Conditions de distribution et de copie

Exercices (II)

Fabrice Rossi

19 décembre 2000

1 Les boucles

1.1 Analyse

Exercice 1.1 :

Donner l'organigramme du programme suivant :

Analyse

```
1  import dauphine.util.*;
2  public class Analyse {
3    public static void main(String[] args) {
4      Console.start();
5      int i,j;
6      System.out.print("Début : ");
7      i=Console.readInt();
8      System.out.print("Fin : ");
9      j=Console.readInt();
10     while (i!=j) {
11       if (i%7==0)
12         System.out.println(i+" est multiple de 7");
13       i++;
14     }
15   }
16 }
```

Que se passe-t-il si l'utilisateur entre une valeur de `i` supérieure strictement à celle de `j` ? Comment modifier le programme pour lui donner un comportement plus cohérent ?

Exercice 1.2 :

Donner l'organigramme du programme suivant :

Analyse2

```
1  import dauphine.util.*;
2  public class Analyse2 {
3    public static void main(String[] args) {
4      Console.start();
5      int n;
6      double u=1,v=1,w;
7      System.out.print("Combien de valeurs : ");
8      n=Console.readInt();
```

```
9       System.out.println("1 : "+u);
10      System.out.println("2 : "+v);
11      for(int i=3;i<=n;i++) {
12        w=v+2*u;
13        System.out.println(i+" : "+w);
14        u=v;
15        v=w;
16      }
17    }
18  }
```

Que fait le programme ?

Exercice 1.3 :

Pour chacune des boucles suivantes, indiquer la valeur de `m` en fin de boucle et le nombre d'exécutions de la boucle. Remplacer toutes les boucles par des boucles `while` donnant les mêmes résultats.

1. ```
   m = 2;
   for (int i = 1; i<=10; i++) {
     m = m +2;
   }
   ```
2. ```
   i = 1; m = 1;
   do {
     m = i;
     i++;
   }  while (i <= 10);
   ```
3. ```
   m = 1;
   i = 1;
   do {
     m = m + 2;
     i++;
   }  while (i < 5);
   ```
4. ```
   m = 0;
   i = 1;
   do {
     m = m + 2;
     i++;
   }  while (i > m);
   ```
5. ```
   m = 2;
   i = 1;
   do {
     m = m + 2;
     i = i + 1;
   }  while (i <= m);
   ```

Exercice 1.4 :

Pour le programme suivant, indiquez :
   ```

1. l'évolution des variables `j`, `u` et `k` ;
2. l'affichage produit par le programme.

Analyse5

```
1  public class Analyse5 {
2    public static void main(String[] arg) {
3      int j=0,k=0;
4      do {
5        if(j%2==1) {
6          int u=j;
7          while(u>0) {
8            k=k+1;
9            u=u/2;
10         }
11       } else {
12         for(int v=-1;v<j;v++) {
13           k=k+2;
14         }
15       }
16       j=j+1;
17     } while(j<5);
18     System.out.println(k);
19   }
20 }
```

Exercice 1.5 :

Donner l'affichage produit par le programme suivant :

Analyse6

```
1  public class Analyse6 {
2    public static void main(String[] arg) {
3      int i=3,j=0,k=0;
4      while(i>0) {
5        if(j%3==0) {
6          i=i+1;
7        } else {
8          i=i-1;
9        }
10       j=j+1;
11       int u=-2;
12       do {
13         k=k+1;
14         u=u+j/2+1;
15       } while(u<2*i);
16       System.out.println(k);
17     }
18   }
19 }
```

Exercice 1.6 :

On considère le programme suivant :

Analyse3

```
1  import dauphine.util.*;
2  public class Analyse3 {
3    public static void main(String[] arg) {
4      Console.start();
5      int n=Console.readInt();
6      int p;
7      for(int i=0;i<n;i++) {
8        System.out.println(i);
9        p=0;
10       while(p<i*i) {
11         System.out.println(p);
12         if (p%2==0) {
13           p=p+1+i/2;
14         } else {
15           p=p*2;
16         }
17       }
18       System.out.println("---");
19     }
20   }
21 }
```

Dessiner l'organigramme du programme. Donner l'affichage produit par le programme quand l'utilisateur saisit la valeur 6.

Exercice 1.7 :

On considère le programme suivant :

Analyse4

```
1  import dauphine.util.*;
2  public class Analyse4 {
3    public static void main(String[] arg) {
4      Console.start();
5      int n=Console.readInt();
6      int p=0;
7      int k=1;
8      while(k<n) {
9        p=p+1;
10       k=k*10;
11     }
12     p=p-1;
13     System.out.println(p);
14     k=k/10;
15     while(p>=0) {
16       System.out.println(n/k);
17       n=n%k;
18       k=k/10;
```

```
      p=p-1;
    }
  }
}
```

1. Dessiner l'organigramme du programme.
2. Donner l'affichage produit quand l'utilisateur saisit la valeur 27.
3. Donner l'affichage produit quand l'utilisateur saisit la valeur 3289.
4. Expliquer brièvement ce que fait le programme en justifiant la réponse proposée.

Exercice 1.8 :

Pour chacun des programmes, dessiner l'organigramme correspondant et en donner une interprétation.

1. `Programmation1.java` :

Programme1

```
import dauphine.util.*;
public class Programme1 {
  public static void main(String[] args) {
    Console.start();
    double s=1;
    int n;
    do {
      System.out.print("Valeur du paramètre n : ");
      n=Console.readInt();
    } while (n<=0);
    for (int i=2;i<=n;i++) {
      int t=i;
      for (int j=2;j<=i;j++)
        t=t*i;
      s=s+1.0/t;
    }
    System.out.println("Le résultat final est : ");
    System.out.println(s);
  }
}
```

2. `Programmation2.java` :

Programme2

```
import dauphine.util.*;
public class Programme2 {
  public static void main(String[] args) {
    Console.start();
    double s=1;
    int n;
    do {
      System.out.print("Valeur du paramètre n : ");
      n=Console.readInt();
    } while (n<=0);
```

```
11      for (int i=2;i<=n;i++) {
12        int t=i;
13        for (int j=2;j<=n;j++)
14          t=t*i;
15        s=s+1.0/t;
16      }
17      System.out.println("Le résultat final est : ");
18      System.out.println(s);
19    }
20  }
```

1.2 Calcul de suites

Exercice 1.9 :

Pour chacune des suites définies dans la suite de l'énoncé, écrire un programme qui en affiche les N premiers termes :

$$u_n = \frac{1}{n}\sum_{k=1}^{n}\cos^2(\frac{k\pi}{n}) \qquad v_n = \sum_{k=1}^{n} u_k \text{ où } u_k = \frac{1}{(k+1)!}\sum_{p=1}^{n} p!$$

$$v_n = \cos(n\pi\sqrt{2})$$

$$w_n = \frac{1}{(n+1)!}\sum_{k=1}^{n} k! \qquad \begin{cases} u_0 = 1 \\ u_{n+1} = 1 + \frac{x u_n}{n+1}, \quad x \in \mathbb{R} \end{cases}$$

Exercice 1.10 :

Écrire un programme qui calcule x^n où x est un réel et n un entier relatif.

Exercice 1.11 :

Proposer un algorithme général pour calculer les valeurs d'une suite définie par récurrence et de la forme $u_n = f(u_{n-1}, u_{n-2})$. Comment généraliser ce schéma au cas où $u_n = f(u_{n-1}, u_{n-2}, \ldots, u_{n-k})$, pour k un entier fixé ?

Exercice 1.12 :

On considère les suites (u_n) et (v_n) définies à partir de u_0 et v_0 tels que $0 < u_0 < v_0$ par :

$$u_{n+1} = \frac{u_n + v_n}{2} \quad \text{et} \quad v_{n+1} = \sqrt{u_n v_n}$$

Étudier expérimentalement la convergence des suites (u_n) et (v_n).

Exercice 1.13 :

On considère la méthode suivante :

```
1  public static double g(int n) {
2    double r=0;
3    double f=1;
4    for(int i=1;i<=n;i++) {
5      f*=i;
6      r+=1/f;
7    }
8    return r;
9  }
```

1. Quand on lui transmet un entier `n` strictement positif, la méthode renvoie :
   (a) $\sum_{k=1}^{n} \frac{1}{k^2}$
   (b) $\sum_{k=1}^{n} \frac{1}{k!}$
   (c) $\sum_{k=1}^{n} \frac{1}{k^n}$
2. Quand on lui transmet un entier `n` négatif ou nul, la méthode :
   (a) plante (provoque un arrêt du programme)
   (b) renvoie une valeur qui dépend de `n`
   (c) renvoie toujours 0
3. On remplace la ligne `3` par `int f=1 ;`.
   (a) la nouvelle méthode ne compile plus
   (b) la nouvelle méthode plante (provoque un arrêt du programme)
   (c) la nouvelle méthode se comporte exactement comme la version d'origine
   (d) la nouvelle méthode renvoie 0 quand `n` est négatif ou nul, et 1 dans les autres cas
   (e) la nouvelle méthode renvoie toujours 1

Exercice 1.14 :

On considère la méthode suivante :

```
1 public static int l(int n,int s) {
2   int u=s,v=s,w=s;
3   for(int i=2;i<=n;i++) {
4     w=2*u+v;
5     u=v;
6     v=w;
7   }
8   return w;
9 }
```

1. L'appel `l(n,s)`, avec `n` positif ou nul, renvoie le `n`-ième terme de la suite définie par :
   (a) $u_0 = \mathtt{s}$, $u_1 = \mathtt{s}$ et pour $k > 1$, $u_k = 2u_{k-1} + u_{k-2}$
   (b) $u_1 = \mathtt{s}$, $u_2 = \mathtt{s}$ et pour $k > 2$, $u_k = 2u_{k-1} + u_{k-2}$
   (c) $u_0 = \mathtt{s}$, $u_1 = \mathtt{s}$ et pour $k > 1$, $u_k = u_{k-1} + 2u_{k-2}$
   (d) $u_1 = \mathtt{s}$, $u_2 = \mathtt{s}$ et pour $k > 2$, $u_k = u_{k-1} + 2u_{k-2}$
2. Dans la ligne 2, on remplace `w=s` par `w=0`. La nouvelle méthode :
   (a) donne exactement les mêmes résultats que l'ancienne
   (b) calcule la même suite que l'ancienne méthode, mais peut donner des résultats faux quand `n` est égal à 1 ou à 2
   (c) calcule la même suite que l'ancienne méthode, mais peut donner des résultats faux quand `n` est égal à 0 ou à 1
   (d) calcule une suite différente de celle calculée par l'ancienne méthode (pour tout `n`)

1.3 Arithmétique

Exercice 1.15 :

Écrire un programme qui calcule la somme de tous les nombres entiers positifs pairs inférieurs à 20.

Exercice 1.16 :

Écrire un programme qui affiche tous les multiples d'un entier n inférieurs à un entier m.

Exercice 1.17 :

Écrire un programme qui affiche tous diviseurs d'un entier n.

Exercice 1.18 :

Écrire un programme qui permet de calculer tous les entiers k compris entre 1 et un entier N (dont la valeur sera saisie au clavier), tels que la somme des cubes des chiffres composant k est égale à k lui-même. Ex : $153 = 1^3 + 5^3 + 3^3$.

Exercice 1.19 :

Tout nombre réel x strictement positif peut s'écrire sous forme scientifique $x = m.10^{ex}$ où la mantisse m vérifie $1 \leq m < 10$ et où l'exposant ex est un entier.

Écrire un programme qui demande un réel à l'utilisateur et en détermine la mantisse m et l'exposant ex, sans utiliser la fonction log.

Exercice 1.20 :

Écrivez un programme qui calcule la valeur de $n!$. On fera déterminer par l'ordinateur un entier F_{max} tel que si $n > F_{max}$, la valeur de $n!$ est supérieure à `Integer.MAX_VALUE`.

Écrivez un programme qui calcule la valeur de C_n^p ou indique une erreur si $n < 0$, si $p < 0$ ou si $n > p$, ou encore si $C_n^p >$`Integer.MAX_VALUE`.

Essayez plusieurs stratégies de calcul de C_n^p. On rappelle les relations :

$$C_n^p = \frac{n!}{p!(n-p)!} = \frac{n(n-1)(n-2)\dots(n-p+1)}{p(p-1)(p-2)\dots 2 \times 1}$$

Exercice 1.21 :

Deux nombres entiers a et b sont amis si et seulement si la somme des diviseurs stricts de a vaut b et réciproquement si la somme des diviseurs stricts de b vaut a. Le nombre entier a est parfait s'il est ami avec lui-même (d est un diviseur strict de n s'il existe un entier $k > 1$ tel que $n = kd$).

Exemple : 220 et 284 sont des nombres amis en effet :
– les diviseurs stricts de 220 sont 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 et 110 dont la somme est 284 ;
– les diviseurs stricts de 284 sont 1, 2, 4, 71 et 142 dont la somme est 220.

Écrivez un programme qui affiche la liste de tous les nombres amis inférieurs à un entier N qui sera saisi au clavier. Écrivez un programme qui détermine tous les couples de nombres amis (p, q) tels que $p \leq q \leq N$.

Exercice 1.22 :

On rappelle que pour tout couple $(a, b) \in \mathbb{Z}^* \times \mathbb{N}^*$, il existe un couple unique (q, r) d'entiers relatifs tels que $a = bq + r$, $0 \leq r < b$ (q est le quotient, et r le reste, de la division euclidienne de a par b).

Écrivez un programme qui saisit deux entiers a et b, tels que $(a, b) \in \mathbb{Z}^* \times \mathbb{N}^*$ (prévoir un test sur les données) et qui calcule le quotient q et le reste r de la division de a par b, **sans utiliser les opérateurs / et %**.

Exercice 1.23 :

Pour multiplier entre eux deux entiers non négatifs x et y, on procède ainsi :
- à chaque étape, on divise x par 2 (division entière sans tenir compte du reste) et on multiplie y par 2, ceci jusqu'à ce que la valeur de x soit égale à 1 ;
- la valeur de xy est alors la somme des valeurs de y correspondant à des x impairs.

Exemple : $17 \times 13 = 221$

17	13	←
8	26	
4	52	
2	104	
1	208	← on a bien $221 = 13 + 208$

Justifiez la méthode proposée et écrivez un programme qui la met en oeuvre.

Exercice 1.24 :

Étant donnés deux nombres entiers naturels a et b, on appelle PGCD de a et b le plus grand commun diviseur de a et b. De même, le PPCM de a et b est le plus petit commun multiple de a et b. Pour calculer les PGCD et PPCM, on utilise les propriétés suivantes :
- le PGCD de a et b est aussi celui de b et a,
- si $a = bq + r$ avec $r < b$, le PGCD de a et b est aussi celui de b et r,
- le PPCM de a et b est le quotient du produit ab par le PGCD de a et b.

Vérifiez mathématiquement les propriétés ci-dessus. Écrivez un programme qui saisit les nombres a et b, et affiche leur PGCD et PPCM.

Exercice 1.25 :

Étant donné un nombre entier naturel a, on dit que a est premier s'il n'admet pas d'autres diviseurs que 1 et lui-même. Écrivez un programme qui saisit un nombre entier a, détermine si a est premier ou non, et affiche le résultat du test à l'écran.

Complétez le programme précédent en faisant afficher dans le cas où a n'est pas premier, la décomposition de a en facteurs premiers. Vous utiliserez à cet effet l'algorithme suivant :
- on cherche le plus petit diviseur premier de a soit d ;
- si $d < a$, on affiche d, on divise a par d et on recommence, sinon on affiche a.

Exercice 1.26 :

On montre qu'un entier naturel est divisible par 11 si la somme de ses chiffres de rang pair est égale à celle de ses chiffres de rang impair, modulo 11. Écrire un programme qui saisit un nombre entier naturel a et indique s'il est divisible par 11.

Exercice 1.27 :

On considère la méthode suivante :

```
1  public static int a(int n) {
2    int k=0;
3    while(n>0) {
4      k++;
5      n/=10;
6    }
7    return k;
8  }
```

1. Quand on appelle la méthode avec comme paramètre un entier positif n, elle renvoie :
   (a) la somme des chiffres de n
   (b) le nombre de chiffres de n
   (c) le nombre de chiffres non nuls de n
2. Quand on appelle la méthode avec comme paramètre un entier strictement négatif n, elle :
   (a) plante (provoque l'arrêt du programme)
   (b) ne s'arrête jamais
   (c) renvoie 0
3. On remplace la ligne 3 par `while(n>=0) {`. La nouvelle méthode :
   (a) fonctionne exactement comme avant
   (b) ne s'arrête jamais quand son paramètre est positif ou nul (et fonctionne comme avant dans les autres cas)
   (c) fonctionne exactement comme avant sauf quand son paramètre est nul (auquel cas la méthode ne s'arrête jamais)

Exercice 1.28 :

On considère la méthode suivante :

```
1  public static int b(int n,int pos) {
2    for(int i=1;i<pos;i++) {
3      n/=10;
4    }
5    if (n>0) {
6      return n%10;
7    } else {
8      return -1;
9    }
10 }
```

1. On effectue l'appel `b(n,p)` où `n` désigne un entier strictement positif. La méthode renvoie :
   (a) le `p`-ième chiffre de `n` (en comptant à partir de la **droite**, par exemple 2 est le premier chiffre de 512) et `-1` si un tel chiffre n'existe pas
   (b) le `p`-ième chiffre de `n` (en comptant à partir de la **gauche**, par exemple 5 est le premier chiffre de 512) et `-1` si un tel chiffre n'existe pas

2. On effectue l'appel `b(n,p)` où `n` désigne un entier strictement négatif.
   (a) la méthode plante (provoque l'arrêt du programme)
   (b) le programme ne compile pas
   (c) la méthode renvoie toujours `-1`
3. On effectue l'appel `b(n,-1)` où `n` désigne un entier strictement positif.
   (a) la méthode plante (provoque l'arrêt du programme)
   (b) la méthode renvoie toujours `-1` (pour toute valeur de `n`)
   (c) la méthode renvoie le chiffre le plus à **droite** de `n`
   (d) la méthode renvoie le chiffre le plus à **gauche** de `n`

Exercice 1.29 :

On considère la méthode suivante :

```
1 public static int c(int n) {
2   int a=0;
3   while(n>0) {
4     a+=n%10;
5     n/=10;
6   }
7   return a;
8 }
```

1. On effectue l'appel `c(n)` où `n` désigne un entier strictement positif. La méthode renvoie :
   (a) le nombre d'apparition du chiffre `0` dans `n`
   (b) la somme des chiffres de `n`
   (c) `0`
2. On échange les lignes 4 et 5, la méthode devenant :

```
1 public static int c(int n) {
2   int a=0;
3   while(n>0) {
4     n/=10;
5     a+=n%10;
6   }
7   return a;
8 }
```

   On effectue l'appel `c(n)` où `n` désigne un entier strictement positif. La nouvelle méthode renvoie :
   (a) le même résultat que la méthode d'origine
   (b) un résultat strictement plus petit que celui qu'aurait renvoyé la méthode d'origine
   (c) un résultat plus petit ou égal à celui qu'aurait renvoyé la méthode d'origine

3. On reprend la méthode d'origine et on la modifie en :

```
1  public static int c(int n) {
2    int a=0;
3    while(n>0) {
4      a+=n%10;
5      n/=10;
6      return a;
7    }
8    return 0;
9  }
```

On effectue l'appel `c(n)` où `n` désigne un entier strictement positif. La nouvelle méthode renvoie :

(a) `0`

(b) la même chose que la première version de `c`

(c) la liste des chiffres de `n`

(d) le chiffre le plus à **droite** de `n`

(e) le chiffre le plus à **gauche** de `n`

4. On conserve la version de la question précédente en supprimant la ligne `8`. La nouvelle méthode (utilisée avec un paramètre strictement positif) :

(a) fonctionne comme la version avec la ligne `8`

(b) plante (provoque l'arrêt du programme)

(c) ne compile plus

2 Les méthodes

2.1 Analyse

Exercice 2.1 :

Les méthodes qui suivent comportent chacune une erreur. Indiquer l'erreur et un moyen simple de la corriger.

missing11

```
1  public static first(double x) {
2    return x*2.5;
3  }
```

missing12

```
1  public static double first(x) {
2    return 1.5*x;
3  }
```

missing21

```
1  public static double second(double x)
2    return 1.5*x;
```

Exercice 2.2 :

Donner l'affichage produit par le programme suivant :

Modification

```
1  public class Modification {
2    public static void modif1(int x) {
3      x+=2;
4    }
5    public static void modif2(int x) {
6      x=5;
7    }
8    public static void main(String[] args) {
9      int x=24;
10     modif1(x);
11     System.out.println(x);
12     x=24;
13     modif2(x);
14     System.out.println(x);
15   }
16 }
```

Exercice 2.3 :

Donner l'affichage produit par le programme suivant :

Calcul

```
1  public class Calcul {
2    public static int calcul(int a,int b) {
3      return a-b;
4    }
5    public static int f(int c,int d) {
6      return calcul(d,d*c);
7    }
8    public static void main(String[] args) {
9      System.out.println(calcul(2,3));
10     System.out.println(f(2,3));
11     int a=3,b=5;
12     System.out.println(calcul(b,a));
13     System.out.println(f(b,a));
14     int c=2,d=4;
15     System.out.println(f(d,c));
16   }
17 }
```

Exercice 2.4 :

Donner l'affichage produit par le programme suivant :

Calcul2

```
1  public class Calcul2 {
2    public static double f(double x) {
3      return x*x+1;
4    }
5    public static double g(double x) {
6      if (x>0)
7        return -x*x;
8      else
9        return 4*x;
10   }
11   public static double h(double x) {
12     return 2*g(x*x);
13   }
14   public static void main(String args[]) {
15     System.out.println(f(f(3)));
16     System.out.println(g(g(2)));
17     System.out.println(g(h(2)));
18   }
19 }
```

Exercice 2.5 :

On considère les deux classes suivantes :

Truc

```
1  public class Truc {
2    public static int f(int a) {
3      return 2*a;
4    }
5    public static int g(int a) {
6      return a+f(a/2);
7    }
8  }
```

Bidule

```
1  public class Bidule {
2    public static int f(int a) {
3      return 3*a;
4    }
5    public static int g(int a) {
6      return Truc.f(2*a);
7    }
8    public static void main(String[] args) {
9      System.out.println(f(2));
10     System.out.println(g(3));
11     System.out.println(Truc.f(2));
12     System.out.println(Truc.g(3));
```

```
13    }
14  }
```

Quel est l'affichage produit par la classe `Bidule`.

2.2 Fonctions mathématiques

Exercice 2.6 :

1. Écrire une méthode qui calcule la fonction f de $\mathbb{R}$ dans $\mathbb{R}$ définie par $f(x) = \sqrt{\sin^2(x) + \pi}$.
2. Écrire une méthode qui calcule la fonction g de $\mathbb{R} \times \mathbb{N}$ dans $\mathbb{R}$ définie par $g(x) = \cos^2\left(\frac{2\pi x}{n}\right)$.

Exercice 2.7 :

Écrire une méthode `suite` qui à un réel `x` et un entier positif `n` associe le terme $u_{\texttt{n}}$ de la suite $(u_k)_{k\in\mathbb{N}}$ définie par :

$$\begin{aligned} u_0 &= \texttt{x} \\ u_k &= 3u_{k-1}(u_{k-1} - 1) \text{ pour } k > 0 \end{aligned}$$

2.3 Arithmétique

Exercice 2.8 :

Écrire une méthode qui à deux entiers `n` et `i` associe le `i`-ème chiffre de `n`, en numérotant les chiffres à partir de la gauche. Par exemple, le 2ème chiffre de `3456` est `4`.

Exercice 2.9 :

Écrire une méthode qui à deux entiers `n` et `i` associe le nombre de fois où le chiffre `i` apparaît dans le nombre `n`. Par exemple, si `n` vaut 12332 et si `i` vaut `3`, on obtient comme résultat 2.

Exercice 2.10 :

Écrire une méthode qui à un entier `n` associe le nombre de chiffres pairs apparaissant dans le nombre `n`. Par exemple, si `n` vaut 12332, on obtient comme résultat 3.

2.4 Méthodes utilitaires

Exercice 2.11 :

Écrire une méthode de saisie contrôlée prenant comme paramètre deux réels a et b. Cette méthode doit renvoyer un réel saisi par l'utilisateur et appartenant à l'intervalle $[a, b]$. Si l'utilisateur ne fournit pas un réel valide, la méthode doit recommencer la saisie.

Exercice 2.12 :

Proposez une méthode qui permet la saisie d'une valeur réelle comprise entre deux bornes et représentant une note définie au quart de point près. On ne pourra donc pas saisir n'importe quel réel, mais seulement les réels qui multipliés par quatre donnent un entier. Généraliser la méthode en ajoutant un paramètre qui règle la précision de la note (au n-ième de point).

2.5 Algorithmes numériques

Exercice 2.13 :

On suppose données deux méthodes `f` et `df` qui représentent respectivement une fonction f de $\mathbb{R}$ dans $\mathbb{R}$, et sa dérivée f'. On souhaite trouver $x \in \mathbb{R}$ tel que $f(x) = 0$, par la méthode de Newton.

En partant d'un point x_0, calculer $x = x_0 - \frac{f(x_0)}{f'(x_0)}$. Si la valeur absolue de $f(x)$ est inférieure à un certain seuil, x est la valeur cherchée. Sinon recommencer avec x comme point de départ. Si aucune valeur n'est trouvée après un nombre maximum d'itérations (à définir) la méthode s'arrête.

Écrire une méthode `newton` pour la fonction f, permettant de spécifier sous forme de paramètre le point de départ, le seuil désiré et le nombre maximal d'itérations. La méthode devra renvoyer la valeur de x découverte.

Exercice 2.14 :

On suppose donnée une méthode `f` qui représente une fonction f de $\mathbb{R}$ dans $\mathbb{R}$. Écrire une méthode `intégrale` qui calcule l'intégrale de f sur l'intervalle $[a, b]$ par la méthode des trapèzes. La méthode prendra comme paramètres les réels a et b, ainsi qu'un entier n indiquant le nombre de trapèzes à utiliser dans le calcul. Elle devra renvoyer l'intégrale obtenue.

Exercice 2.15 :

L'algorithme de la section dorée permet de trouver un minimum local d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$. En voici une description :

Données :
- un intervalle $[a, b]$ de $\mathbb{R}$;
- un point $x \in]a, b[$;
- une fonction f continue de $[a, b]$ dans $\mathbb{R}$ et telle que $f(a) > f(x)$ et $f(b) > f(x)$;
- un réel $\epsilon > 0$ représentant la précision de résolution.

Résultats : un intervalle $[u, v]$ de $\mathbb{R}$ et un point $y \in]u, v[$ tels que :
- $(v - u) < \epsilon \frac{v+u}{2}$;
- $f(u) > f(y)$ et $f(v) > f(y)$

1. on pose `u=`a, `v=`b et `y=`x
2. on pose `g=`$\frac{\sqrt{5}-1}{2}$
3. tant que `v-u`$\geq \epsilon$`(v+u)/2` :
   (a) si l'intervalle `]u,y[` est plus grand que l'intervalle `]y,v[` :
      i. placer dans `w` le résultat de `u+g*(y-u)`
   (b) sinon
      i. placer dans `w` le résultat de `v-g*(v-y)`
   (c) si f(`w`) $> f$(`y`)
      i. si `w<y`, placer `w` dans `u`
      ii. sinon, placer `w` dans `v`
   (d) sinon (on fait l'hypothèse simplificatrice que f(`w`) $\neq f$(`y`)) :
      i. si `w<y`, placer `y` dans `v` et `w` dans `y`

ii. sinon, placer `y` dans `u` et `w` dans `y`

4. Résultat : les contenus des variables `u`, `v` et `y`.

Questions :

1. représenter graphiquement les quatre cas possibles dans l'étude de $f(\mathtt{w})$;
2. montrer que cet algorithme s'arrête toujours après un nombre fini d'itérations qu'on peut calculer à l'avance, en fonction de a, b et ϵ, si on suppose que x vaut $a + (b - a)\frac{\sqrt{5}-1}{2}$;
3. montrer que les conditions énoncées sont bien vérifiées par le résultat produit par l'algorithme ;
4. programmer une méthode de minimisation par section dorée. Cette méthode supposera donnée une méthode `f` qui représente la fonction f à minimiser. Elle prendra comme paramètre les bornes a et b de l'intervalle, ainsi que le réel ϵ. Elle devra renvoyer l'estimation de la position du minimum.