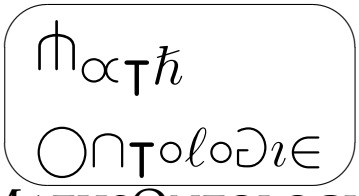

MATHSONTOLOGIE
UNE ONTOLOGIE POUR LES MATHS

MANUEL DE RÉFÉRENCE

Alain Busser
IREM de la Réunion
2013

Chapitre 1

Introduction

1.1 C'est quoi MathsOntologie ?

1.1.1 Ontologie

En informatique, une ontologie est définie comme «un ensemble structuré de concepts permettant de donner un sens aux informations». Les concepts de MathsOntologie sont des concepts mathématiques, et leur structure est définie par la notion d'héritage dans le langage Smalltalk qui est un langage objet [1]. En bref, une ontologie est une sorte de langage de programmation incomplet.

1.1.2 Mathématiques

MathsOntologie est dédié aux mathématiques, et ses objets sont créés pour aider à résoudre des problèmes de mathématiques.

1.1.3 En français

MathsOntologie est avant tout une traduction française partielle de Smalltalk, et se veut un ascenseur pour mettre le pied à l'étrier du «vrai» Smalltalk. Il est d'ailleurs possible de programmer en Smalltalk depuis MathsOntologie, sans aucune restriction.

1. et même plus ou moins le premier langage objet, créé par Alan Kay et son équipe au début des années 1970.

1.2 Que peut-on faire avec MathsOntologie ?

Avec MathsOntologie on peut faire des maths, mais aussi de l'algorithmique :

1.2.1 Programmation en MathsOntologie

Avec des tests

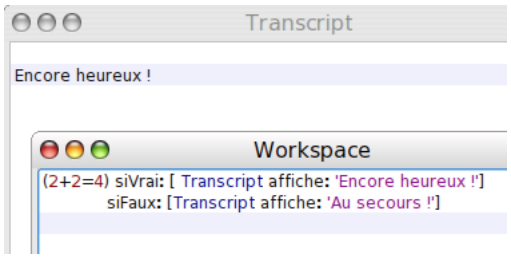

FIGURE 1.1 – on apprend que 2+2 est bien égal à 4

Avec des boucles à nombre prédéterminé d'exécutions

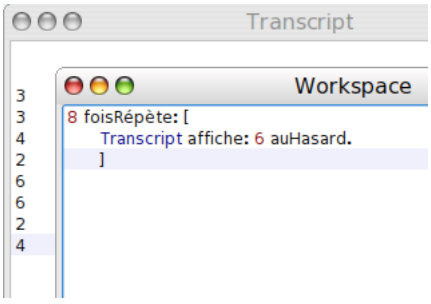

FIGURE 1.2 – Lancer un dé 8 fois

Avec des boucles à condition de sortie

Par exemple, on voudrait savoir combien de fois il faut lancer un dé pour que la probabilité d'avoir un 6 dépasse 0,99 ; autrement dit, que la probabilité de ne ***jamais*** avoir un 6 passe en -dessous de 0,01 :

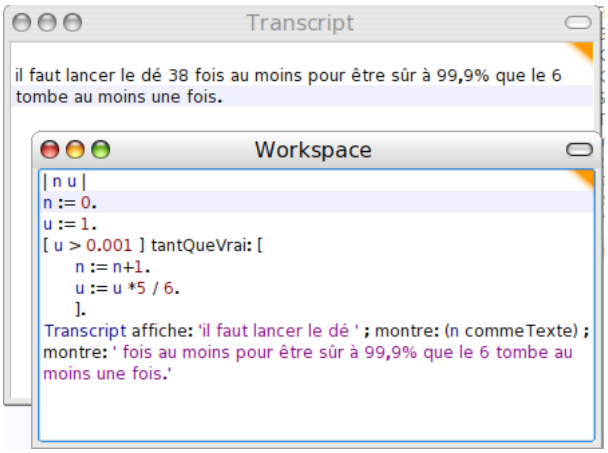

FIGURE 1.3 – Pas de chance au jeu des petits chevaux

1.2.2 Des maths avec MathsOntologie

Tableau de valeurs d'une fonction

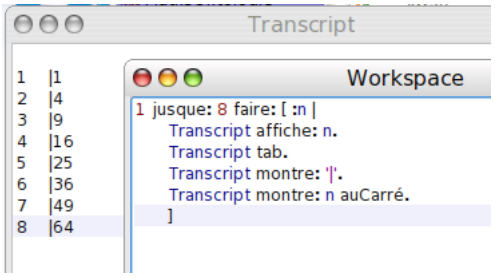

FIGURE 1.4 – Table des carrés

Affichage des nombres en toutes lettres

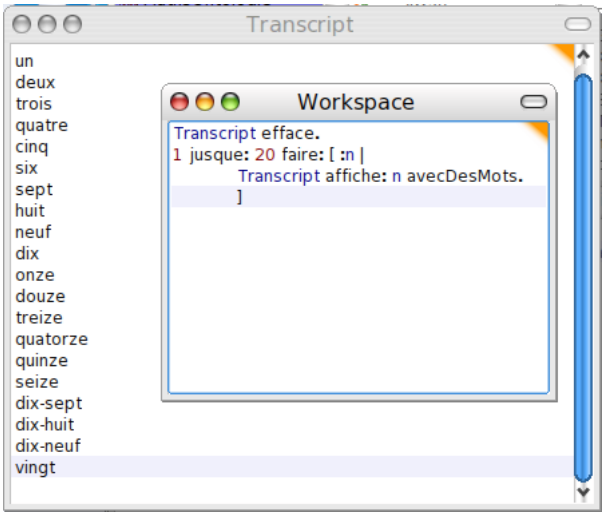

FIGURE 1.5 – MathsOntologie est fort en orthographe

et même en chiffres romains

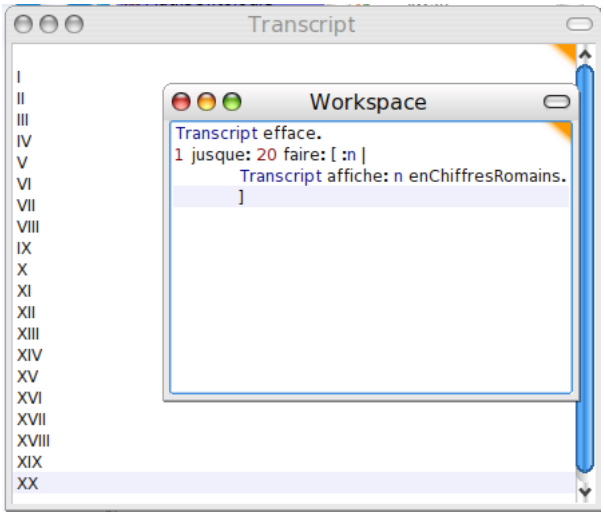

FIGURE 1.6 – Par Toutatis, MathsOntologix sait même faire ça !

Des statistiques

Statistique sur la liste des 96 diviseurs de 40320 :

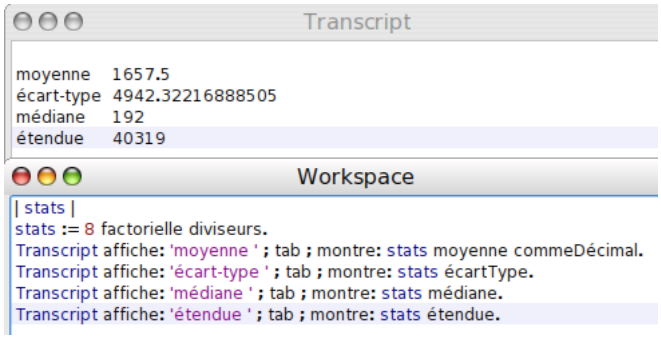

FIGURE 1.7 – arithmétique et statistiques en même temps

Des probabilités

Comptage par boucle (combien de fois le 8 apparaît) :

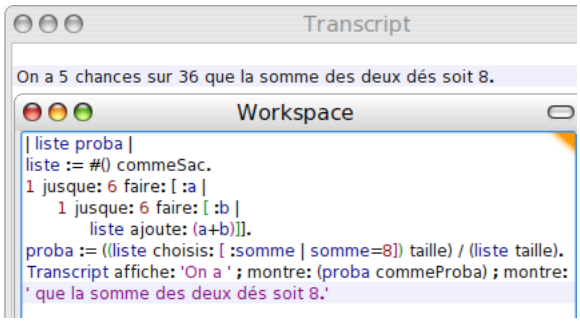

FIGURE 1.8 – Calcul de la probabilité d'avoir un 8 en lançant 2 dés

Chapitre 2

Démarrage

2.1 Lancer Pharo

2.1.1 mais sans lance-pierre !

Le logiciel Pharo est fourni "tout-en-un" : Les fichiers importants ou utiles sont les mêmes sous Windows, Linux ou Mac. Seul le minimum est différent selon le système d'exploitation :

- Sous Windows c'est `MathsOntologie.exe` sur lequel il convient de double-cliquer.
- Sous Linux c'est `MathsOntologie.sh` sur lequel il convient de double-cliquer [1] ;
- Sous MacOS il faut glisser l'application vers le dossier des applications.

2.1.2 Description de Pharo

Pharo ressemble un peu à un système d'exploitation, avec une sorte de "bureau" sur lequel s'ouvrent des fenêtres au fur et à mesure qu'on lance des outils. Les fenêtres cachées sont représentées par des miniatures en bas de l'écran ; il suffit d'approcher la souris de l'une de ces miniatures pour rouvrir la fenêtre voulue. Pour lancer un outil il suffit de faire un clic sur une partie du bureau qui n'est pas cachée par une fenêtre. Ce qui fait apparaître un menu déroulant, au sein duquel il ne reste plus qu'à choisir l'outil dont on a besoin. Pour MathsOntologie, on a besoin essentiellement de deux de ces outils :

1. ou lancer depuis une console si on préfère

2.2 Le workspace

Le workspace est l'endroit où on écrit son programme. Normalement il est présent au démarrage de MathsOntologie, mais on peut en ouvrir un en cliquant sur le bureau et en choisissant "Workspace" :

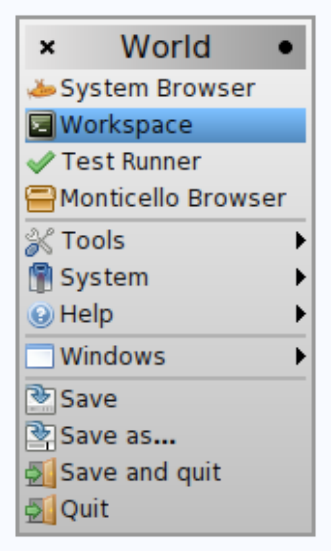

FIGURE 2.1 – En cas de perte de workspace, s'adresser au bureau

2.3 Le transcript

Le transcript est l'endroit où les données produites par un programme sont préférentiellement affichées. Normalement, il est ouvert au démarrage de MathsOntologie, mais au besoin, on peut le rouvrir en cliquant sur une partie vide du bureau, et en choisissant le transcript parmi les outils ("tools") :

Pour tester la configuration on peut essayer d'écrire ceci dans le workspace (histoire de calculer une hypoténuse) :

```
Transcript affiche: (5 auCarré + 12 auCarré) racine.
```

Pour lancer ce programme en MathsOntologie, on le sélectionne, soit avec la souris, soit en faisant `Alt-A`, et on clique droit, ce qui permet de choisir ce qu'on veut faire avec ce programme :

– l'exécuter : "Do it" (ou `Alt-D`)

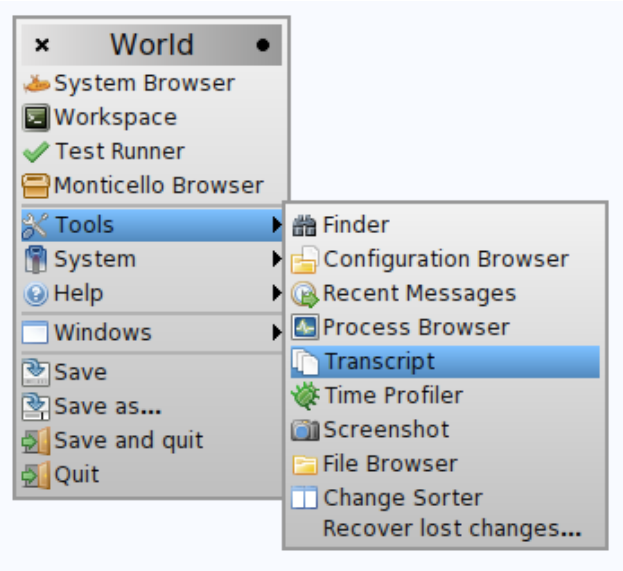

FIGURE 2.2 – La boîte à outil de MathsOntologie est bien fournie

– l'afficher dans le workspace : "Print it" (ou `Alt-P`)
– l'inspecter sous toutes les coutures : "Inspect it" (ou `Alt-I`)
– l'explorer : "Explore it" (ou `Alt-L`)

Dans le cas présent, "Do it" suffit à faire afficher sur le transcript la valeur de $\sqrt{5^2 + 12^2}$.

2.3.1 Puissance du transcript

La méthode `affiche` va à la ligne ce qui permet d'écrire en colonnes. La méthode `montre` fait de même sauf qu'elle ne va pas à la ligne ; elle permet donc d'écrire plusieurs choses sur une même ligne. La méthode `cr`, au contraire, va à la ligne sans rien écrire. La méthode `tab` va à la tabulation suivante ; elle est donc utile pour écrire des tableaux. Enfin on peut effacer le tableau plein de craie avec

```
Transcript efface.
```

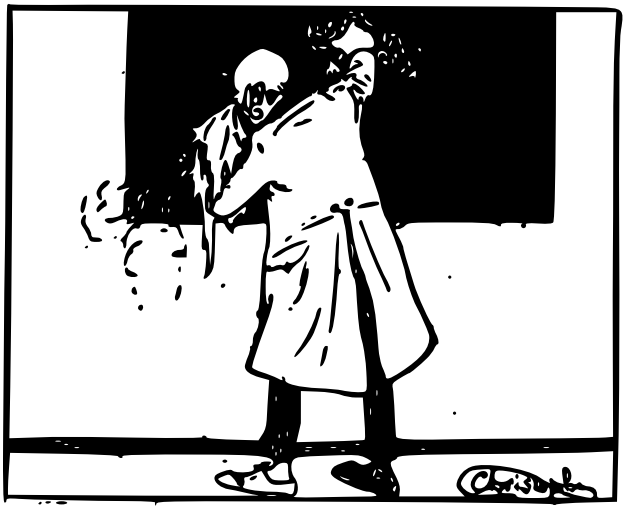

FIGURE 2.3 – En effaçant le transcript on ne fait pas de nuage de craie

2.4 Se passer du transcript

2.4.1 Afficher

En choisissant "Print it", on affiche la valeur produite par le programme (ce qui suppose qu'il en produise une, et une seule) ; par exemple, si on veut savoir si le triangle de côtés 5, 12 et 13 est rectangle, on peut entrer dans le workspace :

```
5 auCarré + 12 auCarré = 13 auCarré.
```

puis cliquer-droit :

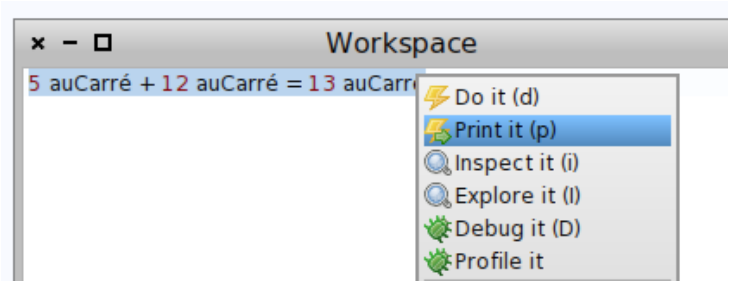

FIGURE 2.4 – J'ai comme l'impression que ça va imprimer

La réponse `true` apparaît alors collée au programme, ce qui est peu lisible. Alors on peut ajouter des lignes vides en-dessous du programme, ou inspecter au lieu d'afficher :

2.4.2 Inspecter

Si, au lieu de choisir "Print it", on choisit "Inspect it", une nouvelle fenêtre s'ouvre, dans laquelle se trouve non seulement la réponse elle-même mais aussi une fiche complète sur la réponse :

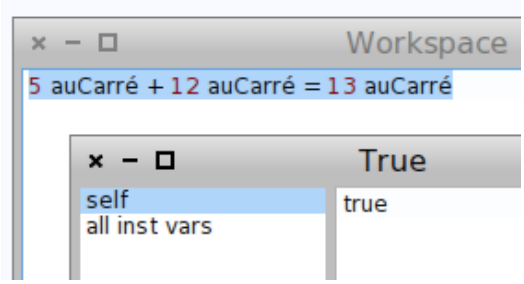

FIGURE 2.5 – La police de l'inspecteur n'a pas de serif

2.5 Quitter Pharo

Lorsqu'on quitte le logiciel, on a le choix entre
- laisser le logiciel dans l'état où on l'a trouvé en entrant : Tout ce qu'on a fait durant la séance disparaît alors dans les limbes de l'oubli ;
- enregistrer l'«image» dans son état actuel, avec le souvenir de la séance, et le transcript et le workplace pleins d'affichage. Toutefois, en quittant Pharo, on enregistre un message de départ sur le transcript.

Chapitre 3

Problèmes de trains

> Un train quitte la gare de Citypolis à 2 heures 53 minutes pour la ville de Policité, située à 245 km de Citypolis. Sachant que l'arrivée à Policité se fait à 5 heures 14 minutes, quelle est la vitesse moyenne du train ?

3.1 temps

3.1.1 Mois et année

Pour savoir en quel mois on est, on fait [1] :

```
Transcript affiche: Time now commeDate nomDuMois.
```

Pour savoir en quelle année on est, on fait :

```
Transcript affiche: Time now commeDate année.
```

Pour savoir si l'année courante est bissextile :

```
Transcript affiche: Time now commeDate année estBissextile.
```

3.1.2 Jour

Pour connaître le numéro d'aujourd'hui dans le mois, on fait :

1. `Time now` est une méthode de l'objet `Time`, qui renvoie l'information stockée dans l'horloge de l'ordinateur. `commeDate` transforme ces informations en objet `Date`.

```
Transcript affiche: Time now commeDate quantième.
```

Pour connaître le numéro du jour dans l'année, on fait :

```
Transcript affiche: Time now commeDate jourDansLannée.
```

Pour connaître le nom du jour d'hui, on fait :

```
Transcript affiche: Time now commeDate nomDuJour.
```

Enfin, pour avoir un calendrier, on fait :

```
Transcript affiche: Time now commeDate aujourdHui.
```

3.2 Durées

3.2.1 Jours

```
 Un robot martien parcourt un triangle
équilatéral; sachant qu'il lui faut 20 heures
pour parcourir un côté, combien mettra-t-il
pour parcourir tout le périmètre ?
```

Il suffit de multiplier 20 heures par 3 :

```
Transcript affiche: ((3*20) heures).
```

La réponse `2:12:00:00` se lit «deux jours et douze heures» : Ce robot est nettement plus lent que le train qui a quitté Citypolis il y a trois quarts-d'heures déjà. Mais combien de minutes cela fait-il, trois quarts-d'heures au juste ?

3.2.2 heures

```
Transcript affiche: (3/4) heures.
```

La réponse `0:00:45:00` se lit «quarante-cinq minutes». Pour la vérifier, on peut faire

```
Transcript affiche: 45 avecDesMots.
```

Pour la conversion dans l'autre sens, on peut faire

```
Transcript affiche: 45 minutes enHeures.
```

MathOntologie sait aussi convertir `enSecondes`, `enMinutes`, ou `enJours`, et arrondir ou tronquer à une durée, par exemple à 10 minutes près[2].

3.3 Solution du problème

3.3.1 Distance

Pour rappel, la distance parcourue est, en kilomètres, 245. On va donc diviser ce nombre par la durée en heures pour avoir la vitesse en kilomètres par heures. Reste donc à calculer la durée.

3.3.2 Durée

Pour calculer la durée du trajet, il suffit de soustraire l'heure de départ de l'heure d'arrivée :

```
Transcript affiche: ((5 heures + 14 minutes) − (2 heures + 53 minutes)).
```

Il subsiste un problème : *MathOntologie* ne sait pas diviser par `0:02:21:00` parce que `2 heures 21 minutes` n'est pas un nombre, et que *MathOntologie* ne sait diviser que par des nombres. C'est pourquoi il faut encore convertir cette durée en un nombre d'heures, avec `enHeures` :

3.3.3 Calcul de la vitesse

Finalement :

```
| d t |
d := 235.
t := (2 heures + 21 minutes) enHeures.
Transcript affiche: 'La vitesse moyenne est '.
Transcript montre: (d/t).
```

2. avec `arrondiA:` ou `tronquéA:` respectivement

Chapitre 4

Logique

4.1 Booléens

4.1.1 Exemple

Si on entre

```
Transcript affiche: 2+2=4.
```

l'affichage sur le transcript dit `True`. Donc, dans la langue de George Boole, «vrai» se dit "true". De même, «faux» se dit "false".

4.1.2 Opérations booléennes

4.1.3 négation

Le contraire de la vérité est quelque chose de faux :

```
Transcript affiche: (2+2=4) contraire.
Transcript affiche: (2+2=4) contraire contraire.
```

De façon similaire, le contraire de quelque chose de faux, est vrai :

```
Transcript affiche: (2+2=5) contraire.
```

4.1.4 Conjonction

Une conjonction n'est vraie que si tous ses constituants le sont. Ainsi, bien que 2+2 soit égal à 4, «2+2=4 et -2 est positif» est une proposition fausse puisque son second constituant est faux :

```
Transcript affiche: ((2+2=4) et: (−2>0)).
```

4.1.5 Disjonction

Par contre, une disjonction est vraie dès que l'un au moins de ses constituants l'est :

```
Transcript affiche: (2+2=4) ou: (−2>0).
```

4.1.6 Tiers exclu

Avec *MathOntologie*, on peut vérifier que la disjonction entre une proposition et son contraire, est toujours vraie :

```
| a b c |
10 foisRépète: [
   a := 9 auHasard.
   b := 9 auHasard.
   c := 18 auHasard.
   Transcript affiche: 'Ou bien ',(a commeTexte,'+',(b commeTexte),' est égal à ',(c
     commeTexte),', '.
   Transcript montre: 'ou bien ',(a commeTexte),'+',(b commeTexte),' est différent de ',(
     c commeTexte),': '.
   Transcript montre: ((a+b=c) ou: (a+b∼=c)).
   ].
```

4.2 Tests

En *MathOntologie*, les tests se rédigent avec `siVrai`, `siFaux` et leurs combinaisons.

4.2.1 Tests simples

```
((6 auHasard)=6) siVrai: [
   Transcript affiche: 'gagné'.
      ].
```

Dans la plupart des cas, il ne se passe rien, puisque le dé ne tombe pas sur le 6.

```
((6 auHasard)=6) siFaux: [
   Transcript affiche: 'essayez une autre fois'.
      ].
```

Ici c'est le contraire, il ne se passe quelque chose que si on perd (c'est-à-dire dans la plupart des cas).

4.2.2 Tests multiples

```
((6 auHasard)=6) siVrai: [
   Transcript affiche: 'gagné'.
      ] siFaux: [
      Transcript affiche: 'perdu'.].
```

Chapitre 5

Algèbre et fonctions

5.1 Opérations

5.1.1 priorités opératoires

Les opérations se notent respectivement $+$, $-$, $*$ et $/$. Mais en *MathOntologie*, l'addition n'est pas nécessairement prioritaire sur la multiplication : Les opérations se font de gauche à droite. Ainsi,

```
Transcript affiche: 2 + 3 * 4.
```

donne $20 = (2 + 3) \times 4$ et non $12 = 2 + (3 \times 4)$: *MathOntologie* est nul en algèbre ! Cela est la conséquence du fait que le message qui vient en premier est "`+ 3`", envoyé au nombre 2, ce qui a pour effet d'effectuer l'addition, avant que parvienne le message suivant (`* 4`) au résultat. Et la multiplication est effectuée en dernier, simplement parce qu'elle arrive en dernier.

Une remédiation est proposée ci-dessous, avec l'algorithmique.

5.1.2 puissances

En *MathOntologie*, l'élévation à la puissance ne se note **pas** comme avec les outils classiques (chapeau ou double astérisque) mais s'écrit en toutes lettres :

```
Transcript affiche: (2 puissance: 4).
```

Lorsque l'exposant est petit, on peut utiliser une abréviation :

1. Si l'exposant est 3 :

   ```
   Transcript affiche: 2 auCube.
   ```

2. Si l'exposant est 2 :

```
Transcript affiche: 2 auCarré.
```

3. Si l'exposant est -1 :

```
Transcript affiche: 2 inverse.
```

Un exemple : La réciproque du théorème de Pythagore :

```
| a b c |
a := 5.
b := 12.
c := 13.
(a auCarré + b auCarré = (c auCarré))
   siVrai: [
      Transcript affiche: 'Le triangle est rectangle.'.
      ].
   siFaux: [
      Transcript affiche: 'Le triangle est quelconque.'.
      ].
```

5.2 Fonctions

Pour appliquer une fonction à un nombre, on écrit le nom de la fonction après le nombre. Les exemples des fonctions *carré*, *cube* et *inverse* ont été vus ci-dessus.

5.2.1 Racine carrée

Pour calculer la racine de 2, il suffit de le lui demander :

```
Transcript affiche: 2 racine.
```

5.2.2 Fonctions trigonométriques

Pour calculer les sinus, cosinus et tangente d'un angle en degrés, on écrit

```
Transcript affiche: 30 sinus.
Transcript affiche: 30 cosinus.
Transcript affiche: 30 tangente.
```

D'autres fonctions sont décrites dans le chapitre sur les décimaux.

5.2.3 Fonctions définies par l'utilisateur

Pour *MathOntologie*, une fonction est un *bloc* noté entre crochets. Le début du bloc est la liste des variables, dont le nom est précédé d'un double-point, et un trait vertical pour séparer cette définition du corps de la fonction, lequel est juste un programme de Smalltalk (ou de *MathOntologie*) :

```
| f |
f := [ :x | x − (x auCarré)].
Transcript affiche: (f valeur: 0.75).
```

Pour appliquer une fonction, on lui envoie le message `valeur` suivi du nombre auquel on veut appliquer la fonction. Une fonction peut très bien avoir plusieurs variables :

```
| hypoténuse |
hypoténuse := [ :a :b | (a auCarré + b auCarré) racine].
```

5.2.4 Représentation graphique

MathOntologie n'est pas un grapheur, mais il est possible d'avoir l'allure de la représentation graphique d'une fonction sur un intervalle en entrant `dessineDe: xmin jusque: xmax` (`xmin` et `xmax` étant les bornes de l'intervalle).

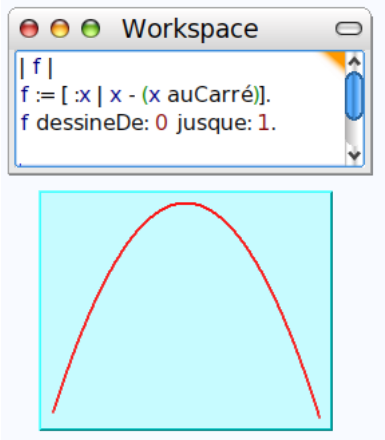

FIGURE 5.1 – la parabole de la part à Bowl

Une représentation graphique peut être déplacée à la souris, et elle est translucide [1]. Pour s'en débarasser, il faut faire un `Alt-Shift-Clic` dessus, ce qui fait apparaître un "halo" avec, notamment, un bouton de fermeture.

5.3 Affectations successives

MathOntologie est peut-être nul en algèbre, mais il est fort en algorithmique. Ainsi, alors que

```
| x |
x := 2 + 3 * 4.
Transcript affiche: x.
```

donne un résultat faux, on peut corriger avec

```
| x y |
x := 2.
y := 3 * 4.
Transcript affiche: x + y.
```

L'affectation se note donc " : =" en *MathOntologie* [2].

5.4 Étude de cas : Sujet Brevet Polynésie septembre 2012

Sujet :

1. Alors que l'auteur de ces lignes est à la fois transcendant et lucide, mais pas translucide...
2. Une réminiscence du langage de programmation *Algol*, que *Squeak*, le Smalltalk du *MIT*, représente automatiquement par une flèche : ←.

On donne le programme de calcul suivant :

- Choisir un nombre.
- Lui ajouter 1.
- Calculer le carré de cette somme.
- Enlever 16 au résultat obtenu.

1. Vérifier que, lorsque le nombre de départ est 4, on obtient comme résultat 9.
2. Lorsque le nombre de départ est (- 1), quel résultat obtient-on ?
3. Le nombre de départ étant x, exprimer le résultat final en fonction de x.

5.4.1 Solution par affectations

1.
```
| x |
x := 4.
x := x + 1.
x := x auCarré.
x := x − 16.
Transcript affiche: x.
```

Ce script répond à la question mais n'explique pas pourquoi la réponse est 9. Pour en savoir plus, il suffit d'insérer des affichages dans le transcript :

```
| x |
x := 4.
Transcript affiche: 'À ce stade, x vaut ' ; montre: x.
x := x + 1.
Transcript affiche: 'À ce stade, x vaut ' ; montre: x.
x := x auCarré.
Transcript affiche: 'À ce stade, x vaut ' ; montre: x.
x := x − 16.
Transcript affiche: 'Finalement, x vaut ' ; montre: x.
```

2.

3. *MathOntologie* ne connaissant pas le calcul formel[3], il faut faire la question à la main : $(x+1)^2 - 16 = x^2 + 2x + 1 - 16 = x^2 + 2x - 15$.

3. Décidément l'algèbre n'est pas son fort !

5.4.2 Avec des fonctions

Une fonction peut s'implémenter en *MathOntologie* par un *bloc* du langage Smalltalk. Les variables d'entrée sont précédées d'un double-point [4] et leur liste est séparée du bloc proprement dit par un trait vertical [5]. La dernière variable citée est renvoyée par le bloc, lorsqu'il reçoit le message `valeur:`. Dans le cas présent, comme il est hors de question de modifier l'antécédent de la fonction, on doit créer une variable provisoire notée `t` [6] qui sera affectée au cours de l'exécution du bloc. Le bloc `programme` est lui-même une variable, citée en haut du script, et affectée en première ligne de celui-ci :

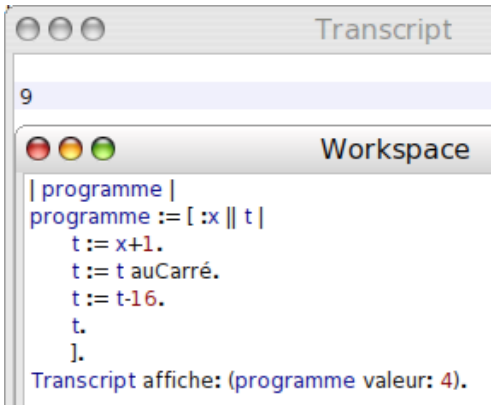

FIGURE 5.2 – Les programmes de calcul sont aussi des objets *MathOntologie*

4. Pour expliquer à *MathOntologie* qu'il ne faut pas les calculer tout de suite...
5. Sur PC, appuyer sur `AltGr` et sur le 6 du haut du clavier
6. comme «temporaire»

On peut vérifier expérimentalement le développement de la question 3. En effet, $x^2 + 2x - 15$ est une autre fonction, dont on peut vérifier qu'elle retourne les mêmes valeurs que la fonction *programme* :

```
| programme f e |
programme := [ :x || t |
    t := x+1.
    t := t auCarré.
    t := t − 16.
    t.
    ].
f := [ :x | x auCarré + (2 * x) − 15].
10 foisRépète: [
    e := 1000 auHasard.
    Transcript affiche: (programme valeur: e = (f valeur: e)).
    ].
```

5.4.3 Comparaison des deux algorithmes

L'algorithme *programme* utilise deux additions (une par 1, l'autre par -16) et une multiplication (une élévation au carré pour être précis). Alors que l'algorithme f utilise deux multiplications (l'une par x, l'autre par 2) et deux additions (l'une par $2x$, l'autre par -15). On s'attend donc à ce que, malgré son apparente simplicité, l'algorithme f prenne plus de temps que le programme de calcul initial. C'est apparemment bien le cas :

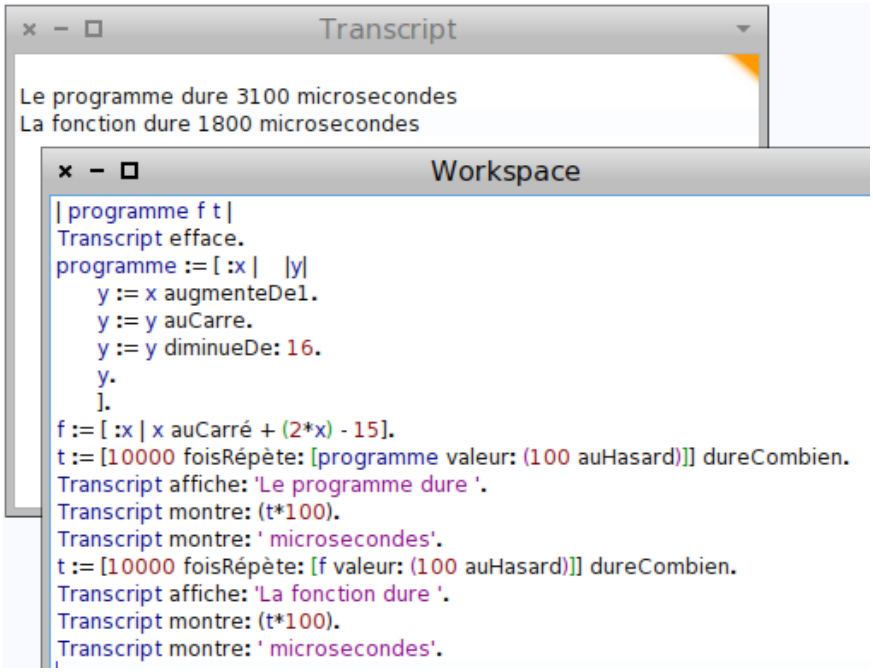

FIGURE 5.3 – Pour demander à un bloc combien de millisecondes il lui faut pour s'exécuter, on lui demande *dureCombien*. Pour mesurer ce temps, on peut exécuter le bloc 10000 fois et diviser le résultat par 10000.

Chapitre 6

Suites et algorithmique

6.1 Boucles à nombre prédéterminé d'exécutions

Pour boucler, on utilise en général une variable spéciale appelée «indice», et qui est destinée à être incrémentée ou décrémentée. Bref, une variable qui sert à compter, et c'est compliqué. Mais pas toujours :

6.1.1 Boucles simples

Pour tourner la langue 7 fois dans la bouche, on peut faire

```
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
Transcript affiche: 'Je tourne la langue dans la bouche'.
```

Mais aussi :

```
7 foisRépète: [Transcript affiche: 'Je tourne la langue dans la bouche'].
```

ce qui est plus simple à programmer [1]. Cette répétition sans avoir à compter soi-même est particulièrement utile en probabilités ; par exemple, pour lancer 8 fois un dé, on répète 8 fois l'action de lancer le dé :

```
8 foisRépète: [Transcript affiche: 6 auHasard].
```

1. Ce genre de boucle a été introduite par Seymour Papert dans le langage *LOGO* au début des années 1960.

6.1.2 Boucles à indice

```
 Calculer la somme des 100 premiers nombres
entiers non nuls:
1+2+3+...+99+100
```

Pour additionner toutes les valeurs de l'indice, on utilise une variable `somme`, initialisée à zéro, et qui va accumuler toutes les valeurs successives de l'indice. Ces valeurs vont de 1 à 100 par pas de 1 :

```
| somme |
somme := 0.
1 jusque: 100 faire: [ :indice |
   somme := somme + indice.
   ].
Transcript affiche: '1+2+3+...+100=' ; montre: somme.
```

Mais il y a d'autres façons de résoudre ce problème.

```
Transcript affiche: ([1 jusque: 100] injecte: 0 dans [ :i :s | s+i.]).
```

On injecte des doses successives de l'indice i dans la somme s. À la fin, on a la somme de tous les éléments de la liste `[1 jusque: 100]`. Plus simple (mais pas généralisable) :

```
Transcript affiche: ((1 jusque: 100) somme).
```

On peut aussi utiliser les boucles sans référence à l'indice, juste en explicitant l'incrémentation de l'indice :

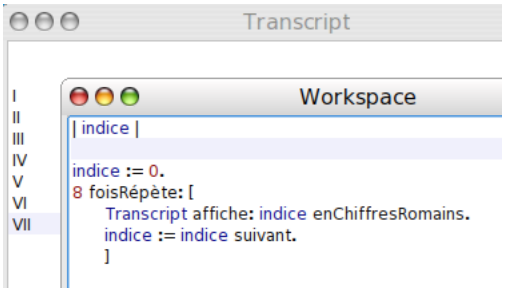

FIGURE 6.1 – On peut utiliser des boucles sans indice avec un indice

Les boucles à indice vont de 1 en 1 (incrémentation de l'indice). Si ce n'est pas ce qu'on veut, on peut préciser un *pas* autre que 1. Par exemple, pour transformer le transcript en une table de logarithmes, on peut faire

```
(1 jusque: 2 parPasDe: 0.1) fais: [ :x |
    Transcript affiche: (x commeFractionApprochée commeDécimal).
    Transcript tab.
    Transcript montre: '|'.
    Transcript montre: x log.
    ].
```

La suite de nombres ainsi calculés est une *suite arithmétique*. Son premier terme est le nombre de départ (ci-dessus, 1) et le pas de la suite s'appelle aussi sa *raison*.

6.1.3 Boucles sur ensembles

MathOntologie permet aussi de boucler sur des ensembles, par exemple des puissances de 10 :

```
#(10 100 1000 1000000) fais: [ :p |
  Transcript affiche: (p avecDesMots).
  ].
```

6.1.4 Représentation graphique d'une suite positive

Lorsque pour tout n, u_n est positif, on peut représenter graphiquement la suite avec `suite diagrammeBatonsTrié`[2]. Pour cela, la suite doit être créée comme un `Dictionnaire`, par `Dictionary new` :

2. Le nom de la méthode indique pourquoi elle a été créée uniquement pour des suites positives...

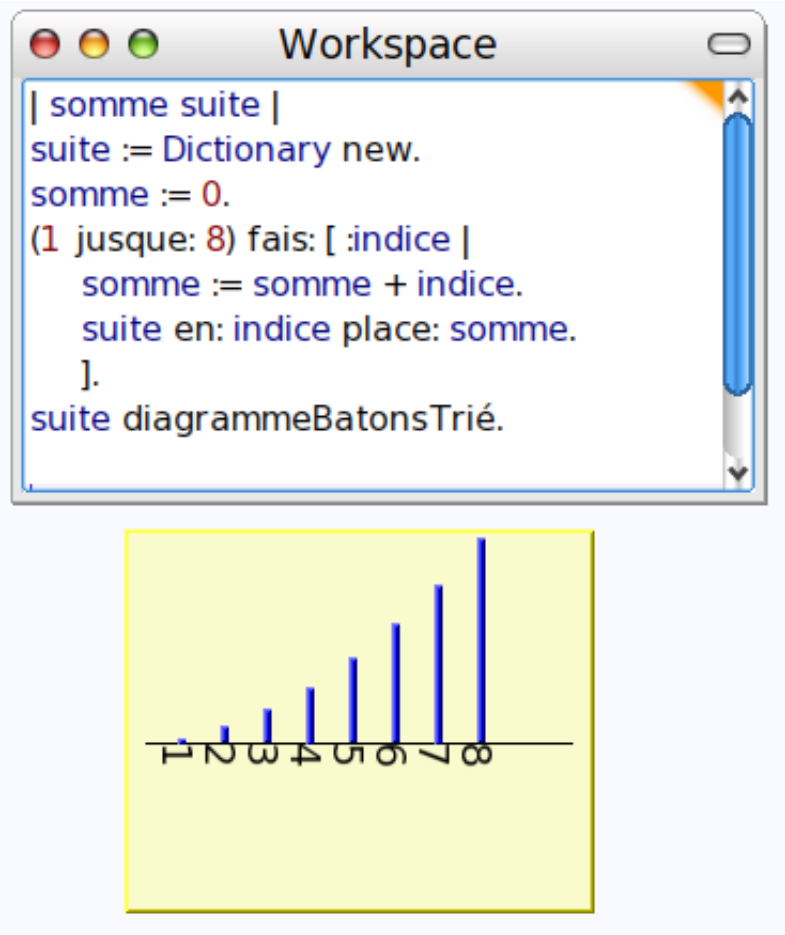

FIGURE 6.2 – On constate une nette tendance à l'augmentation lorsqu'on additionne des nombres positifs...

6.2 Boucles à condition de sortie

> On place 800 € à 2 % d'intérêts par an. Au bout de combien d'années le capital aura-t-il doublé?

Pour résoudre ce genre de problème, on doit boucler (c'est-à-dire incrémenter l'indice) jusqu'à ce que le capital ait doublé, ou, ce qui revient au même, tant que le capital n'a pas encore doublé. La condition de bouclage sera donc le fait que le capital est encore inférieur à 1600 €. Mais comme le capital ne cesse d'évoluer, il faut refaire le test à chaque passage dans la

boucle. Le test ne sera donc pas une proposition mais un *bloc* de Smalltalk, noté entre crochets :

```
| capital indice |
capital := 800.
indice := 0.
[ capital < 1600 ] tantQueVrai: [
   capital := capital augmentéDePourcents: 2.
   indice := indice suivant.
   ].
Transcript affiche: 'Il aura fallu '.
Transcript montre: indice.
Transcript montre: ' ans pour doubler le capital.'.
```

On peut remplacer `[capital<1600] tantQueVrai` par `[capital>1600] tantQueFaux`. Et les boucles à condition de sortie permettent aussi de calculer la somme 1+2+3+...+100 :

```
| somme indice |
somme := 0.
indice := 0.
[ indice <= 100] tantQueVrai: [
      somme :=  somme + indice.
      indice := indice suivant.
      ].
Transcript affiche: '1+2+3+...+100=' ; montre: somme
```

Ceci permet de trouver la prochaine année première :

```
| an |
an := 2013.
[ an estPremier ] tantQueFaux: [
   an := an suivant.]
Transcript affiche: an ; montre: ' sera une année première.'.
```

Chapitre 7

Arithmétique

7.1 Affichage

7.1.1 En chiffres

Pour afficher un nombre entier dans le transcript, on peut faire

```
Transcript affiche: 2013.
```

Mais on peut aussi afficher le nombre 2013 en base 2 (par exemple) :

```
Transcript affiche: (2013 afficheEnBase: 2).
```

7.1.2 En lettres

En Français

Pour afficher 2013 en lettres dans le transcript, on peut faire

```
Transcript affiche: 2013 avecDesMots.
```

Ceci affiche `deux mille treize` dans le transcript.

En latin

MathOntologie peut aussi afficher en chiffres romains :

```
Transcript affiche: 2013 enChiffresRomains.
```

Ceci affiche `MMXIII` dans le transcript :

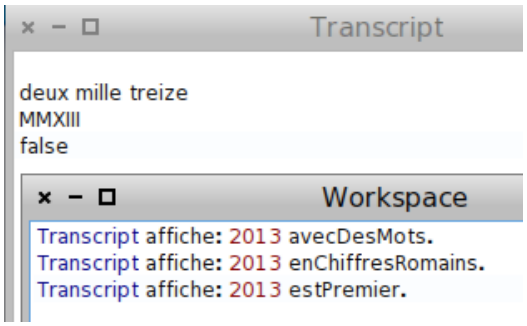

FIGURE 7.1 – affichages de 2013 dans le transcript

7.2 Opérations

7.2.1 Opérations élémentaires

Les opérations arithmétiques de base se notent $+$, $-$, $*$ et $/$ respectivement. Mais la division est exacte, et son résultat est parfois donné sous forme de fraction. Il y a aussi une division euclidienne, notée $//$ pour le quotient et $\backslash\backslash$ pour le reste. Pour le reste euclidien, on utilise plutôt `modulo:`. Par exemple, pour calculer le reste de 2013 dans la division euclidienne par 23, on peut faire

```
Transcript affiche: (2013 modulo: 23).
```

7.2.2 Diviseurs

Lorsque `n modulo: p` est nul, on dit que n est divisible par p. Le test se note `estDivisiblePar` en *MathOntologie* :

```
Transcript affiche: (2013 estDivisiblePar: 3).
```

affiche `true` parce que 2+0+1+3=6.
La liste des diviseurs d'un entier s'obtient par `diviseurs` :

```
Transcript affiche: 2013 diviseurs.
```

Comme c'est une liste[1], on peut faire des statistiques sur cette liste, comme le montre l'un des exemples du premier chapitre. On peut également obtenir la liste des diviseurs communs à deux nombres en envoyant le message `inter` à leurs listes de diviseurs :

1. en fait c'est un sac mais on peut quand même faire des statistiques dessus ; voir le chapitre sur les statistiques.

```
| l1 l2 |
l1 := 64 diviseurs.
l2 := 48 diviseurs.
Transcript affiche: (l1 inter: l2).
```

Le plus grand des éléments de cette liste commune est le *pgcd* :

```
Transcript affiche: (64 pgcd: 48).
```

De même, le plus petit des multiples communs à deux entiers s'obtient avec `ppcm` :

```
Transcript affiche: (64 ppcm: 48).
```

Lorsque leur pgcd est égal à 1, deux nombres sont dits *premiers entre eux* :

```
Transcript affiche: (2013 estPremierAvec: 23).
```

Si on note g le pgcd de deux nombres a et b, l'équation $au + bv = g$ admet une solution $(u; v)$:

```
Transcript affiche: (2013 BezoutAvec: 23).
```

Le diagnostic est assez complet : L'équation est affichée, ainsi que la solution trouvée par *MathOntologie*.

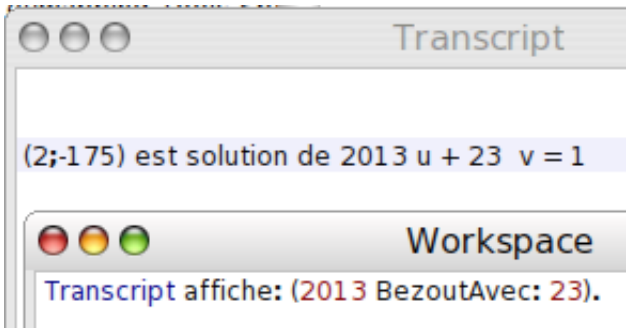

FIGURE 7.2 – Résolution de l'équation de Bezout

7.3 Nombres premiers

7.3.1 primalité d'un entier

Pour savoir si un entier est premier, il suffit de le lui demander :

```
Transcript affiche: 2013 estPremier.
```

Seulement la réponse est rédhibitoirement longue pour les grands entiers, alors dans ce cas on leur demande seulement si la probabilité qu'ils soient premiers est grande :

```
Transcript affiche: ((2 puissance: 127 − 1) estProbablementPremier).
```

7.3.2 Décomposition

La liste des facteurs premiers d'un nombre n s'obtient avec `facteursPremiers` :

```
Transcript affiche: 2013 facteursPremiers.
```

On peut donc calculer les quartiles, médiane, moyenne et écart-type de ces facteurs...

7.3.3 Corps finis

Si p est premier, l'ensemble des restes modulo p est muni d'une division comme les réels : Tout nombre non nul modulo p possède un inverse modulo p. Par exemple, 127 étant premier, les opérations entre 36 et 67 modulo 127 se font ainsi :

1. Addition :

   ```
   Transcript affiche: ((36 + 67) modulo: 127).
   ```

2. Soustraction :

   ```
   Transcript affiche: ((36 − 67) modulo: 127).
   ```

3. Multiplication :

   ```
   Transcript affiche: ((36 * 67) modulo: 127).
   ```

4. Division :

   ```
   Transcript affiche: (36 * (67 inverseModulo: 127 ) modulo: 127).
   ```

5. Puissance :

```
Transcript affiche: (36 puissance: 67 modulo: 127).
```

7.4 Une étude de cas : L'indicatrice d'Euler

.

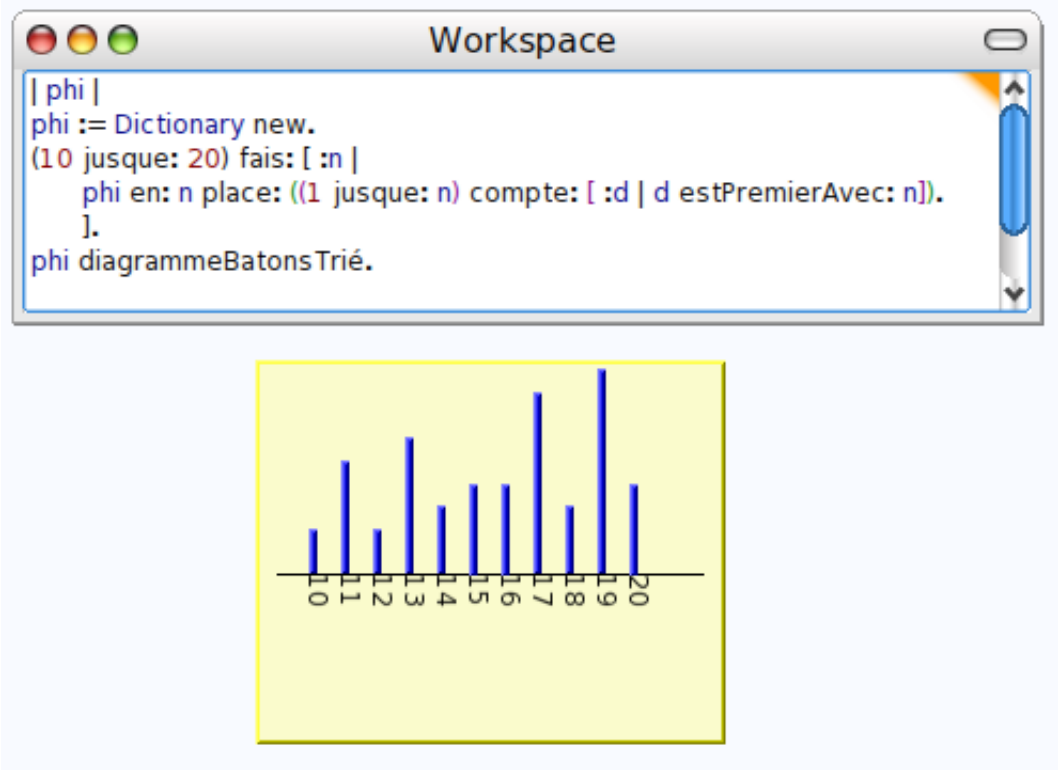

FIGURE 7.3 – De l'air avec Euler, l'air de rien...

7.4.1 Définition

Voici comment on peut retrouver expérimentalement l'indicatrice d'Euler, qui est le nombre de nombres premiers avec un nombre donné. Par exemple, l'indicatrice d'Euler de 40 est le nombre de nombres entre 1 et 40 qui sont premiers avec 40. Pour voir quels sont ces nombres on peut demander à *MathOntologie* d'extraire parmi tous les nombres entre 1 et 40, ceux qui sont premiers avec 40 :

```
Transcript affiche: ((1 jusque: 40) choisis: [ :d | d estPremierAvec: 40]).
```

Pour savoir combien ils sont, il suffit, au lieu de les choisir, de les compter :

```
Transcript affiche: ((1 jusque: 40) compte: [ :d | d estPremierAvec: 40]).
```

On apprend ainsi que parmi les nombres de 1 à 40, il y en a 16 qui sont premiers avec 40.

7.4.2 Calcul

Euler prétend que son indicatrice est le produit de 40 par le produit de tous les $\left(1-\frac{1}{p}\right)$ pour tous les diviseurs premiers de n. La vérification de l'égalité est assez rapide avec *MathOntologie*.

1. Facteurs premiers

   ```
   Transcript affiche: 40 facteursPremiers.
   ```

   On constate que la liste obtenue ne fait pas l'affaire, le diviseur 2 apparaissant trois fois, donc deux fois de trop.

2. Diviseurs premiers

   ```
   Transcript affiche: 40 facteursPremiers commeEnsemble.
   ```

   Cette fois-ci, on a $\{2,5\}$ donc les deux diviseurs premiers de 40 sont bien affichés une fois chacun. On a maintenant un algorithme pour calculer l'indicatrice d'Euler :
   – initialiser le produit φ à 40 ;
   – pour chaque diviseur premier p de 40, multiplier φ par $\left(1-\frac{1}{p}\right)$;
   – La valeur finale de φ est l'indicatrice d'Euler de 40.
   La traduction de cet algorithme en *MathOntologie* donne :

   ```
   | phi |
   phi := 40.
   40 facteursPremiers commeEnsemble fais: [ :p | phi := phi * ( 1 − (1/p))].
   Transcript affiche: phi.
   ```

7.4.3 Étude expérimentale

On peut recommencer tout ça dans une boucle :

```
| phi |
10 jusque: 30 faire: [ :n |
   Transcript affiche: n.
   Transcript montre: ' : '.
   Transcript montre: ((1 jusque: n) compte: [ :d |
      d estPremierAvec: n]).
   Transcript montre: ' = '.
   phi := n.
   n facteursPremiers commeEnsemble fais: [ :p |
      phi := phi*(1-(1/p))
      ].
   Transcript montre: phi.
   ].
```

7.5 Probabilités

7.5.1 Factorielle

Pour calculer une factorielle, on le demande à un entier :

```
Transcript affiche: 20 factorielle.
```

Exemple : Comme $8! = 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$, $8! + 1$ n'est divisible ni par 2, ni par 3, ni par 4, ni par 5, ni par 6, ni par 7, ni par 8. Ce nombre ne peut donc avoir que de grands diviseurs premiers. Vérification :

```
2 jusque: 10 faire: [ :n |
   Transcript affiche: (n factorielle + 1) facteursPremiers.
   ]
```

Cette constatation a mené Euclide à démontrer qu'il y a une infinité de nombres premiers.

7.5.2 Combinaisons

Pour savoir combien il y a de manières de choisir 5 cartes parmi 32, on entre

```
Transcript affiche: 5 parmi: 32.
```

On en déduit une façon de résoudre ce problème :

```
Quelle est la probabilité d'avoir un carré
d'as en puisant 5 cartes au hasard parmi 32 ?
```

Le nombre de mains gagnantes est le nombre de manières de choisir la cinquième carte (celle qui n'est pas un as) de la main ; pour avoir la probabilité voulue, on divise ce nombre par le nombre total de combinaisons calculé ci-dessus :

```
Transcript affiche: (((1 parmi: 28)/(5 parmi: 32)) commeProba).
```

Les coefficients binomiaux se représentent graphiquement comme une courbe en cloche :

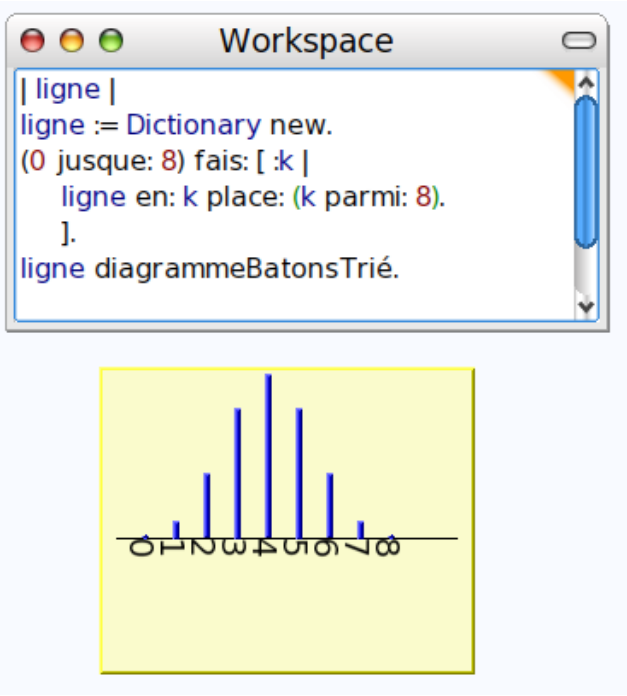

FIGURE 7.4 – La ligne 8 du triangle de Pascal

7.5.3 Alea

Pour choisir un nombre entre 1 et n au hasard, il suffit de le demander :

```
Transcript affiche: 12 auHasard.
```

Chapitre 8

Fractions

8.1 Affichages

Pour entrer une fraction, on la met entre parenthèses. Par exemple, pour deux tiers, on écrit `(2/3)`.

8.1.1 En toutes lettres

La méthode `avecDesMots` permet d'afficher le nom de la fraction, en toutes lettres. Par exemple, en choisissant de façon répétitive un numérateur et un dénominateur au hasard, on peut créer un exercice de grammaire :

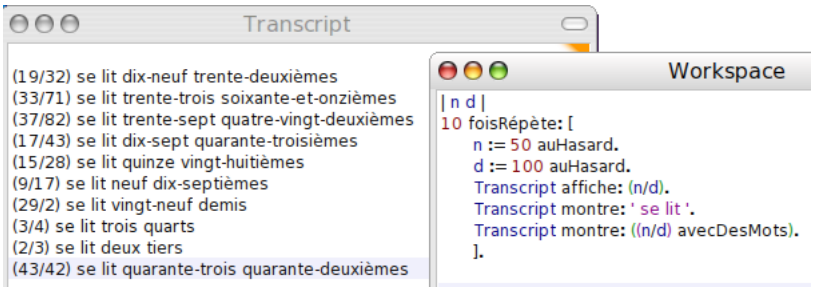

FIGURE 8.1 – *MathOntologie* permet de conceptualiser les fractions en leur associant des mots

8.1.2 En probabilités

On peut afficher une fraction inférieure à 1 comme le font les journalistes et les bookmakers. Par exemple, lorsqu'on lance un dé équilibré, la probabilité de gagner en obtenant un 6 est $\frac{1}{6}$ qui s'écrit «une chance sur 6» avec

```
Transcript affiche: (1/6) commeProba.
```

```
Comment parier équitablement sur l'obtention
d'un 6 en lançant un dé équilibré ?
```

Pour répondre à cette question, il suffit d'entrer

```
Transcript affiche: (1/6) commePari.
```

On apprend alors qu'il faut parier à 5 contre 1 (par exemple, pour chaque euro parié par le joueur sur le 6, la banque doit verser 5 € si le 6 sort) [1].

8.1.3 À l'ancienne

Pour préciser l'ordre de grandeur d'une fraction, on l'écrivait autrefois comme somme d'un entier et d'une fraction inférieure à 1. Par exemple, au lieu de $\frac{8}{5}$, on écrivait «1 3/5» parce que $\frac{8}{5} = 1 + \frac{3}{5}$. Pour retrouver cet affichage, on écrit

```
Transcript affiche: ((8/5) àLancienne).
```

Pour écrire une fraction comme somme de fractions «égyptiennes» [2], on peut écrire

```
Transcript affiche: ((355/113) àLégyptienne).
```

8.2 Opérations

8.2.1 fraction

Pour obtenir le numérateur d'une fraction, on lui envoie le message `numérateur`. De même, le message `dénominateur` permet d'obtenir le dénominateur d'une fraction :

1. En effet, en appelant p la somme pariée, l'espérance de gain est $5p \times \frac{1}{6} - p \times \frac{5}{6} = 0$ comme il se doit pour un jeu équitable

2. de numérateur égal à 1, ce sont donc des inverses d'entiers, garantis d'origine cairote© authentique.

```
Transcript affiche: (10/6) numérateur.
Transcript affiche: (10/6) dénominateur.
```

On constate que le numérateur de $\frac{10}{6}$ n'est pas 10 et que son dénominateur n'est pas 6.

```
 Le dictateur gentil président d'un état du Sud
a reçu 60 000 € d'aide alimentaire. Il consacre
les deux tiers de cette aide à la construction
d'une piscine dans son palais, et le cinquième
à l'achat d'un véhicule tout terrain «4×4».
Combien a-t-il dépensé en tout ?
```

Pour avoir le détail de la facture, on peut chercher à savoir combien font $\frac{2}{3}$ de 60000 et combien font $\frac{1}{5}$ de 60000 :

```
Transcript affiche: ((2/3) de: 60000).
Transcript affiche: ((1/5) de: 60000).

Transcript affiche: 40000+12000.
```

Après avoir dépensé 52 000 € de l'aide alimentaire, le dictateur gentil président ne laisse que 8 000 € à son peuple...

8.2.2 Opérations sur les fractions

On peut aussi résoudre le problème ci-dessus par une addition des deux fractions $\frac{2}{3}$ et $\frac{1}{5}$:

```
Transcript affiche: ((2/3)+(1/5)).
Transcript affiche: ((13/15) de: 60000).
```

Cet exemple tend à rendre les fractions plus concrètes... De même, on peut soustraire, multiplier et diviser des fractions avec les opérations $-$, $*$ et $/$:

```
Transcript affiche: ((2/3)+(1/5)).
Transcript affiche: ((2/3)-(1/5)).
Transcript affiche: ((2/3)*(1/5)).
Transcript affiche: ((2/3) de: (1/5)).
Transcript affiche: ((2/3) / (1/5)).
Transcript affiche: ((2/3) inverse).
```

On peut les élever à une puissance entière avec `puissance` :

```
Transcript affiche: ((2/3) puissance: 5).
Transcript affiche: ((2 puissance: 5)/(3 puissance: 5)).
Transcript affiche: ((2/3) auCarré).
Transcript affiche: ((2/3) auCube).
```

La multiplication se notant également `de`, on peut donc aussi aborder les fractions de fractions. Par exemple, pour avoir les deux cinquièmes des trois quarts de 60 000, on peut écrire

```
Transcript affiche: ((2/5) de: ((3/4) de: 60000)).
```

À propos, le nombre 3 % désignant en fait la fraction $\frac{3}{100}$, la notation `pourcentsDe` est réservée aux fractions de dénominateur 100 :

```
Transcript affiche: ((3/100) de: 60000).
Transcript affiche: (3 pourcentsDe: 60000).
```

8.3 Conversion

8.3.1 Fractions entières

Alors que le quotient de 2 par 6 n'est pas un nombre décimal, et que l'écriture $\frac{2}{6}$ est la meilleure façon d'écrire sa valeur exacte, il n'en est pas de même pour son inverse, le quotient de 6 par 2, qui lui est entier [3]. *MathOntologie* fait la conversion automatiquement, comme on l'a vu avec les exemples ci-dessus, le résultat étant affiché comme un entier et non comme une fraction [4].

8.3.2 Écriture décimale

Il subsiste encore un problème : Alors que le quotient de 2 par 7 par exemple, ne peut être donné que par une écriture fractionnaire parce qu'il n'est pas décimal, le quotient de 6 par 5, lui, est égal à 1,2. On ne peut s'empêcher de trouver que c'est un peu compliqué d'écrire $\frac{6}{5}$ pour ce quotient, alors qu'il est plus facile d'écrire 1,2. Pour *MathOntologie* qui est un champion des fractions, on peut convertir en décimal avec `commeDécimal`.

```
Transcript affiche: (100/40).
Transcript affiche: (100/40) commeDécimal.
Transcript affiche: (2/7) commeDécimal.
```

3. Plutôt qu'acheter 6 demi-baguettes, on préfère en général acheter 3 baguettes...
4. *Python* lui, répond `Fraction(52000,1)`...

Chapitre 9

Nombres réels

9.0.3 Approximations

Le gros problème avec *MathOntologie*, c'est que les calculs sont faits à l'intérieur en binaire, et que l'écriture binaire de 0,1 ne finit jamais :

```
Transcript affiche: (0.1 afficheEnBase: 2).
```

Le résultat `0.0001100110011001100110011001100110011001100110011001100110011` montre que l'écriture binaire est périodique et tronquée à 53 chiffres binaires (ou «bits»). Du coup, tout calcul avec des multiples ou sous-multiples de 0,1 est approché, et lors de l'affichage, des chiffres parasites apparaissent à la fin :

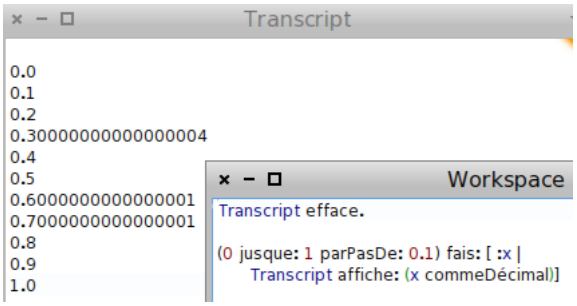

FIGURE 9.1 – Des nombres décimaux surprenants

Donc, pour comparer deux décimaux, il vaut mieux utiliser `estProcheDe` que l'égalité. Et pour afficher correctement un décimal, on peut imposer le nombre de décimales affichées :

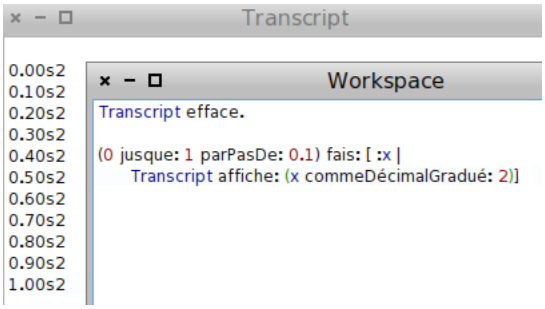

FIGURE 9.2 – Affichage à deux chiffres ; le «s» signifiant «significatifs»

L'affichage n'étant pas nécessairement satisfaisant, et ne s'appliquant pas au résultat calculé, *MathOntologie* peut approcher ou tronquer avec toutes sortes de méthodes, appelées `arrondi`, `arrondiA:` (qui arrondit à n chiffres après la virgule), `arrondiDéfautA:`, `arrondiExcèsA:`, `arrondiEntier` et `tronqué`, `tronquéA:`, `troncatureEntière`, `partieEntière` et `partieFractionnaire`.

```
Transcript affiche: (Float pi arrondiA: 0.01).
```

Il est bien connu que $\frac{6}{5} = 1,2$ mais *MathOntologie* ne le sait pas :

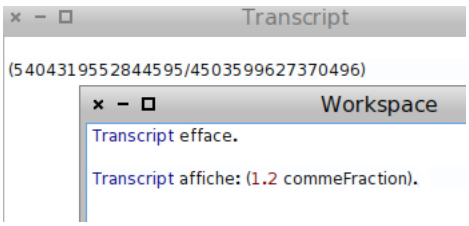

FIGURE 9.3 – Pourtant, quand on divise 6 par 5, on trouve 1,2

Pour se sortir de ce problème d'approximation de fractions, on se contente d'une fraction approchée :

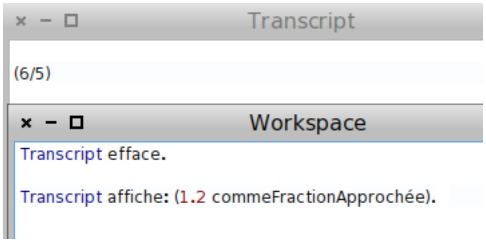

FIGURE 9.4 – Oui, là ça va mieux

9.0.4 Pour avoir des nombres vraiment décimaux

Le simple fait d'arrondir à peu de chiffres ne résout pas toujours le problème précédent :

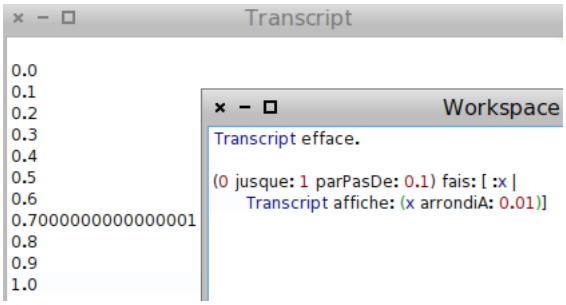

FIGURE 9.5 – 0,7 est un nombre magique ... ou ensorcelé !

La meilleure solution est finalement le passage par des fractions approchées puis le retour aux nombres décimaux :

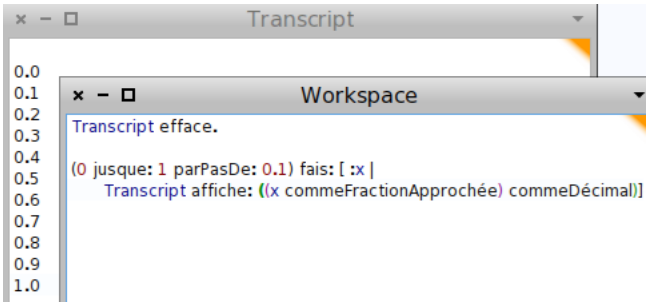

FIGURE 9.6 – Comment additionner 0,1 sans heurts ni erreurs ni horreurs

9.1 Proportionnalité

9.1.1 Règle de trois

```
 Sachant que 3 litres de peinture coûtent 8 €,
combien coûtent 4,5 litres de peinture ?
```

On entre

```
Transcript affiche: (8 devientCombienSachantQue: 3 devient: 4.5).
```

Les méthodes `pourcentsDe:` et `augmentéDePourcents:` répondent aussi à des problèmes de proportionnalité.

9.1.2 Angles

Pour savoir combien de degrés il y a dans $\frac{\pi}{6}$ radians, on entre

```
Transcript affiche: ((Float pi / 6) degrés).
```

Pour savoir combien il y a de radians dans 45°, on fait au contraire

```
Transcript affiche: (45 radians).
```

Pour avoir la mesure principale de 10 radians, on fait

```
Transcript affiche: (10 reste: (Float pi * 2)).
```

9.2 Fonctions

Outre les fonctions trigonométriques `sin`, `cos` et `tan` en radians, *MathOntologie* possède aussi des fonctions en degrés comme `sinus`, `cosinus` et `tangente`. *MathOntologie* possède aussi des fonctions `log`, `exp` (logarithme népérien et exponentielle), `puissance:`, `auCarré` et `auCube` déjà vues. Mais aussi `abs` pour la valeur absolue, `signe` pour le signe (1 ou -1), `opposé` pour l'opposé :

```
Transcript affiche: ((−3) opposé).
```

Pour trouver un nombre dont le carré est 6,25 on peut faire

```
Transcript affiche: (6.25 racine).
```

Pour trouver un nombre dont le cube est 125 on peut faire

```
Transcript affiche: (125 racineCubique).
```

Pour trouver un nombre dont la puissance cinquième est 32, on peut faire

```
Transcript affiche: (32 racineNième: 5).
```

On peut représenter graphiquement une fonction définie par un bloc, avec `bloc dessineDe: a jusque: b`, où a et b sont les bornes de l'intervalle sur lequel la fonction est dessinée. Le graphique obtenu est à considérer comme une esquisse (sans axe dessiné, sans gestion des erreurs) que l'on peut déplacer avec la souris, et fermer en effectuant un `Alt-Shift-Clic` dessus, puis en cliquant sur le bouton de fermeture (en haut à gauche).

9.2.1 incrémentation et décrémentation

Pour ajouter 1 à un nombre, on peut lui envoyer le message `suivant` :

```
Transcript affiche: (7 suivant).
```

De même pour soustraire 1 à un nombre on peut lui envoyer le message `précédent`.

9.2.2 Analyse numérique

```
 Quels sont le minimum et le maximum de la
fonction x ↦ x − x^3/25 sur l'intervalle [-5;5]?
Quelle est sa dérivée? Quelle est son intégrale
de 0 jusqu'à 5?
```

Pour étudier une fonction, il faut la définir par un bloc (entre crochets) ; dans l'exemple présent, on le rédige ainsi :

```
| f |
f := [ :x | x − (x auCube / 25)].
```

1. Recherche d'un zéro
   Pour savoir où la fonction f s'annule sur [a ;b], on peut tenter un

   ```
   Transcript affiche: (f valeurAnnulantDe: a jusque: b).
   ```

2. Extrema sur un intervalle
   Pour obtenir le maximum de la fonction sur l'intervalle [a ;b], on utilise `maximumDe: a jusque: b:`

   ```
   | f |
   f := [ :x | x − (x auCube / 25)].
   Transcript affiche: ((f maximumEntre: −5 et: 5) arrondiA: 0.001).
   ```

   Et pour savoir où la fonction atteint son maximum :

   ```
   | f |
   f := [ :x | x − (x auCube / 25)].
   Transcript affiche: ((f valeurMaximisantEntre: −5 et: 5) arrondiA: 0.001).
   ```

   De même, le minumum s'obtient avec `minimumDe: a jusque: b:`

   ```
   | f |
   f := [ :x | x − (x auCube / 25)].
   Transcript affiche: ((f minimumEntre: −5 et: 5) arrondiA: 0.001).
   ```

   Pour savoir où le minimum est atteint on peut entrer

   ```
   | f |
   f := [ :x | x − (x auCube / 25)].
   Transcript affiche: ((f valeurMinimisantEntre: −5 et: 5) arrondiA: 0.001).
   ```

3. Pour obtenir le nombre dérivé en a, on utilise `dérivéeEn: a` :

   ```
   | f |
   f := [ :x | x − (x auCube / 25)].
   Transcript affiche: (f dérivéeEn: (5/ (3 racine))).
   ```

   On peut également, par une définition de fonction en bloc, obtenir la ***fonction dérivée*** de f :

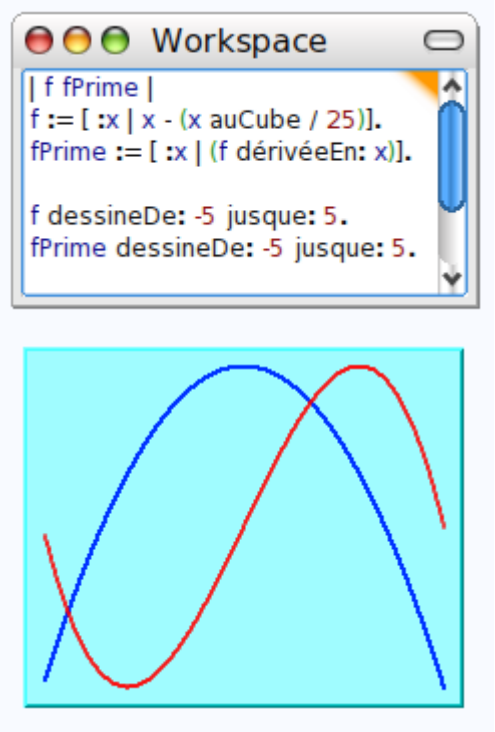

FIGURE 9.7 – La dérivée est représentée en bleu, par-dessus la fonction qui elle, est représentée en rouge

4. Pour obtenir l'intégrale entre a et b, on utilise `intégraleDe: a jusque: b:`

```
| f |
f := [ :x | x − (x auCube / 25)].
Transcript affiche: ((f intégraleDe: 0 jusque: 5) arrondiA: 0.001).
```

9.3 tests sur les nombres

MathOntologie peut répondre à beaucoup de questions sur les nombres (la réponse est `true` ou `false`) :

1. `estUnNombre` répond toujours vrai, et `nEstPasUnNombre` répond toujours faux.
2. `positif`, `strictementPositif`, `négatif` et `estNul` répondent à des questions sur le signe d'un nombre. De plus, $\leqslant$ se note $<=$ et $\neq$ se note $\sim=$.

3. `estInfini` évoque le fait que sous *MathOntologie*, l'infini est considéré comme un nombre :

```
Transcript affiche: (0 In estInfini).
```

Chapitre 10

Géométrie repérée

Dans un repère orthonormé, on considère les points A(−1;3), B(5;1) et C(1;5).

1. Quelle est la nature du triangle ABC?
2. En déduire les coordonnées du centre M de son cercle circonscrit.
3. Calculer l'aire de ABC.
4. Donner l'équation réduite de la droite (AB).

10.1 Points

10.1.1 Définitions

Pour entrer un point en *MathOntologie*, on écrit son abscisse suivie du caractère `arobase`, suivi de son ordonnée. Une affectation permet de stocker ces coordonnées dans des variables qui sont des points :

```
| A B C |
A := −1@3.
B := 5@1.
C := 1@5.
```

Remarque : Si l'ordonnée est négative il faut la mettre entre parenthèses, à cause d'une ambigüité entre les deux opérateurs `arobase` et `moins`.

Pour avoir l'abscisse d'un point, on fait suivre son nom de la lettre "x" ; et de même, l'ordonnée d'un point s'obtient avec "y" :

```
| A B C |
A := −1@3.

Transcript affiche: (A x).
Transcript affiche: (A y).
```

10.1.2 Milieu

Pour avoir le milieu du segment [AB], on entre `A milieu: B`:

```
| A B C M |
A := −1@3.
B := 5@1.
C := 1@5.
M := A milieu: B.
Transcript affiche: M.
```

10.1.3 Symétrique

Pour avoir le symétrique de A par rapport à B, on entre

```
| A B |
A := −1@3.
B := 5@1.

Transcript affiche: (A symétriqueParRapportà: B).
```

Pour avoir le symétrique de A par rapport à la droite (BC), on entre

```
| A B C |
A := −1@3.
B := 5@1.
C := 1@5.

Transcript affiche: (A symétriqueParRapportàLaDroitePassantPar: B et: C).
```

10.1.4 Distances

Pour avoir la distance AB, on entre `A dist: B`:

```
| A B C |
A := −1@3.
B := 5@1.
```

```
C := 1@5.
Transcript affiche: (A dist: B).
Transcript affiche: (A dist: C).
Transcript affiche: (B dist: C).
```

Ces distances ne permettent pas de répondre à la première question de l'exercice ouvrant ce chapitre (sauf à dire que le triangle n'est visiblement pas isocèle). Alors autant utiliser Pythagore, en calculant le carré des distances :

```
| A B C |
A := −1@3.
B := 5@1.
C := 1@5.
Transcript affiche: (A distanceAuCarréJusque: B).
Transcript affiche: (A distanceAuCarréJusque: C).
Transcript affiche: (B distanceAuCarréJusque: C).
```

Comme 8+32=40, on sait maintenant que le triangle est rectangle et que son hypoténuse est [AB]. On en déduit donc que le centre de son cercle circonscrit est le milieu de [AB], soit le point M trouvé précédemment.

10.1.5 Situation

Pour vérifier que M est dans le cercle circonscrit à ABC, on peut utiliser un test :

```
| A B C M |
A := −1@3.
B := 5@1.
C := 1@5.
M := A milieu: B.
Transcript affiche: (M estDansLeCerclePassantPar: A et: B et: C).
```

De même, on peut vérifier que M est à l'intérieur du triangle ABC, on peut faire ce test :

```
| A B C M |
A := −1@3.
B := 5@1.
C := 1@5.
M := A milieu: B.
Transcript affiche: (M estDansLeTriangleDeSommets: A et: B et: C).
```

Enfin, on peut vérifier que M est sur la droite (AB) avec

```
| A B C M |
A := −1@3.
B := 5@1.
C := 1@5.
M := A milieu: B.
Transcript affiche: (M estAlignéAvec: A et: B).
```

10.2 Triangles

Outre les tests sur l'intérieur du triangle et son cercle circonscrit vus précédemment, *MathOntologie* peut calculer l'aire d'un triangle :

```
| A B C |
A := −1@3.
B := 5@1.
C := 1@5.

Transcript affiche: (A aireDuTriangleAvec: B et: C).
```

Remarque : Le calcul exact peut être mené avec les carrés des distances, puisqu'on sait que ABC est rectangle, et que les côtés de son angle droit valent $\sqrt{32}$ et $\sqrt{8}$. Alors son aire vaut

$$\frac{\sqrt{32}\times\sqrt{8}}{2}=\frac{\sqrt{32\times 8}}{2}=\frac{\sqrt{256}}{2}=\frac{16}{2}=8.$$

On peut alors calculer la hauteur issue de C dans le triangle ABC : Puisque le produit de cette hauteur par la base AB correspondante est 16, et que $AB=\sqrt{40}$, la hauteur vaut

$$\frac{16}{\sqrt{40}}=\frac{16}{2\sqrt{10}}=\frac{8}{\sqrt{10}}$$

On peut vérifier cela en écrivant que la hauteur est la distance de C à H (pied de la hauteur issue de C), et en calculant H comme point de (AB) le plus proche de C :

```
| A B C H |
A := −1@3.
B := 5@1.
C := 1@5.
H := C pointLePlusProcheSurLaDroitePassantPar: A et: B.
Transcript affiche: (C dist: H).
Transcript affiche: (8 / (10 racine)).
```

10.3 Vecteurs

Les vecteurs se traitent exactement comme des points : abscisse suivie du caractère `arobase` puis ordonnée ; l'abscisse de v s'obtient avec `v x` et son ordonnée par `v y`.

1. Calculs
   Pour calculer et afficher les coordonnées du vecteur $\overrightarrow{AB}$, on fait

```
| A B |
A := −1@3.
B := 5@1.

Transcript affiche: (A vecteur: B).
```

   Pour additionner ou soustraire deux vecteurs :

```
| u v |
u := 4@1.
v := 2@(−3).

Transcript affiche: (u+v).
Transcript affiche: (u−v).
```

   .
   Pour multiplier un vecteur par un réel :

```
| u v |
u := 4@1.
v := 2.5*u.

Transcript affiche: v.
```

   Pour appliquer une translation à un point A, on fait

```
A := −1@3.
u := 4@1.

Transcript affiche: (A translateSelon: u).
```

2. Coordonnées polaires
   Pour avoir le rayon vecteur d'un point (ou la norme d'un vecteur) on utilise la lettre `r` ; pour avoir son angle, on utilise le mot `theta` ou le mot `angle` :

```
| u |
u := 4@1.


Transcript affiche: (u r).
Transcript affiche: (u theta).
```

Pour obtenir l'image C de B dans une rotation de 30° autour de A, on peut faire

```
| A B C |
A := −1@3.
B := 5@1.
C := B tourneDe: (30 radians) autourDe: A.
Transcript affiche: C.
```

3. Colinéarité

   Deux vecteurs sont colinéaires lorsque leur déterminant est nul ; celui-ci se calcule avec

```
| u v |
u := 4@1.
v := 2@(−3).

Transcript affiche: (u déterminant: v).
```

   Pour savoir si deux vecteurs sont colinéaires, il suffit de le demander :

```
| u v |
u := 4@1.
v := 2@(−3).

Transcript affiche: (u estColinéaireAvec: v).
```

4. Produit scalaire

   Pour calculer le produit scalaire de deux vecteurs, on fait :

```
| u v |
u := 4@1.
v := 2@(−3).

Transcript affiche: (u scalaire: v).
```

10.4 Droites

1. Vecteur directeur
   Pour avoir un vecteur directeur de la droite (AB), on calcule juste le vecteur $\overrightarrow{AB}$ (voir plus haut).
2. Vecteur normal
   Un vecteur normal unitaire de la droite (AB) s'obtient en envoyant le message `normal` à un de ses vecteurs directeurs :

```
| A B |
A := −1@3.
B := 5@1.

Transcript affiche: ((A vecteur: B) normal).
```

3. Équation réduite
   Pour calculer l'équation réduite de la droite (AB), on écrit

```
| A B |
A := −1@3.
B := 5@1.

Transcript affiche: (A équationDroiteJusque: B).
```

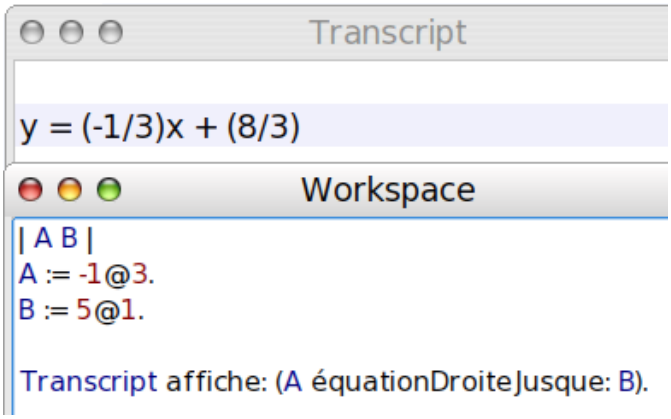

FIGURE 10.1 – Les coefficients de l'équation réduite sont donnés en valeur exacte

Chapitre 11

Probabilités

11.1 Listes et autres collections

11.1.1 Listes

Une liste est notée par un «dièse» suivi de parenthèses en *MathOntologie*. Les éléments de la liste sont séparés par des espaces. La liste des résultats de lancers de dés peut s'écrire ainsi :

```
Transcript affiche: #(1 2 3 4 5 6).
```

Mais il y a plus simple avec une conversion en liste :

```
Transcript affiche: (1 jusque: 6) commeTableau.
```

Les éléments d'une liste sont numérotés à partir de 1 [1]. Pour accéder à l'élément numéro 3 d'une liste *l* on peut faire

```
Transcript affiche: l en: 3.
```

Pour modifier l'élément numéro 3 de la liste *l*, on y pose un nouvel élément :

```
l en: 3 pose: 'joker'.
```

11.1.2 Collections ordonnées

La principale différence entre une liste et une collection ordonnée c'est que cette dernière n'a pas une taille fixe, et qu'on peut y ajouter des éléments au fur et à mesure. Ce qui permet de créer par simulation des échantillons statistiques à étudier ; par exemple pour simuler 1000 lancers de

1. et non de zéro comme avec la plupart des outils logiciels

deux dés, on peut créer une collection vide, puis y placer l'un après l'autre les résultats des 1000 lancers :

```
| résultats |
résultats := #() commeCollectionOrdonnée.
1000 foisRépète: [ résultats ajoute: (6 auHasard + (6 auHasard)).].
Transcript affiche: (résultats compte: [ :x | x=6]).
```

Ci-dessus on a compté le nombre de 6 dans la collection. Au besoin, on peut convertir une collection ordonnée en liste ou *vice-versa*.

11.1.3 Multiensembles

Le multiensemble est une collection où l'ordre ne compte pas, par exemple une urne dont on a mélangé les boules. Pour faire des statistiques, on n'a en général pas besoin de l'ordre dans lequel les éléments ont été placés dans l'historique, et on peut utiliser un multiensemble ou *sac*. Donc pour lancer 1000 fois deux dés, on peut créer une liste vide, la convertir en un sac vide, puis 1000 fois de suite, répéter l'opération consistant à mettre un résultat de lancer de dés dans le sac :

```
| résultats |
résultats := #() commeSac.
1000 foisRépète: [ résultats ajoute: (6 auHasard + (6 auHasard))].
Transcript affiche: (résultats compte: [ :x | x pair]).
```

Cette fois-ci, on compte combien de résultats sont pairs.

Pour ajouter plusieurs exemplaires d'un même objet, on peut faire `ajoute: 'joker' fois: 5`.

11.1.4 Ensembles

Un ensemble veille à ne pas avoir deux fois le même élément. Cette notion sert à modéliser les ensembles de solutions d'équations, ou les évènements en théorie des probabilités. On en a vu un exemple dans le chapitre sur l'arithmétique avec l'indicatrice d'Euler. Pour avoir un ensemble, on peut convertir une liste ou autre collection avec `commeEnsemble`, ou ajouter des éléments [2]. Les ensembles peuvent aussi se construire à partir d'autres ensembles par intersection ($\cap$ noté `inter:`) ou par réunion ($\cup$ noté `union`). On peut aussi ajouter un objet s'il n'est pas déjà dans l'ensemble, avec `ajouteAuBesoin`.

2. ou en retirer d'ailleurs

Pour compter le nombre total d'éléments d'un ensemble, on lui demande sa taille :

```
Transcript affiche: (((1 jusque: 100) choisis: [ :p | p estPremier]) taille).
```

On peut alors définir la probabilité d'un évènement comme le quotient de sa taille par celle de l'univers :

```
| univers evt |
univers := (1 jusque: 12) commeEnsemble.
evt := univers choisis: [ :p | p estPremier].
Transcript affiche: 'En lançant un dé dodécaédrique, la probabilité que le résultat soit
    premier est de '.
Transcript montre: ((evt taille/(univers taille)) commeProba).
```

11.1.5 Collections triées

En envoyant à une collection (liste ou autre) le message `commeCollectionTriée`, on le transforme en collection qui se trie automatiquement à chaque ajout d'un élément. Cette collection est de peu d'utilité en *MathOntologie* puisqu'il est possible de trier une collection après coup, en lui envoyant le message `trié`.

11.1.6 Dictionnaires

Un dictionnaire est une sorte de tableau dont les index ne sont pas nécessairement des nombres qui se suivent. Cette notion modélise la notion mathématique de fonction entre ensembles finis. L'ensemble de départ s'obtient par `keys` et l'ensemble d'arrivée, par `values`. On peut utiliser les dictionnaires pour formaliser la notion de variables aléatoires, comme dans cet exemple où on paye 2 € pour avoir le droit de parier sur un 6, et dans ce cas, on gagne 10 euros :

```
| X univers |
X := Dictionary newFrom: { 1−> −2 . 2−> −2 . 3−> −2 . 4−> −2 . 5−> −2 . 6−>10}.
univers := X keys.
Transcript affiche: univers.
```

Les tableaux d'effectifs et de fréquences (voir chapitre «statistiques») sont des dictionnaires de *MathOntologie*. Lorsque ses valeurs sont des nombres positifs, un dictionnaire peut être représenté graphiquement par `dictionnaire diagrammeBatons` ; lorsque, de plus, ses clés sont aussi des nombres, le dictionnaire est avantageusement repreésenté graphiquement par `dictionnaire diagrammeBatonsTrié` ou par `dictionnaire dessineHistogramme`.

11.2 Sous-ensembles

11.2.1 Ensemble vide

```
Transcript affiche: (#(1 3 5) inter: #(2 4 6)) estVide.
```

Pour savoir si un sous-ensemble est vide, on peut faire

```
Transcript affiche: ((1 jusque: 100) aucunNeVérifie: [ :p | p estPremier]).
```

11.2.2 Élément

Pour savoir si une collection contient un objet on peut faire

```
Transcript affiche: ((1 jusque: 100) contient: 2013).
```

11.2.3 Choix d'un élément au hasard

`auHasard` déjà vu plusieurs fois, permet de choisir un élément d'une collection au hasard. Par exemple, pour extraire une boule d'une urne contenant 10 boules rouges et 90 boules blanches, on peut procéder ainsi :

```
| urne |
urne := #() commeSac.
90 foisRépète: [urne ajoute: 'blanche'].
10 foisRépète: [urne ajoute: 'rouge'].
8 foisRépète: [Transcript affiche: urne auHasard].
```

11.2.4 Extraction

Pour extraire les éléments premiers d'une liste de nombres, on peut faire ainsi :

```
Transcript affiche: ((1 jusque: 100) choisis: [ :p | p estPremier]).
```

Pour savoir si une liste contient au moins un nombre premier, on peut faire

```
Transcript affiche: ((1 jusque: 100) unVérifie: [ :p | p estPremier]).
```

Pour savoir si tous les éléments d'une liste sont premiers, on peut faire

```
Transcript affiche: ((1 jusque: 100) tousVérifient: [ :p | p estPremier]).
```

Pour compter les éléments premiers d'une liste, on peut faire :

```
Transcript affiche: ((1 jusque: 100) compte: [ :p | p estPremier]).
```

Pour obtenir le complémentaire d'un sous-ensemble (le contraire d'un évènement en probabilités), on fait

```
Transcript affiche: ((1 jusque: 100) exclus: [ :p | p estPremier]).
```

11.2.5 Enlever

Pour ne garder que les nombres composés dans une collection de nombres, on peut faire :

```
Transcript affiche: ((1 jusque: 100) commeSac retireTousTelsQue: [ :p | p estPremier]).
```

Pour vider une collection («vider son sac»), il suffit d'écrire

```
Transcript affiche: ((1 jusque: 100) commeSac retireTout).
```

11.3 Opérations sur les listes

11.3.1 faire agir chaque élément

On peut demander à tous les éléments d'une collection d'exécuter un bloc, ce qui permet d'appliquer une fonction non seulement à un nombre mais également à une liste de nombres :

```
Transcript affiche: (#(2 3 5 7) fais: [ :x | Transcript affiche: (x auCarré)]).
```

Ceci affiche les carrés des nombres premiers à un chiffre.

11.3.2 Machine de Turing

`picore` a le même effet que `fais` sauf que les résultats sont stockés dans une collection. Son nom vient d'une ressemblance avec le comportement d'une machine de Turing, une collection initialement vide qui va récolter l'un après l'autre les éléments d'une liste, et pour chacun d'eux stocker le résultat de l'exécution d'un bloc.

Par exemple, on peut récolter, à partir des numéros d'un dé, leurs noms :

```
Transcript affiche: ((1 jusque: 6) picore: [ :x | x avecDesMots]).
```

Pour fabriquer un dé en chiffres romains :

```
| ALEA |
ALEA := ((1 jusque: 6) picore: [ :x | x enChiffresRomains]).
Transcript affiche: ALEA.
8 foisRépète: [ Transcript affiche: ALEA auHasard.].
```

`picore` permet d'appliquer plusieurs fonctions directement à tous les éléments d'une liste de nombres : `opposés`, `abs`, `inverses`, `arrondis`, `auCarré` ou `puissances` (ce dernier, suivi d'un exposant). Les fonctions `cosinus` et `sinus` également peuvent s'appliquer à une liste. Ce qui permet de fabriquer assez facilement une table trigonométrique :

```
Transcript affiche: ((1 jusque: 90) cosinus).
```

11.3.3 injecte

Les méthodes précédentes utilisaient une variable (dans le bloc, ci-dessus notée `x`) qui prenait successivement les valeurs de la liste. La présente autilise deux variables, l'une qui parcourt les éléments de la liste, l'autre qui se met à jour (typiquement la somme) dans le bloc. Par exemple pour calculer la factorielle de 8, on peut injecter les facteurs (à partir de la valeur initiale 1) allant de 1 à 8 dans un produit qui, à la fin, sera la factorielle voulue :

```
Transcript affiche: ((1 jusque: 8) injecte: 1 dans: [ :p :f | p*f]).
```

Pour avoir la factorielle de 8, il est plus simple de faire :

```
Transcript affiche: (8 factorielle).
```

Et pour additionner les éléments d'une liste, on fait

```
Transcript affiche: ((1 jusque: 10) somme).
```

Mais les codes sources de ces méthodes utilisent `injecte`.

Chapitre 12

Statistiques

```
 Quel est le diviseur médian de la factorielle
de 10 ? Quels sont les quartiles ? Quel est le
diviseur moyen ?
```

12.1 Quantiles

12.1.1 Valeurs extrêmes d'une liste

Pour avoir le plus petit diviseur de $10!$, on peut faire

```
Transcript affiche: (10 factorielle diviseurs min).
```

De même, le plus grand des diviseurs de $10!$ est obtenu avec

```
Transcript affiche: (10 factorielle diviseurs max).
```

Ces méthodes sont plus pratiques qu'elles paraissent, parce que chercher directement dans la liste des diviseurs est difficile si celle-ci est longue [1] :

```
Transcript affiche: (10 factorielle diviseurs taille).
```

Enfin, l'étendue s'obtient ainsi :

```
Transcript affiche: (10 factorielle diviseurs étendue).
```

1. Sauf que dans le cas présent, on sait bien que le plus petit diviseur de $10!$ est 1 et que son plus grand diviseur est $10!$...

12.1.2 Quartiles

Pour avoir la médiane, on fait

```
Transcript affiche: (10 factorielle diviseurs médiane).
```

Pour les deux autres quartiles :

```
Transcript affiche: (10 factorielle diviseurs premierQuartile).
Transcript affiche: (10 factorielle diviseurs troisièmeQuartile).
```

12.2 Moyenne et écart-type

12.2.1 Moyenne

```
Transcript affiche: (10 factorielle diviseurs moyenne).
```

12.2.2 Écart-type

```
Transcript affiche: (10 factorielle diviseurs écartType).
```

12.3 Statistiques

On jette deux dés mille fois de suite ; on souhaite avoir le tableau des effectifs et le tableau des fréquences.

12.3.1 Effectifs

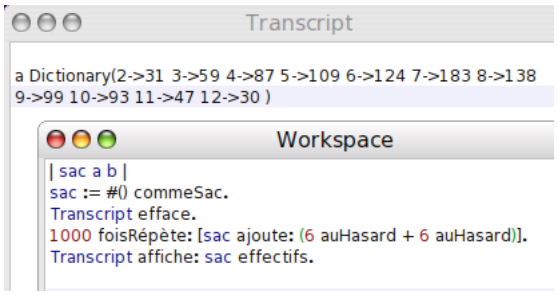

FIGURE 12.1 – le 7 sort nettement plus souvent que le 3

12.3.2 Fréquences

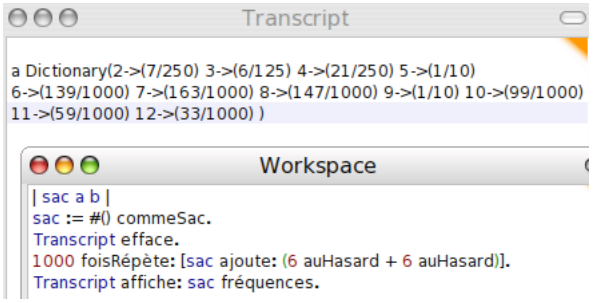

FIGURE 12.2 – le 7 sort vraiment plus souvent que le 3

12.3.3 Graphiques

1. Le ***diagramme en batons*** sert à représenter la répartition de caractères qualitatifs :

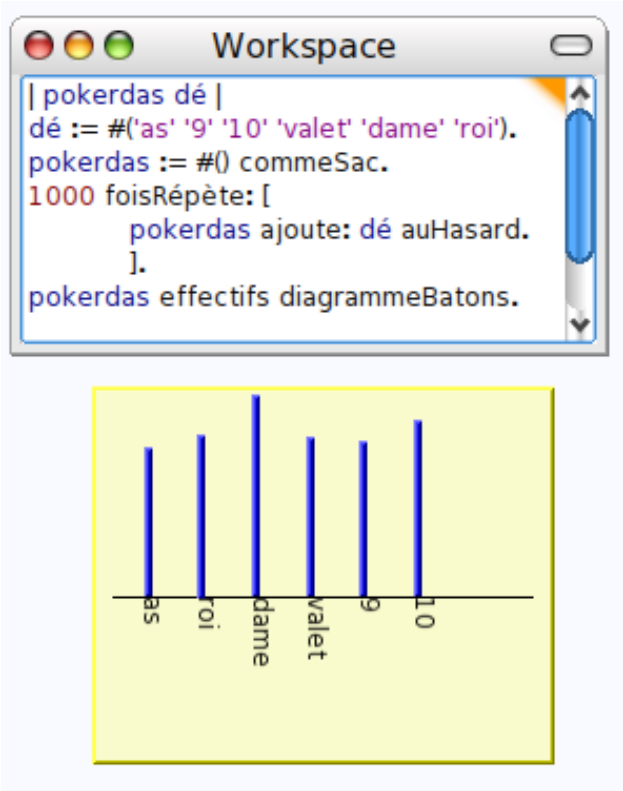

FIGURE 12.3 – Plus de chances avec les dames qu'au poker d'as...

On obtient un diagramme en batons avec stats effectifs diagrammeBatons ; on l'efface avec `Alt-Shift-clic` (du bouton gauche de la souris) puis en cliquant sur le bouton de fermeture en haut à gauche.

2. Le ***diagramme en batons trié*** sert à représenter les effectifs (ou fréquences) d'un caractère quantitatif discret (ce sont les valeurs du caractère qui sont triées). On l'utilise avec `stats fréquences diagrammeBatonsTrié`. Un exemple se trouve à la fin du chapitre sur l'arithmétique. En voici un autre, toujours avec le lancer de deux dés :

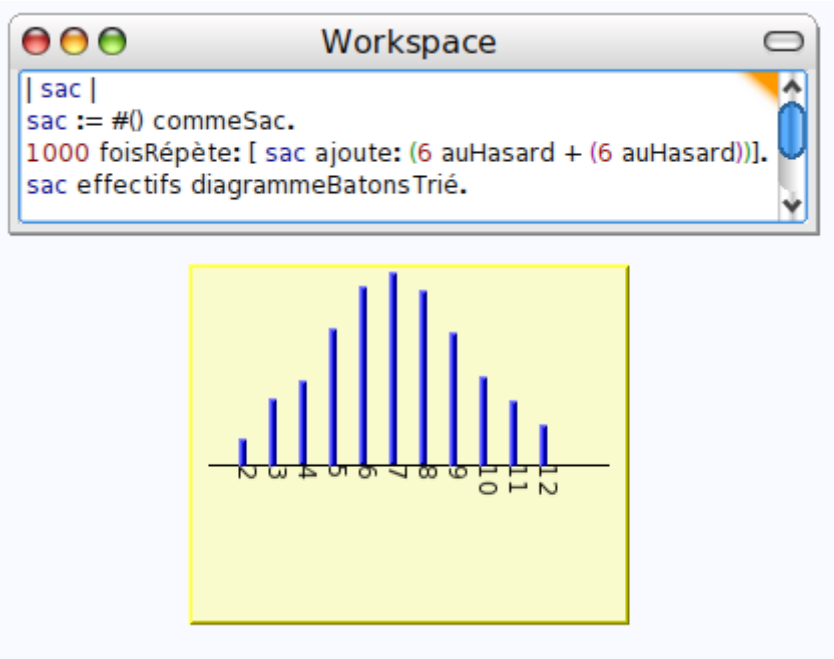

FIGURE 12.4 – Mais puisqu'on vous dit que le 7 sort plus souvent que le 3

3. L'***histogramme*** sert à représenter graphiquement les effectifs (ou fréquences) des caractères quantitatifs continus ; on peut toutefois l'utiliser aussi avec des caractères discrets, comme ici, pour montrer qu'avec deux dés, le 7 sort nettement plus souvent que le 3 :

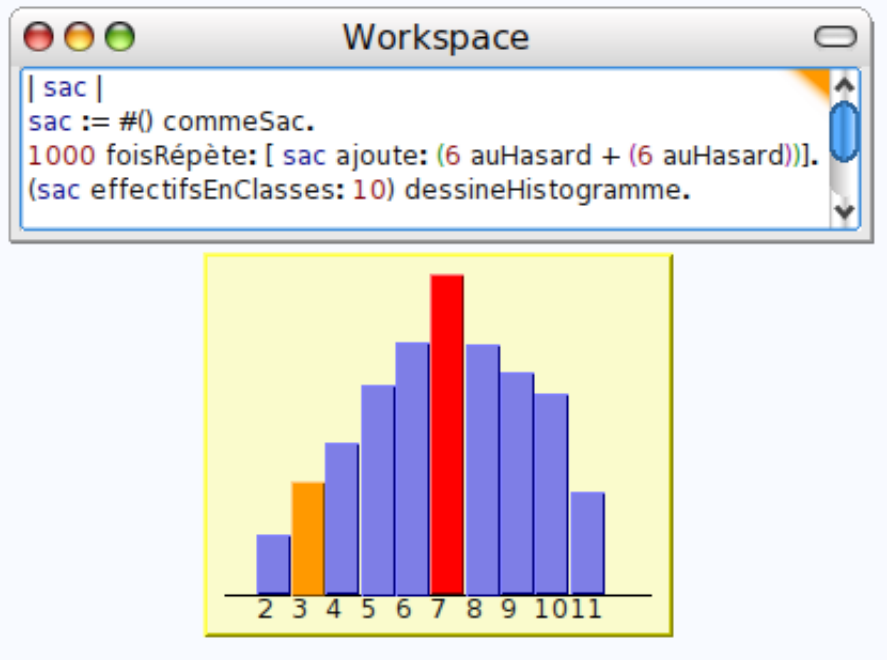

FIGURE 12.5 – Avec le `Alt-Shift-Clic` sur l'un des rectangles de l'histogramme, on peut changer sa couleur et ainsi, mettre en exergue le fait que le 7 sort plus souvent que le 3

Pour représenter en histogramme une série statistique numérique, il faut la transformer avec `effectifsEnClasses`, qui attend un paramètre entier : Le nombre de classes à afficher.

Chapitre 13

Nombres complexes

13.1 Obtention

13.1.1 Affixe

Pour avoir l'affixe d'un point, il suffit de transformer le point en un nombre complexe :

```
Transcript affiche: ((−1@3) commeComplexe).
```

Un réel peut aussi être considéré comme complexe (en fait, transformé en complexe) avec cette méthode.

13.1.2 Parties réelle et imaginaire

```
| z |
z := (−1@3) commeComplexe.

Transcript affiche: (z partieRéelle).
Transcript affiche: (z partieImaginaire).
```

13.1.3 Test

Le test `estComplexe` donne `true` s'il est appliqué à un complexe, et uniquement dans ce cas-là.

Exemple : Si on essaye de calculer la racine de -1, on a un message d'erreur :

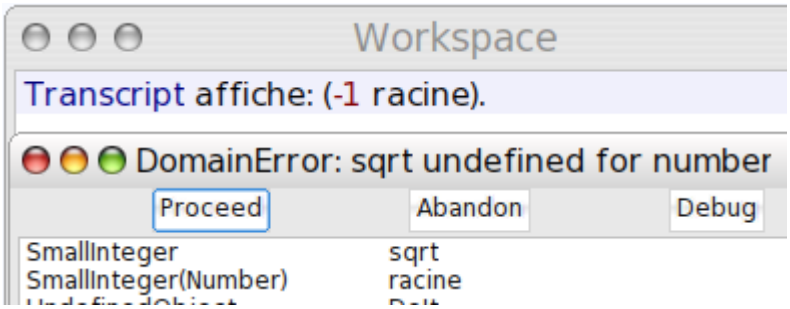

FIGURE 13.1 – Le fameux bug de Bombelli

Mais il suffit de dire que -1 doit être considéré comme complexe, pour avoir un résultat plus intéressant :

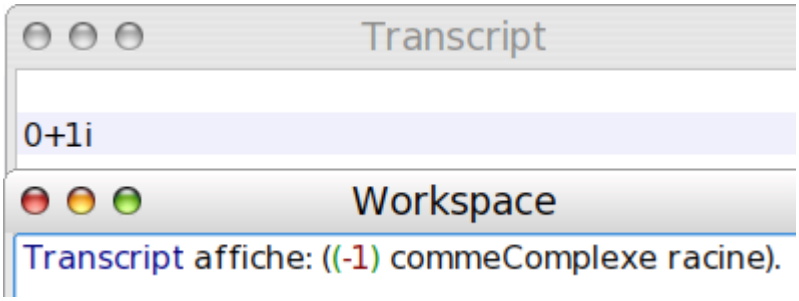

FIGURE 13.2 – On fait comme si : i

13.1.4 Conjugué, module et argument

Le conjugué d'un nombre complexe s'obtient ainsi :

```
Transcript affiche: ((−1@3) commeComplexe conjugué).
```

Le module et l'argument :

```
Transcript affiche: ((−1@3) commeComplexe module).
Transcript affiche: ((−1@3) commeComplexe argument).
Transcript affiche: ((−1@3) commeComplexe moduleAuCarré).
```

13.2 Opérations

13.2.1 Addition et soustraction

```
| a b |
a := (−1@3) commeComplexe.
b := (5@1) commeComplexe.

Transcript affiche: (a+b).
Transcript affiche: (a−b).
Transcript affiche: (a opposé).
```

13.2.2 Multiplication et division

```
| a b |
a := (−1@3) commeComplexe.
b := (5@1) commeComplexe.

Transcript affiche: (a*b).
Transcript affiche: (a/b).
Transcript affiche: (a inverse).
```

13.2.3 Puissances

Un nombre complexe peut être élevé à une puissance entière (éventuellement négative) avec `puissance:` comme les réels. Dans les cas particuliers où l'exposant est -1, 2 ou 3, on peut utiliser `inverse`, `auCarré` ou `auCube`.

13.3 Fonctions

13.3.1 Racines

On peut calculer une racine carrée avec `racine` comme on l'a vu plus haut. Voici comment on peut résoudre des équations du second degré avec ça :

1. La version réelle :

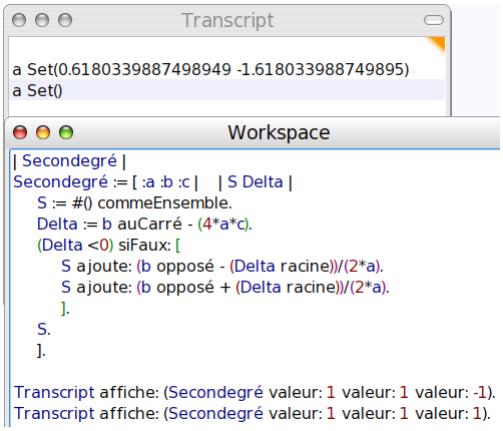

FIGURE 13.3 – Si Δ est négatif, on n'a pas de solution (ensemble vide)

2. Et la version complexe :

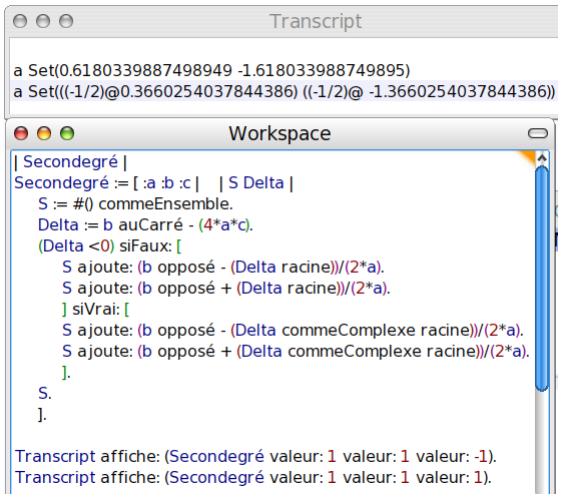

FIGURE 13.4 – Si $\Delta < 0$, les solutions sont complexes (et conjuguées)

On peut également obtenir une racine cubque d'un nombre complexe avec `racineCubique`, voire une racine quatrième avec `racineNième`:

```
4
```

13.4 Logarithme et exponentielle

13.4.1 Logarithme

Le logarithme népérien de z s'obtient par `z ln`.

13.4.2 exponentielle

L'exponentielle de z s'obtient par `z exp`.

13.5 Trigonométrie

13.5.1 Sinus

Le sinus de z s'obtient par `z sin`.

13.5.2 Cosinus

Le cosinus de z s'obtient par `z cos`.

13.6 Représentation graphique

Pour représenter graphiquement une fonction sur $\mathbb{C}$, il suffit d'entrer des bornes complexes pour l'intervalle de définition du tracé :

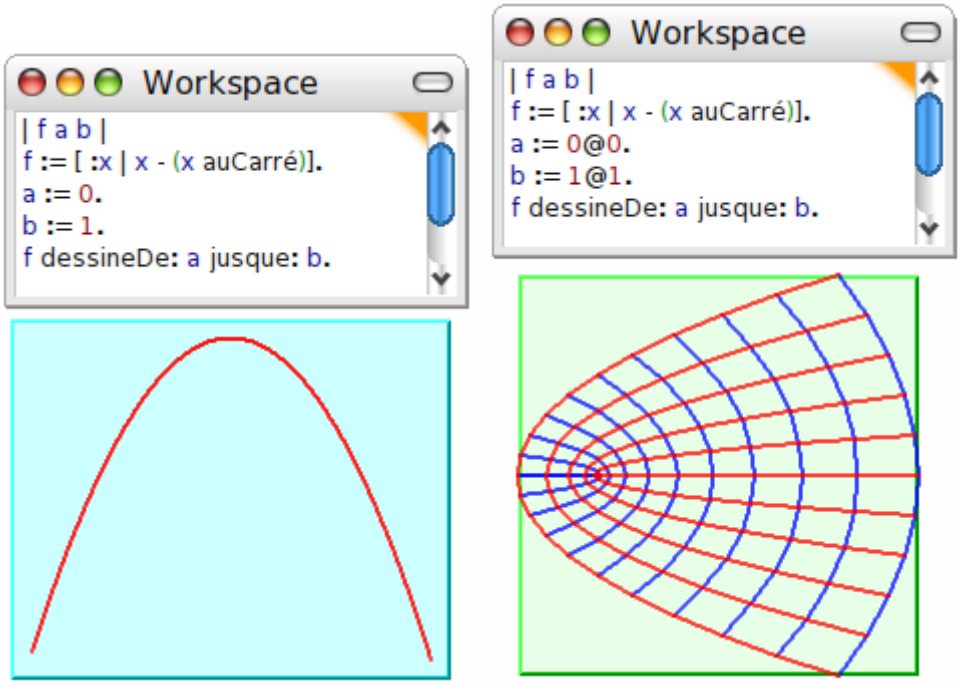

FIGURE 13.5 – En changeant le domaine de définition d'une fonction, on change beaucoup sa représentation graphique.

Chapitre 14

Matrices

> L'archipel de Passy-Ficka est formé de deux atolls: L'atoll de Passy comptant initialement 900 000 habitants, et l'atoll de Ficka comptant initialement 100 000 habitants. Chaque année, 20 % des habitants de Passy déménagent vers Ficka, et 60 % des habitants de Ficka émigrent vers Passy. Le ministre de l'intérieur de Passy-Ficka souhaite savoir quelles seront les conséquences à long terme de ces exodes entre atolls.

Dans *MathOntologie*, les matrices sont uniquement carrées d'ordre 2 [1].

14.1 Création et modification

14.1.1 Création

Pour créer une matrice, on écrit `Matrice new` :

```
| M |
M := Matrice new.
Transcript affiche: M.
```

L'affichage de la matrice montre que ses coefficients sont alors tous nuls. On peut les lire comme des variables appelées `M a11`, `M a12` etc.

1. en réalité ce sont des matrices rectangulaires de 2 lignes et 3 colonnes, mais la troisième colonne étant peu utilisée, n'est pas affichée : Tout se passe, sauf exception, comme si la matrice était effectivement carrée.

14.1.2 Modification

Pour changer la valeur du coefficient `a12` (par exemple pour le remplacer par un 5), il suffit de faire `M a12: 5`

Pour engendrer une matrice aléatoire, on peut donc faire comme ceci :

```
| M |
M := Matrice new.
M a11: (10 auHasard).
M a12: (10 auHasard).
M a21: (10 auHasard).
M a22: (10 auHasard).
Transcript affiche: M.
```

1. `M metAngleà: t` transforme `M` en la matrice $\begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix}$
2. `M metEchelleà: k` transforme `M` en la matrice $\begin{pmatrix} k & 0 \\ 0 & k \end{pmatrix}$
3. `M rendIdentité` fait la même chose que `M metEchelleà: 1`.

14.2 Fonctions

14.2.1 Arrondis

`M arrondieA: 0.001` donne une matrice identique à `M` sauf que ses coefficients sont arrondis à 0,001 près. De même, `M tronquéeA: 0.001` tronque les coefficients de `M`. `M tronquée` enlève tout ce qui est après la virgule dans les coefficients de `M`; de même, `M arrondieEntier` répond une matrice identique à `M` à ceci près que ses coefficients sont arrondis à l'entier le plus proche.

14.2.2 partie fractionnaire

Pour annuler la chiffres avant la virgule dans `M`, on écrit `M modulo: 1`. On peut réaliser cette opération `modulo` avec d'autres modules que 1, par exemple 12.

14.3 Opérations

14.3.1 Addition, soustraction et multiplication

Pour additionner, soustraire ou multiplier deux matrices, on les écrit avec, entre elles, le symbole "+", "-" ou "*".

On peut vérifier que la multiplication des matrices n'est pas commutative :

```
| M N |
M := Matrice new.
N := Matrice new.
M a11: (10 auHasard).
M a12: (10 auHasard).
M a21: (10 auHasard).
M a22: (10 auHasard).
N a11: (10 auHasard).
N a12: (10 auHasard).
N a21: (10 auHasard).
N a22: (10 auHasard).
Transcript affiche: (M*N).
Transcript affiche: (N*M).
```

On peut élever une matrice à une puissance entière (par exemple, 4) avec `M puissance: 4` ; dans le cas particulier où l'exposant est égal à 2 ou 3, on a aussi `auCarré` et `auCube`.

14.3.2 Inverse et division

Pour avoir l'inverse de la matrice `M`, on écrit `M inverse`. Pour avoir le produit de `M` par l'inverse de `N`, on écrit `M/N`.

14.4 Produit d'un vecteur par une matrice

14.4.1 Appliquer une matrice à un vecteur

Le produit de la matrice $M = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix}$ par le vecteur $\vec{u}\begin{pmatrix} x \\ y \end{pmatrix}$ est le vecteur $M\vec{u}\begin{pmatrix} a11 \times x + a12 \times y \\ a21 \times x + a22 \times y \end{pmatrix}$. On l'obtient avec `M transformeUnVecteur: u`.

Pour résoudre le problème directeur de ce chapitre, on peut écrire la matrice de Markov [2] du processus : $M = \begin{pmatrix} 0,8 & 0,6 \\ 0,2 & 0,4 \end{pmatrix}$ et le vecteur $\vec{v}$ décrivant la répartition initiale de la population, ramené à la populaion totale de 1 000 000 [3] ; chaque année, les populations des atolls de Passy et de Ficka seront calculées par un produit de la matrice et du vecteur des populations de l'année précédente :

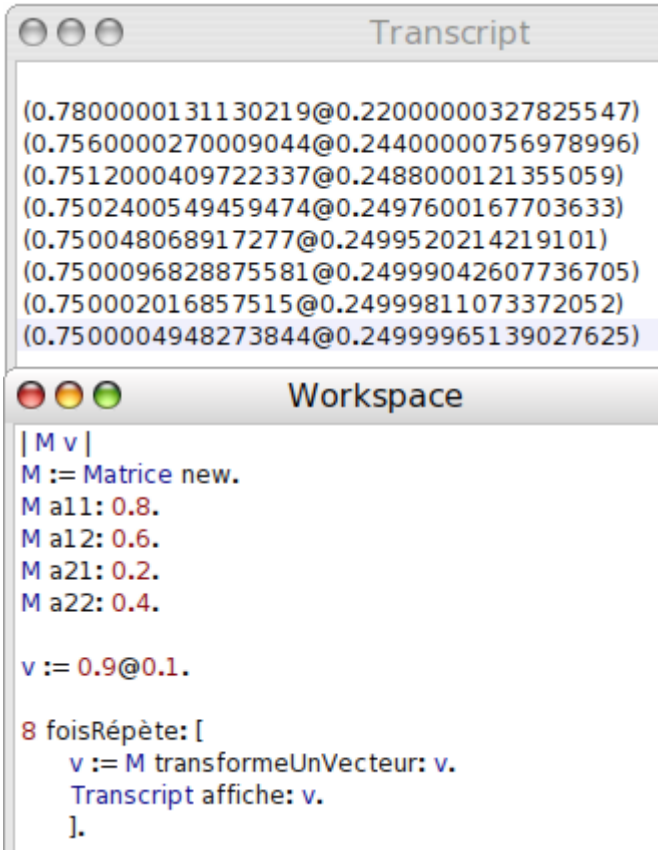

FIGURE 14.1 – On voit assez vite une probabilité stationnaire

Par exemple, au bout d'un an, l'atoll de Passy compte 780 000 habitants et l'atoll de Ficka en compte 220 000. Au bout de deux ans, les populations

2. Théoricien des probabilités, il est l'initiateur de la théorie des matrices stochastiques, et cet exercice est en réalité un exercice sur les probabilités conditionnelles, un peu déguisé.

3. c'est donc un vecteur de probabilités, de coordonnées $\begin{pmatrix} 0,9 \\ 0,1 \end{pmatrix}$

sont passées respectivement à 756 000 et 244 000. On constate numériquement qu'au bout de quelques années, les populations des deux atolls se stabilisent à 750 000 et 250 000 habitants respectivement.

14.4.2 Produit et décalage

Pour additionner un vecteur fixe au produit $M\vec{u}$, on utilise `M transformeUnPoint: u` (les coordonnées du vecteur fixe sont les coefficients invisibles de `M`, appelés `M a13` et `M a23`).

14.4.3 Résolution de systèmes

Le produit $M^{-1}\vec{u}$ s'obtient par `M inverséeTransformeUnPoint: u`. Par exemple, pour résoudre le système

$$\begin{cases} 3x - 2y &= -1 \\ x + y &= 8 \end{cases}$$

on peut écrire les seconds membres comme un vecteur $\vec{u}\begin{pmatrix} -1 \\ 8 \end{pmatrix}$ et les autres coeficients du système dans une matrice $M = \begin{pmatrix} 3 & -2 \\ 1 & 1 \end{pmatrix}$, puis

```
| M v |
M := Matrice new.
M a11: 3.
M a12: −2.
M a21: 1.
M a22: 1.
v := −1@8.
Transcript affiche: (M inverséeTransformeUnPoint: v).
```

On obtient la solution du système sous forme d'un vecteur.

Bien entendu, on a le même effet en appliquant `transformeUnVecteur: v` à l'inverse de `M`.

Pour le problème qui ouvre ce chapitre, une population stable $\begin{pmatrix} x \\ y \end{pmatrix}$ se caractérise par le fait qu'une fois multipliée par la matrice, elle reste la même [4], et que, comme c'est un vecteur de proportions, $x + y = 1$. On peut donc déterminer la population stable en résolvant le système

$$\begin{cases} 0,8x + 0,6y &= x \\ x + y &= 1 \end{cases}$$

4. Autrement dit, c'est un vecteur propre associé à la valeur propre 1

qu'on peut réécrire

$$\begin{cases} -0,2x + 0,6y &= 0 \\ x + y &= 1 \end{cases}$$

La proportion (0,75 ;0,25) semble bien en être la solution : Le problème

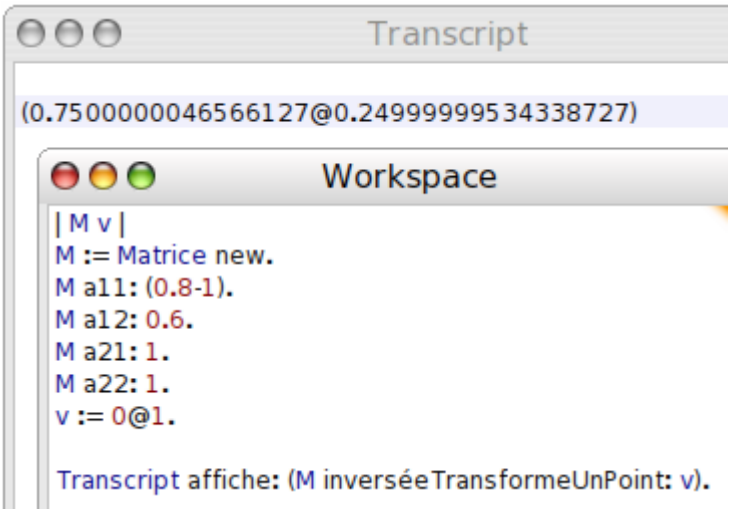

FIGURE 14.2 – On dirait que la proprtion est "trois quarts - un quart"

habituel d'affichage des arrondis se pose ici, mais on peut vérifier avec un calcul de fraction, que le vecteur $\left(\frac{3}{4}; \frac{1}{4}\right)$ est stable par la matrice :

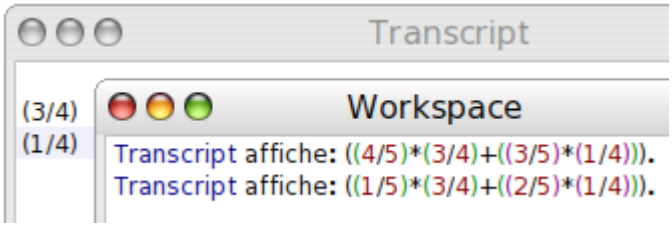

FIGURE 14.3 – C'est bien $\left(\frac{3}{4}; \frac{1}{4}\right)$

14.5 Algèbre linéaire

14.5.1 Déterminant

`M déterminant` renvoie le déterminant de `M` (le déterminant de $M = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix}$ étant égal à $\begin{vmatrix} a11 & a12 \\ a21 & a22 \end{vmatrix} = a11 \times a22 - a21 \times a12$)

14.5.2 Trace

`M trace` permet d'avoir la trace de la matrice `M` (la somme de ses éléments diagonaux)

14.5.3 Polynôme caractéristique

`M polynomeCaractéristique` renvoie le polynôme caractéristique de `M` (c'est une fonction). On peut calculer l'image d'un nombre par cette fonction, chercher quand elle s'annule, calculer sa dérivée etc. Par exemple, on peut la représenter graphiquement avec

```
| M N |
M := Matrice new.
M a11: (10 auHasard).
M a12: (10 auHasard).
M a21: (10 auHasard).
M a22: (10 auHasard).
(M polynomeCaractéristique) dessineDe: −4 jusque: 4.
```

14.5.4 Valeurs propres

Les racines réelles, si elles existent, du polynôme caractéristique de `M`, s'obtiennent avec `M valeursPropresRéelles` (c'est un ensemble contenant 0 à 2 nombres réels).