```
#load "nums.cma";;
open Num;;

(* Polynome caractéristique et interpolation de Lagrange *)


let zero = num_of_int(0);;
let un   = num_of_int(1);;


let copie m =
  let n = Array.length m
  and p = Array.length m.(0) in
  let t = Array.make_matrix n p m.(0).(0) in
  for i=0 to n-1 do
    for j=0 to p-1 do
      t.(i).(j) <- m.(i).(j)
    done
  done;
  t;;


(* Déterminant par pivot *)
let det(t) =
  let m = copie t in
  let n = Array.length m in
  let d = ref un in
  let j = ref 0 in
    while (!j < n) && (!d <>/ zero) do
      let i = ref (!j) in
        while (!i < n) && (m.(!i).(!j) =/ zero) do i := !i+1 done;
        if !i < n then
          begin
            if !i > !j then
              begin
                for k = !j to n-1 do
                  let x = m.(!i).(k) in m.(!i).(k) <- m.(!j).(k); m.(!j).(k) <- x
                done;
                d := minus_num (!d);
              end;
            d := !d */ m.(!j).(!j);
            for k = n-1 downto !j do m.(!j).(k) <- m.(!j).(k)//m.(!j).(!j) done;
            for l = !j+1 to n-1 do
              for k = n-1 downto !j do m.(l).(k) <- m.(l).(k) -/ m.(l).(!j)*/m.(!j).(k) done
            done
          end
        else d := zero;
        j := !j + 1;
    done;
    !d ;;

let m = Array.map (Array.map num_of_int)
          [| [| 1;2;3 |];
             [| 4;5;6 |];
             [| 7;8;9 |] |];;

det m;;

let m' = Array.map (Array.map num_of_int)
          [| [| 1;0;0 |];
             [| 0;2;0 |];
             [| 0;0;1 |] |];;
det m';;

(* Evaluation d'un polynôme *)
let horner p x =
  let n = Array.length(p) in
  let y = ref(zero)      in
  for i=n-1 downto 0 do y := !y */ x +/ p.(i) done;
  !y
;;
```

```
(* Polynôme d'interpolation de Lagrange *)
let lagrange x y =
  let n = Array.length x   in
  let p = Array.make n zero in
  let q = Array.make n zero in

  p.(0) <- y.(0); q.(0) <- un;
  for i=1 to n-1 do
    for j = i downto 1 do q.(j) <- q.(j-1) -/ x.(i-1)*/q.(j) done;
    q.(0) <- minus_num(x.(i-1) */ q.(0));
    let z = (y.(i) -/ (horner p x.(i))) // (horner q x.(i)) in
    for j=0 to i do p.(j) <- p.(j) +/ z*/q.(j) done;
  done;
  p;;


lagrange (Array.map num_of_int [|0;1;2;-1|])
  (Array.map num_of_int [|-1;2;11;-4|]);;

(* polynôme caractéristique *)

let add_diag m x =
  let n = Array.length (m)       in
  let p = Array.make_matrix n n zero in
  for i=0 to n-1 do for j=0 to n-1 do p.(i).(j) <- m.(i).(j) done done;
  for i=0 to n-1 do p.(i).(i) <- p.(i).(i) +/ x done;
  p
;;


let poca_2 m =
  let n = Array.length(m)       in
  let x = Array.make (n+1) zero in
  let y = Array.make (n+1) zero in
  for i=0 to n do
    x.(i) <- num_of_int i;
    y.(i) <- det(add_diag m (minus_num x.(i)))
  done;
  lagrange x y
;;

poca_2(m);;
poca_2 m';;

let m_test =
  let a = Array.make_matrix 10 10 zero in
    for i=0 to 9 do
      for j=0 to 9 do
        a.(i).(j) <- num_of_int (((i+1)*(j+2)*17+3)mod 9)
      done
    done;
    a;;

poca_2 m_test;;

(* Méthode de Fadeev *)

let tr(m) =
  let n = Array.length(m) in
  let t = ref(zero)      in
  for i=0 to n-1 do t := !t +/ m.(i).(i) done;
  !t
;;

let prod a b =
  let n =  Array.length(a)       in
  let p =  Array.length(a.(0))   in
  let q =  Array.length(b.(0))   in
  let c = Array.make_matrix n q zero in
  for i=0 to n-1 do for j=0 to q-1 do
    for k=0 to p-1 do c.(i).(j) <- c.(i).(j) +/ a.(i).(k)*/b.(k).(j) done
```

```
  done done;
  c
;;


let poca_1 m =
  let n =  Array.length(m)       in
  let p = Array.make (n+1) zero in
  p.(n) <- un;
  let a = ref(m) in
  for i=1 to n do
    p.(n-i) <- minus_num ((tr !a) // (num_of_int i));
    a := prod m (add_diag !a p.(n-i))
  done;
  p
;;

poca_1 m;;
poca_1 m';;




(* Manipulation de sev de Q^n *)

let transpose(m) =
  let n = Array.length(m)
  and p = Array.length(m.(0))        in
  let t = Array.make_matrix p n m.(0).(0) in
  for i=0 to n-1 do for j=0 to p-1 do t.(j).(i) <- m.(i).(j) done done;
  t
;;

let concat a b =
  let n = Array.length(a)
  and p = Array.length(a.(0))
  and q = Array.length(b.(0)) in
  let t = Array.make_matrix n (p+q) a.(0).(0) in
  for i=0 to n-1 do
    for j=0 to p-1 do t.(i).(j)   <- a.(i).(j) done;
    for j=0 to q-1 do t.(i).(j+p) <- b.(i).(j) done
  done;
  t
;;

concat m m';;


(* Echelonnement par rapport aux lignes *)
let echelonne(t) =

  let m = copie(t)           in
  let n = Array.length(m)
  and p = Array.length(m.(0)) in

  let l = ref(0)             in
  for j = 0 to p-1 do

    let i = ref(!l) in
    while (!i < n) && (m.(!i).(j) =/ zero) do i := !i+1 done;

    if !i < n then begin
      if !i > !l then (let x = m.(!i) in m.(!i) <- m.(!l); m.(!l) <- x);
      for k = p-1 downto j do m.(!l).(k) <- m.(!l).(k)//m.(!l).(j) done;
      for a = 0 to n-1 do if a <> !l then
        for k = p-1 downto j do m.(a).(k) <- m.(a).(k) -/ m.(a).(j)*/m.(!l).(k) done
      done;
      l := !l+1;
    end

  done;
  m
```

```
;;

let m2 = Array.map (Array.map num_of_int) [| [| 1;2;3 |]; [| 2;4;5 |]; [| 3;6;7 |] |];;
echelonne(m);;
echelonne(m2);;

let matrice_grosse = Array.map (Array.map num_of_int) [| [| 1;-1;2;5;2 |]; [| 2;4;0;1;3 |]; [|
3;-2;-1;6;7 |] |];;
echelonne(matrice_grosse);;


(* Extrait les pivots et les inconnues secondaires *)
let pivots(m) =
  let n = Array.length(m)
  and p = Array.length(m.(0))     in
  let r = Array.make p 0          in
  let i = ref(0) and j = ref(-1) in
  while (!j < p) && (!i < n) do
    j := !j + 1;
    while (!j < p) && (m.(!i).(!j) =/ zero) do j := !j+1 done;
    if !j < p then begin r.(!i) <- !j; i := !i+1 end;
  done;
  Array.sub r 0 !i
;;

let seconds(m) =
  let n = Array.length(m)
  and p = Array.length(m.(0)) in
  let r = Array.make p 0      in
  let k = ref(0)
  and i = ref(0)
  and j = ref(-1)            in

  while (!j < p) && (!i < n) do
    j := !j + 1;
    while (!j < p) && (m.(!i).(!j) =/ zero) do
      r.(!k) <- !j; k := !k + 1; j := !j + 1
    done;
    i := !i + 1
  done;
  j := !j + 1;
  while !j < p do r.(!k) <- !j; j := !j + 1; k := !k + 1 done;
  Array.sub r 0 !k
;;

pivots(echelonne(m2));;
seconds(echelonne(m2));;

(* Extrait une base d'une famille génératrice *)
let base(m) =
  let n   = Array.length(m)
  and piv = pivots(echelonne(m)) in
  let r   = Array.length(piv)     in
  let t   = Array.make_matrix n r zero in
  for i=0 to n-1 do for j=0 to r-1 do t.(i).(j) <- m.(i).(piv.(j)) done done;
  t
;;

base(m2);;

(* Dimension *)

let dim a = Array.length(pivots(echelonne a));;

dim m2;;

(* Somme *)

let somme a b = base(concat a b);;

(* Inclusion *)
```

```
let inclus a b = dim a = dim (somme a b);;


(* Noyau d'une application linéaire *)
let noyau(m) =
  let p   = Array.length(m.(0))
  and t   = echelonne(m)       in

  let piv = pivots(t)
  and snd = seconds(t)         in
  let r   = Array.length(piv)   in

  let  s = Array.make_matrix p (p-r) zero in
  for i=0 to r-1   do for j=0 to p-r-1 do s.(piv.(i)).(j) <- t.(i).(snd.(j)) done done;
  for j=0 to p-r-1 do s.(snd.(j)).(j) <- num_of_int(-1) done;
  s
;;

noyau(m2);;

(* Orthogonal *)

let orth a = noyau(transpose a);;

orth m2;;

(* Intersection *)

let inters a b = orth(somme (orth a) (orth b));;

let egal a b = let d = dim(somme a b) in (d = dim a) && (d = dim b);;

let a = Array.map (Array.map num_of_int)
        [| [| 1;2;3 |];
           [| 2;3;4 |];
           [| 4;5;6 |];
           [| 0;1;2 |] |];;
let b = Array.map (Array.map num_of_int)
        [| [| 1 |];
           [| 1 |];
           [| 1 |];
           [| 1 |] |];;
let c = Array.map (Array.map num_of_int)
        [| [| 1;2 |];
           [| 1;3 |];
           [| 1;5 |];
           [| 1;1 |] |];;

dim(a), dim(b), dim(c);;
base(a), base(b), base(c);;
somme a b;;
somme a c;;
inters a b;;
Array.map (Array.map string_of_num) (inters a c);;
orth a;;
orth b;;
orth c;;
egal a b, egal a c;;
echelonne(a);;
```