COURSE CONTENT

Computational Mathematics Lab (Using Python software)								
II year/ I Semester: CE/CSD/CSM/EEE/ME								
II year/ II Semester: CSE/ECE/EEE								
CourseCode	**Category**	**Hours/ Week**			**Credits**	**MaximumMarks**		
2530075	**Basic sciences**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	2	1	40	60	100
ContactClasses:Nil	**TutorialClasses:Nil**	**PracticalClasses:32**				**TotalClasses:32**		
Prerequisites: Matrices, Iterative methods and ordinary differential equations								

Course Overview:

This course introduces the mathematical foundations of numerical methods for solving systems of equations, root-finding problems, eigenvalue analysis, and differential equations. It emphasizes the development and implementation of standard numerical algorithms such as LU decomposition, Bisection, and Newton-Raphson methods using Python. Learners will model and analyze real-world scientific and engineering phenomena through exponential and differential equations.

Course Objectives:

1. Understand the mathematical foundations of numerical methods used for solving systems of equations, root-finding, and differential equations. Also, apply computational tools (Python and relevant libraries) to model, analyze, and solve scientific and engineering problems.
2. Develop programs that implement standard numerical algorithms such as LU decomposition, Bisection, and Newton–Raphson methods.
3. Analyze and interpret eigenvalues and eigenvector computations for system stability and physical modeling.
4. Model real-world phenomena (e.g., population growth, drug decay, cooling, and radioactive decay) using exponential and differential equations.
5. Formulate, verify, and solve both exact and non-exact differential equations symbolically and numerically using Python. Methods of solving the differential equations of first order and first degree.

Course Outcomes: After Completion the experiments, Students should be able to

1. Solve non-homogeneous linear systems using LU decomposition and interpret the computational results.
2. Compute real and complex eigenvalues and eigenvectors of matrices using Python.
3. Apply Bisection and Newton–Raphson methods to determine roots of nonlinear equations with specified accuracy.
4. Identify, verify, and solve exact and non-exact differential equations symbolically and numerically. Also, implement homogeneous and non-homogeneous linear ordinary differential equations using analytical and numerical techniques.
5. Model and simulate natural processes such as population growth, drug decay, and radioactive half-life using exponential functions and estimate parameters like the cooling constant in Newton's law of cooling and predict system behavior over time.

List of Experiments:

1. Solve a non-homogeneous linear system using LU decomposition in Python.
   Input: matrix A and right-hand side vector b.

Output: solution vector x, and the L and U factors (with pivoting if needed).

2. Write a general Python program that accepts a square matrix from the user and returns its eigenvalues (including complex values when they occur).
   Input: square matrix.
   Output: list of eigenvalues (real and complex).
3. Implement computational tools in Python to compute eigenvectors for a given square matrix.
   Input: square matrix (and optionally selected eigenvalues).
   Output: corresponding eigenvectors (normalized or not, as specified).
4. Implement the Bisection method in Python to find a root of any continuous function on a given interval.
   Input: function f, interval [a, b] with $f(a)f(b) < 0$, tolerance, and max iterations.
   Output: approximate root and iteration info.
5. Implement the Newton–Raphson method in Python to find a root of any differentiable function.
   Input: function f, its derivative f', initial guess x0, tolerance, and max iterations.
   Output: approximate root and iteration info.
6. Detect and solve exact differential equations in Python.
   Input: first-order differential equation $M(x,y) + N(x,y)\, y' = 0$.
   Output: determine if the equation is exact; if so, provide the general solution (implicit or explicit).
7. Solve non-exact first-order differential equations in Python by finding and applying an integrating factor when possible.
   Input: $M(x,y) + N(x,y)\, y' = 0$.
   Output: integrating factor (if found) and the general solution.
8. Write a Python program that models exponential processes, such as population growth, drug decay in blood, or radioactive decay.
   Input: model parameters (initial value, rate or half-life, time span).
   Output: time series (table or plot) and parameter estimates where applicable.
9. Compute the cooling constant k from two temperature measurements and predict future temperature using Newton's law of cooling in Python.
   Input: ambient temperature, two measurements
   Output: estimated k and predicted temperature for requested times.
10. Derive and solve homogeneous linear ordinary differential equations in Python.
    Input: linear ODE with constant coefficients (specify order).
    Output: general solution (analytic where possible) and particular initial/boundary value solutions.
11. Solve non-homogeneous linear ordinary differential equations with constant coefficients in Python.
    Input: Linear ODE with constant coefficients and a non-homogeneous term, including initial or boundary conditions if provided.
    Output: general solution and particular solution; numeric solution if analytic form is not available.
12. Solve non-homogeneous linear ordinary differential equations with variable coefficients in Python.
    Input: Linear ODE with variable coefficients and a non-homogeneous term, including initial or boundary conditions if provided.
    Output: general solution and particular solution; numeric solution if analytic form is not available.

Open Ended Experiments:

1. Solve a System of Linear Equations using the Gauss–Jordan Elimination Method in Python
   Input: Coefficient matrix $A$and right-hand side vector b.
   Output: Solution vector x; reduced row echelon form of the augmented matrix; verification of results using built-in functions (if desired).
2. Solve a System of Linear Equations using the Gauss–Seidel Iterative Method in Python

Input: Coefficient matrix A, right-hand side vector b, initial guess x_0, tolerance, and maximum number of iterations.
Output: Approximate solution vector $x$and number of iterations required for convergence.

TEXTBOOKS:

1. Kenneth A. Lambert, The fundamentals of Python: First Programs, 2011, Cengage Learnings.
2. Think Python First Edition, by Allen B. Downey, Orielly publishing.
3. IntroductiontoPythonProgramming,WilliamMitchell,PovelSolin,MartinNovaketal.,NC Lab Public Computing, 2012.
4. IntroductiontoPythonProgramming,©JacobFredslund,2007.

REFERENCEBOOKS:

1. An Introduction to Python, John C. Lusth, The UniversityofAlabama,2011. IntroductiontoPython,©DaveKuhlman,2008.

ELECTRONIC RESOURCES:

1. **https://www.youtube.com/watch?v=wtUk7CqbAt4**
2. **https://www.youtube.com/watch?v=OZ0JM9RAa00**
3. **https://www.youtube.com/watch?v=vTUPQq2mdbY**
4. **https://www.youtube.com/watch?v=oVLhKP_JfnE&t=2s**
5. **https://www.youtube.com/watch?v=fOdM9HKRtbs&t=122s**

MATERIALS ONLINE:

1. Course template
2. Lab Manual
3. Open-ended experiments
4. e-Learning Readiness Videos(ELRV)