SÉMINAIRE DE THÉORIE DES NOMBRES DE GRENOBLE

JEAN-RENÉ JOLY

Analyse numérique p-adique des nombres de Bernoulli et des séries *L* de Dirichlet

Séminaire de théorie des nombres de Grenoble, tome 10 (1981-1982), exp. n° 6, p. 1-14

<http://www.numdam.org/item?id=STNG_1981-1982__10__A6_0>

Séminaire de Théorie des Nombres
18 et 25 mars 1982
Grenoble

ANALYSE NUMERIQUE p-ADIQUE DES NOMBRES DE BERNOULLI ET DES SERIES L DE DIRICHLET

par Jean-René JOLY

I. - NOMBRES DE BERNOULLI modulo p^m

La méthode le plus couramment utilisée pour tester la régularité d'un couple (p,k) -c'est-à-dire, essentiellement, pour calculer B_k modulo p- met en œuvre une série de congruences dues à E. Lehmer [8] et dont la plus importante est

$$(1) \qquad S_{k-1} \equiv \tfrac{1}{2} C_{k-1} \frac{B_k}{k} \pmod{p^2}$$

avec par définition

$$(2) \qquad S_{k-1} = \sum_a (p-6a)^{k-1} , \quad C_{k-1} = 6^{k-1}+3^{k-1}+2^{k-1}-1 ;$$

la sommation sur a est étendue à tous les entiers de 1 à $[p/6]$. Ces congruences, ainsi que d'autres relatives aux quotients de Fermat, de Wilson, etc. . sont démontrées dans [8] à partir de la formule des différences finies pour les polynômes $B_k(X)$; cette méthode ne se généralise pas directement à un module p^m , $m \geq 2$. Cependant, la congruence (1) et les autres congruences d'E. Lehmer sont visiblement des formules d'Euler-Maclaurin tronquées à l'ordre 1 ou 2 ; une façon de calculer $B_k \pmod{p^m}$ est donc d'écrire la formule d'Euler-Maclaurin à l'ordre k pour les sommes telles que S_{k-1} figurant dans le membre de gauche des congruences

d'E. Lehmer, et de remarquer que, dans le membre de droite : 1) le reste intégral est nul ; 2) les deux premiers termes sont "en général" congrus à 0 modulo p^{k-1} ; 3) les autres termes sont "en général" de la forme $A_j B_\ell p^j$, avec $\ell = k-j$, $j = 0,1,2,\ldots$ ou $j = 0,2,4,\ldots$, les A_j étant des p-entiers explicites. Par suppression des termes d'ordre $j \geq m$, et en supposant $m < k-1$, on arrive ainsi à des congruences généralisant celles d'E. Lehmer. Bien entendu, chaque congruence fait maintenant intervenir plusieurs nombres de Bernoulli. (Par exemple, avec la somme S_{k-1} introduite en (2), on obtient, pour m pair, la congruence

$$(3) \qquad S_{k-1} \equiv \tfrac{1}{2} C_{k-1} \left(\frac{B_k}{k} + \sum_{j=2}^{m-2} \binom{k-1}{j} 6^{\ell-1} \frac{B_\ell}{\ell} p^j \pmod{p^m} \right) .$$

Pour k fixé, le calcul de $B_k \pmod{p^m}$, se fera donc en calculant plusieurs sommes $S_{\ell-1} \pmod{p^m}$ et en écrivant plusieurs congruences telles que (3) (correspondant aux indices $\ell = k-2, k-4, \ldots$), puis en éliminant les nombres de Bernoulli B_ℓ parasites (correspondant à $\ell = k-2, k-4, \ldots$). Pratiquement, cette élimination se limite d'ailleurs à la résolution d'un système linéaire triangulaire sur $\mathbb{Z}/p^m\mathbb{Z}$, ou, ce qui revient au même, sur $\mathbb{Z}_p$; c'est là une opération facile à programmer et d'exécution rapide, à condition de disposer d'un ordinateur qui sache "faire directement de l'arithmétique sur les entiers p-adiques", ce qu'on supposera dans toute la suite (v. §3).

Les considérations ci-dessus résolvent complètement (en principe) le problème du calcul numérique de $B_k \pmod{p^m}$, ou, si l'on veut, du calcul numérique p-adique approché de B_k ; pour plus amples détails, voir [2a], [2b], et le §3 ci-dessous. Pour des méthodes de calcul essentiellement différentes, voir la thèse de M. Chellali [3].

2. – NOMBRES DE BERNOULLI GENERALISES modulo p^m

La méthode décrite au §1 s'étend, moyennant quelques modifications de détail, au calcul de $B_k(\chi)$ modulo $\mathfrak{P}^m$, où χ est un caractère de Dirichlet à valeurs dans $\mathbb{Q}_p^{alg}$, et où $\mathfrak{P}$ est un idéal premier de $\mathbb{Z}_p[\chi]$ au-dessus du nombre premier rationnel p. Si N désigne le conducteur de χ, on est amené à distinguer deux cas :

a) $N = p$ On a alors $\chi = \omega^h$, ω désignant le caractère de Teichmüller, et on sait (v. [5], [6], [2b]) qu'il existe entre nombres de Bernoulli ordinaires et nombres de Bernoulli généralisés des congruences telles que

$$(4) \qquad \frac{B_k(\omega^h)}{k} \equiv \frac{B_{k+hp^m}}{k+hp^m} \pmod{p^{m+1}}$$

(pour $k+h \not\equiv 0 \pmod{p-1}$), etc. On est donc ramené ici à la situation du §1.

b) $N \neq p$ Quitte éventuellement à remplacer χ par $\chi\omega^0$, on peut supposer $N \equiv 0 \pmod{p}$. Pour simplifier l'exposition, on supposera également $N \not\equiv 0 \pmod{p^2}$, χ pair, k pair ; enfin, si d désigne l'ordre de χ, on supposera que d divise $p-1$, donc que χ est à valeurs dans $\mu_{p-1} \subset \mathbb{Z}_p$. On a alors $\mathbb{Z}_p[\chi] = \mathbb{Z}_p$ et $\mathfrak{P} = p\mathbb{Z}_p$, de sorte qu'on travaille encore dans $\mathbb{Z}_p$. Cela étant, l'idée générale esquissée au §1 s'adapte de la façon suivante

1) on introduit les <u>fonctions</u> de Bernoulli attachées à χ, dépendant de la variable <u>réelle</u> t, et définies <u>ici</u> par

$$(5) \qquad \underline{B}_k(t,\chi) = N^{k-1} \sum_{a=0}^{N-1} \chi(a)\, B_k(\langle \frac{a+t}{N} \rangle)$$

(v. [1a], [1b] et [6]) ;

2) on introduit d'autre part les sommes

$$S_k(\chi) = \sum_{a=0}^{N-1} \chi(a)\, a^k ,$$

3) on applique aux $S_k(\chi)$ la formule d'Euler-Maclaurin "avec caractère"

(v. [1a], [1b]) à l'ordre $k+1$, avec $f(x) = x^k$. On a $B_0(\chi) = 0$, $f^{(k+1)} = 0$: les deux termes intégraux sont nuls, et il vient (avec les bornes de sommation $A = 0$, $B = N$).

$$(6) \qquad \sum_{a=A}^{B} \chi(a)f(a) = \sum_{j=1}^{k+1} \frac{(-1)^j}{j!} \left[\underline{B}_j(t,\chi) f^{(j-1)}(t) \right]_{t=A}^{t=B} \quad ;$$

4) on remarque enfin que $\underline{B}_j(0,\chi) = \underline{B}_j(N,\chi) = B_j(\chi)$ et que $N = pN_1$, $N_1 \not\equiv 0 \pmod p$.

En posant pour simplifier $S_k^0(\chi) = p^{-1} S_k(\chi)$ (ce qui est "en général" un p-entier), en faisant le changement d'indice $j := k-j$, et en remarquant que $B_j(\chi) = 0$ pour j impair, on arrive à l'égalité suivante :

$$(7) \qquad S_k^0(\chi) = \sum_{j=0}^{k-2} \binom{k}{j+1} N_1^{j+1} \frac{B_{k-j}(\chi)}{k-j} p^j \quad .$$

En supposant (comme au §1) m pair $\leq k-1$, en supprimant les termes d'ordre $j \geq m$, et en faisant toujours $\ell = k-j$, on arrive à

$$(8) \qquad S_k^0(\chi) \equiv kN_1 \frac{B_k(\chi)}{k} + \sum_{j=2}^{m-2} \binom{k}{j+1} N_1^{j+1} \frac{B_\ell(\chi)}{\ell} p^j \pmod{p^m} \quad .$$

Cette congruence est analogue à (3) ; les principales différences sont : 1) la "longueur" de la somme dans le membre de gauche (pN_1 termes au lieu de $[p/6]$) ; 2) la forme des coefficients, plus simple ici (en particulier, le "coefficient de Lehmer" C_{k-1} n'existe pas dans ce type de formule). Bien entendu, pour que la méthode soit effectivement utilisable, il faut que N_1 ne soit pas trop grand devant p ; et que le calcul de l'entier p-adique $\chi(a)$ à partir de a soit suffisamment rapide. Mais ces deux contraintes interviendraient sans doute dans toute autre méthode.

3. - CALCULS p-ADIQUES

Les calculs décrits aux §§1-2 ont été testés sur calculatrice programmable HP 41 (pour $m = 4$), puis, dans un premier temps, programmés en BASIC et exécutés sur micro-ordinateur MZ 80 : voir [2a], [2b]. Le BASIC convient au traitement du système linéaire, mais sa pauvreté en ce qui concerne les appels de procédures et les passages de paramètres rend pénible la programmation des subroutines p-adiques. De ce point de vue, le Pascal est évidemment préférable. Une autre solution, moins élégante mais plus rapide à l'exécution, consiste à programmer toutes les opérations p-adiques en Langage-machine et à ne confier au BASIC que les entrées-sorties et la résolution du système linéaire : schématiquement, l'arithmétique p-adique est alors le domaine du Langage-machine (L.M.) et l'algèbre celui du BASIC ; le passage d'un langage à l'autre se faisant par les instructions classiques PEEK/POKE et USR/RET. L'implantation des diverses opérations p-adiques (permettant donc essentiellement d'accélérer le calcul des S_{k-1} et des $S_k(\chi)$) peut être décrite de la façon suivante :

1) p et m sont fixés ; on définit un type scalaire (entier entre 0 et p-1, considéré comme reste modulo p, et mémorisé sur un registre ou sur une case-mémoire), un type hensel (suite de m scalaires, considérée comme suite des m premiers coefficients du développement de Hensel d'un entier p-adique) et un type grand-entier (suite d'entiers entre 0 et 2^8-1, considérés comme chiffres en base $2^8 = 256$ d'un entier positif ; le "2^8" est contingent et tient au fait que la case-mémoire du MZ 80 est d'un octet ; ceci introduit d'ailleurs la limitation $p < 2^8$; à ce propos, v. §5) ;

2) on définit également une méthode générale de rangement en mémoire des scalaires, des hensels et des grands-entiers, rapidement utilisable à la fois par le BASIC et le L.M.

3) on implante enfin un programme BASIC permettant d'entrer les données et en particulier de convertir les grands entiers (au sens ordinaire, écrits en base 10 et considérés comme chaînes de caractères) en grands-entiers

(au sens ci-dessus, c'est-à-dire écrits en base 256).

Cela étant,

4) on implante deux sous-programmes L.M. de multiplication scalaire et de réduction modulo p (avec reste) :

MULTSC : $x,y,z \longmapsto xy + z$;

REDSC : $u \longmapsto z, x$;

x,y,z sont des scalaires (un registre ou une case-mémoire, pratiquement, un octet) ; $xy + z$ et u sont des "doubles-scalaires" (un double-registre ou deux cases-mémoires ; pratiquement, deux octets) ; dans REDSC , z et x sont le quotient et le reste de u modulo p ;

5) on implante ensuite des sous-programmes L.M. d'addition, d'inversion,... "hensélienne" : ADDHE, INVHE,... ; on implante également deux sous-programmes L.M. de multiplication "hensélienne" :

MULTMX : $x , (y_i) \longmapsto (z_i)$

donnant le produit du scalaire x par le hensel (y_i) , et

MULTHE : $(x_i) , (y_i) \longmapsto (z_i)$

donnant le produit du hensel (x_i) par le hensel (y_i) ; naturellement, MULTMX et MULTHE font appel à MULTSC / REDSC ; il y a $m(m+1)/2$ appels dans le second cas contre m seulement dans le premier, ce qui justifie la procédure LEHMER décrite plus loin ;

6) on implante un sous-programme L.M. d'exponentiation "hensélienne"

EXPHE : $(x_i) , k \longmapsto (z_i)$

où k est un grand-entier, où (x_i) et (z_i) sont des hensels, et où (z_i) égale (x_i) à la puissance k ; EXPHE fait naturellement appel à MULTHE et procède par "décalages à droite" de k , selon une technique bien connue (la même remarque s'applique d'ailleurs à MULTSC et REDSC) ;

7) on implante enfin un sous-programme L.M. baptisé LEHMER et dont le rôle est de permettre le calcul rapide et simultané des sommes S_{k-1-j} ou $S_{k-j}(\chi)$ $(j=0,2,4,\ldots)$ intervenant comme "seconds membres" dans les systèmes linéaires introduits implicitement aux §§1-2. LEHMER est une procédure travaillant sur les données p, m et k et "retournant" les sommes S_{k-1-j} ou $S_{k-j}(\chi)$ $(\mathrm{mod}\ p^m)$. En "pidgin-Pascal", LEHMER pourrait être rédigé de la façon suivante (en assimilant pour simplifier le type entier au type scalaire) :

```
procédure lehmer ( k : grand-entier ; var s : tableau [1..m] de
hensel) .

var r,i,a,x : scalaire ; b : hensel ; k0 : grand-entier ;

début
pour i := 2 à m incrément 2 faire s(i) := 0 ;
k0 := k-m-1 ; r := int(p/6) ;
pour a := 1 à r faire

   début
   x := p-6*a ; b := exphe(x,k0) ;
   pour i := 1 à m faire

      début
      b := multmx(x,b) ;
      si i mod 2 = 0 alors s[i] := addhe(s[i],b)
      fin
   fin
fin ;
```

En fin d'exécution, on a évidemment $s[i] = S_{\ell-1}$ $(\mathrm{mod}\ p^m)$ avec toujours $\ell = k-j$ (et $j = m-i$) (voir §1). Naturellement, il faut également prévoir de la même manière le calcul des $C_{\ell-1}$ (dans la situation du §1). Dans la situation du §2, LEHMER est à modifier de façon évidente :

remplacer "$k_0 := k-m-1$" par "$k_0 := k-m$" ;
remplacer la double instruction "x := p-6*a ; b := exphe(x,k_0) ; "
par "x := exphe(a,k_0) ; b := multhe(χ[a],x) ; "

naturellement, il faut aussi dans ce cas avoir prévu un sous-programme (ou un fichier, ou un tableau) fournissant les valeurs du caractère χ lorsque LEHMER les appelle.

L'intérêt de LEHMER est que, pour une valeur de $x=p-6a$ (ou de a), on appelle une seule fois EXPHE, et que les puissances $x^{\ell-1}$ ou a^{ℓ} sont ensuite calculées à partir de là par MULTMX, du fait que x ou a est un scalaire : on court-circuite donc (sauf dans EXPHE) la multiplication "lourde" MULTHE. Au total, il n'est donc guère plus long, avec LEHMER, de calculer la collection des sommes $S_{k-1}, S_{k-3},\ldots,S_{k-m+1}$ que de calculer une seule d'entre elles (et de même pour les $S_{\ell}(\chi)$). Le temps de résolution des sytèmes linéaires étant quasi-négligeable, on peut donc dire que le temps de calcul de $B_k \pmod{p^m}$ ou de $B_k(\chi) \pmod{p^m}$ est sensiblement égal à celui de $S_{k-1} \pmod{p^m}$ ou de $S_k^0(\chi) \pmod{p^m}$.

4. - EXEMPLES EXPLICITES

On s'est limité aux §§1 à 3 à des descriptions en termes très généraux. Voici maintenant quelques indications donnant une idée précise des ordres de grandeur (pour p , k , m) , des temps de calcul et des tailles-mémoire intervenant effectivement. (Rappelons que les calculs ont été effectués sur un MZ 80/2 Mhz/ 48 Ko ; les programmes p-adiques en L.M. ont donc été rédigés en Assembleur Z 80).

Encombrements-mémoire (des opérations p-adiques) :

MULTSC : 13 octets ; REDSC : 21 octets ; MULTMX : 42 octets ; MULTHE : 80 octets ; EXPHE : 98 octets ; LEHMER : 151 octets.

En fait, on peut affirmer que l'arithmétique p-adique tout entière (y compris inversion, Teichmüller, etc.) occuperait environ 1,5 kilo-octet de mémoire, ce qui est extrêmement faible.

Temps d'exécution :

MULTSC et REDSC : environ 165 microsecondes.

Les temps d'exécution des autres opérations dépendent évidemment des valeurs de m et de k. Prenons deux exemples, correspondant (comme dans [2a], [2b]) à l'étude du couple irrégulier $(37,32)$:

Exemple 1. - On fait $p = 37$, $m = 32$. (On a donc $p^m = 37^{32} > 10^{50}$: on travaille virtuellement avec des entiers de 50 chiffres). On prend d'autre part $k = 1\,853\,540 = 32 + 36.37^3$. Voici les temps d'exécution :

EXPHE : environ 3 secondes (pour le calcul de chacune des six exponentielles $(p-6a)^{k_0} \pmod{p^m}$; LEHMER : environ 22 secondes.

Le temps de calcul de $S_{k-1} \pmod{p^m}$ est donc de 18 secondes environ, et le temps de calcul simultané des seize sommes $S_{k-1}, S_{k-3}, \ldots, S_{k-31} \pmod{p^m}$ est seulement de quatre secondes supérieur ; en d'autres termes : la première somme coûte 18 secondes, chaque somme suivante coûte 1/4 de seconde ! Ceci illustre bien la remarque finale du § 3.

Exemple 2. - On fait toujours $p = 37$. L'exposé [2a] (Annexe 3, pp. 17-18) donne une table de valeurs k_m congrues à 32 (mod 36) et solutions de la congruence $B_k \equiv 0 \pmod{p^m}$, avec $m = 2,3,4,6,8$. L'utilisation des deux valeurs

$$k'_6 = 325\,656\,968 \quad , \quad k''_6 = 2\,822\,039\,420$$

permet (voir [2b], §5) de calculer k_{12}, plus petit indice k congru à 32 (mod 36) tel que

$$(*) \qquad B_k \equiv 0 \pmod{37^{12}} \ ;$$

on trouve

$$k_{12} = 4\,841\,789\,050\,865\,438\,960 \ .$$

La vérification (que k_{12} satisfait effectivement à $(*)$) peut se faire avec les programmes BASIC décrits dans [2a], [2b] ; le temps d'exécution (sous interpréteur) est de 21 minutes, soit :

16 minutes pour le calcul des $S_{\ell-1}$ et des $C_{\ell-1}$, c'est-à-dire pour l'équivalent de LEHMER ;

1 minute pour les entrées-sorties ;

4 minutes pour le système linéaire ;

la même vérification en langage-machine pour les calculs p-adiques et en BASIC compilé pour les autres opérations prend seulement une minute et demie, soit :

9 secondes pour LEHMER ;

20 secondes pour les entrées-sorties ;

1 minute environ pour le système linéaire.

Ces deux exemples montrent que l'emploi du langage-machine pour l'arithmétique procure un important gain de place en mémoire (facteur 3 à 6), et surtout un énorme gain de temps (facteur 20 à 150).

5. - QUELQUES REMARQUES

REMARQUE SUR LE §2. - Les congruences du type (4) appartiennent à l'arithmétique des séries $L_p(s,\chi)$ (v. [5], [6], [2b]) : les considérations des §§ 1 à 4 permettent donc effectivement l'"Analyse numérique p-adique des séries L de Dirichlet", du moins sous la condition restrictive que l'ordre d de χ divise p-1. Cette restriction peut être supprimée à condition de résoudre deux problèmes "auxiliaires" :

- détermination d'une base de $\mathbb{Z}_p[\chi] = \mathbb{Z}_p[\mu_d]$ sur $\mathbb{Z}_p$: c'est un problème classique de cyclotomie p-adique ;
- mise en œuvre effective du caractère χ : tout dépend évidemment de la manière dont χ est défini (ou donné).

Bornons-nous ici à signaler que si ces deux problèmes sont résolus, la méthode des §§ 1-2 s'applique sans modification, à condition évidemment de travailler "composante par composante" :

si $\mathbb{Z}_p[\chi] = \oplus_i \mathbb{Z}_p \cdot e_i$, et si $\sum_i \chi_i \cdot e_i$ (les χ_i étant donc des fonctions arithmétiques N-périodiques à valeurs dans $\mathbb{Z}_p$), on introduit les $S_k^0(\chi_i)$, on leur applique la formule d'Euler-Maclaurin périodique (v. [1a], [1b]), on en déduit (par la méthode des §§ 1 à 3) les $B_k(\chi_i)$, et on conclut par

$$B_k(\chi) = \sum_i B_k(\chi_i) \cdot e_i \ .$$

REMARQUE SUR LE § 3. - L'implantation des opérations p-adiques en L.M. telle qu'on l'a décrite au § 3 est essentiellement basée sur l'identification

type entier = type scalaire (= entier modulo p)
= simple-registre (du microprocesseur Z 80)
= octet-mémoire (de la mémoire vive) ;

cette identification est naturelle, puisque le Z80 est en principe un microprocesseur 8 bits. Toutefois, grâce à la possibilité de coupler les six registres B , C , D , E , H , L du Z 80 en trois doubles-registres BC , DE , HL (v. [10a] , [10b]), et plus généralement grâce à la grande variété des introductions d'adressage du Z 80, il est assez facile de travailler en 16 bits, en accordant donc un double-registre ou deux octets-mémoire au type scalaire. Cela permet de travailler dans la zone $2^8 < p < 2^{16}$. Si on reste dans la zone $p < 2^8$, cela permet d'autre part de redéfinir le type scalaire comme entier de 0 à p^2-1 , considéré comme reste modulo p^2, et par conséquent de "compacter" le type hensel et de tirer parti -pour les nombres de Bernoulli ordinaires- du fait que la formule d'Euler-Maclaurin donne un développement par puissances paires de p : on exploite ainsi en L.M. une particularité utilisée en BASIC dans [2a] , [2b] .

Il est intéressant de noter que (pour une même valeur de m, supposé pair) les programmations L. M. 8 bits et 16 bits donnent à peu près la même vitesse d'exécution : par exemple, MULTSC prend (en 16 bits) environ 700 microsecondes, ce qui est voisin de $4 \times 165 = 660$ microsecondes (en 8 bits) (v. § 4 ; bien entendu, il faut introduire un facteur $4 = 2^2$ correspondant au "compactage" des hensel).

BIBLIOGRAPHIE

[1a] JOLY J.R., Formules sommatoires périodiques et applications arithmétiques, Sémin. Th. Nombres, Grenoble, 1976-1977.

[1b] JOLY J.R., Suites périodiques, linéarité, et inégalité de Polya, Bull. Sci. Math., 102, 1978, pp. 3-13.

[2a] JOLY J.R., Calcul des nombres de Bernoulli modulo p^m et application à l'étude des nombres premiers irréguliers, Sém. Th. Nombres, Grenoble, 1980-1981.

[2b] JOLY J.R., Compléments à l'exposé précédent ([2a]) : même titre, manuscrit ronéotypé, Grenoble, Laboratoire de Mathématiques Pures, Novembre 1981.

[2c] JOLY J.R., Calcul des nombres de Bernoulli modulo p^m, Sém. Delange-Pisot-Poitou, Paris, 1981-1982.

[3] CHELLALI M., Nombres de Bernoulli modulo p^m et nombres premiers irréguliers, Doctorat de Spécialité, Grenoble, 1982.

[4a] WELLS JOHNSON, Irregular prime divisors of the Bernoulli numbers, Math. Comp., 28, 1974, pp. 653-657.

[4b] WELLS JOHNSON, Irregular primes and cyclotomic invariants, Math. Comp., 29, 1975, pp. 113-120.

[5] KOBLITZ N., p-adic Numbers, p-adic Analysis and Zeta-Functions, Springer Verlag.

[6] LANG S., Cyclotomic Fields I, II, Springer Verlag.

[7] LEHMER D.H., LEHMER E., VANDIVER H.S., An application of high-speed computers to Fermat's last theorem, Proc. Nat. Acad. Sci. U.S.A., 40, 1954, pp. 25-33.

[8] LEHMER E., On congruences involving Bernoulli numbers and the quotients of Fermat and Wilson, Ann. of Math. 39, 1938, pp. 350-360.

[9a] WAGSTAFF S.S., Zeroes of p-adic L-functions, Math. Comp. 29, 1975, pp. 1138-1143.

[9b] WAGSTAFF S.S., The irregular primes to 125 000 , Math. Comp. 32, 1978, pp. 583-591.

[10a] PINAUD A., Programmer en Assembleur, Editions du P.S.I.

[10b] ZAKS R., Programmation du Z80, Sybex.
