Informatique en Degead 1

—

Algorithmes en Maple (2/3)

Responsable : Denis Cornaz

denis.cornaz@dauphine.fr
01 44 05 41 83
P 409 bis

http://www.lamsade.dauphine.fr/∼cornaz/Enseignement/UV21-Degead1/

Université Paris-Dauphine

Planning (rappel)

- Du 24 Septembre au 28 Octobre puis du 5 au 11 Novembre :
  Semaines 1-6 (S1A)
  1h30 d'informatique par semaine en salle info :
  - Semaines 1-2 = prise en main Excel
  - Semaines 3-6 = Algorithmique en Maple
  - 3 amphis avec moi = 3, 10 et 24 Octobre (slides sur mon site)
- Du 29 Octobre au 4 Novembre : Semaine de consolidation
- Du 12 Novembre au 23 Décembre : Semaines 7-12 (S1B)
  3h d'informatique par semaine =
  1h30 en salle info + 1h30 en salle de cours
  - Algorithmique en Maple
- 4 dernières semaines du semestre =
  2 de vacances + 1 de révision + 1 d'examen.

Évaluation (rappel)

- Note de Contrôle continu sur 20 : 3 interrogations en TD = 1 sur 8 (45 min en semaine 4) + 1 sur 12 (1h30 en semaine 8)
- Note d'examen sur 20 en Amphi (1h30 sans aucun document) = 5 points sur Excel + 15 points sur Maple
- Note finale = 0.5CC + 0.5E
  (ou zéro en cas de 6 absences justifiées ou non)

Mots clés en Maple

Seuls sont autorisés :

- les séparateurs
  `, [ ] ( ) : ;`
- l'affectation
  `:=`
- les opérateurs arithmétiques
  `+ - * / < > <= >= = <>`
- les mots clés suivants
  `proc local RETURN end if then fi true false and or not while do od for from to op nops NULL`

Coder des algorithmes en Maple : multiplication "bizarre" (rappel)

Un algorithme manipulant deux entiers x et y :

Diviser x par 2 en arrondissant à l'inférieur et multiplier y par 2 jusqu'à ce que x vaille 1, puis additionner les valeurs prises par y lorsque x avait une valeur impaire.

On obtient, avec $x = 11$ et $y = 13$:

x	11	5	2	1
y	13	26	52	104

$13 + 26 + 104 = 143 = x \times y$.

Boucle `while`

Syntaxe

`while` *condition* `do`
    instructions
`od:`

Ex :

```
> x:=3:
> while x <= 10 do x:=x+2; od:
> x;
                          11
```

Procédure

Syntaxe

nom de la procédure :=proc(*paramètre d'entrée*)
instructions
RETURN(*paramètre de sortie*)
end:

Ex :

```
> Plus4:=proc(x)
      RETURN(x+4):
    end:
> Plus4(6);
                                  10
> x:=3: x:=Plus4(x):
> x;
                                   7
```

Procédure appelant d'autres procédures

Ex : Procédure `Plus6()` appelant la procédure `Plus4()`

```
> Plus6:=proc(x)
      RETURN(Plus4(x)+2):
    end:
> Plus6(6);
                                    12
```

Variables locales

Syntaxe

nom de la procédure :=proc(*paramètre d'entrée*)
    déclaration des variables locales
    instructions
RETURN(*paramètre de sortie*)
end:

Ex : Redéfinir `Plus4()`

```
> Plus4:=proc(x)
      local y:                          #variable locale
      y:=x+4:
      RETURN(y):
   end:
```

Quotient entier

Soit a, b deux entiers, il existe un unique entier q tel que

$$a = qb + r, \qquad \text{avec } 0 \leq r < b$$

Entrée : deux entiers a et b **Sortie : le quotient entier q**

```
> MyIquo:=proc(a,b)
      local q:
      q:=0:
      while q*b <= a do
         q:=q+1:
      od:
      RETURN(q-1):
   end:
```

Ex. $a = 17$, $b = 5$

q	0	1	2	3	4

Reste entier

Soit a, b deux entiers, il existe un unique entier r tel que

$$a = qb + r, \qquad \text{avec } 0 \leq r < b$$

Entrée : deux entiers a et b **Sortie : le reste entier r**

```
> MyIrem:=proc(a,b)
      RETURN(a-MyIquo(a,b)*b):
  end:
```

Test if

Syntaxe

if *expression booléenne* then
    instructions
fi:

Ex :

```
> x:=3:
> if x <= 10 then x:=x+2: fi:
> x;
                          5
> if x > 10 then x:=0: fi:
> x;
                          5
```

Test de parité

Entrée : un entier a — Sortie : booléen a impair ?

```
> IsOdd:=proc(a)
      if MyIrem(a,2)=0 then RETURN(false):fi:
      RETURN(true):
    end:
> IsOdd(16);

                                  false
```

Multiplication bizarre

Entrée : deux entiers a et b — Sortie : le produit a * b

```
> MyMult:=proc(a,b)
    local x,y,z:
    x:=a: y:=b:
    z:=0:
    while x >= 1 do
       if IsOdd(x) then z:=z+y: x:=x-1: fi:
       x:=x/2:
       y:=y*2:
    od:
    RETURN(z):
  end:
```

```
> MyMult(11,13);
```

143

Boucle `for`

Syntaxe

`for` *variable* `from` *valeur initiale* `to` *valeur finale* `do:`
    instructions
`od:`

Ex :

```
> x:=0:
> for i from 1 to 100 do: x:= x+i: od:
> x;
                              5050
```

Racine carrée (rappel)

Un algorithme manipulant un entier y :

Soit $x = 10$. *Modifier 5 fois la valeur de* x *en lui attribuant à chaque fois la nouvelle valeur de*

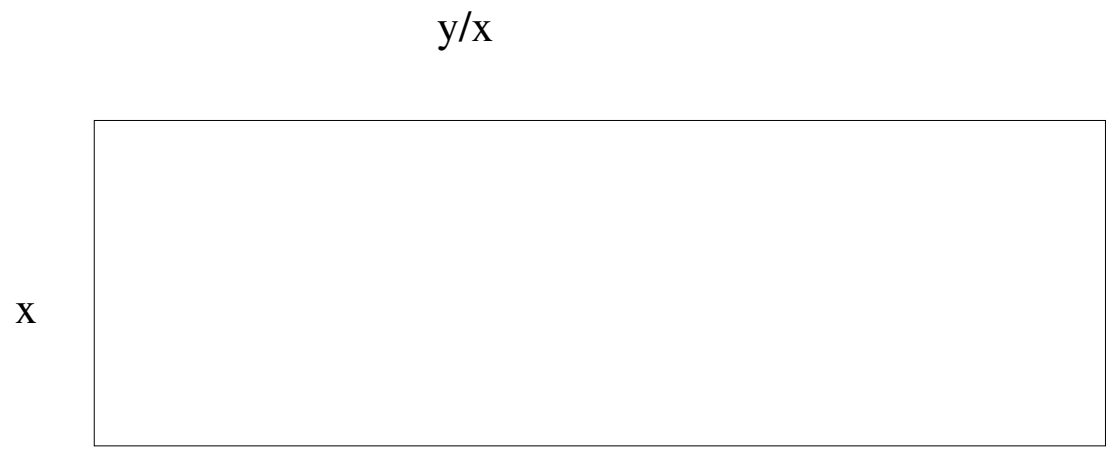
$$\frac{x + \frac{y}{x}}{2}$$

Pour $y = 4$, on obtient :

$$x = \quad 10 \quad 5.2 \quad 2.98 \quad 2.16 \quad 2.00$$

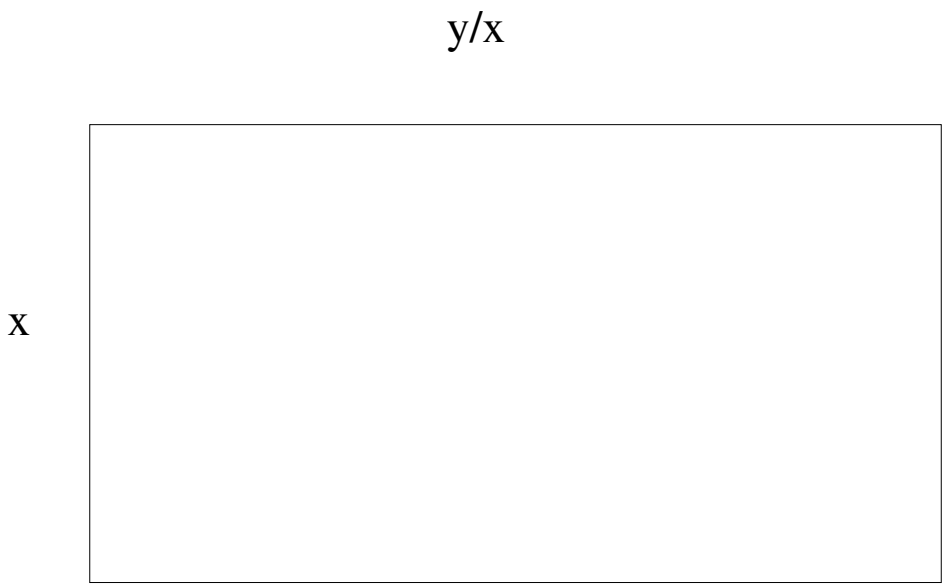
Racine carrée (rappel)

Un rectangle dont l'aire vaut y quelque soit x, Faire $x \leftarrow \frac{x+\frac{y}{x}}{2}$



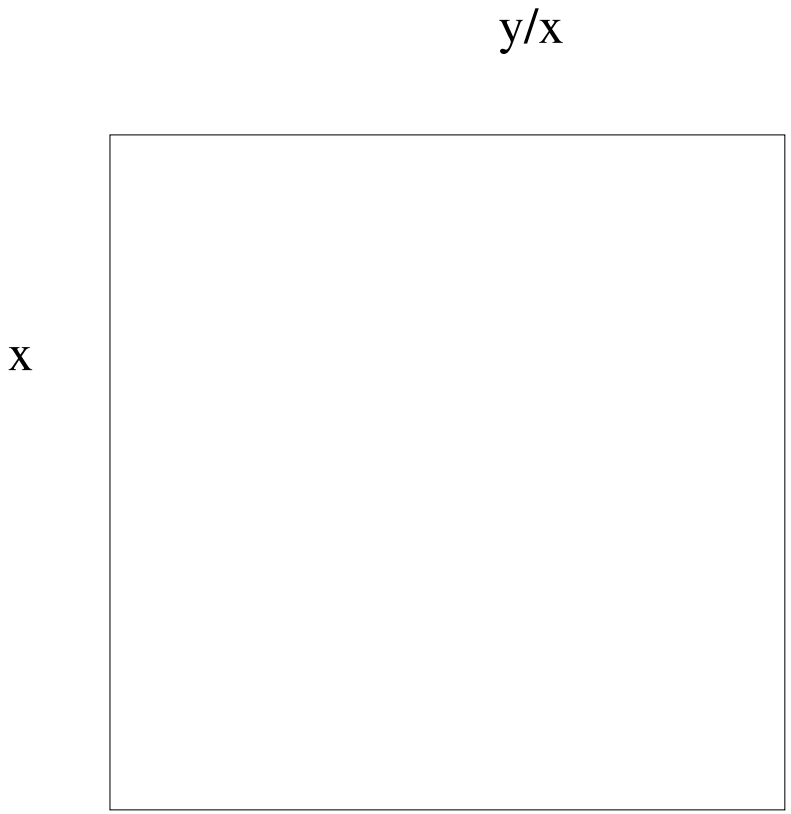
Racine carrée (rappel)

Un rectangle dont l'aire vaut y quelque soit x, Faire $x \leftarrow \frac{x+\frac{y}{x}}{2}$



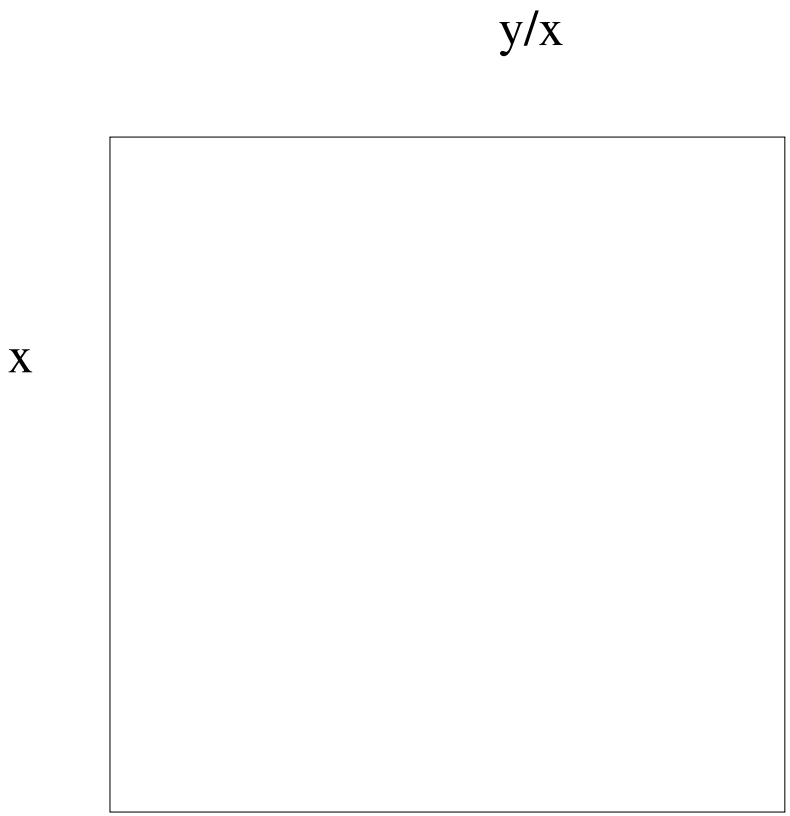
Racine carrée (rappel)

Un rectangle dont l'aire vaut y quelque soit x, Faire $x \leftarrow \frac{x+\frac{y}{x}}{2}$



Racine carrée (rappel)

Un rectangle dont l'aire vaut y quelque soit x, Faire $x \leftarrow \frac{x+\frac{y}{x}}{2}$



Si le rectangle est un carré : $x = \sqrt{y}$

Racine carrée

Entrée : un réel y Sortie : une approximation de sa racine carrée

```
> MySqrt:=proc(y)
      local x,i:
      x:= 10.0:
      for i from 1 to 5 do
                x:=(x+(y/x))/2:
     od:
     RETURN(x);
   end:
```

```
> x:= 16:
> MySqrt(x);
                    4.000010363
```

Que fait l'algorithme suivant ?

Entrée : deux entiers x et p — Sortie : ? ?

```
> Mystery:=proc(x,p)
      local i,r:
      r:=1:
      for i from 1 to p do:
                r:= r * x:
     od:
     RETURN(r);
   end:
```

Séquences

Séquences : concaténation opérateur " ,"

```
> S1:=NULL:
> S2:=3,4,5,1,2:
> S1:=S1,6:
> S1;
                              6
> S:=S2,S1;
                    S:=3,4,5,1,2,6
```

`NULL` : séquence vide

`S1,S2` : séquence constituée des éléments de `S1` puis de `S2`

Listes

Listes : accès direct `[]`, longueur `nops`

```
> L1:=[]:
> L2:=[S]:
> L2[3];
                              5
> L1:=[op(L2),4,5];
          L1:=[3,4,5,1,2,6,4,5]
> nops(L1);
                             8
```

`[]` : liste vide

`L[i]` : ième élément de la liste `L`

`nops(L)` : nombre d'éléments de la liste `L`

`op(L)` : séquence constituée des éléments de la liste `L`

`[S]` : liste constituée des éléments de la séquence `S`

Séquence des listes de valeurs dans MyMult

```
> MyMult:=proc(a,b)
        local x,y,z,S:
        x:=a: y:=b:
        z:=0:
        S:=NULL:
        while x>= 1 do
                S:=S,[x,y]:
                if IsOdd(x) then z:=z+y: x:=x-1: fi:
                x:=x/2:
                y:=y*2:
        od:
        RETURN(S,z):
    end:
> MyMult(11,13);
              [11, 13], [5, 26], [2, 52], [1, 104], 143
```

Séquence des listes de valeurs dans MySqrt

```
> MySqrt:=proc(y)
            local x,i,S:
            x:= 10.0:
            S:=NULL:
            for i from 1 to 5 do
                      S:=S,[x,y/x]:
                      x:=(x+(y/x))/2:
            od:
            RETURN(S);
      end:
 > MySqrt(16);
                  [10.0, 1.600000000],
                  [5.800000000, 2.758620690],
                  [4.279310345, 3.738920225],
                  [4.009115286, 3.990905439],
                  [4.000010363, 3.999989637]
```

Supprimer le *n*ème élément d'une liste

```
> MyRemove:= proc(L,n)
          local i,S:
          S:=NULL:
          if nops(L) < n then RETURN(L): fi:
          for i from 1 to nops(L) do:
                    if i <> n then S:=S,L[i]: fi:
          od:
          RETURN([S]):
     end:
> L:=[a,b,c,d,e]:
> MyRemove(L,3);
                              [a, b, d, e]
```

Différence symétrique

Entréee : deux booléens a et b **Sortie : leur différence symétrique**

```
> MyDifSym:= proc(a,b)
      RETURN((a and not b) or (b and not a)):
    end:
```

```
>  a:=true: b:=true:
> MyDifSym(a,b);
                                  false
> b:= false: MyDifSym(a,b);
                                  true
```