Variations autour d'un sujet de Bac S

Guillaume Connan

Irem de Nantes

4 juillet 2013

Résumé

Les sujets de Bac posent souvent quelques problèmes mathématiques et informatiques : le recours exclusif à des Entrée/Sortie, l'utilisation de tests sur des flottants. Ces dangers ont été illustrés dans un article précédent publié sur **Mathématice**.

Il y a beaucoup de discussions depuis des années autour du langage de programmation utilisé mais dans la majorité des cas vus en lycée, ce choix est peu important : c'est la modélisation en amont et éventuellement la vérification en aval qui prime.

1 Extrait du sujet de Bac S Polynésie 2013

Voici l'énoncé complet de l'exercice 1 proposé en 2013 au Bac S Polynésie :

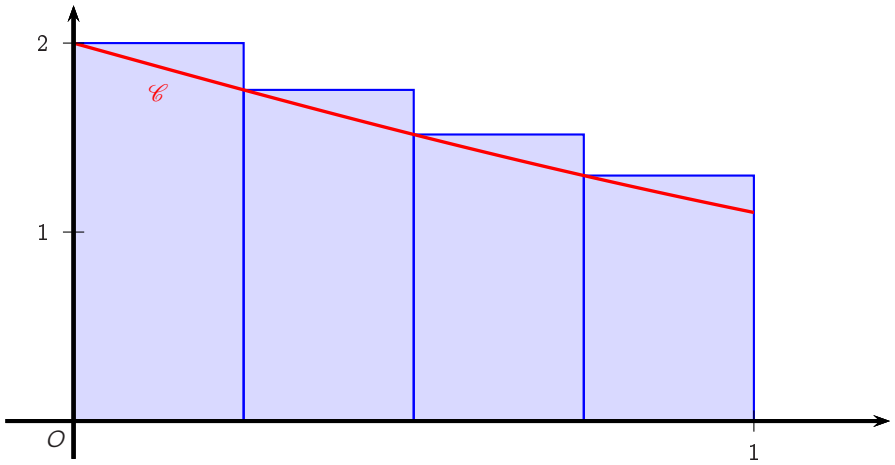
On considère la fonction f définie sur $\mathbb{R}$ par

$$f(x) = (x+2)\,e^{-x}.$$

On note $\mathscr{C}$ la courbe représentative de la fonction f dans un repère orthogonal.

1. *Étude de la fonction f.*
   (a) *Déterminer les coordonnées des points d'intersection de la courbe $\mathscr{C}$ avec les axes du repère.*
   (b) *Étudier les limites de la fonction f en $-\infty$ et en $+\infty$. En déduire les éventuelles asymptotes de la courbe $\mathscr{C}$.*
   (c) *Étudier les variations de f sur $\mathbb{R}$.*
2. *Calcul d'une valeur approchée de l'aire sous une courbe.*
   On note $\mathscr{D}$ le domaine compris entre l'axe des abscisses, la courbe $\mathscr{C}$ et les droites d'équation $x = 0$ et $x = 1$. On approche l'aire du domaine $\mathscr{D}$ en calculant une somme d'aires de rectangles.
   (a) *Dans cette question, on découpe l'intervalle $[0\ ;\ 1]$ en quatre intervalles de même longueur :*
   - *Sur l'intervalle $\left[0\ ;\ \frac{1}{4}\right]$, on construit un rectangle de hauteur $f(0)$*
   - *Sur l'intervalle $\left[\frac{1}{4}\ ;\ \frac{1}{2}\right]$, on construit un rectangle de hauteur $f\left(\frac{1}{4}\right)$*
   - *Sur l'intervalle $\left[\frac{1}{2}\ ;\ \frac{3}{4}\right]$, on construit un rectangle de hauteur $f\left(\frac{1}{2}\right)$*
   - *Sur l'intervalle $\left[\frac{3}{4}\ ;\ 1\right]$, on construit un rectangle de hauteur $f\left(\frac{3}{4}\right)$*

   Cette construction est illustrée ci-dessous :



L'algorithme ci-dessous permet d'obtenir une valeur approchée de l'aire du domaine $\mathscr{D}$ en ajoutant les aires des quatre rectangles précédents :

Variables :	*k est un nombre entier*
	S est un nombre réel
Initialisation :	*Affecter à S la valeur 0*
Traitement :	*Pour k variant de 0 à 3*
	Affecter à S la valeur $S + \frac{1}{4} f\left(\frac{k}{4}\right)$
	Fin Pour
Sortie :	*Afficher S*

Donner une valeur approchée à 10^{-3} près du résultat affiché par cet algorithme.

(b) Dans cette question, N est un nombre entier strictement supérieur à 1. On découpe l'intervalle $[0; 1]$ en N intervalles de même longueur. Sur chacun de ces intervalles, on construit un rectangle en procédant de la même manière qu'à la question 2.a).

Modifier l'algorithme précédent afin qu'il affiche en sortie la somme des aires des N rectangles ainsi construits.

3. *Calcul de la valeur exacte de l'aire sous une courbe.*

Soit g la fonction définie sur $\mathbb{R}$ par

$$g(x) = (-x - 3)e^{-x}.$$

On admet que g est une primitive de la fonction f sur $\mathbb{R}$.

(a) Calculer l'aire $\mathscr{A}$ du domaine $\mathscr{D}$, exprimée en unités d'aire.

(b) Donner une valeur approchée à 10^{-3} près de l'erreur commise en remplaçant $\mathscr{A}$ par la valeur approchée trouvée au moyen de l'algorithme de la question 2.a), c'est-à-dire l'écart entre ces deux valeurs.

2 Méthode des rectangles

2.1 Cas général

On reconnaît bien sûr la méthode des rectangles. L'objet du problème est de calculer la somme et de comparer *numériquement* la valeur approchée calculée à celle de l'intégrale à 10^{-3} près. Cette somme est donnée par :

$$S^+(f, a, b, N) = \frac{b-a}{N} \sum_{k=0}^{N-1} f\left(a + \frac{k(b-a)}{N}\right)$$

qui permet d'obtenir une majoration de l'intégrale :

$$I(f,a,b)=\int_a^b f(x)\,\mathrm{d}x$$

Dans l'exercice, $a=0$ et $b=1$. Comme la fonction est décroissante sur $[0,1]$, on obtient ainsi une majoration de l'intégrale.
On peut difficilement aujourd'hui étudier l'erreur commise au lycée. Dans une classe motivée, on peut malgré tout étudier l'inégalité des accroissements finis qui était un outil phare du lycée il y a une dizaine d'années...
Elle permet de montrer que :

$$|f(x)-f(a)|\leqslant (b-a)\sup_{x\in[a,b]}|f'(x)|$$

puis :

$$\left|\int_a^b f(x)\,\mathrm{d}x-(b-a)f(a)\right|\leqslant (b-a)^2\sup_{x\in[a,b]}|f'(x)|$$

et plus généralement :

$$\left|I(f,a,b)-S^+(f,a,b,N)\right|\leqslant \frac{(b-a)^2}{N}\sup_{x\in[a,b]}|f'(x)|$$

Bon, ça, nous le savons. Nous pouvons éventuellement demander aux élèves d'admettre que l'on peut démontrer ce résultat. Cela nous donne un contrôle mathématique de l'erreur commise. Cela exige malgré tout de connaître un majorant de la dérivée de f sur l'intervalle d'étude...

2.2 Cas des fonctions monotones

Ici, l'erreur peut être étudiée en Terminale car la fonction est décroissante sur [0,1].
Notons :

$$S^-(f,a,b,N)=\frac{b-a}{N}\sum_{k=1}^{N} f\left(a+\frac{k(b-a)}{N}\right)$$

et $h=\frac{b-a}{N}$.
La fonction étant décroissante et à valeurs positives sur $[a,b]$, on a, pour tout entier naturel $k\leqslant N-1$ et tout réel $x\in[a+kh,a+(k+1)h]$:

$$f(a+(k+1)h)\leqslant f(x)\leqslant f(a+kh)$$

On intègre sur $[a+kh,a+(k+1)h]$:

$$h\cdot f(a+(k+1)h)\leqslant \int_{a+kh}^{a+(k+1)h} f(x)\,\mathrm{d}x\leqslant h\cdot f(a+kh)$$

On somme de $k=0$ jusque $N-1$:

$$0\leqslant S^-(f,a,b,N)\leqslant I(f,a,b)\leqslant S^+(f,a,b,N)$$

Finalement :

$$0\leqslant S^+(f,a,b,N)-I(f,a,b)\leqslant S^+(f,a,b,N)-S^-(f,a,b,N)=\frac{b-a}{N}(f(a)-f(b))$$

qui nous donnera une majoration de l'erreur utilisable par les élèves.

3 Rapide étude du problème avec XCAS et le calcul formel

Pendant l'année, puisqu'on dispose d'ordinateurs, on peut les utiliser comme super-calculatrices pour étudier des fonctions.
On définit la fonction :

XCAS
```
f := x -> (x + 2)*exp(-x)
```

Les intersections avec les axes :

XCAS
```
resoudre(f(x) = 0)
f(0)
```

Les limites :

XCAS
```
limite(f(x),x,-infinity)
limite(f(x),x,+infinity)
```

Le signe de la dérivée :

XCAS
```
resoudre(f'(x) < 0)
```

La représentation graphique sur $[-2, 5]$:

XCAS
```
graphe(f(x),x = -2 .. 5)
```

Le calcul de l'aire par la méthode des rectangles :

XCAS
```
aire(f(x),x=0..1,4,rectangle_gauche)
```

L'illustration graphique :

XCAS
```
tracer_aire(f(x),x=0..1,4,rectangle_gauche,affichage=jaune)
```

La valeur exacte de l'intégrale :

XCAS
```
integrer(f(x),x,0,1)
```

Une valeur approchée à 10^{-4} près :

XCAS
```
approx(integrer(f(x),x,0,1) , 4)
```

4 L'algorithme du sujet en différents langages

4.1 Parlons fonction

En cours de mathématiques et également en informatique, il est très maladroit d'utiliser des sorties du style *Afficher S* car cela enlève toute possibilité d'utiliser les résultats dans une nouvelle fonction : l'ordinateur ne va pas marcher sur ses petites jambes et aller regarder ce qui se passe à l'écran, c'est absurde...
Il est indispensable de n'utiliser que des fonctions pour traiter efficacement les résultats obtenus. Pour des problèmes plus complexes, il faudra les diviser en sous-tâches plus simples et donc être capable de composer les résultats obtenus. Cette modularité est la base de la programmation.
Il s'agit donc de créer une fonction $s(n)$ qui calcule la somme demandée en fonction du nombre n de rectangles.
On peut même créer une fonction plus générale définie par $S^+(f,a,b,N)$ qui calcule la somme demandée pour toute fonction f sur tout intervalle $[a,b]$ où f est suffisamment régulière.
On peut également éviter de multiplier par $1/N$ à chaque itération comme proposé dans le texte : cela demande des calculs inutiles pouvant de plus créer des problèmes d'arrondis. On multipliera donc par $1/N$ en sortie de boucle.
Notre problème devient donc :

Fonction `rect_gauche( f:`*fonction [a,b] -> R* `a,b:`*flottants* `N:`*entier > 0* `):`*flottant*
`S ← 0`
`h ← (b-a)/N`
Pour `k` **variantDe** `0` **Jusque** `N-1` **Faire**
  `S ← S + f(a + k * h)`
FinPour
Retourner `h * S`

Ensuite, on peut traduire ça dans 100000 langages, ça revient toujours au même...

4.2 Avec XCAS

XCAS n'est pas seulement un logiciel de calcul formel : il permet également de travailler les problèmes d'algorithmique en français.

XCAS

```
rect_gauche(f,a,b,N) := {
  S := 0;
  h := (b - a)/N;
  pour k de 0 jusque N - 1 faire
    S := S + f(a + k*h)
  fpour
  retourne h*S
 }:;
```

Pour notre exemple :

XCAS

```
rect_gauche(x -> (x + 2)*exp(-x) , 0. , 1. , 4)
```

renvoie bien `1.64190910781`. Notez bien les · après 0 et 1 : les bornes sont des flottants. Si nous les utilisons sans ·, XCAS considérera 0 et 1 comme des entiers formels et le traitement sera plus long mais exact :

XCAS

```
rect_gauche(x -> (x + 2)*exp(-x) , 0 , 1 , 4)
```

$$\frac{1}{4}\cdot\left(2+\frac{9}{4}\cdot e^{-\frac{1}{4}}+\frac{5}{2}\cdot e^{-\frac{1}{2}}+\frac{11}{4}\cdot e^{-\frac{3}{4}}\right)$$

Vous noterez la possibilité de donner une fonction anonyme créée « à la volée ».
Si vous préférez, XCAS parle aussi anglais :

XCAS

```
left_rec(f,a,b,N) := {
  S := 0;
  h := (b - a)/N
  for(k := 0; k < N; k++){
    S := S + f(a + k*h)
  }
  return h*S
}:;
```

4.3 Avec Python

On peut faire la même chose :

Python

```
def rect_gauche(f,a,b,N):
    S = 0
    h = (b - a)/N
    for k in range(N):
        S += f(a + k*h)
    return h*S
```

et on obtient :

Python

```
In [2]: rect_gauche(lambda x: (x + 2)*exp(-x) , 0., 1., 4)
Out[2]: 1.6419091078075088
```

Veillez bien à « pointer » les bornes sinon Python va effectuer une division euclidienne pour calculer h et vous obtiendrez 0...
Notez l'emploi du `+=` qui permet d'incrémenter rapidement, `i += 2` signifiant par exemple `i = i + 2`.
Comme XCAS, Python permet de donner en argument de fonction une fonction créée à la volée avec l'opérateur `lambda`.

4.4 Avec...

Cet algorithme est tellement basique qu'il peut être écrit à l'identique dans tout langage pouvant prendre une fonction en argument (et vous aurez bien noté qu'il s'agit d'une fonction et pas d'une expression : c'est la même bataille que vous menez avec les élèves pour qu'ils ne confondent pas f et $f(x)$...).

5 L'algorithme du sujet sans boucle « for »

Le sujet propose une boucle « for » mais on peut s'en passer, notamment si le langage accepte quelques outils de programmation fonctionnelle.
On travaille alors sur des listes de valeurs de x. Là encore, peu importe le langage de programmation choisi : c'est le paradigme de programmation et les structures utilisées qui guident l'algorithme.

5.1 Avec Python

On utilise la fonction de « pliage » `reduce` qui réduit (ou plie) une liste en appliquant une fonction de deux variables successivement sur les arguments d'une liste.
Par exemple :

Python
```
reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
```

calcule $((((1+2)+3)+4)+5)$.
Cela donne ici :

Python
```
def frect_gauche(f,a,b,N):
    h = (b - a)/N
    return h * reduce(lambda x, S: S + x, [f(a + k*h) for k in range(N)])
```

De nombreux langages disposent d'une fonction `sum`, en particulier Python, ce qui simplifie ici les choses :

Python
```
def srect_gauche(f,a,b,N):
    h = (b - a)/N
    return h * sum( [f(a + k*h) for k in range(N)] )
```

5.2 Avec Haskell

Haskell est un langage purement fonctionnel. C'est un vrai régal à la fois pour le mathématicien et pour le programmeur. Vous pouvez rapidement vous initier à ce bijou sur cette page.
Ici, le pliage s'écrit `foldl` (fold comme *plier* et l comme *left*).

Haskell
```
rect_gauche f a b n =
  h * foldl (\s x -> s + x) 0 [f(a + k*h) | k <- [0..n-1]]
  where h = (b - a)/n
```

ce qui donne :

Haskell
```
*Main> rect_gauche (\x -> (x + 2)*exp(-x))  0  1  4
1.6419091078075088
```

et avec `sum` :

Haskell
```
srect_gauche f a b n =
  h * sum [f(a + k*h) | k <- [0..n-1]]
  where h = (b - a)/n
```

6 Combien d'itérations pour une précision donnée ?

6.1 Différence des sommes...

Nous avons démontré au début de cet article que lorsque la fonction étudiée était décroissante, on obtenait :

$$0 \leqslant S^+(f,a,b,N) - I(f,a,b) \leqslant S^+(f,a,b,N) - S^-(f,a,b,N)$$

Ainsi, la « distance » de I à S^+ est majorée par la différence $S^+(f,a,b,N) - S^-(f,a,b,N) = \frac{b-a}{N}(f(a) - f(b))$.
Il **suffit** donc d'incrémenter N jusqu'à ce que la différence $S^+ - S^-$ soit suffisamment petite.

6.2 Si l'on connaît la valeur exacte de l'intégrale

Notre algorithme perd de son intérêt mais il nous permet de mesurer la qualité de notre approximation. C'est d'ailleurs l'esprit du sujet de Bac si l'on regarde la question 3.

6.3 Oui mais les flottants de la machine ne sont pas des réels...

Comme expliqué dans cet article, il est parfois très dangereux de travailler sans précaution avec les flottants de la machine qui ne sont pas des réels et ont leur propre arithmétique parfois déroutante. Par exemple :

1. 3×0.1 n'est pas égal à 0.3 :

Python

```
In [1]: 0.1 + 0.1 + 0.1 == 0.3
Out[1]: False

In [2]: 3*0.1
Out[2]: 0.30000000000000004
```

   En effet, le processeur compte en base 2 (les flottants sont codés sur 64 bits (ou 32)) et 0.1 « ne tombe pas juste » en base 2 puisqu'il s'agit de la division de 1 par 1010...
   On essaiera dans la mesure du possible de ne diviser que par des puissances de 2. D'une part cela évitera les erreurs d'arrondi et d'autre part cela transformera une division en un simple changement de l'exposant du flottant dans sa représentation standard (signe, mantisse, exposant).
   Un nombre flottant *double précision* est en effet représenté sous la forme :

$$[b_0, b_1, ..., b_{52}, b_{53}, ..., b_{63}] = (-1)^{b_0} 0.1 b_1 b_2 ... b_{52} \cdots 2^{e-1023}$$

   où e est l'exposant codé sur 11 bits.

2. Comment interpréter cette fonction :

Python

```
def test1(x):
    k   = 1
    val = x
    while x + val > x:
        val = val * 0.5
        k = k + 1
    return val,k
```

et ce résultat :

Python

```
In [4]: test1(0.1)
Out[4]: (5.551115123125783e-18, 55)
```

Ce test ne devrait jamais s'arrêter...et pourtant... En effet, pour le processeur, le flottant 1 a un successeur qu'on appelle le plus souvent $1+\varepsilon$. Cet ε vaut environ 2.2×10^{-16}.

Deux nombres distants de moins de ε sont alors considérés comme égaux.

On veillera donc à ne pas faire de tests d'égalité sur des flottants mais des comparaisons $<$ ou $>$.

3. Voici un autre test perturbant :

Python

```
def test2(xo):
    x = xo
    for i in range(1,100):
        x = 4*x - 1
        print x
```

Normalement, avec $1/3$ comme argument, cette fonction doit renvoyer un résultat constant...et pourtant :

Python

```
In [6]: test2(1./3.)
0.333333333333
0.333333333333
0.333333333333
.
.
.
0.33332824707
0.333312988281
0.333251953125
.
.
.
-1398101.0
-5592405.0
-22369621.0
.
.
.
-1.16149714576e+41
-4.64598858303e+41
-1.85839543321e+42
-7.43358173284e+42
```

Fichtre : en 100 itérations, on arrive à des résultats totalement aberrants !

Et il y a encore bien d'autres anomalies...qui n'en sont pas si on est averti du fonctionnement du processeur et de la norme IEEE754.

6.4 Tests raisonnés

Compte tenu de ces remarques, on peut proposer au choix, avec un logiciel de calcul formel qui nous permettra de valider nos hypothèses :

6.4.1 Avec XCAS

Si l'on est sûr que notre fonction est positive, décroissante et que eps n'est pas trop petit :

XCAS

```
test_arret(f,a,b,eps) := {
  m := 0;
  num := (b - a)*(f(a) - f(b))
    tantque num/2^m > eps faire
      m := m + 1
    ftantque
  retourne 2^m, rect_gauche(f,a,b,2^m), integrer(f(x),x,a,b)
}:;
```

Alors :

XCAS

```
test_arret1(x -> (x + 2)*exp(-x) , 0. , 1. , 0.0001)
                                    16384,1.5285095899,1.52848223531
```

6.4.2 Avec Sage

Sage est un logiciel de calcul mathématique dont la syntaxe de programmation est celle de Python. Vous saurez tout sur Sage en lisant ce livre disponible sur cette page.

Sage

```
def test_arret(f,a,b,eps):
    m = 0
    num = (b - a) / (f(a) - f(b))
    while num / 2^m > eps:
        m += 1
    return 2^m, rect_gauche(f,a,b,2^m), integrate(f(x),x,a,b)
```

Sage

```
sage: test_arret(lambda x:  (x + 2)*exp(-x) , 0. , 1. , 0.0001)
(16384, 1.52850959018375, 1.52848223531)
```

C'est comme avec XCAS.

7 Un peu de dessin avec la bibliothèque Pygal de Pyhton

La bibliothèque Pygal permet d'obtenir de jolis dessin au format SVG pour une lecture dynamique dans un navigateur.

Python

```
from pygal import *
from pygal.style import NeonStyle
```

```
def rectangles(s,f,a,b,N):
    # on crée un objet graphique de type courbe en (x,y)
    g = XY(fill=True ,style=NeonStyle,show_dots=False)
    g.title = 'Approximation de l\'integrale avec '+ str(N) +  ' rectangles'
    h = (b - a)/N
    # on ajoute la liste des couples de points des rectangles gauche en les classant selon leur ordonnée
    g.add('I <= ' + str(srect_gauche(f,a,b,N)),sorted([(a + k*h, f(a + k*h)) for k in range(N)] + [ (a +
        (k+1)*h, f(a + k*h)) for k in range(N)] , key = lambda point: point[1], reverse = True))
    # idem avec les rectangles droite
    g.add('I >= ' + str(srect_droite(f,a,b,N)),sorted([(a + k*h, f(a + (k+1)*h)) for k in range(N)] + [ (a +
        (k+1)*h, f(a + (k+1)*h)) for k in range(N)] , key = lambda point: point[1], reverse = True))
    # on rajoute les points de la courbe
    g.add('I',[(a + k*h, f(a+k*h)) for k in range(N+1)] + [(b,0.)])
    # on exporte au format svg
    g.render_to_file(s + '.svg')
```

Si vous travaillez dans ipython, vous pouvez alors lancer firefox directement :

Python

```
In [168]: rectangles('Methode_des_rectangles',lambda x: (x+2)*exp(-x),0.,1.,10)

In [169]: !firefox Methode_des_rectangles.svg
```

Et vous obtenez ce fichier dans votre navigateur où il se passe de jolies choses quand vous y promenez la souris.