COMPRENDRE

SCILAB : DES POINTS QUI VOUS TENDENT LES BRAS

Si vous ne deviez retenir que 3 choses des pages qui suivent, ce serait que :
1) Scilab prend de plus en plus de place dans les sujets de concours et dans le décompte des points. (EDHEC 2017 cause toujours des cauchemars à certains.)
2) Les questions Scilab sont (encore pour le moment) d'une facilité déconcertante, à la limite de la trivialité.
3) En conséquence, il serait profondément stupide de faire l'impasse sur un sujet dont le ratio points rapportés / investissement en temps de travail est extrêmement élevé...

PAR FRÉDÉRIC BROSSARD, PROFESSEUR DE MATHÉMATIQUES À INTÉGRALE PRÉPA, BASÉE À PARIS

Scilab, c'est avant tout des points faciles à prendre !

Pourquoi Scilab ?

L'algorithmique est une composante à part entière du programme de mathématiques du secondaire dans nombre de sections, il n'est donc pas illogique, en continuité, d'en retrouver en classes préparatoires. En remplacement de Turbo Pascal devenu obsolète, le choix du langage support s'est porté en 2014 sur Scilab qui présente deux avantages majeurs : ses larges capacités de modélisation en analyse, algèbre, probabilités, statistiques et... sa gratuité. On pourrait ajouter également sa relative simplicité qui permet à celles et ceux qui y consacrent des périodes de travail régulières d'en acquérir rapidement une maîtrise certaine.

A ceux qui remettraient en cause l'utilité de son apprentissage, on peut arguer que, d'une part, savoir modéliser une situation ou un problème est l'un des fondamentaux de la vie professionnelle et que le raisonnement algorithmique permet de bien structurer ce type de démarche. D'autre part, un diplômé d'une Grande École de management sera amené à utiliser des outils informatiques basés sur les mathématiques et la modélisation (fast trading, simulation de gains ...) et, de fait, il est bon d'en connaître quelque peu le fonctionnement.

Et l'argument massue est sans aucun doute qu'être noté sur 18 au lieu de 20 aux concours parce qu'on n'a pas fait de Scilab pendant deux ans n'est sans doute pas l'idée du siècle quand une admissibilité se joue au quart de demi-point.

Comment travailler Scilab efficacement ?

Le mot qui fâche et à double titre : ordinateur. Tout d'abord, parce qu'il n'y a évidemment pas aux concours d'épreuve informatique devant console mais surtout parce qu'en dépit de cela, il faut néanmoins consacrer un peu de temps à la pratique, celle-ci étant le meilleur moyen de capitaliser les connaissances acquises.

Une heure et demi par semaine est le temps idéal à dédier à Scilab pendant les deux années de prépa, moitié théorie, moitié pratique.

Passée une première phase d'apprentissage de l'ensemble des commandes listées dans le Programme Officiel (comme pour tout langage, il convient d'avoir un peu de vocabulaire avant de... parler), je conseille de « travailler Scilab » en même temps que le cours de maths correspondant. Par exemple, vous révisez les suites / séries, vous en profitez pour étudier, apprendre par cœur, programmer et faire tourner les algorithmes (cf Programme Officiel) inhérents à ce thème et que vos enseignants vous ont présenté en classe : calcul du n-ième terme d'une suite, conjecture graphique de la limite d'une suite un=f(n), rang à partir duquel un est proche de sa limite à ε près, calcul du rang à partir duquel une somme partielle est supérieure à un réel donné... L'apprentissage par cœur est un impératif, ne faites pas dans un à peu près qui ne vous rapportera pas de point.

En deuxième année, celles et ceux qui visent les Parisiennes auront tout intérêt à passer du temps sur les TD Scilab du Programme (6x3h dans l'année). Ceux-ci sont, en effet, une source inépuisable d'épreuves de maths car ils introduisent des notions à la marge que prisent les concepteurs de sujets HEC ou ESSEC. En voie E, par exemple, le TD « Chaînes de Markov » a été traité au moins deux fois de manière théorique dans les 10 dernières années (une épreuve ESSEC et une épreuve HEC). Idem pour le TD « Simulation de Lois » (ESSEC 2014). La composante mathématique de ces TD est donc d'une rare richesse.

Scilab, trous et interprétation

Les questions Scilab posées aux concours ne sont pas très variées. Elles peuvent se résumer à 1) compléter un programme donné par l'énoncé dans lequel se sont glissés quelques trous 2) savoir expliquer ce que réalise un algorithme donné 3) savoir interpréter le résultat graphique produit par un ou des algorithmes (par exemple : comparaison de fonctions de répartition pour prouver une convergence en loi). Et pour savoir y répondre à coup sûr, rien de plus simple... appliquez simplement pendant deux ans le plan de travail énoncé dans le paragraphe précédent.

Super calculatrice... programmable et graphique

Scilab est à la fois une calculatrice permettant de réaliser toute sorte d'opérations tant sur des scalaires que des matrices et un langage de programmation structurée qui autorise la conception d'algorithmes complexes et la création de fonctions spécifiques par l'utilisateur dans le mode éditeur. Scilab permet aussi d'afficher toutes sortes de graphiques : graphes de fonctions, histogrammes, diagrammes en bâtons...

La dorsale algorithmique de Scilab est relativement pauvre car elle ne repose que sur les trois piliers traditionnels vus dans le secondaire que sont la boucle inconditionnelle « pour... » (for), la boucle conditionnelle « tant que » (while) et l'instruction conditionnelle « si alors sinon » (if then else). Rien de nouveau donc. On ajoute au tout l'instanciation de variables et on a fait le tour ou presque.

Scilab en mode calculs

La liste des opérateurs de calculs analytiques à connaître est courte : addition (+), soustraction (-), multiplication (*), division (/), puissance (^), racine (sqrt( )), partie entière (floor( )), valeur absolue (abs ( )), exponentielle (exp ( )) et logarithme népérien ln (log ( )). On notera également que π s'écrit %pi et e %e. Ce n'est donc pas là qu'on va mesurer l'étendue des immenses possibilités offertes par Scilab.
Celles-ci sont conséquence de la structure opératoire matricielle de Scilab qui considère en effet tous les éléments utilisés dans les calculs comme des matrices. Définir une matrice est très simple.

A=[1 1 1 ; 2 1 2] (espace entre deux valeurs sur la même ligne et ; pour changer de ligne)

Scilab propose également des matrices prédéfinies : eye(n,n) identité de taille (n,n), zeros(n,m) matrice (n,m) remplie de zéros ou ones(n,m) matrice remplie de 1.
Si A et B sont deux matrices : A+B va renvoyer la matrice somme, A*B (sous réserve de compatibilité) va renvoyer le produit, A^n la puissance n-ième... sum(A) va renvoyer la somme des coefficients de A. On peut également calculer des sommes par colonnes sum(A,'r') ou lignes, diag(A) va extraire sous forme d'un vecteur colonne, les éléments de la diagonale, spec(A) va donner les valeurs propres, A' va donner la matrice transposée de A, size(A) va renvoyer les dimensions de A, mean(A) la moyenne des coefficients etc.
Mais beaucoup plus intéressant, on peut également réaliser des opérations termes à termes avec l'opérateur pointé. Ainsi A.*B va renvoyer une matrice dont les coefficients sont les produits aij*bij ou encore .log(A) va renvoyer une matrice où chaque coefficient est le logarithme népérien (sous réserve d'existence) des coefficients de A.

Un exemple d'application : soit à calculer les 10 sommes S1, ..., S10 définies par Si=31+...+3i pour i appartenant à {1,...,10} :

```
q=3
M=zeros(10)
for k=1:10
    M(k)=sum(q.^[1:k])
end
disp(M)
```

Qui renvoie bien :

```
   3.
   12.
   39.
   120.
   363.
   1092.
   3279.
   9840.
   29523.
   88572.
```

Quant à la résolution de systèmes linéaires AX=B, c'est un jeu d'enfant avec, cerise sur le gâteau, deux instructions possibles : A\B ou (A,-B)

Scilab en mode probas

Deux générateurs de nombres aléatoires suivant une loi donnée sont au menu : rand(n,m) qui génère une matrice de nombres aléatoires de taille (n,m) suivant une loi uniforme sur]0,1[et grand(n,m,'loi',paramètres) qui génère une matrice de nombres aléatoires de taille (n,m) suivant une loi (qui peut être discrète : 'uin', 'bin', 'geom', 'poi' ou continue 'exp', 'nor','unf').
Pour analyser une série statistique générée par rand ou grand, on a à disposition tous les outils possibles et imaginables par exemple (X et Y étant des vecteurs ou matrices de données aléatoires) :

- La moyenne empirique mean(X)
- La variance empirique mean(X.^2)-mean(X)^2
- La covariance empirique mean(X.*Y)-mean(X)*mean(Y)
- L'affichage d'histogrammes histplot(nombre de classes, X) ou histplot([bornes des classes],X) où [bornes des classes] est un vecteur ligne délimitant les classes souhaitées
- L'affichage de diagrammes en bâtons : bar
- Le classement par ordre croissant ou décroissant : gsort
- Le tri ordonné avec calcul des fréquences d'apparition : tabul(X,'i') (ordre croissant)

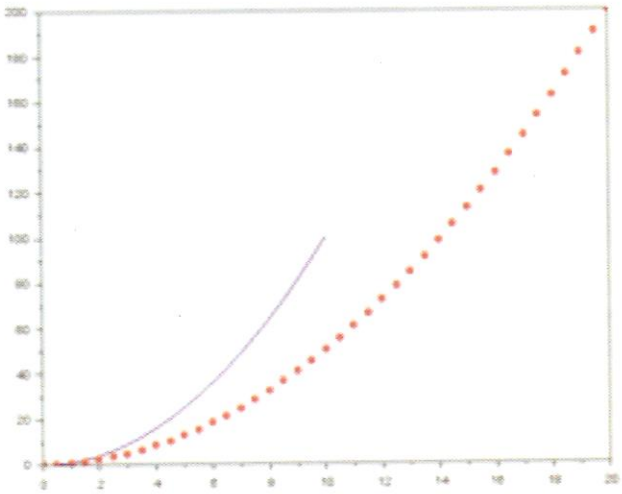
On peut également programmer ses propres fonctions d'analyse telles que la fonction de répartition empirique :

```
function repart(X)
    N=input("nombre de simulations ?")
    L=length(X)
    X=tabul(X,"i")
    X(:,2)=cumsum(X(:,2))
    plot(X(:,1),X(:,2)/N)
endfunction
```

Scilab en mode graphique

Scilab est incapable d'afficher un graphe « continu » tel que peut le faire Geogebra. Il ne sait qu'afficher des couples de points reliés par des segments. Aussi, toutes les instructions graphiques de

Scilab telles que plot qui permet d'afficher un graphe de fonction f d'une variable nécessitent d'abord, la définition de la fonction, la définition d'un vecteur des abscisses X (par exemple un vecteur à pas constant X=-1 :0.1 :1 qui donne [-1, -0.9, -0.8,...,0.9,1] ou V=linspace(1,0.5,10) qui donne 10 valeurs équiréparties entre 1 et 0.5 dans cet ordre) puis plot(X,f(X)) pour afficher le graphe.



Scilab en mode programmation

Le maniement des boucles et de l'instruction conditionnelle est le même que celui que vous avez étudié dans le secondaire, à une exception intéressante près : la boucle for qui utilise à plein la structure matricielle de Scilab. Ainsi for i=1 :0.1 :10 va faire prendre successivement à i les valeurs 1, 1.1, 1.2,...,10. Le pas de boucle est donc à l'entière disposition de l'utilisateur.

Le conditionnement des boucles se fait avec les opérateurs de comparaison traditionnels. Ainsi, while (u>1) va autoriser la boucle while à tourner tant que la valeur de la variable u n'est pas inférieure ou égale à 1. De même, if x==0 then (condition 1) else (condition 2) va réaliser la condition 1 si la variable x prend la valeur 1 et la condition 2 sinon.
Scilab offre à l'utilisateur la possibilité de développer ses propres fonctions puis de les utiliser ensuite à sa guise dans tout calcul.

Exemple :

```
function [V,u]=exemple2(n,u0,q)
    //On va stocker dans le vecteur V les sommes partielles
    //u va être la valeur du n-ième terme de la suite
    //une suite géométrique est définie par u0 et q
    V=zeros(n+1) //on initie un vecteur V (ça ne mange pas de pain)
    u=u0
    V(1)=u0 //la première somme partielle est u0
    //Problème de Scilab, on ne peut pas commencer un vecteur à 0 ...
    for i=1:n
        u=u*q //on calcule par itération
        V(i+1)=V(i)+u //on passe d'une somme partielle à la suivante en
                      // ajoutant le terme qu'on vient de calculer
    end
    V=V' //on transpose V pour le mettre en ligne
    disp("une=",u)
    disp(V)
endfunction
```

Cette fonction calcule le terme de rang n de n'importe quelle suite géométrique définie par son terme initial et sa raison et stocke dans un vecteur ligne les valeurs des sommes partielles de la série correspondante jusqu'à l'ordre n.

Et un autre joli exemple pour conclure :

```
clf()
n=input("entrer le paramètre n de la loi binomiale ")
lambda=input("entrer le paramètre lambda de la Poisson ")
N=input("entrer le nombre de tirages de la simulation ")
p=lambda/n
//On réalise N tirages de chacune des 2 lois
//On stocke les valeurs obtenues dans des vecteurs
B=grand(1,N,'bin',n,p)
POIS=grand(1,N,'poi',lambda)
Br=-gsort(-B) //on classe les valeurs de B par ordre croissant
//on aurait pu utiliser la syntaxe gsort(B,'lc','i')
Pr=-gsort(-POIS)
plot2d2(Br,[1:N]/N)
plot2d2(Pr,[1:N]/N,2)
```

Ce script compare les fonctions de répartitions empiriques d'une binomiale (n,λ) (pour n grand) avec une loi de Poisson de paramètre λ. Et vous connaissez le résultat ...

Le minimum minimorum...

Et si vraiment vous n'avez décidément pas envie de passer du temps sur Scilab (c'est votre – mauvais – choix), je vous conseille au moins d'apprendre par cœur les algorithmes qui « tombent aux concours » (sic), ce qu'un petit coup d'œil aux annales récentes vous confirmera.

- Calcul d'une intégrale par la méthode des rectangles
- Calcul d'une intégrale par la méthode de Monte Carlo
- Calcul du n-ième terme d'une suite
- Calcul des sommes partielles (avec ou sans l'opérateur pointé)
- Approximation d'une limite
- Résolution d'une équation f(x)=0 par dichotomie
- Génération des lois classiques avec grand
- Simulation de lois par inversion de la fonction de répartition
- Fonction de répartition empirique
- Comparaison de lois par analyse :
  - Des fonctions de répartition empiriques
  - Des histogrammes ou diagrammes en bâtons
- Simulation d'une loi normale à partir d'une loi uniforme via le TCL
- Affichage du graphe d'une fonction de deux variables et traçage des lignes de niveaux

Cette liste n'est sans doute pas exhaustive mais elle me semble un viatique vaguement idoine. Et à la question « *où trouver ces algorithmes ?* » eh bien... dans le cours fait par votre professeur bien sûr ! Where else ?

Alors convaincu.e.s ? Des points qui se ramassent vraiment à la pelle. Toujours envie de faire l'impasse ?