Liste des savoir-faire en Informatique et Algorithmique (2)

En Informatique et Algorithmique en ECS1, je dois savoir aussi :

V Matrices et systèmes linéaires

1. construire une matrice par extension : on donne la liste des éléments de la matrice entre crochets et en les séparant par des virgules, chaque ligne étant séparée par des points virgules.
2. différencier une matrice/vecteur ligne (ce qu'on appelait simplement vecteur ou liste jusqu'à présent) d'une matrice/vecteur colonne.
3. construire une matrice vide avec `[]`.
4. construire une matrice par blocs.
5. construire des matrices avec les commandes `rand(n,p)`, `ones(n,p)`, `zeros(n,p)`, `eye(n,p)` et `diag(v)` lorsque `v` désigne un vecteur.
6. obtenir la taille d'une matrice `A` avec `size(A)` et différencier cette commande de `length(A)`.
7. accéder à la coordonnée d'indice (`i`,`j`) d'une matrice `A` avec la commande `A(i,j)`, et modifier ou supprimer sa valeur.
8. obtenir la transposée d'une matrice `A` avec `A'`.
9. effectuer des opérations matricielles (addition avec `+`, soustraction avec `-`, produit avec `*`, élévation à la puissance d'un nombre entier naturel `k` avec $\wedge$`k` ou `**`) lorsqu'elles ont un sens.
10. différencier les opérations matricielles des opérations coordonnées par coordonnées sur des matrices de même taille avec `.*`, `./`, `.**`, `.`$\wedge$.
11. effectuer des tests de comparaison sur des matrices de même taille. On obtient alors des matrices contenant des variables booléennes. La commande `find(M)` renvoie un vecteur contenant les positions des coefficients d'une matrice booléenne `M` ayant la valeur `T` (vrai) et la commande `length(find(M))` renvoie donc le nombre de coefficients d'une matrice booléenne `M` ayant la valeur `T` (vrai).
12. utiliser les commandes `min`, `max`, `sum`, `prod`.
13. calculer l'inverse d'une matrice inversible avec la commande `inv`.
14. calculer le rang d'une matrice avec la commande `rank`.
15. obtenir une base du noyau d'une matrice A codée en Scilab par `A` (c'est-à-dire une base du noyau de l'application linéaire $X \in \mathcal{M}_{p,1}(\mathbb{K}) \longmapsto AX$ si la matrice A possède p colonnes) avec la commande `kernel(A)`.
16. créer une matrice `A` associée à un système linéaire homogène et créer un vecteur colonne `B` associé au second membre.
17. résoudre ce système avec la commande `inv(A)*B` dans le cas où `A` est une matrice carrée inversible.
18. résoudre ce système avec la commande `[X0,noyau]=linsolve(A,-B)` sinon.
19. comprendre l'implémentation Scilab de l'algorithme du pivot de Gauss.

VI Résolution de manière approchée d'une équation du type $f(x) = 0$

20. savoir mettre en place un algorithme de recherche d'une solution particulière par dichotomie. Si on sait qu'une fonction `f` (préalablement implémentée en Scilab) s'annule sur $[$`a`$,$`b`$]$ en un point `x0`, alors l'algorithme suivant calcule un encadrement de `x0` à `eps`-près :

```
while  b-a>eps
       if  f((a+b)/2)==0   then
             a=(a+b)/2; b=a;
       elseif   f(a)*f((a+b)/2)<0   then
             b=(a+b)/2;
       else
             a=(a+b)/2;
       end
end
disp(string(a)+' <= x0 <= '+string(b)+'.')
```

21. construire un algorithme permettant de calculer une valeur approchée de `x0` en calculant les termes successifs d'une suite récurrente (dont on a préalablement montré théoriquement qu'elle converge vers `x0` avec une vitesse donnée, cf. méthode du point fixe). On renvoie pour cela au paragraphe IV.
22. ajouter un compteur dans un algorithme contenant une boule `while` afin de calculer le nombre d'itérations nécessaires.

VII Calcul approché d'une intégrale par la méthode des rectangles

23. mettre en place un algorithme de calcul des valeurs approchées d'une intégrale par la méthode des rectangles (utilisant les sommes de Riemann). Si on souhaite calculer l'intégrale d'une fonction `f` (préalablement implémentée en Scilab) sur un segment $[\mathtt{a}, \mathtt{b}]$, alors on utilise l'algorithme suivant :

```
S=0; T=0
for  k=1:n
     S=S+f(a+k*(b-a)/n);
     T=T+f(a+(k+1)*(b-a)/n);
end
S=S*(b-a)/n; T=T*(b-a)/n;
```

Plus précisément `S` (resp. `T`) contient une valeur approchée obtenue par la méthode des rectangles à gauche (resp. à droite). La variable `n` doit avoir été déterminée au préalable en fonction de la précision recherchée (il y a des formules dans le cas où `f` est monotone ou `f` est de classe C^1) ou sinon on prend `n` très grand.

24. calculer notamment une valeur approchée de $\Phi(x) = \displaystyle\int_{-\infty}^{x} e^{-t^2/2}\,\frac{dt}{\sqrt{2\pi}}$ pour $x \in \mathbb{R}$. Attention il faut d'abord se placer sur un segment :
    - On sait que, si $x \leqslant -4$ (resp. $x \geqslant 4$), alors $\Phi(x) \approx 0$ (resp. $\Phi(x) \approx 1$). Si $x \in [-4, 4]$, alors $\Phi(x) \approx \displaystyle\int_{-4}^{x} e^{-t^2/2}\,\frac{dt}{\sqrt{2\pi}}$.
    - On sait (cf. feuille d'exercices n° 28) que
        - Si $x \geqslant 0$, alors $\Phi(x) = \dfrac{1}{2} + \displaystyle\int_{0}^{x} e^{-t^2/2}\,\frac{dt}{\sqrt{2\pi}}$.
        - Si $x \leqslant 0$, alors $\Phi(x) = 1 - \Phi(-x)$.

    Voici par exemple un algorithme qui calcule $\Phi(\mathtt{x})$:

```
//Choix de a, b et n (avec n grand)
a=0; b=abs(x); n=1000;

//Définition de la fonction f
function y=f(t)
      y=exp(-t**2/2)/sqrt(2*%pi);
endfunction;

//Méthode des rectangles sur [a,b]
S=0;
for  k=1:n
     S=S+f(a+k*(b-a)/n);
end
S=S*(b-a)/n;
Phi=1/2+S;

//Calcul final
if   x>0   then
  disp(Phi);
else
  disp(1-Phi);
end
```

VIII Probabilités

1) Simulation de variables aléatoires

25. simuler une variable aléatoire de loi uniforme sur $[0,1]$ avec `X=rand()`.
26. simuler une variable aléatoire de loi uniforme sur $[a,b]$ avec `X=a+(b-a)*rand()`.
27. simuler une variable aléatoire de loi uniforme sur $[\![a,b]\!]$ avec `X=a+floor((b-a+1)*rand())`.
28. simuler une variable aléatoire de loi de Bernoulli de paramètre `p` avec `X=(rand()<p)`.
29. construire, pour les quatre loi précédentes, une matrice de taille `m`×`n` constituée de simulations de variables aléatoires indépendantes en remplaçant `rand()` par `rand(m,n)`.
30. simuler une variable aléatoire de loi binomiale de paramètres `n` et `p` avec `X=sum(rand(1,n)<p)`.
31. simuler une variable aléatoire de loi géométrique de paramètre `p` avec

    `X=1; while rand()>p; X=X+1; end; X` ou `X=0; while rand()>p; X=X+1; end; X=X+1`.

32. faire des simulations avec la fonction `grand`, notamment pour des variables aléatoires de loi de Poisson, de loi Normale et de loi exponentielle (la commande `X=-log(1-rand())/a` qui simule une v.a de loi exponentielle de paramètre `a` n'est pas clairement au programme).

2) Représentation graphiques de lois théoriques

33. construire un diagramme en bâton avec `plot2d3(val,prob)` ou `bar(val,prob)` où `val` est un vecteur contenant la liste des valeurs possibles d'une variable aléatoire réelle finie X et `prob` est un vecteur contenant les probabilités respectives de ces valeurs. Si X prend un nombre dénombrable de valeurs, alors on se limite aux valeurs que X prend avec des probabilités significatives ($> 10^{-3}$ par exemple).
34. tracer la fonction de répartition de X avec la commande `plot2d2(val,cumsum(prob))`.
35. tracer la courbe d'une densité et de la fonction de répartition d'une variable aléatoire de loi admettant une densité.

3) Représentation graphique de données empiriques

36. construire un tableau d'effectif à partir de réalisations indépendantes d'une variable aléatoire réelle finie X, stockées dans un vecteur Scilab `X` avec la commande `T=tabul(X,'i')`. La commande `T(:,2)/sum(T(:,2))` remplace les effectifs respectifs par les fréquences. La commande `plot2d3(T(:,1),T(:,2))` construit alors le diagramme en bâton empirique des réalisations de X. on peut ensuite lui superposer un diagramme en bâton d'une loi théorique (suggérée par l'énoncé).
37. réaliser un histogramme (à r classes) à partir de réalisations indépendantes d'une variable aléatoire réelle X, stockées dans un vecteur Scilab `X` avec la commande `histplot(r,X)`. On peut ensuite superposer la courbe d'une densité théorique (suggérée par l'énoncé) à l'histogramme.
38. obtenir le vecteur des sommes cumulées d'un autre vecteur `X` avec la commande `cumsum(X)`.
39. construire la fonction de répartition empirique de X à partir des commandes

    `T=tabul(X,'i'); T(:,2)=T(:,2)/sum(T(:,2)); plot2d2(T(:,1),cumsum(T(:,2)))`

    et lui superposer la courbe de la fonction de répartition d'une loi théorique (suggérée par l'énoncé).

4) Convergence et approximation de variables aléatoires

40. Simuler une variable aléatoire de loi de Poisson de paramètre $a > 0$ en simulant une variable aléatoire de loi binomiale de paramètre n et a/n avec n grand.
41. représenter graphiquement la loi des grands nombres et le théorème central limite (du moins les versions faibles vu en cours).
42. calculer une valeur approchée de la probabilité p d'un événement en réalisant n (avec n grand) fois une expérience de façons indépendantes et en comptant le nombre c d'expériences pour lesquelles l'événement en question a été réalisé (on approxime alors p par c/n en vertu de la loi faible des grands nombres pour la binomiale).