In this notebook, we will investigate the characteristics of npn high voltage from ST technology

---

$$ Specification $$ Model name: NPN high voltage

$E_l = 1 \mu m$

$E_w = 0.27 \mu m$

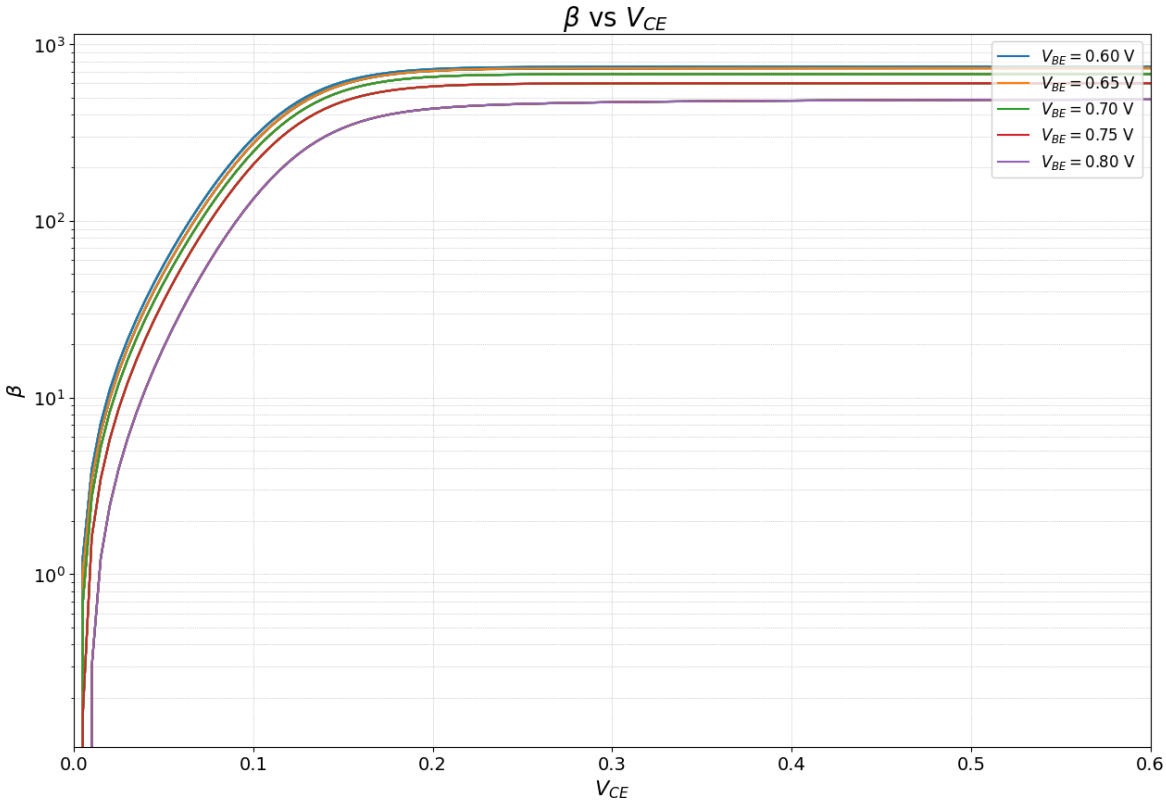
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd
```

1. keep V_{BE}, sweep V_{CE}

In [2]:
```
# Read the CSV file into a pandas DataFrame
data = pd.read_csv('gummel_beta_vs_VCE.csv', skiprows=1, header=None) # N
```

In [3]:
```
VBE = [0.6, 0.65, 0.7, 0.75, 0.8]
n = np.linspace(1, 10, 10)
count = 1
count = int(count)

plt.figure(figsize=(15, 10))  # Create the figure only once
plt.title(r' $\beta$ vs $V_{CE}$',fontsize = 20)
for V in VBE:
    try:
        plt.semilogy(data[0], data[count], label=f'$V_{{BE}} = {V:.2f}$ V

        for i in range(count, count + 10):
            plt.semilogy(data[0], data[i])

    except IndexError:
        print(f"Error: Index {count} or {i} out of range for data")
        break

    count = count + 10

plt.xlabel(r'$V_{CE} $', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
plt.xlim(0,0.6)
```

Out[3]:
```
(0.0, 0.6)
```



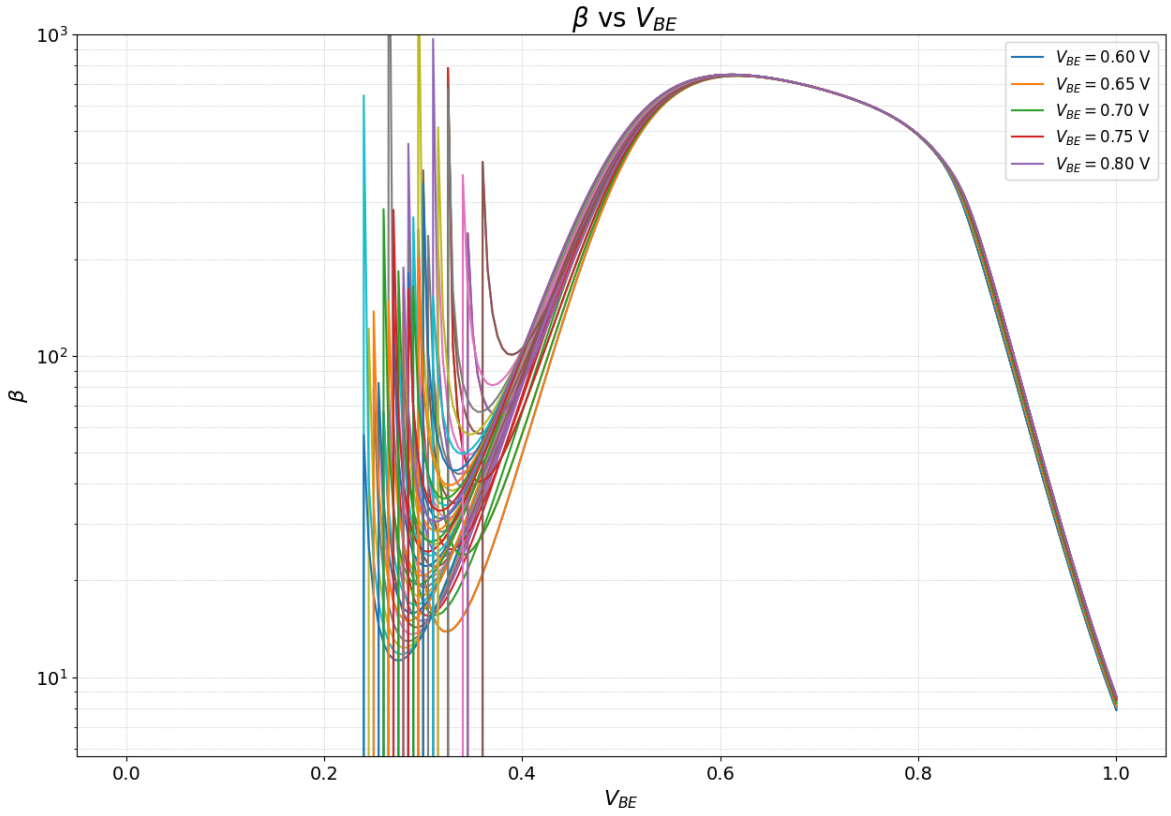
2. keep V_{CE}, sweep V_{BE}

In [4]:
```
# Read the CSV file into a pandas DataFrame
data2 = pd.read_csv('gummel_beta_vs_VBE.csv', skiprows=1, header=None) #
```

In [5]:
```
VCE = [0.6, 0.65, 0.7, 0.75, 0.8]
n = np.linspace(1, 10, 10)
count = 1
count = int(count)

plt.figure(figsize=(15, 10))  # Create the figure only once
plt.title(r' $\beta$ vs $V_{BE}$',fontsize = 20)
for V in VCE:
    try:
        plt.semilogy(data2[0], data2[count], label=f'$V_{{BE}} = {V:.2f}$

        for i in range(count, count + 10):
            plt.semilogy(data2[0], data2[i])

    except IndexError:
        print(f"Error: Index {count} or {i} out of range for data")
        break

    count = count + 10

plt.xlabel(r'$V_{BE} $', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
plt.ylim(0,1e3)
```

```
/var/folders/v4/bwhrmml56zdgg4rlt5zykqp00000gp/T/ipykernel_25872/110730034
5.py:27: UserWarning: Attempt to set non-positive ylim on a log-scaled axi
s will be ignored.
  plt.ylim(0,1e3)
```

Out[5]: (5.67106559144008, 1000.0)



3. Set $V_{BE} = V_{CE}$

In [6]:
```
data3 = pd.read_csv('IcIb_bellshape.csv', skiprows=1, header=None) # No h
```

In [7]:
```
#group of Ic
gr_Ic = np.arange(1,21,1)
Ic = data3[gr_Ic]

gr_Ib = np.arange(21,41,1)
print(gr_Ib)
Ib = data3[gr_Ib]
beta = []

for i in range(20):
    beta.append(data3[gr_Ic[i]] / data3[gr_Ib[i]])

VBE = [0.6,0.65,0.7,0.75,0.8]
n = np.linspace(1,20,20)
# plt.plot(beta)
# plt.show()
```

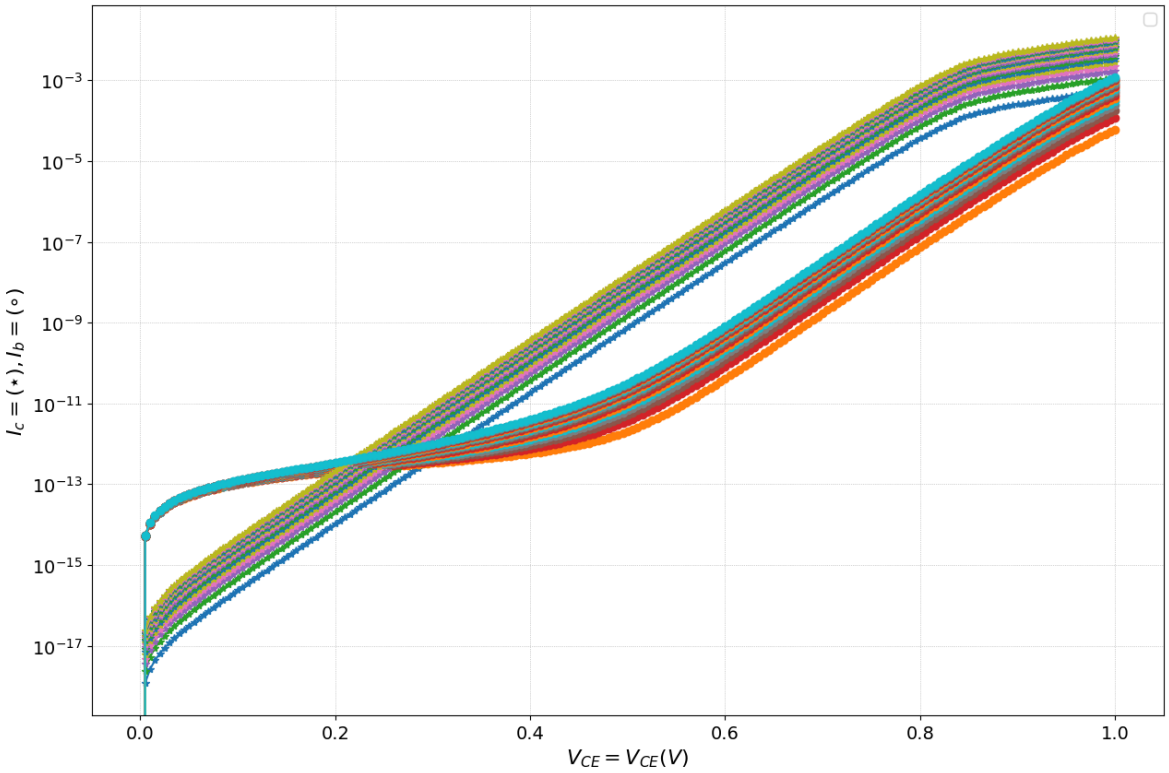
```
[21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40]
```

In [8]:
```
plt.figure(figsize = (15,10))
for i in range(20):
    plt.semilogy(data3[0], data3[gr_Ic[i]], marker = '*')#, label = f' m
    plt.semilogy(data3[0], data3[gr_Ib[i]], marker = 'o')
```

```
plt.ylabel(r'$I_c = (\star),  I_b = (\circ$)', fontsize=16)
plt.xlabel(r'$V_{CE} = V_{CE} (V)$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
```

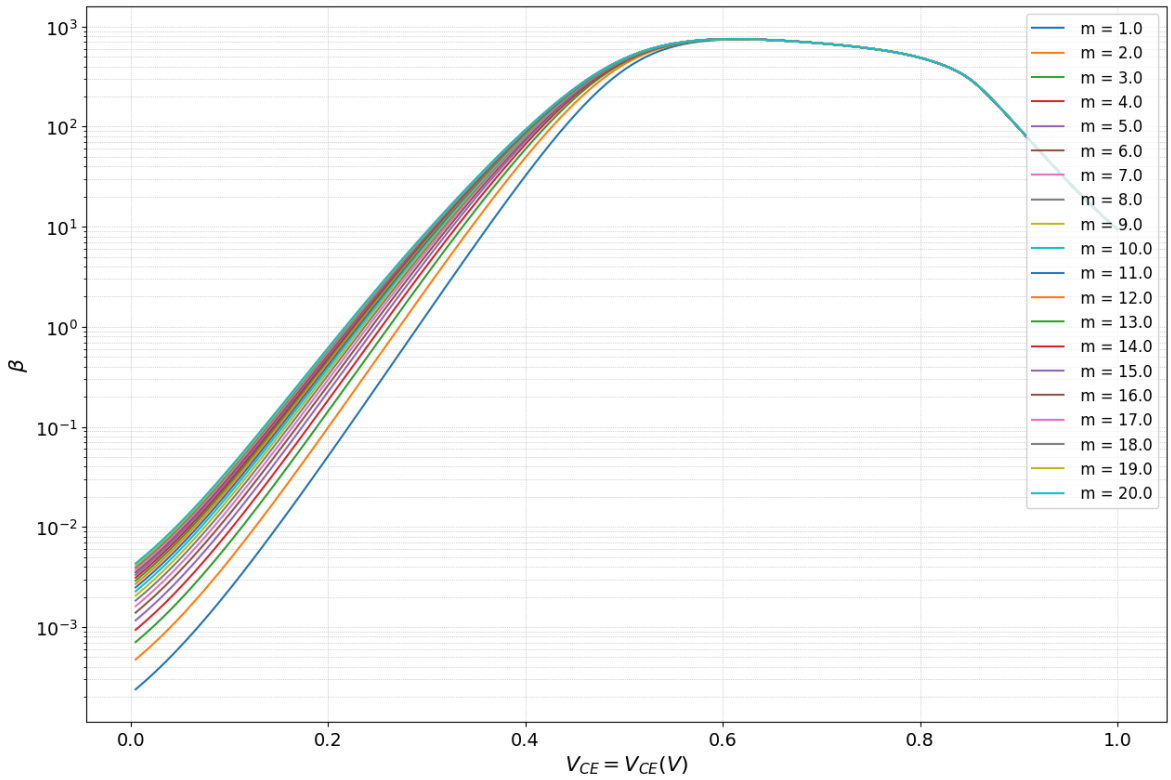
```
/var/folders/v4/bwhrmml56zdgg4rlt5zykqp00000gp/T/ipykernel_25872/795322686.py:10: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend(loc='upper right', fontsize=12)
```

Out[8]: <matplotlib.legend.Legend at 0x12d50d060>



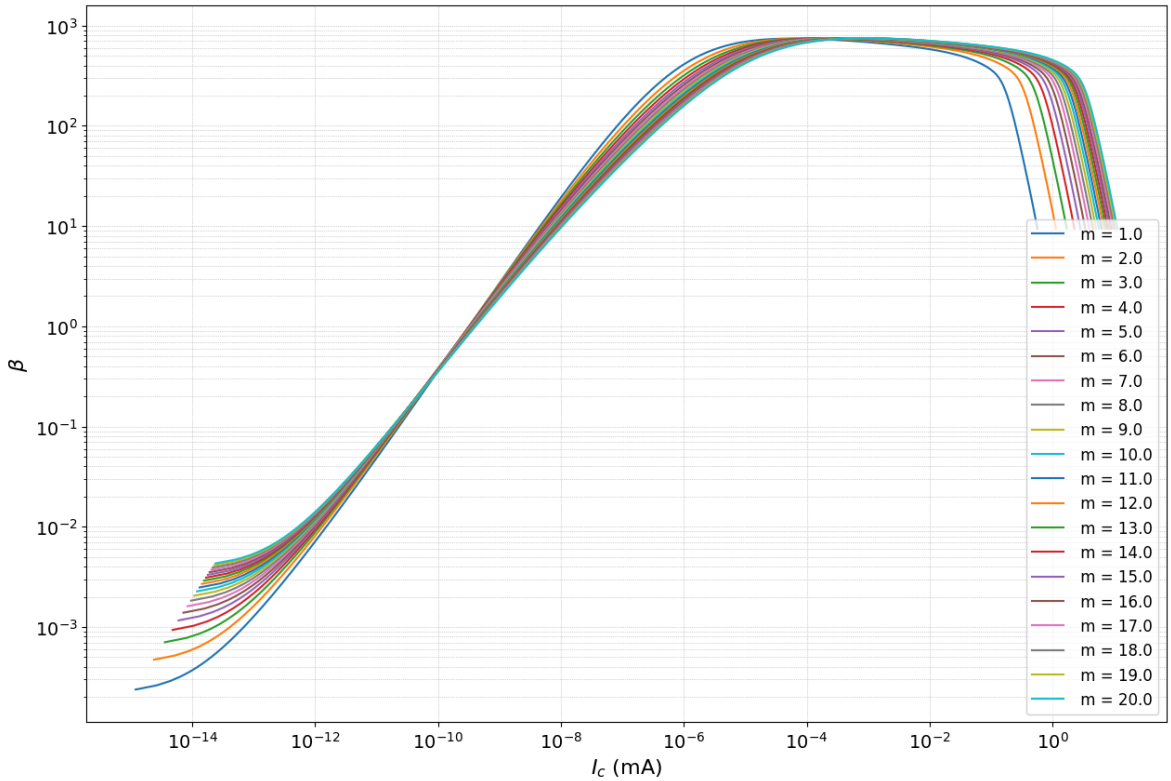
In [9]:
```
plt.figure(figsize = (15,10))
for i in range(20):
    plt.semilogy(data3[0], beta[i], label = f' m = {n[i]}' )
plt.xlabel(r'$V_{CE} = V_{CE} (V)$', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
# plt.xlim(0,2)
```

Out[9]: <matplotlib.legend.Legend at 0x12ebf3d30>



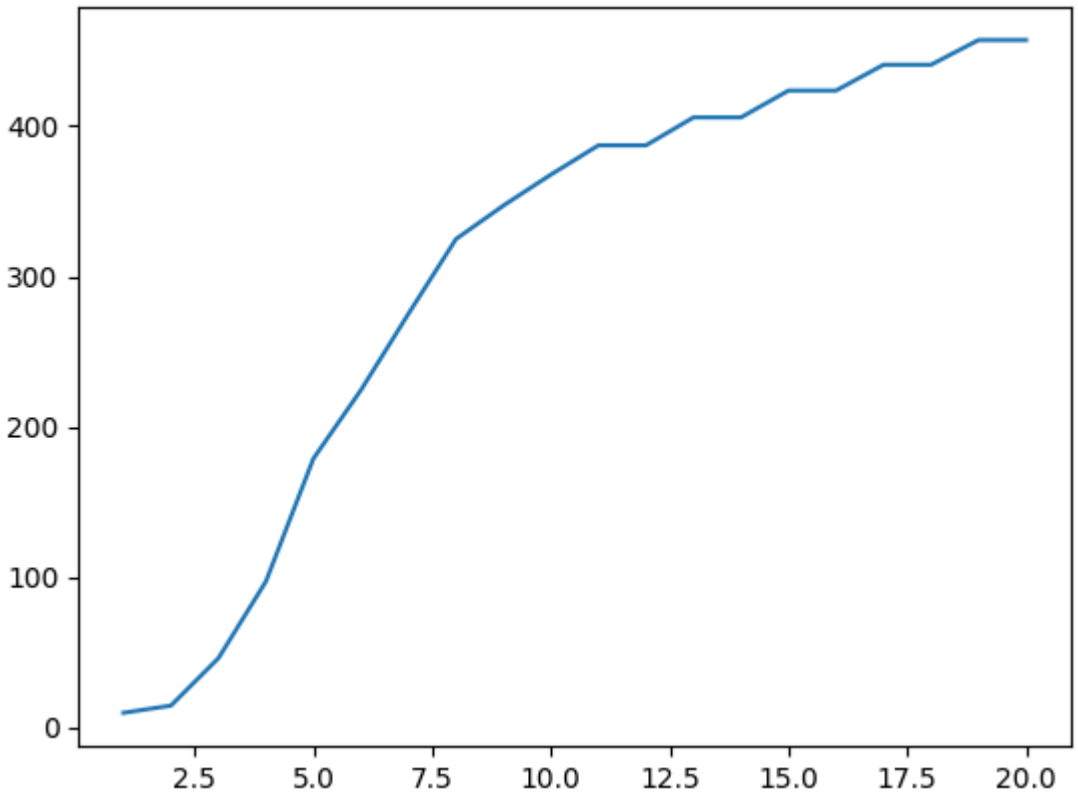
In [10]:
```
plt.figure(figsize = (15,10))
for i in range(20):
    plt.loglog(data3[gr_Ic[i]]*1e3, beta[i], label = f' m = {n[i]}' )
plt.xlabel(r'$I_c$ (mA)', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='lower right', fontsize=12)
# plt.xlim(0,2)
```

Out[10]: <matplotlib.legend.Legend at 0x12ed4e650>



In [11]:
```
import numpy as np

def find_closest_index(array, value):
  """
  Finds the index of the number in an array that is closest to a given va

  Args:
    array: The array to search in.
    value: The value to find the closest match to.

  Returns:
    The index of the closest number in the array.
  """
  array = np.asarray(array)  # Convert to NumPy array for easier calculat
  index = (np.abs(array - value)).argmin()
  return index


# Example usage
beta_1mA = []
ind = []
value = 1e-3
for i in range(20):
    # plt.loglog(data3[gr_Ic[i]]*1e3, beta[i], label = f' m = {n[i]}' )
    ind = find_closest_index(data3[gr_Ic[i]], value)
    beta_1mA.append(beta[i][ind])

plt.plot(n, beta_1mA)
```

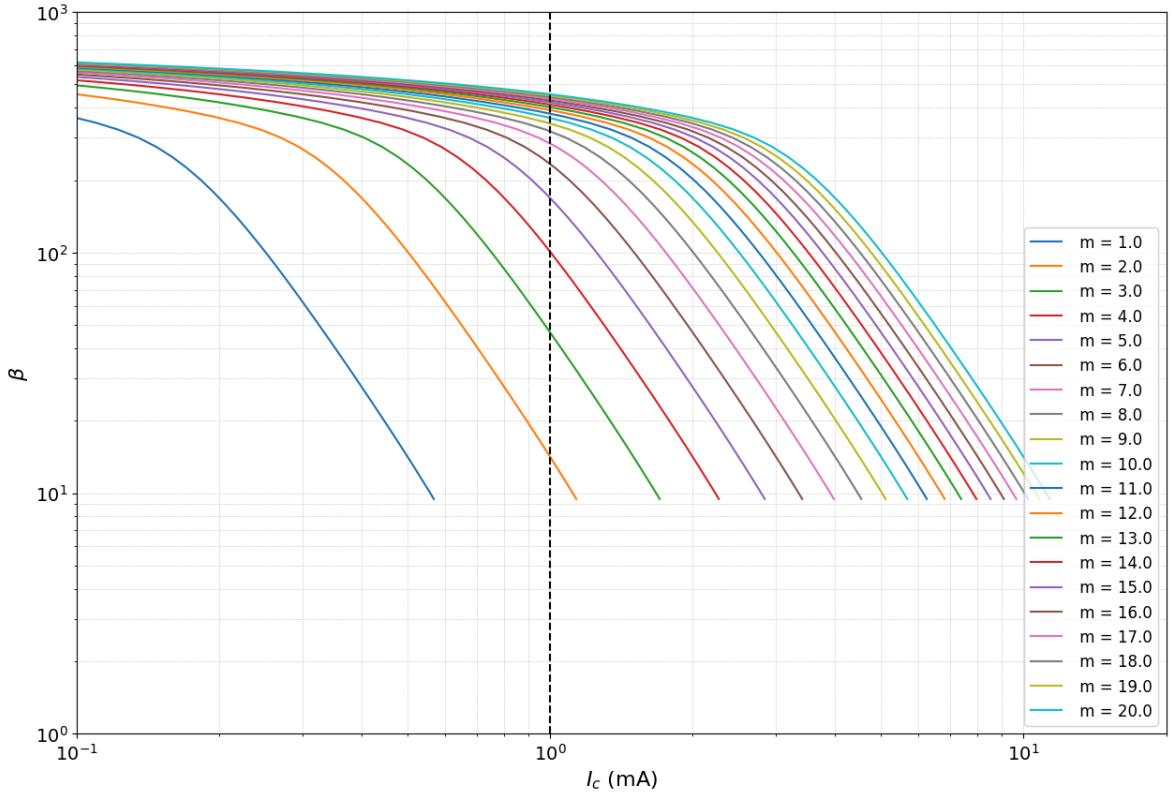
Out[11]:
```
[<matplotlib.lines.Line2D at 0x12d753df0>]
```



In [12]:
```
plt.figure(figsize = (15,10))
for i in range(20):
    plt.loglog(data3[gr_Ic[i]]*1e3, beta[i], label = f' m = {n[i]}' )
plt.xlabel(r'$I_c$ (mA)', fontsize=16)
```

```
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='lower right', fontsize=12)
plt.xlim(1e-1, 20)
plt.ylim(1,1e3)
plt.axvline(x = 1, color = 'black', linestyle ='dashed')
```

Out[12]: <matplotlib.lines.Line2D at 0x12f06d930>



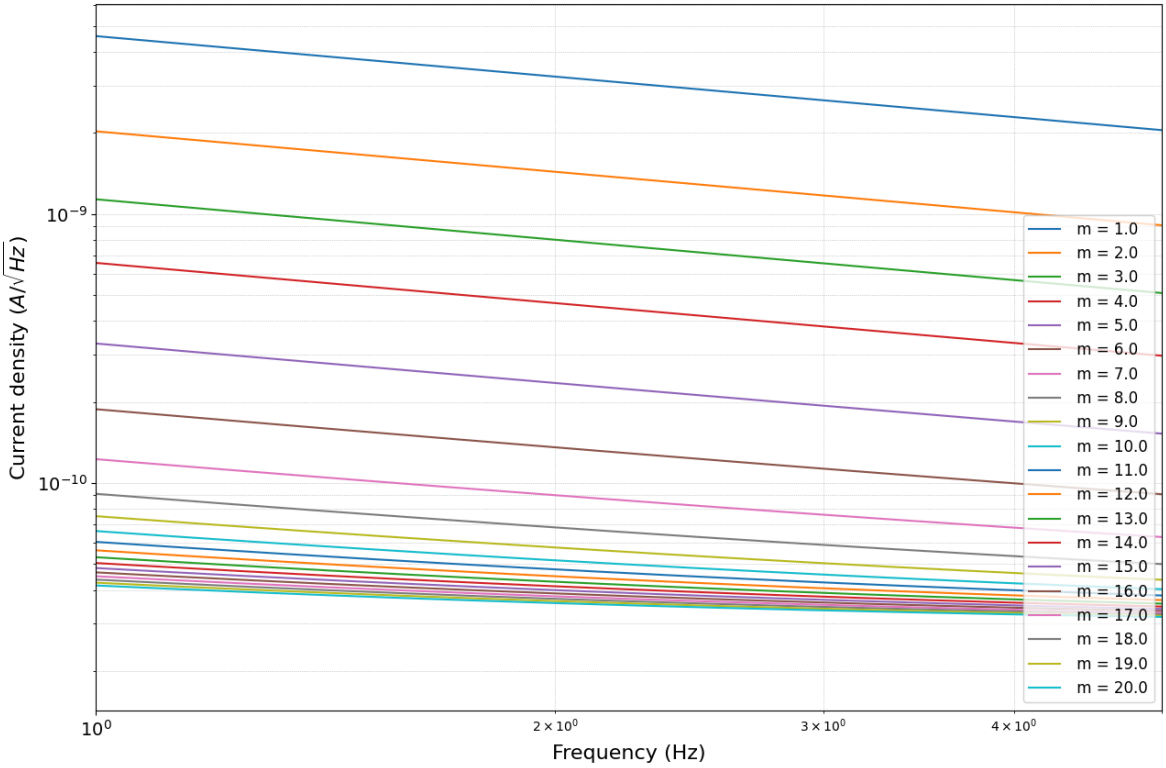
In [ ]:

4. Noise plot

In [13]:
```
data4 = pd.read_csv('current_mirror_noise_number_sweep.csv', skiprows=1,
```

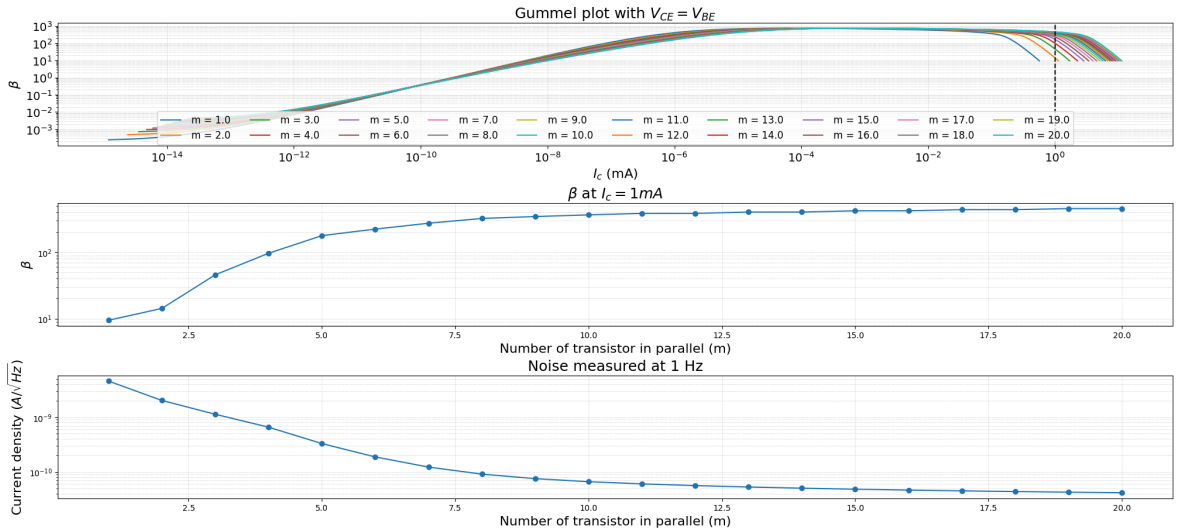
In [14]:
```
noise_1Hz = []
plt.figure(figsize = (15,10))
for i in range(20):
    plt.loglog(data4[0], data4[i+1], label = f' m = {n[i]}' )
    noise_1Hz.append(data4[i+1][0])
plt.xlabel(r'Frequency (Hz)', fontsize=16)
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='lower right', fontsize=12)
plt.xlim(1,5)
```

Out[14]: (1, 5)



In [15]:

```
plt.figure(figsize = (20,9), constrained_layout=True)
plt.subplot(311)
plt.gca().set_title(r'Gummel plot with $V_{CE} = V_{BE}$',fontsize = 18)
for i in range(20):
    plt.loglog(data3[gr_Ic[i]]*1e3, beta[i], label = f' m = {n[i]}' )
plt.xlabel(r'$I_c$ (mA)', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
# plt.legend(loc='lower right', fontsize=12)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 0.32), ncol=10, fonts
# plt.xlim(1e-1, 20)
# plt.ylim(1,1e3)
plt.axvline(x = 1, color = 'black', linestyle ='dashed')
plt.subplot(312)
plt.gca().set_title(r'$\beta$ at $I_c = 1mA$',fontsize = 18)
plt.semilogy(n, beta_1mA, marker ='o')
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'$\beta$', fontsize=16)
plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
plt.subplot(313)
plt.gca().set_title('Noise measured at 1 Hz',fontsize = 18)
plt.semilogy(n, noise_1Hz, marker ='o')
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
plt.savefig('gummel_et_noise.png')
```

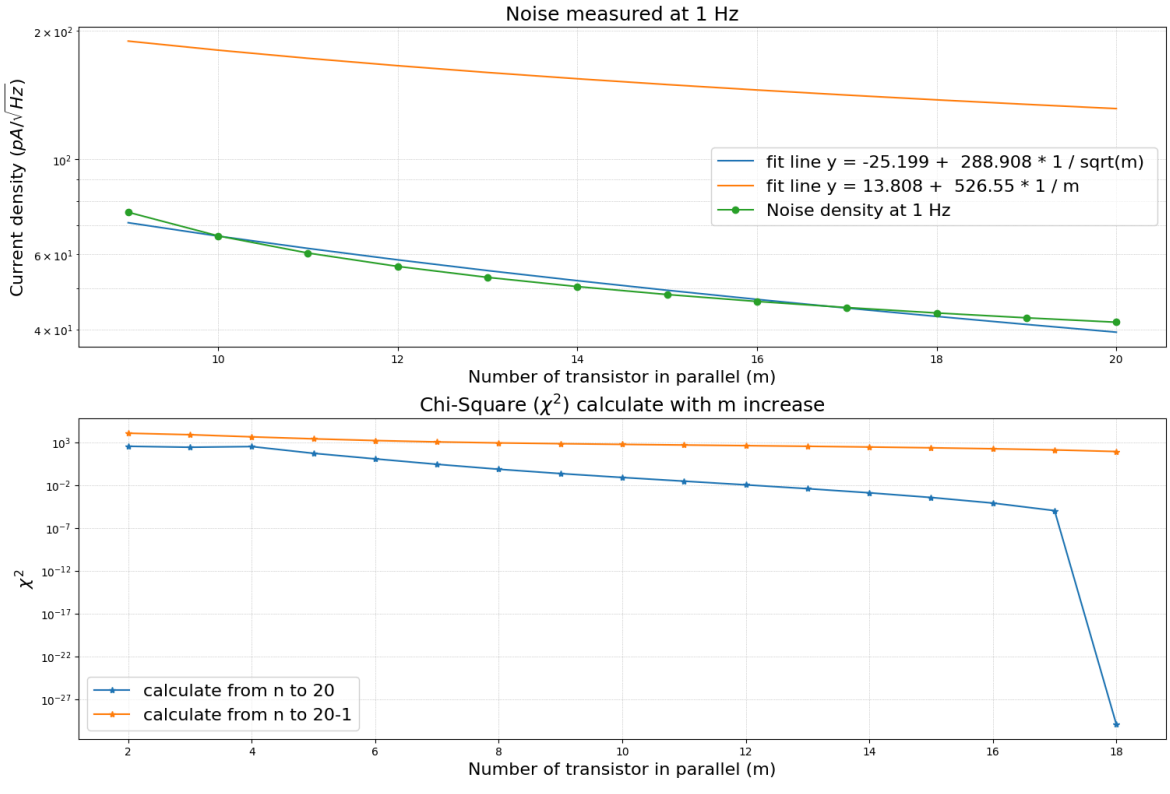


In [16]:
```python
def chi_square(observed, expected):
  """
  Calculates the chi-square statistic.

  Args:
    observed: Array of observed frequencies.
    expected: Array of expected frequencies.

  Returns:
    The chi-square statistic.
  """
  observed = np.asarray(observed)
  expected = np.asarray(expected)

  # Calculate the chi-square statistic
  chi2 = np.sum((observed - expected)**2 / expected)
  return chi2
```

In [17]:
```python
from scipy.optimize import curve_fit
def objective(x,a,b):
    return a + b * (1 / np.sqrt(x))
def objective1(x,a,b):
    return a + b * (1 / (x))
plt.figure(figsize = (15,10), constrained_layout=True)
n1 = 8
noise_to_plot = noise_1Hz[n1:20]
noise_to_plot = np.array(noise_to_plot)*1e12
plt.subplot(211)

# fit curve
popt, _ = curve_fit(objective, n[n1:20], noise_to_plot)
a, b = popt
plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y =
popt, _ = curve_fit(objective1, n[n1:20], noise_to_plot)
a, b = popt
plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y =

plt.gca().set_title('Noise measured at 1 Hz',fontsize = 18)
plt.semilogy(n[n1:20], noise_to_plot, marker ='o', label = 'Noise density
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
plt.legend(fontsize = 16)
```

```python
# plt.savefig('gummel_et_noise.png')
# n1 = np.arange(9,20,1)
plt.subplot(212)
plt.gca().set_title(r'Chi-Square ($\chi^2$) calculate with m increase', f
chi = []
chi1 =[]
n_chi = np.arange(2,19,1)
for n_c in n_chi:
    noise_to_plot = noise_1Hz[n_c:20]
    noise_to_plot = np.array(noise_to_plot)*1e12
    popt, _ = curve_fit(objective, n[n_c:20], noise_to_plot)
    a, b = popt
    y1 = objective(n[n_c:20], a, b)
    chi.append(chi_square(noise_to_plot, y1))

    popt, _ = curve_fit(objective1, n[n_c:20], noise_to_plot)
    a, b = popt
    y1 = objective(n[n_c:20], a, b)
    chi1.append(chi_square(noise_to_plot, y1))
plt.semilogy(n_chi, chi,marker = '*', label = 'calculate from n to 20')
plt.semilogy(n_chi, chi1,marker = '*', label = 'calculate from n to 20-1'
plt.ylabel(r'$\chi^2$', fontsize=16)
plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
plt.legend(fontsize = 16)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.savefig('noise_fit.png')
```

```
/var/folders/v4/bwhrmml56zdgg4rlt5zykqp00000gp/T/ipykernel_25872/183874337
9.py:36: OptimizeWarning: Covariance of the parameters could not be estima
ted
  popt, _ = curve_fit(objective, n[n_c:20], noise_to_plot)
/var/folders/v4/bwhrmml56zdgg4rlt5zykqp00000gp/T/ipykernel_25872/183874337
9.py:41: OptimizeWarning: Covariance of the parameters could not be estima
ted
  popt, _ = curve_fit(objective1, n[n_c:20], noise_to_plot)
```

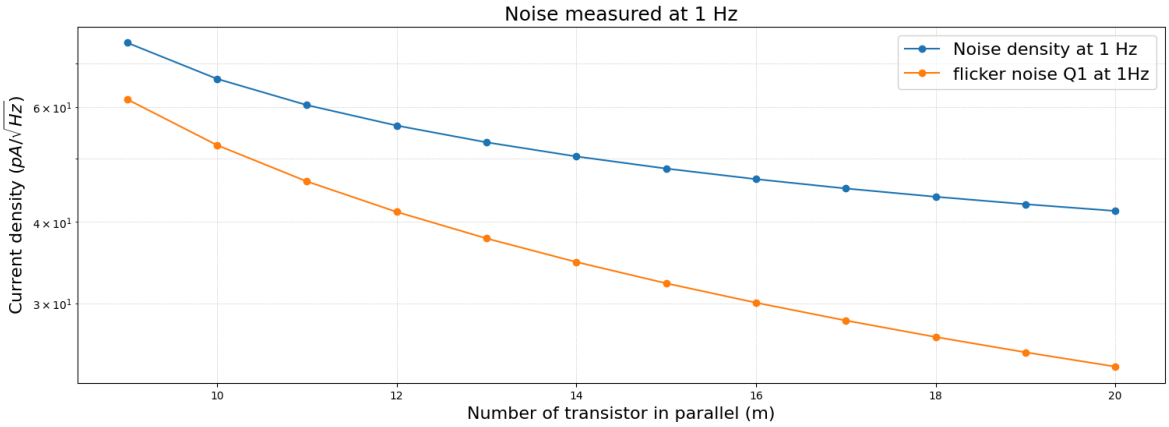


In [18]:
```
fn = [6.16241e-11, 5.24309e-11, 4.61346e-11, 4.14338e-11, 3.77342e-11, 3.
fn = np.array(fn)*1e12
```

In [19]:
```
plt.figure(figsize = (15,10), constrained_layout=True)
n1 = 8
noise_to_plot = noise_1Hz[n1:20]
noise_to_plot = np.array(noise_to_plot)*1e12
plt.subplot(211)

# # fit curve
# popt, _ = curve_fit(objective, n[n1:20], noise_to_plot)
# a, b = popt
# plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y
# popt, _ = curve_fit(objective1, n[n1:20], noise_to_plot)
# a, b = popt
# plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y
print(n[n1:20])
plt.gca().set_title('Noise measured at 1 Hz',fontsize = 18)
plt.semilogy(n[n1:20], noise_to_plot, marker ='o', label = 'Noise density
plt.semilogy(n[n1:20], fn, marker ='o', label = 'flicker noise Q1 at 1Hz'
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
plt.legend(fontsize = 16)
```

```
[ 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20.]
```

Out[19]: <matplotlib.legend.Legend at 0x12d5d7fa0>



In [20]:
```
# plt.figure(figsize = (15,10), constrained_layout=True)
# n1 = 8
# noise_to_plot = noise_1Hz[n1:20]
# noise_to_plot = np.array(noise_to_plot)*1e12
# noise_to_plot = fn
# plt.subplot(211)

# # fit curve
# popt, _ = curve_fit(objective, n[n1:20], noise_to_plot)
# a, b = popt
# plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y
# popt, _ = curve_fit(objective1, n[n1:20], noise_to_plot, p0 = [n1])
# a, b = popt
# plt.semilogy(n[n1:20], objective(n[n1:20], a, b), label = f'fit line y

# plt.gca().set_title('Noise measured at 1 Hz',fontsize = 18)
# plt.semilogy(n[n1:20], noise_to_plot, marker ='o', label = 'Noise densi
# plt.grid(which='both', linestyle=':', linewidth='0.5')
```

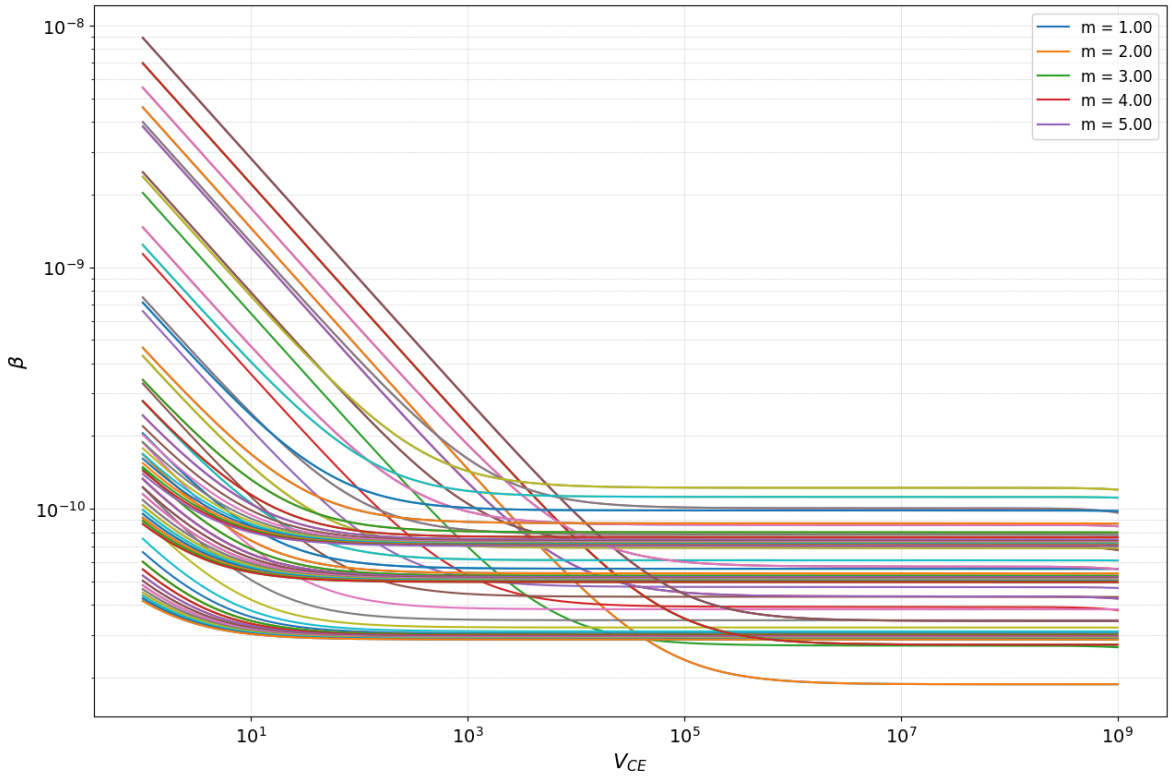
```
# plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
# plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
# plt.legend(fontsize = 16)
# # plt.savefig('gummel_et_noise.png')
# # n1 = np.arange(9,20,1)
# # plt.subplot(212)
# # plt.gca().set_title(r'Chi-Square ($\chi^2$) calculate with m increase
# # chi = []
# # chi1 =[]
# # n_chi = np.arange(9,19,1)
# # for n_c in n_chi:
# #     noise_to_plot = fn
# #     popt, _ = curve_fit(objective, n[n_c:20], noise_to_plot)
# #     a, b = popt
# #     y1 = objective(n[n_c:20], a, b)
# #     chi.append(chi_square(noise_to_plot, y1))

# #     popt, _ = curve_fit(objective1, n[n_c:20], noise_to_plot)
# #     a, b = popt
# #     y1 = objective(n[n_c:20], a, b)
# #     chi1.append(chi_square(noise_to_plot, y1))
# # plt.semilogy(n_chi, chi,marker = '*', label = 'calculate from n to 20
# # plt.semilogy(n_chi, chi1,marker = '*', label = 'calculate from n to 2
# # plt.ylabel(r'$\chi^2$', fontsize=16)
# # plt.xlabel('Number of transistor in parallel (m)', fontsize=16)
# # plt.legend(fontsize = 16)
# # plt.grid(which='both', linestyle=':', linewidth='0.5')
```

In [21]:
```
data5 = pd.read_csv('current_mirror_noise_number_sweep_withmandn.csv', sk
```

In [22]:
```
N = [1, 2, 3, 4, 5]
n = np.linspace(1, 20, 20)
print(n)
count = 1
count = int(count)

plt.figure(figsize=(15, 10))  # Create the figure only once
# plt.title(r' $\beta$ vs $V_{CE}$',fontsize = 20)
for i in N:
    try:
        plt.semilogy(data5[0], data5[count], label=f'm = {i:.2f}')

        for i in range(count, count + 20):
            plt.loglog(data5[0], data5[i])

    except IndexError:
        print(f"Error: Index {count5} or {i} out of range for data")
        break

    count = count + 10

plt.xlabel(r'$V_{CE} $', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
```

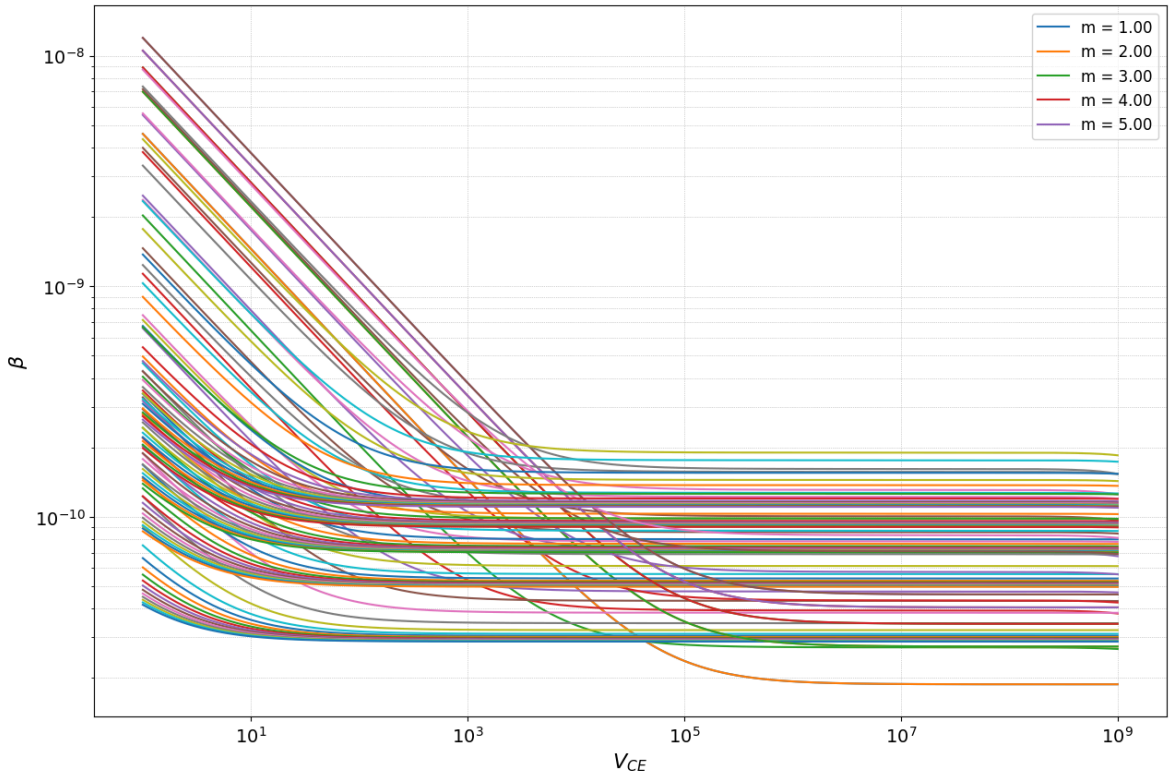
```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 19. 20.]
```

Out[22]: <matplotlib.legend.Legend at 0x12e9cabc0>



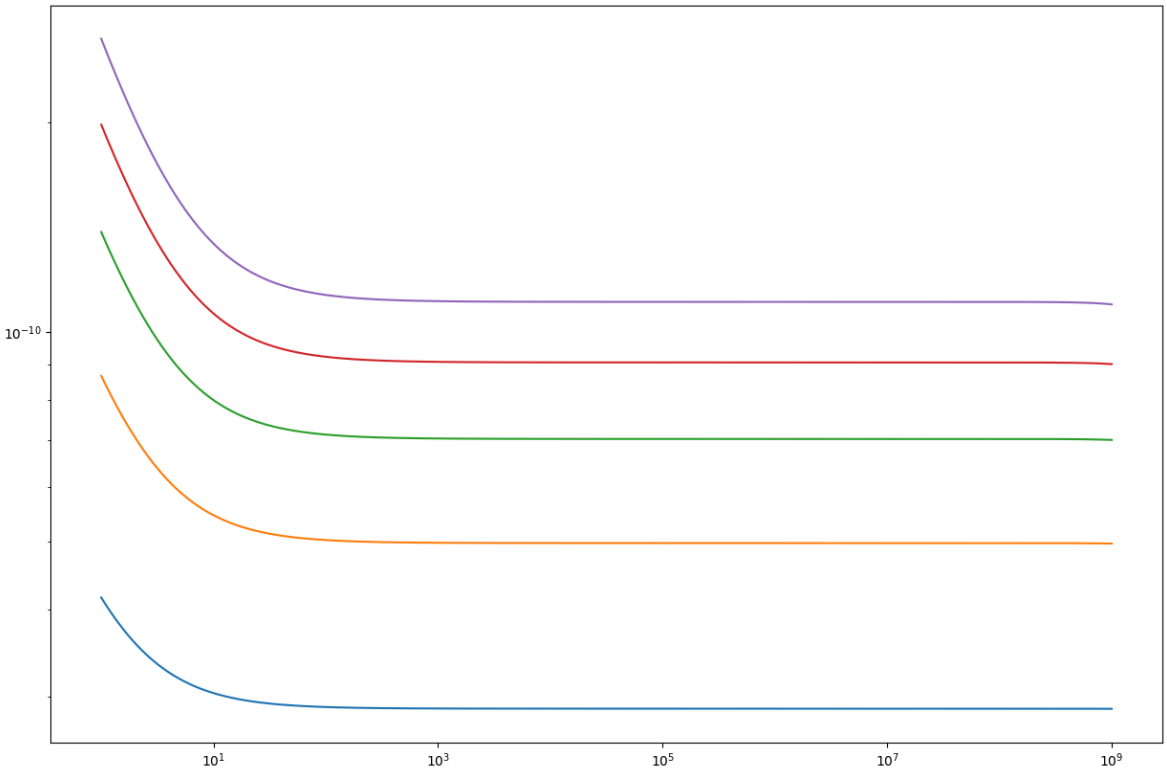
In [23]:
```
N = [1, 2, 3, 4, 5]
n = np.linspace(1, 20, 20)
count = 1
count = int(count)
N1 = []
N2 = []
N3 = []
N4 = []
N5 = []
N_add = [N1, N2, N3, N4, N5]

plt.figure(figsize=(15, 10))

for i in N:
    try:
        plt.semilogy(data5[0], data5[count], label=f'm = {i:.2f}')

        for j in range(count, count + 20):
            plt.loglog(data5[0], data5[j])
            N_add[i-1].append(data5[j])  # Append to the correct list bas

    except IndexError:
        print(f"Error: Index {count} or {j} out of range for data5")
        break

    count = count + 20

plt.xlabel(r'$V_{CE} $', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.legend(loc='upper right', fontsize=12)
```

Out[23]: <matplotlib.legend.Legend at 0x12f2e97e0>



In [24]:
```
plt.figure(figsize = (15,10))
n = [1,2,3,4,5]
plt.loglog(data5[0], N_add[0][19])
plt.loglog(data5[0], N_add[1][19])
plt.loglog(data5[0], N_add[2][19])
plt.loglog(data5[0], N_add[3][19])
plt.loglog(data5[0], N_add[4][19])
for i in range(5):
    print(N_add[i][19][10])
KF = 6.2e-10 / 0.27
ic = 1e-3
beta = 400
for i in n:
    print('------')
    print(f' N = {i}')
    m = 10
    i1 = np.sqrt(KF * ic**2 / beta**2 *1 * 1/m)
    i2 = np.sqrt(KF * ic**2 / beta**2 *1 * i**2/m)
    print(f'sqrt(i1**2) = {i1*np.sqrt(i)}')
    print(f'sqrt(i2**2) = {i2}')
    print(f'sqrt(i1**2 + i2**2) = {np.sqrt(i1**2 + i2**2)}')
```

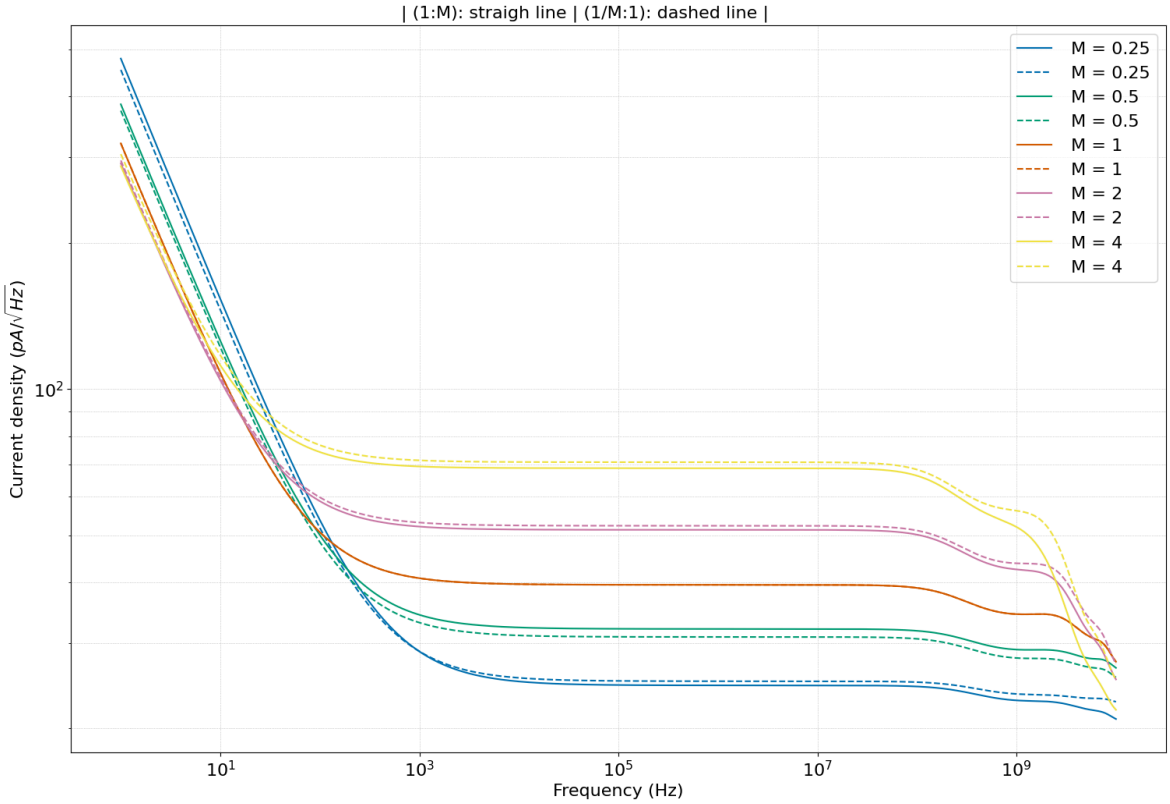
```
3.9289e-11
8.03702e-11
1.28142e-10
1.81758e-10
2.4053e-10
------
 N = 1
sqrt(i1**2) = 3.788383804718293e-11
sqrt(i2**2) = 3.788383804718293e-11
sqrt(i1**2 + i2**2) = 5.3575837561071974e-11
------
 N = 2
sqrt(i1**2) = 5.3575837561071974e-11
sqrt(i2**2) = 7.576767609436587e-11
sqrt(i1**2 + i2**2) = 8.471083712209392e-11
------
 N = 3
sqrt(i1**2) = 6.561673228343176e-11
sqrt(i2**2) = 1.136515141415488e-10
sqrt(i1**2 + i2**2) = 1.1979921473804346e-10
------
 N = 4
sqrt(i1**2) = 7.576767609436587e-11
sqrt(i2**2) = 1.5153535218873173e-10
sqrt(i1**2 + i2**2) = 1.561990657723283e-10
------
 N = 5
sqrt(i1**2) = 8.471083712209392e-11
sqrt(i2**2) = 1.8941919023591466e-10
sqrt(i1**2 + i2**2) = 1.9317042945237453e-10
```



03/12/2024

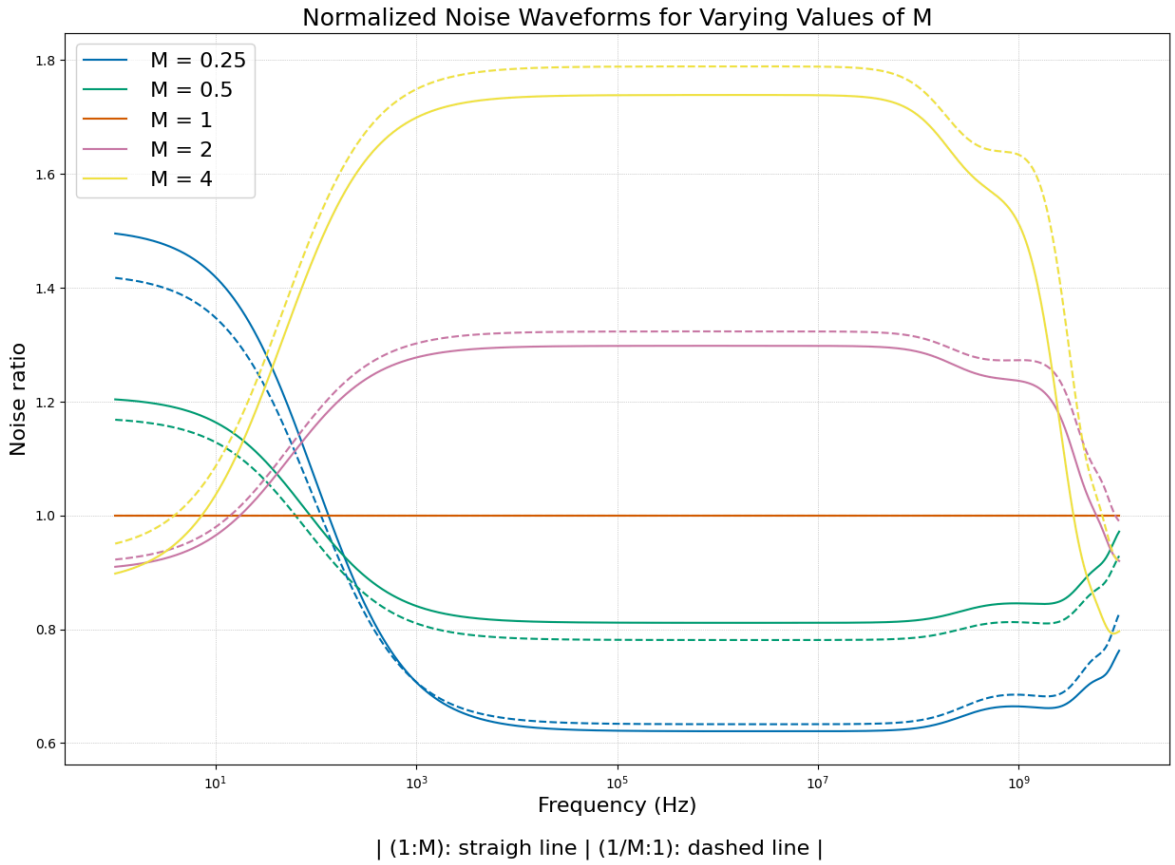
In [25]:
```
noise1 = pd.read_csv('noise1.csv', skiprows=1, header=None) # No header
noise2 = pd.read_csv('noise2.csv', skiprows=1, header=None) # No header
```

In [26]:
```
M = [0.25,0.5,1,2,4]
col = ["#0072B2", "#009E73", "#D55E00", "#CC79A7", "#F0E442", "#56B4E9"]
nnpn = 20
npnp = 22
El = 1
note = f'Nnpn = {nnpn}, Npnp = {npnp}, El = {El} um'
note = 'line = case 1, dashed = case 2'
plt.figure(figsize = (15,10), constrained_layout=True)

for i in range(len(M)):
    plt.loglog(noise1[0],noise1[i+1]*1e12, color = col[i], label = f' M =
    plt.loglog(noise2[0],noise2[i+1]*1e12, color = col[i], linestyle = 'd

plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Frequency (Hz)', fontsize=16)
plt.xticks(fontsize = 16)
plt.yticks(fontsize = 16)
plt.legend(fontsize = 16)
note =  '| (1:M): straigh line | (1/M:1): dashed line |'
plt.figtext(0.5, 1, note , ha="center", fontsize=16, va="bottom")

plt.savefig('noise_M.png')
```



In [27]:
```
plt.figure(figsize = (15,10))
for i in range(len(M)):
    plt.semilogx(noise1[0],(noise1[i+1]/noise1[3]), color = col[i], label
    plt.semilogx(noise2[0],(noise2[i+1]/noise1[3]), linestyle = 'dashed',
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Noise ratio', fontsize=16)
plt.xlabel('Frequency (Hz)', fontsize=16)
plt.legend(fontsize = 16)
note =  '| (1:M): straigh line | (1/M:1): dashed line |'
plt.figtext(0.5, 0.01, note , ha="center", fontsize=16, va="bottom")
plt.title('Normalized Noise Waveforms for Varying Values of M', fontsize
# plt.savefig('noise_ratio.png')
```

Out[27]:
```
Text(0.5, 1.0, 'Normalized Noise Waveforms for Varying Values of M')
```



varies nnpn = 8,12,16,20,24,28,32,36,40

In [28]:
```
noise_1 = pd.read_csv('noise1_tot.csv', skiprows=1, header=None) # No hea
noise_2 = pd.read_csv('noise2_tot.csv', skiprows=1, header=None) # No hea
```

In [29]:
```
# Spnp =
# Snpn =
# def sur_area_case1(npnp,nnpn,M):
```

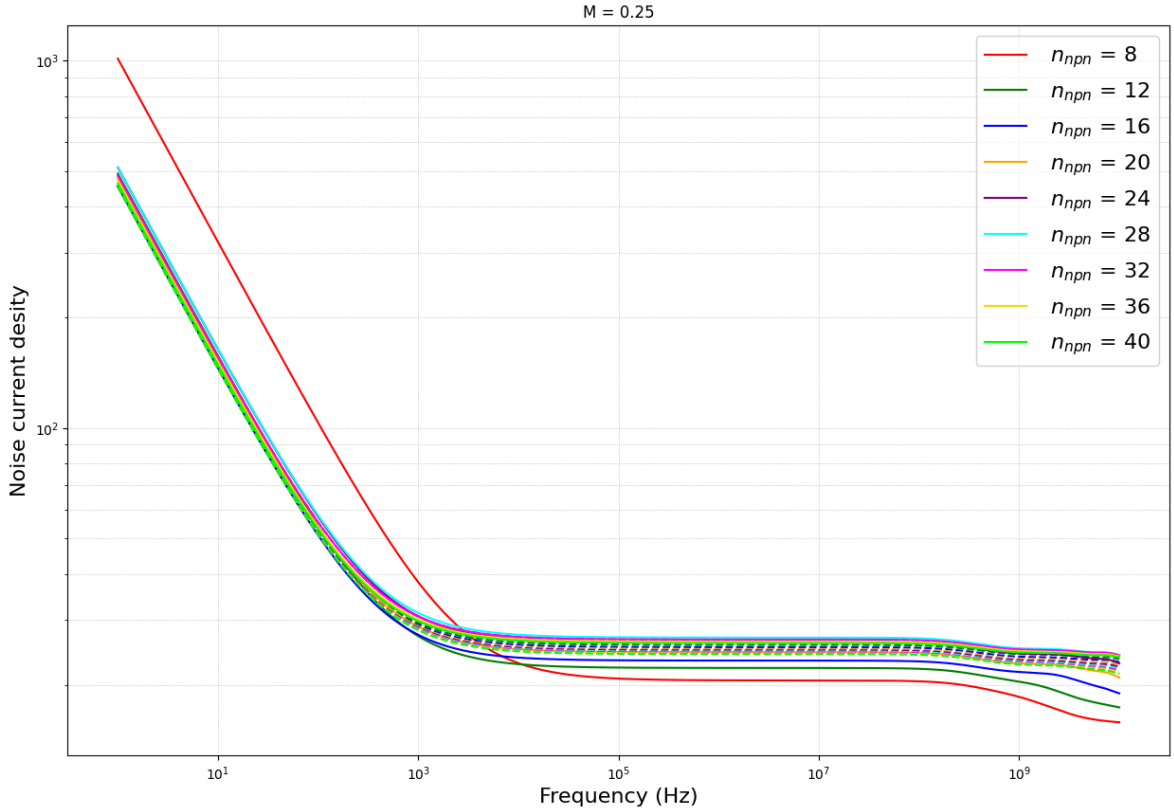
In [ ]:
```
```

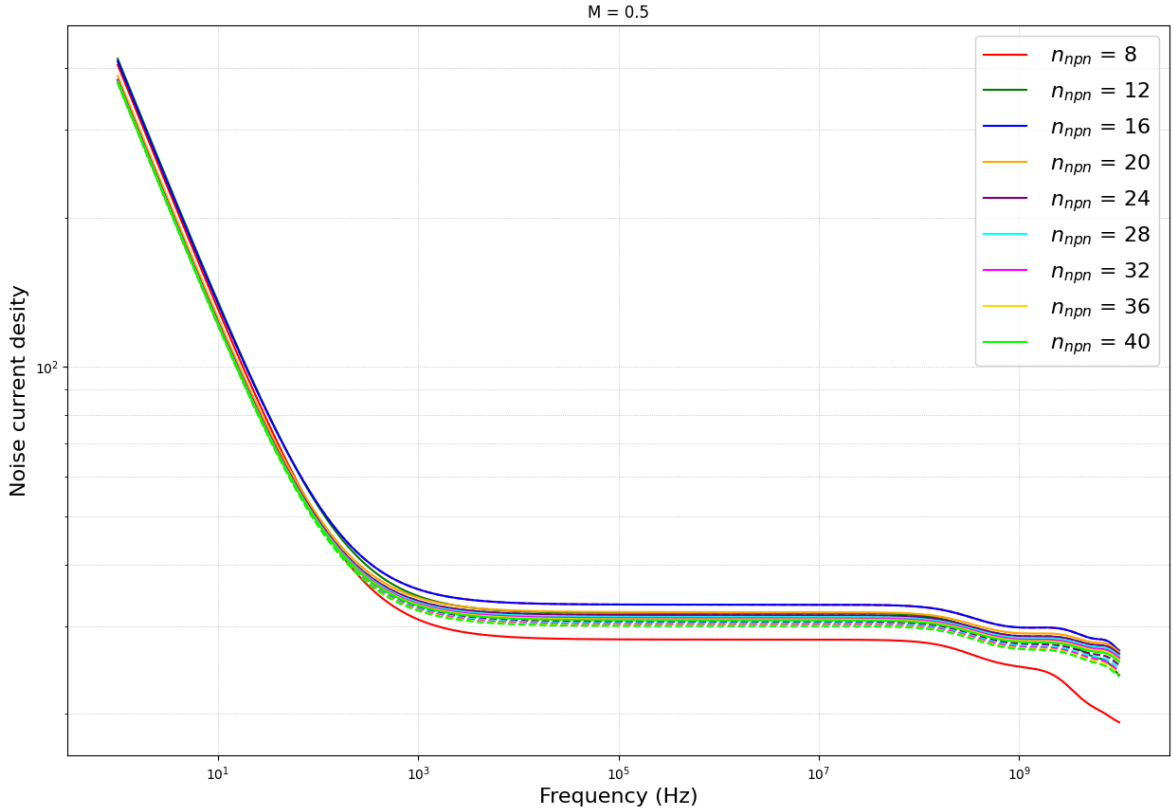
In [30]:
```
import matplotlib.pyplot as plt
import numpy as np

# ... (Your existing code to load or define 'noise_1' should be here) ...
```

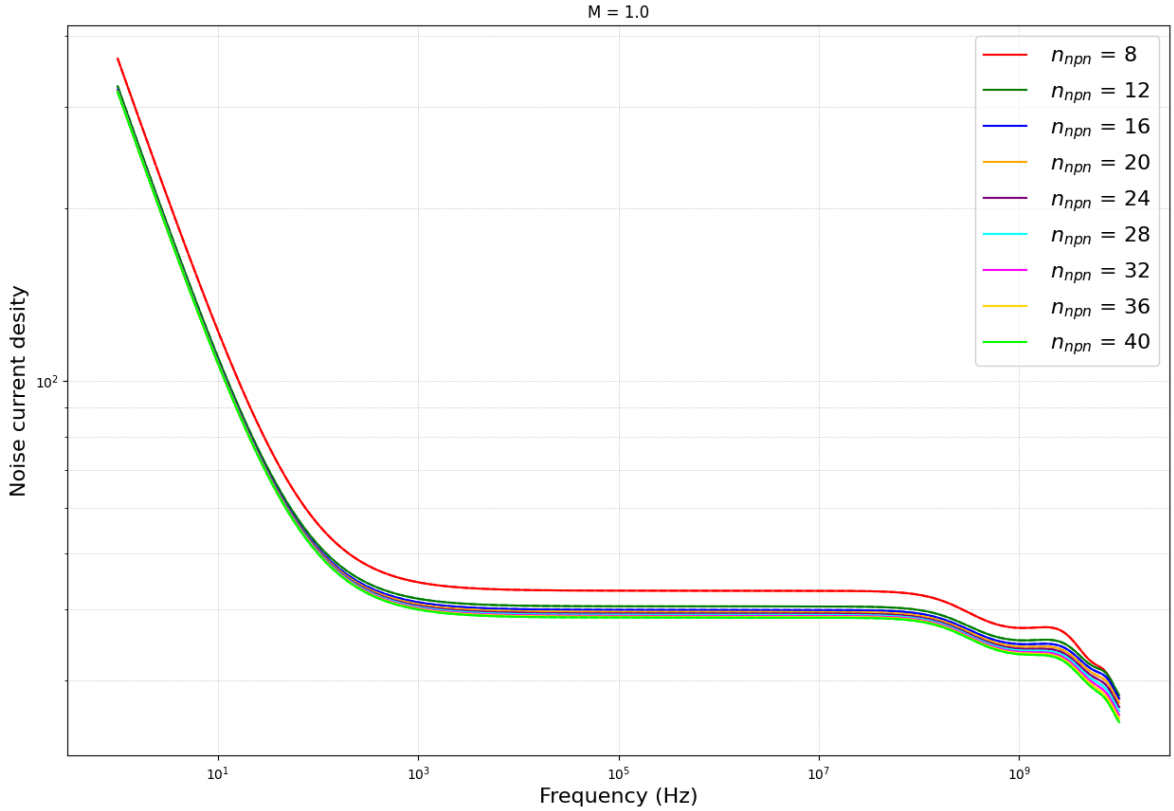
```
nnpn = [8, 12, 16, 20, 24, 28, 32, 36, 40]
M = [0.25, 0.5, 1, 2, 4]
M = np.array(M)
n = np.linspace(1, 9, 9)

M1_1 = []
M2_1 = []
M3_1 = []
M4_1 = []
M5_1 = []
M_add1 = [M1_1, M2_1, M3_1, M4_1, M5_1]
M1_2 = []
M2_2 = []
M3_2 = []
M4_2 = []
M5_2 = []
M_add2 = [M1_2, M2_2, M3_2, M4_2, M5_2]

for i in range(5):
    for j in range(9):
        try:
            index = 1 + i + j * 5  # Calculate the index directly
            M_add1[i].append(noise_1[index])
            M_add2[i].append(noise_2[index])
        except IndexError:
            print(f"Error: Index {index} out of range for noise_1")
            break
```

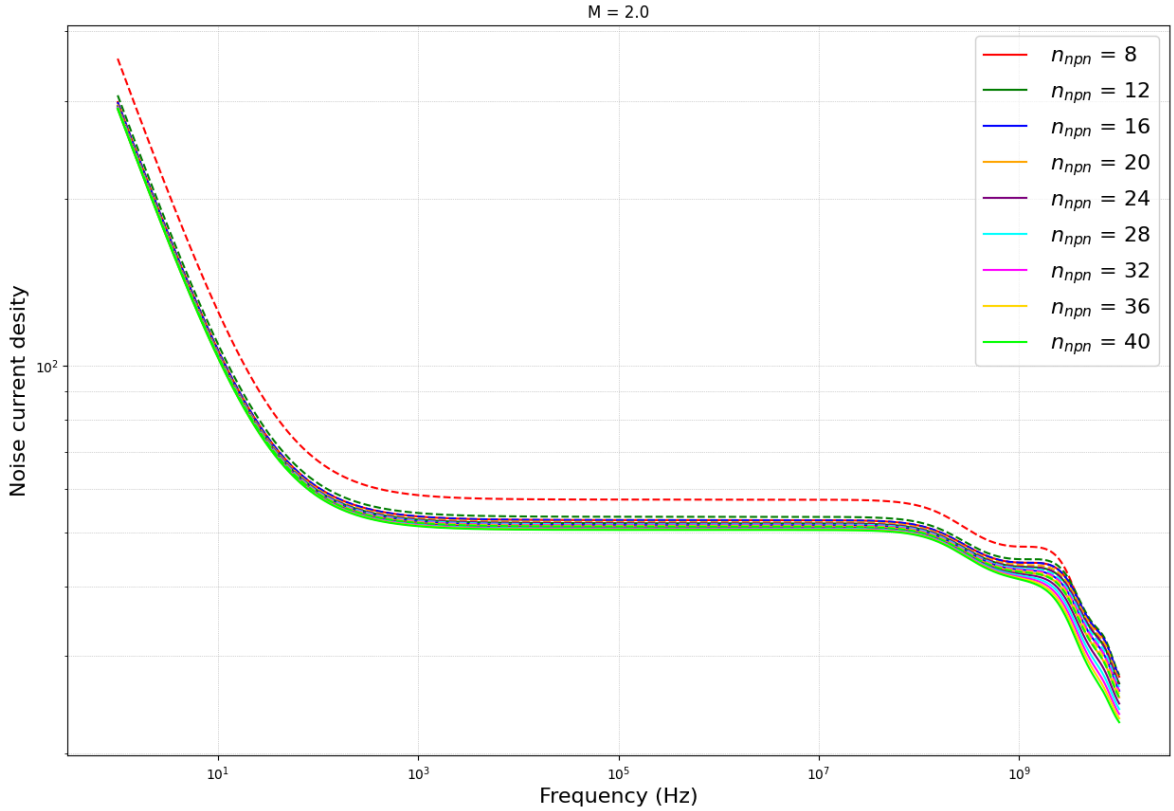
In [31]:

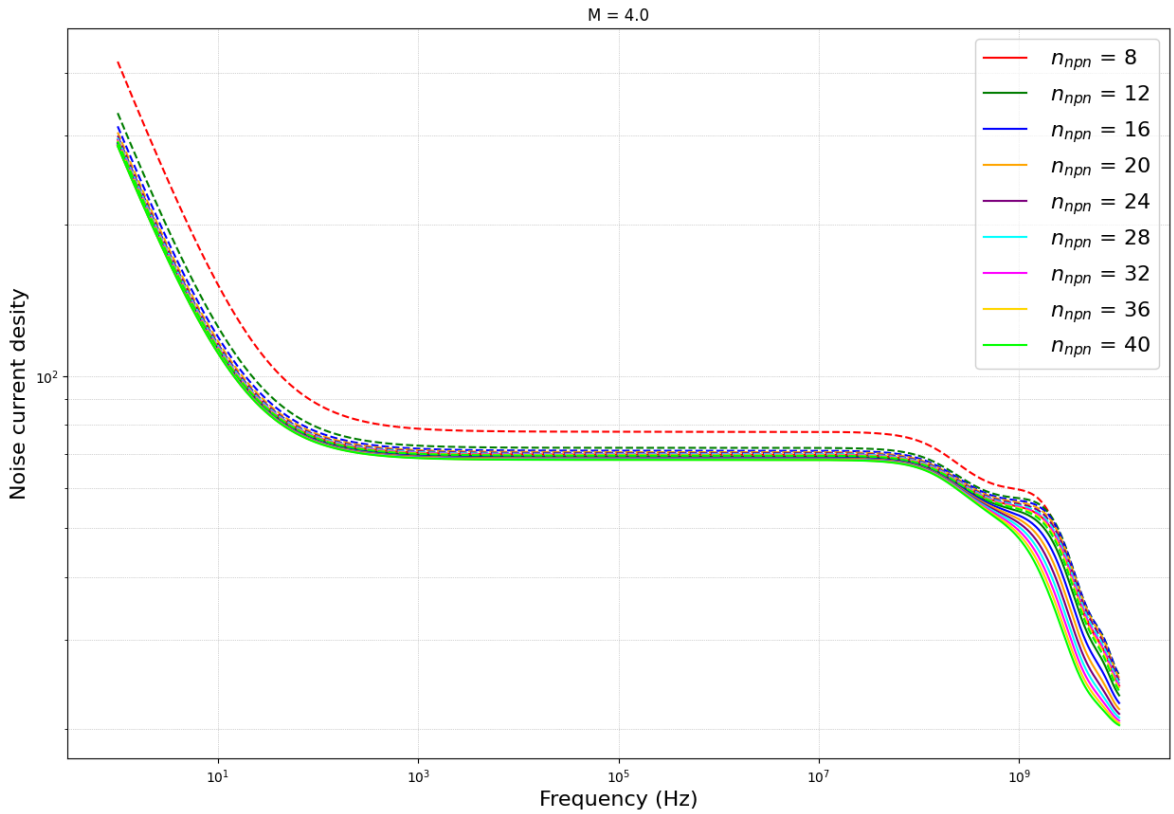
```
colors = [
    'red', 'green', 'blue', 'orange', 'purple', 'cyan', 'magenta', 'gold'
]
for i in range(5):
    plt.figure(figsize = (15,10))
    plt.title(f'M = {M[i]}')
    for j in range(0,9):
        # plt.semilogx(noise_1[0], M_add1[i][j]/M_add1[i][1], color = col
        # plt.semilogx(noise_2[0], M_add2[i][j]/M_add2[i][1], color = col
        plt.loglog(noise_1[0], M_add1[i][j]*1e12, color = colors[j],  lab
        plt.loglog(noise_2[0], M_add2[i][j]*1e12, color = colors[j], line
    plt.grid(which='both', linestyle=':', linewidth='0.5')
    plt.ylabel(r'Noise current desity', fontsize=16)
    plt.xlabel('Frequency (Hz)', fontsize=16)
    plt.legend(fontsize = 16)
    note =  '| (1:M): straigh line | (1/M:1): dashed line |'
    # plt.figtext(0.5, 0.01, note , ha="center", fontsize=16, va="bottom"
```


M = 0.25
Noise current desity
Frequency (Hz)
n_{npn} = 8
n_{npn} = 12
n_{npn} = 16
n_{npn} = 20
n_{npn} = 24
n_{npn} = 28
n_{npn} = 32
n_{npn} = 36
n_{npn} = 40


M = 0.5
Noise current desity
Frequency (Hz)
n_{npn} = 8
n_{npn} = 12
n_{npn} = 16
n_{npn} = 20
n_{npn} = 24
n_{npn} = 28
n_{npn} = 32
n_{npn} = 36
n_{npn} = 40


M = 1.0
Noise current desity
10^2
10^1
10^3
10^5
10^7
10^9
Frequency (Hz)
n_{npn} = 8
n_{npn} = 12
n_{npn} = 16
n_{npn} = 20
n_{npn} = 24
n_{npn} = 28
n_{npn} = 32
n_{npn} = 36
n_{npn} = 40


M = 2.0
Noise current desity
10^2
10^1
10^3
10^5
10^7
10^9
Frequency (Hz)
n_{npn} = 8
n_{npn} = 12
n_{npn} = 16
n_{npn} = 20
n_{npn} = 24
n_{npn} = 28
n_{npn} = 32
n_{npn} = 36
n_{npn} = 40



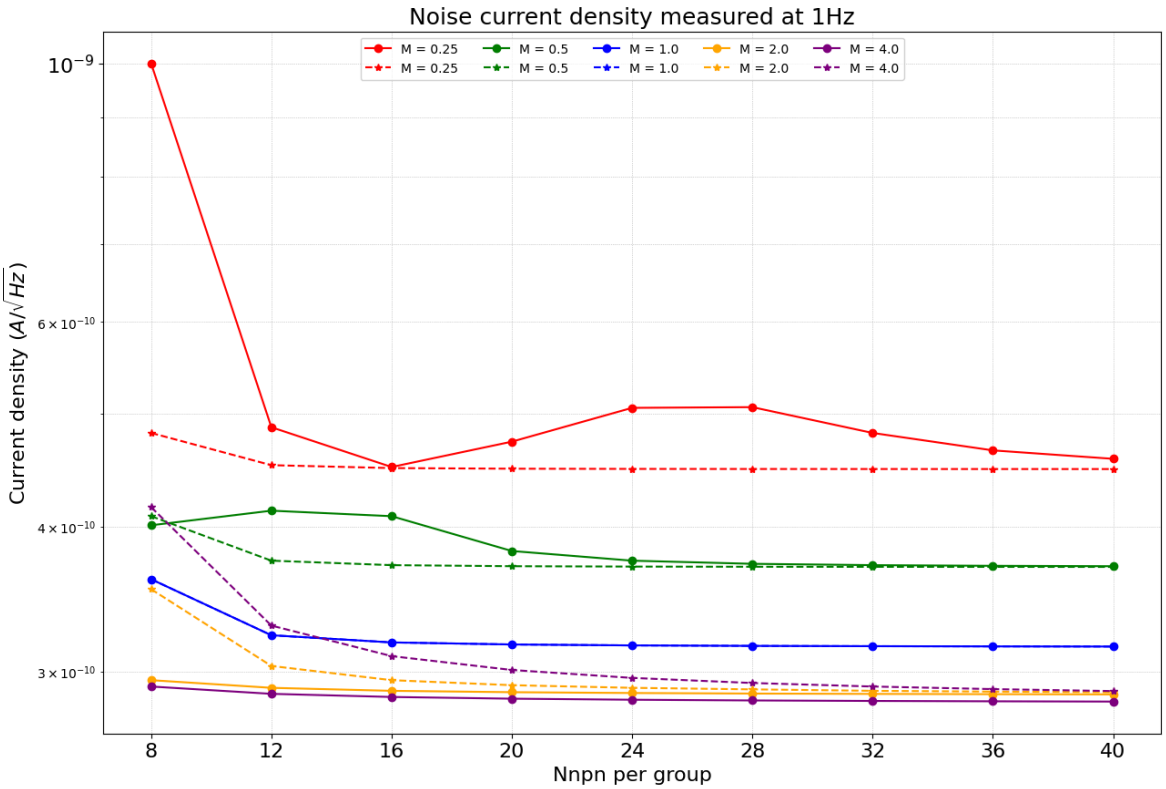
compare in fraction of noise variation

Observe the variation in white noise region (f = 10MHz) and low noise region (f = 1 Hz)

In [32]:
```
#1 Hz
noise1Hz_1 = [[],[],[],[],[]]
noise10MHz_1 = [[],[],[],[],[]]
noise1Hz_2 = [[],[],[],[],[]]
noise10MHz_2 = [[],[],[],[],[]]
for i in range(5):
    for j in range(9):
        noise1Hz_1[i].append(M_add1[i][j][1])
        noise10MHz_1[i].append(M_add1[i][j][800])
        noise1Hz_2[i].append(M_add2[i][j][1])
        noise10MHz_2[i].append(M_add2[i][j][800])
```

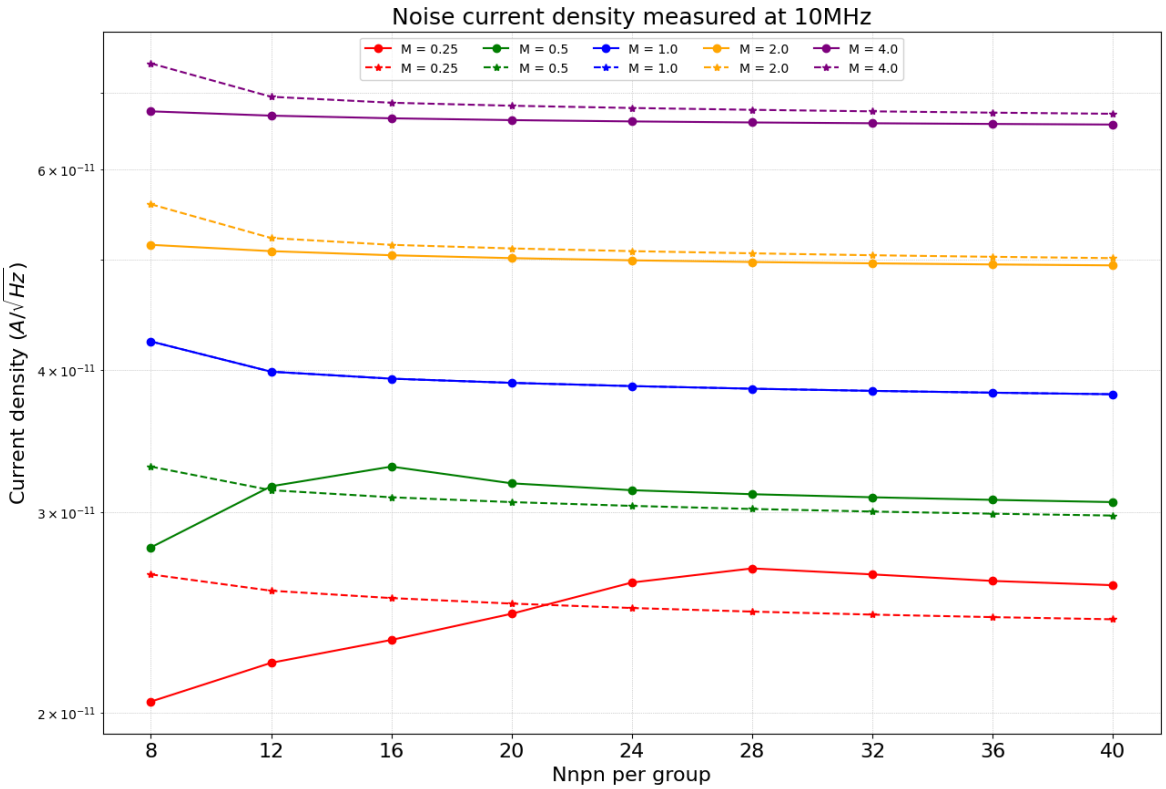
In [33]:
```
plt.figure(figsize = (15,10))
for i in range(5):
    plt.semilogy(nnpn, noise1Hz_1[i],color = colors[i], marker = 'o', lab
    plt.semilogy(nnpn, noise1Hz_2[i],color = colors[i], marker = '*', lin
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Nnpn per group', fontsize=16)
plt.xticks(nnpn,fontsize = 16)
plt.yticks(fontsize = 16)
plt.legend(loc='upper center', ncol=5, bbox_to_anchor=(0.5, 1.0))
plt.title('Noise current density measured at 1Hz',fontsize = 18)
# note =  '| (1:M): straigh line | (1/M:1): dashed line |'
# plt.figtext(0.5, 1, note , ha="center", fontsize=16, va="bottom")
```

Out[33]:
```
Text(0.5, 1.0, 'Noise current density measured at 1Hz')
```



In [34]:
```
plt.figure(figsize = (15,10))
for i in range(5):
    plt.semilogy(nnpn, noise10MHz_1[i],color = colors[i], marker = 'o', l
    plt.semilogy(nnpn, noise10MHz_2[i],color = colors[i], marker = '*', l
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Nnpn per group', fontsize=16)
plt.xticks(nnpn, fontsize = 16)
plt.yticks(fontsize = 16)
plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=5, bbox_to_anchor=(0.5, 1.0))
plt.title('Noise current density measured at 10MHz',fontsize = 18)
```

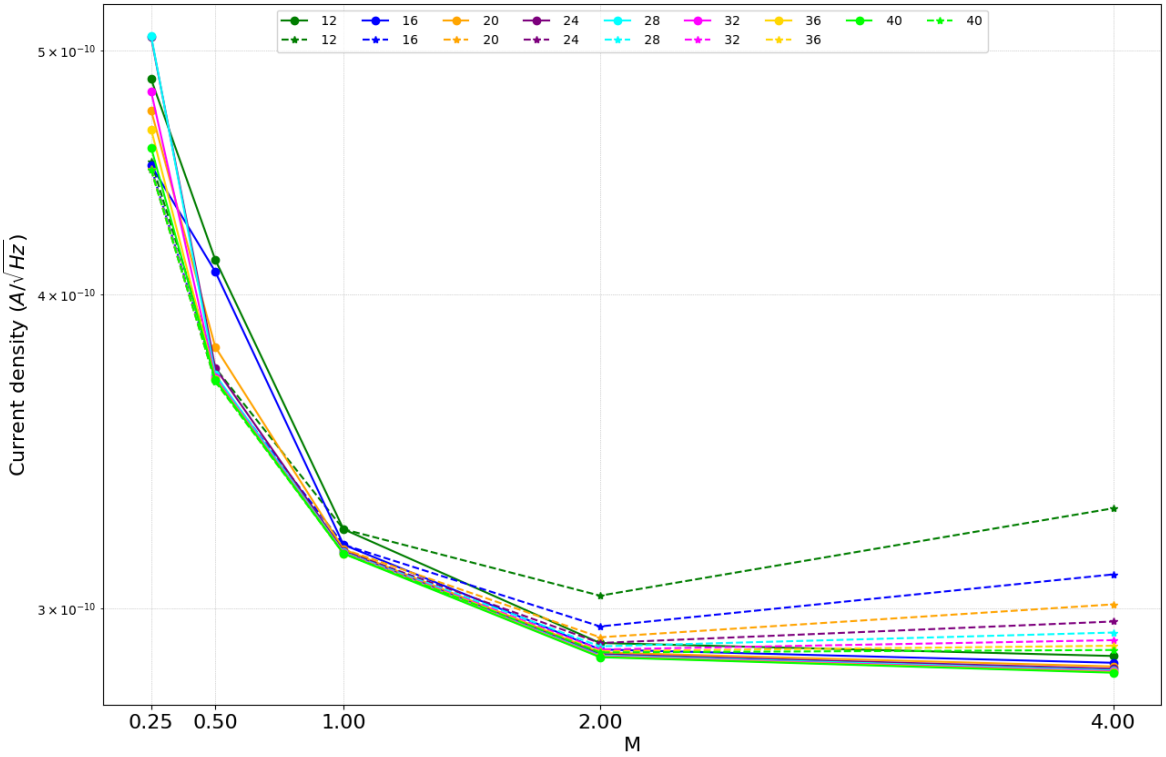
Out[34]:
```
Text(0.5, 1.0, 'Noise current density measured at 10MHz')
```

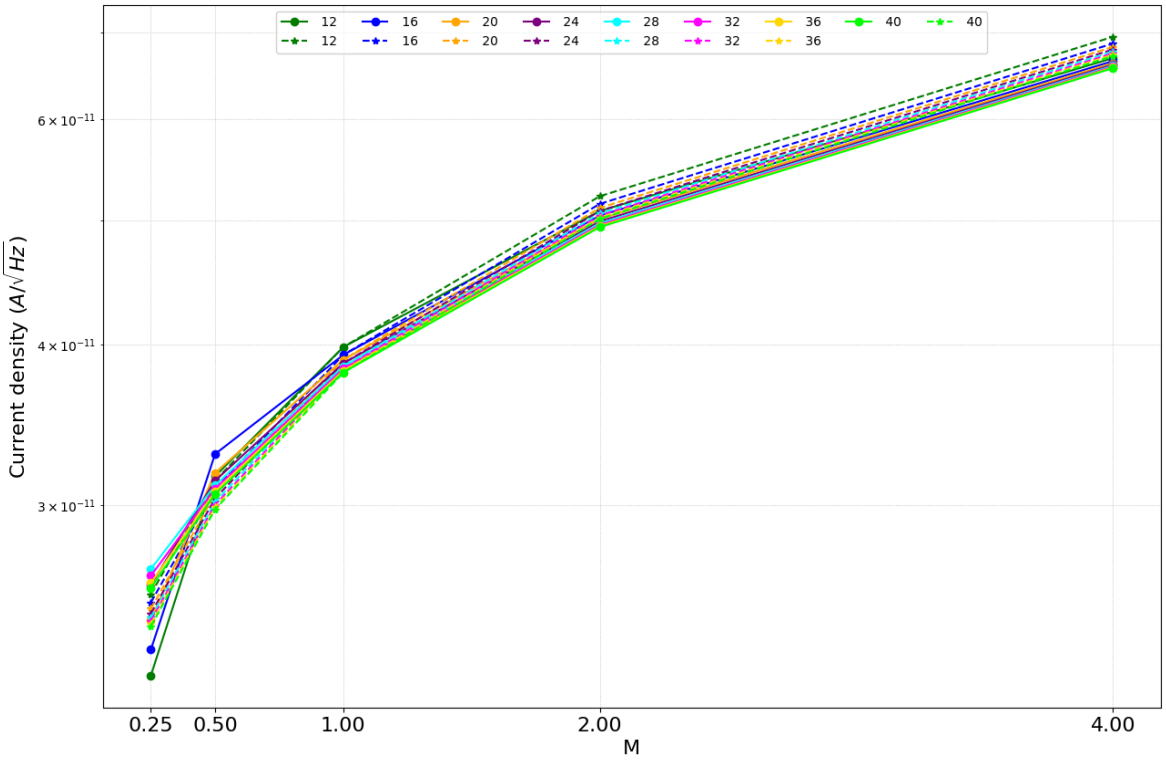


In [35]:
```
n_plot1Hz_1 = [[],[],[],[],[],[],[],[],[]]
n_plot1Hz_2 = [[],[],[],[],[],[],[],[],[]]
n_plot10MHz_1 = [[],[],[],[],[],[],[],[],[]]
n_plot10MHz_2 = [[],[],[],[],[],[],[],[],[]]
# noise1Hz_1
# noise1Hz_2
for i in range(5):  # Loop through the outer lists (5 of them)
    for j in range(9):  # Loop through the inner lists (9 of them)
        n_plot1Hz_1[j].append(noise1Hz_1[i][j])
        n_plot1Hz_2[j].append(noise1Hz_2[i][j])
        n_plot10MHz_1[j].append(noise10MHz_1[i][j])
        n_plot10MHz_2[j].append(noise10MHz_2[i][j])
```

In [36]:
```
plt.figure(figsize = (15,10))
for i in range(1,9):
    plt.semilogy(M, n_plot1Hz_1[i],color = colors[i], marker = 'o', label
    plt.semilogy(M, n_plot1Hz_2[i],color = colors[i], marker = '*', lines
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('M', fontsize=16)
plt.xticks(M, fontsize = 16)
plt.yticks(fontsize = 16)
# plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=9, bbox_to_anchor=(0.5, 1.0))
plt.figure(figsize = (15,10))
for i in range(1,9):
    plt.semilogy(M, n_plot10MHz_1[i],color = colors[i], marker = 'o', lab
    plt.semilogy(M, n_plot10MHz_2[i],color = colors[i], marker = '*', lin
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('M', fontsize=16)
plt.xticks(M, fontsize = 16)
plt.yticks(fontsize = 16)
# plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=9, bbox_to_anchor=(0.5, 1.0))
```

Out[36]: <matplotlib.legend.Legend at 0x1574ab970>





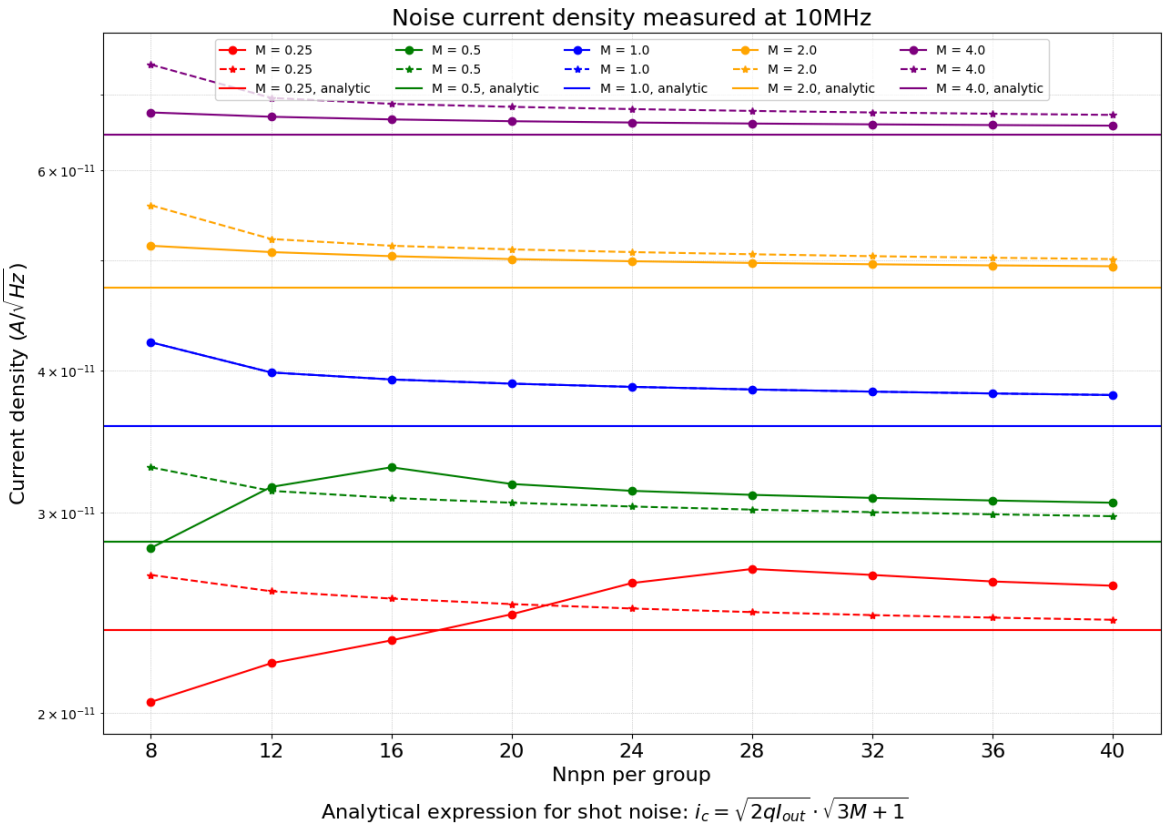
Analytical

In [37]:
```
Iout = 1e-3 #A

q = 1.6e-19 #C

def white_noise1(M, Iout):
    Iref = Iout/M
    # return np.sqrt(3*(np.sqrt(M*2*q*Iout))**2 + 2*q*Iout)
    return np.sqrt(2*q*Iout*(3 * M + 1))
```

```
wn_analytic = white_noise1(M, Iout)
print(wn_analytic)
```

```
[2.36643191e-11 2.82842712e-11 3.57770876e-11 4.73286383e-11
 6.44980620e-11]
```

In [38]:
```
plt.figure(figsize = (15,10))
for i in range(5):
    plt.semilogy(nnpn, noise10MHz_1[i],color = colors[i], marker = 'o', l
    plt.semilogy(nnpn, noise10MHz_2[i],color = colors[i], marker = '*', l
    plt.axhline(y = white_noise1(M[i], Iout), color = colors[i], marker =

plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($A/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel('Nnpn per group', fontsize=16)
plt.xticks(nnpn, fontsize = 16)
plt.yticks(fontsize = 16)
plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=5, bbox_to_anchor=(0.5, 1.0))
plt.title('Noise current density measured at 10MHz',fontsize = 18)
note = r'Analytical expression for shot noise: $i_c = \sqrt{2qI_{out}} \c
plt.figtext(0.5, 0.01, note, ha="center", fontsize=16, va="bottom")
plt.savefig('noise10MHz.png')
```



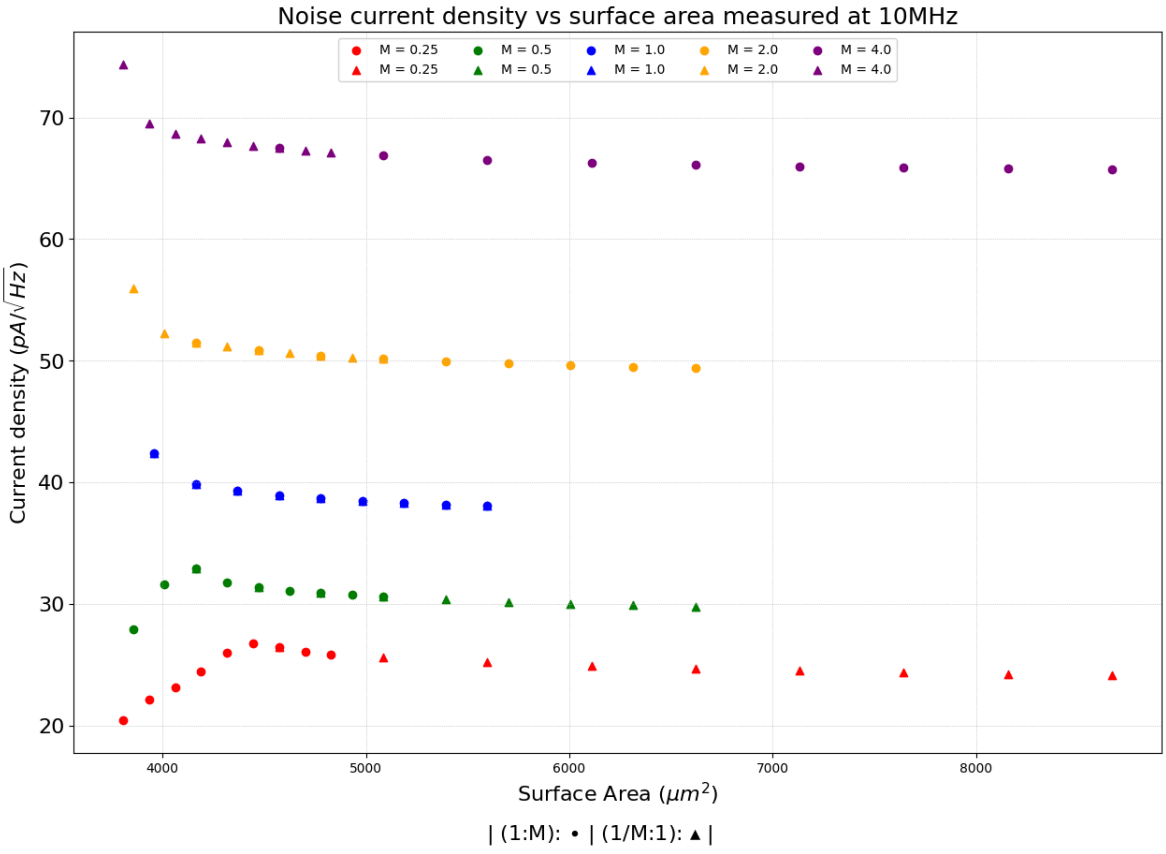
Calculate surface area

In [39]:
```
Spnp = 9.49 * 8.5 #um^2 (can touch btrench?)
Snpn = 5.7 * 4.49 #um^2 doesn't count the infusion
def S1(M, nnpn):
    return 22 * Spnp * 2 +   nnpn * Snpn + M * nnpn * Snpn
def S2(M, nnpn):
    return 22 * Spnp * 2 +   nnpn * Snpn / M +  nnpn * Snpn
```

In [40]:
```
#calculate the surface area
A1 = [[],[],[],[],[]]
A2 = [[],[],[],[],[]]
for i in range(5):
    for j in range(9):
        A1[i].append(S1(M[i], nnpn[j]))
        A2[i].append(S2(M[i], nnpn[j]))
```

In [41]:
```
#scatter plot?

plt.figure(figsize = (15,10))
for i in range(5):
    plt.scatter(A1[i], np.array(noise10MHz_1[i])*1e12,color = colors[i],
    plt.scatter(A2[i], np.array(noise10MHz_2[i])*1e12,color = colors[i],
    # plt.axhline(y = white_noise1(M[i], Iout), color = colors[i], marker
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel(r'Surface Area $(\mu m^2)$', fontsize=16)
# plt.xticks(nnpn, fontsize = 16)
plt.yticks(fontsize = 16)
# plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=5, bbox_to_anchor=(0.5, 1.0))
plt.title('Noise current density vs surface area measured at 10MHz',fonts
# # note = r'Analytical expression for shot noise: $i_c = \sqrt{2qI_{out}
# plt.figtext(0.5, 0.01, note, ha="center", fontsize=16, va="bottom")
note =  r'| (1:M): $\bullet$ | (1/M:1): $\blacktriangle$ |'
plt.figtext(0.5, 0.01, note , ha="center", fontsize=16, va="bottom")
```

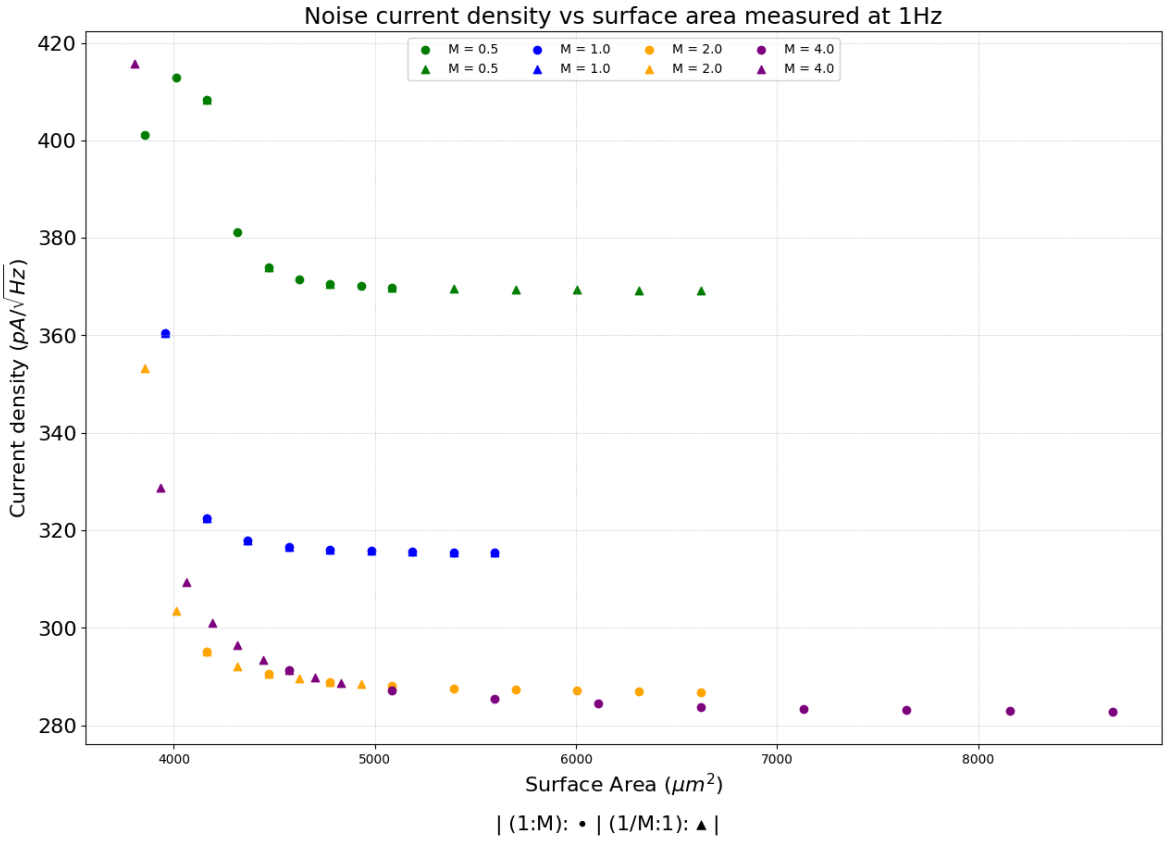
Out[41]:
```
Text(0.5, 0.01, '| (1:M): $\\bullet$ | (1/M:1): $\\blacktriangle$ |')
```



In [42]:
```
plt.figure(figsize = (15,10))
for i in range(1,5):
```

```
    plt.scatter(A1[i], np.array(noise1Hz_1[i])*1e12,color = colors[i], ma
    plt.scatter(A2[i], np.array(noise1Hz_2[i])*1e12,color = colors[i], ma
    # plt.axhline(y = white_noise1(M[i], Iout), color = colors[i], marker
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'Current density ($pA/ \sqrt{Hz}$)', fontsize=16)
plt.xlabel(r'Surface Area $(\mu m^2)$', fontsize=16)
# plt.xticks(nnpn, fontsize = 16)
plt.yticks(fontsize = 16)
# plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=4, bbox_to_anchor=(0.5, 1))
plt.title('Noise current density vs surface area measured at 1Hz',fontsiz
# # note = r'Analytical expression for shot noise: $i_c = \sqrt{2qI_{out}
# plt.figtext(0.5, 0.01, note, ha="center", fontsize=16, va="bottom")
note =  r'| (1:M): $\bullet$ | (1/M:1): $\blacktriangle$ |'
plt.figtext(0.5, 0.01, note , ha="center", fontsize=16, va="bottom")
```

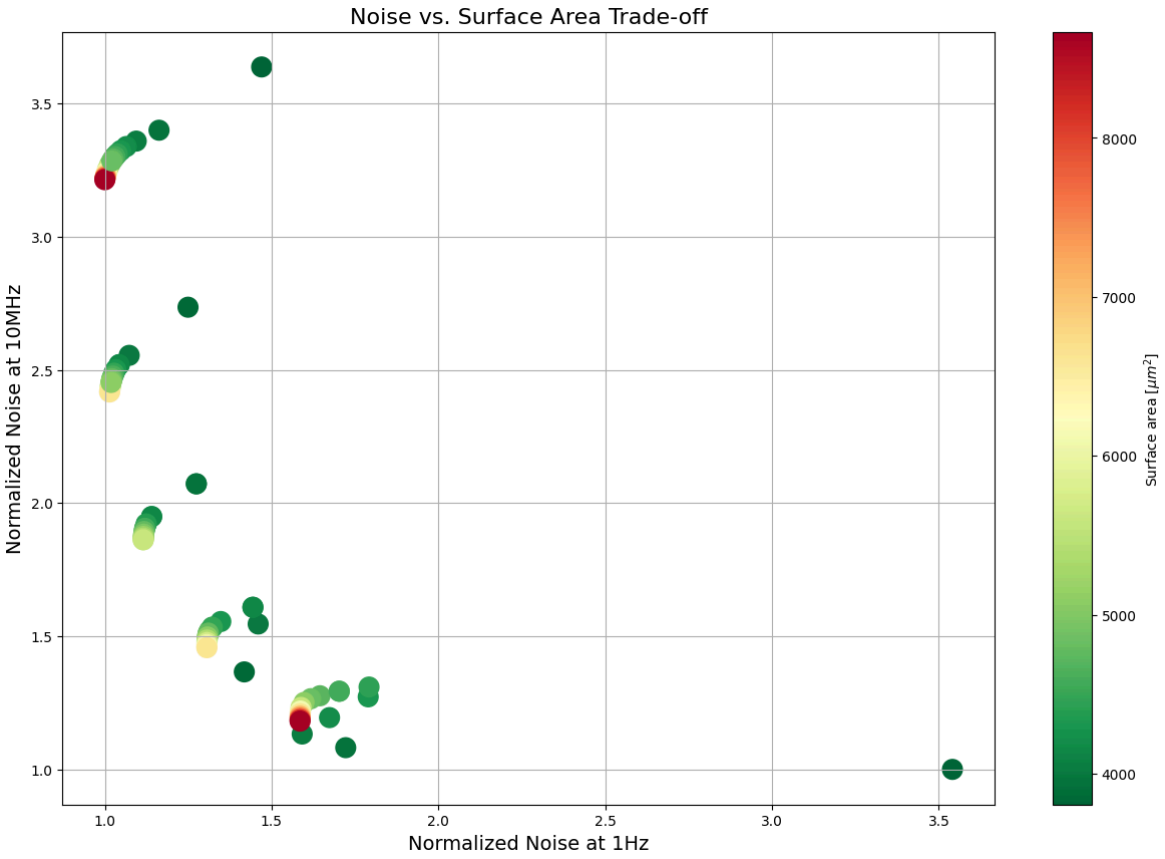
Out[42]: `Text(0.5, 0.01, '| (1:M): $\\bullet$ | (1/M:1): $\\blacktriangle$ |')`



In [106…

```
# Sample data (flatten data)
noise_1Hz = [noise1Hz_1, noise1Hz_2]
noise_1Hz = np.array(noise_1Hz).flatten()

noise_10MHz = [noise10MHz_1, noise10MHz_2]
noise_10MHz = np.array(noise_10MHz).flatten()

area = [A1,A2]
area = np.array(area).flatten()

# Normalize the data
noise_1Hz_norm = noise_1Hz / np.min(noise_1Hz)
noise_10MHz_norm = noise_10MHz / np.min(noise_10MHz)
area_norm = area / np.min(area)

# Weights (adjust these based on your priorities)
w1 = 0.5  # Weight for noise at 1Hz   (we focuse on flicker noise)
```

```python
w2 = 0.5  # Weight for noise at 10MHz
# w3 = 0.3  # Weight for surface area

# Calculate the combined metric (weighted sum)
combined_metric = w1 * noise_1Hz_norm + w2 * noise_10MHz_norm

# Rank the data points (lower is better)
rank_indices = np.argsort(combined_metric)
ranked_noise_1Hz = noise_1Hz[rank_indices]
ranked_noise_10MHz = noise_10MHz[rank_indices]
# ranked_area = area[rank_indices]
# ranked_metric = combined_metric[rank_indices]

# Scatter plot with color-coded combined metric
plt.figure(figsize=(15,10))
scatter = plt.scatter(noise_1Hz_norm, noise_10MHz_norm, c=area, cmap='RdY
plt.colorbar(scatter, label=r'Surface area [$\mu m^2$] ')
plt.xlabel('Normalized Noise at 1Hz', fontsize=14)
plt.ylabel('Normalized Noise at 10MHz', fontsize=14)
plt.title('Noise vs. Surface Area Trade-off', fontsize=16)

# Annotate points with rank, surface area, and original index
for i, txt in enumerate(ranked_area):
    original_index = rank_indices[i]  # Get the original index
    # plt.annotate(f'{i+1} ({original_index})',
                 # (noise_1Hz_norm[original_index], noise_10MHz_norm[orig
                 # textcoords="offset points", xytext=(5,5), ha='center',
# plt.xlim(1,2)
plt.grid(True)
plt.savefig('Noise_optimize.png')
plt.show()

# Print ranked data with original index
print("Ranked Data:")
best_index = rank_indices[0]  # Get the original index of the best-ranked
print(f"The best-ranked solution is at index: {best_index}")

for i in range(len(ranked_metric)):
    original_index = rank_indices[i]
    print(f"Rank {i+1} (Index: {original_index}): Noise 1Hz = {ranked_noi
```


Noise vs. Surface Area Trade-off
Normalized Noise at 10MHz
Normalized Noise at 1Hz
Surface area [μm^2]
1.0
1.5
2.0
2.5
3.0
3.5
4000
5000
6000
7000
8000

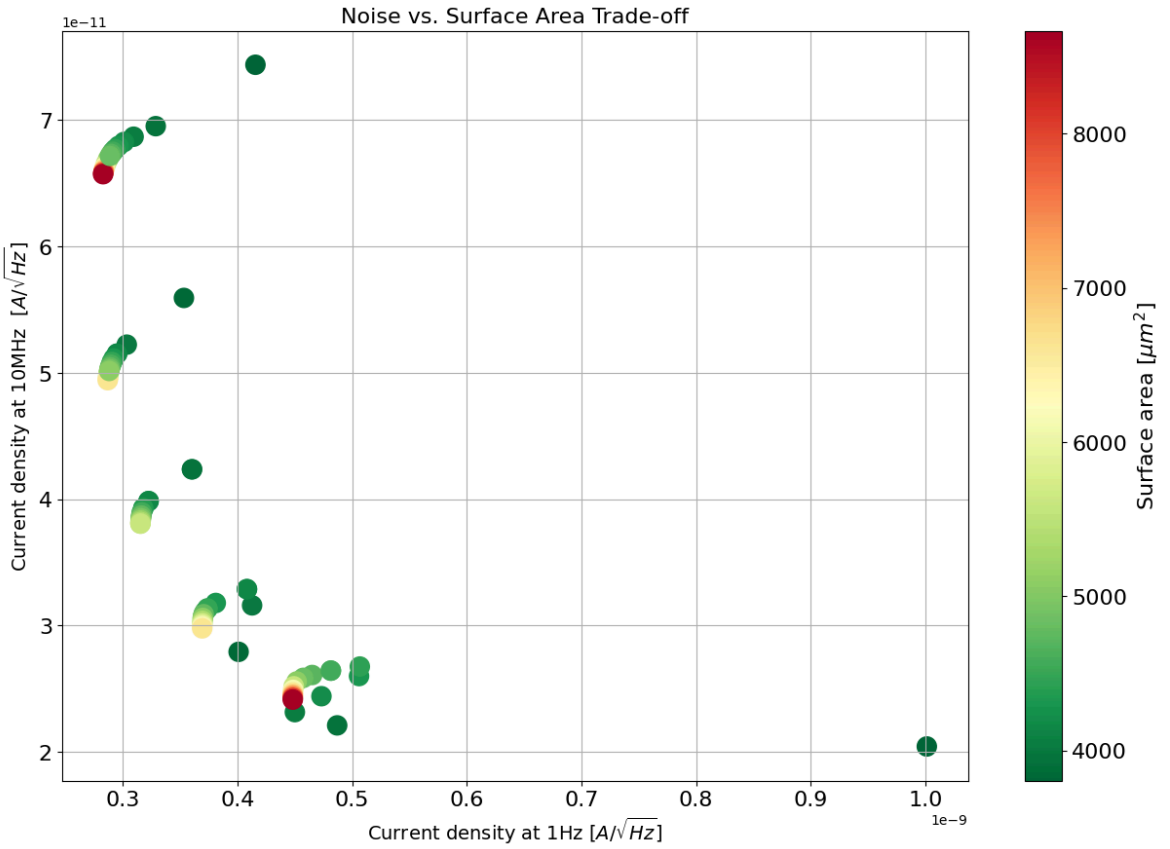
```
Ranked Data:
The best-ranked solution is at index: 2
Rank 1 (Index: 2): Noise 1Hz = 4.50e+02 pA/sqrt(Hz), Noise 10MHz = 2.32e+0
1 pA/sqrt(Hz)
Rank 2 (Index: 62): Noise 1Hz = 3.69e+02 pA/sqrt(Hz), Noise 10MHz = 2.98e+
01 pA/sqrt(Hz)
Rank 3 (Index: 53): Noise 1Hz = 4.48e+02 pA/sqrt(Hz), Noise 10MHz = 2.41e+
01 pA/sqrt(Hz)
Rank 4 (Index: 61): Noise 1Hz = 3.69e+02 pA/sqrt(Hz), Noise 10MHz = 2.99e+
01 pA/sqrt(Hz)
Rank 5 (Index: 52): Noise 1Hz = 4.48e+02 pA/sqrt(Hz), Noise 10MHz = 2.43e+
01 pA/sqrt(Hz)
Rank 6 (Index: 60): Noise 1Hz = 3.69e+02 pA/sqrt(Hz), Noise 10MHz = 3.00e+
01 pA/sqrt(Hz)
Rank 7 (Index: 51): Noise 1Hz = 4.48e+02 pA/sqrt(Hz), Noise 10MHz = 2.44e+
01 pA/sqrt(Hz)
Rank 8 (Index: 59): Noise 1Hz = 3.69e+02 pA/sqrt(Hz), Noise 10MHz = 3.02e+
01 pA/sqrt(Hz)
Rank 9 (Index: 9): Noise 1Hz = 4.01e+02 pA/sqrt(Hz), Noise 10MHz = 2.79e+0
1 pA/sqrt(Hz)
Rank 10 (Index: 50): Noise 1Hz = 4.48e+02 pA/sqrt(Hz), Noise 10MHz = 2.45e
+01 pA/sqrt(Hz)
Rank 11 (Index: 58): Noise 1Hz = 3.70e+02 pA/sqrt(Hz), Noise 10MHz = 3.04e
+01 pA/sqrt(Hz)
Rank 12 (Index: 49): Noise 1Hz = 4.48e+02 pA/sqrt(Hz), Noise 10MHz = 2.47e
+01 pA/sqrt(Hz)
Rank 13 (Index: 1): Noise 1Hz = 4.87e+02 pA/sqrt(Hz), Noise 10MHz = 2.21e+
01 pA/sqrt(Hz)
Rank 14 (Index: 57): Noise 1Hz = 3.70e+02 pA/sqrt(Hz), Noise 10MHz = 3.06e
+01 pA/sqrt(Hz)
Rank 15 (Index: 17): Noise 1Hz = 3.70e+02 pA/sqrt(Hz), Noise 10MHz = 3.06e
+01 pA/sqrt(Hz)
Rank 16 (Index: 48): Noise 1Hz = 4.49e+02 pA/sqrt(Hz), Noise 10MHz = 2.49e
+01 pA/sqrt(Hz)
Rank 17 (Index: 16): Noise 1Hz = 3.70e+02 pA/sqrt(Hz), Noise 10MHz = 3.07e
+01 pA/sqrt(Hz)
Rank 18 (Index: 47): Noise 1Hz = 4.49e+02 pA/sqrt(Hz), Noise 10MHz = 2.52e
+01 pA/sqrt(Hz)
Rank 19 (Index: 56): Noise 1Hz = 3.71e+02 pA/sqrt(Hz), Noise 10MHz = 3.09e
+01 pA/sqrt(Hz)
Rank 20 (Index: 15): Noise 1Hz = 3.71e+02 pA/sqrt(Hz), Noise 10MHz = 3.09e
+01 pA/sqrt(Hz)
Rank 21 (Index: 14): Noise 1Hz = 3.72e+02 pA/sqrt(Hz), Noise 10MHz = 3.11e
+01 pA/sqrt(Hz)
Rank 22 (Index: 46): Noise 1Hz = 4.52e+02 pA/sqrt(Hz), Noise 10MHz = 2.56e
+01 pA/sqrt(Hz)
Rank 23 (Index: 55): Noise 1Hz = 3.74e+02 pA/sqrt(Hz), Noise 10MHz = 3.14e
+01 pA/sqrt(Hz)
Rank 24 (Index: 13): Noise 1Hz = 3.74e+02 pA/sqrt(Hz), Noise 10MHz = 3.14e
+01 pA/sqrt(Hz)
Rank 25 (Index: 3): Noise 1Hz = 4.73e+02 pA/sqrt(Hz), Noise 10MHz = 2.44e+
01 pA/sqrt(Hz)
Rank 26 (Index: 8): Noise 1Hz = 4.57e+02 pA/sqrt(Hz), Noise 10MHz = 2.59e+
01 pA/sqrt(Hz)
Rank 27 (Index: 12): Noise 1Hz = 3.81e+02 pA/sqrt(Hz), Noise 10MHz = 3.18e
+01 pA/sqrt(Hz)
Rank 28 (Index: 7): Noise 1Hz = 4.65e+02 pA/sqrt(Hz), Noise 10MHz = 2.61e+
01 pA/sqrt(Hz)
Rank 29 (Index: 71): Noise 1Hz = 3.15e+02 pA/sqrt(Hz), Noise 10MHz = 3.81e
+01 pA/sqrt(Hz)
```

Rank 30 (Index: 26): Noise 1Hz = 3.15e+02 pA/sqrt(Hz), Noise 10MHz = 3.81e+01 pA/sqrt(Hz)
Rank 31 (Index: 70): Noise 1Hz = 3.15e+02 pA/sqrt(Hz), Noise 10MHz = 3.82e+01 pA/sqrt(Hz)
Rank 32 (Index: 25): Noise 1Hz = 3.15e+02 pA/sqrt(Hz), Noise 10MHz = 3.82e+01 pA/sqrt(Hz)
Rank 33 (Index: 69): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.83e+01 pA/sqrt(Hz)
Rank 34 (Index: 24): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.83e+01 pA/sqrt(Hz)
Rank 35 (Index: 6): Noise 1Hz = 4.82e+02 pA/sqrt(Hz), Noise 10MHz = 2.64e+01 pA/sqrt(Hz)
Rank 36 (Index: 45): Noise 1Hz = 4.82e+02 pA/sqrt(Hz), Noise 10MHz = 2.64e+01 pA/sqrt(Hz)
Rank 37 (Index: 68): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.85e+01 pA/sqrt(Hz)
Rank 38 (Index: 23): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.85e+01 pA/sqrt(Hz)
Rank 39 (Index: 10): Noise 1Hz = 4.13e+02 pA/sqrt(Hz), Noise 10MHz = 3.16e+01 pA/sqrt(Hz)
Rank 40 (Index: 67): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.87e+01 pA/sqrt(Hz)
Rank 41 (Index: 22): Noise 1Hz = 3.16e+02 pA/sqrt(Hz), Noise 10MHz = 3.87e+01 pA/sqrt(Hz)
Rank 42 (Index: 21): Noise 1Hz = 3.17e+02 pA/sqrt(Hz), Noise 10MHz = 3.90e+01 pA/sqrt(Hz)
Rank 43 (Index: 66): Noise 1Hz = 3.17e+02 pA/sqrt(Hz), Noise 10MHz = 3.90e+01 pA/sqrt(Hz)
Rank 44 (Index: 65): Noise 1Hz = 3.18e+02 pA/sqrt(Hz), Noise 10MHz = 3.93e+01 pA/sqrt(Hz)
Rank 45 (Index: 20): Noise 1Hz = 3.18e+02 pA/sqrt(Hz), Noise 10MHz = 3.93e+01 pA/sqrt(Hz)
Rank 46 (Index: 54): Noise 1Hz = 4.08e+02 pA/sqrt(Hz), Noise 10MHz = 3.29e+01 pA/sqrt(Hz)
Rank 47 (Index: 11): Noise 1Hz = 4.08e+02 pA/sqrt(Hz), Noise 10MHz = 3.29e+01 pA/sqrt(Hz)
Rank 48 (Index: 4): Noise 1Hz = 5.06e+02 pA/sqrt(Hz), Noise 10MHz = 2.60e+01 pA/sqrt(Hz)
Rank 49 (Index: 64): Noise 1Hz = 3.23e+02 pA/sqrt(Hz), Noise 10MHz = 3.98e+01 pA/sqrt(Hz)
Rank 50 (Index: 19): Noise 1Hz = 3.23e+02 pA/sqrt(Hz), Noise 10MHz = 3.98e+01 pA/sqrt(Hz)
Rank 51 (Index: 5): Noise 1Hz = 5.07e+02 pA/sqrt(Hz), Noise 10MHz = 2.68e+01 pA/sqrt(Hz)
Rank 52 (Index: 18): Noise 1Hz = 3.60e+02 pA/sqrt(Hz), Noise 10MHz = 4.24e+01 pA/sqrt(Hz)
Rank 53 (Index: 63): Noise 1Hz = 3.60e+02 pA/sqrt(Hz), Noise 10MHz = 4.24e+01 pA/sqrt(Hz)
Rank 54 (Index: 35): Noise 1Hz = 2.87e+02 pA/sqrt(Hz), Noise 10MHz = 4.94e+01 pA/sqrt(Hz)
Rank 55 (Index: 34): Noise 1Hz = 2.87e+02 pA/sqrt(Hz), Noise 10MHz = 4.95e+01 pA/sqrt(Hz)
Rank 56 (Index: 33): Noise 1Hz = 2.87e+02 pA/sqrt(Hz), Noise 10MHz = 4.96e+01 pA/sqrt(Hz)
Rank 57 (Index: 32): Noise 1Hz = 2.87e+02 pA/sqrt(Hz), Noise 10MHz = 4.98e+01 pA/sqrt(Hz)
Rank 58 (Index: 31): Noise 1Hz = 2.88e+02 pA/sqrt(Hz), Noise 10MHz = 4.99e+01 pA/sqrt(Hz)
Rank 59 (Index: 30): Noise 1Hz = 2.88e+02 pA/sqrt(Hz), Noise 10MHz = 5.01e+01 pA/sqrt(Hz)

Rank 60 (Index: 80): Noise 1Hz = 2.88e+02 pA/sqrt(Hz), Noise 10MHz = 5.01e+01 pA/sqrt(Hz)
Rank 61 (Index: 79): Noise 1Hz = 2.88e+02 pA/sqrt(Hz), Noise 10MHz = 5.03e+01 pA/sqrt(Hz)
Rank 62 (Index: 29): Noise 1Hz = 2.89e+02 pA/sqrt(Hz), Noise 10MHz = 5.04e+01 pA/sqrt(Hz)
Rank 63 (Index: 78): Noise 1Hz = 2.89e+02 pA/sqrt(Hz), Noise 10MHz = 5.04e+01 pA/sqrt(Hz)
Rank 64 (Index: 77): Noise 1Hz = 2.90e+02 pA/sqrt(Hz), Noise 10MHz = 5.06e+01 pA/sqrt(Hz)
Rank 65 (Index: 28): Noise 1Hz = 2.91e+02 pA/sqrt(Hz), Noise 10MHz = 5.09e+01 pA/sqrt(Hz)
Rank 66 (Index: 76): Noise 1Hz = 2.91e+02 pA/sqrt(Hz), Noise 10MHz = 5.09e+01 pA/sqrt(Hz)
Rank 67 (Index: 75): Noise 1Hz = 2.92e+02 pA/sqrt(Hz), Noise 10MHz = 5.11e+01 pA/sqrt(Hz)
Rank 68 (Index: 27): Noise 1Hz = 2.95e+02 pA/sqrt(Hz), Noise 10MHz = 5.15e+01 pA/sqrt(Hz)
Rank 69 (Index: 74): Noise 1Hz = 2.95e+02 pA/sqrt(Hz), Noise 10MHz = 5.15e+01 pA/sqrt(Hz)
Rank 70 (Index: 73): Noise 1Hz = 3.03e+02 pA/sqrt(Hz), Noise 10MHz = 5.22e+01 pA/sqrt(Hz)
Rank 71 (Index: 72): Noise 1Hz = 3.53e+02 pA/sqrt(Hz), Noise 10MHz = 5.59e+01 pA/sqrt(Hz)
Rank 72 (Index: 44): Noise 1Hz = 2.83e+02 pA/sqrt(Hz), Noise 10MHz = 6.57e+01 pA/sqrt(Hz)
Rank 73 (Index: 43): Noise 1Hz = 2.83e+02 pA/sqrt(Hz), Noise 10MHz = 6.58e+01 pA/sqrt(Hz)
Rank 74 (Index: 42): Noise 1Hz = 2.83e+02 pA/sqrt(Hz), Noise 10MHz = 6.59e+01 pA/sqrt(Hz)
Rank 75 (Index: 41): Noise 1Hz = 2.83e+02 pA/sqrt(Hz), Noise 10MHz = 6.60e+01 pA/sqrt(Hz)
Rank 76 (Index: 40): Noise 1Hz = 2.84e+02 pA/sqrt(Hz), Noise 10MHz = 6.61e+01 pA/sqrt(Hz)
Rank 77 (Index: 39): Noise 1Hz = 2.84e+02 pA/sqrt(Hz), Noise 10MHz = 6.63e+01 pA/sqrt(Hz)
Rank 78 (Index: 38): Noise 1Hz = 2.85e+02 pA/sqrt(Hz), Noise 10MHz = 6.65e+01 pA/sqrt(Hz)
Rank 79 (Index: 37): Noise 1Hz = 2.87e+02 pA/sqrt(Hz), Noise 10MHz = 6.69e+01 pA/sqrt(Hz)
Rank 80 (Index: 89): Noise 1Hz = 2.89e+02 pA/sqrt(Hz), Noise 10MHz = 6.72e+01 pA/sqrt(Hz)
Rank 81 (Index: 88): Noise 1Hz = 2.90e+02 pA/sqrt(Hz), Noise 10MHz = 6.73e+01 pA/sqrt(Hz)
Rank 82 (Index: 36): Noise 1Hz = 2.91e+02 pA/sqrt(Hz), Noise 10MHz = 6.75e+01 pA/sqrt(Hz)
Rank 83 (Index: 87): Noise 1Hz = 2.91e+02 pA/sqrt(Hz), Noise 10MHz = 6.75e+01 pA/sqrt(Hz)
Rank 84 (Index: 86): Noise 1Hz = 2.93e+02 pA/sqrt(Hz), Noise 10MHz = 6.77e+01 pA/sqrt(Hz)
Rank 85 (Index: 85): Noise 1Hz = 2.96e+02 pA/sqrt(Hz), Noise 10MHz = 6.79e+01 pA/sqrt(Hz)
Rank 86 (Index: 84): Noise 1Hz = 3.01e+02 pA/sqrt(Hz), Noise 10MHz = 6.83e+01 pA/sqrt(Hz)
Rank 87 (Index: 83): Noise 1Hz = 3.09e+02 pA/sqrt(Hz), Noise 10MHz = 6.87e+01 pA/sqrt(Hz)
Rank 88 (Index: 0): Noise 1Hz = 1.00e+03 pA/sqrt(Hz), Noise 10MHz = 2.04e+01 pA/sqrt(Hz)
Rank 89 (Index: 82): Noise 1Hz = 3.29e+02 pA/sqrt(Hz), Noise 10MHz = 6.95e+01 pA/sqrt(Hz)

```
Rank 90 (Index: 81): Noise 1Hz = 4.16e+02 pA/sqrt(Hz), Noise 10MHz = 7.44e
+01 pA/sqrt(Hz)
```

In [116…
```
plt.figure(figsize = (15,10))
# plt.scatter(noise_1Hz, noise_10MHz)
scatter = plt.scatter(noise_1Hz, noise_10MHz, c=area, cmap='RdYlGn_r', s=
cbar = plt.colorbar(scatter, label=r'Surface area [$\mu m^2$]')
cbar.ax.tick_params(labelsize=16)  # Set the font size of the colorbar ti
cbar.ax.set_ylabel(r'Surface area [$\mu m^2$]', fontsize=16)  # Set the f
plt.xlabel(r'Current density at 1Hz [$A/\sqrt{Hz}$]', fontsize=14)
plt.ylabel('Current density at 10MHz  [$A/\sqrt{Hz}$]', fontsize=14)
plt.title('Noise vs. Surface Area Trade-off', fontsize=16)
plt.xticks(fontsize = 16)
plt.yticks(fontsize = 16)
# plt.xlim(1,2)
plt.grid(True)
plt.savefig('Noise_optimize.png')
```



In [44]:
```
# noise_square_1 = noise_1
# noise_square_2 = noise_2
# for i in range(1,45):
#     noise_square_1[i] = noise_square_1[i]**2
#     noise_square_2[i] = noise_square_2[i]**2
# print(noise_square_1)
```

In [45]:
```
def integral(frequency, data):
    S = 0
    for i in range(1,len(frequency)):
        S = S + 0.5 * (data[i-1]**2 + data[i]**2) * (frequency[i] - frequ
    return S
```

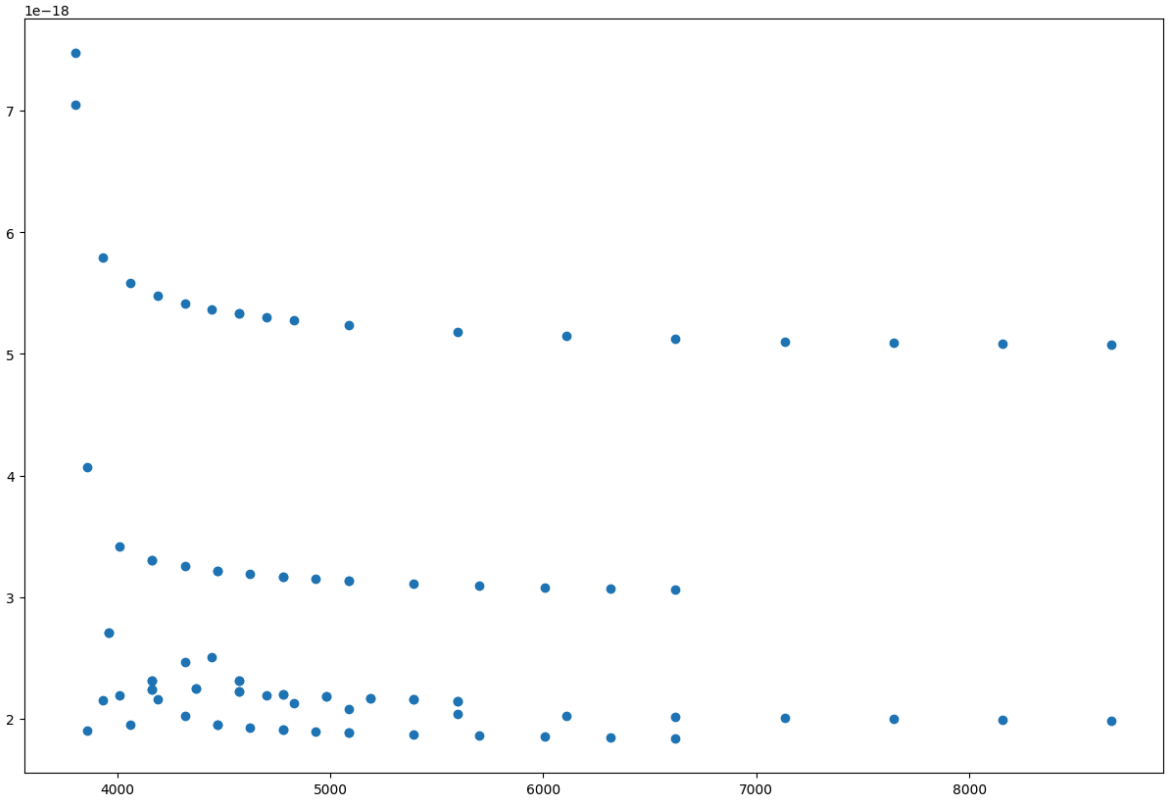
In [46]:
```
RMS_1 = [[],[],[],[],[]]
RMS_2 = [[],[],[],[],[]]
```

```
#we want to limit the bw from 1 to 1000


for i in range(5):
    for j in range(9):
        try:
            index = 1 + i + j * 5  # Calculate the index directly
            RMS_1[i].append(integral(noise_1[0][0:300],noise_1[index][0:3
            RMS_2[i].append(integral(noise_1[0][0:300],noise_2[index][0:3
        except IndexError:
            print(f"Error: Index {index} out of range for noise_1")
            break
```

In [47]:
```
RMS = np.array([RMS_1,RMS_2]).flatten()
```

In [48]:
```
plt.figure(figsize = (15,10))
plt.scatter(area, RMS)
```

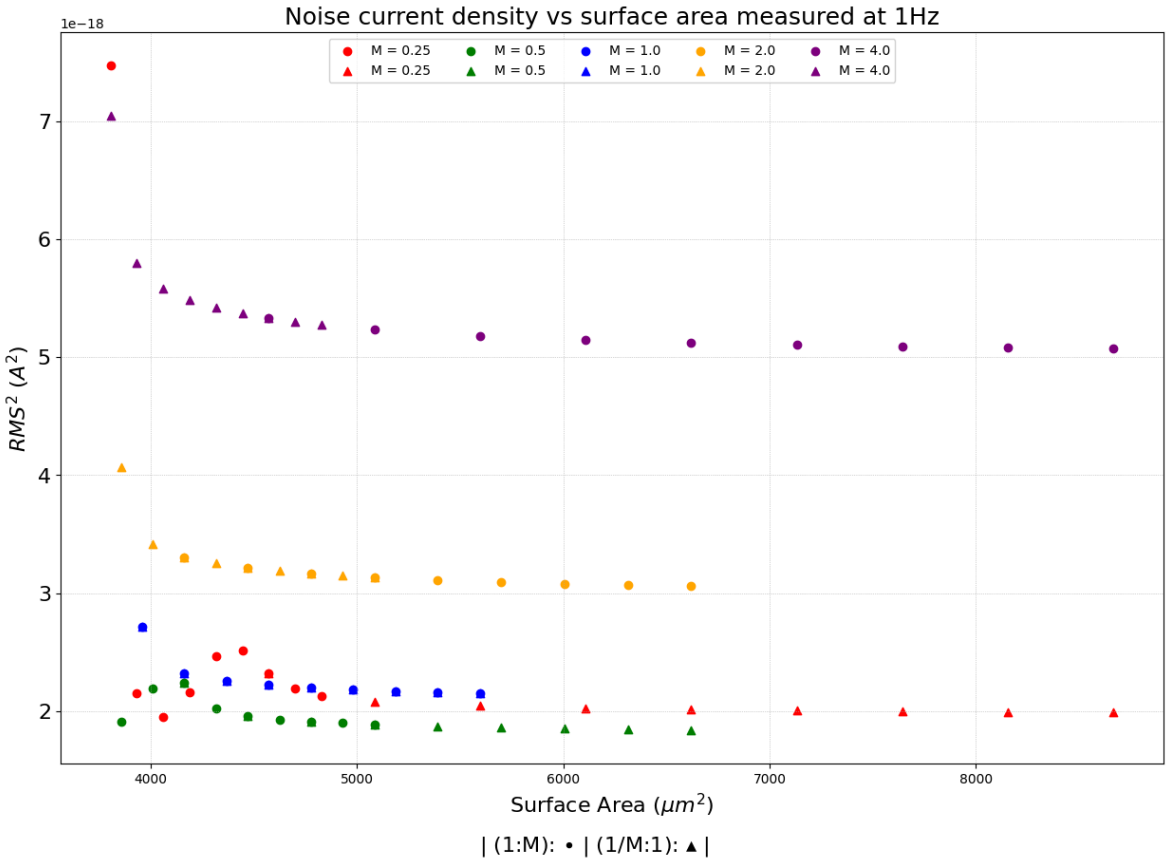
Out[48]: <matplotlib.collections.PathCollection at 0x1471d2b90>



In [49]:
```
plt.figure(figsize = (15,10))
for i in range(5):
    plt.scatter(A1[i], RMS_1[i],color = colors[i], marker = 'o', label =
    plt.scatter(A2[i], RMS_2[i],color = colors[i], marker = "^",  label =
    # plt.axhline(y = white_noise1(M[i], Iout), color = colors[i], marker
plt.grid(which='both', linestyle=':', linewidth='0.5')
plt.ylabel(r'$RMS^2$ ($A^2$)', fontsize=16)
plt.xlabel(r'Surface Area $(\mu m^2)$', fontsize=16)
# plt.xticks(nnpn, fontsize = 16)
plt.yticks(fontsize = 16)
# plt.legend(fontsize = 16, loc = 'upper right')
plt.legend(loc='upper center', ncol=5, bbox_to_anchor=(0.5, 1))
plt.title('Noise current density vs surface area measured at 1Hz',fontsiz
# # note = r'Analytical expression for shot noise: $i_c = \sqrt{2qI_{out}
# plt.figtext(0.5, 0.01, note, ha="center", fontsize=16, va="bottom")
```

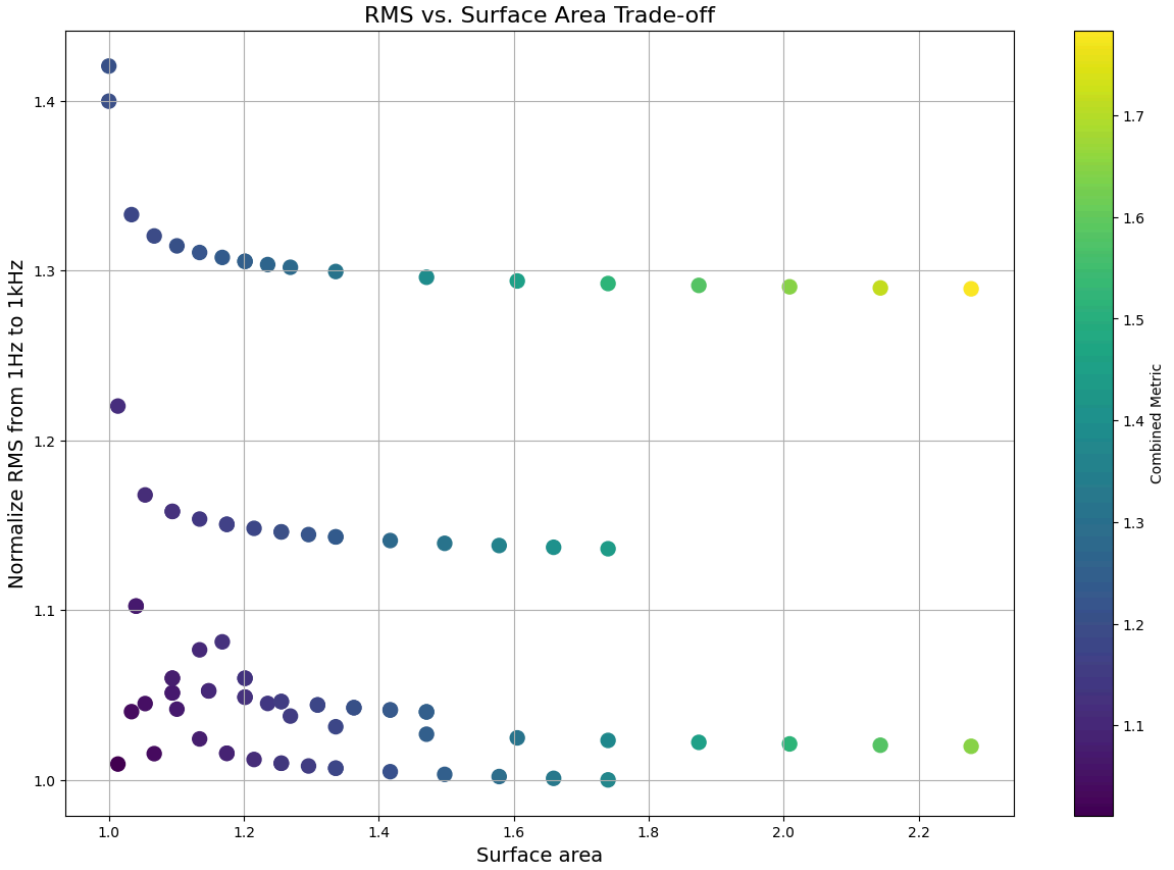
```
note =  r'| (1:M): $\bullet$ | (1/M:1): $\blacktriangle$ |'
plt.figtext(0.5, 0.01, note , ha="center", fontsize=16, va="bottom")
```

Out[49]: Text(0.5, 0.01, '| (1:M): $\\bullet$ | (1/M:1): $\\blacktriangle$ |')



In [52]:
```
# Normalize the data
RMS = np.sqrt(RMS)
RMS_norm = RMS / np.min(RMS)
# Calculate the combined metric (weighted sum)
w1 = 0.5
w2 = 0.5
combined_metric = w1 * RMS_norm + w2 * area_norm

# Rank the data points (lower is better)
rank_indices = np.argsort(combined_metric)
ranked_RMS = RMS[rank_indices]
ranked_area = area[rank_indices]
ranked_metric = combined_metric[rank_indices]

# Scatter plot with color-coded combined metric
plt.figure(figsize=(15,10))
scatter = plt.scatter(area_norm, RMS_norm, c=combined_metric, cmap='virid
plt.colorbar(scatter, label='Combined Metric')
plt.xlabel('Surface area', fontsize=14)
plt.ylabel('Normalize RMS from 1Hz to 1kHz', fontsize=14)
plt.title('RMS vs. Surface Area Trade-off', fontsize=16)

# Annotate points with rank, surface area, and original index
for i, txt in enumerate(ranked_area):
    original_index = rank_indices[i]  # Get the original index
    plt.annotate(f'{i+1} ({original_index})',
                 (RMS[original_index], noise_10MHz_norm[original_index]),
                 textcoords="offset points", xytext=(5,5), ha='center', f
# plt.xlim(1,2)
```

```
plt.grid(True)
plt.show()

# Print ranked data with original index
print("Ranked Data:")
best_index = rank_indices[0]  # Get the original index of the best-ranked
print(f"The best-ranked solution is at index: {best_index}")

for i in range(len(ranked_metric)):
    original_index = rank_indices[i]
    print(f"Rank {i+1} (Index: {original_index}): RMS = {ranked_RMS[i]:.2
```



```
Ranked Data:
The best-ranked solution is at index: 9
Rank 1 (Index: 9): RMS = 3.72e-05 A, normalise_RMS = 1.01e+00, Area = 385
6.4 um^2, normalised area = 1.0
Rank 2 (Index: 1): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 393
3.2 um^2, normalised area = 1.0
Rank 3 (Index: 2): RMS = 3.74e-05 A, normalise_RMS = 1.02e+00, Area = 406
1.1 um^2, normalised area = 1.1
Rank 4 (Index: 10): RMS = 3.85e-05 A, normalise_RMS = 1.04e+00, Area = 400
9.9 um^2, normalised area = 1.1
Rank 5 (Index: 3): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 418
9.1 um^2, normalised area = 1.1
Rank 6 (Index: 18): RMS = 4.06e-05 A, normalise_RMS = 1.10e+00, Area = 395
8.7 um^2, normalised area = 1.0
Rank 7 (Index: 63): RMS = 4.06e-05 A, normalise_RMS = 1.10e+00, Area = 395
8.7 um^2, normalised area = 1.0
Rank 8 (Index: 11): RMS = 3.87e-05 A, normalise_RMS = 1.05e+00, Area = 416
3.5 um^2, normalised area = 1.1
Rank 9 (Index: 54): RMS = 3.87e-05 A, normalise_RMS = 1.05e+00, Area = 416
3.5 um^2, normalised area = 1.1
Rank 10 (Index: 19): RMS = 3.90e-05 A, normalise_RMS = 1.06e+00, Area = 41
63.5 um^2, normalised area = 1.1
Rank 11 (Index: 64): RMS = 3.90e-05 A, normalise_RMS = 1.06e+00, Area = 41
63.5 um^2, normalised area = 1.1
Rank 12 (Index: 12): RMS = 3.77e-05 A, normalise_RMS = 1.02e+00, Area = 43
17.1 um^2, normalised area = 1.1
Rank 13 (Index: 55): RMS = 3.74e-05 A, normalise_RMS = 1.02e+00, Area = 44
70.6 um^2, normalised area = 1.2
Rank 14 (Index: 13): RMS = 3.74e-05 A, normalise_RMS = 1.02e+00, Area = 44
70.6 um^2, normalised area = 1.2
Rank 15 (Index: 20): RMS = 3.87e-05 A, normalise_RMS = 1.05e+00, Area = 43
68.2 um^2, normalised area = 1.1
Rank 16 (Index: 65): RMS = 3.87e-05 A, normalise_RMS = 1.05e+00, Area = 43
68.2 um^2, normalised area = 1.1
Rank 17 (Index: 4): RMS = 3.96e-05 A, normalise_RMS = 1.08e+00, Area = 431
7.1 um^2, normalised area = 1.1
Rank 18 (Index: 73): RMS = 4.30e-05 A, normalise_RMS = 1.17e+00, Area = 40
09.9 um^2, normalised area = 1.1
Rank 19 (Index: 14): RMS = 3.73e-05 A, normalise_RMS = 1.01e+00, Area = 46
24.2 um^2, normalised area = 1.2
Rank 20 (Index: 72): RMS = 4.49e-05 A, normalise_RMS = 1.22e+00, Area = 38
56.4 um^2, normalised area = 1.0
Rank 21 (Index: 5): RMS = 3.98e-05 A, normalise_RMS = 1.08e+00, Area = 444
5.0 um^2, normalised area = 1.2
Rank 22 (Index: 66): RMS = 3.86e-05 A, normalise_RMS = 1.05e+00, Area = 45
73.0 um^2, normalised area = 1.2
Rank 23 (Index: 21): RMS = 3.86e-05 A, normalise_RMS = 1.05e+00, Area = 45
73.0 um^2, normalised area = 1.2
Rank 24 (Index: 27): RMS = 4.26e-05 A, normalise_RMS = 1.16e+00, Area = 41
63.5 um^2, normalised area = 1.1
Rank 25 (Index: 74): RMS = 4.26e-05 A, normalise_RMS = 1.16e+00, Area = 41
63.5 um^2, normalised area = 1.1
Rank 26 (Index: 6): RMS = 3.90e-05 A, normalise_RMS = 1.06e+00, Area = 457
3.0 um^2, normalised area = 1.2
Rank 27 (Index: 45): RMS = 3.90e-05 A, normalise_RMS = 1.06e+00, Area = 45
73.0 um^2, normalised area = 1.2
Rank 28 (Index: 56): RMS = 3.72e-05 A, normalise_RMS = 1.01e+00, Area = 47
77.7 um^2, normalised area = 1.3
Rank 29 (Index: 15): RMS = 3.72e-05 A, normalise_RMS = 1.01e+00, Area = 47
77.7 um^2, normalised area = 1.3
```

Rank 30 (Index: 7): RMS = 3.85e-05 A, normalise_RMS = 1.05e+00, Area = 470
0.9 um^2, normalised area = 1.2
Rank 31 (Index: 75): RMS = 4.25e-05 A, normalise_RMS = 1.15e+00, Area = 43
17.1 um^2, normalised area = 1.1
Rank 32 (Index: 67): RMS = 3.85e-05 A, normalise_RMS = 1.05e+00, Area = 47
77.7 um^2, normalised area = 1.3
Rank 33 (Index: 22): RMS = 3.85e-05 A, normalise_RMS = 1.05e+00, Area = 47
77.7 um^2, normalised area = 1.3
Rank 34 (Index: 16): RMS = 3.71e-05 A, normalise_RMS = 1.01e+00, Area = 49
31.3 um^2, normalised area = 1.3
Rank 35 (Index: 8): RMS = 3.82e-05 A, normalise_RMS = 1.04e+00, Area = 482
8.9 um^2, normalised area = 1.3
Rank 36 (Index: 76): RMS = 4.24e-05 A, normalise_RMS = 1.15e+00, Area = 44
70.6 um^2, normalised area = 1.2
Rank 37 (Index: 28): RMS = 4.24e-05 A, normalise_RMS = 1.15e+00, Area = 44
70.6 um^2, normalised area = 1.2
Rank 38 (Index: 57): RMS = 3.71e-05 A, normalise_RMS = 1.01e+00, Area = 50
84.8 um^2, normalised area = 1.3
Rank 39 (Index: 17): RMS = 3.71e-05 A, normalise_RMS = 1.01e+00, Area = 50
84.8 um^2, normalised area = 1.3
Rank 40 (Index: 23): RMS = 3.84e-05 A, normalise_RMS = 1.04e+00, Area = 49
82.5 um^2, normalised area = 1.3
Rank 41 (Index: 68): RMS = 3.84e-05 A, normalise_RMS = 1.04e+00, Area = 49
82.5 um^2, normalised area = 1.3
Rank 42 (Index: 77): RMS = 4.23e-05 A, normalise_RMS = 1.15e+00, Area = 46
24.2 um^2, normalised area = 1.2
Rank 43 (Index: 82): RMS = 4.91e-05 A, normalise_RMS = 1.33e+00, Area = 39
33.2 um^2, normalised area = 1.0
Rank 44 (Index: 46): RMS = 3.80e-05 A, normalise_RMS = 1.03e+00, Area = 50
84.8 um^2, normalised area = 1.3
Rank 45 (Index: 83): RMS = 4.86e-05 A, normalise_RMS = 1.32e+00, Area = 40
61.1 um^2, normalised area = 1.1
Rank 46 (Index: 81): RMS = 5.15e-05 A, normalise_RMS = 1.40e+00, Area = 38
05.2 um^2, normalised area = 1.0
Rank 47 (Index: 29): RMS = 4.22e-05 A, normalise_RMS = 1.15e+00, Area = 47
77.7 um^2, normalised area = 1.3
Rank 48 (Index: 78): RMS = 4.22e-05 A, normalise_RMS = 1.15e+00, Area = 47
77.7 um^2, normalised area = 1.3
Rank 49 (Index: 24): RMS = 3.84e-05 A, normalise_RMS = 1.04e+00, Area = 51
87.2 um^2, normalised area = 1.4
Rank 50 (Index: 69): RMS = 3.84e-05 A, normalise_RMS = 1.04e+00, Area = 51
87.2 um^2, normalised area = 1.4
Rank 51 (Index: 84): RMS = 4.84e-05 A, normalise_RMS = 1.31e+00, Area = 41
89.1 um^2, normalised area = 1.1
Rank 52 (Index: 0): RMS = 5.23e-05 A, normalise_RMS = 1.42e+00, Area = 380
5.2 um^2, normalised area = 1.0
Rank 53 (Index: 58): RMS = 3.70e-05 A, normalise_RMS = 1.00e+00, Area = 53
92.0 um^2, normalised area = 1.4
Rank 54 (Index: 79): RMS = 4.21e-05 A, normalise_RMS = 1.14e+00, Area = 49
31.3 um^2, normalised area = 1.3
Rank 55 (Index: 85): RMS = 4.82e-05 A, normalise_RMS = 1.31e+00, Area = 43
17.1 um^2, normalised area = 1.1
Rank 56 (Index: 70): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 53
92.0 um^2, normalised area = 1.4
Rank 57 (Index: 25): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 53
92.0 um^2, normalised area = 1.4
Rank 58 (Index: 86): RMS = 4.81e-05 A, normalise_RMS = 1.31e+00, Area = 44
45.0 um^2, normalised area = 1.2
Rank 59 (Index: 30): RMS = 4.21e-05 A, normalise_RMS = 1.14e+00, Area = 50
84.8 um^2, normalised area = 1.3

Rank 60 (Index: 80): RMS = 4.21e-05 A, normalise_RMS = 1.14e+00, Area = 50
84.8 um^2, normalised area = 1.3
Rank 61 (Index: 47): RMS = 3.78e-05 A, normalise_RMS = 1.03e+00, Area = 55
96.7 um^2, normalised area = 1.5
Rank 62 (Index: 59): RMS = 3.69e-05 A, normalise_RMS = 1.00e+00, Area = 56
99.1 um^2, normalised area = 1.5
Rank 63 (Index: 36): RMS = 4.81e-05 A, normalise_RMS = 1.31e+00, Area = 45
73.0 um^2, normalised area = 1.2
Rank 64 (Index: 87): RMS = 4.81e-05 A, normalise_RMS = 1.31e+00, Area = 45
73.0 um^2, normalised area = 1.2
Rank 65 (Index: 71): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 55
96.7 um^2, normalised area = 1.5
Rank 66 (Index: 26): RMS = 3.83e-05 A, normalise_RMS = 1.04e+00, Area = 55
96.7 um^2, normalised area = 1.5
Rank 67 (Index: 88): RMS = 4.80e-05 A, normalise_RMS = 1.30e+00, Area = 47
00.9 um^2, normalised area = 1.2
Rank 68 (Index: 31): RMS = 4.20e-05 A, normalise_RMS = 1.14e+00, Area = 53
92.0 um^2, normalised area = 1.4
Rank 69 (Index: 89): RMS = 4.79e-05 A, normalise_RMS = 1.30e+00, Area = 48
28.9 um^2, normalised area = 1.3
Rank 70 (Index: 60): RMS = 3.69e-05 A, normalise_RMS = 1.00e+00, Area = 60
06.2 um^2, normalised area = 1.6
Rank 71 (Index: 48): RMS = 3.77e-05 A, normalise_RMS = 1.02e+00, Area = 61
08.6 um^2, normalised area = 1.6
Rank 72 (Index: 37): RMS = 4.78e-05 A, normalise_RMS = 1.30e+00, Area = 50
84.8 um^2, normalised area = 1.3
Rank 73 (Index: 32): RMS = 4.19e-05 A, normalise_RMS = 1.14e+00, Area = 56
99.1 um^2, normalised area = 1.5
Rank 74 (Index: 61): RMS = 3.68e-05 A, normalise_RMS = 1.00e+00, Area = 63
13.3 um^2, normalised area = 1.7
Rank 75 (Index: 33): RMS = 4.19e-05 A, normalise_RMS = 1.14e+00, Area = 60
06.2 um^2, normalised area = 1.6
Rank 76 (Index: 62): RMS = 3.68e-05 A, normalise_RMS = 1.00e+00, Area = 66
20.4 um^2, normalised area = 1.7
Rank 77 (Index: 49): RMS = 3.77e-05 A, normalise_RMS = 1.02e+00, Area = 66
20.4 um^2, normalised area = 1.7
Rank 78 (Index: 38): RMS = 4.77e-05 A, normalise_RMS = 1.30e+00, Area = 55
96.7 um^2, normalised area = 1.5
Rank 79 (Index: 34): RMS = 4.19e-05 A, normalise_RMS = 1.14e+00, Area = 63
13.3 um^2, normalised area = 1.7
Rank 80 (Index: 35): RMS = 4.18e-05 A, normalise_RMS = 1.14e+00, Area = 66
20.4 um^2, normalised area = 1.7
Rank 81 (Index: 50): RMS = 3.76e-05 A, normalise_RMS = 1.02e+00, Area = 71
32.3 um^2, normalised area = 1.9
Rank 82 (Index: 39): RMS = 4.76e-05 A, normalise_RMS = 1.29e+00, Area = 61
08.6 um^2, normalised area = 1.6
Rank 83 (Index: 51): RMS = 3.76e-05 A, normalise_RMS = 1.02e+00, Area = 76
44.1 um^2, normalised area = 2.0
Rank 84 (Index: 40): RMS = 4.76e-05 A, normalise_RMS = 1.29e+00, Area = 66
20.4 um^2, normalised area = 1.7
Rank 85 (Index: 52): RMS = 3.76e-05 A, normalise_RMS = 1.02e+00, Area = 81
56.0 um^2, normalised area = 2.1
Rank 86 (Index: 41): RMS = 4.75e-05 A, normalise_RMS = 1.29e+00, Area = 71
32.3 um^2, normalised area = 1.9
Rank 87 (Index: 53): RMS = 3.75e-05 A, normalise_RMS = 1.02e+00, Area = 86
67.9 um^2, normalised area = 2.3
Rank 88 (Index: 42): RMS = 4.75e-05 A, normalise_RMS = 1.29e+00, Area = 76
44.1 um^2, normalised area = 2.0
Rank 89 (Index: 43): RMS = 4.75e-05 A, normalise_RMS = 1.29e+00, Area = 81
56.0 um^2, normalised area = 2.1

```
Rank 90 (Index: 44): RMS = 4.75e-05 A, normalise_RMS = 1.29e+00, Area = 86
67.9 um^2, normalised area = 2.3
```

In [104…

```python
# to check exactly the case of the index
def analyze_index(index, M, N):
    """
    Analyzes an index and prints corresponding information based on its v

    Args:
      index: The input index.
      M: A known array of M values.
      N: A known array of N values.
    """

    index += 1  # Add 1 to the index


    if index < 46:
        print("Case 1")
        M_value = M[index // 9]
        b = index - ((index // 9) * 9)  # Calculate b for Case 1
        print(f'M = {M_value}')
        print(f'nnpn = {nnpn[int(b) - 1]}')  # Adjust index for nnpn (0-b
        print(' |                   | ')
        print(' |                   | ')
        print(' |                   | ')
        print('  \|                 |/ ')
        print('   |-------|-------|  ')
        print('  /|       |       |\ ')
        print(' |_________|         | ')
        print(' |                   | ')
        print(' |                   | ')
        print(f' |   {nnpn[int(b) - 1]}   :    {int(M_value*nnpn[int(b) -
    else:
        print("Case 2")
        M_value = M[(index - 45) // 9]
        b = index - 45 - ((index - 45) // 9) * 9  # Calculate b for Case
        print(f'M = {M_value}')
        print(f'nnpn = {nnpn[int(b) - 1]}')  # Adjust index for nnpn (0-b
        print(' |                   | ')
        print(' |                   | ')
        print(' |                   | ')
        print('  \|                 |/ ')
        print('   |-------|-------|  ')
        print('  /|       |       |\ ')
        print(' |_________|         | ')
        print(' |                   | ')
        print(' |                   | ')
        print(f' |   {int(nnpn[int(b) - 1]/M_value)}   :    {nnpn[int(b)


analyze_index(5,M,nnpn)
```

```
Case 1
M = 0.25
nnpn = 28
 |                   |
 |                   |
 |                   |
  \|                 |/
   |-------|-------|
  /|       |       |\
 |_________|         |
 |                   |
 |                   |
 |   28   :    7     |
```

In [ ]:
```
P
```