

1.00 Lecture 22

Matrices

Reading for next time: Numerical Recipes, pp. 37-42

Matrices

- Matrix is 2-D array of m rows by n columns

$$\begin{array}{|c|c|c|c|c|} \hline a_{00} & a_{01} & a_{02} & a_{03} \dots & a_{0n-1} \\ \hline a_{10} & a_{11} & a_{12} & a_{13} \dots & a_{1n-1} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} \dots & a_{2n-1} \\ \hline \dots & \dots & \dots & \dots \dots & \dots \\ \hline a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & a_{m-1,3} \dots & a_{m-1,n-1} \\ \hline \end{array}$$

- In math notation, we use index 1, ... m and 1, ... n.
- In Java, we usually use index 0, ... m-1 and 0, ...n-1
- They often represent a set of linear equations:

$$a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \dots + a_{0,n-1}x_{n-1} = b_0$$

$$a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \dots + a_{1,n-1}x_{n-1} = b_1$$

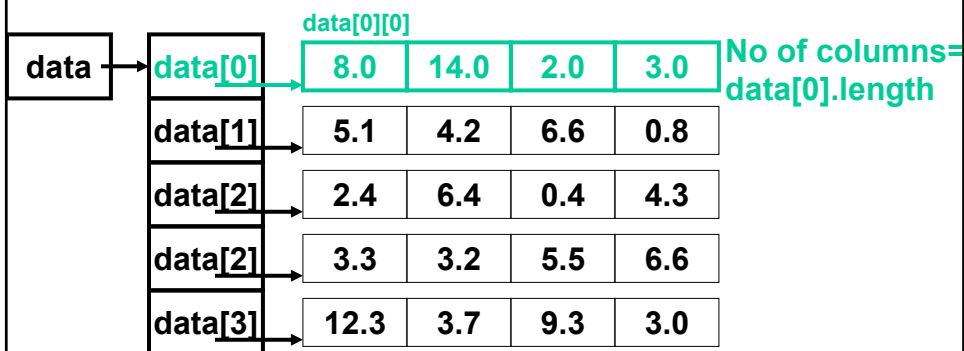
$$\dots$$
$$a_{m-1,0}x_0 + a_{m-1,1}x_1 + a_{m-1,2}x_2 + \dots + a_{m-1,n-1}x_{n-1} = b_{m-1}$$

- n unknowns x are related by m equations
- Coefficients a are known, as are right hand side b

Matrices, p.2

- In this lecture we cover basic matrix representation and manipulation
 - Used most often to prepare matrices for use in solving linear systems, which we cover in next lecture
- Java has 2-D arrays, declared as, for example
`double[][] squareMatrix= new double[5][5];`
 - But there are no built-in methods for them
- So, it's helpful to create a Matrix class:
 - Create methods to add, subtract, multiply, form identity matrix, etc.
- Sparse matrices are handled differently:
 - Almost all large matrices are extremely sparse (99%+ of entries are zeros)
 - Store (i, j, value) in a list or 1-D array or other data structure

2-D Arrays



No. of rows=
data.length

A 2-D array is:

a reference to a 1-D array of references to 1-D arrays of data.
This is how we'll store the matrix data in class Matrix

Matrix class, p.1

```
public class Matrix {
    private double[][] data;    // Reference to array

    public Matrix(int m, int n) {
        data = new double[m][n];
    }

    public void setIdentity() {
        int i,j;
        int nrows = data.length;
        int ncols = data[0].length;
        for(i=0; i<nrows; i++)
            for(j=0; j<ncols; j++)
                if(i == j)
                    data[i][j]= 1.0;
                else
                    data[i][j]= 0.0; // Not really needed
                                     // since initialized=0
    }
}
```

Matrix class, p.2

```
public int getNumRows() {
    return data.length; }
public int getNumCols() {
    return data[0].length; }
public double getElement(int i, int j) {
    return data[i][j]; }
public void setElement(int i, int j, double val) {
    data[i][j] = val; }

public Matrix addMatrices(Matrix b) {
    Matrix result = null;
    int nrows = data.length;
    int ncols = data[0].length;
    if (nrows==b.data.length && ncols==b.data[0].length) {
        result = new Matrix(nrows, ncols);
        for(int i=0; i<nrows; i++)
            for(int j=0; j<ncols; j++)
                result.data[i][j]= data[i][j]+ b.data[i][j]; }
    return result;
}
```

Matrix class, p.3

```
public Matrix multMatrices(Matrix b) {
    Matrix result = null;
    int nrows = data.length;
    int p = data[0].length;
    if (p == b.data.length) {
        result = new Matrix(nrows, b.data[0].length);
        for(int i=0; i<nrows; i++)
            for(int j=0; j<result.data[0].length; j++){
                double t = 0.0;
                for(int k=0; k<p; k++) {
                    t += data[i][k] * b.data[k][j];
                }
                result.data[i][j]= t;
            }
        return result;
    }
    public void print() {
        for(int i=0; i< data.length; i++) {
            for(int j=0; j< data[0].length; j++)
                system.out.print(data[i][j] + " ");
            system.out.println();
        }
        system.out.println();
    }
}
```

MatrixTest

```
public class MatrixTest {
    public static void main(String argv[]) {
        Matrix mat1 = new Matrix(3,3);
        Matrix mat2 = new Matrix(3,3);
        mat1.setIdentity();
        mat2.setIdentity();
        Matrix res;

        res = mat1.addMatrices(mat2);

        System.out.println("mat1:");
        mat1.print();
        System.out.println("mat2:");
        mat2.print();
        System.out.println("mat1 + mat2:");
        res.print();

        // Similar code for multiplication
        // Add your exercise code here
    }
}
```

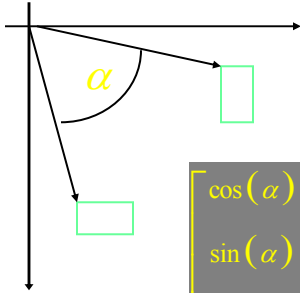
Exercise

- Download `Matrix` and `MatrixTest` from the Web site
- Write a method `multScalar()` to multiply a matrix by a scalar in class `Matrix`
- Invoke your method from your `MatrixTest` class' main method
- Hints for writing `multScalar()`
 - Use `addMatrix()` as a rough guide
 - Find the number of rows and columns in the matrix
 - Create a new `Matrix` object to return as the result
 - Loop through all entries (nested for loops) to multiply by the scalar
 - Return the result
- Modifying `MatrixTest` `main()` method:
 - Add a line to use the `multScalar()` method
 - Add another line to print the resulting matrix, using its `print()` method

Exercise 2

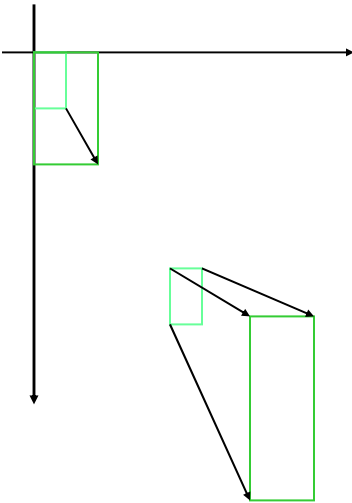
- Implement graphics transforms from last lecture
- Instead of using Java's `rotate()` and `scale()` methods, you'll create matrices to represent rotation and scaling, multiply them, and apply them to a shape.
- With some perseverance, your matrix manipulations will yield the same result as Java's methods!

Rotation



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

Scaling



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

Composing Transformations

- Suppose we want to scale point (x, y) by 2 and then rotate by 90 degrees.

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$$

rotate

scale

Composing Transformations, 2

Because matrix multiplication is associative, we can rewrite this as

$$\left(\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Exercise

- **Download TransformTest, TransformPanel**
 - TransformPanel rotates (by 18 °) and scales (by 2) a rectangle using Java affine transforms
 - Run it to see the result
- **Download TransformTest1, TransformPanel1**
 - These are skeletons for doing the rotations and scaling through matrix multiplication yourself. You will:
 - Create two matrices (rotate, scale) with your Matrix class
 - Scale by 2 in the x and y directions and rotate by 18° ($\pi/10$)
 - Look at the scaling and rotation matrices in previous slides
 - Multiply the 2 matrices, save them in Matrix result. Order matters!
 - Try it both ways if you aren't sure
 - Pass the values as arguments to AffineTransform() as shown in TransformPanel1 code on the next slides
 - See if your AffineTransform produces the same result!

Exercise

```
import java.awt.*;
import javax.swing.*;

public class TransformTest {
    public static void main(String args[]) {
        JFrame frame = new JFrame("Rectangle transform");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500,500);
        Container contentPane= frame.getContentPane();
        TransformPanel panel = new TransformPanel();
        contentPane.add(panel);
        frame.setVisible(true);
    }
}
```


Exercise

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;    // For 2D classes

public class TransformPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2= (Graphics2D) g;
        Rectangle2D rect= new Rectangle2D.Double(0, 0, 50, 100);

        g2.setPaint(Color.BLUE);
        AffineTransform baseXf = new AffineTransform();
        // Rotate by 18 degrees, then scale by 2 in x, y directions
        baseXf.rotate(Math.PI/10.0);
        baseXf.scale(2.0, 2.0);
        g2.transform(baseXf);
        g2.draw(rect);
    }
}
```

Replace these lines

Exercise

```
public class TransformPanel1 extends JPanel {
    public void paintComponent(Graphics g) {
        // Same initial lines: superclass, cast g2, new rectangle
        g2.setPaint(Color.MAGENTA);
        Matrix s = new Matrix(3, 3);
        // Set its elements to scale rectangle by 2
        // Your code here
        s.print();
        Matrix r = new Matrix(3, 3);    // Rotate
        double a = Math.PI / 10;        // 18 degree angle
        // Set elements to rotate 18 degrees; use Math.sin() and cos()
        // Your code here
        r.print();
        // Multiply r and s to get Matrix result
        // Your code here.
        result.print();
        double m00 = result.getElement(0, 0);
        double m01 = result.getElement(0, 1);
        // Etc. Coefficients inserted in COLUMN order. Done for you.
        AffineTransform baseXf =
            new AffineTransform(m00, m10, m01, m11, m02, m12);
        g2.transform(baseXf);    // Only 6 elements vary in xform
        g2.draw(rect);    } }
}
```

Exercise

