# 5 Oct 23 - Activity: Solving PDEs in Spherical Coordinates

We can extend the ideas we developed from Separation of Variables to solve PDEs in spherical coordinates. We propose a similar solution to the one we used for Cartesian coordinates, functions of a single variable.

We have [shown that we can solve this problem in general](../../assets/notes/Notes-Separation_of_Variables_Spherical.pdf). We will now apply this to a specific problem. Recall these general solutions are written in the following ways:

where are the [associated Legendre polynomials](https://en.wikipedia.org/wiki/Associated_Legendre_polynomials). So that the general solution is the product of these for different values of , , and . One solution is some combination of , , and , written like this:

where and are constants. Because Laplace’s equation is linear, we can write the general solution as a sum of these solutions:

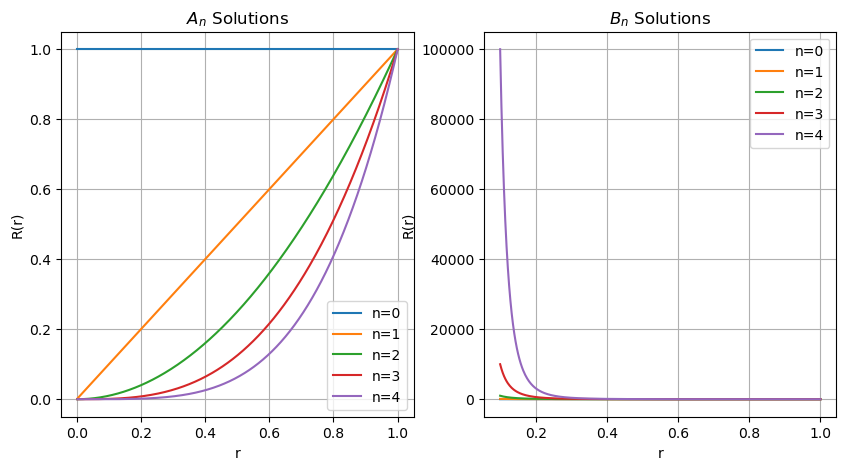
## Visualizing the Solutions

One of the major challenges with these general solutions is seeing what the solutions look like. We can use these picture to help us dtermine if we can remove one or more terms before trying to match boundary conditions. We will plot each of the solutions below with respect to their relative variables.

### Radial Solutions

Below, we are plotting the solutions and for several choices of . You can see where the solutions blow up. This is helpful for matching boundary conditions.

import matplotlib.pyplot as plt  
import numpy as np  
  
max\_n = 4  
step = 500  
max\_x = 1  
start\_x = max\_x/step  
  
# Radial coordinate  
r = np.linspace(start\_x, max\_x, step)  
  
plt.figure(figsize=(10,5))  
plt.subplot(1,2,1)  
plt.title(r'$A\_n$ Solutions')  
plt.xlabel('r')  
plt.ylabel('R(r)')  
  
for n in range(max\_n + 1):  
 radial\_solution = r\*\*n  
 plt.plot(r, radial\_solution, label=f'n={n}')  
  
plt.legend()  
plt.grid(True)  
  
  
max\_n = 4  
step = 500  
max\_x = 1  
start\_x = 0.1  
  
# Radial coordinate  
r = np.linspace(start\_x, max\_x, step)  
  
plt.subplot(1,2,2)  
plt.title(r'$B\_n$ Solutions')  
plt.xlabel('r')  
plt.ylabel('R(r)')  
  
for n in range(max\_n + 1): # for negative n  
  
 radial\_solution = 1/(r\*\*(n+1))  
 plt.plot(r, radial\_solution, label=f'n={n}')  
  
plt.legend()  
plt.grid(True)

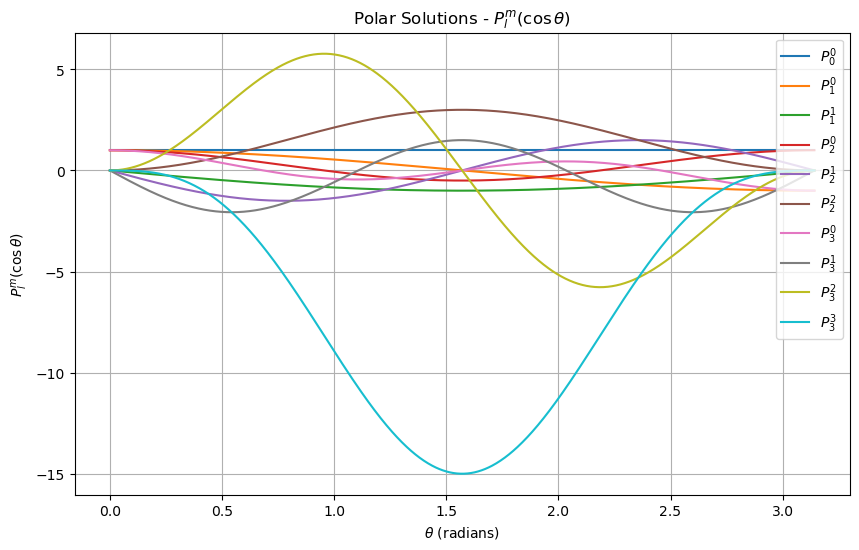


png

## The Polar Angle Solutions

These solutions are the associated Legendre Polynomials, presented below.

from scipy.special import lpmv  
  
theta = np.linspace(0, np.pi, 1000) # theta from 0 to pi  
x = np.cos(theta) # x = cos(theta)  
  
# Define a set of (l, m) pairs to plot  
lm\_pairs = [(0, 0), (1, 0), (1, 1), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2), (3, 3)]  
  
plt.figure(figsize=(10,6))  
for l, m in lm\_pairs:  
 if m <= l: # The associated Legendre functions are only defined for m <= l  
 plt.plot(theta, lpmv(m, l, x), label=f'$P\_{{{l}}}^{{{m}}}$')  
  
plt.title(r'Polar Solutions - $P\_l^m(\cos\theta)$')  
plt.xlabel(r'$\theta$ (radians)')  
plt.ylabel(r'$P\_l^m(\cos\theta)$')  
plt.legend(loc='upper right')  
plt.grid(True)

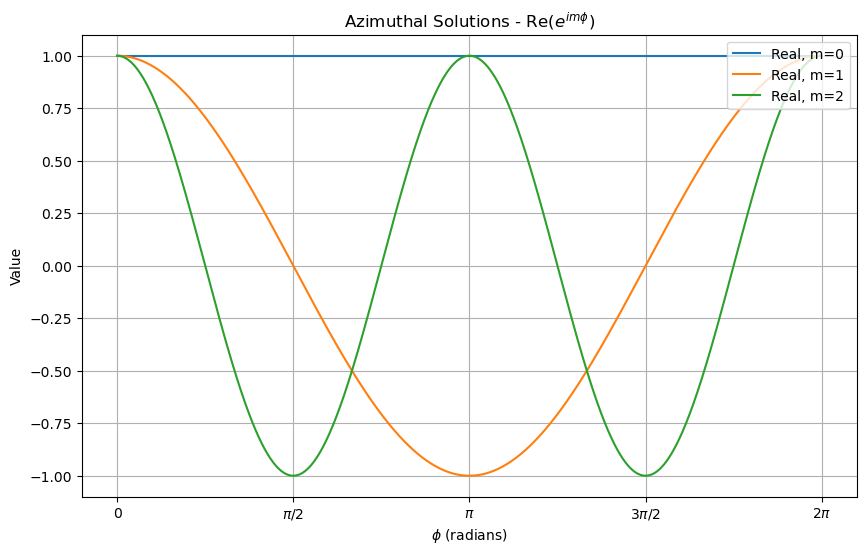


png

### The Azimuthal Angle Solutions

These solutions are the exponential functions, presented below.

# Define phi from 0 to 2\*pi  
phi = np.linspace(0, 2 \* np.pi, 1000)  
  
# Define a set of m values to plot  
m\_values = [0, 1, 2]  
  
plt.figure(figsize=(10,6))  
plt.xticks([0, np.pi/2, np.pi, 3\*np.pi/2, 2\*np.pi], ['0', r'$\pi/2$', r'$\pi$', r'$3\pi/2$', r'$2\pi$'])  
  
# Plot the real and imaginary parts of e^(im\*phi) for different m values  
for m in m\_values:  
 plt.plot(phi, np.cos(m \* phi), linestyle='-', label=f'Real, m={m}')  
  
plt.title('Azimuthal Solutions - Re($e^{im\phi}$)')  
plt.xlabel(r'$\phi$ (radians)')  
plt.ylabel('Value')  
plt.legend(loc='upper right')  
plt.grid(True)  
plt.show()



png

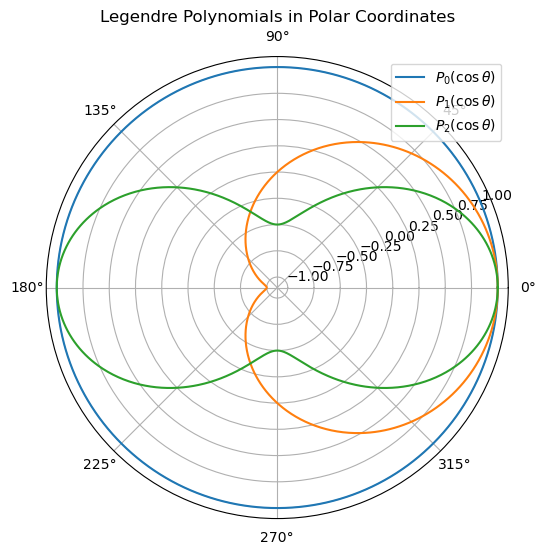
## Azimuthally Symmetric Solutions

In the case of azimuthally symmetric solutions, the potential is independent of and the solutions are of the form:

where are the [Legendre polynomials](https://en.wikipedia.org/wiki/Legendre_polynomials). The solutions are plotted below, but this time in 2D. You can imagine those solutions spun around the z-axis. We will exclusively use these solutions for the remainder of this activity because they make visualization easier.

We can see here why is the only solution for a perfectly spherical charge distribution. The other solutions are not spherically symmetric. It’s also possible to see where the dipole moment is in the solution; notice it’s spread out a bit.

from scipy.special import legendre  
  
theta = np.linspace(0, 2 \* np.pi, 1000)  
x = np.cos(theta)  
  
# Define a set of l values to plot  
l\_values = [0, 1, 2]  
  
plt.figure(figsize=(10,6))  
  
# Create a polar subplot  
ax = plt.subplot(111, projection='polar')  
  
for l in l\_values:  
 # Generate Legendre Polynomial  
 P\_l = legendre(l)  
 # Evaluate the polynomial at x = cos(theta)  
 y = P\_l(x)  
 ax.plot(theta, y, label=f'$P\_{l}(\cos \\theta)$')  
  
ax.set\_title('Legendre Polynomials in Polar Coordinates')  
ax.legend(loc='upper right')  
  
plt.show()



png

## Matching Boundary Conditions and Plotting the Potential

**✅ Do this**

### Sphere of constant surface potential

Consider a sphere of with a radius . If the potential on the surface is , what is the potential inside and outside the sphere?

1. Consider the radial solutions (what doesn’t blow up?)
2. Consider the polar angle solutions, what has to be true? What does that say about terms with ?
3. Write down the solution inside and outside the sphere.
4. Make a heat map plot (in and ) of the potential inside and outside the sphere. You can set and if that helps.

## your code here