National University of Computer and Emerging Sciences

Islamabad Campus

Numerical Computing (CS-2008)

Date: Dec 17, 2024

Course Instructors

Dr.-Ing. Mukhtar Ullah, Dr. Ir. Imran Ashraf,

Dr. Mir Suleman Sarwar, M. Almas

Final Exam

Total Time (Hrs): 3

Total Marks: 100

Total Questions: 6

Roll No ____________ Section ____________ Student Signature ____________________

Attempt all the questions
Use answer sheet to answer all questions
Answer MCQs on the bubble sheet
Clearly present all formulas and calculation steps
Bonus 2 marks by solving questions and their parts in sequential order
Attach the bubble sheet to your answer sheet before submission
DO NOT WRITE BELOW THIS LINE

CLO # 1 & 6

Question # 1 [6+6+ 6 = 18 Marks]

a. The implementation of the function $f(x) = \frac{1-\cos(x)}{x^2}$ for $x = 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}, 1 \times 10^{-8}, 1 \times 10^{-9}, 1 \times 10^{-10}, 1 \times 10^{-11}$, results in catastrophic cancellation. Express the function into a form that is stable and has an implementation that does not lead to catastrophic cancellation. *(Algorithmic stability)*

(a) Solution

The given function is:

$$f(x) = \frac{1-\cos(x)}{x^2}$$

When x is small, direct computation can lead to catastrophic cancellation due to the subtraction of nearly equal terms. To stabilize the function, we multiply both the numerator and the denominator by $1 + \cos(x)$:

$$f(x) = \frac{(1-\cos(x))(1+\cos(x))}{x^2(1+\cos(x))}$$

Using the trigonometric identity $(1-\cos(x))(1+\cos(x)) = \sin^2(x)$, we get:

$$f(x) = \frac{\sin^2(x)}{x^2(1+\cos(x))}$$

Simplifying further:

$$f(x) = \left(\frac{\sin(x)}{x}\right)^2 \frac{1}{1+\cos(x)}$$

This form avoids catastrophic cancellation and ensures numerical stability, especially for small values of x.

b. Approximate the root of the function $f(x) = \ln(x) + x$, using Secant's Method, for $\text{tol} = 1 \times 10^{-4}$, between $x_1 = 1.0$ and $x_2 = 2.0$.

(b) Solution

Given initial guesses $x_1 = 1.0$ and $x_2 = 2.0$.
The Secant Method formula is given by:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{1}$$

We start with initial guesses: $x_1 = 1.0$ and $x_2 = 2.0$.

The function is defined as $f(x) = \ln(x) + x$.

Calculate $f(x_1)$ and $f(x_2)$:

$$f(x_1) = \ln(1.0) + 1.0 = 1.0$$
$$f(x_2) = \ln(2.0) + 2.0 = 2.6931$$

The remaining iterations are calculated as follows:

Iteration (n)	x_n	$f(x_n)$	Error ($\lvert x_{n+1} - x_n \rvert$)
0	1.0000	1.0000	-
1	2.0000	2.6931	-
2	0.4094	-0.4837	1.5906
3	0.6516	0.2232	0.2422
4	0.5751	0.0219	0.0765
5	0.5668	-0.0010	0.0083
6	0.5671	0.000004	0.00036
7	0.5671	0	0.000002

Thus, the approximate root of the function is $x \approx 0.5671$, with the error being sufficiently small to meet the required tolerance.

c. Write down code to find the root of the function $f(x) = \ln(x) + x$, using Secant's Method, for $\text{tol} = 1 \times 10^{-4}$, $x_1 = 1.0$, $x_2 = 2.0$.

(c) Python code

```python
from scipy.optimize import newton
import numpy as np

# Define the function
def f(x):
    return np.log(x) + x

# Initial guesses
x1 = 1.0
x2 = 2.0
tol = 1e-4

# Solve for the root using the secant method
root = newton(f, x0=x1, x1=x2, tol=tol)
```

```python

# Print the result
print(f"The root of the function is approximately: {root:.6f}")
```

CLO # 2

Question # 2 [3 + 3 + 1 + 2 = 9 Marks]

If we are given three points (x_0, y_0), (x_1, y_1), (x_2, y_2), we can construct a quadratic polynomial and write the following equations:

$$\begin{cases} y_0 = a_0 + a_1 x_0 + a_2 x_0^2 \\ y_1 = a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 = a_0 + a_1 x_2 + a_2 x_2^2 \end{cases}$$

a. Write this system of equations in the matrix form.

b. Write the Vandermonde matrix for this system.

c. What is an advantage of using Vandermonde matrix?

d. What are the two disadvantages of using Vandermonde matrix?

Solution a)

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

Solution b)

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix}$$

Solution c) Simple and straightforward to use.

Solution d)

- Makes a dense matrix, which is slow to solve.
- The matrix is ill-conditioned and the calculations are sensitive to floating-point errors.

CLO # 3

Question # 3 [8 Marks]

The magnitude of the force $f(x_i)$ measured in Newtons at the locations x_i, $i = 0, 1, \dots, 15$ measured in meters is given below:

x_i	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4
$f(x_i)$	0.0	0.45	1.45	2.3	3.1	3.1	3.1	2.5	1.1	1.1	1.1	0.8	0.6	0.3	0.0

Write a Python program to estimate the work done (that is, an integral of this force over displacement). You are supposed to use the module `scipy.integrate` for this program.

Solution

Python code

```
from scipy.integrate import trapezoid
import numpy as np

x = np.arange(0.0, 1.5, 0.1)
y = np.array([0.0, 0.45, 1.45, 2.3, 3.1, 3.1, 3.1,
              2.5, 1.1, 1.1, 1.1, 0.8, 0.6, 0.3, 0.0])

print(trapezoid(y, x))
```

CLO # 4

Question # 4 [3+3+3+3+4+4= 20 Marks]

Consider the system of equations $Ax = b$ with matrix A and b given.

$$A = \begin{bmatrix} 11 & 2 \\ 2 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 9 \\ 4 \end{bmatrix}$$

a. Solve above given system of equation via Cholesky factorization.

a) Solution:

$$A = \begin{bmatrix} 11 & 2 \\ 2 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 9 \\ 4 \end{bmatrix}.$$

Using Cholesky factorization, $A = HH^T$, we find:

$$H = \begin{bmatrix} h_{11} & 0 \\ h_{21} & h_{22} \end{bmatrix}, \quad H^T = \begin{bmatrix} h_{11} & h_{21} \\ 0 & h_{22} \end{bmatrix}.$$

After computation:

$$H = \begin{bmatrix} \sqrt{11} & 0 \\ \frac{2}{\sqrt{11}} & \sqrt{5 - \frac{4}{11}} \end{bmatrix}, \quad H^T = \begin{bmatrix} \sqrt{11} & \frac{2}{\sqrt{11}} \\ 0 & \sqrt{5 - \frac{4}{11}} \end{bmatrix}.$$

Forward substitution: Solve $Hy = b$:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad y_1 = \frac{9}{\sqrt{11}}, \quad y_2 = \frac{4 - \frac{2}{\sqrt{11}} \cdot 9}{\sqrt{5 - \frac{4}{11}}}.$$

Backward substitution: Solve $H^T x = y$:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad x_2 = \frac{y_2}{\sqrt{5 - \frac{4}{11}}}, \quad x_1 = \frac{y_1 - \frac{2}{\sqrt{11}} x_2}{\sqrt{11}}.$$

b. Solve above given system of equation via LU factorization.

b) Solution

Using LU factorization, $A = LU$, we find:

$$L = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}.$$

After computation:

$$L = \begin{bmatrix} 1 & 0 \\ \frac{2}{11} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 11 & 2 \\ 0 & 5 - \frac{4}{11} \end{bmatrix}.$$

Forward substitution: Solve $Ly = b$:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad y_1 = 9, \quad y_2 = 4 - \frac{2}{11} \cdot 9.$$

Backward substitution: Solve $Ux = y$:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad x_2 = \frac{y_2}{5 - \frac{4}{11}}, \quad x_1 = \frac{y_1 - 2x_2}{11}.$$

c. Approximate x_1,x_2 using the Jacobi method for two iterations using initial guess $\begin{bmatrix} 1, 1 \end{bmatrix}$.

c) Solution:

$$A = \begin{bmatrix} 11 & 2 \\ 2 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 9 \\ 4 \end{bmatrix}.$$

The Jacobi iteration formula is:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right).$$

With the initial guess $x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, the iterations proceed as follows:

Iteration 1:

$$x_1^{(1)} = \frac{1}{11} (9 - 2 \cdot 1) = \frac{7}{11}, \quad x_2^{(1)} = \frac{1}{5} (4 - 2 \cdot 1) = \frac{2}{5}.$$

Iteration 2:

$$x_1^{(2)} = \frac{1}{11} \left(9 - 2 \cdot \frac{2}{5} \right) = \frac{43}{55}, \quad x_2^{(2)} = \frac{1}{5} \left(4 - 2 \cdot \frac{7}{11} \right) = \frac{26}{55}.$$

d. Approximate x_1,x_2 using the Gauss-Seidel method for two iterations using initial guess $\begin{bmatrix} 1, 1 \end{bmatrix}$.

d) Solution:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right).$$

With the initial guess $x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, the iterations proceed as follows:

Iteration 1:

$$x_1^{(1)} = \frac{1}{11}(9 - 2 \cdot 1) = \frac{7}{11}, \quad x_2^{(1)} = \frac{1}{5}\left(4 - 2 \cdot \frac{7}{11}\right) = \frac{13}{55}.$$

Iteration 2:

$$x_1^{(2)} = \frac{1}{11}\left(9 - 2 \cdot \frac{13}{55}\right) = \frac{452}{605}, \quad x_2^{(2)} = \frac{1}{5}\left(4 - 2 \cdot \frac{452}{605}\right) = \frac{151}{605}.$$

Final Note: Both methods approximate the solution iteratively, with the Gauss-Seidel method often converging faster due to immediate updates of intermediate results.

e. For 10,000 moderate-sized linear systems $Ax = b$, having non-symmetric A (same A in all systems, varying b), from (a), (b), (c), (d) above suggest optimal method (Method name + one-line justification).

e) Answer

For solving 10,000 moderate-sized linear systems $Ax = b$, with the same non-symmetric A and varying b, the optimal method is:

Method: LU Factorization

Justification: LU factorization efficiently reuses the decomposition $A = LU$ across all systems, requiring only forward and backward substitution for each new b, minimizing computational cost.

f. Review the following code. Identify the exact incorrect code and propose corrections.

Figure 1: Jacobi Method

```python
def jacobi(A, b, x, tol = 1.e-5, maxit = 100):
    d = np.copy(np.diag(A))
    np.fill_diagonal(A, 0.0)
    err = 1.0
    iters = 0
    while (err < tol and iters > maxit):
        iters += 1
        xnew = (x + np.dot(A, b)) / d
        err = np.linalg.norm(xnew-x, np.inf)
        x = np.copy(xnew)
        print('iterations :', iters)
    return x
```

Correction in above code

```python

while (err > tol and iters < maxit):
xnew = (b - np.dot(A, x)) / d
```

Figure 2: Gauss-Seidel Method

```python
def gauss_seidel(A, b, x, tol = 1.e-5, maxit = 100):
    n = len(b)
    err = 1.0
    iters = 0
    # Initialize the solution with the initial guess
    # xnew = np.zeros_like(x)
    # Extract the lower triangular part of A
    M = np.tril(A)
    # Construct the upper triangular part of A
    U = A - M
    while (err < tol and iters > maxit):
        iters += 1
        # Compute the new approximation
        xnew = np.dot(np.linalg.inv(M), b + np.dot(U, x))
        # Estimate convergence
        err = np.linalg.norm(xnew-x, np.inf)
        x = np.copy(xnew)
    print('iterations required for convergence:', iters)
    return x
```

Correction in above code

```python

while ( err > tol and iters < maxit ) :
xnew = np . dot ( np . linalg . inv ( M ) , b - np . dot (U , x ) )
```

CLO # 4 & 5

Question # 5 [Marks = 15]

Assume that the following functions have been provided to you.

The function `UTS()` takes:

- an upper triangular matrix **A** as the first input
- a column matrix **b** as the second input

and returns:

- a solution using backward substitution to the linear system associated with matrices (**A**,**b**) as the output

The function `LLS()` takes:

- a matrix **A** as the first input
- a column matrix b as the second input

and returns:

- a solution to the linear least squares problem associated with matrices (A,b) as the first output
- the minimized error in the linear least squares problem as the second output

The function `GMST()` takes:

- a matrix A as the input

and returns:

- a matrix with orthonormal columns generated from columns of A as the output

You are given a linear system $\mathbf{Ax} = \mathbf{b}$ with:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

Questions:

a. Write Python code using only (one or both of) the two functions listed above to compute $\mathbf{A}^{\dagger}\mathbf{b}$.

b. Write Python code using only (one or both of) the functions listed above, together with the `numpy.inner`, to compute matrices $\mathbf{Q}$ and $\mathbf{R}$ of the QR factorization of $\mathbf{A}$.

c. Write Python code using the computed matrices $\mathbf{Q}$ and $\mathbf{R}$ to compute the solution to the least square problem associated with the given linear system.

Solution

Given the linear system and The provided functions are:

- `UTS()` Solves an upper triangular system using backward substitution.
- `LLS()` Solves the linear least squares problem.
- `GMST()` Computes the orthonormal matrix Q from A.

(a) Solution

To compute $A^T b$, we will use the `LLS()` function, which returns the solution to the linear least squares problem for A and b.

The Python code for this is:

```
Apb = LLS(A,b)
```

(b) Solution

To compute the QR factorization, we use the `GMST()` function to calculate the orthonormal matrix Q. Then, R can be computed as $R = Q^T A$.

The Python code for this is:

```
Q = GMST(A)
R = np.inner(Q,A)
```

(c) Solution

Now, solve the system $Rx = Q^T b$ using the `UTS()` function, which solves upper triangular systems.

The Python code for this is:

```
Qtrp = np. transpose(Q)
y = np.dot(Qtrp,b)
x = UTS(R,y)
```

CLO # 1,3,4,5,6

Question # 6 [Marks = 30]

1. `my_name = 'James'; print(my_name[3:4])` will print:
   a. James
   b. Jame
   c. e
   d. es
2. `c=np.array([2,4,3]); A=circulant(c); print(A)` will print:
   a [3, 4, 2]
   b [[2, 4, 3] [3, 2, 4] [4, 3, 2]]
   c [234]
   d [[2, 3, 4] [4, 2, 3] [3, 4, 2]]
3. `A = np.array([2,-1,0],[-1,2,-1],[0,-1,2]); print(det(A))` will print:
   a. 65
   b. 4
   c. 64
   d. None of these
4. If `x=3.141596`, and `t=5` then chopping gives:
   a. 3.1415
   b. 3.1416
   c. 3
   d. 3.5
5. ______________is the phenomenon where the subtraction of two almost equal numbers leads to a large error, due to finite precision arithmetic.
   a. Catastrophic cancellation
   b. Relative error
   c. Chopping
   d. Absolute error
6. In ___________ the endpoints are not included in an interval such that $a < x < b$.
   a. Open interval (a, b)
   b. Closed interval $[a, b]$
   c. Open interval $[a, b]$
   d. Closed interval (a, b)
7. The first term in the Taylor expansion for the function $f(x) = \cos(x)$ at $x = \pi/2$ is:
   a. 1
   b. 2

c. 0
d. 4

8. In the interval $[1, 2]$ for any function, using Bolzanos theorem, which of the following cannot be a root?
   a. 1.5
   b. 2.5
   c. 1.75
   d. 1.25

9. When you want to estimate the value of a function at a point between two known values, which of the following methods is most commonly used?
   (a) Interpolation
   (b) Matrix Multiplication
   (c) Eigenvalue Decomposition
   (d) Fourier Series Expansion

10. What is the main benefit of using cubic spline interpolation over linear interpolation?
    (a) Cubic Spline capture smoothness better than Linear interpolation
    (b) Both are best option for approximating the rate of change
    (c) Both are approximating integral value
    (d) Both are same

[From 11^{th} to 15^{th} MCQs only] Consider the data for the following questions. Choose the right option.

X	Y
5	4
7	6

11. What will be the coefficient a_0 using Newton divided and difference method?
    (a) 4
    (b) 1
    (c) 0
    (d) None of a, b, c

12. What will be the coefficient a_1 using Newton divided and difference method?
    (a) 4
    (b) 1
    (c) 0
    (d) None of a, b, c

13. Using Lagrange interpolation $P_1(6) = \sum_{i=0}^{N} y_i \ell_i(6)$ in the case of the above data, what will be the value of $\ell_0(x)$ if $x = 6$?
    (a) 0.5
    (b) 0.254

(c) 0.75
(d) None of a, b, c

14. On comparison of $\ell_1(6)$ for $x = 6$ with the above a_1 you computed in question 12.
   (a) a_1 equal to $\ell_1(x)$
   (b) a_1 less than $\ell_1(x)$
   (c) a_1 greater than $\ell_1(x)$
   (d) None of a, b, c

15. What will be the $p(6)$ by using $P_1(6) = \sum_{i=0}^{N} y_i\ell_i(6)$?
   (a) 5.1
   (b) 5
   (c) 5.2
   (d) None of A, B, C

16. What does the trapezoidal rule use to approximate the area under a curve?
   (a) Trapezoids
   (b) Squares
   (c) Circles
   (d) Triangles

17. Which method gives a more accurate result for smooth functions in numerical integration?
   (a) Simpson's Rule
   (b) Rectangle Rule
   (c) Midpoint Rule
   (d) Trapezoidal Rule

18. The trapezoidal rule is used for:
   (a) Numerical integration
   (b) Solving linear equations
   (c) Solving differential equations
   (d) Polynomial fitting

19. What does the forward difference method estimate?
   (a) The first derivative of a function
   (b) approximate root
   (c) The integral of a function
   (d) The value of a function at a new point

20. Why is the central difference method more accurate for derivatives?
   (a) It uses both the forward and backward points
   (b) It uses only the forward points
   (c) It uses only the backward points
   (d) It is not used for derivatives

21. The linear least squares problem associated with the linear system is the problem of finding:
    (a) a solution x^* that minimizes the error $\|Ax - b\|_2^2$
    (b) the minimum value of the error $\|Ax - b\|_2^2$
    (c) the exact solution x^* that satisfies $Ax^* = b$
    (d) a solution x^* such that error $\|b - Ax\|_2$ is a linear function.

22. The pseudoinverse A^+ is useful in solving:
    a. the linear system $Ax = b$ because A^+ approximates x within floating-point error.
    b. the linear system $Ax = b$ if A is invertible.
    c. the linear system $Ax = b$ if $A^T A$ is invertible.
    d. the linear system $Ax = b$ if the augmented matrix $[A|b]$ can be row-reduced.

23. When it exists, the solution x to the linear least square problem associated with the linear system $Ax = b$ can be expressed as:
    a. $x = (A^T A)^{-1} A^T b$
    b. $x = A^T (AA^T)^{-1} b$
    c. $x = A^T A (A^T)^{-1} b$
    d. $x = A^{-1} A^T b$

24. To verify that the vectors $q_1, q_2, \ldots, q_n$ have been correctly generated by Gram-Schmidt orthonormalization, we must ensure that:
    a. $q_i \cdot q_j = 0$ for $i \neq j$
    b. $\|q_i\| = 1$
    c. $A = QR$
    d. $A^T A = QQ^T$

25. LU factorization can be applied to which type of matrix?
    a. Only column matrices
    b. Only row matrices
    c. Only diagonal matrices
    d. Non of above a,b,c

26. Cholesky decomposition can be applied to which type of matrix?
    a. Any square matrix
    b. Any skew symmetric matrix
    c. Any symmetric positive-definite matrix
    d. Any diagonal matrix

27. What is the main advantage of using Cholesky decomposition for solving a system of linear equations $Ax = b$?
    a. It provides an approximation to the solution
    b. It is faster and more numerically stable for symmetric positive-definite matrices compared to other methods
    c. It guarantees a unique solution for any matrix

d. It can be applied to non-square matrices

28. The Jacobi method for solving a system of linear equations converges if which of the following conditions is met?
    a. The matrix is sparse
    b. The matrix is symmetric
    c. The matrix is diagonally dominant or symmetric positive-definite
    d. All of above

29. QR factorization is characterized by the following observations:
    a. The Q matrix is lower triangular
    b. The Q matrix is upper triangular
    c. The R matrix is lower triangular with nonzero diagonal entries.
    d. The R matrix is upper triangular with nonzero diagonal entries.

30. Given a QR factorization of a matrix A, the pseudoinverse A^+ can be efficiently computed as:
    a. $A^+ = R^{-1}Q^T$
    b. $A^+ = QR^T$
    c. $A^+ = Q^T R$
    d. $A^+ = QR^{-1}$

Q No	Correct
CS2008: Numerical Computing - .	
1	C
2	B
3	B
4	A
5	A
6	A
7	C
8	B
9	A
10	A
11	A
12	B
13	A
14	C
15	B
16	A
17	A
18	A
19	A
20	A
21	A
22	C
23	A
24	A, B
25	D
26	C
27	B
28	D
29	D
30	A

Figure 1: Key for 30 MCQs

Useful Formulae and Algorithms

$$x_i^{(k+1)} = \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right) \Big/ a_{ii}, \quad i = 1, \ldots, n$$

Figure 2: Jacobi iterative method

Algorithm 32 Cholesky factorization of symmetric and positive definite matrix $\boldsymbol{A}$

```
for i = 1 : n do
    H(i,i) = sqrt( A(i,i) − Σ_{k=1}^{i−1} H^2(i,k) )
    for j = i+1 : n do
        H(j,i) = ( A(j,i) − Σ_{k=1}^{i−1} H(j,k)H(i,k) ) / H(i,i)
    end for
end for
```

- **for** $i = 1 : n$ **do**
  - $\boldsymbol{H}(i,i) = \sqrt{\boldsymbol{A}(i,i) - \sum_{k=1}^{i-1} \boldsymbol{H}^2(i,k)}$
  - **for** $j = i+1 : n$ **do**
    - $\boldsymbol{H}(j,i) = \left(\boldsymbol{A}(j,i) - \sum_{k=1}^{i-1} \boldsymbol{H}(j,k)\boldsymbol{H}(i,k)\right) / \boldsymbol{H}(i,i)$
  - **end for**
- **end for**

Figure 3: Cholesky factorization algorithm

$$\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} - \boldsymbol{L})^{-1} \boldsymbol{U} \boldsymbol{x}^{(k)} + (\boldsymbol{D} - \boldsymbol{L})^{-1} \boldsymbol{b}$$

Figure 4: Gauss-Seidel iterative method

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \Big/ a_{ii}. \quad i = 1, \ldots, n$$

Figure 5: Gauss-Seidel iterative method

$$proj_{\mathbf{v}} \mathbf{a} = \frac{\mathbf{a} \cdot \mathbf{v}}{\|\mathbf{v}\|^2} \mathbf{v}$$

Figure 6: Projection of vector a on v

$$x_{k+1} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})}$$

Figure 7: Secant method

Algorithm 31 *LU* factorization with partial pivoting

Given the array $\boldsymbol{A}$
for $k = 1 : n-1$ **do**
  Find p such that $|\boldsymbol{A}(p,k)| = \max_{k \le p \le n} |\boldsymbol{A}(k:n,k)|$
  Swap rows $\boldsymbol{A}(k,:) \leftrightarrow \boldsymbol{A}(p,:)$
  Swap rows $perm(k) \leftrightarrow perm(p)$
  for $i = k+1 : n$ **do**
    if $\boldsymbol{A}(i,k) \neq 0$ **then**
      $m_{ik} = \boldsymbol{A}(i,k)/\boldsymbol{A}(k,k)$
      $\boldsymbol{A}(i,k+1:n) = \boldsymbol{A}(i,k+1,n) - m_{ik} \cdot \boldsymbol{A}(k,k+1:n)$
      $\boldsymbol{A}(i,k) = m_{ik}$
    end if
  end for
end for

Figure 8: LU factorization algorithm