

BeyondPPM_updatd

February 15, 2023

1 Installation and Import of packages, mount the drive

pip install dataframe_image, the package to display the table from dataframe

```
[1]: pip install dataframe-image
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-  
wheels/public/simple/
```

```
Collecting dataframe-image
```

```
  Downloading dataframe_image-0.1.5-py3-none-any.whl (6.6 MB)
```

```
        6.6/6.6 MB
```

```
37.0 MB/s eta 0:00:00
```

```
Requirement already satisfied: aiohttp in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (3.8.3)
```

```
Requirement already satisfied: matplotlib>=3.1 in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (3.2.2)
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (2.25.1)
```

```
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (4.6.3)
```

```
Requirement already satisfied: mistune in /usr/local/lib/python3.8/dist-packages  
(from dataframe-image) (0.8.4)
```

```
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (23.0)
```

```
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (5.6.1)
```

```
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.8/dist-  
packages (from dataframe-image) (1.3.5)
```

```
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in  
/usr/local/lib/python3.8/dist-packages (from matplotlib>=3.1->dataframe-image)  
(3.0.9)
```

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-  
packages (from matplotlib>=3.1->dataframe-image) (0.11.0)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in  
/usr/local/lib/python3.8/dist-packages (from matplotlib>=3.1->dataframe-image)  
(1.4.4)
```

```
Requirement already satisfied: python-dateutil>=2.1 in  
/usr/local/lib/python3.8/dist-packages (from matplotlib>=3.1->dataframe-image)
```

(2.8.2)

Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=3.1->dataframe-image) (1.21.6)

Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (1.5.0)

Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (0.4)

Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (6.0.0)

Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (0.7.1)

Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (2.6.1)

Requirement already satisfied: testpath in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (0.6.0)

Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (2.11.3)

Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (5.7.3)

Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (5.7.1)

Requirement already satisfied: jupyter-core in /usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe-image) (5.2.0)

Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=0.24->dataframe-image) (2022.7.1)

Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (1.3.3)

Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (22.2.0)

Requirement already satisfied: charset-normalizer<3.0,>=2.0 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (2.1.1)

Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (1.8.2)

Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (1.3.1)

Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (4.0.2)

Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe-image) (6.0.4)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->dataframe-image) (2022.12.7)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->dataframe-image) (1.24.3)

Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->dataframe-image) (4.0.0)

Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->dataframe-image) (2.10)

Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2>=2.4->nbconvert>=5->dataframe-image) (2.0.1)

Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.8/dist-packages (from nbformat>=4.4->nbconvert>=5->dataframe-image) (4.3.3)

Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.8/dist-packages (from nbformat>=4.4->nbconvert>=5->dataframe-image) (2.16.2)

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib>=3.1->dataframe-image) (1.15.0)

Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from bleach->nbconvert>=5->dataframe-image) (0.5.1)

Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.8/dist-packages (from jupyter-core->nbconvert>=5->dataframe-image) (3.0.0)

Requirement already satisfied: pyparsing!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert>=5->dataframe-image) (0.19.3)

Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert>=5->dataframe-image) (5.10.2)

Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.8/dist-packages (from importlib-resources>=1.4.0->jsonschema>=2.6->nbformat>=4.4->nbconvert>=5->dataframe-image) (3.12.1)

Installing collected packages: dataframe-image

Successfully installed dataframe-image-0.1.5

```
[2]: # Import the packages
import math
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pandas.plotting import table
import dataframe_image as dfi
```

```
[3]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

2 A scheme Beyond PPM

We temporarily call the scheme as Jonas. \ We have 4 photons 14 timeslots 1 symbol created by 14 timeslots

A1. The number of slots n with Jonas method. $n = \text{Sum}(\text{First } k \text{ integers in the series}) \setminus$

A2. The number of possible symbols, m , will be $m=k \setminus$

A3. The number of bits (the information content per symbol) will be $b=\log_2(m) \setminus$

2.1 Sequence to design the length of time slot / time bin

We can generate the desired number of sequence we want from the below code.

A010672 A B_2 sequence: $a(n)$ = least value such that sequence increases and pairwise sums of distinct elements are all distinct. \ Author: Dan Hoey

1. <https://oeis.org/A010672>
2. <https://oeis.org/A011185>
3. <https://oeis.org/A010672>

The below function generates a list of 100 sequences.

```
[4]: from itertools import islice

def agen(): # generator of terms

    aset, sset, k = set(), set(), 0

    while True:

        k += 1

        while any(k+an in sset for an in aset): k += 1

        yield k; sset.update(k+an for an in aset); aset.add(k)

a = list(islice(agen(), 100))
b = list(map(lambda v: v-1, a))
print(b)
```

```
[0, 1, 2, 4, 7, 12, 20, 29, 38, 52, 73, 94, 127, 151, 181, 211, 257, 315, 373,
412, 475, 530, 545, 607, 716, 797, 861, 964, 1059, 1160, 1306, 1385, 1434, 1555,
1721, 1833, 1933, 2057, 2260, 2496, 2698, 2873, 3060, 3196, 3331, 3628, 3711,
3867, 4139, 4446, 4639, 5021, 5064, 5322, 5613, 6003, 6273, 6493, 6641, 6979,
7275, 7587, 8017, 8373, 9071, 9167, 9760, 10105, 10489, 11109, 11374, 11516,
12101, 12330, 12867, 13426, 13923, 14535, 14911, 15469, 15904, 16136, 16900,
17041, 17822, 19421, 19933, 20288, 20996, 21491, 22065, 22612, 22659, 23724,
24399, 24969, 25360, 26071, 26680, 27601]
```

```
[5]: len(b)
```

```
[5]: 100
```

2.2 The Number of Time Bin (Time Slot)

Let us consider the following sequence: [0, 1, 2, 4, 7, 12, 20, 29, 38, 52, 73, 94, 127] as an example. Since we should have at least 1 photon in a time slot, we drop out the 1st term and start with the value 1.

The length of the nth time bin is the sum of the first nth sequence.

```
[6]: l = [1, 2, 4, 7, 12, 20, 29, 38, 52, 73, 94, 127, 151, 181, 211] # or l =   
      ↪ range(1, 21)  
      series = [sum(l[:i]) for i in range(1, len(l)+1)]  
      print (series)
```

```
[1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]
```

If we have 4 photons to encode into the superblock. We represent them into a sequence in a list as [1,2,4,7], which is encoded as follow: \ H H0 H000 V000000, which comprie of 14 time slots. We can find all the permutation, i.e. the number of ways to order them, which is $4! = 24$. It means that we have 24 possible way to represent a symbol in this time bin.

```
[7]: # A Python program to print all  
      # permutations using library function  
      from itertools import permutations  
  
      # Get all permutations of [1, 2, 3]  
      perm = permutations([1, 2, 4, 7])  
  
      lists = []  
      # Print the obtained permutations  
      for i in (perm):  
          #print (i)  
          lists.append(i)  
  
      lists
```

```
[7]: [(1, 2, 4, 7),  
      (1, 2, 7, 4),  
      (1, 4, 2, 7),  
      (1, 4, 7, 2),  
      (1, 7, 2, 4),  
      (1, 7, 4, 2),  
      (2, 1, 4, 7),  
      (2, 1, 7, 4),  
      (2, 4, 1, 7),  
      (2, 4, 7, 1),  
      (2, 7, 1, 4),  
      (2, 7, 4, 1),  
      (4, 1, 2, 7),
```

```
(4, 1, 7, 2),
(4, 2, 1, 7),
(4, 2, 7, 1),
(4, 7, 1, 2),
(4, 7, 2, 1),
(7, 1, 2, 4),
(7, 1, 4, 2),
(7, 2, 1, 4),
(7, 2, 4, 1),
(7, 4, 1, 2),
(7, 4, 2, 1)]
```

3 Number of ways to order the photons

$$n!$$

```
[8]: def ways(n):
      return math.factorial(n)
```

4 Number of Bits per Symbol

It is the information content per symbol

$$\log_2 n!$$

```
[9]: def bps(W):
      return math.log2(W)
```

5 Number of Bits per Photon

$\text{math.log2}(\text{Jonasways}) / n$

$$\log_2 \frac{n!}{n}$$

```
[10]: def bpph(b,n):
       return b/n
```

6 Number of Bits per Timeslot

$$\log_2 \frac{n!}{n} \times \frac{n}{T}$$

```
[11]: def bpt(B, n, T):  
       return B*n/T
```

```
[12]: bpph(bps(24), 4)*4/14
```

```
[12]: 0.3274973214800826
```

7 Putting all the functions together

```
[13]: series, series[4]
```

```
[13]: ([1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002], 26)
```

```
[14]: Jonasnumber = []  
JonasPermutation = []  
JonasBPS = []  
JonasBPP = []  
JonasBPT = []  
  
series = [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]  
  
df_Jonas = pd.DataFrame(columns=['|Number of Photon|',  
                                'Time Bins|',  
                                'Permutation|',  
                                'Bits/Symbol|',  
                                'Bits/Photon|',  
                                'Bits/Time Bin|'])  
  
for n in range(1,11):  
    Jonasways = math.factorial(n)           # A2. The number of possible symbols,  $m$ , will be  $m=k$  \\  
    JonasTimeBin = series[n-1]              # A1. The number of slots  $n$  with  $n = \text{Sum}(\text{First } k \text{ integers in the series})$  \\  
    Jonasbps = math.log2(Jonasways)         # A3. The number of bits (the information content per symbol) will be  $b = \log_2(m)$   
    Jonasbpp = math.log2(Jonasways) / n  
    Jonasbpt = Jonasbpp * n/series[n-1]  
  
    JonasTimeBin_SigFig = "{:.2e}".format(JonasTimeBin)  
    Jonasways_SigFig = "{:.2e}".format(Jonasways)  
    Jonasbps_round = round(Jonasbps, 2)  
    Jonasbpp_round = round(Jonasbpp, 2)  
    Jonasbpt_round = round(Jonasbpt, 2)
```

```
df_Jonas.loc[len(df_Jonas)] = [n, JonasTimeBin, Jonasways_SigFig, ↵
↵Jonasbps_round, Jonasbpp_round, Jonasbpt_round]
```

```
[15]: df_Jonas # 4th one data is correct
```

```
[15]: |Number of Photon| Time Bins| Permutation| Bits/Symbol| Bits/Photon| \
0          1          1      1.00e+00      0.00      0.00
1          2          3      2.00e+00      1.00      0.50
2          3          7      6.00e+00      2.58      0.86
3          4         14      2.40e+01      4.58      1.15
4          5         26      1.20e+02      6.91      1.38
5          6         46      7.20e+02      9.49      1.58
6          7         75      5.04e+03     12.30      1.76
7          8        113      4.03e+04     15.30      1.91
8          9        165      3.63e+05     18.47      2.05
9         10        238      3.63e+06     21.79      2.18
```

```
Bits/Time Bin|
0          0.00
1          0.33
2          0.37
3          0.33
4          0.27
5          0.21
6          0.16
7          0.14
8          0.11
9          0.09
```

```
[16]: df_Jonas[df_Jonas.columns[0]]
```

```
[16]: 0      1
      1      2
      2      3
      3      4
      4      5
      5      6
      6      7
      7      8
      8      9
      9     10
      Name: |Number of Photon|, dtype: object
```

```
[17]: df_Jonas[df_Jonas.columns[1]]
```

```
[17]: 0      1
      1      3
```



```

2      7
3     14
4     26
5     46
6     75
7    113
8    165
9    238
Name: Time Bins|, dtype: object

```

```
[18]: df_Jonas[df_Jonas.columns[2]]
```

```

[18]: 0      1.00e+00
      1      2.00e+00
      2      6.00e+00
      3      2.40e+01
      4      1.20e+02
      5      7.20e+02
      6      5.04e+03
      7      4.03e+04
      8      3.63e+05
      9      3.63e+06
Name: Permutation|, dtype: object

```

```
[19]: df_Jonas[df_Jonas.columns[3]]
```

```

[19]: 0      0.00
      1      1.00
      2      2.58
      3      4.58
      4      6.91
      5      9.49
      6     12.30
      7     15.30
      8     18.47
      9     21.79
Name: Bits/Symbol|, dtype: float64

```

```
[20]: df_Jonas[df_Jonas.columns[4]]
```

```

[20]: 0      0.00
      1      0.50
      2      0.86
      3      1.15
      4      1.38
      5      1.58
      6      1.76

```

```

7    1.91
8    2.05
9    2.18
Name: Bits/Photon|, dtype: float64

```

```
[21]: df_Jonas[df_Jonas.columns[5]]
```

```

[21]: 0    0.00
      1    0.33
      2    0.37
      3    0.33
      4    0.27
      5    0.21
      6    0.16
      7    0.14
      8    0.11
      9    0.09
Name: Bits/Time Bin|, dtype: float64

```

```

[22]: #df_Jonas = df_Jonas.style.background_gradient() #adding a gradient based on
      ↪ values in cell
dfi.export(
    df_Jonas,
    "Jonas_Table.png",
    table_conversion="matplotlib"
)

```

```
[23]: df_Jonas.columns[1]
```

```
[23]: 'Time Bins|'
```

8 Plot Graph

```

[24]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↪ columns[2]],linewidth=3,zorder=1, label = "bits")
axis[0, 0].set_title('Permutation(in Log Scale Vs Number of Photon)', fontsize=
    ↪ 16)
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

```

```

axis[0, 1].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↪columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Symbol Vs Number of Photon', fontsize = 16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

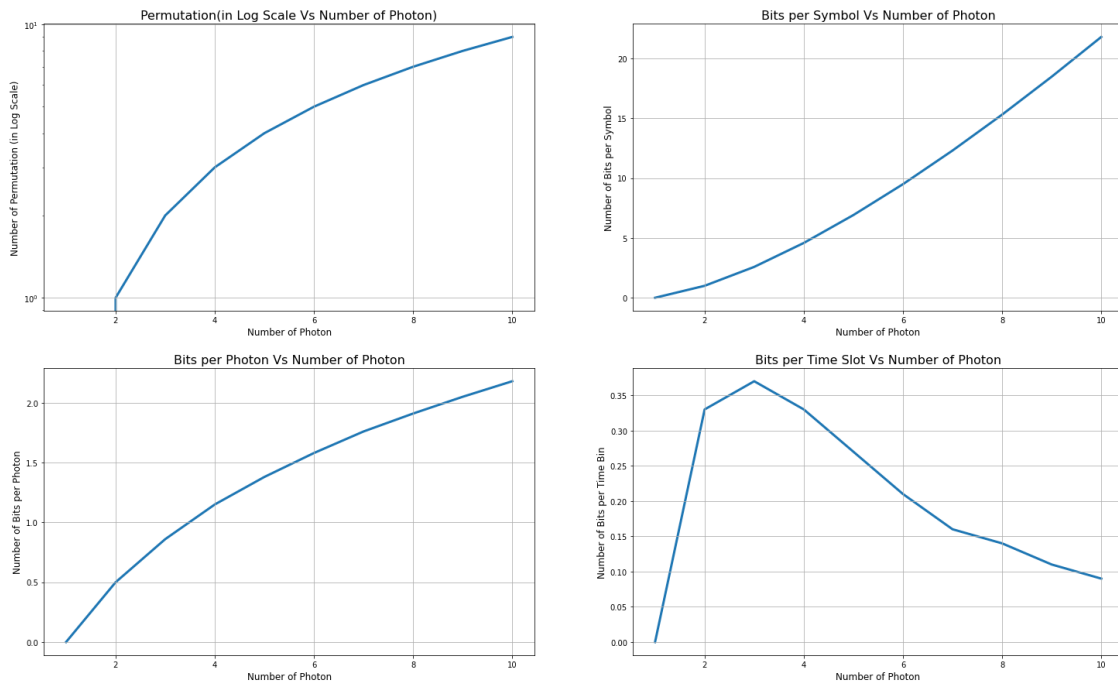
axis[1, 0].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↪columns[4]],linewidth=3,zorder=1, label = "bits")

axis[1, 0].set_title('Bits per Photon Vs Number of Photon', fontsize = 16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Time Slot Vs Number of Photon', fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('Jonas_Photon_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



```

[25]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↳columns[2]],linewidth=3,zorder=1, label = "bits")
axis[0, 0].set_title('Permutation(in Log Scale Vs Number of Time Bin)',
    ↳fontsize = 16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↳columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Symbol Vs Number of Time Bin', fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

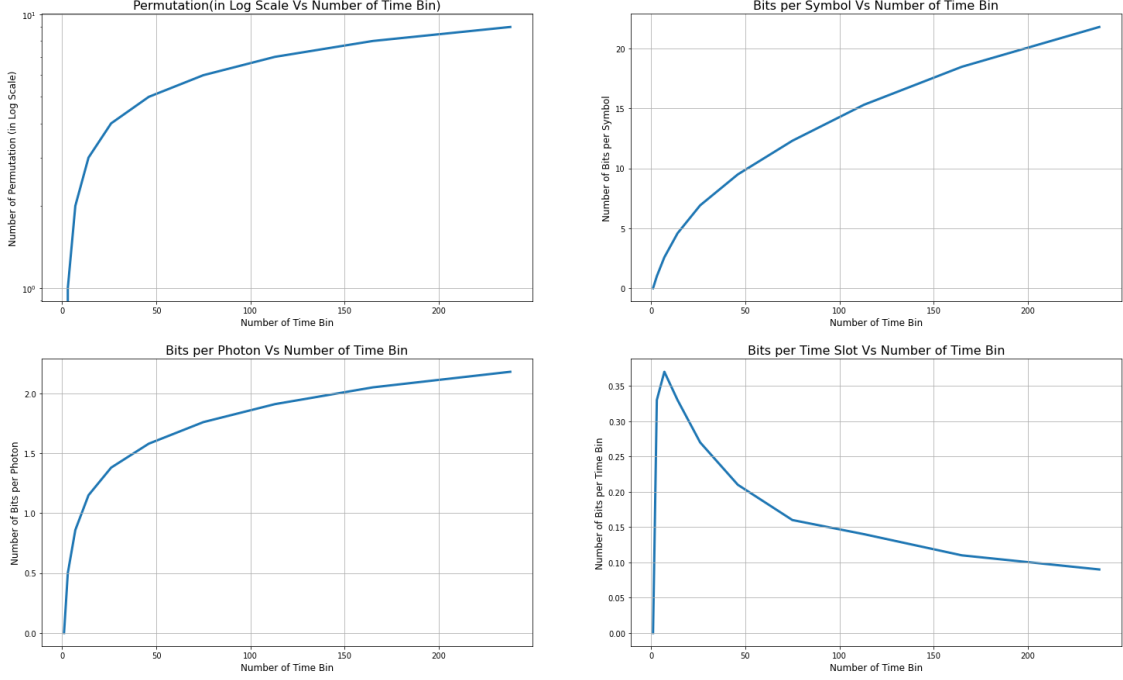
axis[1, 0].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↳columns[4]],linewidth=3,zorder=1, label = "bits")

axis[1, 0].set_title('Bits per Photon Vs Number of Time Bin', fontsize = 16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Time Slot Vs Number of Time Bin', fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('Jona_Time_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



9 General

The number of symbols m in general method will be the number of ways k photons can be placed in n bins, i.e.

$$m = \binom{n}{k}$$

If we have 4 photons, 14 timeslots, then there will be 1,0001 ways to order them by binomial where order does not matter.

The number of bits (the information content per symbol) will be

$$b = \log_2 m$$

However, if we consider the repetition are not allowed, then we use Permutations = $r!$ x Combinations

B1. Compare with “general” method using same number of slots, n , and same number of photons, k \

B2. The number of symbols m in general method will be the number of ways k photons can be placed in n bins, i.e. $m = \binom{n}{k}$ \

B3. The number of bits (the information content per symbol) will be $b = \log_2(m)$

9.0.1 Combination

```
[26]: def combination(n,r):  
        return math.factorial(n) / (math.factorial(n-r)*math.factorial(r))  
  
combination(14,4)
```

[26]: 1001.0

```
[27]: Generalnumber_c = []  
GeneralPermutation_c = []  
GeneralBPS_c = []  
GeneralBPP_c = []  
GeneralBPT_c = []  
  
series = [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]  
  
df_General_c = pd.DataFrame(columns=['|Number of Photon|',  
                                     '|Time Bins|',  
                                     '|Permutation|',  
                                     '|Bits/Symbol|',  
                                     '|Bits/Photon|',  
                                     '|Bits/Time Bin|'])  
  
for n in range(1,11):  
    Generalways_c = combination(series[n-1],n)    # B2. The number of symbols m  
    ↪ in general method will be the number of ways k photons can be placed in n  
    ↪ bins, i.e.  $m = \binom{n}{k}$  \\  
    GeneralTimeBin = series[n-1]                # B1. Compare with "general"  
    ↪ method using same number of slots, n, and same number of photons, k \\  
    Generalbps_c = math.log2(Generalways_c)      # B3. The number of bits (the  
    ↪ information content per symbol) will be  $b = \log_2(m)$   
    Generalbpp_c = math.log2(Generalways_c) / n  
    Generalbpt_c = Generalbpp_c * n/series[n-1]  
  
    GeneralTimeBin_SigFig_c = "{:.2e}".format(GeneralTimeBin)  
    Generalways_SigFig_c = "{:.2e}".format(Generalways_c)  
    Generalbps_round_c = round(Generalbps_c, 2)  
    Generalbpp_round_c = round(Generalbpp_c, 2)  
    Generalbpt_round_c = round(Generalbpt_c, 2)  
  
    df_General_c.loc[len(df_General_c)] = [n, GeneralTimeBin,  
    ↪ Generalways_SigFig_c, Generalbps_round_c, Generalbpp_round_c,  
    ↪ Generalbpt_round_c]
```

```
#round(answer, 2)
```

```
[28]: # The first is zero because, the first item in the time bin is 1, which give
      ↪ the log based 2 to be 0.
      math.log2(combination(1,1))
```

```
[28]: 0.0
```

```
[29]: df_General_c # the 4 data is correct
```

```
[29]: |Number of Photon| Time Bins| Permutation| Bits/Symbol| Bits/Photon| \
0          1          1      1.00e+00      0.00      0.00
1          2          3      3.00e+00      1.58      0.79
2          3          7      3.50e+01      5.13      1.71
3          4         14      1.00e+03      9.97      2.49
4          5         26      6.58e+04     16.01      3.20
5          6         46      9.37e+06     23.16      3.86
6          7         75      1.98e+09     30.89      4.41
7          8        113      5.12e+11     38.90      4.86
8          9        165      2.00e+14     47.51      5.28
9         10        238      1.33e+17     56.88      5.69
```

```
      Bits/Time Bin|
0          0.00
1          0.53
2          0.73
3          0.71
4          0.62
5          0.50
6          0.41
7          0.34
8          0.29
9          0.24
```

```
[30]: dfi.export(
      df_General_c,
      "General_c_Table.png",
      table_conversion="matplotlib"
      )
```

```
[31]: figure, axis = plt.subplots(2,2,figsize=(25,15))

      axis[0, 0].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
      ↪ columns[2]],linewidth=3,zorder=1, label = "bits")
      axis[0, 0].set_title('Permutation(in Log Scale Vs Number of Photon)', fontsize_
      ↪ = 16)
      axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
```

```

axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

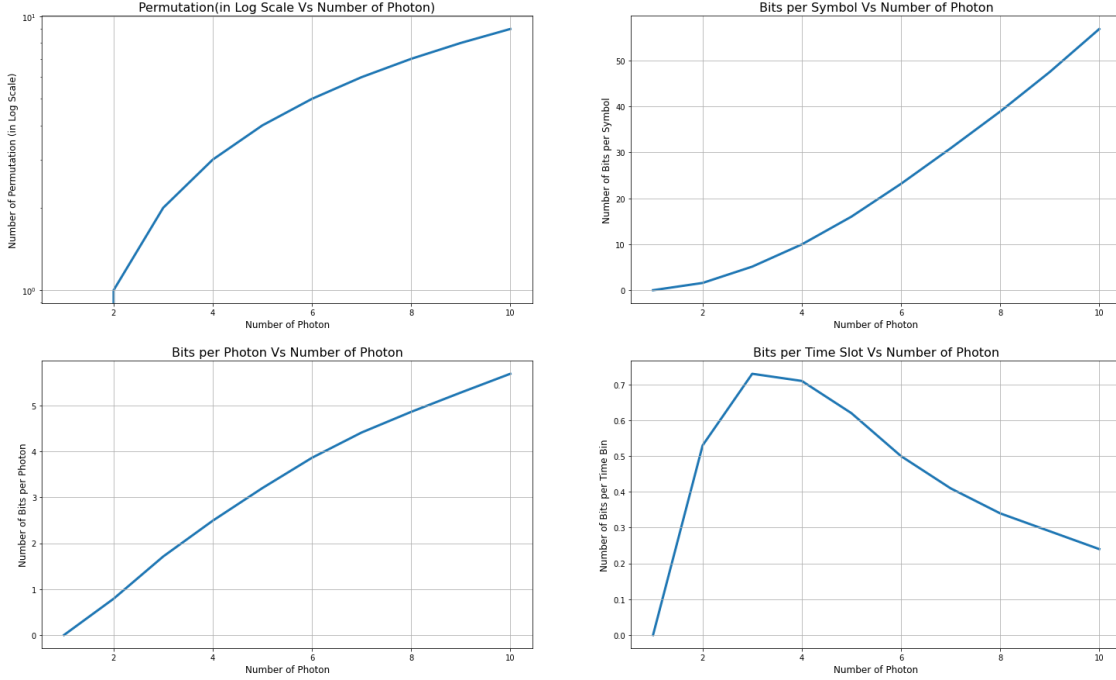
axis[0, 1].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
    ↪columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Symbol Vs Number of Photon', fontsize = 16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
    ↪columns[4]],linewidth=3,zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Photon Vs Number of Photon', fontsize = 16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Time Slot Vs Number of Photon', fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
#plt.xticks(x)
plt.savefig('General_c_Photon_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```

```
[32]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↪columns[2]],linewidth=3,zorder=1, label = "bits")
axis[0, 0].set_title('Permutation(in Log Scale Vs Number of Time Bin)',
    ↪fontsize = 16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↪columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Symbol Vs Number of Time Bin', fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↪columns[4]],linewidth=3,zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Photon Vs Number of Time Bin', fontsize = 16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
```

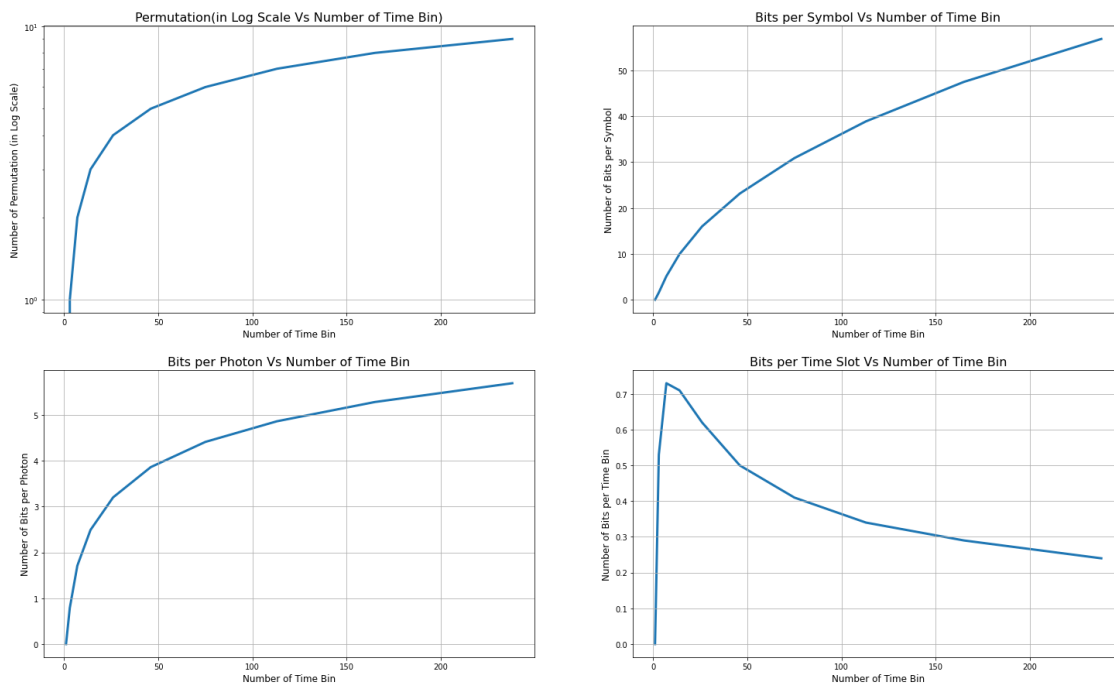
```

axis[1, 0].grid(True)

axis[1, 1].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Time Slot Vs Number of Time Bin', fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
#plt.xticks(x)
plt.savefig('General_c_TIme_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



9.0.2 Permutation

```

[33]: # A Python program to print all permutations of given length
from itertools import permutations

# Get all permutations of length 2
#perm = permutations([1, 2, 3], 2)

# Print the obtained permutations
#for i in list(perm):
#    print (i)

```

```
[34]: def permutation(n,r):
        return math.factorial(n)/math.factorial(n-r)
```

```
[35]: Generalnumber_p = []
GeneralPermutation_p = []
GeneralBPS_p = []
GeneralBPP_p = []
GeneralBPT_p = []

series = [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]

df_General_p = pd.DataFrame(columns=['|Number of Photon|',
                                     'Time Bins|',
                                     'Permutation|',
                                     'Bits/Symbol|',
                                     'Bits/Photon|',
                                     'Bits/Time Bin|'])

for n in range(1,11):
    Generalways_p = permutation(series[n-1],n)
    GeneralTimeBin = series[n-1]
    Generalbps_p = math.log2(Generalways_p)
    Generalbpp_p = math.log2(Generalways_p) / n
    Generalbpt_p = Generalbpp_p * n/series[n-1]

    GeneralTimeBin_SigFig_p = "{:.2e}".format(GeneralTimeBin)
    Generalways_SigFig_p = "{:.2e}".format(Generalways_p)
    Generalbps_round_p = round(Generalbps_p, 2)
    Generalbpp_round_p = round(Generalbpp_p, 2)
    Generalbpt_round_p = round(Generalbpt_p, 2)

    df_General_p.loc[len(df_General_p)] = [n, GeneralTimeBin_SigFig_p,
    ↪Generalways_SigFig_p, Generalbps_round_p, Generalbpp_round_p,
    ↪Generalbpt_round_p]

    #round(answer, 2)
```

```
[36]: df_General_p # the 4 data is not correct
```

```
[36]: |Number of Photon| Time Bins| Permutation| Bits/Symbol| Bits/Photon| \
0          1    1.00e+00    1.00e+00          0.00          0.00
1          2    3.00e+00    6.00e+00          2.58          1.29
2          3    7.00e+00    2.10e+02          7.71          2.57
3          4    1.40e+01    2.40e+04         14.55          3.64
```

4	5	2.60e+01	7.89e+06	22.91	4.58
5	6	4.60e+01	6.74e+09	32.65	5.44
6	7	7.50e+01	1.00e+13	43.19	6.17
7	8	1.13e+02	2.06e+16	54.20	6.77
8	9	1.65e+02	7.26e+19	65.98	7.33
9	10	2.38e+02	4.81e+23	78.67	7.87

	Bits/Time Bin
0	0.00
1	0.86
2	1.10
3	1.04
4	0.88
5	0.71
6	0.58
7	0.48
8	0.40
9	0.33

```
[37]: dfi.export(
        df_General_p,
        "General_p_Table.png",
        table_conversion="matplotlib"
    )
```

```
[38]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_General_p[df_General_p.columns[0]],df_General_p[df_General_p.
    ↪columns[1]],linewidth=3,zorder=1, label = "bits")
axis[0, 0].set_title('Number of Permutation(in Log Scale Vs Number of Photon)',
    ↪fontsize = 16)
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_General_p[df_General_p.columns[0]],df_General_p[df_General_p.
    ↪columns[2]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Number of Bits per Symbol Vs Number of Photon',
    ↪fontsize = 16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)
```

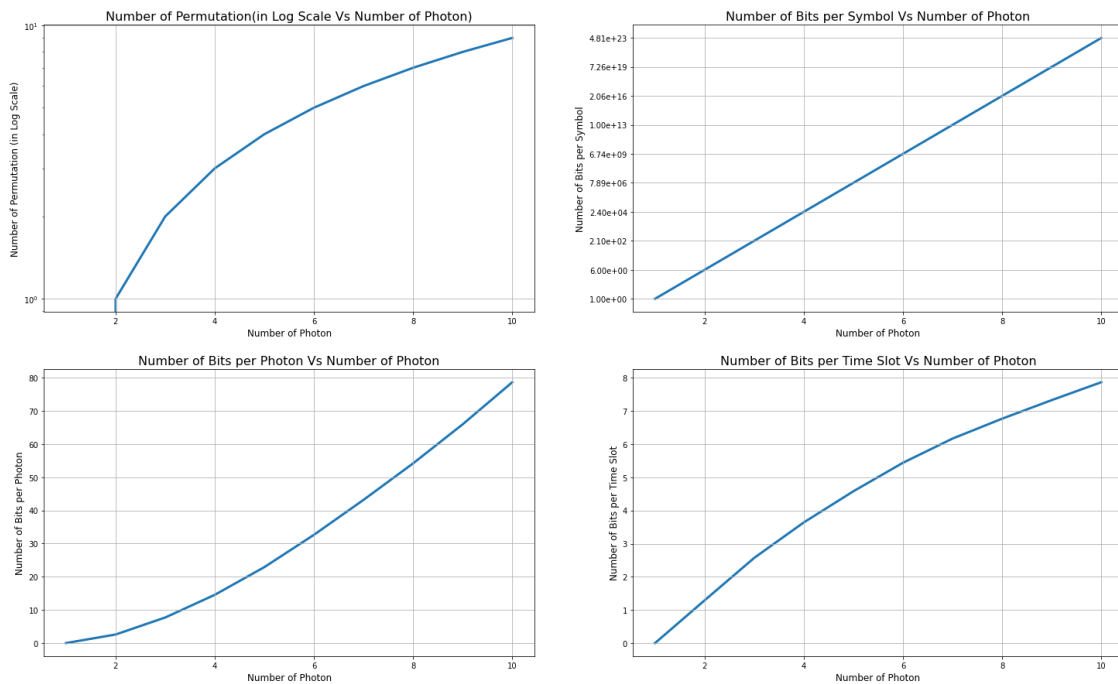
```

axis[1, 0].plot(df_General_p[df_General_p.columns[0]],df_General_p[df_General_p.
    ↪columns[3]],linewidth=3,zorder=1, label = "bits")
axis[1, 0].set_title('Number of Bits per Photon Vs Number of Photon', fontsize_
    ↪= 16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_General_p[df_General_p.columns[0]],df_General_p[df_General_p.
    ↪columns[4]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Number of Bits per Time Slot Vs Number of Photon',_
    ↪fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Slot', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('General_p_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



10 PPM

We have 1 photon 14 timeslots 14 ways to order them

C1. Compare with PPM method using same number of slots, n. It will only use k=1 photons per

symbol \

C2. The number of symbols m in PPM will be the number of ways 1 photon can be placed in n bins, i.e. $m=n$ \

C3. The number of bits (the information content per symbol) will be $b=\log_2(m)$ \

```
[39]: PPMnumber = []
PPMPermutation = []
PPMBPS = []
PPMBPP = []
PPMBPT = []

series = [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]

df_PPM = pd.DataFrame(columns=['|Number of Photon|',
                                '|Time Bins|',
                                '|Permutation|',
                                '|Bits/Symbol|',
                                '|Bits/Photon|',
                                '|Bits/Time Bin|'
                                ])

for n in range(1,11):
    PPMways = series[n-1]
    #PPMways = n # C2 The number of symbols m in PPM will
    #be the number of ways 1 photon can be placed in n bins, i.e. m=n \
    PPMTimeBin = series[n-1] # C1 Compare with PPM method using same
    #number of slots, n. It will only use k=1 photons per symbol \
    PPMbps = math.log2(PPMTimeBin) # C3 The number of bits (the
    #information content per symbol) will be b=log2(m)
    PPMbpp = math.log2(PPMTimeBin) / 1 # C1 Compare with PPM method using
    #same number of slots, n. It will only use k=1 photons per symbol \
    PPMbpt = PPMbpp * 1/series[n-1]

    PPMTimeBin_SigFig = "{:.2e}".format(PPMTimeBin)
    #PPMways_SigFig = "{:.2e}".format(PPMways)
    PPMways_SigFig = PPMways
    PPMbps_round = round(PPMbps, 2)
    PPMbpp_round = round(PPMbpp, 2)
    PPMbpt_round = round(PPMbpt, 2)

    df_PPM.loc[len(df_PPM)] = [1, PPMTimeBin, PPMways_SigFig, PPMbps_round ,
    #PPMbpp_round, PPMbpt_round]

    #round(answer, 2)
```

```
[40]: df_PPM.style # the 4th data is not correct
```

```
[40]: <pandas.io.formats.style.Styler at 0x7f5a45a3c2b0>
```

```
[41]: math.log2(15)
```

```
[41]: 3.9068905956085187
```

```
[42]: df_PPM.columns[0]
```

```
[42]: '|Number of Photon|'
```

```
[43]: dfi.export(  
    df_PPM,  
    "PPM_Table.png",  
    table_conversion="matplotlib"  
)
```

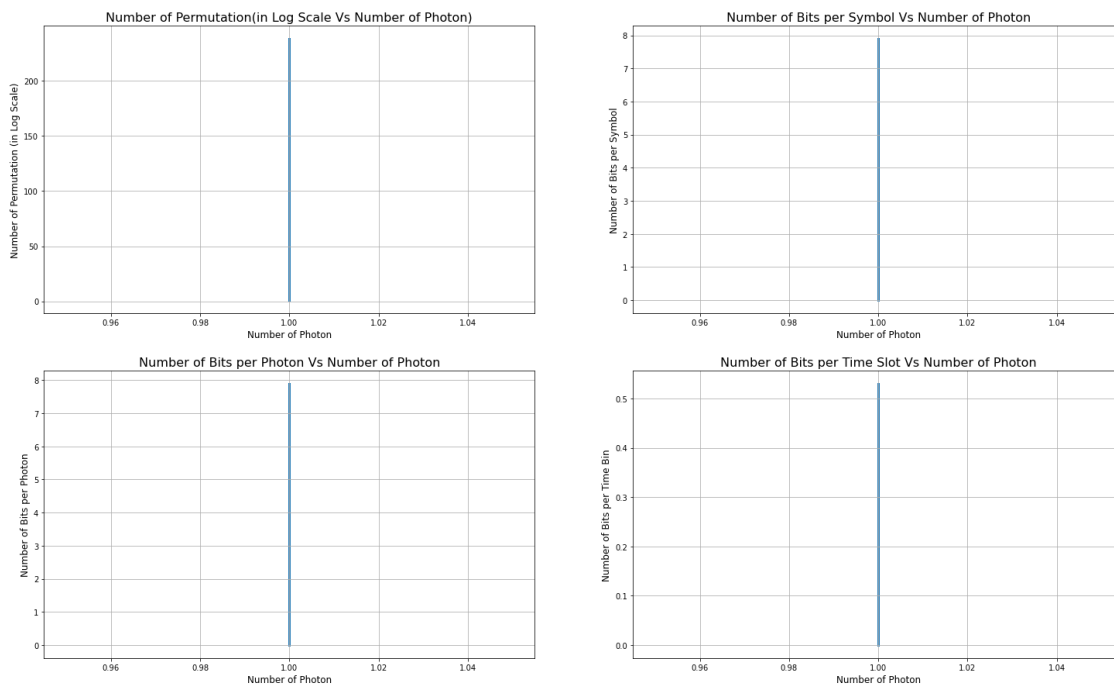
```
[44]: figure, axis = plt.subplots(2,2,figsize=(25,15))  
  
axis[0, 0].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.  
    ↳columns[2]],linewidth=3,zorder=1, label = "bits")  
axis[0, 0].set_title('Number of Permutation(in Log Scale Vs Number of Photon)',  
    ↳fontsize = 16)  
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)  
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)  
#axis[0, 0].set_yscale('log')  
#axis[0, 0].set_xlim([0, 21])  
axis[0, 0].grid(True)  
  
axis[0, 1].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.  
    ↳columns[3]],linewidth=3,zorder=1, label = "bits")  
axis[0, 1].set_title('Number of Bits per Symbol Vs Number of Photon', fontsize=  
    ↳= 16)  
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)  
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)  
axis[0, 1].grid(True)  
  
axis[1, 0].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.  
    ↳columns[4]],linewidth=3,zorder=1, label = "bits")  
axis[1, 0].set_title('Number of Bits per Photon Vs Number of Photon', fontsize=  
    ↳= 16)  
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)  
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)  
axis[1, 0].grid(True)
```

```

axis[1, 1].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Number of Bits per Time Slot Vs Number of Photon',
    ↳fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('PPM_Photon_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



```

[45]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↳columns[2]],linewidth=3,zorder=1, label = "bits")
axis[0, 0].set_title('Number of Permutation(in Log Scale Vs Number of Time_
    ↳Bin', fontsize = 16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

```



```

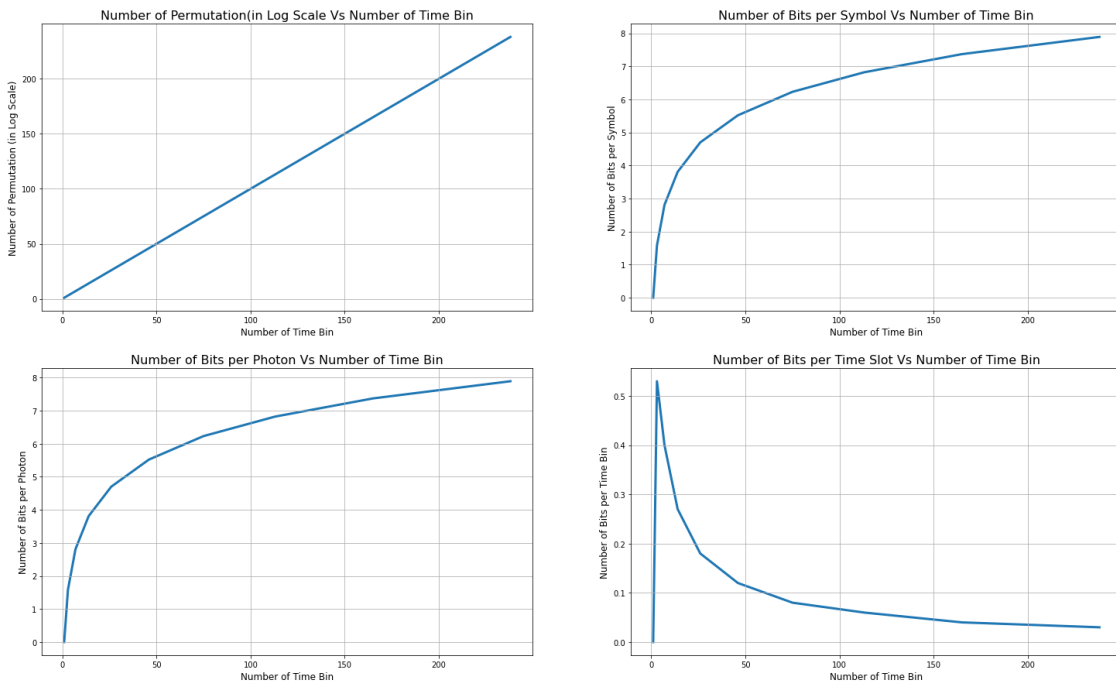
axis[0, 1].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↳columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Number of Bits per Symbol Vs Number of Time Bin',
    ↳fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↳columns[4]],linewidth=3,zorder=1, label = "bits")
axis[1, 0].set_title('Number of Bits per Photon Vs Number of Time Bin',
    ↳fontsize = 16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Number of Bits per Time Slot Vs Number of Time Bin',
    ↳fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('PPM_TimeBin_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



11 OOK

D1. Compare with OOK method using same number of slots, n . It will in average use $k=n/2$ photons per symbol \

D2. The number of symbols m in PPM will be $m=2^n$ \

D3. The number of bits (the information content per symbol) will be $b=\log_2(m)=n$ \

```
[46]: OOKnumber = []
      OOKPermutation = []
      OOKBPS = []
      OOKBPP = []
      OOKBPT = []

      series = [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]

      df_OOK = pd.DataFrame(columns=['|Number of Photon|',
                                     'Time Bins|',
                                     'Permutation|',
                                     'Bits/Symbol|',
                                     'Bits/Photon|',
                                     'Bits/Time Bin|']
                             )

      for n in range(1,11):
          #OOKways = (2)**series[n-1]
          OOKTimeBin = series[n-1] # D1. Compare with OOK
          ↪method using same number of slots, n. It will in average use k=n/2 photons
          ↪per symbol
          OOKWays = (2)**series[n-1] # D2. The number of
          ↪symbols m in PPM will be m=2^n, n is the number of time bin
          OOKbps = math.log2(OOKWays) # D3. The number of bits
          ↪(the information content per symbol) will be b=log2(m)=n, n is the number of
          ↪time bin
          #OOKbpp = (math.log2(OOKWays) * series[n-1] / n
          OOKbpp = 2
          OOKbpt = OOKbpp * n / series[n-1]

          OOKTimeBin_SigFig = "{:.2e}".format(OOKTimeBin)
          OOKWays_SigFig = "{:.2e}".format(OOKWays)
          #OOKWays_SigFig = round(OOKWays, 2)
          OOKbps_round = round(OOKbps, 2)
          OOKbpp_round = round(OOKbpp, 2)
          OOKbpt_round = round(OOKbpt, 2)
```

```
df_00K.loc[len(df_00K)] = [00KTimeBin/2, 00KTimeBin, 00K Ways_SigFig,
↪00Kbps_round, 00Kbpp_round, 00Kbpt_round]
```

```
[47]: #df_00K.style
```

```
[48]: df_00K # not sure if it is correct
```

```
[48]:
```

	Number of Photon	Time Bins	Permutation	Bits/Symbol	Bits/Photon	\
0	0.5	1	2.00e+00	1.0	2	
1	1.5	3	8.00e+00	3.0	2	
2	3.5	7	1.28e+02	7.0	2	
3	7.0	14	1.64e+04	14.0	2	
4	13.0	26	6.71e+07	26.0	2	
5	23.0	46	7.04e+13	46.0	2	
6	37.5	75	3.78e+22	75.0	2	
7	56.5	113	1.04e+34	113.0	2	
8	82.5	165	4.68e+49	165.0	2	
9	119.0	238	4.42e+71	238.0	2	

	Bits/Time Bin
0	2.00
1	1.33
2	0.86
3	0.57
4	0.38
5	0.26
6	0.19
7	0.14
8	0.11
9	0.08

```
[49]: dfi.export(
        df_00K,
        "00K_Table.png",
        table_conversion="matplotlib"
    )
```

```
[50]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_00K[df_00K.columns[0]],df_00K[df_00K.
↪columns[2]],linewidth=3,zorder=1, label = "bits")

axis[0, 0].set_title('Number of Permutation(in Log Scale Vs Number of Photon)',
↪fontsize = 16)
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
```

```

axis[0, 0].set_yscale('log')
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

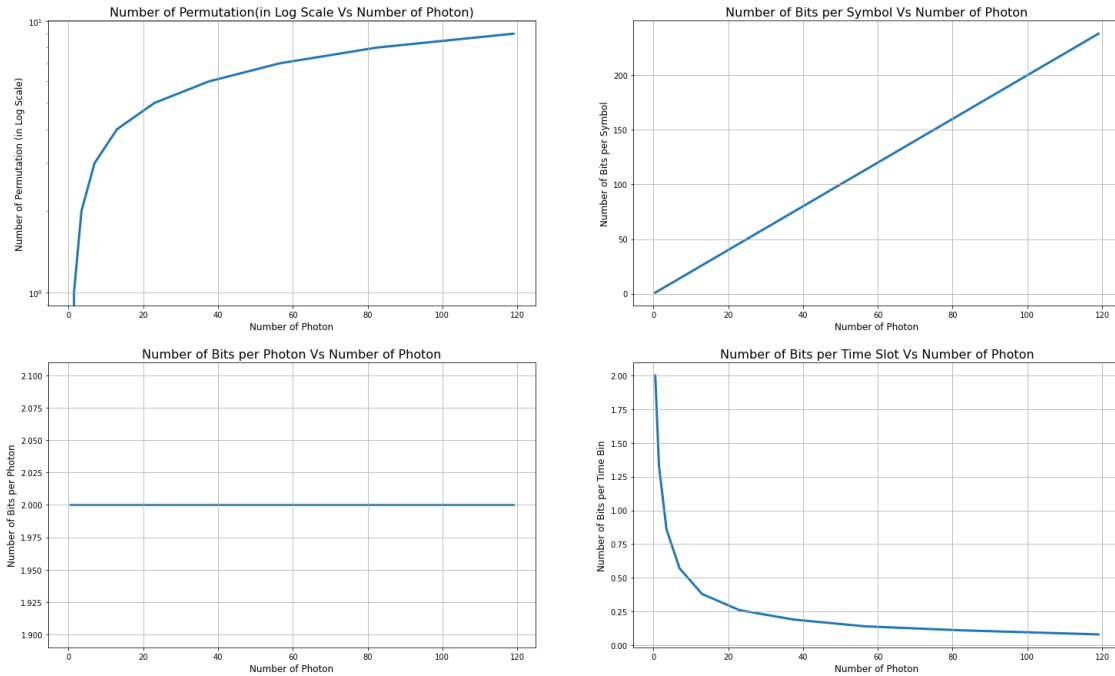
axis[0, 1].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
    ↳columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Number of Bits per Symbol Vs Number of Photon', fontsize=
    ↳16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
    ↳columns[4]],linewidth=3,zorder=1, label = "bits")
axis[1, 0].set_title('Number of Bits per Photon Vs Number of Photon', fontsize=
    ↳16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Number of Bits per Time Slot Vs Number of Photon',
    ↳fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('OOK_Photon_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



```
[51]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
    ↳columns[2]],linewidth=3,zorder=1, label = "bits")

axis[0, 0].set_title('Number of Permutation(in Log Scale Vs Number of Time_
    ↳Bin', fontsize = 16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
axis[0, 0].set_yscale('log')
#axis[0, 0].set_yscale('log')
#axis[0, 0].set_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
    ↳columns[3]],linewidth=3,zorder=1, label = "bits")
axis[0, 1].set_title('Number of Bits per Symbol Vs Number of Time Bin',
    ↳fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Symbol', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
    ↳columns[4]],linewidth=3,zorder=1, label = "bits")
```

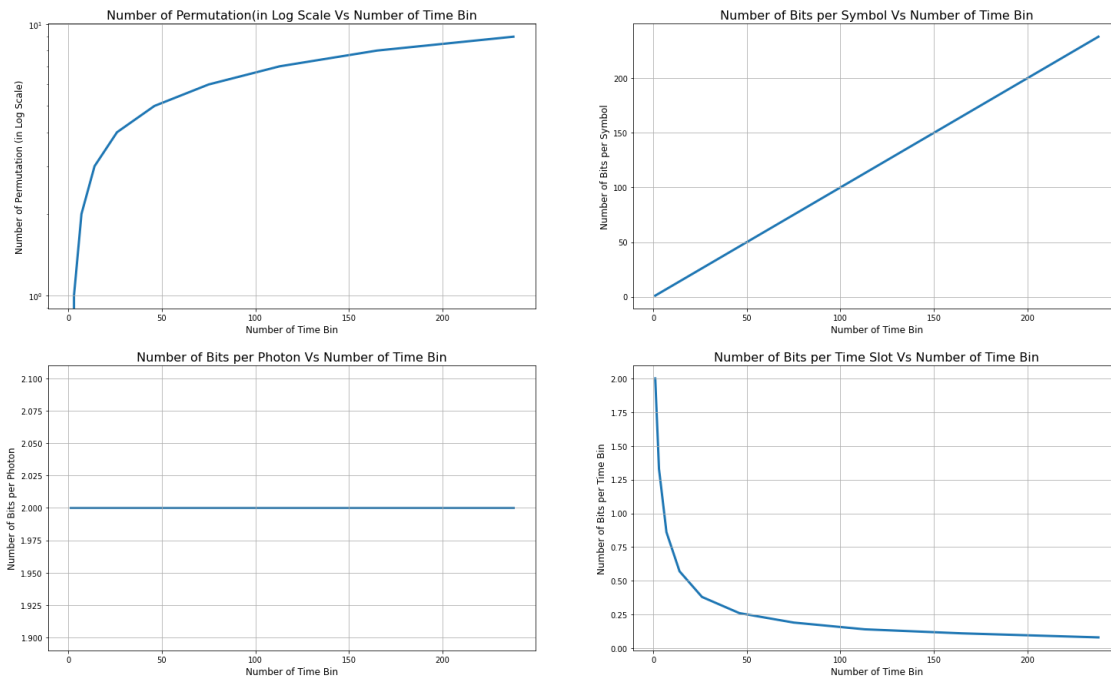
```

axis[1, 0].set_title('Number of Bits per Photon Vs Number of Time Bin',
    ↳fontsize = 16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Number of Bits per Time Slot Vs Number of Time Bin',
    ↳fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('OOK_TimeBin_Plot.png', dpi=450, bbox_inches='tight')
plt.show()

```



12 Bits/Photon over 4 Schemes

```

[52]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↳columns[4]],linewidth=3, zorder=1, label = "bits")

```

```

axis[0, 0].set_title('Bits per Photon Vs Number of Photon (Jonas)' , fontsize = 16)
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 0].grid(True)

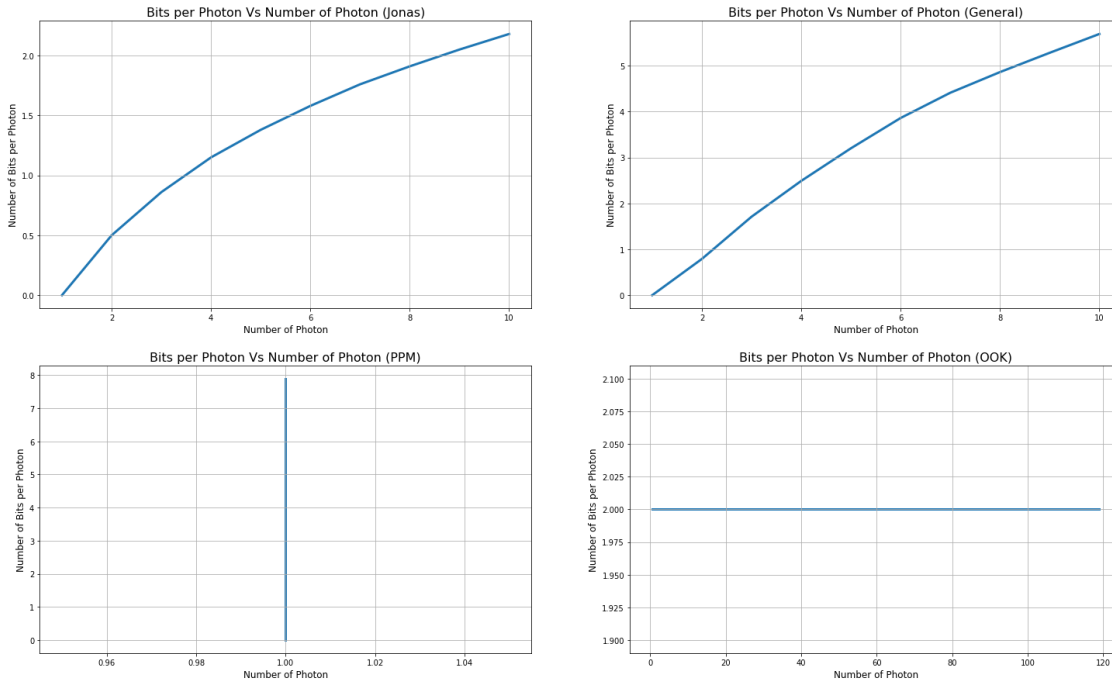
axis[0, 1].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
    columns[4]],linewidth=3, zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Photon Vs Number of Photon (General)' , fontsize = 16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
    columns[4]],linewidth=3, zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Photon Vs Number of Photon (PPM)' , fontsize = 16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
    columns[4]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Photon Vs Number of Photon (OOK)' , fontsize = 16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('4SchemesBPP_Photon.png', dpi=450, bbox_inches='tight')
plt.show()

```



```
[53]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↳columns[4]],linewidth=3, zorder=1, label = "bits")
axis[0, 0].set_title('Bits per Photon Vs Number of Time Bin (Jonas)' , fontsize=
    ↳16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 0].grid(True)

axis[0, 1].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↳columns[4]],linewidth=3, zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Photon Vs Number of Time Bin (General)' ,
    ↳fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↳columns[4]],linewidth=3, zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Photon Vs Number of Time Bin (PPM)' , fontsize =
    ↳16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)
```

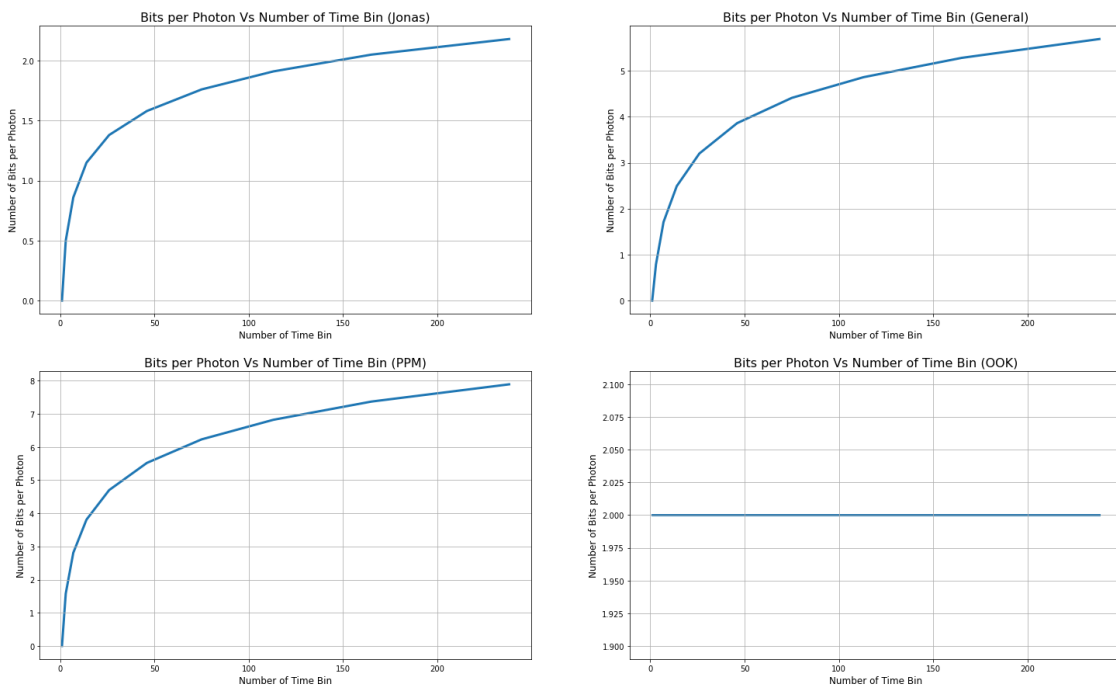


```

axis[1, 1].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
    ↳columns[4]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Photon Vs Number of Time Bin (OOK)' , fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('4SchemesBPP_TImeBin.png', dpi=450, bbox_inches='tight')
plt.show()

```



13 Bits/ Time Bin over 4 Schemes

```

[54]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[0, 0].set_title('Bits per Time Slot Vs Number of Photon (Jonas)', fontsize=16)
axis[0, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 0].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[0, 0].grid(True)

```

```

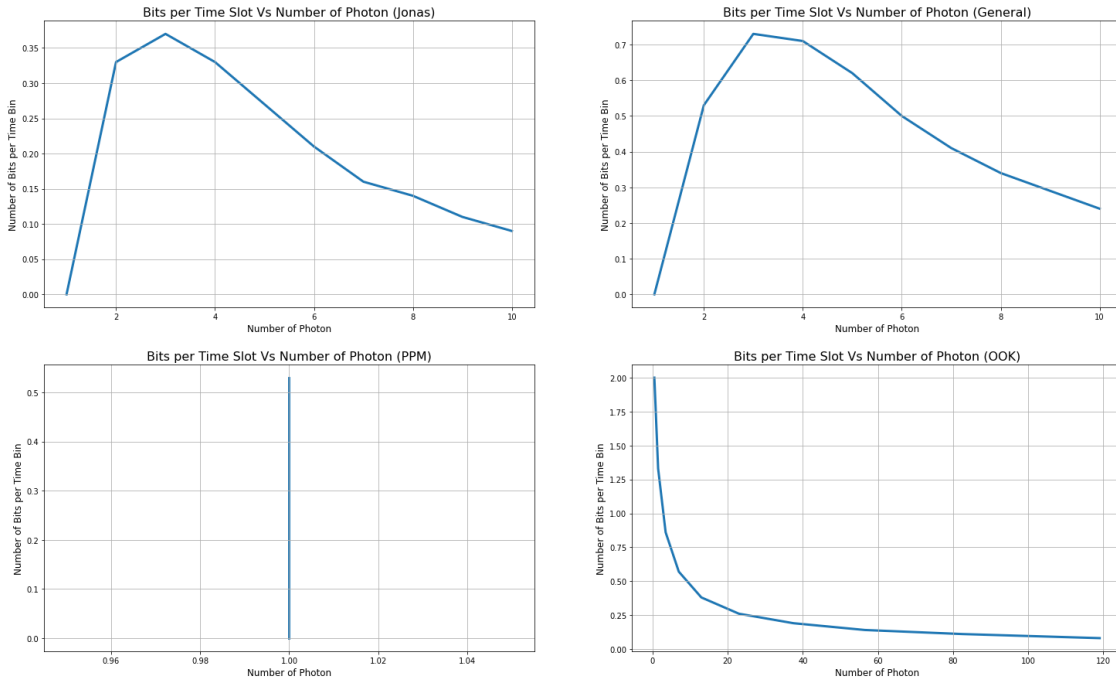
axis[0, 1].plot(df_General_c[df_General_c.columns[0]],df_General_c[df_General_c.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Time Slot Vs Number of Photon (General)',↳
    ↳fontsize = 16)
axis[0, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[0, 1].grid(True)

axis[1, 0].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Time Slot Vs Number of Photon (PPM)', fontsize =↳
    ↳16)
axis[1, 0].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 0].grid(True)

axis[1, 1].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
    ↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Time Slot Vs Number of Photon (OOK)', fontsize =↳
    ↳16)
axis[1, 1].set_xlabel('Number of Photon', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Time Bin', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
#plt.xticks(x)
plt.savefig('4schemesBPT_Photon.png', dpi=450, bbox_inches='tight')
plt.show()

```



```
[55]: figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[1]],df_Jonas[df_Jonas.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[0, 0].set_title('Bits per Photon Vs Number of Time Bin (Jonas)' , fontsize=
    ↪= 16)
axis[0, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 0].grid(True)

axis[0, 1].plot(df_General_c[df_General_c.columns[1]],df_General_c[df_General_c.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[0, 1].set_title('Bits per Photon Vs Number of Time Bin (General)' ,
    ↪fontsize = 16)
axis[0, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[0, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[0, 1].grid(True)

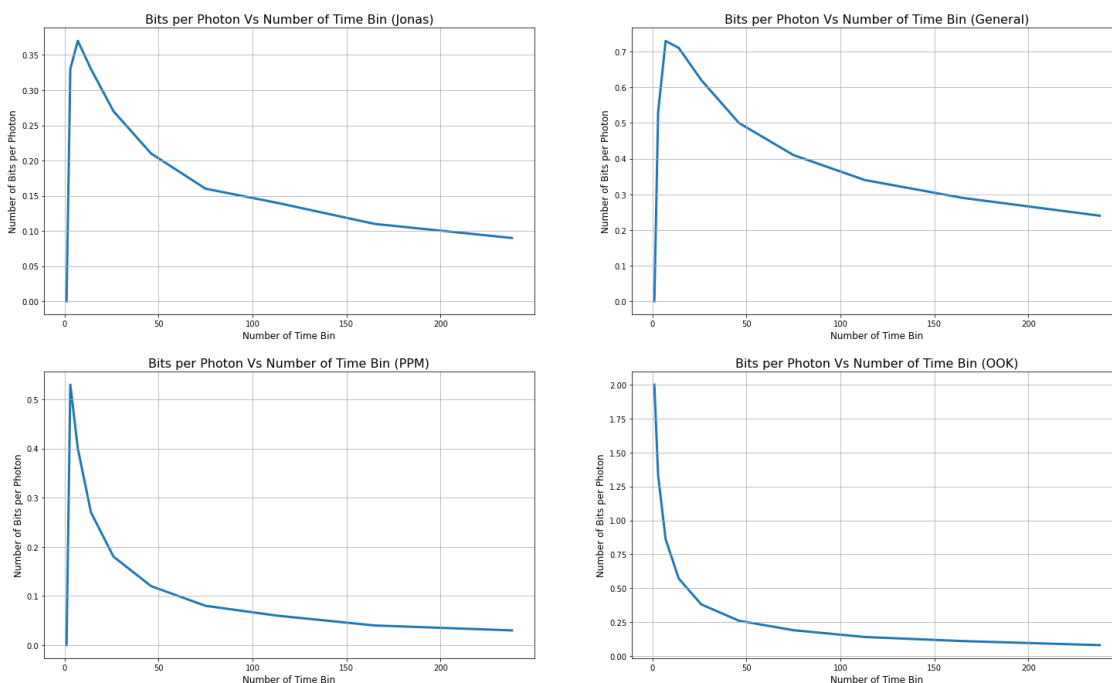
axis[1, 0].plot(df_PPM[df_PPM.columns[1]],df_PPM[df_PPM.
    ↪columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 0].set_title('Bits per Photon Vs Number of Time Bin (PPM)' , fontsize =
    ↪16)
axis[1, 0].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 0].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 0].grid(True)
```

```

axis[1, 1].plot(df_OOK[df_OOK.columns[1]],df_OOK[df_OOK.
↳columns[5]],linewidth=3, zorder=1, label = "bits")
axis[1, 1].set_title('Bits per Photon Vs Number of Time Bin (OOK)' , fontsize = 16)
axis[1, 1].set_xlabel('Number of Time Bin', fontsize = 12)
axis[1, 1].set_ylabel('Number of Bits per Photon', fontsize = 12)
axis[1, 1].grid(True)

figure.set_facecolor("white")
plt.savefig('4SchemesBPP_TimeBin.png', dpi=450, bbox_inches='tight')
plt.show()

```



[56]: # https://www.geeksforgeeks.org/pandas-set_option-function-in-python/
https://www.geeksforgeeks.org/pandas-set_option-function-in-python/
<https://stackoverflow.com/questions/35634238/how-to-save-a-pandas-dataframe-table-as-a-png>
↳ [how-to-save-a-pandas-dataframe-table-as-a-png](https://stackoverflow.com/questions/35634238/how-to-save-a-pandas-dataframe-table-as-a-png)
<https://towardsdatascience.com/6-pandas-display-options-you-should-memories-84adf8887bc3>
↳ [6-pandas-display-options-you-should-memories-84adf8887bc3](https://towardsdatascience.com/6-pandas-display-options-you-should-memories-84adf8887bc3)
<https://dash.plotly.com/datatable/width>
<https://randomds.com/2021/12/23/visualize-and-save-full-pandas-dataframes-as-images/>
↳ [visualize-and-save-full-pandas-dataframes-as-images/](https://randomds.com/2021/12/23/visualize-and-save-full-pandas-dataframes-as-images/)
<https://randomds.com/2021/12/23/visualize-and-save-full-pandas-dataframes-as-images/>
↳ [visualize-and-save-full-pandas-dataframes-as-images/](https://randomds.com/2021/12/23/visualize-and-save-full-pandas-dataframes-as-images/)

```
# https://www.adamsmith.haus/python/answers/  
→ how-to-print-a-number-in-scientific-notation-in-python  
# https://www.scaler.com/topics/python-scientific-notation/  
# https://stackoverflow.com/questions/20457038/  
→ how-to-round-to-2-decimals-with-python  
# https://pythonfix.com/pkg/d/dataframe-image/
```

[56] :