Communications Project

Antenna Arrays

Homework Results

Authors:

Saqib Faraz	458317
Ankita Kannan Iyer	457620
Harsh Godhani	461986
Richard Grünert	289427
Kanaiyalal Thummar	435534

1.6.2023

Homework Tasks

```
HW6)

due June 1st 23:59 LT

Simulate the radiation pattern (Array factor) of different antenna array
arrangements.

1a) 5 x 5 antenna , squared grid, 0.5 lambda wavelength (WL) spacing for vertical
and 60° off-vertical beam pointing along one array axis Plot the resulting
radiation pattern in spherical top view (directional cosines) and ONE cross
section through the main beam.

1b) same as 1a, but for 1.0 and 1.5 WL

2) Plot the array geometry and mark the phase distribution for the off-vertical
scenario for 0.5 WL and 1.5 WL.

3a) same as 1a) but equilateral grid structure , e.g. 19 antennas

3b) same as 3a), but for 1.0 and 1.5 WL
```

1 Introduction

Antenna arrays can be constructed from multiple similar antennas to increase the gain in comparison to a single of these antennas. [1] Under the assumption that every antenna has the same radiation pattern and that there is no coupling between the antennas, one can calculate an *Array Factor* (AF) which is simply multiplied with the base field pattern $E(\phi, \theta)$ to yield the pattern of the total array $Y(\phi, \theta)$.

$$Y(\phi, \theta) = E(\phi, \theta) \cdot AF$$

The array factor can be calculated with knowledge of the positions of the elements, e.g. given as a complex numbers, and their input signals (magnitude and phase), which depend on the steering direction (θ_d, ϕ_d). It is independent of the fundamental radiation pattern used, which makes it so useful.

$$AF(\hat{r}) = \sum_{n=0}^{N-1} w_n \cdot e^{-jk\hat{r}r_n}$$

Where N is the total number of antennas, w_n are the weights, i.e. the phases required to steer the array factor in a certain direction, r_n are the individual antenna positions, and $\hat{r}$ is the unit vector pointing to the observation location. For a planar (2D) array this turns out to be

$$AF(\theta, \phi) = \sum_{n=0}^{N-1} w_n \cdot e^{-jk \cdot \sin\theta (x_n \cdot \cos\phi + y_n \cdot \sin\phi)}$$

with

$$w_n = e^{jk \cdot \sin\theta_d (x_n \cdot \cos\phi_d + y_n \cdot \sin\phi_d)}$$

The array factor is commonly written in terms of the directional cosines u, v

$$AF(\theta, \phi) = \sum_{n=0}^{N-1} w_n \cdot e^{-jk \cdot (x_n u + y_n v)}$$

$$u = \sin\theta \cos\phi$$

$$v = \sin\theta \sin\phi$$

Additionally, one has to technically normalize this factor in order to not increase the power artificially when multiplying with the actual radiation pattern.

2 Square 5x5 Array

The following figures show the array factor patterns for a 5x5 planar antenna array with different spacings (d) and steering angles (either vertical or $\theta_d = 60\,°$), calculated with python. The array geometry can be seen in figure 7.

One can see grating lobes appear on the edges of some radiation patterns due to the finite antenna distances involved. As the distance between the antennas goes down, the angular distance towards the grating lobes increases, i.e. angular ambiguities appear "later" and the beam can be safely steered to greater angles. However, this also results in a greater beamwidth which might not be desired.

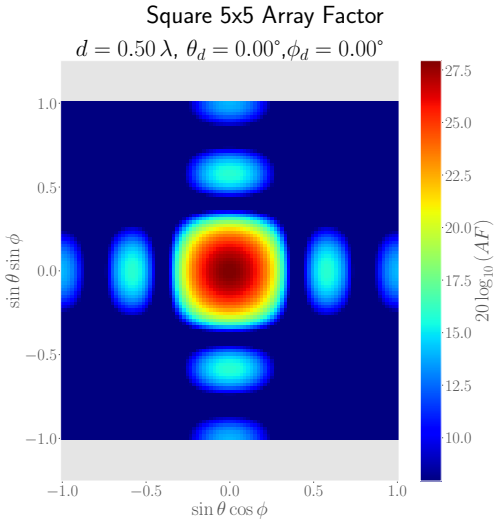

Figure 1: Square 5x5 vertically steered radiation pattern for 0.5λ spacing.

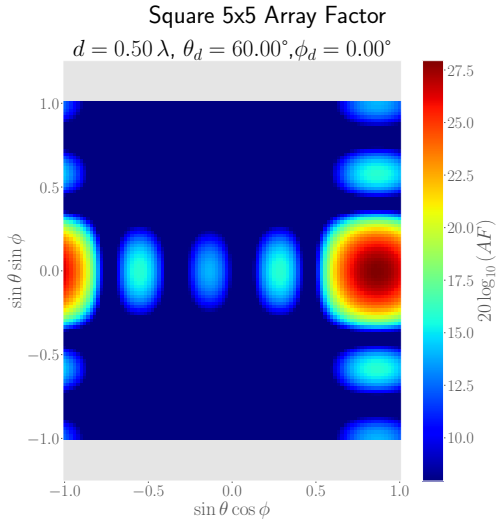

Figure 2: Square 5x5 off-vertically steered radiation pattern for 0.5λ spacing.

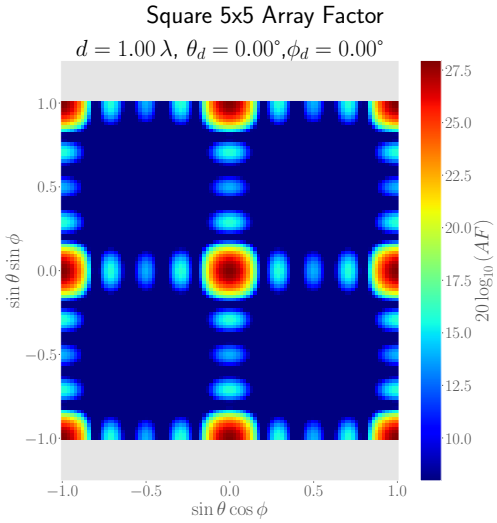

Figure 3: Square 5x5 vertically steered radiation pattern for 1.0λ spacing.

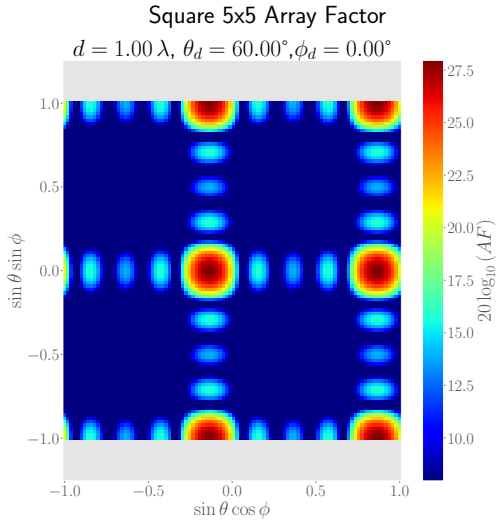

Figure 4: Square 5x5 off-vertically steered radiation pattern for 1.0λ spacing.

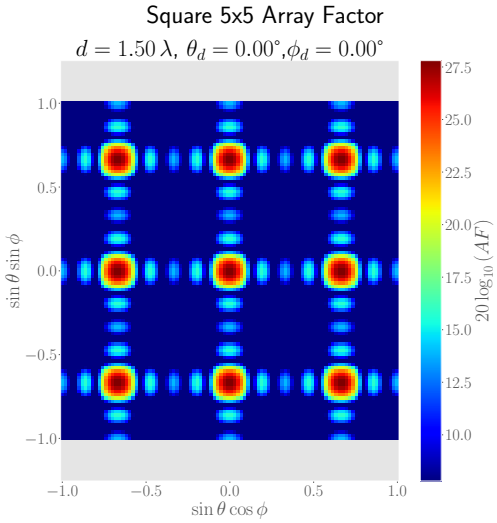

Figure 5: Square 5x5 vertically steered radiation pattern for 1.5λ spacing.

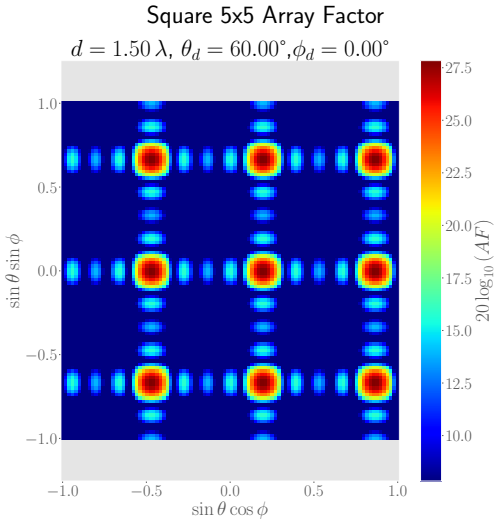

Figure 6: Square 5x5 off-vertically steered radiation pattern for 1.5λ spacing.

In the cases of figure 3 and 4, the grating lobes become even more apparent. Although, technically the corner ones do not count since they lie beyond the array horizon. In this case and, also with $1.5\,\lambda$, spacing, you cannot distinguish the pattern resulting from the (wanted) $60\,°$ steering angle from a smaller one (e.g. $10\,°$), leading to unwanted ambiguities.

2.1 Phase Distribution

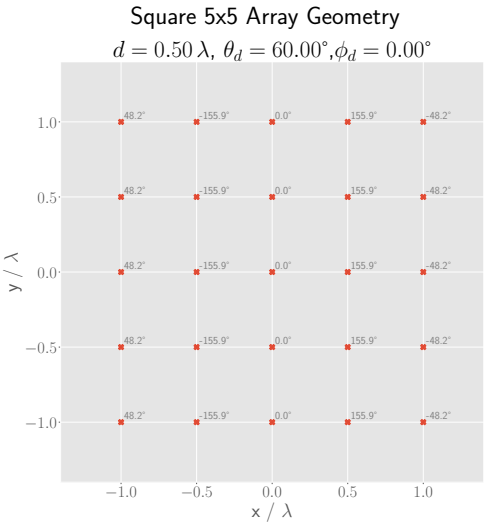

Figure 7: Square 5x5 Geometry with steering phases for 0.5λ spacing.

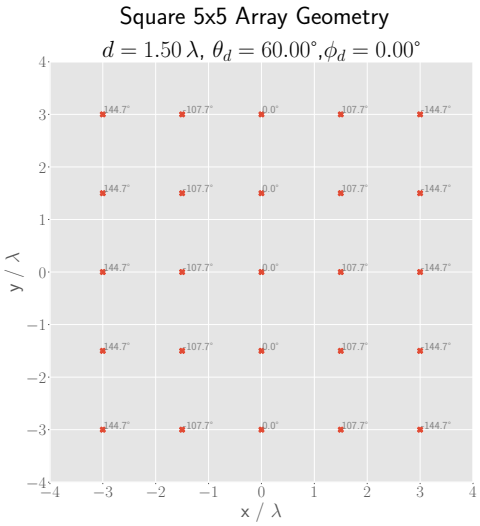

Figure 8: Square 5x5 Geometry with steering phases for 1.5λ spacing.

In figures 7 and 8, the array geometries along with the individual steering phases are shown for a target position of $\theta_d = 60\,°, \phi_d = 0\,°$. You can see that the phases are constant along the columns since any incident wave from a far-away target will have the same time of arrival for these antennas. Comparing

the different spacings, the angle differences in the $d = 1.5\lambda$ case are greater, which makes sense since their spacing is larger which results in more time lag.

3 Equilateral Array

The same figures have been generated for a 25-antenna array where the spacings were still equal but the grid of antennas is made from equilateral triangles. This reduces the spacing between the rows but also enlargens the array towards the left and right. This "squishing" effect can also be seen in the radiation patterns. The geometry can be seen in figure 9.

The same principle regarding the spacing, grating lobes and beamwidth as in the square array case applies. However, the grating lobes along the u-direction are farther away and so appear at a greater angle.

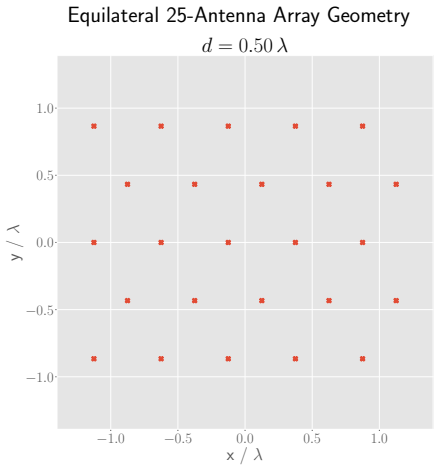

Figure 9: Equilateral grid array structure for $d = 0.5\,\lambda$

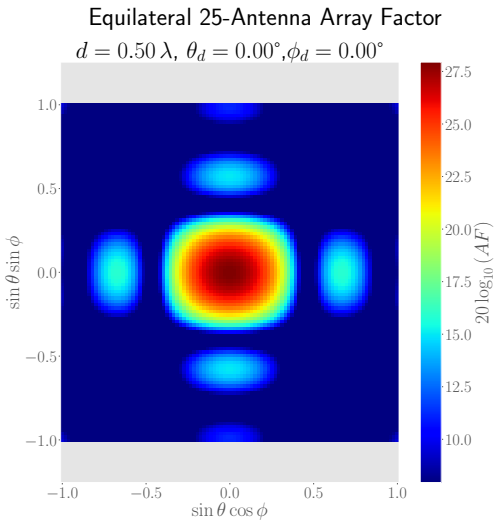

Figure 10: Equilateral vertically steered radiation pattern for 0.5λ spacing.

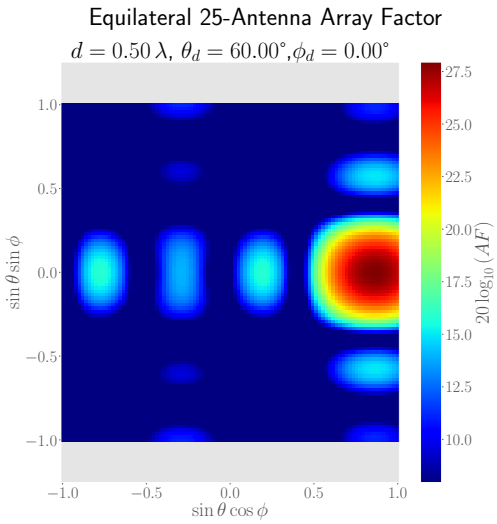

Figure 11: Equilateral off-vertically steered radiation pattern for 0.5λ spacing.

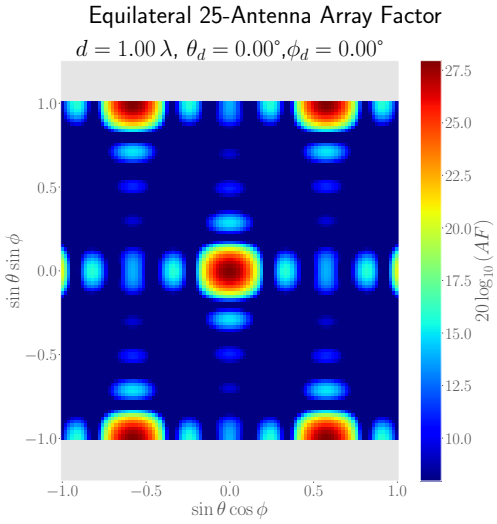

Figure 12: Equilateral vertically steered radiation pattern for 1.0λ spacing.

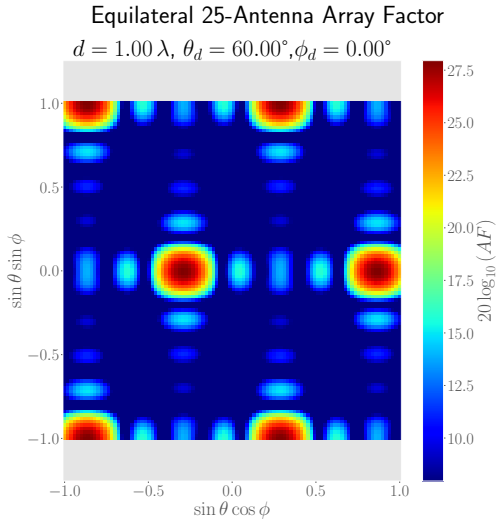

Figure 13: Equilateral off-vertically steered radiation pattern for 1.0λ spacing.

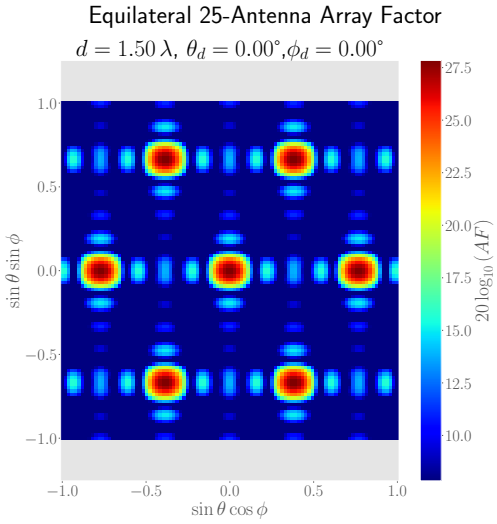

Figure 14: Equilateral vertically steered radiation pattern for 1.5λ spacing.

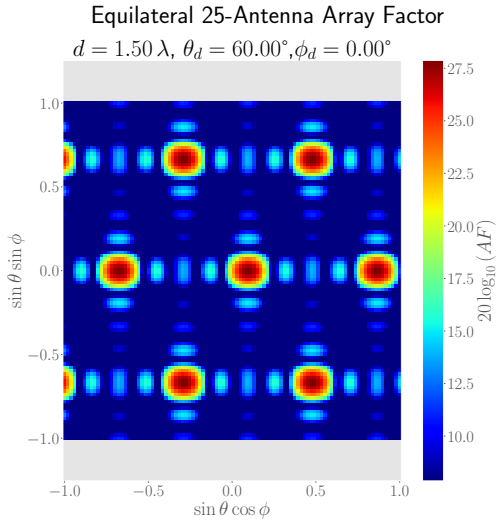

Figure 15: Equilateral off-vertically steered radiation pattern for 1.5λ spacing.

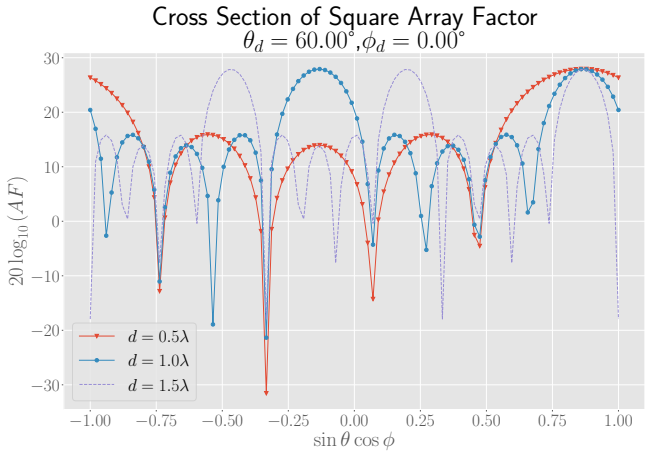

Figure 16: Comparison of cross-sections of the square array factor for different spacings.

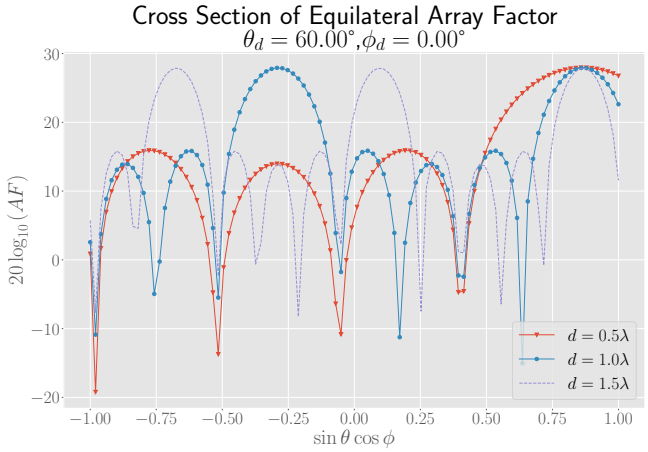

Figure 17: Comparison of cross-sections of the equilateral array factor for different spacings.

References

Literature

[1] Constantine A. Balanis. *Antenna Theory, Analysis and Design*. ISBN 0-471-66782-X. John Wiley & Sons, Inc., Feb. 2005.

[2] Peter Joseph Bevelacqua. *The Array Factor*. URL: https://www.antenna-theory.com/arrays/arrayfactor.php (visited on 05/28/2023).

[3] David Hysell. *Antennas and Radar for Environmental Scientists and Engineers*. ISBN 9781108164122. Cambridge University Press, Feb. 2018. URL: https://doi.org/10.1017/9781108164122.

Software Used

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.7. Python Software Foundation. 2019. URL: https://www.python.org/.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Python Code

```python
import matplotlib.pyplot as plt
import numpy as np


class AntennaArray:
    def __init__(self, lam):
        self.pos = self.get_geometry()
        self.lam = lam

    def get_geometry(self):
        """
        should return the array of complex positions
        overridden by child classes
        """
        return np.zeros(1)

    def get_array_factor_dcos(self, dcosx, dcosy, target_theta, target_phi):
        """
        calculates the array factor for a specific position
        (directional cosines)
        """
        steering_weights = self.get_steering_weights_for_target(target_theta, target_phi)

        xn = self.pos.real
        yn = self.pos.imag

        AF = np.sum(steering_weights * np.exp(-1j*2*np.pi/self.lam * (xn * dcosx + yn * dcosy)))

        return AF

    def get_array_factor(self, theta, phi, target_theta, target_phi):
        """
        calculates the array factor for a specific position (set of angles)
        TODO: normalize this. e.g. theta=0, phi=0 leads to AF=9 (3x3) which
        skews the power
        """
        steering_weights = self.get_steering_weights_for_target(target_theta, target_phi)

        xn = self.pos.real
        yn = self.pos.imag

        AF = np.sum(steering_weights * np.exp(-1j*2*np.pi/self.lam * np.sin(theta) * (xn * np.cos(phi)
        ↪  + yn * np.sin(phi))))

        return AF

    def get_steering_weights_for_target(self, target_theta, target_phi):
        """
        calculates weights (phases) based on target location
        (phased weights)
        """

        xn = self.pos.real
        yn = self.pos.imag

        angle = 2*np.pi / self.lam * np.sin(target_theta) * (xn * np.cos(target_phi) + yn *
        ↪  np.sin(target_phi))

        return np.exp(1j*angle)


class SquareArray(AntennaArray):
    def __init__(self, lam, size, spacing):
        self.size = size
```

```
        self.spacing = spacing
        super().__init__(lam)

    def get_geometry(self):
        nx, ny = self.size
        x = np.linspace(0, 1, nx)
        y = np.linspace(0, 1, ny)
        xv, yv = np.meshgrid(x, y)

        # define complex number and scale
        pos = self.spacing * ((nx-1) * xv + 1j * (ny-1) * yv)
        # center around zero
        pos = pos - ((np.max(pos.real) + 1j*np.max(pos.imag))) / 2

        return pos


class EquilateralArray(AntennaArray):
    def __init__(self, lam, size, spacing):
        self.size = size
        self.spacing = spacing
        super().__init__(lam)

    def get_geometry(self):
        nx, ny = self.size
        x = np.linspace(0, 1, nx)
        y = np.linspace(0, 1, ny)
        xv, yv = np.meshgrid(x, y)

        x_shift = np.zeros(self.size)
        # modify every second row
        x_shift[1::2,:] = -self.spacing / 2

        # define complex number and scale
        pos = self.spacing * ((nx-1) * xv + 1j * (ny-1) * yv * np.sqrt(3)/2)
        pos.real = pos.real - x_shift
        # center around zero
        pos = pos - ((np.max(pos.real) + 1j*np.max(pos.imag))) / 2

        return pos


def plot_phase_numbers(array, target_theta, target_phi, scale=1):
    weights = array.get_steering_weights_for_target(
        np.deg2rad(target_theta),
        np.deg2rad(target_phi))
    equilat_phases = np.rad2deg(np.angle(weights)).flatten()
    text_offs = 0.02
    for i, pos in enumerate((array.pos / scale).flatten()):
        plt.text(pos.real + text_offs, pos.imag + text_offs,
                 "%.1f°" % equilat_phases[i],
                 color="gray",
                 size=20)


def main():

    f = 100e6
    c = 3e8
    lam = c / f

    target_phi = 0
    target_theta = 60
    target_azm = np.mod(90 - target_phi, 360)

    spacing = 0.5 * lam

```

```
    square_array = SquareArray(lam, (5, 5), spacing)
    equilat_array = EquilateralArray(lam, (5, 5), spacing)

    dcx = np.linspace(-1, 1, 100)
    dcy = np.linspace(-1, 1, 100)

    # ugly but does the job
    afs_square = np.zeros([100, 100])
    afs_equilat = np.zeros([100, 100])
    for i, dx in enumerate(dcx):
        for k, dy in enumerate(dcy):
            af1 = square_array.get_array_factor_dcos(dx, dy,
                                                     np.deg2rad(target_theta),
                                                     np.deg2rad(target_azm))
            af2 = equilat_array.get_array_factor_dcos(dx, dy,
                                                      np.deg2rad(target_theta),
                                                      np.deg2rad(target_azm))
            afs_square[i, k] = np.abs(af1)
            afs_equilat[i, k] = np.abs(af2)

    afs_square_pow = 20*np.log10(afs_square)
    afs_equilat_pow = 20*np.log10(afs_equilat)


    # ------------------------------------------------------------------------
    # Plotting
    # ------------------------------------------------------------------------

    # does not work for some reason
    # dist = np.sqrt(dcx**2 + dcy**2)
    # afs_pow[dist < 1] = np.NAN

    # radiation pattern color plot
    plt.figure()
    plt.suptitle("Square %dx%d Array Factor"
                 % (square_array.size))
    plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
              % (spacing / lam, target_theta, target_phi))
    plt.axis('equal')
    plt.pcolor(dcx, dcy, afs_square_pow, cmap='jet')
    cbar = plt.colorbar()
    cbar.ax.set_ylabel("$20\\log_{10}{(AF)}$")
    plt.clim(np.max(afs_square_pow)-20, np.max(afs_square_pow))
    plt.xlabel("$\\sin{\\theta}\\cos{\\phi}$")
    plt.ylabel("$\\sin{\\theta}\\sin{\\phi}$")
    plt.savefig("outputs/square-%.2f-lambda-%.2f-theta-%.2f-phi-radpat.pdf"
                % (spacing/lam, target_theta, target_phi))

    plt.figure()
    plt.suptitle("Equilateral %d-Antenna Array Factor"
                 % (np.prod(square_array.size)))
    plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
              % (spacing / lam, target_theta, target_phi))
    plt.axis('equal')
    plt.pcolor(dcx, dcy, afs_equilat_pow, cmap='jet')
    cbar = plt.colorbar()
    cbar.ax.set_ylabel("$20\\log_{10}{(AF)}$")
    plt.clim(np.max(afs_equilat_pow)-20, np.max(afs_equilat_pow))
    plt.xlabel("$\\sin{\\theta}\\cos{\\phi}$")
    plt.ylabel("$\\sin{\\theta}\\sin{\\phi}$")
    plt.savefig("outputs/equilat-%.2f-lambda-%.2f-theta-%.2f-phi-radpat.pdf"
                % (spacing/lam, target_theta, target_phi))


    # radiation pattern cross section plot
    plt.figure()
    plt.plot(dcx, afs_square_pow[int((len(dcx)-1)/2+1),:])
```

```
    plt.suptitle("Cross Section of Square Array Factor")
    plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
              % (spacing / lam, target_theta, target_phi))
    plt.ylabel("$20\\log_{10}{(AF)}$")
    plt.xlabel("$\\sin{\\theta}\\cos{\\phi}$")
    plt.savefig("outputs/square-%.2f-lambda-%.2f-theta-%.2f-phi-cross.pdf"
                % (spacing/lam, target_theta, target_phi))

    plt.figure()
    plt.plot(dcx, afs_square_pow[int((len(dcx)-1)/2+1),:])
    plt.suptitle("Cross Section of Equilateral Array Factor")
    plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
              % (spacing / lam, target_theta, target_phi))
    plt.ylabel("$20\\log_{10}{(AF)}$")
    plt.xlabel("$\\sin{\\theta}\\cos{\\phi}$")
    plt.savefig("outputs/equilat-%.2f-lambda-%.2f-theta-%.2f-phi-cross.pdf"
                % (spacing/lam, target_theta, target_phi))


    # geometry plot
    plt.figure()
    plt.tight_layout()
    plt.scatter(square_array.pos.real / lam,
                square_array.pos.imag / lam,
                marker="X", s=100)
    plt.suptitle("Square %dx%d Array Geometry"
                 % (square_array.size))
    plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
              % (spacing / lam, target_theta, target_phi))
    plt.xlabel("x / $\\lambda$")
    plt.ylabel("y / $\\lambda$")
    # add phase numbers as text
    plot_phase_numbers(square_array, target_theta, target_phi, lam)
    plt.xlim([-1.4, 1.4])
    plt.ylim([-1.4, 1.4])
    plt.savefig("outputs/square-%.2f-lambda-%.2f-theta-%.2f-phi-geometry.pdf"
                % (spacing/lam, target_theta, target_phi))

    plt.figure()
    plt.tight_layout()
    plt.scatter(equilat_array.pos.real / lam,
                equilat_array.pos.imag / lam,
                marker="X", s=100)
    plt.suptitle("Equilateral %d-Antenna Array Geometry"
                 % (np.prod(square_array.size)))
    #plt.title("$d=%.2f\,\\lambda$, $\\theta_d=%.2f$°,$\\phi_d=%.2f$°"
    #          % (spacing / lam, target_theta, target_phi))
    plt.title("$d=%.2f\,\\lambda$" % (spacing / lam))
    plt.xlabel("x / $\\lambda$")
    plt.ylabel("y / $\\lambda$")
    plt.axis('equal')
    # add phase numbers as text
    #plot_phase_numbers(equilat_array, target_theta, target_phi, lam)
    plt.xlim([-1.4, 1.4])
    plt.ylim([-1.4, 1.4])
    plt.savefig("outputs/equilat-%.2f-lambda-%.2f-theta-%.2f-phi-geometry.pdf"
                % (spacing/lam, target_theta, target_phi))

    plt.show()


if __name__ == "__main__":
    main()
```