

## 7.4.33

Harsha-EE25BTECH11026

September 8,2025

# Question

A circle  $C$  of radius 1 unit is inscribed in an equilateral triangle  $PQR$ . The points of contact of  $C$  with sides  $PQ$ ,  $QR$ ,  $RP$  are  $D$ ,  $E$ ,  $F$  respectively. The line  $PQ$  is given by the equation  $\sqrt{3}x + y - 6 = 0$  and the point  $D$  is  $\left(\frac{3\sqrt{3}}{2}, \frac{3}{2}\right)$ . Further, it is given that the origin and the centre of  $C$  are on same side of line  $PQ$ . The equation of circle  $C$  is

- ①  $(x - 2\sqrt{3})^2 + (y - 1)^2 = 1$
- ②  $(x - 2\sqrt{3})^2 + \left(y + \frac{1}{2}\right)^2 = 1$
- ③  $(x - \sqrt{3})^2 + (y - 1)^2 = 1$
- ④  $(x - \sqrt{3})^2 + (y + 1)^2 = 1$

# Theoretical Solution

According to the question,

$$\text{Equation of tangent PQ : } \mathbf{n}^T \mathbf{x} = c \quad (1)$$

$$\text{where } \mathbf{n} = \begin{pmatrix} \sqrt{3} & 1 \end{pmatrix}^T \text{ and } c = 6$$

$$\text{Point of tangency (D) : } \begin{pmatrix} \frac{3\sqrt{3}}{2} \\ \frac{3}{2} \end{pmatrix} \quad (2)$$

$$\text{radius}(r) = 1 \quad (3)$$

# Theoretical Solution

As the point of tangency  $\mathbf{D}$  and centre of circle  $\mathbf{u}$  are along the direction of the vector  $\mathbf{n}$ ,

$$\therefore \mathbf{D} - \mathbf{u} = \lambda \mathbf{n}, \text{ for some scalar } \lambda \quad (4)$$

$$\implies \mathbf{u} = \mathbf{D} - \lambda \mathbf{n} \quad (5)$$

Also,

$$\frac{|\mathbf{n}^T \mathbf{u} - c|}{\|\mathbf{n}\|} = r \quad (6)$$

$$|\mathbf{n}^T \mathbf{u} - c| = r \|\mathbf{n}\| \quad (7)$$

$$\mathbf{n}^T \mathbf{u} = c \pm r \|\mathbf{n}\| \quad (8)$$

# Theoretical Solution

To decide the sign, we need to use the fact that the origin and centre of circle are on the same side of the line PQ.

$$\therefore (\mathbf{n}^\top \mathbf{u} - c) \left( \mathbf{n}^\top \begin{pmatrix} 0 \\ 0 \end{pmatrix} - c \right) > 0 \quad (9)$$

$$\implies \mathbf{n}^\top \mathbf{u} < c \quad (10)$$

$$\therefore \mathbf{n}^\top \mathbf{u} = c - r\|\mathbf{n}\| \quad (11)$$

Substituting value of  $\mathbf{u}$ ,

$$\mathbf{n}^\top (\mathbf{D} - \lambda \mathbf{n}) = c - r\|\mathbf{n}\| \quad (12)$$

$$\implies \lambda = \frac{\mathbf{n}^\top \mathbf{D} + r\|\mathbf{n}\| - c}{\mathbf{n}^\top \mathbf{n}} \quad (13)$$

$$\mathbf{u} = \mathbf{D} - \frac{\mathbf{n}^\top \mathbf{D} + r\|\mathbf{n}\| - c}{\mathbf{n}^\top \mathbf{n}} \mathbf{n} \quad (14)$$

Substituting the values,

$$\therefore \mathbf{u} = \begin{pmatrix} \frac{3\sqrt{3}}{2} \\ \frac{3}{2} \end{pmatrix} - \frac{1}{2} \begin{pmatrix} \sqrt{3} \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{3} \\ 1 \end{pmatrix} \quad (15)$$

$$\therefore \text{Required equation of circle : } \|\mathbf{x}\|^2 - 2 \begin{pmatrix} \sqrt{3} & 1 \end{pmatrix} \mathbf{x} + 3 = 0 \quad (16)$$

# C Code -Finding the equation of circle

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

// structure for 2D point
typedef struct {
    double x, y;
} Point;

// normalize vector
void normalize(double *x, double *y) {
    double norm = sqrt((*x)*(*x) + (*y)*(*y));
    if (norm > 1e-12) {
        *x /= norm;
        *y /= norm;
    }
}
```

# C Code -Finding the equation of circle

```
// rotate by theta
Point rotate(Point v, double theta) {
    Point r;
    double c = cos(theta), s = sin(theta);
    r.x = c*v.x - s*v.y;
    r.y = s*v.x + c*v.y;
    return r;
}

// solve 2x2 system
Point intersect(double a1,double b1,double c1, double a2,double
    b2,double c2) {
    double det = a1*b2 - a2*b1;
    Point P;
    P.x = (b1*c2 - b2*c1)/det;
    P.y = (c1*a2 - c2*a1)/det;
    return P;
}
```



## C Code -Finding the equation of circle

```
// main solver: fills arrays with results
void solve_geometry(double *results) {
    // Given: line PQ:  $\sqrt{3}x + y - 6 = 0$ 
    double a = sqrt(3.0), b = 1.0, c = -6.0;
    Point D = {3*sqrt(3.0)/2.0, 1.5};
    double r = 1.0;
    // unit normal
    double nx=a, ny=b;
    normalize(&nx,&ny);
    // candidate centers
    Point C1 = {D.x + r*nx, D.y + r*ny};
    Point C2 = {D.x - r*nx, D.y - r*ny};
    // check side with origin
    double origin_side = a*0 + b*0 + c;
    double side1 = a*C1.x + b*C1.y + c;
    Point O = (side1*origin_side > 0) ? C1 : C2;
```

## C Code -Finding the equation of circle

```
// tangency vectors
Point vD = {D.x - O.x, D.y - O.y};
Point vE = rotate(vD, 2*M_PI/3.0);
Point vF = rotate(vD, -2*M_PI/3.0);

Point E = {O.x + vE.x, O.y + vE.y};
Point F = {O.x + vF.x, O.y + vF.y};

// tangent lines: ax+by+c=0
double cPQ = -(vD.x*D.x + vD.y*D.y);
double cQR = -(vE.x*E.x + vE.y*E.y);
double cRP = -(vF.x*F.x + vF.y*F.y);

// vertices
Point P = intersect(vD.x, vD.y, cPQ, vF.x, vF.y, cRP);
Point Q = intersect(vD.x, vD.y, cPQ, vE.x, vE.y, cQR);
Point R = intersect(vE.x, vE.y, cQR, vF.x, vF.y, cRP);
```

## C Code -Finding the equation of circle

```
// fill results array (order: O,D,E,F,P,Q,R)
results[0]=O.x; results[1]=O.y;
results[2]=D.x; results[3]=D.y;
results[4]=E.x; results[5]=E.y;
results[6]=F.x; results[7]=F.y;
results[8]=P.x; results[9]=P.y;
results[10]=Q.x; results[11]=Q.y;
results[12]=R.x; results[13]=R.y;
}
```

# Python+C code

```
import ctypes
import numpy as np
import math as m
import matplotlib.pyplot as plt

lib = ctypes.CDLL("./libcircle_solver.so")
# prepare result array (14 doubles: O,D,E,F,P,Q,R)
results = (ctypes.c_double * 14)()
lib.solve_geometry(results)

vals = np.array(results)
O = vals[0:2]
D = vals[2:4]
E = vals[4:6]
F = vals[6:8]
P = vals[8:10]
Q = vals[10:12]
R = vals[12:14]
```

# Python+C code

```
# circle parameters
h, k, r = 0[0], 0[1], 1.0
print("Equation of circle:")
print(f"(x-{h:.2f})^2+(y-{k:.2f})^2={r}")

# --- Plot ----
theta_vals = np.linspace(0, 2*np.pi, 600)
xc = 0[0] + r * np.cos(theta_vals)
yc = 0[1] + r * np.sin(theta_vals)

plt.figure(figsize=(7,7))
# incircle
plt.plot(xc, yc, label="Incircle (r=1)")

# triangle edges
tri_x = [P[0], Q[0], R[0], P[0]]
tri_y = [P[1], Q[1], R[1], P[1]]
plt.plot(tri_x, tri_y, 'm--', linewidth=1.8, label="Equilateral Triangle")
```

```
# --- Reconstruct lines PQ, QR, RP ---
# PQ given:  $\sqrt{3}x + y - 6 = 0$ 
a, b, c = np.sqrt(3), 1.0, -6.0
xvals = np.linspace(min(P[0],Q[0],R[0],O[0]) - 3,
                    max(P[0],Q[0],R[0],O[0]) + 3, 400)
yvals_pq = -(a*xvals + c)/b
plt.plot(xvals, yvals_pq, 'g-', label="Given side PQ")

# lines QR, RP from tangency points
def line_from_points(A, B):
    # equation through points A,B :  $(y_2-y_1)x - (x_2-x_1)y + (x_2-x_1)y_1 - (y_2-y_1)x_1 = 0$ 
    x1,y1 = A; x2,y2 = B
    a = y1 - y2
    b = x2 - x1
    c = x1*y2 - x2*y1
    return np.array([a,b,c])
```

```
def plot_line(L, style, label):
    aL,bL,cL = L
    if abs(bL) > 1e-8:
        y = -(aL*xvals + cL)/bL
        plt.plot(xvals, y, style, label=label)
    else:
        plt.axvline(-cL/aL, linestyle=style, label=label)

L_QR = line_from_points(Q,R)
L_RP = line_from_points(R,P)
plot_line(L_QR, 'k--', 'QR (tangent E)')
plot_line(L_RP, 'c--', 'RP (tangent F)')
# points
plt.scatter([D[0],E[0],F[0]], [D[1],E[1],F[1]],
            c=['red','orange','purple'], zorder=5, label='Tangency
            points D,E,F')
plt.scatter(O[0], O[1], c='black', s=40, label='Center O')
```

# Python+C code

```
plt.scatter([P[0],Q[0],R[0]],[P[1],Q[1],R[1]], c='blue', s=30,
            label='Vertices P,Q,R')
plt.scatter(0,0, c='green', s=40, label="Origin (0,0)")
# --- Labels ---
def add_label(pt, text, dx=0.15, dy=0.15):
    plt.text(pt[0]+dx, pt[1]+dy, text, fontsize=11, fontweight='
            bold',
            bbox=dict(facecolor='white', edgecolor='none', alpha
                    =0.7))

add_label(P, "P")
add_label(Q, "Q")
add_label(R, "R")
add_label(D, "D")
add_label(E, "E")
add_label(F, "F")
add_label(O, "O", dx=0.2, dy=-0.2)
add_label([0,0], "Origin", dx=0.2, dy=-0.2)
```



```
# axes
plt.axhline(0, color='gray', linewidth=1) # x-axis
plt.axvline(0, color='gray', linewidth=1) # y-axis

plt.xlim(min(P[0],Q[0],R[0], 0[0]) - 3, max(P[0],Q[0],R[0], 0[0])
          + 3)
plt.ylim(min(P[1],Q[1],R[1], 0[1]) - 3, max(P[1],Q[1],R[1], 0[1])
          + 3)
plt.gca().set_aspect("equal", adjustable="box")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.title("Equilateral triangle PQR with incircle C and tangency
          points D,E,F")
plt.savefig("/home/user/Matrix Theory: workspace/
            Matgeo_assignments/7.4.33/figs/figure_1.png")
plt.show()
```

# Python code

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mp
mp.use("TkAgg")

a, b, c = np.sqrt(3), 1.0, -6.0
D = np.array([3*np.sqrt(3)/2.0, 3.0/2.0])

def unit(v):
    n = np.linalg.norm(v)
    return v / n if n != 0 else v

def line_from_normal_and_point(nvec, P):
    # a x + b y + c = 0 with normal nvec = [a,b]; c = -(a*Px + b*Py)
    a,b = nvec[0], nvec[1]
    c = -(a*P[0] + b*P[1])
    return np.array([a,b,c])
```

```
def intersect(L1, L2):
    A = np.array([[L1[0], L1[1]], [L2[0], L2[1]]])
    B = -np.array([L1[2], L2[2]])
    return np.linalg.solve(A, B)

def rotate(v, theta):
    c,s = np.cos(theta), np.sin(theta)
    R = np.array([[c,-s],[s,c]])
    return R @ v

n = np.array([a,b])
n_unit = unit(n)

# two candidate centers:
C1 = D + r * n_unit
C2 = D - r * n_unit
```

# Python code

```
# Choose centre on same side of PQ as origin
def side_sign(point):
    return np.sign(a*point[0] + b*point[1] + c)
origin_side = side_sign(np.array([0.0,0.0]))
if side_sign(C1) == origin_side:
    O = C1
else:
    O = C2

h,k = O[0], O[1]

vD = D - O # vector from centre to D
theta = 2*np.pi/3.0 # 120 degrees
vE = rotate(vD, theta)
vF = rotate(vD, -theta)

E = O + vE
F = O + vF
```

# Python code

```
assert np.allclose(np.linalg.norm(vD), r, atol=1e-8)
assert np.allclose(np.linalg.norm(vE), r, atol=1e-8)
assert np.allclose(np.linalg.norm(vF), r, atol=1e-8)

L_PQ_from_tangent = line_from_normal_and_point(vD, D)
scale_given = np.sqrt(a*a + b*b)
L_PQ_normed = L_PQ_from_tangent / scale_given

L_QR = line_from_normal_and_point(vE, E)
L_RP = line_from_normal_and_point(vF, F)
# side names: PQ (tangent at D), QR (tangent at E), RP (tangent
# at F)
P = intersect(L_PQ_from_tangent, L_RP) # P = PQ  RP
Q = intersect(L_PQ_from_tangent, L_QR) # Q = PQ  QR
R = intersect(L_QR, L_RP) # R = QR  RP

u_vec = np.array([h,k])
f_const = h*h + k*k - r*r
```

```
print("\nExpanded scalar form:")
print(f"x^2 + y^2 - 2*{h:.2g}*x - 2*{k:.2g}*y + {f_const:.2g} = 0")
# Print the three tangent lines in normalized readable form
def pretty_line(L):
    # scale to unit normal for readability
    nrm = np.linalg.norm(L[:2])
    La = L / nrm
    a_s, b_s, c_s = La[0], La[1], La[2]
    return f"{a_s:.12g} x + {b_s:.12g} y + {c_s:.12g} = 0"
```

# Python code

```
#plot
theta_vals = np.linspace(0, 2*np.pi, 600)
xc = 0[0] + r * np.cos(theta_vals)
yc = 0[1] + r * np.sin(theta_vals)
plt.figure(figsize=(7,7))
# incircle
plt.plot(xc, yc, label="Incircle (r=1)")
# triangle edges
tri_x = [P[0], Q[0], R[0], P[0]]
tri_y = [P[1], Q[1], R[1], P[1]]
plt.plot(tri_x, tri_y, 'm--', linewidth=1.8, label="Equilateral
Triangle")
# tangency lines (extended for visibility)
xlim_min = min(P[0],Q[0],R[0], 0[0]) - 3
xlim_max = max(P[0],Q[0],R[0], 0[0]) + 3
xvals = np.linspace(xlim_min, xlim_max, 400)
```

```
# PQ from given coeffs
yvals_pq = -(a*xvals + c)/b
plt.plot(xvals, yvals_pq, 'g-', label="Given side PQ")
# ---- function to plot any line ----
def plot_line(L, style, label):
    aL,bL,cL = L
    if abs(bL) > 1e-8:
        y = -(aL*xvals + cL)/bL
        plt.plot(xvals, y, style, label=label)
    else:
        xconst = -cL/aL
        plt.axvline(x=xconst, linestyle=style, label=label)

# tangent lines via their normals
plot_line(L_QR, 'k--', 'QR (tangent E)')
plot_line(L_RP, 'c--', 'RP (tangent F)')
```



```
# points
plt.scatter([D[0], E[0], F[0]], [D[1], E[1], F[1]],
            c=['red', 'orange', 'purple'], zorder=5, label='Tangency
            points D,E,F')
plt.scatter(O[0], O[1], c='black', s=40, label='Center O')
plt.scatter([P[0],Q[0],R[0]], [P[1],Q[1],R[1]], c='blue', s=30,
            label='Vertices P,Q,R')
plt.scatter(0,0, c='green', s=40, label="Origin (0,0)")
# --- Add labels near points ---
def add_label(pt, text, dx=0.1, dy=0.1):
    plt.text(pt[0]+dx, pt[1]+dy, text, fontsize=11, fontweight='
            bold',
            bbox=dict(facecolor='white', edgecolor='none', alpha
            =0.7))
```

# Python code

```
add_label(P, "P")
add_label(Q, "Q")
add_label(R, "R")
add_label(D, "D")
add_label(E, "E")
add_label(F, "F")
add_label(0, "0", dx=0.2, dy=-0.2)
add_label([0,0], "Origin", dx=0.2, dy=-0.2)
# axes
plt.axhline(0, color='gray', linewidth=1) # x-axis
plt.axvline(0, color='gray', linewidth=1) # y-axis
```

```
plt.xlim(xlim_min, xlim_max)
plt.ylim(min(P[1],Q[1],R[1], O[1]) - 3, max(P[1],Q[1],R[1], O[1])
        + 3)
plt.gca().set_aspect("equal", adjustable="box")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.title("Equilateral triangle PQR with incircle C and tangency
        points D,E,F")
plt.savefig("/home/user/Matrix Theory: workspace/
        Matgeo_assignments/7.4.33/figs/Figure_1.png")
plt.show()
```

