B_worksheet

February 2, 2025

1 Worksheet B

variationalform https://variationalform.github.io/

Just Enough: progress at pace https://variationalform.github.io/

https://github.com/variationalform

Simon Shaw https://www.brunel.ac.uk/people/simon-shaw.

This document uses python

and also makes use of LaTeX

in Markdown

1.1 What this is about:

This worksheet is based on the material in the notebooks

- matrices: matrix concepts and algebra
- systems: systems of linear equations, under- and over-determined cases
- decomp: eigensystem decomposition and SVD.

Note that while the 'lecture' notebooks are prefixed with `1_`, `2_` and so on, to indicate the order in which they should be studied, the worksheets are prefixed with `A_`, `B_`, …

1.2 Exercise 1

Rebuild $\boldsymbol{L}$ from the notes in `numpy`,

$$\boldsymbol{L} = \begin{pmatrix} 1 & 5 & -7 \\ -3 & -4 & 0 \end{pmatrix}.$$

1. What does `L.shape` give? (Look at the dimensions of $\boldsymbol{L}$.)
2. What does `L.ndim` give? ($\boldsymbol{L}$ is two-dimensional.)
3. Examine these statements. What do they do?

- `np.ones(3)`

- `np.ones([3,1])`
- `np.ones([2,4])`
- `np.zeros([2,4])`
- `np.eye(4)`

4. Try these - explain the results:

- `D = L.reshape([3,2])`
- `E = L.reshape([6,])`

5. What about these?

- `np.arange(4)`
- `np.arange(2,8)`
- `np.arange(2,9,2)`
- `np.linspace(4,9,num=6)`
- `np.linspace(4,9,num=11)`

`numpy` is very powerful: see this introduction for more details `https://numpy.org/doc/stable/user/absolute_beginners.html`

1.3 Exercise 2

In the lecture we decided that this is an under-determined system:

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{f} \qquad \text{for } \boldsymbol{B} = \begin{pmatrix} 3 & -2 & 4 \\ -6 & 6 & -11 \\ 6 & -4 & 8 \end{pmatrix} \text{ and } \boldsymbol{f} = \begin{pmatrix} 2 \\ -1 \\ 4 \end{pmatrix}$$

We claimed that $x = (10 - 2z)/6$ and $y = (9 + 9z)/6$ will solve this system for any choice of z

1. Derive these solutions (Hint: move z to the right and solve the 2×2 system).
2. For large matrices finding the inverse is not practical. Fortunately, numpy can deal with under-determined systems using a least squares procedure. Try the following code. It's non-trivial, but it does illustrate this very advanced functionality.

```
u,_,_,_ = np.linalg.lstsq(B,f,rcond=None)
print(u)
print(f-B.dot(u))
```

1.4 Exercise 3

We decided in the lecture that this is an over-determined system

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{f} \qquad \text{for } \boldsymbol{B} = \begin{pmatrix} 3 & -2 \\ -6 & 6 \\ 6 & 2 \end{pmatrix} \text{ and } \boldsymbol{f} = \begin{pmatrix} 6 \\ -18 \\ 7 \end{pmatrix}$$

and that a solution cannot exist. Try the *least squares* routine from `numpy` here and see what it produces.

1.5 Exercise 4

In the lecture we saw that

$$\text{if } \boldsymbol{A} = \begin{pmatrix} 3 & -2 & 4 \\ -2 & 6 & 2 \\ 4 & 2 & 5 \end{pmatrix}$$

then (with some rounding),

$$\boldsymbol{D} \approx \begin{pmatrix} -1.217 & 0 & 0 \\ 0 & 8.217 & 0 \\ 0 & 0 & 7 \end{pmatrix} \quad \text{and} \quad \boldsymbol{V} \approx \begin{pmatrix} .726 & .522 & -.447 \\ .363 & .261 & .894 \\ -.584 & .812 & 0 \end{pmatrix}$$

- verify this with code.
- verify that $\boldsymbol{V}^{-1} = \boldsymbol{V}^T$. Hint: `D=np.diag(w)`.
- verify also that $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$

1.6 Exercise 5

In the lecture we saw that after

```
A = np.array([[3,-2,4],[-2,6,2],[4,2,5]])
w, V = np.linalg.eig(A)
D=np.diag(w)
```

we could re-construct $\boldsymbol{A}$ with

```
D[0,0]*V[:,0:1]*V[:,0:1].T + D[1,1]*V[:,1:2]*V[:,1:2].T + D[2,2]*V[:,2:3]*V[:,2:3].T
```

Investigate *python for loops* and re-code this using such a loop.

Hint: a python for loop takes the form

```
for k in range(0,3):
    do something with k = 0,1,2 in turn
now carry on with something else
```

The indent is important here - it is the *loop body*. Jupyer gives you a four-space indent, but you can use more or less (but at least one) spaces.

See e.g. https://www.learnpython.org/en/Loops

1.7 Exercise 6

Find the eigen-decomposition of the matrix

$$\boldsymbol{T} = \begin{pmatrix} 7 & -3 & -9 \\ -3 & -5 & 2 \\ -9 & 2 & 10 \end{pmatrix}$$

Order the eigenvalues so that $|\lambda_1| \geq |\lambda_2| \geq |\lambda_3|$ and determine the partial recontructions,

- $\boldsymbol{T}_1 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T$
- $\boldsymbol{T}_2 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \lambda_2 \boldsymbol{v}_2 \boldsymbol{v}_2^T$

- $\boldsymbol{T}_3 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \lambda_2 \boldsymbol{v}_2 \boldsymbol{v}_2^T + \lambda_3 \boldsymbol{v}_3 \boldsymbol{v}_3^T$

Finally, check that $\boldsymbol{T}_3 = \boldsymbol{T}$.

1.8 Exercise 7

1. If $\boldsymbol{A}$ is $n \times n$ symmetric then how many independent quantities does it contain?
2. If we approximate

$$\boldsymbol{A} = \sum_{k=1}^{n} \lambda_k \boldsymbol{v}_k \boldsymbol{v}_k^T \qquad \text{by} \qquad \boldsymbol{A} \approx \sum_{k=1}^{m} \lambda_k \boldsymbol{v}_k \boldsymbol{v}_k^T$$

   for some $k < n$, then how many independent quantities does this expression contain?
3. What is the ratio of approximate size to exact size?
4. Evaluate that ratio when $m = 5$ and $n = 1000$

1.9 Exercise 8

Based originally on https://stackoverflow.com/questions/8092920/sort-eigenvalues-and-associated-eigenvectors-after-using-numpy-linalg-eig-in-pyt

Let's think about …

- How can we use code to re-order the eigenvalues in `w` so that their absolute values are in descending order?
- How can we use that re-ordering to correctly re-order the eigenvectors that are in the columns of V?

First of all let's use the matrix $\boldsymbol{A}$ that we saw in lectures.

[1]:
```python
import numpy as np
A = np.array([[3,-2,4],[-2,6,2],[4,2,5]])
w, V = np.linalg.eig(A)
D=np.diag(w)

print('V = \n', V)
print('D = \n', D)
print('    w  = ', w)
print('abs(w) = ', abs(w))
```

```
V =
 [[ 7.26085219e-01  5.22302838e-01 -4.47213595e-01]
 [ 3.63042610e-01  2.61151419e-01  8.94427191e-01]
 [-5.83952325e-01  8.11787954e-01  2.85088132e-16]]
D =
 [[-1.21699057  0.          0.        ]
 [ 0.          8.21699057  0.        ]
 [ 0.          0.          7.        ]]
    w  =  [-1.21699057  8.21699057  7.        ]
abs(w) =  [1.21699057 8.21699057 7.        ]
```

Now let's see how we can sort the eigenvalues by descending absolute value.

We get them first in ascending order, and then flip them.

[2]:
```
# ascending...
print('abs(w).argsort()            = ', abs(w).argsort())
# flip to get descending
print('np.flip( abs(w).argsort()) = ', np.flip( abs(w).argsort()))
# or, alternatively,
print('abs(w).argsort()[::-1]      = ', abs(w).argsort()[::-1])
# the first is tidier - use it to store the indices.
indx = np.flip( abs(w).argsort() )
print('indx = ', indx)
```

```
abs(w).argsort()            =  [0 2 1]
np.flip( abs(w).argsort()) =  [1 2 0]
abs(w).argsort()[::-1]      =  [1 2 0]
indx =  [1 2 0]
```

Now we can reassign `w` and `V` in descending order.

[3]:
```
w = w[indx]
V = V[:,indx]
print('lambda = ', w)
print('V = \n', V)
```

```
lambda =  [ 8.21699057  7.          -1.21699057]
V =
 [[ 5.22302838e-01 -4.47213595e-01  7.26085219e-01]
 [ 2.61151419e-01  8.94427191e-01  3.63042610e-01]
 [ 8.11787954e-01  2.85088132e-16 -5.83952325e-01]]
```

Re-visit the work above to find the eigen-decomposition of the matrix $\boldsymbol{T}$ as given by,

$$\boldsymbol{T} = \begin{pmatrix} 7 & -3 & -9 \\ -3 & -5 & 2 \\ -9 & 2 & 10 \end{pmatrix}$$

Use code, as descibed above, to order the eigenvalues so that $|\lambda_1| \geq |\lambda_2| \geq |\lambda_3|$ and determine the partial recontructions,

- $\boldsymbol{T}_1 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T$
- $\boldsymbol{T}_2 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \lambda_2 \boldsymbol{v}_2 \boldsymbol{v}_2^T$
- $\boldsymbol{T}_3 = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \lambda_2 \boldsymbol{v}_2 \boldsymbol{v}_2^T + \lambda_3 \boldsymbol{v}_3 \boldsymbol{v}_3^T$

Finally, check that $\boldsymbol{T}_3 = \boldsymbol{T}$.

Recall that in the lecture we sorted the eigenvalues manually...

We have now seen how to automate the sorting.

1.9.1 Extra Challenge...

Use a python for loop to determine those partial sums for $\boldsymbol{T}_i$, $i - 1, 2, 3$.

Illustrate the relative errors in the partial and full reconstructions.

Hint: you can use `np.allclose(A,B)` to see if two matrices are the same up toa given tolerance.

And: you can use `np.linalg.norm(A-B)` to check the norm, or *size*, of $\boldsymbol{A} - \boldsymbol{B}$.

2 Outline solutions

First import what we need...

[4]:
```
import numpy as np
```

2.1 Exercise 1 - outline solutions

[5]:
```
L = np.array([[1, 5, -7],[-3, -4, 0]])
print(L.ndim)
print(L.shape)
print(np.ones(3))
print(np.ones([3,1]))
print(np.ones([2,4]))
print(np.zeros([2,4]))
print(np.eye(4))

print(L)
D = L.reshape([3,2])
print(D)
E = L.reshape([6,])
print(E)

print(np.arange(4))
print(np.arange(2,8))
print(np.arange(2,9,2))
print(np.linspace(4,9,num=6))
print(np.linspace(4,9,num=11))
```

```
2
(2, 3)
[1. 1. 1.]
[[1.]
 [1.]
 [1.]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
[[ 1  5 -7]
 [-3 -4  0]]
[[ 1  5]
 [-7 -3]
 [-4  0]]
[ 1  5 -7 -3 -4  0]
[0 1 2 3]
[2 3 4 5 6 7]
[2 4 6 8]
[4. 5. 6. 7. 8. 9.]
[4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5 9. ]
```

2.2 Exercise 2 - outline solutions

In the lecture we decided that this is an under-determined system:

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{f} \qquad \text{for } \boldsymbol{B} = \begin{pmatrix} 3 & -2 & 4 \\ -6 & 6 & -11 \\ 6 & -4 & 8 \end{pmatrix} \text{ and } \boldsymbol{f} = \begin{pmatrix} 2 \\ -1 \\ 4 \end{pmatrix}$$

Write this as

$$\begin{pmatrix} 3 & -2 \\ -6 & 6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2-4z \\ -1+11z \end{pmatrix}$$

and then use the inverse matrix (easy for a 2×2) to get

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{6} \begin{pmatrix} 6 & 2 \\ 6 & 3 \end{pmatrix} \begin{pmatrix} 2-4z \\ -1+11z \end{pmatrix}$$

These are the formulae we saw in the lecture notes.

[6]:
```python
B = np.array( [[3, -2, 4],[-6, 6, -11],[ 6, -4, 8 ]])
f = np.array([[2], [-1], [4]])
```

[7]:
```python
z = 0
x = (10-2*z)/6
y = (9+9*z)/6
u = np.array([[x],[y],[z]])
print(B.dot(u))
```

```
[[ 2.]
 [-1.]
 [ 4.]]
```

On the other hand, for large matrices finding the inverse is not practical.

Fortunately, `numpy` can deal with under-determined systems using a *least squares* procedure.

```
[8]: u,_,_,_ = np.linalg.lstsq(B,f,rcond=None)
     print(u)
     print(f-B.dot(u))
```

```
[[ 1.83471074]
 [ 0.74380165]
 [-0.50413223]]
[[-2.66453526e-15]
 [ 6.21724894e-15]
 [-5.32907052e-15]]
```

Note that the result of $\boldsymbol{f} - \boldsymbol{B}\boldsymbol{u} = \boldsymbol{0}$ up to rounding error. We'd expect this because we know that solutions exist.

2.3 Exercise 3 - outline solution

For this over-determined system

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{f} \qquad \text{for } \boldsymbol{B} = \begin{pmatrix} 3 & -2 \\ -6 & 6 \\ 6 & 2 \end{pmatrix} \text{ and } \boldsymbol{f} = \begin{pmatrix} 6 \\ -18 \\ 7 \end{pmatrix}$$

a solution cannot exist. However, we can try the *least squares* routine from `numpy` here and see what it produces…

```
[9]: B = np.array( [[3, -2],[-6, 6],[ 6, 2]])
     f = np.array([[6], [-18], [7]])

     u,_,_,_ = np.linalg.lstsq(B,f,rcond=None)
     print(u)
     print(f-B.dot(u))
```

```
[[ 1.58108108]
 [-1.33108108]]
[[-1.40540541]
 [-0.52702703]
 [ 0.17567568]]
```

Note that the result of $\boldsymbol{f} - \boldsymbol{B}\boldsymbol{u} \neq \boldsymbol{0}$. Again, we expect this because we know that no solution can exist.

So, what's happening here? Well `numpy` is trying to give us a *best* attempt at a solution…

2.4 Exercise 4 - outline solution

[10]:
```
A = np.array([[3,-2,4],[-2,6,2],[4,2,5]])
w, V = np.linalg.eig(A)
D=np.diag(w)
print('A=\n',A,'\n','w=\n',w,'\n','D=\n',D,'\n','V=\n',V)
print('V V^T = ...\n',V.dot(V.T))
print('\n error ...\n',A-V.dot(D.dot(V.T)))

print('sum =', np.sum(A-V.dot(D.dot(V.T))) )
assert np.sum(A-V.dot(D.dot(V.T))) < 0.0001
```

```
A=
 [[ 3 -2  4]
 [-2  6  2]
 [ 4  2  5]]
 w=
 [-1.21699057  8.21699057  7.        ]
 D=
 [[-1.21699057  0.          0.        ]
 [ 0.          8.21699057  0.        ]
 [ 0.          0.          7.        ]]
 V=
 [[ 7.26085219e-01  5.22302838e-01 -4.47213595e-01]
 [ 3.63042610e-01  2.61151419e-01  8.94427191e-01]
 [-5.83952325e-01  8.11787954e-01  2.85088132e-16]]
V V^T = …
 [[ 1.00000000e+00 -1.55686504e-16  3.16593922e-16]
 [-1.55686504e-16  1.00000000e+00 -3.00120936e-16]
 [ 3.16593922e-16 -3.00120936e-16  1.00000000e+00]]

 error …
 [[-5.32907052e-15  2.66453526e-15 -3.55271368e-15]
 [ 2.22044605e-15  2.66453526e-15  4.44089210e-16]
 [-4.44089210e-15  6.66133815e-16  8.88178420e-16]]
sum = -3.774758283725532e-15
```

2.5 Exercise 5 - outline solution

[11]:
```
A = np.array([[3,-2,4],[-2,6,2],[4,2,5]])
w, V = np.linalg.eig(A)
D=np.diag(w)
print(A)
# from the lecture
print('\n The reconstruction of A ...')
print(D[0,0]*V[:,0:1]*V[:,0:1].T + D[1,1]*V[:,1:2]*V[:,1:2].T + D[2,2]*V[:,2:
 ↪3]*V[:,2:3].T)
```

```
# using a for loop
M = np.zeros(A.shape)
for k in range(0,3):
    M += D[k,k]*V[:,k:k+1]*V[:,k:k+1].T
print(M)
```

```
[[ 3 -2  4]
 [-2  6  2]
 [ 4  2  5]]

 The reconstruction of A …
[[ 3. -2.  4.]
 [-2.  6.  2.]
 [ 4.  2.  5.]]
[[ 3. -2.  4.]
 [-2.  6.  2.]
 [ 4.  2.  5.]]
```

2.6 Exercise 6 - outline solution

[12]:
```
# possible solution
T = np.array([[7,-3,-9],[-3,-5,2],[-9,2,10]])
w, V = np.linalg.eig(T)
# look at the eigenvalues to manually sort them...
print('lambda = ', w)
```

```
lambda =  [18.14414013 -0.43436748 -5.70977265]
```

We can see that $|\lambda_1| > |\lambda_3| > |\lambda_2|$ and so, we can select them out in the correct order manually like this…

[13]:
```
D=np.diag(w)
T1 = D[0,0]*V[:,0:1]*V[:,0:1].T       # using lambda_1
T2 = T1 + D[2,2]*V[:,2:3]*V[:,2:3].T  # using lambda_2
T3 = T2 + D[1,1]*V[:,1:2]*V[:,1:2].T  # using lambda_3
print('T1 = \n', T1)
print('T2 = \n', T2)
print('T-T3 = \n', T-T3)
```

```
T1 =
 [[ 7.55329941 -1.7372557  -8.77369555]
 [-1.7372557   0.39956808  2.0179463 ]
 [-8.77369555  2.0179463  10.19127264]]
T2 =
 [[ 7.22886142 -3.05895812 -8.79129842]
 [-3.05895812 -4.98481151  1.94623536]
 [-8.79129842  1.94623536 10.19031757]]
T-T3 =
 [[-4.44089210e-15  2.66453526e-15  7.10542736e-15]
 [ 2.66453526e-15  8.88178420e-16 -8.88178420e-16]
 [ 7.10542736e-15 -8.88178420e-16 -8.88178420e-15]]
```

2.7 Exercise 7 - outline solution

1. There are n^2 entries in the matrix but those under the diagonal are replicated above the diagonal. Hence, by adding the length of the diagonal to the first, and then second, and then third, …, super-diagonals there are just

$$n + (n-1) + (n-2) + \cdots + 2 + 1 = \frac{(n+1)n}{2}.$$

2. $\boldsymbol{v}_k$ has n entries, and λ_k is just one number. Hence this expression has just $m(n+1)$.
3. The ratio is therefore

$$m(n+1) \div \frac{n(n+1)}{2} = \frac{2m(n+1)}{n(n+1)} = \frac{2m}{n}.$$

4. $m = 5$ and $n = 1000$ gives the ratio $\dfrac{2m}{n} = \dfrac{2 \times 5}{1000} = 1\%$.

2.8 Exercise 8 - outline solution

[14]:
```
T = np.array([[7,-3,-9],[-3,-5,2],[-9,2,10]])
print(T)
w, V = np.linalg.eig(T)
indx = np.flip( abs(w).argsort() )
w = w[indx]
V = V[:,indx]
D=np.diag(w)
T1 = D[0,0]*V[:,0:1]*V[:,0:1].T
T2 = T1 + D[1,1]*V[:,1:2]*V[:,1:2].T
T3 = T2 + D[2,2]*V[:,2:3]*V[:,2:3].T
print('T1 = \n', T1)
print('T2 = \n', T2)
print('T-T3 = \n', T-T3)
```

```
[[ 7 -3 -9]
 [-3 -5  2]
 [-9  2 10]]
T1 =
 [[ 7.55329941 -1.7372557  -8.77369555]
 [-1.7372557   0.39956808  2.0179463 ]
 [-8.77369555  2.0179463  10.19127264]]
T2 =
 [[ 7.22886142 -3.05895812 -8.79129842]
 [-3.05895812 -4.98481151  1.94623536]
 [-8.79129842  1.94623536 10.19031757]]
T-T3 =
 [[-4.44089210e-15  2.66453526e-15  7.10542736e-15]
 [ 2.66453526e-15  8.88178420e-16 -8.88178420e-16]
 [ 7.10542736e-15 -8.88178420e-16 -8.88178420e-15]]
```

2.8.1 Extra - outline solution

[15]:
```
E = np.zeros(T.shape)
for i in range(0,3):
    E = E + w[i] * V[:,i:i+1] @ V[:,i:i+1].T
    print('E=\n',E)
    print('(A-E)/A', (T-E)/T)
    print('max{ (A-E)/A }', np.max( (T-E)/T) )
```

```
E=
 [[ 7.55329941 -1.7372557  -8.77369555]
 [-1.7372557   0.39956808  2.0179463 ]
 [-8.77369555  2.0179463  10.19127264]]
(A-E)/A [[-0.07904277  0.42091477  0.02514494]
 [ 0.42091477  1.07991362 -0.00897315]
 [ 0.02514494 -0.00897315 -0.01912726]]
max{ (A-E)/A } 1.0799136164905145
E=
 [[ 7.22886142 -3.05895812 -8.79129842]
 [-3.05895812 -4.98481151  1.94623536]
 [-8.79129842  1.94623536 10.19031757]]
(A-E)/A [[-0.03269449 -0.01965271  0.02318906]
 [-0.01965271  0.0030377   0.02688232]
 [ 0.02318906  0.02688232 -0.01903176]]
max{ (A-E)/A } 0.026882320613163202
E=
 [[ 7. -3. -9.]
 [-3. -5.  2.]
 [-9.  2. 10.]]
(A-E)/A [[-6.34413157e-16 -8.88178420e-16 -7.89491929e-16]
 [-8.88178420e-16 -1.77635684e-16 -4.44089210e-16]
 [-7.89491929e-16 -4.44089210e-16 -8.88178420e-16]]
max{ (A-E)/A } -1.7763568394002506e-16
```

2.9 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 B_worksheet.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

Some of this originated from

https://stackoverflow.com/questions/38540326/save-html-of-a-jupyter-notebook-from-within-the-r

These lines create a back up of the notebook. They can be ignored.

At some point this is better as a bash script outside of the notebook

[16]:
```
%%bash
NBROOTNAME='B_worksheet'
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/html/

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

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

```
Not Generating html, pdf and py output versions
```