CORRIGE du TP 6

B. Landreau

Calculs de valeurs et vecteurs propres

```
> with(linalg):
Warning, the protected names norm and trace have been
redefined and unprotected
```

Exercice 1 : Methode de la puissance

On suit la methode de la puissance.

On introduit une variable μ qui permet de stocker la valeur de λ avant de la remplacer
par la valeur calculee au rang suivant.

On met la variable *erreur* a ∞ pour que a coup sur on rentre dans la boucle while.

Pour le produit scalaire, on peut utiliser *innerprod* ou *dotprod* (produit hermitien) .

ATTENTION :

- pour des varaiables structurees (tableau, vecteur, matrices...) les affectations se font avec *copy, on fait*
  A:=copy(B) et non A:=B,
- il faut imperativement forcer l'evaluation par un *evalm* a chaque
calcul avant affectation sinon il enregistre la formule telle quelle et s'il l'evalue un peu plus loin dans la procedure, il prendra alors la valeur presente des variables qui aura probablement change.

C’est une source frequente d’erreurs.
Il est bon de mettre un garde fou sur le nombre d’iterations maximum.
On pourrait utiliser la commande *normalize* qui rend un vecteur de norme 1 mais c’est uniquement pour la norme infinie.

```
> Puiss:=proc(M,X0,c)
  local u,v,erreur,lambda,mu,k,K;
  u:=copy(X0);
  v:=evalm(u/norm(u,infinity));
  mu:=1.;
  erreur:=infinity;
  k:=0;K:=100;
  while (erreur>c) and (k<=K) do
    k:=k+1;
    u:=evalm(M&*v);

  lambda:=evalf(innerprod(v,u)/innerprod(v,
  v));
    v:=evalm(u/norm(u,infinity));
    erreur:=abs((lambda-mu)/mu);
    mu:= lambda;
    #print(lambda,v);
  od;
  if k<=K then
    printf("\n plus grande valeur propre :
  %f",lambda);
    printf("\n vecteur propre : ");
    print(evalm(v));
    printf("\n iterations : %d",k);
    RETURN([lambda,v]);
  else
```

```
    printf("desole, la methode ne converge
  pas");
  fi;
  end:
```

Faisons un test sur une matrice dont on connait a priori le spectre.

```
> B:=vector(4,i->1.);
```

$$B := [1., 1., 1., 1.]$$

```
> P:=randmatrix(4,4):d:=diag(-7,3,4,1):
  M:=evalm(inverse(P)&*d&*P):
> Puiss(M,B,0.001);

 valeur propre : -6.998305
 vecteur propre :
```

[-.4227642198, -.5273074992, 1.000000000, .2846164907]

```
 iterations : 17
```

$$[-6.998305323, v]$$

Nouvel essai (avec non unicite de la plus grande valeur propre en module)

```
> P:=randmatrix(4,4):d:=diag(-7,7,4,1):
  M:=evalm(inverse(P)&*d&*P):
> Puiss(M,B,0.001);
desole, la methode ne converge pas
```

C’est normal.

Nouvel essai (avec la plus grande valeur propre multiple)

```
> P:=randmatrix(4,4):d:=diag(-7,-7,1,1):
  M:=evalm(inverse(P)&*d&*P):
> Puiss(M,B,0.0001);

 valeur propre : -7.000043
```

```
 vecteur propre :
```

[-.01124026215, .4333233424, -1.000000000, -.2925690237]

```
 iterations : 7
```

$$[-7.000042977, v]$$

Ca marche bien!

Nouvel essai au hasard.

```
> M:=randmatrix(4,4);
```

$$M := \begin{bmatrix} -80 & -44 & -21 & -5 \\ 0 & 21 & -92 & -70 \\ -95 & -47 & 51 & 47 \\ 5 & 33 & 9 & 26 \end{bmatrix}$$

```
> Puiss(M,B,0.0001);

 valeur propre : -113.257834
 vecteur propre :
```

[-1.000000000, -.4188875167, -.7504709970, .1836026024]

```
 iterations : 17
```

$$[-113.2578339, v]$$

```
>
> evalf(Eigenvals(M));
```

$$[-113.2650707, 61.27503142 + 27.22302859\,I,$$

$$61.27503142 - 27.22302859\,I, 8.715008254]$$

Pas mal!

Exercice 2 : Methode de la puissance

inverse

On suit la methode de la puissance inverse. Ce qui change, c'est juste de remplacer A par $A^{(-1)}$.

De plus, on evite de calculer $A^{(-1)}$ on resoud plutot le systeme Au=v par linsolve par exemple.

Il ne faut pas oublier non plus de sortir $\frac{1}{\lambda}$ au lieu de λ.

```
> Puiss_inverse:=proc(M,X0,c)
  local u,v,erreur,lambda,mu,k,K;
  u:=copy(X0);
  v:=normalize(u);
  mu:=1.;
  erreur:=infinity;
  k:=0;K:=100;
  while (erreur>c) and (k<=K) do
    k:=k+1;
    u:=linsolve(M,v);

  lambda:=evalf(innerprod(v,u)/innerprod(v,
  v));
    v:=normalize(u);
    erreur:=abs((lambda-mu)/mu);
    mu:= lambda;
    #print(lambda,v);
  od;
  if k<=K then
    printf("\n plus petite valeur propre :
  %f",1./lambda);
    printf("\n vecteur propre : ");
```

```
    print(evalm(v));
    printf(" iterations : %d",k);
    RETURN([1./lambda,v]);
  else
    printf("desole, la methode ne converge
  pas");
    RETURN(0);
  fi;
  end:
> B:=vector(4,i->1.);
```

$$B := [1., 1., 1., 1.]$$

Faisons un test sur une matrice dont on connait a priori le spectre.

```
> P:=randmatrix(4,4):d:=diag(-7,3,4,1):
  M:=evalm(inverse(P)&*d&*P):

> Puiss_inverse(M,B,0.01);

 plus petite valeur propre : 1.004227
 vecteur propre :
```

$$[.4320899129, .3509101423, .2874362438, .7794490264]$$

```
 iterations : 8
```

$$[1.004226850, v]$$

Nouvel essai

```
> P:=randmatrix(4,4):d:=diag(12,5,1,1):
  M:=evalm(inverse(P)&*d&*P):
> Puiss_inverse(M,B,0.01);

 plus petite valeur propre : .999404
 vecteur propre :
```

$$[-.1759440411, .3536908792, .7966616164, -.4574677324]$$

```
 iterations : 5
```

$$[.9994037917, v]$$

Nouvel essai

```
> P:=randmatrix(4,4):d:=diag(12,5,-1,1):
  M:=evalm(inverse(P)&*d&*P):
> Puiss_inverse(M,B,0.01);
desole, la methode ne converge pas
```

$$0$$

Nouvel essai aleatoire

```
> M:=randmatrix(4,4);
```

$$M := \begin{bmatrix} 65 & 81 & 80 & -19 \\ -54 & 54 & -78 & 27 \\ 98 & -22 & 33 & 74 \\ 83 & 13 & -66 & -77 \end{bmatrix}$$

```
> Puiss_inverse(M,B,0.001);
desole, la methode ne converge pas
```

$$0$$

```
> evalf(Eigenvals(M));
```

$$[-72.07790100 + 85.73023315\,I,$$

$$-72.07790100 - 85.73023315\,I, 109.5779008 + 60.83886350\,I,$$

$$109.5779008 - 60.83886350\,I]$$

OK !

Exercice 3 : Methode generale

On va utiliser a procedure precedente sur les $M - \gamma\, Id$ ou γ est une valeur approchee d'une valeur propre.

On prendre a defaut d'informations particulieres les centres des disques de Gerschgorin.

Il ne faut pas oublier de rajouter γ a la valeur trouvee.

Pour savoir si la procedure *Puiss_inverse* a converge il suffit de savoir si elle a retourne 0 ou non.

Je modifie legerement la procedure *Puiss_inverse* pour eviter qu'elle imprime le texte.

```
> Puiss_inverse:=proc(M,X0,c)
  local u,v,erreur,lambda,mu,k,K;
  u:=copy(X0);
  v:=normalize(u);
  mu:=1.;
  erreur:=infinity;
  k:=0;K:=100;
  while (erreur>c) and (k<=K) do
    k:=k+1;
    u:=linsolve(M,v);

  lambda:=evalf(innerprod(v,u)/innerprod(v,
  v));
    v:=normalize(u);
    erreur:=abs((lambda-mu)/mu);
    mu:= lambda;
  od;
  if k<=K then
      RETURN([1./lambda,v,k]);
  else
     RETURN(0);
  fi;
  end:
```

La procedure *vecteurs_propres*

```
> VecteursPropres:=proc(M,c)
  local i,n,W,B;
  n:=rowdim(M);
  B:=vector(n,i->1);
  for i to n do
   W:=Puiss_inverse(M-M[i,i],B,c);
   if W<>0 then
    printf("\n**********\n valeur propre :
  %f",W[1]+M[i,i]);
    printf("\n vecteur propre : ");
    print(evalm(W[2]));
    printf(" iterations : %d",W[3]);
   fi;
  od;
  end:
```

Essais, attention on n'est pas du tout assure que ca marche.
Ca marche bien si tous les disques de Gerschgorin sont disjoints.

```
> P:=randmatrix(4,4):d:=diag(12.,5,3,1):
  M:=evalm(inverse(P)&*d&*P);
```

$M :=$

$$[15.92049462, -24.58200778, 26.80255398, -7.672805342]$$

$$[-13.38576417, 80.95329077, -75.79994355, 18.08638176]$$

$$[-14.45591865, 75.88870920, -71.31635556, 17.94615299]$$

$$[25.30552574, -9.306139139, 17.04544292, -4.557429842]$$

```
> VecteursPropres(M,0.0001);

**********
 valeur propre : 12.000214
 vecteur propre :
  [.1909548149, -.7077754050, -.6800594981, -.01045533230]
```

```
 iterations : 11
**********
 valeur propre : 12.066807
 vecteur propre :
```

[.1897146842, -.7079683964, -.6801509896, -.01355432823]

```
 iterations : 57
**********
 valeur propre : .880397
 vecteur propre :
```

[-.2701264241, .6396000295, .6494337419, -.3101279276]

```
 iterations : 73
**********
 valeur propre : .999452
 vecteur propre :
```

[-.2695199796, .6419414911, .6542444539, -.2953545270]

```
 iterations : 17
>
```

Exercice 4 : Essais

```
> M1:=matrix([[14.,0,1],[1,-1,0],[-1,1,5]])
  ;evalf(Eigenvals(M1));
```

$$M1 := \begin{bmatrix} 14. & 0 & 1 \\ 1 & -1 & 0 \\ -1 & 1 & 5 \end{bmatrix}$$

[13.89512641, -.988982994, 5.093856586]

Dans ce cas, ca devrait marcher, il y a une seule valeur propre par disque.

```
> VecteursPropres(M1,0.0001);

**********
 valeur propre : 13.895126
 vecteur propre :
```

```
        [.9923361518, .06662153456, -.1040698459]
 iterations : 4
**********
 valeur propre : -.988983
 vecteur propre :
        [.01086915547, .9865798100, -.1629175864]
 iterations : 4
**********
 valeur propre : 5.093857
 vecteur propre :
       [-.1115621737, -.01830727513, .9935888107]
 iterations : 4
```

Youpi!

```
> M2:=matrix([[4.,0,1],[1,-1,0],[-1,1,5]]);
  evalf(Eigenvals(M2));
```

$$M2 := \begin{bmatrix} 4. & 0 & 1 \\ 1 & -1 & 0 \\ -1 & 1 & 5 \end{bmatrix}$$

$[4.483682575 + .7558485668\,I,\ 4.483682575 - .7558485668\,I,$

$-.9673651423]$

Dans ce cas, ca ne devrait pas marcher, il y a 2 valeurs propre pres de 4 mais elles sont complexes conjuguees.

```
> VecteursPropres(M2,0.0001);

**********
 valeur propre : 10.000000
 vecteur propre :
        [-.9733285269, -.1622214211, -.1622214211]
 iterations : 2
**********
```

```
valeur propre : -.967365
vecteur propre :
        [.03219761506, .9866019318, -.1599373065]
iterations : 4
```

On retrouve bien la valeur propre -.967365 mais l'autre est completement erronee.
Il est meme curieux que celadonne quelque chose en dehors.
Pouquoi 10 ? A vous de trouver.

```
>
>
```

Exercice 5 : (supplement) methode QR

On utilise ici la fonction QRdecomp. Attention a la syntaxe.
QRdecomp retourne la matrice R et le second argument Q=' q' a pour effet de mettre
la valeur de Q calculee par cette procedure dans la variable q.
Attention, on ne peut pas faire Q='Q', il faut utiliser une autre etiquette.
l'algorithme suit la methode exposee en cours, la suite des matrices A_i converge vers une matrice triangulaire superieure dont les elements diagonaux sont les valeurs propres de A.

```
> QR:=proc(A,n)
  local B,q,R,k;
  B:=copy(A);
  for k to n do
    R:=QRdecomp(B,Q='q');
    B:=evalm(R&*q);
  od;
  RETURN(evalm(B));
  end:
```

Prenons une matrice aleatoire inversible *P*, une matrice diagonale *d* et considerons $P^{\{-1\}}\, d\, P$.

```
> P:=randmatrix(4,4);det(P);
```

$$P := \begin{bmatrix} 43 & -62 & 77 & 66 \\ 54 & -5 & 99 & -61 \\ -50 & -12 & -18 & 31 \\ -26 & -62 & 1 & -47 \end{bmatrix}$$

$$-38052263$$

```
> d:=diag(1,2,3,4);A:=evalm(1.*inverse(P)&*
  d&*P);
```

$$d := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$A :=$

[3.018271292 , .4998823329 , .08991241861 , -1.174006576]

[.5013132596 , 2.998549285 , .3165684522 , 1.769584584]

[-.6109557794 , .4269666695 , 1.648156116 , .9361686846]

[-.1312219460 , 1.053616391 , -.5173789533 , 2.335023307]

```
> QR(A,20);
```

[4.006453954 , -.5420967678 , -1.239814512 , .3404788961]

[.01198029902 , 2.993707122 , -1.419179751 , .2621160337]

[$-.9983820582\ 10^{-6}$, .0001144516948 , 1.999839154 ,

.4159938619]

[-.4594794974 10^{-12}, .1864607926 10^{-9}, -.5640006373 10^{-6},

.9999997653]

On voit apparaitre en effet les valeurs propres 4,3,2,1 en diagonale.

>

FIN