Tronc Commun Maple
Opérations sur les matrices

Corrigé

25 novembre 2005

1 Somme matricielle

1.

```
somme_matricielle := proc(A,B)
  local res, i, j ;
  res := evalm(A) ;
  for i from 1 to rowdim(A) do
    for j from 1 to coldim(A) do
      res[i,j] :=A[i,j]+B[i,j] ;
    od ;
  od ;
  evalm(res) ;
end ; ;
```

2. `matadd(A,B) ; ;`

2 Produit matriciel

1.

```
produit_matriciel := proc(A,B)
  local res,i,j, k,terme ;
  res := evalm(A) ;
  for i from 1 to rowdim(A) do
    for j from 1 to coldim(A) do
      terme := 0 ;
```

```
        for k from 1 to coldim(A) do
          terme := (terme + A[i,k] * B[k,j]) ;
        od ;
        res [i,j] := terme ;
      od ;
    od ;
    evalm(res) ;
  end ; ;
```

2. Pour multiplier une matrice $m \times n$ par une matrice $n \times p$ en appliquant la définition du produit matriciel, on effectue mpn multiplications et $mp(n-1)$ additions (pour être précis il faudrait en faite compter entre autres les incrémentations des variables de boucles, mais on s'en moque). Cela donne donc du $\Theta(mnp)$.
3. `multiply(A,B) ; ;`

3 Calcul de déterminant

3.1 Méthode naïve : le développement par rapport à une ligne (ou une colonne)

1.

```
extrait := proc (A,a,b)
  local res,n,i,j ;
  n := rowdim(A) ;
  res :=matrix(n-1,n-1) ;
  for i from 1 to a-1 do
    for j from 1 to b-1 do
      res[i,j] :=A[i,j] ;
    od ;
    for j from b+1 to n do
      res[i,j-1] :=A[i,j] ;
    od ;
  od ;
  for i from a+1 to n do
    for j from 1 to b-1 do
      res[i-1,j] :=A[i,j] ;
    od ;
    for j from b+1 to n do
      res[i-1,j-1] :=A[i,j] ;
    od ;
  od ;
  evalm(res) ;
end ; ;
```

```
determinant_developpement := proc(A)
  local res,n,signe,i ;
  n := rowdim(A) ;
  if n = 1 then A[1,1] ;
  else
    res := 0 ;
    for i from 1 to n do
      signe := (-1)^(i+1) ;
      res :=res+((A[i,1]*signe)
        *determi-
nant_developpement(extrait(A,i,1))) ;
    od ;
    eval(res) ;
  fi ;
end ; ;
```

2. L'algorithme procède de la manière suivante : calcul de n déterminants d'ordre $n-1$, suivi de n additions. On obtient donc $\forall n, T(n) > nT(n-1) + Kn$ pour une certaine constante K. On vérifie facilement qu'il existe une constante K' telle que $\forall n, T(n) > K'n!$. De plus on montre de même que $\exists K'', \forall n, T(n) < K''n!$. La complexité de cet algorithme est donc $\Theta(n!)$. Il est donc inutilisable dès que la taille de la matrice dépasse quelques unités.
3. `det(A)`

3.2 Méthode efficace : le pivot de Gauss

1.

```
triangule := proc (A)
  local piv,abs_piv,n,i,j,k,l,nv_val ;
  n :=rowdim(A) ;
  for i from 1 to n do
    piv := i ;
    abs_piv :=abs(A[piv,i]) ;
    for j from i+1 to n do
      if (abs(A[j,i])>abs_piv) then
        piv :=j ;
        abs_piv := abs(A[piv,i]) ;
      fi ;
    od ;
    if (piv = 0) then RE-
TURN(evalm(A),false) fi ;
    for j from i+1 to n do
      nv_val := -1*A[j,i]/A[i,i] ;
      for l from i to n do
        A[j,l] := A[j,l] + nv_val * A[i,l] ;
      od ;
    od ;
  od ;
  (evalm(A),true) ;
end ; ;
```

```
determinant_gauss := proc(A)
  local a,b,i,res ;
  (a,b) := triangule(A) ;
  if (not b) then RETURN(0)
  else
    res :=1 ;
    for i from 1 to coldim(A) do
      res :=res*A[i,i] ;
    od ;
    res ;
  fi ;
end ; ;
```

2. À la $k^{\text{ème}}$ itération, on soustraie la $^{\text{ème}}$ ligne (multipliée par un coefficient adéquat) à $n-k$ autres lignes, ce qui donne du $\Theta((n-k)^2)$ et au total du $\Theta(n^3)$.

4 Inversion de matrice

1.
```
inverse_gauss := proc(A)
  local i,j,k,temp,n,W,I ;
  n := rowdim(A) ;
  W :=matrix(n,2*n) ;
  I :=matrix(n,n) ;

  for i from 1 to n do
    for j from 1 to n do
      W[i,j] :=A[i,j] ;
      if i = j then
        W[i,j+n] := 1 ;
      else
        W[i,j+n] :=0 ;
      fi ;
    od ;
  od ;

  for i from 1 to n do
   j :=i ;
   while W[j,i]=0 do
     j :=(j+1) ;
     if j>n then RETURN(evalm(I)) fi ;
   od ;

   if i<>j then
     for k from i to 2*n do
       temp :=W[i,k] ;
       W[i,k] :=W[j,k] ;
       W[j,k] :=temp ;
     od ;
   fi ;

   if W[i,i]<>1 then
     temp := W[i,i] ;
     for j from i to 2*n do
       W[i,j] :=W[i,j]/temp ;
     od ;
   fi ;
   for j from i+1 to n do
     if W[j,i] <> 0 then
       temp :=W[j,i] ;
       for k from i to 2*n do
         W[j,k] :=W[j,k]-W[i,k]*temp ;
       od ;
     fi ;
   od ;
  od ;

   for i from 0 to (n-2) do
     for j from 1 to ((n-i)-1) do
       if W[j,(n-i)]<>0 then
         temp :=W[j,(n-i)] ;
         for k from (n-i) to 2*n do
           W[j,k] :=W[j,k]-W[(n-
i),k]*temp ;
         od ;
       fi ;
     od ;
   od ;


   for i from 1 to n do
     for j from 1 to n do
       I[i,j] :=W[i,j+n] ;
     od ;
   od ;

   evalm(I) ;

 end ; ;
```
2. inverse(A).