LSINF1113 : Exercices

Denis Mortier

Décembre 2016

Ce document reprends les réponses aux questions des exercices proposés pour chaque *lecture*. Malgré le soin apporté à sa réalisation, aucune garantie n'est apportée quant à l'exactitude des réponses et chacun est invité à aider à compléter ou corriger ce document pour le rendre meilleur. Le code LaTeXde ce document est disponible sur GitHub[1].

Les projets Java liés se trouvent dans le même dossier que ce fichier. Pour les toute petites classes indépendantes, elles ont toutes été rassemblées dans le projet "littleclasses". A nouveau, aucune garantie n'est apportée sur le fait que le code soit correct, encore moins sur le fait qu'il soit optimal. Ici aussi chacun peut aider à les améliorer.

1 Introduction and number representation

1.1 Question 1

$$\begin{aligned} 0.1 &= \frac{1}{2^5} + \frac{1}{2^6} + \frac{1}{2^8} + RE \\ &= 0.10156251 \end{aligned} \tag{1}$$

L'erreur est donc de $|\frac{0.1015625-0.1}{0.1}| = 0.015625$ soit une erreur de moins de 5 %.

Rien que pour représenter la partie décimale, il faut donc 8 bits. Cela ne tiens pas compte ni du bit pour le signe ni des bits pour représenter la partie entière.

1.2 Question 2

$$\begin{aligned} x &= [\underline{x}, \overline{x}] \\ y &= [\underline{y}, \overline{y}] \end{aligned} \tag{2}$$

On peut donc calculer la somme et le produit de x et y de cette manière :

$$\begin{aligned} x + y &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\ x \cdot y &= [\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}] \end{aligned} \tag{3}$$

1.3 Question 3

// TO DO

[1] https://github.com/Zibeline/LSINF1113-Exercices/

1.4 Question 4

La méthode main contient deux variables, `start` et `end` qui déterminent les bornes où évaluer la fonction. La variable `pointsToCalculate` détermine le nombre de points à calculer. Les points seront répartis de manière uniforme entre `start` et `end`.

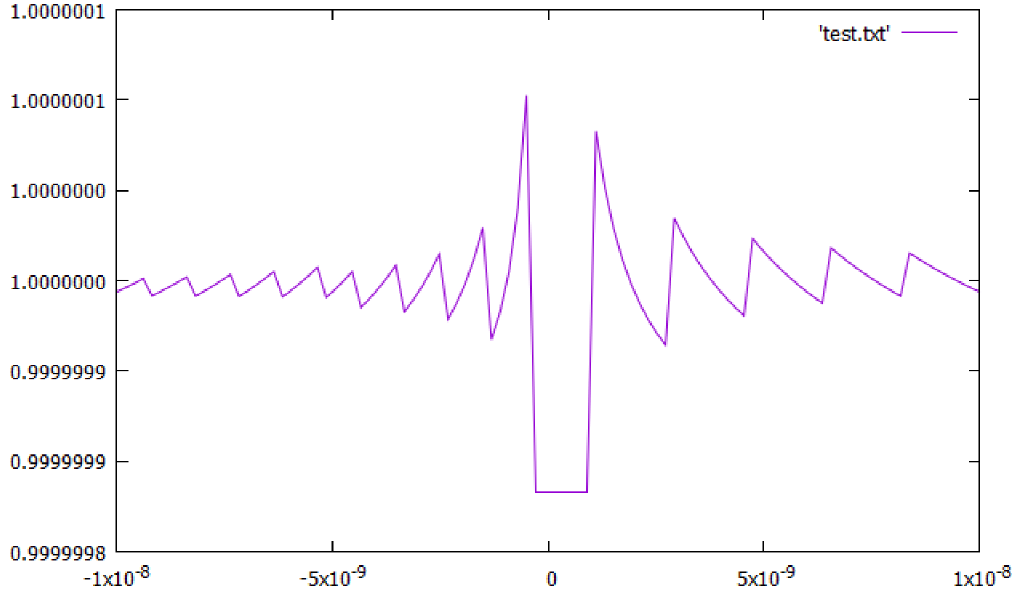
Le code se trouve dans le projet `littleclasses` dans la classe `expo.java`.

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
/**
 *
 * @author DenisM
 * @version December 2016
 */
public class expo {
    /**
     * Renvoie (e exp(x) - 1) / x
     */
    public static double expo(double x) {
        return (Math.exp(x)-1)/x;
    }

    public static void main(String[] args) {
        String filename = "test.txt";
        PrintStream stream = null;
        double start = Math.pow(10, -8)*-1;
        double end = Math.pow(10, -8);
        int pointsToCalculate = 100;
        double step = Math.abs(end-start)/(pointsToCalculate-1);
        try {
            FileOutputStream fileWriter = new FileOutputStream(filename, true);
            stream = new PrintStream(fileWriter);

            for (int i = 0; i < pointsToCalculate; i++) {
                double x = start + i*step;
                stream.println(x+" "+expo(x));
            }
        } catch (IOException e) {
            System.out.println("Une erreur s'est produite :/");
        } finally {
            if (stream != null) stream.close();
        }
        System.out.println("Et voila, tout ce trouve dans "+filename);
    }
}
```

Et voici le graphique généré via gnuplot[2] avec comme paramètre `pointsToCalculate` à 100 :

[2] http://www.gnuplot.info

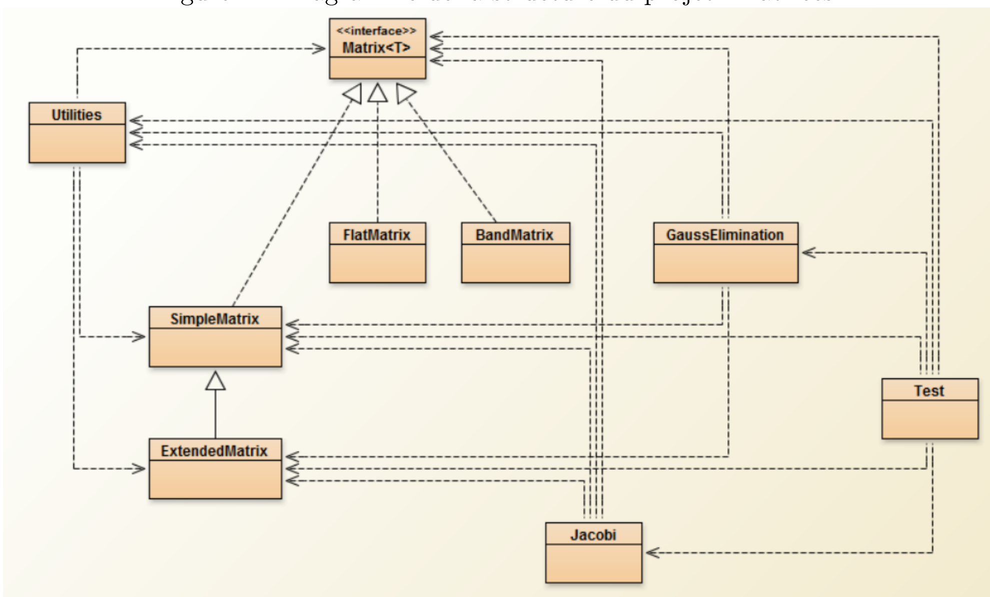
Figure 1: Plot avec 100 points de calcul



2 Matrix representation

Tous les codes de cette leçon sont dans le projet "matrices". En plus des codes demandés, plusieurs choses ont été imlémentées. D'une part une interface "Matrix" qui représente une matrice générique et une classe "Utilities" qui implémente plusieurs opérations sur les matrices (addition, affichage, ...).

Figure 2: Diagramme de la structure du projet ”matrices”



Tous ce code est expérimental et ne peut pas être utilisé tel quel dans un projet. En effet, ils risquent de générer des erreurs, par exemple parce qu'ils ne vérifient jamais si les données sont correctement formatées. Le but est avant tout de se concentrer sur les algos et leur fonctionnement.

2.1 Question 1

Le code se trouve dans le projet `matrices` dans la classe `FlatMatrix.java`.

```java
/**
 * Question 1
 * FlatMatrix
 *
 * @author DenisM
 * @version December 2016
 */
public class FlatMatrix implements Matrix<Double> {
    int m, n;
    double[] data;

    public FlatMatrix(int m, int n) {
        this.m = m;
        this.n = n;
        this.data = new double[m*n];
    }

    public FlatMatrix(int m, int n, double[][] vals) {
        this(m, n);

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                this.set(vals[i][j], i, j);
            }
        }
```

```java
    }

    public Double get(int i, int j) {
        int pos = j;
        if (i>0) pos += (i-1)*this.n;

        return data[pos];
    }

    public void set(Double value, int i, int j) {
        int pos = j;
        if (i>0) pos += (i-1)*this.n;

        data[pos] = value;
    }

    public int getWidth() { return this.n; }
    public int getHeight() { return this.m; }
}
```

2.2 Question 2

// TO DO

2.3 Question 3

2.3.1 Java representation of a band matrix

Le code se trouve dans le projet `matrices` dans la classe `BandMatrix.java`.

```java
/**
 * Question 3
 * BandMatrix
 *
 * @author DenisM
 * @version December 2016
 */
public class BandMatrix implements Matrix<Double> {
    int n;
    double[][] data;

    public BandMatrix(int n) {
        this.n = n;
        this.data = new double[2][n];
    }

    public BandMatrix(int n, double[][] vals) {
        this(n);

        for (int i = 0; i < n; i++) {
            for (int j = i; j < i+2; j++) {
                this.set(vals[i][j], i, j);
            }
        }
    }

    public Double get(int i, int j) {
        if (j-i==0 || j-i==1 || i<0 || j<0) return 0.;

        return data[j-i][j];
    }

```

```java
    public void set(Double value, int i, int j) {
        if (j-i==0 || j-i==1) return;

        data[j-i][j] = value;
    }

    public int getWidth() { return this.n; }
    public int getHeight() { return this.n; }
}
```

2.3.2 Multiplication by a band matrix

Le code se trouve dans le projet `matrices` dans la classe `Utilities.java`.

```java
public static Matrix multiplyByBand(Matrix base, Matrix band) {
    SimpleMatrix ret = new SimpleMatrix(base.getHeight(), base.getWidth());
    for (int i = 0; i < base.getHeight(); i++) {
        for (int j = 0; j < base.getWidth(); j++) {
            double pos = (double) base.get(i, j-1)* (double) band.get(j-1, j) + (double) base.
            get(i, j)*(double) band.get(j, j);
            ret.set(pos, i, j);
        }
    }
    return ret;
}
```

3 Linear systems

Les codes les codes de cette leçon sont également dans le projet "matrices". Dans la classe *Test.java* se trouvent deux méthodes `test_gauss` et `test_jacobi` qui contiennent des matrices de test pour tester les deux méthodes.

3.1 Question 1

Le code se trouve dans le projet `matrices` dans la classe `GaussElimination.java`.

```java
/**
 * Gauss elimination
 * 3 - Question 1
 *
 * @author DenisM
 * @version December 2016
 */
public class GaussElimination {
    public static Matrix process(ExtendedMatrix matrice, double[] bs) {
        Matrix res = new SimpleMatrix(matrice.getWidth(), 1);

        return forwardSubstitution(matrice, bs, res);
    }

    private static Matrix forwardSubstitution(ExtendedMatrix matrice, double[] bs, Matrix res)
    {
        for (int i = 0; i < matrice.getHeight(); i++) {
            double pivot = (double) matrice.get(i, i);
            // Switch rows
            int maxRow = getRowWhereColumnIsMax(matrice, i, i);
```

```java
            matrice.switchRows(i, maxRow);
            double b = bs[i];
            bs[i] = bs[maxRow];
            bs[maxRow] = b;

            pivot = matrice.get(i, i); // update pivot value

            // Divide actual row
            matrice.divideRow(i, pivot);
            bs[i] /= pivot;

            pivot = matrice.get(i, i); // update pivot value

            for (int j = i+1; j < matrice.getHeight(); j++) {
                double fact = matrice.get(j, i)/pivot;
                matrice.substractRows(j, i, fact);
                bs[j] -= bs[i]*fact;
            }
        }
        return backwardSubstitution(matrice, bs, res);
    }

    private static Matrix backwardSubstitution(ExtendedMatrix matrice, double[] bs, Matrix res)
     {
        for (int i = matrice.getHeight()-1; i>-1; i--) {
            double min = bs[i];

            for (int j = i+1; j<matrice.getWidth()-1; j++) {
                min -= matrice.get(i, j)*(double)res.get(j, 0);
            }

            res.set(min/matrice.get(i, i), i, 0);
        }
        return res;
    }

    private static int getRowWhereColumnIsMax(ExtendedMatrix matrice, int j, int offset) {
        double max = matrice.get(offset, j);
        int pos = offset;
        for (int i = offset; i < matrice.getHeight(); i++) {
            if (max<matrice.get(i, j)) {
                max = matrice.get(i, j);
                pos = i;
            }
        }
        return pos;
    }
}
```

3.2 Question 2

Le code se trouve dans le projet `matrices` dans la classe `Jacobi.java`.

```java
/**
 * Jacobi
 * 3 - Question 2
 *
 * @author DenisM
 * @version December 2016
 */
public class Jacobi {
    public static Matrix process(Matrix matrice, double[] bs, int iter) {
        Matrix prec = new SimpleMatrix(matrice.getWidth(), 1);

```

```java
        return iterate(matrice, bs, prec, iter);
    }

    private static Matrix iterate(Matrix matrice, double[] bs, Matrix prec, int iter) {
        Matrix res = new SimpleMatrix(matrice.getWidth(), 1);

        for (int i = 0; i < res.getHeight(); i++) {
            double sum = 0;
            for (int j = 0; j < matrice.getHeight(); j++) {
                if (i!=j) sum += (double) matrice.get(i, j)*(double) prec.get(j ,0);
            }
            double ici = (1/(double)matrice.get(i, i))*(bs[i]-sum);
            res.set(ici, i, 0);
        }
        if (iter==0) return res;
        return iterate(matrice, bs, res, -- iter);
    }
}
```

4 Linear regression

4.1 Question 1

$$\begin{aligned}
X^TXA &= X^TY \\
\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 95 & 85 & 80 & 70 & 60 & 70 \end{pmatrix} \cdot \begin{pmatrix} 1 & 95 \\ 1 & 85 \\ 1 & 80 \\ 1 & 70 \\ 1 & 60 \\ 1 & 70 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 95 & 85 & 80 & 70 & 60 & 70 \end{pmatrix} \cdot \begin{pmatrix} 85 \\ 95 \\ 70 \\ 65 \\ 70 \\ 80 \end{pmatrix} \\
\begin{pmatrix} 6 & 460 \\ 460 & 36050 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 465 \\ 36100 \end{pmatrix} \\
\begin{pmatrix} 1 & 36050/460 \\ 6 & 460 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 36100/460 \\ 465 \end{pmatrix} \\
\begin{pmatrix} 1 & 36050/460 \\ 0 & 460 - 6 \cdot \frac{36050}{460} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 36100/460 \\ 465 - 6 \cdot \frac{36100}{460} \end{pmatrix} \\
\begin{pmatrix} 1 & 36050/460 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 36100/460 \\ \frac{465 - 6 \cdot \frac{36100}{460}}{460 - 6 \cdot \frac{36050}{460}} \end{pmatrix} \\
\begin{pmatrix} 1 & 36050/460 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 36100/460 \\ \frac{27}{47} \end{pmatrix} \\
\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} 36100/460 - \frac{27}{47} \cdot 36050/460 \\ \frac{27}{47} \end{pmatrix}
\end{aligned} \tag{4}$$

$$\begin{aligned}
a_1 &= 33,457 \\
a_2 &= 0.5745
\end{aligned} \tag{5}$$

4.2 Question 2

4.2.1 Explicit normal equation for the model $y = a_1 + a_2 x$

$$\begin{pmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{pmatrix} \tag{6}$$

4.2.2 Explicit normal equation for the model $y = a_1 + a_2 x + a_3 x^2$

$$\begin{pmatrix} m & \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i^3 \\ \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i^3 & \sum_{i=1}^{m} x_i^4 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \\ \sum_{i=1}^{m} x_i^2 y_i \end{pmatrix} \tag{7}$$

4.3 Question 3

4.3.1 Point 1

$$\begin{aligned} A &= \begin{pmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{pmatrix}; L_A = \begin{pmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{pmatrix} \\ B &= \begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix}; L_B = \begin{pmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{pmatrix} \end{aligned} \tag{8}$$

Le fonctionnement de l'algorithme est développé en détail sur Wikipédia [3].
Le code se trouve dans le projet `matrices` dans la classe `CholeskyFactorization.java`.

```java
/**
 * Cholesky Factorization
 * 4 - Question 3
 *
 * @author DenisM
 * @version December 2016
 */
public class CholeskyFactorization {
    private static double somme(int i, int j, Matrix L) {
        double ret = 0;
        for (int k = 0; k < j; k++) {
            ret += (double) L.get(i, k)*(double) L.get(j, k);
        }
        return ret;
    }

    public static Matrix getL(Matrix C) {
        SimpleMatrix L = new SimpleMatrix(C.getHeight(), C.getWidth());

```

[3] https://fr.wikipedia.org/wiki/Factorisation_de_Cholesky#Algorithme

```
        for (int j = 0; j < C.getWidth(); j++) {
            double diag = Math.sqrt((double)C.get(j, j)-somme(j, j, L));
            L.set(diag, j, j);

            for (int i = j+1; i < C.getHeight(); i++) {
                double val = ((double) C.get(i, j)-somme(i, j, L))/diag;
                L.set(val, i, j);
            }
        }
        return L;
    }

    public static void main(String[] args) {
        double[][] tableau1 = {
            {25, 15, -5},
            {15, 18, 0},
            {-5, 0, 11}
        };
        SimpleMatrix matrice1 = new SimpleMatrix(tableau1);

        double[][] tableau2 = {
            {4, 12, -16},
            {12, 37, -43},
            {-16, -43, 98}
        };
        SimpleMatrix matrice2 = new SimpleMatrix(tableau2);

        // Exemple de Wikipédia
        double[][] tableau3 = {
            {1, 1, 1, 1},
            {1, 5, 5, 5},
            {1, 5, 14, 14},
            {1, 5, 14, 15}
        };
        SimpleMatrix matrice3 = new SimpleMatrix(tableau3);

        Matrix cholesky1 = getL(matrice1);
        Matrix cholesky2 = getL(matrice2);

        System.out.println("===== Matrice 1 =====");
        Utilities.printMatrix(matrice1);
        System.out.println("Sa version L :");
        Utilities.printMatrix(cholesky1);
        System.out.println("===== Matrice 2 =====");
        Utilities.printMatrix(matrice2);
        System.out.println("Sa version L :");
        Utilities.printMatrix(cholesky2);
    }
}
```

4.3.2 Point 2

// TO DO

4.3.3 Point 3

// TO DO

5 Matrix norm and condition

5.1 Question 1

Nous allons donc vérifier les 3 propriétés :

1. $||A + B|| \leq ||A|| + ||B||$
2. $||A \cdot W|| \leq ||A|| \cdot ||W||$
3. $||A \cdot B|| \leq ||A|| \cdot ||B||$

1e propriété

$$\begin{aligned} ||A + B|| &= max_{||V||=1}||(A + B)V|| \\ &= max_{||V||=1}||AV + BV|| \\ &= max_{||V||=1}||AV|| + max_{||V||=1}||BV|| \end{aligned} \tag{9}$$

On peut passer de la deuxième à la troisième ligne car quand on multiplie une matrice par un vecteur on obtient un vecteur, et on peut donc appliquer les propriétés des vecteurs.

D'autre part on a que :

$$||A|| + ||B|| = max_{||V||=1}||AV|| + max_{||V||=1}||AV|| \tag{10}$$

2e propriété Pour le cas ou $W = 0$, on sait que multiplier une matrice par un vecteur nul donne un vecteur nul, et que la norme d'un vecteur nul est nul.

$$\begin{aligned} ||A \cdot W|| &= ||A \cdot 0|| \\ &= ||0|| \\ &= 0 \end{aligned} \tag{11}$$

$$\begin{aligned} ||A|| \cdot ||W|| &= ||A|| \cdot ||0|| \\ &= ||A|| \cdot ||0|| \\ &= 0 \end{aligned} \tag{12}$$

Pour le cas ou $W \neq 0$

3e propriété // TO DO

5.2 Question 2

Trouvons juste un contre exemple.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}; B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}; C = A \cdot B = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \tag{13}$$

$$\begin{gathered} ||A \cdot B|| = ||C|| = max_{i,j}|c_{i,j}| = 50 \\ ||A|| \cdot ||B|| = max_{i,j}|a_{i,j}| \cdot max_{i,j}|b_{i,j}| = 4 \cdot 8 = 32 \end{gathered} \tag{14}$$

Pour que la norme soit sub-multiplicative il faut que $||A \cdot B|| \leq ||A|| \cdot ||B||$. Or 50 n'est pas plus petit ou égal à 32, donc cette norme ne peut être sub-multiplicative.

5.3 Question 3

Pour cette question toutes les matrices inverses ont été calculées via Matrixcalc.org. Pour des matrices 2x2 on peut aisément le faire ”à la main” avec la formule suivantes :

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}; A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{(-1)} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{15}$$

La condition pour appliquer cette méthode étant que $ad - bc \neq 0$ (sinon on divise par zéro, et ça on ne peut pas faire).

5.3.1 Point 1

$$\begin{aligned} Cond(A) &= ||A|| \cdot ||A^{-1}|| \\ || \cdot ||_1 &= max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| \\ \begin{pmatrix} 2 & 3 \\ 3 & 4 \end{pmatrix}^{(-1)} &= \begin{pmatrix} -4 & 3 \\ 3 & -2 \end{pmatrix} \\ ||A|| &= 7 \\ ||A||^{(-1)} &= 7 \\ Cond(A) &= 7 \cdot 7 = 49 \end{aligned} \tag{16}$$

5.3.2 Point 2

$$\begin{aligned} \begin{pmatrix} 2 & 2.2 \\ 3 & 3.2 \end{pmatrix}^{(-1)} &= \begin{pmatrix} -16 & 11 \\ 15 & -10 \end{pmatrix} \\ ||A|| &= 5.4 \\ ||A||^{(-1)} &= 31 \\ Cond(A) &= ||A|| \cdot ||A^{-1}|| \\ &= 5.4 \cdot 31 = 167.4 \end{aligned} \tag{17}$$

On ne peut rien dire sur le condition number sans avoir calculé A^{-1}.

5.3.3 Point 3

$$X = \begin{pmatrix} 2 & 2.2 \\ 3 & 3.2 \end{pmatrix}; Y = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \tag{18}$$

// TO DO

6 Interpolation

6.1 Question 1

6.1.1 Vandermonde

$$\begin{pmatrix} 1 & 12 & 144 \\ 1 & 20 & 400 \\ 1 & 24 & 576 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 16 \\ 17 \end{pmatrix} \tag{19}$$

On peut ensuite trouver les valeurs a_1, a_2 et a_3 en résolvant le système.

Ici, nous n'avons pas fait ce développement, nous avons simplement entré la matrice dans la méthode de résolution par Gauss-Jordan implémentée dans la première question de la leçon 3.

Nous obtenons ainsi :

$$\begin{aligned} a_1 &= -39 \\ a_2 &= 4.833 \\ a_3 &= -0.104 \end{aligned} \tag{20}$$

6.1.2 Lagrange

$$f(x) = (y^{(1)} \cdot \Phi_1(x)) + (y^{(2)} \cdot \Phi_2(x)) + (y^{(3)} \cdot \Phi_3(x)) \tag{21}$$

$$\begin{aligned} \Phi_1(x) &= \frac{(x - x^{(2)})(x - x^{(3)})}{(x^{(1)} - x^{(2)})(x^{(1)} - x^{(3)})} = \frac{(x-20)(x-24)}{96} = \frac{x^2}{96} - \frac{11}{24}x + 5 \\ \Phi_1(x) &= \frac{(x - x^{(1)})(x - x^{(3)})}{(x^{(2)} - x^{(1)})(x^{(2)} - x^{(3)})} = \frac{(x-12)(x-24)}{-32} = \frac{-3x^2}{96} + \frac{27}{24}x - 9 \\ \Phi_1(x) &= \frac{(x - x^{(1)})(x - x^{(2)})}{(x^{(3)} - x^{(1)})(x^{(3)} - x^{(2)})} = \frac{(x-12)(x-20)}{48} = \frac{2x^2}{96} - \frac{16}{24}x + 5 \end{aligned} \tag{22}$$

$$f(x) = \frac{-5}{48}x^2 + \frac{29}{6}x - 144 \tag{23}$$

6.1.3 Newton

$$\begin{aligned} \Psi_1(x) &= 1 \\ \Psi_2(x) &= \Psi_1(x) \cdot (x - 12) \\ \Psi_3(x) &= \Psi_2(x) \cdot (x - 20) = (x-12)(x-20) \end{aligned} \tag{24}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 8 & 0 \\ 1 & 12 & 48 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 16 \\ 17 \end{pmatrix} \tag{25}$$

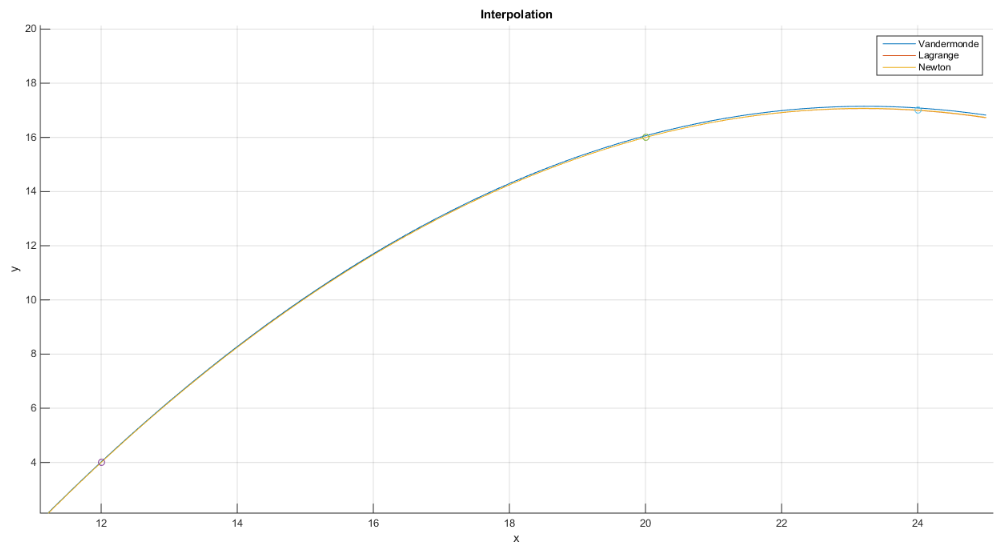
A l'aide de notre méthode Java pour Gauss-Jordan, nous obtenons que

$$\begin{aligned} c_1 &= 4 \\ c_2 &= 1.5 \\ c_3 &= -0.104 \end{aligned} \tag{26}$$

Nous pouvons alors définir $f(x)$:

$$f(x) = 4 + 1.5(x - 12) - 0.104(x - 12)(x - 20) \tag{27}$$

Figure 3: Interpolation avec 3 points



6.1.4 Vandermonde avec une mesure en plus

$$\begin{pmatrix} 1 & 12 & 144 & 1728 \\ 1 & 20 & 400 & 8000 \\ 1 & 24 & 576 & 13824 \\ 1 & 16 & 256 & 4096 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 16 \\ 17 \\ 13 \end{pmatrix} \tag{28}$$

On peut ensuite trouver les valeurs a_1, a_2, a_3 et a_4 en résolvant le système. Nous obtenons ainsi :

$$\begin{aligned} a_1 &= -99 \\ a_2 &= 46/3 \\ a_3 &= -11/16 \\ a_4 &= 1/96 \end{aligned} \tag{29}$$

6.1.5 Newton avec une mesure en plus

$$\begin{aligned}
&f(x) = 4 + 1.5(x-12) - 0.104(x-12)(x-20) + c_4(x-12)(x-20)(x-24) \\
&f(16) = 13 \\
&13 = 10 + \frac{80}{48} + 128c_4 \\
&c_4 = \frac{3 - 80/48}{128} = \frac{1}{96}
\end{aligned} \tag{30}$$

Figure 4: Interpolation avec 4 points

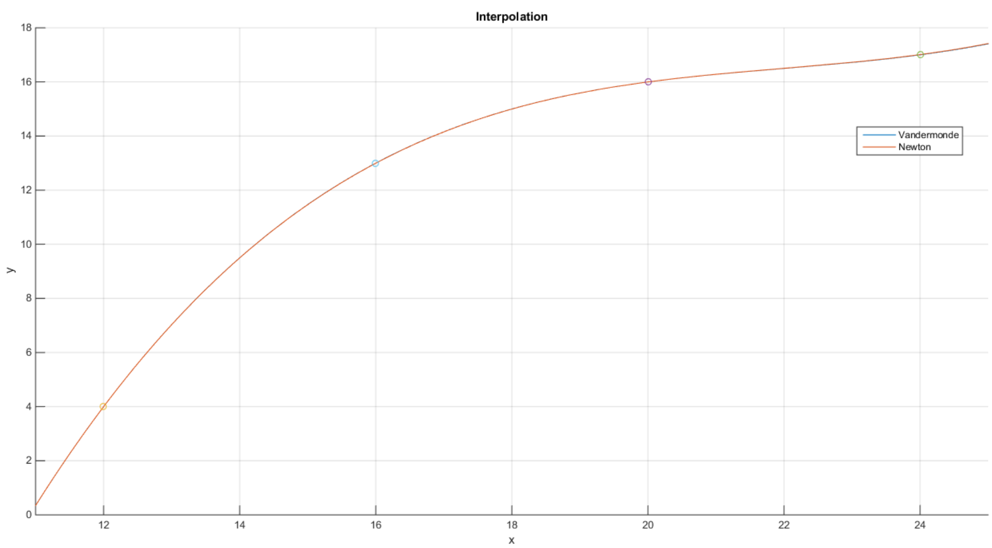
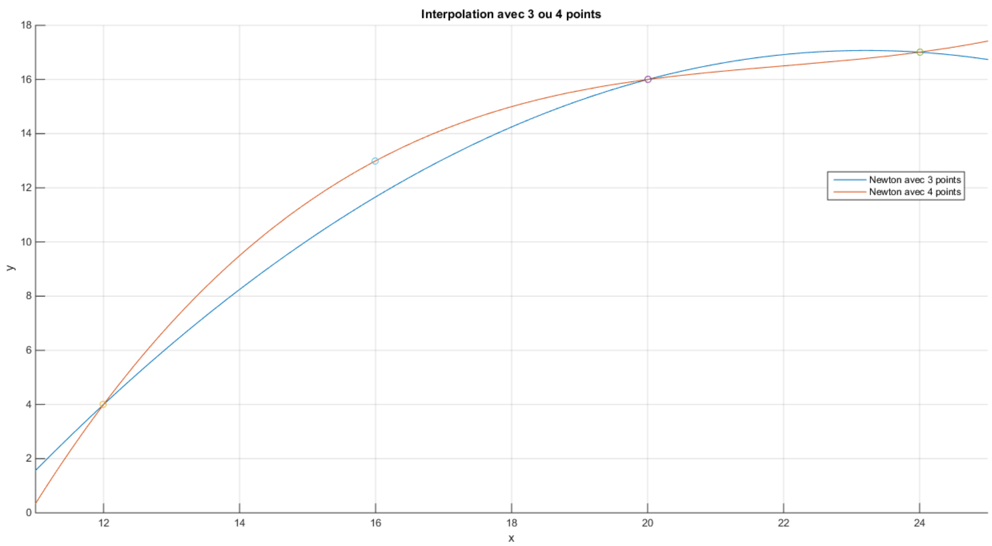


Figure 5: Interpolation via Newton avec 3 ou 4 points



6.2 Question 2

Tous les codes de cette question sont dans le projet `numalgoplotter` qui est basé sur les codes fournis sur moodle.

6.2.1 NumAlgoPlotter

Le code se trouve dans le projet `numalgoplotter` dans la classe `NumAlgoPlotter.java`.

```
import java.awt.Color;
import java.util.LinkedList;

/**
 *
 * @author Ramin Sadre
 */
public class NumAlgoPlotter {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        LinkedList<ControlPoint> controlPoints=new LinkedList<>();

        controlPoints.add(new ControlPoint(-300,-100));
        controlPoints.add(new ControlPoint(-100,0));
        controlPoints.add(new ControlPoint(100,0));
        controlPoints.add(new ControlPoint(300,100));

        MainFrame mainFrame=new MainFrame(controlPoints);

        mainFrame.add(new SimpleCurvePlotter() {
            @Override
            public double calculateY(double x) {
```

```java
                // tan:
                // return Math.tan(x/200.0)*100.0;
                // runge function:
                return -1.0/(1.0+x*x/3600.0)*100.0;
            }
        }.setColor(Color.green));

        mainFrame.add(new VandermondePlotter().setColor(Color.red));
        mainFrame.add(new NewtonPlotter().setColor(Color.blue));
        mainFrame.add(new CubicSplinePlotter());

        // Ne pas oubliez de commenter tous les autres plotters pour utiliser BSplinePlotter
        // mainFrame.add(new BSplinePlotter());

        mainFrame.setVisible(true);
    }
}
```

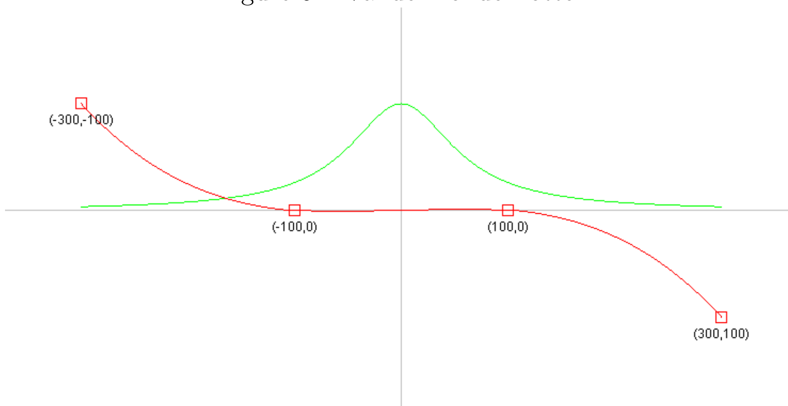
6.2.2 VandermondePlotter

Le code se trouve dans le projet `numalgoplotter` dans la classe `VandermondePlotter.java`.

```java
/**
 * A plotter for polynomials.
 * The polynomial is fitted to the control points by solving the Vandermonde system X*A=Y
 * @author Ramin Sadre
 */
public class VandermondePlotter extends SimpleCurvePlotter {
    // The coefficients of the polynomial y = a_0 + a_1*x + a_3*x^2 + ...
    // Note that we start with a_0 (not a_1 like in the course) because
    // Java arrays start with index 0.
    private double[] vectorA;

    // This method is called whenever a control point is changed or deleted or added.
    // Here is a good place to do calculations that you need to do only once.
    @Override
    public void doPreparations(ControlPoint[] controlPoints) {
        // Construct and solve the linear system X*A=Y (where X is the Vandermonde matrix).
        // The degree of the polynomial is the number of control points minus 1.

        // n = the number of points
        int n=controlPoints.length;

        double[][] A = new double[n][n];
        double[] B = new double[n];

        for (int i = 0; i < n; i++) {
            B[i] = controlPoints[i].getY();
            double base = controlPoints[i].getX();

            for (int j = 0; j<n; j++) {
                A[i][j] = Math.pow(base, j);
            }
        }
        vectorA= GaussianElimination.solve(A, B);
    }

    // This method calculates y=a0+a1*x+a2*x^2+...
    public double calculateY(double x) {
      double y = 0;

        for (int i = 0; i < vectorA.length; i++) {
            y += vectorA[i] * Math.pow(x, i);
        }
```

```
        return y;
    }
}
```

Figure 6: VandermondePlotter



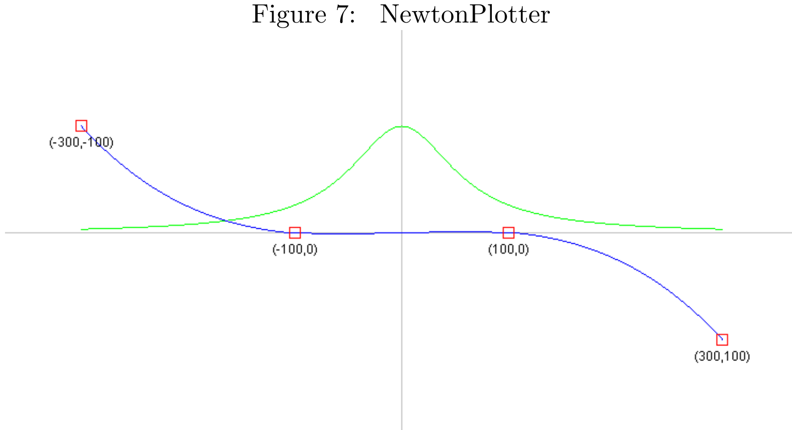
6.2.3 NewtonPlotter

Le code se trouve dans le projet `numalgoplotter` dans la classe `NewtonPlotter.java`.

```
/**
 * A plotter for polynomials.
 * The polynomial is fitted to the control points by solving the Newton system
 * @author DenisM
 */
public class NewtonPlotter extends SimpleCurvePlotter {
    private double[] ci;
    private double[] xi;

    @Override
    public void doPreparations(ControlPoint[] controlPoints) {
        int n=controlPoints.length;

        double[][] A = new double[n][n];
        double[] B = new double[n];
        xi = new double[n];
        for (int i = 0; i < n; i++) {
            double diable = 1;
            xi[i] = controlPoints[i].getX();
            for (int j = 0; j<i+1; j++) {
                if (j>0) diable *= (controlPoints[i].getX()-controlPoints[j-1].getX());
                A[i][j] = diable;
            }
            B[i] = controlPoints[i].getY();
        }
        ci= GaussianElimination.solve(A, B);
    }

    public double calculateY(double x) {
        double y = 0;
        double diable = 1;

        for (int i = 0; i < ci.length; i++) {
```

```java
            if (i>0) diable *= (x-xi[i-1]);
            y += ci[i] * diable;
        }

        return y;
    }
}
```

Figure 7: NewtonPlotter



7 Splines

Tous les codes de cette leçon sont comme pour la dernière leçon dans le projet `numalgoplotter` qui est basé sur les codes fournis sur moodle.

7.1 Question 1

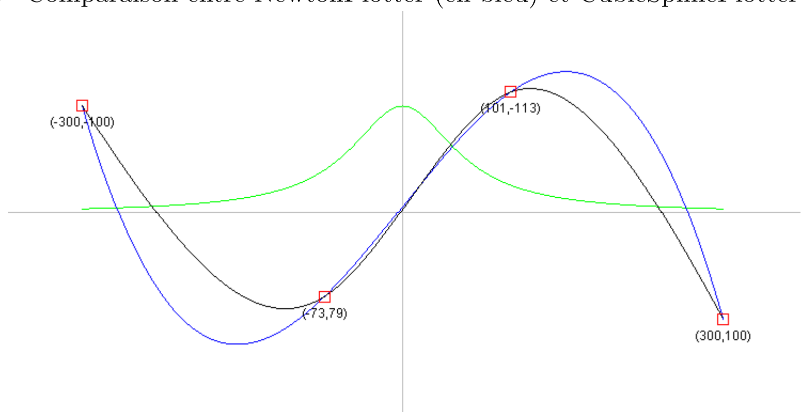
Le code se trouve dans le projet `numalgoplotter` dans la classe `CubicSplinePlotter.java`.

```java
import java.awt.Color;
import java.awt.Graphics;
import java.util.List;

/**
 *
 * @author Ramin Sadre
 */
public class CubicSplinePlotter extends CurvePlotter {
    private ControlPoint[] controlPoints;
    private int n;
    private double[] B; // the b_i coefficients of the spline
    private double[] C; // the c_i coefficients of the spline
    private double[] D; // the d_i coefficients of the spline

    @Override
    public void controlPointsChanged(List<ControlPoint> lcp) {
        // sort the control points by x position.
        // this makes things easier later when we paint the function.
        lcp.sort(
```

```java
                (point1,point2) -> Integer.compare(point1.getX(),point2.getX())
        );

        // collect the points in an array before doing the calculations.
        // again, this makes things easier for us.
        controlPoints=lcp.toArray(new ControlPoint[0]);
        n=controlPoints.length;
        if(n<2) {
            // nothing to do here
            return;
        }

        // Construct the linear system S*C=T with n-2 equations.
        // The matrix S contains the left hand side of the equations on slide 13.
        // The matrix Z contains the right hand side of the equations on slide 13.
        double[][] S=new double[n-2][n-2];
        double[] Z=new double[n-2];

        for (int i = 0; i < n-2; i++) {
            int realX = i+1;
            double ha = (lcp.get(realX+0).getX()-lcp.get(realX-1).getX());
            double hb = (lcp.get(realX+1).getX()-lcp.get(realX+0).getX());

            double ya = (lcp.get(realX+0).getY()-lcp.get(realX-1).getY());
            double yb = (lcp.get(realX+1).getY()-lcp.get(realX+0).getY());

            if (i>0) S[i][i-1] = ha;
            S[i][i] = 2*(ha+hb);
            if (i<n-3) S[i][i+1] = hb;

            Z[i] = 3*((yb/hb)-(ya/ha));
        }

        double[] ci= GaussianElimination.solve(S, Z);
        B = new double[n];
        C = new double[n];
        D = new double[n];
        for (int i = 0; i < ci.length; i++) {
            C[i+1] = ci[i];
        }

        for (int i = 0; i < n-1; i++) {
            double ha = (lcp.get(i+1).getX()-lcp.get(i).getX());
            D[i] = (C[i+1]-C[i])/(3*ha);
            B[i] = (lcp.get(i+1).getY()-lcp.get(i).getY())/ha-C[i]*ha-D[i]*Math.pow(ha, 2);
        }
    }

    @Override
    public void paint(Graphics g) {
        g.setColor(Color.black) ;

        // Paint the n-1 polynomials
        for(int i=0;i<n-1;i++) {
            int x_i=controlPoints[i].getX();
            int x_iplus1=controlPoints[i+1].getX();
            double a_i=controlPoints[i].getY();

            int previousX=0,previousY=0;
            for(int x=x_i;x<x_iplus1;x++) {
               double y = a_i + B[i]*(x-x_i) + C[i]*Math.pow((x-x_i), 2) + D[i]*Math.pow((x-x_i), 3);

                // draw a line between this (x,y) and the previous (x,y)
                if(x!=x_i) {
                    g.drawLine(previousX,previousY,x,(int)y);
                }
                previousX=x;
                previousY=(int)y;
```

```
            }
        }
    }
}
```

Figure 8: Comparaison entre NewtonPlotter (en bleu) et CubicSplinePlotter (en noir)



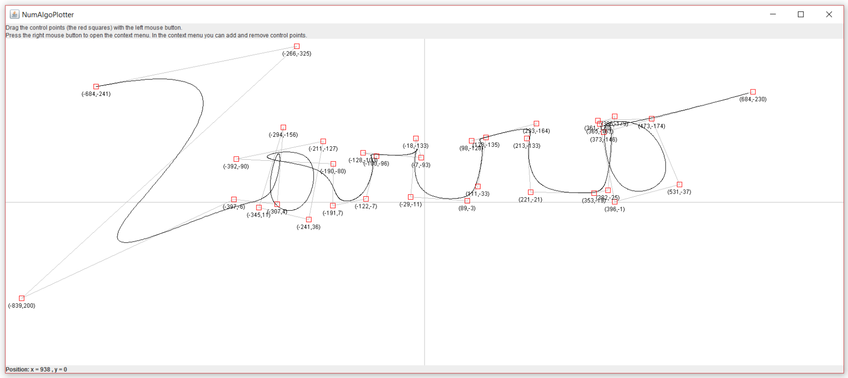
7.2 Question 2

Le code se trouve dans le projet `numalgoplotter` dans la classe `BSplinePlotter.java`.

```
import java.awt.Color;
import java.awt.Graphics;
import java.util.List;

/**
 *
 * @author Ramin Sadre
 */
public class BSplinePlotter extends CurvePlotter {
    private ControlPoint[] controlPoints;
    private int n;
    private final static int m = 3;  // the degree of the B-spline

    @Override
    public void controlPointsChanged(List<ControlPoint> cp) {
        controlPoints=cp.toArray(new ControlPoint[0]);
        n=controlPoints.length;
    }

    @Override
    public void paint(Graphics g) {
        // draw lines between the control points (so the user can see the order of the points)
        g.setColor(Color.lightGray) ;
        for(int i=0;i<n;i++) {
            if(i!=0) {
                ControlPoint c1=controlPoints[i-1];
                ControlPoint c2=controlPoints[i];
                g.drawLine(c1.getX(),c1.getY(),c2.getX(),c2.getY());
            }
        }

```

```
        // now, draw the curve
        g.setColor(Color.black) ;

        int previousX=0,previousY=0;
        for(double t=0.0;t<=n-m;t+=0.01) {

            // Missing code:
            // Calculate x and y
            double x = 0, y = 0;

            for (int i = 0; i < n; i++) {
                ControlPoint here = controlPoints[i];
                double b = b(i, m, t);
                x += here.getX()*b;
                y += here.getY()*b;
            }

            // draw a line between this (x,y) and the previous (x,y)
            if(t>0.0) {
                g.drawLine(previousX,previousY,(int)x,(int)y);
            }
            previousX=(int)x;
            previousY=(int)y;
        }
    }

    private int t(int i) {
        // Missing code:
        if (i <= m) return 0;
        if (i >= n) return n-m;
        return i-m;
    }

    private double b(int i,int k,double t) {
        // Missing code:
        if (k==0) {
            if (t>=t(i) && t<=t(i+1)) return 1;
            return 0;
        }

        double b = 0;

        double denom1 = t(i+k)-t(i);
        if (denom1!=0) {
            b += ((t-t(i))/denom1)*b(i, k-1, t);
        }

        double denom2 = t(i+k+1)-t(i+1);
        if (denom2!=0) {
            b += ((t(i+k+1)-t)/denom2)*b(i+1, k-1, t);
        }
        return b;
    }
}
```

Figure 9:   Exemple de B-Spline



7.3 Question 3

// TO DO

7.4 Question 4

// TO DO

8 Numerical integration

8.1 Question 1

8.1.1 Point 1

Données :

$$\begin{aligned}
&\int_a^b f(x) \approx \sum_{i=1}^{n} f(x^{(i)}) \cdot h \cdot \int_1^n \frac{\prod_{j\neq i}(t-j)}{\prod_{j\neq i}(i-j)} dt \\
&n = 3 \\
&h = \frac{b-a}{n-1} = \frac{b-a}{2}
\end{aligned} \tag{31}$$

Résolution :

$$\begin{aligned}
\int_a^b f(x) &\approx \sum_{i=1}^{3} f(x^{(i)}) \cdot \frac{b-a}{2} \cdot \int_1^3 \frac{\prod_{j\neq i}(t-j)}{\prod_{j\neq i}(i-j)} dt \\
&= f(x^{(1)}) \cdot \frac{b-a}{2} \cdot \omega_1 + f(x^{(2)}) \cdot \frac{b-a}{2} \cdot \omega_2 + f(x^{(3)}) \cdot \frac{b-a}{2} \cdot \omega_3
\end{aligned} \tag{32}$$

On peut calculer les ω_i :

$$\begin{aligned}
\omega_1 &= \int_1^3 \frac{(t-2)(t-3)}{(1-2)(1-3)} dt \\
&= \frac{1}{2} \int_1^3 t^2 - 5t + 6dt \\
&= \frac{1}{2} \left[\frac{t^3}{3} - \frac{5t^2}{2} + 6t \right]_1^3 \\
&= \frac{1}{2} \left[\frac{9}{2} - \frac{23}{6} \right] = \frac{1}{3} \\
\omega_2 &= \int_1^3 \frac{(t-3)(t-3)}{(2-1)(2-3)} dt \\
&= -1 \int_1^3 t^2 - 4t + 3dt \\
&= -1 \left[\frac{t^3}{3} - 2t^2 + 3t \right]_1^3 \\
&= -1 \left[0 - \frac{4}{3} \right] = \frac{4}{3} \\
\omega_3 &= \int_1^3 \frac{(t-1)(t-2)}{(3-1)(3-2)} dt \\
&= \frac{1}{2} \int_1^3 t^2 - 3t + 2dt \\
&= \frac{1}{2} \left[\frac{t^3}{3} - \frac{3t^2}{2} + 2t \right]_1^3 \\
&= \frac{1}{2} \left[\frac{3}{2} - \frac{5}{6} \right] = \frac{1}{3}
\end{aligned} \tag{33}$$

On peut alors remplacer les ω_i pour obtenir :

$$\begin{aligned}
\int_a^b f(x) &\approx f(a) \cdot \frac{b-a}{2} \cdot \frac{1}{3} + f(\frac{a+b}{2}) \cdot \frac{b-a}{2} \cdot \frac{4}{3} + f(b) \cdot \frac{b-a}{2} \cdot \frac{1}{3} \\
&= \frac{b-a}{6} \left(f(a) + 4f(\frac{a+b}{2}) + f(b) \right)
\end{aligned} \tag{34}$$

8.1.2 Point 2

// TO DO

8.2 Question 2

8.2.1 Point 1

// TO DO

8.2.2 Point 2

// TO DO

8.2.3 Point 3

// TO DO

8.3 Question 3

8.3.1 Newton-Cotes with $n = 2$

$$\begin{aligned} \int_{-1}^{2} e^x dx &\approx \sum_{i=1}^{2} f(x^{(i)}) \cdot h \cdot \int_{1}^{2} \frac{\prod_{j\neq i}(t-j)}{\prod_{j\neq i}(i-j)} dt \\ &= f(x^{(1)}) \cdot h \cdot \omega_1 + f(x^{(2)}) \cdot h \cdot \omega_2 \end{aligned} \tag{35}$$

On a que :

$$\begin{aligned} n &= 2 \\ [a, b] &= [-1, 2] \\ h &= \frac{b-a}{n-1} = 3 \end{aligned} \tag{36}$$

On peut calculer les ω_i :

$$\begin{aligned} \omega_1 &= \int_{1}^{2} \frac{(t-2)}{(1-2)} dt \\ &= -\left[\frac{t^2}{2} - 2t\right]_1^2 \\ &= -[0 - (-3/2)] = \frac{-3}{2} \\ \omega_2 &= \int_{1}^{2} \frac{(t-1)}{(2-1)} dt \\ &= \left[\frac{t^2}{2} - t\right]_1^2 \\ &= [0 - (-1/2)] = \frac{1}{2} \end{aligned} \tag{37}$$

On peut dès lors résoudre :

$$\begin{aligned} \int_{-1}^{2} e^x dx &\approx f(x^{(1)}) \cdot h \cdot \omega_1 + f(x^{(2)}) \cdot h \cdot \omega_2 \\ &= f(-1) \cdot 3 \cdot \frac{-3}{2} + f(2) \cdot 3 \cdot \frac{1}{2} \\ &= e^{(-1)} \cdot 3 \cdot \frac{-3}{2} + e^2 \cdot 3 \cdot \frac{1}{2} \\ &= 9.42812 \end{aligned} \tag{38}$$

L'erreur est donc de $9.42812 - 7.02118 = 2.40694$.

8.3.2 Gauss-Legendre with $n = 2$

Si on connais ω_i et $x^{(i)}$ pour $\int_{-1}^{1}$ on a que

$$\begin{aligned}\int_a^b f(x)dx &= \frac{b-a}{2}\sum_{i=1}^{n}\omega_i f\left(\frac{b-a}{2}x^{(i)} + \frac{a+b}{2}\right)\\ a &= -1\\ b &= 2\\ n &= 2\end{aligned} \tag{39}$$

On va donc commencer par trouver ω_i et $x^{(i)}$.

$$\begin{aligned}\sum_{j=1}^{n}\omega_j(x^{(i)})^{i-1} &= \frac{(b^i - a^i)}{i}\\ i &= 1, ..., 2n\\ \begin{cases}\omega_1 + \omega_2 = 2\\ \omega_1 x^{(1)} + \omega_2 x^{(2)} = 0\\ \omega_1 {x^{(1)}}^2 + \omega_2 {x^{(2)}}^2 = \frac{2}{3}\\ \omega_1 {x^{(1)}}^3 + \omega_2 {x^{(2)}}^3 = 0\end{cases} \qquad \omega_1 &= 1\\ \omega_2 &= 1\\ x^{(1)} &= -\frac{1}{\sqrt{3}}\\ x^{(2)} &= \frac{1}{\sqrt{3}}\end{aligned} \tag{40}$$

On peut alors appliquer la formule pour obtenir l'intégrale par Gauss-Legendre :

$$
\begin{aligned}
\int_a^b f(x)dx &= \frac{b-a}{2} \sum_{i=1}^{n} \omega_i f\left(\frac{b-a}{2} x^{(i)} + \frac{a+b}{2}\right) \\
&= \frac{2-(-1)}{2} \sum_{i=1}^{2} \omega_i f\left(\frac{2-(-1)}{2} x^{(i)} + \frac{(-1)+2}{2}\right) \\
&= \frac{3}{2}\left[\omega_1 f\left(\frac{3}{2} x^{(1)} + \frac{1}{2}\right) + \omega_2 f\left(\frac{3}{2} x^{(2)} + \frac{1}{2}\right)\right] \\
&= \frac{3}{2}\left[1 \cdot f\left(\frac{3}{2} \cdot \frac{-1}{\sqrt{3}} + \frac{1}{2}\right) + 1 \cdot f\left(\frac{3}{2} \cdot \frac{-1}{\sqrt{3}} + \frac{1}{2}\right)\right] \\
&= \frac{3}{2}\left[e^{\frac{-\sqrt{3}+1}{2}} + e^{\frac{\sqrt{3}+1}{2}}\right] \\
&= \frac{3}{2}\left[e^{\frac{-\sqrt{3}+1}{2}} + e^{\frac{\sqrt{3}+1}{2}}\right] \\
&= \frac{3}{2}\left[0.693485 + 3.919740\right] \qquad = 6.91983
\end{aligned}
\tag{41}
$$

L'erreur est donc de $6.91983 - 7.02118 = -0.10135$, soit une erreur ± 23 fois plus petite qu'avec la méthode de Newton-Cotes. La méthode de Gauss-Legendre donne donc le meilleur résultat.

8.4 Question 4

// TO DO

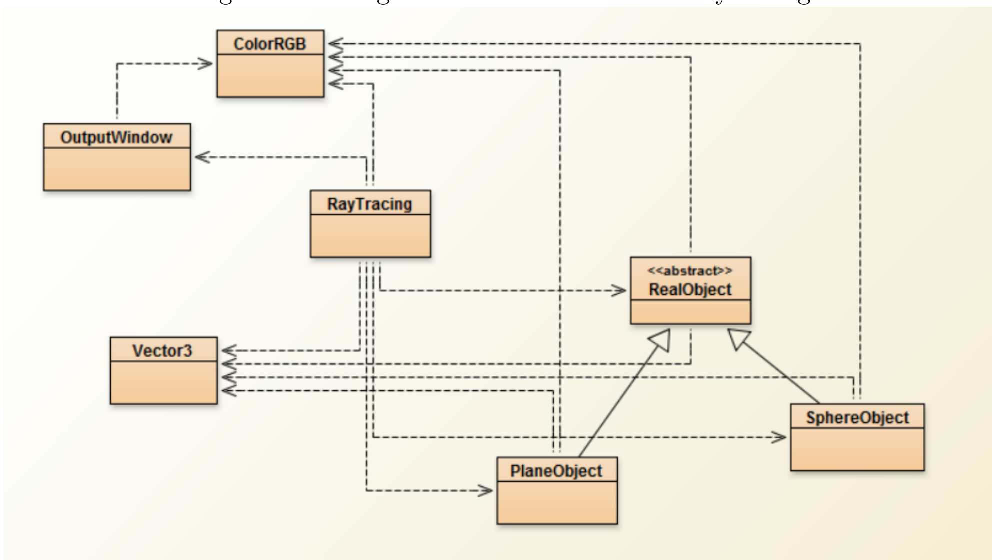
8.5 Question 5

// TO DO

9 Ray tracing (part 1)

Tous les codes de cette leçon sont dans le projet `raytracing` qui est basé sur les codes fournis sur moodle.

Figure 10: Diagramme de la structure de raytracing



9.1 Question 1

9.1.1 Vector3

Il y a dans cette classe quelques méthodes en plus de ce qui est demandé dans l'énoncé. Il s'agit de méthodes qui se sont révélées utiles par la suite lors de l'implémentation des autres méthodes.

Le code se trouve dans le projet `raytracing` dans la classe `Vector3.java`.

```java
/**
 * Class for representing vectors
 *
 * @author DenisM
 * @version December 2016
 */
public class Vector3 {
    public double x,y,z;

    public Vector3(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    /**
     * Addition de deux vecteurs
     */
    public static Vector3 add(Vector3 v1, Vector3 v2) {
        return new Vector3(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);
    }

    /**
     * Addition de 4 vecteurs
     */
    public static Vector3 add(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4) {
        return add(add(v1, v2), add(v3, v4));
```

```java
    }

    /**
     * Soustraction de deux vecteurs
     */
    public static Vector3 subst(Vector3 v1, Vector3 v2) {
        return new Vector3(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);
    }

    /**
     * Multiplication d'un vecteur par un facteur constant
     */
    public static Vector3 multiply(Vector3 v1, double factor) {
        return new Vector3(v1.x * factor, v1.y * factor, v1.z * factor);
    }

    /**
     * Division d'un vecteur par un facteur constant
     */
    public static Vector3 divide(Vector3 v1, double factor) {
        if (factor==0) return v1;
        return new Vector3(v1.x / factor, v1.y / factor, v1.z / factor);
    }

    /**
     * Renvoie la longueur du vecteur
     */
    public static double getLength(Vector3 v1) {
        return Math.sqrt(Math.pow(v1.x, 2) + Math.pow(v1.y, 2) + Math.pow(v1.z, 2));
    }

    /**
     * Renvoie la version normalisée d'un vecteur
     */
    public static Vector3 normalize(Vector3 v1) {
        double length = getLength(v1);
        if (length==0) return new Vector3(0, 0, 0);
        return divide(v1, length);
    }

    /**
     * Renvoie le ploduit scalaire de deux vecteurs
     */
    public static double dotProduct(Vector3 v1, Vector3 v2) {
        return v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
    }

    /**
     * Renvoie le produit vectoriel de deux vecteurs
     */
    public static Vector3 crossProduct(Vector3 v1, Vector3 v2) {
        return new Vector3(v1.y*v2.z-v1.z*v2.y, v1.z*v2.x-v1.x*v2.z, v1.x*v2.y-v1.y*v2.x);
    }
}
```

9.1.2 ColorRGB

Ici aussi, quelques méthodes complémentaires, qui seront utiles par la suite, ont été implémentées.

Le code se trouve dans le projet `raytracing` dans la classe `ColorRGB.java`.

```java
/**
 * Class for representing colors
 *
 * @author DenisM
```

```
 * @version December 2016
 */
public class ColorRGB {
    public double r,g,b;

    public ColorRGB(double r, double g, double b) {
        this.r = r;
        this.g = g;
        this.b = b;
    }

    /**
     * Ajoute la couleur passée en argument à la couleur actuelle
     */
    public void add(ColorRGB color) {
        this.r += color.r;
        this.g += color.g;
        this.b += color.b;
    }

    /**
     * Multiplie les composantes de deux couleurs
     */
    public static ColorRGB multiply(ColorRGB a, ColorRGB b) {
        return new ColorRGB(a.r*b.r, a.g*b.g, a.b*b.b);
    }

    /**
     * Multiplie les composantes d'une couleur par un facteur constant
     */
    public static ColorRGB multiply(ColorRGB a, double factor) {
        return new ColorRGB(a.r*factor, a.g*factor, a.b*factor);
    }

    /**
     * Multiplie les composantes de deux couleurs d'abord entre elles puis par un facteur
     constant
     */
    public static ColorRGB multiply(ColorRGB a, ColorRGB b, double factor) {
        return multiply(multiply(a, b), factor);
    }
}
```

9.1.3 PlaneObject

Le code se trouve dans le projet `raytracing` dans la classe `PlaneObject.java`.

```
/**
 * Class for representing plane objects
 *
 * @author DenisM
 * @version December 2016
 */
public class PlaneObject extends RealObject {
    public final Vector3 c;
    public final Vector3 n;

    public PlaneObject(Vector3 c, Vector3 n, ColorRGB color) {
        super(color);
        this.c  = c;
        this.n  = n;
    }

    public double testCollision(Vector3 rayPosition,Vector3 rayDirection) {
```

```
        double dn = Vector3.dotProduct(rayDirection, this.n);
        if (dn==0) return -1;
        Vector3 min = Vector3.subst(this.c, rayPosition);
        return Vector3.dotProduct(min, this.n)/dn;
    }

    public Vector3 getNormal(Vector3 pointOnSurface) {
        return this.n;
    }

    public ColorRGB getColor(Vector3 pointOnSurface) {
        return this.color;
    }
}
```

9.1.4 SphereObject

Le code se trouve dans le projet `raytracing` dans la classe `SphereObject.java`.

```
/**
 * Class for a sphere object
 *
 * @author DenisM
 * @version December 2016
 */
public class SphereObject extends RealObject {
    public Vector3 c;
    public double radius;
    public ColorRGB color;

    public SphereObject(Vector3 c, double radius, ColorRGB color) {
        super(color);
        this.color = color;
        this.c = c;
        this.radius = radius;
    }

    /**
     * Renvoie -1 si pas de collision ou la distance si il y a une collision
     * Pour le fonctionnement, voir l'algo dans les slides
     */
    public double testCollision(Vector3 rayPosition, Vector3 rayDirection) {
        Vector3 f = Vector3.subst(rayPosition, this.c);
        double a = Vector3.dotProduct(f, rayDirection);
        if (a>0) return -1;

        double f2 = Vector3.dotProduct(f, f);
        double b = f2 - Math.pow(this.radius, 2);

        double delta = Math.pow(a, 2) - b;

        if (delta<0) return -1;
        if (delta==0) return -1*a;
        return -1*a-Math.sqrt(delta);
    }

    /**
     * Renvoie le vecteur normal à la position passée en argument
     */
    public Vector3 getNormal(Vector3 pointOnSurface) {
        Vector3 min = Vector3.subst(pointOnSurface, c);
        return new Vector3(min.x/radius, min.y/radius, min.z/radius);
    }

```

```
    public ColorRGB getColor(Vector3 pointOnSurface) {
        return this.color;
    }
}
```

9.1.5 RayTracing

La classe `RealObject` a du être adaptée pour que la couleur soit de type `ColorRGB` et non pas un `Vector3`.

Le code se trouve dans le projet `raytracing` dans la classe `RayTracing.java`.

```
import java.util.LinkedList;

/**
 *
 * @author ramin
 */
public class RayTracing {
    private static final int screenWidth=800;
    private static final int screenHeight=600;

    // Camera : position, direction and up
    private Vector3 pc;
    private final Vector3 dc;
    private final Vector3 up;

    private final OutputWindow outputWindow;

    // List of all objects
    LinkedList<RealObject> objects=new LinkedList<>();

    public RayTracing() {
        outputWindow=new OutputWindow(screenWidth,screenHeight);

        // setting up camera
        pc = new Vector3(0, 0, 15);
        dc = new Vector3(0, 0, -1);
        up = new Vector3(0, 1, 0);

        ColorRGB dark_grey = new ColorRGB(0.2, 0.2, 0.2);
        ColorRGB red       = new ColorRGB(1, 0, 0);
        ColorRGB yellow    = new ColorRGB(1, 1, 0);

        // initialize scene
        objects.add(new SphereObject(new Vector3(0, 2, 0), 2, dark_grey));
        objects.add(new SphereObject(new Vector3(4, 0, 2), 2, red));
        objects.add(new PlaneObject(new Vector3(0, -4, 0), new Vector3(0, 1, 0), yellow));
    }

    private ColorRGB traceRay(Vector3 rayPosition,Vector3 rayDirection) {
        RealObject hittedObject = null;
        double tret = Double.MAX_VALUE;
        for (int i = 0; i < objects.size(); i++) {
            RealObject here = objects.get(i);
            double thit = here.testCollision(rayPosition, rayDirection);

            if (thit>0 && thit<tret) {
                tret = thit;
                hittedObject = here;
            }
        }

        if (hittedObject==null) return new ColorRGB(0, 0, 0);
```

```java

        ColorRGB objectColor = hittedObject.color;

        double distanceFactor = 120/Math.pow(tret, 2);
        return ColorRGB.multiply(objectColor, distanceFactor);
    }

    private void renderScreen() {
      Vector3 right = Vector3.crossProduct(dc, up);

        // calculate delta_x and delta_y
        Vector3 delta_x = new Vector3(right.x/screenHeight, right.y/screenHeight, right.z/
        screenHeight);
        Vector3 delta_y = new Vector3(up.x/screenHeight, up.y/screenHeight, up.z/screenHeight);
        //Vector3.multiply(up, 1/screenHeight);

        // go through all points of the image plane
        for (int y = -screenHeight/2; y < screenHeight/2-1; y++) {
            for (int x = -screenWidth/2; x < screenWidth/2; x++) {
                Vector3 xdeltax = Vector3.multiply(delta_x, x);
                Vector3 ydeltay = Vector3.multiply(delta_y, y);
                Vector3 toNormalize = new Vector3(dc.x+xdeltax.x+ydeltay.x, dc.y+xdeltax.y+
                ydeltay.y, dc.z+xdeltax.z+ydeltay.z);
                Vector3 normalized = Vector3.normalize(toNormalize);
                ColorRGB color = traceRay(pc, normalized);

                int hx = x+screenWidth/2;
                int hy = screenHeight/2-1-y;
                outputWindow.setPixel(hx,hy,color);
            }
        }

        // show the result
        outputWindow.showResult();
    }

    public static void main(String[] args) {
        new RayTracing().renderScreen();
    }

}
```

9.2 Question 2

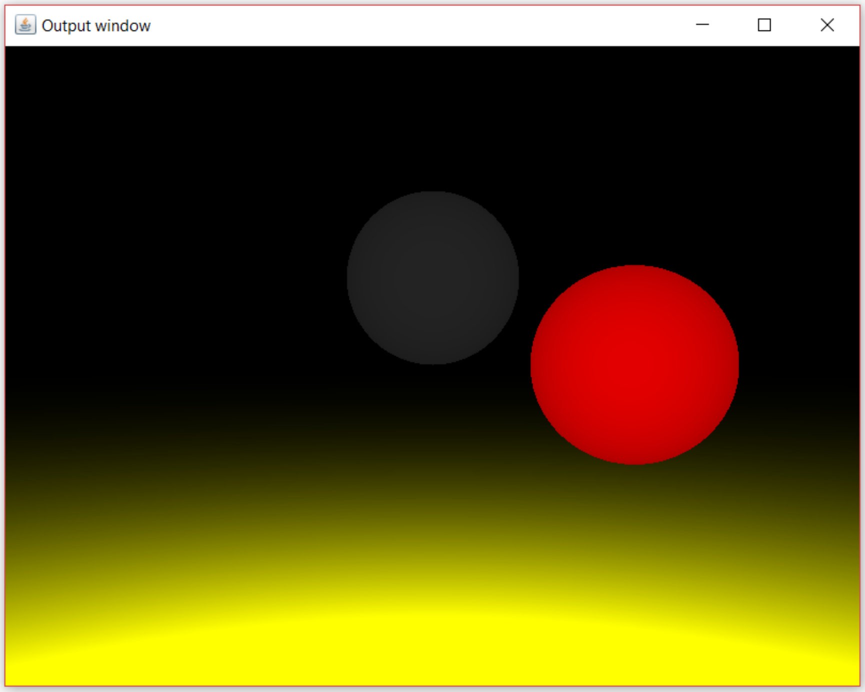
// TO DO

9.3 Question 3

Le code a directement été implementé. Il s'agit des deux denirères lignes de la méthode `traceRay` dans la classe `RayTracing`

```java
double distanceFactor = 120/Math.pow(tret, 2);
return ColorRGB.multiply(objectColor, distanceFactor);
```

Figure 11: Résultat obtenu



10 Ray tracing (part 2)

Tous les codes de cette leçon sont dans le projet `raytracing-extended` qui est basé sur le projet `raytracing` implémenté dans la leçon précédente.

10.1 Question 1

On va ici pour chaque étape décrire les modifications apportées au code, sans mettre les codes complets, pour les codes complets il suffira de se réferrer aux codes sources.

Dans les modifictions que nous allons apporter au code, nous allons devoir étendre la méthode `traceRay` de la classe `RayTracing`. Pour cela nous allons juste après le code qui permets de tester les collisions créer quatre variables qui nous servirons tout au long de ces modifications ;

```
Vector3 ip = Vector3.add(rayPosition, Vector3.multiply(rayDirection, tret));
Vector3 normal = Vector3.normalize(hittedObject.getNormal(ip));

ColorRGB finalColor = new ColorRGB(0, 0, 0);
ColorRGB objectColor = hittedObject.getColor(ip);
```

10.1.1 Diffuse reflection and ambient light

On commence par créer une classe qui représente une lampe.

Le code se trouve dans le projet `raytracing-extended` dans la classe `Lamp.java`.

```java
/**
 * Class for representing a lamp
 *
 * @author DenisM
 * @version December 2016
 */
public class Lamp {
    public Vector3 pos;
    public ColorRGB color;

    public Lamp(Vector3 pos, ColorRGB color) {
        this.pos = pos;
        this.color = color;
    }
}
```

Une `LinkedList` de lampes est créée au début de `RayTracing`, ainsi qu'une couleur pour la lumière ambiante.

```java
LinkedList<Lamp> lamps=new LinkedList<>();
private ColorRGB ambientLight;
```

Dans le constructeur de RayTracing on peut alors ajouter des lampes et définir la couleur de la lumière ambiante.

```java
lamps.add(new Lamp(new Vector3(-10, 10, 10), new ColorRGB(0.9, 0.9, 0.9)));
lamps.add(new Lamp(new Vector3(5, 10, 10), new ColorRGB(0.5, 0.5, 0.5)));
ambientLight = new ColorRGB(0.3, 0.3, 0.3);
```

Les méthodes `getNormal` dans les classes représentant les objets avaient déjà été implémentées lors de la leçon précédente. Par exemple, voici la méthode pour la classe `SphereObject` :

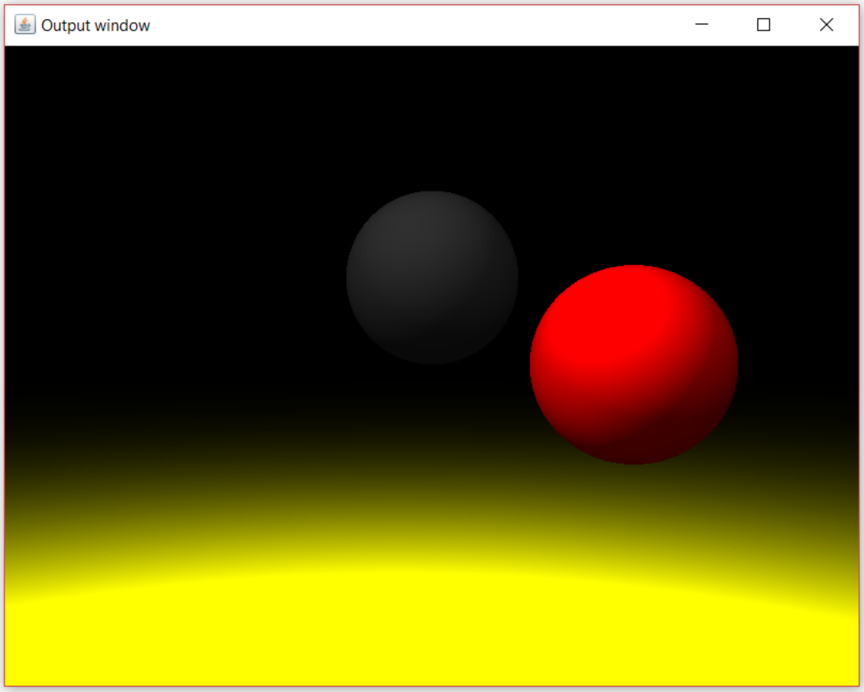
```java
public Vector3 getNormal(Vector3 pointOnSurface) {
    Vector3 min = Vector3.subst(pointOnSurface, c);
    return new Vector3(min.x/radius, min.y/radius, min.z/radius);
}
```

On peut alors parcourir la liste des lampes pour calculer la couleur. Pour le fonctionnement du code, se réferrer à l'algorithme dans les slides. Pour finir, on ajoute également la lumière ambiante.

```java
// ***** Diffuse reflection and ambient light *****
for (int i = 0; i < lamps.size(); i++) {
    Lamp light = lamps.get(i);
    Vector3 dl = Vector3.normalize(Vector3.subst(light.pos, ip));
    double cosa = Vector3.dotProduct(normal, dl)/(Vector3.getLength(normal) * Vector3.getLength
    (dl));
    if (cosa>0) {
        finalColor.add(ColorRGB.multiply(light.color, objectColor, cosa));
    }
}
```

```
finalColor.add(ColorRGB.multiply(ambientLight, objectColor));
```

Figure 12: Réflexion diffuse et lumière ambiante



10.1.2 Shadows

Pour créer des ombres, il faut que dans le code qu'on vient d'ajouter, la couleur ne soit modifié pour une lampe que si il n'y a aucun objet entre la lampe et l'endroit ou l'on se trouve.

On modifie alors la condition qui devient

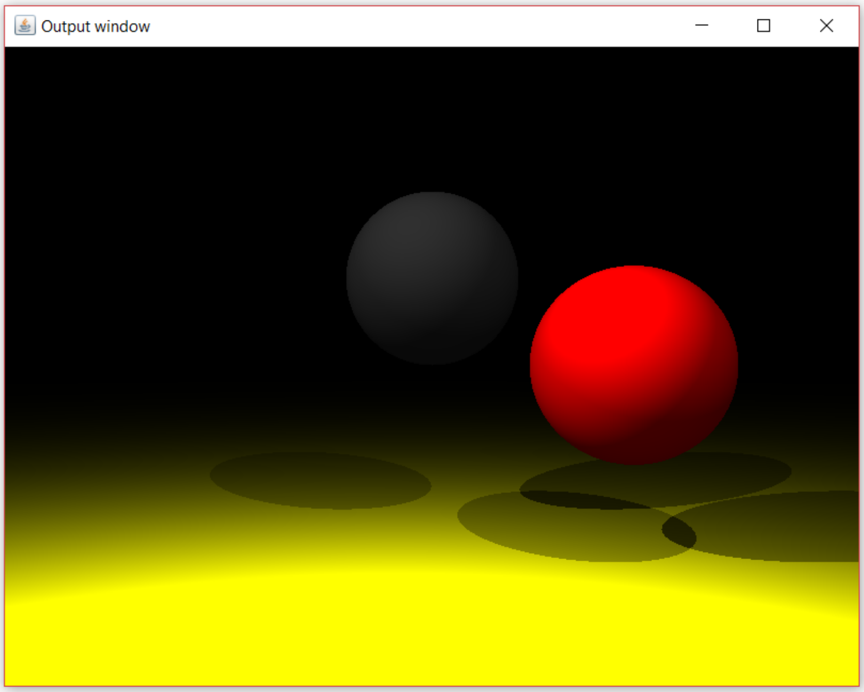
```
if (cosa>0 && !testShadow(ip, dl)) {
```

Et dans la classe `RayTracing` on implémente la méthode `testShadow`. A nouveau, le fonctionnement de cette méthode est décrit dans les slides.

```
private boolean testShadow(Vector3 pos, Vector3 dl) {
    for (int i = 0; i < objects.size(); i++) {
        RealObject here = objects.get(i);
        double thit = here.testCollision(pos, dl);

        if (thit>0) {
            return true;
```

```
        }
    }
    return false;
}
```

Figure 13: Ombres



10.1.3 Perfect reflection

Pour implémenter la réflexion, on va devoir modifier les classes représentant les objets pour que leurs constructeurs prennent un paramètre qui sera le coefficient de réflection `kr`. Voici le code ajouté dans la classe `PlaneObject` (tout le code n'y est pas, ici se trouve uniquement ce qu'on ajoute pour ajouter le coefficient de réflexion)

```
public class PlaneObject extends RealObject {
    public double kr;

    public PlaneObject(Vector3 c, Vector3 n, ColorRGB color, double kr) {
        super(color, kr);
        this.c  = c;
        this.n  = n;
        this.kr = kr;
    }

    public PlaneObject(Vector3 c, Vector3 n, ColorRGB color) {
```

```
        this(c, n, color, 0.);
    }
}
```

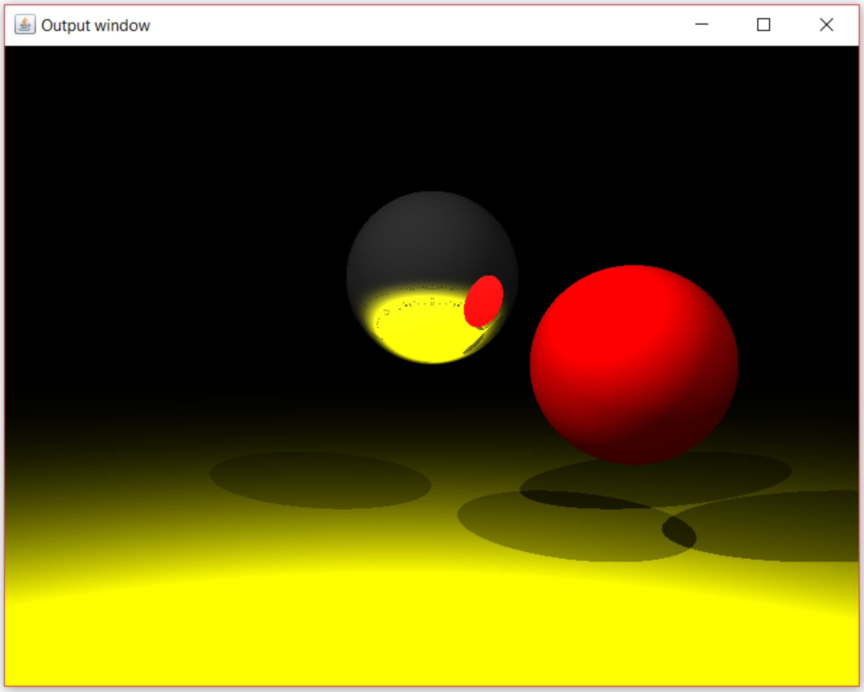
On modifie également `RealObject` pour y ajouter ce coefficient :

```java
public final ColorRGB color;
public double kr;

public RealObject(ColorRGB color, double kr) {
    this.color=color;
    this.kr = kr;
}
```

On peut alors continuer d'étendre la méthode `traceRay` pour y ajouter une condition qui, si il y a un coefficient de réflexion, va modifier la couleur en conséquence.

```java
// ***** Perfect reflexion *****
if (hittedObject.kr>0) {
    double dn =2*Vector3.dotProduct(rayDirection, normal);
    Vector3 reflectionDirection = Vector3.subst(rayDirection, Vector3.multiply(normal, dn));
    ColorRGB reflection = traceRay(ip, reflectionDirection);

    finalColor.add(ColorRGB.multiply(reflection, hittedObject.kr));
}
```

Figure 14: Réflexion parfaite



10.1.4 Texturing

Pour créer des surfaces avec des textures, on crée une classe `TexturizedPlaneObject` qui étends `PlaneObject`.

Le code se trouve dans le projet `raytracing-extended` dans la classe `TexturizedPlaneObject.java`.

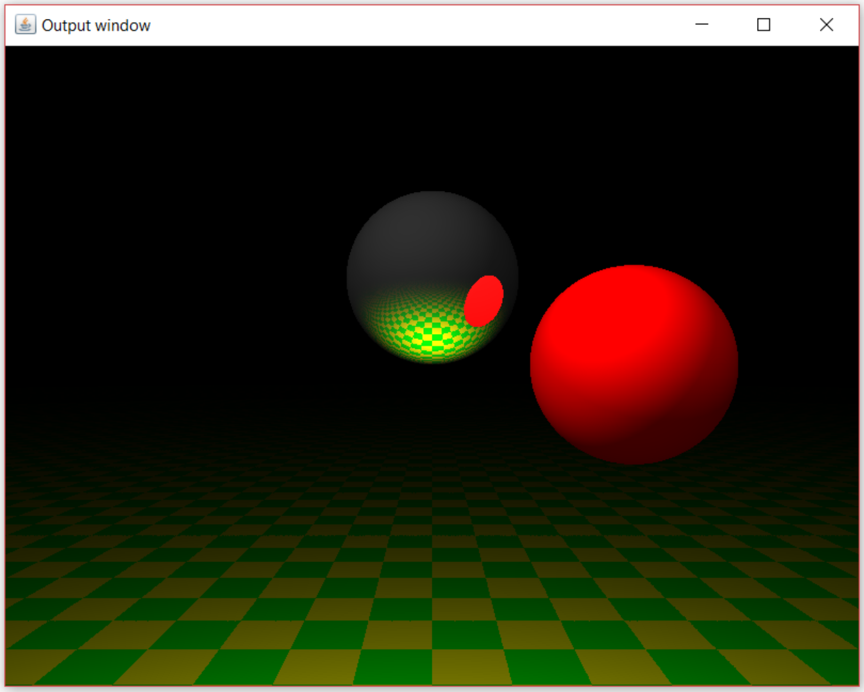
```java
/**
 * Class for a texturized plane
 *
 * @author DenisM
 * @version December 2016
 */
public class TexturizedPlaneObject extends PlaneObject {
    public final Vector3 c;
    public final Vector3 n;
    public final Vector3 v1;
    public final Vector3 v2;
    public double kr;

    public TexturizedPlaneObject(Vector3 c, Vector3 v1, Vector3 v2, ColorRGB color) {
        this(c, v1, v2, color, 0.);
    }

    public TexturizedPlaneObject(Vector3 c, Vector3 v1, Vector3 v2, ColorRGB color, double kr)
    {
```

```java
        super(c, Vector3.normalize(Vector3.crossProduct(v1, v2)), color, kr);
        this.c  = c ;
        this.n  = Vector3.normalize(Vector3.crossProduct(v1, v2));
        this.v1 = v1;
        this.v2 = v2;
        this.kr = kr;
    }

    public double testCollision(Vector3 rayPosition,Vector3 rayDirection) {
        double dn = Vector3.dotProduct(rayDirection, this.n);
        if (dn==0) return -1;
        Vector3 min = Vector3.subst(this.c, rayPosition);
        return Vector3.dotProduct(min, this.n)/dn;
    }

    public Vector3 getNormal(Vector3 pointOnSurface) {
        return this.n;
    }

    public ColorRGB getColor(Vector3 pointOnSurface) {
        Vector3 pc = Vector3.subst(pointOnSurface, c);

        double u = (Vector3.dotProduct(pc, v1))/(Math.pow(Vector3.getLength(v1), 2));
        double v = (Vector3.dotProduct(pc, v2))/(Math.pow(Vector3.getLength(v2), 2));

        return ((int) (Math.floor(u)+Math.floor(v)) & 1) == 0 ? new ColorRGB(1, 1, 0) : color;
    }
}
```

On peut alors modifier notre scène dans le constructeur

```java
objects.add(new PlaneObject(new Vector3(0, -4, 0), new Vector3(0, 1, 0), yellow));
// devient
objects.add(new TexturizedPlaneObject(new Vector3(0, -4, 0), new Vector3(1, 0, 0), new Vector3
(0, 0, 1), green));
```

Figure 15: Textures



10.1.5 Parallelograms

Pour créer des parallèlogrames, on créer une classe `ParallelogramObject`. Son fonctionnement est semblable au fonctionnement des plans infinis. La principale différence est la méthode `testCollision` dont le fonctionnement est développé dans les slides.

Le code se trouve dans le projet `raytracing-extended` dans la classe `ParallelogramObject.java`.

```java
/**
 * Class for a parallelogram object
 *
 * @author DenisM
 * @version December 2016
 */
public class ParallelogramObject extends RealObject {
    public final Vector3 c;
    public final Vector3 n;
    public final Vector3 v1;
    public final Vector3 v2;
    public double kr;

    public ParallelogramObject(Vector3 c, Vector3 v1, Vector3 v2, ColorRGB color) {
        this(c, v1, v2, color, 0.);
    }

    public ParallelogramObject(Vector3 c, Vector3 v1, Vector3 v2, ColorRGB color, double kr) {
```

```java
        super(color, kr);
        this.c  = c;
        this.v1  = v1;
        this.v2  = v2;
        this.n = Vector3.normalize(Vector3.crossProduct(v1, v2));
        this.kr = kr;
    }

    public double testCollision(Vector3 rayPosition,Vector3 rayDirection) {
        double dn = Vector3.dotProduct(rayDirection, this.n);
        if (dn==0) return -1;
        Vector3 min = Vector3.subst(this.c, rayPosition);
        double thit = Vector3.dotProduct(min, this.n)/dn;
        Vector3 point = Vector3.add(rayPosition, Vector3.multiply(rayDirection, thit));
        Vector3 pmoinsc = Vector3.subst(point, c);
        double u = Vector3.dotProduct(pmoinsc, v1)/ Math.pow(Vector3.getLength(v1), 2);
        double v = Vector3.dotProduct(pmoinsc, v2)/ Math.pow(Vector3.getLength(v2), 2);
        if (u>=0 && u<=1 && v>=0 && v<=1) return thit;
        return -1;
    }

    public Vector3 getNormal(Vector3 pointOnSurface) {
        return this.n;
    }

    public ColorRGB getColor(Vector3 pointOnSurface) {
        return this.color;
    }
}
```

On peut alors ajouter un parallèlogramme dans notre scène :

```java
objects.add(new ParallelogramObject(new Vector3(-5, -1, 0), new Vector3(0, 0, 5), new Vector3
(0, 5, 0), green));
```

Figure 16: Parallèlogrames

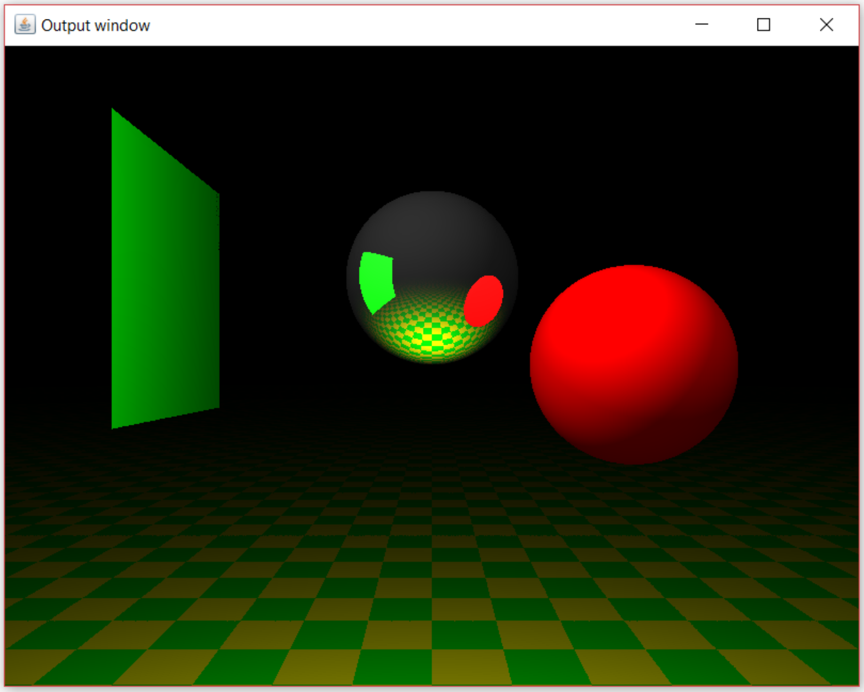
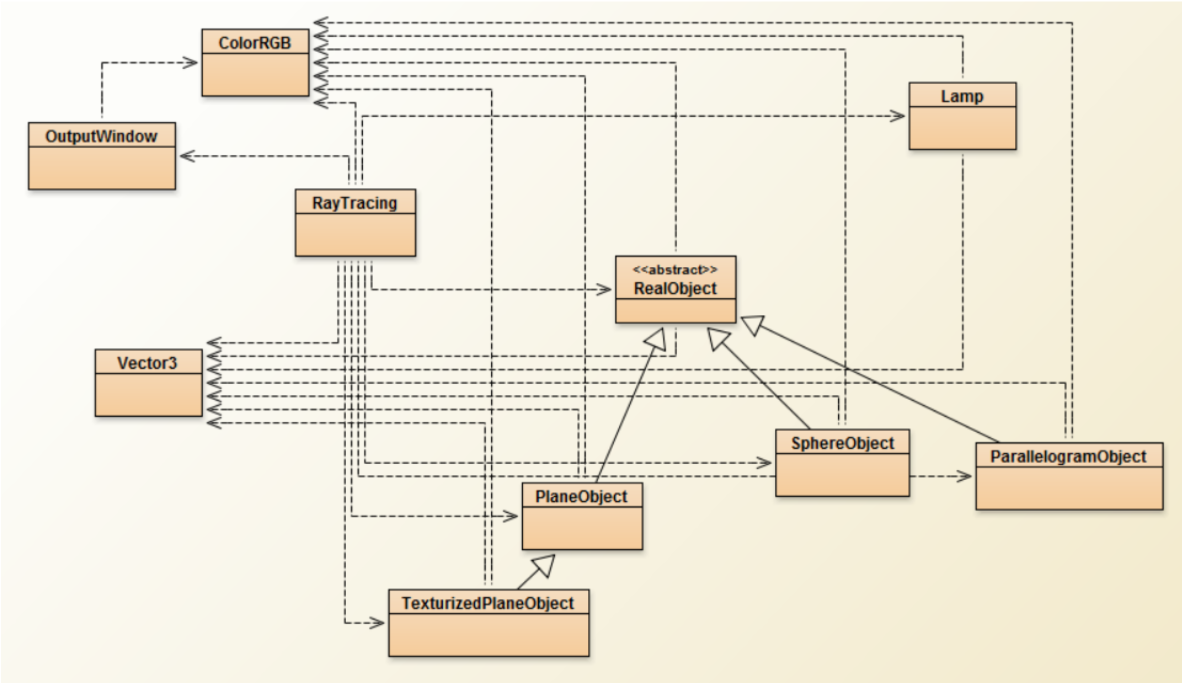


Figure 17: Diagramme de la structure de raytracing-extended



10.2 Question 2

10.2.1 Point 1

Parce que si $cos(\alpha)$ est plus petit que 0, on est déjà sur que la lampe n'a pas d'influence sur l'objet. Il est donc inutile d'appeler la méthode `testShadow` qui ferais des calculs inutiles.

10.2.2 Point 2

Si l'on place plusieurs objetx avec une réflexion parfaite, on peut avoir des réflexions infinies et donc ne jamais s'arrêter. On peut d'une part arrêter de modifier la couleur si elle est déjà totalement blanche, et d'autre part on peut implémenter un compteur qui limite le nombre de réflexions en chaine à calculer.

11 Numerical differentiation and solution of nonlinear equations

11.1 Question 1

$$
\begin{aligned}
p(x) &= \sum_{i=1}^{3} f(x_i)\Phi_i(x \\
f'(x) &= \sum_{i=1}^{3} f(x_i)\Phi_i'(x) \\
\Phi_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = (x^2-(x_2+x_3)x+x_2x_3)\frac{1}{(x_1-x_2)(x_1-x_3)} \\
\Phi_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} \\
\Phi_3(x) &= \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \\
\Phi_1(x)' &= \frac{2x-(x_2+x_3)}{(x_1-x_2)(x_1-x_3)} \\
\Phi_2(x)' &= \frac{2x-(x_1+x_3)}{(x_2-x_1)(x_2-x_3)} \\
\Phi_3(x)' &= \frac{2x-(x_1+x_2)}{(x_3-x_2)(x_3-x_1)} \\
f'(x) &= f(x_1)\frac{2x-(x_2+x_3)}{(x_1-x_2)(x_1-x_3)} + f(x_2)\frac{2x-(x_1+x_3)}{(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{2x-(x_1+x_2)}{(x_3-x_2)(x_3-x_1)}
\end{aligned}
\tag{42}
$$

$$
\begin{aligned}
Erreur &= \frac{f^{(3)}(\xi(x))}{3!}(\prod_{i=1}^{3}(x-x^{(i)}))' \\
&= \frac{f^{(3)}(\xi(x))}{3!}((x-x_1)(x-x_2)(x-x_3))' \\
&= \frac{f^{(3)}(\xi(x))}{3!}(x^3-(x_1+x_2+x_3)x^2+(x_1x_2+(x_1-x_2)x_3)x-x_1x_2x_3)' \\
&= \frac{f^{(3)}(\xi(x))}{3!}(3x^2-2(x_1+x_2+x_3)x+x_1x_2+(x_1+x_2)x_3)
\end{aligned}
\tag{43}
$$

Si l'on remplace x_1, x_2 et x_3 par $x_1 = x-h$, $x_2 = x$ et $x_3 = x+h$, ont obtient :

$$
\begin{aligned}
f'(x) = f(x-h)&\frac{2x-(x+x+h)}{(x-h-x)(x-h-(x+h))} + f(x)\frac{2x-(x+h+x+h)}{(x-(x-h))(x-(x+h))} \\
&+ f(x+h)\frac{2x-(x-h+x)}{(x+h-x)(x+h-(x-h))} \\
&= f(x-h)\frac{-h}{2h^2} + f(x+h)\frac{h}{2h^2} \\
&= \frac{f(x+h)-f(x-h)}{2h}
\end{aligned}
\tag{44}
$$

L'interpolation devient alors identique à la méthode du "two sided centered differencing" vue au cours.

11.2 Question 2

11.2.1 Point 1

$$\begin{aligned} T(h) &= \frac{16S(h) - S(2h)}{15} = \frac{2^4 S(h) - S(2h)}{2^4 - 1} \\ P(h) &= \frac{2^6 T(h) - T(2h)}{2^6 - 1} = \frac{64T(h) - T(2h)}{63} \end{aligned} \tag{45}$$

11.2.2 Point 2

$$\begin{cases} R_0(h) = D(h) \\ R_i(h) = \frac{2^{2i} R_{(i-1)}(h) - R_{(i-1)}(2h)}{2^{2i} - 1} \end{cases} \tag{46}$$

11.2.3 Point 3

$$\begin{cases} R_0(h) = D(h) \\ R_i(h) = \frac{2^{2i} R_{(i-1)}(h) - R_{(i-1)}(\alpha h)}{2^{2i} - 1} \end{cases} \tag{47}$$

// A CONFIRMER

11.3 Question 3

11.3.1 Point 1

f differentiable but f' is expensive to evaluate Si f est diiférentiable mais chèr à évaluer, on peut utiliser la 'secant method'. Avec cette méthode on va approximer en ré-utilisant ce qu'on a déjà calculé lors des itérations précédentes.

$$\begin{aligned} f'(x_k) &\approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \\ x_{k+1} &= x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \end{aligned} \tag{48}$$

f is continuous but not differentiable Si f est continue mais pas différentiable, on peut utiliser le 'bisection algorithm'.

$$\begin{aligned} f(a) < u < f(b) \quad &\text{ou} \quad f(a) > u > f(b) \\ &\exists z \in [a, b] \text{ tq } f(z) = u \end{aligned} \tag{49}$$

11.3.2 Point 2

// TO DO

11.3.3 Point 3

$$\begin{aligned} Cost(f(x)) &= t \\ Cost(f'(x)) &= \frac{t}{2} \end{aligned} \tag{50}$$

On va calculer pour chaque méthode combien de fois on doit calculer $f(x)$ et $f'(x)$ lors d'une itération pour pouvoir estimer le cout de chaque méthode. La méthode qui aura le cout d'une itération le plus bas sera préférée.

Newton-Raphson Avec la méthode de Newton-Raphson, lors d'une itération on calcule une fois $f(x)$ et une fois $f'(x)$

$$\begin{aligned} Cost(Newton - Raphson) &= 1 \cdot Cost(f(x)) + 1 \cdot Cost(f'(x)) \\ &= t + \frac{t}{2} = \frac{3t}{2} \end{aligned} \tag{51}$$

Secant method Avec la méthode sécante, lors d'une itération on calcule trois fois $f(x)$ et on ne calcule jamais $f'(x)$.

$$\begin{aligned} Cost(Secant) &= 3 \cdot Cost(f(x)) + 0 \cdot Cost(f'(x)) \\ &= 3t \end{aligned} \tag{52}$$

Pour les couts donnés, la méthode sécante coute deux fois plus que la méthode de Newton-Raphson, qui sera donc préférée dans ce cas.

11.4 Question 4

11.4.1 Point 1

Le code se trouve dans le projet `littleclasses` dans la classe `NewtonRaphson.java`.

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
/**
 * Newton-Raphson method
 * Lecon 11 - Question 4
 *
 * @author DenisM
 * @version December 2016
 */
public class NewtonRaphson {
    private static double a = 1.;
    private static double b = 0.;
    private static double c = -2.;
    private static double d = 2.;
    private static int max_iterations = 200;

    /**
     * Renvoie f(x) = ax^3 + bx^2 + cx + d
     */
    private static double f(double x) {
        return a*Math.pow(x, 3) + b*Math.pow(x, 2) + c*x + d;
    }

```

```
    /**
     * Renvoie f'(x)
     * On aurais pu utiliser une des méthodes vues à la lecon 8 pour
     * calculer cette dérivée automatiquement
     */
    private static double der(double x) {
        return 3*a*Math.pow(x, 2) + b*x + c;
    }

    /**
     * Renvoie x_{x+1} pour un x_k donné
     */
    private static double getIteration (double oldx) {
        return oldx-(f(oldx)/der(oldx));
    }

    /**
     * Calcule la racine pour un x0 donné et une erreur acceptable
     * Renvoie un tableau avec les données suivantes
     * [0] : la racine calculée
     * [1] : f(racine calculée)
     * [2] : le nombre d'itérations pour trouver ce résultat
     * [3] : x0 (= argument recu)
     * [4] : erreur acceptable (= argument recu)
     */
    public static double[] process(double x0, double acceptable_error) {
        int iterations = 0;

        double xhere = x0;
        double fhere = f(x0);
        while (Math.abs(fhere)>acceptable_error && iterations<max_iterations) {
            double newx = getIteration(xhere);
            double newf = f(newx);
            xhere = newx;
            fhere = newf;
            iterations++;
        }

        double[] ret = {xhere, fhere, iterations, x0, acceptable_error};
        return ret;
    }

    /**
     * Affiche le tableau renvoyé par la méthode process proprement dans la console
     */
    public static void displayResult(double[] res) {
        System.out.println("===== Newton-Raphson \n\tx0 = "+res[3]+";\n\tacceptable_error = "+res[4]+"\n");
        if (Math.abs(res[1])<=res[4]) {
            System.out.println("\t"+res[2]+" itérations,\n\tracine : "+res[0]);
        }
        else {
            System.out.println("\t"+res[2]+" itérations, rien trouvé de concluant");
        }
        System.out.println("\tf("+res[0]+") = "+res[1]);
    }

    /**
     * Enregistre dans un fichier texte le nombre d'itérations nécessaire pour chaque x0
     * Ne mets pas les points ou le nombre d'itérations = nombre max d'itérations
     * (on considère que dans ce cas la méthode ne converge pas)
     */
    public static void saveData(double[][] data) {
        String filename = "newton_raphson.txt";
        PrintStream stream = null;

        try {
            FileOutputStream fileWriter = new FileOutputStream(filename, true);
            stream = new PrintStream(fileWriter);
```

```java

            for (int i = 0; i < data.length; i++) {
                if (Math.abs(data[i][1])<=data[i][4]) {
                    stream.println(data[i][3]+" "+data[i][2]);
                }
            }
        } catch (IOException e) {
            System.out.println("Une erreur s'est produite :/");
        } finally {
            if (stream != null) stream.close();
        }
        System.out.println("Et voila, tout se trouve dans "+filename);
    }

    /**
     * Affiche la liste des points ou il n'y a pas de convergence
     */
    public static void displayNoConvergence(double[][] datas) {
        for (int i = 0; i < datas.length; i++) {
            if (Math.abs(datas[i][1])>datas[i][4]) {
                System.out.println(datas[i][3]);
            }
        }
    }

    public static void main(String[] args) {
        double acceptable_error = 0.;
        double begin = -10.;
        double end = 10;
        int points = 100;

        double step = Math.abs(end-begin)/(points-1);
        double[][] datas = new double[points][5];

        for (int i = 0; i < points; i++) {
            double x0 = begin+i*step;
            datas[i] = process(x0, acceptable_error);
            displayResult(datas[i]);
        }

        saveData(datas);
    }
}
```

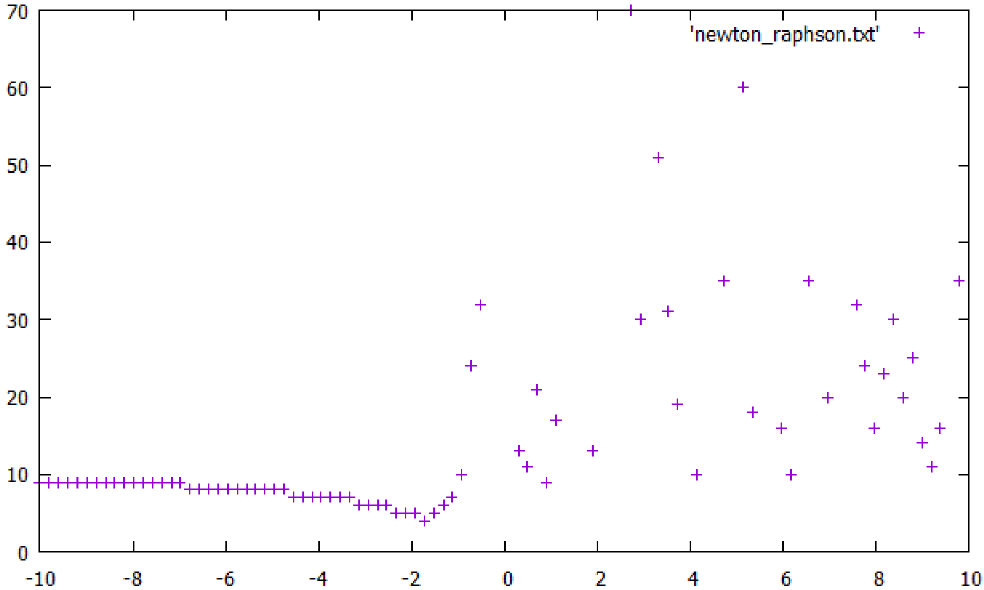
Quelques commentaires sur le code ;

- La méthode `getIteration` calcule une itération
- `process` calcule la racine pour un x_0 donné et une erreur acceptable
- En plottant le fichier créé par la méthode `saveData` on peut voir le nombre d'itérations nécessaire en fonction du x_0

11.4.2 Point 2

Avec la méthode `main`, on va calculer la racine avec la méthode de Newton-Raphson pour 100 points répartis uniformément entre -10 et 10. On peut voir sur le graphique le nombre d'itérations nécessaires en fonction du x_0 choisi.

Figure 18: Nombre d'itérations en fonction du x_0 choisi



A l'aide de la méthode `displayNoConvergence` on peut récupérer les x_0 pour lesquels Newton-Cotes ne converge pas. Voici la liste des points qui après 200 itérations n'ont pas trouvé de solution :

-0.30303030303030276
-0.10101010101010033
0.10101010101010033
1.3131313131313131
1.5151515151515156
1.7171717171717162
2.121212121212121
2.3232323232323235
2.525252525252524
3.1313131313131315
3.9393939393939394
4.3434343434343425
4.545454545454545
4.94949494949495
5.555555555555555
5.757575757575758
6.363636363636363
6.767676767676768
7.171717171717173
7.373737373737374
9.595959595959595
10.0

11.4.3 Point 3

On peut déduire du graphique qu'au plus x_0 est loin de la racine, au plus le nombre d'itérations nécessaires est élevé.

11.5 Question 5

11.5.1 Point 1

// TO DO

11.5.2 Point 2

// TO DO

12 Initial value problems

12.1 Question 1

12.1.1 Point 1

$$\begin{aligned} f(x) &= \frac{1}{C-x} + x \\ f(2) &= 1 \\ 1 &= \frac{1}{C-2} + 2 \\ C &= 1 \end{aligned} \tag{53}$$

Pour que l'IVP soit stable, il faut que $\frac{\delta F}{\delta f}|_{(x_i,\zeta_i)} < 0$

$$\begin{aligned} \frac{\delta F}{\delta f}|_{(x_i,\zeta_i)} &= \frac{\delta}{\delta f}(1 + (x - f(x))^2) \\ &= (2f(x) - 2x)|_{(x_i,\zeta_i)} \\ (2f(x) - 2x) &< 0 \\ f(x) - x &< 0 \\ \frac{1}{1-x} + x - x &< 0 \end{aligned} \tag{54}$$

Si on vérifie cette dernière inéquation pour tous les x d'intéret (à savoir $x \geq 2$), on constate qu'elle est toujours vérifiée. L'IVP est donc stable pour les valeurs d'intérêt.

12.1.2 Point 2

$$\begin{aligned} f_2 &= 1 \\ f_{i+1} &= h(1 + (x_i - f_i)^2) + f_i \end{aligned} \tag{55}$$

Pour que les itérations soient stable, il faut que $h < \frac{-2}{\frac{\delta F}{\delta f}|_{(x_i,\zeta_i)}}$

$$\begin{aligned} h &< \frac{-2}{\frac{\delta F}{\delta f}|_{(x_i,\zeta_i)}} \\ &< \frac{-2}{2f(x) - 2x} \\ &< \frac{-1}{f(x) - x} \\ &< \frac{-1}{\frac{1}{1-x} + x - x} \\ &< x - 1 \end{aligned} \tag{56}$$

Il faut donc choisir un h plus petit que 1 pour que les itérations soient stable.

12.1.3 Point 3

Le code se trouve dans le projet `littleclasses` dans la classe `ForwardEuler.java`.

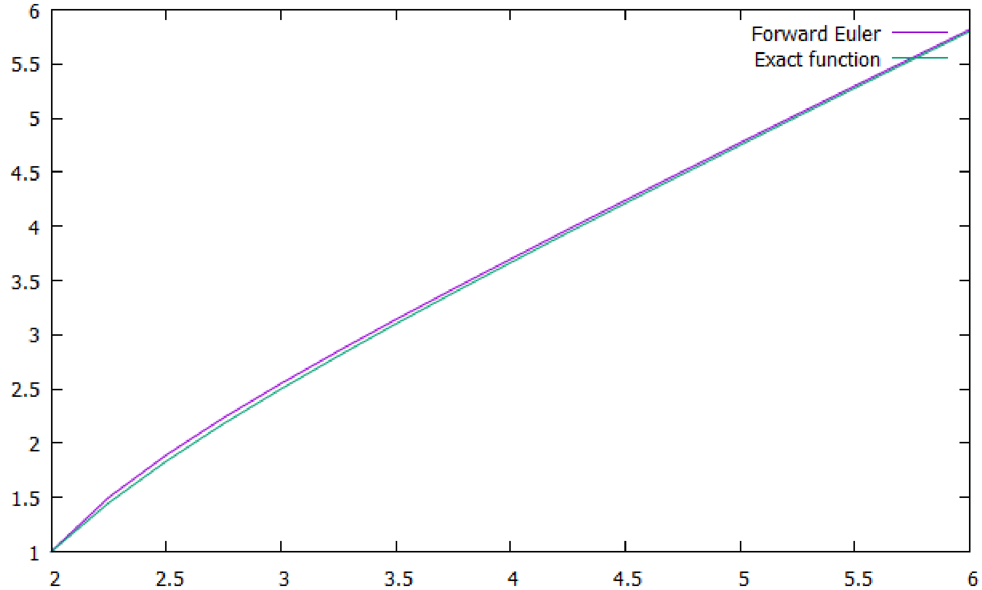
```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

/**
 * Calculates the Forward Euler
 *
 * @author DenisM
 * @version December 2016
 */
public class ForwardEuler {
    // Si on a la fonction exacte, mettre à true pour aussi calculer les points
    // via la fonction exacte
    // Permets par exemple de créeer un graphe comparatif
    private static boolean compareWithExactFunction = true;

    /**
     * Renvoie le résultat de la dérivée pour des paramètres donnés
     * F pour xi et fi
     */
    public static double F(double xi, double fi) {
        return 1 + Math.pow((xi - fi), 2);
    }

    /**
     * Si on a la fonction réelle, on peut la renvoyer ici (permets de créer un
     * graphe comparatif par exemple)
     */
    public static double exactFunction(double x) {
        return x+1/(1-x);
    }

    /**
     * Renvoie le résultat d'une itération
     * f_{i+1} pour x_i et f_i donnés
     */
    public static double getIteration(double xi, double fi, double h) {
        return h * F(xi, fi) + fi;
    }

    /**
     * Applique la méthode Forward Euler
     * Valeur initiale : f(a) = v
     * h: time step
     * maxX : valeur ou l'on veut évaluer f
     * Renvoie un tableau de tous les points évalués, la dernière entrée du tableau
     * est donc {maxX, f(maxX)}
     * Si compareWithExactFunction est à true, le tableu renvoyé contient une 3e
     * colonne avec la valeur exacte pour x
     */
    public static double[][] process(double a, double v, double h, double maxX) {
        int points = (int) 1+ (int) ((maxX-a)/h);

        int width = (compareWithExactFunction) ? 3 : 2;
        double[][] ret = new double[points][width];

        ret[0][0] = a;
        ret[0][1] = v;
        if (compareWithExactFunction) ret[0][2] = v;

        for (int i = 1; i < points; i++) {
            double xi = a+((maxX-a)*i/(points-1));
            double fi = getIteration(ret[i-1][0], ret[i-1][1], h);

```

```java
            ret[i][0] = xi;
            ret[i][1] = fi;
            if (compareWithExactFunction) ret[i][2] = exactFunction(xi);
        }

        return ret;
    }

    public static void saveData(double[][] data) {
        String filename = "forward_euler.txt";
        PrintStream stream = null;

        try {
            FileOutputStream fileWriter = new FileOutputStream(filename, true);
            stream = new PrintStream(fileWriter);

            for (int i = 0; i < data.length; i++) {
                for (int j = 0; j < data[i].length; j++) {
                    stream.print(data[i][j]+" ");
                }
                stream.println();
            }
        } catch (IOException e) {
            System.out.println("Une erreur s'est produite :/");
        } finally {
            if (stream != null) stream.close();
        }
        System.out.println("Et voila, tout se trouve dans "+filename);
    }

    public static void main(String[] args) {
        // Point connu : f(a) = v
        double a = 2;
        double v = 1;

        double maxX = 6.; // Valeur max de x jusqu'ou il faut calculer
        double h = 0.25; // Time step h
        System.out.println("h = "+h);

        double[][] data = process(a, v, h, maxX);

        for (int i = 0; i < data.length; i++) {
            System.out.println(data[i][0]+" - "+data[i][1]);
        }
        saveData(data);
    }
}
```

Pour adapter le code à une autre fonction, il faut modifier les variables dans la méthode `main` ainsi que la méthode `F`. Il est également possible de mettre la fonction exacte dans la méthode `exactFunction` et de mettre alors le booléen `compareWithExactFunction` à `true`. Dans ce cas, le tableau renvoyé par la méthode `process` contiendra une troisième colonne avec les évaluations pour ces x de la fonction exacte. On peut alors créer un graphe pour comparer les résultats obtenus par Forward Euler et les valeurs exactes avec gnuplot par exemple via la commande suivante :

```
plot 'forward_euler.txt' using 1:2 with lines title 'Forward Euler', '' using 1:3 with lines
title 'Exact function'
```

Figure 19: Forward Euler avec $h = 0.25$



12.2 Question 2

$$\begin{cases} f'(x) = x + \sqrt{f(x)} \\ f(1) = 2 \end{cases} \tag{57}$$

Pour l'implémentation de Forward Euler, il nous suffit d'adapter l'implémentation de la question précédente en modifiant les variables de la valeur initiale et la méthode `F`. Pour le Classical Runge-Kutta on peut très simplement l'implémenter en se basant sur la structure de notre implémentation pour Forward Euler.

Le code se trouve dans le projet `littleclasses` dans la classe `RK4.java`.

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

/**
 * Calculates the Classical Runge Kutta
 *
 * @author DenisM
 * @version December 2016
 */
public class RK4 {
    /**
     * Renvoie le résultat de la dérivée pour des paramètres donnés
     * F pour xi et fi
     */
    public static double F(double xi, double fi) {
        return xi+Math.sqrt(fi);
    }

    /**
```

```java
     * Renvoie le résultat d'une itération
     * f_{i+1} pour x_i et f_i donnés
     */
    public static double getIteration(double xi, double fi, double xi1, double h) {
        double k1 = F(xi, fi);
        double k2 = F(xi+(h/2), fi+(h/2)*k1);
        double k3 = F(xi+(h/2), fi+(h/2)*k2);
        double k4 = F(xi1, fi+h*k3);
        return fi+ (h/6) * (k1+2*k2+2*k3+k4);
    }

    /**
     * Applique la méthode Classic Runge Kutta
     * Valeur initiale : f(a) = v
     * h: time step
     * maxX : valeur ou l'on veut évaluer f
     * Renvoie un tableau de tous les points évalués, la dernière entrée du tableau
     * est donc {maxX, f(maxX)}
     */
    public static double[][] process(double a, double v, double h, double maxX) {
        int points = (int) 1+ (int) ((maxX-a)/h);

        double[][] ret = new double[points][2];

        ret[0][0] = a;
        ret[0][1] = v;

        for (int i = 1; i < points; i++) {
            double xi = a+((maxX-a)*i/(points-1));
            double fi = getIteration(ret[i-1][0], ret[i-1][1], xi, h);

            ret[i][0] = xi;
            ret[i][1] = fi;
        }

        return ret;
    }

    public static void saveData(double[][] data) {
        String filename = "classic_runge_kutta.txt";
        PrintStream stream = null;

        try {
            FileOutputStream fileWriter = new FileOutputStream(filename, true);
            stream = new PrintStream(fileWriter);

            for (int i = 0; i < data.length; i++) {
                for (int j = 0; j < data[i].length; j++) {
                    stream.print(data[i][j]+" ");
                }
                stream.println();
            }
        } catch (IOException e) {
            System.out.println("Une erreur s'est produite :/");
        } finally {
            if (stream != null) stream.close();
        }
        System.out.println("Et voila, tout se trouve dans "+filename);
    }

    public static void main(String[] args) {
        // Point connu : f(a) = v
        double a = 1;
        double v = 2;

        double maxX = 3.; // Valeur max de x jusqu'ou il faut calculer
        double h = 0.125; // Time step h
        System.out.println("h = "+h);

```

```java
        /*double[][] data = process(a, v, h, maxX);

        for (int i = 0; i < data.length; i++) {
            System.out.println(data[i][0]+" - "+data[i][1]);
        }
        saveData(data);*/

        double exactSolution = 106446;
        double actualSolution = -1;
        int pointsBegin = 2;
        int points = pointsBegin;
        do {
            h = (maxX-a)/(points-1);
            double[][] data = process(a, v, h, maxX);
            actualSolution = data[data.length-1][1];
            System.out.println(h+" - "+actualSolution);
            points++;
        } while (points<pointsBegin+100 && Math.abs(Math.round(actualSolution*100)-exactSolution)>=1);
            System.out.println(actualSolution);

        System.out.println("h = "+h);
        System.out.println("points = "+points);
    }
}
```

12.2.1 Point 1

// TO DO

12.2.2 Point 2

Pour calculer le plus grand h qui nous permette d'obtenir une solution exacte arrondie à 4 décimales, on peut utiliser une boucle qui commence avec un grand h et le réduit tant que la différence entre la solution obtenue et le solution exacte est trop grande. Cette technique a été implémentée dans la méthode `calculate_biggest_h` et on peut ainsi déterminer que le plus grand h qui nous permets d'obtenir une solution exacte est 0.2.

12.2.3 Point 3

// TO DO

12.3 Question 3

$$\begin{cases} f'(x) = sin(f(x)) \\ f(0) = 1 \end{cases} \tag{58}$$

$$\begin{aligned} f(x_{i+1}) \approx f_{i+1} &= hF(x_{i+1}, f_{i+1}) + f_i \\ &= h \cdot sin(f_{i+1}) + f_i \end{aligned} \tag{59}$$

// TO DO

12.4 Question 4

12.4.1 Point 1

// TO DO

12.4.2 Point 2

// TO DO

13 Initial value problems (part 2)

13.1 Question 1

// TO DO