Initiation à Mathematica: Compléments

Le but de ce chapitre est d'introduire quelques fonctions supplémentaires permettant de réaliser de la programmation légèrement plus avancée et de présenter certaines applications de Mathematica aux chapitres du cours de mathématiques de base.

§ 1 Quelques fonctions supplémentaires

§ 1-1 Module

Lorsqu'on désire que plusieurs instructions soient effectuées successivement par Mathematica sans que les variables de ces instructions n'entrent en conflit avec les variables du reste du document, on peut regrouper ces commandes dans une structure appelée **`Module`**.
La structure de **`Module`** est la suivante:

```
Module[
    {var1,var2,...},
    expr1;
    expr2;
    ...
]
```

Les variables **`var1,var2,...`** sont les *variables locales* du module, c'est-à-dire qu'il s'agit de variables utilisées uniquement à l'intérieur du module. Si une de ces variables a déjà été défini à l'extérieur du module, Mathematica en fait abstraction à l'intérieur du module et inversement, Mathematica ne tiendra pas compte des définitions de ces variables faites à l'intérieur du module lorsqu'on se trouve à l'extérieur du module.
Les expressions **`expr1;expr2;...`** sont les instructions à effectuer par Mathematica; Mathematica n'affichera que le résultat de l'évaluation de la dernière expression. Attention: il est important de terminer chaque ligne sauf la dernière par le symbole ;.
Si on souhaite créer un output contenant plusieurs éléments, on peut aisément le faire en les rassemblant dans une liste.

Lorsque l'on a défini un module, il est indispensable de le tester dans des situations connues afin de s'assurer qu'il fonctionne bien comme prévu.

Dans le cas de modules plus complexes, il est utile de ne pas définir directement le module voulu, mais de commencer par un module plus simple et d'ajouter des complications à fur et à mesure en le testant à chaque étape.

Exemple 1

1. Connaissant le nombre de côtés *n* d'un polygone régulier ainsi que le rayon *r* de son cercle circonscrit, on souhaite calculer le périmètre de ce polygone. Définissez un module **perimetrePolygone** qui calcule ce périmètre en fonction de *n* et *r*.

```
In[1]:= perimetrePolygon[r_, n_] := Module[
                                    module
          {a, c},
          a = 2 Pi/n;  (* angle au centre *)
          c = 2 r Sin[a / 2];  (* côté *)
                  sinus
          n c  (* Output: le perimètre *)
        ]

In[2]:= perimetrePolygon[x, 6]
        (* test: perimètre d'un hexagone inscrit dans un cercle de rayon x *)

Out[2]= 6 x
```

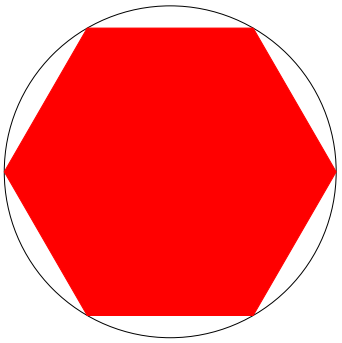
Nous pouvons constater qu'aucune valeur n'a été affecté aux variables **a** et **c** en dehors du module:

```
In[3]:= a
        c

Out[3]= a

Out[4]= c
```

2. Modifiez le module de telle sorte qu'on obtienne le périmètre ainsi qu'une représentation graphique du polygone inscrit dans le cercle comme output.

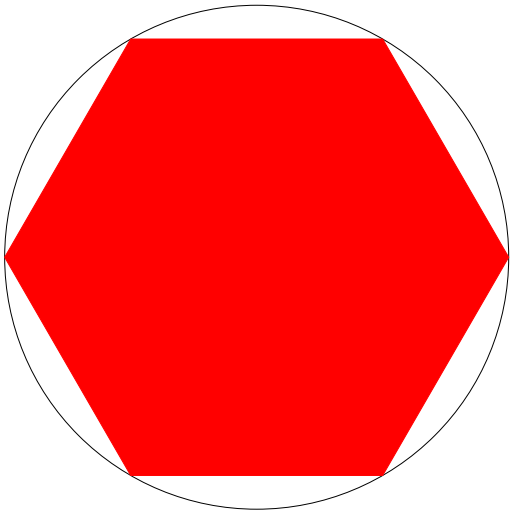
```
In[5]:= perimetreImagePolygon[r_, n_] := Module[
                                         module
          {a, c, image},
          a = 2 Pi/n;  (* angle au centre *)
          c = 2 r Sin[a / 2];  (* côté *)
                  sinus
          image = Graphics[{Circle[{0, 0}, r], Red, Polygon[CirclePoints[r, n]]}];
                  graphique  cercle              ro···  polygone  points de cercle
          {n c, image}
          (* output: liste contenant le périmètre et la représentation graphique *)
        ]
```

In[6]:=
```
perimetreImagePolygon[2, 6]
(* Test avec l'hexagone inscrit dans un cercle de rayon 2 *)
```

Out[6]= {12,  }

Et on a bien sûr la possibilité de n'afficher qu'un seul élément de l'output:

In[7]:=
```
perimetreImagePolygon[2, 6][[2]]  (* on extrait l'image de l'output *)
```

Out[7]= 

In[8]:=
```
Clear[perimetrePolygon, perimetreImagePolygon]
```
efface

§ I-2 Print

Lorsque l'on crée des modules plus sophistiqués, il peut être utile d'afficher les valeurs de certaines variables locales à un moment précis de l'exécution des instructions afin de chercher une éventuelle erreur de programmation. La commande **Print** permet de le faire:

Exemple

In[9]:=
```
perimetrePolygon[r_, n_] := Module[
                            module
  {a, c},
  a = 2 Pi/n;  (* angle au centre *)
  Print[a];
  imprime
  c = 2 r Sin[a/2];  (* côté *)
          sinus
  n c  (* perimètre *)
 ]
```

In[10]:=
```
perimetrePolygon[x, 6]
```

$\frac{\pi}{3}$

Out[10]= 6 x

§ I-3 If et Which

Dans certaines situations on souhaite spécifier sous quelles conditions une instruction (ou une suite d'instructions) doit être exécutée. Les commandes suivantes permettent de le faire:

- **`If[condition,t,f]`** retourne l'expression **`t`** si l'expression **`condition`** est vraie et **`f`** si elle est fausse.
- **`Which[condition1,expression1,condition2,expression2,...]`** retourne la première expression pour laquelle la condition est vraie et ne retourne rien si toutes les conditions sont fausses.

Exemple 2

Modifiez le module perimetrePolygon de l'exemple 1 de telle sorte qu'il retourne un message d'erreur lorsque l'utilisateur donne un rayon négatif.

```
In[11]:= perimetrePolygonMessage[r_, n_] := Module[
                                             module
   {a, c},
   If[r ≥ 0,
   si
    (* Instructions à exécuter lorsque r≥0: *)
    a = 2 Pi/n;  (* angle au centre *)
    c = 2 r Sin[a / 2];   (* côté *)
            sinus
    n c ,
    (* Instruction à exécuter lorsque r<0: *)
    "ERREUR: Le rayon doit être positif"
   ]
  ]

In[12]:= (* Tests: *)
perimetrePolygonMessage[2, 6]
perimetrePolygonMessage[-2, 6]

Out[12]= 12

Out[13]= ERREUR: Le rayon doit être positif

In[14]:= Clear[perimetrePolygonMessage]
         efface
```

Exercices

Exercice 1-1

Nous considérons que la liste d'instructions ci-dessous a été évaluée par Mathematica:

```
In[15]:= Clear[a, b, c, d, f, x]
         efface
         a = 3;
         b = 10;
         f[x_] := Module[
                  module
           {a, d},
           a = 100;
           c = 1000;
           d = 10 000;
           x + a + b
          ]
```

1. Devinez ce que retourne Mathematica lors de l'évaluation de `{a,b,c,d}`. Expliquez.
2. Devinez ce que retourne Mathematica si vous évaluez ensuite `f[1]`. Expliquez.
3. Devinez ce que retourne Mathematica si vous évaluez finalement `f[{a,b,c,d}]`. Expliquez.
4. Devinez ce que retourne Mathematica si vous évaluez d'abord `c=2`, puis `f[c]`. Expliquez.

Exercice 1-2

Créez un module **`aireImagePolygone[r_,n_,couleur_]`** qui, pour **`r`**, **`n`**, et **`couleur`** donnés par l'utilisateur, retourne l'aire du polygone à **`n`** côtés inscrit dans un cercle de rayon **`r`** ainsi qu'une représentation graphique du polygone avec sons cercle circonscrit, la couleur du polygone pouvant être choisi par l'utilisateur.
Testez votre module pour un hexagone bleu inscrit dans un cercle de rayon 2.

Indication: Pour la représentation du polygone, utilisez la commande **`Polygon`**.

Exercice 1-3

1. Définissez une fonction **`appreciation[note_]`** qui retourne **`insuffisant`** si **`note`** est inférieure à 4 et **`suffisant`** sinon. Définissez cette fonction une fois à l'aide de la commande **`If`** et une fois à l'aide de la commande **`Which`**.
2. Modifiez les deux fonctions de la partie précédente de telle sorte que le message d'erreur "la note doit être une nombre entre 0 et 6" est affiché si on applique la fonction à un nombre trop grand ou trop petit.

Exercice 1-4

1. Soit la fonction $f(x) = 2\sin(x) - x$.
   Tracez le graphe de cette fonction en y ajoutant en point à la position de chacun de ses zéros se trouvant entre $a = -3$ et $b = 3$.
   Prenez garde à définir vous différentes variables de telle façon à ce qu'une modification des définitions de `f`, `a` ou `b` modifie automatiquement le résultat lors d'une réévaluation du cahier.
2. Définissez un module `grapheZeros[g_,{min_,max_}]` qui trace le graphe d'une fonction `g` entre `min` et `max` en y ajoutant un point à la position de chacun de ses zéros. Testez votre module avec différentes fonctions dont vous connaissez les zéros.
3. Définissez un module `grapheZeros2[g_,{min_,max_}]` dont l'output contient la même représentation graphique que le module `grapheZeros` ainsi que les valeurs des différents zéros de `g` en code à virgule.
4. Définissez un module `grapheZeros3[g_,{min_,max_}]` qui retourne le même graphe que `grapheZeros2` ainsi que les valeurs des zéros de `g` situés dans l'intervalle `[min;max]` lorsque `g` possède des zéros dans cet l'intervalle. Si `g` ne possède pas de zéros dans cet intervalle, le module retournera uniquement le graphe de `g` agrémenté du message "g ne possède pas de zéros dans l'intervalle donné" sur fond jaune.

   Indication: la commande `Text` *permet d'afficher un texte et "fond" se dit "background" en anglais.*

Exercice 1-5

1. Déterminez la fonction cubique dont le graphe passe par les points (-1;1), (0;-1), (1;1) et (2;-1). Tracez le graphe de cette fonction et positionnez les points en rouge sur le graphe.
2. Définissez un module `cubique[listePoints_]` qui détermine l'équation de la fonction cubique dont le graphe passe par quatre points donnés par l'utilisateur dans `listePoints` et trace le graphe de la fonction ainsi que les quatre points dans un intervalle approprié. Les points doivent être rouges et bien visibles. L'output sera constitué du polynôme cubique et de la représentation graphique.

§ 2 Polynômes et fonctions rationnelles

§ 2-1 Fonctions définies par morceaux

Définition à l'aide de Piecewise

Mathematica offre une commande spécialement dédiée à la définition d'une fonction définie par morceaux: **`Piecewise[{{val1,cond1},{val2,cond2},...}]`**

Exemple 1

Soit la fonction f définie sur $\mathbb{R}$ avec $f(x) = \begin{cases} x^2 & \text{si } x \leq 1, \\ -x & \text{si } x > 1. \end{cases}$

On peut la définir de la manière suivante à l'aide de **`Piecewise`**:

```
In[19]:= f[x_] := Piecewise[{{x^2, x ≤ 1}, {-x, x > 1}}]
```
fonction par morceaux

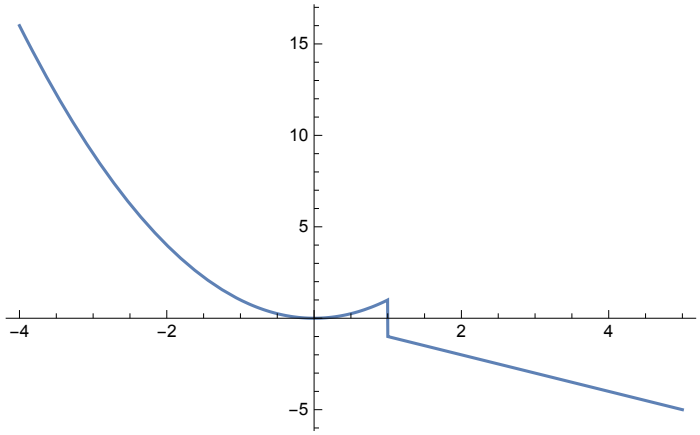
On peut contrôler qu'il s'agit bien de la fonction voulue:

```
In[20]:= Map[f, {-2, 1, 3}]
```
applique

```
Out[20]= {4, 1, -3}
```

```
In[21]:= Plot[f[x], {x, -4, 5}]
```
tracé de courbes

Out[21]=

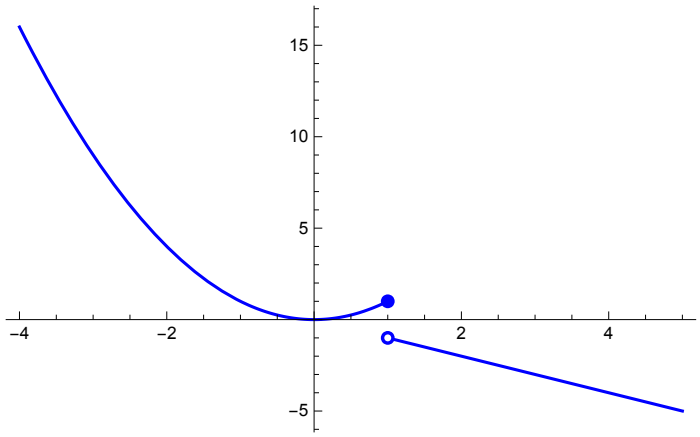


Si nous voulons mettre en évidence graphiquement où est situé $f(1)$, nous devons le faire manuellement:

In[22]:=
```
Plot[f[x], {x, -4, 5},
 (* On prescrit la couleur du graphe afin
  de pouvoir choisir la même couleur pour les points: *)
 PlotStyle → Blue,
 Epilog → {
   (* On place un point plein à l'endroit de l'image de 1: *)
   Blue, PointSize[0.02], Point[{1, 1}],
   (* Afin d'obtenir le rendu habituel d'un point vide,
   on superpose un point bleu d'un point blanc plus petit: *)
   Blue, PointSize[0.02], Point[{1, -1}],
   White, PointSize[0.01], Point[{1, -1}]
  }
]
```

tracé de courbes | active les messages | active les messages | style de tracé | bleu | épilogue | active les messages | bleu | taille des points | point | bleu | taille des points | point | blanc | taille des points | point

Out[22]=



In[23]:=
```
Clear[f]
```

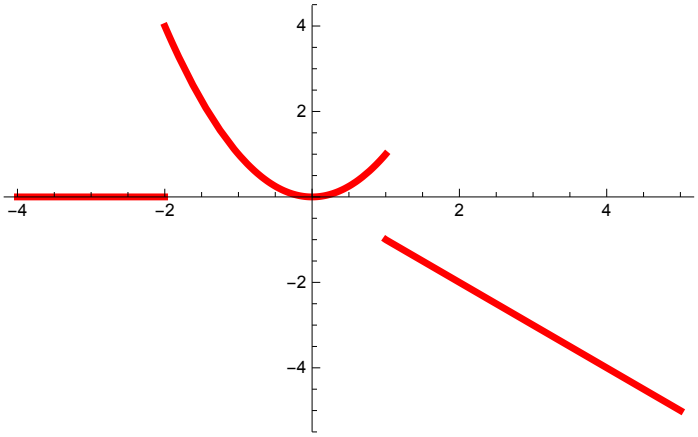
efface

Exemple 2

Attention néanmoins si la fonction n'est pas définie sur ℝ, par exemple $f(x) = \begin{cases} x^2 & \text{si } -2 \leq x \leq 1, \\ -x & \text{si } x > 1. \end{cases}$

In[24]:=
```
f[x_] := Piecewise[{{x^2, -2 ≤ x ≤ 1}, {-x, x > 1}}]
```
fonction par morceaux
```
Map[f, {-3, -2, 4}]
```
applique
```
Plot[f[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
```
tracé de courbes style de tracé ro··· épaisseur

Out[25]= {0, 4, -4}

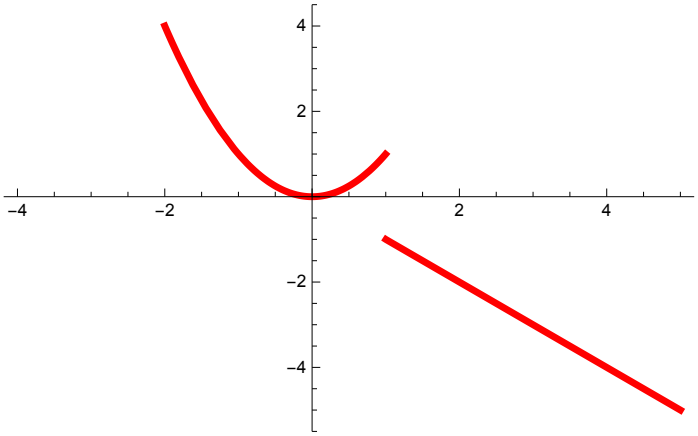
Out[26]=


In[27]:=
```
Clear[f]
```
efface

Nous constatons que **Piecewise** attribue par défaut la valeur 0 aux domaine où aucune valeur n'a été spécifiée. Pour y remédier, on y attribue la valeur **Null**:

In[28]:=
```
f[x_] := Piecewise[{{x^2, -2 ≤ x ≤ 1}, {-x, x > 1}}, Null]
```
fonction par morceaux expressi
```
Map[f, {-3, -2, 4}]
```
applique
```
Plot[f[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
```
tracé de courbes style de tracé ro··· épaisseur

Out[29]= {Null, 4, -4}

Out[30]=


In[31]:=
```
Clear[f]
```
efface

Méthodes alternatives pour définir des fonctions par morceaux: If, Which, /;

On aurait également pu utiliser les commandes **`If`** ou **`Which`** ou la notation raccourcie **`/;`**, comme illustré dans les exemples ci-dessous.

Exemple 3

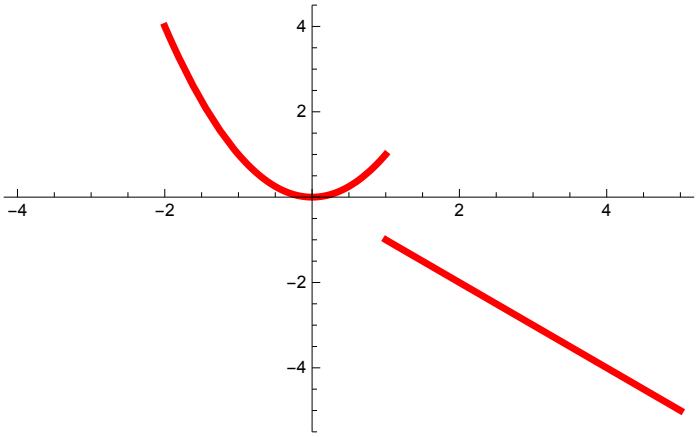
On souhaite à nouveau définir la fonction $f(x) = \begin{cases} x^2 & \text{si } -2 \leq x \leq 1, \\ -x & \text{si } x > 1. \end{cases}$

Définition à l'aide de la commande **`If`**:

In[32]:=
```
Clear[f]
efface
f[x_] := If[-2 ≤ x ≤ 1, x^2, If[x > 1, -x]]
         si                 si
Map[f, {-3, -2, 4}]
applique
Plot[f[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
tracé de courbes       style de tracé  ro···  épaisseur
```

Out[34]= {Null, 4, -4}

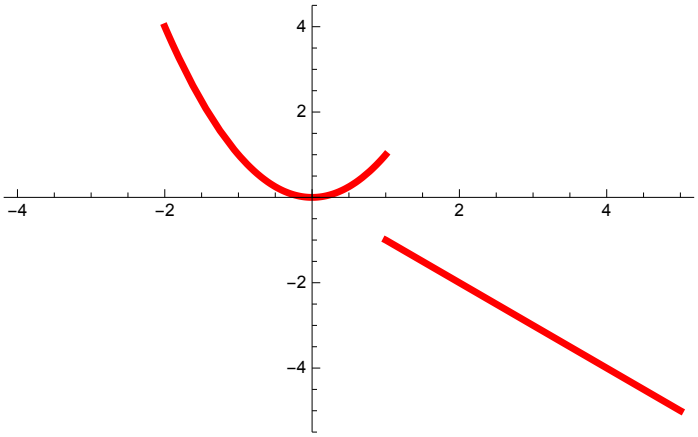
Out[35]=



Définition à l'aide de la commande **`Which`**:

In[36]:=
```
Clear[f]
efface
f[x_] := Which[-2 ≤ x ≤ 1, x^2, x > 1, -x]
          quel
Map[f, {-3, -2, 4}]
applique
Plot[f[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
tracé de courbes        style de tracé   ro···  épaisseur
```

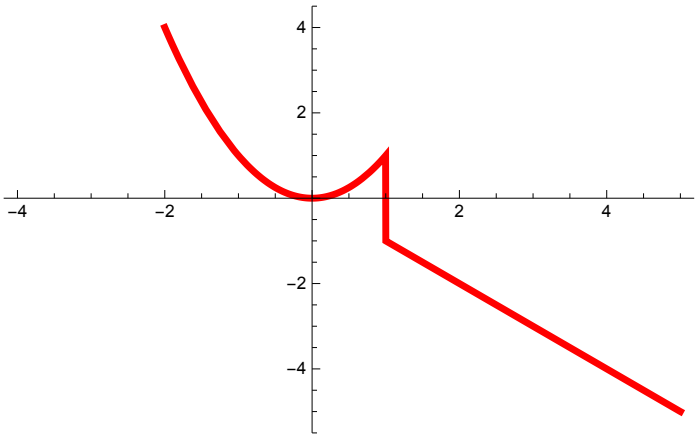
Out[38]= {Null, 4, -4}

Out[39]=


Définition à l'aide de /;

In[40]:=
```
Clear[f]
efface
f[x_] := x^2 /; -2 ≤ x ≤ 1
f[x_] := -x /; x > 1
Map[f, {-3, -2, 4}]
applique
Plot[f[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
tracé de courbes        style de tracé   ro···  épaisseur
```

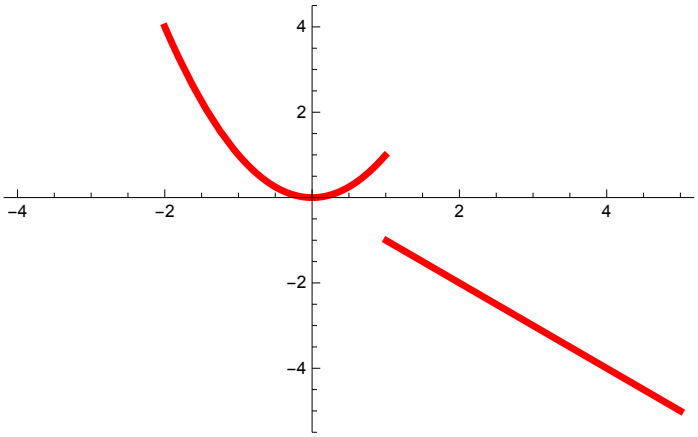
Out[43]= {f[-3], 4, -4}

Out[44]=


Nous pouvons constater qu'avec cette méthode, Plot gère mal le le comportement en $x = 1$. Afin d'obtenir un meilleur rendu, nous pouvons recourir à une astuce: au lieu d'afficher le graphe de la fonction f, on affiche le graphe d'une fonction g égale à f à l'exception près qu'elle n'est pas définie dans un petit intervalle autour de la valeur problématique:

In[45]:=
```
g[x_] := f[x] /; x < 0.99 || x > 1.01
Plot[g[x], {x, -4, 5}, PlotStyle → {Red, Thickness[0.01]}]
```
tracé de courbes style de tracé ro··· épaisseur

Out[46]=



In[47]:=
```
Clear[f, g]
```
efface

La valeur absolue

La valeur absolue pourrait être définie par morceaux, mais elle est déjà implémentée dans Mathematica. Il s'agit de la fonction **`Abs`**.

§ 2-2 Division euclidienne

Les fonctions **`PolynomialQuotient`** et **`PolynomialRemainder`** permettent d'obtenir le quotient, respectivement le reste de la division euclidienne de polynômes. Les arguments sont le dividende, le diviseur et la variable.

Exemple

On détermine le quotient et le reste de la division $\frac{x^2-3x+5}{x-2}$:

In[48]:=
```
PolynomialQuotient[x^2 - 3 x + 5, x - 2, x]
```
Quotient du polynômes
```
PolynomialRemainder[x^2 - 3 x + 5, x - 2, x]
```
reste de polynômes

Out[48]= $-1 + x$

Out[49]= 3

Pour rappel: ceci signifie que $x^2 - 3x + 5 = (x-1)(x-2) + 5$, c'est-à-dire que $\frac{x^2-3x+5}{x-2} = x - 1 + \frac{3}{x-2}$.

§ 2-3 Limites

L'instruction `Limit[f[x],x->a]` calcule la limite $\lim_{x\to a} f(x)$, où `a` peut être un nombre, une expression algébrique, `Infinity` (∞) ou `-Infinity` (-∞).
Lorsque `a` est fini, `Limit[f[x],x->a]` détermine la limite lorsque `x` s'approche de `a` par la droite; on obtiendrait le même résultat par `Limit[f[x],x->a,Direction->-1]`. Si on souhaite calculer la limite à gauche, il faut le spécifier: `Limit[f[x],x->a,Direction->1]`.

Exemple

Déterminez le comportement de la fonction $f(x) = \frac{x}{2x-4}$ au bords de son domaine de définition.

Puisque $D_f = \mathbb{R}\backslash\{2\}$, on commence par calculer les limites de $f(x)$ lorsque x tend vers 2:

In[50]:=
```
f[x_] := x/(3 x - 6)
Limit[f[x], x → 2, Direction → 1]
Limit[f[x], x → 2, Direction → -1]
```
limite direction
limite direction

Out[51]= $-\infty$

Out[52]= ∞

La fonction f possède donc une asymptote verticale d'équation $x = 2$ du type "puits infini".

On détermine ensuite le comportement de f à l'infini:

In[53]:=
```
Limit[f[x], x → -Infinity]
Limit[f[x], x → Infinity]
```
limite infini
limite infini

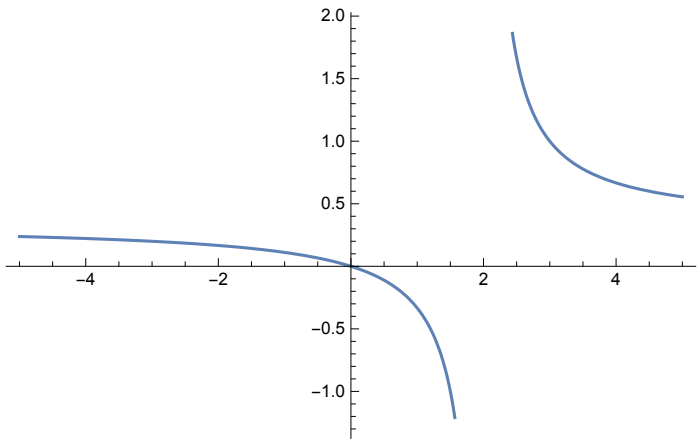
Out[53]= $\frac{1}{3}$

Out[54]= $\frac{1}{3}$

La fonction f admet donc une asymptote d'équation $y = \frac{1}{3}$ à gauche et à droite.

In[55]:=
```
Plot[f[x], {x, -5, 5}]
```
tracé de courbes

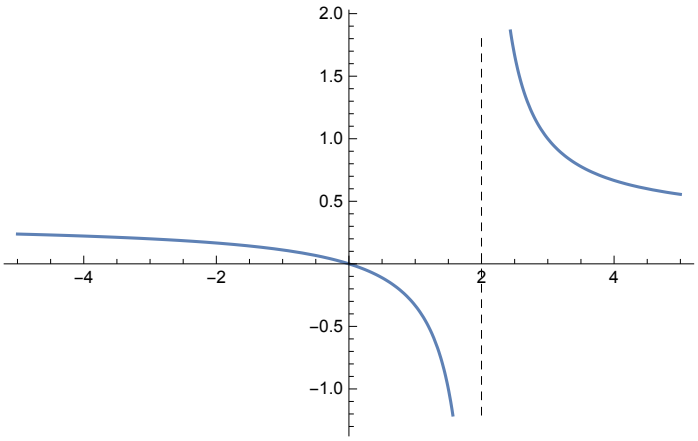
Out[55]=



Depuis Mathematica 11, les problèmes d'affichage des fonctions comportant des asymptotes verticales ont été résolus et il est possible de spécifier les style d'affichage de ces asymptotes à l'aide de **`ExclusionsStyle`**:

In[56]:=
```
Plot[f[x], {x, -5, 5}, ExclusionsStyle → Dashed]
```
tracé de courbes style d'exclusions rayé

Out[56]=

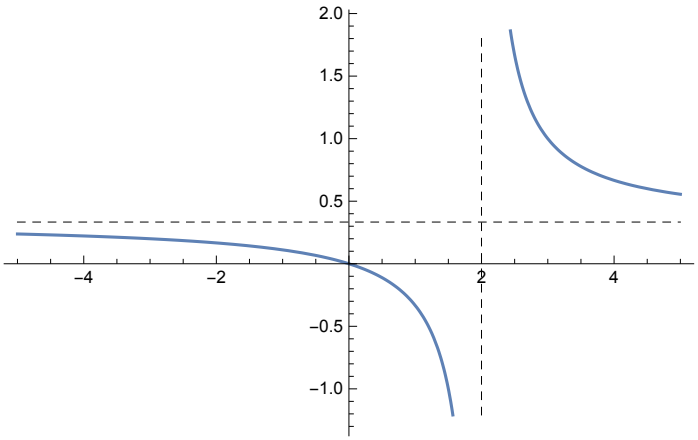


Il faut en revanche ajouter l'asymptote horizontale à la main:

In[57]:=
```
xMin = -5;
xMax = 5;
Plot[f[x], {x, xMin, xMax}, ExclusionsStyle → Dashed,
 Epilog → {Dashed, Line[{{xMin, 1/3}, {xMax, 1/3}}]}]
```
tracé de courbes style d'exclusions rayé

épilogue rayé ligne

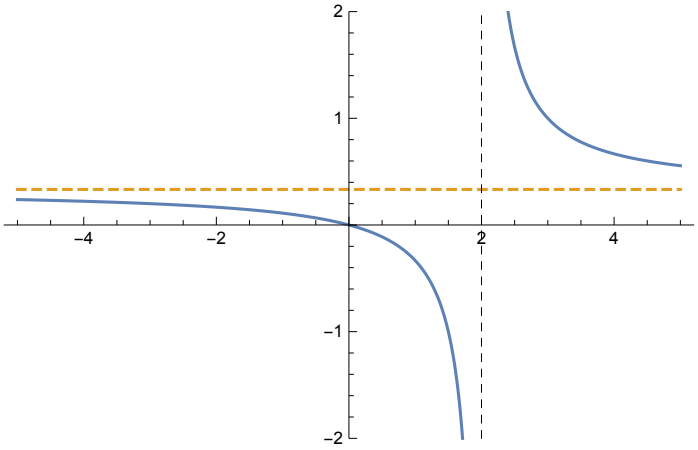
Out[59]=



Deuxième possibilité: on ajoute l'asymptote horizontale en tant que graphe d'une fonction:

In[60]:=
```
Plot[{f[x], 1/3}, {x, xMin, xMax}, ExclusionsStyle → Dashed,
 PlotStyle → {Automatic, Dashed}, PlotRange → {-2, 2}]
```
tracé de courbes | style d'exclusions | rayé | style de tracé | automatique | rayé | zone de tracé

Out[60]=



In[61]:=
```
Clear[f, xMin, xMax]
```
efface

Exercices

Exercice 2-1

Tracez le graphe de la fonction $f(x) = \begin{cases} -1 & \text{si } x \leq -1, \\ x^2 & \text{si } -1 < x < 1, \\ 1 & \text{si } x \geq 1. \end{cases}$

Pour ce faire, procédez de quatre manières différentes.

Exercice 2-2

Tracez le graphe de la fonction $g(x) = |\sin(x)|$ pour $x \in [-4\pi; 4\pi]$ en graduant l'axe des abscisses tous les $\frac{\pi}{2}$.

Exercice 2-3

Jusqu'à la version 10, Mathematica gère mal l'affichage du graphe d'une fonction rationnelle: Mathematica calcule les coordonnées d'un grand nombre de points puis relie ceux-ci. Aux abscisses où f possède une asymptote verticale, ceci donne lieu à une droite verticale étant interprétée comme faisant partie du graphe, alors qu'une asymptote verticale n'en fait bien sûr pas partie. Pour éviter ce problème, il suffit de ne pas définir la fonction dans un petit intervalle autour de la valeur interdite. Si par exemple -2 est une valeur interdite, on définit f uniquement sur $\mathbb{R}\backslash]-2.1;-1.9[$.

Dans Mathematica 11, le problème a été corrigé et on peut spécifier la représentation de l'asymptote à l'aide de la commande **`ExclusionsStyle`** (voir l'exemple du paragraphe sur les limites).

Tracez le graphe de la fonction $f(x) = \frac{x}{x-1}$ à l'aide de Mathematica 10 en tenant compte de la remarque ci-dessus, puis à l'aide de Mathematica 11. Dans les deux cas, affichez l'asymptote verticale en traitillés.

Exercice 2-4

On considère la fonction $f(x) = \frac{\sin(x)}{x}$.

1. Calculer $\lim_{x\to-\infty} f(x)$ et $\lim_{x\to+\infty} f(x)$ et interprétez les résultats graphiquement.
2. Calculez $\lim_{x\uparrow 0} f(x)$ et $\lim_{x\downarrow 0} f(x)$ et interprétez les résultats graphiquement.
3. Affichez le graphe de f en y ajoutant les interprétations graphiques trouvées ci-dessus.

Exercice 2-5

1. Rappelez et justifiez comment la division euclidienne permet de déterminer l'éventuelle asymptote affine d'une fonction rationnelle.
2. On considère la fonction $f(x) = \frac{x^2-2x+1}{x-3}$.
   - **2.1.** Déterminez l'équation de chacune des asymptotes de la fonction f.
   - **2.2.** Illustrez votre résultat graphiquement.

Exercice 2-6

Définissez un module **`asymptoteAffine[f_,{a_,b_}]`** dont l'argument **`f`** est une fonction rationnelle. Le module décidera si la fonction **`f`** admet une asymptote affine. Si tel est le cas, il fournira un output contenant l'équation de cette asymptote ainsi qu'une représentation graphique de **`f`** sur l'intervalle **`[a;b]`** avec son asymptote affine en traitillés. Si la fonction **`f`** n'admet pas d'asymptote affine, le module fournira comme output uniquement le graphe de **`f`** avec le message "la fonction f n'admet pas d'asymptote affine" sur fond jaune.

Indications:

- *Commencez par créer un module qui fonctionne lorsque l'asymptote affine existe.*
- *La commande **`Exponent`** permet de déterminer le degré d'un polynôme.*

Exercice 2-7

1. Définissez des fonctions booléennes **`fctPaireQ[f_]`** et **`fctImpaire[f_]`** qui déterminent si la fonction **`f`** est paire, respectivement impaire.

   *Indication: Travaillez avec **`Reduce`** et **`TrueQ`**.*

2. Utilisez les fonctions précédentes pour déterminer la parité des fonctions $f(x) = \sin(x)\cos(10\,x)$ et $g(x) = \sqrt[5]{\sin(x) + 1}$.

   *Indication: Pour la racine cinquième, il est nécessaire d'utiliser la commande **`Surd`** car Mathematica calcule la racine cinquième de nombres négatifs dans les nombres complexes si on travaille avec des exposants rationnels. Par exemple l'expression* $(-32)^{1/5}$ retourne $2\,(-1)^{1/5}$. Si on demande à Mathematica de développer cette expression (avec la fonction **`ComplexExpand`**)*, on peut constater que ce nombre possède bien une partie imaginaire.*

3. Définissez une fonction **`pariteFct[f_]`** qui retourne la réponse "paire" si **`f`** est paire, la réponse "impaire" si **`f`** est impaire et la réponse "ni paire, ni impaire" sinon.

§ 3 Composition de fonctions et fonction réciproque

§ 3-1 Composition de fonctions

Une composition de deux fonctions se réalise de manière tout à fait naturelle à l'aide de Mathematica:

Exemple

Déterminez l'équation de la fonction $h = f \circ g$ pour $g(x) = \sqrt{x+3}$ et $f(x) = \frac{4}{x^2-1}$.

```
In[62]:= f[x_] := 4/(x^2 - 1)
         g[x_] := Sqrt[x + 3]
                  racine carrée
         h[x_] := f[g[x]]
         h[x]
```

Out[65]= $\frac{4}{2+x}$

Rappel: La détermination du domaine de définition d'une composition de deux fonctions doit se faire <u>avant</u> de simplifier la nouvelle équation.
Afin qu'un nombre x appartienne au domaine de définition de $h = f \circ g$, les deux étapes de la composition $x \to g(x) \to f(g(x))$ doivent être réalisables, c'est-à-dire x doit remplir deux conditions:

$$x \in \mathcal{D}_g \quad \text{et} \quad g(x) \in \mathcal{D}_f.$$

Pour les fonctions définies ci-dessus, $\mathcal{D}_g = [-3; +\infty[$ et $\mathcal{D}_f = \mathbb{R}\backslash\{-1; 1\}$. On peut facilement déterminer les valeurs exclues du domaine de définition de h en raison de la deuxième étape:

```
In[66]:= Reduce[g[x]^2 - 1 == 0]
         réduis
```

Out[66]= $x == -2$

Ainsi, $\mathcal{D}_h = [-3; +\infty[\backslash\{-2\}$.

Nous pouvons constater que Mathematica gère mal le domaine de définition d'une composition:

```
In[67]:= h[-5]
```

Out[67]= $-\frac{4}{3}$

```
In[68]:= FunctionDomain[h[x], x]
         domaine de fonction
```

Out[68]= $x < -2 \,||\, x > -2$

```
In[69]:= Clear[f, g, h]
         efface
```

§ 3-2 Fonction réciproque

Mathematica propose une commande permettant de déterminer la réciproque d'une fonction bijective: Il s'agit de la commande **`InverseFunction`**

Exemple

Déterminez la fonction réciproque de la fonction bijective $f(x) = 3x + 5$.

```
In[70]:= f[x_] := 3 x + 5
         rf[x_] := InverseFunction[f][x]
                   fonction inverse
         rf[x]
```

Out[72]= $\frac{1}{3}(-5 + x)$

```
In[73]:= Clear[f, rf]
         efface
```

Exercices

Exercice 3-1

On considère la fonction $g(x) = \frac{x-1}{x+2}$.

1. Déterminez l'ensemble de départ A et l'ensemble d'arrivée B tels que la fonction $g : A \to B$ soit bijective.
2. Déterminez la fonction réciproque h de g sans utiliser la commande **`InverseFunction`**.
3. Montrez que la fonction h de l'item précédent est bien la réciproque de g en montrant que $h \circ g(x) = x$ et $g \circ h(y) = y$.
4. Définissez une fonction **`reciproque[f_]`** qui retourne l'équation de la fonction réciproque d'une fonction bijective **`f`**.

§ 4 Fonctions exponentielles et logarithmes

§ 4-1 Logarithmes

Calculer un logarithme se fait à l'aide de la commande **Log[b,x]** ou **b** est la base et **x** est l'argument. Si on ne précise pas la base, Mathematica calcule par défaut le logarithme naturel ln.
On peut constater que puisque Mathematica calcule avec des nombres complexes, des arguments strictement négatifs ne sont pas interdits.

Exemples 1

In[74]:= $\mathbf{Log}\left[3, \frac{1}{81}\right]$
logarithme

Out[74]= -4

In[75]:= **Log[E]**
lo··· nombre e

Out[75]= 1

In[113]:= **Log[-3]**
logarithme

Out[113]= $i\,\pi + \mathrm{Log}[3]$

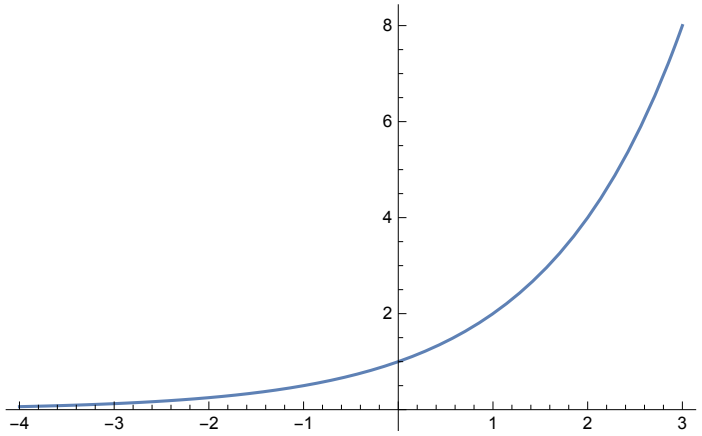
§ 4-2 Fonctions exponentielles

Les fonctions exponentielles se définissent de manière tout à fait naturelle:

Exemple 2

In[77]:=
```
f[x_] := (2)^x
Plot[f[x], {x, -4, 3}]
```
tracé de courbes

Out[78]=



La fonction exponentielle en base *e*, c'est-à-dire la fonction exp, est déjà définie dans Mathematica, mais on peut également obtenir l'expression e^x en utilisant le nombre *e* (qui s'écrit évidemment en majuscule dans Mathematica...):

In[79]:=
```
Exp[x]
```
exponent
```
E^x
```
nombre e

Out[79]= e^x

Out[80]= e^x

Lorsque nous résolvons des équations exponentielles, nous devons nous rappeler que par défaut, Mathematica calcule avec des nombres complexes. Si nous ne nous intéressons qu'aux solutions réelles d'une équation, nous devons le préciser.

Exemple 3

In[81]:=
```
Reduce[2^x == 3, x]
```
réduis
```
Solve[2^x == 3, x]
```
résous

Out[81]= $C[1] \in \text{Integers} \,\&\&\, x == \frac{2\, i\, \pi\, C[1]}{\text{Log}[2]} + \frac{\text{Log}[3]}{\text{Log}[2]}$

Out[82]= $\left\{\left\{x \to \text{ConditionalExpression}\left[\frac{2\, i\, \pi\, C[1]}{\text{Log}[2]} + \frac{\text{Log}[3]}{\text{Log}[2]},\ C[1] \in \text{Integers}\right]\right\}\right\}$

Pour n'obtenir que la solution réelle:

```
In[83]:= Reduce[2^x == 3, x, Reals]
         réduis          nombres
         Solve[2^x == 3, x, Reals]
         résous          nombres r
```

Out[83]= $x == \frac{\text{Log}[3]}{\text{Log}[2]}$

Out[84]= $\left\{\left\{x \to \frac{\text{Log}[3]}{\text{Log}[2]}\right\}\right\}$

Mathematica donne la solution en utilisant le logarithme naturel ln que Mathematica désigne par **Log**. Mais depuis le chapitre sur les fonctions exponentielles et logarithmes du cours de mathématiques de base nous savons que $\frac{\ln(3)}{\ln(2)} = \log_2(3)$,

---

Exercices

Exercice 4-1

A l'aide de Mathematica, résolvez les équations suivantes dans $\mathbb{R}$ après avoir déterminé leurs domaines de définition. Donnez à chaque fois l'ensemble des solutions sous forme exacte (si possible) et sous forme arrondie à 3 chiffres significatifs.

1. $e^x = \frac{15}{e^{2x}}$
2. $8 \cdot 2^{3x} - 2^{2x} - 40 \cdot 2^x + 5 = 0$
3. $\ln(x^2 - 81) = 2$
4. $\log_5\left(\frac{x+2}{10-x}\right) + \log_5(x-1) = \log_5(x^2) - 1$
5. $\log\left(\log_{\frac{1}{10}}(x)\right) = 2$

§ 5 Fonctions trigonométriques

Nous avons déjà utilisé les fonctions trigonométriques **Sin**, **Cos** et **Tan** dans Mathematica, tout en constatant que par défaut, Mathematica calcule en radians.

Mathematica est capable de résoudre des équations trigonométriques:

Exemples

In[85]:=
```
Reduce[Cos[x] == 1/2, x, Reals]
Solve[Cos[x] == 1/2, x, Reals]
```
réduis cosinus nombres
résous cosinus nombres r

Out[85]= $C[1] \in \text{Integers} \,\&\&\, \left(x == -\frac{\pi}{3} + 2\pi C[1] \,||\, x == \frac{\pi}{3} + 2\pi C[1]\right)$

Out[86]= $\left\{\left\{x \to \text{ConditionalExpression}\left[-\frac{\pi}{3} + 2\pi C[1], C[1] \in \text{Integers}\right]\right\}, \left\{x \to \text{ConditionalExpression}\left[\frac{\pi}{3} + 2\pi C[1], C[1] \in \text{Integers}\right]\right\}\right\}$

Mathematica nous donne bien les deux familles infinies de solutions:
$S = \left\{-\frac{\pi}{3} + k \cdot 2\pi \mid k \in \mathbb{Z}\right\} \cup \left\{\frac{\pi}{3} + k \cdot 2\pi \mid k \in \mathbb{Z}\right\}$.

Nous pouvons déterminer des solutions particulières en choisissant une valeur précise de $C[1]$ dans les solutions données par Mathematica:

In[87]:=
```
Solve[Cos[x] == 1/2, x, Reals] /. C[1] → 1
```
résous cosinus nombres r··· constante C

Out[87]= $\left\{\left\{x \to \frac{5\pi}{3}\right\}, \left\{x \to \frac{7\pi}{3}\right\}\right\}$

Mais Mathematica ne nous donne pas toujours l'ensemble des solutions sous la forme la plus simple:

In[88]:=
```
Reduce[Sin[x] == 0, x, Reals]
```
réduis sinus nombres

Out[88]= $C[1] \in \text{Integers} \,\&\&\, \left(x == 2\pi C[1] \,||\, x == \pi + 2\pi C[1]\right)$

Dans ce cas-ci nous pouvons regrouper les deux familles de solutions: $S = \{k \cdot \pi \mid k \in \mathbb{Z}\}$.

Exercices

Exercice 5-1

1. Déterminez l'ensemble des solutions de l'équation $\sin(x)\cos^2(x) = \frac{1}{4}\sin(x)$.
2. Représentez le cercle trigonométrique dans son repère orthonormé en y ajoutant les points associés aux solutions de l'équation de la partie 1 bien visibles en rouge. A l'aide de cette représentation, déterminez l'ensemble des solutions sous sa forme la plus simple.

§ 6 Suites et séries

§ 6-1 Suites

Intuitivement, une suite numérique est une liste ordonnée de nombres réels $a_0,\ a_1,\ a_2,\ a_3,\ \ldots$

Elle peut être finie (c'est-à-dire comporter un nombre fini d'éléments) ou infinie (c'est-à-dire comporter un nombre infini d'éléments). Mais on peut définir la notion de suite de manière plus précise:

Définition:

Une *suite numérique* est une fonction

$$\begin{aligned} a:\ \mathbb{N} &\to \mathbb{R} \\ n &\mapsto a(n) = a_n \end{aligned}$$

Le nombre a_n est appelé le *n-ième terme* de la suite. La suite entière est dénotée par $(a_n)_{n\in\mathbb{N}}$ ou simplement (a_n).

L'expression algébrique décrivant le terme a_n en fonction de n s'appelle *terme général* de la suite.

Parfois une suite peut être décrite par une *relation de récurrence*, c'est-à-dire par le procédé permettant de calculer un terme à partir de son (ou ses) prédécesseur(s):

$$a_0 \text{ (valeur initiale)}, \quad a_{n+1} = f(a_n) \text{ pour } n \geq 0.$$

Exemple

Soit $(a_n)_{n\in\mathbb{N}}$ la suite des nombres pairs supérieurs ou égaux à 0, c'est-à-dire

$$a_0 = 0,\ a_1 = 2,\ a_2 = 4,\ a_3 = 6,\ \ldots$$

Le terme général de cette suite est $a_n = 2\,n$.

La relation de récurrence permettant de définir cette suite est $a_0 = 0,\quad a_{n+1} = a_n + 2$ pour $n \geq 0$.

---

§ 6-2 Calcul des termes d'une suite à l'aide de Mathematica

Lorsque la suite est définie par son terme général, on peut facilement déterminer autant de termes que souhaité à l'aide de la fonction **`Table`**:

Exemple

Déterminez les 10 premiers termes de la suite $(a_n)_{n \in \mathbb{N}}$ avec $a_n = 2\,n$.

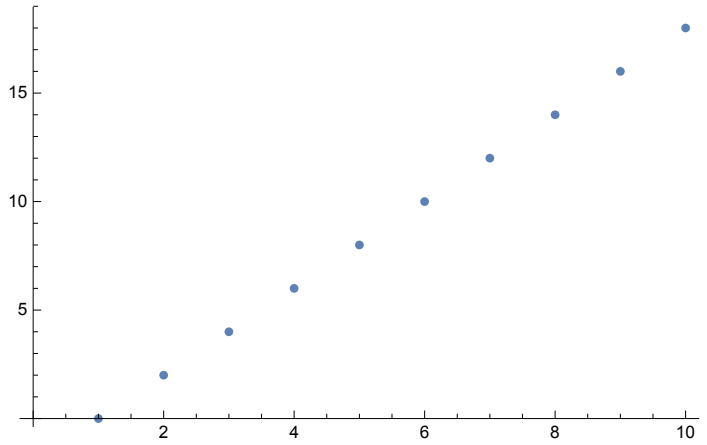
In[89]:=
```
suite = Table[2 n, {n, 0, 9}]
```
table

Out[89]= {0, 2, 4, 6, 8, 10, 12, 14, 16, 18}

Nous pouvons également représenter ces termes graphiquement à l'aide de **`ListPlot`**. Attention néanmoins: si nous ne spécifions pas les abscisses, Mathematica considère que la première abscisse vaut 1:

In[90]:=
```
ListPlot[suite]
```
tracé de liste

Out[90]=

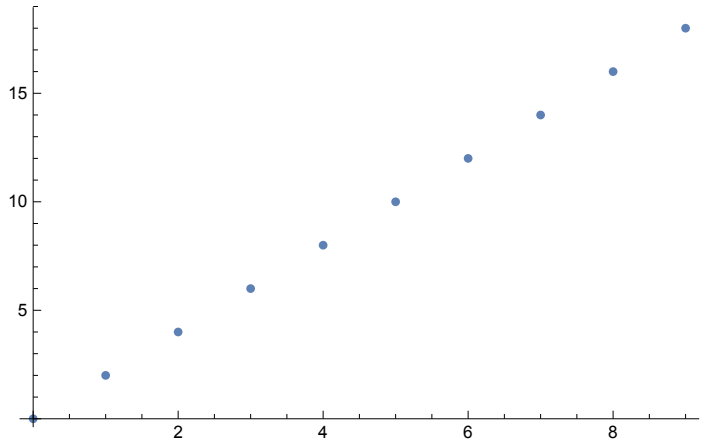


Mais nous pouvons facilement y remédier:

In[91]:=
```
points = Transpose[{Range[0, 9], suite}];
```
transposée plage

```
ListPlot[points]
```
tracé de liste

Out[92]=



In[93]:=
```
Clear[suite, points]
```
efface

Si la suite est défini par une relation de récurrence, nous pouvons travailler avec la fonction **NestList** que nous connaissons déjà: nous avons vu que la fonction **NestList[f,expr,n]** retourne la liste des résultats de l'application de la fonction **f** à **expr** de 0 à **n** fois. Si nous nous intéressons uniquement au dernier terme, nous utilisons **Nest**.

Exemple 1

Nous considérons la suite $(a_n)_{n\in\mathbb{N}}$ définie par récurrence de la manière suivante:

$$a_0 = 3, \qquad a_{n+1} = \frac{2 + a_n^2}{2\, a_n} \quad \text{pour } n \geq 0.$$

On a donc $a_1 = \frac{2+a_0^2}{2\,a_0} = \frac{2+3^2}{2\cdot 3} = \frac{11}{6}$, $a_2 = \frac{2+a_1^2}{2\,a_1} = \frac{2+\frac{11}{6}}{2\cdot\frac{11}{6}} = \frac{193}{132}$ et ainsi de suite, chaque terme se calcule à partir du précédent.

Donnez la liste des termes a_0 jusqu'à a_{10} puis définir la fonction **a[n_]** donnant la valeur du terme a_n (donner les résultats sous forme de code à virgule).

In[94]:=
```
succ[x_] := (2 + x^2)/(2 x)
(* Fonction permettant de calculer le successeur d'un terme x *)
NestList[succ, 3., 10]  (* génération des termes d'indices 0 à 10 *)
```
liste d'imbrication

Out[95]=
```
{3., 1.83333, 1.46212, 1.415, 1.41421,
 1.41421, 1.41421, 1.41421, 1.41421, 1.41421}
```

In[96]:=
```
a[n_] := Nest[succ, 3., n]  (* Fonction déterminant uniquement le terme d'indice n *)
```
imbrique
```
a[4]  (* test *)
```
SetDelayed: Tag Integer in 3[n_] is Protected.

Out[96]=
```
$Failed
```

Out[97]=
```
3[4]
```

Plutôt que de spécifier le nombre de fois qu'une fonction doit être appliquée à une expression, on peut imposer une condition sur le nombre d'applications. Ceci se fait avec la fonction **NestWhile[f,expr,test]** qui applique la fonction **f** à l'expression **expr** tant que ("while" en anglais) la valeur de la fonction booléenne **test** appliquée au dernier résultat est vrai. Mathematica retourne alors le résultat de la dernière application de **f**. Si on souhaite connaître tous les termes calculés, Mathematica propose la fonction **NestWhileList**. Ces deux fonctions nous seront par exemple utiles dans le chapitre suivant lorsque nous allons définir des suites permettant d'approcher les solutions d'une équation: nous allons arrêter la procédure lorsque la qualité de l'approximation sera suffisamment bonne.

Exemple 2

Déterminez le premier terme qui est inférieur à 2 dans la suite $(a_n)_{n\in\mathbb{N}}$ définie de la façon suivante:

$$a_0 = 1000, \qquad a_{n+1} = \sqrt{a_n}\,.$$

In[98]:=
```
(* Critère d'arrêt: x<2; on doit continuer tant que x≥2 *)
crit[x_] := x ≥ 2
NestWhile[Sqrt, 1000., crit]
```
imbrique p··· racine carrée

Out[99]= 1.53993

In[100]:=
```
NestWhileList[Sqrt, 1000., crit]
```
liste de résultats ··· racine carrée

Out[100]= {1000., 31.6228, 5.62341, 2.37137, 1.53993}

In[101]:=
```
Clear[crit]
```
efface

§ 6-3 Limite d'une suite

Dans le cas d'une suite définie par son terme général, le calcul de sa limite se fait de la même façon que pour le calcul de la limite à l'infini d'une fonction (ce qui est normal puisqu'une suite n'est rien d'autre qu'une fonction définie sur $\mathbb{N}$).

Dans le cas où la suite est définie par récurrence, c'est-à-dire en donnant une valeur initiale a_0 et une fonction f telle que $a_{n+1} = f(a_n)$, on ne peut pas utiliser la commande **`Limit`**. Mais si la fonction **`f`** est continue, la limite L de la suite (si elle existe) est un point fixe de la fonction f, c'est-à-dire que

$$L = f(L) \qquad \text{car} \quad a_{n+1} = f(a_n) \implies \lim_{n\to+\infty} a_{n+1} = \lim_{n\to+\infty} f(a_n)$$

La fonction **`FixedPoint`** permet d'obtenir une estimation numérique de la valeur de la limite. La fonction **`FixedPointList`** donnera en plus tous les termes avant l'approximation de la limite. Il ne faut pas oublier de faire travailler Mathematica avec des nombres en code à virgule lors de l'utilisation de **`FixedPoint`** ou **`FixedPointList`**, sinon le point fixe ne sera jamais atteint.
Si un tel calcul ne s'arrête pas, il est parfois possible de forcer l'arrêt en sélectionnant Abort evaluation dans le menu Evaluation.

ATTENTION!!! L'évaluation de **`FixedPoint`** ou **`FixedPointList`** peut saturer la mémoire de l'ordinateur lorsque la limite n'existe pas ou que la suite converge trop lentement. Parfois il est impossible de forcer l'arrêt. Avant d'évaluer ces fonctions, mieux vaut se convaincre que la suite converge vraiment, soit par un raisonnement mathématique, soit en calculant d'abord un nombre déterminé de termes. Et avant chaque évaluation de ses deux fonctions, on s'assure qu'on a bien sauvegardé son cahier!!!

Exemple

Déterminer une approximation de la limite de la suite $(a_n)_{n\in\mathbb{N}}$ définie par

$$a_0 = 1000, \qquad a_{n+1} = \sqrt{a_n}\,.$$

In[102]:=
```
a = FixedPointList[Sqrt, 1000.]
```
liste de point fixe racine carrée

Out[102]=
```
{1000., 31.6228, 5.62341, 2.37137, 1.53993, 1.24094, 1.11397, 1.05545, 1.02735,
 1.01358, 1.00677, 1.00338, 1.00169, 1.00084, 1.00042, 1.00021, 1.00011, 1.00005,
 1.00003, 1.00001, 1.00001, 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}
```

Pour mieux comprendre la situation, affichons toutes les décimales calculées par Mathematica:

In[103]:= **NumberForm[a, 16]**
apparence numérique

Out[103]//NumberForm=

```
{1000., 31.62277660168379, 5.623413251903491, 2.371373705661656,
 1.539926526059492, 1.24093776075172, 1.113973859994802, 1.055449600878603,
 1.027350768179303, 1.013583133334066, 1.006768659292722,
 1.003378622102705, 1.001687886570815, 1.000843587465501, 1.000421704815275,
 1.000210830182954, 1.000105409535892, 1.000052703379123, 1.000026351342365,
 1.000013175584384, 1.000006587770493, 1.000003293879822, 1.000001646938555,
 1.000000823468938, 1.000000411734384, 1.000000205867171, 1.00000010293358,
 1.000000051466789, 1.000000025733394, 1.000000012866697, 1.000000006433349,
 1.000000003216674, 1.000000001608337, 1.000000000804169, 1.000000000402084,
 1.000000000201042, 1.000000000100521, 1.00000000005026, 1.00000000002513,
 1.000000000012565, 1.000000000006283, 1.000000000003141, 1.00000000000157,
 1.000000000000785, 1.000000000000393, 1.000000000000196, 1.000000000000098,
 1.000000000000049, 1.000000000000024, 1.000000000000012, 1.000000000000006,
 1.000000000000003, 1.000000000000001, 1.000000000000001, 1., 1.}
```

Nous constatons que Mathematica calcule des termes de la suite jusqu'à ce que la différence de deux termes consécutifs soient égaux (à la précision par défaut de 16 chiffres significatifs près). Nous en déduisons que $\lim_{n \to +\infty} a_n = 1$.

Si on avait utilisé la commande **FixedPointList[Sqrt,1000]**, c'est-à-dire qu'on avait donné la valeur initiale sous forme exacte, Mathematica aurait produit une liste de la forme suivante (voici les 20 premiers termes):

In[104]:= **NestList[Sqrt, 1000, 20]**
liste d'imb··· racine carrée

Out[104]= $\{1000, 10\sqrt{10}, 10^{3/4}, 10^{3/8}, 10^{3/16}, 10^{3/32}, 10^{3/64}, 10^{3/128}, 10^{3/256}, 10^{3/512}, 10^{3/1024}, 10^{3/2048}, 10^{3/4096}, 10^{3/8192}, 10^{3/16384}, 10^{3/32768}, 10^{3/65536}, 10^{3/131072}, 10^{3/262144}, 10^{3/524288}, 10^{3/1048576}\}$

Puisque tous les termes sont différents, **FixedPointList** ne s'arrêterait jamais et il aurait fallu interrompre l'évaluation manuellement (ce qui peut s'avérer impossible si la mémoire de l'ordinateur est complètement saturée).

§ 6-4 Sommes et séries

La fonction **Sum[a[k],{k,min,max}]** retourne la valeur de la somme $\sum_{k=\min}^{\max} a_k$. Il est à noter que **max** peut prendre la valeur **Infinity**, ce qui permet de déterminer la série correspondante. Pour calculer la valeur d'une somme il est aussi possible d'utiliser le symbole de sommation se trouvant dans une des palettes.

Exemple

Calculer la série $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$

In[105]:=
```
Sum[(-1)^(k+1)/k, {k, 1, Infinity}]
somme                  infini
```

Out[105]= Log[2]

Le résultat signifie que la série harmonique alternée converge vers ln(2).

La fonction **Sum** peut seulement être appliquée lorsque le terme général de la suite est connue. Si on dispose uniquement d'une relation de récurrence, il est toujours possible de calculer des sommes finies, par exemple à l'aide de la fonction **Total**:

Exemple

Calculer la somme des 20 premiers termes de la suite définie par la relation de récurrence suivante:

$$a_0 = 2, \quad a_{n+1} = \frac{a_n}{2} + 3$$

On commence par créer la suite des 20 premiers termes:

In[106]:=
```
succ[x_] := x/2 + 3
suite = NestList[succ, 2, 19];
        liste d'imbrication
```

Ensuite on calcule leur somme:

In[108]:=
```
Total[suite]
total
```

Out[108]= $\frac{14\,680\,065}{131\,072}$

Une deuxième possibilité serait d'utiliser **Apply** et **Plus** (comme nous avions vu précédemment):

In[109]:=
```
Apply[Plus, suite]
remp··· plus
```

Out[109]= $\frac{14\,680\,065}{131\,072}$

In[110]:=
```
Clear[succ, suite]
efface
```

Exercices

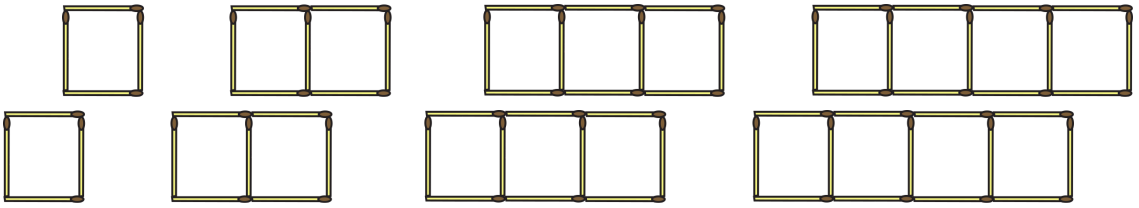
Exercice 6-1

Pour chacune des suites suivantes, déterminez la relation de récurrence ainsi que le terme général. Déterminez ensuite les 15 premiers termes en utilisant `Table` et en utilisant `NestList` et calculez la limite de chacune des deux suites.

1. La suite $(a_n)_{n\in\mathbb{N}}$ décrit le nombre d'allumettes nécessaires à la construction de chacune des figures de la séquence ci-dessous:

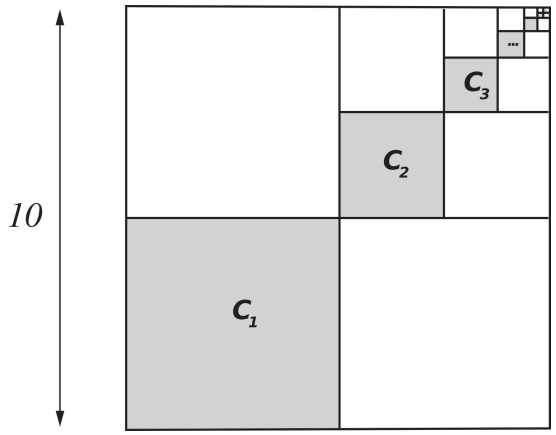
In[111]:=

Out[111]=

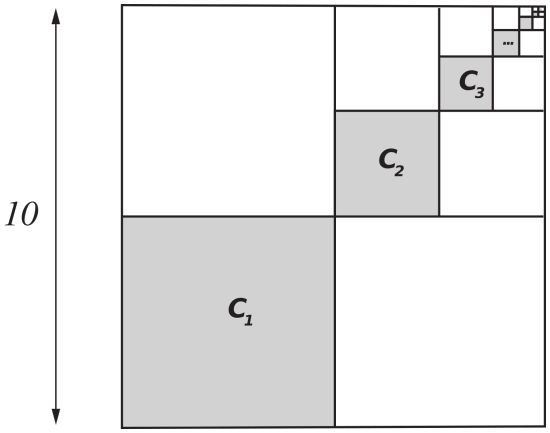


2. La suite $(c_n)_{n\in\mathbb{N}}$ décrit la mesure des côtés des carrés grisés C_n représentés ci-dessous:

In[112]:=



Out[112]=



Exercice 6-2

Soit la suite $(a_n)_{n\in\mathbb{N}}$ définie de la façon suivante:

$$a_0 = 1, \qquad a_{n+1} = \frac{2 + 2\,a_n^3}{3\,a_n^2}$$

1. Calculez a_{10} avec une précision de 15 chiffres significatifs.
2. Représentez graphiquement les 10 premiers termes de cette suite.
3. Déterminez une estimation numérique de la limite de cette suite.
4. Définissez un module `a[n_]` donnant la valeur exacte de a_n.

Exercice 6-3

Soit la suite $(a_n)_{n\in\mathbb{N}^*}$ définie de la façon suivante :

$$a_1 = 1, \qquad a_n = \frac{1}{1 + a_{n-1}}$$

(Attention: cette fois-ci l'indice du premier terme est 1 et non 0 !)

1. Définissez un module `a[n_]` donnant la valeur exacte de a_n.
2. Représentez graphiquement les 10 premiers termes de cette suite (graduer l'axe des abscisses toutes les unités). Que peut-on supposer à partir de ce graphe?
3. Déterminez une estimation numérique de la limite de cette suite.
4. Calculer la valeur exacte de la limite de cette suite.

Exercice 6-4

On considère la suite $(a_n)_{n\in\mathbb{N}^*}$ définie de la façon suivante:

$$a_1 = 150, \qquad a_{n+1} = \frac{2}{3} \cdot a_n.$$

1. Calculez les 15 premiers termes de la suite $(a_n)_{n\in\mathbb{N}^*}$ en code à virgule.
2. Déterminez le premier terme de la suite qui est inférieur à 0.1 à l'aide de la fonction **`NestWhile`**.
3. Déterminez le terme général de la suite, puis, en utilisant le terme général, déterminez le premier terme de la suite qui est inférieur à 0.1 d'abord sans Mathematica et ensuite avec Mathematica.

Exercice 6-5

On considère la somme suivante:

$$\frac{4}{3} + \frac{4}{9} + \frac{4}{27} + \ldots + \frac{4}{59049}.$$

1. Calculer cette somme à l'aide du terme général et à l'aide de la relation de récurrence.
2. Calculer la série vers laquelle converge cette somme lorsque le nombre de termes tend vers l'infini.

Exercice 6-6

On considère la somme suivante:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots - \frac{1}{99}.$$

1. Approximez la valeur de cette somme.
2. Calculer la série vers laquelle converge cette somme lorsque le nombre de termes tend vers l'infini.

Exercice 6-7

Un jardinier doit arroser 32 arbres alignés et distants les uns des autres de 8 m. Chaque arbre doit recevoir un arrosoir d'eau. Quelle distance le jardinier parcourra-t-il pour réaliser sa tâche s'il n'a qu'un arrosoir à disposition et que son périple commence et se termine au bassin d'eau se trouvant à 5 m du premier arbre?

Résolvez le problème de deux manières différentes: d'une part à l'aide d'une relation de récurrence, d'autre part à l'aide du terme général.

Exercice 6-8

1. Soit la suite $(a_n)_{n\in\mathbb{N}^*}$ définie de la façon suivante :

$$a_1 = 7, \quad a_2 = 10, \quad a_n = a_{n-1} + a_{n-2} \text{ pour } n > 2.$$

1.1. Définissez un module `a[n_]` donnant la valeur exacte de a_n. Pour ceci, utilisez la fonction `next[{a_,b_}]:={b,a+b}` et `Nest`.

1.2. Construisez la liste **aa** contenant les 30 premiers termes de la suite $(a_n)_{n\in\mathbb{N}^*}$.

1.3. On définit une nouvelle suite $(b_n)_{n\in\mathbb{N}^*}$ par $b_n = \frac{a_{n+1}}{a_n}$.

A partir de la liste **aa**, construisez la liste **bb** contenant les 29 premiers termes de la suite $(b_n)_{n\in\mathbb{N}^*}$ en code à virgule. Que peut-on supposer à partir de cette liste?

1.4. Rappel: le nombre d'or ϕ est défini par $\phi = \frac{1+\sqrt{5}}{2}$. Affichez ce nombre en code à virgule. Que pouvez-vous constater?

1.5. Que se passe-t-il si vous changez les valeurs initiales a_1 et a_2?

2. Pour $x, y \in \mathbb{R}$ on définit la suite $(u_n)_{n\in\mathbb{N}^*}$ par $u_n = x\cdot\phi^n + y\cdot\left(\frac{-1}{\phi}\right)^n$, où $\phi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or.

2.1. Montrez à l'aide de Mathematica que la suite $(u_n)_{n\in\mathbb{N}^*}$ est *une* suite de Fibonacci, c'est-à-dire que tout terme est la somme de ses deux prédécesseurs.

2.2. Soit c_1 et c_2 deux nombres réels. Montrez à l'aide de Mathematica qu'il existe deux nombres x et y tels que $u_1 = c_1$ et $u_2 = c_2$. Que peut-on déduire de cet énoncé?

2.3. Soit la suite $(p_n)_{n\in\mathbb{N}^*}$ définie par $p_n = \frac{u_{n+1}}{u_n}$. Calculez $\lim_{n\to+\infty} p_n$ à l'aide de Mathematica, puis démontrez votre résultat. Que peut-on déduire de cet énoncé?