1. Soit d un réel fixé. On considère un système d'inéquations aux inconnues réelles x et y.
$$\begin{cases} 0 \leq x \leq |d| \\ 0 \leq y \leq |d| \\ x - y = d \end{cases}$$
En considérant deux cas, résoudre ce système d'inéquations.
2. Soit $n > 1$ entier et $(a_1, a_2, \cdots, a_n)$ un n-uplet quelconque de nombres réels. Montrer qu'il existe un unique couple $((b_1, b_2, \cdots, b_n), (c_1, c_2, \cdots, c_n))$ de n-uplets vérifiant
$$\begin{aligned} &b_1 = a_1, \quad c_1 = 0 \\ &\forall k \in \{2, \cdots n\},\ 0 \leq b_k - b_{k-1} \leq |a_k - a_{k-1}| \\ &\forall k \in \{2, \cdots n\},\ 0 \leq c_k - c_{k-1} \leq |a_k - a_{k-1}| \\ &\forall k \in \{1, \cdots n\},\ a_k = b_k - c_k \end{aligned}$$
La démonstration devra utiliser la question 1 présenter en langage usuel l'algorithme définissant le seul couple de n-uplets possible.
3. Pour n entier fixé représenté en Maple par la variable globale `n`, on identifie un n-uplet à un tableau indexé de 1 à n.
Écrire en langage Maple une procédure nommée `decdif` telle que l'appel `decdif(A)` renvoie une liste `[B,C]` de tableaux associés aux n-uplets comme dans la question 2.

Corrigé

1. Supposons que x et y vérifient les relations.
Si $d \geq 0$ alors $x = d + y \geq d = |d|$ donc $y = 0$ et $x = d = |d|$.
Si $d < 0$ alors $y = -d + x \geq -d = |d|$ donc $x = 0$ et $y = -d = |d|$.
Réciproquement, on vérifie facilement que $(|d|, 0)$ est solution quand $d \geq 0$ et que $(0, |d|)$ est solution lorsque $d < 0$. Cette question montre donc que le système admet un unique couple de solutions.
2. Supposons que $b_1, \cdots, b_n, c_1, \cdots, c_n$ vérifient les relations.
Les dernières relations signifient exactement que $(b_k - b_{k-1}, a_k - a_{k-1})$ est un couple de solutions d'un système analogue à celui de la question 1. avec $d = a_k - a_{k-1}$. Ce système admet un unique couple solution. À partir des conditions initiales a_1 et 0, on construit ainsi de proche en proche l'unique solution possible selon l'algorithme suivant
   - `B[1] <-- A[1]`
   - `C[1] <-- 0`
   - Pour `k` de 2 à `n`
     - Si `A[k] >= A[k-1]`
       - `B[k] = B[k-1] + A[k] - A[k-1]`
       - `C[k] = C[k-1]`
     - Sinon
       - `B[k] = B[k-1]`
       - `C[k] = C[k-1] - A[k] + A[k-1]`
3. On peut implémenter la procédure demandée par le code suivant

```
decdiff := proc(A)
 global n;
 local B,C,i;
 B := array(1..n);
 C := array(1..n);
 B[1] := A[1];
 C[1] := 0;
 for i from 2 to n do
   if A[i] >= A[i-1] then
     B[i] := B[i-1] + A[i] - A[i-1];
     C[i] := C[i-1];
   else
     B[i] := B[i-1];
     C[i] := C[i-1] - A[i] + A[i-1];
   fi;
```


Rémy Nicolai Avarbor

```
  od;
  return [eval(B),eval(C)];
end proc:
```