PythonMathsPrimer

August 27, 2024

1 An Introduction to Python for Mathematics

1.1 a crash course

Simon Shaw

- https://www.brunel.ac.uk/people/simon-shaw
- https://github.com/variationalform

Version 1

This document uses python

and also makes use of LaTeX

in Markdown

1.2 Contents

This is a very quick run through of how to use some basic features of python in a Jupyter notebook. We'll cover:

1. markdown
2. linear algebra
3. plotting
4. prob/stat simulations
5. some other stuff... if time...

There is a lot we don't touch...

There is a beamer/PDF slide show with some background to the code below.

This material is on git at: https://github.com/variationalform/PythonMathsPrimer

you can run the notebook on binder using this:

https://mybinder.org/v2/gh/variationalform/PythonMathsPrimer/HEAD

This code creates a button - but it breaks LaTeX output

```
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/variationalform/Py
```

1.3 Markdown

First note this cell - it is a **markdown** cell, not a 'code' cell. This allows for *literate programming*: we can explain our algorithm with bullets,

- step 1
- step 2
  - substep 2.1
- step 3

or with enumeration,

1. step 1
2. step 2
3. substep 2.1
4. step 3

We can type maths in LaTeX, like this: find $\boldsymbol{u} \in V$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = \langle \neg, \boldsymbol{v} \rangle \qquad \forall \boldsymbol{v} \in V.$$

There is no equation numbering, BiBTeX, graphics etc. AFAIK though… perhaps see **bookdown**. HTML is allowed.

1.4 Leontief Input-Output Models

This *input-output* problem has been set up in the accompanying slide show.

We have $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{d}$ or, alternatively, $(\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} = \boldsymbol{d}$

First some simple arithmetic as a warm up… And an introduction to `numpy`

[1]:
```
0.9*50000 - 0.5*40000
```

[1]:
```
25000.0
```

[2]:
```
-0.3*50000 + 0.8*40000
```

[2]:
```
17000.0
```

[3]:
```
print(0.9*50000 - 0.5*40000, -0.3*50000 + 0.8*40000)
```

```
25000.0 17000.0
```

Note that

$$\boldsymbol{A} = \begin{pmatrix} 0.1 & 0.5 \\ 0.3 & 0.2 \end{pmatrix} \Longrightarrow \boldsymbol{I} - \boldsymbol{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.1 & 0.5 \\ 0.3 & 0.2 \end{pmatrix} = \begin{pmatrix} 0.9 & -0.5 \\ -0.3 & 0.8 \end{pmatrix}$$

and therefore

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = \frac{1}{(0.9)(0.8) - (0.5)(0.3)} \begin{pmatrix} 0.8 & 0.5 \\ 0.3 & 0.9 \end{pmatrix} = \frac{10}{57} \begin{pmatrix} 8 & 5 \\ 3 & 9 \end{pmatrix}$$

What are these?

$$\frac{10}{57}\begin{pmatrix}8 & 5\\ 3 & 9\end{pmatrix}\begin{pmatrix}35000\\ 29000\end{pmatrix} \qquad \text{and} \qquad \frac{10}{57}\begin{pmatrix}8 & 5\\ 3 & 9\end{pmatrix}\begin{pmatrix}2500\\ 1900\end{pmatrix}$$

[4]:
```
d1=35000; d2=29000; print(10/57*(8*d1 + 5*d2), 10/57*(3*d1 + 9*d2))
```

```
74561.40350877192 64210.52631578947
```

[5]:
```
d1=2500; d2=1900; print(10/57*(8*d1 + 5*d2), 10/57*(3*d1 + 9*d2))
```

```
5175.438596491228 4315.78947368421
```

Let's introduce `numpy` - for arrays, and hence linear algebra. We'll duplicate the calculation above...

[6]:
```
import numpy as np
A = np.array([[0.1, 0.5],[0.3, 0.2]]) # use tab completion on 'np.a'
Id = np.eye(2)
print(Id-A)
print(5.7*np.linalg.inv(Id-A))   # using inverse matrices is usually bad!
d = np.array([[35000],[29000]])
print(np.linalg.solve(Id-A, d))
Dd = np.array([[2500],[1900]])
print(np.linalg.solve(Id-A, Dd))
```

```
[[ 0.9 -0.5]
 [-0.3  0.8]]
[[8. 5.]
 [3. 9.]]
[[74561.40350877]
 [64210.52631579]]
[[5175.43859649]
 [4315.78947368]]
```

Back to the Leontief IO problem...

[7]:
```
print('This is the technical matrix: A = ')
A = np.array([[0.15, 0.12, 0.05, 0.03],
              [0.17, 0.16, 0.04, 0.04],
              [0.03, 0.08, 0.18, 0.22],
              [0.07, 0.18, 0.03, 0.19]])
print(A)
print('Here are the total output levels: x = ')
x = np.array([[89000], [55000], [47000], [76000]])
print(x.T)
print('Note the transpose - a tidier output...')
```

```
This is the technical matrix: A =
[[0.15 0.12 0.05 0.03]
 [0.17 0.16 0.04 0.04]
```

```
 [0.03 0.08 0.18 0.22]
 [0.07 0.18 0.03 0.19]]
Here are the total output levels: x =
[[89000 55000 47000 76000]]
Note the transpose - a tidier output…
```

[8]:
```
print('This is the Leontiev matrix: I-A = ')
Id = np.eye(4)
print(Id-A)
print('The amounts available for external demand are: d = ')
print((Id-A).dot(x).T)
```

```
This is the Leontiev matrix: I-A =
[[ 0.85 -0.12 -0.05 -0.03]
 [-0.17  0.84 -0.04 -0.04]
 [-0.03 -0.08  0.82 -0.22]
 [-0.07 -0.18 -0.03  0.81]]
The amounts available for external demand are: d =
[[64420. 26150. 14750. 44020.]]
```

[9]:
```
print('The required total output is: x = ')
d = np.array([[55000],[24000],[18000],[40000]])
print(np.linalg.solve(Id-A, d).T)
print('The change in external demand: Dd = ')
Dd = np.array([[-5000], [350], [2300],[-500]])
print(Dd.T)
print(np.linalg.solve(Id-A, Dd).T)
```

```
The required total output is: x =
[[76985.04648009 49721.31303541 48094.90788649 68866.21831676]]
The change in external demand: Dd =
[[-5000   350  2300  -500]]
[[-5899.0465539   -730.25568572  2193.68789267 -1208.10943879]]
```

1.5 Eigensystems and SVD

While we are looking at linear algebra it's useful to look at eigenvalue problems and the related SVD. We'll stay with the technical matrix $\boldsymbol{A}$ as defined above…

[10]:
```
w, V = np.linalg.eig(A)
print('The eigenvalues are ', w) # the eigenvalues
print(f'the shape of the eigenvector matrix V is {V.shape}')
print('and the first two eigenvectors are')
print(V[:,:2])
```

```
The eigenvalues are  [0.42116703+0.j         0.00461882+0.j
0.12710707+0.07455003j
 0.12710707-0.07455003j]
the shape of the eigenvector matrix V is (4, 4)
```

```
and the first two eigenvectors are
[[-0.36169836+0.j -0.53830234+0.j]
 [-0.41503348+0.j  0.60726175+0.j]
 [-0.65491995+0.j  0.37609264+0.j]
 [-0.51768849+0.j -0.44723381+0.j]]
```

We can check that $\boldsymbol{AV} = \boldsymbol{VD}$...

See e.g. https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

[11]:
```
D=np.diag(w)
print('D = \n', D)
# print(A@V - V@D)  # uncomment it if you like, but this is better...
print(f"Frobenius norm, ||AV-VD||F = {np.linalg.norm(A@V - V@D, ord='fro')}")
```

```
D =
 [[0.42116703+0.j         0.        +0.j         0.        +0.j
  0.        +0.j        ]
 [0.        +0.j         0.00461882+0.j         0.        +0.j
  0.        +0.j        ]
 [0.        +0.j         0.        +0.j         0.12710707+0.07455003j
  0.        +0.j        ]
 [0.        +0.j         0.        +0.j         0.        +0.j
  0.12710707-0.07455003j]]
Frobenius norm, ||AV-VD||F = 4.548644071527765e-16
```

The SVD ... There are a couple of *gotchas* for the SVD, it works like this...

[12]:
```
K = np.array([[1,2,5],[5,-6,1]])
U, S, VT = np.linalg.svd(K)
print('U is what we expect', U)
print('But S is not!', S)
print('V-transpose gets returned, not V', VT)
```

```
U is what we expect [[-0.06213744  0.9980676 ]
 [ 0.9980676   0.06213744]]
But S is not! [7.88191065 5.4658471 ]
V-transpose gets returned, not V [[ 0.62525456 -0.77553283  0.08720987]
 [ 0.23944227  0.29699158  0.9243719 ]
 [-0.74278135 -0.55708601  0.37139068]]
```

We can stack `S` to get what we expect ...

[13]:
```
S = np.hstack(( np.diag(S), np.zeros((2,1)) ))
print('S is now the correct shape\n', S)

# print(K - U @ S @ VT) # again, uncomment if you like but this is better
print(f"The inf-norm ||K - U S V^T||inf = {np.linalg.norm(K - U @ S @ VT, np.
 ↪inf)}")
```

```
S is now the correct shape
 [[7.88191065 0.         0.        ]
 [0.         5.4658471  0.        ]]
The inf-norm ||K - U S V^T||inf = 2.7755575615628914e-15
```

1.5.1 Exercises 1

Open a new cell under this and attempt these…

1. Find the required total output for $\boldsymbol{d} = (51k, 26k, 15k, 48k)^T$.
2. Find (and verify) the eigensystem for the $N \times N$ tridiagonal Laplacian matrix

$$\boldsymbol{A} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & & & & \\ -1 & 2 & -1 & 0 & \cdots & & & \\ 0 & -1 & 2 & -1 & 0 & \cdots & & \\ \cdots & 0 & -1 & 2 & -1 & 0 & \cdots & \\ & & & \ddots & \ddots & \ddots & & \\ & & \cdots & 0 & -1 & 2 & -1 & 0 \\ & & & \cdots & 0 & -1 & 2 & -1 \\ & & & & \cdots & 0 & -1 & 2 \end{pmatrix}$$

where $h = (N+1)^{-2}$. Hint:

```
diag = np.ones(4)
print(2*np.diag(diag)-np.diag(diag[1:],-1)-np.diag(diag[1:],1))
```

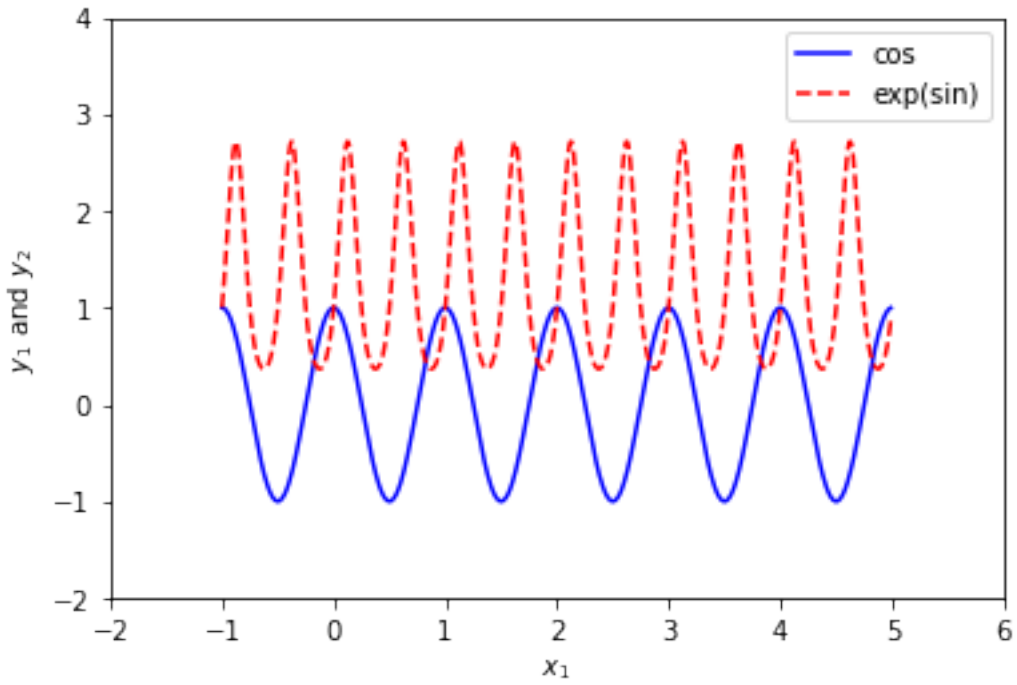
1.6 Plotting in 2D

We introduce `matplotlib`. There are others but this seems to be the most common.

Here is a simple example

[14]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-1,5,0.01)
y1, y2 = np.cos(2*np.pi*x), np.exp(np.sin(4*np.pi*x))
plt.plot(x,y1, 'b-')
plt.plot(x,y2, 'r--')
plt.axis([-2, 6, -2, 4])
plt.legend(['cos', 'exp(sin)'])
plt.xlabel(r'$x_1$'); plt.ylabel('$y_1$ and $y_2$')
plt.savefig('./gfx/my2Dplot.png', dpi=600)
plt.savefig('./gfx/my2Dplot.eps', dpi=600)
```

```
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
```

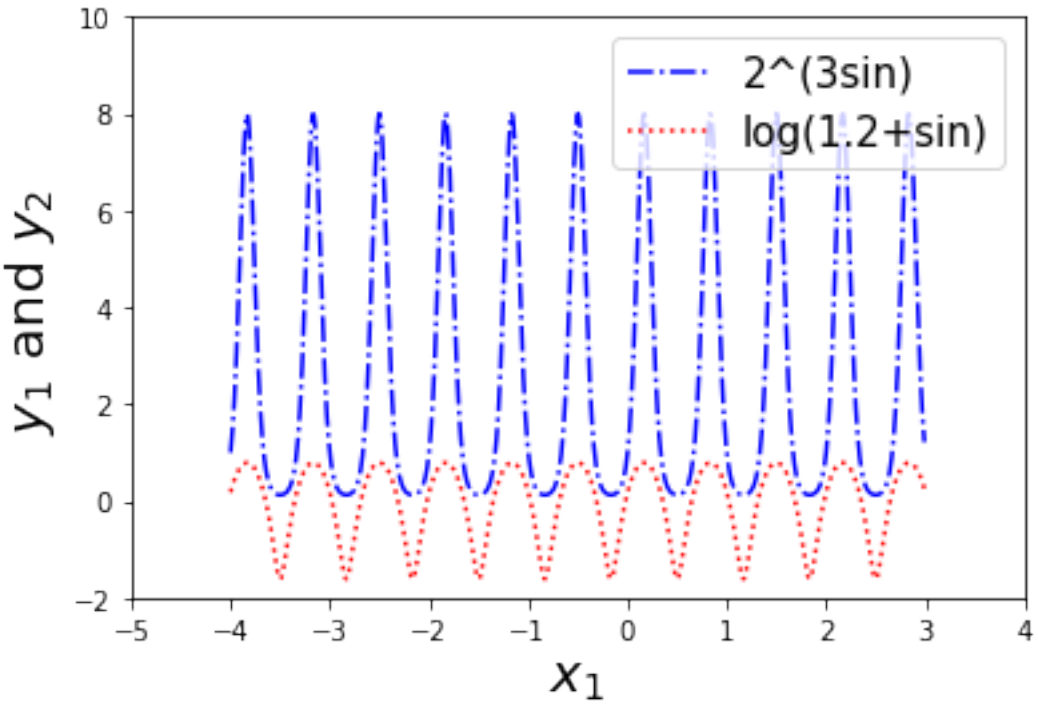


Note that once we have imported we don't have to do it again.

However - make sure you execute the notebook from the top down!

[15]:
```
x = np.arange(-4,3,0.01)
y1, y2 = 2**(3*np.sin(3*np.pi*x)), np.log(1.2+np.sin(3*np.pi*x))
plt.plot(x,y1, 'b-.')
plt.plot(x,y2, 'r:')
plt.axis([-5, 4, -2, 10])
plt.legend(['2^(3sin)', 'log(1.2+sin)'], fontsize=15)
plt.xlabel(r'$x_1$', fontsize=20); plt.ylabel('$y_1$ and $y_2$', fontsize=20)
plt.savefig('./gfx/my2Dplot2.png', dpi=600)
plt.savefig('./gfx/my2Dplot2.eps', dpi=600)
```

```
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
```



1.6.1 Exercises 2

1. Find (and verify) the SVD of

$$\boldsymbol{K} = \begin{pmatrix} 2 & -5 & 7 & -9 \\ -1 & 2 & -1 & 0 \end{pmatrix}$$

   Find a rank one approximation to $\boldsymbol{K}$ and determine the error in the Frobenius norm (Hint: look up `np.linalg.norm()`).

2. Consider `y = np.heaviside(np.sin(2*np.pi*x),0)` and plot a square waves with periods π and π^2 and amplitudes 2 and 5. (Hint: use `np.pi**2`)

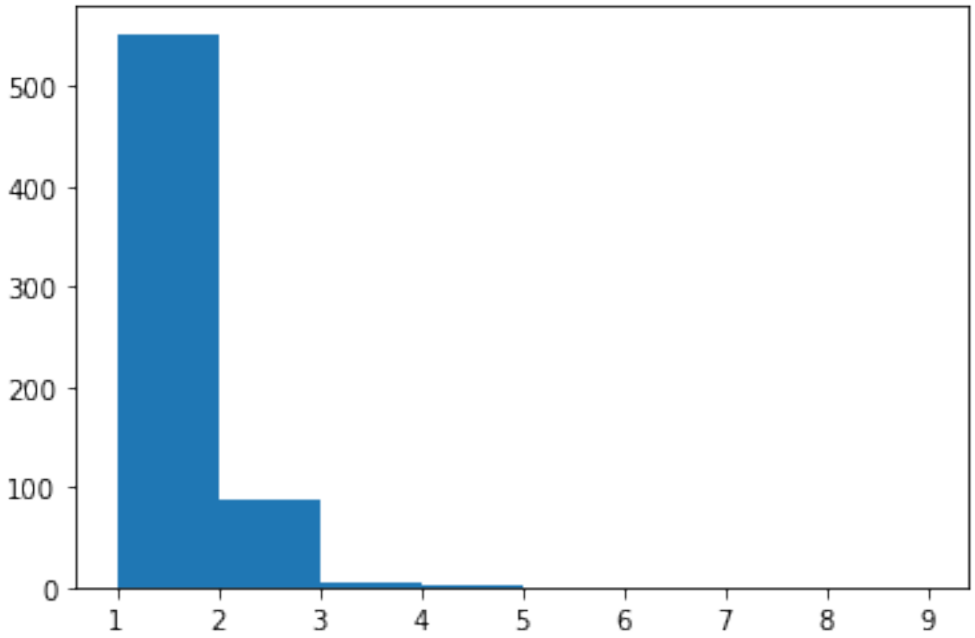
1.7 Anonymity

Based on

- https://www.johndcook.com/blog/2018/12/07/simulating-zipcode-sex-birthdate/
- https://techscience.org/a/2015092903/

[16]:
```
from random import randrange
import matplotlib.pyplot as plt
import numpy as np

d = 365*3*2
N = 750
```

```
buckets = np.zeros(d)

for _ in range(N):
    z = randrange(d)
    buckets[z] += 1

plt.hist(buckets, range(1,10)); # note the semi-colon (try without)
```



[17]:
```
loners = len(buckets[buckets==1])
print('Probability that anonymous data occurs only once: ', loners/N)
print('Nearly exact probability that anonymous data occurs only once: ',
       np.exp(-N/d))
loners2 = len(buckets[buckets==2])
print('Probability that anonymous data occurs at most twice: ',
      (loners+2*loners2)/N)
loners3 = len(buckets[buckets==3])
print('Probability that anonymous data occurs at most three times: ',
      (loners+2*loners2+3*loners3)/N)
loners4 = len(buckets[buckets==4])
print('Probability that anonymous data occurs at most four times: ',
      (loners+2*loners2+3*loners3+4*loners4)/N)
```

```
Probability that anonymous data occurs only once:  0.736
Nearly exact probability that anonymous data occurs only once:
0.7100174346347615
Probability that anonymous data occurs at most twice:  0.9706666666666667
```

```
Probability that anonymous data occurs at most three times:  0.9946666666666667
Probability that anonymous data occurs at most four times:  1.0
```

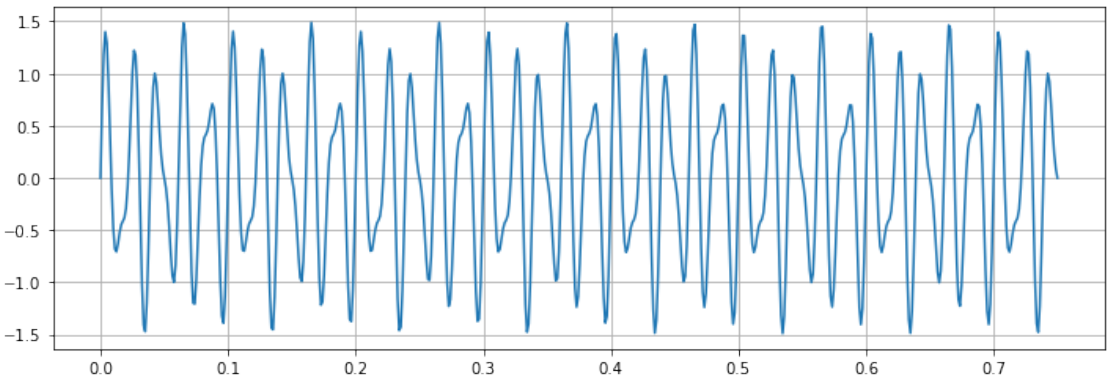
[18]:
```
print( sum(buckets) )
print( len(buckets[buckets!=0]), end = ', ')
print( len(buckets[buckets==1]), end = ', ' )
print( len(buckets[buckets==2]), end = ', ' )
print( len(buckets[buckets==3]), end = ', ' )
print( len(buckets[buckets==4]), end = ', ' )
print( len(buckets[buckets>4]), end = ', ' )

# a check
print( len(buckets[buckets==1])+2*len(buckets[buckets==2])
       +3*len(buckets[buckets==3])+4*len(buckets[buckets==4]) )
```

```
750.0
647, 552, 88, 6, 1, 0, 750
```

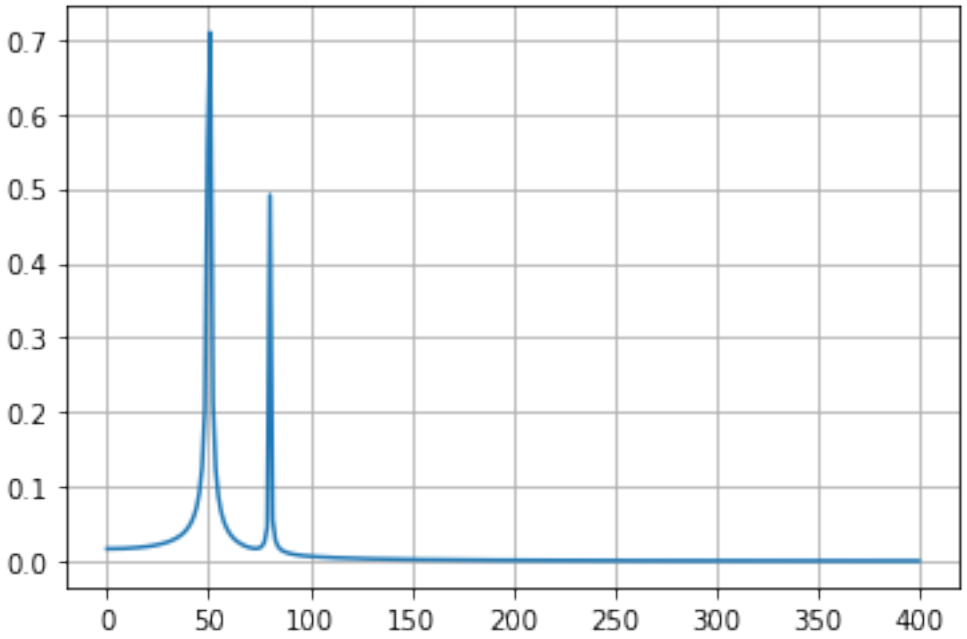
1.8 Discrete Fourier Transform

Here's an example from the docs: - https://docs.scipy.org/doc/scipy/reference/fft.html#module-scipy.fft - https://docs.scipy.org/doc/scipy-1.5.2/reference/tutorial/fft.html

[19]:
```
from scipy.fft import fft, fftfreq, ifft
N = 600; dt = 1.0 / 800.0   # number of sample points and sample spacing
t = np.linspace(0.0, N*dt, N)
y = np.sin(50.0 * 2.0*np.pi*t) + 0.5*np.sin(80.0 * 2.0*np.pi*t)
plt.figure(figsize=(12, 4)); plt.plot(t, y);  plt.grid(); plt.show()
```

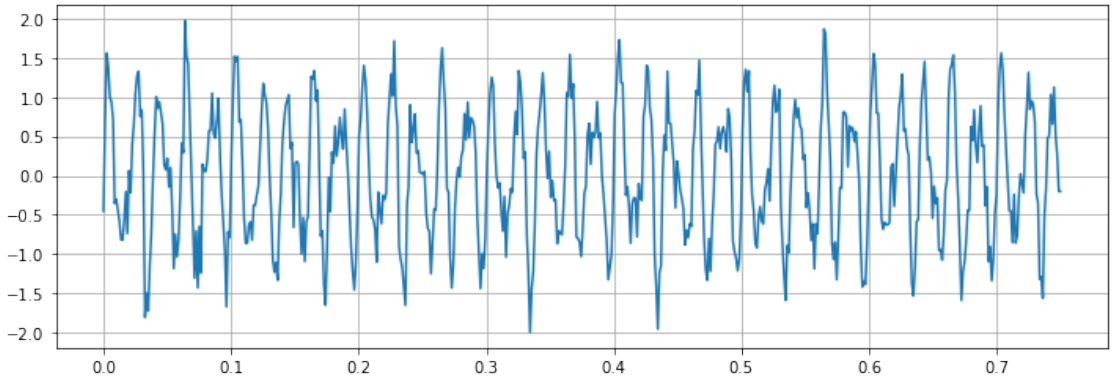


[20]:
```
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*dt), N//2) # or xf = fftfreq(N,T)[:N//2]
import matplotlib.pyplot as plt
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2])); plt.grid(); plt.show()
```

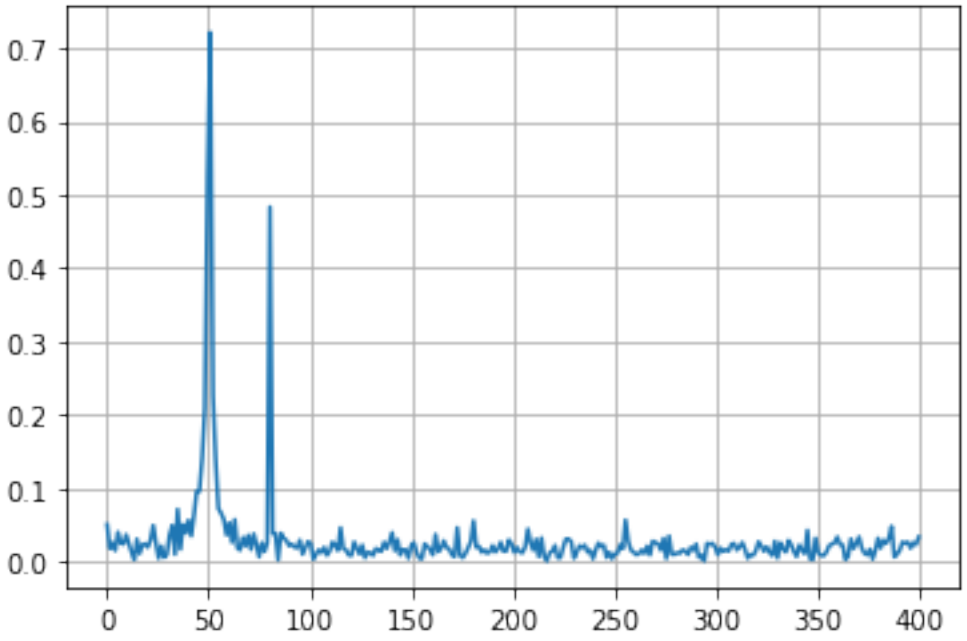


Let's noise it up a bit...

[21]:
```
#import numpy.random.normal
yn  = np.sin(50.0 * 2.0*np.pi*t) + 0.5*np.sin(80.0 * 2.0*np.pi*t)
yn += np.random.normal(0,200*dt,yn.shape)
plt.figure(figsize=(12, 4)); plt.plot(t, yn);  plt.grid(); plt.show()
```



[22]:
```
ynf = fft(yn)
xf = np.linspace(0.0, 1.0/(2.0*dt), N//2) # or xf = fftfreq(N,T)[:N//2]
import matplotlib.pyplot as plt
plt.plot(xf, 2.0/N * np.abs(ynf[0:N//2])); plt.grid(); plt.show()
```



1.9 ODE Solver - Huen's method

Given $\dot{\boldsymbol{z}} = \boldsymbol{f}(t, \boldsymbol{z}(\boldsymbol{t}))$ Heun's method, with $\boldsymbol{z}(0) = \boldsymbol{z}_0$ is,

$$\begin{aligned}\boldsymbol{y}_{n+1} &= \boldsymbol{z}_n + k\boldsymbol{f}(t_n, \boldsymbol{z}_n),\\ \boldsymbol{z}_{n+1} &= \boldsymbol{z}_n + \frac{k}{2}\Big(\boldsymbol{f}(t_n, \boldsymbol{z}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\Big),\end{aligned}$$

for $n = 0, 1, 2, 3, ..., N-1$ and where $k = T/N$ for the final time T.

Example (from Example 8.4B in Numerical Analysis, a practical approach, MJ Maron and RJ Lopez, Wordsworth Publishing Company, 1991):

$$\dot{\boldsymbol{z}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} 0 \\ t \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ t \end{pmatrix} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{f}(t, \boldsymbol{z})$$

with $\boldsymbol{z}(0) = (1, -1)^T$ has solution

$$\boldsymbol{z} = \frac{1}{2}\begin{pmatrix} \exp(t) + \exp(-t) - 2t \\ \exp(t) - \exp(-t) - 2 \end{pmatrix}.$$

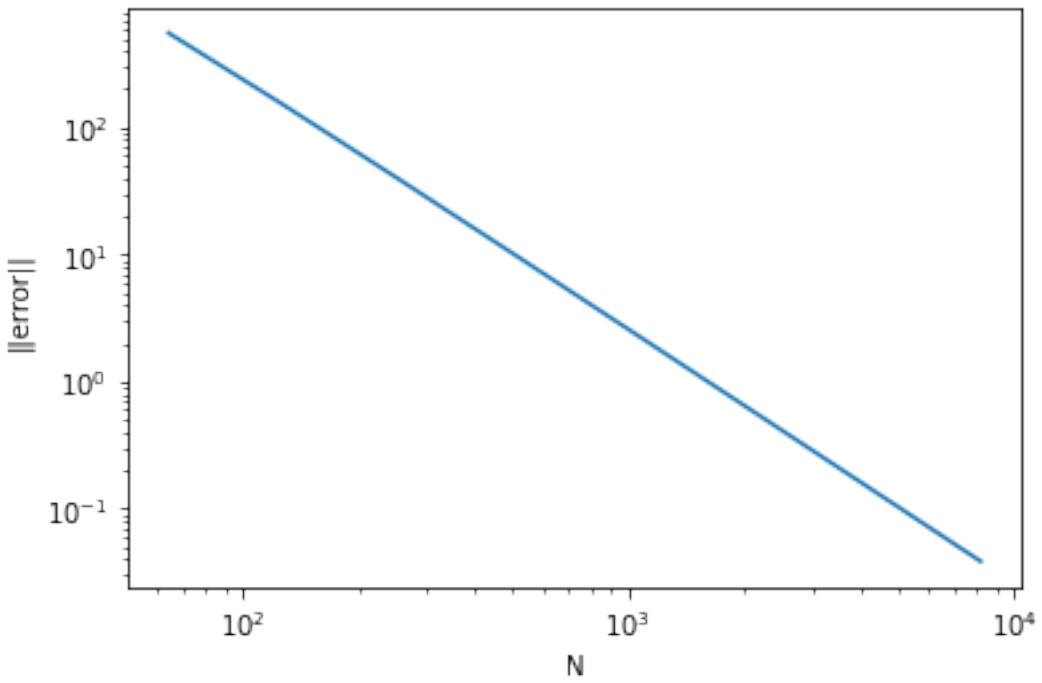
Here is the code... With an example definition of a function...

[23]:
```
# NOTE the use of [:,[n]] rather that [:,n] - a slice gives a column...
def Huen(N,k,B):
```

```
  # allocate solution vector and set up initial condition
  z = np.zeros((2,N+1)); z[0,0] = 1; z[1,0] = -1
  for n in range(N):
    tn = n*k
    f = np.array([[0],[tn]]) + B @ z[:,[n]]
    z[:,[n+1]] = z[:,[n]] + k*f
    f = f + np.array([[0],[tn+k]]) + B @ z[:,[n+1]]
    z[:,[n+1]] = z[:,[n]] + 0.5*k*f
  return z[:,[N]]

T = 10; N1 = 64; B = np.array([[0,1],[1,0]])
zexact  = 0.5*np.array([[np.exp(T)+np.exp(-T)-2*T],[np.exp(T)-np.exp(-T)-2]])
maxit = 8; Nvals = np.zeros(maxit); errors = np.zeros(maxit)
for i in range(maxit):
    N = N1*2**i; k = T/N; Nvals[i] = N;
    zapprox = Huen(N,k,B)
    errors[i] = np.linalg.norm(zexact - zapprox)
    print(errors[i])
```

```
553.9969262552904
148.72911957118532
38.41966013228909
9.755381185582275
2.4573348385089204
0.6166231615350984
0.1544405550689361
0.03864564398922986
```

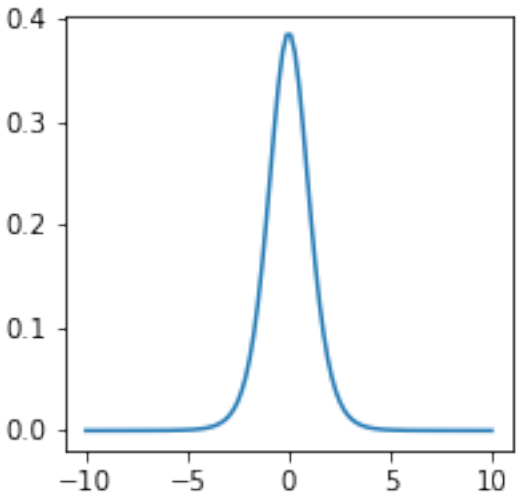
[24]:
```
plt.loglog(Nvals,errors); plt.xlabel('N'); plt.ylabel('||error||');
```



2 Hypothesis Testing

Example taken from: https://www.jmp.com/en_gb/statistics-knowledge-portal/t-test/one-sample-t-test.html and with some input from https://www.geeksforgeeks.org/how-to-find-the-t-critical-value-in-python/.

[25]:
```
from scipy.stats import t
x = np.linspace(-10,10,100); y = t.pdf(x, df=8); plt.figure(figsize=(3,3)); plt.
↪plot(x,y);
```



[26]:
```
prot = np.array([20.70, 27.46, 22.15, 19.85, 21.29, 24.75, 20.75, 22.91, 25.34,␣
↪20.33, 21.54, 21.08,
                 22.14, 19.56, 21.10, 18.04, 24.12, 19.95, 19.72, 18.28, 16.26,␣
↪17.46, 20.53, 22.12,
                 25.06, 22.44, 19.08, 19.88, 21.39, 22.33, 25.79])
N = prot.shape[0]
null_mean = 20
print(f'There are {N} samples with mean {prot.mean()}, std dev = {prot.
↪std(ddof=1)} and variance {prot.var()}')
print(f'Note std is unbiased: direct calc gives ', end=' ')
print(np.sqrt(np.power(prot,2).sum()/N - np.power(prot.sum()/N,2)), end=', ')
print(f'or just use ', prot.std())
diff_mean = prot.mean() - null_mean
print(f'difference from null mean: {prot.mean()} - {null_mean} = {diff_mean}')
std_err = prot.std(ddof=1)/np.sqrt(N)
tstat = diff_mean/std_err
print(f'standard error = {std_err}, and t statistic = {tstat}')
alpha = 0.05
tcrit = t.ppf(q=1-alpha/2, df=N-1)  # q is lower tail probability: => two-sided
print(f'with alpha = {alpha}, critical t is {tcrit}')
print(f'{tstat} > {tcrit} so we reject the null hypothesis at {100*alpha}%')
print(f'the p value is {2*t.cdf(-tstat, df=N-1)}')
```

```
There are 31 samples with mean 21.400000000000002, std dev = 2.5416687431685507
and variance 6.251690322580647
Note std is unbiased: direct calc gives  2.5003380416616756, or just use
2.5003380416616965
difference from null mean: 21.400000000000002 - 20 = 1.4000000000000021
standard error = 0.4564971822688009, and t statistic = 3.0668316352840814
with alpha = 0.05, critical t is 2.0422724563012373
3.0668316352840814 > 2.0422724563012373 so we reject the null hypothesis at 5.0%
the p value is 0.004552621060635394
```

[30]:
```
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_1samp.
↪html#scipy.stats.ttest_1samp
from scipy.stats import ttest_1samp
#stat, pval = ttest_1samp(prot, 20, axis=0, alternative='two-sided')
stat, pval = ttest_1samp(prot, 20, axis=0)
print(f'results are stat = {stat} and p value = {pval}')
#ttest_1samp(prot, 20, axis=0, alternative='two-sided')
ttest_1samp(prot, 20, axis=0)
```

```
results are stat = 3.0668316352840814 and p value = 0.004552621060635394
```

[30]:
```
Ttest_1sampResult(statistic=3.0668316352840814, pvalue=0.004552621060635394)
```

3 Coin tossing

Toss a coin 12 times, with a bias. A head is 1, a tail is zero.

[31]:
```
# random integers from low (inclusive) to high (exclusive).
bias = 64 #  bias+1 is probability of a head.
tosses = np.random.randint(0, high=100, size=12, dtype=int)
print(tosses)
indx = tosses>bias
print(indx)
tosses[indx] = 1
tosses[~indx] = 0
print(tosses)
```

```
[25 25 12  5 72 68 35 14 55 35 57 79]
[False False False False  True  True False False False False False  True]
[0 0 0 0 1 1 0 0 0 0 0 1]
```

3.0.1 Exercise

- test the hypothesis that the coin is fair.
- put the test in a loop. How many times would you expect to reject the null?

4 Photo Compression

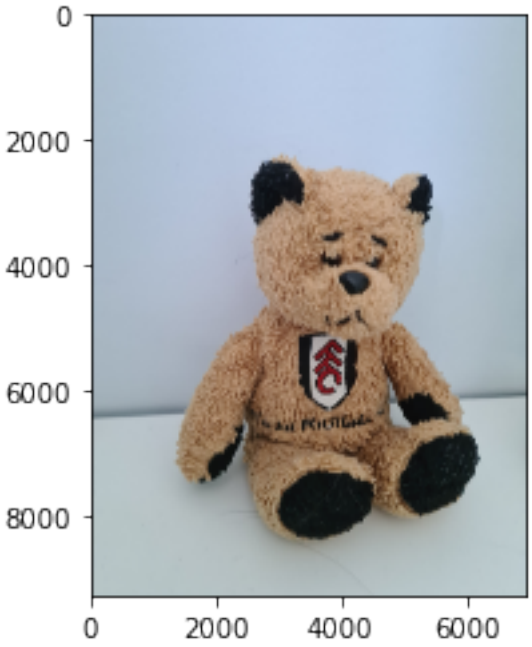
Either get your own jpeg or use one of the supplied ones...

[32]:
```
from PIL import Image
import IPython.display
# Use a jpeg photo - ffc.jpg is about 6.2MB (use your own path/filename here)
IPython.display.Image(filename='./gfx/ffc.jpg', width = 150)
```

[32]:

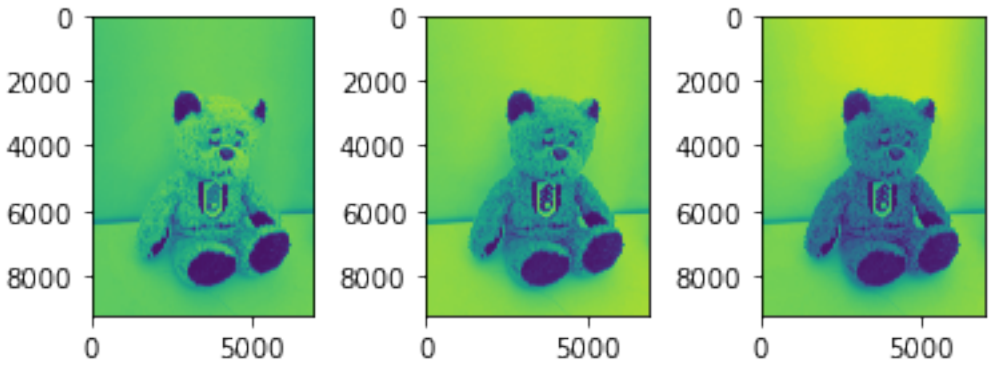
[33]:
```
# that is just a display, so ... load in the FFC bear - Roy - and visually␣
↪check him.
```

```
img = Image.open('./gfx/ffc.jpg')
# convert him to a numpy array for processing as a matrix
a = np.asarray(img)
im_orig = Image.fromarray(a)
plt.imshow(im_orig);
```



This image is made up of pixels where each pixel has a value for RED, GREEN and BLUE. We can get these colour bands and show them as follows...

[34]:
```
# convert band 'bnd' to a numpy array and show them...
for bnd in range(3):
  plt.subplot(1, 3, 1+bnd)
  img_mat = np.array(list(img.getdata(bnd)), float)
  img_mat = np.matrix(img_mat)
  img_mat.shape = (img.size[1], img.size[0])
  plt.imshow(img_mat)
plt.subplots_adjust(wspace=0.5)
```



[35]:
```
# get the red, green and blue bands as separate objects...
rband =img.getdata(band=0)
gband =img.getdata(band=1)
bband =img.getdata(band=2)
# and convert each to a numpy arrays for maths processing
imgr_mat = np.array(list(rband), float)
imgg_mat = np.array(list(gband), float)
imgb_mat = np.array(list(bband), float)
# each of these is about 64k elements
print('sizes = ', imgr_mat.size, imgg_mat.size, imgb_mat.size)
print('shapes = ', imgr_mat.shape, imgg_mat.shape, imgb_mat.shape)
```

```
sizes =  64144128 64144128 64144128
shapes =  (64144128,) (64144128,) (64144128,)
```

[36]:
```
# get image shape - we can assume they are all the same
imgr_mat.shape = imgg_mat.shape = imgb_mat.shape = (img.size[1], img.size[0])
print('imgr_mat.shape = ', imgr_mat.shape)
print('imgg_mat.shape = ', imgg_mat.shape)
print('imgb_mat.shape = ', imgb_mat.shape)
# convert these 1D-arrays to matrices
imgr_mat1D = np.matrix(imgr_mat)
imgg_mat1D = np.matrix(imgg_mat)
imgb_mat1D = np.matrix(imgb_mat)
print(type(imgb_mat))
```

```
imgr_mat.shape =  (9248, 6936)
imgg_mat.shape =  (9248, 6936)
imgb_mat.shape =  (9248, 6936)
<class 'numpy.ndarray'>
```

Using the Singular Value Decomposition we can hope to compress these objects.

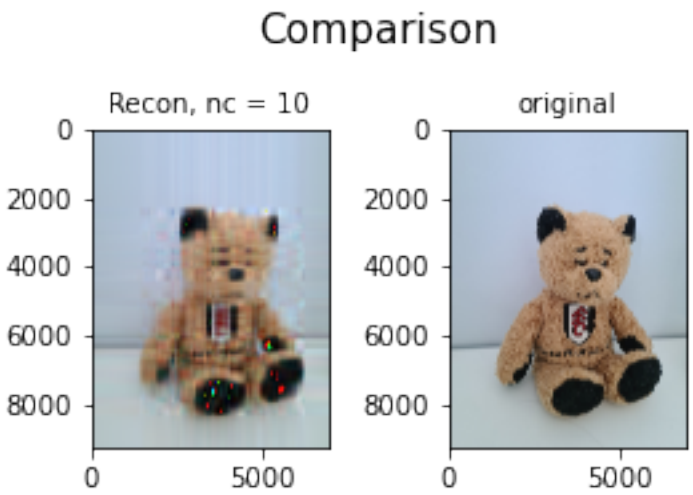
First get the SVD's of the R, G and B layers… (takes a while)

[37]:
```
Ur, Sr, VTr = np.linalg.svd(imgr_mat)
Ug, Sg, VTg = np.linalg.svd(imgg_mat)
Ub, Sb, VTb = np.linalg.svd(imgb_mat)
print(f'RED: shapes of Ur, Sr, VTr = {Ur.shape}, {Sr.shape}, {VTr.shape}')
print(f'GREEN: shapes of Ug, Sg, VTg = {Ug.shape}, {Sg.shape}, {VTg.shape}')
print(f'BLUE: shapes of Ub, Sb, VTb = {Ub.shape}, {Sb.shape}, {VTb.shape}')
```

```
RED: shapes of Ur, Sr, VTr = (9248, 9248), (6936,), (6936, 6936)
GREEN: shapes of Ug, Sg, VTg = (9248, 9248), (6936,), (6936, 6936)
BLUE: shapes of Ub, Sb, VTb = (9248, 9248), (6936,), (6936, 6936)
```

[38]:
```
# choose the number of components to use in the reconstruction
nc = 10 # 1387
rec_imgr = np.matrix(Ur[:, :nc]) * np.diag(Sr[:nc]) * np.matrix(VTr[:nc, :])
rec_imgg = np.matrix(Ug[:, :nc]) * np.diag(Sg[:nc]) * np.matrix(VTg[:nc, :])
rec_imgb = np.matrix(Ub[:, :nc]) * np.diag(Sb[:nc]) * np.matrix(VTb[:nc, :])
img_all = np.array([rec_imgr, rec_imgg, rec_imgb]).T
img_all = np.swapaxes(img_all,0,1)
PIL_image = Image.fromarray(np.uint8(img_all)).convert('RGB')
PIL_image.show()   # uncomment this to spawn an external viewer
PIL_image.save("ffc_recon.jpg")  # save the reconstruction if you like
```

[39]:
```
fig=plt.figure(figsize=(4, 3)); fig.suptitle('Comparison', fontsize=15)
plt.subplot(1,2,1); ax = plt.gca(); plt.subplots_adjust(wspace=0.5)
im = Image.fromarray(np.uint8(img_all)).convert('RGB'); ax.imshow(im)
ax.set_title(f'Recon, nc = {nc}', fontsize=10)
plt.subplot(1,2,2); ax = plt.gca(); ax.imshow(im_orig); ax.
↪set_title('original', fontsize=10);
```



Exercise Calculate the memory saving as a function of `nc` and plot it.

4.1 Going Further

if you want to see uses of python in machine learning and data science then you can look at my MA5634 binder page here:

https://mybinder.org/v2/gh/variationalform/FML.git/HEAD

This code creates a button - but it breaks LaTeX output

```
<p><a rel="noopener" href="https://mybinder.org/v2/gh/variationalform/FML.git/HEAD">https://myb
<p>Or just click this button: <a rel="noopener" href="https://mybinder.org/badge_logo.svg"
<p></p>
```

The raw materials are on git:

- https://variationalform.github.io
- https://github.com/variationalform/PythonMathsPrimer

4.2 Technical Notes, Production and Archiving

Ignore the material below. What follows is not relevant to the material being taught.

Production Workflow

- Finalise the notebook material above
- Clear and fresh run of entire notebook
- Create html slide show:
  - `jupyter nbconvert --to slides PythonMathsPrimer.ipynb`
- Set `OUTPUTTING=1` below
- Comment out the display of web-sourced diagrams
- Clear and fresh run of entire notebook
- Comment back in the display of web-sourced diagrams
- Clear all cell output
- Set `OUTPUTTING=0` below
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
  - git add, commit and push
- rebuild binder

[40]:
```
%%bash
NBROOTNAME=PythonMathsPrimer
OUTPUTTING=1

if [ $OUTPUTTING -eq 1 ]; then
  #jupyter nbconvert --to html $NBROOTNAME.ipynb
  #cp $NBROOTNAME.html ./backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.html
  #mv -f $NBROOTNAME.html ./formats/

  jupyter nbconvert --to slides $NBROOTNAME.ipynb
```

```
  cp $NBROOTNAME.slides.html ./backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.
 ↪slides.html
  mv -f $NBROOTNAME.slides.html ./formats/

  jupyter nbconvert --to pdf $NBROOTNAME.ipynb
  cp $NBROOTNAME.pdf ./backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.pdf
  mv -f $NBROOTNAME.pdf ./formats/

  jupyter nbconvert --to script $NBROOTNAME.ipynb
  cp $NBROOTNAME.py ./backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.py
  mv -f $NBROOTNAME.py ./formats/
else
  echo 'Not Generating html, pdf and py output versions'
fi
```

```
[NbConvertApp] Converting notebook PythonMathsPrimer.ipynb to slides
[NbConvertApp] Writing 9407933 bytes to PythonMathsPrimer.slides.html
[NbConvertApp] Converting notebook PythonMathsPrimer.ipynb to pdf
[NbConvertApp] Support files will be in PythonMathsPrimer_files/
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Making directory ./PythonMathsPrimer_files
[NbConvertApp] Writing 94293 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 6617295 bytes to PythonMathsPrimer.pdf
[NbConvertApp] Converting notebook PythonMathsPrimer.ipynb to script
[NbConvertApp] Writing 22206 bytes to PythonMathsPrimer.py
```

[ ]: