1. **Matrix Operations and Properties**
   * **Objective**: Implement basic matrix operations (addition, multiplication, transpose).
   * **Task**: Write functions for matrix addition, matrix multiplication, and transposition in a programming language like Python. Test your functions on randomly generated matrices.
2. **Solving Linear Systems**
   * **Objective**: Solve systems of linear equations using Gaussian elimination.
   * **Task**: Implement Gaussian elimination and back-substitution to solve a system of linear equations. Compare your solution with built-in functions (e.g., numpy.linalg.solve in Python).
3. **Numerical Differentiation**
   * **Objective**: Understand numerical methods for approximating derivatives.
   * **Task**: Use finite difference approximations (forward, backward, central differences) to estimate the derivative of a given function. Compare the numerical results with the analytical derivative.
4. **Root-Finding with Bisection Method**
   * **Objective**: Implement and apply the bisection method for solving nonlinear equations.
   * **Task**: Write a program to find the root of a non-linear function using the bisection method. Test it on simple functions like f(x)=x2−2f(x) = x^2 - 2f(x)=x2−2.

**Intermediate Level**

1. **Polynomial Interpolation (Lagrange and Newton)**
   * **Objective**: Explore polynomial interpolation techniques.
   * **Task**: Implement the Lagrange and Newton interpolation methods to interpolate a set of given data points. Compare the performance and accuracy of both methods.
2. **Numerical Integration (Trapezoidal and Simpson’s Rule)**
   * **Objective**: Apply numerical integration methods.
   * **Task**: Implement the trapezoidal rule and Simpson’s rule for approximating definite integrals. Test the methods on well-known functions, like ∫01exdx\int\_0^1 e^x dx∫01​exdx.
3. **Jacobi and Gauss-Seidel Iterative Methods**
   * **Objective**: Solve linear systems using iterative methods.
   * **Task**: Implement both Jacobi and Gauss-Seidel methods for solving systems of linear equations. Compare the convergence rates for different initial guesses and matrix types.
4. **Eigenvalue Calculation (Power Method)**
   * **Objective**: Approximate dominant eigenvalues of a matrix.
   * **Task**: Implement the Power Method to find the dominant eigenvalue of a matrix. Test it on matrices with known eigenvalues for verification.

**Advanced Level**

1. **Numerical Solutions to ODEs (Euler and Runge-Kutta Methods)**
   * **Objective**: Solve ordinary differential equations using numerical techniques.
   * **Task**: Implement Euler’s method and the fourth-order Runge-Kutta method to solve a first-order ODE. Compare the accuracy of both methods for different step sizes.
2. **LU Decomposition and Solving Linear Systems**
   * **Objective**: Understand and implement LU decomposition.
   * **Task**: Write a program to perform LU decomposition of a matrix and use it to solve linear systems. Compare it with direct methods like Gaussian elimination in terms of computational efficiency.
3. **Optimization using Gradient Descent**
   * **Objective**: Apply gradient descent to minimize functions.
   * **Task**: Implement gradient descent to find the local minima of functions (e.g., quadratic functions or multivariable functions). Analyze the effect of the learning rate on convergence.
4. **Finite Element Method for 1D PDEs**
   * **Objective**: Apply the finite element method (FEM) to solve partial differential equations.
   * **Task**: Implement FEM for solving a simple 1D PDE (e.g., heat equation or Poisson’s equation). Analyze the accuracy by comparing the numerical solution with the exact solution.

**Expert Level**

1. **Sparse Matrix Computations**
   * **Objective**: Work with large sparse matrices.
   * **Task**: Implement efficient matrix operations (e.g., multiplication, LU decomposition) for large sparse matrices using appropriate data structures. Test the efficiency against standard dense matrix methods.
2. **Fast Fourier Transform (FFT)**
   * **Objective**: Implement and apply the FFT algorithm.
   * **Task**: Implement the Cooley-Tukey Fast Fourier Transform algorithm. Apply it to a signal (e.g., sine wave) and analyze the frequency components.
3. **Multigrid Methods for Solving PDEs**
   * **Objective**: Solve large-scale PDEs efficiently.
   * **Task**: Implement a multigrid method for solving a simple 2D PDE (e.g., Laplace's equation). Compare the computational efficiency and accuracy with classical iterative methods.
4. **Monte Carlo Simulation for Numerical Integration**
   * **Objective**: Use stochastic methods to approximate integrals.
   * **Task**: Implement a Monte Carlo simulation to estimate the value of a multi-dimensional integral. Compare the results with deterministic methods like trapezoidal or Simpson's rule.