Consolidated NumPy & Matplotlib Practice Exercises

I. Fundamental Array Mechanics (NumPy Basics)

These exercises focus on the essential creation, properties, indexing, and manipulation skills.

1. **Array Creation and Inspection:**
   - Create a **1D array** of integers from 0 to 9. Print its shape, ndim, and dtype.
   - Create a **2D array** (3x5) filled with random floating-point numbers between 0 and 1.
   - Create a **5x5 identity matrix** using np.eye().
   - Create an array of **20 equally spaced numbers** between 5 and 50 using np.linspace.
2. **Indexing, Slicing, and Selection:**
   - Given a 5x5 array A, extract the **3rd row** and the **2nd column**.
   - Create a 1D array of 10 random integers. Use **Boolean indexing** to select and print only the elements that are greater than 7.
   - Create a 5x5 array. Use **Fancy indexing** to select rows 0, 2, and 4.
3. **Manipulation and Broadcasting:**
   - Take a 1D array of 12 elements and **reshape** it into a 4x3 array.
   - Create two 4-element 1D arrays, A and B. **Stack them** vertically (np.vstack) and horizontally (np.hstack).
   - Create a 3x3 array M. Add a 1D array V of shape (3,) to every **row of M** using **broadcasting**.

II. Data Fitting and Visualization (Least Squares)

This exercise combines data generation, curve fitting, and comparative plotting.

1. **Generate Noisy Data:**
   - Create a 1D array of 50 equally spaced *x*-values between 0 and 10.
   - Calculate the theoretical *y*-array: y = 3x + 5.
   - Add **Gaussian noise** (mean $\mu=0$, standard deviation $\sigma=2$) to the *y*-values using np.random.normal() to create y_{noisy}.
2. **Perform Fit:**
   - Use **np.polyfit()** to perform a linear least-squares fit (degree 1) on the $(\text{x}, \text{y}_{\text{noisy}})$ data to find the fitted slope (m_{fit}) and intercept (c_{fit}).
3. **Visualize:**
   - Create a single **Matplotlib** plot:
     - Plot the y_{noisy} data as **scatter points**.
     - Plot the y_{fit} line (calculated with m_{fit} and c_{fit}) as a **solid line**.

        - Plot the **original theoretical line** ($y = 3x + 5$) as a **dashed line**.
    - Include a **legend** and display the fitted parameters in the title.

III. Spectral Analysis and Subplots

Focuses on the transformation and visualization of time-series data.

1. **Create Complex Signal:**
    - Generate a time array t (1000 points over 2 seconds).
    - Create the signal $f(t)$ by summing two sine waves: $f(t) = \sin(2\pi \cdot 10t) + 0.5 \cdot \sin(2\pi \cdot 50t)$.
2. **Calculate Spectrum:**
    - Apply the **FFT** using **np.fft.fft()**.
    - Calculate the corresponding frequency array using **np.fft.fftfreq()** and shift it using **np.fft.fftshift()**.
    - Calculate the **power spectrum** (magnitude squared).
3. **Visualize:**
    - Use **Matplotlib** to create **two subplots** stacked vertically:
        - **Top Subplot:** Plot the original signal $f(t)$ vs. time t.
        - **Bottom Subplot:** Plot the power spectrum vs. the shifted frequency array, clearly showing the two expected frequency peaks.

IV. Multidimensional Visualization and Linear Algebra

These exercises cover crucial skills for numerical physics and computational modeling.

1. **Solving Linear Systems:**
    - Create a random 4×4 coefficient matrix **A** and a 4×1 result vector **b**.
    - Solve the system $A\textbf{x} = \textbf{b}$ for **$\textbf{x}$** using **np.linalg.solve()**.
    - Calculate the **residual** $\textbf{r} = A\textbf{x} - \textbf{b}$ to verify the solution (should be close to zero).
    - Visualize the four components of the solution vector $\textbf{x}$ using a **Matplotlib bar chart**.
2. **2D Potential Field Visualization:**
    - Use **np.linspace** and **np.meshgrid()** to create 2D coordinate arrays X and Y for the domain $[-5, 5] \times [-5, 5]$.
    - Calculate the 2D Gaussian function: $Z = \exp(-(X^2 + Y^2) / 2)$.
    - Use **plt.imshow()** to create a **heatmap** of Z. Ensure the x and y axes are correctly labeled using the extent parameter.
    - Add a **colorbar**.
    - *Bonus:* Create a **contour plot** (plt.contourf()) of the same data Z.

V. Practical Data Preprocessing (Normalization & Missing Data)

Simulates common steps required before feeding data into models.

1. **Data Normalization:**
   - Create a 1D array X of 10 random values.
   - **Normalize** the array X to the range $[0, 1]$ using the formula $\text{X}_{norm} = \frac{\text{X} - \text{X}_{\text{min}}}{\text{X}_{\text{max}} - \text{X}_{\text{min}}}$.
2. **Handling Missing Data:**
   - Create a 5×5 array and manually insert three **np.nan** values.
   - Write code to **count** the number of missing values.
   - **Impute** the missing values by replacing them with the **mean** of the non-missing values in the array.

Would you like the Python code solution for the **Data Fitting and Visualization** exercise (Section II) to get started?