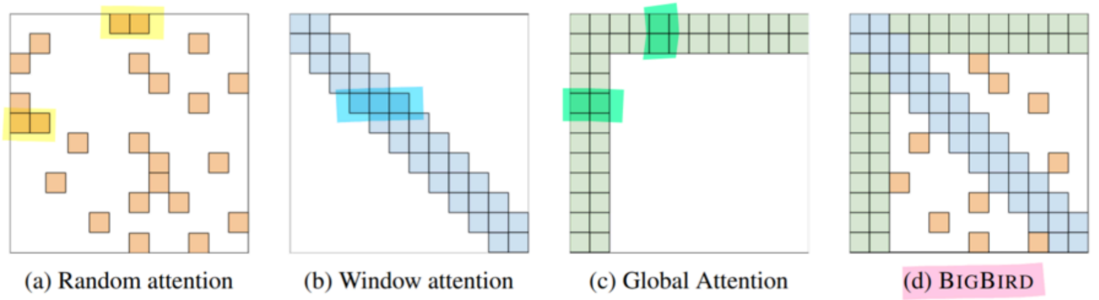
Previous work



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Introduction

My goal was to generate python functions, using only math, numpy and pure python, with a algorithm approche, simplicity and easy to port to C. For each attention mask (a, b, c and d), i have written the following features :

- generate a boolean mask, using the corresponding parameter
- same, but based on a given sparsity (by having a function : given sparsity -> corresponding parameter)
- generate artificial matrix of only ones and zero based on the mask
- a test function which shows how to use it and the output

Code

Imports

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import math # comment
```

UTILS

In [2]:

```
def get_nb_non_zero(matrix):
    return np.count_nonzero(matrix)
```

In [3]:

```
def get_density(matrix, length):
    return float(get_nb_non_zero(matrix)) / float(length * length)
```

In [4]:

```
def get_sparsity(matrix, length):
    return 1.0 - get_density(matrix, length)
```

In [5]:

```
def show_matrix_infos(matrix, length, given_sparsity = -1.0): # -1.0 correspond to None.
    # conditions : shape(matrix) = (length, length)
    real_sparsity = get_sparsity(matrix, length)
    given_text = "None" if given_sparsity < 0 else f"{given_sparsity:.2f}"
    text = f"Length: {length}, Given Sparsity: {given_text}, Real Sparsity: {real_sparsity:.4f}"
    plt.title(label=text)
    plt.imshow(matrix, cmap='gray', interpolation='nearest', vmin=0, vmax=1)
    plt.colorbar()
    plt.show()
```

RANDOM ATTENTION

By number of non-zeros per row

In [6]:

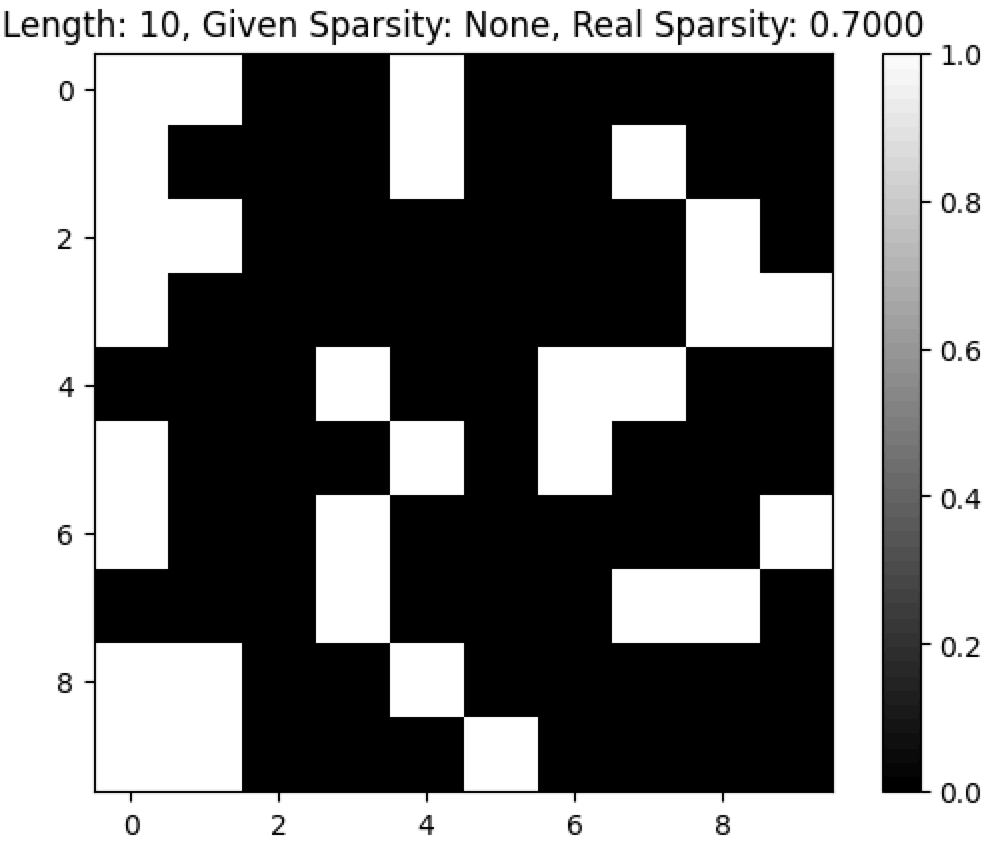
```
def get_random_attention_mask(length, nz_per_row):
    # conditions : nz_per_row <= length
    rng = np.random.default_rng()
    mask = rng.multivariate_hypergeometric([1]*length, nz_per_row, size=length).astype(bool)
    return mask
```

In [7]:

```
def generate_matrix_with_random_attention_mask(length, nz_per_row):
    # conditions : nz_per_row <= length
    matrix = np.ones((length, length))
    mask = get_random_attention_mask( length=length, nz_per_row=nz_per_row)
    matrix[~mask] = 0
    return matrix
```

In [8]:

```
def test_generate_matrix_with_random_attention_mask():
    length = 10
    nz_per_row = 3
    matrix = generate_matrix_with_random_attention_mask(length=length, nz_per_row=nz_per_row)
    show_matrix_infos(matrix=matrix, length=length)

test_generate_matrix_with_random_attention_mask()
```



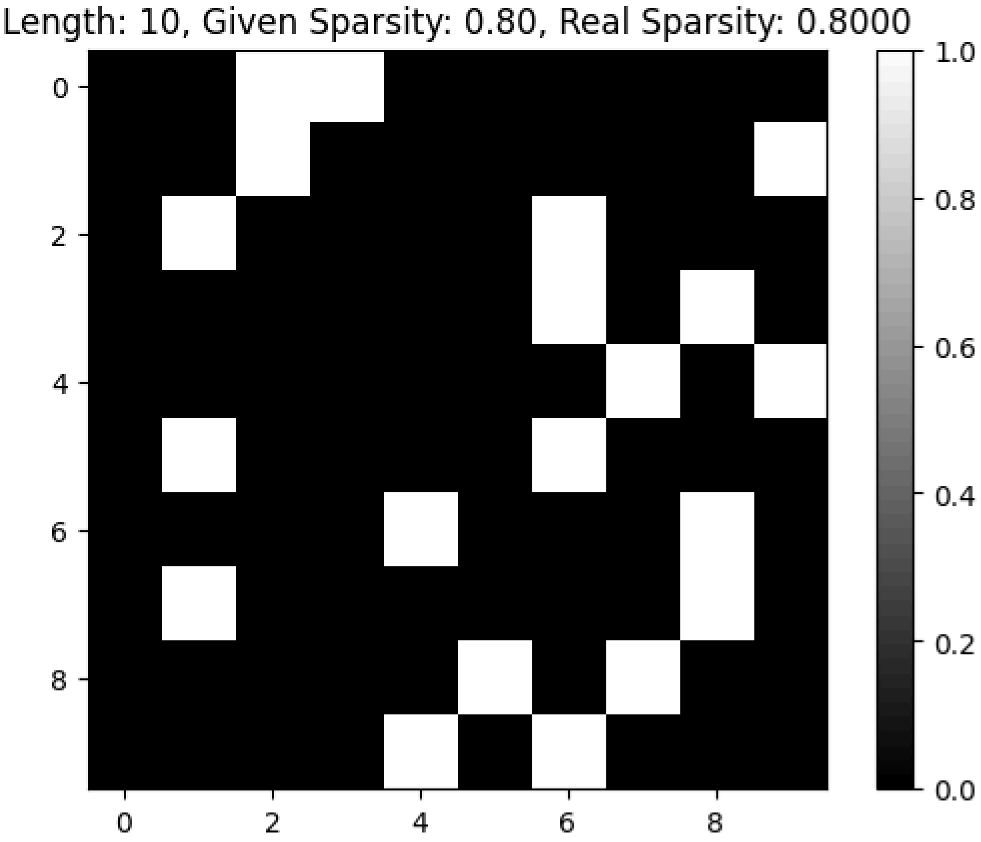
By sparsity

In [9]:
```
def best_nz_per_row_from_sparsity(length, sparsity):
    # conditions : 0 <= sparsity <= 1
    return round(length * (1 - sparsity))
```

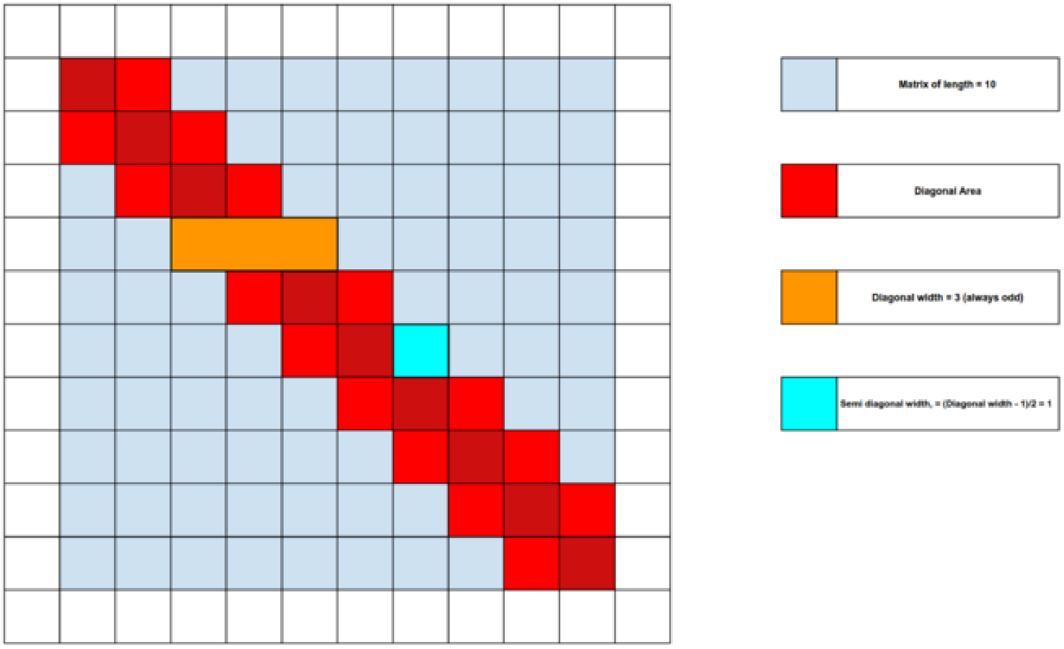
In [10]:
```
def get_random_attention_mask_with_sparsity(length, sparsity):
    # conditions : 0 <= sparsity <= 1
    nz_per_row=best_nz_per_row_from_sparsity(length=length, sparsity=sparsity)
    return get_random_attention_mask( length=length, nz_per_row=nz_per_row)
```

In [11]:
```
def generate_matrix_with_random_attention_mask_with_sparsity(length, sparsity):
    # conditions : 0 <= sparsity <= 1
    matrix = np.ones((length, length))
    mask = get_random_attention_mask_with_sparsity(length=length, sparsity= sparsity)
    matrix[~mask] = 0
    return matrix
```

In [12]:
```
def test_generate_matrix_with_random_attention_mask_with_sparsity():
    length = 10
    sparsity = 0.8
    matrix = generate_matrix_with_random_attention_mask_with_sparsity(length=length, sparsity=sparsity)
    show_matrix_infos(matrix=matrix, length=length, given_sparsity=sparsity)

test_generate_matrix_with_random_attention_mask_with_sparsity()
```



WINDOW ATTENTION


Matrix of length = 10
Diagonal Area
Diagonal width = 3 (always odd)
Semi diagonal width, = (Diagonal width - 1)/2 = 1

Utils

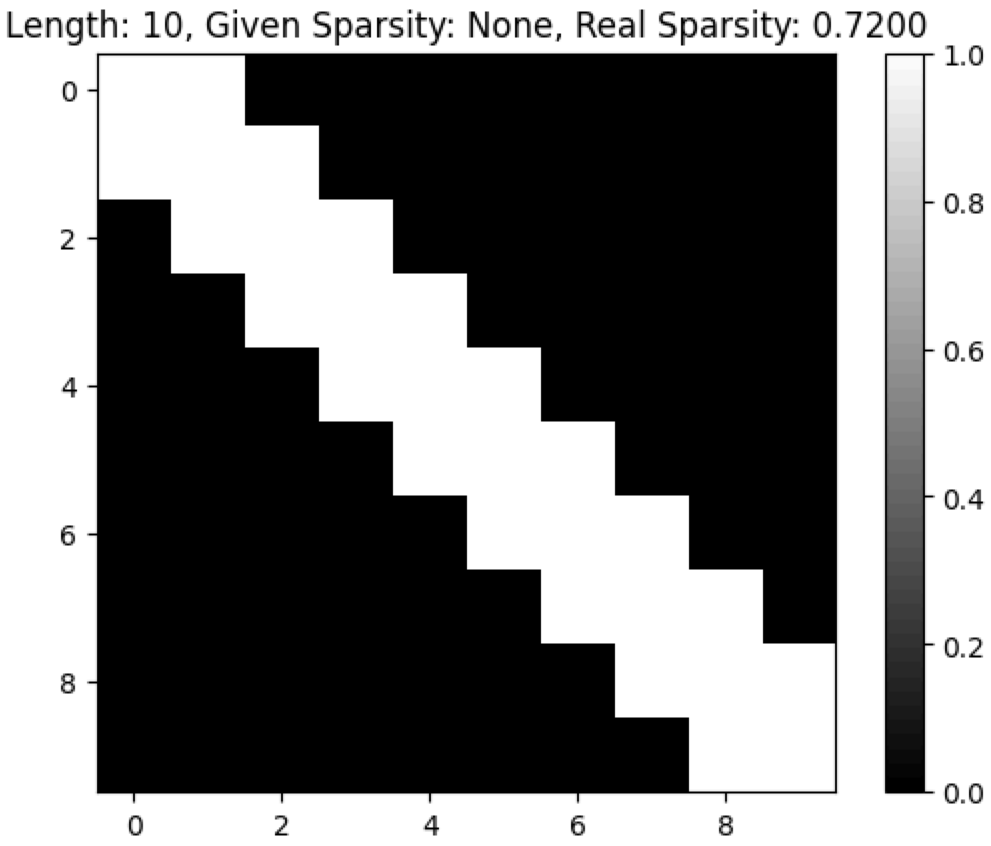
In [13]:
```python
def diagonal_area(length,diagonal_width):
    # conditions : length
    if(diagonal_width == 0):
        return 0
    else:
        n = length
        #semi diagonal widht
        sdw = diagonal_width // 2 # (diagonal_width / 2 - 1 because is odd)
        da = n * ( 1 + 2 * sdw ) - sdw * (sdw + 1)
        return da
```

By diagonal width

In [14]:
```python
def get_window_attention_mask (length, diagonal_width):
    # conditions : shape(matrix) = (length, length), 0 <= diagonal_width <= 2*length - 1 (cover full matrix), diagonal_widt
    mask = np.zeros(shape=(length, length), dtype=bool)
    if (diagonal_width > 0):
        sdw = diagonal_width // 2
        if diagonal_width == 1:
            mask = np.fromfunction(lambda i, j:  j == i,shape=(length, length), dtype=int)
        else :
            mask = np.fromfunction(lambda i, j:  np.abs(i - j) <= sdw ,shape=(length, length), dtype=int)
    return mask
```

In [15]:
```python
def generate_matrix_with_window_attention_mask(length, diagonal_width):
    # conditions :  0 <= diagonal_width <= 2*length - 1 (cover full matrix), diagonal_width is odd
    matrix = np.ones((length, length))
    mask = get_window_attention_mask( length= length, diagonal_width=diagonal_width)
    matrix[~mask] = 0
    return matrix
```

In [16]:
```python
def test_generate_matrix_with_window_attention_mask():
    length = 10
    diagonal_width = 3
    matrix = generate_matrix_with_window_attention_mask(length=length, diagonal_width=diagonal_width)
    show_matrix_infos(matrix,length)

test_generate_matrix_with_window_attention_mask()
```



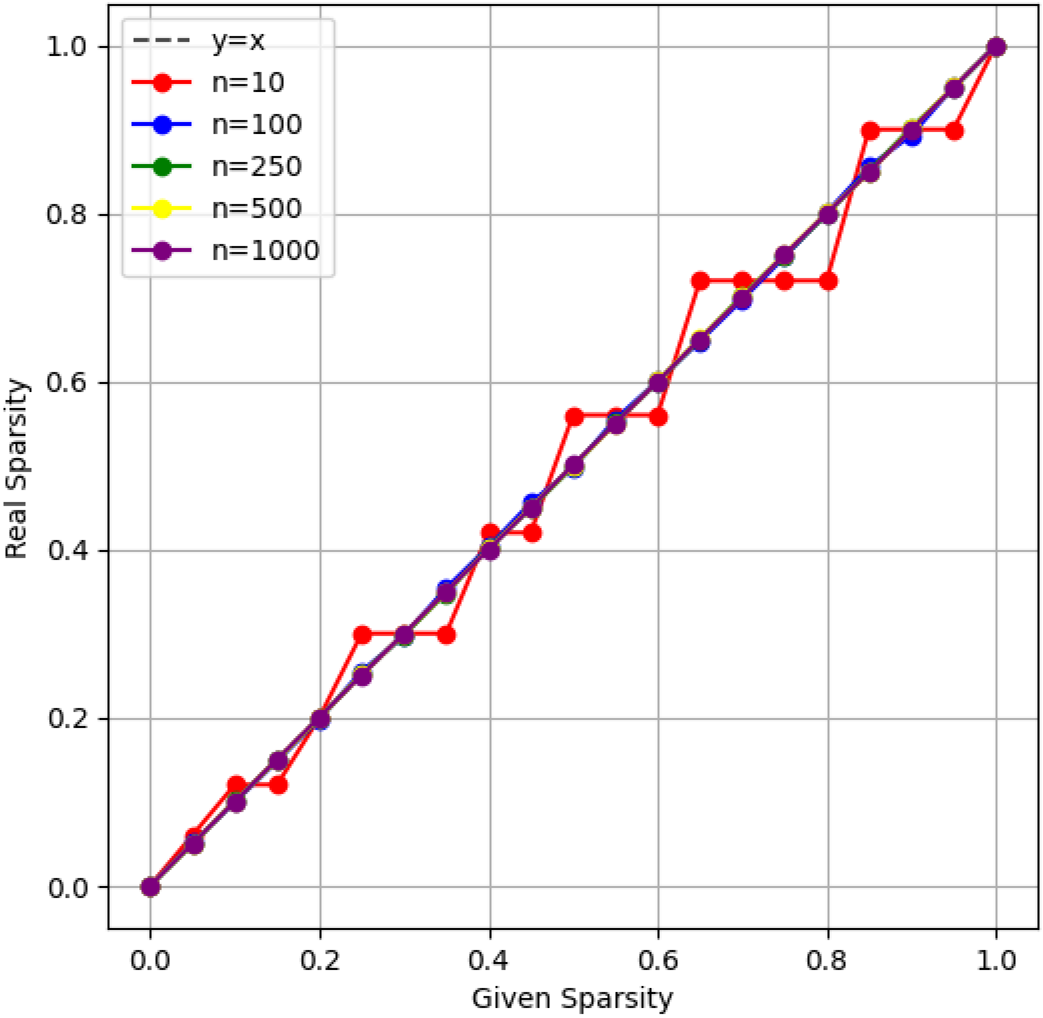
By sparsity

In [17]:

```
def best_diagonal_width_from_sparsity(length, sparsity):
    n = length
    density = 1.0 - sparsity
     # ideal diagonal aera
    da = n * n * density
    # from this point, all is explained in the related document
    a = -1
    b = 2 * n - 1
    c = n - da
    det = b * b - 4 * a * c
    x = (-b + math.sqrt(det))/(2 * a)

    sdw = round(x)

    dw = 2 * sdw + 1

    if(dw < 0) : dw = 0
    elif(dw > 2*n - 1): dw = 2*n - 1
    # print(f"For matrix of size: {n} and given sparsity: {sparsity}, ideal semi diagonal width is : {x}, chosen dw is {dw}
    return dw
```

In [19]:

```
# test function hided for render
test_diagonal_width_from_sparsity()
```



In [20]:

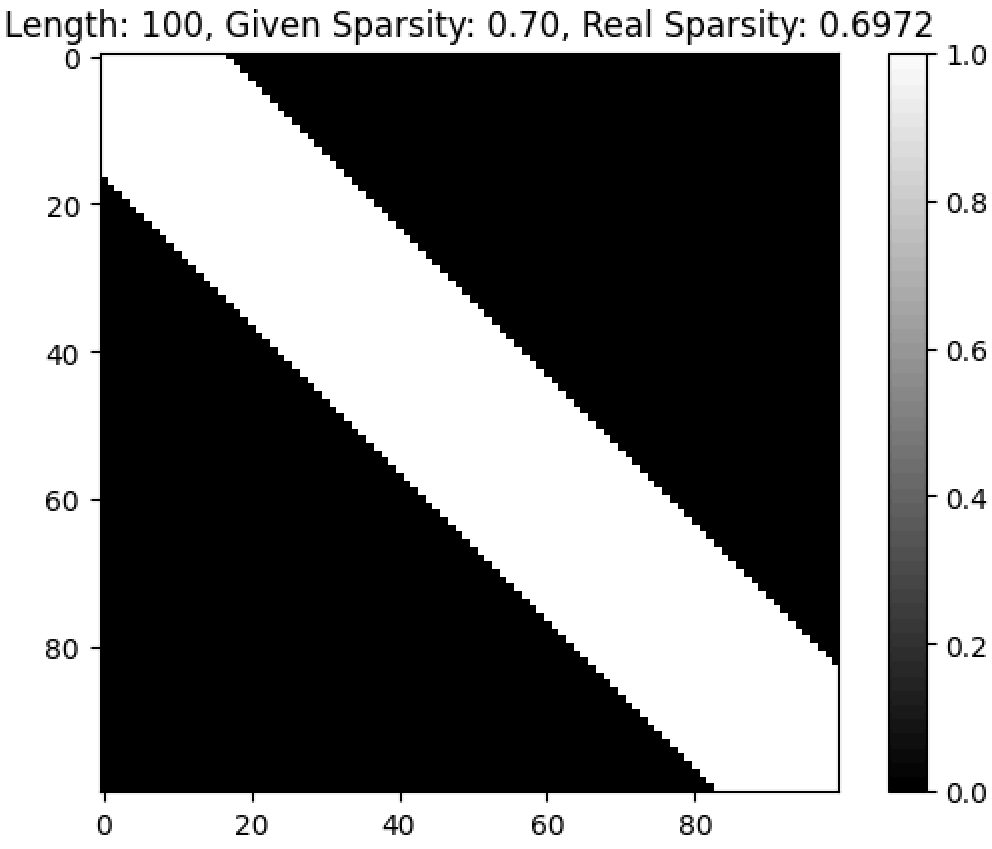
```
def get_window_attention_mask_with_sparsity( length, sparsity):
    # conditions : 0 <= sparsity <= 1
    dw = best_diagonal_width_from_sparsity(length, sparsity)
    return get_window_attention_mask( length=length, diagonal_width=dw)
```

In [21]:

```
def generate_matrix_with_window_attention_mask_with_sparsity(length, sparsity):
    # conditions :  0 <= diagonal_width <= 2*length - 1 (cover full matrix), diagonal_width is odd
    matrix = np.ones((length, length))
    mask = get_window_attention_mask_with_sparsity( length= length, sparsity=sparsity)
    matrix[~mask] = 0
    return matrix
```

In [22]:

```
def test_generate_matrix_with_window_attention_mask_with_sparsity():
    length = 100
    sparsity = 0.7
    matrix = generate_matrix_with_window_attention_mask_with_sparsity(length=length, sparsity=sparsity)
    show_matrix_infos(matrix,length, sparsity)

test_generate_matrix_with_window_attention_mask_with_sparsity()
```



GLOBAL ATTENTION

Utils

In [23]:

```
def global_attention_aera(length,global_attention_width):
    w = global_attention_width
    n = length
    return (2 * w * n) - (w * w)

print(global_attention_aera(10,2))
```

```
36
```

In [24]:

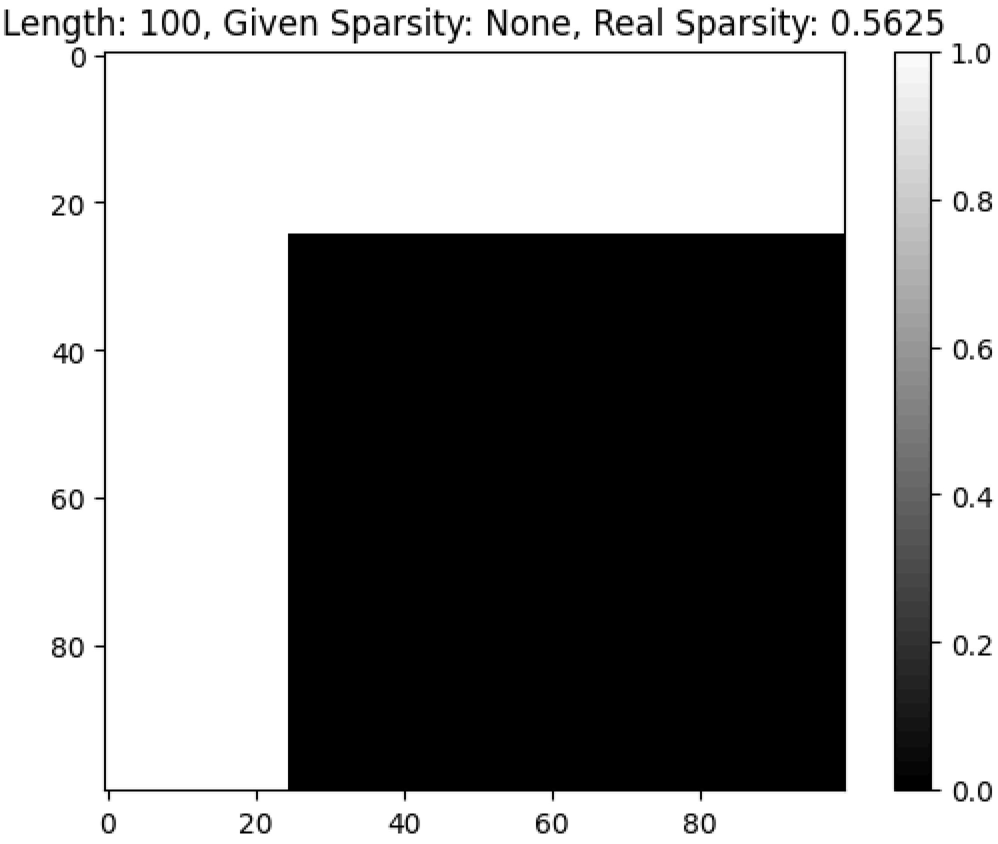
```
def get_global_attention_mask( length, global_width):
    mask = np.zeros(shape=(length,length), dtype=bool)
    mask[:global_width,:] = True
    mask[global_width : , : global_width] = True
    return mask
```

In [25]:

```
def generate_matrix_with_global_attention_mask(length, global_width):
    matrix = np.ones((length, length))
    mask = get_global_attention_mask( length=length, global_width=global_width)
    matrix[~mask] = 0
    return matrix
```

In [26]:

```
def test_generate_matrix_with_global_attention_mask():
    length = 100
    global_width = 25
    matrix = generate_matrix_with_global_attention_mask(length=length, global_width=global_width)
    show_matrix_infos(matrix=matrix, length=length)

test_generate_matrix_with_global_attention_mask()
```

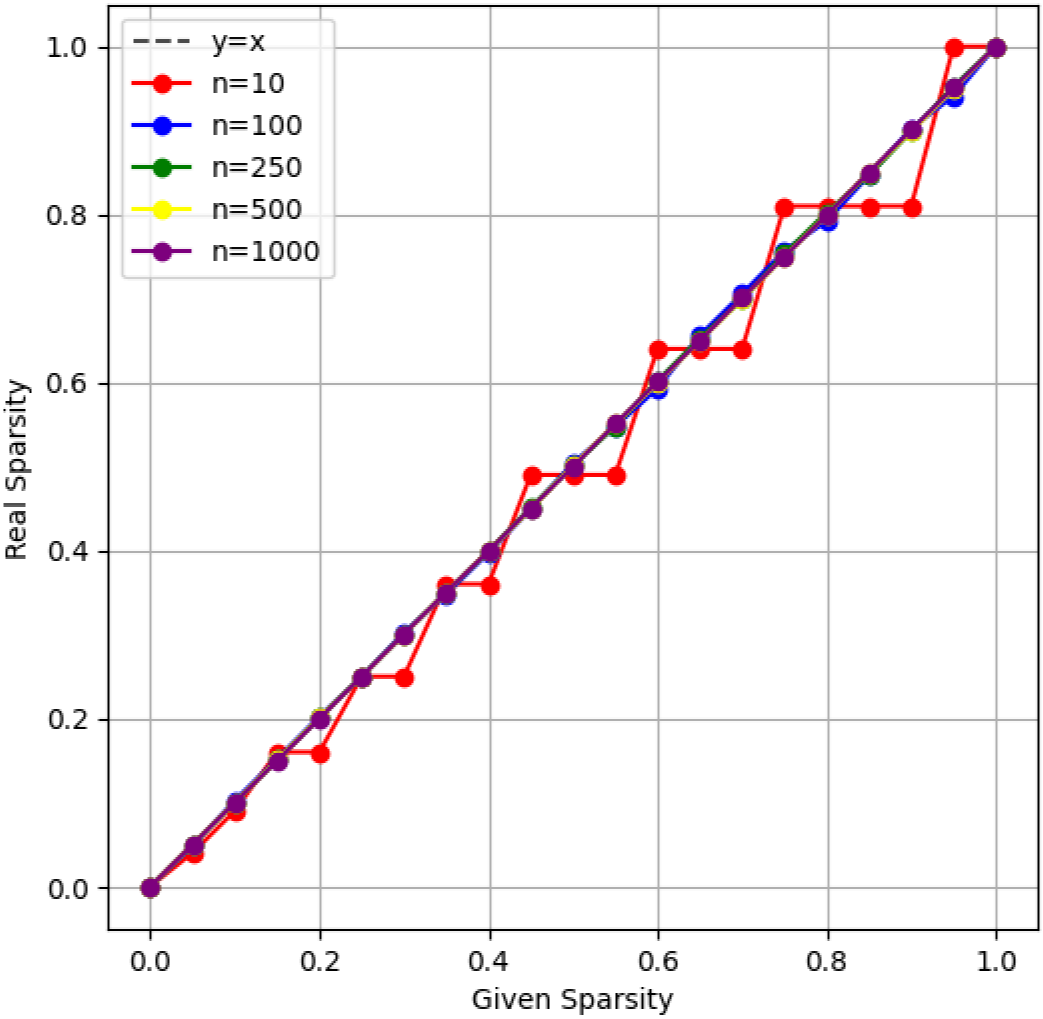


In [27]:

```
def best_global_width_from_sparsity(length, sparsity):
    n = length
    density = 1.0 - sparsity
     # ideal diagonal aera
    ga = n * n * density
    # same as window mask but easier
    a = -1
    b = 2 * n
    c = - ga
    det = b * b - 4 * a * c
    x = (-b + math.sqrt(det))/(2 * a)
    gw = round(x)
    if(gw < 0) : gw = 0
    elif(gw > n * n ): gw = n * n
    return gw
```

In [29]:

```
# test function hided for render
test_global_width_from_sparsity()
```



In [30]:

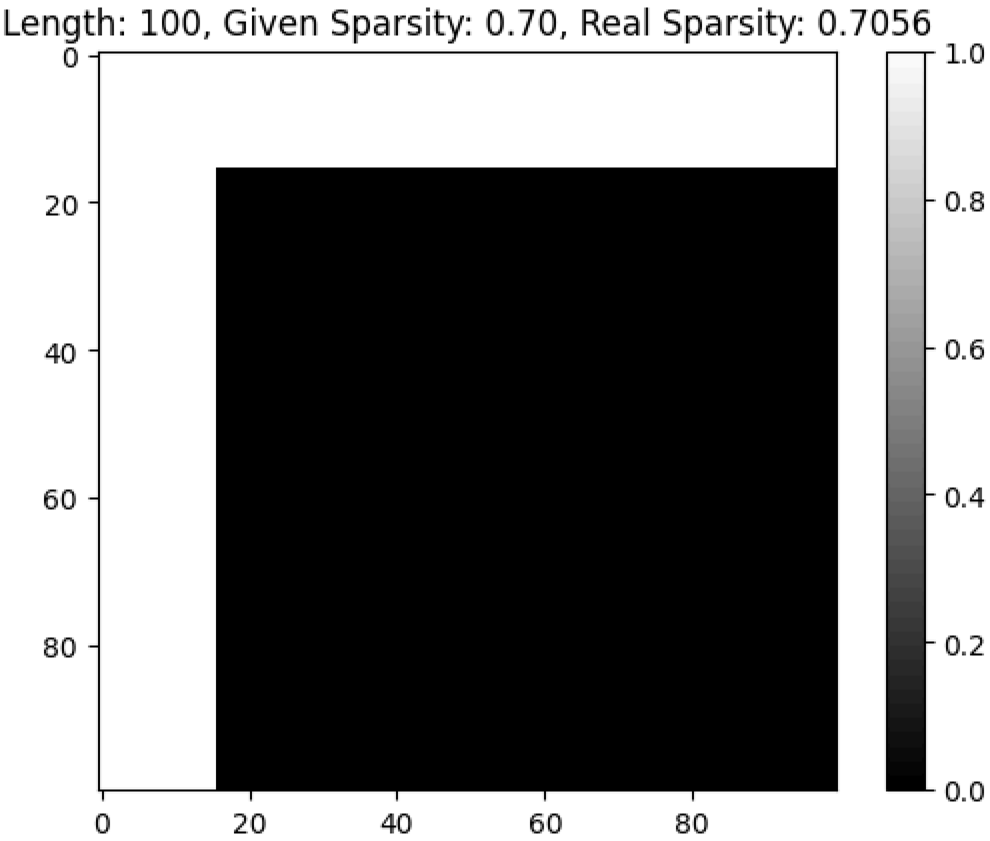
```
def get_global_attention_mask_with_sparsity( length, sparsity):
    # conditions : 0 <= sparsity <= 1
    gw = best_global_width_from_sparsity(length, sparsity)
    return get_global_attention_mask( length=length, global_width=gw)
```

In [31]:

```
def generate_matrix_with_global_attention_mask_with_sparsity(length, sparsity):
    matrix = np.ones((length, length))
    mask = get_global_attention_mask_with_sparsity(length=length, sparsity=sparsity)
    matrix[~mask] = 0
    return matrix
```

In [32]:

```
def test_generate_matrix_with_global_attention_mask_with_sparsity():
    length = 100
    sparsity = 0.7
    matrix = generate_matrix_with_global_attention_mask_with_sparsity(length=length, sparsity=sparsity)
    show_matrix_infos(matrix=matrix, length=length, given_sparsity= sparsity)

test_generate_matrix_with_global_attention_mask_with_sparsity()
```

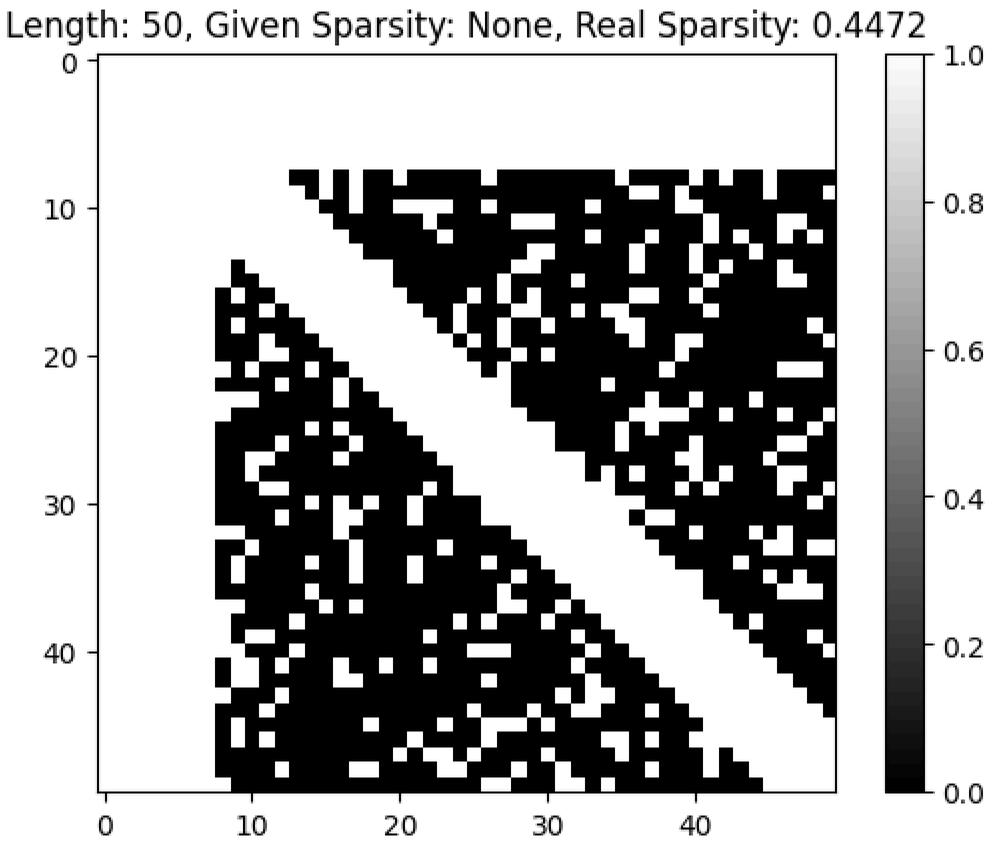

Length: 100, Given Sparsity: 0.70, Real Sparsity: 0.7056

BIG BIRD (combination of all above)

In [33]:
```
def get_big_bird_mask(length, nz_per_row, diagonal_width, global_width):
    am = get_random_attention_mask( length= length, nz_per_row=nz_per_row)
    wm = get_window_attention_mask( length= length, diagonal_width=diagonal_width)
    gm = get_global_attention_mask( length=length, global_width= global_width)
    total_mask = am | wm | gm
    return total_mask
```

In [34]:
```
def generate_big_bird(length, nz_per_row, diagonal_width, global_width ):
    matrix = np.ones((length, length))
    mask = get_big_bird_mask( length=length, nz_per_row= nz_per_row, diagonal_width=diagonal_width, global_width= global_wi
    matrix[~mask] = 0
    return matrix
```

In [35]:
```
def test_generate_big_bird():
    length = 50
    nz_per_row = 10
    diagonal_width = 8
    global_width = 8
    matrix = generate_big_bird(length=length,nz_per_row=nz_per_row, diagonal_width=diagonal_width, global_width=global_widt
    show_matrix_infos(matrix=matrix, length= length)

test_generate_big_bird()
```



In [36]:

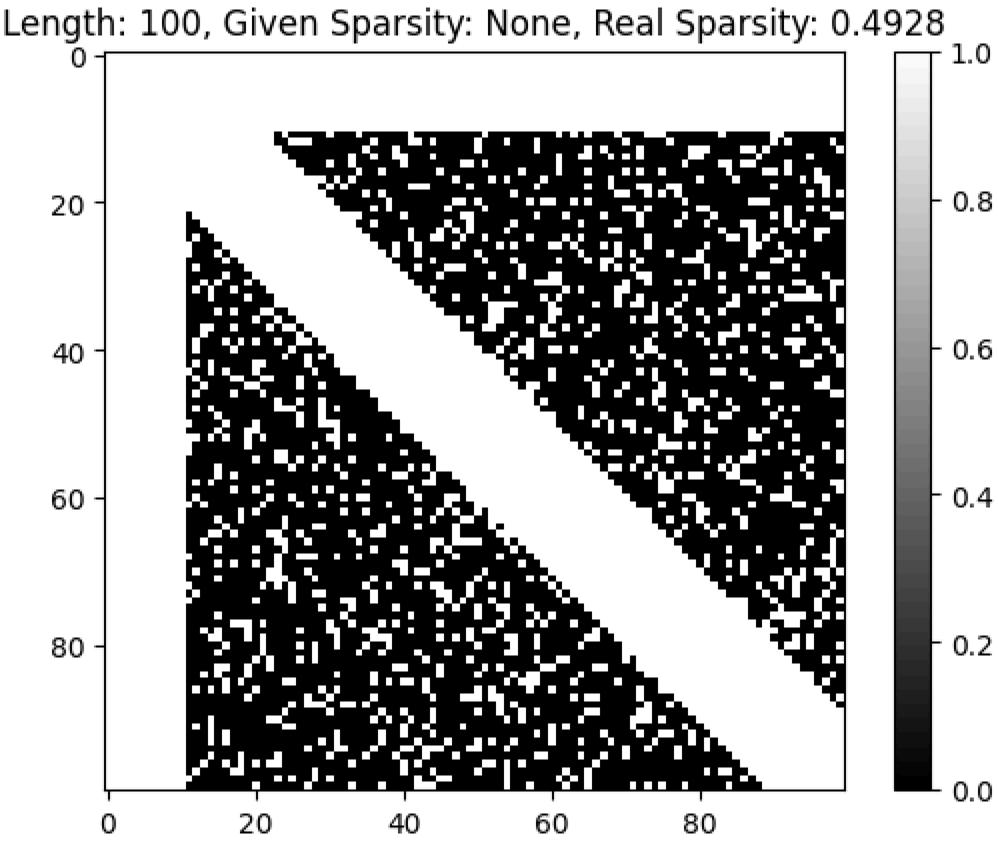
```
def get_big_bird_mask_with_sparsity( length, random_sparsity, window_sparsity, global_sparsity):
    am = get_random_attention_mask_with_sparsity( length= length, sparsity=random_sparsity)
    wm = get_window_attention_mask_with_sparsity( length= length,sparsity=window_sparsity )
    gm = get_global_attention_mask_with_sparsity(length=length, sparsity=global_sparsity)
    total_mask = am | wm | gm
    return total_mask
```

In [37]:

```
def generate_big_bird_with_sparsity(length, random_sparsity, window_sparsity, global_sparsity):
    matrix = np.ones((length, length))
    mask = get_big_bird_mask_with_sparsity(length, random_sparsity, window_sparsity, global_sparsity)
    matrix[~mask] = 0
    return matrix
```

In [38]:

```
def test_generate_big_bird_with_sparsity():
    length = 100
    random_sparsity =  0.8
    window_sparsity = 0.8
    global_sparsity = 0.8
    matrix = generate_big_bird_with_sparsity(length=length,random_sparsity=random_sparsity, window_sparsity=window_sparsity
    show_matrix_infos(matrix=matrix, length= length)

test_generate_big_bird_with_sparsity()
```



In [39]:
```
def adjust_total_sparsity(total_sparsity):
    x = total_sparsity
    # degree = 3
    # a = 2.61815675
    # b = -4.77052715
    # c = 2.98999146
    # d = 0.19945692
    # res =  a  * ( x ** 3 )  + b * (x ** 2) + c * x
    # degree = 5
    a = 24.08862473
    b = -65.2963488
    c = 64.48601296
    d = -28.42365239
    e = 5.98076684
    f = 0.17082526
    poly =  a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f
    res = min(max(poly, 0.0), 1.0)
    return res
```

In [40]:
```
def get_big_bird_mask_with_total_sparsity( length, total_sparsity, adjust):
    if adjust :
        total_sparsity = adjust_total_sparsity(total_sparsity)
    random_sparsity = total_sparsity
    window_sparsity = total_sparsity
    global_sparsity = total_sparsity
    total_mask = get_big_bird_mask_with_sparsity( length, random_sparsity, window_sparsity, global_sparsity)
    return total_mask
```

In [41]:
```
def generate_big_bird_with_total_sparsity(length,total_sparsity, adjust):
    matrix = np.ones((length, length))
    mask = get_big_bird_mask_with_total_sparsity(length, total_sparsity, adjust)
    matrix[~mask] = 0
    return matrix
```

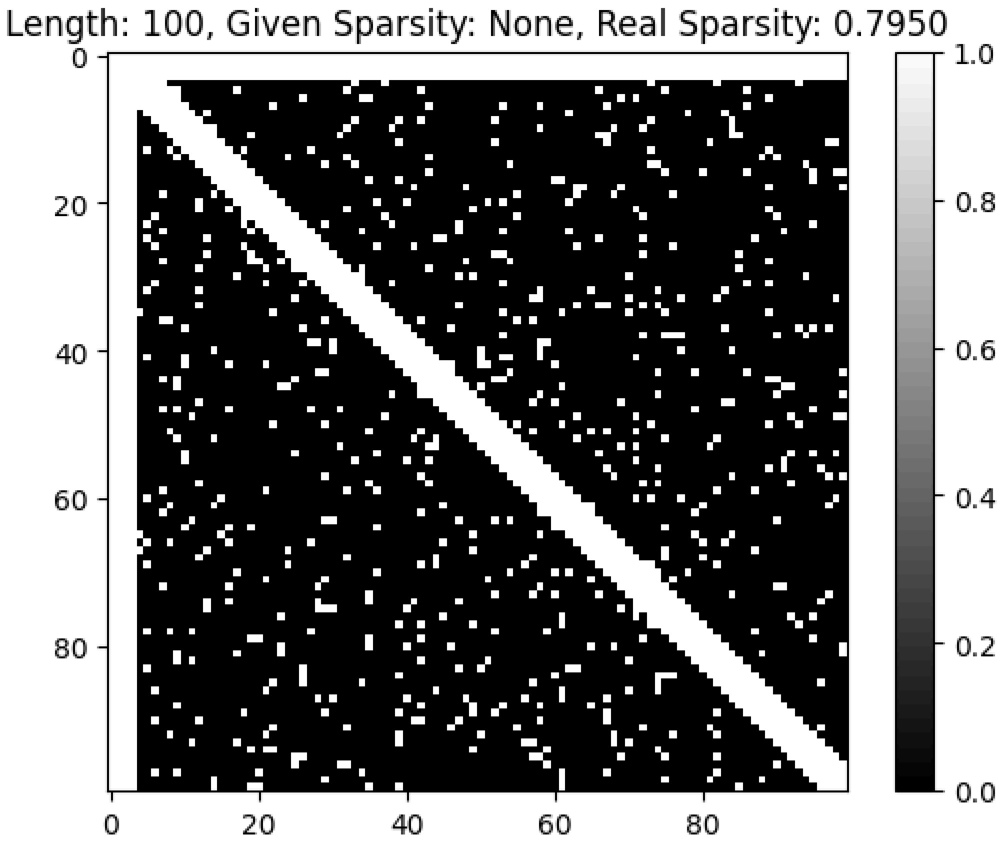
In [42]:
```
def find_approximation():
    sparsity_values = [x / 100.0 for x in range(0, 101, 5)]
    length = 2000

    given_sparsity = []
    real_sparsity = []

    for sparsity in sparsity_values:
        matrix = generate_big_bird_with_total_sparsity(length, sparsity, adjust=False)
        real_sp = get_sparsity(matrix, length)
        given_sparsity.append(sparsity)
        real_sparsity.append(real_sp)

    sort_idx = np.argsort(real_sparsity)
    real_sorted = np.array(real_sparsity)[sort_idx]
    given_sorted = np.array(given_sparsity)[sort_idx]

    coeffs = np.polyfit(real_sorted, given_sorted, 5)

    print(f"a = {coeffs[0]}")
    print(f"b = {coeffs[1]}")
    print(f"c = {coeffs[2]}")
    print(f"d = {coeffs[3]}")
    print(f"e = {coeffs[4]}")
    print(f"f = {coeffs[5]}")
    print(f"poly = a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f")

find_approximation()
```

```
a = 24.080837401839318
b = -65.27459395374147
c = 64.46369710621178
d = -28.413497112960528
e = 5.978905086888747
f = 0.17087493282757565
poly = a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f
```

In [43]:

```
def test_generate_big_bird_with_total_sparsity():
    length = 100
    total_sparsity = 0.8
    adjust = True
    matrix = generate_big_bird_with_total_sparsity(length, total_sparsity, adjust)
    print(f"Given sparsity = {total_sparsity}")
    show_matrix_infos(matrix, length)

test_generate_big_bird_with_total_sparsity()
```

```
Given sparsity = 0.8
```

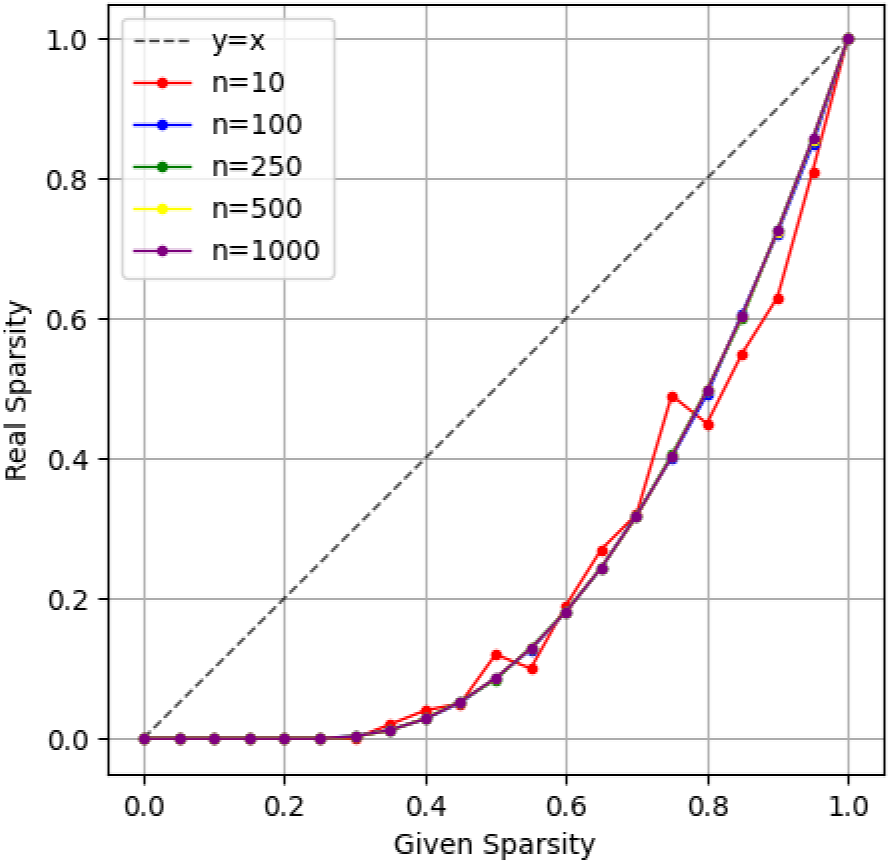

Length: 100, Given Sparsity: None, Real Sparsity: 0.7950

In [45]:

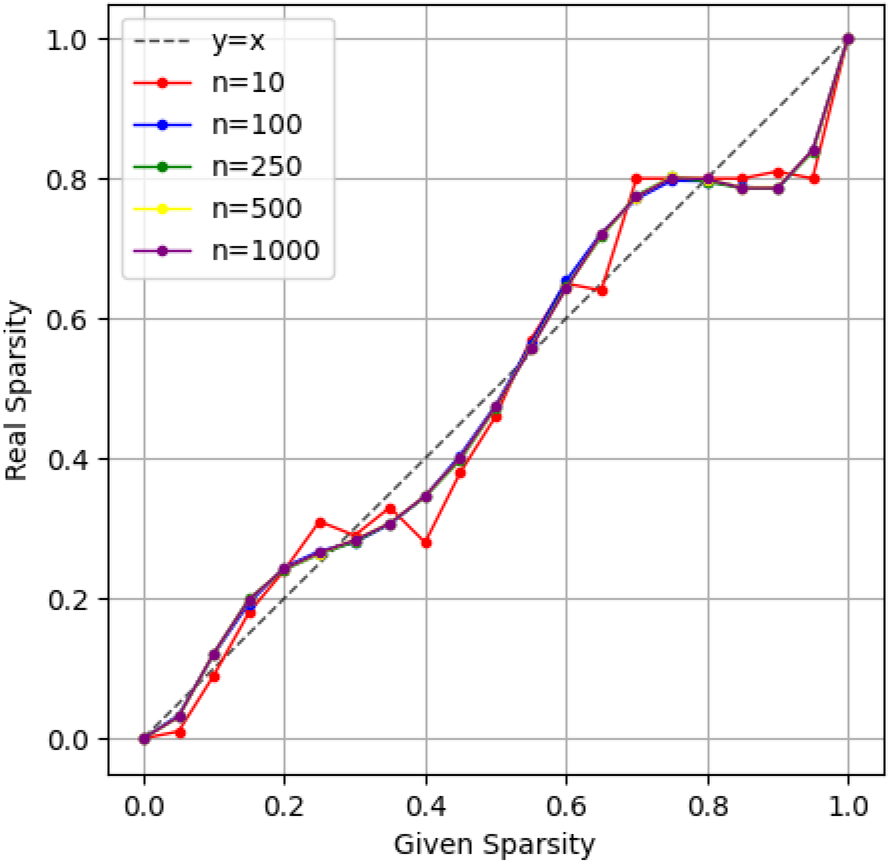
```
# def test_adjust_total_sparsity(adjust):
# ... (hided for rendering)

print("Without adjusting :")
```

```
test_adjust_total_sparsity(False)
print("When adjusting the given sparsity : ")
test_adjust_total_sparsity(True)
```

```
Without adjusting :
```



```
When adjusting the given sparsity :
```


y=x
n=10
n=100
n=250
n=500
n=1000
Real Sparsity
0.0
0.2
0.4
0.6
0.8
1.0
Given Sparsity
0.0
0.2
0.4
0.6
0.8
1.0