

BROUILLON - DES COEFFICIENTS DE BACHET-BÉZOUT POUR LES HUMAINS

CHRISTOPHE BAL

*Document, avec son source L^AT_EX, disponible sur la page
<https://github.com/bc-writing/drafts>.*

Mentions « légales »

Ce document est mis à disposition selon les termes de la licence Creative Commons “Attribution - Pas d’utilisation commerciale - Partage dans les mêmes conditions 4.0 International”.



TABLE DES MATIÈRES

1.	Où allons-nous ?	2
2.	L’algorithme « human friendly » appliqué de façon magique	3
2.1.	Un exemple complet façon « <i>diaporama</i> »	3
2.2.	Phase 1 – Au début était l’algorithme d’Euclide...	3
2.3.	Phase 2 – Remontée facile des étapes	3
2.4.	Et voilà comment conclure !	5
3.	Pourquoi cela marche-t-il ?	6
3.1.	Avec des arguments élémentaires	6
3.2.	Avec des matrices pour y voir plus clair	7
4.	Des coefficients via des algorithmes programmables	8
4.1.	La version « humaine » à la main	8
4.2.	La version « humaine » via les matrices	9
4.3.	Tailles des coefficients lors de la remontée	9
4.4.	Pas terribles...	11
5.	Un algorithme classique bien plus efficace	12
5.1.	On peut faire mieux !	12
5.2.	Tailles des coefficients de la méthode efficace	13
6.	Une infinité de coefficients de Bachet-Bézout	15
7.	Nombre d’étapes de l’algorithme d’Euclide	16
7.1.	Une première estimation	16
8.	AFFAIRE À SUIVRE...	16

1. OÙ ALLONS-NOUS ?

Un résultat classique d'arithmétique dit qu'étant donné $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$, il existe $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$. Les entiers u et v seront appelés « *coefficients de Bachet-Bézout* » et $au + bv = \text{pgcd}(a; b)$ sera nommée « *relation de Bachet-Bézout* ». Notons qu'il n'y a pas unicité car nous avons par exemple :

$$(-3) \times 12 + 1 \times 42 = 4 \times 12 + (-1) \times 42 = 6 = \text{pgcd}(12; 42)$$

Nous allons voir comment trouver de tels entiers u et v tout d'abord de façon humainement rapide puis ensuite via un algorithme programmable efficace.

2. L'ALGORITHME « HUMAN FRIENDLY » APPLIQUÉ DE FAÇON MAGIQUE

2.1. **Un exemple complet façon « *diaporama* ».** Sur le lieu de téléchargement de ce document se trouve un fichier PDF de chemin relatif `bezout-coef-for-human/slide-version.pdf` présentant la méthode sous la forme d'un diaporama. Nous vous conseillons de le regarder avant de lire les explications ci-après.

2.2. **Phase 1 – Au début était l'algorithme d'Euclide...** Pour chercher des coefficients de Bachet-Bézout pour $(a; b) = (27; 141)$, on commence par appliquer l'algorithme d'Euclide « *verticalement* » comme suit.

141

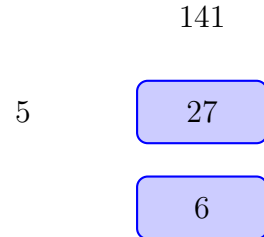
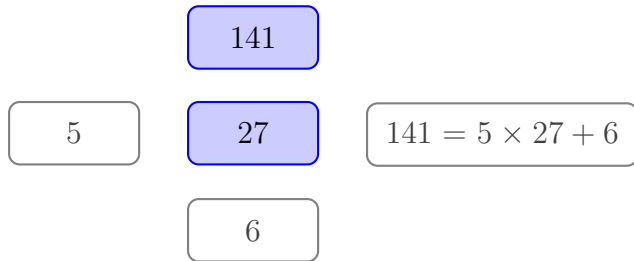
141

27

27

Étape 1 : le plus grand naturel est mis au-dessus.

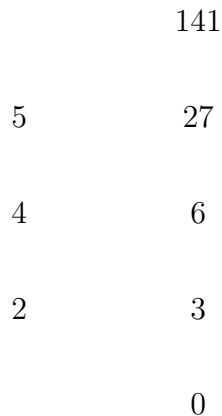
Étape 2 : deux naturels à diviser.



Étape 3 : première division euclidienne.

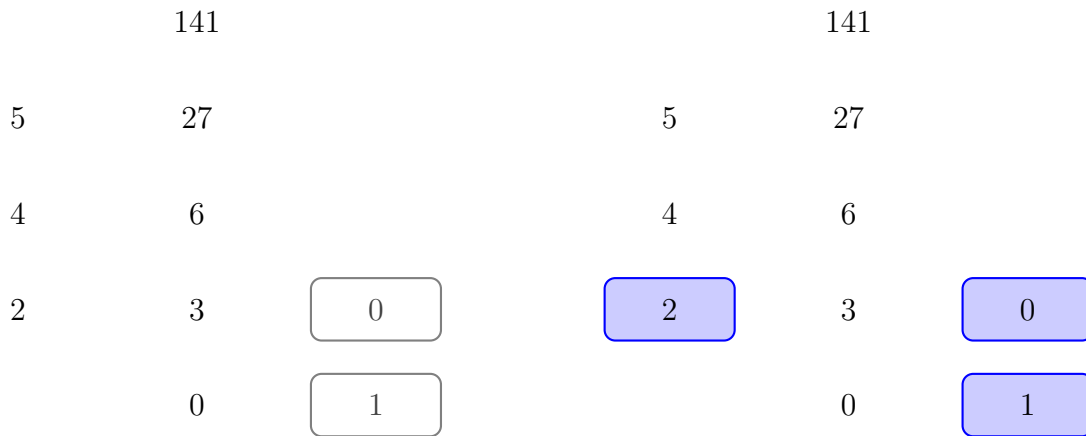
Étape 4 : on passe aux deux naturels suivants.

En répétant ce processus, nous arrivons à la représentation suivante où nous n'avons pas besoin de garder la trace des divisions euclidiennes.



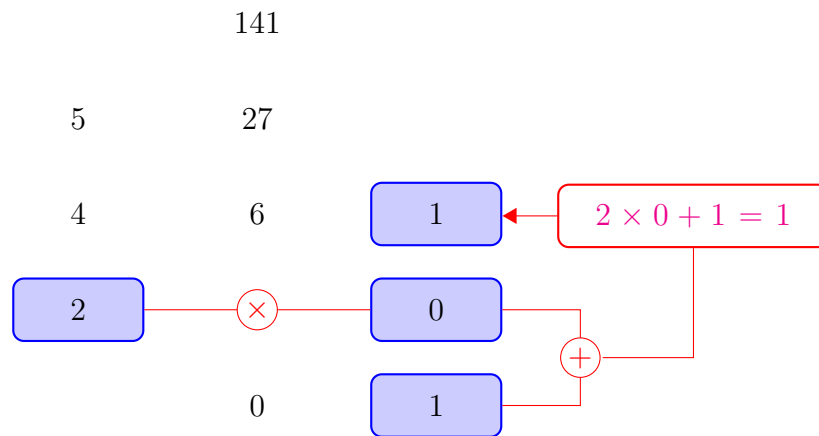
Étape finale (1^{re} phase) : l'algorithme d'Euclide « vertical ».

2.3. **Phase 2 – Remontée facile des étapes.** La méthode classique consiste à remonter les calculs. Mais comment faire cette remontée tout en évitant un claquage neuronal ? L'astuce est la suivante.

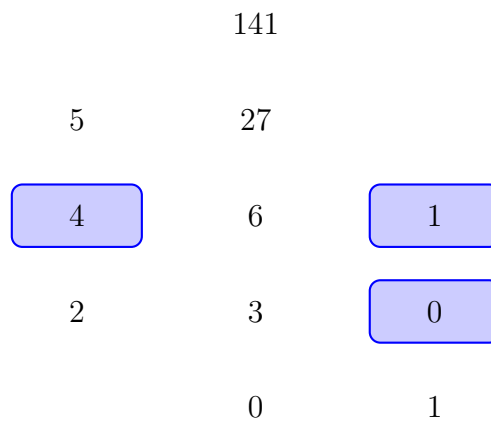


Étape 1 : ajout d'une nouvelle colonne.

Étape 2 : on n'utilise pas la colonne centrale.



Étape 3 : on fait une sorte de division « inversée ».



Étape 4 : on passe aux trois naturels suivants.

En répétant ce processus, nous arrivons à la représentation suivante.

	141	21
5	27	4
4	6	1
2	3	0
	0	1

Étape finale (2^e phase) : remontée en voie libre des calculs.

Remarque 1. *En remontant les calculs sur la colonne centrale, on dispose d'un moyen simple de construire deux entiers a et b de pgcd fixé et avec des valeurs des quotients intermédiaires q_k choisis.*

2.4. Et voilà comment conclure !

	141	21	
5	27	4	$141 \times 4 - 21 \times 27 = -3$
4	6	1	
2	3	0	
	0	1	

Étape finale (la vraie) : on finit avec un produit en croix.

Des coefficients de Bachet-Bézout s'obtiennent sans souci via l'équivalence suivante où nous avons $3 = \text{pgcd}(27, 141)$.

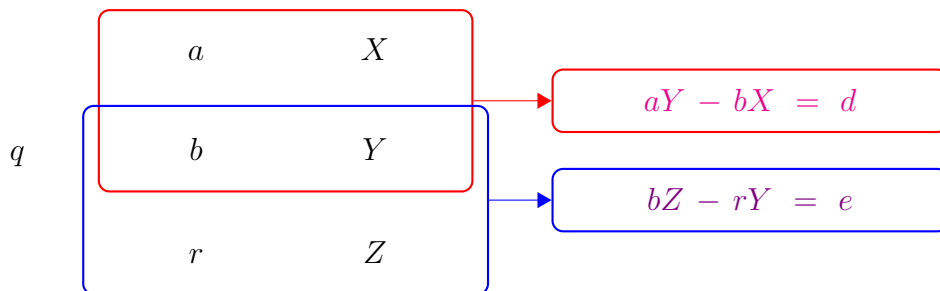
$$141 \times 4 - 27 \times 21 = -3 \iff 27 \times 21 - 141 \times 4 = 3$$

Nous allons voir, dans la section qui suit, que l'on obtient forcément à la fin $\pm \text{pgcd}(27, 141)$.

En pratique, nous n'avons pas besoin de détailler les calculs comme nous l'avons fait à certains moments afin d'expliquer comment procéder. Avec ceci en tête, on comprend toute l'efficacité de la méthode présentée, mais pas encore justifiée, car il suffit de garder une trace minimale, mais complète, des étapes tout en ayant à chaque étape des opérations assez simples à effectuer. Il reste à démontrer que notre méthode marche à tous les coups. Ceci est le propos de la section suivante.

3. POURQUOI CELA MARCHE-T-IL ?

3.1. **Avec des arguments élémentaires.** Commençons par une preuve vérificative qui malheureusement ne nous permet pas de voir d'où vient l'astuce (*nous explorerons ceci dans la sous-section suivante*).



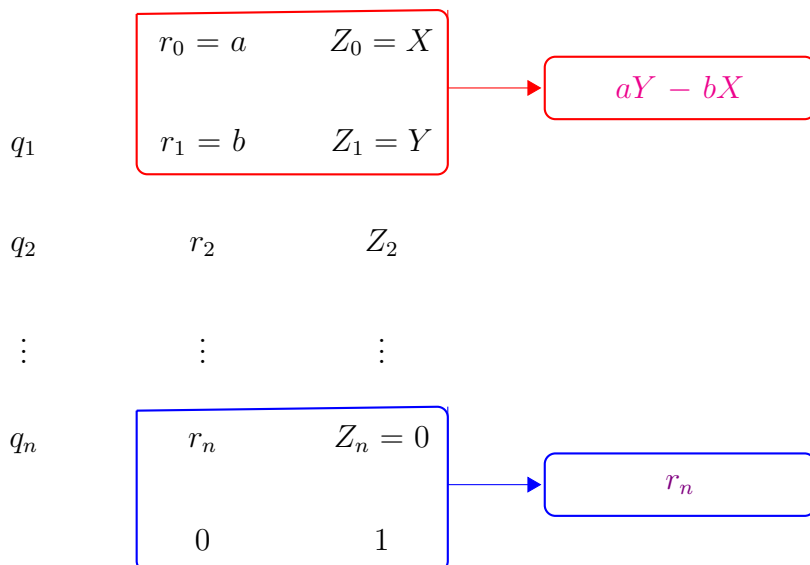
Calculs faits dans les deux phases.

Par construction, nous avons $a = qb + r$ et $X = qY + Z$. Ceci nous donne :

$$\begin{aligned}
 d &= aY - bX \\
 &= (qb + r)Y - b(qY + Z) \\
 &= rY - bZ \\
 &= -e
 \end{aligned}$$

Donc si l'on fait « *glisser* » des carrés sur les deux colonnes de droite, les produits en croix dans ces carrés ne différeront que par leur signe.

La représentation symbolique « *complète* » ci-dessous donne $aY - bX = \pm \text{pgcd}(a; b)$ car le dernier reste non nul de l'algorithme d'Euclide est $\text{pgcd}(a; b)$. Ceci prouve la validité de la méthode dans le cas général. On comprend au passage l'ajout initial du 0 et du 1 dans la 3^e colonne. Bien entendu, (-1) aurait pu convenir à la place de 1, et l'on constate que 0 peut être remplacé par n'importe quelle valeur entière.



Représentation symbolique au complet.

3.2. Avec des matrices pour y voir plus clair. Oublions tout ce que nous avons vu précédemment. Soit $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $a > b$. Nous cherchons $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$. La petite astuce est de noter que $au + bv = \det M$ où $M = \begin{pmatrix} a & -v \\ b & u \end{pmatrix}$.

Nous allons poser $X = -v$ et $Y = u$ de sorte que $M = \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$ et raisonner en supposant l'existence de u et v ¹.

Soit ensuite $a = qb + r$ la division euclidienne de a par b . L'algorithme d'Euclide nous fait alors travailler avec $(b; r)$ au lieu de $(a; b)$. Comme $r = a - qb$, on peut considérer la matrice $N = \begin{pmatrix} a - qb & X - qY \\ b & Y \end{pmatrix}$ qui vérifie $\det N = \det M$ puis, afin d'avoir b en haut, la matrice $P = \begin{pmatrix} b & Y \\ a - qb & X - qY \end{pmatrix}$ qui vérifie $\det P = -\det M$.

Notant $Z = X - qY$, de sorte que $P = \begin{pmatrix} b & Y \\ r & Z \end{pmatrix}$, nous avons $X = Z + qY$. Ceci justifie la construction utilisée lors de la phase de remontée.

Pour passer à une nouvelle preuve, notons que $\begin{pmatrix} b & Y \\ r & Z \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \cdot \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$ puis introduisons les notations suivantes.

- $r_0 = a$, $r_1 = b$, $Z_0 = X$ et $Z_1 = Y$ où Z_0 et Z_1 ne sont pas connus pour le moment.
- Pour $k \in \mathbb{N}^*$, on note $r_{k-1} = r_k q_k + r_{k+1}$ la division euclidienne de r_{k-1} par r_k , puis ensuite on pose $Z_{k+1} = Z_{k-1} - Z_k q_k$ de sorte que $Z_{k-1} = Z_k q_k + Z_{k+1}$.
- On note enfin $M_k = \begin{pmatrix} r_k & Z_k \\ r_{k+1} & Z_{k+1} \end{pmatrix}$ pour $k \in \mathbb{N}$ et $Q_k = \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix}$ pour $k \in \mathbb{N}^*$ de sorte que nous avons $M_{k+1} = Q_{k+1} \cdot M_k$ pour $k \in \mathbb{N}$. Il est immédiat que Q_k est inversible d'inverse $R_k = \begin{pmatrix} q_k & 1 \\ 1 & 0 \end{pmatrix}$ avec $\det R_k = -1$.

L'algorithme d'Euclide nous donne l'existence de $n \in \mathbb{N}$ un indice minimal tel que $r_{n+1} = 0$, et que pour cet indice nous avons $r_n = \text{pgcd}(a; b)$.

Comme $M_n = \prod_{k=n}^1 Q_k \cdot M_0$, nous avons $M_0 = \prod_{k=1}^n R_k \cdot M_n$ avec $M_0 = \begin{pmatrix} r_0 & Z_0 \\ r_1 & Z_1 \end{pmatrix} = \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$ et $M_n = \begin{pmatrix} r_n & Z_n \\ 0 & Z_{n+1} \end{pmatrix} = \begin{pmatrix} \text{pgcd}(a; b) & Z_n \\ 0 & Z_{n+1} \end{pmatrix}$.

Comme $\det M_0 = \pm 1 \det M_n$ car $\det R_k = -1$, il suffit de choisir $Z_{n+1} = 1$, avec Z_n quelconque, pour avoir $aY - bX = \pm \text{pgcd}(a; b)$. Voilà comment découvrir la méthode visuelle vue précédemment où le choix particulier $Z_n = 0$ sert juste à simplifier les premiers calculs.

Remarque 2. Comme Z_n peut être quelconque, nous pouvons produire une infinité de coefficients de Bachet-Bézout. En effet, les étapes de « remontée » sont du type $Z_{k-1} = Z_k q_k + Z_{k+1}$ avec toujours $q_k > 0$, donc des valeurs naturelles non nulles² différentes de Z_n produiront des valeurs différentes de Z_0 .

1. Ce n'est qu'à la fin de la preuve que nous aurons effectivement prouver l'existence de u et v .

2. En fait, il n'est pas besoin d'imposer une condition particulière à Z_n . Voyez-vous pourquoi?

4. DES COEFFICIENTS VIA DES ALGORITHMES PROGRAMMABLES

4.1. **La version « humaine » à la main.** Il n'est pas dur de coder directement la méthode humaine par descente puis remontée³. Voici un algorithme, peu efficace mais instructif, où \star est un symbole à part, $R[-1]$ le dernier élément de la liste R et $R[-2]$ l'avant-dernier, et enfin $[x, y] + [r, s, t] \stackrel{\text{déf}}{=} [x, y, r, s, t]$ (*additionner des listes c'est les concaténer et donc $R + [r]$ est un raccourci pour « on ajoute l'élément r à droite de la liste R »*).

Algorithme 1 : Descente et remontée avec du papier et un crayon

Donnée : $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $b \leq a$

Résultat : $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$

Actions

```
# Phase de descente
# Q est une liste qui va stocker les quotients entiers  $q_k$ .
# R est une liste qui va stocker les restes  $r_k$  (rappelons que
#  $r_0 = a$  et  $r_1 = b$ ).
 $Q \leftarrow [\star]$  ;  $R \leftarrow [a, b]$ 
Tant Que  $R[-1] \neq 0$  :
     $\alpha \leftarrow R[-2]$  ;  $\beta \leftarrow R[-1]$ 
     $\alpha = q\beta + r$  est la division euclidienne standard.
     $Q \leftarrow Q + [q]$  ;  $R \leftarrow R + [r]$ 

# Phase de remontée
# Z est une liste qui va stocker les entiers tout à droite.
 $\varepsilon \leftarrow 1$  ;  $Z \leftarrow [1, 0]$  ;  $c \leftarrow (-1)$ 
Tant Que  $Q[c] \neq \star$  :
     $z \leftarrow Q[c] \cdot Z[-2] + Z[-1]$ 
     $Z \leftarrow Z + [z]$ 
     $\varepsilon \leftarrow (-\varepsilon)$  ;  $c \leftarrow c - 1$ 

# On gère le signe devant le pgcd grâce à  $\varepsilon$ .
 $u \leftarrow \varepsilon \cdot Z[-2]$  ;  $v \leftarrow (-\varepsilon \cdot Z[-1])$ 
Renvoyer  $(u; v)$ 
```

Nous avons traduit brutalement ce que l'on fait humainement mais à bien y regarder, la seule liste dont nous avons réellement besoin est Q . On peut donc proposer la variante suivante programmable qui est à la fois proche de la version de descente et remontée tout en limitant l'impact sur la mémoire.

3. Sur le lieu de téléchargement du document que vous lisez, se trouvent les fichiers `down.py` et `up.py` dans le dossier `bezout-coef-for-human/euclid2tikz`. Ces codes traduisent directement la méthode à la main par descente puis remontée.

Algorithme 2 : Descente et remontée moins mémophage**Donnée** : $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $b \leq a$ **Résultat** : $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$ **Actions**

Phase de descente

 $Q \leftarrow [\star]$ **Tant Que** $b \neq 0$: $a = qb + r$ est la division euclidienne standard. $Q \leftarrow Q + [q]$ $a \leftarrow b ; b \leftarrow r$

Phase de remontée

 $u \leftarrow 1 ; v \leftarrow 0$ $\varepsilon \leftarrow 1 ; c \leftarrow (-1)$ **Tant Que** $Q[c] \neq \star$: $temp \leftarrow Q[c]v + u ; u \leftarrow v ; v \leftarrow temp$ $\varepsilon \leftarrow (-\varepsilon) ; c \leftarrow c - 1$ $u \leftarrow \varepsilon \cdot u ; v \leftarrow (-\varepsilon \cdot v)$ **Renvoyer** $(u; v)$

4.2. **La version « humaine » via les matrices.** Voici la version matricielle de l'algorithme de remontée et descente où de nouveau on limite l'impact sur la mémoire. La matrice R correspond au produit cumulé des matrices R_k .

Algorithme 3 : Descente et remontée via les matrices**Donnée** : $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $b \leq a$ **Résultat** : $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$ **Actions** $a_0 \leftarrow a ; \varepsilon \leftarrow 1$ $R \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ **Tant Que** $b \neq 0$: $a = qb + r$ est la division euclidienne standard. $R \leftarrow R \cdot \begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$ $a \leftarrow b ; b \leftarrow r$ $\varepsilon \leftarrow (-\varepsilon)$ $R \leftarrow R \cdot \begin{pmatrix} a_0 & 0 \\ 0 & \varepsilon \end{pmatrix}$ $u \leftarrow R_{22} ; v \leftarrow (-R_{12})$ **Renvoyer** $(u; v)$

4.3. **Tailles des coefficients lors de la remontée.** Nous redonnons la représentation symbolique complète, ceci afin de rappeler les notations utilisées.

$$\begin{array}{ccc}
& r_0 = a & Z_0 = X \\
q_1 & r_1 = b & Z_1 = Y \\
q_2 & r_2 & Z_2 \\
\vdots & \vdots & \vdots \\
q_n & r_n & Z_n = 0 \\
0 & 1 &
\end{array}$$

Représentation symbolique au complet.

Pour $k \in \llbracket 1; n \rrbracket$, nous savons que $Z_{k-1} = q_k Z_k + Z_{k+1}$ avec $(Z_n; Z_{n+1}) = (0; 1)$, et aussi que $r_{k-1} = q_k r_k + r_{k+1}$ avec $(r_n; r_{n+1}) = (\text{pgcd}(a; b); 0)$.

- Le cas minimal est $n = 1$ puisque $1 \stackrel{\text{hyp}}{\leq} b \stackrel{\text{hyp}}{\leq} a$ (en fait $n = 1$ correspond au cas où $b \mid a$). Nous devons donc calculer au moins un quotient q_k .
- Nous avons clairement $0 \leq Z_n < r_n$.
- Comme $Z_{n-1} = q_n Z_n + Z_{n+1} = 1$, $r_{n-1} = q_n r_n + r_{n+1} = q_n r_n$ et $r_n < r_{n-1}$ par définition de la division euclidienne standard, nous avons $1 \leq Z_{n-1} < r_{n-1}$. Notons au passage que $q_n \geq 2$.
- Supposons maintenant que $n > 1$. Comme $Z_{n-2} = q_{n-1} Z_{n-1} + Z_n$, il est clair que $Z_{n-2} \geq 1$. De plus, nous avons :

$$\begin{aligned}
Z_{n-2} &= q_{n-1} Z_{n-1} + Z_n \\
&< q_{n-1} r_{n-1} + r_n \\
&= r_{n-2}
\end{aligned}$$

- Une récurrence descendante finie nous donne que $\forall k \in \llbracket 0; n \rrbracket$, $0 \leq Z_k < r_k$.

D'après ce qui précède et comme de plus la suite finie $(r_k)_{0 \leq k \leq n+1}$ est décroissante, nous savons que $\forall k \in \llbracket 1; n \rrbracket$, $0 \leq Z_k < r_1 = b$ et $0 \leq Z_0 < r_0 = a$. En particulier $(u; v) = \pm(Z_1; -Z_0)$ qui est tel que $au + bv = \text{pgcd}(a; b)$ vérifie aussi $0 \leq |u| < b$ et $0 \leq |v| < a$.

Remarque 3. *Le résultat précédent empêche toute explosion en taille des calculs intermédiaires. Ceci est une très bonne chose !*

Remarque 4. *Il n'est pas dur de vérifier que la suite $(Z_k)_{0 \leq k \leq n}$ est strictement décroissante.*

Remarque 5. *Notant $d = \text{pgcd}(a; b)$, nous avons en fait $0 \leq |u| < \frac{b}{2d}$ et $0 \leq |v| < \frac{a}{2d}$, et plus généralement $\forall k \in \llbracket 1; n \rrbracket$, $0 \leq Z_k < \frac{b}{2d}$. Ceci vient des deux constatations suivantes.*

- (1) *Tout d'abord en notant que $r_n \leq \frac{1}{2} r_{n-1}$, nous avons $0 \leq |u| < \frac{b}{2}$ et $0 \leq |v| < \frac{a}{2}$.*
- (2) *Posons $a' = \frac{a}{d}$ et $b' = \frac{b}{d}$. Les suites $(q'_k)_k$ et $(r'_k)_k$ associées à a' et b' sont tout simplement $(q_k)_k$ et $(\frac{r_k}{d})_k$, la deuxième suite n'étant pas utilisée pour la phase de remontée. Pour comprendre cela il suffit de noter que si $a = bq + r$ désigne la division euclidienne standard alors $\frac{a}{d} = \frac{b}{d}q + \frac{r}{d}$ en est aussi une.*

4.4. **Pas terribles...** Les algorithmes 2 et 3 vus ci-dessus sont informatiquement très maladroits. Voici pourquoi.

- (1) L'algorithme 2 utilise une liste de taille la somme de 1 et du nombre d'étapes de l'algorithme d'Euclide pour calculer $\text{pgcd}(a; b)$. Dans la section 7, nous verrons que ce nombre d'étapes est environ égal à 5 fois le nombre de chiffres de l'écriture décimale du plus petit des deux entiers a et b . Donc si l'on travaille avec des entiers de tailles assez grandes, la taille de la liste risque de devenir problématique sur du matériel où l'usage de la mémoire est critique (*penser aux objets connectés*).
- (2) Le problème avec l'algorithme 3 est le produit cumulé des matrices qui cache beaucoup d'opérations intermédiaires. Il serait bien de pouvoir s'en passer !

Dans la section qui suit nous allons voir que l'on peut chercher plus efficacement des coefficients de Bachet-Bézout, et ceci sans faire appel ni à des raisonnements avancés, ni à un algorithme complexe dans sa structure.

5. UN ALGORITHME CLASSIQUE BIEN PLUS EFFICACE

5.1. **On peut faire mieux !** Dans la section 3.1, nous avons vu que la clé de la réussite de l'algorithme de descente et remontée est l'égalité $aY - bX = bZ - rY$ dans la représentation ci-dessous où $a = qb + r$ est la division euclidienne standard et $X = qY + Z$.

$$\begin{array}{cc}
 & \boxed{\begin{array}{cc} a & X \end{array}} \\
 q \quad & \boxed{\begin{array}{cc} b & Y \\ r & Z \end{array}}
 \end{array}$$

Calculs faits dans les deux phases.

Nous avons donc exhibé un invariant et dès que l'on arrive à $r = 0$, c'est à dire à la fin de la phase de descente, nous pouvons avoir $bZ - rY = \text{pgcd}(a; b)$ grâce au choix $Z = 1$, et du coup en remontant les calculs nous arrivons à nos fins (*au signe près*).

La méthode précédente est peu efficace à cause de la nécessité de mémoriser certains calculs pour la phase de remontée. Ceci est une contrainte forte ! Nous allons essayer de nous passer de cette nécessité de mémoriser des choses. Pour cela repartons de la représentation symbolique « complète » ci-dessous où $r_{n+1} = 0$ et $r_n = \text{pgcd}(a; b)$.

$$\begin{array}{ccc}
 & r_0 = a & Z_0 \\
 q_1 & r_1 = b & Z_1 \\
 q_2 & r_2 & Z_2 \\
 \vdots & \vdots & \vdots \\
 q_n & r_n & Z_n \\
 & r_{n+1} & Z_{n+1}
 \end{array}$$

Représentation symbolique au complet où $r_{n+1} = 0$.

Ce qui fait fonctionner l'algorithme de descente puis remontée c'est que les r_k et les Z_k vérifient la même relation de récurrence.

- (1) $r_{k+2} = r_k - q_{k+1}r_{k+1}$ car $r_k = q_{k+1}r_{k+1} + r_{k+2}$ est la division euclidienne standard.
- (2) $Z_{k+2} = Z_k - q_{k+1}Z_{k+1}$ soit $Z_k = q_{k+1}Z_{k+1} + Z_{k+2}$ par définition.

Nous allons essayer de construire deux suites (u_k) et (v_k) telles que $au_k + bv_k = r_k$ car nous aurons alors la relation de Bachet-Bézout $au_n + bv_n = r_n = \text{pgcd}(a; b)$. Étant donné ce qui précède, il est maintenant naturel de supposer que $u_{k+2} = u_k - q_{k+1}u_{k+1}$ et $v_{k+2} = v_k - q_{k+1}v_{k+1}$.

En effet, ceci nous donne :

$$\begin{aligned}
 au_{k+2} + bv_{k+2} &= a(u_k - q_{k+1}u_{k+1}) + b(v_k - q_{k+1}v_{k+1}) \\
 &= au_k + bv_k - q_{k+1}(au_{k+1} + bv_{k+1}) \\
 &= r_k - q_{k+1}r_{k+1} \\
 &= r_{k+2}
 \end{aligned}$$

Il nous reste à trouver les valeurs initiales. Ceci est immédiat puisque nous avons :

- (1) $au_0 + bv_0 = r_0 = a$ donc $(u_0; v_0) = (1; 0)$ s'impose.
- (2) $au_1 + bv_1 = r_1 = b$ donc $(u_1; v_1) = (0; 1)$ s'impose.

Notons que nécessairement $n \geq 1$. Nous voilà prêts à proposer un algorithme classique et efficace pour déterminer des coefficients de Bachet-Bézout.

Algorithme 4 : Classique et efficace

Donnée : $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $b \leq a$

Résultat : $(u; v) \in \mathbb{Z} \times \mathbb{Z}$ tel que $au + bv = \text{pgcd}(a; b)$

Actions

$u' \leftarrow 1; u'' \leftarrow 0$

$v' \leftarrow 0; v'' \leftarrow 1$

Tant Que $b \neq 0$:

$a = qb + r$ est la division euclidienne standard.

$temp_u \leftarrow u' - qu''; u' \leftarrow u''; u'' \leftarrow temp_u$

$temp_v \leftarrow v' - qv''; v' \leftarrow v''; v'' \leftarrow temp_v$

Renvoyer $(u'; v')$

5.2. Tailles des coefficients de la méthode efficace.⁴

Dans l'algorithme précédent nous avons défini les suites (u_k) et (v_k) par les relations de récurrence $u_{k+2} = u_k - q_{k+1}u_{k+1}$ et $v_{k+2} = v_k - q_{k+1}v_{k+1}$ couplées avec les conditions initiales $(u_0; v_0) = (1; 0)$ et $(u_1; v_1) = (0; 1)$ où les q_k sont les quotients intermédiaires de l'algorithme d'Euclide. Ces suites fournissent les coefficients de Bachet-Bézout u_n et v_n .

Reprenons une interprétation matricielle comme nous l'avons fait de façon féconde pour l'algorithme de descente puis remontée. Posant $A_k = \begin{pmatrix} u_k & v_k \\ u_{k+1} & v_{k+1} \end{pmatrix}$, nous avons $A_{k+1} = Q_k A_k$ où $Q_k = \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix}$. Ceci nous fournit l'invariant $\det A_{k+1} = -\det A_k = \pm \det A_0 = \pm 1$ ⁵.

Commençons par tenter d'évaluer la taille de u_{n+1} . Par construction, notant $d = \text{pgcd}(a; b)$, nous avons $A_n \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r_n \\ r_{n+1} \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}$.

Posant ensuite $a' = \frac{a}{d}$ et $b' = \frac{b}{d}$, nous avons $A_n \begin{pmatrix} a' \\ b' \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Ceci nous permet d'avoir $\begin{pmatrix} a' \\ b' \end{pmatrix} = A_n^{-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \pm \begin{pmatrix} v_{n+1} & -v_n \\ -u_{n+1} & u_n \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \pm \begin{pmatrix} v_{n+1} \\ -u_{n+1} \end{pmatrix}$ grâce à $\det A_n = \pm 1$. Nous avons

4. Nous reprenons, en la précisant, la preuve page 50 du livre « *Cours de calcul formel – Algorithmes fondamentaux* » de Philippe Saux Picart aux éditions Ellipses.

5. Nous venons d'établir que $u_k v_{k+1} - u_{k+1} v_k = \pm 1$, une identité qui ne coule pas de source sans utiliser des matrices.

établi que $a' = \pm v_{n+1}$ et $b' = \mp u_{n+1}$. Il nous reste à « remonter la récurrence » pour déduire de $u_{n+1} = \pm b'$ des majorations des valeurs des précédents u_k . Le cas des v_k sera similaire à traiter comme nous allons le constater.

Une étude informatique⁶ permet de conjecturer rapidement les choses suivantes.

- $\forall k \in \llbracket 0; n+1 \rrbracket$, $u_{2p} > 0$ si $k = 2p$ est pair, $u_{2p+1} \leq 0$ si $k = 2p+1$ est impair, et même $u_{2p+1} < 0$ si $2p+1 \geq 3$.
- La suite $(|u_k|)_{k \in \llbracket 1; n+1 \rrbracket}$ semble être croissante. Si tel est le cas, nous aurons la majoration $|u_k| \leq b'$ pour $k \in \llbracket 1; n+1 \rrbracket$ (en réalité, nous allons faire un peu mieux).

Commençons par démontrer par récurrence sur $p \in \mathbb{N}$ que si $2p \leq n+1$ alors $u_{2p} > 0$, et si $2p+1 \leq n+1$ alors $u_{2p+1} \leq 0$ avec aussi $u_{2p+1} < 0$ si de plus $2p+1 \geq 3$.

- **Cas de base :** comme $(u_0; u_1) = (1; 0)$, nous avons bien le début de la récurrence.
- **Hérédité :** supposons avoir $2(p+1) \leq n+1$. Comme $2p < 2p+1 \leq n$, nous avons par hypothèse de récurrence $u_{2p} \geq 0$ et $u_{2p+1} \leq 0$.

$u_{2p+2} = u_{2p} - q_{2p+1}u_{2p+1}$ implique que $u_{2p+2} > -q_{2p+1}u_{2p+1} = |q_{2p+1}u_{2p+1}| > 0$. Nous obtenons au passage une information précise à savoir que $|u_{2p+2}| > q_{2p+1}|u_{2p+1}|$.

Supposons ensuite avoir $2(p+1)+1 \leq n+1$. De nouveau $u_{2p} > 0$ et $u_{2p+1} \leq 0$ par hypothèse de récurrence.

$u_{2p+3} = u_{2p+1} - q_{2p+2}u_{2p+2}$ implique que $u_{2p+3} < -q_{2p+2}u_{2p+2} \leq 0$. Nous obtenons au passage une information précise à savoir que $|u_{2p+3}| > q_{2p+2}|u_{2p+2}|$.

La preuve par récurrence précédente nous a fourni $|u_{k+1}| > q_k|u_k|$ dès que $k \in \llbracket 1; n \rrbracket$. Ceci implique la croissance stricte de la suite $(|u_k|)_{k \in \llbracket 1; n+1 \rrbracket}$. De plus dans la section 4.3, nous avons vu que $q_n \geq 2$. Combiné avec $|u_{n+1}| > q_n|u_n|$, ceci nous donne $|u_n| < \frac{1}{2}|u_{n+1}| = \frac{b'}{2}$ puis $\forall k \in \llbracket 1; n \rrbracket$, $|u_k| < \frac{b'}{2}$.

On prendra garde au cas particulier de $|u_0| = 1$. En fait, cela est sans intérêt car nous avons forcément $n \geq 1$ et donc on se soucie peu du cas particulier de $|u_0|$.

Le cas de la suite (v_k) est similaire à traiter, sans particularité pour v_0 , via la propriété suivante à démontrer : $\forall p \in \mathbb{N}$, si $2p \leq n+1$ alors $v_{2p} \leq 0$ avec de plus $u_{2p} < 0$ dès que $2p \geq 2$, et si $2p+1 \leq n+1$ alors $v_{2p+1} > 0$. On obtient alors $\forall k \in \llbracket 0; n \rrbracket$, $|v_k| < \frac{a'}{2}$.

En résumé, nous n'avons de nouveau pas d'explosion des tailles des nombres u_k et v_k . Ceci rend donc l'algorithme 4 est à la fois efficace dans sa gestion de la mémoire et peu gourmand en calculs intermédiaires.

6. Sur le lieu de téléchargement de ce document, voir le fichier Python ayant pour chemin relatif `bezout-coef-for-human/algo-efficient/bachetbezout.py`.

6. UNE INFINITÉ DE COEFFICIENTS DE BACHET-BÉZOUT

Les algorithmes 2, 3 et 4 donnent chacun l'existence de coefficients de Bachet-Bézout u et v pour $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $b \leq a$ de sorte que l'on ait $au + bv = d$ où $d = \text{pgcd}(a; b)$.

Considérons un autre couple $(x; y) \in \mathbb{Z} \times \mathbb{Z}$ tel que $ax + by = d$. Par soustraction, nous avons $a(x - u) + b(y - v) = 0$ soit $a(x - u) = -b(y - v)$ puis $a'(x - u) = -b'(y - v)$ en posant $a' = \frac{a}{d}$ et $b' = \frac{b}{d}$.

Comme $\text{pgcd}(a'; b') = 1$, d'après le lemme de Gauss $a' \mid (y - v)$ soit $\exists k \in \mathbb{Z}$ tel que $y - v = ka'$ d'où $a'(x - u) = -b'ka'$ puis $x - u = -kb'$.

En résumé, il est nécessaire que $x = u - kb'$ et $y = v + ka'$ où $k \in \mathbb{Z}$. Cette condition étant clairement suffisante, nous savons que tous les coefficients de Bachet-Bézout sont du type $(x; y) = (u - k\frac{b}{d}; v + k\frac{a}{d})$ avec $k \in \mathbb{Z}$.

Remarque 6. *Comme pour deux valeurs consécutives de $k \in \mathbb{Z}$, les $(u - kb')$ et $(v + ka')$ associés ne diffèrent en valeur absolue que de b' et a' respectivement, nous pouvons affirmer qu'il n'existe qu'un seul couple $(u; v)$ de coefficients de Bachet-Bézout vérifiant $|u| < \frac{b'}{2}$ et $|v| < \frac{a'}{2}$.*

7. NOMBRE D'ÉTAPES DE L'ALGORITHME D'EUCLIDE

Tous les algorithmes vus précédemment s'appuient sur l'algorithme d'Euclide. Nous allons donc chercher à évaluer le nombre d'étapes de l'algorithme d'Euclide ce qui permettra d'estimer, sans effort, la complexité des algorithmes présentés pour déterminer des coefficients de Bacht-Bézout.

7.1. Une première estimation. Dans la représentation ci-dessous, $(q_k)_{1 \leq k \leq n}$ est la suite des quotients intermédiaires et $(r_k)_{2 \leq k \leq n}$ celle des restes.

$$\begin{array}{rcl}
 & & r_0 = a \\
 & & \\
 q_1 & & r_1 = b \\
 & & \\
 q_2 & & r_2 \\
 & & \\
 \vdots & & \vdots \\
 & & \\
 q_n & & r_n > 0 \\
 & & \\
 & & 0
 \end{array}$$

L'algorithme d'Euclide au complet.

Nous savons que $(q_k)_{1 \leq k \leq n} \subseteq \mathbb{N}^*$ et que $(r_k)_{1 \leq k \leq n} \subseteq \mathbb{N}^*$ est strictement décroissante. Comme $r_k = q_{k+1}r_{k+1} + r_{k+2}$ si $1 \leq k \leq n-2$, nous avons $r_k \geq r_{k+1} + r_{k+2} > 2r_{k+2}$ d'où $r_k > 2r_{k+2}$ dès que $1 \leq k \leq n-2$. Ceci nous donne les majorations suivantes.

- **Cas $n = 2p \geq 2$ est pair.** Comme $b > r_2 > 2r_4 > \dots > 2^{p-1}r_{2p} \geq 2^{p-1}$ où $r_{2p} = r_n$, nous avons $\log(2^{p-1}) < \log b$ puis $p < 1 + \frac{\log b}{\log 2}$ où \log désigne le logarithme décimal. Donc $n < 2 + 2\frac{\log b}{\log 2}$ ici.
- **Cas $n = 2p + 1$ est impair.** Comme $b = r_1 > 2r_3 > \dots > 2^p r_{2p+1} \geq 2^p$ où $r_{2p+1} = r_n$, nous avons $\log(2^p) < \log b$ puis $p < \frac{\log b}{\log 2}$. Donc $n < 1 + 2\frac{\log b}{\log 2}$ ici.

Dans les deux cas, nous avons $n < 2 + 2\frac{\log b}{\log 2}$. Comme $\frac{2}{\log 2} \approx 6,65$, notant d le nombre de chiffres décimaux de b , de sorte que $\log b \leq d + 1$, nous avons l'estimation $n < 2 + 14(d + 1)$ soit $n < 14d + 16$ puis $n \leq 14d + 15$.

En résumé, l'algorithme d'Euclide appliqué à $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ avec $a > b$ demandera au maximum $14d + 15$ étapes où d est le nombre de chiffres décimaux de b .

Remarque 7. *Cette estimation rapide à établir est un peu grossière. Elle est à tout de même le mérite de montrer que le nombre d'étapes augmente de façon logarithmique par rapport à b , le plus petit des entiers naturels a et b .*

8. AFFAIRE À SUIVRE...