1 Un exemple

1.1 Une feuille de calculs

Regardons d'abord la séquence Maple :

```
> y:=sqrt(1-x**2);
```

$$y := \sqrt{1-x^2}$$

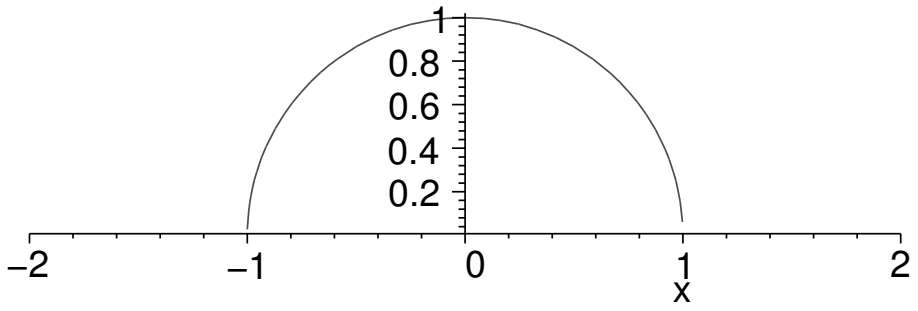
```
> subs(x=0,y);
```

$$1$$

```
> evalf(subs(x=2,y));
```

$$1.732050808\, I$$

```
> plot(y,x=-2..2,scaling=constrained);
```



1.2 Analyse

Regardons Maple tracer le graphe. Il ne trace évidemment que les points qui correspondent à des y réels ! On peut donc penser que les programmeurs de Maple disposaient d'une instruction du type :

si *telle condition* **alors** *telle action* **sinon** *tele autre action*

C'est ce qu'on appelle une structure **alternative**.

Par ailleurs, Maple ne trace pas une infinité de points – ce qui prendrait un temps infini – mais un nombre assez grand de petits segments de droites pour qu'on ait l'impression qu'il trace la courbe.

Si, par exemple ici, Maple trace 200 segments de droites, il relie :

le point $(-1;0)$ au point $(-0.99;\sqrt{1-0.99^2})$, puis,
le point $(-0.99;\sqrt{1-0.99^2})$ au point $(-0.98;\sqrt{1-0.98^2})$, puis,
le point $(-0.98;\sqrt{1-0.98^2})$ au point $(-0.97;\sqrt{1-0.97^2})$, puis,
le point $(-0.97;\sqrt{1-0.97^2})$ au point $(-0.96;\sqrt{1-0.96^2})$, puis,
le point $(-0.96;\sqrt{1-0.96^2})$ au point $(-0.95;\sqrt{1-0.95^2})$, puis ...
$\vdots$
le point $(0.98;\sqrt{1-0.98^2})$ au point $(0.99;\sqrt{1-0.99^2})$, et enfin,
le point $(0.99;\sqrt{1-0.99^2})$ au point $(1;0)$.

On peut penser donc que cette fois ci, les programmeurs de Maple utilisent une structure qui permet de répéter un certain nombre de fois une certaine action, du type :

pour *les valeurs de la variable* i **depuis** -1 **jusqu'à** 0.99 **par pas de** 0.01
faire *le tracé du segment qui relie* $(i;\sqrt{1-i^2})$ à $(i+0.01;\sqrt{1-(i+0.01)^2})$

C'est ce qu'on appelle une structure **itérative**.

2 Structures de controle

2.1 Notion de procédure

C'est la fonction `proc` qui permet de créer une fonction, encore appelée procédure, en complément du procédé simplifié déjà.
Il s'agit de donner un nom à une séquence de calcul en lui indiquant les paramètres utilisés, c'est la première phase d'écriture de la procédure.
Ensuite, en utilisant le nom donné auparavent, avec des valeurs des paramètres, on fera éxécuter la séquence de calcul sans avoir besoin de la réécrire, c'est la phase d'utilisation de la procédure.
Dans l'utilisation d'une procédure, il n'y a pas de différence visible entre une fonction Maple d'origine et une procédure (ou fonction) que vous avez créée... On dit qu'on procède par extention du langage.

La syntaxe générale est :

```
> nom:=proc(paramètre(s))
> local variable(s) locale(s);
> corps de la procédure
> variable(s) « résultat »
> end;
```

Les différentes variables sont éventuellement séparées par des virgules.

2.2 Structure alternative

La syntaxe générale de la structure alternative est :

```
> if condition then instructions cas vrai else instructions cas faux fi
```

Les instructions soulignées, ici et dans la suite de ce document, sont optionelles et peuvent être omises.
Maple teste la condition et éxécute l'un ou l'autre des paquets d'instruction selon les cas.

On va maintenant créer une procédure qui calcule, pour une expression usuelle du second degré

$$ax^2 + bx + c$$

à coefficients réels, le nombre de ses racines réelles.

```
> nbrac:=proc(a,b,c) local Delta,n; Delta:=b*b-4*a*c; if Delta > 0
then n:=2 else if Delta = 0 then n:=1 else n:=0 fi fi; n end;
```

$nbrac := \mathbf{proc}(a, b, c)$
local Δ, n;
$\Delta := b^2 - 4 * a * c$;
if $0 < \Delta$ **then** $n := 2$ **else if** $\Delta = 0$ **then** $n := 1$ **else** $n := 0$ **end if end if**;
n
end proc

Essayons notre procédure :

```
> nbrac(1,1,1);
```

$$0$$

```
> nbrac(1,2,1);
```

$$1$$

```
> nbrac(1,-3,2);
```

$$2$$

2.3 Structure itérative

La syntaxe générale de la structure itérative, ou répétitive, est :

```
> for variable from valeur début by valeur du pas to valeur fin while condition
> do corps de la boucle od
```

Pour toutes les valeurs de la variable,
depuis la valeur de début (1 si absente),
jusque la valeur de fin, en augmentant du pas donné (1 si absent),
tant que la condition est vérifiée,
les instructions du corps de la boucle sont exécutées.

On veillera à ce que la boucle ait un sens et s'arrête !...

On va d'abord créer une procédure qui calcule, pour un terme donné, le terme suivant de la suite

$$u_{n+1} = \frac{u_n + \dfrac{a}{u_n}}{2}$$

On a donc deux paramètres : le terme donné de la suite et a.
C'est la programmation d'un simple calcul direct.

```
> suivant:=proc(u,a) local v; v:=u; v:=(v+a/v)/2; v end;
```

$$suivant := \mathbf{proc}(u, a)\ \mathbf{local}\ v;\ v := u\ ;\ v := 1/2 * v + 1/2 * a\ /v\ ;\ v\ \mathbf{end\ proc}$$

Essayons cette procédure :

```
> suivant(2,3);
```

$$\frac{7}{4}$$

a) Strucure itérative à nombre de pas connu

On va maintenant créer une procédure qui, pour la même suite, calcule u_n pour u_0 et n donnés.
On a donc trois paramètres : le premier terme de la suite, a et le rang du terme cherché n.
A chaque passage dans la boucle, on calcule le terme suivant de `v`, que l'on réaffecte ensuite à `v`. La syntaxe Maple veut que cette réaffectation s'écrive **avant** le calcul du terme suivant...

```
> suite:=proc(u,a,n) local i,v; v:=u; for i from 1 to n do
v:=suivant(v,a) od; v end;
```

$$suite := \mathbf{proc}(u, a, n)\ \mathbf{local}\ i, v;\ v := u\ ;\ \mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ \ v := \mathrm{suivant}(v, a)\ \mathbf{end\ do}\ ;\ v\ \mathbf{end\ proc}$$

Essayons cette procédure :

```
> suite(2,3,5);
```

$$\frac{1002978273411373057}{579069776145402304}$$

Le résultat n'est pas très exploitable... Maple fait à priori du calcul exact, et ce n'est pas ce que l'on veut ici... Réécrivons une procédure qui fait la même chose en calcul approché « flottant ».

```
> suitef:=proc(u,a,n) local i,v; v:=evalf(u); for i from 1 to n do
v:=suivant(v,a) od; v end;
```

$$
\begin{aligned}
&\mathit{suitef} := \mathbf{proc}(u, a, n)\\
&\mathbf{local}\, i, v;\\
&\quad v := \mathrm{evalf}(u)\,;\ \mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, v := \mathrm{suivant}(v, a)\ \mathbf{end\ do}\,;\ v\\
&\mathbf{end\ proc}
\end{aligned}
$$

Essayons la :

```
> suitef(2,3,5);
```

$$1.732050808$$

C'est mieux !... En fait, ici, la suite converge vers $\sqrt{3}$. (ici $a = 3$)

```
> evalf(sqrt(3));
```

$$1.732050808$$

b) Strucure itérative à nombre de pas inconnu

On reprendra ce problème lors de la prochaine séance... Soyez patients !

3 A retenir

1) La **notion de procédure**, c'est à dire : la fonction `proc...end`, la notion de paramètre et de variable locale.

2) La **structure alternative** : `if...then...else...fi`.

3) La double **boucle** pour le calcul **répétitif** :

a) à nombre d'itérations **connu** :
`for...from...by...to...do...od`

b) à nombre d'itérations **inconnu** :
`while...do...od`

ainsi que la possibilité de combiner les deux.

4) L'intérêt de passer *le plus tôt possible* en calcul flottant quand on cherche un résultat en calcul approché.