Matrix of creation operator 4

November 22, 2023

[1]:
```
from numpy import *
import numpy as np
from math import *
import pandas as pd
from numpy.linalg import det
```

[2]:
```
n = 10
a1 = np.zeros((n,n))
a2 = np.zeros((n,n))
for i in range(n-1):
    a1[i+1,i]=sqrt(i+1)
    a2[i,i+1]=sqrt(i+1)
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 \end{pmatrix} = a^{\dagger}$$

[3]:
```
a = 1j*(a1 - a2)
print(pd.DataFrame(around(a1,4)))
```

```
     0       1       2    3       4       5       6       7    8    9
0  0.0  0.0000  0.0000  0.0  0.0000  0.0000  0.0000  0.0000  0.0  0.0
1  1.0  0.0000  0.0000  0.0  0.0000  0.0000  0.0000  0.0000  0.0  0.0
2  0.0  1.4142  0.0000  0.0  0.0000  0.0000  0.0000  0.0000  0.0  0.0
3  0.0  0.0000  1.7321  0.0  0.0000  0.0000  0.0000  0.0000  0.0  0.0
4  0.0  0.0000  0.0000  2.0  0.0000  0.0000  0.0000  0.0000  0.0  0.0
5  0.0  0.0000  0.0000  0.0  2.2361  0.0000  0.0000  0.0000  0.0  0.0
6  0.0  0.0000  0.0000  0.0  0.0000  2.4495  0.0000  0.0000  0.0  0.0
7  0.0  0.0000  0.0000  0.0  0.0000  0.0000  2.6458  0.0000  0.0  0.0
8  0.0  0.0000  0.0000  0.0  0.0000  0.0000  0.0000  2.8284  0.0  0.0
9  0.0  0.0000  0.0000  0.0  0.0000  0.0000  0.0000  0.0000  3.0  0.0
```

[4]:
```
def fak(n):
    a=1
    for i in range(n):
        a *= (i+1)
    return a
```

[5]:
```
def series(n, a):
    ident = eye(n,n)
    a_j = ident
    result = 0
    for j in range(n):
        a_j = matmul(a_j, a)
        result += a_j/fak(j+1)
    return result + ident
```

The exponential of the operator associated with

$$i\left(a^{\dagger}-a\right)$$

is indeed hermitian.

When the matrix is transposed, the imaginary part gets negated. As you can see below:

[6]:
```
result = series(n, a)
print(pd.DataFrame(around(result)))
```

```
          0         1         2          3          4          5          6  \
0  2.0+0.0j  0.0-2.0j -1.0+0.0j   0.0+1.0j   0.0+0.0j   0.0-0.0j  -0.0+0.0j
1  0.0+2.0j  3.0+0.0j  0.0-3.0j  -3.0+0.0j   0.0+2.0j   1.0+0.0j   0.0-0.0j
2 -1.0+0.0j  0.0+3.0j  6.0+0.0j   0.0-6.0j  -5.0+0.0j   0.0+3.0j   2.0+0.0j
3  0.0-1.0j -3.0+0.0j  0.0+6.0j   9.0+0.0j  0.0-10.0j  -8.0+0.0j   0.0+6.0j
4  0.0+0.0j  0.0-2.0j -5.0+0.0j  0.0+10.0j  14.0+0.0j  0.0-15.0j -13.0+0.0j
5  0.0+0.0j  1.0+0.0j  0.0-3.0j  -8.0+0.0j  0.0+15.0j  21.0+0.0j  0.0-22.0j
6 -0.0+0.0j  0.0+0.0j  2.0+0.0j   0.0-6.0j -13.0+0.0j  0.0+22.0j  30.0+0.0j
7  0.0-0.0j -0.0+0.0j  0.0+1.0j   3.0+0.0j   0.0-9.0j -19.0+0.0j  0.0+31.0j
8  0.0+0.0j  0.0-0.0j -0.0+0.0j   0.0+2.0j   5.0+0.0j  0.0-13.0j -25.0+0.0j
9  0.0+0.0j  0.0+0.0j  0.0-0.0j  -1.0+0.0j   0.0+2.0j   7.0+0.0j  0.0-14.0j

           7          8          9
0   0.0+0.0j   0.0+0.0j   0.0-0.0j
1  -0.0+0.0j   0.0+0.0j   0.0+0.0j
2   0.0-1.0j  -0.0+0.0j   0.0+0.0j
3   3.0+0.0j   0.0-2.0j  -1.0+0.0j
4   0.0+9.0j   5.0+0.0j   0.0-2.0j
5 -19.0+0.0j  0.0+13.0j   7.0+0.0j
6  0.0-31.0j -25.0+0.0j  0.0+14.0j
7  39.0+0.0j  0.0-37.0j -23.0+0.0j
```

```
8   0.0+37.0j   40.0+0.0j   0.0-26.0j
9 -23.0+0.0j    0.0+26.0j   18.0+0.0j
```

$ \ $ $ \ $ And the exponential of just the creation operator, gives this:

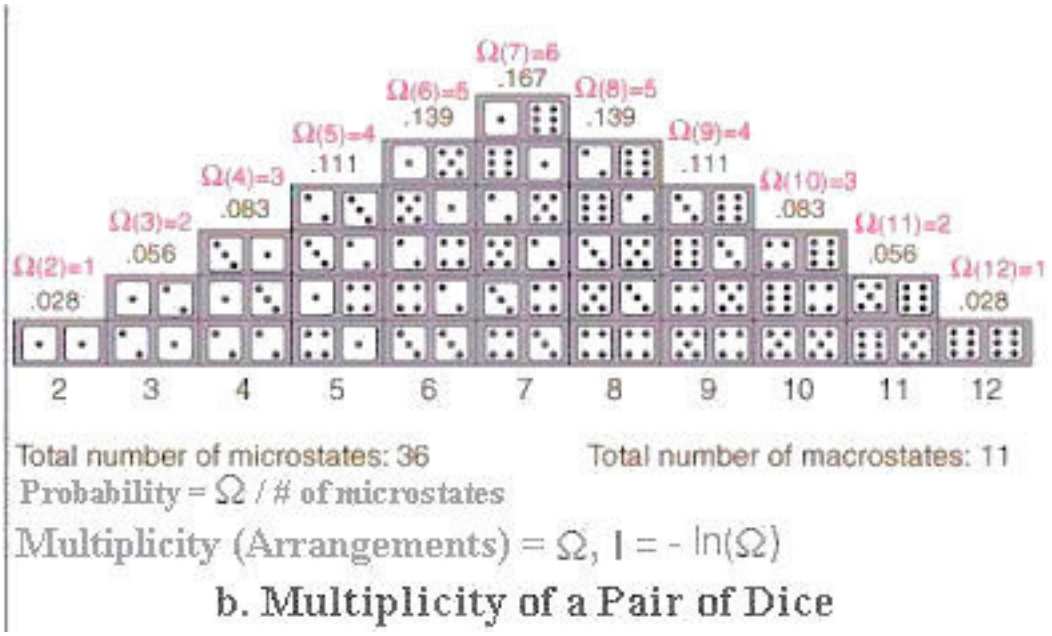
[7]:
```
a = a1
result = series(n, a)
print(pd.DataFrame(around(result,4)))
```

```
        0       1       2       3       4       5       6       7    8    9
0  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0  0.0
1  1.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0  0.0
2  0.7071  1.4142  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0  0.0
3  0.4082  1.2247  1.7321  1.0000  0.0000  0.0000  0.0000  0.0000  0.0  0.0
4  0.2041  0.8165  1.7321  2.0000  1.0000  0.0000  0.0000  0.0000  0.0  0.0
5  0.0913  0.4564  1.2910  2.2361  2.2361  1.0000  0.0000  0.0000  0.0  0.0
6  0.0373  0.2236  0.7906  1.8257  2.7386  2.4495  1.0000  0.0000  0.0  0.0
7  0.0141  0.0986  0.4183  1.2076  2.4152  3.2404  2.6458  1.0000  0.0  0.0
8  0.0050  0.0398  0.1972  0.6831  1.7078  3.0551  3.7417  2.8284  1.0  0.0
9  0.0017  0.0149  0.0845  0.3416  1.0247  2.2913  3.7417  4.2426  3.0  1.0
```

"Hidden" Pascals triangle: - Binomial coefficients in numerator - "Reverse" factorials in denominator - Square root of all entries

$$\begin{pmatrix} \sqrt{\frac{\binom{0}{0}}{0!}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{\frac{\binom{1}{0}}{1!}} & \sqrt{\frac{\binom{1}{1}}{0!}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{\frac{\binom{2}{0}}{2!}} & \sqrt{\frac{\binom{2}{1}}{1!}} & \sqrt{\frac{\binom{2}{2}}{0!}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{\frac{\binom{3}{0}}{3!}} & \sqrt{\frac{\binom{3}{1}}{2!}} & \sqrt{\frac{\binom{3}{2}}{1!}} & \sqrt{\frac{\binom{3}{3}}{0!}} & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & 1 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & 1 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix} = \exp(a^\dagger)$$

Because binomials are related to combinatorics, my first guess was, it could have something to do with entropy and the possible numbers of microstates.
Just by applying ln(x) after $\exp(x)$ again:



A more likely interpretation is, that it has to do, with the number of paths a particle can take.

So it's the sum over all paths, with a particle that can choose of two states:

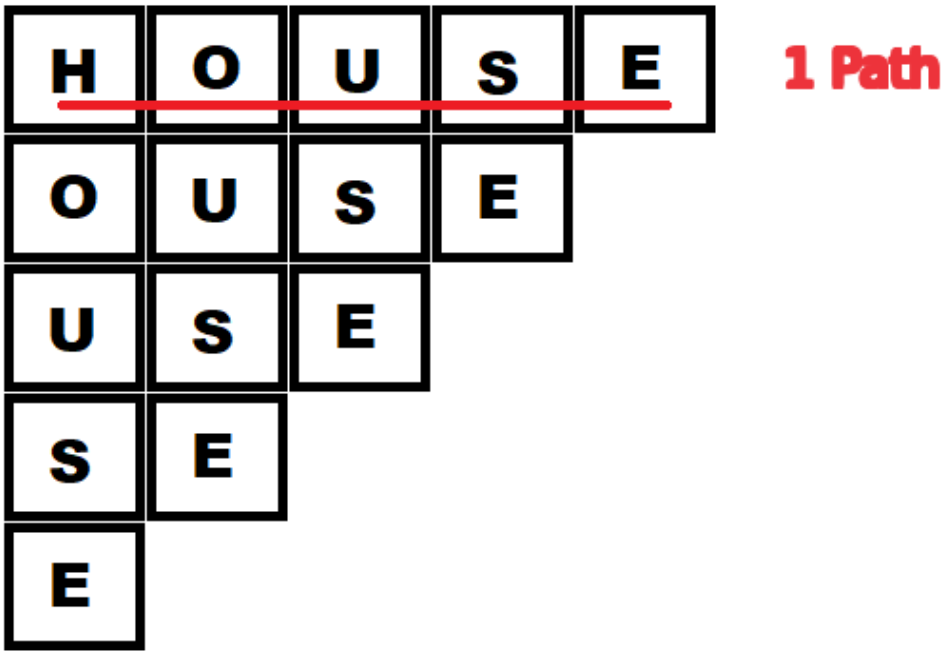
↑↓

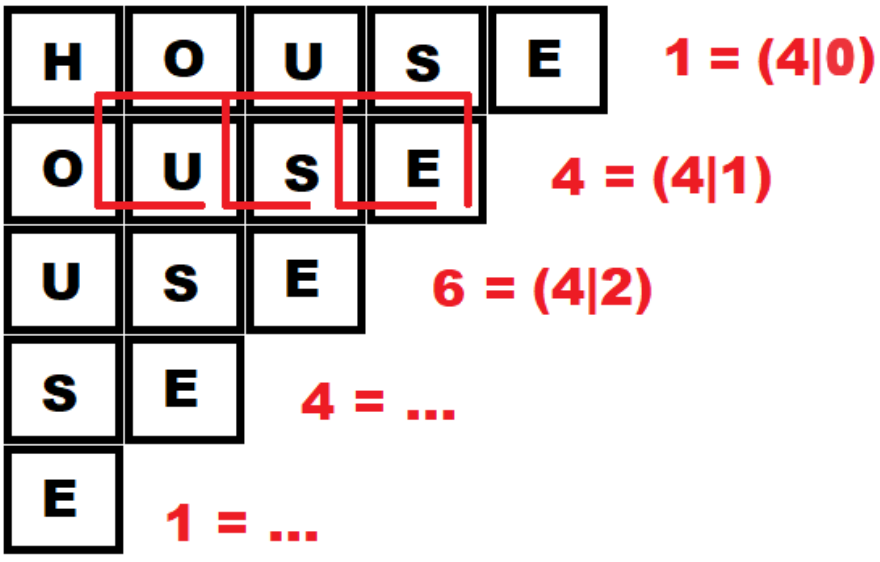
Like in this puzzle:

Assuming a particle can choose between the two possibilities of going right, or going down.

How many ways/paths are there, to write the word "House"

The result gives you a pascals triangle.





Possible ways for further research.

Taylor-series:

$$\exp z := \sum_{k=0}^{\infty} \frac{z^k}{k!}$$

The series you get by multiplying out the definition:

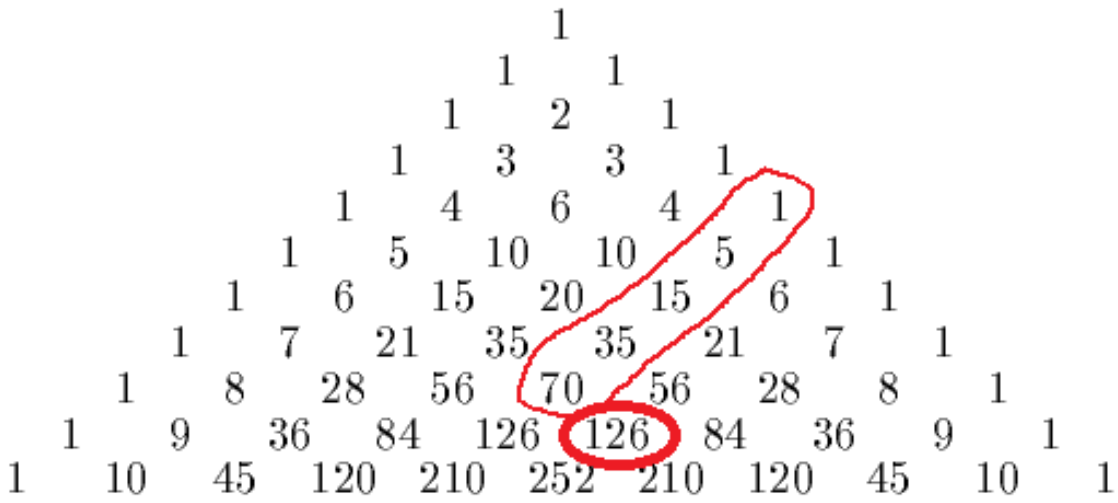
$$\exp z := \lim_{n\to\infty} \left(1+\frac{z}{n}\right)^n = \left(1+\frac{z}{n}\right)\left(1+\frac{z}{n}\right)\left(1+\frac{z}{n}\right)...$$

Another series I found, with a special kind of Monte-Carlo itegration:

$$\exp z := 1 + \lim_{n\to\infty} \frac{z}{n} \sum_{k=1}^{n} \left(1+\frac{z}{n}\right)^k$$

By comparing the last two series, it's easy to prove, that the binomials are the sum of the diagonals above.

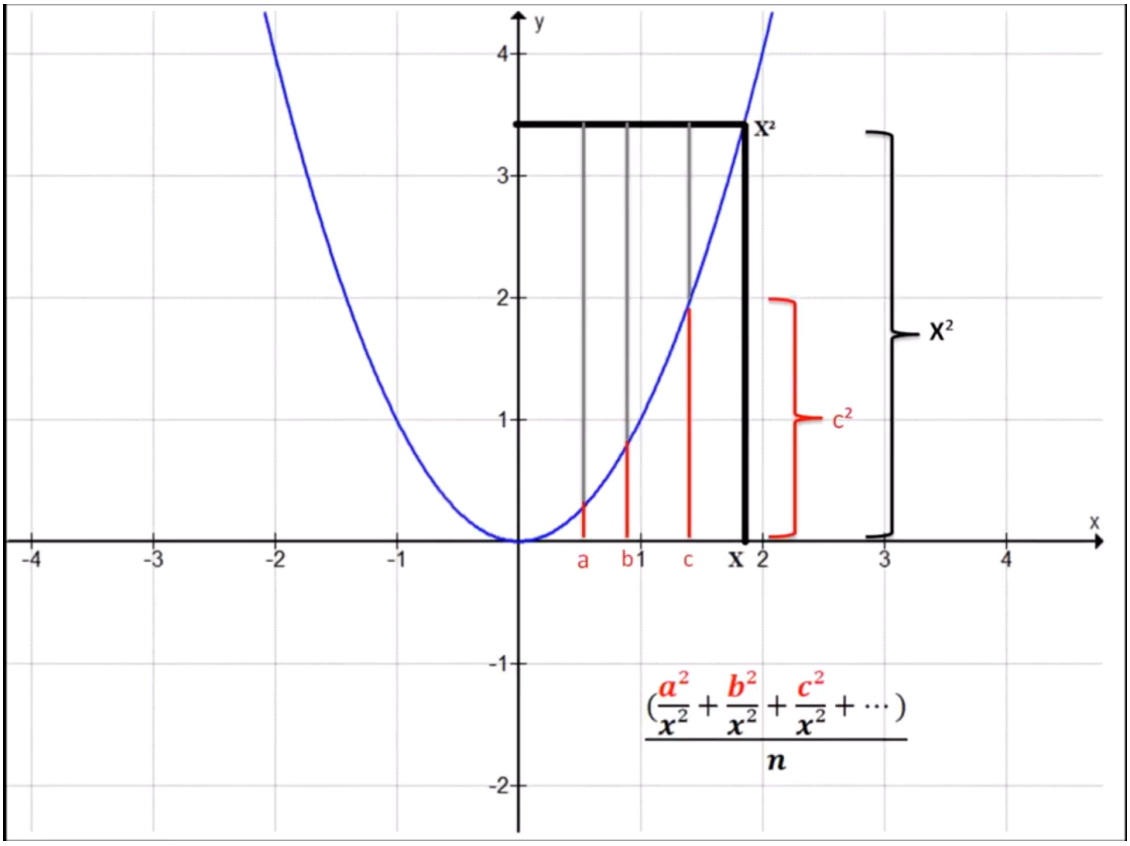
(This can also be proven by induction)



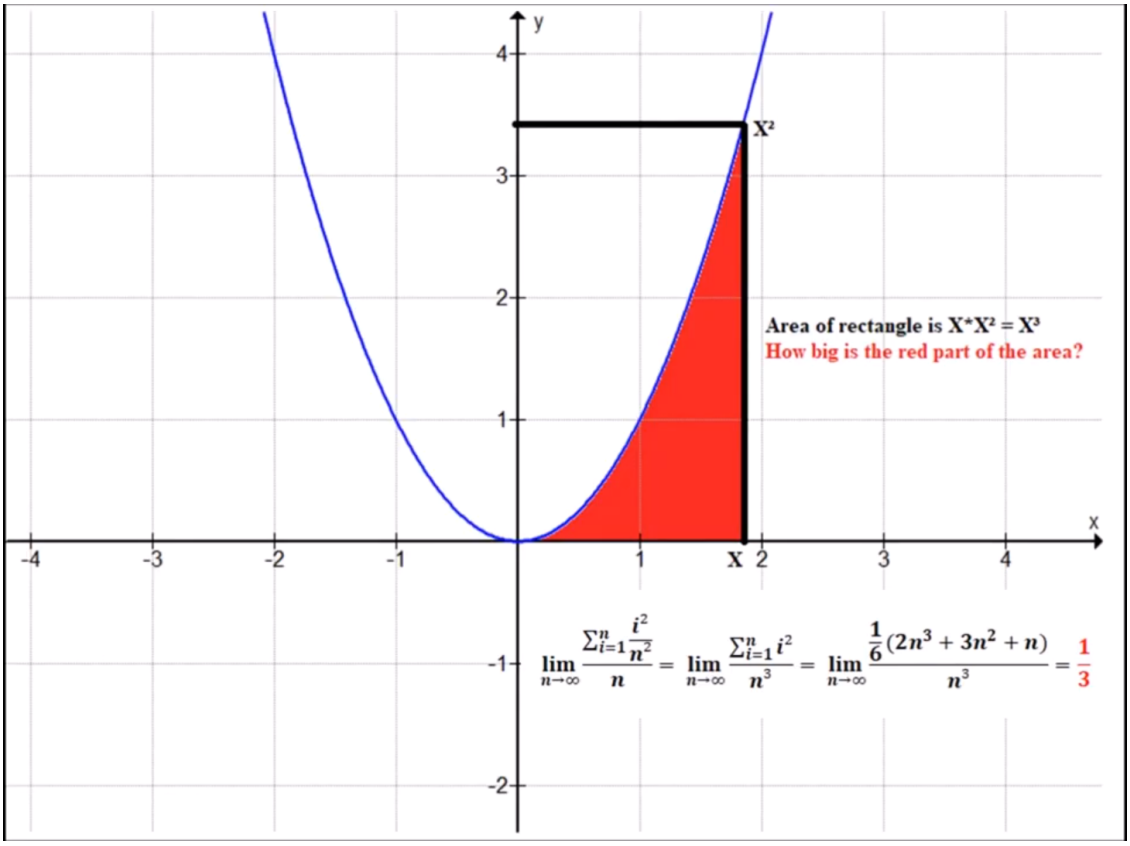
My integration method works by averaging the height-ratios. You get the ratio of red area to total rectangle.

Example:

$$f(x) = x^2$$



$Areaofrectangle = X^3$



Area is:

$$A = \frac{1}{3}x^3$$

Maybe, this Monte-Carlo Integration, or average, can also be interpreted as expectation value of position-ratios:

$$\left\langle \hat{R} \right\rangle = \int \psi^* \hat{R} \psi dx$$

So: Average = Expectation

Investigating the determinants of these operators, lead to a strange anomaly that needs further research.

[8]:
```
from numpy import *
import numpy as np
from math import *
import pandas as pd
from numpy.linalg import det
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

[9]:
```
def fak(n):
    a=1
    for i in range(n):
```

```
        a *= (i+1)
    return float(a)
```

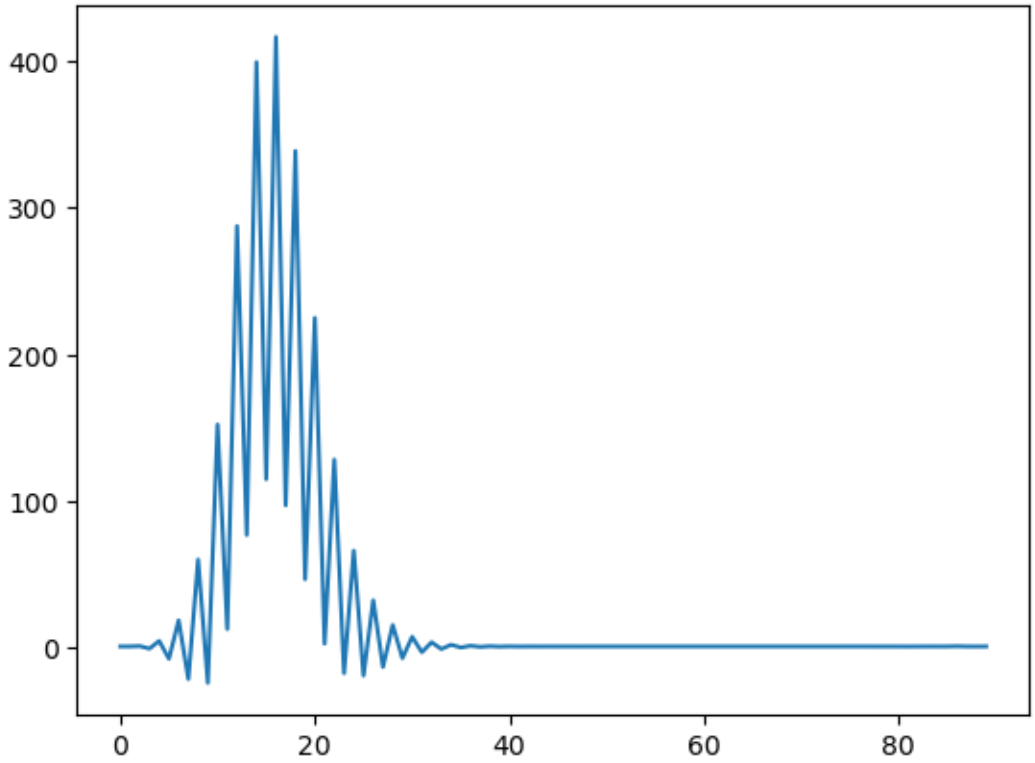
[10]:
```
def series(n, a):
    ident = eye(n,n)
    a_j = ident
    result = 0
    for j in range(n):
        a_j = matmul(a_j, a)
        result += a_j/fak(j+1)
    return result + ident
```

[11]:
```
def operator(n):
    a1 = np.zeros((n,n))
    a2 = np.zeros((n,n))
    for i in range(n-1):
        a1[i+1,i]=sqrt(i+1)
        a2[i,i+1]=sqrt(i+1)
    a = (a1 + a2)
    return a

def determ(n):
    a = operator(n)
    return det(series(n,a))

print(determ(20))
list = [determ(i) for i in range(90)]
print(list)
plt.plot(list)
```

```
225.02662696146044
[1.0, 1.0, 1.25, -0.5000000000000017, 4.790608723958337, -7.4160698784722365,
18.829702997289413, -21.24979920491955, 60.214256360818155, -23.835844916056196,
152.29703909998736, 12.845715193237847, 287.60654883775857, 76.95364538926766,
399.4010101025449, 115.00088640026607, 416.57456394151257, 97.19802632767268,
338.8478449509994, 46.79737543612804, 225.02662696146044, 2.874643870786153,
128.37608614737943, -17.262990440118376, 66.30059044521, -18.829374239077694,
32.48275887377732, -13.040626687289395, 15.624445715553213, -6.999516291195833,
7.574086322948935, -2.953876091245787, 3.839117304189545, -0.7611103574145045,
2.1675368893287144, 0.27774171373196943, 1.454675830631436, 0.7238027851773653,
1.1673761307280783, 0.9007574292415986, 1.0582789561478976, 0.9663254899324295,
1.0192288839733143, 0.9891723428675055, 1.0060239926211711, 0.9966897159343571,
1.001797777700402, 0.9990393851721844, 1.000511153087915, 0.999727157342325,
1.0001268109590133, 0.9999151690753648, 1.000033731144042, 0.9999889552379235,
1.0000327612159334, 0.9999850114368164, 1.000007827172898, 1.0001189311605594,
0.9999837866011797, 1.0000560341256535, 1.0001546792295748, 0.9998812059174452,
0.9999249251704063, 0.9999551119446249, 0.9996176731609557, 1.000344658117848,
```

```
0.9995226732675253, 1.0006561471398436, 1.0004911374284062, 1.000571714221554,
0.9989288354565281, 0.9981745498373039, 0.9954436874967737, 0.9947036493729546,
0.9976827631987242, 1.0010963060608236, 1.008178398436711, 0.990997442697231,
0.9827006046602136, 0.9966750120252672, 0.9778532879452503, 0.96520006914627,
0.9774837233824674, 1.0178678066008144, 0.9964523804662981, 1.0103746654358223,
1.1372144114773235, 1.0099213806137457, 0.9643230798501125, 1.0106887738160895]
```

[11]: `[<matplotlib.lines.Line2D at 0x7fd9dd17f950>]`



The determinant starts at '1', raises to some random maximum of ~400, and then goes back to '1'.

So the main question is: WTF does this anomaly mean????

[ ]: