

Projet - Problèmes Inverses

September 6, 2026

Inverse Problems for images

Antoine Lunven & Colin Coërchon

Project : Reflector imaging from ambient noise signals

0.1 Bibliothèques

```
[1]: # === Librairies standards ===  
import random  
import time  
  
# === Calcul scientifique ===  
import numpy as np  
from scipy import integrate  
from functools import partial  
  
# === Compilation et accélération ===  
from numba import jit, prange  
from numba import float64, complex128, int32  
  
# === Visualisation ===  
import matplotlib.pyplot as plt  
from mpl_toolkits.mplot3d import Axes3D  
  
# === Barre de progression ===  
from tqdm import tqdm
```

0.2 Constantes

```
[2]: N = 200  
c0 = 1  
sigma_r = 0.001  
zr = np.array([-5,0,65])
```

1 1/ Setup Géométrique

```
[3]: def receivers_coordinates():  
    """Returns the receivers position"""  
    x = np.zeros((5,3))  
    for j in range(1,6): # There are 5 receivers  
        x[j-1,:] = np.array([-30 + 7.5*(j-1),0,100])  
    return x  
  
def sources_coordinates(N):  
    """Returns sources position"""  
    sources = np.zeros((N,3))  
    for i in range(N):  
        x = np.random.uniform(-50,50)  
        y = np.random.uniform(-10,10)  
        z = np.random.uniform(185,200)  
        sources[i] = np.array([x,y,z])  
    return sources  
  
def random_field_sources(t,x):  
    pass  
  
def PSD_sources(omega):  
    """Returns the power spectral density of the noise sources"""  
    return np.exp(-omega**2)  
  
def geometry_plot(receivers, sources, zr):  
    # x = np.arange(-50,50)  
    # z = np.arange(185,200)  
    # X,Z = np.meshgrid(x,z)  
    plt.scatter(sources[:,2], sources[:,0], label = "Sources", marker="v")  
    plt.scatter(receivers[:,2], receivers[:,0], label = "Receivers", marker="o")  
    plt.scatter(zr[2], zr[0], label = "Reflector", marker = "x")  
    plt.xlabel("z")  
    plt.ylabel("x")  
    plt.xlim(0,210)  
    plt.ylim(-55,55)  
    plt.legend()  
    plt.show()  
  
def geometry_plot_3D(receivers, sources, zr):  
    fig = plt.figure(figsize=(10, 7))  
    ax = fig.add_subplot(111, projection='3d')  
  
    # Plot des sources (triangles)  
    ax.scatter(sources[:, 2], sources[:, 1], sources[:, 0], label="Sources",  
↪marker="o", c='r')
```

```

# Plot des récepteurs (cercles)
ax.scatter(receivers[:, 2], receivers[:, 1], receivers[:, 0],
→label="Receivers", marker="^", c='black')

# Plot du réflecteur (croix)
ax.scatter(zr[2], zr[1], zr[0], label="Reflector", marker="d", c='b', s=50)

ax.set_xlabel("Z")
ax.set_ylabel("Y")
ax.set_zlabel("X")

ax.set_xlim([0, 210])
ax.set_ylim([-15, 15])
ax.set_zlim([-55, 55])

ax.set_title("3D Geometry of Receivers, Sources, and Reflector")
ax.legend()

plt.show()

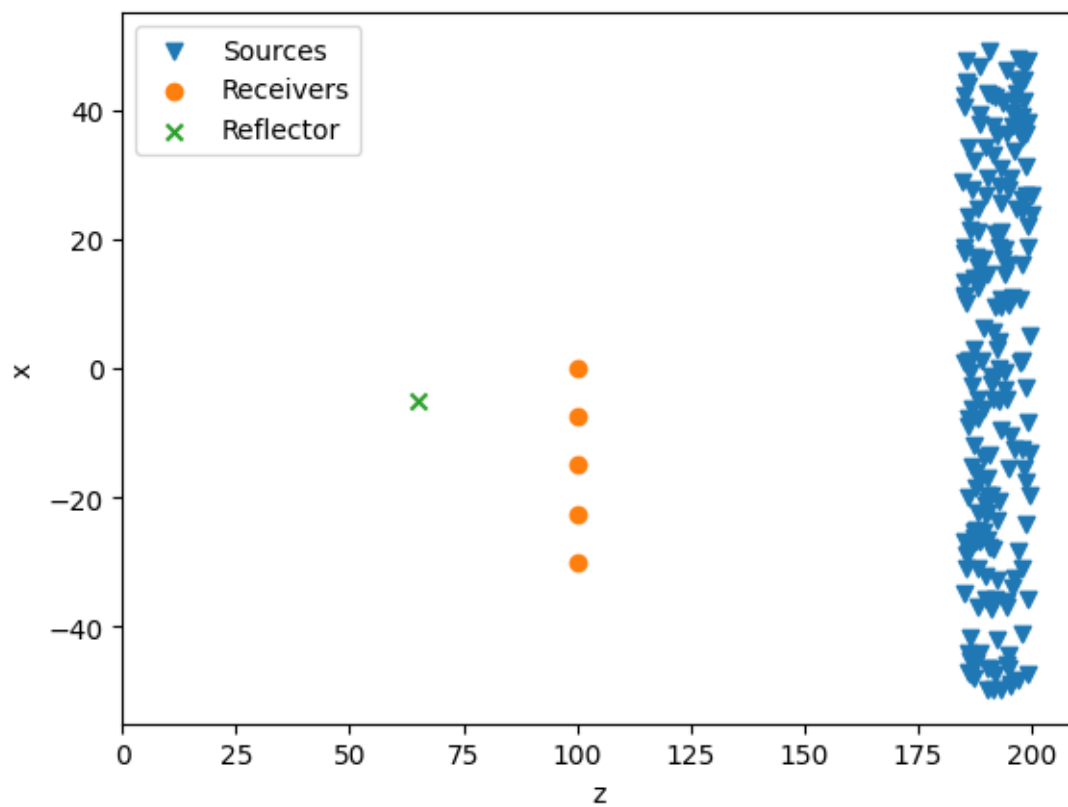
```

```

[4]: sources_position = sources_coordinates(N)
new_sources_position = sources_position[:, [0,2]]
receivers_position = receivers_coordinates()

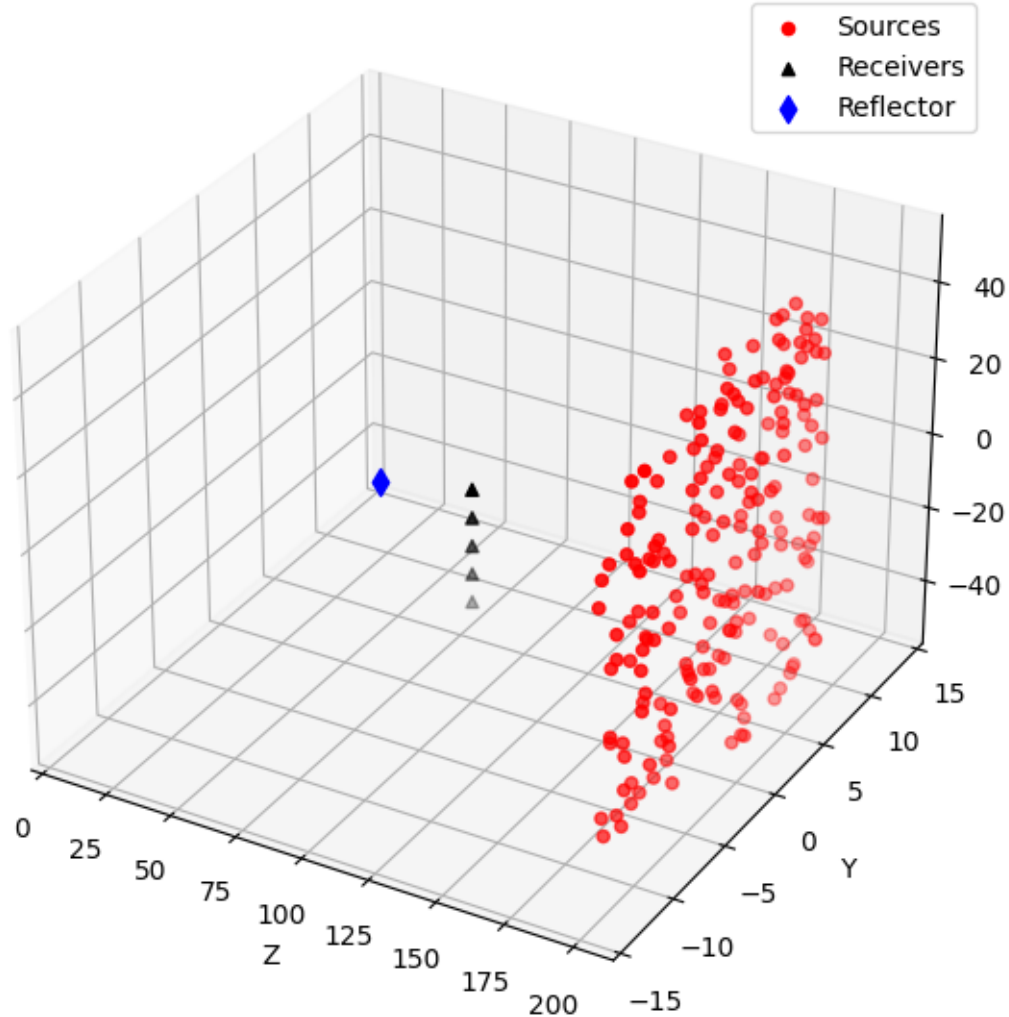
geometry_plot(receivers_position, sources_position, zr)

```



```
[5]: geometry_plot_3D(receivers_position, sources_position, zr)
```

3D Geometry of Receivers, Sources, and Reflector



2 2/ Préliminaires

2.1 Notre méthode numérique d'intégration : la méthode 3/8 de Simpson

```
[6]: @jit(nopython=True)
def simpson_rule(y, x, axis):
    """
    Perform numerical integration using Simpson's rule on the function values y_
    ↪ at points x.
    Assumes that x is evenly spaced.
    """
    # print("y", y.shape, "x", x.shape)
```

```

n = len(y)
h = (x[-1] - x[0]) / (n - 1)

integral = y[0] + y[-1]
integral += 4 * np.sum(y[1:n-1:2], axis = axis) # Sum over odd indices (1,
↪3, 5, ...)
integral += 2 * np.sum(y[2:n-2:2], axis = axis) # Sum over even indices (2,
↪4, 6, ...)

integral *= h / 3
return integral

```

2.2 Les autres fonctions

```

[7]: @jit(nopython=True)
def Green_function(omega, x, y):
    """Computes the Green function"""
    x = x[np.newaxis, :] # Reshape x to (1, 2)
    r = np.sqrt(np.sum((x - y)**2, axis=1))[:, np.newaxis]
    # r = np.linalg.norm(x - y, axis=1)[:, np.newaxis]
    G0_hat = np.exp(1j * (omega[np.newaxis, :] / c0) * r) / (4 * np.pi * r)
    return G0_hat

@jit(nopython=True)
def Born_approximation(omega, x, y, z_r):
    """Computes the Born approximation """
    return Green_function(omega, x, y) +
↪sigma_r*(omega**2)*Green_function(omega, x, z_r)*Green_function(omega, z_r, y)

@jit(nopython=True)
def PSD(omega):
    """Computes the power density function"""
    return np.exp(-omega**2)

@jit(nopython = True)
def CN_tau_x_xprime(tau, x, xprime, sources_position, omega, z_r):
    """Coomputes the signal correlation CN(tau, x, x_prime)"""
    Born_x = Born_approximation(omega, x, sources_position, z_r)[:,:,np.newaxis]
    Born_xprime = Born_approximation(omega, xprime, sources_position, z_r)[:,:
↪,np.newaxis]
    exp_term = np.exp(-1j * np.outer(omega, tau))

    integrand = PSD(omega)[:, np.newaxis] * np.sum(Born_x.
↪conjugate()*Born_xprime, axis = 0)*exp_term
    CN_tau = simpson_rule(integrand, omega, axis = 0)

    return (1 / (2 * np.pi * len(sources_position))) * CN_tau

```

```
[8]: omega = np.linspace(-10, 10, 1500)
tau = np.linspace(-200, 200, 2000).reshape(-1, 1)
```

3 Partie 1

3.1 Question 1.a

3.1.1 The three-dimensional Green's function $\hat{G}_0(\omega, x, y)$ is the solution of

$$\Delta_x \hat{G}_0(\omega, x, y) + \frac{\omega^2}{c_0^2} \hat{G}_0(\omega, x, y) = -\delta(x - y), \quad x \in \mathbb{R}^2.$$

3.1.2 Sommerfeld radiation condition

$$\hat{G}_0(\omega, x, y) = \frac{1}{4\pi |x - y|} \exp\left(i \frac{\omega}{c_0} |x - y|\right)$$

3.1.3 Born approximation : it is used to generate data in the presence of a reflector z_r

$$\hat{G}(\omega, x, y) = \hat{G}_0(\omega, x, y) + \sigma_r \hat{G}_0(\omega, x, z_r) \hat{G}_0(\omega, z_r, y)$$

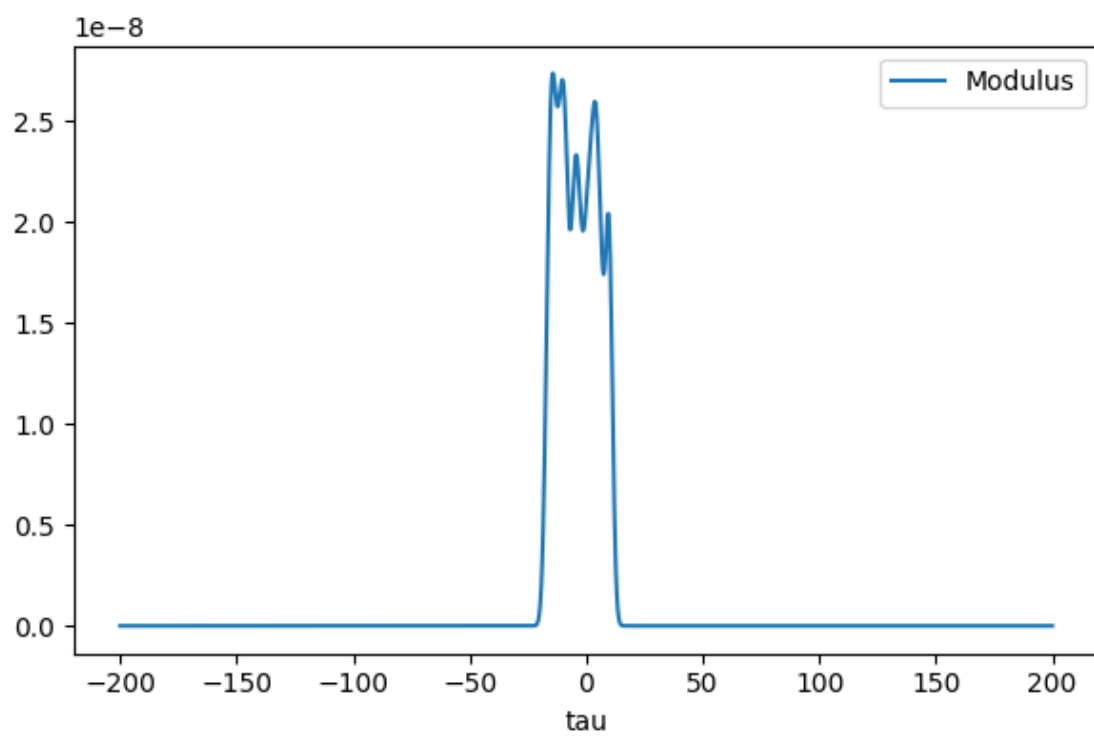
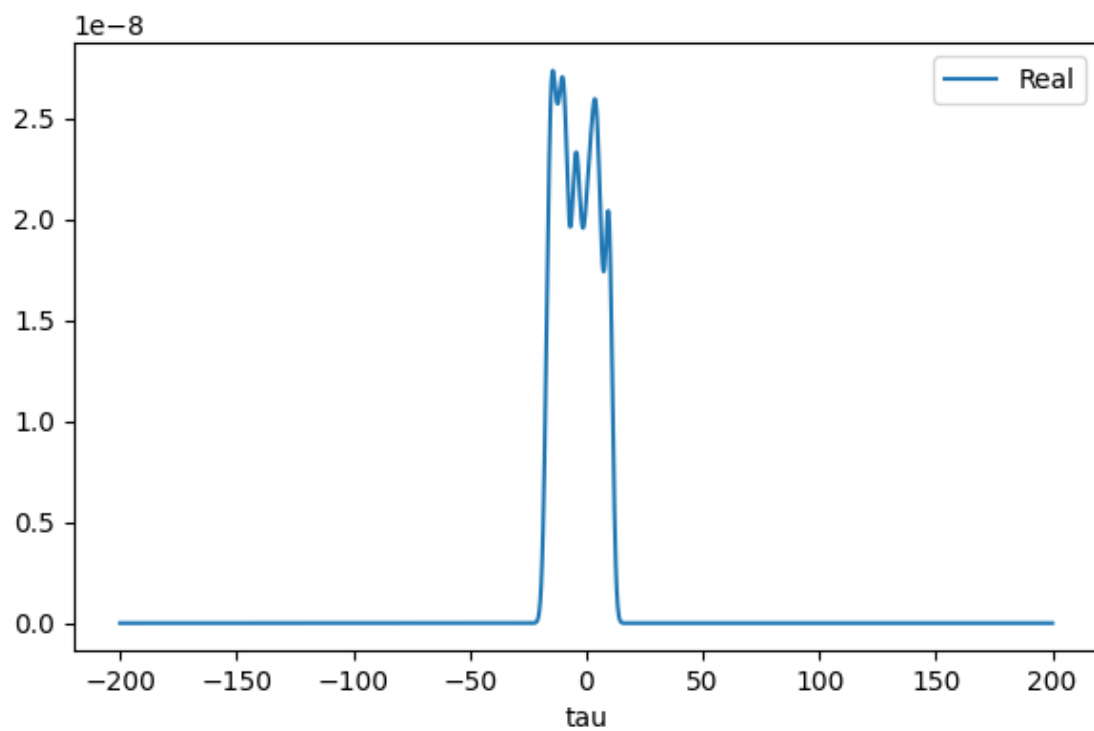
3.1.4 Expectation of the empirical cross correlated signals, recorded at x_1 and x_2 with respect to the distribution of the emitted signals

$$C_N(\tau, x_1, x_5) = \frac{1}{2\pi N} \sum_{s=1}^N \int \hat{F}(\omega) \hat{G}(\omega, x_1, y_s) \hat{G}(\omega, x_5, y_s) e^{-i\omega\tau} d\omega$$

```
[9]: CN_tau = CN_tau_x_xprime(tau, receivers_position[0][[0,2]],
    ↪ receivers_position[4][[0,2]], new_sources_position, omega, zr[[0,2]])
print(CN_tau.shape)
```

(2000,)

```
[10]: fig, ax = plt.subplots(2,1, figsize = (6,8))
# ax[0].plot(tau, CN_tau.imag, label = "Imaginary")
# ax[0].set_xlabel("tau")
# ax[0].legend()
ax[0].plot(tau, CN_tau.real, label = "Real") # x axis
ax[0].set_xlabel("tau")
ax[0].legend()
ax[1].plot(tau, np.abs(CN_tau), label = "Modulus") # x axis
ax[1].set_xlabel("tau")
ax[1].legend()
plt.tight_layout()
```



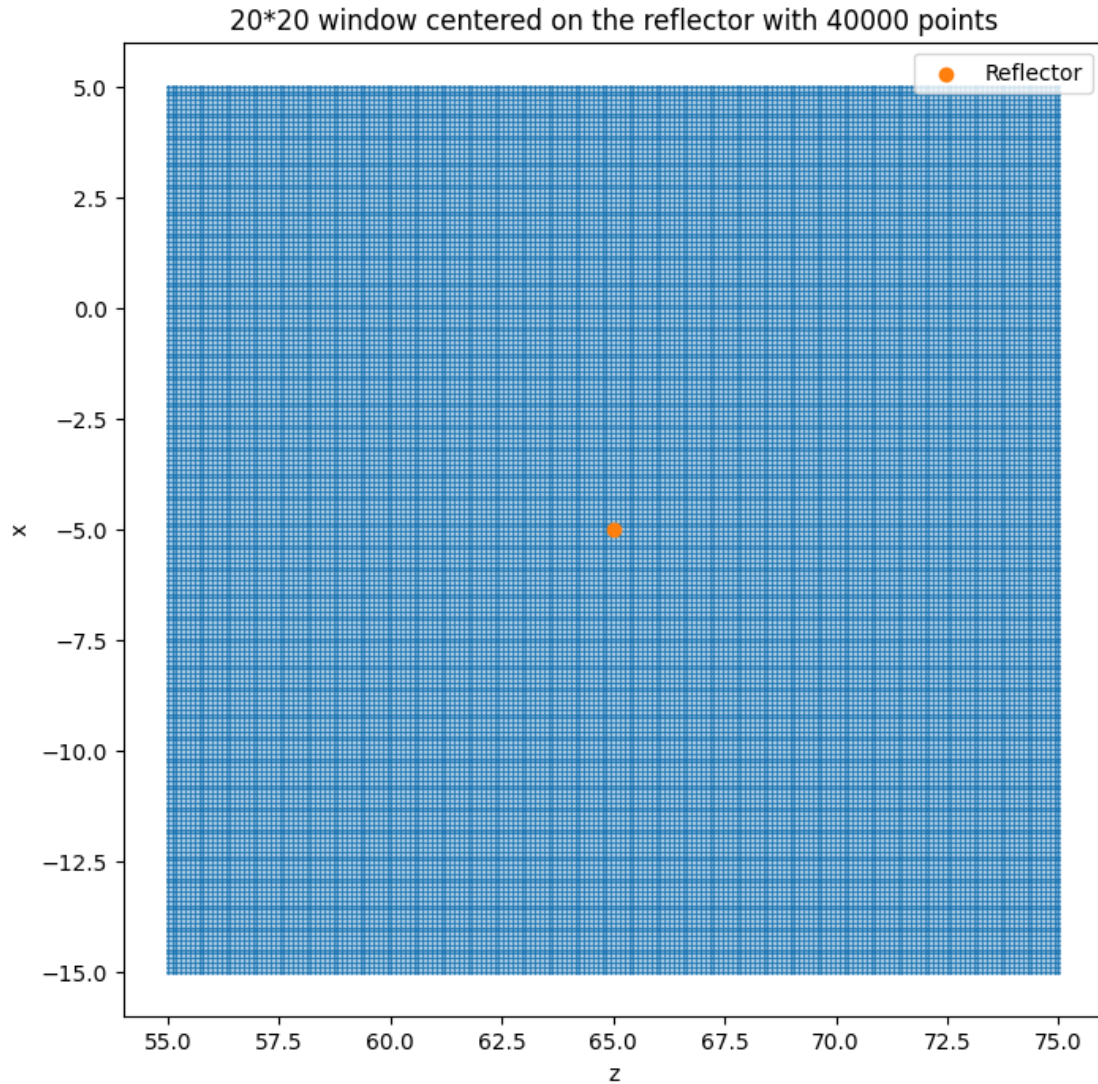
3.2 Question 1.b

```
[11]: window_size = 20
      number_elements = 200

      def ys_plane(z_r, window_size = 20, number_elements = 20):
          """Returns a ys plane in a square window with size 20*20 centered on the_
          ↪reflector location"""
          ys = np.zeros((number_elements, number_elements, 2))
          print("x and z reflector location", z_r[[0, 2]])
          x_val = np.linspace(z_r[0]-window_size/2, z_r[0]+window_size/2, number_elements)
          z_val = np.linspace(z_r[2]-window_size/2, z_r[2]+window_size/2, number_elements)
          for i, x in enumerate(x_val):
              for j, z in enumerate(z_val):

                  ys[i,j] = np.array([x, z])
          return ys
      plt.figure(figsize=(8,8))
      ys = ys_plane(zr, window_size, number_elements = number_elements)
      plt.scatter(ys[:, :, 1], ys[:, :, 0], s=1)
      plt.scatter(zr[2], zr[0], label = f"Reflector")
      plt.title(f"{window_size}*{window_size} window centered on the reflector with_
      ↪{number_elements*number_elements} points")
      plt.xlabel("z")
      plt.ylabel("x")
      plt.legend()
      plt.show()
```

x and z reflector location [-5 65]



```
[12]: def KM_image (ys, receiver_position, sources_position, omega, zr):
        image = np.zeros((ys.shape[0]*ys.shape[1]), dtype=np.complex128)
        ys = ys.reshape(ys.shape[0]*ys.shape[1], 2) # Flatten the grid to compute
        ↪ for each point
        for j, xj in tqdm(enumerate(receiver_position),
        ↪ total=len(receiver_position)):
            for l, xl in enumerate(receiver_position):

                tau = (np.linalg.norm(xj[np.newaxis, [0,2]]-ys, axis = 1) + np.
        ↪ linalg.norm(ys-xl[np.newaxis, [0,2]], axis = 1))[:, np.newaxis] # Select x and
        ↪ z coordinates
```

```

        CN_tau = CN_tau_x_xprime(tau, xj[[0,2]], xl[[0,2]],  

↪sources_position, omega, zr[[0,2]])  

        image += CN_tau  

    return image

```

```

[23]: def plot_image(matrix, title=None, imaginary=None):
    image_plot = matrix.reshape(ys.shape[0], ys.shape[1])

    Z = ys[:, :, 0] # Z-coordinates
    X = ys[:, :, 1] # X-coordinates

    if imaginary is not None:
        fig, ax = plt.subplots(1, 3, figsize=(15, 8))

        im1 = ax[0].imshow(image_plot.real, extent=[X.min(), X.max(), Z.min(), Z.  

↪max()],
                            origin='lower', cmap='viridis')
        ax[0].set_title("Real Part of Image")
        ax[0].set_xlabel("Z Coordinate")
        ax[0].set_ylabel("X Coordinate")
        fig.colorbar(im1, ax=ax[0], label="Real Value", fraction=0.045)

        im2 = ax[1].imshow(image_plot.imag, extent=[X.min(), X.max(), Z.min(), Z.  

↪max()],
                            origin='lower', cmap='viridis')
        ax[1].set_title("Imaginary Part of Image")
        ax[1].set_xlabel("Z Coordinate")
        ax[1].set_ylabel("X Coordinate")
        fig.colorbar(im2, ax=ax[1], label="Imaginary Value", fraction=0.045)

        im3 = ax[2].imshow(np.abs(image_plot), extent=[X.min(), X.max(), Z.  

↪min(), Z.max()],
                            origin='lower', cmap='viridis')
        ax[2].set_title("Image modulus")
        ax[2].set_xlabel("Z Coordinate")
        ax[2].set_ylabel("X Coordinate")
        fig.colorbar(im3, ax=ax[2], label="Imaginary Value", fraction=0.045)

    else:
        fig, ax = plt.subplots(1, figsize=(8, 8))

        im1 = ax.imshow(image_plot.real, extent=[X.min(), X.max(), Z.min(), Z.  

↪max()],
                            origin='lower', cmap='viridis')
        ax.set_title("Real Part of Image")
        ax.set_xlabel("Z Coordinate")
        ax.set_ylabel("X Coordinate")

```

```

fig.colorbar(im1, ax=ax, label="Real Value", fraction=0.045)

if title is not None:
    fig.suptitle(title, fontsize=18, fontweight="bold", y=0.75)
    plt.tight_layout()
    plt.subplots_adjust(top=0.92)
    plt.show()

plt.tight_layout()
plt.show()

```

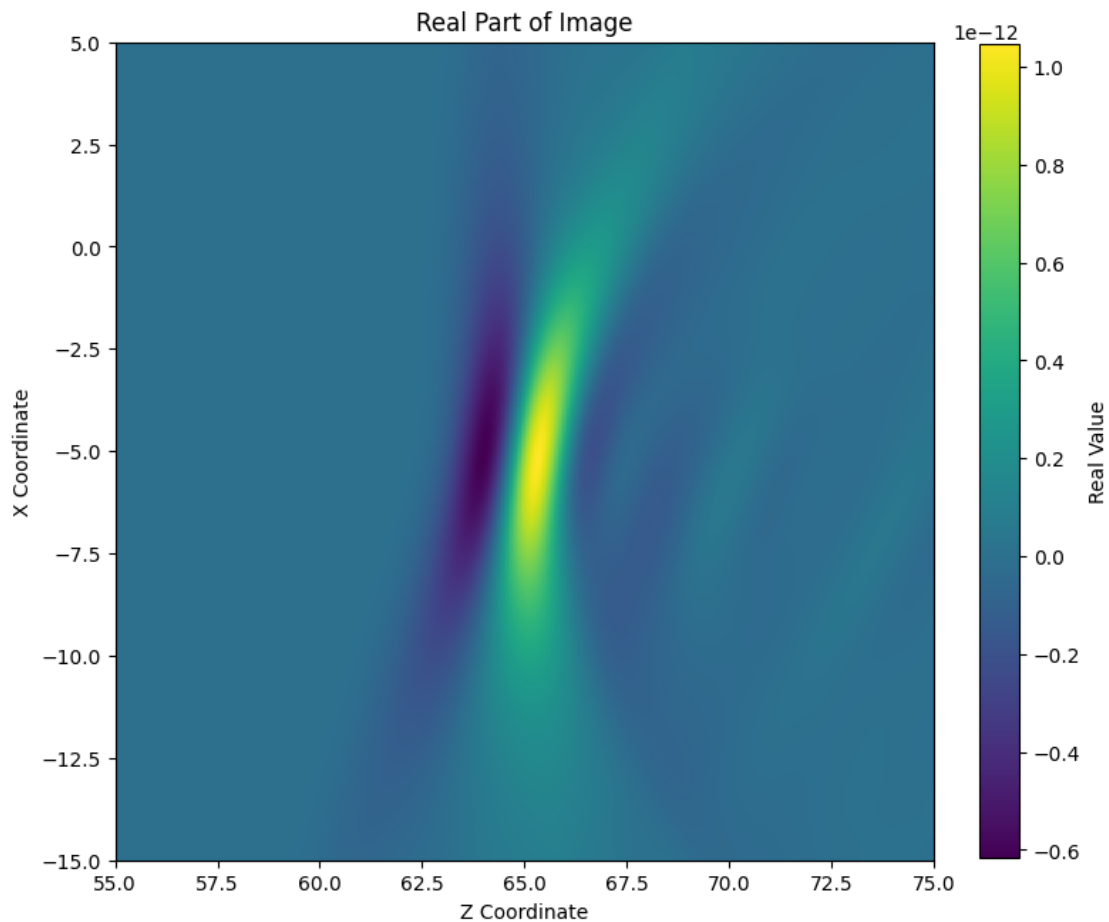
3.2.1 Image zoomée sur une zone de 20*20 autour du réflecteur

```

[42]: image = KM_image(ys, receivers_position, new_sources_position, omega, zr)
      plot_image(image)

```

100%|| 5/5 [01:38<00:00, 19.80s/it]

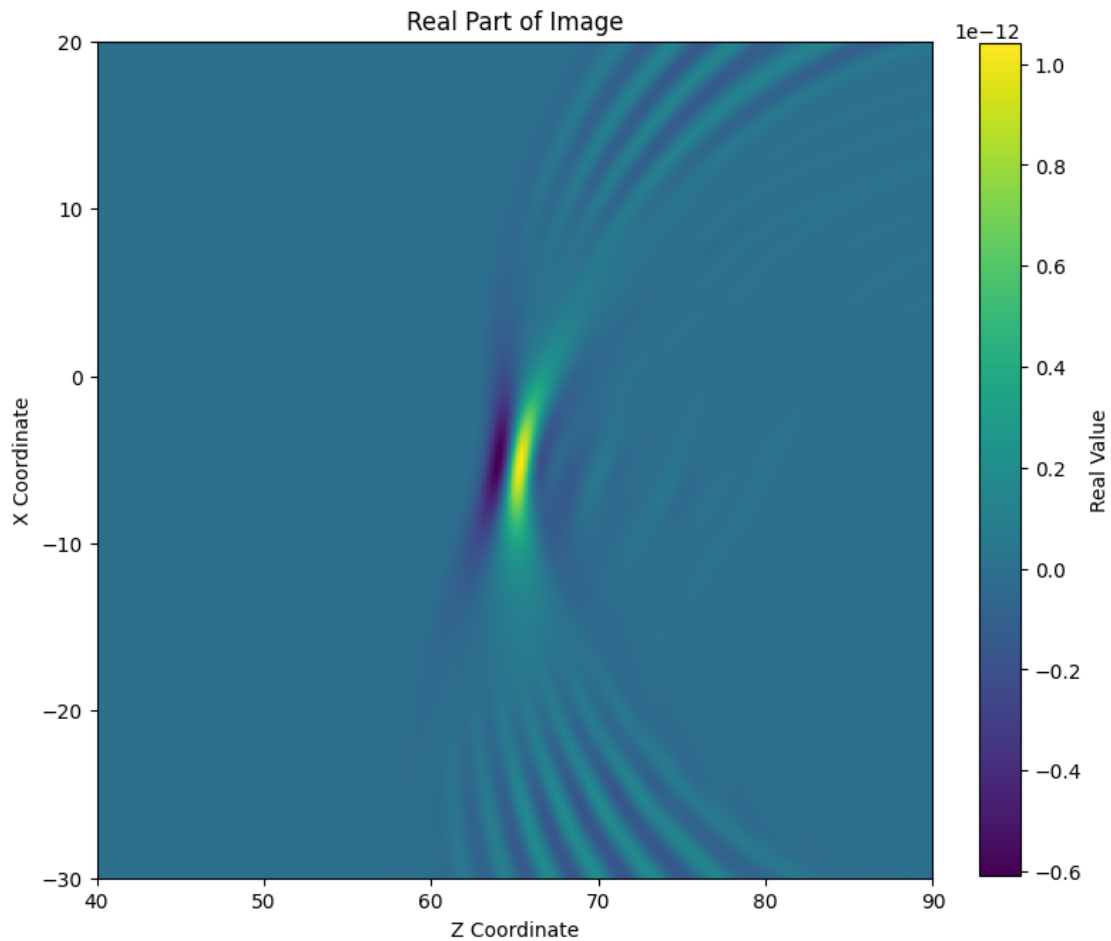


3.2.2 Image dézoomée sur une zone de 50*50 autour du réflecteur

```
[43]: ys = ys_plane(zr, window_size = 50, number_elements = number_elements)
      image = KM_image(ys, receivers_position, new_sources_position, omega, zr)
      plot_image(image)
```

x and z reflector location [-5 65]

100%|| 5/5 [00:57<00:00, 11.45s/it]



3.3 Question 1.c

```
[ ]: # def variation_parametres_image(omega_list,
      #                               N_list,
      #                               receivers_position,
      #                               zr,
      #                               window_size=50,
      #                               number_elements=100,
```

```

#                                     omega_min=-10,
#                                     omega_max=10):
#     ys_local = ys_plane(zr, window_size=window_size,
# ↪number_elements=number_elements)
#     global ys
#     ys = ys_local

#     fig, axs = plt.subplots(len(omega_list), len(N_list), figsize=(40,10))

#     idx = 0 # ligne dans la figure

#     for i, nb_omega in enumerate(omega_list):
#         omega = np.linspace(omega_min, omega_max, nb_omega)

#         for k, N in enumerate(N_list):
#             print(f"Calcul pour N = {N}, #ω = {nb_omega}")

#             sources_position = sources_coordinates(N)
#             new_sources_position = sources_position[:, [0, 2]]

#             image_complex = KM_image(ys, receivers_position,
# ↪new_sources_position, omega, zr)
#             image_plot = image_complex.reshape(number_elements,
# ↪number_elements)

#             Z = ys[:, :, 0]
#             X = ys[:, :, 1]

#             titles = ["Partie réelle"]
#             datas = [image_plot.real]
#             cbar_labels = ["Real value"]

#             im = axs[i,k].imshow(image_plot.real,
#                                     extent=[X.min(), X.max(), Z.min(), Z.max()],
#                                     origin='lower',
#                                     cmap='viridis')

#             # Ajout de la barre de couleur à droite de chaque image

#             axs.set_title(f"N={N}, #ω={nb_omega}", fontsize=9)
#             fig.colorbar(im, ax=axs, fraction=0.046, pad=0.04)

#             idx += 1

#     plt.tight_layout()

```

```
# plt.show()
```

```
[14]: def variation_parametres_image(omega_list,
                                     N_list,
                                     receivers_position,
                                     zr,
                                     window_size=50,
                                     number_elements=100,
                                     omega_min=-10,
                                     omega_max=10):
    ys_local = ys_plane(zr, window_size=window_size,
↪number_elements=number_elements)
    global ys
    ys = ys_local

    total_rows = len(omega_list) * len(N_list)
    fig, axs = plt.subplots(total_rows, 3, figsize=(12, total_rows * 2.5))

    idx = 0 # ligne dans la figure

    for nb_omega in omega_list:
        omega = np.linspace(omega_min, omega_max, nb_omega)

        for N in N_list:
            print(f"Calcul pour N = {N}, # $\omega$  = {nb_omega}")

            sources_position = sources_coordinates(N)
            new_sources_position = sources_position[:, [0, 2]]

            image_complex = KM_image(ys, receivers_position,
↪new_sources_position, omega, zr)
            image_plot = image_complex.reshape(number_elements, number_elements)

            Z = ys[:, :, 0]
            X = ys[:, :, 1]

            titles = ["Partie réelle", "Partie imaginaire", "Module"]
            datas = [image_plot.real, image_plot.imag, np.abs(image_plot)]
            cbar_labels = ["Valeur réelle", "Valeur imaginaire", "Module"]

            for j in range(3):
                ax = axs[idx, j]
                im = ax.imshow(datas[j],
                               extent=[X.min(), X.max(), Z.min(), Z.max()],
                               origin='lower',
                               cmap='viridis')
```

```

        ax.set_title(titles[j], fontsize=9)
        ax.set_xticks([])
        ax.set_yticks([])

        # Ajout de la barre de couleur à droite de chaque image
        fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04,
↪label=cbar_labels[j])

        if j == 0:
            ax.set_ylabel(f"N={N}, # $\omega$ = {nb_omega}", fontsize=9)

        idx += 1

    plt.tight_layout()
    plt.show()

```

```

[ ]: omega_list = [500, 1000, 1500, 2000]
     N_list = [1, 200, 1000]

     variation_parametres_image(omega_list,
                                N_list,
                                receivers_position=receivers_position,
                                zr=zr,
                                window_size=30,
                                number_elements=100,
                                omega_min=-10,
                                omega_max=10)

```

x and z reflector location [-5 65]

Calcul pour N = 1, # ω = 500

100%|| 5/5 [00:10<00:00, 2.18s/it]

Calcul pour N = 200, # ω = 500

100%|| 5/5 [00:05<00:00, 1.13s/it]

Calcul pour N = 1000, # ω = 500

100%|| 5/5 [00:07<00:00, 1.41s/it]

Calcul pour N = 1, # ω = 1000

100%|| 5/5 [00:09<00:00, 1.86s/it]

Calcul pour N = 200, # ω = 1000

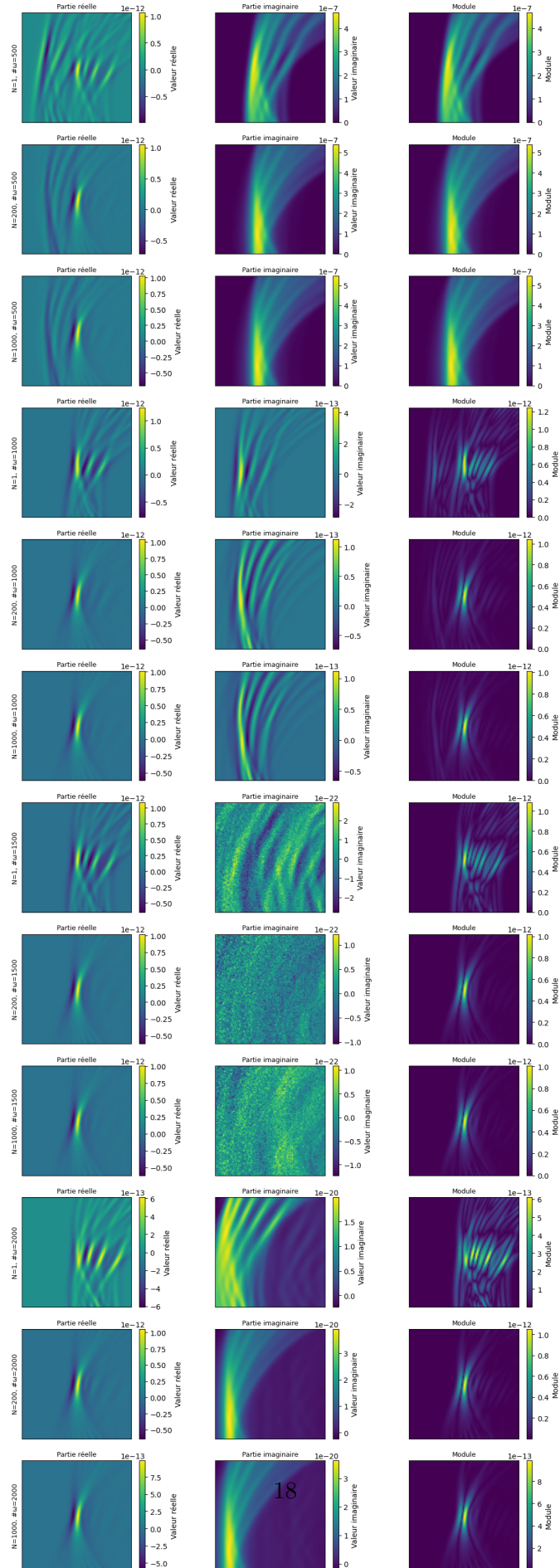
100%|| 5/5 [00:10<00:00, 2.17s/it]

Calcul pour N = 1000, # ω = 1000

100%|| 5/5 [00:14<00:00, 2.84s/it]

Calcul pour N = 1, # ω = 1500

100%|| 5/5 [00:15<00:00, 3.04s/it]
Calcul pour $N = 200$, $\#\omega = 1500$
100%|| 5/5 [00:15<00:00, 3.19s/it]
Calcul pour $N = 1000$, $\#\omega = 1500$
100%|| 5/5 [00:21<00:00, 4.20s/it]
Calcul pour $N = 1$, $\#\omega = 2000$
100%|| 5/5 [00:19<00:00, 3.99s/it]
Calcul pour $N = 200$, $\#\omega = 2000$
100%|| 5/5 [00:18<00:00, 3.77s/it]
Calcul pour $N = 1000$, $\#\omega = 2000$
100%|| 5/5 [00:26<00:00, 5.30s/it]



3.3.1 Choix de l'échantillonnage en fréquence ω

On choisit **1500 points pour échantillonner ω** .

Pour $\omega < 1500$, la partie imaginaire est faussée avec des valeurs trop élevées. Au-delà, les résultats sont stables.

3.3.2 Influence du nombre de sources N

Entre $N = 200$ et $N = 1000$, l'image change peu.

Mais pour $N = 1$, la position du réflecteur $\mathbf{z}_r$ devient difficile à localiser à cause du bruit.

4 Partie 2

```
[15]: tau = np.linspace(-150, 150, 1500)
```

4.1 Question 2.a

4.1.1 Recorded signals $(u(t, x_j))_{j=1,\dots,5, t \in [0, T]}$

$$\begin{aligned} u(t, x_j) &= \frac{1}{2\pi} \iint \hat{G}(\omega, x_j; y) \hat{n}(\omega, y) \exp(-i\omega t) dy d\omega \\ &= \frac{1}{2\pi\sqrt{N}} \sum_{s=1}^N \int \hat{G}(\omega, x_j; y_s) \hat{n}_s(\omega) \exp(-i\omega t) d\omega. \end{aligned}$$

4.1.2 Empirical cross correlation of the signals recorded at x_1 and x_2 for the recording time T

$$C_{T,N}(\tau, x_1, x_2) = \frac{1}{T - |\tau|} \int_{\tau}^{T-\tau} u(t, x_1) u(t + \tau, x_2) dt.$$

where $\tau_+ = \max(\tau, 0)$ and $\tau_- = -\min(\tau, 0)$, $\tau \in [-150, 150]$

```
[16]: @jit(nopython=True)
def integrand_omega(omega, t, x, sources_position, z_r, n_hat):
    Born = Born_approximation(omega, x, sources_position, z_r)[:,:,np.
    ↪newaxis]
    # print("nhat", n_hat.shape, 'born', Born.shape, np.exp(-1j * omega[np.
    ↪newaxis, :, np.newaxis] * t[np.newaxis, np.newaxis, :]).shape)
    integrand_val = np.sum(Born * n_hat, axis=0) * np.exp(-1j * omega[:, np.
    ↪newaxis] * t[np.newaxis, :])

    integrand_val = simpson_rule(integrand_val, omega, axis =0)
    return integrand_val

@jit(nopython=True)
def DFT(omega, signal):
```

```

""" print discrete fourier transform """
dft = np.zeros((signal.shape[0], signal.shape[1]), dtype = np.complex128)
# dft = np.array([np.exp(-2j*np.pi*i*omega/signal.shape[0])*signal[i] for i in
→ range(signal.shape[0])])
    for i in range(signal.shape[1]):
        dft += np.exp(-2j*np.pi*i*omega[np.newaxis, :]/signal.
→ shape[1])*signal[:,i][:, np.newaxis]
    return dft
# @jit(nopython=True)
@jit(nopython=True, parallel = True)
def cross_correlation(T, tau, n_hat, x1, x2, sources_position, omega, z_r):
    total = np.zeros(tau.shape[0], dtype=np.complex128)

    sources_position = np.asarray(sources_position, dtype=np.float64)
    # TODO : peut-être à changer
    # n_hat = np.random.randn(omega.shape[0])[np.newaxis, :, np.newaxis] * np.
→ sqrt(PSD(omega))[np.newaxis, :, np.newaxis]
    for i in prange(tau.shape[0]):
        tau_val = tau[i]
        low_t = -min(tau_val, 0.)
        high_t = T - max(tau_val, 0.)
        t = np.linspace(low_t, high_t, 1500)
        u1 = integrand_omega(omega, t, x1, sources_position, z_r, n_hat) / (2*np.
→ pi*np.sqrt(len(sources_position)))
        u2 = integrand_omega(omega, t + tau_val, x2, sources_position, z_r,
→ n_hat) / (2*np.pi*np.sqrt(len(sources_position)))
        integrand = u1*u2
        result = simpson_rule(integrand, t, axis =0)
        total[i] = result / (T - abs(tau_val))
    return total

```

[23]: T = [500, 1000, 2000, 3000]

```

def plot_cross_correlation():
    fig, ax = plt.subplots(1, 3, figsize=(14, 6))
    for t in tqdm(T):
        dft_random = DFT(omega, np.random.randn(200, omega.shape[0]))[:, :, np.
→ newaxis]
        print("fft", dft_random.shape)
        # dft_random = DFT(omega, np.random.randn(omega.shape[0]))[np.newaxis, :
→ , np.newaxis]
        n_hat = dft_random * np.sqrt(PSD(omega))[np.newaxis, :, np.newaxis]
        C_T_N = cross_correlation(t, tau, n_hat, receivers_position[4][[0,2]],
→ receivers_position[0][[0,2]], sources_position[:, [0,2]], omega, zr[[0,2]])
        ax[0].plot(tau, C_T_N.real, label = t)
        ax[0].set_title("Real Part of the signal", fontsize = 14)

```

```

ax[0].set_xlabel("Tau", fontsize = 14)
ax[0].legend()

ax[1].plot(tau, C_T_N.imag, label = t)
ax[1].set_title("Imaginary Part of the signal", fontsize = 14)
ax[1].set_xlabel("Tau", fontsize = 14)
ax[1].legend()

ax[2].plot(tau, np.abs(C_T_N), label = t)
ax[2].set_title("Modulus of the signal", fontsize = 14)
ax[2].set_xlabel("Tau", fontsize = 14)
ax[2].legend()

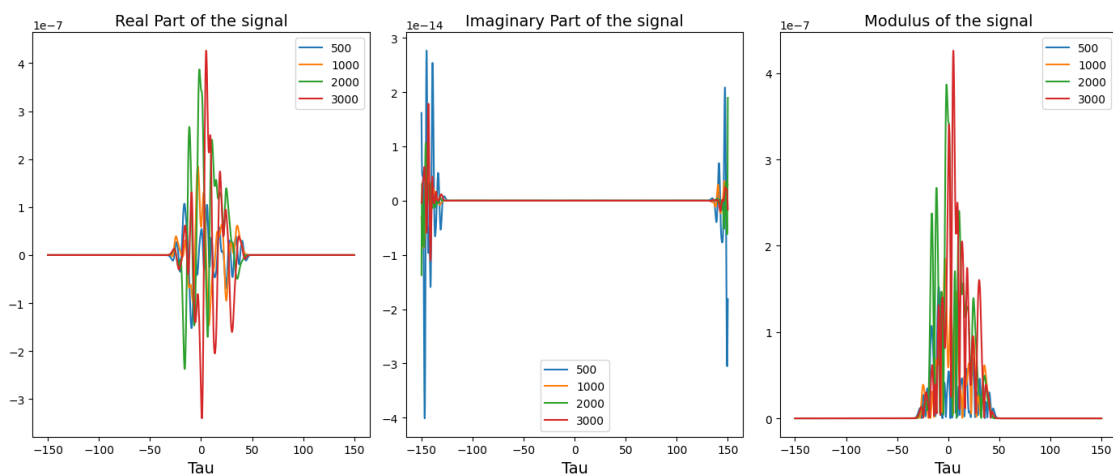
plt.tight_layout()
plt.show()
plot_cross_correlation()

```

```

0%|          | 0/4 [00:00<?, ?it/s]
fft (200, 1500, 1)
25%|         | 1/4 [01:46<05:19, 106.66s/it]
fft (200, 1500, 1)
50%|        | 2/4 [03:22<03:20, 100.40s/it]
fft (200, 1500, 1)
75%|       | 3/4 [04:59<01:38, 98.70s/it]
fft (200, 1500, 1)
100%|| 4/4 [06:31<00:00, 97.90s/it]

```



4.1.3 To increase stability, we consider M independent time windows of size T to compute M empirical cross correlations

$$C_{T,N}^{(m)}(\tau, x_j, x_l) = \frac{1}{T - |\tau|} \int_{\tau_-}^{T-\tau_+} u^{(m)}(t, x_j) u^{(m)}(t + \tau, x_l) dt.$$

4.1.4 Then, we compute the average of these cross correlations

$$C_{T,N,M}(\tau, x_j, x_l) = \frac{1}{M} \sum_{m=1}^M C_{T,N}^{(m)}(\tau, x_j, x_l).$$

```
[ ]: def windows_cross_correlations(T, M):
    """ T : window size
        M : Number of windows"""
    C_T_N_M = []
    for m in tqdm(range(M)):
        compteur = 0
        C_T_N_m = []
        dft_random = DFT(omega, np.random.randn(200, omega.shape[0]))[:, :, np.
        ↪newaxis]
        n_hat = dft_random * np.sqrt(PSD(omega))[np.newaxis, :, np.newaxis]
        for j, xj in enumerate(receivers_position):
            for l, xl in enumerate(receivers_position):

                compteur+=1
                new_xj = np.array([xj[0], xj[2]])
                new_xl = np.array([xl[0], xl[2]])
                new_zr = np.array([zr[0], zr[2]])
                C_T_N = cross_correlation(T, tau, n_hat, new_xj, new_xl, ↪
        ↪new_sources_position, omega, new_zr)
                C_T_N_m.append(C_T_N)
        C_T_N_m = np.array(C_T_N_m)

        print("C_T_N_m", C_T_N_m.shape, "compteur", compteur)
        C_T_N_M.append(C_T_N_m)

    C_T_N_M = np.array(C_T_N_M)
    print("C_T_N_M", C_T_N_M.shape)
    return C_T_N_M
```

```
C_T_N_M = windows_cross_correlations(T=500, M=1)
```

```
0%|          | 0/1 [00:00<?, ?it/s]
```

4.2 Question 2.b

4.2.1 We compute the CC-KM images

$$I_{T,N,M}(y^S) = \sum_{j=1}^5 C_{T,N,M}(|x_j - y^S| + |y^S - x_l|, x_j, x_l).$$

```
[17]: window_size = 20
      number_elements = 40

      ys = ys_plane(zr, window_size = window_size, number_elements = number_elements)
```

x and z reflector location [-5 65]

```
[18]: ys0 = ys.shape[0]
      ys1 = ys.shape[1]

      @jit(nopython =True)
      def CC_KM_image(T, M, ys):
          """ T : window size
              M : Number of windows"""
          C_T_N_M = np.zeros((ys0*ys1), dtype=np.complex128)
          for m in range(M):
              C_T_N_m = np.zeros((ys0*ys1), dtype=np.complex128)
              ys_reshape = ys.reshape(ys0*ys1, 2)
              compteur = 0
              dft_random = DFT(omega, np.random.randn(200,omega.shape[0]))[:, :, np.
→newaxis]
              n_hat = dft_random * np.sqrt(PSD(omega))[np.newaxis, :, np.newaxis]
              for j, xj in enumerate(receivers_position):
                  for l, xl in enumerate(receivers_position):

                      new_xj = np.array([xj[0], xj[2]])
                      new_xl = np.array([xl[0], xl[2]])
                      new_zr = np.array([zr[0], zr[2]])
                      tau = np.sqrt(np.sum((new_xj[np.newaxis, :]-ys_reshape)**2, axis_
→= 1)) + np.sqrt(np.sum((ys_reshape-new_xl[np.newaxis, :])**2, axis = 1))

                      print("M", m, "COMPTEUR", compteur)
                      compteur+=1
                      C_T_N = cross_correlation(T, tau, n_hat, new_xj, new_xl,
→new_sources_position, omega, new_zr)
                      C_T_N_m += C_T_N

              C_T_N_M += C_T_N_m

      return C_T_N_M / M
```

4.3 Question 2.c

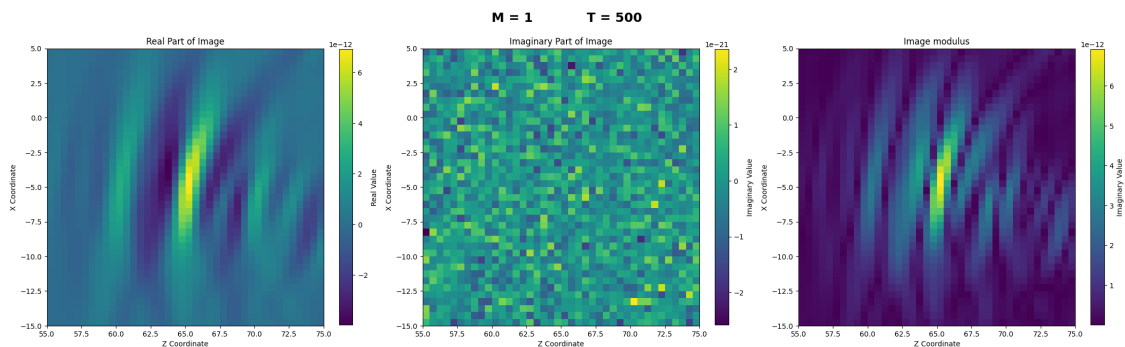
$T = 500$ and $M = [1, 3, 6, 10]$

```
[ ]: T = 500
M = 1

cc_km_imege1 = CC_KM_image(T, M, ys)

print(cc_km_imege1.shape, ys.shape)
plot_image(cc_km_imege1, f"M = {M}"          T = {T}", imaginary=True)
```

```
M 0 COMPTEUR 0
M 0 COMPTEUR 1
M 0 COMPTEUR 2
M 0 COMPTEUR 3
M 0 COMPTEUR 4
M 0 COMPTEUR 5
M 0 COMPTEUR 6
M 0 COMPTEUR 7
M 0 COMPTEUR 8
M 0 COMPTEUR 9
M 0 COMPTEUR 10
M 0 COMPTEUR 11
M 0 COMPTEUR 12
M 0 COMPTEUR 13
M 0 COMPTEUR 14
M 0 COMPTEUR 15
M 0 COMPTEUR 16
M 0 COMPTEUR 17
M 0 COMPTEUR 18
M 0 COMPTEUR 19
M 0 COMPTEUR 20
M 0 COMPTEUR 21
M 0 COMPTEUR 22
M 0 COMPTEUR 23
M 0 COMPTEUR 24
(1600,) (40, 40, 2)
```



<Figure size 640x480 with 0 Axes>

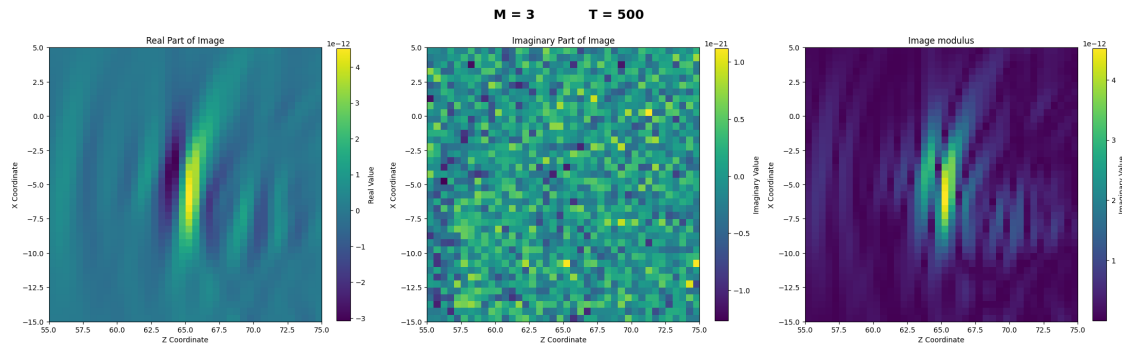
```
[20]: T = 500
      M = 3

      cc_km_imege2 = CC_KM_image(T, M , ys)

      plot_image(cc_km_imege2, f"M = {M}          T = {T}", imaginary=True)
```

```
M 0 COMPTEUR 0
M 0 COMPTEUR 1
M 0 COMPTEUR 2
M 0 COMPTEUR 3
M 0 COMPTEUR 4
M 0 COMPTEUR 5
M 0 COMPTEUR 6
M 0 COMPTEUR 7
M 0 COMPTEUR 8
M 0 COMPTEUR 9
M 0 COMPTEUR 10
M 0 COMPTEUR 11
M 0 COMPTEUR 12
M 0 COMPTEUR 13
M 0 COMPTEUR 14
M 0 COMPTEUR 15
M 0 COMPTEUR 16
M 0 COMPTEUR 17
M 0 COMPTEUR 18
M 0 COMPTEUR 19
M 0 COMPTEUR 20
M 0 COMPTEUR 21
M 0 COMPTEUR 22
M 0 COMPTEUR 23
M 0 COMPTEUR 24
M 1 COMPTEUR 0
M 1 COMPTEUR 1
M 1 COMPTEUR 2
M 1 COMPTEUR 3
M 1 COMPTEUR 4
M 1 COMPTEUR 5
M 1 COMPTEUR 6
M 1 COMPTEUR 7
M 1 COMPTEUR 8
M 1 COMPTEUR 9
M 1 COMPTEUR 10
M 1 COMPTEUR 11
```

M 1 COMPTEUR 12
M 1 COMPTEUR 13
M 1 COMPTEUR 14
M 1 COMPTEUR 15
M 1 COMPTEUR 16
M 1 COMPTEUR 17
M 1 COMPTEUR 18
M 1 COMPTEUR 19
M 1 COMPTEUR 20
M 1 COMPTEUR 21
M 1 COMPTEUR 22
M 1 COMPTEUR 23
M 1 COMPTEUR 24
M 2 COMPTEUR 0
M 2 COMPTEUR 1
M 2 COMPTEUR 2
M 2 COMPTEUR 3
M 2 COMPTEUR 4
M 2 COMPTEUR 5
M 2 COMPTEUR 6
M 2 COMPTEUR 7
M 2 COMPTEUR 8
M 2 COMPTEUR 9
M 2 COMPTEUR 10
M 2 COMPTEUR 11
M 2 COMPTEUR 12
M 2 COMPTEUR 13
M 2 COMPTEUR 14
M 2 COMPTEUR 15
M 2 COMPTEUR 16
M 2 COMPTEUR 17
M 2 COMPTEUR 18
M 2 COMPTEUR 19
M 2 COMPTEUR 20
M 2 COMPTEUR 21
M 2 COMPTEUR 22
M 2 COMPTEUR 23
M 2 COMPTEUR 24



<Figure size 640x480 with 0 Axes>

```
[21]: T = 500
      M = 6

      cc_km_imege3 = CC_KM_image(T, M , ys)

      plot_image(cc_km_imege3, f"M = {M}          T = {T}", imaginary=True)
```

```
M 0 COMPTEUR 0
M 0 COMPTEUR 1
M 0 COMPTEUR 2
M 0 COMPTEUR 3
M 0 COMPTEUR 4
M 0 COMPTEUR 5
M 0 COMPTEUR 6
M 0 COMPTEUR 7
M 0 COMPTEUR 8
M 0 COMPTEUR 9
M 0 COMPTEUR 10
M 0 COMPTEUR 11
M 0 COMPTEUR 12
M 0 COMPTEUR 13
M 0 COMPTEUR 14
M 0 COMPTEUR 15
M 0 COMPTEUR 16
M 0 COMPTEUR 17
M 0 COMPTEUR 18
M 0 COMPTEUR 19
M 0 COMPTEUR 20
M 0 COMPTEUR 21
M 0 COMPTEUR 22
M 0 COMPTEUR 23
M 0 COMPTEUR 24
M 1 COMPTEUR 0
```

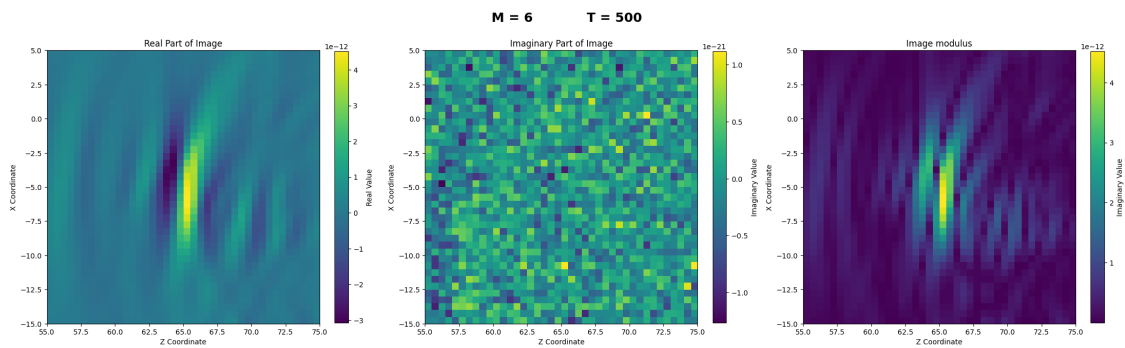
M 1 COMPTEUR 1
M 1 COMPTEUR 2
M 1 COMPTEUR 3
M 1 COMPTEUR 4
M 1 COMPTEUR 5
M 1 COMPTEUR 6
M 1 COMPTEUR 7
M 1 COMPTEUR 8
M 1 COMPTEUR 9
M 1 COMPTEUR 10
M 1 COMPTEUR 11
M 1 COMPTEUR 12
M 1 COMPTEUR 13
M 1 COMPTEUR 14
M 1 COMPTEUR 15
M 1 COMPTEUR 16
M 1 COMPTEUR 17
M 1 COMPTEUR 18
M 1 COMPTEUR 19
M 1 COMPTEUR 20
M 1 COMPTEUR 21
M 1 COMPTEUR 22
M 1 COMPTEUR 23
M 1 COMPTEUR 24
M 2 COMPTEUR 0
M 2 COMPTEUR 1
M 2 COMPTEUR 2
M 2 COMPTEUR 3
M 2 COMPTEUR 4
M 2 COMPTEUR 5
M 2 COMPTEUR 6
M 2 COMPTEUR 7
M 2 COMPTEUR 8
M 2 COMPTEUR 9
M 2 COMPTEUR 10
M 2 COMPTEUR 11
M 2 COMPTEUR 12
M 2 COMPTEUR 13
M 2 COMPTEUR 14
M 2 COMPTEUR 15
M 2 COMPTEUR 16
M 2 COMPTEUR 17
M 2 COMPTEUR 18
M 2 COMPTEUR 19
M 2 COMPTEUR 20
M 2 COMPTEUR 21
M 2 COMPTEUR 22
M 2 COMPTEUR 23

M 2 COMPTEUR 24
M 3 COMPTEUR 0
M 3 COMPTEUR 1
M 3 COMPTEUR 2
M 3 COMPTEUR 3
M 3 COMPTEUR 4
M 3 COMPTEUR 5
M 3 COMPTEUR 6
M 3 COMPTEUR 7
M 3 COMPTEUR 8
M 3 COMPTEUR 9
M 3 COMPTEUR 10
M 3 COMPTEUR 11
M 3 COMPTEUR 12
M 3 COMPTEUR 13
M 3 COMPTEUR 14
M 3 COMPTEUR 15
M 3 COMPTEUR 16
M 3 COMPTEUR 17
M 3 COMPTEUR 18
M 3 COMPTEUR 19
M 3 COMPTEUR 20
M 3 COMPTEUR 21
M 3 COMPTEUR 22
M 3 COMPTEUR 23
M 3 COMPTEUR 24
M 4 COMPTEUR 0
M 4 COMPTEUR 1
M 4 COMPTEUR 2
M 4 COMPTEUR 3
M 4 COMPTEUR 4
M 4 COMPTEUR 5
M 4 COMPTEUR 6
M 4 COMPTEUR 7
M 4 COMPTEUR 8
M 4 COMPTEUR 9
M 4 COMPTEUR 10
M 4 COMPTEUR 11
M 4 COMPTEUR 12
M 4 COMPTEUR 13
M 4 COMPTEUR 14
M 4 COMPTEUR 15
M 4 COMPTEUR 16
M 4 COMPTEUR 17
M 4 COMPTEUR 18
M 4 COMPTEUR 19
M 4 COMPTEUR 20
M 4 COMPTEUR 21

```

M 4 COMPTEUR 22
M 4 COMPTEUR 23
M 4 COMPTEUR 24
M 5 COMPTEUR 0
M 5 COMPTEUR 1
M 5 COMPTEUR 2
M 5 COMPTEUR 3
M 5 COMPTEUR 4
M 5 COMPTEUR 5
M 5 COMPTEUR 6
M 5 COMPTEUR 7
M 5 COMPTEUR 8
M 5 COMPTEUR 9
M 5 COMPTEUR 10
M 5 COMPTEUR 11
M 5 COMPTEUR 12
M 5 COMPTEUR 13
M 5 COMPTEUR 14
M 5 COMPTEUR 15
M 5 COMPTEUR 16
M 5 COMPTEUR 17
M 5 COMPTEUR 18
M 5 COMPTEUR 19
M 5 COMPTEUR 20
M 5 COMPTEUR 21
M 5 COMPTEUR 22
M 5 COMPTEUR 23
M 5 COMPTEUR 24

```



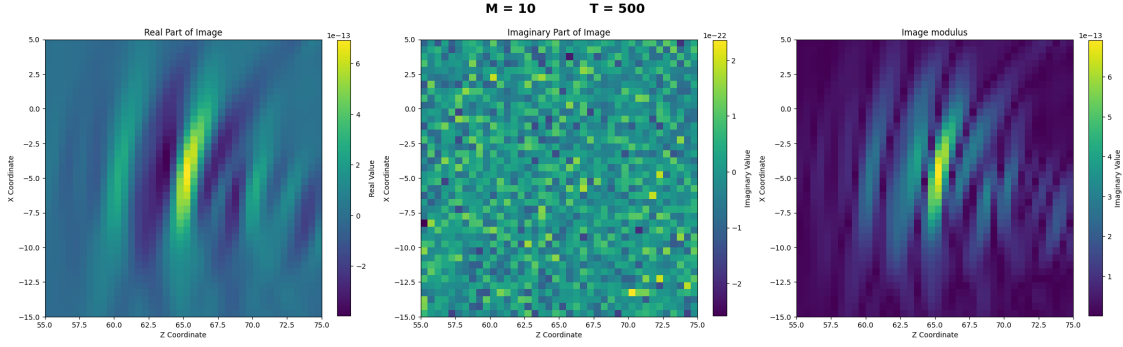
<Figure size 640x480 with 0 Axes>

```

[22]: cc_km_10 = (cc_km_imege1 + cc_km_imege2*3 * cc_km_imege3*6)/10

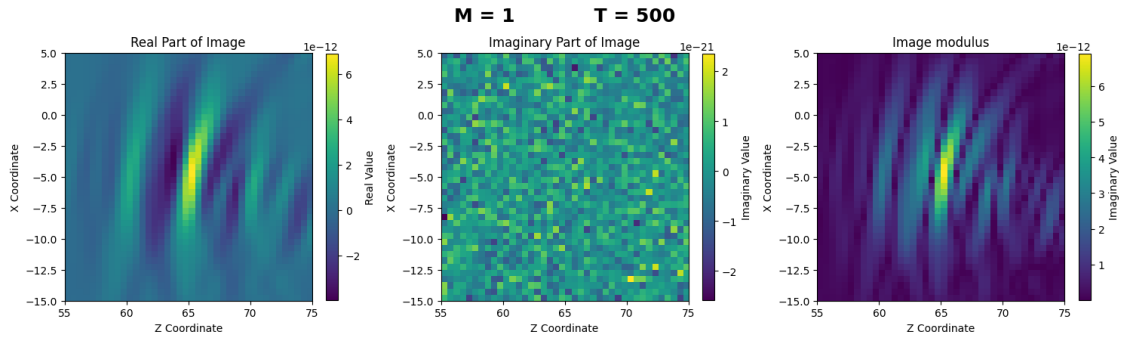
plot_image(cc_km_10, f"M = {10}                      T = {T}", imaginary=True)

```

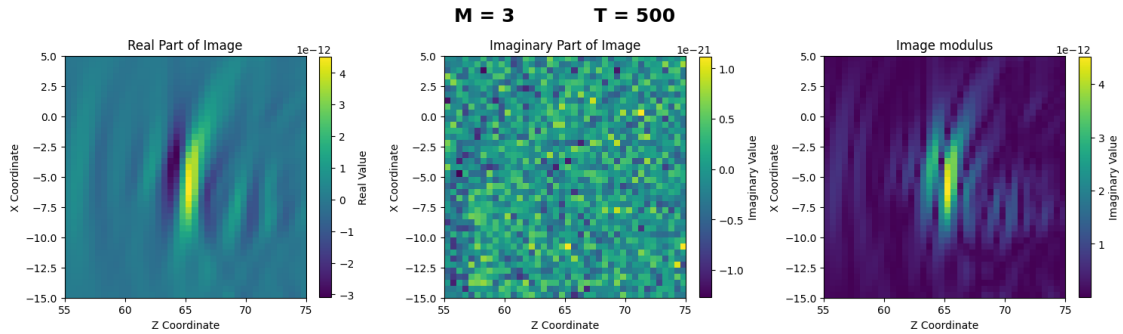


<Figure size 640x480 with 0 Axes>

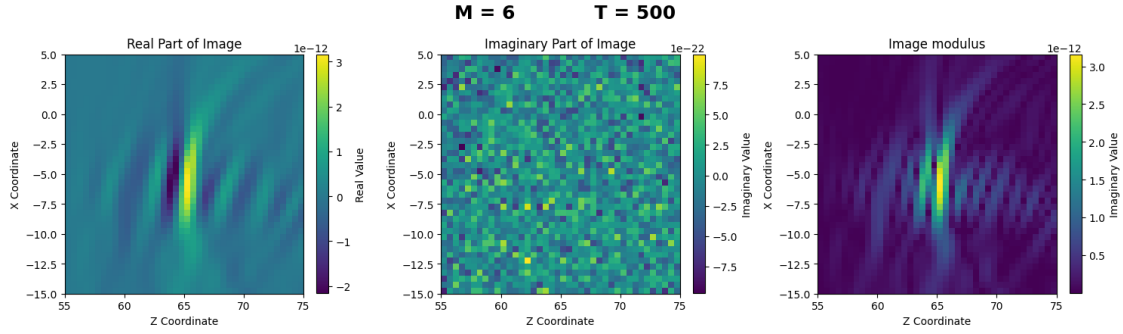
```
[25]: plot_image(cc_km_imege1, f"M = {1}          T = {500}", imaginary=True)
      plot_image(cc_km_imege2, f"M = {3}          T = {500}", imaginary=True)
      plot_image(cc_km_imege3, f"M = {6}          T = {500}", imaginary=True)
      plot_image(cc_km_10,   f"M = {10}         T = {500}", imaginary=True)
```



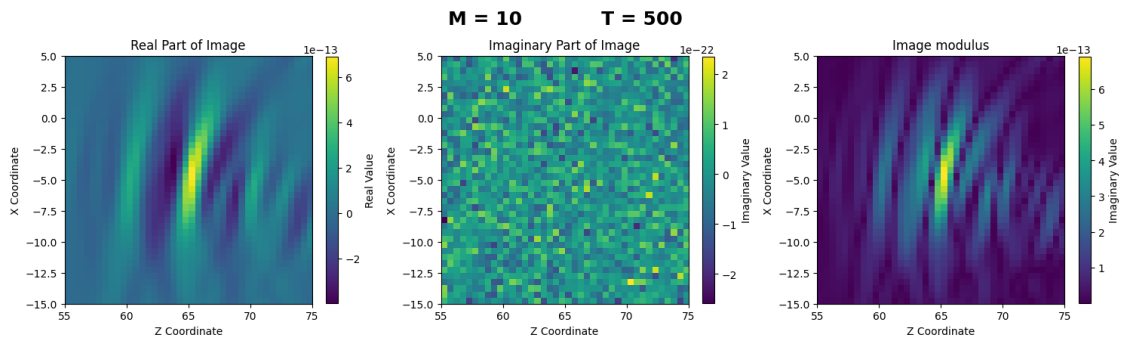
<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



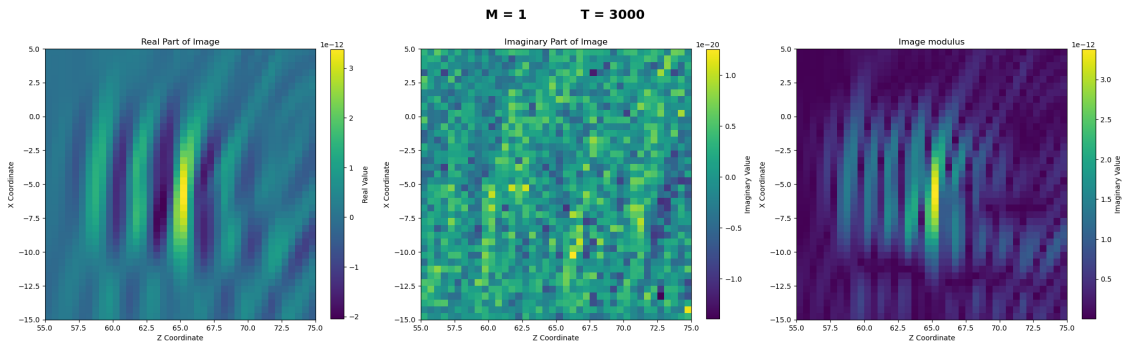
<Figure size 640x480 with 0 Axes>



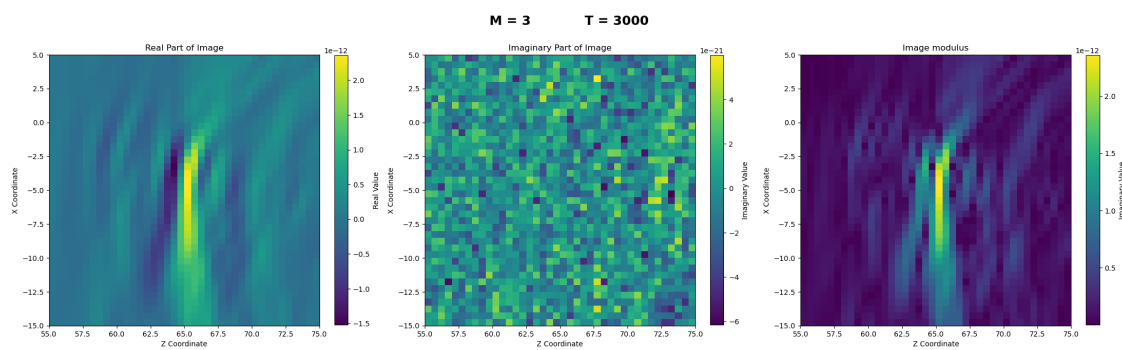
<Figure size 640x480 with 0 Axes>

$T = 3000$ and $M = [1, 3, 6, 10]$

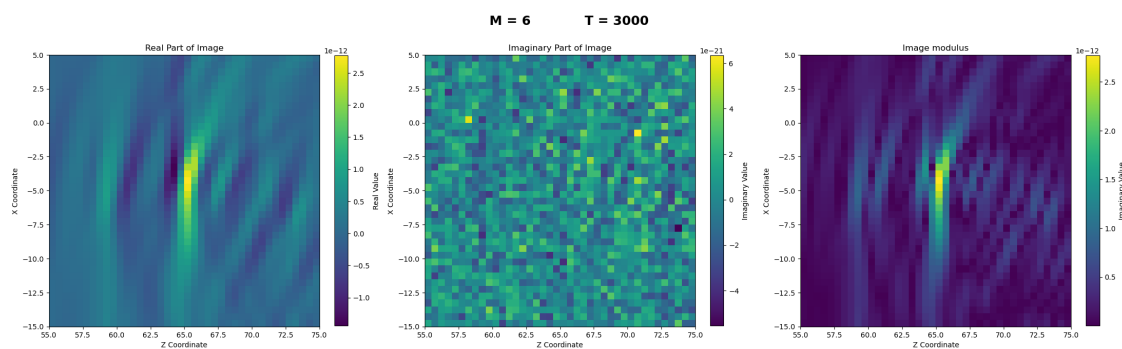
```
[ ]: plot_image(cc_km_imege4, f"M = {1}          T = {3000}", imaginary=True)
      plot_image(cc_km_imege5, f"M = {3}          T = {3000}", imaginary=True)
      plot_image(cc_km_imege6, f"M = {6}          T = {3000}", imaginary=True)
      plot_image(cc_km_10_3000, f"M = {10}       T = {3000}", imaginary=True)
```



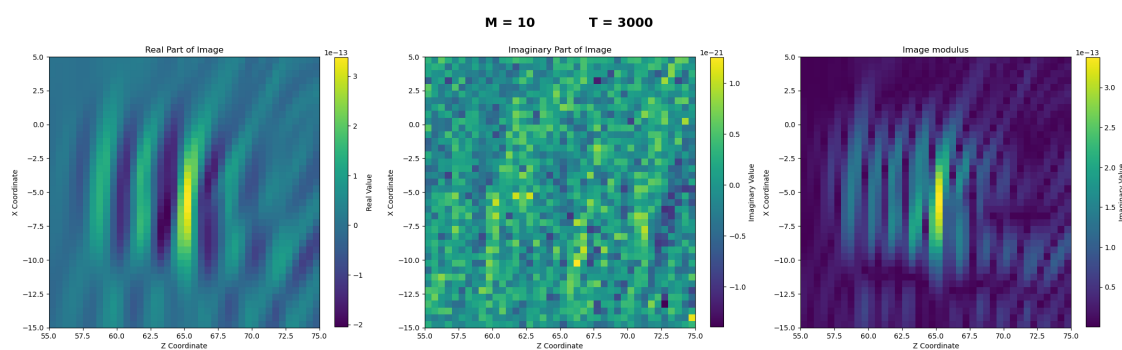
<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>