Chapter 9

Gauss Elimination

Chapter Objectives

- Solve small sets of linear equations with Cramer's rule.
- Implement forward elimination and back substitution as in Gauss elimination.
- Recognize singular and ill-conditioned problems.
- Implement partial pivoting.

$$[A]\{x\} = \{b\}$$

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$D = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Cramer's Rule

Cramer's Ruleeach unknown in a system of linear algebraic equations may be expressed as a fraction of two determinants with denominator *D* and with the numerator obtained from *D* by replacing the column of coefficients of the unknown in question by the constants b_1, b_2, …, b_n.

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{D}$$

Cramer's Rule Example

- Find x_2 in the following system of equations:

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

- Find the determinant *D*

$$D = \begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix} = 0.3\begin{vmatrix} 1 & 1.9 \\ 0.3 & 0.5 \end{vmatrix} - 0.52\begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1\begin{vmatrix} 0.5 & 1 \\ 0.1 & 0.4 \end{vmatrix} = -0.0022$$

- Find determinant D_2 by replacing *D*'s second column with *b*

$$D_2 = \begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix} = 0.3\begin{vmatrix} 0.67 & 1.9 \\ -0.44 & 0.5 \end{vmatrix} - 0.01\begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1\begin{vmatrix} 0.5 & 0.67 \\ 0.1 & -0.44 \end{vmatrix} = 0.0649$$

- Divide

$$x_2 = \frac{D_2}{D} = \frac{0.0649}{-0.0022} = -29.5$$

Naïve Gauss Elimination

- For larger systems, Cramer's Rule can become awkward.
- Instead, a sequential process of removing unknowns from equations using *forward elimination* followed by *back substitution* may be used - this is Gauss elimination.
- "Naïve" Gauss elimination simply means the process does not check for potential problems resulting from division by zero.

Elimination of Unknowns

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

$$a_{21}a_{11}x_1 + a_{21}a_{12}x_2 = a_{21}b_1$$
$$a_{11}a_{21}x_1 + a_{11}a_{22}x_2 = a_{11}b_2$$

$$a_{11}a_{22}x_2 - a_{21}a_{12}x_2 = a_{11}b_2 - a_{21}b_1$$

$$x_2 = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{21}a_{12}}$$

$$x_1 = \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - a_{21}a_{12}}$$

Forward Elimination of Unknowns

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\
\vdots \qquad\qquad & \vdots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

$$
a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \frac{a_{21}}{a_{11}}a_{13}x_3 + \cdots + \frac{a_{21}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}b_1
$$

$$
\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \cdots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n}\right)x_n = b_2 - \frac{a_{21}}{a_{11}}b_1
$$

$$
a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2
$$

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2
\end{aligned}
$$

Forward Elimination of Unknowns (cont)

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2 \\
a'_{32}x_2 + a'_{33}x_3 + \cdots + a'_{3n}x_n &= b'_3 \\
\vdots \qquad\qquad \vdots & \\
a'_{n2}x_2 + a'_{n3}x_3 + \cdots + a'_{nn}x_n &= b'_n
\end{aligned}
$$

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2 \\
a''_{33}x_3 + \cdots + a''_{3n}x_n &= b''_3 \\
\vdots \qquad\qquad & \vdots \\
a''_{n3}x_3 + \cdots + a''_{nn}x_n &= b''_n
\end{aligned}
$$

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2 \\
a''_{33}x_3 + \cdots + a''_{3n}x_n &= b''_3 \\
\ddots \qquad\qquad &\vdots \\
a_{nn}^{(n-1)}x_n &= b_n^{(n-1)}
\end{aligned}
$$

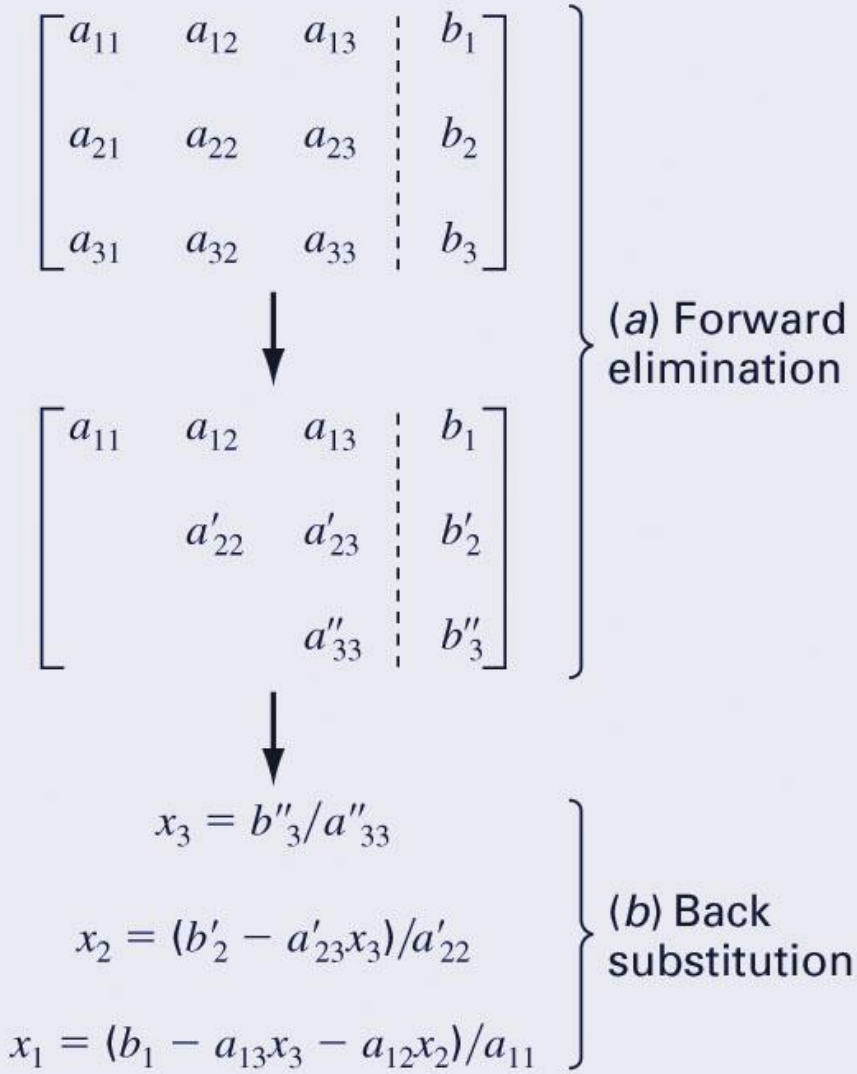
Back Substitution

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \qquad \text{for } i = n-1, n-2, \ldots, 1$$

Naïve Gauss Elimination (cont)

- Forward elimination
  - Starting with the first row, add or subtract multiples of that row to eliminate the first coefficient from the second row and beyond.
  - Continue this process with the second row to remove the second coefficient from the third row and beyond.
  - Stop when an upper triangular matrix remains.
- Back substitution
  - Starting with the *last* row, solve for the unknown, then substitute that value into the next highest row.
  - Because of the upper-triangular nature of the matrix, each row will contain only one more unknown.



```python
def gaussnaive(A,b):
    """
    gaussnaive: naive Gauss elimination
    input:
    A = coefficient matrix
    b = constant vector
    output:
    x = solution vector
    """
    (n,m) = A.shape
    #n = nm[0]
    #m = nm[1]
    if n != m:
        return 'Coefficient matrix A must be square'
    nb = n+1
    # build augmented matrix
    Aug = np.hstack((A,b))
    # forward elimination
    for k in range(n-1):
        for i in range(k+1,n):
            factor = Aug[i,k]/Aug[k,k]
            Aug[i,k:nb]=Aug[i,k:nb]-factor*Aug[k,k:nb]
    # back substitution
    x = np.zeros([n,1]) # create empty x array
    x = np.matrix(x) # convert to matrix type
    x[n-1]=Aug[n-1,nb-1]/Aug[n-1,n-1]
    for i in range(n-2,-1,-1):
        x[i]=(Aug[i,nb-1]-Aug[i,i+1:n]*x[i+1:n,0])/Aug[i,i]
    return x
```

Pivoting

- Problems arise with naïve Gauss elimination if a coefficient along the diagonal is 0 (problem: division by 0) or close to 0 (problem: round-off error)
- One way to combat these issues is to determine the coefficient with the largest absolute value in the column below the pivot element. The rows can then be switched so that the largest element is the pivot element. This is called *partial pivoting.*
- If the rows to the right of the pivot element are also checked and columns switched, this is called *complete pivoting*.

Example

Solve the equations using (a) Direct substition (b) Gauss Partial Pivot

$$-y + z = 1$$
$$2x - z = 3$$
$$-x + 2y - 3z = 0$$
$$A = [0 \ -1 \ \ 1; 2 \ \ 0 \ \ -1; -1 \ \ 2 \ \ -3]$$
$$b = [1; 3; 0]$$

Direct Substitution: x=1/3, y=-10/3, z=-7/3

Gauss Elimination

We have the augmented matrix

$$Aug = \begin{bmatrix} 0 & -1 & 1 & 1 \\ 2 & 0 & -1 & 3 \\ -1 & 2 & -3 & 0 \end{bmatrix}$$

Pivot interchanging 1st and 2nd rows

$$Aug = \begin{bmatrix} 2 & 0 & -1 & 3 \\ 0 & -1 & 1 & 1 \\ -1 & 2 & -3 & 0 \end{bmatrix}$$

Subtract (1st row)*(-1)/2 from 3rd row

$$Aug = \begin{bmatrix} 2 & 0 & -1 & 3 \\ 0 & -1 & 1 & 1 \\ 0 & 2 & -3.5 & 1.5 \end{bmatrix}$$

- Partial pivot for 2nd row by interchanging 2nd and 3rd rows:

$$Aug = \begin{bmatrix} 2 & 0 & -1 & 3 \\ 0 & 2 & -3.5 & 1.5 \\ 0 & -1 & 1 & 1 \end{bmatrix}$$

- Subtract (2nd row)*(-1)/2 from 3rd row

$$Aug = \begin{bmatrix} 2 & 0 & -1 & 3 \\ 0 & 2 & -3.5 & 1.5 \\ 0 & 0 & -0.75 & 1.75 \end{bmatrix}$$

Backward substitution

$z = 1.75/(-0.75) = -2.3333$

$y = 0.5*(1.5-3.5*2.3333)= -3.3333$

$x = 0.5*(3-2.3333) = 0.3333$

```python
import numpy as np

def gausspivot(A, b):
    """
  gausspivot: Gauss elimination with partial pivoting
    Input:
        A = coefficient matrix (n x n)
        b = constant vector (n x 1)
    Output:
        x = solution vector (n x 1)
    """
    (n, m) = A.shape
    if n != m:
        raise ValueError("Coefficient matrix A must be square")
    # Build augmented matrix
    Aug = np.hstack((A, b))
    # Forward elimination
    for k in range(n - 1):
        # Partial pivoting: Find the row with the maximum pivot element
        max_row_index = np.argmax(abs(Aug[k:n, k])) + k
        # Swap rows if needed
        if max_row_index != k:
            Aug[[k, max_row_index]] = Aug[[max_row_index, k]]
        # Perform elimination
        for i in range(k + 1, n):
            factor = Aug[i, k] / Aug[k, k]
            Aug[i, k:] -= factor * Aug[k, k:]
    # Back substitution
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (Aug[i, -1] - np.dot(Aug[i, i + 1:n], x[i + 1:])) / Aug[i, i]
    return x
# Example usage
A = np.array([[0, -1, 1], [2, 0, -1], [-1, 2, -3]], dtype=float)
b = np.array([1, 3, 0], dtype=float).reshape(-1, 1)
x = gausspivot(A, b)
print("Solution vector x:\n", x)
```

Computer Calculation

x=GaussPivotNew(A,b,0.0001)
gives 0.333, -3.333, -2.333

x=GaussNaive(A,b)

gives NaN,NaN,NaN

Note that you can solve the same problem using fsolve.

In general, it is faster to use Gauss elimination method for linear system than to use fsolve.

Use the latter for non-linear problems.

Note also for fsolve one needs to provide initial guess.

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

def f(x):
    y = np.zeros(3)
    y[0] = -x[1] + x[2] - 1
    y[1] = 2 * x[0] - x[2] - 3
    y[2] = -x[0] + 2 * x[1] - 3 * x[2]
    return y

def example_fsolve_linear():
    x0 = np.array([0.5, -0.5, 1])
    x = fsolve(f, x0)
    return x

result = example_fsolve_linear()
print(result)
```

```
[ 0.33333333 -3.33333333 -2.33333333]
```

Tridiagonal Systems

- A *tridiagonal* system is a banded system with a bandwidth of 3:

$$\begin{bmatrix} f_1 & g_1 & & & & & \\ e_2 & f_2 & g_2 & & & & \\ & e_3 & f_3 & g_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & & e_{n-1} & f_{n-1} & g_{n-1} \\ & & & & & & e_n & f_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ \cdot \\ \cdot \\ \cdot \\ r_{n-1} \\ r_n \end{Bmatrix}$$

- Tridiagonal systems can be solved using the same method as Gauss elimination, but with much less effort because most of the matrix elements are already 0.

```
def tridiag(e,f,g,r):
    """
    tridiag: solves a set of n linear algebraic equations
    with a tridiagonal-banded coefficient matris
    input:
    e = subdiagonal vector of length n, first element = 0
    f = diagonal vector of length n
    g = superdiagonal vector of length n, last element = 0
    r = constant vector of length n
    output:
    x = solution vector of length n
    """
    n = len(f)
    # forward elimination
    x = np.zeros([n])
    for k in range(1,n):
        factor = e[k]/f[k-1]
        f[k] = f[k] - factor*g[k-1]
        r[k] = r[k] - factor*r[k-1]
    # back substitution
    x[n-1] = r[n-1]/f[n-1]
    for k in range(n-2,-1,-1):
        x[k] = ( r[k] - g[k]*x[k+1] )/f[k]
    return x
```

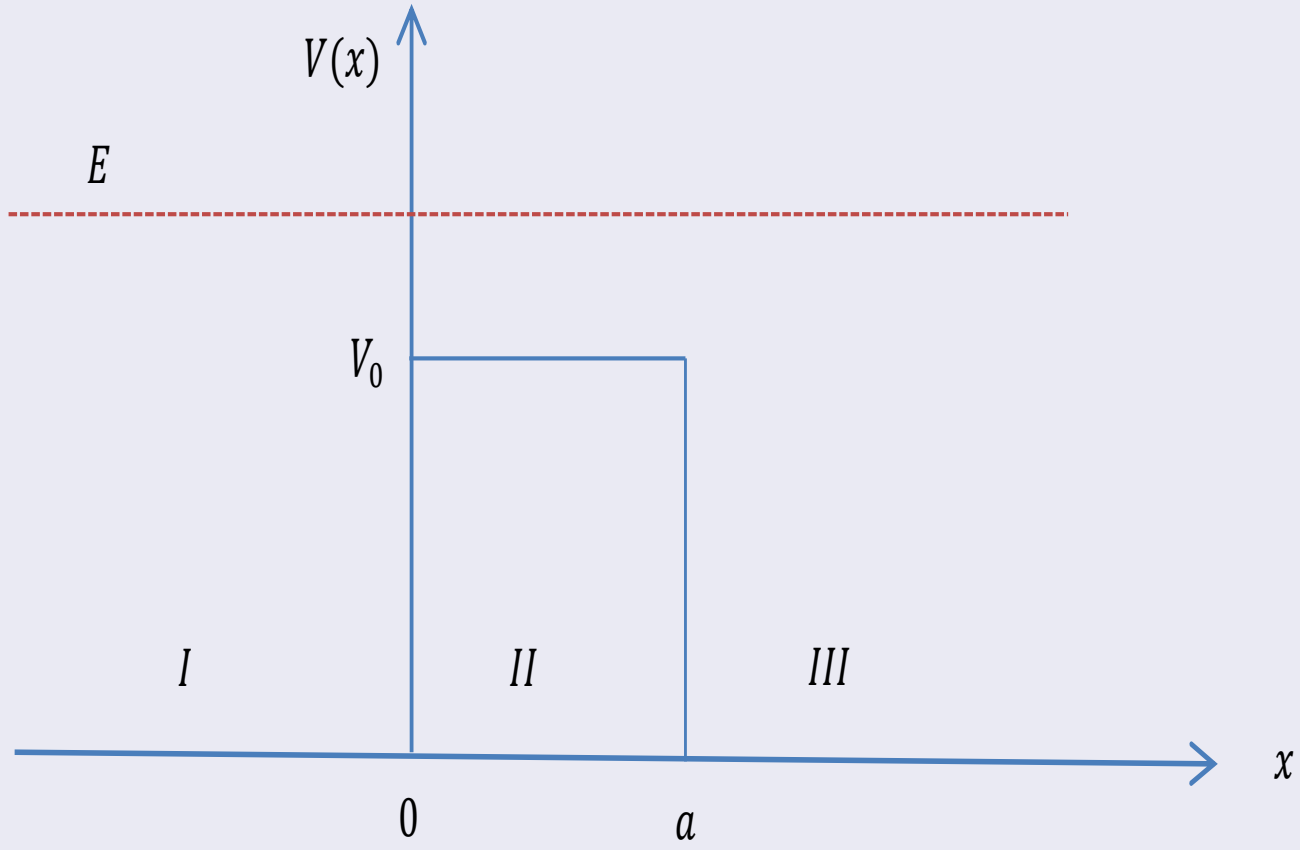
Example: Tunneling

Consider a particle incident from the left on the potential barrier with potential energy

$$V(x) = \begin{cases} 0 & x \leq 0 \\ V_0 & 0 \leq x \leq a \\ 0 & x \geq a \end{cases}$$

Plot the transmission and reflection coefficients as a function of energy for

$$b_0 = \sqrt{\frac{2ma^2V_0}{\hbar^2}} = 5$$


V(x)
E
V0
I
II
III
x
0
a

The boundary conditions at $x = 0,\ x = a$ give

$$A + B = C + D$$
$$kA - kB = KC - KD$$
$$C\exp(iKa) + D\exp(-iKa) = F\exp(ika)$$
$$KC\exp(iKa) - KD\exp(-iKa) = kF\exp(ika)$$

In this case the reflection and transmission coefficients are given by

$$R = \left|\frac{B}{A}\right|^2, \qquad T = \left|\frac{F}{A}\right|^2$$

Rewrite the equations as (by setting $A = 1$)

$$B - C - D + 0F = -1$$

$$kB + KC - KD + 0F = k$$

$$0B + \exp(iKa)C + \exp(-iKa)D - \exp(ika)\,F = 0$$

$$0B + K\exp(iKa)\,C - K\exp(-iKa)\,D - k\exp(ika)F = 0$$

Written as a matrix equation

$$\begin{bmatrix} 1 & -1 & -1 & 0 \\ k & K & -K & 0 \\ 0 & \exp(iKa) & \exp(-iKa) & -\exp(ika) \\ 0 & K\exp(iKa) & -K\exp(-iKa) & -k\exp(ika) \end{bmatrix} \begin{pmatrix} B \\ C \\ D \\ F \end{pmatrix} = \begin{pmatrix} -1 \\ k \\ 0 \\ 0 \end{pmatrix}$$

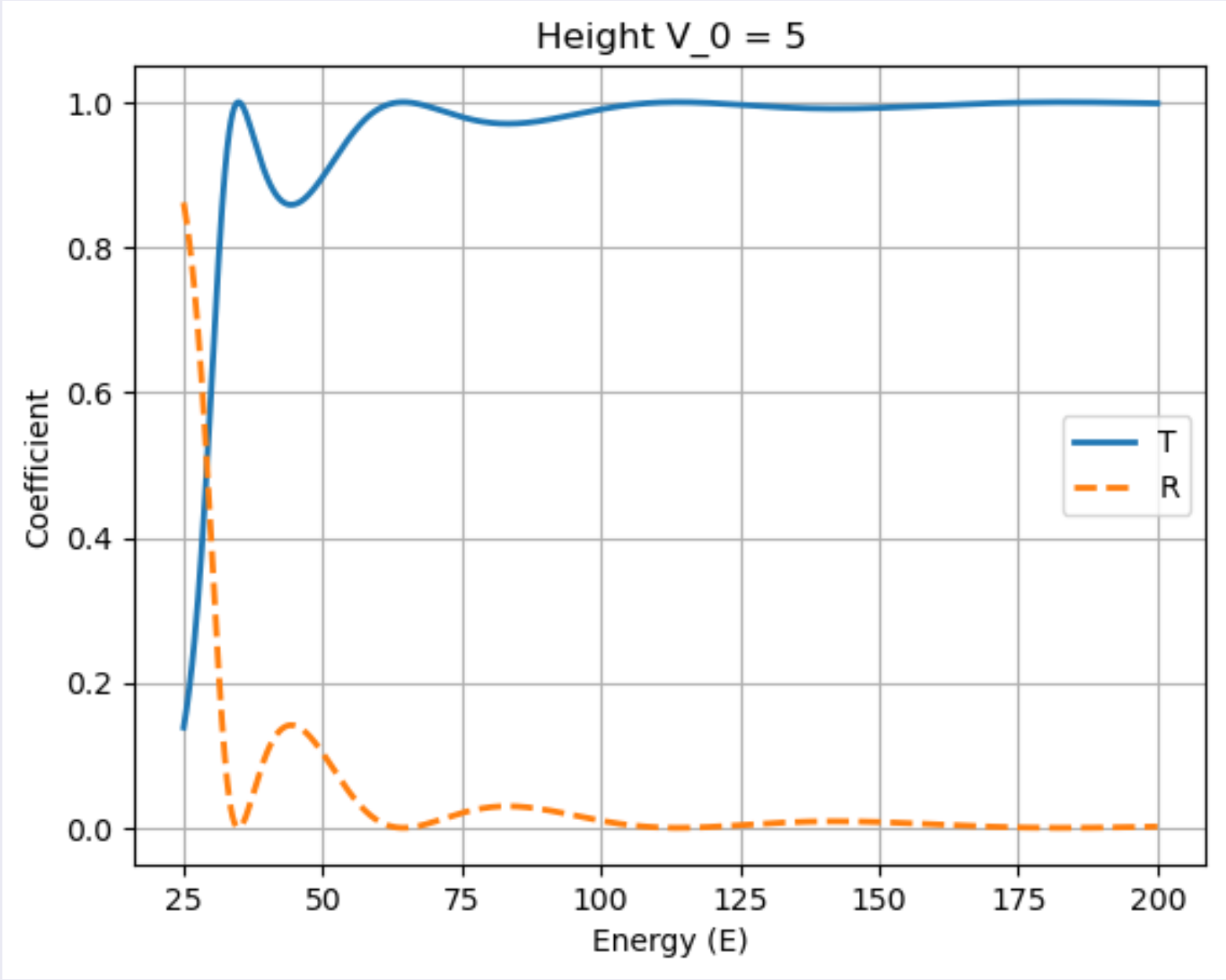
Write $ka = b$, $Ka = \sqrt{b^2 - b_0^2}$, where

$$b_0^2 = 2ma^2V_0/\hbar^2$$

The equation in matrix form is then (with $c = \sqrt{b^2 - b_0^2}$

$$\begin{bmatrix} 1 & -1 & -1 & 0 \\ b & c & -c & 0 \\ 0 & \exp(ic) & \exp(-ic) & -\exp(ib) \\ 0 & c\exp(ic) & -c\exp(-ic) & -b\exp(ib) \end{bmatrix} \begin{pmatrix} B \\ C \\ D \\ F \end{pmatrix} = \begin{pmatrix} -1 \\ b \\ 0 \\ 0 \end{pmatrix}$$

```python
def tunneling():
    # Parameters
    Emax = 200, b0 = 5
    # Lists to hold energy, transmission, and reflection coefficients
    Ep = []
    Tp = []
    Rp = []
    # Energy values from 0.01 to Emax with a step of 0.1
    for E in np.arange(0.01, Emax + 0.1, 0.1):
        b = np.sqrt(E)
        c = np.sqrt(b**2 - b0**2)
        # Constructing the matrix M
        M = np.array([[1, -1, -1, 0],
                      [b, c, -c, 0],
                      [0, np.exp(1j * (c - b)), np.exp(-1j * (c + b)), -1],
                      [0, c * np.exp(1j * (c - b)), -c * np.exp(-1j * (c + b)), -b]])
        # Right-hand side vector B
        B = np.array([-1, b, 0, 0])
        # Solve for X
        X = np.linalg.solve(M, B)
        # Calculate Transmission and Reflection coefficients
        T = abs(X[3])**2
        R = abs(X[0])**2
        # Append values to lists
        Ep.append(E)
        Tp.append(T)
        Rp.append(R)
    # Plotting
    plt.plot(Ep, Tp, '-', label='T', linewidth=2)
    plt.plot(Ep, Rp, '--', label='R', linewidth=2)
    plt.title(f'Height V_0 = {b0}')
    plt.legend()
    plt.xlabel('Energy (E)')
    plt.ylabel('Coefficient')
    plt.grid()
    plt.show()
tunneling()
```


Height V_0 = 5
Coefficient
Energy (E)
T
R
0.0
0.2
0.4
0.6
0.8
1.0
25
50
75
100
125
150
175
200

Homework

Solve 9.14 using GaussNaive,