ErrorCorrectionipynb

February 9, 2023

[1]:
```
pip install dataframe_image
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: dataframe_image in /usr/local/lib/python3.8/dist-
packages (0.1.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (23.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.8/dist-packages
(from dataframe_image) (3.8.3)
Requirement already satisfied: mistune in /usr/local/lib/python3.8/dist-packages
(from dataframe_image) (0.8.4)
Requirement already satisfied: matplotlib>=3.1 in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (3.2.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (4.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (2.25.1)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (1.3.5)
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.8/dist-
packages (from dataframe_image) (5.6.1)
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: 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: numpy>=1.11 in /usr/local/lib/python3.8/dist-
packages (from matplotlib>=3.1->dataframe_image) (1.21.6)
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: pandocfilters>=1.4.1 in
```

```
/usr/local/lib/python3.8/dist-packages (from nbconvert>=5->dataframe_image)
(1.5.0)
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: bleach in /usr/local/lib/python3.8/dist-packages
(from nbconvert>=5->dataframe_image) (6.0.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-
packages (from nbconvert>=5->dataframe_image) (2.6.1)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-
packages (from nbconvert>=5->dataframe_image) (0.7.1)
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: testpath in /usr/local/lib/python3.8/dist-
packages (from nbconvert>=5->dataframe_image) (0.6.0)
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: charset-normalizer<3.0,>=2.0 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe_image) (2.1.1)
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: 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: aiosignal>=1.1.2 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->dataframe_image) (1.3.1)
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: 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: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-
packages (from requests->dataframe_image) (2.10)
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: 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: fastjsonschema in /usr/local/lib/python3.8/dist-
```

```
packages (from nbformat>=4.4->nbconvert>=5->dataframe_image) (2.16.2)
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: 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) (2.6.2)
Requirement already satisfied: pyrsistent!=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)
(3.12.0)
```

[2]:
```
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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

1 Sequence

[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]
```

2 The Number of Time Slot

[0, 1, 2, 4, 7, 12, 20, 29, 38, 52, 73, 94, 127]

[5]:
```
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]
```

[1,2,4,7] is encoded as follow: \ H H0 H000 V000000, which comprie of 14 time slots. We can find all the permutation as follow:

[6]:
```
# 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
```

[6]:
```
[(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 1 symbol created by 14 timeslots

4 Jonas

We have 4 photons 14 timeslots 1 symbol created by 14 timeslots

5 Number of ways to order the photons

[7]:
```
def ways(n):
    return math.factorial(n)
```

6 Number of Bits per Symbol

$$log_2 n!$$

[8]:
```
def bps(W):
    return math.log2(W)
```

7 Number of Bits per Photon

[9]:
```
def bpph(b,n):
    return b/n
```

8 Number of Bits per Timeslot

[10]:
```
def bpt(B, n, T):
    return B*n/T
```

[11]:
```
bpph(bps(24), 4)*4/14
```

[11]: 0.3274973214800826

9 Putting all the functions together

[12]:
```
series
```

[12]: [1, 3, 7, 14, 26, 46, 75, 113, 165, 238, 332, 459, 610, 791, 1002]

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

[13]: 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=['Photon Number ',
                           'Permutation ',
                           'Bits/Symbol ',
                           'Bits/Photon ',
                           'Bits/Time Slots ']
                  )

for n in range(1,11):
    Jonasways = math.factorial(n)
    Jonasbps = math.log2(Jonasways)
    Jonasbpp = math.log2(Jonasways) / n
    Jonasbpt = round(Jonasbpp * n/series[n-1], 3)
    df_Jonas.loc[len(df_Jonas)] = [n, Jonasways, Jonasbps, Jonasbpp, Jonasbpt]
```

[15]:
```
# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_Jonas, loc='center', fontsize=12) # where df is your data frame
# plt.savefig('Jonas_Table.png', dpi = 450, figsize=(10, 10))
```

[85]:
```
# df_Jonas
```

[17]:
```
# table(ax, df_Jonas)
# plt.show()
# plt.savefig('mytable.png')
```

[88]:
```
#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"
)
```

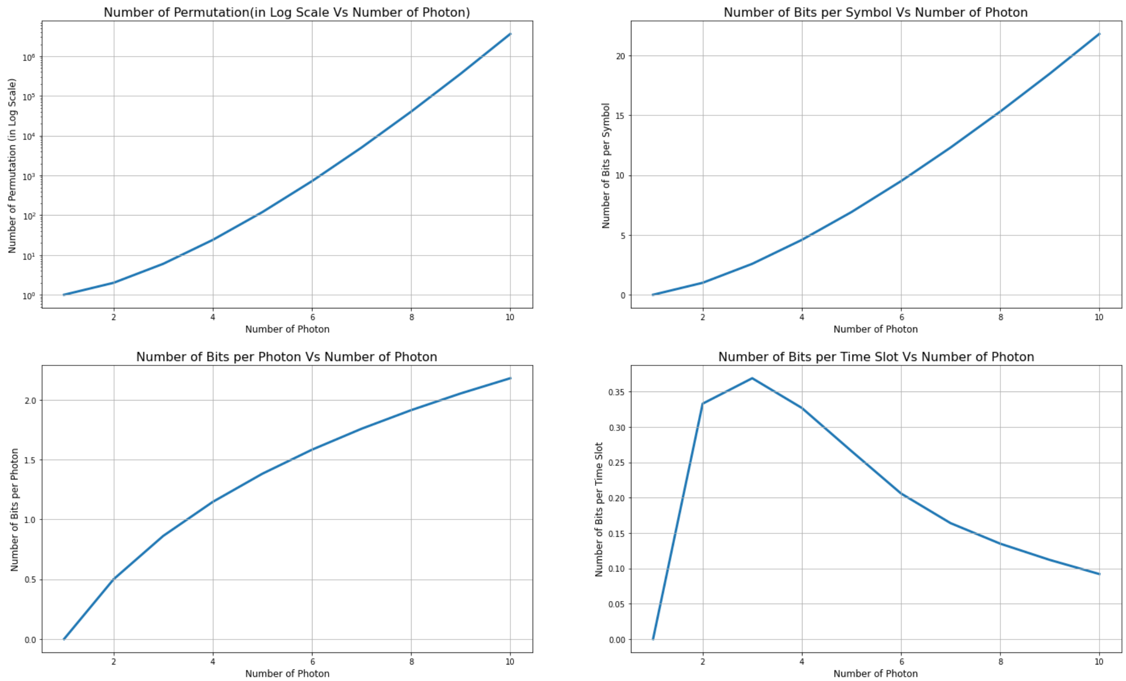
[91]:
```
# display(file="Jonas_Table.png")
```

[19]:
```
# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_Jonas, loc='center', fontsize=16) # where df is your data frame
# plt.savefig('Jonas_Table.png', dpi = 450, figsize=(10, 10))
```

[20]:
```
# import matplotlib
# import seaborn as sns

# def save_df_as_image(df, path):
#     # Set background to white
#     norm = matplotlib.colors.Normalize(-1,1)
#     colors = [[norm(-1.0), "white"],
#             [norm( 1.0), "white"]]
#     cmap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)
#     # Make plot
#     plot = sns.heatmap(df, annot=True, cmap=cmap, cbar=False)
#     fig = plot.get_figure()
#     fig.savefig(path)
```

10 Plot Graph

[21]:
```
figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
↪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_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
↪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_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
↪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_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
↪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('Jonas_Plot.png', dpi=450, bbox_inches='tight')
plt.show()
```



[22]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.columns[1]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Permutation(in Log Scale)', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Permutation (in Log Scale)', fontsize = 12)
# ax1.set_yscale('log')
# ax1.set_xlim([0, 21])
# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('Jonas_PhotonVsPermutation.png', dpi=300, bbox_inches='tight')
```

[23]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
```

```
# ax1.plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
 ↪columns[2]],linewidth=3,zorder=1, label = "bits")



# ax1.set_title('Number of Photon Vs Number of Bits per Symbol', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Symbol', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# #figure.set_facecolor("white")
# plt.savefig('Jonas_PhotonVsBpS.png', dpi=300, bbox_inches='tight')
```

[24]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
 ↪columns[3]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Bits per Photon', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Photon', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('Jonas_PhotonVsBpP.png', dpi=300, bbox_inches='tight')
```

[25]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_Jonas[df_Jonas.columns[0]],df_Jonas[df_Jonas.
 ↪columns[4]],linewidth=3, zorder=1, label = "bits")



# ax1.set_title('Number of Photon Vs Number of Bits per Time Slot', fontsize =␣
 ↪16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
```

```
# ax1.set_ylabel('Number of Bits per Time Slot', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('Jonas_PhotonVsTime.png', dpi=300, bbox_inches='tight')
```

11 PPM

We have 1 photon 14 timeslots 14 ways to order them

[26]:
```
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',
                               'Number of Permutation',
                               'Number of Bits per Symbol',
                               'Number of Bits per Photon',
                               'Number of Bits per Time Slots']
                     )
T =14
for n in range(1,11):
    PPMways = series[n-1]
    PPMbps = math.log2(PPMways)
    PPMbpp = math.log2(PPMways) / 1
    PPMbpt = round(PPMbpp * n/series[n-1], 3)
    df_PPM.loc[len(df_PPM)] = [n, PPMways, PPMbps, PPMbpp, PPMbpt]
```

[27]:
```
PPMways
```

[27]: 238

[28]:
```
#def PPMbpt(T, n):
#    def PPMbpph(n):
#        def PPMbps(n):
#            def PPMways(T):
#                return T
#            return math.log2(T)
```

```
#        return math.log2(T) / n
#    return print([n], "Number of wyas:", ways(n),
#                 "Number of Bits per Symbol:", math.log2(ways(n)),
#                 "Number of Bits per Photon: ", math.log2(ways(n)) / n,
#                 "Number of Bits per Timeslot", math.log2(ways(n)) / n*(n/T)
#                )
```

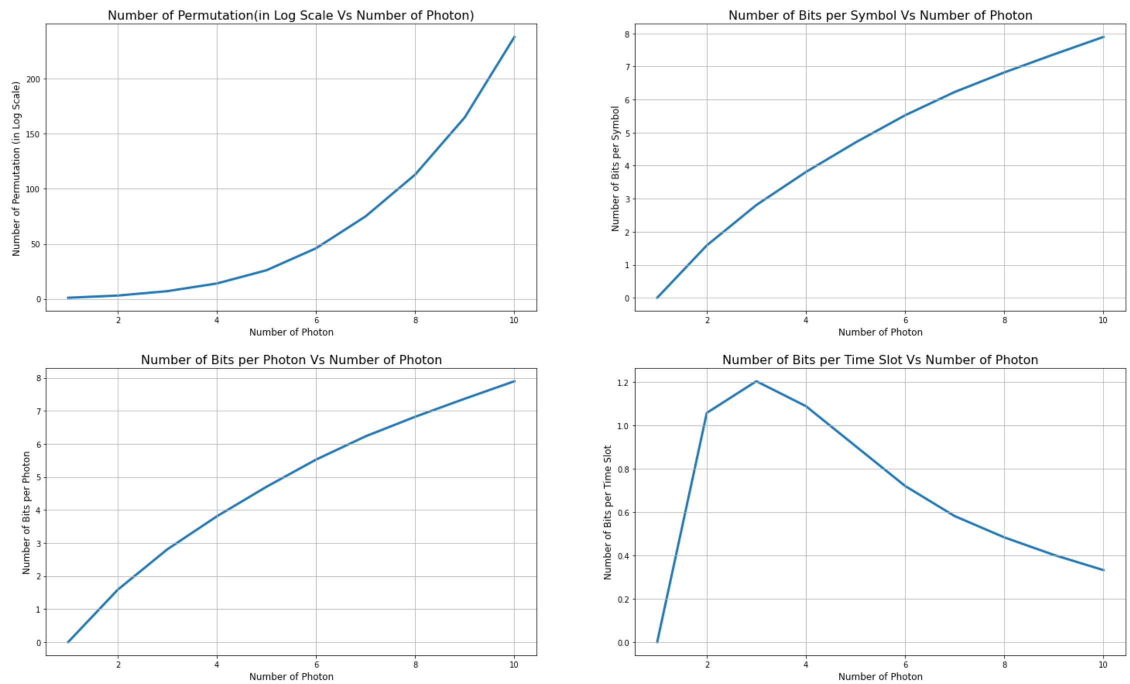
[82]:
```
# df_PPM.style
```

[84]:
```
# ax = plt.subplot(111, frame_on=False)

# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_PPM, loc='center',fontsize=12) # where df is your data frame
# plt.savefig('PPM_Table.png', dpi = 450)
```

[92]:
```
dfi.export(
    df_PPM,
    "PPM_Table.png",
    table_conversion="matplotlib"
)
```

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

axis[0, 0].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
 ↪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_xlim([0, 21])
axis[0, 0].grid(True)

axis[0, 1].plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
 ↪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_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
 ↪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_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
↪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('PPM_Plot.png', dpi=450, bbox_inches='tight')
plt.show()
```



[32]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
↪columns[1]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Permutation', fontsize = 16)
```

```
# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Permutation', fontsize = 12)
# #ax1.set_yscale('log')
# ax1.set_ylim([0,20])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('PPM_PhotonVsPermutation.png', dpi=300, bbox_inches='tight')
```

[33]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
 ↪columns[2]],linewidth=3,zorder=1, label = "bits")



# ax1.set_title('Number of Photon Vs Number of Bits per Symbol', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Symbol', fontsize = 12)

# ax1.set_ylim([0,5.0])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# #figure.set_facecolor("white")
# plt.savefig('PPM_PhotonVsBpS.png', dpi=300, bbox_inches='tight')
```

[34]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
 ↪columns[3]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Bits per Photon', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Photon', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)
```

```
# figure.set_facecolor("white")
# plt.savefig('PPM_PhotonVsBpP.png', dpi=300, bbox_inches='tight')
```

[35]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_PPM[df_PPM.columns[0]],df_PPM[df_PPM.
↪columns[4]],linewidth=3,zorder=1, label = "bits")


# ax1.set_title('Number of Photon Vs Number of Bits per Time Slot', fontsize =␣
↪16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Time Slot', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('PPM_PhotonVsTime.png', dpi=300, bbox_inches='tight')
```

12 OOK

[36]:
```
#def OOKbpt(T, n):
#  def OOKbpph(n, T):
#    def OOKbps(T):
#      def OOKways(T):
#        return (2)**T
#      return math.log2((2)**T)
#    return math.log2((2)**T) / n
#  return print(
#                  [n],
#                  "Number of Permutation:", OOKways(T),
#                  "Number of Bits per Symbol:", math.log2(OOKways(T)),
#                  "Number of Bits per Photon: ", math.log2(OOKways(T)) / n,
#                  "Number of Bits per Timeslot", math.log2(OOKways(T)) / n*(n/T)
#                  )
```

[37]:
```
OOKnumber = []
OOKPermutation = []
OOKBPS = []
OOKBPP = []
```

```
OOKBPT = []

df_OOK = pd.DataFrame(columns=['Number of Photon',
                              'Number of Permutation',
                              'Number of Bits per Symbol',
                              'Number of Bits per Photon',
                              'Number of Bits per Time Slots']
                     )
T =14
for n in range(1,11):
    OOKways = (2)**series[n-1]
    OOKbps = math.log2(OOKways)
    #OOKbpp = (math.log2(OOKways) / n)/7
    OOKbpp = 2
    OOKbpt = round(OOKbpp * n/series[n-1], 3)
    df_OOK.loc[len(df_OOK)] = [n, OOKways, OOKbps, OOKbpp, OOKbpt]
```

[93]:
```
# df_OOK.style
```

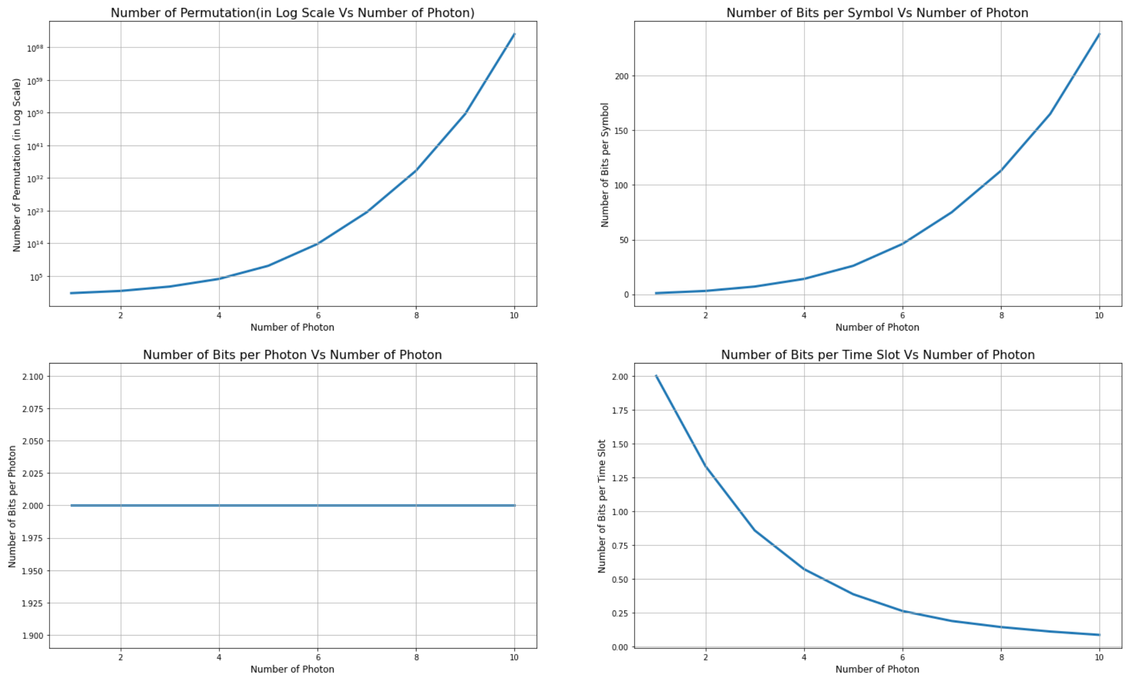
[94]:
```
# df_OOK
```

[95]:
```
# ax = plt.subplot(111, frame_on=False)

# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_OOK, loc='center') # where df is your data frame
# plt.savefig('OOK_Table.png', dpi = 450)
```

[97]:
```
dfi.export(
    df_OOK,
    "OOK_Table.png",
    table_conversion="matplotlib"
)
```

[41]:
```
figure, axis = plt.subplots(2,2,figsize=(25,15))

axis[0, 0].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
 ↪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])
```

```python
axis[0, 0].grid(True)

axis[0, 1].plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
 ↪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_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
 ↪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_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
 ↪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('OOK_Plot.png', dpi=450, bbox_inches='tight')
plt.show()
```



[42]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
↪columns[1]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Permutation', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Permutation', fontsize = 12)
# #ax1.set_yscale('log')
# #ax1.set_ylim([0,20])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('OOK_PhotonVsPermutation.png', dpi=300, bbox_inches='tight')
```

[43]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
↪columns[2]],linewidth=3,zorder=1, label = "bits")
```

```
# ax1.set_title('Number of Photon Vs Number of Bits per Symbol', fontsize = 16)

# ax1.set_xlabel('Number of Photon')
# ax1.set_ylabel('Number of Bits per Symbol')

# #ax1.set_ylim([0,5.0])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# #figure.set_facecolor("white")
# plt.savefig('OOK_PhotonVsBpS.png', dpi=300, bbox_inches='tight')
```

[44]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
↪columns[3]],linewidth=3,zorder=1, label = "bits")

# ax1.set_title('Number of Photon Vs Number of Bits per Photon', fontsize = 16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Photon', fontsize = 12)

# #ax1.set_ylim([6000,7000])
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('OOK_PhotonVsBpP.png', dpi=300, bbox_inches='tight')
```

[45]:
```
# figure, ax1 = plt.subplots(figsize=(18,10))

# #ax = df1.plot(, xticks=range(0, 61), title = 'Detection Rate in Data 1')
# ax1.plot(df_OOK[df_OOK.columns[0]],df_OOK[df_OOK.
↪columns[4]],linewidth=3,zorder=1, label = "bits")


# ax1.set_title('Number of Photon Vs Number of Bits per Time Slot', fontsize =␣
↪16)

# ax1.set_xlabel('Number of Photon', fontsize = 12)
# ax1.set_ylabel('Number of Bits per Time Slot', fontsize = 12)

# #ax1.set_ylim([6000,7000])
```

```
# #ax2.set_ylim([6000,7000])

# ax1.grid(True)

# figure.set_facecolor("white")
# plt.savefig('OOK_PhotonVsTime.png', dpi=300, bbox_inches='tight')
```

13 General

We have 4 photons 14 timeslots 1,0001 ways to order them by binomial where order does not matter and repeition are not allowed. Permutations = r! x Combinations

[46]:
```
# A Python program to print all
# permutations of given length
from itertools import permutations

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

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

```
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

[47]:
```
# # A Python program to print all
# # permutations of given length
# from itertools import permutations

# # Get all permutations of length 2
# # and length 2
# perm = permutations([1, 2, 4, 7, 0,0,0,0,0], )2

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

[48]:
```
def permutation(n,r):
  return math.factorial(n)/math.factorial(n-r)
```

[49]:
```
def combination(n,r):
  return math.factorial(n) / (math.factorial(n-r)*math.factorial(r))
```

[50]:
```
combination(14,4)
```

[50]: 1001.0

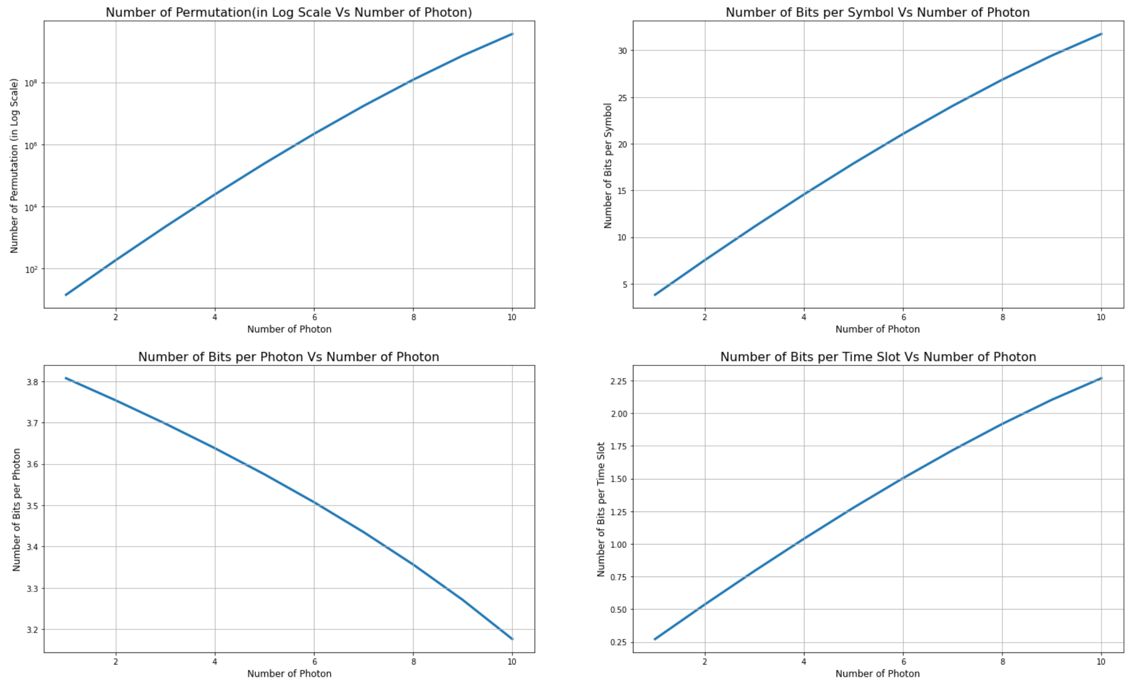
[51]:
```
Generalnumber_p = []
GeneralPermutation_p = []
GeneralBPS_p = []
GeneralBPP_p = []
GeneralBPT_p = []

df_General_p = pd.DataFrame(columns=['Number of Photon',
                           'Number of Permutation',
                           'Number of Bits per Symbol',
                           'Number of Bits per Photon',
                           'Number of Bits per Time Slots']
                  )

for n in range(1,11):
    Generalways_p = permutation(14,n)
    Generalbps_p = math.log2(Generalways_p)
    Generalbpp_p = math.log2(Generalways_p) / n
    Generalbpt_p = round(Generalbpp_p * n/14, 3)
    df_General_p.loc[len(df_General_p)] = [n, Generalways_p, Generalbps_p,␣
 ↪Generalbpp_p, Generalbpt_p]
```

[100]:
```
dfi.export(
    df_General_p,
    "General_p_Table.png",
    table_conversion="matplotlib"
)
```

[101]:
```
# ax = plt.subplot(111, frame_on=False)

# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_General_p, loc='center', fontsize=12) # where df is your data␣
 ↪frame
# plt.savefig('General_p_Table.png', dpi = 450)
```

[53]:
```
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()
```



[54]:
```
GeneralNumber_c = []
GeneralPermutation_c = []
GeneralBPS_c = []
GeneralBPP_c = []
GeneralBPT_c = []

df_General_c = pd.DataFrame(columns=['Number of Photon',
                              'Number of Permutation',
                              'Number of Bits per Symbol',
                              'Number of Bits per Photon',
                              'Number of Bits per Time Slots']
                   )

for n in range(1,11):
    Generalways_c = combination(14,n)
    Generalbps_c = math.log2(Generalways_c)
    Generalbpp_c = math.log2(Generalways_c) / n
    Generalbpt_c = round(Generalbpp_c * n/14, 3)
    df_General_c.loc[len(df_General_c)] = [n, Generalways_c, Generalbps_c, Generalbpp_c, Generalbpt_c]
```

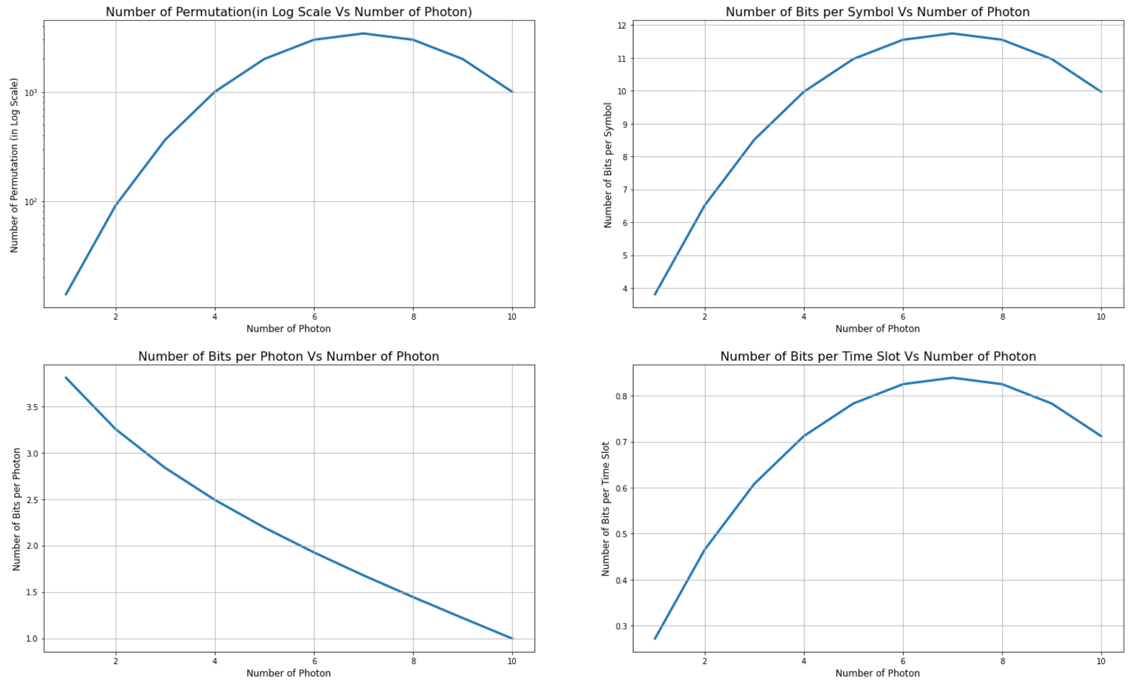
[102]:
```
dfi.export(
    df_General_c,
    "General_c_Table.png",
    table_conversion="matplotlib"
```

```
)
```

[103]:
```
# ax = plt.subplot(111, frame_on=False)

# ax = plt.subplot(111, frame_on=False) # no visible frame
# ax.xaxis.set_visible(False) # hide the x axis
# ax.yaxis.set_visible(False) # hide the y axis
# table(ax, df_General_c, loc='center') # where df is your data frame
# plt.savefig('General_c_Table.png', dpi = 450)
```

[56]:
```
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[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_c[df_General_c.columns[0]],df_General_c[df_General_c.
 ↪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_c[df_General_c.columns[0]],df_General_c[df_General_c.
 ↪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_c[df_General_c.columns[0]],df_General_c[df_General_c.
 ↪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_c_Plot.png', dpi=450, bbox_inches='tight')
plt.show()
```



14 Hamming Distance

[57]:
```
# Using scipy to calculate the Hamming distance
from scipy.spatial.distance import hamming

values1 = [1, 1, 0, 0, 1]
values2 = [0, 1, 0, 0, 0]

values3 = [1, 0, 1, 1, 0, 1]
values4 = [0, 0, 1, 1, 0, 0]

hamming_distance_1 = hamming(values1, values2) * len(values1)

hamming_distance_2 = hamming(values3, values4) * len(values3)

print(hamming_distance_1)

print(hamming_distance_2)
```

```
2.0
2.0
```

[58]:
```
hamming(values1, values2)
```

[58]:
```
0.4
```

[59]:
```
hamming(values3, values4)
```

[59]:
```
0.3333333333333333
```

15 TC-MPPM Constellations

Practical PPM systems with w = 1 typically use n = 2,4,6,8, 16, etc.

5C3

[60]:
```
# # # A Python program to print all
# # # permutations using library function
# from itertools import permutations, combinations


# # # Get all permutations of [1, 2, 3]
# perm = permutations([1, 1, 1, 0, 0])
# # com = combinations([1, 1, 1, 0, 0],3)
# # # Print the obtained permutations
# for i in list(perm):
#      print (i)
```

[61]:
```
values51 = [1, 1, 1, 0, 0]
values52 = [0, 0, 1, 1, 1]
values53 = [1, 1, 0, 0, 0]
values54 = [1, 1, 1, 0, 0]
values55 = [0, 1, 0, 1, 0]
values56 = [0, 1, 1, 1, 0]

hamming_distance_51 = hamming(values51, values52) * len(values51) # 4/5␣
 ↪difference * 5
hamming_distance_52 = hamming(values51, values53) * len(values51) # 1/5␣
 ↪difference * 5
hamming_distance_53 = hamming(values51, values54) * len(values54) # 0/5␣
 ↪difference * 5
hamming_distance_54 = hamming(values52, values53) * len(values52) # 5/5␣
 ↪difference * 5
hamming_distance_55 = hamming(values54, values55) * len(values52) # 3/5␣
 ↪difference * 5
```

```
hamming_distance_56 = hamming(values51, values56) * len(values52) # 2/5␣
↪difference * 5

print(hamming(values51, values52),
      hamming(values51, values53),
      hamming(values51, values54),
      hamming(values52, values53),
      hamming(values51, values56))

print(len(values51),
      len(values52),
      len(values53),
      len(values54),
      len(values55),
      len(values56))

print(hamming_distance_53)
print(hamming_distance_52)
print(hamming_distance_56)
print(hamming_distance_55)
print(hamming_distance_51)
print(hamming_distance_54)
```

```
0.8 0.2 0.0 1.0 0.4
5 5 5 5 5 5
0.0
1.0
2.0
3.0
4.0
5.0
```

16 Draft

[62]:
```
# 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
```

[62]:
```
[(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)]
```

[63]:
```
def differences(a, b):
    if len(a) != len(b):
        raise ValueError("Lists of different length.")
    return sum(i != j for i, j in zip(a, b))
    return
```

[64]:
```
# differences(lists[0], lists[1])
# for i in range()
```

[65]:
```
List1 = [10,10,11,12,15,16,18,19]
List2 = [10,11,13,15,16,19,20]
List3 = [10,11,11,12,15,19,21,23]

inter = set(List1).intersection(List2, List3)

diff3 = set(List3).difference(inter)

print(diff3)
set([12, 21, 23])
```

{12, 21, 23}

[65]: {12, 21, 23}

[66]:
```python
# A Python program to print all
# permutations using library function
from itertools import permutations


# Get all permutations of [1, 2, 3]
perm = permutations([1, 1, 0, 0])

lists = []
# Print the obtained permutations
for i in (perm):
    #print (i)
    lists.append(i)
lists
```

[66]:
```
[(1, 1, 0, 0),
 (1, 1, 0, 0),
 (1, 0, 1, 0),
 (1, 0, 0, 1),
 (1, 0, 1, 0),
 (1, 0, 0, 1),
 (1, 1, 0, 0),
 (1, 1, 0, 0),
 (1, 0, 1, 0),
 (1, 0, 0, 1),
 (1, 0, 1, 0),
 (1, 0, 0, 1),
 (0, 1, 1, 0),
 (0, 1, 0, 1),
 (0, 1, 1, 0),
 (0, 1, 0, 1),
 (0, 0, 1, 1),
 (0, 0, 1, 1),
 (0, 1, 1, 0),
 (0, 1, 0, 1),
 (0, 1, 1, 0),
 (0, 1, 0, 1),
 (0, 0, 1, 1),
 (0, 0, 1, 1)]
```

[67]:
```python
from sympy.utilities.iterables import multiset_permutations
from sympy import factorial
from pprint import pprint
```

[68]:
```python
pprint(list(multiset_permutations([1,1,1,0,0,0])))
```

```
[[0, 0, 0, 1, 1, 1],
 [0, 0, 1, 0, 1, 1],
 [0, 0, 1, 1, 0, 1],
 [0, 0, 1, 1, 1, 0],
 [0, 1, 0, 0, 1, 1],
 [0, 1, 0, 1, 0, 1],
 [0, 1, 0, 1, 1, 0],
 [0, 1, 1, 0, 0, 1],
 [0, 1, 1, 0, 1, 0],
 [0, 1, 1, 1, 0, 0],
 [1, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 1],
 [1, 0, 0, 1, 1, 0],
 [1, 0, 1, 0, 0, 1],
 [1, 0, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 0],
 [1, 1, 0, 0, 0, 1],
 [1, 1, 0, 0, 1, 0],
 [1, 1, 0, 1, 0, 0],
 [1, 1, 1, 0, 0, 0]]
```

[70]:
```python
# len(a[0])
```

[71]:
```python
for i in a:
    print(i)
```

```
1
2
3
5
8
13
21
30
39
53
74
95
128
152
182
212
258
316
374
413
```

476
531
546
608
717
798
862
965
1060
1161
1307
1386
1435
1556
1722
1834
1934
2058
2261
2497
2699
2874
3061
3197
3332
3629
3712
3868
4140
4447
4640
5022
5065
5323
5614
6004
6274
6494
6642
6980
7276
7588
8018
8374
9072
9168
9761
10106

```
10490
11110
11375
11517
12102
12331
12868
13427
13924
14536
14912
15470
15905
16137
16901
17042
17823
19422
19934
20289
20997
21492
22066
22613
22660
23725
24400
24970
25361
26072
26681
27602
```

[72]:

```
a = list(multiset_permutations([1,1,1,0,0,0]))
for i in a:
    print(len(i))
```

```
6
6
6
6
6
6
6
6
6
6
6
```

```
6
6
6
6
6
6
6
6
6
```

6C3

[73]:
```
differences(a[1], a[6])
```

[73]:
```
4
```

[74]:
```
# [''.join(i) for i in multiset_permutations('aab')]
# ['aab', 'aba', 'baa']
# len(list(multiset_permutations('banana')))
```

[75]:
```
pprint(list(multiset_permutations([1,1,1,0,0,0])))
```

```
[[0, 0, 0, 1, 1, 1],
 [0, 0, 1, 0, 1, 1],
 [0, 0, 1, 1, 0, 1],
 [0, 0, 1, 1, 1, 0],
 [0, 1, 0, 0, 1, 1],
 [0, 1, 0, 1, 0, 1],
 [0, 1, 0, 1, 1, 0],
 [0, 1, 1, 0, 0, 1],
 [0, 1, 1, 0, 1, 0],
 [0, 1, 1, 1, 0, 0],
 [1, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 1],
 [1, 0, 0, 1, 1, 0],
 [1, 0, 1, 0, 0, 1],
 [1, 0, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 0],
 [1, 1, 0, 0, 0, 1],
 [1, 1, 0, 0, 1, 0],
 [1, 1, 0, 1, 0, 0],
 [1, 1, 1, 0, 0, 0]]
```

[76]:
```
values61 = [1, 1, 1, 0, 0, 0]
values65 = [1, 0, 1, 0, 1, 0]
values62 = [1, 1, 1, 0, 0, 0]
values63 = [0, 1, 1, 1, 0, 0]
values64 = [0, 0, 1, 1, 1, 0]
```

```
values64 = [0, 0, 0, 1, 1, 1]


hamming_distance_61 = hamming(values61, values62) * len(values61) # 0/6␣
 ↪difference
hamming_distance_62 = hamming(values61, values63) * len(values61) # 1/6␣
 ↪difference
hamming_distance_63 = hamming(values61, values64) * len(values61) # 2/6␣
 ↪difference=
hamming_distance_64 = hamming(values61, values65) * len(values61) #


print(hamming(values61, values62),
      hamming(values61, values63),
      hamming(values61, values64),
      hamming(values61, values65)
      )

print(len(values61),
      len(values62),
      len(values63),
      len(values64)
      )

print(hamming_distance_61)
print(hamming_distance_62)
print(hamming_distance_63)
print(hamming_distance_64)
```

```
0.0 0.3333333333333333 1.0 0.3333333333333333
6 6 6 6
0.0
2.0
6.0
2.0
```

[77]:
```
import itertools
a = [1,2,3]
b = [4,5]
c = [-1]
# result contains all possible combinations.
combinations = list(itertools.product(a,b,c))
combinations
```

[77]:
```
[(1, 4, -1), (1, 5, -1), (2, 4, -1), (2, 5, -1), (3, 4, -1), (3, 5, -1)]
```

[78]:
```
combinations = list(itertools.product(a,b,c))
```

7C3

[79]:
```python
values71 = [1, 0, 1, 0, 0, 1, 0]
values72 = [0, 1, 0, 0, 1, 0, 1]
hamming_distance_7 = hamming(values71, values72) * len(values71)
print(hamming_distance_7)
```

```
6.0
```

[80]:
```python
from itertools import combinations
s1 = pd.Series([1,1,1,0,0])
s2 = pd.Series([1,1,1,0,0])

x = combinations(s1, 2)
y = combinations(s2, 2)

dfx = pd.DataFrame(list(x)).rename(columns=lambda x: x+1).add_prefix('x')
dfy = pd.DataFrame(list(y)).rename(columns=lambda x: x+1).add_prefix('y')
df = pd.concat([dfx, dfy], axis=1)

m1 = (df.x1 > df.x2) & (df.y1 > df.y2)
m2 = (df.x1 < df.x2) & (df.y1 < df.y2)
# m3 =
# m4 =
# m5 =
m = m1 | m2

print (m)
```

```
0    False
1    False
2     True
3     True
4    False
5     True
6     True
7     True
8     True
9    False
dtype: bool
```

[81]:
```python
df['score'] = np.where(m, m.cumsum(), 0)
print (df)
```

```
   x1  x2  y1  y2  score
0   1   1   1   1      0
1   1   1   1   1      0
2   1   0   1   0      1
```

```
3  1  0  1  0    2
4  1  1  1  1    0
5  1  0  1  0    3
6  1  0  1  0    4
7  1  0  1  0    5
8  1  0  1  0    6
9  0  0  0  0    0
```

[81]: