Emmanuelle BOYER,
Professeure agrégée de mathématiques,
Lycée Emile Duclaux,
15000 AURILLAC

Juin 2020

Projet de fin de formation DIU pour l'enseignement de NSI

Algorithme de la division simple par différence décrite dans un manuscrit du XI ième siècle par un moine dénommé Bernelin, élève de Gerbert d'Aurillac pape de l'an 1000 sous le nom de Sylvestre 2.

Remerciements :

Je remercie tout particulièrement Aurélie Lagoutte pour son expertise et son aide sur la partie théorique qui m'ont permis de cibler le cœur de l'algorithme.

Je remercie également Malika More, sans qui aucune exploration de ce type ne m'aurait paru envisageable il y a seulement un an et demi et Alex Esbelin pour son accompagnement du côté historique et épistémologique.

Je dédicace tout ce travail à Jean Cassinet, décédé en 1998, qui a su, il y a plus de 25 ans lors des commémorations du millénaire autour du pape Gerbert d'Aurillac, transmettre son enthousiasme pour la diffusion des méthodes de division sur abaque auprès de la jeune professeure de mathématiques que j'étais. Il est à l'origine du livre autour du manuscrit de Bernelin et a contribué à sa traduction.

Table des matières :

I- Description du projet envoyé à Alex Esbelin en février 2020 (feuille de route)

Emmanuelle BOYER
Professeure agrégée de mathématiques
Lycée Emile Duclaux
15000 AURILLAC
En charge de la moitié des heures d'une classe de première NSI pour l'année 2019-2020.
Niveau : débutant

Formation NSI Clermont : grandes lignes du projet envisagé pour la fin de deuxième année juin 2020
(version du 4 février 2020)

Après avoir exploré un peu l'année dernière les pages web, serveurs et bases de données, je me lance cette année dans un projet individuel sur la méthode de la division sur abaque selon Gerbert d'Aurillac pape de l'an mil.

Sujet : étude d'un algorithme de division sur abaque décrit dans un manuscrit en latin du 11^{ième} siècle d'un disciple de Gerbert d'Aurillac appelé Bernelin : la « division simple par différence » obtenue par déplacements de jetons portant des chiffres arabes dans un tableau (abaque). La méthode nécessite des multiplications à un chiffre utilisant une demi-table de multiplication et d'additions.

Objectifs :

1) Réalisation : mise au point de l'algorithme décrit dans le manuscrit, traduction dans le langage python et présentation de la méthode (déplacement des jetons sur l'abaque) à l'aide d'une IHM utilisant Tkinter (la multiplication à 1 chiffre et l'addition seront obtenues par les opérations machines ordinaires).

2) Intérêt pédagogique pour l'enseignement:
- utilisation des éléments de programmation de base (au programme de la classe de première) pour évaluer les difficultés de codage, de faisabilité et les limites de conception que pourraient rencontrer mes élèves.
- Exploration et approfondissement des possibilités de Tkinter avant les vacances de Pâques pour aider les élèves de première NSI dans leur projet de fin d'année (troisième trimestre).
- Réutilisation possible de l'algorithme dans le cadre de projets interdisciplinaires locaux (Aurillac/Moyen age/ pape Gerbert)

3) Approfondissement de la formation NSI

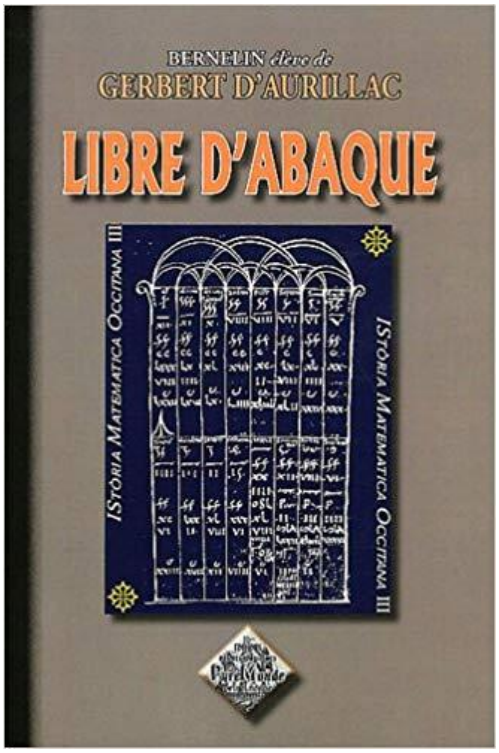
- Donner une estimation de la complexité de l'algorithme comme vu en cours avec Aurélie Lagoutte. En profiter pour explorer l'implémentation machine de la division de deux nombres dans la documentation de vulgarisation disponible et les méthodes de division en base 2. Explorer les différences entre coût des affectations de variables dans un tableau et coût des boucles avec opération. (Réflexion sur le « coût humain » de la méthode au Moyen Age : la méthode n'a pas eu le succès escompté. Relecture prévue en histoire des maths de la querelle entre abacistes et algoristes)

- En juin, essayer de reprendre le projet en programmation orientée objet avec la création du modèle (comme vu en formation avec Marc Chevaldonné). Une approche de la POO est au programme de terminale NSI :
Gestion de l'interface graphique et construction des abaques / méthodes de déplacement et remplacements des jetons/gestion animée de la présentation de la méthode : sur un exemple ou de façon plus simple sur une division avec des nombres donnés par l'utilisateur/gestion des cases du tableau/etc … Selon le temps qui restera…

Dans cette phase de conception, explorer des possibilités d'extension de la méthode comme énoncé dans le manuscrit de Bernelin sur les fractions d'unités anciennes (drachme, once, etc …) avec la manipulation des tables de calcul fournies. Mais qui ne seront pas codées par manque de temps.

Bref, un projet ambitieux (trop ?) qui se limitera peut-être à la première partie par manque de temps disponible. J'ai donc privilégié un planning par difficulté croissante.

Livre support : http://www.sudoc.fr/160168805



Libre d'abaque / Bernelin, élève de Gerbert d'Aurillac ; texte latin établi, traduit et annoté par Béatrice Baklouche [i.e. Bakhouche] ; notes complémentaires de Jean Cassiut [i.e. Cassinet], réalisé d'après le manuscrit du XIe siècle, H 491 de la Bibliothèque de l'Ecole de médecine de Montpellier.

II- Description du contenu technique du projet et des trois dossiers fournis

1- Dossier de fichiers python pour l'algorithme de base utilisé dans le manuscrit, nommé :

projetAbaqueBase

Le code est réparti dans le fichier du programme principal et un fichier contenant les fonctions.

- En entrée : dividende et diviseur.
- L'abaque est représenté par trois tableaux (liste de liste): bloc 1, 2 et 3.
- L'objectif est de respecter au maximum la méthode proposée par Bernelin et de mettre en œuvre l'algorithme correspondant en machine pour construire en console les tableaux successifs obtenus à chaque étape. Seules les quelques multiplications, additions et soustractions seront demandées à l'ordinateur. Avec du temps, il suffirait de les programmer par lecture du résultat dans les tables de multiplications données par Bernelin (tables ordinaires de multiplication à un chiffre) avec gestion de la retenue.
- La priorité est donnée aux déplacements des jetons (affectations des entiers dans les cases des tableaux).
- Les tests en fin de programme permettent de vérifier la bonne mise en œuvre de la division.

2- Dossier de fichiers python, images et textes pour la présentation de la méthode, nommé :

projetAbaqueCompletTkinter

lancer python sur le fichier progPrincipalAbaqueTkinter.py

L'objectif est de présenter la méthode de division sur l'abaque à l'aide de fenêtres Tkinter avec les références pas à pas du manuscrit de Bernelin quand ce n'est pas trop long. Pour faciliter la compréhension, j'ai réalisé un simulateur qui permet de faire fonctionner la méthode pas à pas sur n'importe quel exemple de division.
Le code de base du programme précédent est réutilisé.

De principe, le code a été construit pas à pas, comme le ferait un élève de première NSI en utilisant les fonctions et les variables globales nécessaires à la gestion des évènements permettant le passage de pages en pages. L'idée est d'utiliser un minimum de connaissances de Tkinter disponibles sur le site : http://tkinter.fdex.eu/index.html accessibles en autonomie. (dans la cadre du programme de première : exploration d'une bibliothèque python) Cela m'a permis de mettre en évidence les difficultés que pourraient rencontrer mes élèves pour mener à bien un projet de ce type (des variables globales innombrables, ou des passages de paramètres en trop grande quantité). Seule entorse à mon objectif, une variable stringvar pour la gestion des titres dans les widgets Label.

L'élaboration des pages ayant été améliorée et les pages complétées au fur et à mesure, la cohérence d'ensemble en pâtit.

3- Dossier de fichiers pour la partie du programme correspondant à l'animation élaborée en programmation orientée objet pour faciliter l'utilisation des objets de la bibliothèque Tkinter.

projetAbaqueAnimationPOO

Cette animation a été intégrée au programme final précédent et apparait à la page 4.
Le diagramme de classe de cette partie est donné en annexe

III - Analyse de l'algorithme de la division simple avec différence sur l'abaque de Gerbert.

D'après le manuscrit de Bernelin du XI iéme siècle (écrit entre 999 et 1003 au temps où Gerbert était pape).

(s'aider du logiciel de présentation de l'algorithme « projetAbaqueCompletTkinter » pour comprendre le vocabulaire et les résultats)

1) Preuve de l'algorithme : retour sur les nombres entiers en base 10 formés par les lignes de jetons.

a) Préliminaires importants : retour à la numération décimale de position pour les calculs.

- dividende de longueur n (n chiffres)
- diviseur de longueur p (p chiffres)

Attention : 10^p contient p+1 chiffres

Un nombre de longueur p est donc compris entre 10^{p-1} et 10^p exclu

Analyse des nombres de l'abaque en base 10 et noms utilisés:
Bloc 1 :

Ligne 1 : le diviseur opératoire est 10^p de longueur p+1
Ligne 2 : le diviseur a p chiffres
Ligne 3 : la différence entre le diviseur opératoire et le diviseur a au maximum p chiffres appelée differencediviseur.
Ligne 4 : le dividende a n chiffres

Bloc 2 : à une étape quelconque numérotée q, le dividende partiel obtenu à l'étape précédente est de longueur k.

Ligne 1 : le dividende partiel précédent a pour longueur k auquel on retire le premier jeton donc de longueur maxi k-1.

Le jeton déplacé a pour numéro = E ($\frac{\text{dividende partiel}}{10^{k-1}}$) que l'on déplace dans le bloc 3 avec autant de colonnes d'écart à partir de la colonne de droite qu'entre la longueur du dividende partiel k et la longueur du dividende opératoire p+1 donc la dénomination est égale à : numéro x $10^{k-(p+1)}$

En première ligne le nombre écrit est : dividende partiel – numéro x 10^{k-1}

Ligne 2 : la multiplication est égale à numéro x $10^{k-(p+1)}$ x la differencediviseur
Multiplication= numéro x $10^{k-(p+1)}$ x (10^p – diviseur) = numéro x $10^{k-(p+1)}$ x 10^p – numéro x $10^{k-(p+1)}$ x diviseur
= numéro x 10^{k-1} – numéro x $10^{k-(p+1)}$ x diviseur

Ligne 3 : la somme des deux lignes précédentes est :
dividende partiel – numéro x 10^{k-1} + numéro x 10^{k-1} – numéro x $10^{k-(p+1)}$ x diviseur
= dividende partiel– numéro x $10^{k-(p+1)}$ x diviseur
Or cette somme est le nouveau dividende partiel pour l'étape suivante q + 1

b) Preuve de la terminaison de l'algorithme : fin de la boucle Tant Que

Le processus s'arrête quand dividende partiel < diviseur opératoire pour la boucle Tant Que principale (bloc 1,2 et 3) .
Prenons comme variant de boucle :
la valeur du dividende partiel (ligne 1 du bloc 2 avant qu'on enlève le jeton de gauche) ≥ diviseur opératoire

La suite des dividendes partiels obtenus à chaque étape est une suite de nombres entiers strictement décroissante donc la boucle se termine dès que le variant est false c'est-à-dire dès que le dividende partiel est inférieur au diviseur opératoire.
En effet : en ligne 3 le nouveau dividende partiel est égal à l'ancien dividende partiel diminué du nombre positif :
numéro x $10^{k-(p+1)}$ x diviseur

Si en fin de boucle Tant Que principale, le dividende Partiel est inférieur au diviseur opératoire mais supérieur au diviseur, le processus continue dans une autre boucle Tant que dont il est aisé de voir comme la précédente qu'elle se termine.

c) Preuve de l'algorithme de la boucle principale :
Choisissons comme invariant de boucle la relation :

dividende partiel + dénomination totale x diviseur = dividende
et longueur du dividende partiel ≥ longueur du diviseur opératoire.

Initialisation :
à la première étape, le dividende partiel est égal au dividende et il n'y a pas de jetons dans le bloc 3 , la dénomination totale vaut 0 et la relation est vraie.

Transmission :
Supposons qu'à l'étape q , le dividende Partiel obtenu par la somme du bloc2 de l'étape précédente (c'est-à-dire la ligne 1 du bloc 2 sans le jeton enlevé) vérifie la relation et a pour longueur k:

$(\text{dividende partiel})_q + (\text{dénomination totale})_q \times \text{diviseur} = \text{dividende}$

et longueur du $(\text{dividende partiel})_q \geq$ longueur du diviseur opératoire.

En utilisant les résultats des lignes énoncés en préliminaires le nouveau dividende partiel de l'étape suivante q+1 se trouve en ligne 3 et est égal à :

$(\text{dividende partiel})_{q+1} = (\text{dividende partiel})_q - (\text{numéro})_q \times 10^{k-(p+1)} \times \text{diviseur}$

La dénomination totale de l'étape suivante q+1 qui se trouve en ligne 3 est donc égale à :

$(\text{dénomination totale})_{q+1} = (\text{dénomination totale})_q + (\text{numéro})_q \times 10^{k-(p+1)}$

Ainsi : $(\text{dividende partiel})_{q+1} + (\text{dénomination totale})_{q+1} \times \text{diviseur}$

$= (\text{dividende partiel})_q - (\text{numéro})_q \times 10^{k-(p+1)} \times \text{diviseur}$

$+ ((\text{dénomination totale})_q + (\text{numéro})_q \times 10^{k-(p+1)}) \times \text{diviseur}$

$= (\text{dividende partiel})_q - (\text{numéro})_q \times 10^{k-(p+1)} \times \text{diviseur}$

$+ (\text{dénomination totale})_q \times \text{diviseur} + (\text{numéro})_q \times 10^{k-(p+1)} \times \text{diviseur}$

$= (\text{dividende partiel})_q + (\text{dénomination totale})_q \times \text{diviseur} = \text{dividende}$

et la relation est vraie .
la condition sur la longueur du dividende partiel ne sera vraie que pour effectuer l'étape q+1 sinon le processus s'arrête.

De même si il est nécessaire de rentrer dans la deuxième boucle Tant Que de fin dans le cas où le dividende partiel est inférieur au diviseur opératoire mais supérieur au diviseur, l'invariant de boucle reste le même et la justification de la transmission est la même que précédemment , seule la condition change avec dividende partiel ≥ diviseur.
En sortie de boucle (des deux boucles si nécessaire)on obtient donc :
dividende partiel + dénomination totale x diviseur = dividende avec dividende partiel <diviseur.

On retrouve la définition de la division euclidienne avec :
Quotient = dénomination totale
Reste = dernier dividende partiel

2) Majoration du nombre d'étapes dans la boucle principale.

a) Majoration du nombre d'étapes par colonne de remplissage du bloc 3.

(c'est-à-dire le nombre maximal de jetons pouvant être rajoutés dans une colonne du bloc 3)

Tout va venir de la l'étude des dividendes partiels successifs.
On a obtenu ligne 3 du bloc2 la relation :

(dividende partiel) $_{q+1}$ = (dividende partiel) $_q$ – (numéro) $_q$ x $10^{k-(p+1)}$ x diviseur

L'écart entre deux dividendes partiels consécutifs décroissants est donc de :

(numéro) $_q$ x $10^{k-(p+1)}$ x diviseur = E ($\frac{\text{dividende partiel}}{10^{k-1}}$) x $10^{k-(p+1)}$ x diviseur

= E ($\frac{\text{dividende partiel}}{10^{k-1}}$) x 10^{k} x $\frac{\text{diviseur}}{10^{p+1}}$

= E ($\frac{\text{dividende partiel}}{10^{k-1}}$) x 10^{k} x $\frac{\text{diviseur}}{10^{p-1}}$ x 10^{-2}

= E ($\frac{\text{dividende partiel}}{10^{k-1}}$) x $\frac{\text{diviseur}}{10^{p-1}}$ x 10^{-2} x 10^{k}

Or E ($\frac{\text{dividende partiel}}{10^{k-1}}$) et $\frac{\text{diviseur}}{10^{p-1}}$ sont deux nombres compris entre 1 et 10 exclu, mais 1 est compris dans les deux cas.

Donc l'écart minimal est de 10^{-2} x 10^{k} et les écarts vont s'accumuler de proche en proche.

Or tous les nombres de longueur k sont compris entre 10^{k-1} inclus et 10^{k} exclu, il faut donc (en comptant large !) au maximum 10^{2} écarts de 10^{-2} x 10^{k} pour que le dividende partiel de longueur k devienne au fil du processus de longueur k-1 et donc que la dénomination (jeton déplacé) soit transférée dans la colonne suivante.

On obtient donc une majoration de 100 jetons par colonnes du bloc 3, (contrairement à la première intuition qui donnerait au premier abord une majoration de 10 mais que la simulation infirme).

Remarque :
Le logiciel permet de nombreuses simulations et je pense que la majoration faite a été trop brutale. Les observations laissent envisager un nombre d'étapes maximal de l'ordre d'une petite trentaine par colonne.
Une observation plus fine des simulations donne une piste sur le maximum égal à la somme des multiples des entiers inférieurs à 10 correspondant à la remontée des numéros des dénominations dans la multiplication. Je pense que c'est ce qui se passe dans une majorité de cas.
Mais dans certains cas, le jeton de dénomination suivant se distribue soit dans la valeur inférieure, soit dans la valeur supérieure, il faudrait faire une analyse plus poussée de ces comportements pour une éventuelle démonstration. Elle est due à la somme avec le dividende partiel tronqué. Par exemple : si le jeton de dénomination est 9 , alors ce 9 est multiplié par un « différencediviseur » de premier chiffre au plus égal à 9. Pour 9, 9x9 = 81, le premier chiffre passe à 8… et la dénomination suivante sera de l'ordre de 8 ou inférieure dans le cas général. Mais il faut ajouter le dividende partiel tronqué de la ligne 1 du bloc 2, son premier chiffre sera alors ajouté au deuxième chiffre du 81 soit 1. Dans certains cas , cette somme sera juste au dessus du 90, ce qui n'influera pas la dénomination qui suit sauf dans quelques cas. Bref une démonstration à poursuivre ;-)

1+1+1+1+1+1+1+1+1+1 = 10 pour un totale de 10
2+2+2+2+2 =10 pour un total de 5
3+3+3 = 9 pour un total de 3
4+4 = 8 pour un total de 2
5+5 = 10 pour un total de 2
6 pour un total de 1
7 pour un total de 1
8 pour un total de 1
9 pour un total de 1
(10 + 5 + 3 + 2 + 2 + 1 + 1 + 1 + 1 = 26 …) donc un maximum autour de 26 ?

b) Majoration du nombre d'étapes pour la boucle principale.

A chaque étape, le jeton de dénomination correspondant au jeton gauche du dividende partiel est déplacé dans la colonne du bloc3 avec autant d'écart entre le jeton de gauche du dividende partiel et le jeton 1 du diviseur opératoire. Le dividende partiel diminuant à chaque étape et ayant pour valeur de départ le dividende de la division, le jeton de dénomination va donc se placer successivement de gauche à droite dans n-p colonnes avec un maximum d'au plus 100 jetons par colonnes.

Ce qui donne une majoration du nombre d'étapes de la boucle principale de 100x(n-p)

La colonne vide de gauche sera éventuellement remplie par les retenues de la somme finale des dénominations, elle ne compte pas pour une étape.

3) Estimation large du coût pour effectuer cette division pour un moine du Xième siècle :
n : la longueur du dividende
p : la longueur du diviseur

Je compterai en pire cas le nombre de positionnement de jetons ou de déplacement de jeton manuellement et de calculs simples sur les chiffres (addition de 2 chiffres ou lectures de multiplications à 2 chiffres dans la table)
Je compterai quand même un jeton « 0 » même si la place est vide et ne demande pas de mouvement. J'aurai certainement du compter aussi la lecture dans la table des multiplications (mais je fais confiance au « par cœur » du cerveau humain !). Mais je donnerai un total large pour ne garder que l'ordre de grandeur du coût pour ne pas alourdir le calcul. Je ne donnerai donc qu'une majoration raisonnable du décompte des opérations en pire cas.

- Construction abaque vide , le nombre de colonnes doit être un multiple de 3 : tracer E((n+1)/3+1)x3 colonnes , 3 lignes et E((n+1)/3+1)x2 cercles si ils ne sont pas déjà construits.
Total large: 5x E((n+1)/3+1) + 3 opérations donc 2n +3

- Construction bloc 1 :
  - positionner n jetons pour le dividende
  + 1 jeton numéroté « 1 » pour le diviseur opératoire (c'est facultatif dans le manuscrit)
  + p jetons pour le diviseur
  + effectuer la soustraction entre diviseur opératoire et diviseur et placer au maximum les p jetons obtenus.
    La différence se fait par complément à 10 dans chaque colonne avec une retenue éventuelle donc pour chaque colonne : une addition de retenue sauf celle de droite + un complément à 10 + un placement de jeton donc 3p-1 opérations.
  Total dans pire cas : n + 4p

-Construction bloc 2 :
- ligne 1 : positionner les n jetons du dividende à la première étape ou remonter les au plus p+1 jetons de la troisième ligne de l'étape précédente qui contient le dividende partiel.(en effet, si j'appelle k la longueur du dividende partiel ($p \leq k \leq n$) alors les (k-p-1) jetons de droite ne bougent pas.) donc p+1 opérations.
- déplacer le jeton le plus à gauche dans le bloc 3 avec autant d'écart de colonnes depuis la droite qu'il y en a entre le jeton enlevé et la position du jeton du diviseur opératoire numéroté « 1 ». Ce jeton est appelé dénomination. donc 1 opération.
- ligne 2 : multiplication à un chiffre correspondant au numéro du jeton déplacé par la différence du bloc 1 (au maximum p opérations lues dans la table des multiplications avec éventuelles retenues à ajouter) et placement. donc 3p opérations.
- ligne 3 : somme des deux lignes précédentes, colonne par colonne avec retenue éventuelle (sauf celle de droite) donc au maximum (p+1) additions chiffre à chiffre et placement d'au maximum (p+1) jetons de résultat. (Si on réfléchit bien, il est inutile de faire la somme avec remplacement des jetons : il suffit de faire des tas de jetons dans les colonnes qui peuvent dépasser un total de 10 dans la colonne !) donc 3(p+1) -1 = 3p+2

Total dans pire cas : p + 1 + 1 + 3p + 3p + 2 = 7p + 4
La construction du bloc 2 sera répétée tant que la longueur du dividende partiel est inférieure au dividende donc au plus 100x(n-p) fois (voir détail du calcul donné précédemment)
Total dans pire cas : (7p+4) x 100 x (n-p)

- Eventuellement la construction des blocs 1bis, bloc2bis et rajouts des dénominations obtenues dans le bloc 3 de la division principale. (au plus 10 étapes avec la même quantité d'opérations que précédemment)
Total dans pire cas : 1+ 3p+1 + 10 x 3p = 33 p + 2

-Construction bloc 3 : rien de plus sauf une addition à la fin de toutes les dénominations pour obtenir le quotient final.
Total dans pire cas de 100x(n-p)jetons

_____

Au total, en comptant large en pire cas,
2n + 3 + n + 4p + (7p+4) x 100 x (n-p) + 100x(p+1)

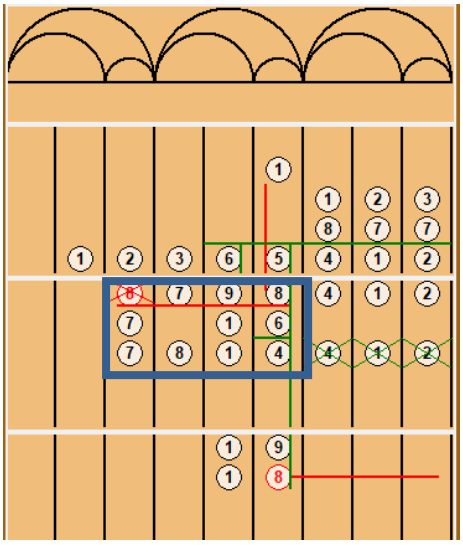
la complexité est de l'ordre de 700 x p x (n-p) en comptant large.

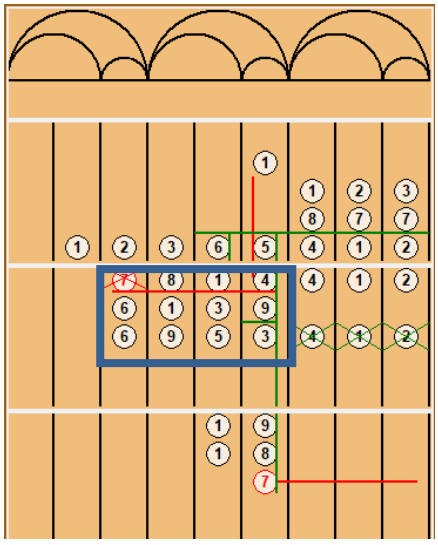
C'est-à-dire 700 x longueur du diviseur x (écart de longueur entre dividende et diviseur)
donc en O(px(n-p))

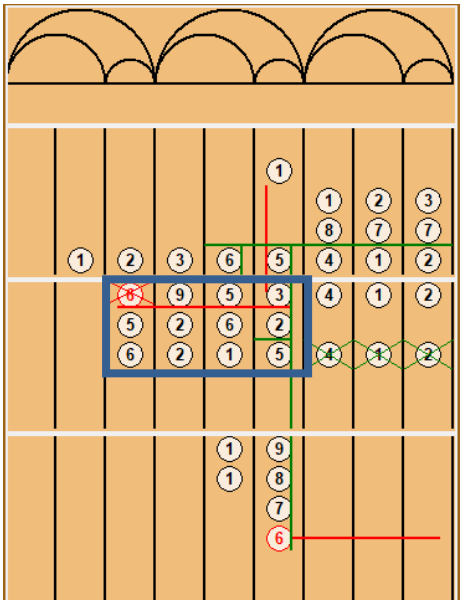
Commentaires : L'observation des simulations confirme que la complexité va dépendre principalement de la longueur du diviseur pour les calculs et de l'écart de longueur entre le dividende et le diviseur pour le nombre d'étapes. En effet, les calculs à chaque étape vont se situer sur une zone de l'abaque comportant 3 lignes et p+1 colonnes. Ce bloc rectangulaire de calcul se déplace de la gauche vers la droite sur n-p colonnes.

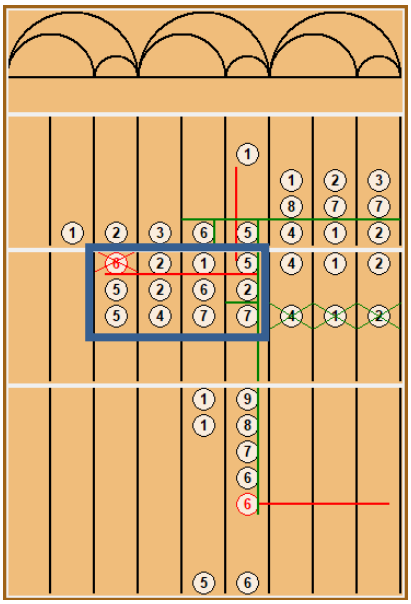
Le code python que j'ai établi à partir du texte du manuscrit (traduction du latin), ne met pas l'accent sur cette zone de calcul, car j'ai privilégié le remplissage de tableaux lignes par lignes avec les calculs de multiplications et d'additions faits en machine (donc sur des conversions de la totalité d'une ligne en nombre entier). Cette approche m'a caché la zone de calcul plus restreinte mise en évidence par la simulation. En reprenant le texte original, on voit bien cette préoccupation de Bernelin, de positionner des doigts (digitis) pour origine des calculs… Mais encore fallait-il, le comprendre et s'y retrouver dans les explications !!!
Suite à cette observation, la conception de l'animation réalisée en POO en tient compte à la fois dans la présentation que dans l'élaboration du code.

Extraits de l'abaque issus de la simulation









4) Calcul de la complexité sur un extrait du programme :

```
def remplissageBLOC2etBLOC3(dividendePartiel,differenceDiviseur,placeDiviseurOP1,tabBLOC3,nbcolonnes):
    dividendePartielChaine=str(dividendePartiel)
    L6=remplissageLigne(dividendePartielChaine,nbcolonnes)

    premierJeton=dividendePartielChaine[0]
    positionPremierJeton=nbcolonnes-(len(dividendePartielChaine)-1)
    ecartPremierJetonPlaceDiviseurOP1=len(dividendePartielChaine)-placeDiviseurOP1
    tabBLOC3=remplissageBLOC3(tabBLOC3,premierJeton,ecartPremierJetonPlaceDiviseurOP1,nbcolonnes)
    dividendeIntermediaireSansPremierJetonChaine=""
    for k in range(len(dividendePartielChaine)-1):
        dividendeIntermediaireSansPremierJetonChaine=dividendeIntermediaireSansPremierJetonChaine+dividendePartielChaine[k+1]
    L6=remplissageLigne(dividendeIntermediaireSansPremierJetonChaine,nbcolonnes)

    multiplication=int(premierJeton)*differenceDiviseur
    multiplicationChaine=str(multiplication)
    if ecartPremierJetonPlaceDiviseurOP1!=0:
        multiplicationChaine=multiplicationChaine+"0"*ecartPremierJetonPlaceDiviseurOP1

    L7=remplissageLigne(multiplicationChaine,nbcolonnes)

    dividendePartiel=int(dividendeIntermediaireSansPremierJetonChaine)+int(multiplicationChaine) # pas du jeu car calcul ordi
    dividendePartielChaine=str(dividendePartiel)

    L8=remplissageLigne(dividendePartielChaine,nbcolonnes)

    tabBLOC2=[L6,L7,L8]
    return (dividendePartiel,tabBLOC2,tabBLOC3)
```

```
# remplissage des BLOC2 et BLOC3 par deplacement des jetons sur l'abaque et multiplication a un chiffre
dividendePartiel=dividende
while dividendePartiel>=diviseur and len(str(dividendePartiel))-placeDiviseurOP1>=0 :
    (dividendePartiel,tabBLOC2,tabBLOC3)=remplissageBLOC2etBLOC3(dividendePartiel,differenceDiviseur,placeDiviseurOP1,tabBLOC3,nombrecolonnesAbaque)
    print(tabBLOC2,tabBLOC3)
```

Une rapide évaluation de la complexité de l'algorithme python sur un extrait correspondant au remplissage des blocs 2 et 3. Le code est issu du programme de base : **projetAbaqueBase**(sans les fenêtres Tkinter)

Les fonctions remplissageBLOC et remplissageLigne remplissent les listes python.
Donc à chaque étape :
- les tableaux de l'abaque sont régénérés à partir des entiers même si les changements dans les tableaux sont peu nombreux : longueur du dividende x 3 x 2 pour le bloc 2 + nombres de jetons dans les colonnes du bloc 3 (le nombre de lignes du bloc 3 augmente à la demande avec la fonction .append, ce nombre est majoré par 100).
- La conversion des listes en chaines de caractères et inversement qui seront elle-même retraduites pour 3 d'entre elles en entiers pour deux opérations de calcul machine: une multiplication à 1 chiffre et une addition par tour. (que j' aurais pu remplacer par des lectures dans des tableaux d'addition ou de multiplication avec gestion de la retenue mais ce n'a pas été mon choix.)
Ainsi le coût en affectations et lecture des tableaux est très grand contrairement à la manipulation humaine sur l'abaque (mais quand même plus rapide sur ordinateur !).
Donc un total d'affectations des cases du tableau de l'ordre (n+1) x3 pour la création du bloc 2 + 2 opérations de calcul + 4n fabrication des chaines de caractères et d'entiers + 100 parcours maximal des n-p colonnes du bloc 3 + création d'une ligne supplémentaire à n éléments.
Chaque tour de boucle sera répété au maximum 100x(n-p) fois en comptant large.

Donc un total pour une majoration estimée à 10nx100x(n-p) = 1000 x n x (n-p) en comptant les affectations et les conversions listes/chaines/entiers et le peu d'opérations de calcul donc un O(nx(n-p))

La grande différence avec l'approche manuelle d'un moine du Xième siècle est sur l'ordre en nx(n-p) plus grand que le px(n-p) qui est due à la régénération de l'ensemble des tableaux de l'abaque. Si j'avais vu la différence au début, j'aurais peut-être géré les tableaux autrement en ne générant pour les calculs machines que le cadre coulissant des (p+1)x3 cases qui concernent les calculs… Ce que j'ai fait plus tard avec l'animation !

5) Le désir de preuve dans le manuscrit de Bernelin.

Page 40 : « *Là-dessus, je vous supplie d'attendre de moi non point ce qu'offre la vérité approfondie, mais ce que notre intelligence est capable de concevoir* ».

Page 41 : « *A quoi il faut aussi ajouter, selon moi, que comme nous l'avons dit, toutes les lignes de l'abaque sont des multiples ou des sous multiples de 10.* ».

Il fait le lien dans le chapitre 1 et le chapitre 2 entre la position des jetons sur l'abaque et la numération décimale de position qu'il devait bien connaître et qu'il essaie d'expliquer.

Page 33 : « *Voilà ce qu'il faut savoir, d'une manière générale, pour tout, car tout nombre (dizaine, centaine, millier ou plus), multiplié par une unité, placera le doigt dans la même colonne que l'unité et l'article dans la seconde* » etc... de proche en proche

D'après le commentaire de bas de page : digitus et articulus servent à noter des réalités numériques à valeur variable, suivant la multiplication et donc leur place dans l'abaque. (c'est le principe de positionnement du jeton déplacé et des calculs faits en bloc 2 à partir de lui). Seule constante : l'articulus vaut dix digiti.

Pages 49-50 : Il fait la vérification sur son exemple 668/6 en remontant l'algorithme à partir de la fin.

« Dans le cas où tu voudrais vérifier si tu as bien divisé par le calcul des valeurs numériques, voici la preuve qui le confirmera : multiplie le diviseur par les dénominations si ce produit ajouté au reste 2 donne un nombre égal aux premiers dividendes tu as bien réussi ; sinon c'est le contraire. De fait , le diviseur, multiplié par la dénomination de l'unité et additionné aussi au reste 2, devient égal au premier 8. Semblablement aussi, multiplié par la dénomination des dizaines, il revient égal à 6. Et si tu fais de même avec la dénomination des centaines, tu as intégralement reconstitué le troisième 6. Vois-tu que de cette multiplication et du reste 2 on aboutit un nombre égal aux premiers dividendes ? tu as donc réussi.

A qui divise de cette façon pour tout division simple, aucun nuage d'erreur ne viendra faire obstacle, sauf quand on propose un nombre tel que diviseur et dividende tombent dans la même colonne. »

Commentaires : On voit dans ces extraits que Bernelin ne se limite pas à des recettes de cuisine (« la division sur abaque pour les nuls ») mais essaie aussi de justifier les calculs en revenant aux nombres écrits en base décimale et en étant le plus pédagogue possible pour convaincre le lecteur de la validité de sa méthode. En ayant jeté un rapide coup d'œil sur les « œuvre complètes de Gerbert », on voit bien que celui-ci , maître de Bernelin, pose les principes du calcul de façon plus générale anticipant la numération de position (introduction du zéro pour l'écriture des nombres).

IV : Une rapide mise en perspective avec l'implémentation actuelle de la division en machine.

L'étude précédente permet de montrer que l'algorithme de la division sur abaque est plus tourné du côté de l'analyse matricielle : positionnement d'un chiffre parmi 10 dans chacune des cases d'un tableau et donc relève en machine de l'affectation bien ciblée de ces dix chiffres en case mémoire.
En effet, peu de calculs sont effectués, pour rappel :
1 complément du diviseur à la puissance de 10 supérieure.
La répétition à chaque tour de boucle d'une multiplication à un chiffre suivi d'une addition (qui toutes les deux peuvent être lues dans un tableau simple avec gestion des retenues). La condition de cette répétition se faisant non pas sur la comparaison des nombres eux-mêmes mais sur la comparaison de la longueur des nombres (ou simplement le positionnement des « digitis » dans le tableau).
L'essentiel des opérations élémentaires me parait donc être des affectations et des lectures mais peu de calculs.

En flânant sur internet pour explorer ce thème et guider ma réflexion, je tombe sur le site http://compoasso.free.fr/primelistweb/page/prime/euclide.php
qui détaille l'implémentation en java de plusieurs algorithmes classiques de division, de celui issu de la définition de la division euclidienne à la division égyptienne en passant par notre division manuelle classique actuelle. L'auteur compare les vitesses d'exécution sur différents jeux de tests.

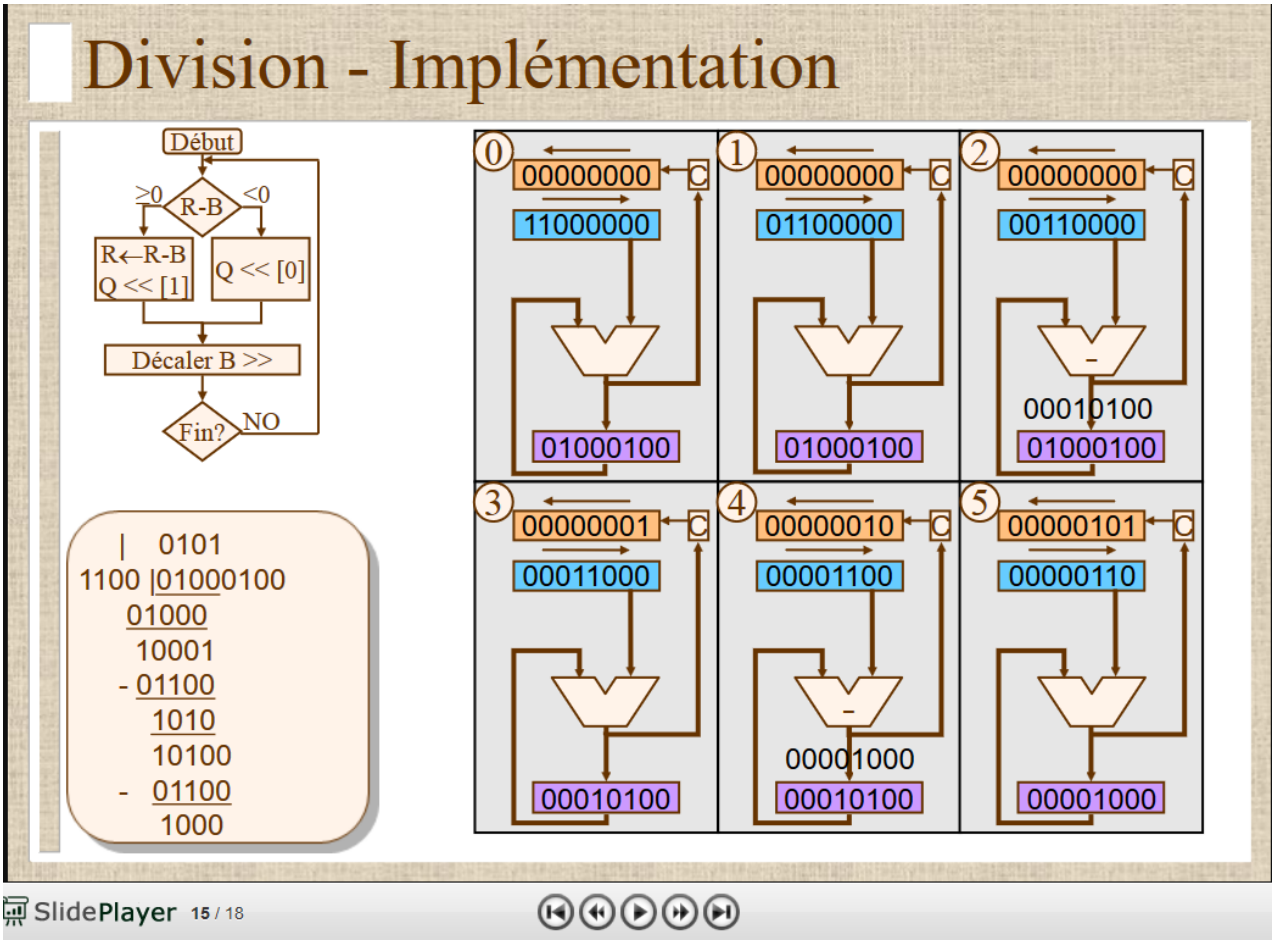
Ceux-ci reposent principalement sur des tests de comparaison de nombres :
-« combien il y a de fois … dans … » pour la division ordinaire
-le « combien de fois » sera lu par test dans une table des multiples de 2 au fur et a mesure pour la division égyptienne.
- L'utilisation de la dichotomie sous forme de choix par nature pour la recherche de bornes du quotient sous forme de puissances de 2 par approximation successives.

Or la méthode de division simple par différence sur abaque présentée par Bernelin ne se base pas sur des tests, on répète le procédé sans se poser de questions : « mécaniquement » ! Inutile de travailler sur les nombres eux-même mais seulement sur les jetons portant un chiffre placés dans des colonnes de l'abaque.

Je poursuis ma réflexion en explorant l'implémentation machine de la division et trouve un diaporama explicatif de Cécile Potier qui me permet de me donner une idée de comment peut être programmée la division en assembleur :
https://slideplayer.fr/slide/1326347/ (voir extrait page suivante)

Là, je retrouve vraiment les décalages et calculs élémentaires (comme sur l'abaque) codés en machine. Ici, l'explication est sur la base de la division en binaire avec le même procédé que notre division décimale ordinaire. Est-ce un exemple d'enseignement pour comprendre le principe ou bien est-ce vraiment l'implémentation machine sur mon ordinateur. Est-ce la même implémentation sur ma calculatrice ?
Précédemment dans le diaporama, l'auteur nous montre une méthode optimisée pour la multiplication, il en existe certainement aussi pour la division. Mais elle nous donne aussi des indices sur l'optimisation en terme de ressources mémoires.

On voit sur cet exemple que la notion de test que j'avais mentionné plus haut dans le langage de haut niveau lié à l'idée de « combien de fois » n'apparait plus au niveau de l'implémentation machine puisqu'on en revient à deux issues (le test n'étant que sur le 0 ou 1 et le calcul se fait à ce niveau à l'intérieur de la boucle). La répétition « automatique » du décalage jusqu'à la fin me met en lumière que le principe de division sur abaque présenté par Bernelin parait plus proche du langage de bas niveau que du langage de haut niveau.
C'est un peu comme si Bernelin faisait de l'assembleur non pas avec la base 2 (avec des 0 et des 1)mais avec la base 10 (avec ses 10 jetons) !
C'est cette dernière remarque qui me donne l'idée de l'animation intégrée à la présentation. En effet, les pages d'explication sont plus tournées du côté « nombre ».



Cette réflexion sur les langages de haut niveau et de bas niveau me poussent à explorer les derniers cours de la formation NSI sur les notions d'automates/langages/compilation. Cela m'a permis de clarifier un peu certaines de mes idées qui ont émergé lors de ce projet et dont je n'avais pas le vocabulaire suffisant à poser dessus pour les expliciter.

L'efficacité de la méthode de la division simple par différence pour effectuer les calculs sur abaque est réelle et très facile à expliquer et à automatiser avec très peu de calculs sur les nombres (voir aucun au-delà de la multiplication à un chiffre par lecture dans une table ou somme chiffre à chiffre par empilement), elle ne demande pas de connaissances préalables. Pourtant Bernelin ne s'arrête pas là, s'il la qualifie de simple c'est qu'il présente dans le chapitre suivant la division « complexe » par différence. Celle-ci repose plus sur l'idée de « nombre » car le diviseur opératoire n'est plus pris arrondi à la puissance de 10 supérieur mais à la valeur par excès à un seul chiffre significatif (pour 668 on prend 700 donc un diviseur opératoire de 7 placé dans la colonne au dessus du début du diviseur). Le nombre d'étapes est de ce fait beaucoup plus réduit mais demande plus de calculs sur les nombres à chaque étape. Même si cette méthode est aussi facilement automatisable, elle me parait plus tirer vers des calculs sur la représentation des nombres et donc s'éloigner du langage de base de la méthode de division simple (décalage /lecture/ empilement). C'est aussi le cas des méthodes de divisions simple ou complexe sans différence qui repose sur le choix « combien de fois » qu'il présente aussi. Je pense que Bernelin met en évidence que le travail mental fait en amont sur le calcul sur les nombres réduit la manipulation des jetons sur l'abaque donc pour un gain de temps de manipulation…

V Conclusion :

Je me suis tenue à la feuille de route que j'avais établie en janvier 2020 après la lecture du manuscrit de Bernelin. De l'intuition des éléments qui constitueraient ma réflexion à la réalisation d'un projet en cohérence, le travail a été long.
Le défrichage et déchiffrage de l'algorithme dans le texte du XIième siècle m'a certainement caché les difficultés que j'allais rencontrer dans l'élaboration de mon programme informatique. En effet, j'ai toujours essayé de rester en équilibre sur le fil séparant le codage de l'algorithme de la division en python et la présentation de son explication pour diffusion (je participe au groupe Irem AHMES travaillant sur l'histoire des mathématiques) donc entre « faire coller le code à la méthode décrite par Bernelin pour réaliser la division» et « donner les résultats intermédiaires en console ou par fenêtres Tkinter pour expliquer la méthode».
En restant sur ce fil, j'ai travaillé sur des tableaux constitués de lignes de chiffres correspondant aux nombres. Ces tableaux simulent l'abaque en bois partagé en colonnes et je suis passée à côté du cœur de la méthode qui est plus tournée vers la manipulation des jetons en tant que chiffres d'une matrice indépendants du nombre formé. Mais d'un autre côté, c'est grâce à la simulation que j'ai compris le cœur de l'algorithme! Peut-être est-ce à cause de mes cheveux blancs et de l'origine de ma discipline première d'enseignement mais je n'avais pas expérimenté ce va et vient entre avancée conceptuelle et simulation. L'enseignement de cas d'école liant exercices de mathématiques et simulations dans les cours de mathématiques n'est pas de cet ordre là. J'ai donc rajouté l'animation qui parait être plus en cohérence avec l'application de la méthode sur abaque présentée par Bernelin.

La mise en perspective des différentes approches de la division (côté « nombres » ou côté position des « chiffres ») me montre bien que l'important auprès des élèves est de bien positionner le problème pour ce qu'il est au niveau des méthodes. L'essentiel est d'expliciter la façon de l'aborder et pour quels objectifs. On voit bien que traiter le problème de la division vu du côté des méthodes dans des langages de haut niveau n'a rien à voir avec l'implémentation machine en langage de bas niveau. Pourtant ce « rien à voir » est à relativiser car les liens entre ces deux niveaux sont pourtant essentiels et utilisent chacun la même notion d'algorithme. En relisant la querelle historique entre abacistes et algoristes (couvrant une période large du 13^{ième} siècle au 18^{ième} siécle) https://ljk.imag.fr/membres/Bernard.Ycart/mel/ar/node17.html , je me dis que les débats sont un peu de cet ordre là. Les abacistes pencheraient plutôt du côté des langages de bas niveau et les algoristes plutôt du côté des langages de haut niveau avec forcément une frontière poreuse qui n'est certainement due qu'à des différences d'approche.
Je pense que Bernelin a été confronté à ce problème dans l'écriture de son manuscrit entre la notion de « nombre » écrit en numération décimale (même si c'est nouveau en occident à l'époque, il est généralement admis que Bernelin comme son maître Gerbert la maitrisaient) et la manipulation des jetons. Peut-être que ça justifie ses remarques du premier chapitre poussant le lecteur à accéder au raisonnement sur les nombres écrits en numération de position au-delà du calcul technique : passage du langage de bas niveau à un langage de haut niveau. Dans les ordinateurs d'aujourd'hui, on utilise plutôt une « redescente » du langage de haut niveau maîtrisée par l'utilisateur vers le langage de bas niveau utilisé en machine.

Entre les maîtres (génies) qui ont su inventer des méthodes soit en créant de nouveaux concepts soit en synthétisant les connaissances disponibles et les disciples qui ont su diffuser les connaissances, les progrès dans la connaissance et la technique ont permis de créer des ordinateurs manipulables par chacun. Alors comment ne pas rendre hommage à ce moine dénommé Bernelin dont on ne connait presque rien de la vie au Xième siècle sauf qu'il préférait faire les vendanges plutôt que de se lancer dans l'écriture d'un manuscrit détaillant le principe de la division sur abaque qui serait recopié dans les monastères pour diffusion dans la papauté, mais aussi à Cécile Potier que je ne connais pas et qui laisse à disposition de tous sur internet son cours sur l'assembleur. A ces deux là, pris en exemple vont s'ajouter les myriades de passeurs de connaissances de génération en génération, j'en prends une infime partie de bas niveau d'enseignement en espérant que les graines ressemées depuis des millénaires ne se perdent pas dans l'oubli.

Dans le manuscrit, Bernelin détaille aussi la méthode de la division complexe et la poursuit au-delà des entiers sur les fractions correspondant aux différents systèmes de mesure utilisés à son époque. Les calculs s'obtiennent à l'aide de la lecture de tables fournies dans le manuscrit. Mais là, ce serait pour un autre projet !

Et que dire du travail sur le calcul dans les différentes bases de Gerbert dont la réputation sulfureuse de connivence avec le diable lui aurait permis de construire une machine préfigurant un ordinateur…

Courte bibliographie :

- Mes cours de la formation NSI à Clermont (2018-2020)
- Libre d'abaque / Bernelin, élève de Gerbert d'Aurillac ; texte latin établi, traduit et annoté par Béatrice Bakliouche [i.e. Bakhouche] ; notes complémentaires de Jean Cassiut [i.e. Cassinet], réalisé d'après le manuscrit du XIe siècle, H 491 de la Bibliothèque de l'Ecole de médecine de Montpellier. Editions des Régionalismes.
- Histoire universelle des chiffres / Georges Ifrah . Edition Robert Laffont

Sitographie (entre autres…):
Pour les pages générales : Wikipédia :
Sur Gerbert : https://fr.wikipedia.org/wiki/Sylvestre_II
Sur l'implémentation de la division en machine : https://fr.wikipedia.org/wiki/Division
Sur le calcul sur abaque : https://fr.wikipedia.org/wiki/Abaque_(calcul)

Cours Assembleur de Cécile Potier sur slide player : https://slideplayer.fr/slide/1326347/

Œuvres de Gerbert, pape sous le nom de Sylvestre II... / précédées de sa biographie, suivies de notes critiques & historiques par Alexandre Olleris, edition 1867 F.Thibaud à Clermont
https://gallica.bnf.fr/ark:/12148/bpt6k408322c/f521.item

sur la querelle entre algoristes et abacistes :
https://ljk.imag.fr/membres/Bernard.Ycart/mel/ar/node17.html
http://www.encyclopedie-universelle.net/abaque-menu.html
et en particulier http://www.encyclopedie-universelle.net/abaque-calcul7-moyen-age-occidental.html

Annexe

Diagramme de classes pour la partie animation des jetons en programmation objet