Lab 2

November 27th, 2017

Schedule

Date	Time	Room
November 27th, 2017	11:30 - 13:00	Q117 and Q120
November 27th, 2017	13:15 - 14:45	Q117 and Q120
December 1st, 2017	9:45 - 11:15	Q117 and Q120

Students assigned to Room Q120 - please, don't forget your notebooks!

Task 2.1 : Getting started

In the labs, you will use Eclipse for writing Java programs. If you have not worked on Lab 1, you should still have a running version of Eclipse. For easy user interaction, import the `SimpleIO` package. Follow the guidelines given in Task 1.1.

Task 2.2 : Modifying BMI calculation

In case you have not solved Task 1.6 from Lab 1 completely, use the code in Listing 1 to create a new program: This program reads weight and height of different persons from the user input to calculate the body mass index (BMI). Subsequent BMI values are summed up and divided by the number of specified persons `NUM_PERSONS` to obtain the average BMI.

Modify the program to support input data of three persons. No change of the number of persons during runtime is needed.

To ensure legal input values, two while loops have been added to check for a reasonable range of input values. The tallest man in the world was *Robert Wadlow* with a height of 2.72 m. Add a constant `MAX_HEIGHT=2.72` to the program and use it to modify the range check in order to reject values greater than specified in `MAX_HEIGHT`.

Task 2.3 : Storing values in an one-dimensional array

Instead of summing up values, we now want to store the BMI for each person separately. For that purpose, replace the double variable `bmi` by a suitable array `arr_bmi` to store the value for each person.

Once all values for the persons have been entered by the user, use your new array to calculate the average BMI and print it to the console.

```java
package de.tuhh.diss.apps;
import de.tuhh.diss.io.SimpleIO;

public class BmiCalculation {

  final public static int NUM_PERSONS = 2;

  public static void main(String[] args) {

    double weight;
    double height;
    double bmi = 0;

    for(int i = 0 ; i < NUM_PERSONS ; i++){

      SimpleIO.print("Enter weight in [kg] for person " + (i+1) + ":");
      weight = SimpleIO.readDouble();
      while (weight < 0) {
        SimpleIO.print("Enter non-negative value: ");
        weight = SimpleIO.readDouble();
      }

      SimpleIO.print("Enter height in [m] of person " + (i+1) + ":");
      height = SimpleIO.readDouble();
      while (height <= 0){
        SimpleIO.print("Height has to be greater "
          + "than zero, enter value in [m] again: ");
        height = SimpleIO.readDouble();
      }

      bmi += weight / (height*height);

    }

    SimpleIO.print("Their average BMI is: " + bmi / NUM_PERSONS);
  }
}
```

Listing 1: BMI calculation

Task 2.4 : Upper-bounding eigenvalues

Let λ be an eigenvalue of $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\|\mathbf{A}\|_F$ the Frobenius norm of the matrix $\mathbf{A}$, then

$$|\lambda|_F \leq \|\mathbf{A}\|_F \tag{1}$$

holds. Thus, for deriving an upper bound of the eigenvalues of $\mathbf{A}$ the calculation of the Frobenius norm of the matrix $\mathbf{A}$ can be used. The Frobenius norm of any matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined as:

$$\| \mathbf{A}\|_F := \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2}, \tag{2}$$

where a_{ij} are the elements in the matrix $\mathbf{A}$. In words, the Frobenius norm calculates the square root of the sum of squares of all elements in the matrix.

Create a new class `EigenvalueUpperBound` that lets the user define the entries of matrix $\mathbf{A} \in \mathbb{R}^{3 \times 3}$ with `SimpleIO.readDouble()`. The matrix is defined as

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tag{3}$$

and will be stored in a two-dimensional array `A`. The class also gives an upper bound on the magnitude of eigenvalues of $\mathbf{A}$ by calculating its Frobenius norm. Use two nested for-loops for accessing all rows i and all columns j in your matrix. Use a constant `MAX_MATRIXSIZE=3` to allocate enough memory and to avoid out-of-bound indexation of your array.

Think about how many for-loops you *really* need and optimize your program accordingly.

Hint: Once the user specified all entries of the matrix, you can use the methods provided by the Java `Math` library:

- `Math.sqrt(b)` to calculate the square root.
- `Math.pow(b, 2)` to calculate the square.

Note `b` might be a double variable declared first or any other double value.

Task 2.5 : Upper-bounding eigenvalues in methods

For good software engineering, it is recommended to separate user input and calculation. Thus, create a new method `obtainFrobeniusNorm()`, which accepts an array `matrixA` as input and returns a double value containing the Frobenius norm of the matrix. Call this method from `main()` and pass the user-entered matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ to the new method. Use the return value to generate the output text.

Hint: Use the following method.

```
1  public static double obtainFrobeniusNorm(double[][] matrixA) {
2    double frobeniusNorm = 0;
3    //Put your calculation here
4    return frobeniusNorm;
5  }
```

Task 2.6 : VO_2 max

In training theory, the maximal oxygen consumption (VO_2 max) is an important factor to assess your aerobic fitness. Research has shown that VO_2 max, maximum heart rate (HR_{max}) and resting heart rate (HR_{rest}) are related by the following equation:

$$VO_2\ \text{max} \approx 15 \times \frac{HR_{max}}{HR_{rest}} \quad [\text{mL}/(\text{kg}\,\text{min})] \tag{4}$$

HR_{max} and HR_{rest} mainly relate to the age of the person, via equation

$$HR_{max} \approx 206.9 - (0.67 \times \text{age}) \quad [1/\text{min}] \tag{5}$$

and by the given (rough) classification table:

age	18-35	36-55	56+
HR_{rest}	71	73	76

Write a class `MaxOxygen` that calculates VO_2 max based on the age of a person. Your program contains three methods for:

- VO_2 max calculation,
- HR_{max} calculation,
- HR_{rest} decision.

To ensure all methods are accessible by each other, declare them as `public static` and use a suitable return type. Write a method `main()`, which queries the age of the user and print its VO_2 max to the console.

Task 2.7 : Newton's method

Write a program that approximates the root ($f(x_r) = 0, x_r \in \mathbb{R}$) of the function:

$$f(x) := x^4 - 2x^3 - x^2 - 2x + 2\,. \tag{6}$$

The root of the function is approximated using the algorithm called "Newton's method", which works as follows:

Algorithm 1 Newton's method

while $f(x_{r,n}) \neq 0$ **do**
  $x_{r,n+1} \leftarrow x_{r,n} - f(x_{r,n})/\mathrm{d}f(x_{r,n})$
  $x_{r,n} \leftarrow x_{r,n+1}$
end while

Use an initial value $x_0 = 10$. Count the number of iterations your program needs and print it to the console. How many iterations do you need?

Is $f(x_{r,n}) \neq 0$ a useful choice to end the repetition loop of your program? Think of other exit statements and implement them accordingly.

Hint: Use the following implementations of the methods for $f(x)$ and its derivative.

```
1  /**
2  * @returns f(x) = x^4 - 2x^3 - x^2 - 2x + 2
3  */
4  public static double f(double x) {
5    return Math.pow(x, 4) - 2*Math.pow(x, 3) - Math.pow(x, 2) - 2*x + 2;
6  }
7
8  /**
9  * @returns f'(x) = 4x^3 - 6x^2 - 2x - 2
10 */
11 public static double df(double x) {
12   return 4*Math.pow(x, 3) - 6*Math.pow(x, 2) - 2*x - 2;
13 }
```