Some Math Tricks

KochiTechGroup

Introduction to Complex Numbers

Complex Numbers

Definition of Complex Numbers

A complex number is written as:

$$z = a + bi$$

where:

- a is the **real part.**
- b is the **imaginary part.**
- i is the imaginary unit ($i^2 = -1$).

Complex Numbers – Addition and Substraction

Addition and Subtraction

For two complex numbers:

$$z_1 = a + bi, \quad z_2 = c + di$$

Addition

$$z_1 + z_2 = (a + c) + (b + d)i$$

Each component is added separately.

Subtraction

$$z_1 - z_2 = (a - c) + (b - d)i$$

Each component is subtracted separately.

Complex Numbers – Multiplication

Multiplication

Multiplication follows the distributive property:

$$z_1 \cdot z_2 = (a + bi) \cdot (c + di)$$

Expanding:

$$(ac - bd) + (ad + bc)i$$

Remember, $i^2 = -1$, which simplifies the result.

Complex Numbers - Division

Division

To divide two complex numbers, multiply the numerator and denominator by the conjugate of the denominator:

$$\frac{z_1}{z_2} = \frac{(a + bi)}{(c + di)}$$

Multiply by $\frac{c - di}{c - di}$ (the conjugate of z_2):

$$\frac{(a + bi)(c - di)}{c^2 + d^2}$$

Expanding:

$$\frac{(ac + bd) + (bc - ad)i}{c^2 + d^2}$$

Complex Numbers – Modulus and Conjugate

- **Modulus (Absolute Value)**

The modulus of a complex number is:

$$|z| = \sqrt{a^2 + b^2}$$

It represents the magnitude (distance from the origin).

- **Conjugate**

The conjugate of $z = a + bi$ is:

$$z = a - bi$$

This is useful in division and simplifying expressions.

Complex Numbers – Polar Form

Polar Form

A complex number can be written in polar form:

$$z = re^{i\theta} \quad \text{or} \quad z = r(\cos\theta + i\sin\theta)$$

where:

- $r = |z| = \sqrt{a^2 + b^2}$.
- $\theta = \tan^{-1}(b/a)$ (argument).

A Simple Complex class

```
class Complex {
    private final double real;
    private final double imaginary;
    public Complex(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }
    public double getReal() { return real; }
    public double getImaginary() { return imaginary;}
    public Complex add(Complex other) {
        return new Complex(this.real + other.real,
                this.imaginary + other.imaginary);
    }
    public Complex multiply(Complex other) {
        double realPart = this.real * other.real - this.imaginary * other.imaginary;
        double imaginaryPart = this.real * other.imaginary + this.imaginary * other.real;
        return new Complex(realPart, imaginaryPart);
    }
    public double modulusSquared() {
        return this.real * this.real + this.imaginary * this.imaginary;
    }
}
```

Euler’s Identity – Most beautiful equations in Mathematics

$$e^{i\pi} + 1 = 0$$

Euler’s Identity elegantly combines **five** of the most fundamental numbers in mathematics:

e – The base of natural logarithms, crucial in growth and decay models.

i – The imaginary unit, defined as $i^2 = -1$.

π – The ratio of a circle’s circumference to its diameter, fundamental in geometry.

1 – The multiplicative identity, the foundation of counting.

0 – The additive identity, representing nothingness.

Derivation of Euler’s Identity

$$e^{i\theta} = \cos\theta + i\sin\theta$$

Step-by-Step Derivation

1. Substituting $\theta = \pi$: $$e^{i\pi} = \cos\pi + i\sin\pi$$
2. Evaluating trigonometric values: $$\cos\pi = -1, \quad \sin\pi = 0$$
3. This simplifies to: $$e^{i\pi} = -1 + 0i$$
4. Adding 1 to both sides: $$e^{i\pi} + 1 = 0$$

Quadratic Formula

Quadratic Equation Form

A quadratic equation is generally written as:

$$ax^2 + bx + c = 0$$

The roots of the equation are found using the **quadratic formula**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If the **discriminant** $(b^2 - 4ac)$ is negative, the roots will be **complex**.

Quadratic Equation – Example #2

Example 1: $x^2 + 2x + 5 = 0$

Step 1: Identify coefficients

Here, we have:

- $a = 1$
- $b = 2$
- $c = 5$

Step 2: Compute Discriminant $\Delta = b^2 - 4ac = (2)^2 - 4(1)(5) = 4 - 20 = -16$

Since $\Delta < 0$, the roots are **complex**.

Step 3: Compute Square Root $\sqrt{-16} = 4i$

Step 4: Compute Roots $x = \dfrac{-2 \pm 4i}{2(1)}$

$$x = \frac{-2 \pm 4i}{2} = -1 \pm 2i$$

Final Answer: $x = -1 + 2i, \quad x = -1 - 2i$

Quadratic Equation – Example #2

$2x^2 + 4x + 7 = 0$

Step 1: Identify coefficients

- $a = 2$
- $b = 4$
- $c = 7$

Step 2: Compute Discriminant $\Delta = (4)^2 - 4(2)(7) = 16 - 56 = -40$

Since $\Delta < 0$, the roots are complex.

Step 3: Compute Square Root $\sqrt{-40} = \sqrt{40}\,i = 2\sqrt{10}\,i$

Step 4: Compute Roots $x = \dfrac{-4 \pm 2\sqrt{10}\,i}{4}$

$$x = -1 \pm \frac{\sqrt{10}\,i}{2}$$

Final Answer: $x = -1 + \dfrac{\sqrt{10}\,i}{2}, \quad x = -1 - \dfrac{\sqrt{10}\,i}{2}$

Elementary Transformation Matrices

Translation Matrices and Homogeneous Coordinates

Translation Matrix

Translation moves a point (x, y) by a given distance (t_x, t_y):

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Applying this to a point $(x, y, 1)$:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Scale Matrix

Scaling Matrix

Scaling changes the size of an object by factors s_x and s_y:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Applying scaling to a point:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix}$$

Rotation Matrices

Rotation Matrix

Rotation by an angle θ about the origin:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Applying rotation:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}$$

Transformation Matrices

General Transformation Matrix

Combining all transformations:

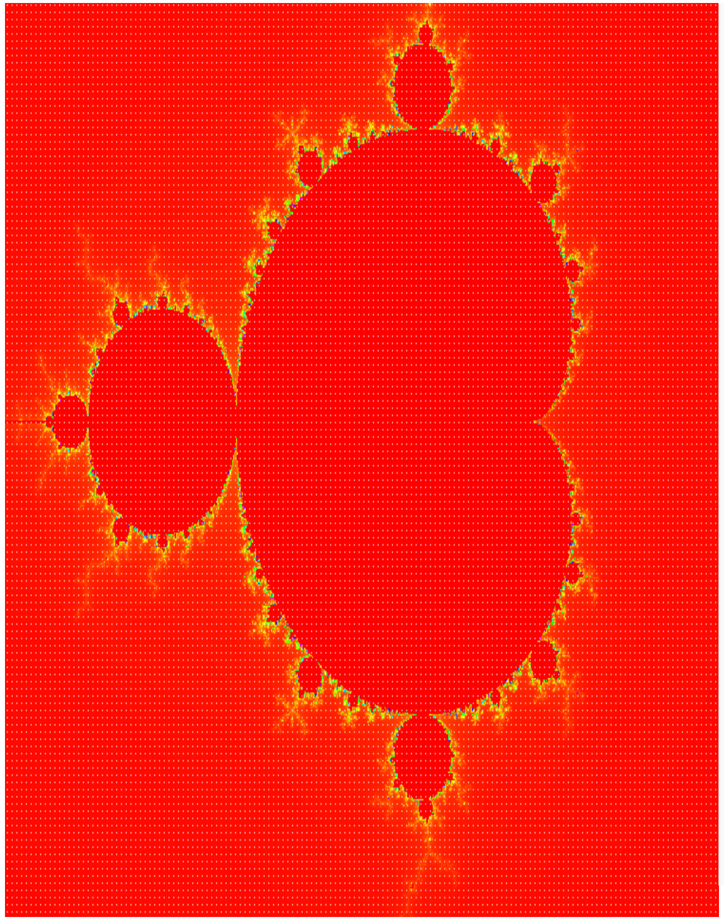
$$T = R \cdot S \cdot T = \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta & t_x \\ s_x \sin\theta & s_y \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix Class – a simple one

```
class Matrix {
    private final double[][] mat;
    public Matrix(double[][] values) { this.mat = values;}
    public static Matrix identity() {
        return new Matrix(new double[][]{
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        });
    }
    public static Matrix translation(double tx, double ty) {
        return new Matrix(new double[][]{
            {1, 0, tx},
            {0, 1, ty},
            {0, 0, 1}
        });
    }
    public static Matrix scaling(double sx, double sy) {
        return new Matrix(new double[][]{
            {sx, 0, 0},
            {0, sy, 0},
            {0, 0, 1}
        });
    }
```

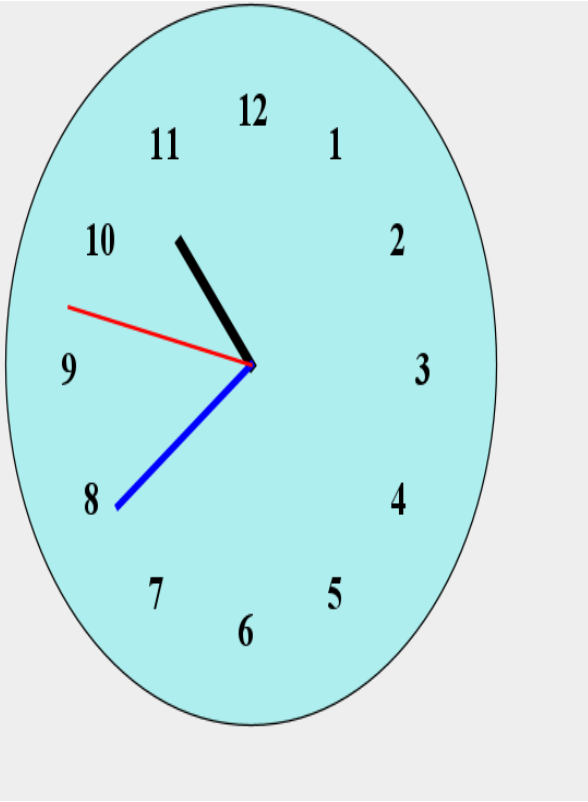
```
    public Matrix multiply(Matrix other) {
        double[][] result = new double[3][3];
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                for (int k = 0; k < 3; k++)
                    result[i][j] += this.mat[i][k] * other.mat[k][j];
        return new Matrix(result);
    }

    public double[] transformPoint(double x, double y) {
        double[] point = {x, y, 1};
        double[] result = new double[3];

        for (int i = 0; i < 3; i++) {
            result[i] = mat[i][0] * point[0] + mat[i][1] * point[1] + mat[i][2] * point[2];
        }
        return new double[]{result[0], result[1]}; // Return transformed (x,y)
    }
}
```

Application – MandelBrot Set with explicit co-ordinate computation



```java
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    for (int x = 0; x < WIDTH; x++) {
        for (int y = 0; y < HEIGHT; y++) {
            double real = (x - WIDTH / 2) / ZOOM;
            double imaginary = (y - HEIGHT / 2) / ZOOM;
            Complex c = new Complex(real, imaginary);
            int iter = mandelbrotIterations(c);

            // Color mapping based on iterations
            g.setColor(new Color(iter % 256, iter % 256, iter % 256));
            g.drawRect(x, y, 1, 1);
        }
    }
}
```

Analog Clock – With Trignometric Functions

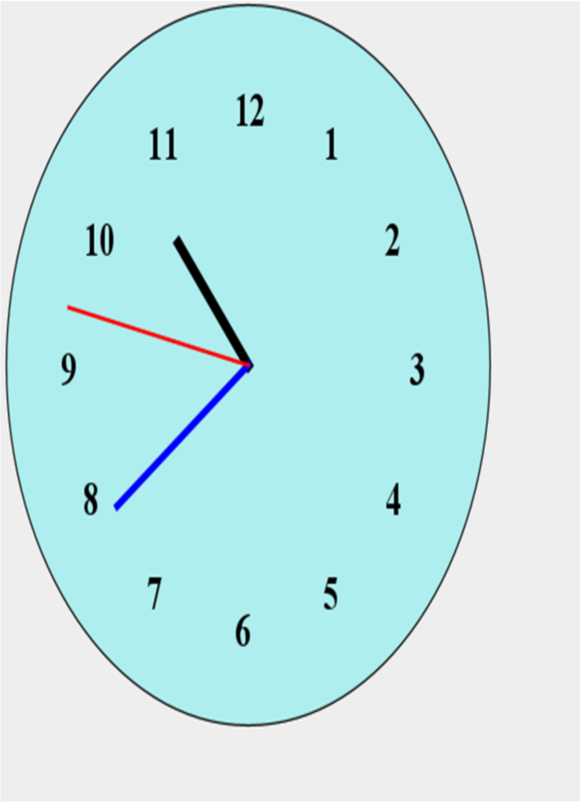



```java
    // Get current time
    Calendar now = Calendar.getInstance();
    int hour = now.get(Calendar.HOUR) % 12;
    int minute = now.get(Calendar.MINUTE);
    int second = now.get(Calendar.SECOND);

    // Convert time to angles
    double angleHour = Math.toRadians((hour + minute / 60.0) * 30 - 90);
    double angleMinute = Math.toRadians(minute * 6 - 90);
    double angleSecond = Math.toRadians(second * 6 - 90);

    // Draw clock hands
    drawHand(g2d, angleHour, CLOCK_RADIUS * 0.5, 6, Color.BLACK);
    drawHand(g2d, angleMinute, CLOCK_RADIUS * 0.75, 4, Color.BLUE);
    drawHand(g2d, angleSecond, CLOCK_RADIUS * 0.85, 2, Color.RED);
}

private void drawHand(Graphics2D g2d, double angle, double length, int thickness, Color color) {
    g2d.setColor(color);
    g2d.setStroke(new BasicStroke(thickness));
    int x = 200, y = 200;
    int xEnd = (int) (x + length * Math.cos(angle));
    int yEnd = (int) (y + length * Math.sin(angle));
    g2d.draw(new Line2D.Double(x, y, xEnd, yEnd));
}
```

A Matrix class with Affine Transformation

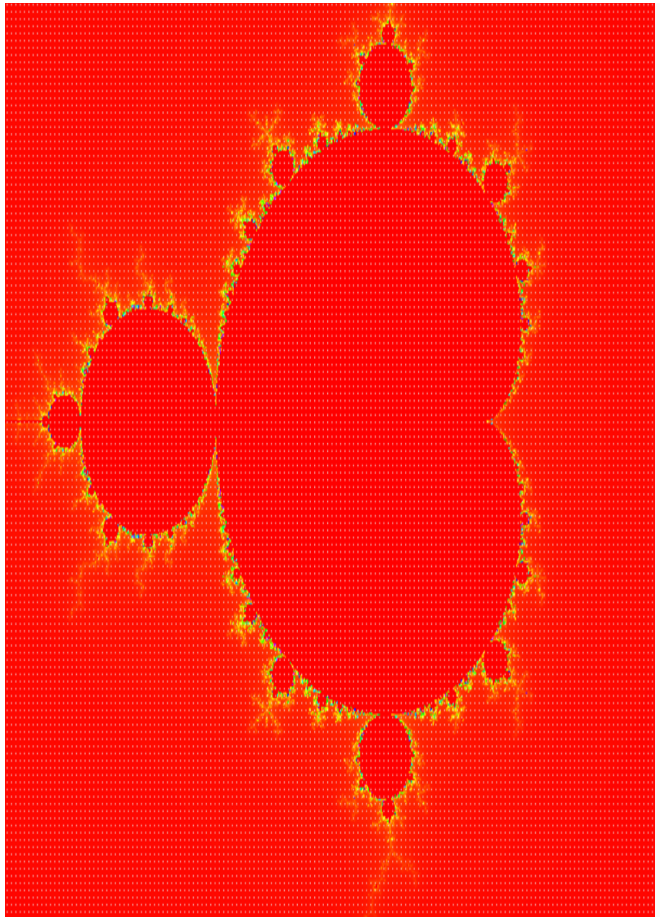
```java
class Matrix {
    private final double[][] mat;
    public Matrix(double[][] values) { this.mat = values;}
    public static Matrix identity() {
        return new Matrix(new double[][]{
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        });
    }
    public static Matrix translation(double tx, double ty) {
        return new Matrix(new double[][]{
            {1, 0, tx},
            {0, 1, ty},
            {0, 0, 1}
        });
    }
    public static Matrix scaling(double sx, double sy) {
        return new Matrix(new double[][]{
            {sx, 0, 0},
            {0, sy, 0},
            {0, 0, 1}
        });
    }
    public Matrix multiply(Matrix other) {
        double[][] result = new double[3][3];
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                for (int k = 0; k < 3; k++)
                    result[i][j] += this.mat[i][k] * other.mat[k][j];
        return new Matrix(result);
    }

    public double[] transformPoint(double x, double y) {
        double[] point = {x, y, 1};
        double[] result = new double[3];

        for (int i = 0; i < 3; i++) {
            result[i] = mat[i][0] * point[0] + mat[i][1] * point[1] + mat[i][2] * point[2];
        }
        return new double[]{result[0], result[1]}; // Return transformed (x,y)
    }
}
```

Analog Clock with Affine Transformation




```
    double angleHour = Math.toRadians((hour + minute / 60.0) * 30);
    double angleMinute = Math.toRadians(minute * 6);
    double angleSecond = Math.toRadians(second * 6);

    // Apply matrix-based transformations for clock hands
    drawHand(g2d, angleHour, CLOCK_RADIUS * 0.5, 6, Color.BLACK);
    drawHand(g2d, angleMinute, CLOCK_RADIUS * 0.75, 4, Color.BLUE);
    drawHand(g2d, angleSecond, CLOCK_RADIUS * 0.85, 2, Color.RED);
}

private void drawHand(Graphics2D g2d, double angle, double length, int thickness, Color color) {
    g2d.setColor(color);
    g2d.setStroke(new BasicStroke(thickness));

    // Transformation: Scaling, Rotation, Translation
    Matrix scale = Matrix.scaling(length, length);
    Matrix rotation = Matrix.rotation(angle - Math.PI / 2);
    Matrix translation = Matrix.translation(200, 200);
    Matrix transform = translation.multiply(rotation).multiply(scale);

    double[] endpoint = transform.transformPoint(1, 0);
    g2d.draw(new Line2D.Double(200, 200, endpoint[0], endpoint[1]));
}
```

MandelBrot with Affine Transformation



```
public MandelbrotRenderer2() {
    // Define transformation: scale and translate viewport
    Matrix scale = Matrix.scaling(1.0 / ZOOM, 1.0 / ZOOM);
    Matrix translate = Matrix.translation(-WIDTH / 2.0, -HEIGHT / 2.0);
    transformation = scale.multiply(translate);
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    for (int x = 0; x < WIDTH; x++) {
        for (int y = 0; y < HEIGHT; y++) {
            // Transform pixel coordinates to complex plane
            double[] transformed = transformation.transformPoint(x, y);
            Complex c = new Complex(transformed[0], transformed[1]);
            int iter = mandelbrotIterations(c);

            // Color mapping
            float hue = iter / (float) MAX_ITER;
            g.setColor(Color.getHSBColor(hue, 1, iter > 0 ? 1 : 0));
            g.drawRect(x, y, 1, 1);
        }
    }
}
```

Questions, if any