Biot Savart Law

```
#| edit: false
#| echo: false
#| execute: true

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Set default plotting parameters
plt.rcParams.update({
    'font.size': 10,
    'lines.linewidth': 1,
    'lines.markersize': 5,
    'axes.labelsize': 11,
    'xtick.labelsize': 10,
    'ytick.labelsize': 10,
    'xtick.top': True,
    'xtick.direction': 'in',
    'ytick.right': True,
    'ytick.direction': 'in',
})

def get_size(w, h):
    return (w/2.54, h/2.54)
```

The law of Biot Savart allows the calculation of the magnetic field around arbitrary currents. Each element of a conductor at a point $\vec{r}'$ will contribute to the magnetic field at a point $\vec{r}$ an amount $\mathrm{d}\vec{B}$ according to

$$\mathrm{d}\vec{B}(\vec{r}) = \frac{\mu_0}{4\pi} I \, \mathrm{d}\vec{l} \times \frac{\vec{r} - \vec{r}'}{\left|\vec{r} - \vec{r}'\right|^3}$$

which will then lead to the total magnetic field one integrated over the whole volume.

$$\vec{B}(\vec{r}) = \frac{\mu_0}{4\pi} \int_V \vec{j}(\vec{r}\,') \times \frac{\vec{r} - \vec{r}\,'}{|\vec{r} - \vec{r}\,'|^3} \, \mathrm{d}V'$$

We would like to explore this relation by calculating the magnetic field around a circular loop of radius R centered at the origin.

Calculating the Magnetic Field of a Circular Current Loop

We'll use the Biot-Savart law to calculate the magnetic field of a circular current loop of radius R in the xy-plane, centered at the origin. We'll divide the loop into small segments and sum their individual contributions to find the total magnetic field.

```
#| autorun: false
# Physical constants and parameters
mu0 = 4 * np.pi * 1e-7  # Vacuum permeability in H/m
I = 1.0  # Current in amperes
R = 1.0  # Radius of the loop in meters

N_segments = 100

# Function to calculate magnetic field from a current loop using Biot-Savart law
def calculate_B_field(positions):

    # Ensure positions is properly shaped for calculation
    original_shape = positions.shape

    # Handle single position vector
    if len(original_shape) == 1:
        positions = positions.reshape(1, 3)
    else:
        # Reshape to 2D array (N_points, 3) for calculation
        positions = positions.reshape(-1, 3)

    # Parametrize the circular loop
    theta = np.linspace(0, 2*np.pi, N_segments, endpoint=False)
    dl_magnitude = 2*np.pi*R/N_segments  # Length of each segment

    # Create arrays for positions on the loop
    r_prime = np.zeros((N_segments, 3))
```

```
    r_prime[:, 0] = R * np.cos(theta)  # x-coordinates
    r_prime[:, 1] = R * np.sin(theta)  # y-coordinates

    # Create arrays for dl vectors
    dl = np.zeros((N_segments, 3))
    dl[:, 0] = -R * np.sin(theta) * dl_magnitude  # dx
    dl[:, 1] = R * np.cos(theta) * dl_magnitude   # dy

    # Prepare for broadcasting:
    # positions: (n_points, 1, 3)
    # r_prime: (1, N_segments, 3)
    positions_expanded = positions[:, np.newaxis, :]
    r_prime_expanded = r_prime[np.newaxis, :, :]
    dl_expanded = dl[np.newaxis, :, :]

    # Calculate r_diff for all positions and all segments
    r_diff = positions_expanded - r_prime_expanded  # shape: (n_points, N_segments, 3)

    # Calculate |r-r'| for all positions and segments
    r_magnitude = np.sqrt(np.sum(r_diff**2, axis=2))  # shape: (n_points, N_segments)

    # Avoid division by zero
    valid_mask = r_magnitude > 1e-10
    r_magnitude_safe = np.where(valid_mask, r_magnitude, np.inf)

    # Calculate scaling factor
    scaling = np.where(
        valid_mask,
        (mu0 * I / (4 * np.pi)) / r_magnitude_safe**3,
        0.0
    )  # shape: (n_points, N_segments)

    # Calculate cross products
    cross_products = np.cross(
        np.broadcast_to(dl_expanded, r_diff.shape),
        r_diff
    )  # shape: (n_points, N_segments, 3)

    # Apply scaling to cross products
    scaled_cross_products = cross_products * scaling[:, :, np.newaxis]

    # Sum over all segments to get the total field at each position
```

```
    B_field = np.sum(scaled_cross_products, axis=1)  # shape: (n_points, 3)

    # Reshape back to original dimensions
    if len(original_shape) == 1:
        return B_field[0]  # Return a single vector
    else:
        return B_field.reshape(original_shape[:-1] + (3,))

# Calculate field in the xz-plane (y=0)
grid_points = 20
x_range = np.linspace(-2*R, 2*R, grid_points)
z_range = np.linspace(-2*R, 2*R, grid_points)
X, Z = np.meshgrid(x_range, z_range)
Y = np.zeros_like(X)

# Create a grid of position vectors
positions = np.stack((X, Y, Z), axis=-1)

# Calculate the magnetic field at all grid points
B_vectors = calculate_B_field(positions)

# Extract components for plotting
BX = B_vectors[:, :, 0]
BY = B_vectors[:, :, 1]
BZ = B_vectors[:, :, 2]

# Calculate the magnitude of the magnetic field
B_mag = np.sqrt(BX**2 + BY**2 + BZ**2)
```

```
#| autorun: false

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

# Normalize the arrows for better visualization
arrow_scale = 5.0 / np.max(B_mag)
BX_norm = BX * arrow_scale
BZ_norm = BZ * arrow_scale

# Plot the magnetic field vectors
plt.quiver(X, Z, BX_norm, BZ_norm, B_mag,cmap="viridis")

# Plot the current loop (it appears as two points in the xz-plane)
```

```
plt.plot([-R, R], [0, 0], 'ro', markersize=8)


# Set labels and title
plt.xlabel('x (m)')
plt.ylabel('z (m)')
plt.axis('equal')

plt.tight_layout()
plt.show()
```

```
#| autorun: false

z_axis = np.linspace(-3*R, 3*R, 100)
Bz_along_z = np.zeros_like(z_axis)

for i, z in enumerate(z_axis):
    _, _, Bz_along_z[i] = calculate_B_field(0, 0, z)

# Create a figure for the field along the z-axis
plt.figure(figsize=get_size(10, 10))
plt.plot(z_axis, Bz_along_z, 'b-')
plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)
plt.xlabel(r'$z (m)$')
plt.ylabel(r'$B_z$ (T)')
plt.title('Magnetic Field along z-axis')

plt.tight_layout()
plt.show()
```

This simulation divides a circular current loop into small segments and applies the Biot-Savart law to each segment. The first plot shows the magnetic field vectors in the xz-plane, while the second plot shows how the z-component of the field varies along the z-axis.

For the field along the z-axis, the numerical results can be compared with the analytical formula:

$$B_z = \frac{\mu_0 I R^2}{2(R^2 + z^2)^{3/2}}$$

The characteristic “dipole” pattern is clearly visible in the vector plot, with field lines emerging from the center of the loop and curving around to form closed loops.