MATGEO Presentation: 5.5.7

Subhodeep Chakraborty
ee25btech11055,
IIT Hyderabad.

October 10, 2025

Problem Statement

Find the inverse of the following matrix, using elementary transformations $(12, 2019)$

$$\begin{pmatrix} 2 & 3 & 1 \\ 2 & 4 & 1 \\ 3 & 7 & 2 \end{pmatrix}$$

Given data

Given:

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 1 \\ 2 & 4 & 1 \\ 3 & 7 & 2 \end{pmatrix} \tag{3.1}$$

Formulae

Let $\mathbf{A}^{-1}$ be inverse of $\mathbf{A}$ We know

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \tag{3.2}$$

Thus augmented matrix to find $\mathbf{A}^{-1}$ is given by: $\left(\mathbf{A} \,\middle|\, \mathbf{I}\right)$

Solving

$$\left(\begin{array}{ccc|ccc} 2 & 3 & 1 & 1 & 0 & 0 \\ 2 & 4 & 1 & 0 & 1 & 0 \\ 3 & 7 & 2 & 0 & 0 & 1 \end{array}\right) \xleftrightarrow{R1 \leftrightarrow R3; R1 = R1/3} \tag{3.3}$$

$$\left(\begin{array}{ccc|ccc} 1 & 7/3 & 2/3 & 0 & 0 & 1/3 \\ 2 & 4 & 1 & 0 & 1 & 0 \\ 2 & 3 & 1 & 1 & 0 & 0 \end{array}\right) \xleftrightarrow{R2 = R2 - 2R1; R3 = R3 - 2R1} \tag{3.4}$$

$$\left(\begin{array}{ccc|ccc} 1 & 7/3 & 2/3 & 0 & 0 & 1/3 \\ 0 & -2/3 & -1/3 & 0 & 1 & -2/3 \\ 0 & -5/3 & -1/3 & 1 & 0 & -2/3 \end{array}\right) \xleftrightarrow{R3 = -5/3R3} \tag{3.5}$$

$$\left(\begin{array}{ccc|ccc} 1 & 7/3 & 2/3 & 0 & 0 & 1/3 \\ 0 & -2/3 & -1/3 & 0 & 1 & -2/3 \\ 0 & 1 & 1/5 & -3/5 & 0 & 2/5 \end{array}\right) \xleftrightarrow{R1 = R1 - 7/3R3; R2 = 2/3R3} \tag{3.6}$$

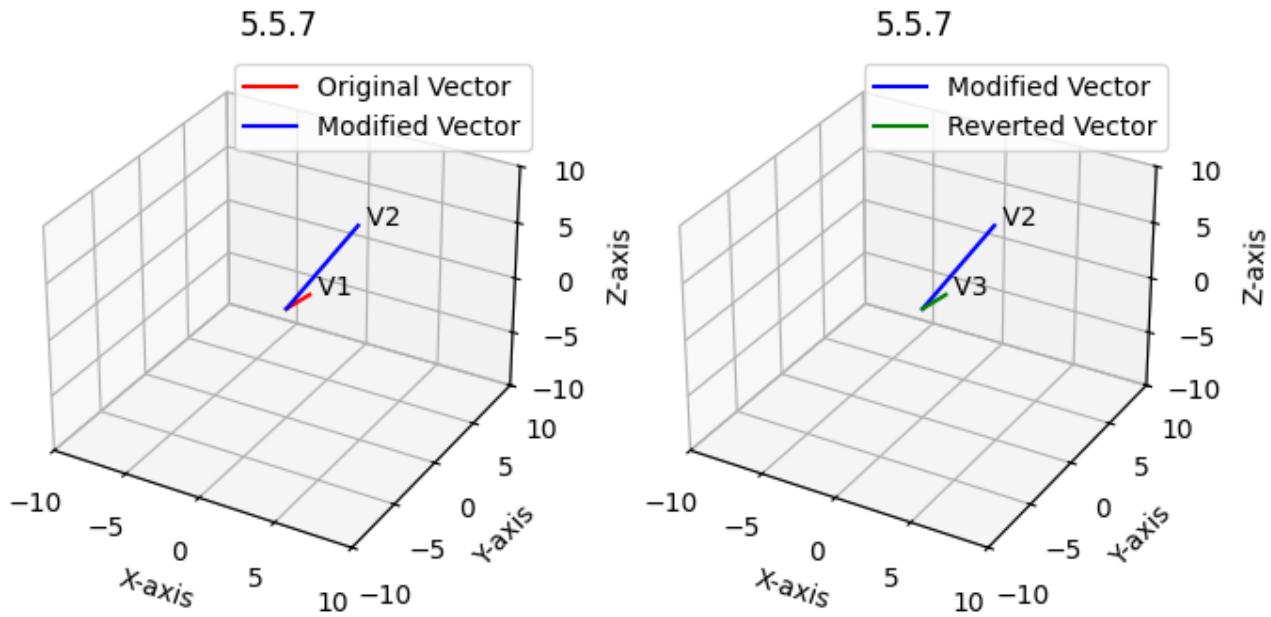
$$\left(\begin{array}{ccc|ccc} 1 & 0 & 1/5 & 7/5 & 0 & -3/5 \\ 0 & 0 & -1/5 & -2/5 & 1 & -2/5 \\ 0 & 1 & 1/5 & -3/5 & 0 & 2/5 \end{array}\right) \xleftrightarrow{R2 \leftrightarrow R3} \tag{3.7}$$

Result

So we have:

$$\mathbf{A^{-1}} = \begin{pmatrix} 1 & 1 & -1 \\ -1 & 1 & 0 \\ 2 & -5 & 2 \end{pmatrix} \tag{3.9}$$

Plot


5.5.7
Original Vector
Modified Vector
V2
V1
X-axis
Y-axis
Z-axis
5.5.7
Modified Vector
Reverted Vector
V2
V3
X-axis
Y-axis
Z-axis

C code for generating points on line

```
void point_gen(const double* P1, const double* P2, double t, double*
     result_point) {
    result_point[0] = P1[0] + t * (P2[0] - P1[0]);
    result_point[1] = P1[1] + t * (P2[1] - P1[1]);
    result_point[2] = P1[2] + t * (P2[2] - P1[2]);
}
```

C Code for matrix functions

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int find_inverse(const double *input_matrix, double *inverse_matrix, int
    n) {
    // Read up on memory
    double *augmented = (double *)malloc(n * 2 * n * sizeof(double))
        ;

    int i, j, k;
    const int width = 2 * n;
```

```
for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j) {
        augmented[i * width + j] = input_matrix[i * n + j];
    }
    for (j = n; j < width; ++j) {
        augmented[i * width + j] = (i == (j - n)) ? 1.0 : 0.0;
    }
}

for (i = 0; i < n; ++i) {
    // check why this matters. smtg to do with errors sir said iirc
    int max_row = i;
    for (k = i + 1; k < n; ++k) {
        if (fabs(augmented[k * width + i]) > fabs(augmented[
            max_row * width + i])) {
            max_row = k;
        }
    }
```

```
if (max_row != i) {
    for (k = 0; k < width; ++k) {
        double temp = augmented[i * width + k];
        augmented[i * width + k] = augmented[max_row *
            width + k];
        augmented[max_row * width + k] = temp;
    }
}

// check singular
if (fabs(augmented[i * width + i]) < 1e-9) {
    free(augmented);
    return 0;
}
```

```
        // –make 1
        double pivot = augmented[i * width + i];
        for (j = i; j < width; ++j) {
            augmented[i * width + j] /= pivot;
        }

        // make zero
        for (j = 0; j < n; ++j) {
            if (i != j) {
                double factor = augmented[j * width + i];
                for (k = 0; k < width; ++k) {
                    augmented[j * width + k] -= factor * augmented[i
                        * width + k];
                }
            }
        }
    }
```

```
    // Output inverse
    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            inverse_matrix[i * n + j] = augmented[i * width + (j + n)];
        }
    }

    free(augmented);
    return 1;
}
```

```
void mul(const double *a, const double *b, double *c, int m, int n,
    int p) {
    for (int i = 0; i < m; i++) {
        for (int k = 0; k < p; k++) {
            double temp = 0.0;
            for (int j = 0; j < n; j++) {
                temp += a[i * n + j] * b[j * p + k];
            }
            c[i * p + k] = temp;
        }
    }
}
```

Python code for plotting using C

```
import ctypes
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
import sys

libline = ctypes.CDLL("./line.so")
get_point = libline.point_gen
get_point.argtypes = [
    ctypes.POINTER(ctypes.c_double), # P1
    ctypes.POINTER(ctypes.c_double), # P2
    ctypes.c_double, # t
    ctypes.POINTER(ctypes.c_double), # result_point
]
get_point.restype = None
```

```
libmatrix = ctypes.CDLL("./matrix.so")
libmatrix.find_inverse.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags="
        C_CONTIGUOUS"),
    np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags="
        C_CONTIGUOUS"),
    ctypes.c_int,
]
libmatrix.find_inverse.restype = ctypes.c_int
find_inv = libmatrix.find_inverse
```

```
libmatrix.mul.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags="
        C_CONTIGUOUS"),
    np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags="
        C_CONTIGUOUS"),
    np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags="
        C_CONTIGUOUS"),
    ctypes.c_int,
    ctypes.c_int,
    ctypes.c_int,
]
libmatrix.mul.restype = None
matmul = libmatrix.mul
```

```
DoubleArray3 = ctypes.c_double * 3
a = DoubleArray3(0,0,0)
x = np.array([1,1,1],dtype=np.float64)

A = np.array([[3, 0, -1], [2, 3, 0], [0, 4, 1]],dtype=np.float64)
A_flat = A.flatten()
# flatten() passes a COPY! matrix isn't stored
inv = np.empty_like(A_flat)
if not find_inv(A_flat,inv,3):
    print("Matrix is singular")
    sys.exit()

inv = inv.reshape(A.shape)
b = np.empty_like(x)
c = np.empty_like(x)
matmul(A.flatten(),x,b,3,3,1)
matmul(inv.flatten(),b,c,3,3,1)
```

```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(121, projection="3d")

t_values = np.linspace(0, 1, 100)
line_points_x, line_points_y, line_points_z = [], [], []
for t in t_values:
    result_arr = DoubleArray3()
    get_point(a, DoubleArray3(x[0], x[1], x[2]), t, result_arr)
    line_points_x.append(result_arr[0])
    line_points_y.append(result_arr[1])
    line_points_z.append(result_arr[2])
ax.plot(
    line_points_x,
    line_points_y,
    line_points_z,
    color="red",
    label="Original Vector"
)
ax.text(x[0], x[1], x[2], " V1")
```

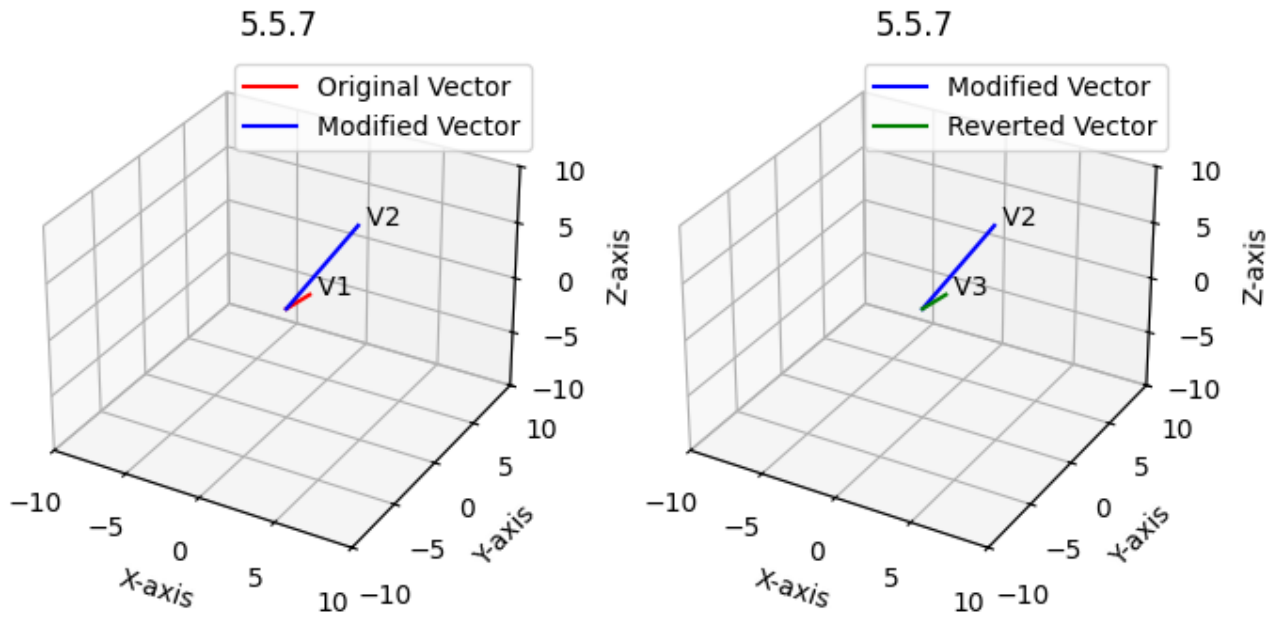
```
t_values = np.linspace(0, 1, 100)
line_points_x, line_points_y, line_points_z = [], [], []

for t in t_values:
    result_arr = DoubleArray3()

    get_point(a, DoubleArray3(b[0], b[1], b[2]), t, result_arr)

    line_points_x.append(result_arr[0])
    line_points_y.append(result_arr[1])
    line_points_z.append(result_arr[2])
ax.plot(
    line_points_x,
    line_points_y,
    line_points_z,
    color="blue",
    label="Modified Vector"
)
ax.text(b[0], b[1], b[2], " V2")
```

```
ax2 = fig.add_subplot(122, projection="3d")

t_values = np.linspace(0, 1, 100)
line_points_x, line_points_y, line_points_z = [], [], []
for t in t_values:
    result_arr = DoubleArray3()
    get_point(a, DoubleArray3(b[0], b[1], b[2]), t, result_arr)
    line_points_x.append(result_arr[0])
    line_points_y.append(result_arr[1])
    line_points_z.append(result_arr[2])
ax2.plot(
    line_points_x,
    line_points_y,
    line_points_z,
    color="blue",
    label="Modified Vector"
)
ax2.text(b[0], b[1], b[2], " V2")
```

```
t_values = np.linspace(0, 1, 100)
line_points_x, line_points_y, line_points_z = [], [], []

for t in t_values:
    result_arr = DoubleArray3()

    get_point(a, DoubleArray3(c[0], c[1], c[2]), t, result_arr)

    line_points_x.append(result_arr[0])
    line_points_y.append(result_arr[1])
    line_points_z.append(result_arr[2])
ax2.plot(
    line_points_x,
    line_points_y,
    line_points_z,
    color="green",
    label="Reverted Vector"
)
ax2.text(c[0], c[1], c[2], " V3")
```

```
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.set_title("5.5.7")
ax.set_xlim([-10, 10])
ax.set_ylim([-10, 10])
ax.set_zlim([-10, 10])
ax.legend()
ax.grid(True)
ax2.set_xlabel("X-axis")
ax2.set_ylabel("Y-axis")
ax2.set_zlabel("Z-axis")
ax2.set_title("5.5.7")
ax2.set_xlim([-10, 10])
ax2.set_ylim([-10, 10])
ax2.set_zlim([-10, 10])
ax2.legend()
ax2.grid(True)
plt.savefig("../figs/plot.png")
```

Plot


5.5.7
Original Vector
Modified Vector
V2
V1
X-axis
Y-axis
Z-axis
5.5.7
Modified Vector
Reverted Vector
V2
V3
X-axis
Y-axis
Z-axis

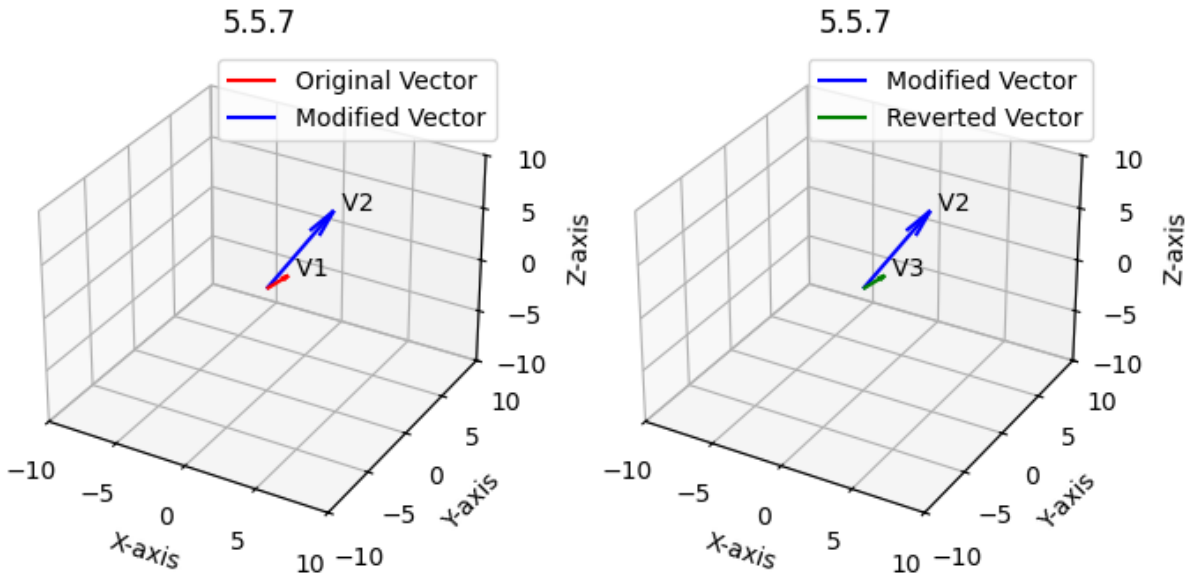
Pure Python code

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt

A = np.array([[3, 0, −1], [2, 3, 0], [0, 4, 1]])

x = np.array([1, 1, 1])
b = A @ x
c = LA.inv(A) @ b
print(x,A,b,c,sep='\n')



fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(121, projection="3d")
```

```
ax.quiver(0, 0, 0, x[0], x[1], x[2], color="red", label="Original Vector")
ax.text(x[0], x[1], x[2], " V1")
ax.quiver(0, 0, 0, b[0], b[1], b[2], color="blue", label="Modified Vector")
ax.text(b[0], b[1], b[2], " V2")

ax2 = fig.add_subplot(122, projection="3d")

ax2.quiver(0, 0, 0, b[0], b[1], b[2], color="blue", label="Modified Vector"
    )
ax2.text(b[0], b[1], b[2], " V2")
ax2.quiver(0, 0, 0, c[0], c[1], c[2], color="green", label="Reverted Vector
    ")
ax2.text(c[0], c[1], c[2], " V3")
```

```
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.set_title("5.5.7")
ax.set_xlim([-10, 10])
ax.set_ylim([-10, 10])
ax.set_zlim([-10, 10])
ax.legend()
ax.grid(True)
ax2.set_xlabel("X-axis")
ax2.set_ylabel("Y-axis")
ax2.set_zlabel("Z-axis")
ax2.set_title("5.5.7")
ax2.set_xlim([-10, 10])
ax2.set_ylim([-10, 10])
ax2.set_zlim([-10, 10])
ax2.legend()
ax2.grid(True)
plt.savefig("../figs/python.png")
```

Plot


5.5.7
Original Vector
Modified Vector
V2
V1
X-axis
Y-axis
Z-axis
5.5.7
Modified Vector
Reverted Vector
V2
V3
X-axis
Y-axis
Z-axis