Çağdaş Güven 2738938

ME 536

Hide Words In High Dimensions

then find and count them :)

In [1]:
```
# only importing from following libraries are allowed. You can add more i
from skimage import io
from skimage.filters import threshold_otsu as otsu
import numpy as np
from scipy.linalg import orth
from numpy.linalg import matrix_rank as rank
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import sklearn


# also import the matrix printing function
!rm bug_numpy_utils.py 2> dump.me # DONT FORGET TO CHANGE % TO ! BEFORE S
!wget https://raw.githubusercontent.com/bugrakoku/bug_python_utils/main/b
from bug_numpy_utils import CData as CMe
from bug_numpy_utils import GenerateDataforImage as GenImMat
from bug_numpy_utils import text2mat
```

```
--2024-11-27 23:35:52--  https://raw.githubusercontent.com/bugrakoku/bug_p
ython_utils/main/bug_numpy_utils.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.111.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18456 (18K) [text/plain]
Saving to: ‘bug_numpy_utils.py’

bug_numpy_utils.py  100%[===================>]  18.02K  --.-KB/s    in 0.0
09s

2024-11-27 23:35:53 (1.87 MB/s) - ‘bug_numpy_utils.py’ saved [18456/18456]
```

Intro to Basics: Assignment has not started yet!

This is the warm up

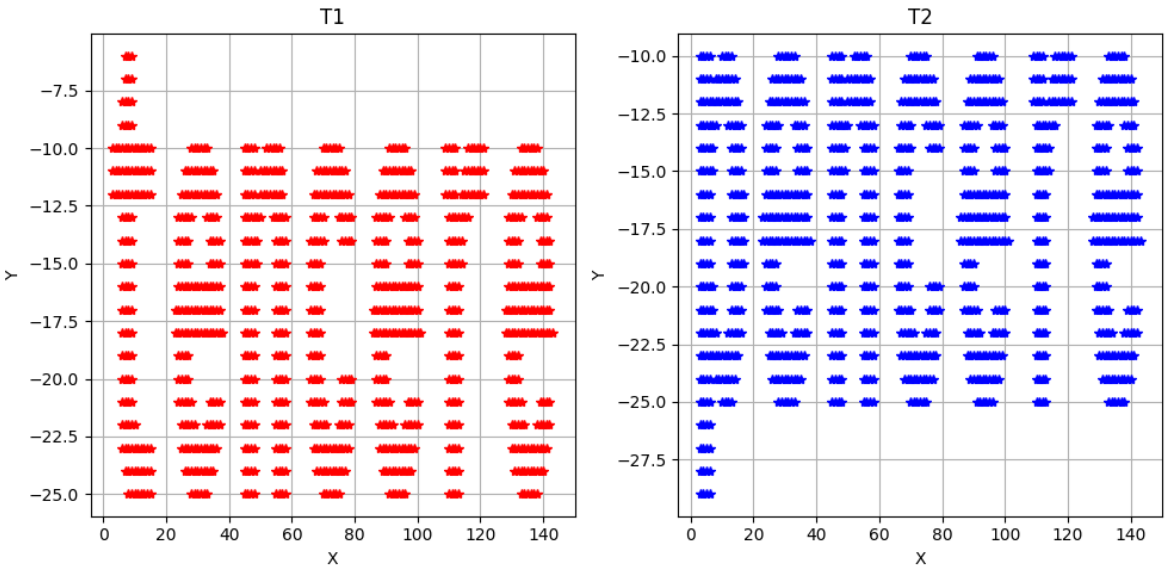
Generate and plot reference text

Generate data matrices based on two strings of same length.
Columns of these matrices are data points, which when plotted is read as the given string. If you prefer strings of different lengths, fill in the short string with a character.
Using pyplot display the data points to make sure that they are readable.

In [2]:
```
S1 = 'tencere'
S2 = 'pencere'
T1, T1num = text2mat(S1)
T2, T2num = text2mat(S2)

# just that we get to understand ```text2mat``` function let's print the
print(f'Shape of T1 = {T1.shape}, where letters of "{S1}" has {T1num} dat
print(f'Shape of T2 = {T2.shape}, where letters of "{S2}" has {T2num} dat
```

```
Shape of T1 = (3, 932), where letters of "tencere" has [121, 153, 142, 120
, 153, 90, 153] data points in each corresponding letter
Shape of T2 = (3, 985), where letters of "pencere" has [174, 153, 142, 120
, 153, 90, 153] data points in each corresponding letter
```

In [3]:
```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Plot T1
axes[0].plot(T1[0, :], T1[1, :], '*r')
axes[0].set_title('T1')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[0].grid(True)

# Plot T2
axes[1].plot(T2[0, :], T2[1, :], '*b')
axes[1].set_title('T2')
axes[1].set_xlabel('X')
axes[1].set_ylabel('Y')
axes[1].grid(True)

plt.tight_layout()
plt.show()
```



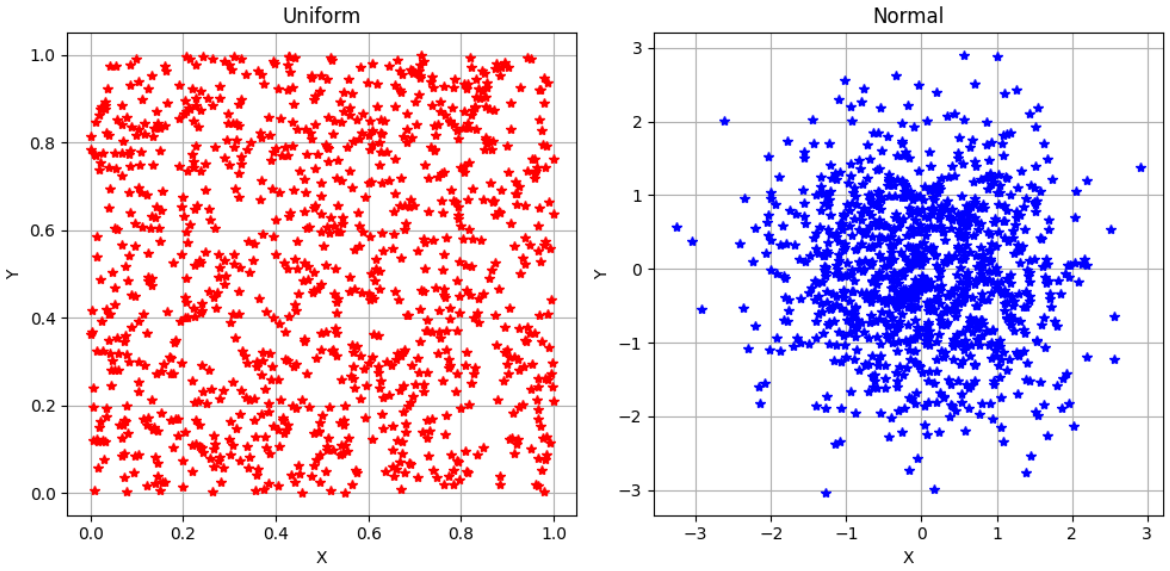
See them in 3D

In [4]:
```
# CELL 1
# also let's see these text in 3D
# note that text2mat actually provides data in 3D but with z=0, hence tex
CMe(T1, f'{S1}')
CMe(T2, f'{S2}')
```

Recall random uniform vs normal distribution

This implies noise will be added later in the assignment :)

In [5]:
```
N1 = np.random.rand(2,1000)
N2 = np.random.randn(2,1000)

fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Plot N1
axes[0].plot(N1[0, :], N1[1, :], '*r')
axes[0].set_title('Uniform')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[0].grid(True)

# Plot N2
axes[1].plot(N2[0, :], N2[1, :], '*b')
axes[1].set_title('Normal')
axes[1].set_xlabel('X')
axes[1].set_ylabel('Y')
axes[1].grid(True)

plt.tight_layout()
plt.show()
```



Assignment starts here

Read the following cells carefully and respond by filling in the code and text cells. Your explanations brief yet clear.

You are to practice and hopefully get better if not excel in SVD usage at the end of this assignment.

Let's generate some reference text as we did above.

I will change this text (i.e. `S1, S2`) while grading, and run the following many times to see how your code works.

You can try likewise, however, make sure that your submission does not involve any of your personal test code.

I will refer the reference text using `T1` or `T2` in the remainder of this assignment. I might also use `S1` and `T1` interchangebly to refer to the same text content.

In [6]:
```
# similar to above, generate reference text
S1 = 'tencere'
S2 = 'pencere'
T1, T1num = text2mat(S1)
T2, T2num = text2mat(S2)

# let's make sure that we start with a 2D array, recall that text2mat fun
T1 = T1[0:2, :] # get only the XY coordinates of the data points, note th
T2 = T2[0:2, :] # get only the XY coordinates of the data points, note th
```

In [9]:
```
# CELL 2
Np = 3 # dimension of the ambient space to which data will be projected
B = np.random.rand(Np,2) # generate a random basis
ND_Data = B@T1 # convert or project 2D data points to Np Dimensions, in o
# we start with 3 dimensions so that we can view
# let's view the projected text in 3D
CMe(ND_Data)
```

Explain - 1:

Run the above code couple of times to observe that the text `T1` is seen as if it is written in *italic*. It might even seem printed in reverse or its mirror image.

Briefly explain, why the text is not as readable as it is seen when 'CELL 1' under "See them in 3D" is run.

HINT: If you do not see the reason why, or the plot does not seem right at firt, try running this cell with `Np=2` couple of times in CELL 2

Fix the code in 'CELL 2' so that

It does look in the same shape despite it is always projected randomly to some plane in 3D.

Note that if 3 is change with N, your code should also work for N-D.

In [ ]:
```
# when this cell is run, similar to 'CELL 2', text in T1 will be project
# in order so that we can generate high dimensional data later in the ass
# you are to fill in the following function

# we are implementing this following function

def DataInND(M, Nd = 3, NoiseLevel = 0.0, NoiseType = 'Normal'):
    '''
    M is a 2xK matrix, indicating that there are K data points in 2D
    Nd is the dimension to which data in M will be projected onto a rando
    After projection Res is a NdxK matrix, it should be rank-2 at this po
    If NoiseLevel is not zero:
        a random matrix that is also NdxK and this matrix is added to the
        after being multiplied by NoiseLevel
    Random matrix is generated using randn() if NoiseType is 'Normal' els

    Finally Res is returned
    '''
    Res = None



    return Res

# do not change the following
# if your function works, following should work
ND_Data = DataInND(T1)

# at the end the following display the data in
if Nd == 3: # let's plot if it si viewable
    CMe(ND_Data)
```

Let's move on to higher dimensions

Now, let's step by step move on to higher dimensions.

First we will look at the no noise case and increase intensity

In [ ]:
```
# using your function let's create data in Nd
ND_Data = DataInND(T1, Nd = 10) # note that no noise is added

# Your code starts here
# starting with ND_Data your code should project the text in T1 to a plan
# extract the X, Y coordinates
X = None
Y = None


# Your code ends here and X and Y cooridantes of the

# Plot the result
plt.plot(X,Y, 'r*')
```

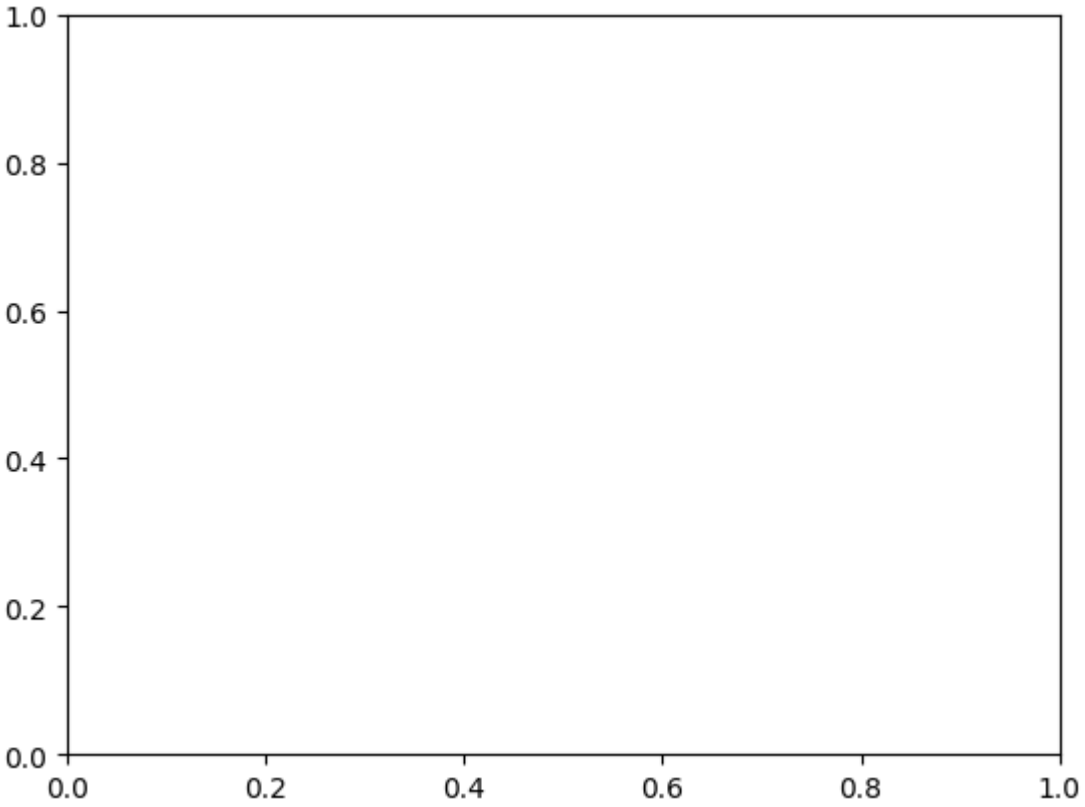
and there comes the noise

In [14]:
```
# using your function let's create data in Nd but this time noise will be
# try differen values of noise, decrease, increase, run this cell many ti
```

```
# get a good understanding of what happens then answer the questions belo
ND_Data = DataInND(T1, Nd = 10, NoiseLevel=1.1) # note that no noise is a

# Your code starts here
# Despite the contamination in ND_Data
# your code should project the text in T1 to a plane
# and extract the X, Y coordinates
X = None
Y = None


# Your code ends here and X and Y cooridantes of the

# Plot the result in 2D
plt.plot(X,Y, 'r*')
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[14], line 17
     11 Y = None
     14 # Your code ends here and X and Y cooridantes of the
     15
     16 # Plot the result in 2D
---> 17 plt.plot(X,Y, 'r*')

File ~/MECH536/.venv/lib/python3.10/site-packages/matplotlib/pyplot.py:379
4, in plot(scalex, scaley, data, *args, **kwargs)
   3786 @_copy_docstring_and_deprecators(Axes.plot)
   3787 def plot(
   3788     *args: float | ArrayLike | str,
   (...)
   3792     **kwargs,
   3793 ) -> list[Line2D]:
-> 3794     return gca().plot(
   3795         *args,
   3796         scalex=scalex,
   3797         scaley=scaley,
   3798         **({"data": data} if data is not None else {}),
   3799         **kwargs,
   3800     )

File ~/MECH536/.venv/lib/python3.10/site-packages/matplotlib/axes/_axes.p
y:1779, in Axes.plot(self, scalex, scaley, data, *args, **kwargs)
   1536 """
   1537 Plot y versus x as lines and/or markers.
   1538
   (...)
   1776 (``'green'``) or hex strings (``'#008000'``).
   1777 """
   1778 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1779 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1780 for line in lines:
   1781     self.add_line(line)

File ~/MECH536/.venv/lib/python3.10/site-packages/matplotlib/axes/_base.p
y:296, in _process_plot_var_args.__call__(self, axes, data, *args, **kwarg
s)
    294     this += args[0],
    295     args = args[1:]
--> 296 yield from self._plot_args(
    297     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datake
y)

File ~/MECH536/.venv/lib/python3.10/site-packages/matplotlib/axes/_base.p
y:447, in _process_plot_var_args._plot_args(self, axes, tup, kwargs, retur
n_kwargs, ambiguous_fmt_datakey)
    444 # Don't allow any None value; these would be up-converted to one
    445 # element array of None which causes problems downstream.
    446 if any(v is None for v in tup):
--> 447     raise ValueError("x, y, and format string must not be None")
    449 kw = {}
    450 for prop_name, val in zip(('linestyle', 'marker', 'color'),
    451                           (linestyle, marker, color)):
```

```
ValueError: x, y, and format string must not be None
```



Explain - 2:

Both clear and noisy data should be handled in a very similar way, simply by using the first 2 left singular vectors and so on...
However, data projected in 2D in both cases are they exactly the same?
What is the difference? What can you say about noise filtering achieved via SVD in this case? Briefly explain.

Let's fancy

Now write another function

In [12]:
```
def TwoOrthStrInND(T1, T2, Nd = 10, NoiseLevel = 0.0, NoiseType = 'Normal
    '''
    Accepts 2 data matrices of shape 2xK, preferably that comes from read
    Let T1, T2 be two matrices of shape 2xK
    This function should project T1 and T2 to seperate and orthogonal pla
    and return Res.
    If we project points in Res on to the plane that is spanned by
    the first 2 lef-singular vectors T1 should be readable on the plot
    Similarly, if we project points in the result on to the plane that is
    the following 2 lef-singular vectors (i.e. the third and the fourth)

    NoiseLevel and NoiseType is just like before, i.e.:
    Noise level generates a random matrix that is also NdxK and this matr
    after being multiplied by NoiseLevel
    Random matrix is generated using randn() if NoiseType is 'Normal' els
    '''
    Res = None
```

```
    return Res
```

Let's try

Let's recall that SVD of matrix $\mathbf{M}$ is:

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

here let u_1 be the first column of $\mathbf{U}$, and in general $\mathbf{u}_i$ is the i^{th} column of $\mathbf{U}$, or in other words $\mathbf{u}_i$ is the i^{th} left-singular vector of $\mathbf{M}$.

If we project the outcome of `TwoOrthStrInNd` to the plane that is spanned by $\mathbf{u}_1, \mathbf{u}_2$, and plotted in 2D the text in `T1` should be readable. `T2` should not be visible even as a line. A point at the origin is of course acceptalbe.

Similarly, if we project the outcome of `TwoOrthStrInNd` to the plane that is spanned by $\mathbf{u}_3, \mathbf{u}_4$, and plotted in 2D the text in `T2` should be readable. Evidently, `T1` should not be visible, other than a point at the origin.

In [ ]:
```
# CELL 3
# get the data in Nd
Nd_Data = TwoOrthStrInND(T1, T2, Nd = 10, NoiseLevel=0.0)
```

In [ ]:
```
# Your code starts here
# proejct all data point in Nd_Data to the plane spanned by u1,u2 first a
# let the coordinates of Nd_Data on this plane be X1, Y1
X1 = None
Y1 = None


# Your code ends here

# let's plot and see
plt.plot(X1, Y1, 'r*')
```

In [ ]:
```
# Your code starts here
# proejct all data point in Nd_Data to the plane spanned by u3,u4 first a
# let the coordinates of Nd_Data on this plane be X1, Y1
X2 = None
Y2 = None


# Your code ends here

# let's plot and see
plt.plot(X2, Y2, 'r*')
```

Explain - 3:

Going back to CELL 3, change noise level as you did before, observe what happens in your plots as noise gets heavier and explain what you see.

Even fancier: Consider this as bonus

In [ ]:
```
def TwoFacesOfAPrism(T1, T2, Nd = 10, NoiseLevel = 0.0, NoiseType = 'Norm
    '''
    Similar to TwoOrthStrInND,
    Accepts 2 data matrices of shape 2xK, preferably that comes from read
    Let T1, T2 be two matrices of shape 2xK
    This function should project T1 and T2 to seperate and orthogonal pla
    and return the resulting matrix: Res
    However, this time it is a bit different:
    If we reconstruct a rank-3 approximation of the result and use the co
    with respoect to the first 3 columns of the U matrix that is in rank-
    this will result in a 3xK matrix and ploting this matrix using CMe()
    T1 and T2 should be on orthgonal planes
    i.e. if we rotate the plot so that T1 is readable, T2 should not be v
    When this scene is propelry rotated by 90 degrees, T2 should be reada

    NoiseLevel and NoiseType is just like before, i.e.:
    Noise level generates a random matrix that is also NdxK and this matr
    after being multiplied by NoiseLevel
    Random matrix is generated using randn() if NoiseType is 'Normal' els

    returns Res, i.e. a matrix of shape NdxK
    '''
    Res = None



    return Res
```

Let's try

In this case lets assume that $\tilde{\mathbf{M}}_3$ is the rank-3 approximation of $\mathbf{M}$.

Given that: $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, then

$$\tilde{\mathbf{M}}_3 = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \mathbf{C}$$

Recall that $\mathbf{u}_i, \mathbf{v}_j$ are column vectors and columns of $\mathbf{C}$ contains the coordinates of points in $\mathbf{M}$ when they are projected down to $3D$.

Therefore, finding $\mathbf{C}$ and plotting as follows:
`CMe(C)`
should provide the plot.
When you rotate this plot so that S1 is readable, S2 should not be visible, and when you rotate the view properly for $90$$ degress, S2 should be readable, but not S1 .
This is as if you are vertically looking onto two neigbouring faces of a rectangular prism, and seeing different strings on each face.

Check out the demo cell below. If you look at XY plane vertically, you should read

FACE , if you rotate it 90 degrees and look at XZ plane, you see just some random points. If you implement this part properly, on XY plane you should read S1 and on XZ plane S2 should be readable.

However, note that this implies more that just printing on the face of a prism, which is not the answer to this question. Becasue if you do so, when S1 is printed, S2 will show up as a line.

What is required is, when we look at the in the direction to see S1 , S2 should not leave a mark on the scened, and vice versa in the no noise case. Of course noise will contaminate the scene and effect the result.

In [ ]:
```
# demo cell
Tdemo, _ = text2mat('FACE')
Tdemo = Tdemo + np.random.randn(*Tdemo.shape)*0.01
CMe(Tdemo)
```

In [ ]:
```
# CELL 4
# get the data in Nd
Nd_Data = TwoFacesOfAPrism(T1, T2, Nd = 10, NoiseLevel=0.0)
```

In [ ]:
```
# Your code starts here
# proejct all data point in Nd_Data to the plane spanned by u1,u2 first a
# let the coordinates of Nd_Data on this plane be X1, Y1
X = None
Y = None
Z = None


# Your code ends here

# let's plot and see
plt.plot(X1, Y1, 'r*')
```

Explain - 4:

Going back to CELL 4, change noise level as you did before, observe what happens in your plots as noise gets heavier and explain what you see.