Presentation - Matgeo

Aryansingh Sonaye
AI25BTECH11032
EE1030 - Matrix Theory

September 29, 2025

Problem Statement

Problem 9.4.4
Find the roots of the following quadratic equation graphically:

$$x^2 - 3x - 10 = 0 \tag{1.1}$$

Description of Variables used

The given quadratic can be written in the conic form

$$\mathbf{x}^T V\mathbf{x} + 2\mathbf{u}^T\mathbf{x} + f = 0 \tag{2.1}$$

where

$$V = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} -\frac{3}{2} \\ 0 \end{pmatrix}, \quad f = -10 \tag{2.2}$$

Since the roots of the quadratic correspond to intersections with the x-axis, we represent the line

$$L : \mathbf{x} = \mathbf{h} + \kappa\mathbf{m} \tag{2.3}$$

with

$$\mathbf{h} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{m} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{2.4}$$

Description of Variables used

Symbol	Value
V	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$
$\mathbf{u}$	$\begin{pmatrix} -\frac{3}{2} \\ 0 \end{pmatrix}$
f	-10
$\mathbf{h}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$\mathbf{m}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Theoretical Solution

The points of intersection of a line with a conic are given by

$$\kappa = \frac{1}{\mathbf{m}^T V \mathbf{m}} \left(-\mathbf{m}^T (V\mathbf{h} + \mathbf{u}) \pm \sqrt{(\mathbf{m}^T (V\mathbf{h} + \mathbf{u}))^2 - g(\mathbf{h})(\mathbf{m}^T V \mathbf{m})} \right), \tag{2.5}$$

where

$$g(\mathbf{h}) = \mathbf{h}^T V \mathbf{h} + 2\mathbf{u}^T \mathbf{h} + f. \tag{2.6}$$

Step 1: Compute $m^T V m$

$$\mathbf{m}^T V \mathbf{m} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 \tag{2.7}$$

Theoretical Solution

Step 2: Compute $Vh + u$

$$V\mathbf{h} + \mathbf{u} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -\frac{3}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{3}{2} \\ 0 \end{pmatrix} \tag{2.8}$$

Step 3: Compute $m^T(Vh + u)$

$$\mathbf{m}^T(V\mathbf{h} + \mathbf{u}) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} -\frac{3}{2} \\ 0 \end{pmatrix} = -\frac{3}{2} \tag{2.9}$$

Step 4: Compute $g(h)$

$$g(\mathbf{h}) = \mathbf{h}^T V\mathbf{h} + 2\mathbf{u}^T\mathbf{h} + f = -10 \tag{2.10}$$

Theoretical Solution

Step 5: Substitute into formula for κ

$$\kappa = -(-\tfrac{3}{2}) \pm \sqrt{\left(-\tfrac{3}{2}\right)^2 - (-10)(1)} \tag{2.11}$$

$$= \tfrac{3}{2} \pm \sqrt{\tfrac{9}{4} + 10} \tag{2.12}$$

$$= \tfrac{3}{2} \pm \sqrt{\tfrac{49}{4}} \tag{2.13}$$

$$= \tfrac{3}{2} \pm \tfrac{7}{2} \tag{2.14}$$

Step 6: Evaluate roots

$$\kappa_1 = \tfrac{3}{2} + \tfrac{7}{2} = 5 \tag{2.15}$$

$$\kappa_2 = \tfrac{3}{2} - \tfrac{7}{2} = -2 \tag{2.16}$$

Theoretical Solution

Step 7: Find intersection points The intersection points are obtained as

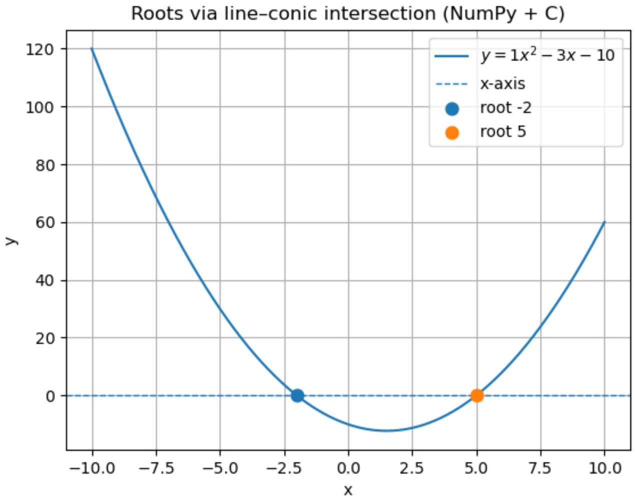
$$\mathbf{x}_1 = \mathbf{h} + \kappa_1 \mathbf{m} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 5 \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix} \tag{2.17}$$

$$\mathbf{x}_2 = \mathbf{h} + \kappa_2 \mathbf{m} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \end{pmatrix} \tag{2.18}$$

Thus, the quadratic $x^2 - 3x - 10 = 0$ intersects the x-axis at

$$\boxed{x = -2 \quad \text{and} \quad x = 5} \tag{3.1}$$

Plot



Roots via line–conic intersection (NumPy + C)
$y = 1x^2 - 3x - 10$
x-axis
root -2
root 5
120
100
80
60
40
20
0
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

Figure

Code - C

```
#include <math.h>

void intersect_line_conic(
    const double V[4], // 2x2 row-major: [V00,V01,V10,V11]
    const double u[2], // size-2
    double f, // scalar
    const double h[2], // line anchor
    const double m[2], // line direction
    double kappa_out[2], // outputs
    double x1[2],
    double x2[2],
    int *status // 2=two real, 1=one (tangent), 0=none
) {
    double Vm0 = V[0]*m[0] + V[1]*m[1];
    double Vm1 = V[2]*m[0] + V[3]*m[1];
    double mTVm = m[0]*Vm0 + m[1]*Vm1;
```

Code - C

```
double Vh0 = V[0]*h[0] + V[1]*h[1];
double Vh1 = V[2]*h[0] + V[3]*h[1];

double Vh_u0 = Vh0 + u[0];
double Vh_u1 = Vh1 + u[1];

double mT_Vh_u = m[0]*Vh_u0 + m[1]*Vh_u1;

double hTVh = h[0]*Vh0 + h[1]*Vh1;
double two_uTh = 2.0*(u[0]*h[0] + u[1]*h[1]);
double g = hTVh + two_uTh + f;

double disc = mT_Vh_u*mT_Vh_u - g*mTVm;
```

Code - C

```
    if (disc > 1e-12) {
        double r = sqrt(disc);
        kappa_out[0] = (-mT_Vh_u + r) / mTVm;
        kappa_out[1] = (-mT_Vh_u - r) / mTVm;
        x1[0] = h[0] + kappa_out[0]*m[0]; x1[1] = h[1] + kappa_out[0]*
            m[1];
        x2[0] = h[0] + kappa_out[1]*m[0]; x2[1] = h[1] + kappa_out[1]*
            m[1];
        *status = 2;
    } else if (fabs(disc) <= 1e-12) {
        kappa_out[0] = kappa_out[1] = (-mT_Vh_u) / mTVm;
        x1[0] = x2[0] = h[0] + kappa_out[0]*m[0];
        x1[1] = x2[1] = h[1] + kappa_out[0]*m[1];
        *status = 1;
    } else {
        *status = 0;}
}
```

Code - Python(with shared C code)

The code to obtain the required plot is

```
import ctypes as ct
import numpy as np
import matplotlib.pyplot as plt
# --- load the shared library (same folder) ---
lib = ct.CDLL("./libsimple_conic.so")
# tell ctypes what arguments the C function expects
lib.intersect_line_conic.argtypes = [
    ct.POINTER(ct.c_double), # V[4]
    ct.POINTER(ct.c_double), # u[2]
    ct.c_double, # f
    ct.POINTER(ct.c_double), # h[2]
    ct.POINTER(ct.c_double), # m[2]
    ct.POINTER(ct.c_double), # kappa[2] out
    ct.POINTER(ct.c_double), # x1[2] out
    ct.POINTER(ct.c_double), # x2[2] out
    ct.POINTER(ct.c_int) # status out
```

Code - Python(with shared C code)

```
]
lib.intersect_line_conic.restype = None

# ---- helper: convert quadratic ax^2+bx+c into conic form ----
def quadratic_to_conic(a, b, c):
    V = np.array([a, 0.0, 0.0, 0.0], dtype=np.double) # [[a,0],[0,0]]
    u = np.array([b/2.0, 0.0], dtype=np.double) # so 2u^T x = b x
    f = np.double(c)
    return V, u, f

# ---- our quadratic: x^2 - 3x - 10 = 0 ----
a, b, c = 1.0, -3.0, -10.0
V, u, f = quadratic_to_conic(a, b, c)

# Intersect with x-axis (y=0): line x = h + k m
h = np.array([0.0, 0.0], dtype=np.double)
m = np.array([1.0, 0.0], dtype=np.double)
```

Code - Python(with shared C code)

```python
# Outputs (allocated for C)
kappa = np.zeros(2, dtype=np.double)
x1 = np.zeros(2, dtype=np.double)
x2 = np.zeros(2, dtype=np.double)
status = ct.c_int(0)
# ---- call the C function ----
lib.intersect_line_conic(
    V.ctypes.data_as(ct.POINTER(ct.c_double)),
    u.ctypes.data_as(ct.POINTER(ct.c_double)),
    ct.c_double(f),
    h.ctypes.data_as(ct.POINTER(ct.c_double)),
    m.ctypes.data_as(ct.POINTER(ct.c_double)),
    kappa.ctypes.data_as(ct.POINTER(ct.c_double)),
    x1.ctypes.data_as(ct.POINTER(ct.c_double)),
    x2.ctypes.data_as(ct.POINTER(ct.c_double)),
    ct.byref(status)
)
```

Code - Python(with shared C code)

```
# Collect results
roots = []
if status.value >= 1:
    roots.append(float(x1[0]))
    if status.value == 2:
        roots.append(float(x2[0]))
roots.sort()
print("status:", status.value)
print("roots:", roots) # expected [-2.0, 5.0]

# ---- plot parabola and roots ----
xs = np.linspace(-10, 10, 600) # simpler fixed range
ys = a*xs*xs + b*xs + c
```

Code - Python(with shared C code)

```
plt.figure()
plt.plot(xs, ys, label=rf"$y={a:.0f}x^2{b:+.0f}x{c:+.0f}$")
plt.axhline(0, linestyle="--", linewidth=1, label="x-axis")
for r in roots:
    plt.scatter([r], [0.0], s=60, zorder=3, label=f"root {r:g}")
plt.xlabel("x"); plt.ylabel("y")
plt.title("Roots via line-conic intersection (NumPy + C)")
plt.grid(True)
plt.legend()
plt.savefig("parabola.png")
plt.show()
```

Code - Python only

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

# ------------ Helpers ------------

def quadratic_to_conic(a, b, c):
    V = np.array([[a, 0.0],
                  [0.0, 0.0]], dtype=float)
    u = np.array([b/2.0, 0.0], dtype=float) # so that 2 u^T x = b x
    f = float(c)
    return V, u, f

def line_conic_intersection(V, u, f, h, m, eps=1e-12):
    # m^T V m
    Vm = V @ m
    mTVm = float(m @ Vm)
```

Code - Python only

```
# Vh + u
Vh_u = V @ h + u
# m^T (Vh + u)
mT_Vh_u = float(m @ Vh_u)
# g(h) = h^T V h + 2 u^T h + f
g = float(h @ (V @ h) + 2.0 * (u @ h) + f)
disc = mT_Vh_u**2 - g * mTVm
if disc > eps:
    r = np.sqrt(disc)
    k1 = (-mT_Vh_u + r) / mTVm
    k2 = (-mT_Vh_u - r) / mTVm
    X1 = h + k1 * m
    X2 = h + k2 * m
    return 2, np.array([k1, k2], dtype=float), np.vstack([X1, X2])
```

Code - Python only

```
    elif abs(disc) <= eps:
        k = (-mT_Vh_u) / mTVm
        X = h + k * m
        return 1, np.array([k, k], dtype=float), np.vstack([X, X])
    else:
        return 0, np.array([np.nan, np.nan]), np.array([[np.nan, np.nan],
                                                        [np.nan, np.
                                                          nan]])


# ----------- Problem setup -----------

# Given quadratic: x^2 - 3x - 10 = 0
a, b, c = 1.0, -3.0, -10.0

# Conic parameters (V, u, f)
V, u, f = quadratic_to_conic(a, b, c)
```

Code - Python only

```
# x-axis as the line: y = 0 -> h = (0,0), m = (1,0)
h = np.array([0.0, 0.0], dtype=float)
m = np.array([1.0, 0.0], dtype=float)

# ------------ Solve via line-conic intersection ------------
status, kappa, X = line_conic_intersection(V, u, f, h, m)

# Roots are the x-coordinates of intersection points with y=0
roots = []
if status >= 1:
    roots = sorted([float(X[0, 0]), float(X[1, 0])]) if status == 2 else [
        float(X[0, 0])]

print("Status (2:two real, 1:tangent, 0:none):", status)
print("kappa values:", kappa)
print("Intersection points (x,y):\n", X)
print("Roots (x-intercepts):", roots)
```

Code - Python only

```
# ----------- Plot -----------
xs = np.linspace(-10, 10, 600)
ys = a*xs**2 + b*xs + c

plt.figure()
plt.plot(xs, ys, label=rf"$y={a:.0f}x^2{b:+.0f}x{c:+.0f}$")
plt.axhline(0, linestyle="--", linewidth=1, label="x-axis")

if status >= 1:
    plt.scatter([X[0,0]], [0.0], s=60, zorder=3, label=f"root-{X[0,0]:g}")
    if status == 2:
        plt.scatter([X[1,0]], [0.0], s=60, zorder=3, label=f"root-{X[1,0]:g
            }")
```

Code - Python only

```
plt.xlabel("x"); plt.ylabel("y")
plt.title("Roots via line—conic intersection (vectors & matrices)")
plt.grid(True)
plt.legend()
plt.savefig("newparabola.png")
plt.show()
```