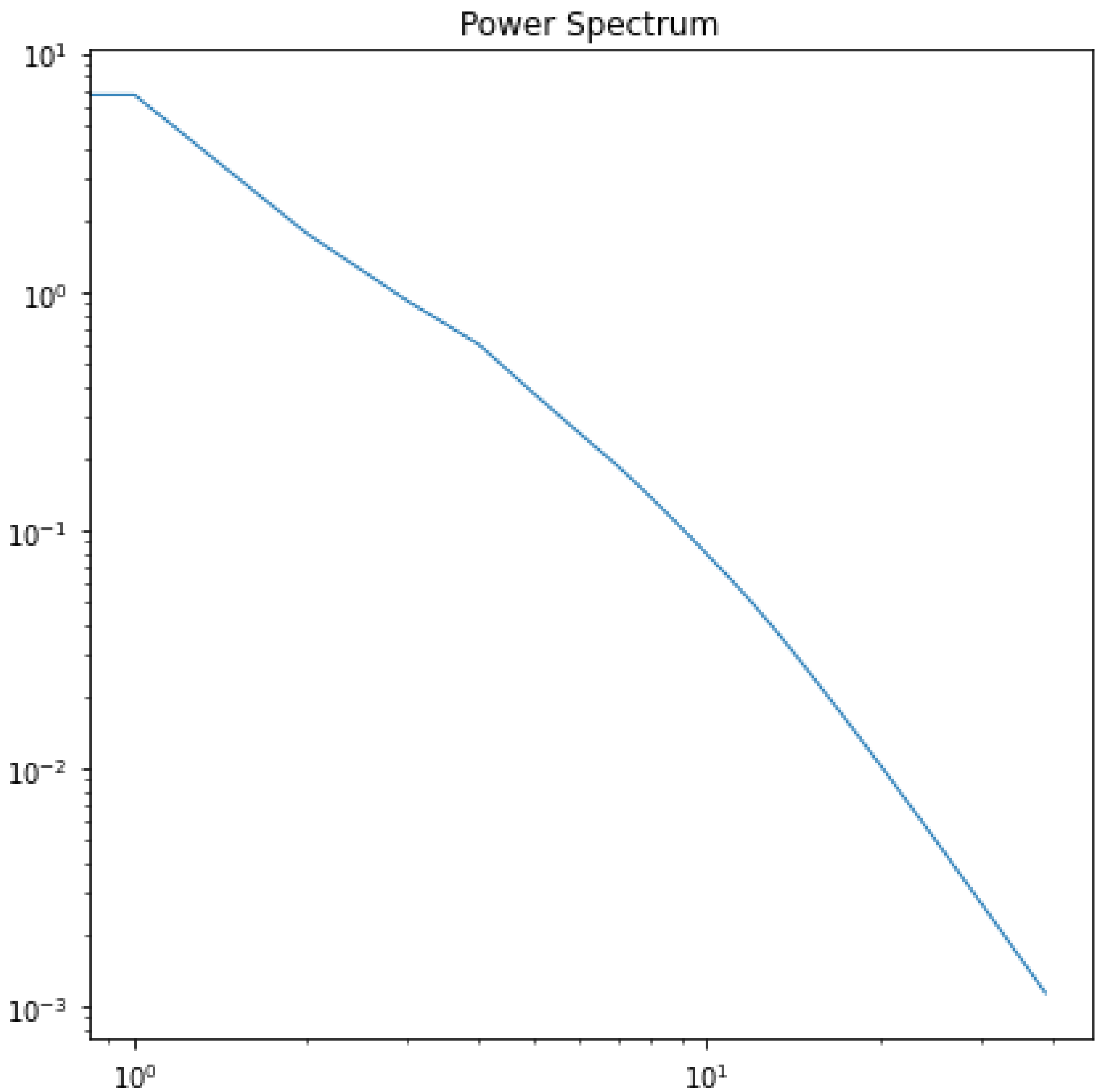


Where we get a familiar power spectrum

[9]:
```
pow_data_file = pd.read_xml(Power_data)
data_pow = np.array(pow_data_file['flux'][1:])

power_space = ift.RGSpace(len(data_pow))
pow_data_field = ift.makeField(power_space, data_pow)

plt.plot(data_pow, 'r.', linewidth = 2)
plt.plot(data_pow, 'k')
plt.xscale('log')
plt.yscale('log')
plt.title("Data Points in Power Space")
plt.xlabel("Kinetic Energy")
```

```
plt.ylabel("Flux")
plt.show()

ift.single_plot(pspec, title='Power Spectrum')
```





0.1.2 Dankness Field

The aim of this project is to track the popularity of a meme. The assumption is that the more popular a meme is the more views it would end up getting.

The popularity of the meme is modelled by a parameter s, which we like to call "dankness". It is defined such that the number of views of this meme is interpreted as its count and it follows the poissonian distribution:

$$P(c) = \frac{\lambda^c e^{-\lambda}}{c!}, \text{ with } \lambda = e^s.$$

If this is now promoted to a field over some domain, the techniques of information field theory can be used to study this dankness field. For now, the dankness field is defined over a one dimensional "time" domain, using the timestamp of instances of the meme. However, extensions to include some kind of "space" domain are conceivable. We imagine something like subreddits are being some such domain.

Now, the dankness field and the mean count rate get promoted to,

$$s(t) \text{ and } \lambda(t),$$

and ift techniques are applied.

Description of dataset The original dataset contains some information about the memes that became popular during the 2016 US presidential elections. The relevant columns are "timestamp" and "link". The link leads to a imgur website which then allows the "views" to be read out.

Note: The "meme" here is something like 'hillary' or 'trump' and meme instances are some pictorial memes relating to them.

Processing the dataset The timestamp is converted to "weeks from 2010 Jan 01". We decided to use a weekly timeframe as the views are only available cumulatively. Being unable to extract the exact time signature of the views we assumed that a meme instance is, on average, popular for about a week and this is mostly when it is viewed. Further work could follow the propagation of a meme instance over time.

The code below is used to initialize the data using the original dataset. An instance of the dataset is shown below.

[4]:
```
def initialize_data(filename):
    # read data from the csv file into pandas dataframe
    data = pd.read_csv(filename)
    data = data[data['network'] == 'imgur'] # only keep imgur data because of␣
↪space coordinate
    data = data[['timestamp','link','author']] # drop unnecessary columns
    data['views'] = np.zeros(data.shape[0]) # add a column of zeros for views
    data['views'] -= 1
    data['weeks'] = np.zeros(data.shape[0])

    # weeks since the beginning of 2010
    for i in data.index:
        data['weeks'][i] = (datetime.strptime(data['timestamp'][i],␣
↪datetime_format) - datetimeorigin).days//7

    data = data.drop('timestamp', 1)

    data.reset_index(drop=True)
    return data

def get_views(data):
    #create a session to download html
    session = HTMLSession()
```

```
    for i in data[data['views'] == -1].index: # only get data when not alrwady␣
↪initaized
        # this is the url which is later to be iterated over
        url = data['link'][i]

        # obtain the html file and then use javascript to render the entire␣
↪html file
        page = session.get(url)
        page.html.render()

        # search for the views field
        index = page.html.html.find('Views</span>')

        if index != -1:
            # now go back to find the span containing the views and then␣
↪convert it into an int
            prior_index = index
            while True:
                prior_index -= 1
                if page.html.html[prior_index] == '>':
                    break

            data['views'][i] = int(page.html.html[prior_index+1:index-1].
↪replace(',',''))
            time.sleep(2)
    return data


data = pd.read_csv('jill_nifty.csv')

data
```

[4]:

	link	author	views	weeks
0	http://imgur.com/a/6lnjL	4chan	5311.0	355.0
1	http://imgur.com/a/hGHd2	libertarianmeme	1052.0	353.0
2	http://imgur.com/xQkfJse	iamverysmart	68.0	353.0
3	http://imgur.com/43LJZG9	comics	49.0	352.0
4	http://imgur.com/9kZb6UD	politicalhumor	134.0	351.0
5	http://imgur.com/kfp40BD	iamverysmart	115.0	351.0
6	http://imgur.com/EhpW75N	FULLCOMMUNISM	115.0	350.0
7	http://imgur.com/odYglAk	me_irl	53.0	347.0
8	http://imgur.com/0UQh5Eo	me_irl	153.0	346.0
9	http://imgur.com/BVzBIgy	politicalhumor	402.0	345.0
10	http://imgur.com/a/uvT68	iamverysmart	2.0	345.0
11	http://imgur.com/Qj9x2wz	forwardsfromgrandma	91.0	344.0
12	http://imgur.com/pyCWWr7	BlackPeopleTwitter	73.0	343.0
13	http://imgur.com/oyjvlEa	conservativecartoons	139.0	343.0

```
14  http://imgur.com/MglKwWP       libertarianmeme    85.0  342.0
15  http://imgur.com/ottxPVC        FULLCOMMUNISM    67.0  341.0
16  http://imgur.com/a/mersz        TumblrInAction   411.0  341.0
17  http://imgur.com/Tf1mhKt        AdviceAnimals    46.0  340.0
18  http://imgur.com/dPkUsp1       politicalhumor   130.0  338.0
19  http://imgur.com/B0jL8er          iamverysmart   388.0  335.0
20  http://imgur.com/WvnX54P                 memes   116.0  335.0
21  http://imgur.com/A5t5Zkf        FULLCOMMUNISM    53.0  333.0
22  http://imgur.com/Xv5RVrR          screenshots    97.0  328.0
23  http://imgur.com/kkxG9Sm          screenshots   228.0  347.0
```

The field In our case the field is restricted to just being over time. A regular grid space of the range of weeks forms the time domain. And the data is just views per week.

[5]:
```python
def create_data(data, spacetime=False):
    if spacetime == False:
        begin_week = min(data['weeks'])
        end_week = max(data['weeks'])

        weeks_list = np.arange(begin_week, end_week)

        views_list = np.zeros(len(weeks_list))

        for week in weeks_list:
            views_list[int(week - begin_week)] = sum(data[(data['weeks'] ==␣
↪week)&(data['views'] != -1)]['views'])

        return views_list
    else:
        print('not yet implemented')
```

Analysis Finally, the analysis is done using the standard techniques developed in the course. The following code is basically a copy of the code used to analyse the cosmic ray data.

[6]:
```python
def make_work(filename):
    data = pd.read_csv(filename)
    input_data = create_data(data)

    # Position space and data field
    position_space = ift.RGSpace(2*len(input_data)) # pad to help with␣
↪periodicity conditions

    # Power Spectrum operator, starting from correlated field
    args = {
            'offset_mean': 17,
            'offset_std': (4, 2),
```

```
        # Amplitude of field fluctuations
        'fluctuations': (3., 0.8),  # 1.0, 1e-2

        # Exponent of power law power spectrum component
        'loglogavgslope': (-3., 1),  # -6.0, 1

        # Amplitude of integrated Wiener process power spectrum component
        'flexibility': (2, 1.),  # 1.0, 0.5

        # How ragged the integrated Wiener process component is
        'asperity': (0.5, 0.4)  # 0.1, 0.5
}

correlated_field = ift.SimpleCorrelatedField(position_space, **args)
exp_correlated_field = ift.exp(correlated_field)
power_spectrum = correlated_field.power_spectrum

# Response and noise
op = ift.SliceOperator(exp_correlated_field.target,input_data.shape) #␣
↪slicing from position_space to data_space
R = ift.GeometryRemover(op.target)
lamb = R(op(exp_correlated_field))
data_space = R.target
data = ift.Field.from_raw(data_space,input_data.astype(int))

# Likelihood energy
likelihood_energy = ift.PoissonianEnergy(data)(lamb)

# Settings for minimization
ic_newton = ift.DeltaEnergyController(
        name='Newton', iteration_limit=100, tol_rel_deltaE=1e-8)
minimizer = ift.NewtonCG(ic_newton)

# Compute MAP solution by minimizing the information Hamiltonian
initial_position = ift.from_random(exp_correlated_field.domain)
H = ift.StandardHamiltonian(likelihood_energy)
H = ift.EnergyAdapter(initial_position, H, want_metric=True)
H, convergence = minimizer(H)

# Plotting
signal = exp_correlated_field(initial_position)
reconstruction = exp_correlated_field(H.position)
pspec = power_spectrum.force(H.position)
plot = ift.Plot()
plot.add(pspec, title='Power Spectrum')
plot.add(R.adjoint(data), title='Data')
plot.add(reconstruction, title='Reconstruction')
```

```
    plot.add(reconstruction - signal, title='Residuals')
    plot.output(xsize=12, ysize=10, name=filename.replace('.csv','_')+'plot.
↪png', title=filename.replace('.csv','').replace('./',''))
    print("Saved results for '{}'.".format(filename))
```

Illustration As an illustration the code is run on one of the datasets.

[2]:
```
ls *.csv
```

```
bern_nifty.csv     donald_nifty.csv  hillary_nifty.csv
clinton_nifty.csv  gary_nifty.csv    jill_nifty.csv
```

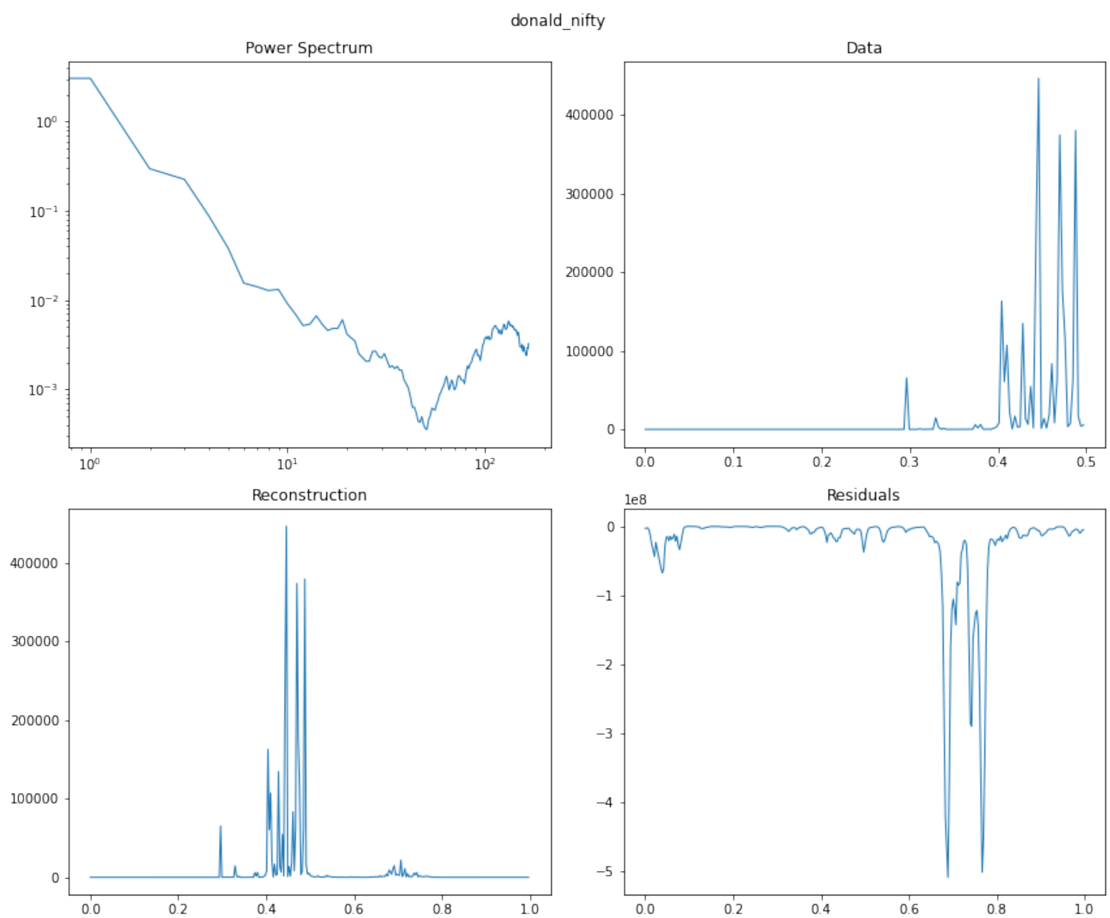
[9]:
```
filename = './donald_nifty.csv'

make_work(filename)
```

```
Newton: Iteration #0 energy=1.239576E+09 reldiff=1.000000E+00 clvl=0
Iteration limit reached. Assuming convergence
Newton: Iteration #1 energy=4.245657E+08 reldiff=6.574911E-01 clvl=0
Newton: Iteration #2 energy=1.310426E+08 reldiff=6.913491E-01 clvl=0
Newton: Iteration #3 energy=2.660679E+07 reldiff=7.969608E-01 clvl=0
Newton: Iteration #4 energy=-9.436804E+06 reldiff=1.354677E+00 clvl=0
Newton: Iteration #5 energy=-2.130947E+07 reldiff=5.571544E-01 clvl=0
Newton: Iteration #6 energy=-2.514998E+07 reldiff=1.527044E-01 clvl=0
Newton: Iteration #7 energy=-2.660196E+07 reldiff=5.458165E-02 clvl=0
Newton: Iteration #8 energy=-2.719193E+07 reldiff=2.169671E-02 clvl=0
Newton: Iteration #9 energy=-2.722081E+07 reldiff=1.060625E-03 clvl=0
Newton: Iteration #10 energy=-2.875460E+07 reldiff=5.334096E-02 clvl=0
Newton: Iteration #11 energy=-2.889957E+07 reldiff=5.016062E-03 clvl=0
Newton: Iteration #12 energy=-2.950985E+07 reldiff=2.068060E-02 clvl=0
Newton: Iteration #13 energy=-2.954307E+07 reldiff=1.124603E-03 clvl=0
Newton: Iteration #14 energy=-2.984438E+07 reldiff=1.009591E-02 clvl=0
Newton: Iteration #15 energy=-2.987210E+07 reldiff=9.279593E-04 clvl=0
Newton: Iteration #16 energy=-3.002240E+07 reldiff=5.006246E-03 clvl=0
Newton: Iteration #17 energy=-3.003437E+07 reldiff=3.987775E-04 clvl=0
Newton: Iteration #18 energy=-3.003512E+07 reldiff=2.476806E-05 clvl=0
Newton: Iteration #19 energy=-3.007027E+07 reldiff=1.169087E-03 clvl=0
Newton: Iteration #20 energy=-3.007085E+07 reldiff=1.928665E-05 clvl=0
Newton: Iteration #21 energy=-3.010357E+07 reldiff=1.086806E-03 clvl=0
Newton: Iteration #22 energy=-3.010378E+07 reldiff=6.918375E-06 clvl=0
Newton: Iteration #23 energy=-3.011357E+07 reldiff=3.253029E-04 clvl=0
Newton: Iteration #24 energy=-3.011359E+07 reldiff=5.646693E-07 clvl=0
Newton: Iteration #25 energy=-3.012751E+07 reldiff=4.620440E-04 clvl=0
Newton: Iteration #26 energy=-3.012753E+07 reldiff=5.345147E-07 clvl=0
Newton: Iteration #27 energy=-3.013271E+07 reldiff=1.718985E-04 clvl=0
Newton: Iteration #28 energy=-3.013282E+07 reldiff=3.726149E-06 clvl=0
Newton: Iteration #29 energy=-3.013347E+07 reldiff=2.166296E-05 clvl=0
Newton: Iteration #30 energy=-3.013347E+07 reldiff=1.328946E-08 clvl=0
```

```
Newton: Iteration #31 energy=-3.013592E+07 reldiff=8.108491E-05 clvl=0
Newton: Iteration #32 energy=-3.013603E+07 reldiff=3.867657E-06 clvl=0
Newton: Iteration #33 energy=-3.013660E+07 reldiff=1.877521E-05 clvl=0
Newton: Iteration #34 energy=-3.013660E+07 reldiff=1.075944E-08 clvl=0
Newton: Iteration #35 energy=-3.013706E+07 reldiff=1.539749E-05 clvl=0
Newton: Iteration #36 energy=-3.013707E+07 reldiff=1.388353E-07 clvl=0
Newton: Iteration #37 energy=-3.013707E+07 reldiff=3.248321E-08 clvl=0
Newton: Iteration #38 energy=-3.013709E+07 reldiff=7.222638E-07 clvl=0
Newton: Iteration #39 energy=-3.013709E+07 reldiff=3.201204E-10 clvl=1

Saved results for './donald_nifty.csv'.
```

[10]:

```
display(Image(filename=filename.replace('.csv','_plot.png')))
```



[16]:

```
display(Image(filename='thanks.jpg'))
```