Racines de polynôme, valeurs propres de matrice et inversion

Arthur Barjot

Avril 2022

Contents

1 Introduction

J'ai souhaité pour un usage pratique implémenter des fonctions sur les matrices en Ocaml, notamment l'inversion ou le calcul des valeurs propres. Mais la méthode de détermination de valeurs propres passe par l'étude du polynôme caractéristique, et donc plus généralement, les valeurs propres étant les racines d'un polynôme, il me faut un programme capable de déterminer celle-ci. J'ai donc développé une méthode pour déterminer les racines de n'importe quel polynôme et je vais présenter celle-ci dans un premier temps. Dans un second temps nous construirons des programmes capables de déterminer le polynôme caractéristique d'une matrice, toutes ses valeurs propres mais aussi son inverse lorsqu'il existe. Je profite de ce projet pour m'exercer à Latex, notamment le module Tikz. J'atteste avoir écrit, créé et eu l'idée seul de tout ce projet.

2 Polynôme

Nous allons voir tout d'abord une première partie informatique sur l'implémentation des polynôme en Ocaml, puis une deuxième partie mathématique sur une méthode d'obtention numérique de racines d'un polynôme, et dans la troisième partie nous implémenterons cette méthode. Enfin nous observerons les résultats de mon programme pour un exemple concret.

2.1 Implémentation

J'ai choisi d'implémenter les polynôme sur Ocaml sous la forme d'une liste de flottants. Le terme de plus haut degré étant en tête de liste, exemple :

$$[22;0;54;3;0;40] \text{ représente } 22x^5 + 54x^3 + 3x^2 + 40$$

On a donc directement les première propriétés :

- Si la liste commence par des zéros, ils sont inutiles et peuvent être supprimés (on appellera cette opération la "normalisation" du polynôme) : [0;0;34;456] = [34;456]

- On connaît rapidement le degré d'un polynôme normalisé, c'est la longueur de la liste moins 1 (avec la convention que l'on adoptera ici : deg(0) = -1) : deg([28;201;34;0;0;2]) = 5

On implemente donc facilement ces deux fonctions :

```
normalise p
deg p
```

On s'intéresse ensuite aux opérations de base, la somme et la multiplication de deux polynômes. Pour la somme de P1 et P2, le principe est de regarder la différence de degré entre les deux polynômes. Supposons deg(P1) > deg(P2) alors on sait que les termes de degré n > deg(P2) du polynôme à renvoyer seront exactement ceux de P1. On ajoute donc ceux-ci au polynôme à renvoyer jusqu'à arriver au termes de degré n = deg(P2), à partir de cet instant, on somme les

termes de degré n de P1 et de P2 et on les ajoute au polynôme à renvoyer. Pour la multiplication, c'est plus délicat car nous avons choisi des listes et non des tableaux pour implémenter nos polynômes. On procède comme dans cet exemple : P1 = [1;3;45] = $x^2 + 3x + 45$ et P2 = [3;56] = $3x + 56$,

P1 × P2 = $x^2\times$ P2 + $3x\times$ P2 + 45 P2 = $1\times$ [3;56;0;0] + $3\times$ [3;56;0] + $45\times$ [3;56]

On va en fait simplement implémenter un fonction qui ajoute n zéros à la fin du polynôme P2, cette opération correspondant à la multiplication par x^n. Puis il nous faut une fonction de multiplication d'un polynôme par un scalaire (multiplication terme à terme).

```
somme p1 p2
ajt0 p k l
mult_scal k poly
mult p1 p2
```

Enfin nous implémentons une fonction qui évalue un polynôme en un point utilisant la méthode de Horner, que je ne vais pas détailler ici. Puis un fonction qui dérive un polynôme (dérivation terme à terme).

```
eval poly x
deriv poly
```

2.2 Racines d'un polynôme

Mon idée est basée sur la recherche dichotomique de racines sur un segment [a,b] (a<b des réels), mais un problème se pose, pour appliquer un algorithme de dichotomie de recherche de zéros d'une fonction, il faut que celle-ci soit monotone, sinon on a de grande chance de ne pas avoir de résultat ou que ceux-ci soient incomplets.

- Méthode dichotomique de recherche de zéro d'une fonction monotone :

  Prenons f une fonction croissante de a vers b, $a < b$. Dans notre cas ici, un polynôme (mise à part les polynôme constant) est strictement monotone entre deux extremums, d'où on prend f strictement croissante, si elle admet une racine alors celle-ci est unique. Le principe est le suivant :

  -Si $f(a)$ et $f(b)$ sont du même signe, on a :

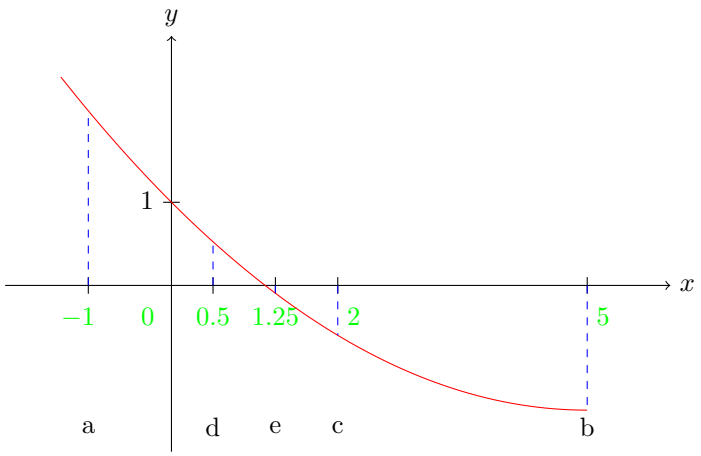
  $\forall x \in [a,b], f(a) \leq f(x) \leq f(b) \Rightarrow f(x)$ du signe de $f(a)$ et $f(b)$. D'où la fonction f ne peux avoir que a comme racine.

  -Si $f(a)$ et $f(b)$ de signe différent, alors $f(a) \leq 0$ et $f(b) \geq 0$. Par théorème des valeurs intermédiaires, tout polynôme étant continu, $\exists x \in [a,b] : f(x) = 0$.

  On prend $c = \frac{a+b}{2}$ le milieu du segment [a,b]. Si f(c) < 0 alors x se trouve dans [c,b], sinon f(c)≥ 0 et x se trouve dans [a,c]. Avec cette méthode répétée de nombreuses fois, l'intervalle de recherche diminue strictement (de manière exponentielle), jusqu'à ce que la valeur soit une suffisamment

bonne approximation (dans l'algorithme que je vais présenter la précision est de 0.0000000000001). Illustration :

On cherche un zéro entre -1 et 5 du polynôme $0.1x^2 - x + 1$ (minimum atteint en 5 donc ce polynôme est strictement décroissant sur [-1,5].



On voit bien que la suite a, b, c, d, e,... tend vers la première racine de $0.1x^2 - x + 1$, celle-ci valant $\frac{1-\sqrt{0.6}}{0.2} \approx 1.127$.

- Découpe de l'intervalle [a,b] en sous intervalles sur lesquels le polynôme est croissant :

  Mon idée pour trouver toutes les racines de tout polynôme est de travailler par récurrence sur le degré du polynôme:

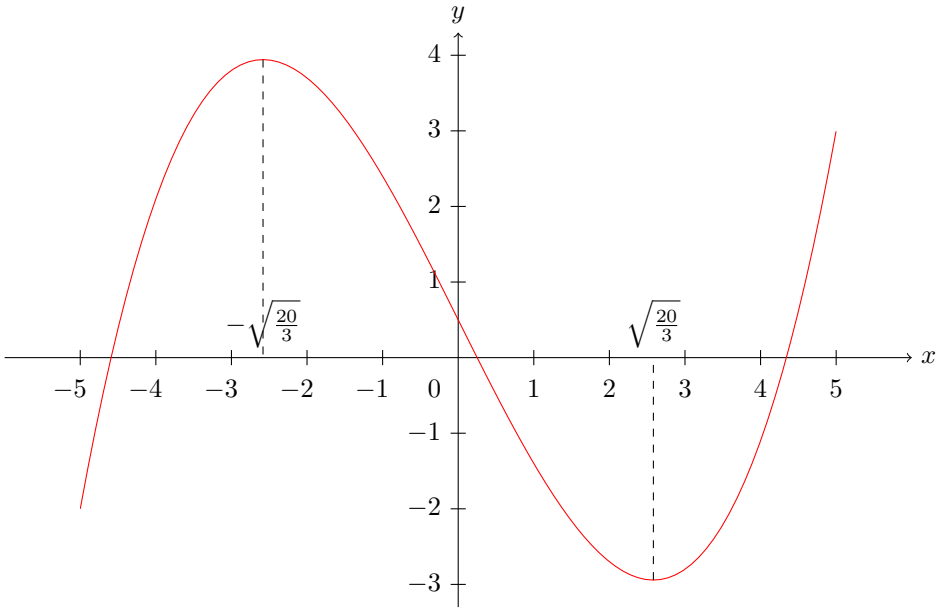
  - Initialisation : Pour un polynôme P de degré 1, celui-ci admet une et une seule racine. En notant $P = \alpha x + \beta$ avec $\alpha \neq 0$, l'unique racine est $\frac{-\beta}{\alpha}$.

  - Hérédité : Supposons que nous ayons un algorithme capable de déterminer toute les racines de tout polynôme de degré $n \in N^*$. Soit P un polynôme de degré n+1, alors sa dérivée P' est un polynôme de degré n. D'où avec notre algorithme on détermine toute les racines de P' : $[r_1, r_2, ..., r_k]$. Avec ces racines on peut dresser le tableau de variation de P, et donc on connaît tout les intervalles sur lesquels P est monotone. Ensuite sur chacun de ces intervalles $[r_i, r_{i+1}]$, on applique l'algorithme de dichotomie pour trouver l'unique racine correspondante. Il reste encore deux potentielles racines sur $]-\infty; r_1]$ et sur $[r_k; +\infty[$, le problème est que notre algorithme de dichotomie n'intervient que sur des segments. Sur $]-\infty; r_1]$ par exemple, il est simple de connaître la monotonie en comparant $r_1 - 1$ et r_1. Si P est croissant sur $]-\infty; r_1]$ par exemple, alors si $r_1 < 0$ il n'y a pas de racine sur cet intervalle. Mais si $r_1 \leq 0$ on va chercher une valeur $x < r_1$ telle que $f(x) < 0$, elle existe car en tant que polynôme non constant il

n'est pas borné. On va donc poser $u_m = r_1 - 2^m$ et trouver le premier m tel que $u_m < 0$. Il reste toujours un problème informatique car il y a un nombre maximum, en Ocaml celui-ci vaut 1.79769313486231571e+308 ce qui correspond à 2^{1023} environ. On est obligé de faire avec ce problème, qui généralement n'a pas beaucoup d'importance. Ayant trouvé u_m, on applique l'algorithme de dichotomie sur $[u_m, r_1]$. Idem avec $[r_k; +\infty[$.

- Exemple : la fonctions $P(x) = 0.1x^3 - 2x + 0.5$:

  On dérive : $P'(x) = 0.3x^2 - 2$, $P''(x) = 0.6x$. On obtient donc que P' est monotone décroissante sur $]-\infty; 0]$ et monotone croissante sur $[0; +\infty[$. P'(0) = -2 donc P' a deux racines. On cherche un $u_m = 0 - 2^m$ tel que $P'(u_m) > 0$ et un $v_m = 0 + 2^m$ tel que $P'(v_m) > 0$. $u_2 = -4$ et $v_2 = 4$ conviennent. On applique l'algorithme de dichotomie sur [-4,0] et sur [0,4]. On devrait obtenir $r_1 = -\sqrt{\frac{20}{3}}$ et $r_2 = \sqrt{\frac{20}{3}}$. D'où P est monotone croissant sur $]-\infty, r_1]$, monotone décroissant sur $[r_1, r_2]$ et monotone croissant sur $[r_2, +\infty[$. De la même façon on applique l'algorithme de dichotomie sur ces trois intervalle que l'on aurait réduit à des segments comme ci-dessus. Mon algorithme trouve : [-4.59225327282205598; 0.25078866710349923; 4.34146460571861237]. Voir le graphe de P :



2.3 Concrétisation

Pour ce qui est de l'implémentation, nous devons commencer par plusieurs fonctions auxiliaires.

- `dicho`

  Tout d'abord la fonction dicho qui prend pour argument a, b les bornes de l'intervalle d'étude (sur lequel le polynôme sera strictement croissant), 'poly' le polynôme P, 'signe' représentant la monotonie du polynôme sur $[a; b]$ et 'precision', la précision souhaitée dans l'approximation des racines. En suite on suis exactement le principe énoncé ci-dessus, la particularité est que pour ne pas traiter plusieurs cas, j'ai remarqué que $P(c) \times signe > 0$ (rappel c est le milieu du segment $[a; b]$) lorsque P(c) est de signe opposé à P et inversement $P(c) \times signe \leq 0$ lorsque P(c) est de signe opposé à P(b). En effet :

  - Si signe > 0, P est croissant sur $[a; b]$ donc on applique l'algorithme parce que $P(a) \leq 0$ et $P(b) \geq 0$. $P(c) \times signe > 0 \Rightarrow P(c) > 0$, et $P(c) \times signe \leq 0 \Rightarrow P(c) \leq 0$ on a le résultat.

  -Si signe ≤ 0, P est décroissant sur $[a; b]$ donc on applique l'algorithme parce que $P(a) \geq 0$ et $P(b) \leq 0$. $P(c) \times signe > 0 \Rightarrow P(c) < 0$, et $P(c) \times signe \leq 0 \Rightarrow P(c) \leq 0$ on a encore le résultat.

  Attention enfin au choix de l précision, plus le polynôme est de grand degré, plus on doit baisser en qualité pour que le programme termine, je pense que c'est encore une fois dû aux problèmes avec les flottants sur Ocaml. J'arrive tout de même à choisir une précision de 0.00000000001 pour un polynôme de degré 7.

- `trouve_v_m, trouve_u_m`

  Ces deux fonctions servent à trouver les racines extérieures (ceci est expliqué plus haut avec les mêmes notations). Les arguments sont 'poly' le polynôme, t la valeur où commencent les recherches, 'signe' indique la monotonie après t pour u_m (qui cherche un croisement avec l'axe dans la direction des $x > 0$) et avant t pour v_m (qui cherche un croisement avec l'axe dans la direction des $x < 0$). 'res' contient la puissance de 2 en court d'analyse par le programme initialisé à 1, le résultat qui sera renvoyé est t-res pour v_m et t+res pour u_m.

- `round`

  Cette fonction prend un réel x en argument, elle renvoie la partie entière supérieur ou inférieur selon laquelle est la plus proche de x.

- `arrondi`

  Cette fonction prend 'poly' le polynôme en argument, et la liste de racine que nous lui auront trouvé. Elle a pour but de tenter de repérer si on ne tombe pas sur des racines entières ou presque, elle teste si l'arrondi au

centième de la racine est une racine réelle (sans aucune approximation) de 'poly'. Le but étant de rendre le rendu plus esthétique lorsque le polynôme admet des racines n'ayant pas plus de deux décimales.

- `est_dans, suppr_doublons`

  Deux fonctions très classique, utile pour les polynôme de degrés deux à discriminant nul par exemple, plus généralement pour les racines doubles, quadruple... Notre algorithme trouverai en fait deux racines, on doit donc supprimer les doublons. Je ne pense pas que l'implémentation d'un programme plus pointu soit nécessaire ici, j'aurai pu par exemple trier la liste avec un tri fusion tout en supprimant directement les doublons.

On arrive enfin à l'aboutissement, le programme :

`racines`

Le programme prend en argument 'poly' le polynôme est 'precision' la précision attendue sur l'approximation des racines. On commence par traiter les cas triviaux : polynôme constant, on considère qu'il n'y a pas de racines et droite, dont on renvoie l'unique racine.

On remarque que j'aurai pu traiter de même le cas des polynôme de degré 2, mais je me suis dis que je préférais rester fidèle à l'algorithme récursif que j'ai trouvé.

On applique donc cet algorithme. Cas particulier :

- P' n'a pas de racines

  Dans ce cas P est monotone, étant un polynôme il a forcément une racine vu les limites en l'infini. D'où on utilise u_m et v_m pour encadrer cette racine et la déterminer pas dichotomie.

- Dernière racine

  C'est le cas où la liste des racines de la dérivée a été vidée dans la fonction aux, elle vaut donc []. On avait gardé en mémoire son dernier élément, donc nous faut donc maintenant un autre point (s'il existe) pour encadrer la dernière racine. On utilise donc u_m

- Première racine

  On doit s'occuper de celle-ci avant de rentrer dans la fonction aux, on s'occupe d'elle avec les même procédés en utilisant v_m.

2.4 Exemples

```
racines [0.1;0.;-.2.;0.5] 0.000000000000001;;

racines [1.;3.7;1./.3.;-.5.;1.] 0.00000000001;;
```

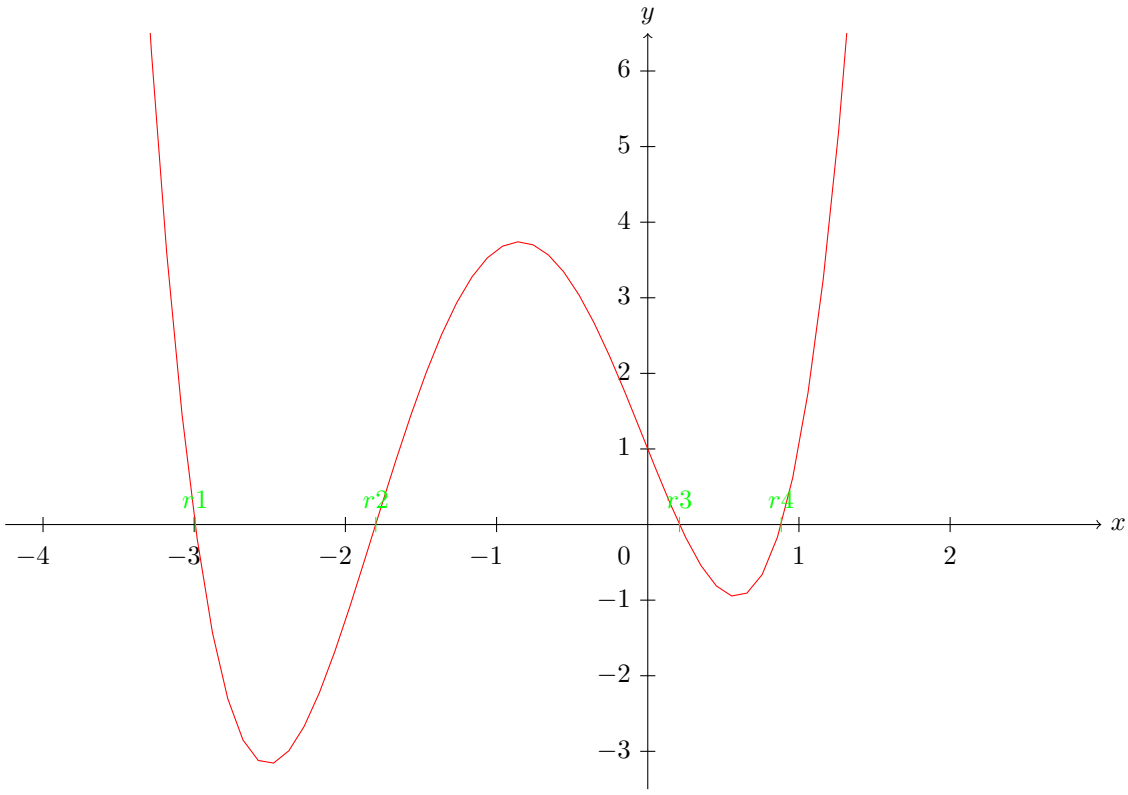
Mon programme a fait ces deux calculs en 0.2 secondes. Il trouve trois racines réelles au premier polynôme de degré 3 (celui de l'exemple ci-dessus), et 4 à l'autre de degré 4 :

```
#       - : float list =
[-4.59225327282207285; 0.250788667103465202; 4.34146460571860793]

#     - : float list =
[-2.99331540582243205; -1.79987569869477; 0.210210142086185825;
 0.882980962431429095]
```

On vérifie bien sur le schéma que les racine de $P = X^4+3.7X^3+\frac{1}{3}X^2-5X+1$ sont :

$$\begin{cases} r1 &= -2.99331540582243205 \\ r2 &= -1.79987569869477 \\ r3 &= 0.210210142086185825 \\ r4 &= 0.882980962431429095 \end{cases}$$



3 Matrices

Tout d'abord nous verrons comment, à l'aide des programmes vu dans la partie polynôme, implémenter un programme qui à une matrice, renvoie son polynôme caractéristique ou ses valeurs propres. Ensuite nous verrons comment obtenir une base propre (on en déduit la matrice de passage de la diagonalisation). Enfin nous traiterons deux exemples détaillés sur des matrices de dimension 3×3: dans le premier nous calculerons à la main (ou nous vérifierons que les résultats renvoyés par le programme sont correctes) un polynôme caractéristique, des valeurs propres et un inverse. Dans le deuxième nous nous intéresserons à la diagonalisation (valeurs propres, vecteurs propres associés) d'une autre matrice 3×3.

3.1 Polynôme caractéristique, valeurs propres

Tout d'abord nous devons implémenter un calcul de déterminant sur Ocaml. Pour ceci on utilise le développement selon la première ligne. On commence par un programme qui a une matrice de taille n et un entier $j \in [0, n-1]$ renvoie la sous matrice obtenue en supprimant de M la première ligne et la $(j+1)^e$ colonne. On peut donc ensuite facilement avec une fonction récursive calculer le développement selon la première ligne d'une matrice :

```
sous_det mat j
det mat
```

Pour la suite nous devons implémenter de nombreuses fonctions de calcul matriciel : Somme, produit de deux matrices, multiplication d'une matrice par un scalaire, une fonction qui à $n \in N^*$ associe la matrice identité de taille n, la matrice zéro de taille n.

Or nous avons besoin d'utiliser le calcul matriciel à la fois sur des matrices de polynômes, et sur des matrices de scalaires. Mis à part la multiplication de deux matrices (qui ne sera utilisé que pour des matrices scalaires), j'ai choisi d'implémenter chacune des fonctions présentée ci-dessus deux fois (une par type), pour s'adapter aux différents besoins. Enfin j'ai créé une fonction faisant la traduction entre ces deux types, à une matrice scalaire elle associe une matrice de polynôme (remplacer a$\in R$ par [a]).

```
somme_mat mat1 mat2
somme_mat_scal mat1 mat2
mult_mat m1 m2
mult_scal_mat k mat
mult_scal_mat_scal k mat
id nid_scal n
zero n
zero_scal n
traduit mat
```

Armés de toutes ces fonctions il est simple de trouver le polynôme caractéristique d'une matrice, et donc ses valeurs propres en appliquant simplement les définitions :

```
let polynome_caract mat0 =
  let mat = traduit mat0 in
  let n = Array.length mat in
    det (somme_mat (mult_scal_mat [1.;0.] (id n)) (mult_scal_mat [-.1.] mat));;


(*renvoie toutes les valeurs propres réelles de mat*)
let valeur_propre mat =
  racines (polynome_caract mat) 0.00000001;;
```

3.2 Vecteurs propres, base propre

Maintenant que nous avons calculé les valeurs propres d'une matrice, il serait intéressant de trouver un programme pour les vecteurs propres associés à ces valeurs propres. Pour ceci j'utilise une méthode standard. On se donne une matrice carré M de dimension n, et $(\lambda_i)_{i\in[\![0,k]\!]}$, $k \in N$ ses valeurs propres distinctes. Tout d'abord les vecteurs propres sont les solutions d'un système, ce système étant celui définissant les sous espaces propres (en effet un sous espace propres $E(\lambda)$ est l'ensemble formé des vecteurs propres associés à λ et du vecteur nul) :

$$E(\lambda) = Ker(M - \lambda I_n) = \{X \in \mathcal{M}_{n,1} | MX = \lambda X\}$$

En notant $S = M - \lambda I_n$, nous devons résoudre le système $SX = 0$ où X est une colonne inconnue, $X = \begin{pmatrix} x_0 \\ x_1 \\ ... \\ x_{n-1} \end{pmatrix}$.

Pour résoudre ce système j'utilise la méthode du pivot de Gauss, mais par un travaille personnel j'ai trouvé comment l'adapter au problème que nous résolvons. Tout d'abord cette méthode nous permet dans tous les cas de trigonaliser le système, le principe par récurrence :

Initialisation : Tout d'abord nous devons trouver un coefficient sur la première colonne du système (première colonne de S) qui soit non nul, il est appelé 'pivot'. Si il n'en existe pas, la première étape est finie. Si il en existe, on échange la ligne sur laquelle se trouve le pivot avec la première ligne, ensuite on annule tous les coefficients de la première colonne de S (hormis le pivot), en faisant des combinaisons linéaires de chacune des lignes avec la première.

Hérédité : Après avoir traité le colonne numéro 0, 1, ..., $k \in N$, pour traiter la colonne k+1 : même principe que lors de l'initialisation, il faut trouver un 'pivot' (coefficient non nul) sur la $(k+1)^e$ colonne de S parmi les lignes numéro k+1 à n-1, ensuite on échange la ligne sur laquelle il se trouve et la $(k+1)^e$. Enfin on annule tous les coefficients de la $(k+1)^e$ colonne, se trouvant sur les lignes numéro k+2, k+3, ..., n-1 par combinaison linéaire de ces lignes avec la

$(k+1)^e$. Ce qui est possible car : Soit nous n'avons pas trouvé de pivot et donc tous ces coefficients étaient déjà nuls. Soit avec le pivot p et l'opération $L_i \leftarrow p \times L_i - S_{i,j} \times L_{k+1}, \forall i \in [\![k+2, n-1]\!]$, nous annulons tous les coefficients.

Après avoir appliqué cet algorithme, on obtient un matrice triangulaire supérieure. Cela est du au fait que chaque étape ne modifie pas le travaille réalisé par les étapes précédentes, l'ordre a une très grande importance.

Ensuite, comme dans la méthode classique du pivot de Gauss, on peut compter le nombre de coefficients non nuls sur la diagonale de S, ce nombre noté rg est le rang de S (car S est triangulaire à cette étape), par théorème du rang, on obtient la dimension de $E(\lambda)$: n - rg. Ce nombre est donc le nombre de variables libres parmi les coefficients de X (défini plus haut). Appliquons maintenant la seconde partie de l'algorithme de Gauss, mais seulement au colonne j dont le coefficient $S_{j,j}$ est non nul (méthode inapplicable sinon). Je ne vais pas la détailler, le principe est le même que pour la première étape, hormis que nous commençons par la dernière colonne est que le but est d'annuler les coefficients "haut dessus" du coefficients sur la diagonale. La dernière étape est de rendre unitaire toutes les lignes du système : Pour la ligne i, si $S_{i,i}$ est non nul, on opère $L_i \leftarrow \frac{1}{S_{i,i}} L_i$ On obtient un système S de la forme :

$$S = \begin{pmatrix} 1 & 0 & c_{0,2} & 0 & ... & c_{0,n-1} \\ 0 & 1 & c_{1,2} & 0 & ... & c_{1,n-1} \\ 0 & 0 & 0 & 0 & ... & c_{2,n-1} \\ 0 & 0 & 0 & 1 & ... & c_{3,n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & ... & 0 \end{pmatrix}$$

Avec S triangulaire supérieure, des coefficients $c_{0,k}$, $c_{1,k}$ ... apparaissent seulement lorsque $S_{k,k} = 0$. Or nous avons vu que le nombre de tels k est la dimension de $E(\lambda)$, le nombre de vecteurs propre libres que nous cherchons. Avec S de cette forme, nous pouvons directement lire les vecteurs propres : Pour toutes colonnes C de S contenant un zéro sur le terme diagonal. Disons que C est la colonne numéro k de S, ainsi $C = \begin{pmatrix} c_{0,k} \\ \vdots \\ c_{k-1,k} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.

Alors : $V = \begin{pmatrix} 0 \\ 0 \\ ... \\ 1 \\ ... \\ 0 \end{pmatrix} - C = \begin{pmatrix} -c_{k,0} \\ -c_{k,1} \\ \vdots \\ -c_{k,k-1} \\ 1 \\ 0 \\ \vdots \end{pmatrix}$ est un vecteur propre de M.

En effet montrons cela avec l'exemple de S représentée ci-dessus (relativement général) :

$$SX = 0 \quad \Leftrightarrow \quad \left\{ \begin{array}{rcl} x_0 + 0 \times x_1 + c_{0,2} \times x_2 + 0 \times x_3 + ... + c_{0,n-1} \times x_{n-1} & = & 0 \\ 0 \times x_0 + x_1 + c_{1,2} \times x_2 + 0 \times x_3 + ... + c_{1,n-1} \times x_{n-1} & = & 0 \\ 0 \times x_0 + 0 \times x_1 + 0 \times x_2 + 0 \times x_3 + ... + c_{2,n-1} \times x_{n-1} & = & 0 \\ 0 \times x_0 + 0 \times x_1 + 0 \times x_2 + 1 \times x_3 + ... + c_{3,n-1} \times x_{n-1} & = & 0 \\ \vdots & & \\ 0 \times x_0 + 0 \times x_1 + 0 \times x_2 + ... + 0 \times x_{n-1} & = & 0 \end{array} \right.$$

$$\Leftrightarrow \quad \left\{ \begin{array}{rcl} x_0 & = & -c_{0,2} \times x_2 - ... - c_{0,n-1} \times x_{n-1} \\ x_1 & = & -c_{1,2} \times x_2 - ... - c_{1,n-1} \times x_{n-1} \\ x_2 & = & x_2 - ... - c_{2,n-1} \times x_{n-1} \\ x_3 & = & ... - c_{3,n-1} \times x_{n-1} \\ \vdots & & \\ x_{n-1} & = & x_{n-1} \end{array} \right.$$

$$\Leftrightarrow \quad X \in Vect \left(\left(\begin{array}{c} -c_{0,2} \\ -c_{1,2} \\ 1 \\ 0 \\ \vdots \\ 0 \end{array} \right), ..., \left(\begin{array}{c} -c_{0,n-1} \\ -c_{1,n-1} \\ -c_{2,n-1} \\ \vdots \\ -c_{n-2,n-1} \\ 1 \end{array} \right) \right)$$

Or les vecteurs trouvés sont en nombre de $\dim(E(\lambda))$ (vu plus haut) et sont clairement libres (on pourrait dire qu'ils sont comme 'échelonnés'), d'où ils forment une base de $E(\lambda)$, nous avons totalement résolu le problème.

J'ai volontairement présenté les étapes dans un formalisme permettant de comprendre l'implémentation en Ocaml (qui suit pas à pas la méthode présentée).

- Tout d'abord nous implémentons les fonctions de calculs sur les systèmes : échange de ligne, combinaison linéaire (de type unit), mais aussi des fonctions prenant en argument un ou deux tableaux représentant des vecteurs lignes ou colonnes, et réalisant la somme de ces lignes, et la multiplication d'une ligne par un scalaire. NB : Ces fonctions ne sont pas les mêmes que celle réalisant la somme de deux polynôme ou le produit d'un polynôme par un scalaire car les polynôme étaient pour nous de type float list, ici nous travaillons avec le type float array.

```
somme_ligne l1 l2
mult_scal_ligne k l
echange_ligne i j mat
combinaison_lin a i b j mat
```

Ensuite nous implémentons un programme qui renvoie le système décrivant $E(\lambda)$ pour un λ et une matrice 'mat' donnés :

```
sys_espace_propre mat lambda
```

Le système est sous la forme d'une matrice car il n'y a pas de second membre à notre système. (Le programme met tout les termes du même côté du '=').

- Le programme

  ```
  trouve_pivot mat j
  ```

  Renvoie l'indice de la ligne du pivot qu'il trouve sur la colonne j de mat, s'il existe, sinon renvoie -1.

  Ensuite on implémente l'algorithme du pivot de Gauss, la première étape : simplifie1 qui à une matrice 'mat' est un indice 'i', représentant les coordonnées d'un pivot (mat.(i).(i)), supprime les coefficients 'en dessous' du pivot. Puis simplifie2 qui opère la seconde étape, il prend pour arguments une matrice 'mat' et un indice 'i', représentant les coordonnées d'un pivot (mat.(i).(i)), elle supprime les coefficients 'au dessus' du pivot. Enfin simplifie3 rend unitaire toute les lignes non nulles du système.

  ```
  simplifie1_sys mat i
  simplifie2_sys mat i
  simplifie3_sys mat
  ```

- Ensuite, j'ai rencontré un problème dû aux approximation faite sur les valeurs propres (voir partie consacrée). Le problème est que le programme ne détecte pas assez de zéros sur la diagonale après les étapes présentées juste avant, et ainsi il ne trouve pas assez de vecteurs propres. Cela est dû au fait qu'à la place d'avoir un zéro, le programme trouve 0.00000001 ou -0.00000001 (erreur de cet ordre, 10^{-8}), ce sont les répercutions des approximations faites avant. Pour résoudre ce problème, je vérifie, pour tout coefficients diagonaux, si sont approximation à 10^{-6} près vaut zéro, si c'est le cas je remplace sa valeur par zéro :

  ```
  arrondi_sys mat
  ```

- En assemblant tout ces programmes, nous pouvons complètement mettre le système sous la forme souhaitée (voir la matrice S ci-dessus) :

  ```
  trigonalise_sys mat
  ```

  Nous avons vu que maintenant, à une petite modification près, nous pouvons lire les vecteurs propres directement à partir des colonne j vérifiant mat.(j).(j) = 0. On en déduit :

```
trouve_vecteur_propre j mat
```

- Je me suis également intéressé à la présentation des résultats, d'où les programmes :

  ```
  print_colonne colonnes
  affiche l1 l2
  ```

  Ils réalisent l'affichage des résultats, saut de ligne, ...

- Finalement nous avons tous les outils pour réaliser notre programme trouvant toutes les valeurs propres et des vecteurs propres associé d'une matrice :

  ```
  val_propre_vect_propre mat
  ```

  Le principe est que nous avons déjà un programme trouvant les valeurs propres d'une matrice. Et un programme qui résout le système $E(\lambda)$, pour toute valeur propre λ. On a donc plus qu'à appliquer ce second programme à toute les valeurs propres trouvée dans le premier.

3.3 Inversion de matrice

En utilisant les programmes que nous venons de créer, nous pouvons appliquer une méthode particulière d'inversion de matrice dont je vais donner un démonstration rapide :

Soit $M \in \mathcal{M}_n(R)$ pour $n \in N^*$. Soit $\chi_M = det(XI_n - M)$ le polynôme caractéristique de M. On sait que le terme de degré 0 de χ_M vaut $(-1)^n det(M)$. Donc avec la donnée de χ_M, nous pouvons rapidement savoir si M est inversible, en vérifiant que son terme de degré 0 est non nul. Supposons donc que M est inversible, on a donc :

$$\chi_M = \sum_{k=0}^{n} a_k X^k, \forall i \in [\![0, n]\!], a_i \in R, a_0 \neq 0$$

Par le théorème de Cayley-Hamilton, χ_M est un polynôme annulateur de M, on a donc :

$$\begin{array}{rcl} \chi_M(M) = \sum_{k=0}^{n} a_k M^k & = & 0_{\mathcal{M}_n(R)} \\ \Rightarrow \sum_{k=1}^{n} a_k M^k & = & -a_0 I_n \\ \Rightarrow M(\frac{-1}{a_0} \sum_{k=0}^{n-1} a_{k+1} M^k) & = & (\frac{-1}{a_0} \sum_{k=0}^{n-1} a_{k+1} M^k) M = I_n \\ \Rightarrow \frac{-1}{a_0} \sum_{k=0}^{n-1} a_{k+1} M^k & = & M^{-1} \end{array}$$

D'où, la connaissance de χ_M (de ses coefficients), nous informe de l'inversibilité de M, mais dans le cas où M est inversible, elle fournit aussi une formule explicite de l'inverse de M, M^{-1}.

D'où il est simple de programmer une fonction d'inversion de matrice en appliquant la formule ci-dessus. On implemente deux fonctions auxiliaires, l'une qui transforme $P = [a_0; a_1; ...; a_m]$ un polynôme en un couple (a_m, P_f) où $P_f = [a_0; a_1; ...; a_{m-1}]$. Celle-ci permet avec a_m de connaître si M est inversible, et si c'est le cas avec P_f appliqué en M, le tout multiplié par $\frac{-1}{a_0}$ d'obtenir l'inverse. La seconde fonction auxiliaire qui prend en argument un polynôme P et une matrice M, et qui renvoie la spécialisation de P en M. J'utilise pour celle-ci à nouveau la méthode de Horner mais appliquée aux matrices, nous devons utiliser les fonctions implémentées précédemment pour les multiplications et somme de matrices.

```
    dernier_suppr l
    eval_mat poly mat
```

Donc il ne nous reste plus qu'à appliquer la formule démontrée ci-dessus :

```
let inverse_matrice mat =
  let poly0 = polynome_caract mat in
  let d,poly = dernier_suppr poly0 in
    if d = 0. then failwith "la matrice n'est pas inversible"
    else mult_scal_mat_scal (-.1./.d) (eval_mat poly mat);;
```

3.4 Exemples

1. Posons M = $\begin{pmatrix} 6 & 5 & 0 \\ 3 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix}$

   - Valeurs propres, polynôme caractéristique
     Nous rentrons la matrice dans le programme :

     ```
     let mat =
       [|[|6.;5.;0.|];
       [|3.;1.;1.|];
       [|2.;0.;1.|]|];;

     polynome_caract mat;;
     valeur_propre mat;;
     ```

     Il renvoie :

     ```
     # - : float list = [1.; -8.; -2.; -1.]
     # - : float list = [8.25688982444504838]
     ```

     Vérifions ce résultat :

$$\chi_M = \begin{vmatrix} X-6 & -5 & 0 \\ -3 & X-1 & -1 \\ -2 & 0 & X-1 \end{vmatrix} = (X-6)\begin{vmatrix} X-1 & -1 \\ 0 & X-1 \end{vmatrix} + 5\begin{vmatrix} -3 & -1 \\ -2 & X-1 \end{vmatrix}$$
$$= (X-6)(X-1)^2 + 5(3-3X-2)$$
$$= X^3 - 8X^2 - 2X - 1$$

Donc le polynôme caractéristique renvoyé est correcte, tester si la valeur propre en est une :

$$\chi_M(8.25688982444504838) = -0.00000000255$$

On pourra donc conclure avec une analyse des variations de χ_M :

$$\chi'_M = 3X^2 - 16X - 2 = 3(X^2 - \tfrac{16}{3}X - \tfrac{2}{3}) = 3[(X - \tfrac{16}{6})^2 - \tfrac{280}{36}]$$
$$= 3(X - \tfrac{8-\sqrt{70}}{3})(X - \tfrac{8+\sqrt{70}}{3})$$

On peut tracer le tableau de variation :

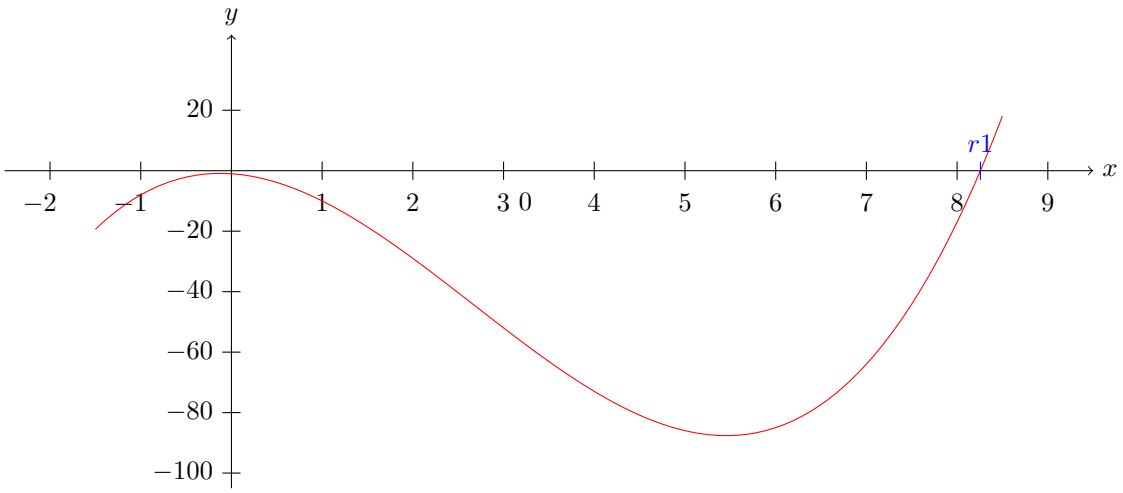
x	$-\infty$		$\frac{8-\sqrt{70}}{3}$		$\frac{8+\sqrt{70}}{3}$		$+\infty$
$\chi'_M(x)$		$+$	0	$-$	0	$+$	
χ_M	$-\infty$	↗	-0.877	↘	-87.64	↗	$+\infty$

On peut conclure de cette étude que χ_M ne s'annule effectivement qu'une seule fois sur R, et c'est pour un $x \in]\frac{8-\sqrt{70}}{3}, +\infty[$ tel que 8.25688982444504838. Soit $\epsilon = 0.0000000001$,

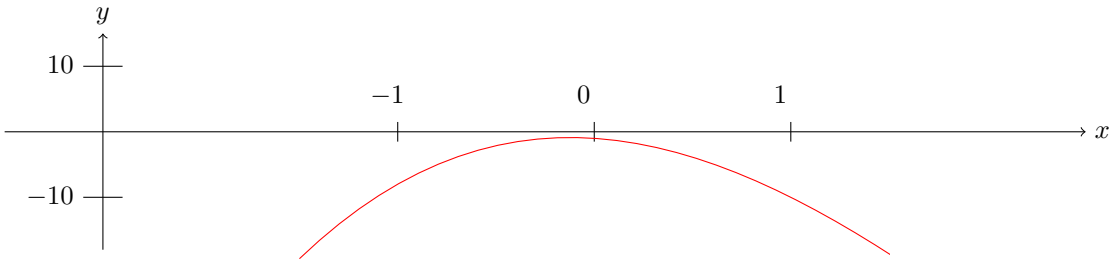
$$\chi_M(8.25688982444504838 - \epsilon) = -0.00000000959244672582571 < 0$$

$$\chi_M(8.25688982444504838 + \epsilon) = 0.00000000449124559764868536 > 0$$

Vu la stricte croissance de χ_M sur $]\frac{8-\sqrt{70}}{3}, +\infty[$, on a bien que 8.25688982444504838 est une très bonne approximation de l'unique racine de χ_M.



Zoom :



- Inversion
  Nous allons demander au programme d'inverser la matrice M :

  ```
  inverse_matrice mat;;
  ```

  Il renvoit :

  ```
  - : float array array =
  [|[|1.; -5.; 5.|]; [|-1.; 6.; -6.|]; [|-2.; 10.; -9.|]|]
  ```

  Vérifions ce résultat en posant M' = $\begin{pmatrix} 1 & -5 & 5 \\ -1 & 6 & -6 \\ -2 & 10 & -9 \end{pmatrix}$:

$$M'M = \begin{pmatrix} 1 & -5 & 5 \\ -1 & 6 & -6 \\ -2 & 10 & -9 \end{pmatrix} \begin{pmatrix} 6 & 5 & 0 \\ 3 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I_3$$

D'où M est inversible à gauche, or $\mathcal{M}_3(R)$ est de dimension finie, donc M est inversible, d'inverse M' car celui-ci est unique.

2. Posons M = $\begin{pmatrix} 9 & 6 & 1 \\ 3 & 1 & 1 \\ 3 & 4 & -1 \end{pmatrix}$

Trouvons ses valeurs propres :

$$\begin{aligned} \chi_M &= \begin{vmatrix} X-9 & -6 & -1 \\ -3 & X-1 & -1 \\ -3 & -4 & X+1 \end{vmatrix} = \begin{vmatrix} X-9 & -6 & -1 \\ -3 & X-1 & -1 \\ -3 & -4 & X+1 \end{vmatrix} \\ &= (X-9)(X^2-1-4)+6(-3X-3-3)-(12+3X-3) \\ &= X^3-9X^2-26X \\ &= X(X^2-9X-26) \\ &= X[(X-\tfrac{9}{2})^2-\tfrac{185}{4}] \\ &= X(X-\tfrac{9}{2}-\tfrac{\sqrt{185}}{2})(X-\tfrac{9}{2}+\tfrac{\sqrt{185}}{2}) \end{aligned}$$

Je trouve $Sp(M) = \{\frac{9-\sqrt{185}}{2}, 0, \frac{9+\sqrt{185}}{2}\}$

Déterminons maintenant une base propre pour M : Soit $\lambda \in Sp(M)$, $E(\lambda)$:

$$\begin{cases} 9x+6y+z = \lambda x \\ 3x+y+z = \lambda y \\ 3x+4y-z = \lambda z \end{cases} \Leftrightarrow \begin{cases} (9-\lambda)x+6y+z = 0 \\ 3x+(1-\lambda)y+z = 0 \\ 3x+4y-(1+\lambda)z = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} (6-\lambda)x+(5+\lambda)y = 0 \\ 3x+(1-\lambda)y+z = 0 \\ (3+\lambda)y-(2+\lambda)z = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} x = \frac{5+\lambda}{\lambda-6}y \\ 3\frac{5+\lambda}{\lambda-6}y+(1-\lambda)y+\frac{3+\lambda}{2+\lambda}y = 0 \\ z = \frac{3+\lambda}{2+\lambda}y \end{cases}$$

$$\Leftrightarrow \begin{cases} x = \frac{5+\lambda}{\lambda-6}y \\ y[-\lambda^3+9\lambda^2+26\lambda] = 0 \\ z = \frac{3+\lambda}{2+\lambda}y \end{cases}$$

$$\Leftrightarrow \begin{cases} x = \frac{5+\lambda}{\lambda-6}y \\ z = \frac{3+\lambda}{2+\lambda}y \end{cases}$$

D'où tous les sous espaces propres sont de dimension 1 et $\begin{pmatrix} \frac{5+\lambda}{\lambda-6} \\ 1 \\ \frac{3+\lambda}{2+\lambda} \end{pmatrix}$ est une base de $E(\lambda)$.

Mon programme renvoie :

```
#     Valeur propre : -2.30073525431
Vecteurs propres associés : (0.139852949136 , -0.430073525454 , 1.)

Valeur propre : 0.
Vecteurs propres associés : (-0.555555555556 , 0.666666666667 , 1.)

Valeur propre : 11.3007352543
Vecteurs propres associés : (2.86014705125 , 0.930073525552 , 1.)
```

Or, $\frac{9-\sqrt{185}}{2} = -2.30073525436772197$

$\frac{5+\frac{9-\sqrt{185}}{2}}{\frac{9-\sqrt{185}}{2}-6} = -0.325183813591930326$ et $\frac{3+\frac{9-\sqrt{185}}{2}}{2+\frac{9-\sqrt{185}}{2}} = -2.32518381359192716$

$$\begin{pmatrix} \frac{5+\lambda}{\lambda-6} \\ 1 \\ \frac{3+\lambda}{2+\lambda} \end{pmatrix} \times (-0.430073525454) = \begin{pmatrix} 0.139852949132057841 \\ -0.430073525454 \\ 1.00000000004005662 \end{pmatrix}$$

$\frac{5+0}{0-6} = 0.83333333333333333$ et $\frac{3+0}{2+0} = 1.5$

$$\begin{pmatrix} \frac{5+\lambda}{\lambda-6} \\ 1 \\ \frac{3+\lambda}{2+\lambda} \end{pmatrix} \times 0.666666666667 = \begin{pmatrix} -0.555555555555833358 \\ 0.666666666667 \\ 1.0000000000005 \end{pmatrix}$$

$\frac{9+\sqrt{185}}{2} = 11.3007352543677229$

$\frac{5+\frac{9+\sqrt{185}}{2}}{\frac{9+\sqrt{185}}{2}-6} = 3.07518381359193$ et $\frac{3+\frac{9+\sqrt{185}}{2}}{2+\frac{9+\sqrt{185}}{2}} = 1.07518381359193049$

$$\begin{pmatrix} \frac{5+\lambda}{\lambda-6} \\ 1 \\ \frac{3+\lambda}{2+\lambda} \end{pmatrix} \times 0.930073525552 = \begin{pmatrix} 2.86014705122789037 \\ 0.930073525552 \\ 1.00000000012389112 \end{pmatrix}$$

D'où le programme renvoie un résultat exact à 10^{-9} près.