Le pivot de Gauss
La factorisation A = L . U d'une matrice inversible quelconque

J'explique comment on passe de A à U sur un exemple.
La question sera : mais où est L ?

```
> with (LinearAlgebra):
```

Je prends un exemple que j'ai préparé. Pour quand on fera le lien avec Python

```
> A := <<100 | 1 | 1>, <1 | 10 | 2>, <2 | 3 | -1>>;
```

$$A := \begin{bmatrix} 100 & 1 & 1 \\ 1 & 10 & 2 \\ 2 & 3 & -1 \end{bmatrix} \quad (1)$$

Le premier pivot = a[1,1] = 100.
On veut faire apparaître des 0 en a[2,1] et en a[3,1].
Il suffit de faire : deuxième ligne de A = deuxième ligne de A – (1/100) * première ligne

Comme je l'ai dit/écrit sur la photo, cette opération peut se coder par une multiplication de matrice. Le codage est important pour comprendre d'où vient L mais vous verrez que, pour l'algorithme, on n'aura pas besoin de faire si compliqué.

Le rapport 1/100 = a[2,1]/a[1,1] porte un nom (on s'en resservira).
C'est un 'multiplicateur'. Je le note m[2,1]

```
> m[2,1] := A[2,1]/A[1,1];
```

$$m_{2,1} := \frac{1}{100} \quad (2)$$

La matrice qui code l'opération. Il faut que je la retrouve :-). Je pars de la matrice identité

```
> G[1] := <<1,-m[2,1],0> | <0,1,0> | <0,0,1>>;
```

$$G_1 := \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{100} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

On vérifie, bien sûr. Ça marche : j'ai un zéro en a[2,1].

```
> G[1] . A;
```

$$\begin{bmatrix} 100 & 1 & 1 \\ 0 & \frac{999}{100} & \frac{199}{100} \\ 2 & 3 & -1 \end{bmatrix} \quad (4)$$

Je place maintenant un zéro en a[3,1]. Le multiplicateur s'obtient en divisant a[3,1]

par le pivot.

```
> m[3,1] := A[3,1] / A[1,1];
```

$$m_{3,1} := \frac{1}{50} \tag{5}$$

La matrice qui code l'opération : soustraire m[3,1] fois la première ligne à la troisième.

```
> G[2] := <<1,0,-m[3,1]> | <0,1,0> | <0,0,1>>;
```

$$G_2 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{50} & 0 & 1 \end{bmatrix} \tag{6}$$

On vérifie

```
> B := G[2] . G[1] . A;
```

$$B := \begin{bmatrix} 100 & 1 & 1 \\ 0 & \frac{999}{100} & \frac{199}{100} \\ 0 & \frac{149}{50} & -\frac{51}{50} \end{bmatrix} \tag{7}$$

On voit que G[2] . G[1] code l'enchaînement des deux opérations.
Il nous reste un zéro à mettre en a[3,2].

Le deuxième pivot doit être pris sur la matrice B ci-dessus, pas sur A.

On choisit un deuxième pivot. On prend b[2,2] = 999/100.

Le multiplicateur est m[3,2] = b[3,2] / b[2,2] (on divise par le pivot)

```
> m[3,2] := B[3,2] / B[2,2];
```

$$m_{3,2} := \frac{298}{999} \tag{8}$$

```
> G[3] := <<1,0,0> | <0,1,-m[3,2]> | <0,0,1>>;
```

$$G_3 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{298}{999} & 1 \end{bmatrix} \tag{9}$$

On place le dernier zéro. Le résultat, c'est la matrice U ! Yes.

```
> U := G[3] . B;
```

(10)

$$U := \begin{bmatrix} 100 & 1 & 1 \\ 0 & \frac{999}{100} & \frac{199}{100} \\ 0 & 0 & -\frac{1612}{999} \end{bmatrix} \quad (10)$$

On a donc :

G[3] . G[2] . G[1] . A = U

```
> G[3] . G[2] . G[1] . A;
```

$$\begin{bmatrix} 100 & 1 & 1 \\ 0 & \frac{999}{100} & \frac{199}{100} \\ 0 & 0 & -\frac{1612}{999} \end{bmatrix} \quad (11)$$

La matrice L, c'est (G[3] . G[2] . G[1])^(-1) (l'inverse).
Mais est-ce que L est triangulaire inférieure ?

```
> L := (G[3] . G[2] . G[1])^(-1);
```

$$L := \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{100} & 1 & 0 \\ \frac{1}{50} & \frac{298}{999} & 1 \end{bmatrix} \quad (12)$$

Miracle ! Et en plus, regardez le contenu de L :
- sur la diagonale, des 1
- sous la diagonale, les multiplicateurs

```
> <<1,m[2,1],m[3,1]> | <0,1,m[3,2]> | <0,0,1>>;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{100} & 1 & 0 \\ \frac{1}{50} & \frac{298}{999} & 1 \end{bmatrix} \quad (13)$$

Vérifions tout de même que A = L . U

```
> A, L . U;
```

$$\begin{bmatrix} 100 & 1 & 1 \\ 1 & 10 & 2 \\ 2 & 3 & -1 \end{bmatrix}, \begin{bmatrix} 100 & 1 & 1 \\ 1 & 10 & 2 \\ 2 & 3 & -1 \end{bmatrix} \quad (14)$$

C'est l'un des "miracles" du pivot de Gauss.

Les opérations de lignes que vous connaissez probablement déjà permettent d'avoir la matrice L sans aucun calcul supplémentaire. Il suffit de mémoriser les multiplicateurs qu'on doit de toutes façons calculer pour obtenir U.

De plus, comme on sait que la diagonale ne comporte que des 1, il n'y a vraiment que les multiplicateurs à mémoriser.

Et comme, en dessous de la diagonale de U, il y a des zéros, on peut utiliser ces emplacements pour stocker les multiplicateurs.

Concrètement, les implantations de la factorisation L . U prennent en entrée la matrice A et la modifient. Dans la matrice résultat, on va trouver U et, en-dessous de la diagonale de U, on va trouver les multiplicateurs. Voici ce que ça donnerait sur notre exemple :

```
> Resultat := Matrix(3, 3, [[100, 1, 1], [m[2,1], 999/100,
  199/100], [m[3,1], m[3,2], -1612/999]]);
```

$$Resultat := \begin{bmatrix} 100 & 1 & 1 \\ \frac{1}{100} & \frac{999}{100} & \frac{199}{100} \\ \frac{1}{50} & \frac{298}{999} & -\frac{1612}{999} \end{bmatrix} \quad (15)$$

En Maple, il existe bien sûr une implantation de la factorisation L . U mais un peu plus générale : c'est une factorisation A = P . L . U

La matrice P code des permutations de lignes.
Même si on suppose A inversible, il peut arriver qu'on trouve un zéro à la place du pivot (en a[i,i] à l'étape i). Dans ce cas, on cherche un autre pivot sur la colonne i, sur les lignes i+1 .. n. On est sûr d'en trouver un puisque A est supposée inversible.

Mais en toute généralité, on a besoin de retourner une information sur les permutations de lignes qu'on a pu avoir à faire. Allons-y :

```
> P, L, U := LUDecomposition (A);
```

$$P, L, U := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{100} & 1 & 0 \\ \frac{1}{50} & \frac{298}{999} & 1 \end{bmatrix}, \begin{bmatrix} 100 & 1 & 1 \\ 0 & \frac{999}{100} & \frac{199}{100} \\ 0 & 0 & -\frac{1612}{999} \end{bmatrix} \quad (16)$$

On retrouve L et U. Pour la matrice P, on voit qu'il s'agit de la la matrice identité, ce qui signifie qu'il n'y a pas eu de permutation de lignes.

Alors, quittons l'algèbre pour le calcul numérique !

Si je prends le même exemple mais avec des nombres à virgule flottante, l'algorithme utilisé n'est plus le même et on va voir qu'il effectue des permutations de lignes. Et pourtant, c'est le même exemple. Oups.

```
> P, L, U := LUDecomposition (evalf(A));
```

$$P, L, U := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1.0 & 0. & 0. \\ 0.0100000000000000 & 1.0 & 0. \\ 0.0200000000000000 & 0.298298298298298 & 1.0 \end{bmatrix}, \tag{17}$$

$$\begin{bmatrix} 100. & 1. & 1. \\ 0. & 9.99000000000000 & 1.99000000000000 \\ 0. & 0. & -1.61361361361361 \end{bmatrix}$$

Ici, il ne le fait pas. Je vais recommencer ci-dessous en permutant deux lignes sur la matrice initiale A.

```
> Abar := <A[3,1..3], A[2,1..3], A[1,1..3]>;
```

$$Abar := \begin{bmatrix} 2 & 3 & -1 \\ 1 & 10 & 2 \\ 100 & 1 & 1 \end{bmatrix} \tag{18}$$

Je fais d'abord la factorisation P . L . U avec des nombres exacts. Comme dans un cours de maths.
Pas de permutation.

```
> P, L, U := LUDecomposition (Abar);
```

$$P, L, U := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 50 & -\frac{298}{17} & 1 \end{bmatrix}, \begin{bmatrix} 2 & 3 & -1 \\ 0 & \frac{17}{2} & \frac{5}{2} \\ 0 & 0 & \frac{1612}{17} \end{bmatrix} \tag{19}$$

Je traite le même exemple avec des flottants. Suspense ...

```
> P, L, U := LUDecomposition(evalf(Abar));
```

$$P, L, U := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1.0 & 0. & 0. \\ 0.0100000000000000 & 1.0 & 0. \\ 0.0200000000000000 & 0.298298298298298 & 1.0 \end{bmatrix}, \tag{20}$$

$$\begin{bmatrix} 100. & 1. & 1. \\ 0. & 9.99000000000000 & 1.99000000000000 \\ 0. & 0. & -1.61361361361361 \end{bmatrix}$$

Yes. La matrice P code l'échange des lignes 1 et 3. Vérification :

```
> P . Abar;
```

$$\begin{bmatrix} 100 & 1 & 1 \\ 1 & 10 & 2 \\ 2 & 3 & -1 \end{bmatrix} \quad \textbf{(21)}$$

L'algorithme cherche le pivot le plus grand possible en valeur absolue (le plus éloigné de 0 possible, si vous voulez) pour que le multiplicateur soit le plus petit possible.

Cette stratégie s'appelle le "**partial pivoting**" en Anglais (pivotage partiel en Français). La stratégie du pivotage total existe aussi (chercher le pivot dans toute la matrice). Elle est utile pour certaines applications mais ne permet pas de calculer la factorisation L . U

En pratique, la matrice P n'a pas besoin d'être codée "comme en maths".
Il n'y a que des zéros sauf un '1' sur chaque ligne / colonne.
On peut complètement retrouver l'information en mémorisant l'indice de ligne du pivot à chaque itération (et on n'a même pas besoin de faire réellement des permutations de ligne).

Je vous montre ça en changeant d'écran. Je passe sur un terminal.

Vous pouvez aller voir le fichier dgetrf.f qui contient la subroutine DGETRF (double precision, general matrix, triangular factorization).

Vous pouvez aussi aller voir scipy.linalg.lu_factor qui appelle dgetrf
et scipy.linalg.lu_solve qui réutilise le résultat de la factorisation LU pour résoudre un système linéaire (en enchaînant deux résolutions de systèmes triangulaires). Cette fonction appelle probablement la subroutine dgetrs de LAPACK