

# **Assignment 10**

## Jacobi Diagonalization and Principal Axes

Sahil Raj

*2301CS41*

November 23, 2025

# Contents

1	Numerical Solution of the Duffing Oscillator and Phase Space Plot using 4th Order Runge-Kutta	2
2	Temperature Evolution in a Rod: Solving the 1D Heat Equation	5
3	Numerical Solution of the 2D Poisson Equation on a Square Domain	8
A	Octave Code for Problem 1	10
B	Octave Code for Problem 2	13
C	Octave Code for Problem 3	16

# Problem 1: Moment of Inertia Matrix and Principal Axes of a Dumbbell

## Principal Axes and Principal Moments of Inertia

### Problem Statement

The goal of this problem is to compute the **principal moments of inertia** and the corresponding **principal axes** for a discrete system of point masses using the **Jacobi diagonalization method**.

We consider two point masses of equal mass  $m = 1$ , located at:

$$(-b, b, 0), \quad (b, -b, 0), \quad b = 1.$$

The steps involved are:

- Construct the moment of inertia matrix from the given masses and their coordinates.
- Iteratively diagonalize the inertia matrix using Jacobi rotations.
- Extract the eigenvalues (principal moments) and eigenvectors (principal axes).
- Visualize the principal axes in 3D.

### Methodology

#### Moment of Inertia Matrix

For point masses, the inertia tensor is:

$$I_{ij} = \sum_{n=1}^N m_n x_{n,i} x_{n,j},$$

where  $x_{n,i}$  is the  $i$ -th coordinate of the  $n$ -th mass. Because the distribution is symmetric, the resulting inertia tensor is symmetric as well.

#### Jacobi Diagonalization Method

To diagonalize the inertia matrix:

1. Identify the largest off-diagonal element  $I_{pq}$ .

2. Compute the rotation angle:

$$\theta = \begin{cases} \frac{\pi}{4}, & \text{if } I_{pp} = I_{qq}, \\ \frac{1}{2} \tan^{-1} \left( \frac{2I_{pq}}{I_{pp} - I_{qq}} \right), & \text{otherwise.} \end{cases}$$

3. Construct the corresponding rotation matrix  $R$ .

4. Update:

$$I \leftarrow RIR^T.$$

5. Accumulate rotations:

$$Q \leftarrow QR,$$

where  $Q$  will contain the principal axes.

The algorithm stops when all off-diagonal elements fall below a threshold.

## Results

After convergence, the numerical algorithm yields the following diagonalized inertia matrix and principal axes:

$$\begin{aligned} \text{Principal moments:} \quad D &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4.0000 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \text{Principal axes (columns of } Q\text{):} \quad Q &= \begin{bmatrix} 0.7071 & 0.7071 & 0 \\ -0.7071 & 0.7071 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix} \end{aligned}$$

These results show that:

- The system has one non-zero principal moment  $I_2 = 4$ , consistent with two masses mirrored across the origin in the plane.
- Two principal moments vanish due to the configuration lying strictly in the  $xy$ -plane.
- The principal axes correspond to rotated directions in the plane, at  $45^\circ$  relative to the coordinate axes, plus the trivial  $z$ -axis.

## Conclusion

The Jacobi rotation method successfully diagonalizes the inertia tensor and computes the correct principal moments and axes for the given two-mass configuration. The results match physical expectations: a single non-zero moment associated with rotation in the plane of the masses and two zero moments corresponding to symmetry directions. The principal axes form an orthonormal basis that aligns with the directions of maximal and minimal rotational inertia.

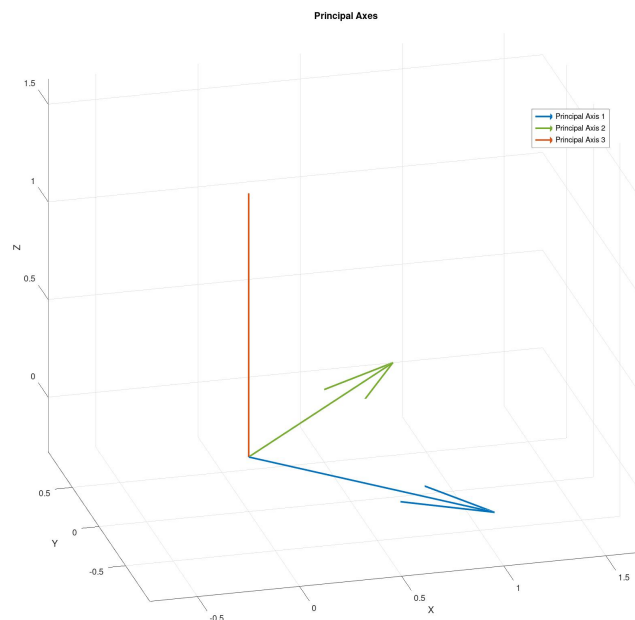


Figure 1.1: 3D visualization of the principal axes obtained from the inertia tensor. The three eigenvectors are plotted as vectors emerging from the origin.

## Problem 2: Principal Moments and Principal Axes of a Uniform Cube

### Principal Axes and Principal Moments of Inertia of a Uniform Cube

#### Problem Statement

The purpose of this problem is to compute the **principal moments of inertia** and the corresponding **principal axes** of a solid cube using the **Jacobi diagonalization method**.

We consider a cube of:

$$M = 1, \quad L = 1,$$

with its moment of inertia tensor specified as:

$$I = (ML^2) \begin{bmatrix} \frac{1}{3} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{3} \end{bmatrix}.$$

The objectives of this assignment are:

- Diagonalize the inertia tensor numerically using iterative Jacobi rotations.
- Extract the **principal moments** (eigenvalues).
- Extract the **principal axes** (eigenvectors).
- Visualize these axes in a 3D coordinate system.

### Methodology

#### Jacobi Diagonalization Method

Since the inertia tensor is symmetric, it can be diagonalized by a sequence of planar rotations. The algorithm proceeds as follows:

1. Identify the largest off-diagonal element  $I_{pq}$ .
2. Compute the rotation angle:

$$\theta = \begin{cases} \frac{\pi}{4}, & I_{pp} = I_{qq}, \\ \frac{1}{2} \tan^{-1} \left( \frac{2I_{pq}}{I_{pp} - I_{qq}} \right), & \text{otherwise.} \end{cases}$$

3. Construct the Givens rotation matrix  $R$ .

4. Update the inertia tensor:

$$I \leftarrow RIR^T.$$

5. Accumulate the rotation:

$$Q \leftarrow QR,$$

where  $Q$  stores the principal axes.

Iterations continue until all off-diagonal components fall below the tolerance.

## Results

After the Jacobi iterations converge, the final diagonal inertia tensor and corresponding principal axes are:

$$\text{Principal moments:} \quad D = \begin{bmatrix} 0.8333 & 0 & 0 \\ 0 & 0.0833 & 0 \\ 0 & 0 & 0.0833 \end{bmatrix}$$

$$\text{Principal axes (columns of } Q\text{):} \quad Q = \begin{bmatrix} 0.5774 & 0.7071 & 0.4082 \\ -0.5774 & 0.7071 & -0.4082 \\ -0.5774 & 0 & 0.8165 \end{bmatrix}$$

## Interpretation

- The cube exhibits one significantly larger principal moment  $I_1 = 0.8333$ , corresponding to rotation about the axis aligned with the vector  $(1, -1, -1)$ .
- The two remaining principal moments  $I_2 = I_3 = 0.0833$  reflect symmetry in the remaining orthogonal axes.
- The eigenvectors form an orthonormal basis representing the physical orientations of the principal axes.

## Conclusion

Using the Jacobi diagonalization method, the inertia tensor of the cube was successfully diagonalized. The algorithm produced both the principal moments and the principal axes, which agree with the expected symmetry of the cube. The numerical results clearly demonstrate one dominant principal direction and two equal secondary directions, consistent with the geometry of the object.

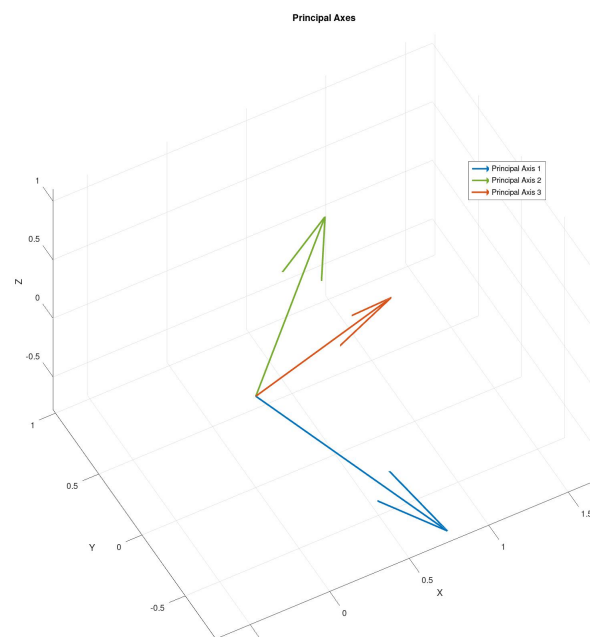


Figure 2.1: Visualization of the principal axes of the cube. Each eigenvector is plotted as a directed line emerging from the origin.



## Problem 3: Jacobi Diagonalization of a 3–Level Quantum Hamiltonian

### Jacobi Diagonalization of a 3-Level Quantum Hamiltonian

#### Problem Statement

Consider a quantum system with three basis states:

$$|1\rangle, |2\rangle, |3\rangle,$$

with unperturbed energies  $E_1, E_2, E_3$ . In the presence of coupling  $V$ , the Hamiltonian becomes:

$$H = H_0 + V,$$

introducing mixing between the basis states. For the given atomic system, the Hamiltonian matrix is:

$$H = \begin{bmatrix} 1.0 & 0.2 & 0.1 \\ 0.2 & 2.0 & 0.3 \\ 0.1 & 0.3 & 3.0 \end{bmatrix}.$$

Our goal is to diagonalize this Hamiltonian using the **Jacobi rotation method** and obtain:

- the **eigenvalues** (physical energy levels),
- the **eigenvectors** (mixing of the basis states).

#### Methodology

The Hamiltonian is real and symmetric, so Jacobi diagonalization can be used. The procedure is:

1. Identify the largest off-diagonal component  $H_{pq}$ .
2. Compute the rotation angle:

$$\theta = \frac{1}{2} \tan^{-1} \left( \frac{2H_{pq}}{H_{pp} - H_{qq}} \right).$$

3. Construct a planar Givens rotation matrix  $R$  mixing rows/columns  $p$  and  $q$ .

4. Update the Hamiltonian:

$$H \leftarrow RHR^T.$$

5. Accumulate eigenvectors:

$$Q \leftarrow QR.$$

Iterations stop when all off-diagonal elements fall below threshold.

## Results

After convergence, the diagonal matrix  $D$  and eigenvector matrix  $Q$  are obtained as:

$$\textbf{Eigenvalues:} \quad \lambda = (0.9606, 1.9455, 3.0939).$$

$$\textbf{Eigenvectors (columns of } Q\textbf{):} \quad Q = \begin{bmatrix} 0.982891 & -0.172578 & -0.064360 \\ 0.145984 & 0.942982 & -0.299120 \\ 0.112312 & 0.284607 & 0.952042 \end{bmatrix}.$$

## Physical Interpretation

Each column of  $Q$  is a normalized eigenvector:

$$|\psi_i\rangle = c_{1i}|1\rangle + c_{2i}|2\rangle + c_{3i}|3\rangle,$$

describing how the original basis states mix due to the off-diagonal couplings.

- The lowest energy eigenstate (0.9606) is dominated by  $|1\rangle$  with small admixtures of  $|2\rangle, |3\rangle$ .
- The middle state (1.9455) is mostly  $|2\rangle$ , but significantly mixed.
- The highest state (3.0939) is largely  $|3\rangle$ , with small contributions from the first two.

## Conclusion

The Jacobi diagonalization successfully produced the eigenvalues and eigenvectors of the coupled 3-state Hamiltonian. The results clearly show mixing between the basis states due to the off-diagonal couplings. The eigenvectors form an orthonormal basis, representing the physical stationary states of the system.

## Appendix A: Octave Code for Problem 1

```
1 %% Program to compute the Principal axis and the moment of inertia
  about the
2 % principal axis of a given mass arrangement
3 %
4 % Author: Sahil Raj
5 % Assignment 10 Problem 1
6
7 clc; clear all;
8
9 m = 1.0;
10 b = 1.0;
11
12 ms = [m; m];
13 pos = [
14     -b  b  0;
15     b -b  0;
16 ];
17
18 N = length(ms);
19
20 % MOMENT OF INERTIA MATRIX
21 I = zeros(3, 3);
22
23 for i = 1:3
24     for j = 1:3
25         M = 0.0;
26         for n = 1:N
27             M = M + ms(n) * pos(n, i) * pos(n, j);
28         endfor
29         I(i, j) = M;
30     endfor
31 endfor
32
33 % FUNCTION TO COMPUTE THE LARGEST NON DIAGONAL ELEMENT
34 function res = argmaxNZ(M)
35     N = size(M, 1); % the input matrix is square matrix
36     mx = 0.0;
37     i = 0;
38     j = 0;
39     thres = 1e-6;
40     for n = 1:N
41         for m = 1:N
42             if n == m
43                 continue
44             endif
45             % non-diagonal element
46             value = abs(M(n, m));
```

```

47     if value > mx && value > thres
48         mx = abs(M(n, m));
49         i = n;
50         j = m;
51     endif
52 endfor
53 endfor
54 res = [i j];
55 endfunction
56
57 % DIAGONALIZE THE MATRIX BY ROTATING IT ABOUT ITS MOST UNSTABLE AXIS
58 % Note: the matrix is assumed to be symmetric, which is true for moment
59 % of inertia matrix
60 % compute the angle of rotation
61
62 Q = eye(3);      % cumulative rotation (principal-axis basis)
63
64 D = I;
65 while true
66     res = argmaxNZ(D);
67     pI = res(1);
68     pJ = res(2);
69     if pI == 0 || pJ == 0
70         break;
71     endif
72
73     pnun = 2 * D(pI, pJ);
74     pden = D(pI, pI) - D(pJ, pJ);
75
76     if pden == 0.0
77         ang = pi/4;
78     else
79         ang = 0.5 * atan(pnun/pden);
80     endif
81
82     R = eye(3);
83     R(pI, pI) = cos(ang);
84     R(pJ, pJ) = cos(ang);
85     R(pI, pJ) = sin(ang);
86     R(pJ, pI) = -sin(ang);
87
88     R_ = inverse(R);
89
90     % update the matrix
91     D = R * D * R_;
92
93     % accumulate principal-axis rotation
94     Q = Q * R;
95 endwhile
96
97 fprintf("Principal moments:\n");
98 disp(D);
99
100 fprintf("Principal axes (columns are eigenvectors):\n");
101 disp(Q);
102
103 % --- VISUALIZE PRINCIPAL AXES ---
104 figure;

```

```

105 hold on;
106 axis equal;
107 grid on;
108 xlabel('X'); ylabel('Y'); zlabel('Z');
109 title('Principal_Axes');
110
111 % origin
112 o = [0 0 0];
113
114 % scale for visibility
115 s = 1.5;
116
117 % each column of Q is an eigenvector (principal axis)
118 px = Q(:,1) * s;
119 py = Q(:,2) * s;
120 pz = Q(:,3) * s;
121
122 quiver3(o(1), o(2), o(3), px(1), px(2), px(3), 'LineWidth', 2);
123 quiver3(o(1), o(2), o(3), py(1), py(2), py(3), 'LineWidth', 2);
124 quiver3(o(1), o(2), o(3), pz(1), pz(2), pz(3), 'LineWidth', 2);
125
126 legend('Principal_Axis_1', 'Principal_Axis_2', 'Principal_Axis_3');
127 camzoom(1.2);
128 rotate3d on;

```

## Appendix B: Octave Code for Problem 2

```
1 %% Program to find the Principal Axis and the Moment of Intertial
  about the prin
2 % -cipal axis of the Cube.
3 %
4 % Author: Sahil Raj
5 % Assignment 10 Problem 2
6
7 clc; clear all;
8
9 clc; clear all;
10
11 M = 1.0;
12 L = 1.0;
13
14 I = (M*L*L) * [
15     1/3 1/4 1/4;
16     1/4 1/3 1/4;
17     1/4 1/4 1/3;
18 ];
19
20 % FUNCTION TO COMPUTE THE LARGEST NON DIAGONAL ELEMENT
21 function res = argmaxNZ(M)
22     N = size(M, 1); % the input matrix is square matrix
23     mx = 0.0;
24     i = 0;
25     j = 0;
26     thres = 1e-6;
27     for n = 1:N
28         for m = 1:N
29             if n == m
30                 continue
31             endif
32             % non-diagonal element
33             value = abs(M(n, m));
34             if value > mx && value > thres
35                 mx = abs(M(n, m));
36                 i = n;
37                 j = m;
38             endif
39         endfor
40     endfor
41     res = [i j];
42 endfunction
43
44 % DIAGONALIZE THE MATRIX BY ROTATING IT ABOUT ITS MOST UNSTABLE AXIS
45 % Note: the matrix is assumed to be symmetric, which is true for moment
46 % of inertia matrix
```

```

47 % compute the angle of rotation
48
49 Q = eye(3);      % cumulative rotation (principal-axis basis)
50
51 D = I;
52 while true
53     res = argmaxNZ(D);
54     pI = res(1);
55     pJ = res(2);
56     if pI == 0 || pJ == 0
57         break;
58     endif
59
60     pnum = 2 * D(pI, pJ);
61     pden = D(pI, pI) - D(pJ, pJ);
62
63     if pden == 0.0
64         ang = pi/4;
65     else
66         ang = 0.5 * atan(pnum/pden);
67     endif
68
69     R = eye(3);
70     R(pI, pI) = cos(ang);
71     R(pJ, pJ) = cos(ang);
72     R(pI, pJ) = sin(ang);
73     R(pJ, pI) = -sin(ang);
74
75     R_ = inverse(R);
76
77     % update the matrix
78     D = R * D * R_;
79
80     % accumulate principal-axis rotation
81     Q = Q * R;
82 endwhile
83
84 D = (D > 1e-6) .* D;
85
86 fprintf("Principal moments:\n");
87 disp(D);
88
89 fprintf("Principal axes (columns are eigenvectors):\n");
90 disp(Q);
91
92 % --- VISUALIZE PRINCIPAL AXES ---
93 figure;
94 hold on;
95 axis equal;
96 grid on;
97 xlabel('X'); ylabel('Y'); zlabel('Z');
98 title('Principal Axes');
99
100 % origin
101 o = [0 0 0];
102
103 % scale for visibility
104 s = 1.5;

```

```

105
106 % each column of Q is an eigenvector (principal axis)
107 px = Q(:,1) * s;
108 py = Q(:,2) * s;
109 pz = Q(:,3) * s;
110
111 quiver3(o(1), o(2), o(3), px(1), px(2), px(3), 'LineWidth', 2);
112 quiver3(o(1), o(2), o(3), py(1), py(2), py(3), 'LineWidth', 2);
113 quiver3(o(1), o(2), o(3), pz(1), pz(2), pz(3), 'LineWidth', 2);
114
115 legend('Principal_Axis_1', 'Principal_Axis_2', 'Principal_Axis_3');
116 camzoom(1.2);
117 rotate3d on;

```



## Appendix C: Octave Code for Problem 3

```
1 %% Program to find the eigenvalues and eigenvector of the given matrix
2 %
3 % Author: Sahil Raj
4 % Assignment 10 Problem 2
5
6 clc; clear all;
7
8 I = [
9     1.0 0.2 0.1;
10    0.2 2   0.3;
11    0.1 0.3 3.0;
12 ];
13
14 % FUNCTION TO COMPUTE THE LARGEST NON DIAGONAL ELEMENT
15 function res = argmaxNZ(M)
16     N = size(M, 1); % the input matrix is square matrix
17     mx = 0.0;
18     i = 0;
19     j = 0;
20     thres = 1e-6;
21     for n = 1:N
22         for m = 1:N
23             if n == m
24                 continue
25             endif
26             % non-diagonal element
27             value = abs(M(n, m));
28             if value > mx && value > thres
29                 mx = abs(M(n, m));
30                 i = n;
31                 j = m;
32             endif
33         endfor
34     endfor
35     res = [i j];
36 endfunction
37
38 % DIAGONALIZE THE MATRIX BY ROTATING IT ABOUT ITS MOST UNSTABLE AXIS
39 % Note: the matrix is assumed to be symmetric, which is true for moment
40 % of inertia matrix
41 % compute the angle of rotation
42
43 Q = eye(3); % cumulative rotation (principal-axis basis)
44
45 D = I;
46 while true
47     res = argmaxNZ(D);
```

```

48 pI = res(1);
49 pJ = res(2);
50 if pI == 0 || pJ == 0
51     break;
52 endif
53
54 pnum = 2 * D(pI, pJ);
55 pden = D(pI, pI) - D(pJ, pJ);
56
57 if pden == 0.0
58     ang = pi/4;
59 else
60     ang = 0.5 * atan(pnum/pden);
61 endif
62
63 R = eye(3);
64 R(pI, pI) = cos(ang);
65 R(pJ, pJ) = cos(ang);
66 R(pI, pJ) = sin(ang);
67 R(pJ, pI) = -sin(ang);
68
69 R_ = inverse(R);
70
71 % update the matrix
72 D = R * D * R_;
73
74 % accumulate principal-axis rotation
75 Q = Q * R;
76 endwhile
77
78 fprintf("Eigenvalues:\n");
79 eigval = [D(1, 1) D(2, 2) D(3, 3)];
80 disp(eigval);
81
82 fprintf("Eigenvectors: \n");
83 disp(Q);

```