Travaux dirigés avec SAGE (partie I)

Math 3 — Année 2010-2011

Sommaire

1 Prise en main **1**
- 1.1 Feuille de calcul . . . 1
- 1.2 Assignation . . . 2
- 1.3 Aide en ligne . . . 3
- 1.4 Expressions symboliques . . . 3
- 1.5 Types composés . . . 4
  - 1.5.1 Définitions . . . 4
  - 1.5.2 Construction des listes . . . 4
- 1.6 Programmation . . . 5

2 Problèmes résolus **5**
- 2.1 Calcul d'une limite . . . 5
- 2.2 Approximation de la tangente . . . 5
- 2.3 Champ newtonien . . . 6

3 Séries numériques **7**
- 3.1 Convergence . . . 7
- 3.2 Calcul de sommes . . . 7
- 3.3 Exercices . . . 8

4 Suites de fonctions **8**
- 4.1 Visualisation . . . 8
  - 4.1.1 Graphes superposés . . . 8
  - 4.1.2 Animation . . . 8
  - 4.1.3 Graphisme interactif . . . 9
- 4.2 Limite simple . . . 9
- 4.3 Convergence uniforme . . . 9
- 4.4 Exercices . . . 10

1 Prise en main

Connectez-vous à l'adresse `http://sage-math.univ-lyon1.fr`*. Après vous être identifié, cliquez sur le lien "New Worksheet" : une feuille de calcul vierge apparaît. Cliquez sur son nom "Untitled" et renommez-la par exemple "TD1". En fin de séance, n'oubliez pas de sauvegarder votre travail.*

1.1 Feuille de calcul

Elle se présente sous la forme de cellules dans lesquelles vous taperez vos commandes. Entrez par exemple l'expression :

```
cos(pi/12)
```

puis validez par les touches `<MAJ><ENTREE>`.

La valeur de l'expression apparaît sous forme littérale (`sqrt` désigne la racine carrée). Pour avoir une approximation numérique avec 10 chiffres décimaux, entrez dans la cellule suivante :

```
cos(pi/12).n(digits=10)
```

puis validez.
A tout moment, on peut :
– insérer une nouvelle cellule : placer la souris entre deux cellules et cliquer sur la ligne bleue qui apparaît,
– supprimer une cellule : effacer d'abord son contenu puis presser la touche `<DELETE>`.
On peut aussi valider une cellule par `<CTRL><ENTREE>`, ce qui a pour effet d'insérer une cellule vide juste en-dessous.
Les boutons du haut proposent certaines actions, comme "Interrupt" pour interrompre le calcul en cours, "Restart" pour effacer de la mémoire tous les calculs effectués, "Save" pour enregistrer la feuille.

1.2 Assignation

Pour des calculs un peu longs, il est pratique de nommer les expressions intermédiaires par l'instruction d'*assignation*, de la forme :

`nom = expression`

Dans une nouvelle cellule, tapez (puis validez) :

```
u = sqrt(5)+sqrt(3) ; v = sqrt(5)-sqrt(3)
y = u/v + v/u ; y
```

Notez au passage que dans une cellule :
– on peut saisir plusieurs lignes,
– sur chaque ligne, on peut saisir plusieurs instructions séparées par des points-virgules,
– le résultat affiché est celui de la dernière ligne, ici la valeur de `y`.
Il est possible d'assigner plusieurs noms à la fois. Dans la cellule précédente, on peut remplacer la première ligne par :

```
u, v = sqrt(5)+sqrt(3), sqrt(5)-sqrt(3)
```

Obtenez un affichage plus sympathique de `y` avec la commande :

```
y.show()
```

Pour SAGE, les expressions mathématiques sont des *objets* au sens informatique du terme. Tout objet a un *type* bien défini (par exemple Integer, Rational etc.), des *attributs* (i.e. données internes) et des *méthodes* (i.e. fonctions qui peuvent lui être appliquées). Pour faire appel à une méthode d'un objet, la syntaxe est de la forme :

`objet.methode(`*paramètres éventuels*`)`

Par exemple, ci-dessus on a fait appel à la méthode `show()` de l'objet `y`.

Remarque.— Certaines méthodes peuvent également être appelées sous une forme plus classique : `methode(objet,` *paramètres éventuels*`)`. Dans l'exemple précédent, la commande `show(y)` a le même effet que `y.show()`.

Pour afficher le type de `y`, vous pouvez entrer la commande :

```
type(y)
```

1.3 Aide en ligne

SAGE contient de nombreuses fonctions et méthodes prédéfinies (on vous distribuera un aide-mémoire avec les commandes de base concernant l'analyse).

Cherchons à simplifier l'expression de `y`. Dans une cellule, tapez :

```
y.simp
```

suivi de la touche `<TAB>` ce qui fait apparaître la liste des méthodes applicables à `y` et commençant par `simp`. Cliquez sur la méthode `simplify_radical`, complétez la cellule par un couple de parenthèses et validez :

```
y.simplify_radical()
```

Le résultat est 8.

Auto-complétion des mots : comme vous l'avez vu, en tapant le début d'une commande suivie de la touche `<TAB>`, on fait apparaître les méthodes correspondantes.

Aide sur une méthode : utilisez le point d'interrogation. Par exemple, tapez :

```
y.simplify_radical?
```

puis la touche `<TAB>` pour faire apparaître la documentation sur `simplify_radical`, avec des exemples d'utilisation.

1.4 Expressions symboliques

SAGE permet de faire du *calcul formel*, c'est-à-dire manipuler des expressions mathématiques contenant des symboles.
Essayez cet exemple (le signe `#` indique le début d'un commentaire ; pour la suite de ce TD, il sera inutile de taper les commentaires) :

```
var('x,y')    # déclare deux variables symboliques x et y
z = sin(x^2/y) ; z
```

Pour remplacer `x` et `y` dans `z`, on utilise la syntaxe de *substitution* :

```
z(x=-1,y=2)
```

On peut aussi définir une fonction `f` :

```
var('x,y')
f(x,y) = sin(x^2/y)
f
```

et calculer ensuite :

```
f(-1,2)
```

Remarque.— Ne pas confondre l'expression `z` et la fonction `f` :
- à partir de l'expression `z`, il serait possible d'obtenir la fonction correspondante `f` par l'instruction :

```
f = z.function(x,y)
```

- inversement, à partir de la fonction `f` on obtient l'expression `z` en écrivant tout simplement :

```
z = f(x,y)
```

1.5 Types composés

1.5.1 Définitions

SAGE utilise le langage de programmation Python, qui lui-même possède trois types de données composés qui vous seront utiles :

1°) *n-uplet* : plusieurs expressions séparées par des virgules, et entourées d'un couple de parenthèses (en général facultatives), par exemple :

```
v = (x,y,z) ; v
```

2°) *liste* : se note avec des crochets, par exemple :

```
L = [1,4,2,8,5,7] ; L
```

Faites afficher le type de `v` et de `L`.

L'accès à un élément d'un n-uplet ou d'une liste se fait en précisant son indice (numéroté à partir de 0). Evaluez par exemple `L[0]`. Le nombre d'éléments de L s'obtient avec `len(L)`. On peut modifier les éléments d'une liste, mais pas ceux d'un n-uplet. Essayez par exemple :

```
L[0] = -1 ; L
```

et vérifiez que la même instruction appliquée à `v` provoque une erreur.

3°) *dictionnaire* : c'est un ensemble de couples de la forme `clef:valeur`, par exemple :

```
d = {4 : 8, 1 : 25}
```

L'accès à un élément se fait en précisant la clef entre crochet. Evaluez par exemple `d[4]`.

Que renvoie la commande suivante ?

```
var('x,y') ; z = sin(x^2/y)
dico = {x : -1, y : 2}
z(dico)
```

(les dictionnaires sont une autre manière d'opérer des substitutions).

1.5.2 Construction des listes

On peut mettre bout-à-bout des listes (ou des n-uplets) avec l'opération `+`. Essayez :

```
M = [-1,4,2] + [8,5,7]
L == M
```

On teste si deux objets sont égaux (resp. différents) avec l'opérateur `==` (resp. `!=`).

Une liste d'entiers successifs s'obtient avec la fonction `range`. Testez les commandes suivantes :

```
range(10)
```

```
range(3,25)
```

```
range(3,25,2)
```

Il y a aussi la construction `[a..b]` qui est équivalente à `range(a,b+1)`. Testez.

Enfin, on peut construire des listes par *compréhension* :

```
[n^3 for n in range(1,11)]
```

Remarque.— Les n-uplets, listes et dictionnaires peuvent contenir des objets de tous types. Par exemple :

```
M = [1.0, x, 'toto', range(4)] ; M
```

1.6 Programmation

Vous serez parfois amené(e) à programmer vos propres fonctions et utiliser des tests ou des boucles. La syntaxe utilisée est celle du langage Python, avec ses structures de contrôle `if, for, while`.

L'exemple suivant définit une fonction `table`. Entrez :

```
def table(n) :
     a,b,c = 0,1,6
     for i in range(1,n+1) :
          a,b,c = a+b, b+c, c+6
          print i, a
```

Notez la présence des deux-points, ainsi que l'indentation des lignes qui doit être rigoureuse car elle délimite chaque bloc d'instructions. Exécutez ensuite :

```
table(10)
```

Que fait la fonction `table` ?

Voici un autre exemple : programmez la fonction $f : \mathbb{R} \to \mathbb{R}$ définie par $f(x) = x$ si $x \in [-1, 1]$, $f(x) = \frac{1}{x}$ sinon. Solution :

```
def f(x) :
     if  -1 <= x <= 1 :
          return x
     else :
          return 1/x
```

Tracez le graphe de cette fonction sur l'intervalle $[-4, 4]$ par la commande `plot(f,-4,4)`.

2 Problèmes résolus

2.1 Calcul d'une limite

Soit $0 < x < \frac{\pi}{2}$. D'après la formule de Taylor-Mac Laurin-Lagrange, il existe un nombre réel θ, $0 < \theta < 1$, dépendant de x, tel que :

$$\cos(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24}\cos(\theta x).$$

Calculer $\lim\limits_{x \to 0_+} \theta$.

Solution :

```
var('x,theta')
eq = cos(x) == 1-x^2/2+x^4/24*cos(theta*x)
s = solve(eq,theta) ; s     # résout l'équation par rapport à theta
```

```
z = s[0].rhs() ; z          # extrait le membre de droite de s[0]
```

```
z.limit(x=0,dir='plus')    # ou : limit(z,x=0,dir='plus')
```

2.2 Approximation de la tangente

Trouver une fonction f de la forme $x \mapsto \frac{ax^3+bx}{x^2+c}$ qui "ressemble" le plus possible à la fonction tangente au voisinage de 0 (i.e. même développement de Taylor à l'ordre 5). Tracer en superposition les graphes de f et de la fonction tangente.

Solution :

```
var('a,b,c,x')
y = (a*x^3+b*x)/(x^2+c)     # expression à déterminer
t1 = taylor(tan(x),x,0,5)
t2 = taylor(y,x,0,5)
print 'dév. de tan :', t1
print 'dév. de f :', t2   # affiche les 2 développements
```

```
# mise en équations du problème et résolution :
eqs = [t1.coeff(x,i) == t2.coeff(x,i) for i in [0..5]]
vars = a,b,c
s = solve(eqs,vars,solution_dict=True)
s
```

```
sol = y(s[1]) ; sol      # c'est l'expression cherchée
```

```
g1 = plot(tan,-4,4,detect_poles=True,hue=0.7)
g2 = plot(sol,-4,4,detect_poles=True,hue=0.0,linestyle='--')
show(g1+g2,ymin=-4,ymax=4,aspect_ratio=1)
```

Notez la superposition des graphes, obtenue avec l'opérateur `+`, ainsi que certaines options de `plot` et `show` (consultez l'aide en ligne pour plus de détails).

2.3 Champ newtonien

On se place dans le plan. Une masse ponctuelle m est soumise à l'attraction newtonienne de trois masses ponctuelles m_1, m_2, m_3. Il s'agit de déterminer les points d'équilibre, pour lesquels la résultante de ces trois forces d'attraction est nulle.
On rappelle que le champ de gravitation dérive d'un potentiel de la forme :

$$U = \sum_i g\frac{m_i}{r_i}\,.$$

1. On note (x_i, y_i) la position de la masse m_i et (x, y) celle de m. Donner l'expression de U et du gradient de U.

2. On prend $g = 1, m = 1, m_1 = 12, x_1 = -5, y_1 = 0, m_2 = 9, x_2 = 3, y_2 = 0, m_3 = 6, x_3 = 0, y_3 = 8$. Exprimer que la position (x, y) de la masse m est une position d'équilibre.

3. Résoudre en s'aidant d'une étude graphique (on trouve deux positions d'équilibre).

Solution :

```
var('x,y,x1,y1,x2,y2,x3,y3,r1,r2,r3,m1,m2,m3,g')
r1 = sqrt((x-x1)^2+(y-y1)^2)
r2 = sqrt((x-x2)^2+(y-y2)^2)
r3 = sqrt((x-x3)^2+(y-y3)^2)
U = g*(m1/r1+m2/r2+m3/r3)
dU = U.gradient()
print 'potentiel = ', U
print 'gradient = ', dU
```

Entrons les valeurs pour l'application numérique :

```
Un = U(g=1,m=1,m1=12,x1=-5,y1=0,m2=9,x2=3,y2=0,m3=6,x3=0,y3=8)
dUn = Un.gradient()
show(Un) ; show(dUn)
```

On cherche les points (x, y) qui annulent le gradient. Traçons les courbes où s'annule chacune des deux composantes :

```
G1 = implicit_plot(dUn[0]==0,(x,-6,3),(y,-1,9),cmap='Reds_r')
G2 = implicit_plot(dUn[1]==0,(x,-6,3),(y,-1,9),cmap='Blues_r')
show(G1+G2, aspect_ratio=1)
```

Les points d'équilibre correspondent aux points d'intersection de ces deux courbes. Sur la figure, hormis les points $(-5,0),(3,0)$ et $(0,8)$ qui correspondent aux trois masses m_i, on voit deux points, l'un voisin de $(-1,0)$ et l'autre voisin de $(0,4)$.

Traçons le graphe du potentiel, puis ses courbes de niveau :

```
plot3d(Un,(x,-4,2),(y,-1,7))
```

```
contour_plot(Un,(x,-4,2.2),(y,-1,7.5),contours=100,fill=False,\
cmap='spectral').show(aspect_ratio=1)
```

(l'antislash \ indique que, faute de place, on poursuit l'instruction à la ligne suivante).

Ces figures confirment la présence de deux points-selle (donc équilibres instables). On peut calculer ces points avec précision en cherchant les minima du carré du module de `dUn` au voisinage des points $(-1,0)$ et $(0,4)$:

```
z = dUn[0]^2 + dUn[1]^2
minimize(z,(-1,0))
```

```
minimize(z,(0,4))
```

3 Séries numériques

3.1 Convergence

On peut s'aider des commandes `limit` et `taylor`. Par exemple, pour la convergence de la série de terme général $(1+\sqrt{n})^{-n}$, on applique le critère de d'Alembert en tapant :

```
var('n')
u = (1+sqrt(n))^(-n)
limit(u(n=n+1)/u,n=infinity)
```

Pour la convergence de la série de terme général $\frac{1}{n}+\ln(1-\frac{1}{n})$, entrez :

```
var('n')
u = 1/n+log(1-1/n)
u.taylor(n,infinity,2)
```

pour avoir un équivalent de u_n au voisinage de $+\infty$.

3.2 Calcul de sommes

Il s'effectue grâce à la commande `sum`. Considérons par exemple la série $\sum \frac{1}{n(n+4)}$. La commande suivante :

```
var('n,N')
s = sum(1/(n*(n+4)),n,1,N) ; s
```

donne l'expression de la somme partielle $\sum_{n=1}^{N} \frac{1}{n(n+4)}$.

Calculez cette somme pour $N=20$:

```
s(N=20)
```

et vérifiez avec l'opérateur `add` :

```
add(1/(n*(n+4)) for n in [1..20])
```

On obtient la somme $\sum_{n=1}^{\infty} \frac{1}{n(n+4)}$ avec la commande :

```
sum(1/(n*(n+4)),n,1,infinity)
```

3.3 Exercices

1. Calculer les sommes suivantes :

$$S_1 = \sum_{k=0}^{n} k^2 \ , \ S_2 = \sum_{k=1}^{n} k\,k! \ , \ S_3 = \sum_{k=0}^{n} \binom{n+k}{k} \ , \ S_4 = \sum_{k=0}^{n} \frac{\binom{2k}{k}^2}{(k+1)\,4^{2k}}$$

   (le coefficient du binôme $\binom{n}{p}$ s'écrit `binomial(n,p)` et la factorielle de n s'écrit `factorial(n)`).

2. Calculer les sommes suivantes :

$$S_5 = \sum_{n=0}^{\infty} 2^{-n} \ , \ S_6 = \sum_{n=3}^{\infty} \frac{2n+1}{n(n^2-4)} \ , \ S_7 = \sum_{n=0}^{\infty} \frac{(4n+1)\,n!}{(2n+1)!} \ , \ S_8 = \sum_{n=0}^{\infty} \frac{1}{n^2+\sqrt{5}\,n-1}$$

4 Suites de fonctions

Soit la suite de fonctions $(f_n)_n$ définies par $f_n(x) = n\,x^2\,\mathrm{e}^{-nx}$, qu'il s'agit d'étudier sur l'intervalle $[0,1]$. Commencez par définir ces fonctions :

```
var('n,x')
f(n,x) = n*x^2*exp(-n*x)
f
```

4.1 Visualisation

Une représentation graphique permettra de voir comment se comporte cette suite de fonctions selon le paramètre n . Vous allez procéder de trois façons différentes.

4.1.1 Graphes superposés

Construisez les graphes superposés de $f_1, \ldots, f_{20}$ sur l'intervalle $[0,1]$ par la commande :

```
g = Graphics()      # crée un objet graphique vide
for n in [1..20] :
     g = g + plot(f(n,x),x,0,1)
```

et tracez :

```
g.show()
```

4.1.2 Animation

Vous la réaliserez grâce à la commande `animate` qui prend en paramètre une liste de graphes :

```
liste = [plot(f(n,x),x,0,1) for n in [1..20]]
animate(liste,ymax=0.4).show(delay=10,iterations=3) # patientez !
```

`delay` permet de fixer la cadence de l'animation (en centièmes de secondes) et `iterations` fixe le nombre de répétitions (avec 0 l'animation tourne en boucle indéfiniment).

4.1.3 Graphisme interactif

Programmez la fonction suivante, qui prend en paramètre n et trace le graphe de f_n :

```
def dessin(n) :
     plot(f(n,x),x,0,1).show(ymax=0.4)
```

et testez-la par :

```
dessin(3)
```

On peut rendre cette fonction interactive grâce au *décorateur* `@interact`. Entrez :

```
@interact
def dessin(n=[1..20]) :
     plot(f(n,x),x,0,1).show(ymax=0.4)
```

Cette étude graphique permet de deviner que la suite $(f_n)_n$ converge uniformément vers la fonction nulle. Il reste à le prouver.

4.2 Limite simple

Si vous tapez :

```
f(n,x)
```

vous constaterez que `n` a conservé la valeur 20 de la construction `for` de **4.1.2**. Il faut donc le désassigner :

```
var('n')
f(n,x)
```

Demandez la limite de $f_n(x)$ quand $n \to \infty$:

```
limit(f(n,x),n=infinity)
```

SAGE répond par une question : x est-il positif, négatif ou nul ?
Fixez la contrainte $x > 0$:

```
assume(x>0)
limit(f(n,x),n=infinity)
```

La limite est 0.
A tout instant, vous pouvez voir la liste des contraintes que vous avez imposées :

```
assumptions()
```

Levez la contrainte avec :

```
forget(x>0)
```

(sachez que la commande `forget()` sans paramètre lève toutes les contraintes effectuées).
Il reste à vérifier la limite pour $x = 0$:

```
assume(x==0) ; limit(f(n,x),n=infinity)
```

4.3 Convergence uniforme

Cherchons à majorer $f_n(x)$ sur $[0, +\infty[$. Pour cela, calculez le point x_n où f_n a son maximum. En ce point la dérivée f'_n s'annule :

```
forget()
df = f(n,x).diff(x) ; df   # expression de la dérivée par rapport à x
```

```
s = solve(df==0,x) ; s
```

d'où le point x_n :

```
xn = s[0].rhs()
```

et la valeur du maximum :

```
maxfn = f(n,xn) ; maxfn
```

La commande suivante permet de conclure que la suite $(f_n)_n$ converge uniformément :

```
limit(maxfn,n=infinity)
```

4.4 Exercices

1. Etudier la convergence sur $\mathbb{R}$ de la suite de fonctions définie par :

$$f_n(x) = \frac{1}{1+(1+nx)^2}$$

2. Etudier la convergence sur $\mathbb{R}_+$ de la suite de fonctions définie par :

$$f_n(x) = \frac{x^2 \mathrm{e}^{-nx}}{n^\alpha}$$

où α est un paramètre réel.