EuclidLib

un ensemble de routines
pour des constructions Euclidiennes
à l'aide d'Applescript

Guide d'utilisation

G. Coray

mai 2019

Table des matières

Introduction

EuclidLib est un ensemble de routines pour dessiner des figures dans le plan Euclidien - notamment celles que l'on peut construire avec la règle et le compas - visualisées dans une fenêtre graphique.

Les étapes d'une telle construction géométrique pourront être consignées dans un script, rédigé en *Applescript*, à l'aide d'instructions basées sur les primitives fournies par EuclidLib*.*
EuclidLib prévoit également la possibilité d'observer certaines propriétés telles que l'incidence, l'alignement ou la distance entre objets.

Les types d'objets manipulés sont le *point*, la *droite*, le *cercle,* le *segment* (une paire de points distincts) et le *polygone*, dont les cas particuliers du triangle et du rectangle. Accessoirement on aura aussi besoin aussi d'un *label* (texte positionné dans le plan) ou d'un *nombre*, pour exprimer un rapport, une distance ou un angle.

EuclidLib se présente comme un script compilé que l'on peut importer dans un script ou une application *Applescript*.
Chaque routine a un nombre fixe de paramètres positionnels, qui sont des objets (à l'exception des routines d'initialisation et de terminaison).
EuclidLib utilise pour cela l'application *Smile*, pilotable en *Applescript*, alors que des versions plus anciennes étaient basées sur *Photoshop*.

Ce guide d'EuclidLib est organisé en sept parties:

1. Le chargement, 2. Les initialisations, préliminaires incontournables.

3. Les objets, 4. Les constructeurs, la véritable matière géométrique.

5. Mesures et rapports, 6. La sauvegarde, pour l'observation des résultats.

7. Exemples, pour terminer.

1. Le chargement

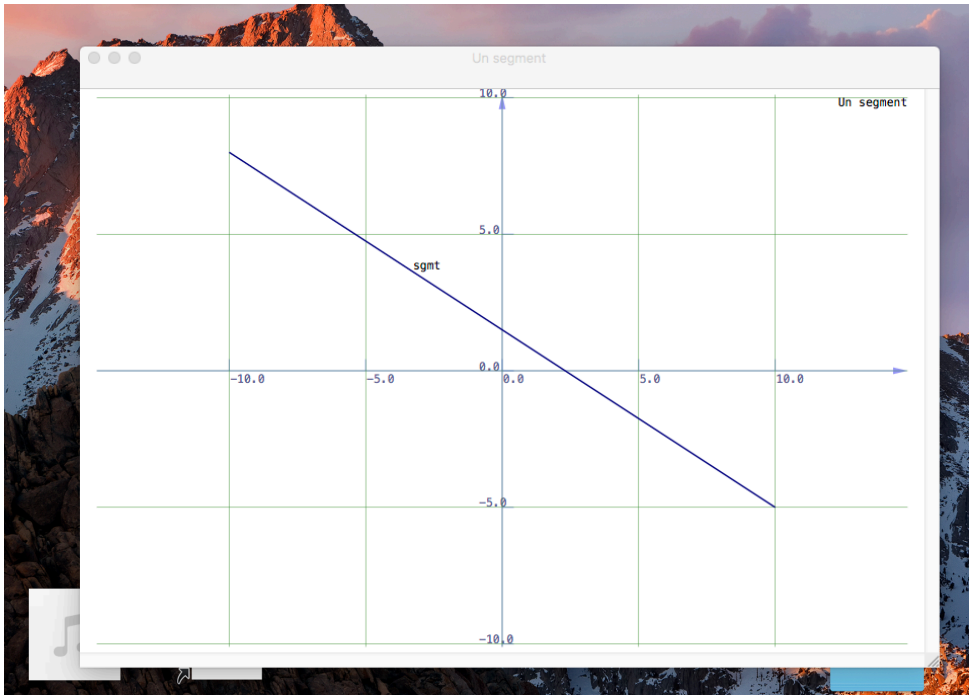
Les préliminaires résumés ici sont incontournables; en première lecture on peut toutefois passer à la suite.
1) Avant de commencer il faut en effet s'assurer que l'application *Smile.app* est installée (la télécharger gratuitement de http://www.satimage.fr/software/en/downloads/index.html) et que
2) le dossier "Macintosh HD:Library:Scripts:Libs" contient bien une copie de "EuclidLib.scpt" en mode compilé.
La figure que l'on désire "scripter" (nous dirons construire ou éditer) requiert le chargement de la bibliothèque *"EuclidLib.scpt"*. Celle-ci sera ainsi mise à disposition, sous un nom interne tel que lib, euclid ou euclidLib, du script édité. Cela s'énonce:

```
set euclidLib to  (load script file "Macintosh
                        HD:Library:Scripts:Libs:EuclidLib.scpt")
```

A partir de là toutes les routines exportées par "EuclidLib.scpt" sont accessibles dans la portée des instructions **tell** euclidLib.

Exemple schématique d'un script basé sur "EuclidLib.scpt" avec la fenêtre qu'il produit à l'écran:

```
set euclidLib to (load script file "Macintosh
      HD:Library:Scripts:Libs:EuclidLib.scpt")
tell euclidLib
      PlanEuclidien("Un segment")  -- crée le graphique, ouvre une fenêtre
      set A to randomPoint()        -- crée un point aléatoirement
      set B to {10,-5}              -- un second point en x=10, y=-5
      trace("sgmt",{A, B})          -- trace le segment AB
end tell
```



2. L'initialisation

Création du Plan Euclidien

planEuclidien(Titel)
C'est la première routine à appeler. Elle prépare un "plan Euclidien" pour y superposer les objets géométriques. En fait elle crée un graphique (objet de l'application *Smile*) et ouvre une fenêtre pour afficher son contenu. Toutes les autres routines de euclidLib se réfèrent implicitement à cette fenêtre. Le graphique est automatiquement muni d'un système de coordonnées cartésien *Oxy*, utilisé pour le positionnement des points par les routines ci-dessous.
Les dimensions cadrent bien avec une feuille A4 horizontale, l'origine étant placée au centre. On travaille en *cm*, les coordonnées x et y peuvent donc varier de -14.5 à +14.5 environ pour x et de -10.5 à +10.5 pour y. Imprimée sur un feuille A4 (orientée en mode paysage) la figure affichée dans la fenêtre occupera exactement la feuille imprimée.

Les paramètres de PlanEuclidien

Le paramètre Titel est le nom porté par la fenêtre du tracé. Il est également affiché à l'angle supérieur droit dans la fenêtre.
Note: cela offre la possibilité d'ouvrir plusieurs fenêtres de nom différent; les routines de euclidLib n'accèdent toutefois qu'à la dernière créée.

Les paramètres d'échelle: xmin, xmax et ymedian

Ces paramètres optionnels peuvent influencer le repérage.
xmin et xmax sont des *variables globales* qui peuvent être modifiées, avant l'appel à PlanEuclidien, si on veut se départir des valeurs par défaut -14.5 et +14.5 fixées par EuclidLib.
On choisira de préférence xmax > xmin suffisamment espacés puisque xmax-xmin détermine l'échelle à laquelle on pourra lire le graphique.
De même la variable globale ymedian, qui repère la mi-hauteur de la fenêtre, est initialisée par défaut à 0; elle peut être affectée d'une autre valeur si l'on désire placer l'origine ailleurs qu'au centre. Pour être effective, l'affectation doit avoir lieu *avant* l'appel à PlanEuclidien.
Les paramètres optionnels suivants précisent le "décor" du graphique.

Les paramètres optionnels: marksOx et marksOy

EuclidLib dessine automatiquement le repère cartésien et place des repères équidistants sur chaque axe. On peut préciser le nombre de repères souhaité sur l'axe horizontal au moyen de la variable globale marksOx (et de même avec marksOy pour l'axe vertical), sachant que les valeurs numériques effectivement marquées seront des multiples d'une puissance de 10 adéquate.
Sa valeur par défaut fixée par EuclidLib est nulle, elle peut être changée par une affectation avant l'appel à PlanEuclidien.
Note: La valeur 0 signifie que l'on renonce au tracé des axes.

Les paramètres optionnels: rulesOx et rulesOy

EuclidLib dessine un quadrillage du plan, avec un nombre de des filets (rules) équidistants, cohérent avec les marques sur les axes. On peut préciser le nombre souhaité de filets du fond quadrillé, au moyen de la variable globale rulesOx (et de même pour rulesOy), sachant que les espaces entre filets seront, à l'instar des valeurs portées par les axes, des multiples d'une puissance adéquate de 10.
Par exemple, si les valeurs des rules est identique à celle des marks, EuclidLib fera passer les filets par les marques sur les axes.
En revanche, l'affectation **set** {rulesOx,rulesOy} **to** {29, 20} produirait une grille centimétrique (29~xmax-xmin).
Pour être effective, l'affectation de rulesOx, rulesOy doit être placée *avant* l'appel à PlanEuclidien.
Note: Une valeur nulle (par défaut) signifie que l'on renonce à une grille de filets.

Les constantes exportées

L'appel à PlanEuclidien a aussi pour effet d'initialiser les constantes xmin, xmax, ymin, ymax, pica et dtp. Elles fournissent l'information exacte des dimensions dans la page, notamment de l'échelle à laquelle on travaille.
Note: Par défaut xmin, xmax valent -14.5, +14.5 respectivement alors que ymin, ymax sont -10.5, 10.5 environ[1].

Les constantes typographiques pica, dtp sont utilisées par la routine label ci-dessous. Elles permettent de positionner du texte (légendes, titre) voire de disposer les figures en fonction du texte produit par la routine label.
Note: pica est la hauteur standard d'un caractère en *corps 12*, alors que dtp vaut un *point* (standard *Digital Typography Point*, 1/12 de pica) soit env. 0.3 mm.

Le changement d'échelle

Un changement d'échelle est parfois souhaitable en cours de dessin pour une meilleure visibilité.

zoom

zoom(Pfixe, ratio) remplace le système de coordonnées existant par un autre, d'échelle multipliée par ratio et tel que le point Pfixe se retrouve au même endroit. Bien entendu xmin, xmax, ymin et ymax seront adaptées au nouveau repère.

---

[1] D'autres variables globales penWidth, pencolor, segmentwidth, segmentcolor, fillcolor, markwidth, markcolor, labelsize, labelcolor, backGroundColor et textBackGround sont utilisées pour le tracé. Cf. appendice sur les couleurs.
Enfin, infini est une constante exportée par PlanEuclidien, elle sert de valeur factice pour la pente de droites verticales.

3. Les objets

Les objets qu'on peut construire, puis utiliser dans la construction d'autres objets, pour former des figures dans le plan euclidien sont soit
- le *point*,
- la *droite*
- le *cercle*
- le *segment*
- le *polygone*, soit une liste d'au mois trois sommets
- un élément *simple*, c'est-à-dire un nombre, l'infini ou un texte
- une *liste*, non vide, à l'exception de polygones, ou enfin
- l'ensemble *vide*, par exemple l'intersection de deux cercles disjoints

Nature d'un objet

Seuls les points, droites, cercles et polygones sont des objets *géométriques* à proprement parler, les autres étant des auxiliaires.
Pour un objet donné la fonction suivante détermine son type ou sa *nature:*

nature(Objet) retourne une des valeurs suivantes: "Point", "Line", "Circle", "Segment", "Polygon" , "Simple", "List", "Empty" selon l'Objet passé en paramètre. Notez l'othographe anglaise (sorry ;-).

Bien qu'en principe euclidLib ne doit pas révéler la représentation *interne* de ses objets, nous n'allons pas faire mystère de la convention suivante: un point P de coordonnées x, y est représenté par la paire de coordonnées: {x,y}. Ainsi, après
set P **to** {0,0} -- l'origine !
on a nature(P) = "Point", alors que nature(x) = nature(y) = "Simple".

Par contre la représentation *externe*, graphique, des objets est primordiale; on l'obtient par l'une des routines traceObj ou trace suivantes.

Tracer un objet

traceObj(Objet) dessine, pour un Objet géométrique donné:
- un minuscule cercle si Objet est un *point*
- un trait fin, rectiligne si Objet est une *droite*, circulaire si c'est un *cercle*, dans les limites de la fenêtre créée par PlanEuclidien
- une ligne brisée, plus épaisse, pour un *polygone*[2] ou un *segment*
- chaque élément de la *liste*, si Objet est une *liste* d'objets géométriques

Paramètres optionnels

opacite, comprise entre 0 et 1, permet d'estomper voire inhiber le tracé de l'Objet, par exemple si ce dernier est une construction auxiliaire.

---

[2] Un polygone *fermé* aura, en outre, l'intérieur "rempli" d'une teinte orange claire.

Au besoin l'épaisseur du trait et sa couleur peuvent également être influencées par les variables globales suivantes.
- penWidth et penColor contrôlent resp. l'épaisseur et la couleur des cercles et droites.
- segmentWidth et segmentColor contrôlent resp. l'épaisseur et la couleur des segments et polygones. De plus, fillcolor indique la couleur de remplissage des triangles et autres polygones fermés.
- Enfin, markwidth et markcolor régissent la représentation d'un point comme un petit cercle de diamètre markwidth.
- L'unité pour les epaisseurs est le point *dtp* (soit env. 1/3 mm)

Nommer un objet

label(nom, Objet) appose le texte nom à proximité du dessin de l'Objet.

Paramètres optionnels

Les variables globales labelColor et labelSize permettent de préciser l'aspect du nom écrit par label (ainsi que par trace ci-dessous).
labelColor, normalement *noir*, définit la couleur[3] du nom.
labelSize, normalement *12*, définit le corps des caractères du nom.

Note: On peut aussi utiliser label sans que l'objet soit tracé, par exemple à côté d'un point virtuel. Ainsi une légende[4] "LEGENDE" peut être placée dans l'angle inférieur gauche de la fenêtre au moyen de:

```
label("LEGENDE", {xmin,ymin})
```

Attention toutefois aux caractères exceptionnels "_", "^" et "\" . "_" est remplacé par "-" alors que "^" est le code pour mettre le caractère suivant en exposant. Enfin "\" est le caractère d'échappement usuel comme dans "\"" pour le guillemet. Ainsi $y=x^2$ pourra être placé auprès d'une parabole (donnée par une liste de points adéquate) au moyen de

```
label("y = x^2", parabole)
```

Tracer et nommer un objet

Pour simplifier, on peut combiner en une seule instruction le dessin de l'objet et le label qu'on lui destine, c'est le rôle de trace.
trace(nom, Objet) dessine l'Objet et place son nom à proximité du tracé. Par exemple trace("Origine", {0,0}) dessine un minuscule cercle à droite du point {0,0} accompagné de l'étiquette "Origine".

Note: trace("", Objet) revient au même que traceObj(Objet).

---

[3] Cf. appendice pour les couleurs. textBackGround (paramètre couleur optionnel) est transparent par défaut. Si sa valeur est différente, label va effacer et colorer le fond de l'espace occupé par le texte nom, afin de rendre ce dernier plus lisible.

[4] Signalons à ce propos la routine legend(message) qui affiche le message à la marge (gauche) du graphique, à la suite des lignes précédentes, depuis le haut de la fenêtre.

4. Les constructeurs

Avec la construction d'objets géométriques on entre dans le vif du sujet. Chaque objet construit peut être mémorisé à l'aide d'une variable, par une affectation, ou passé en paramètre à une routine de construction. Commençons par les points, objets géométriques de base, pour enchaîner avec les droites, notamment la droite passant par deux points donnés, distincts.

Les points

Point de coordonnées x, y

Construire un point à partir de ses coordonnées x et y est simple: on utilise sa représentation interne {x, y} telle quelle.

Note: Dans le même esprit un *segment* dont les extrémités sont les points A, B sera construit en formant la paire {A, B} de ces points. Plus généralement un *polygone* est une liste d'au moins trois points. Mais revenons aux diverses façons de construire *un* point.

Point aléatoire

randomPoint() livre, à chaque appel, un nouveau point de coordonnées pseudo-aléatoires. La routine mémorise les points précédemment générés afin de s'en distancer et favoriser des configurations en position générale.

Point d'intersection

intersection(O1, O2) est le point d'intersection entre les objets O1 et O2 lorsque O1 et O2 sont deux droites non parallèles ou un cercle et une droite tangente au cercle.
Cependant le résultat de l'intersection n'est pas toujours un point. Le résultat est {} si les deux objets ne sont pas incidents. Dans le cas de deux cercles, ou d'une droite et d'un cercle, incidents le résultat est une paire de points (donc un segment: la corde).

Point extrémité

extremite(segm) choisit une des deux extrémités d'un segment segm donné. Le choix favorisera l'extremité la plus éloignée des précédentes. Ainsi deux appels identiques extremite(segm) consécutifs fournissent les deux extrémités distinctes du segm.
extremite(segm) est intéressant si segm est le résultat d'une intersection; extremite choisira alors d'abord le point d'intersection nouveau en évitant ainsi la répétition d'un ancien.

Les droites

droite par deux points

droite(P0, P1) construit la droite passant par les points P0 et P1. On

pourra la tracer à l'aide de traceObj(droite(P0, P1)) ou la garder à disposition au moyen de l'affectation à une variable d:

set d **to** droite(P0, P1)

ou encore la nommer "Droite d" en combinant l'affectation avec le tracé:

set d **to** trace("Droite d", droite(P0, P1)).

parallèle à une droite, par un point

parallele(d, P) construit la parallèle à d, passant par le point P. En géométrie Euclidienne du plan elle est déterminée univoquement par la donnée de d et P. Ici encore on peut mémoriser le résultat par une affectation, tracer la droite obtenue, la nommer ou l'utiliser dans une autre construction.

perpendiculaire à une droite

perpendiculaire(d, P) construit la perpendiculaire à d, abaissée du point P. Le résultat est une droite passant par P et formant un angle droit avec d. C'est un objet géométrique de nature "Line" que l'on peut utiliser comme paramètre d'une autre construction dans le script.

support et mediatrice d'un segment

support(segm) produit la droite qui prolonge le segment donné, dans les deux directions, c'est à dire son support. Une autre droite, qui lui est perpendiculaire est fournie par la:
mediatrice(segm) ou la droite dont les points sont à égale distance des extrémités du segment segm donné.

Les cercles et polygones fermés

Cercle

Cercle(C, P) construit le cercle centré en C et passant par le point P donné.

CerclePar3points

CerclePar3points(A, B, C) construit le cercle circonscrit au triangle A,B,C. Il pourraît aussi être obtenu à l'aide des médiatrices des côtés dont l'intersection donne notoirement le centre du cercle circonscrit.

Triangle

Triangle(A,B,C) le triangle résultant est un polygone fermé[5]

Rectangle

Rectangle(diagonale) construit le rectangle dont les côtés sont parallèles aux axes et deux sommets diagonalement opposés sont donnés par le segment diagonale.

---

[5] Un polygone fermé est un objet géométrique distinct de la liste de ses sommets. Pour EuclidLib, un polygone est *fermé* ssi son dernier sommet coïncide avec le premier. Triangle et Rectangle sont des commodités qui se conforment à cette convention en manipulant pertinemment les listes de points.

5. Mesures et rapports

La géométrie euclidienne permet de munir les objets de propriétés métriques, telles que l'aire d'un triangle, l'angle entre deux droites ou la distance entre deux points.

Compare

Il s'agit d'un outil passe-partout qui, étant donné deux objets Obj1, Obj2 produit un diagnostic, affiché en marge du graphique, rappelant les noms Nom1, Nom2 et la relation entre les deux objets.
La relation est obtenue à l'aide des fonctions ci-dessous et concerne la similitude, l'égalité, l'incidence, la distance, rapport ou angle entre les deux objets, selon leur nature.
On peut ainsi observer les propriétés remarquables des figures construites dans le plan Euclidien.

compare(Nom1, Obj1, Nom2, Obj2) affiche un message où figurent les noms Nom1, Nom2 et un commentaire adéquat caractérisant la relation entre Obj1 et Obj2.

Distances

distance(P0, P1) est la distance entre deux points, exprimée en *cm* ou dans les unités du système cartésien sous-jacent. La routine a été généralisée aux objets P0, P1 suivants:
- distance d'un point à une droite ou à un cercle
- distance entre deux cercles, entre droite et cercle et même entre deux droites parallèles

Note: La distance d'un point à une liste est définie comme le *minimum* des distances à chaque élément de la liste et comprend aussi les cas du segment ou du polygone.

Angles

angle(d0, d1) est l'angle de la droite d1 relativement à d0, exprimé en degrés et ramené (modulo 180) à une valeur comprise entre 0 et 180.
La routine est étendue aux segments, remplacés par leur support.

Rapports

Rapport de similitude

similitude(Obj0, Obj1) donne le *rapport de similitude* pour deux objets Obj0, Obj1 s'ils sont semblables: triangles, rectangles ou polygones. Par contre, si Obj0, Obj1 ne sont pas semblables le résultat est 0.

Cas particuliers.
Le rapport de similitude vaut 1 ssi les objets sont *isométriques*. D'autre part tous les segments sont semblables entre eux, de même que tous les cercles. Le rapport de similitude revient alors au rapport de leur taille.

Rapport d'interpolation

rapport(P0, P1, P) est le rapport entre le segment {P0, P} et {P0, P1} lorsque les trois points sont *alignés*.
Ce rapport est compris entre 0 et 1 si P se trouve sur le segment {P0, P1}; mais il peut être négatif (si P0 sépare P et P1) ou supérieur à 1 (si P1 sépare P0 et P); on parle alors *d'extrapolation*.
Note: plus généralement si les trois points ne sont pas alignés P est remplacé par sa projection sur le support de {P0, P1}. Toutefois le rapport vaut infini si P0 = P1.

Interpolation

interpolation(P0, P1, w) fournit, à l'inverse, le point P dont le rapport(P0, P1, P)=w. C'est le point d'interpolation entre P0 et P1, obtenu par combinaison linéaire pondérée de poids 1-w pour P0 et w pour P1.

Note: *L'interpolation* est une routine très générale, utile par ex. pour construire des figures homothétiques ou pour un dégradé de couleurs comprises entre deux RGB donnés.

Aires

aire(Obj) est l'aire de l'Obj, cercle, triangle, rectangle ou polygone.
Elle est exprimée en *cm carrés,* sauf modification des unités des coordonnées. Le résultat est un nombre réel, positif si le polygone est *simple* (sans croisements).

Note: si Obj est un polygone non fermé le côté manquant sera ajouté. Le résultat est nul pour un point, une droite ou un segment.

6. La sauvegarde

Les routines de service saveinfile et diapo facilitent la sauvegarde des constructions graphiques créées via *Smile.*

saveInFile

La sauvegarde du graphique dans un fichier peut se faire directement par saveinfile:
saveInFile(filePath) conserve l'état actuel du graphique, créé à l'aide des routines précédentes, dans un fichier de type .pdf, .jpg, .tiff ou .png.

Le **paramètre** filePath désigne le fichier destination et doit avoir la forme d'un chemin d'accès (voire d'un alias) connu par l'application *Finder*.
Le suffixe .pdf, .jpg, .tiff ou .png décidera du type de fichier produit.
En l'absence de suffixe, .pdf est automatiquement ajouté. Cette valeur par défaut est recommandée parce que le fichier obtenu est léger et la définition excellente pour le graphique vectoriel produit.
Note: l'option .png pour le suffixe a l'avantage de conserver les parties transparentes du graphique, ce qui peut être utile par ex. pour une incrustation ou dans les animations sur fond fixe.
Une abréviation pratique: si filepath se résume à un identificateur (ne contenant pas de ":") la destination choisie est le bureau.
Par ex. saveInFile("Cercle") crée ou met à jour le fichier "Cerle.pdf" sur le bureau.

Diapos et Diapo

Diapo() et Diapos(folderpath) servent à conserver plusieurs étapes de la construction, destinées par exemple à une présentation par diapositives.

Diapo

Diapo() ajoute un fichier à la collection, numéroté automatiquement, dans l'ordre chronologique. Le dossier recevant la collection doit être créé au préalable à l'aide de Diapos(folderpath).
Les appels à Diapo utiliseront alors, pour chaque fichier stocké, le nom du dossier complété par un numéro.

Diapos

Diapos(folderpath) doit être appelé avant les appels à Diapo, son paramètre principal désignant le dossier destination.
Une fenêtre de dialogue s'ouvrira alors pour permettre à l'utilisateur de modifier le choix du dossier, d'ajouter les nouvelles diapos à un dossier déjà existant ou encore de renoncer à enregistrer des diapos.

Paramètre optionnel

Les diapos stockées lors des appels à Diapo() sont des fichiers de type *.png* par défaut. Au besoin, ce choix peut être modifié par une affectation à la variable globale DiapoFileExtension.

7. Exemples

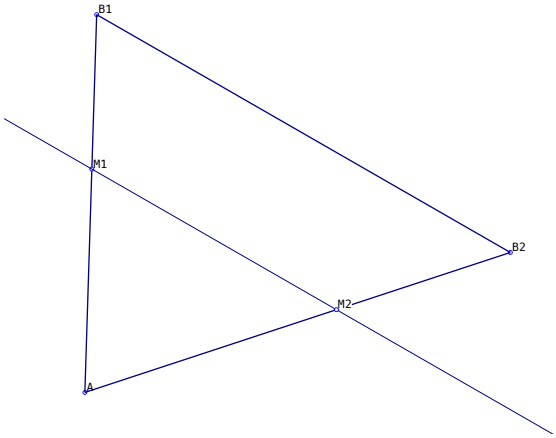
7.1 Thalès

On veut vérifier la propriété, attribuée à Thalès, que deux segments A,B1 et A,B2 issus du même point A sont coupés dans les mêmes proportions par une parallèle à B1,B2.
On utilise la fonction compare aussi bien pour l'égalité des rapports que pour la similitude des deux triangles de la figure.

```
set euclid to load script file "Macintosh HD:Library:Scripts:Libs:euclidLib.scpt"
tell euclid
	planEuclidien("Thales")
	set A to trace("A", randompoint())	-- choisit un point A aléatoirement
	set B1 to trace("B1", randompoint()) -- choisit un point B1 aléatoirement
	set B2 to trace("B2", randompoint()) -- choisit un point B2 aléatoirement
	traceObj({{A, B1}, {B1, B2}, {B2, A}}) -- dessine le triangle A,B1,B2
	-- M1: point aléatoire sur {A,B1}, M2: id. reporté sur {A,B2}
	set M1 to trace("M1", Interpolation(A, B1, random number))
	set p to traceObj(parallele(support({B1, B2}), M1))
	set M2 to trace("M2", intersection(p, {A, B2}))
	compare("rapport(A, B1, M1)", rapport(A, B1, M1),
			"rapport(A, B2, M2)", rapport(A, B2, M2))
	compare("triangle(A,B1,B2)", triangle(A, B1, B2),
			"triangle(A,M1,M2)", triangle(A, M1, M2))
end tell
```

Thales

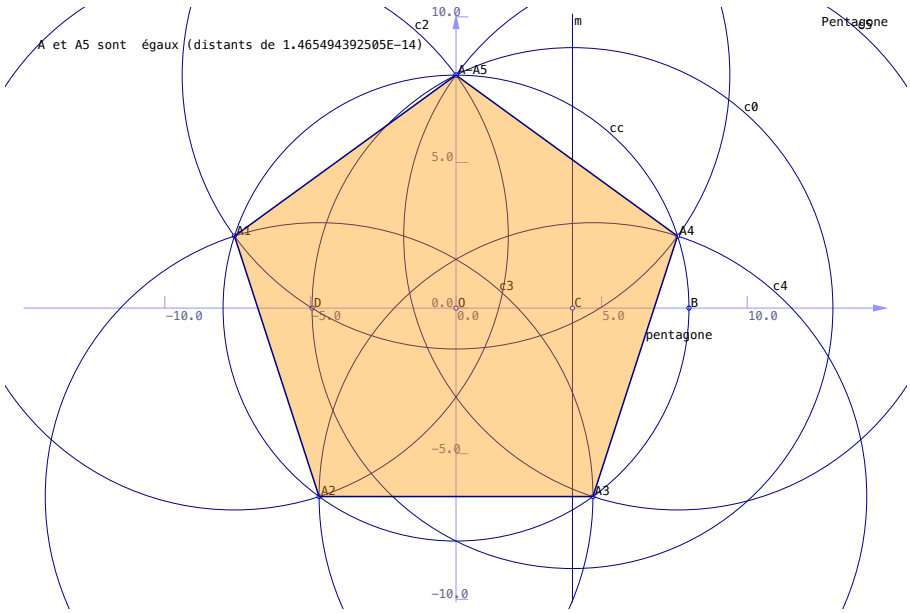
```
rapport(A, B1, M1) et rapport(A, B2, M2) sont égaux (différents de 2.109423746788E-15)
triangle(A,B1,B2) et triangle(A,M1,M2) sont semblables: rapport = 0.591240955411
```



7.2 Pentagone régulier

On commence par exécuter la construction du pentagone inscrit dans le cercle. On constate alors que le polygone est fermé, c'est donc un pentagone régulier.

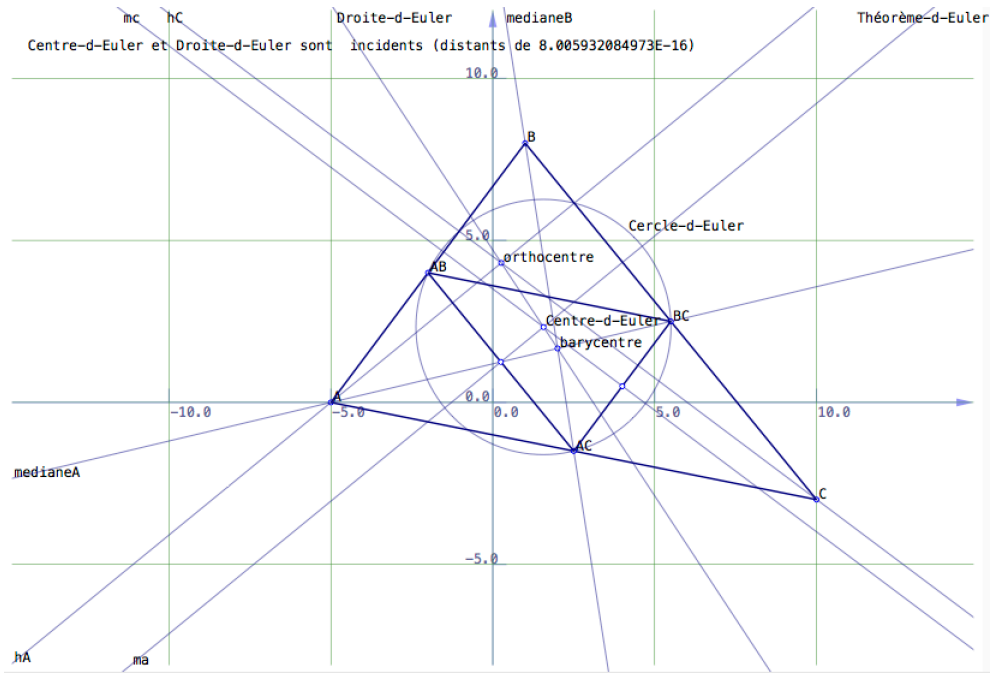
```
set Euclid to (load script file "Macintosh
        HD:Library:Scripts:Libs:EuclidLib.scpt")
set {marksOx, marksOy} to {4,3}
tell Euclid
    planEuclidien("Pentagone")
    set r to 8
    set O to trace("O", {0, 0}) -- origine  = centre
    set A to trace("A", {0, r})  -- rayon vertical
    set cc to trace("cc", cercle(O, A)) -- futur cercle circonscrit
    set B to trace("B", {r, 0})  -- rayon horizontal
    set OB to {O, B}             -- segment horizontal
    set m to trace("m", mediatrice(OB))
    set C to trace("C", intersection(m, OB))
    set c0 to trace("c0", cercle(C, A))
    Extremite(B)
    set D to trace("D", Extremite(intersection(c0, OB)))
    set c1 to trace("c1", cercle(A, D))
    set A1 to trace("A1", Extremite(intersection(c1, cc)))
    set c2 to trace("c2", cercle(A1, A))
    Extremite(A) -- élimine A comme extrémité
    set A2 to trace("A2", Extremite(intersection(c2, cc)))
    set c3 to trace("c3", cercle(A2, A1))
    set A3 to trace("A3", Extremite(intersection(c3, cc)))
    set c4 to trace("c4", cercle(A3, A2))
    set A4 to trace("A4", Extremite(intersection(c4, cc)))
    set pentagone to {A, A1, A2, A3, A4, A}
    set c5 to trace("c5", cercle(A4, A3))
    set A5 to trace(" -A5", Extremite(intersection(c5, cc)))
    compare("A", A, "A5", A5)
    trace("pentagone", {A, A1, A2, A3, A4, A5, A})
end tell
```



7.3 Droite et cercle d'Euler

On veut vérifier le théorème d'Euler, soit l'alignement de *l'orthocentre*, du *barycentre* et du *centre du cercle d'Euler* dans un triangle. Le script suivant illustre les trois points sur la *droite d'Euler*.

```
set euclidLib to load script file "Macintosh
            HD:Library:Scripts:Libs:euclidLib.scpt"
tell euclidLib
      set ymedian to 2
      PlanEuclidien("Théorème_d_Euler")
      set A to trace("A", {-5, 0})
      set B to trace("B", {1, 8})
      set C to trace("C ", {10, -3})
      trace("triangle", {{A, B}, {B, C}, {C, A}})
      -- orthocentre:
      set hA to trace("hA", perpendiculaire({B, C}, A))
      set hC to trace("hC", perpendiculaire({A, B}, C))
      set orthocentre to trace("orthocentre", intersection(hA, hC))
      -- triangle median:
      set AB to trace("AB", intersection({A, B}, mediatrice({A, B}))) --
                  =interpolation(A, B, 1/2)
      set AC to trace("AC", intersection({A, C}, mediatrice({A, C})))
      set BC to trace("BC", intersection({B, C}, mediatrice({B, C})))
      -- centre de gravité:
      set medianeA to trace("medianeA", droite(A, BC))
      set medianeB to trace("medianeB", droite(B, AC))
      set barycentre to trace("barycentre", intersection(medianeA, medianeB))
      set Droite_d_Euler to trace("Droite_d_Euler", droite(barycentre,
                  orthocentre))
      -- cercle d'Euler:
      set opposeDeAB to traceObj({AC, BC}) -- parallèle à AB
      set opposeDeBC to traceObj({AC, AB}) -- parallèle à BC
      traceObj({BC, AB}) -- parallèle à AC, pour l'esthétique

      set mc to trace("mc", mediatrice(opposeDeAB))
      set ma to trace("ma", mediatrice(opposeDeBC))
      traceObj(intersection(mc, opposeDeAB))
      traceObj(intersection(ma, opposeDeBC))
      set centre_d_Euler to trace("Centre_d_Euler", intersection(ma, mc))
      set Cercle_d_Euler to trace("Cercle_d_Euler", Cercle(centre_d_Euler, AC))
      -- test d'alignement (théorème d'Euler):
      compare("Centre_d_Euler", centre_d_Euler, "Droite_d_Euler",
                  Droite_d_Euler)
end tell
```

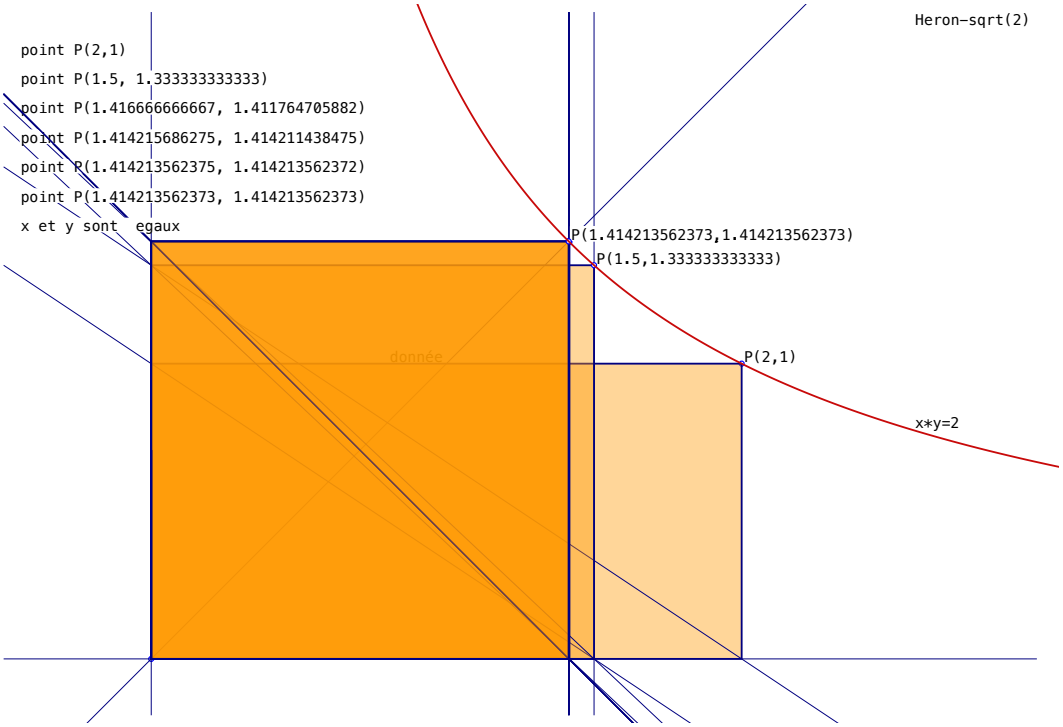


7.4 Le calcul de la racine carrée selon Heron

```
set a to 2 -- la méthode de Heron s'applique à la racine carrée de tout a>1
set Euclid to (load script file "Macintosh
            HD:Library:Scripts:Libs:EuclidLib.scpt")
set Titre to "Heron-sqrt(" & a & ")"
set {xmin, xmax, ymedian} to {-0.5, a + 1, 1} --repère cartésien contenant a
tell Euclid
      Diapos("", Titre) -- on desktop
      planEuclidien(Titre)

      set origine to traceObj({0, 0})
      set Ax to traceObj({0, origine})
      set Ay to traceObj({infini, origine})
      set diagonale to traceObj({1, origine})
      --- sommets du rectangle initial
      set x to a -- as integer -- solves bug
      set y to 1
      set P to trace("P(" & x & "," & y & ")", {x, y})
      trace("donnée", Rectangle({origine, P})) -- remplissage orange
      legend("point P(" & x & "," & y & ")")
      Diapo()
      set B to intersection(Ax, parallele(Ay, P)) -- projection de P sur Ox
      set C to intersection(Ay, parallele(Ax, P)) -- projection de P sur Oy
      --- iteration c.à.d. construction de rectangles de même aire x*y=a
      repeat
            set horizontale to parallele(Ax, P)
            set HD to intersection(horizontale, diagonale)
            set verticale to traceObj(mediatrice({HD, P})) --entre pt diagonal et P

            set D to intersection(Ax, verticale) -- projection du pt. médian sur Ox
            set DC to traceObj(droite(D, C))
            Diapo()
            set EB to traceObj(parallele(DC, B))
            set E to intersection(EB, Ay)
            Diapo()
            set horizontale to parallele(Ax, E) -- nouvelle horizontale
```

```
            set P to intersection(horizontale, verticale) -- ça c'est le nouveau pt. P
            set r to traceObj(Rectangle({{0, 0}, P}))
            set {x, y} to P
            trace("P(" & x & "," & y & ")", P) -- écrit "P(2,1)" si x=2 et y=1
            legend("point P(" & x & ", " & y & ")")
            Diapo()
            -- terminer l'itération ?
            if (abs (x - y)) < 1.0E-12 then -- précision d'au moins 1E-12
                  compare("x", x, "y", y)
                  exit repeat
            end if
            set B to D
            set C to E
      end repeat
      -- lieu des points P = hyperbole y*x=a:
      set segmentcolor to "R"
      trace("x*y=" & a, my hyperbole(a, xmax)) -- polygone approx.
      Diapo()
end tell

on hyperbole(a, xmax) -- points de  y = a/x, pour x entre 1/xmax et xmax
      set courbe to {}
      repeat with i from 1 to 300
            set x to (1 / xmax + i * xmax / 300) as real
            set y to a / x
            set end of courbe to {x, y}
      end repeat
      return courbe -- polygone d'approximation
end hyperbole
```

Sur la dernière diapo ci-dessous on remarque trois itérations oblitérées (visibles sur les diapos intermédiaires):



Appendice: la notation pour les couleurs

Les variables globales backGroundColor, pencolor, segmentcolor, fillcolor, markcolor, labelcolor, et textBackGround sont utilisées respectivement pour le fond, le tracé des objets et l'apparence du texte (labels et légendes). Leurs valeurs sont assignées par défaut et peuvent donc être ignorées. Mais elles peuvent être modifiées par une simple affectation. Par exemple, la couleur du fond est "blanc" par défaut mais peut être choisie "noire", comme pour tracer à la craie sur un tableau, au moyen de l'affectation suivante (qui doit précéder l'appel de planEuclidien):

```
set backGroundColor to "N"  -- fond noir
```

La notation pour les couleurs est soit le code héxadécimal ou RGB standard, soit plus simplement une lettre selon le tableau ci-dessous:

"R"	rouge
"G"	vert (Green)
"B"	bleu
"Y"	jaune (Yellow)
"C"	cyan
"M"	magenta
"O"	orange
"I"	violet (Indigo)
"T"	turquoise
"N"	noir
"W"	blanc (White)

De plus on a les conventions suivantes:

"-"	gris
" "	transparent
"*"	couleur aléatoire