



UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T

G.728 – Annexe G

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

(11/94)

**ASPECTS GÉNÉRAUX DES SYSTÈMES
DE TRANSMISSION NUMÉRIQUES**

**CODAGE DE LA PAROLE À 16 kbit/s EN
UTILISANT LA PRÉDICTION LINÉAIRE À
FAIBLE DÉLAI AVEC EXCITATION PAR CODE**

**ANNEXE G: SPÉCIFICATION
D'UN DISPOSITIF À VIRGULE FIXE
FONCTIONNANT À 16 kbit/s**

Recommandation UIT-T G.728 – Annexe G

(Antérieurement «Recommandation du CCITT»)

AVANT-PROPOS

L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'Union internationale des télécommunications (UIT). Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'études à traiter par les Commissions d'études de l'UIT-T lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution n° 1 de la CMNT (Helsinki, 1^{er}-12 mars 1993).

La Recommandation UIT-T G.728 – Annexe G, que l'on doit à la Commission d'études 15 (1993-1996) de l'UIT-T, a été approuvée le 1^{er} novembre 1994 selon la procédure définie dans la Résolution n° 1 de la CMNT.

NOTE

Dans la présente Recommandation, l'expression «Administration» est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue de télécommunications.

© UIT 1995

Droits de reproduction réservés. Aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'UIT.

TABLE DES MATIÈRES

Page

Annexe G	– Spécification d'un dispositif à virgule fixe fonctionnant à 16 kbit/s	1
G.1	Introduction	1
G.1.1	Principe de base	1
G.1.2	Représentation numérique.....	2
G.1.3	Opérations arithmétiques	3
G.2	Modifications de l'algorithme	8
G.2.1	Modifications de l'adaptateur en boucle du gain vectoriel (bloc 20)	8
G.2.2	Modifications des modules de la récurrence de Levinson-Durbin	12
G.3	Pseudo-code pour les autres modules de la Recommandation G.728.....	18
G.3.1	Bloc 4 – Pseudo-code pour le filtre de pondération	20
G.3.2	Blockzir – Pseudo-code pour les filtres de synthèse et de pondération perceptive pendant le calcul de la réponse à entrée nulle	21
G.3.3	Blocs 9 et 10 – Pseudo-code pour la mise à jour de la mémoire des filtres de synthèse et de pondération perceptive	23
G.3.4	Bloc 11 – Calcul du vecteur quantifié cible	26
G.3.5	Bloc 12 – Calcul du vecteur de réponse impulsionnelle	26
G.3.6	Bloc 13 – Convolution à inversion temporelle	27
G.3.7	Bloc 14 – Convolution des vecteurs code et calcul de l'énergie	27
G.3.8	Bloc 16 – Normalisation du vecteur cible VQ	28
G.3.9	Bloc 17 – Calculateur d'erreur sur la recherche de VQ et sélecteur du meilleur index du répertoire d'excitation.....	29
G.3.10	Bloc 19 – Répertoire des vecteurs d'excitation quantifiés et bloc 21 – Module de normalisation du gain.....	30
G.3.11	Bloc 32 – Filtre de synthèse du décodeur	31
G.3.12	Bloc 36 – Pseudo-code pour le module de fenêtrage hybride.....	33
G.3.13	Bloc 38 – Calculateur des coefficients du filtre de pondération	35
G.3.14	Bloc 43 – Module de fenêtrage hybride.....	36
G.3.15	Bloc 45 – Module d'extension de la largeur de bande	38
G.3.16	Bloc 46 – Prédicteur linéaire de gain logarithmique.....	39
G.3.17	Bloc 49 – Module de fenêtrage hybride pour le filtre de synthèse	41
G.3.18	HWMCORE – Noyau du module de fenêtrage hybride	43
G.3.19	Bloc 51 – Module d'extension de la largeur de bande	47
G.3.20	Blocs 71 et 72 – Postfiltres à long terme et à court terme.....	48
G.3.21	Blocs 73 et 74 – Calculateurs de somme de valeurs absolues.....	49
G.3.22	Bloc 75 – Calculateur de facteur de normalisation	50
G.3.23	Bloc 76 – Filtre passe-bas du premier ordre, et bloc 77 – Module de normalisation du gain de sortie	50
G.3.24	Bloc 81 – Filtre LPC inverse du 10e ordre	51
G.3.25	Bloc 82 – Module d'extraction de la période fondamentale (tonie)	51
G.3.26	Bloc 83 – Calculateur des coefficients de prédicteur de tonie	54
G.3.27	Bloc 84 – Calculateur des coefficients du postfiltre à long terme.....	55
G.3.28	Bloc 85 – Calculateur des coefficients du postfiltre à long terme.....	56
G.4	Représentations des variables de traitement interne LD-CELP	57
G.5	Tables de gain logarithmique pour les vecteurs des répertoires de gain et de forme codés.....	60
G.6	Valeurs entières des tables relatives au répertoire de gain codé	62
G.7	Pseudo-codes pour le programme principal du codeur et du décodeur	62

CODAGE DE LA PAROLE À 16 kbit/s EN UTILISANT LA PRÉDICTION LINÉAIRE À FAIBLE DÉLAI AVEC EXCITATION PAR CODE

(Genève, 1992)

Annexe G

Spécification d'un dispositif à virgule fixe fonctionnant à 16 kbit/s

(Genève, 1994)

(Cette annexe fait partie intégrante de la Recommandation)

G.1 Introduction

La présente annexe a pour objet d'expliquer, de façon suffisamment détaillée, comment la Recommandation UIT-T G.728 relative au codage LD-CELP à 16 kbit/s peut être mise en œuvre sur un dispositif arithmétique du type à virgule fixe. Une telle mise en œuvre devrait permettre l'interfonctionnement intégral avec la version à virgule flottante de la Recommandation G.728 et devrait donner un signal de sortie de qualité équivalente, qu'il s'agisse d'un signal de parole ou d'un signal de données transmis dans la bande. L'expression «arithmétique à virgule fixe» signifie: «mot de 16 bits». La plupart des dispositifs à 16 bits peuvent fonctionner également avec des mots ayant une taille différente. Par exemple, le produit de deux mots de 16 bits est un mot de 32 bits. En conséquence, le registre de produit d'un tel dispositif a normalement une capacité de 32 bits. L'accumulateur stocke la somme des produits; il doit donc aussi avoir une capacité minimale de 32 bits. Ainsi, bien que l'on décrive ici une «forme de mise en œuvre à 16 bits», certaines variables d'état internes ont une précision différente de 16 bits.

On trouvera dans ce document une description complète et exacte sur les bits, de toutes les opérations nécessaires à la mise en œuvre de la Recommandation G.728 sur un processeur de signaux numériques à virgule fixe fonctionnant à 16 bits, comportant un registre de produits à 32 bits et au moins deux accumulateurs à 32 bits (ou plus). Dans de nombreux cas traités dans ce document, on trouvera la description d'autres méthodes qui permettent d'effectuer les opérations de façon telle que l'on obtient exactement le même résultat. En pareils cas, on peut avoir recours à ces autres méthodes. Cependant, il ne faut pas effectuer cette substitution si on n'obtient pas exactement le même résultat pour tous les signaux d'entrée possibles. Comme le nombre des méthodes de substitution possibles est extrêmement grand, on n'a pas cherché ici à spécifier la grande majorité de ces méthodes.

La présente annexe se compose de sept paragraphes. Le premier paragraphe est une introduction, qui renferme des renseignements complémentaires sur le traitement des signaux avec virgule fixe, ainsi que les conventions utilisées dans la présente annexe. Le deuxième paragraphe renseigne sur les modifications apportées aux algorithmes pour obtenir spécifiquement la version G.728 avec virgule fixe. Le troisième paragraphe spécifie le pseudo-code à virgule fixe applicable aux autres modules du codeur. Le quatrième paragraphe donne une récapitulation générale des représentations des variables d'état pour le codeur à virgule fixe. Les derniers paragraphes contiennent des tableaux relatifs à l'adaptateur en boucle du gain vectoriel.

G.1.1 Principe de base

La présente annexe est une annexe à la Recommandation UIT-T G.728. Il est donc inutile de répéter ici tous les renseignements et les développements contenus dans cette Recommandation. Certaines de ces informations seront rappelées, dans la mesure où ces rappels pourront être utiles. En particulier, la Recommandation donne des renseignements complets sur le calcul avec virgule flottante. On ne trouvera pas ici de détails de calcul dans les cas où le seul changement est le remplacement d'opérations arithmétiques avec virgule flottante par des opérations avec virgule fixe.

Les différences les plus importantes entre le codeur à virgule flottante et le codeur à virgule fixe sont les suivantes:

- 1) introduction de différents types d'opérations arithmétiques et de précisions pour les variables d'état;
- 2) application d'une méthode différente, mais équivalente du point de vue mathématique, pour l'adaptation en boucle du gain vectoriel; et
- 3) introduction d'une précision variable pour le calcul des coefficients du prédicteur dans la récurrence de Levinson-Durbin.

Dans la suite du premier paragraphe de cette annexe, on trouvera des indications sur les différentes représentations numériques et l'arithmétique à virgule fixe. Le deuxième paragraphe renseigne sur les deux grandes modifications algorithmiques signalées plus haut, à savoir l'adaptation en boucle du gain vectoriel et la récurrence de Levinson-Durbin. Le troisième paragraphe spécifie le pseudo-code pour le module de fenêtrage hybride (bloc 49 dans la Recommandation G.728). L'algorithme correspondant à ce module est inchangé, mais la mise en œuvre est rendue plus complexe du fait de l'utilisation de l'arithmétique avec virgule fixe. Le pseudo-code en question est un bon exemple des types de modifications qui doivent être apportées dans tous les autres modules du codeur. Dans le quatrième paragraphe, on trouvera un tableau correspondant au Tableau 2/G.728 qui donne la représentation numérique de toutes les variables d'état utilisées dans le codeur et le décodeur.

Aux fins de cohérence dans la présente annexe, toutes les représentations supposent l'utilisation systématique de l'arithmétique fondée sur les compléments à 2. Pour la réalisation du codeur, on peut avoir recours à d'autres représentations capables de donner des résultats mathématiquement équivalents.

G.1.2 Représentation numérique

L'unité de base d'une mise en œuvre avec virgule fixe et 16 bits est le mot de 16 bits. Pour représenter des nombres entiers purs, l'intervalle correspondant va de -32768 à $+32767$. Le nombre 1 est représenté par 0000000000000001 et le nombre -32768 par 1000000000000000. Dans cette représentation, le bit le plus à droite est le bit de plus faible poids (LSB) et le bit le plus à gauche est le bit de plus fort poids (MSB). Dans l'arithmétique fondée sur les compléments à 2, si $MSB = 0$, le nombre est positif; si $MSB = 1$, le nombre est négatif. On peut numéroter les bits de 0 à 15, le bit 0 étant le LSB et le bit 15 étant le MSB.

Pour représenter des nombres ayant une partie décimale, il faut placer une virgule entre deux des bits. Par exemple, pour représenter des nombres compris entre $-1,0$ et $+1,0$, on placera la virgule entre les bits 14 et 15. Ce format particulier est appelé Q15, parce qu'on a 15 bits à la droite de la virgule. Dans le format Qn, il y a n bits à droite de la virgule. Des données correspondantes à des nombres entiers purs seraient représentées par le format Q0.

Pour certaines données, on a besoin d'une représentation meilleure que celle fournie par un mot à 16 bits. Pour la prise en compte de ces données, on définit un format en double précision, caractérisé par la présence de 32 bits d'information. Les mots à 16 bits sont capables de représenter des données avec une précision de 2^{15} ; en revanche, les registres à 32 bits, par exemple le registre de produit ou l'accumulateur de la plupart des puces DSP (traitement numérique des signaux) (*digital signal processing*) du commerce, donnent une précision de 2^{31} . On dit qu'il s'agit de mots à double précision. Ici encore, il faut placer une virgule pour indiquer la gamme dynamique du mot.

Certaines données ont une gamme trop grande pour pouvoir être représentées par un format fixe à 16 bits, quel qu'il soit. Une précision avec 16 bits est peut-être suffisante, mais la mise à l'échelle de la valeur doit être dynamique. Ces données peuvent être représentées avec une virgule flottante en simple précision. Cela signifie que les données sont représentées par deux mots. Le premier mot de 16 bits contient un nombre dont l'amplitude est comprise entre 16384 et 32767. Ce nombre est la mantisse de la valeur; on dit que sa valeur est représentée dans un format normalisé, du fait de la plage occupée par son amplitude. Si la valeur est positive, le bit 14 de la mantisse est un bit 1. Le second mot contient le nombre de décalages à gauche (NLS) (*number of left shifts*) utilisés pour mettre la valeur dans le format normalisé. Le second mot spécifie par conséquent le format Q de la mantisse. Si ce format est utilisé pour une valeur unique, on dit qu'on a affaire à une virgule flottante scalaire.

Il est possible aussi de représenter une table de n valeurs au moyen de n + 1 mots, en utilisant la virgule flottante de bloc. Dans ce format, la plus grande amplitude de la table serait représentée exactement de la même manière que ce qui vient d'être décrit pour la virgule flottante scalaire. Toutes les autres valeurs de la table se partageraient le même NLS. Leurs mantisses ne seraient pas forcément en format normalisé. Il existe une extension de cette représentation, la virgule flottante de bloc segmenté. Dans ce cas, une table de mn valeurs est représentée par m(n + 1) mots. La table est divisée en m sous-tables de taille n et chaque sous-table est représentée, en virgule flottante de bloc, par n mots qui représentent les amplitudes et par 1 mot qui représente le nombre NLS.

L'autre type de représentation utilisé est la virgule flottante à double précision. On utilise ici des nombres entiers en double précision pour les mantisses et un mot en simple précision pour représenter le nombre NLS. Pour récapituler, les différents types de représentation utilisés sont les suivants: le format à virgule fixe en simple précision, le format à virgule fixe en double précision pour les accumulateurs et le registre de produit, le format à virgule flottante scalaire en simple précision, enfin les formats à virgule flottante de bloc en simple précision et double précision.

G.1.3 Opérations arithmétiques

Lorsqu'on multiplie deux mots de 16 bits, le résultat est un mot de 32 bits. C'est la raison pour laquelle les registres de produit sont généralement des modules à double précision. Comme il est possible d'associer ces registres à des accumulateurs, ces derniers doivent aussi avoir une capacité d'au moins 32 bits. Dans un calcul du type somme de produits – par exemple, convolution ou filtrage FIR (réponse impulsionnelle finie) (*finite impulse response*) – l'accumulateur est susceptible de déborder. Ce problème du débordement est traité différemment dans le cas des puces DSP disponibles sur le marché.

Dans le filtrage IIR (réponse impulsionnelle infinie) (*infinite impulse response*) la somme des produits, ou le résultat des opérations multiplication-accumulation, devient partie intégrante de la mémoire du filtre et est réutilisé(e) lors de l'opération de filtrage suivante. Plus précisément, le mot supérieur (16 bits) de la sortie servira d'entrée pour le multiplicateur. On appelle bouclage (wrap around) un débordement qui a pour effet de transformer une grande valeur positive en une grande valeur négative, ou inversement, le résultat étant une grande différence à la sortie du filtre. Pour se prémunir contre ce phénomène, on a recours à l'arithmétique en mode saturation pour tous les filtres IIR et dans tous les autres cas où une somme de produits sera utilisée ultérieurement comme entrée d'un multiplicateur. Le principe du mode saturation est le suivant: si le mot supérieur prend une valeur plus grande que 32767 ou plus petite que -32768, il sera ramené par écrêtage à l'une de ces deux valeurs, selon le cas, afin d'empêcher le bouclage.

G.1.3.1 Décalage et arrondi

Dans l'étude des opérations arithmétiques, on commencera par examiner les processus de décalage et d'arrondi. Si l'on multiplie une valeur au format Q_n par une valeur au format Q_m , le résultat obtenu dans le registre de produit sera au format de double précision $Q_{(n+m)}$. Si ce résultat doit ensuite être mémorisé ou additionné avec une précision différente, il faudra le décaler et/ou l'arrondir à la précision voulue.

On peut appliquer deux types de décalages: les décalages à gauche et les décalages à droite. Dans les puces DSP du commerce, il est possible en général d'effectuer les décalages dans l'accumulateur. De même, il est possible en général de décaler le résultat dans un registre de produit avant de l'additionner à l'accumulateur ou de le stocker dans l'accumulateur. Comme leur nom l'indique, un décalage à gauche consiste à déplacer les bits vers la gauche et un décalage à droite consiste à les déplacer vers la droite. Si l'on décale une valeur de k bits vers la droite, on perd les k bits de moindre poids de l'ancienne valeur. Si l'on décale une valeur vers la gauche, il faut vérifier la présence de débordements éventuels. On utilise l'expression suivante pour indiquer un décalage vers la droite de k bits pour une variable TMP:

$$\text{TMP} = \text{TMP} \gg k$$

et l'expression suivante pour un décalage vers la gauche de k bits:

$$\text{TMP} = \text{TMP} \ll k$$

Dans certains cas, k est une variable et peut même prendre des valeurs négatives. En pareil cas, pour k négatif, un décalage à gauche de k bits se définit comme un décalage à droite de $-k$ bits. De la même façon, un décalage à droite de k bits, pour k négatif, équivaut à un décalage à gauche de $-k$ bits. Dans les circonstances où k peut être négatif, le pseudo-code inclut un test pour vérifier que cette possibilité existe, suivi d'un décalage inverse de $-k$ bits si k est négatif. En dépit de cette définition mathématique des décalages négatifs, ceux-ci ne peuvent pas être mis en œuvre dans la plupart des dispositifs et dans certains langages informatiques.

Il convient de signaler une anomalie spécifique des décalages à droite dans l'arithmétique avec compléments à 2. Supposons que la valeur à décaler vers la droite soit 3 et que le décalage soit de 1 bit. Dans le format à 16 bits, la valeur 3 est représentée par 0000000000000011. Si l'on applique un décalage à droite d'un bit, on obtient: 0000000000000001 = 1. Si la valeur à décaler vers la droite est -3, on obtient la représentation 1111111111111101. Après un décalage à droite, le résultat est 111111111111110 = -2. La première chose à noter, pour le décalage à droite, est l'extension du bit de signe. L'anomalie est la non-concordance des amplitudes des réponses dans ces deux exemples. Il y aurait concordance si on avait utilisé une représentation avec signe et amplitude. Les personnes responsables de la mise en œuvre devront garder cette différence présente à l'esprit.

La simulation du codeur a permis de découvrir une autre différence, plus subtile, qui dépend du compilateur. Il est possible qu'une instruction soit générée dans l'algorithme pour appliquer à un mot un décalage à droite dont l'amplitude est supérieure à la longueur de ce mot. Par exemple, un mot de 16 bits pourrait être décalé de 18 bits. Si cette opération était effectuée sous la forme de 18 décalages à droite individuels de 1 bit chacun, le résultat serait 0 ou -1 , selon le signe des données initiales. Cependant, on a découvert que certains compilateurs considèrent un décalage de 18 bits comme une instruction non réglementaire et donnent des résultats inintelligibles. Il incombe aux responsables de la mise en œuvre de vérifier de quelle manière ce cas serait traité par le matériel et le compilateur qu'ils envisagent d'utiliser.

On appelle arrondi l'opération qui consiste à passer de la double précision à la simple précision dans l'accumulateur. En règle générale, cette opération intervient immédiatement avant la mémorisation de la valeur sous la forme d'un mot de 16 bits. Un accumulateur se compose d'un mot supérieur et d'un mot inférieur (avec, éventuellement, les bits additionnels à la gauche du mot supérieur). D'ordinaire, le mot supérieur ou le mot inférieur peut être stocké en mémoire, ou encore les deux mots peuvent être stockés en réponse à deux instructions successives. Si l'on considère que l'accumulateur possède une virgule placée entre le mot supérieur et le mot inférieur, l'arrondi est l'opération qui consiste à convertir l'accumulateur à la valeur entière la plus proche de la valeur non entière stockée dans les deux mots initiaux. Pour les nombres représentant des compléments à 2, la convention la plus couramment utilisée consiste à tester le bit MSB dans le mot inférieur. Si ce bit est 1, on ajoute 1 à la valeur du mot supérieur, après quoi on met le mot inférieur tout en zéros. Par exemple, si la valeur présente dans l'accumulateur est 1,5, le mot supérieur est donné par 0000000000000001 et le mot inférieur par 1000000000000000. Comme le bit MSB du mot inférieur est 1, on ajoutera 1 au mot supérieur et on mettra le mot inférieur tout en zéros. Le résultat est 0000000000000010 pour le mot supérieur, c'est-à-dire 2. Si la valeur présente dans l'accumulateur est $-1,5$, le mot supérieur est donné par 1111111111111110 et le mot inférieur par 1000000000000000. Comme le bit MSB du mot inférieur est 1, on ajoute 1 au mot supérieur et on met le mot inférieur tout en zéros. Le résultat est 1111111111111111 = -1 . Cela est analogue à l'anomalie des décalages à droite.

Lorsqu'on applique la fonction d'arrondi, il ne faut pas perdre de vue la possibilité de débordement. Supposons par exemple que la valeur du mot supérieur est 0111111111111111 (= 32767) et que le mot inférieur contient un 1 dans le bit MSB; dans ce cas, l'application de la convention usuelle entraîne un débordement. Selon les caractéristiques du processeur, le mot de sortie pourrait prendre la valeur 1000000000000000, ce qui représente -32768 . En l'occurrence, la convention usuelle n'est pas appliquée. Au lieu de cela, la valeur est saturée, afin d'éviter l'obtention d'une valeur non représentable.

Dans les pseudo-codes pris comme exemples, la fonction d'arrondi est symbolisée par RND (.).

Pseudo-code pour VSCALE

Il y a lieu d'introduire ici un nouveau module de pseudo-code qui effectue la normalisation vectorielle dans la représentation avec virgule flottante de bloc. Ce module est appelé VSCALE. Il a pour fonction de normaliser un vecteur de telle manière que la plus grande amplitude de ses éléments soit justifiée à gauche comme on le désire, c'est-à-dire représentée dans un format normalisé. Le module peut être utilisé pour des vecteurs dont on sait que le premier élément est le plus grand ou pour des vecteurs dont on ignore la position du plus grand élément. Les entrées appliquées à VSCALE sont les suivantes: IN, vecteur d'entrée à normaliser; LEN, longueur du vecteur d'entrée; SLEN, longueur de recherche pour trouver la valeur maximale; et MLS, nombre maximal de décalages à gauche permis. Les sorties de VSCALE sont les suivantes: OUT, vecteur de sortie; et NLS, nombre de décalages à gauche utilisés pour normaliser le vecteur d'entrée. On admet par hypothèse que les vecteurs d'entrée et de sortie sont du même type et peuvent être soit la virgule flottante de bloc à simple précision (nombres entiers de 16 bits), soit la virgule flottante de bloc à double précision (entiers de 32 bits). Pour les vecteurs à simple précision, on a $MLS = 14$ et, pour les vecteurs à double précision, $MLS = 30$. Dans certains cas, on souhaite utiliser moins de 16 ou 32 bits pour représenter une variable. Il existe par exemple plusieurs variables qui sont spécifiées avec une précision de 14 ou 15 bits, auquel cas on pose $MLS = 12$ ou 13 respectivement. En raison de cette éventualité, il est possible que l'on soit obligé d'appliquer des décalages à droite, au lieu de décalages à gauche, pour normaliser la variable. En pareils cas, la valeur de NLS renvoyée sera négative. Par exemple, si le système renvoie $NLS = -1$, cela signifie qu'il a fallu appliquer un décalage de 1 bit vers la droite. Le module admet qu'il y a un accumulateur (AA0) disponible pour l'opération de décalage et que cet accumulateur a une précision d'au moins 32 bits. Si l'on sait que l'élément maximal est le premier, on posera $SLEN = 1$. Dans le cas contraire, on posera $SLEN = LEN$ et la recherche de la valeur maximale portera sur la totalité du vecteur.

Le code décrit ci-après respecte la convention selon laquelle les données sont représentées sous la forme des compléments à 2. Il traite séparément les cas dans lesquels les amplitudes maximales sont positives ou négatives.

```

SUBROUTINE VSCALE(IN, LEN, SLEN, MLS, OUT, NLS)
AA0 = IN(1) | Trouver la valeur pos. max. de l'entrée
AA1 = IN(1) | Trouver la valeur nég. max. de l'entrée
If SLEN = 1, skip the next 3 lines
  For I = 2, 3, ..., SLEN, do the next two lines
    If IN(I) > AA0, set AA0 = IN(I)
    If IN(I) < AA1, set AA1 = IN(I)

| Cas 1: vecteur d'entrée de valeur nulle

If AA0 = 0 and AA1 = 0, do the next 3 lines
  For I = 1, 2, ..., LEN, set OUT(I) = 0
  NLS = MLS + 1 | Faire en sorte que 0 ait 1 bit de décalage
  Exit this subroutine | à gauche en plus que 1

NLS = 0 | Initialiser NLS

| Déterminer cas 2 ou cas 3

If AA0 < 0 or AA1 < -AA0, then do the following indented lines
  MAXI = -2MLS | Cas 2: le nombre négatif est le plus grand
  MINI = 2 * MAXI | Limite inférieure de la mantisse après décalage
  If AA1 < MINI, then do the following doubly indented lines to find the number of right shifts needed and then scale the
  elements

LOOP1R: AA1 = AA1 >> 1
NLS = NLS - 1 | NLS négatif ==> décalages à droite
If AA1 < MINI, go to LOOP1R
For I = 1, 2, 3, ..., LEN, do the next line
  OUT(I) = IN(I) >> -NLS
Exit this subroutine

LOOP1L: If AA1 < MAXI, go to SCALE1 | Trouver le nombre de décalages à gauche
AA1 = AA1 << 1
NLS = NLS + 1
Go to LOOP1L

SCALE1: For I = 1, 2, 3, ..., LEN, do the next line
  OUT(I) = IN(I) << NLS
Exit this subroutine

Else, do the following indented lines
| Cas 3: le nombre positif est le plus grand

MINI = 2MLS | Limite inférieure de la mantisse après décalage
MAXI = MINI - 1 | 2 * MIN débordera si MLS = 30
MAXI = MAXI + MINI | Limite supérieure de la mantisse
If AA0 > MAXI, then do the following doubly indented lines to find the number of right shifts needed and then scale the
elements

LOOP2R: AA0 = AA0 >> 1
NLS = NLS - 1
If AA0 > MAXI, go to LOOP2R
For I = 1, 2, 3, ..., LEN, do the next line
  OUT(I) = IN(I) >> -NLS
Exit this subroutine

LOOP2L: If AA0 ≥ MINI, go to SCALE2
AA0 = AA0 << 1
NLS = NLS + 1
Go to LOOP2L

SCALE2: For I = 1, 2, 3, ..., LEN, do the next line
  OUT(I) = IN(I) << NLS
Exit this subroutine

```

Dans certains cas, on constate qu'il n'est pas vraiment souhaitable de renormaliser les données, mais qu'il suffit de déterminer le nombre de décalages à gauche qui seraient nécessaires si l'on voulait effectuer cette nouvelle normalisation. Le programme suivant utilise les mêmes entrées que VSCALE mais fournit seulement NLS comme sortie. La seule différence avec VSCALE est que ce programme ne normalise pas le vecteur d'entrée.

```

SUBROUTINE FINDNLS(IN, SLEN, MLS, NLS)
AA0 = IN(1) | Trouver la valeur pos. max. de l'entrée
AA1 = IN(1) | Trouver la valeur nég. max. de l'entrée
If SLEN = 1, skip the next 3 lines
  For I = 2, 3, ..., SLEN, do the next two lines
    If IN(I) > AA0, set AA0 = IN(I)
    If IN(I) < AA1, set AA1 = IN(I)

If AA0 = 0 and AA1 = 0, do the next 2 lines
  NLS = MLS + 1 | Cas 1: vecteur d'entrée de valeur nulle
  Exit this subroutine | Faire en sorte que 0 ait un bit de décalage
                        | à gauche en plus que 1

NLS = 0 | Initialiser NLS
                        | Déterminer cas 2 ou cas 3

If AA0 < 0 or AA1 < -AA0, then do the following indented lines
  MAXI = -2MLS | Cas 2: le nombre négatif est le plus grand
  MINI = 2 * MAXI | Limite inférieure de la mantisse après décalage
  If AA1 < MINI, then do the following doubly indented lines to find the number of right shifts needed

LOOP1R:  AA1 = AA1 >> 1
         NLS = NLS - 1 | NLS négatif => décalages à droite
         If AA1 < MINI, go to LOOP1R
         Exit this subroutine

LOOP1L:  If AA1 < MAXI, exit this subroutine | Trouver NLS
         AA1 = AA1 << 1
         NLS = NLS + 1
         Go to LOOP1L

Else, do the following indented lines
  MINI = 2MLS | Cas 3: le nombre positif est le plus grand
  MAXI = MINI - 1 | Limite inférieure de la mantisse après décalage
  MAXI = MAXI + MINI | 2 * MIN débordera si MLS = 30
                    | Limite supérieure de la mantisse
  If AA0 > MAXI, then do the following doubly indented lines to find the number of right shifts needed

LOOP2R:  AA0 = AA0 >> 1
         NLS = NLS - 1
         If AA0 > MAXI, go to LOOP2R
         Exit this subroutine

LOOP2L:  If AA0 ≥ MINI, exit this subroutine | Trouver NLS
         AA0 = AA0 << 1
         NLS = NLS + 1
         Go to LOOP2L

```

G.1.3.2 Multiplication

La multiplication de deux nombres à virgule fixe donne un nombre de 32 bits, généralement stocké dans un registre de produit d'une puce DSP. Si les deux nombres à virgule fixe étaient aux formats Q_n et Q_m , le résultat stocké dans le registre de produit est au format $Q_{(n+m)}$. Après l'avoir additionné dans un accumulateur, il peut être nécessaire de décaler le résultat comme expliqué dans le paragraphe précédent.

La multiplication de deux mots à virgule flottante s'effectue de la façon suivante: multiplication en virgule fixe des deux mantisses et addition des deux nombres NLS. Comme indiqué plus haut, le produit est un mot de 32 bits en format $Q(n + m)$. Si ce produit doit être ramené à la version avec virgule flottante, il pourra être nécessaire de lui appliquer une autre normalisation. Par exemple, dans la multiplication de deux mots positifs à virgule flottante, le produit doit avoir un 1 dans le bit 30 ou le bit 29. Une nouvelle normalisation est nécessaire si le bit 30 est un 0. Cela signifie qu'il faut appliquer un décalage à gauche supplémentaire. A la suite de ce décalage, le produit est représenté en format $Q(n + m + 1)$. S'il doit être mémorisé avec virgule flottante scalaire, il devra être arrondi avant la mémorisation. Si le produit doit être mémorisé avec virgule flottante de bloc, il faudra renormaliser l'ensemble de la table en fonction de la valeur qui a maintenant la plus grande amplitude.

Des variables à double précision sont utilisées dans certaines parties de ce codeur, mais il n'y a pas de multiplications à double précision. Ces variables sont multipliées dans certains cas, mais alors seuls les bits de plus fort poids sont utilisés. Ces cas sont consignés dans le pseudo-code.

G.1.3.3 Addition

Pour additionner deux nombres à virgule fixe, il faut que ces nombres soient mémorisés dans le même format Q. En général, c'est la valeur mémorisée avec la plus grande gamme dynamique qui détermine quelle valeur doit être convertie au format Q approprié. Exemple: si on additionne des valeurs mémorisées en formats Q9 et Q11, la valeur en format Q11 doit être décalée à droite de 2 bits avant d'être additionnée avec la valeur mémorisée en format Q9.

L'addition des nombres à virgule flottante scalaire s'effectue de la même façon. Les deux valeurs doivent avoir le même nombre NLS. Ici encore, la valeur affectée du plus grand nombre NLS doit être décalée à droite, pour adaptation au NLS de l'autre valeur. Si la somme nécessite 17 bits pour être représentée, la somme présente dans l'accumulateur peut être décalée d'un bit à droite, puis arrondie pour être ramenée à 16 bits, et le nouveau nombre NLS aura un bit de moins que dans le format précédent. Considérons, par exemple, l'addition de deux valeurs dont les NLS sont 5 et 7. La valeur dont NLS = 7 doit être décalée de 2 bits à droite avant de pouvoir être additionnée avec l'autre valeur. Si les deux valeurs sont du même signe, la somme des deux mantisses peut avoir une amplitude supérieure à 32767. Dans ce cas, la valeur présente dans l'accumulateur doit être décalée d'un bit, puis arrondie. Le nombre NLS de la somme sera 4. Si les deux valeurs sont de signe opposé, le résultat obtenu dans l'accumulateur peut avoir une mantisse d'amplitude inférieure à 16384. Dans ce cas, il faut appliquer au résultat une nouvelle normalisation en effectuant des décalages à gauche jusqu'à ce que l'amplitude soit supérieure ou égale à 16384 et que le NLS soit accru d'une quantité égale au nombre de décalages à gauche. Dans l'exemple considéré, avec des NLS initiaux de 5 et 7, le NLS final ne peut pas être supérieur à 6 ni inférieur à 4.

L'addition des nombres à virgule flottante de bloc se trouve compliquée par le fait que les contraintes sont déterminées sur la base de la valeur qui a la plus grande amplitude. En pareil cas, si deux vecteurs ont des NLS de 5 et 7, celui dont NLS = 7 doit être décalé de 2 bits à droite. On effectue la sommation de chacun des couples. C'est la plus grande des sommes ainsi obtenues qui détermine si une nouvelle normalisation est nécessaire.

G.1.3.4 Division

La division est utilisée beaucoup moins fréquemment que l'addition ou la multiplication. Les seules divisions auxquelles on procède sont des divisions avec virgule flottante scalaire. Le numérateur et le dénominateur sont représentés en format normalisé, de même que le quotient. On calcule le nombre NLS du quotient en retranchant le NLS du dénominateur de celui du numérateur et en ajoutant 14. Pour expliquer ce terme 14, considérons le cas où le numérateur était un peu supérieur au dénominateur, et où NLS = 0 pour les deux. Dans ce cas, on aurait NLS = 14 pour le quotient et ce dernier serait convenablement normalisé. Si la mantisse du numérateur est inférieure à celle du dénominateur, il faut appliquer au numérateur un décalage d'un bit vers la gauche et augmenter son NLS de 1 pour calculer le NLS du quotient. Cela garantit que la mantisse du quotient sera au format normalisé.

La division intervient dans la récurrence de Durbin, programme qui exige que le résultat ait la précision totale sur 16 bits. Les programmes de division approximative ne suffisent donc pas. La mantisse du résultat doit avoir la précision totale sur 16 bits, y compris l'arrondi du résultat à 17 bits. On trouvera plus loin le pseudo-code correspondant à une telle division.

Si le numérateur ou le dénominateur n'est pas stocké au départ avec virgule flottante scalaire, il devra tout d'abord être converti à ce format. La fonction `FLOAT(.)` est utilisée dans le pseudo-code pour représenter ces conversions. L'argument pourrait être une virgule fixe à simple précision ou à double précision.

Pseudo-code pour la division à virgule flottante

Ce programme est utilisé pour la division avec virgule flottante sur un dispositif à virgule fixe à 16 bits. L'hypothèse est que l'on dispose d'au moins un accumulateur à 32 bits. Toutes les entrées et toutes les sorties sont des mots de 16 bits.

Entrée: NUM, NUMNLS, DEN, DENNLS

Sortie: QUO, QUONLS

Fonction: Calculer le quotient. NUM et NUMNLS sont la mantisse et le format Q pour le numérateur. DEN et DENNLS sont la mantisse et le format Q pour le dénominateur. QUO et QUONLS sont la mantisse et le format Q pour le quotient. On admet par hypothèse que tous ces termes sont au format normalisé. Il n'existe pas de test pour DEN = 0; l'hypothèse est DEN ≠ 0.

SUBROUTINE DIVIDE(NUM, MUMNLS, DEN, DENNLS, QUO, QUONLS)

SIGN = 1 | Déterminer d'abord

P = NUM * DEN | le bit de signe

If P < 0, set SIGN = -1 | du quotient

QUONLS = NUMNLS - DENNLS + 14 | Calculer ensuite QUONLS

A0 = | NUM | | A0 est l'accumulateur à 32 bits

| | NUM | est contenu dans les 16 bits inférieurs

A1 = | DEN | | A1 peut être un registre à 16 ou 32 bits,

| si 32 bits, | DEN | est contenu

| dans les 16 bits inférieurs

If A0 < A1, do the next 2 lines

QUONLS = QUONLS + 1

A0 = A0 << 1

QUO = 0 | Initialisation du quotient

I = 0 | Initialisation du compteur de boucle

LOOP: QUO = QUO << 1 | Boucle de division longue

If A0 ≥ A1, do the next 2 lines

QUO = QUO + 1

A0 = A0 - A1

A0 = A0 << 1

I = I + 1

If I < 15, GO TO LOOP

If A0 ≥ A1, set QUO = QUO + 1 | Attention à l'arrondi

If SIGN < 0, set QUO = -QUO | Attention au signe

G.2 Modifications de l'algorithme

G.2.1 Modifications de l'adaptateur en boucle du gain vectoriel (bloc 20)

NOTE – Les informations données dans le présent paragraphe se fondent sur 3.8/G.728. Le lecteur aura intérêt à se familiariser avec ce paragraphe 3.8 avant d'aborder le présent paragraphe. Les modifications signalées ci-après concernent les calculs effectués une fois par vecteur pour l'adaptateur en boucle du gain vectoriel. Dans toute la mesure possible, on a utilisé ici les mêmes notations que dans la Recommandation G.728.

Dans le présent paragraphe, on décrira succinctement les opérations effectuées une fois par vecteur pour l'adaptateur en boucle du gain vectoriel, telles que spécifiées dans la Recommandation G.728 pour la mise en œuvre avec virgule flottante. On décrira ensuite une méthode mathématiquement équivalente, dont la mise en œuvre dans les processeurs du type à virgule fixe est plus aisée et plus précise. L'addendum à la présente annexe contient des tableaux de valeurs nécessaires pour l'application de cette nouvelle méthode.

On trouvera ci-après une brève description des opérations avec virgule flottante. La table GSTATE des variables d'état internes, représentée par δ , contient les 10 valeurs précédentes du gain logarithmique à écart soustrait. Le symbole $\delta(n)$ désigne le gain logarithmique à écart soustrait pour le vecteur n . Le signal de sortie du prédicteur de gain logarithmique [version prédite de $\delta(n)$] a pour expression:

$$\hat{\delta}(n) = - \sum_{i=1}^{10} \alpha_i \delta(n-i) \quad (\text{G-1})$$

La Figure 6/G.728 montre que, avant de convertir $\hat{\delta}(n)$ au domaine linéaire, il faut ajouter un écart de gain de 32 dB et vérifier le résultat pour s'assurer que:

$$0 \leq \hat{\delta}(n) + 32 \leq 60 \quad (\text{G-2})$$

Cela revient à dire que l'intervalle autorisé pour $\hat{\delta}(n)$ est:

$$-32 \leq \hat{\delta}(n) \leq 28 \quad (\text{G-3})$$

L'estimation du gain dans le domaine linéaire est donnée par:

$$\sigma(n) = 10^{(\hat{\delta}(n) + 32)/20} \quad (\text{G-4})$$

Dans un premier temps, la valeur de $\sigma(n)$ sert à normaliser le vecteur cible VQ d'excitation. Une fois achevée la recherche dans le répertoire codé, on utilise $\sigma(n)$ pour normaliser le meilleur vecteur code choisi. Supposons qu'on ait choisi, pour le vecteur n , l'indice i dans le répertoire de gain codé et l'indice j dans le répertoire de forme codé; alors, le vecteur d'excitation $e(n)$ aura pour expression:

$$e(n) = \sigma(n) g_i y_j \quad (\text{G-5})$$

où y_j désigne le j ème vecteur code de forme et g_i le i ème niveau de gain dans le répertoire de gain codé. On se sert ensuite du vecteur d'excitation $e(n)$ pour calculer $\delta(n)$. On calcule tout d'abord le carré de la valeur quadratique moyenne (RMS) de $e(n)$ [c'est-à-dire la «puissance» de $e(n)$], qui a pour expression:

$$P[e(n)] = \frac{1}{5} \sum_{k=1}^5 e_k^2(n) \quad (\text{G-6})$$

Pour un vecteur donné x , le symbole $P[x]$ représente la puissance de x , définie comme étant l'énergie de x divisée par la dimension vectorielle de x . Avant de convertir $P[e(n)]$ à la valeur en dB dans le domaine logarithmique, on force $P[e(n)]$ à 1 si sa valeur est inférieure à 1. L'intervalle autorisé pour $P[e(n)]$ est donc:

$$P[e(n)] \geq 1 \quad (\text{G-7})$$

Cela permet d'éviter le débordement dans la conversion logarithmique, ou l'obtention d'une valeur en dB trop petite. A noter que cette opération de limitation de plage n'est pas représentée explicitement dans la Figure 6/G.728 mais qu'elle est effectuée dans le «pseudo-code» spécifié au 5.7/G.728. On détermine alors le gain logarithmique à écart soustrait à l'aide de la formule:

$$\delta(n) = 10 \log_{10} P[e(n)] - 32 \quad (\text{G-8})$$

On notera que l'équation (G-7) a pour corollaire:

$$\delta(n) \geq -32 \quad (\text{G-9})$$

L'opération suivante consiste à utiliser $\delta(n)$, calculé à l'aide de l'équation (G-8), pour prédire les gains d'excitation suivants et pour mettre à jour les coefficients du prédicteur de gain logarithmique. Ceci est la dernière étape de la brève analyse du fonctionnement avec virgule flottante pour l'adaptateur en boucle du gain vectoriel.

Passons maintenant à la description de la méthode mathématiquement équivalente pour l'opération avec virgule fixe. Soit y_{jk} le k ème élément du j ème vecteur code dans le répertoire de forme codé. En combinant les équations (G-5) et (G-6), on obtient:

$$P[e(n)] = \sum_{k=1}^5 \left(\sigma(n) g_i y_{jk} \right)^2 \quad (\text{G-10})$$

$$= \sigma^2(n) g_i^2 \left(\frac{1}{5} \sum_{k=1}^5 y_{jk}^2 \right) \quad (\text{G-11})$$

$$= \sigma^2(n) g_i^2 P[y_j] \quad (\text{G-12})$$

En portant l'équation (G-12) dans l'équation (G-8), il vient:

$$\delta(n) = 20 \log_{10} \sigma(n) - 32 + 20 \log_{10} |g_i| + 10 \log_{10} P[y_j] \quad (\text{G-13})$$

Compte tenu de l'équation (G-4), $\delta(n)$ peut s'écrire:

$$\delta(n) = \hat{\delta}(n) + 20 \log_{10} |g_i| + \log_{10} P[y_j] \quad (\text{G-14})$$

En d'autres termes, $\delta(n)$ est égal simplement au gain logarithmique $\hat{\delta}(n)$ prédit plus deux «termes de correction»:

- 1) $20 \log_{10} |g_i|$, valeur en dB du meilleur niveau de gain choisi dans le répertoire de gain codé; et
- 2) $10 \log_{10} P[y_j]$, valeur en dB de la puissance du meilleur vecteur code de forme choisi dans le répertoire de forme codé. (Dans un certain sens, on a l'analogie d'un codeur prédictif classique pour le gain, mais fonctionnant dans le domaine logarithmique.)

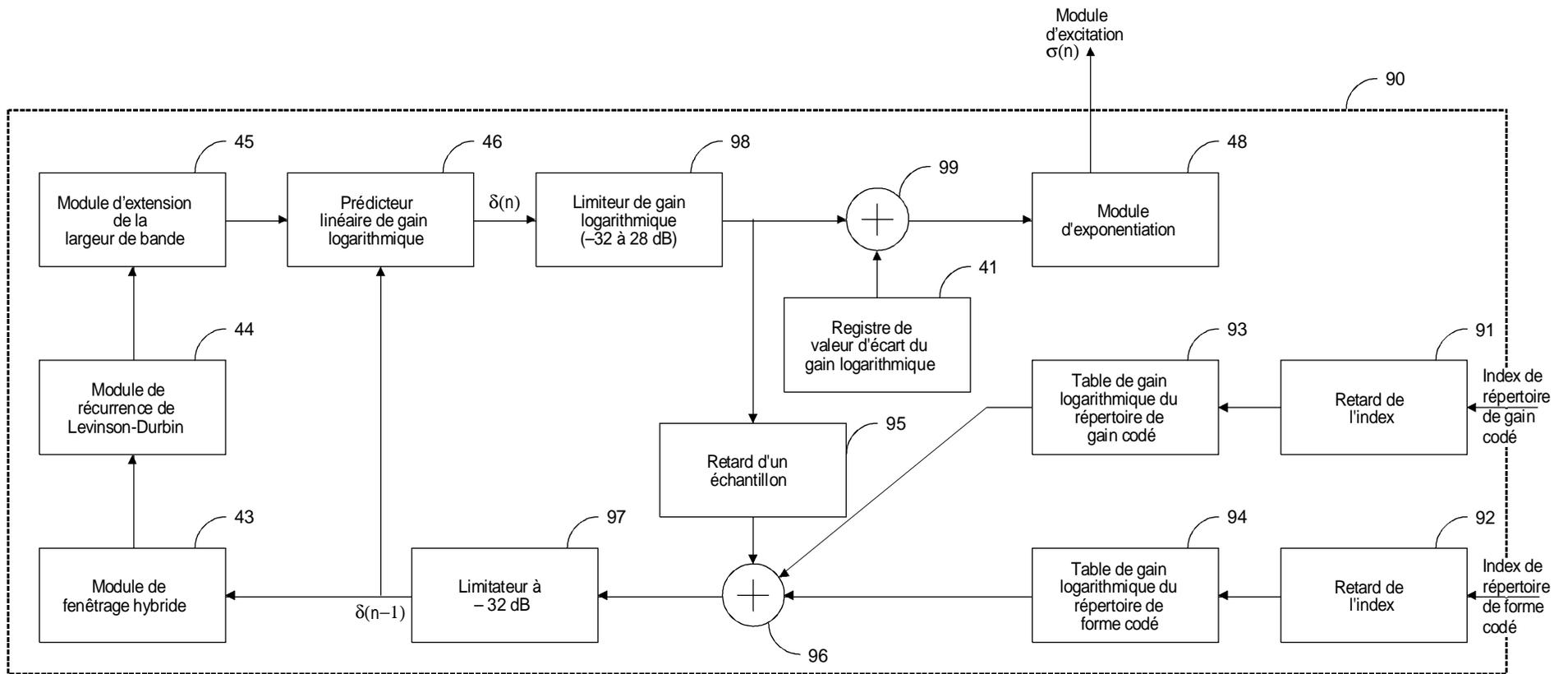
La Figure G.1 montre l'organigramme de cette méthode mathématiquement équivalente. Il n'existe que 4 valeurs possibles de $|g_i|$ et 128 valeurs possibles de $P[y_j]$; on peut donc précalculer les valeurs en dB correspondantes et les stocker dans deux tables de gain logarithmique (blocs 93 et 94 dans la Figure G.1).

Les unités de délai 91 et 92 fournissent les meilleurs index de répertoires de gain codé et de forme codé choisis dans le répertoire d'excitation du vecteur précédent. Ces deux index servent à consulter les valeurs de $20 \log_{10} |g_i|$ et $10 \log_{10} P[y_j]$ dans les tables du gain logarithmique des blocs 93 et 94. L'unité de délai 95 (délai de 1 échantillon) contient le gain logarithmique $\hat{\delta}(n-1)$ prédit précédemment (et, éventuellement, à plage limitée). L'additionneur 96 additionne les signaux de sortie des blocs 93, 94 et 95 pour donner un gain $\hat{\delta}(n-1)$ non échantillonné, selon l'équation (G-14). Ensuite, le limiteur 97 met en œuvre l'inégalité (G-9), en échantillant le signal de sortie de l'additionneur 96 à -32 dB si le niveau de ce signal est inférieur à -32 dB.

Le signal de sortie du limiteur 97 est mathématiquement équivalent au signal de sortie de l'additionneur 42 de la Figure 6/G.728. Les blocs 43 à 46 de la Figure G.1 sont par conséquent identiques à leurs homologues de la Figure 6/G.728. Le fonctionnement du limiteur de gain logarithmique 98 est similaire à celui du limiteur 47 de la Figure 6/G.728, avec cette différence que l'intervalle autorisé a été réduit de 32 dB. L'additionneur 99 ajoute la valeur de l'écart de gain logarithmique (32 dB), stockée dans le bloc 41, au signal de sortie du limiteur de gain logarithmique 98. La valeur de gain logarithmique ainsi obtenue est ensuite convertie au domaine linéaire par l'exponentiateur 48, lequel est identique à ses homologues de la Figure 6/G.728. Ceci est la dernière étape de la description de la méthode mathématiquement équivalente pour la version à virgule fixe.

La méthode équivalente, illustrée par la Figure G.1, présente deux avantages importants par rapport à la méthode initiale de la Figure 6/G.728.

- a) Avec la méthode équivalente, il est inutile de calculer la fonction logarithmique (bloc 40 de la Figure 6/G.728). Dans les versions avec traitement numérique des signaux (DSP) (*digital signal processing*), la fonction logarithmique est généralement calculée au moyen d'un développement de puissance en série; il faut normalement un grand nombre de cycles d'instruction pour effectuer ce calcul. Si on remplace le calcul du logarithme par une consultation de table, il pourrait donc en résulter une très grande économie de cycles DSP. Par ailleurs, il est possible de précalculer avec le maximum de précision souhaité.



T1514700-93/d01

FIGURE G.1/G.728

Adaptateur en boucle de gain vectoriel pour la mise en œuvre avec virgule fixe

- b) La méthode équivalente a des chances de donner des résultats numériques plus précis que la méthode d'origine en cas d'utilisation d'un processeur fonctionnant avec virgule fixe. Du fait de l'adaptation en boucle, il existe une boucle de rétroaction dans le processus d'adaptation du gain. Cette boucle de rétroaction est très longue dans la Figure 6/G.728. Elle s'étend de l'exponentiateur 48 jusqu'au module 21 de normalisation du gain (Figure 2/G.728) et revient ensuite aux blocs 67, 39, 40 et 42 à 48. Plus il y a de calculs dans cette boucle, plus celle-ci a de chances de cumuler des erreurs numériques, en raison de la valeur finie de la précision. Il en est ainsi surtout si le processeur à virgule fixe ne donne pas toujours la précision maximale possible pour la fonction logarithmique. A l'inverse, la boucle de rétroaction de la Figure G.1 est aussi compacte qu'elle peut l'être. On notera que tous les éléments suivants sont ici extérieurs à la boucle: le module de normalisation du gain, le calcul de l'énergie et de la puissance pour $e(n)$, le calculateur de logarithme et même l'additionneur qui a pour fonction de rétablir l'écart du gain logarithmique. A l'exception des blocs 43 à 46, qui sont communs aux deux méthodes, la boucle de rétroaction fait intervenir seulement deux limitations et deux additions, qui peuvent être effectuées avec une très grande précision par des processeurs fonctionnant avec virgule fixe.

Ce changement renforce l'interopérabilité des formes de mise en œuvre avec virgule fixe et avec virgule flottante de la Recommandation G.728. Le principal inconvénient de cette nouvelle méthode est la nécessité de disposer de mots supplémentaires dans la mémoire ROM. On a 128 vecteurs de forme et 4 vecteurs de gain possibles. La capacité de mémoire supplémentaire requise est de $128 + 4 = 132$ mots. Elle ne représente qu'une très petite fraction de l'espace ROM dont on a déjà besoin pour l'application de la Recommandation G.728.

Avec cette nouvelle méthode, l'entrée de l'adaptateur en boucle du gain vectoriel (blocs 20 et 30) n'est plus $e(n)$, mais les indices i et j des répertoires de gain et de forme codés. Pour tenir compte de ce fait, on aurait dû redessiner ici les Figures 1/G.728 à 3/G.728 afin de montrer que l'entrée de cet adaptateur provient du bloc répertoire codé d'excitation VQ. On n'a cependant pas reproduit ici les figures modifiées, parce que la modification nécessaire se résume à peu de chose et que les explications qui précèdent la font parfaitement comprendre.

G.2.2 Modifications des modules de la récurrence de Levinson-Durbin

Ce paragraphe traite des modifications apportées aux modules de la récurrence de Levinson-Durbin (voir la Recommandation G.728). Ces modules sont au nombre de trois (blocs 37, 44 et 50), utilisés respectivement pour le filtre de pondération perceptive, le prédicteur linéaire de gain logarithmique et le filtre de synthèse. Pour plus de détails, on se reportera aux 5.5/G.728 et 5.6/G.728. Dans le présent paragraphe, on considérera à titre d'exemple le pseudo-code relatif au bloc 50 (5.6/G.728) et on indiquera les modifications qu'il faut lui appliquer pour réaliser la mise en œuvre avec virgule fixe. Des modifications analogues doivent être apportées au bloc 37 (filtre de pondération perceptive) et au bloc 44 (prédicteur de gain logarithmique). On commencera par exposer le pseudo-code de la version à virgule flottante.

If RTMP(LPC + 1) = 0, go to LABEL	Si valeur nulle
If RTMP(1) ≤ 0, go to LABEL	Si signal nul
RC1 = -RTMP(2)/RTMP(1)	
ATMP(1) = 1	
ATMP(2) = RC1	Prédicteur du premier ordre
ALPHATMP = RTMP(1) + RTMP(2) * RC1	
If ALPHATMP ≤ 0, go to LABEL	Abandonner si défaut de conditionnement
For MINC = 2, 3, 4, ..., LPC, do the following	
SUM = 0.	
For IP = 1, 2, 3, ..., MINC, do the next 2 lines	
N1 = MINC - IP + 2	
SUM = SUM + RTMP(N1) * ATMP(IP)	
RC = -SUM/ALPHATMP	Facteur de réflexion
MH = MINC/2 + 1	
For IP = 2, 3, 4, ..., MH, do the next 4 lines	
IB = MINC - IP + 2	
AT = ATMP(IP) + RC * ATMP(IB)	
ATMP(IB) = ATMP(IB) + RC * ATMP(IP)	Mettre à jour les coefficients du prédicteur
ATMP(IP) = AT	

ATMP(MINC + 1) = RC	
ALPHATMP = ALPHATMP + RC * SUM	Energie résiduelle de prédiction
If ALPHATMP ≤ 0, go to LABEL	Abandonner si défaut de conditionnement
Repeat the above for the next MINC	
	Fin normale de la récurrence
Exit this program	si l'exécution parvient à ce point

LABEL: Si le programme parvient à ce point, un défaut de conditionnement s'est produit. Dans ce cas, sauter le bloc 51, ne pas mettre à jour les coefficients du filtre de synthèse (c'est-à-dire utiliser les coefficients de filtre du synthèse du précédent cycle d'adaptation).

La meilleure façon de commencer est de considérer les variables à virgule flottante mentionnées dans ce pseudo-code, à savoir RC, RC1, RTMP, SUM, ALPHATMP et ATMP. (Toutes les autres variables du code: MINC, IP, IB et N1 sont des index entiers.)

RC désigne les facteurs de réflexion, qui sont calculés en tant que variable intermédiaire dans ce module. Les facteurs de réflexion ont la propriété suivante: leur amplitude est toujours inférieure à l'unité dans le cas d'un filtre LPC stable. Cela étant, RC peut être représenté dans un format Q15, ce qui signifie qu'un bit est utilisé comme bit de signe et les 15 autres bits servent à représenter la partie fractionnaire de la valeur.

A noter ce qui suit: à chaque itération on calcule RC, on utilise cette variable pour l'itération correspondante et on ne l'utilise plus jamais après. La seule exception est la suivante: pour l'analyse du filtre LPC de synthèse du décodeur, RC1 est mis en réserve pour utilisation ultérieure dans le postfiltre. Afin d'économiser de la mémoire, seule la valeur de RC1 est mise en réserve. Toutes les autres valeurs de RC sont inscrites dans la même position et écrasées à l'itération suivante. Ce procédé est différent du pseudo-code initial pour virgule flottante, mais il est sans effet sur les résultats finals et il peut être appliqué également aux formes de mise en œuvre avec virgule flottante.

RTMP se rapporte aux valeurs de la fonction d'autocorrélation, qui ont une gamme dynamique énorme. On est obligé de conserver RTMP avec la virgule flottante de bloc, ce qui signifie que toutes les valeurs sont normalisées par la même puissance de 2. En théorie, la plus grande valeur devrait être celle de RTMP(1) et l'on sait aussi que cette valeur doit être positive. La représentation utilisée sera telle que l'amplitude maximale de RTMP soit comprise entre 0,5 et 1. Dans ces conditions, la totalité de RTMP peut être mise au format Q15. La totalité d'un RTMP est représentée avec une virgule flottante de bloc dans une fenêtre hybride, mais on a besoin uniquement des mantisses dans la récurrence de Durbin.

Considérons maintenant RTMP(LPC + 1). Comme indiqué à la première ligne, si cette variable a la valeur zéro, il faut arrêter le module. Si RTMP(LPC + 1) est représenté par un entier composé de 16 bits, ce cas a beaucoup plus de chances de se produire que si le même RTMP(LPC + 1) était représenté par un nombre de 32 bits à virgule flottante dans une forme de réalisation à virgule flottante. Cela crée des problèmes au point de vue de l'interopérabilité. Lors du calcul de RTMP(LPC + 1) dans le module précédent (fenêtre hybride, bloc 49), la valeur est stockée dans l'accumulateur, qui a une capacité d'au moins 32 bits dans tous les processeurs numériques de signaux ou processeurs DSP (*digital signal processing*) du type à virgule fixe. Il est proposé de tester cette valeur de 32 bits de l'accumulateur, une fois le calcul terminé, pour voir si elle est égale à zéro. Cette modification permet d'empêcher la terminaison prématurée (et, par voie de conséquence, des difficultés d'interopérabilité). Dans le nouveau code, un test est pratiqué sur une variable logique appelée ILLCOND, pour voir si elle est vraie ou fausse. Sa valeur dépend des résultats du test de RTMP(LPC + 1) effectué dans le module à fenêtre hybride. A un stade ultérieur, on utilise ILLCOND comme variable de sortie pour ce bloc, afin d'indiquer s'il faut utiliser les valeurs de sortie ou ne pas en tenir compte.

Pour le postfiltre, la possibilité existe que le défaut de conditionnement se soit produit après la 10^e itération. Dans ce cas, un nouvel ensemble de coefficients de prédiction à court terme du postfiltre d'adaptation a été déterminé et ces coefficients sont valables; en revanche, les coefficients du filtre de synthèse du 50^e ordre ne sont pas valables. Une seconde variable logique, ILLCOND, indique l'état des coefficients de prédiction du postfiltre.

Les deux variables suivantes sont SUM et ALPHATMP. Ce sont l'une et l'autre des valeurs qui se sont accumulées mais qui ne sont jamais multipliées. La valeur stockée dans un accumulateur se compose de 32 bits. Cependant, ces deux variables sont divisées pour le calcul de RC. Elles sont converties à la virgule fixe 16 bits pour cette division. Le résultat de la division est représenté en format Q15 avec virgule fixe et affecté à RC. La variable SUM n'apparaît pas explicitement dans le pseudo-code à virgule fixe. Dans le format à 32 bits, c'est l'accumulateur AA0 et dans le format à 16 bits, la variable SIGN.

On notera que ALPHATMP est naturellement un nombre de 32 bits qui est accumulé dans un accumulateur. Cependant, dans les formes de mise en œuvre réelles en traitement DSP, il faut conserver ALPHATMP en mémoire pour la récurrence d'ordre immédiatement supérieur; la raison en est qu'on aura besoin de l'accumulateur pour d'autres calculs avant la mise à jour suivante de ALPHATMP. Il est donc possible de sauvegarder quelques cycles de DSP, si l'on sauvegarde et si l'on charge uniquement le mot supérieur de 16 bits (arrondi) de ALPHATMP, au lieu du mot de 32 bits dans sa totalité. On a constaté, dans la pratique, que le fait de sauvegarder uniquement le mot supérieur de ALPHATMP ne nuit pas à la qualité de fonctionnement du codeur. Pour cette raison, on se borne à sauvegarder le mot supérieur après chaque mise à jour de ALPHATMP. Afin de bien préciser à quels moments ALPHATMP est représenté respectivement par 16 bits et par 32 bits, on utilise le nom ALPHATMP dans le pseudo-code seulement lorsque c'est un nombre de 16 bits. Un accumulateur est référencé lorsqu'on a affaire à un nombre de 32 bits.

Le dernier vecteur de variables est ATMP, qui représente les coefficients du prédicteur. Dans toutes les simulations, la valeur maximale observée pour ATMP était inférieure à 4. Cela pourrait donner à penser qu'il faut utiliser uniformément le format Q13. Cependant, les mêmes simulations ont aussi montré que l'utilisation de ce format dans la récurrence de Durbin ne permet pas d'obtenir une interopérabilité suffisante avec les formes de mise en œuvre à virgule flottante. Pour obtenir une meilleure interopérabilité entre les versions à virgule fixe et à virgule flottante, on a constaté qu'il valait mieux recourir au format Q15, sauf dans les cas où ce format est cause de débordements.

Dans une puce du type DSP, les résultats du calcul de ATMP(IB) ou de ATMP(IP) sont, dans un premier temps, placés dans l'accumulateur. Dans la plupart des puces DSP 16 bits à virgule fixe, les accumulateurs ont une capacité d'au moins 32 bits. Les valeurs de ATMP(IB) et ATMP(IP) doivent être arrondies à la précision de 16 bits. Dans le code avec virgule fixe, il est important que l'accumulateur contienne des bits de garde ou présente un fanion de débordement; cela permet de détecter un débordement qui pourrait survenir pendant le calcul de ATMP(IB) ou ATMP(IP).

On a adopté la stratégie décrite ci-après pour choisir le format de ATMP. Les itérations peuvent être numérotées en fonction de la valeur de MINC. On commence avec $\text{MINC} = 2$ et avec le format Q15 pour AMTP. S'il n'y a jamais de débordement, c'est-à-dire si $|\text{ATMP(IP)}| < 1$ quel que soit IP, la représentation finale pour ATMP est Q15. L'autre cas possible est la survenue d'un débordement pendant une des itérations. Supposons qu'il s'agisse de l'itération K. Alors, toutes les valeurs de ATMP calculées au cours des itérations K et K - 1 doivent être converties au format Q14 par le moyen d'un décalage à droite. Ensuite, l'itération K se déroule une seconde fois, avec le format Q14. Dans les itérations subséquentes, à partir de K + 1, les calculs sont aussi effectués en format Q14 et des débordements peuvent également se produire avec ce format. En pareil cas, la même procédure est appliquée une nouvelle fois et le calcul continue en format Q13. On a observé empiriquement que ces débordements ne surviennent jamais en format Q13. On utilise les autres formats pour la raison suivante: le résultat est plus précis en l'absence de débordements. La représentation finale de ATMP, avant la sortie de ce bloc, est Q13, Q14 ou Q15.

On a aussi observé ce qui suit: après les opérations d'extension de la largeur de bande, les coefficients des filtres placés à la sortie des modules d'extension de la largeur de bande (blocs 38, 45, 51 et 85) peuvent toujours être représentés en format Q14. Autre observation: en utilisant, chaque fois que possible, la représentation en format Q15, on n'améliore pas le SNR en décodage croisé par rapport au résultat obtenu en format Q14. En conséquence, les trois modules d'extension de la largeur de bande convertissent toujours leurs tables de coefficients de sortie au format Q14, quel que soit le format (Q13, Q14 ou Q15) de leurs tables de coefficients d'entrée (c'est-à-dire la sortie des modules de la récurrence de Levinson-Durbin). Cela signifie que lorsqu'un de ces modules produit une table de coefficients de sortie en Q13 ou Q15, il doit en informer le module d'extension de la largeur de bande correspondant, pour permettre l'exécution d'un décalage additionnel afin de convertir la table en Q14. C'est la raison pour laquelle, dans le module de récurrence de Levinson-Durbin correspondant décrit ci-après, on a ajouté un fanion supplémentaire, NLSATMP, qui constitue une des sorties de ce module.

Dans le décodeur, la récurrence de Levinson-Durbin est interrompue après le calcul des coefficients de prédiction du 10^e ordre. Les valeurs sont gardées en mémoire à l'intention du postfiltre d'adaptation. Il existe par conséquent deux conditions de démarrage possibles. Dans le cas banal, la récurrence commence avec $\text{MINC0} = 1$. Dans le décodeur, on a une autre possibilité avec $\text{MINC0} = 10$. Dans ce dernier cas, il faut sauvegarder les valeurs de NRS et de ALPHATMP. A noter également que la valeur de NLSATMP doit être sauvegardée jusqu'à ce que $\text{ICOUNT} = 3$ au cours de l'exécution du module d'extension de la largeur de bande.

Enfin, on notera que ce programme met en œuvre trois accumulateurs. Le troisième accumulateur, AA2, a pour fonction d'enregistrer la valeur de RC en précision de 17 bits, aux fins de mise à jour du coefficient de prédiction le plus récent.

Le pseudo-code ci-après décrit la version à virgule fixe des modules de récurrence de Levinson-Durbin.

```

If MINC0 > 1, go to RECURSION
MINC0 = 1                                     | Initialisations pour le
ILLCONDP = .FALSE.                             | décodeur seulement

If ILLCOND = .TRUE., go to FAILED              | Sauter si RTMP(LPC + 1) est nul
If RTMP(1) ≤ 0, go to FAILED                   | Sauter si signal nul
NRS = 0                                         | Format Q15 initialement

DEN = RTMP(1)                                  | Calculer le prédicteur du premier ordre
NUM = RTMP(2)
If NUM < 0, set NUM = -NUM
Call SIMPDIV(NUM, DEN, AA0)                    | | RTMP(2) | /RTMP(1)

AA0 = AA0 << 15
RC1 = RND(AA0)
If RTMP(2) > 0, set RC1 = -RC1                 | Ajouter l'information de signe
RC = RC1                                       | Coefficients du prédicteur de premier ordre
ATMP(2) = RC1

AA0 = RTMP(1) << 16                            |
P = RTMP(2) * RC                              |
AA0 = AA0 + (P << 1)                          |
ALPHATMP = RND(AA0)                            | Stocker en mémoire le mot
                                                | supérieur de l'accumulateur DSP

```

RECURSION:

```

For MINC = MINC0 + 1, MINC0 + 2, ..., LPC, do the following indented lines
  AA0 = 0
  For IP = 2, 3, ..., MINC, do the next 3 lines
    N1 = MINC - IP + 2
    P = RTMP(N1) * ATMP(IP)
    AA0 = AA0 + P                               | 32 bits pour SUM
  AA0 = AA0 << 1

  AA0 = AA0 << NRS
  AA1 = RTMP(MINC + 1) << 16
  AA0 = AA0 + AA1                              |
  SIGN = RND(AA0)                              | Sauvegarder le signe du mot supérieur
  NUM = SIGN
  If NUM < 0, set NUM = -NUM
  If NUM ≥ ALPHATMP, go to FAILED               |
  Call SIMPDIV(NUM, ALPHATMP, AA0)             | Diviser pour obtenir RC
  AA2 = AA0 << 15                               | AA2 stocke RC de 17 bits
  RC = RND(AA2)
  If SIGN > 0, set RC = -RC

                                                | Mise à jour de ALPHATMP

  AA1 = ALPHATMP << 16
  P = RC * SIGN
  AA1 = AA1 + (P << 1)
  If AA1 ≤ 0, go to FAILED
  ALPHATMP = RND(AA1)

  MH = MINC/2 + 1                              | La partie fractionnaire de MINC/2 est tronquée;
                                                | MH = entier
                                                | Commencer la mise à jour des coefficients
                                                | du prédicteur

```

For IP = 2, 3, 4, ..., MH, do the following doubly indented lines

IB = MINC - IP + 2

AA0 = ATMP(IP) << 16

| Charger le mot supérieur de AA0

P = RC * ATMP(IB)

| Q15 RC, donc << 1

AA0 = AA0 + (P << 1)

If AA0 overflowed, then do the following triply indented lines

NRS = NRS + 1

For LP = 2, 3, ..., MINC, set ATMP(LP) = ATMP(LP) >> 1

AA0 = ATMP(IP) << 16

| Nouvelle normalisation de ATMP

P = RC * ATMP(IB)

| Nouveau calcul de

AA0 = AA0 + (P << 1)

| AA0 avec débordement

AA1 = ATMP(IB) << 16

P = RC * ATMP(IP)

AA1 = AA1 + (P << 1)

If AA1 overflowed, then do the following triply indented lines

NRS = NRS + 1

For LP = 2, 3, ..., MINC, set ATMP(LP) = ATMP(LP) >> 1

AA0 = ATMP(IP) << 16

| Nouvelle normalisation de ATMP(IP)

P = RC * ATMP(IB)

| Nouveau calcul de AA0

AA0 = AA0 + (P << 1)

|

AA1 = ATMP(IB) << 16

| Nouvelle normalisation de ATMP(IB)

P = RC * ATMP(IP)

| Nouveau calcul de

AA1 = AA1 + (P << 1)

| AA1 avec débordement

ATMP(IP) = RND(AA0)

ATMP(IB) = RND(AA1)

AA0 = AA2 >> NRS

| Mise à jour de ATMP(MINC + 1)

| AA2 contient RC de 17 bits

AA0 = RND(AA0)

| Signal de sortie dans mot inférieur de AA0

If SIGN > 0, set AA0 = -AA0

ATMP(MINC + 1) = AA0

| Le mot inférieur est stocké dans ATMP

Repeat the above indented lines for the next MINC

NLSATMP = 15 - NRS

If NLSATMP < 13, go to FAILED

Exit this program

| La récurrence est terminée normalement

| si le programme parvient à ce point

FAILED: Set ILLCOND = .TRUE.

If MINC ≤ 10, set ILLCONDP = .TRUE.

Si le programme parvient à ce point, un défaut de conditionnement s'est produit. Dans ce cas, sauter le bloc 51, ne pas mettre à jour les coefficients du filtre de synthèse (c'est-à-dire utiliser les coefficients du filtre de synthèse du précédent cycle d'adaptation).

Pour plus de commodité, on trouvera dans le tableau suivant la liste de toutes les variables du pseudo-code spécifié ci-dessus, avec indication de leur format de représentation.

Variable	Format	Taille	Temp/perm	Ancienne/nouvelle
AA0, AA1, AA2	DP-entier	1	temp	nouvelle
ALPHATMP	SFL	1	temp	ancienne
ATMP	Q13/Q14/Q15	51	perm	ancienne
IB	entier	1	temp	ancienne
ILLCOND	logique	1	perm	nouvelle
ILLCONDP	logique	1	perm	nouvelle
IP	entier	1	temp	ancienne
LP	entier	1	temp	nouvelle
MH	entier	1	temp	ancienne
MINC	entier	1	temp	ancienne
NLSATMP	entier	1	temp	nouvelle
NRS	entier	1	temp	nouvelle
NUM	entier	1	temp	nouvelle
RC	Q15	1	temp	nouvelle
RC1	Q15	1	temp	nouvelle
RTMP	Q15	51	perm	ancienne
SIGN	entier	1	temp	nouvelle
SFL	Virgule flottante scalaire 16 bits			
DP-entier	Registre à 32 bits, par exemple, accumulateur ou registres de produit (AA1, AA2 et P)			
entier	Entier de 16 bits			
Q13/Q14/Q15	Entier de 16 bits avec une de ces représentations			

Le code ci-dessus, écrit pour le bloc 50, utilisait des noms de variable associés à ce bloc. Cependant, il peut servir également pour les blocs 37 et 44. Le tableau ci-après donne la correspondance entre les noms de variable spécifiques au bloc 50 et les noms spécifiques aux autres blocs.

Bloc 50	Bloc 37	Bloc 44
ATMP	AWZTMP	GPTMP
ILLCOND	ILLCONDW	ILLCONDG
NLSATMP	NLSAWZTMP	NLSGPTMP
RTMP	R	R

Ce code fait usage d'un algorithme de division différent de celui qui est utilisé dans tout le reste de l'algorithme, et plus simple. Il est appelé ci-dessus SIMPDIV et son pseudo-code est spécifié ci-près. Les entrées sont NUM et DEN, qui sont tous deux des entiers de 16 bits. La sortie est AA0, avec résultats dans les 17 bits inférieurs.

```

Subroutine SIMPDIV(NUM, DEN, AA0)
AA0 = 0
AA1 = NUM
K = 0
LOOP: AA0 = AA0 << 1
      AA1 = AA1 << 1
      If AA1 ≥ DEN, then set AA1 = AA1 – DEN and AA0 = AA0 + 1
      K = K + 1
      If K < 16, go to LOOP

```

G.3 Pseudo-code pour les autres modules de la Recommandation G.728

On trouvera dans ce paragraphe le pseudo-code pour les autres modules de la Recommandation G.728. Le paragraphe précédent donnait le pseudo-code relatif à la récurrence de Levinson-Durbin, ainsi que les modifications apportées à l'algorithme applicable pour l'adaptateur en boucle du gain vectoriel. Pour chaque module, on indique tout d'abord le pseudo-code pour virgule flottante, suivi d'un commentaire et du pseudo-code pour virgule fixe. Le tableau ci-après peut servir de référence pour la recherche du pseudo-code relatif à tel ou tel module du codeur.

Pour tous les blocs, les pseudo-codes fournissent une spécification exacte sur les bits et représentent la dernière définition du codeur G.728 à virgule fixe. Tout écart par rapport à ces pseudo-codes peut se traduire par une simulation ou une mise en œuvre incorrectes.

Virgule fixe, G.728: Numéro de bloc, désignation et nom du pseudo-code

Bloc	Désignation	Pseudo-code
1	Conversion du format MIC d'entrée	Inutile
2	Tampon vectoriel	Inutile
3	Adaptateur du filtre de pondération	Utiliser le bloc 45
4	Filtre de pondération	Bloc 4
5-7	Interrupteur pour mise à jour de la mémoire du vecteur de réponse à entrée nulle (ZIR) (<i>zero input</i>)	Inutile
8	Décodeur simulé	Voir le détail des blocs ci-après
9	Filtre de synthèse pour ZIR	Blockzir
10	Filtre de pondération pour ZIR	Blockzir
9, 10	Blocs 9 et 10 pour mise à jour de la mémoire	Bloc 9
11	Calcul du vecteur quantifié cible	Bloc 11
12	Calculateur du vecteur de réponse impulsionnelle	Bloc 12
13	Convolution à inversion temporelle	Bloc 13
14	Convolution avec le vecteur code de forme	Bloc 14
15	Calculateur de table d'énergies du répertoire	Bloc 14
16	Normalisation du vecteur quantifié cible	Bloc 16
17	Calculateur d'erreur de recherche de vecteur quantifié (VQ)	Bloc 17
18	Sélecteur du meilleur index du répertoire d'excitation	Bloc 17
19	Répertoire des VQ d'excitation	Bloc 19
20	Adaptateur en boucle du gain vectoriel	Voir le détail des blocs ci-après
21	Module de normalisation du gain	Bloc 19
22	Filtre de synthèse	Utiliser le bloc 9
23	Adaptateur du filtre de synthèse	Voir les blocs 49-51

Virgule fixe, G.728: Numéro de bloc, désignation et nom du pseudo-code (fin)

Bloc	Description	Pseudo-code
24	Module de recherche dans le répertoire	Voir les blocs 12-18
28	Conversion format MIC de sortie	Inutile
29	Répertoire d'excitation du décodeur	Utiliser le bloc 19
30	Adaptateur en boucle de gain du décodeur	Même code que pour le bloc 20
31	Module de normalisation de gain du décodeur	Utiliser le bloc 19
32	Filtre de synthèse du décodeur	Bloc 32
33	Adaptateur du filtre de synthèse du décodeur	Même code que pour le bloc 23
34	Postfiltre	Voir les blocs 71-77
35	Adaptateur du filtre	Voir les blocs 81-85
36	Fenêtre hybride pour $W(z)$	Bloc 36
37	Récurrence de Durbin pour $W(z)$	Voir G.2
38	Calculateur des coefficients de $W(z)$	Bloc 38
43	Fenêtre hybride pour $GP(z)$	Bloc 43
44	Récurrence de Durbin pour $GP(z)$	Voir G.2
45	Extension de la largeur de bande de $GP(z)$	Bloc 45
46	Prédicteur linéaire de gain logarithmique	Bloc 46
48	Exponentiateur	Bloc 46
49	Fenêtre hybride pour $A(z)$	Bloc 49
50	Récurrence de Durbin pour $A(z)$	Voir G.2
51	Calculateur des coefficients de $A(z)$	Bloc 51
71-77	Blocs internes du postfiltre	Blocs correspondants
81-85	Blocs de l'adaptateur du postfiltre	Blocs correspondants
91	Module de délai de l'index du répertoire de gain codé	Inutile
92	Module de délai de l'index du répertoire de forme codé	Inutile
93	Table de gain logarithmique du répertoire de gain codé	Table dans G.5
94	Table de gain logarithmique du répertoire de forme codé	Table dans G.5
95	Délai de l'échantillon pour le gain logarithmique	Inutile
96	Additionneur pour la mise à jour du gain logarithmique	Bloc 46
97	Limiteur du gain logarithmique à -32 dB	Bloc 46
98	Limiteur du gain logarithmique: de -32 à 28 dB	Bloc 46
99	Additionneur pour le rétablissement de l'écart de gain	Bloc 46

G.3.1 Bloc 4 – Pseudo-code pour le filtre de pondération

On trouvera ci-après le pseudo-code avec virgule flottante pour le bloc 4, filtrage du signal de parole d'entrée par le filtre de pondération perceptive.

```
For K = 1, 2, ..., IDIM, do the following
  SW(K) = S(K)
  For J = LPCW, LPCW - 1, ..., 3, 2, do the next 2 lines
    SW(K) = SW(K) + WFIR(J) * AWZ(J + 1)      | Section tous zéros
    WFIR(J) = WFIR(J - 1)                    | du filtre

  SW(K) = SW(K) + WFIR(1) * AWZ(2)           | Traiter le dernier
  WFIR(1) = S(K)                             | vecteur différemment

  For J = LPCW, LPCW - 1, ..., 3, 2, do the next 2 lines
    SW(K) = SW(K) - WIIR(J) * AWP(J + 1)     | Section tous pôles
    WIIR(J) = WIIR(J - 1)                    | du filtre

  SW(K) = SW(K) - WIIR(1) * AWP(2)           | Traiter le dernier
  WIIR(1) = SW(K)                             | vecteur différemment
```

Repeat the above for the next K

Pour la version à virgule fixe de ce pseudo-code, on a les valeurs du nombre de décalages à gauche ou nombre de NLS (*number of left shifts*) associées à WFIR et WIIR. Le calcul doit être effectué de telle manière que la parole d'entrée ait le même NLS que WFIR et que le résultat de ce calcul ait le même NLS que WIIR. Dans ce cas, les valeurs de NLS pour WIIR et WFIR sont fixées à la même valeur que pour la parole d'entrée. AWZ et AWP sont en format Q14. La valeur correspondant à la parole d'entrée, NLSS, est 2. Différents formats d'entrée (linéaire à 16 bits, loi μ , loi A, etc.) sont censés opérer la conversion à la plage [-4096, +4095,75] représentée en format Q2.

Pour une entrée linéaire de K bits, on admet que les données occupent les K bits de plus faible poids d'un mot de 16 bits, K_BIT_SAMPLE. La représentation exacte est:

$$NLS = 15 - K$$

$$S = K_BIT_SAMPLE \ll NLS$$

Pour les signaux d'entrée linéaires de 16 bits (16_BIT_SAMPLE), il faut appliquer un décalage à droite de 1 bit:

$$S = 16_BIT_SAMPLE \gg 1$$

Pour la MIC en loi μ (MULAW_SAMPLE), la valeur de l'échantillon d'amplitude maximale est 4015,5 et on admet par hypothèse que cette valeur serait représentée par 8031 en format Q1. Pour la conversion au format Q2, il faut effectuer un décalage à gauche de 1 bit:

$$S = MULAW_SAMPLE \ll 1$$

Pour la MIC en loi A (ALAW_SAMPLE), la valeur de l'échantillon d'amplitude maximale est 2016, mais certaines valeurs d'échantillon ont une partie fractionnaire égale à 0,5. En conséquence, 2016 serait représenté par 4032 dans un mot de 16 bits. Pour placer cette valeur dans la plage requise, il faut effectuer un décalage à gauche de 2 bits:

$$S = ALAW_SAMPLE \ll 2$$

On trouvera ci-après le pseudo-code à virgule fixe pour le bloc 4, filtrage du signal de parole d'entrée par le filtre de pondération perceptive.

```
For K = 1, 2, ..., IDIM, do the following
  AA0 = S(K)
  AA0 = AA0 << 14
  For J = LPCW, LPCW - 1, ..., 3, 2, do the next 2 lines
    AA0 = AA0 + WFIR(J) * AWZ(J + 1)      | Section tous zéros
    WFIR(J) = WFIR(J - 1)                    | du filtre

  AA0 = AA0 + WFIR(1) * AWZ(2)           | Traiter le dernier
  WFIR(1) = S(K)                             | vecteur différemment
```

For J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines

AA0 = AA0 – WIIR(J) * AWP(J + 1)	Section tous pôles
WIIR(J) = WIIR(J – 1)	du filtre
AA0 = AA0 – WIIR(1) * AWP(2)	Traiter le dernier
AA0 = AA0 >> 14	vecteur différemment
If AA0 > 32767, set AA0 = 32767	Mode saturation pour l'entrée
If AA0 < –32768, set AA0 = –32768	du multiplicateur ultérieurement
WIIR(1) = AA0	Sauvegarde du mot inférieur de 16 bits
SW(K) = AA0	SW est en Q2

Repeat the above for the next K

G.3.2 Blockzir – Pseudo-code pour les filtres de synthèse et de pondération perceptive pendant le calcul de la réponse à entrée nulle

On trouvera ci-après le pseudo-code à virgule flottante pour le bloc 9 (filtre de synthèse) utilisé pendant le calcul de la réponse en entrée nulle.

For K = 1, 2, ..., IDIM, do the following

TEMP(K) = 0.

For J = LPC, LPC – 1, ..., 3, 2, do the next 2 lines

TEMP(K) = TEMP(K) – STATELPC(J) * A(J + 1)	Multiplier – ajouter
STATELPC(J) = STATELPC(J – 1)	Décalage mémoire
TEMP(K) = TEMP(K) – STATELPC(1) * A(2)	Traiter le dernier
STATELPC(1) = TEMP(K)	vecteur différemment

Repeat the above for the next K

Ci-après, le pseudo-code à virgule flottante pour le bloc 10 (filtre de pondération perceptive) utilisé pendant le calcul de la réponse en entrée nulle.

For K = 1, 2, ..., IDIM, do the following

TMP = TEMP(K)

For J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines

TEMP(K) = TEMP(K) + ZIRWFIR(J) * AWZ(J + 1)	Section tous zéros
ZIRWFIR(J) = ZIRWFIR(J – 1)	du filtre
TEMP(K) = TEMP(K) + ZIRWFIR(1) * AWZ(2)	Traiter le dernier vecteur
ZIRWFIR(1) = TMP	
For J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines	
TEMP(K) = TEMP(K) – ZIRWIIR(J) * AWP(J + 1)	Section tous pôles
ZIRWIIR(J) = ZIRWIIR(J – 1)	du filtre
ZIR(K) = TEMP(K) – ZIRWIIR(1) * AWP(2)	Traiter le dernier
ZIRWIIR(1) = ZIR(K)	vecteur différemment

Repeat the above for the next K

Dans le code à virgule fixe, on notera que STATELPC correspond à la virgule flottante de bloc segmenté et qu'il est associé à NLSSTATE. Comme on est en entrée nulle, il est inutile de mettre NLSSTATE en correspondance avec le nombre NLS de l'entrée. Les valeurs de A(), AWZ() et AWP() sont toujours représentées en format Q14.

On trouvera ci-après le pseudo-code à virgule fixe pour le bloc 9 (filtre de synthèse) utilisé pendant le calcul de la réponse en entrée nulle.

NLSSTATE(11) = NLSSTATE(1)

For K = 2, 3, 4, ..., 10, do the next line | Trouver le minimum de NLSSTATE

If NLSSTATE(K) < NLSSTATE(11), set NLSSTATE(11) = NLSSTATE(K)

For K = 1, 2, ..., IDIM, do the following

I = 1

```

L = 6 - K
J = LPC
AA0 = 0
For LL = 1, ..., L, do the next 3 lines
    AA0 = AA0 - STATELPC(J) * A(J + 1)           | Multiplier – ajouter
    STATELPC(J) = STATELPC(J - 1)               | Décalage mémoire
    J = J - 1
NLS = NLSSTATE(I) - NLSSTATE(11)
AA1 = AA0 >> NLS

For I = 2, ..., 10, do the next 8 lines
    AA0 = 0
    For LL = 1, 2, ..., IDIM, do the next 3 lines
        AA0 = 0 - STATELPC(J) * A(J + 1)
        STATELPC(J) = STATELPC(J - 1)           | STATELPC(0) = parasites si J = 1; c'est OK
        J = J - 1
    NLS = NLSSTATE(I) - NLSSTATE(11)
    AA0 = AA0 >> NLS                             | Décaler pour aligner
    AA1 = AA1 + AA0

If K = 1, go to SHIFT2
L = K - 1
AA0 = 0
For LL = 1, 2, ..., L, do the next 3 lines
    AA0 = AA0 - STATELPC(J) * A(J + 1)
    STATELPC(J) = STATELPC(J - 1)             | STATELPC(0) = parasites si J = 1; c'est OK
    J = J - 1
AA1 = AA1 + AA0                                 | Aucun décalage nécessaire pour cette fois

SHIFT2: AA1 = AA1 >> 14                         | A() était en Q14, NLS de AA1
                                                | est maintenant NLSSTATE(11)
If AA1 > 32767, set AA1 = 32767                 | Ecrêter à 16 bits si nécessaire, car
If AA1 < -32768, set AA1 = -32768              | STATELPC(1) sera l'entrée du multiplicateur

STATELPC(1) = AA1                             | Sauvegarder le mot de 16 bits inférieur pour
                                                | STATELPC
IR = NLSSTATE(11) - 2                          | Mettre TEMP au format Q2
If IR > 0, set AA1 = AA1 >> IR                  |
If IR < 0, set AA1 = AA1 << -IR                |
TEMP(K) = AA1

Repeat the above for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 13, STATELPC, NLS)
NLSSTATE(11) = NLSSTATE(11) + NLS              | Renormaliser le nouveau STATELPC à 15 bits

For L = 1, 2, ..., 10, do the next line        | Mettre à jour NLSSTATE
    NLSSTATE(L) = NLSSTATE(L + 1)

Dans le pseudo-code à virgule fixe pour le bloc 10, TEMP, ZIRWFIR et ZIRWIIR sont en format Q2. Dans le bloc
précédent, TEMP a été créé explicitement avec cette valeur. Il est donc inutile de normaliser pour les additionner. On
trouvera ci-après le pseudo-code à virgule fixe pour le bloc 10 (filtre de pondération perceptive) utilisé pendant le calcul
de la réponse en entrée nulle.

For K = 1, 2, ..., IDIM, do the following
    AA0 = TEMP(K) << 14                         | Parce que AWZ est en format Q14
    For J = LPCW, LPCW - 1, ..., 3, 2, do the next 2 lines
        AA0 = AA0 + ZIRWFIR(J) * AWZ(J + 1)     | Section tous zéros
        ZIRWFIR(J) = ZIRWFIR(J - 1)           | du filtre

    AA0 = AA0 + ZIRWFIR(1) * AWZ(2)            | Traiter le dernier vecteur
    ZIRWFIR(1) = TEMP(K)                       | différemment

```

For J = LPCW, LPCW - 1, ..., 3, 2, do the next 2 lines

AA0 = AA0 - ZIRWIIR(J) * AWP(J + 1)	Section tous
ZIRWIIR(J) = ZIRWIIR(J - 1)	pôles du filtre

AA0 = AA0 - ZIRWIIR(1) * AWP(2)	Traiter le dernier
AA0 = AA0 >> 14	vecteur différencement

If AA0 > 32767, set AA0 = 32767	Ecrêter car ZIR et ZIRWIIR
If AA0 < -32768, set AA0 = -32768	seront des entrées du multiplicateur

ZIR(K) = AA0	Sauvegarder le mot de 16 bits
ZIRWIIR(1) = AA0	inférieur pour ZIR et ZIRWIIR

Repeat the above for the next K

G.3.3 Blocs 9 et 10 – Pseudo-code pour la mise à jour de la mémoire des filtres de synthèse et de pondération perceptive

On trouvera ci-après le pseudo-code à virgule flottante pour les blocs 9 et 10 (mise à jour de la mémoire des filtres).

ZIRWFIR(1) = ET(1)	ZIRWFIR devient une table de travail
TEMP(1) = ET(1)	
For K = 2, 3, ..., IDIM, do the following	
A0 = ET(K)	
A1 = 0	
A2 = 0	
For I = K, K - 1, ..., 2, do the next 5 lines	
ZIRWFIR(I) = ZIRWFIR(I - 1)	
TEMP(I) = TEMP(I - 1)	
A0 = A0 - A(I) * ZIRWFIR(I)	Calculer les réponses d'état zéro
A1 = A1 + AWZ(I) * ZIRWFIR(I)	à divers étages
A2 = A2 - AWP(I) * TEMP(I)	du filtre en cascade
ZIRWFIR(1) = A0	
TEMP(1) = A0 + A1 + A2	

Repeat the above indented section for the next K

	Maintenant mettre à jour la mémoire du filtre
	en ajoutant les réponses d'état zéro
	aux réponses à entrée nulle
For K = 1, 2, ..., IDIM, do the next 4 lines	
STATELPC(K) = STATELPC(K) + ZIRWFIR(K)	
If STATELPC(K) > MAX, set STATELPC(K) = MAX	Limitation de plage
If STATELPC(K) < MIN, set STATELPC(K) = MIN	
ZIRWIIR(K) = ZIRWIIR(K) + TEMP(K)	
For I = 1, 2, ..., LPCW, do the next line	Maintenant positionner ZIRWFIR
ZIRWFIR(I) = STATELPC(I)	à la valeur qui convient
I = IDIM + 1	
For K = 1, 2, ..., IDIM, do the next line	Obtenir le signal de parole quantifié en
ST(K) = STATELPC(I - K)	inversant l'ordre de la mémoire du filtre
	de synthèse

On trouvera plus loin le pseudo-code à virgule fixe pour les mêmes blocs. STATELPC possède 10 exposants qui sont stockés dans NLSSTATE(1), ..., NLSSTATE(10). NLSET est associé à la table ET. Après la mise à jour, ZIRWIIR et ZIRWFIR se trouvent en format Q2. Au départ, ZIRWFIR est utilisé comme table de travail. Après l'entrée dans ce code, ET et les 5 éléments supérieurs de STATELPC [STATELPC(1) à STATELPC(5)] sont des tables de 15 bits avec virgule flottante de bloc. Lorsque ET est filtré par le filtre de synthèse LPC sans mémoire, la sortie (à savoir la réponse d'état zéro du filtre LPC) peut dépasser la plage de 15 bits. Lorsque cette situation survient, on décale ET de 1 bit à droite et on répète le calcul jusqu'à ce que la sortie tienne dans 15 bits. On a constaté empiriquement que ce processus se répète

plus 3 fois (ou 4 fois si l'on compte le premier temps de passage). A noter qu'il y a seulement 10 opérations de multiplication-addition pour chaque répétition du calcul; la raison en est que le calcul de la réponse d'état zéro du filtre de pondération a été transféré sur une boucle séparée. La réponse d'état zéro du filtre LPC, calculée comme il vient d'être expliqué, peut toujours être représentée avec 15 bits ou moins. Cette réponse est ensuite ajoutée au STATELPC de 15 bits pour la mise à jour de STATELPC; il est garanti que le résultat de cette addition peut toujours être représenté avec 16 bits. Avant de sortir ce code, STATELPC est normalisé à 14 bits, le but étant de prévenir des débordements dans le calcul ultérieur de la réponse à entrée nulle.

LABEL1: ZIRWFIR(1) = ET(1)	Calculer d'abord la réponse d'état zéro du filtre de synthèse LPC
For K = 2, 3, ..., IDIM, do the following indented lines	
AA0 = ET(K) << 14	Parce que A(1) = 1 en Q14 = 16384
For I = K, K - 1, ..., 2, do the next 3 lines	
ZIRWFIR(I) = ZIRWFIR(I - 1)	
P = A(I) * ZIRWFIR(I)	Multiplication de Q14
AA0 = AA0 - P	Calculer les réponses d'état zéro
AA1 = AA0 << 3	
If AA1 overflowed above, do the next 4 lines	S'assurer que, après AA0 >> 14,
For I = 1, 2, ..., IDIM, do the next line	le résultat ne dépasse pas 15 bits.
ET(I) = ET(I) >> 1	En cas de dépassement, faire ET >> 1
NLSET = NLSET - 1	et répéter le calcul jusqu'à ce que le
GO TO LABEL1	résultat tienne dans 15 bits
AA0 = AA0 >> 14	Compensation du fait que A() est en Q14
ZIRWFIR(1) = AA0	Conserver les 16 bits inférieurs
Repeat the above indented section for the next K	
N = IDIM + 1	Calculer maintenant la réponse d'état zéro
TEMP(1) = ZIRWFIR(IDIM)	du filtre de pondération
For K = 2, 3, ..., IDIM, do the following indented lines	
AA1 = ZIRWFIR(N - K) << 14	Parce que AWZ(1) = 1 (en Q14 = 16384)
M = IDIM - K	
For I = K, K - 1, ..., 2, do the next 5 lines	
TEMP(I) = TEMP(I - 1)	Décaler la section tous pôles de la mémoire
P = AWZ(I) * ZIRWFIR(I + M)	du filtre. Section tous zéros du filtre de pondération
AA1 = AA1 + P	
P = AWP(I) * TEMP(I)	Section tous pôles du filtre de pondération
AA1 = AA1 - P	
AA1 = AA1 >> 14	
If AA1 > 32767, set AA1 = 32767	Ecrêter si nécessaire, car TEMP(1) sera
If AA1 < -32768, set AA1 = -32768	l'entrée de 16 bits du multiplicateur
TEMP(1) = AA1	Conserver les 16 bits inférieurs
Repeat the above indented section for the next K	
IR = NLSET - 2	
For K = 1, ..., IDIM, do the next 2 lines	Décaler maintenant TEMP à Q2 comme
If IR > 0, set TEMP(K) = TEMP(K) >> IR	ZIRWIIR
If IR < 0, set TEMP(K) = TEMP(K) << -IR	
	Maintenant mettre à jour la mémoire du filtre
	en ajoutant les réponses d'état zéro aux
	réponses à entrée nulle. Il faut tout d'abord
	adapter les NLS de ZIRWFIR et STATELPC
If NLSET = NLSSTATE(10), go to LABEL2	Aucun changement nécessaire

<p>If NLSET < NLSSTATE(10), do the next 5 lines NLSD = NLSSTATE(10) – NLSET For K = 1, 2, ..., IDIM, do the next line STATELPC(K) = STATELPC(K) >> NLSD NLSSTATE(10) = NLSET go to LABEL2</p> <p>NLSD = NLSET – NLSSTATE(10) For K = 1, 2, ..., IDIM, do the next line ZIRWFIR(K) = ZIRWFIR(K) >> NLSD</p>	<p> Perte de précision sur STATELPC (perte = bits de NLSD)</p> <p> Seul cas restant: NLSET > NLSSTATE Perte de précision sur ZIRWFIR (perte = bits de NLSD)</p>
<p>LABEL2: AA1 = 4095 If NLSSTATE(10) ≥ 0, set AA1 = AA1 << NLSSTATE(10) If NLSSTATE(10) < 0, set AA1 = AA1 >> –NLSSTATE(10)</p>	<p> Maintenant nous sommes prêts 4095 = niveau d'écrtage de STATELPC Décaler le niveau d'écrtage pour aligner avec STATELPC</p>
<p>For K = 1, 2, ..., IDIM, do the following indented lines AA0 = STATELPC(K) + ZIRWFIR(K) If AA0 > AA1, set AA0 = AA1 If AA0 < –AA1, set AA0 = –AA1</p> <p>If AA0 > 32767, set AA0 = 32767 If AA0 < –32768, set AA0 = –32768 STATELPC(K) = AA0</p> <p>AA0 = ZIRWIIR(K) + TEMP(K) If AA0 > 32767, set AA0 = 32767 If AA0 < –32768, set AA0 = –32768 ZIRWIIR(K) = AA0</p>	<p> Mettre à jour la mémoire du filtre LPC. Si nécessaire, effectuer l'écrtage comme indiqué dans G.728 pour la virgule flottante. A noter que ces valeurs ont été normalisées. Ainsi, si 32767 < AA0 < AA1, il faut écrtéer AA0 à 16 bits parce que STATELPC(K) sera ultérieurement une entrée de 16 bits pour le multiplicateur</p> <p> Mettre à jour la section tous pôles de la mémoire de W(z). Ici encore, écrtéer à 16 bits si nécessaire, parce que ZIRWIIR(K) sera ultérieurement une entrée de 16 bits pour le multiplicateur</p>
<p>Repeat the above indented section for the next K</p>	
<p>Call VSCALE(STATELPC, IDIM, IDIM, 12, STATELPC, NLS) NLSSTATE(10) = NLSSTATE(10) + NLS</p>	<p> Normaliser STATELPC à 14 bits pour prévenir des débordements dans le calcul ultérieur de la réponse à entrée nulle</p>
<p>IR = NLSSTATE(10) – 2</p>	
<p>For I = 1, 2, ..., 5, do the next 4 lines AA0 = STATELPC(I) If IR > 0, set AA0 = AA0 >> IR If IR < 0, set AA0 = AA0 << –IR ZIRWFIR(I) = AA0 IR = NLSSTATE(9) – 2 For I = 6, 7, ..., 10, do the next 4 lines AA0 = STATELPC(I) If IR > 0, set AA0 = AA0 >> IR If IR < 0, set AA0 = AA0 << –IR ZIRWFIR(I) = AA0</p>	<p> Maintenant positionner ZIRWFIR, section tous zéros de la mémoire W(z), sur les valeurs qui conviennent en format Q2 </p>
<p>I = IDIM + 1 For K = 1, 2, ..., IDIM, do the next line ST(K) = STATELPC(I – K) NLSST = NLSSTATE(10)</p>	<p> On obtient le signal de parole quantifié en inversant l'ordre des 5 positions supérieures de la mémoire du filtre de synthèse NLSST est utilisé seulement dans le décodeur</p>

G.3.4 Bloc 11 – Calcul du vecteur quantifié cible

Ci-après, le pseudo-code à virgule flottante pour le bloc 11 (calcul du vecteur quantifié cible).

```
For K = 1, 2, ..., IDIM, do the next line
    TARGET(K) = SW(K) – ZIR(K)
```

Pour le code à virgule fixe, SW et ZIR sont l'un et l'autre en format Q2, comme dans le signal de parole d'entrée. Donc: NLSTARGET = 2. Ci-après, le pseudo-code à virgule fixe.

```
set NLSTARGET = 2
```

```
For K = 1, 2, ..., IDIM, do the next 6 lines
    AA0 = SW(K)
    AA1 = ZIR(K)
    AA0 = AA0 – AA1
    If AA0 > 32767, set AA0 = 32767 | Ecrêter si nécessaire
    If AA0 < –32768, set AA0 = –32768
    TARGET(K) = AA0
```

G.3.5 Bloc 12 – Calcul du vecteur de réponse impulsionnelle

Ci-après, le pseudo-code à virgule flottante pour le bloc 12.

```
TEMP(1) = 1 | TEMP = mémoire du filtre de synthèse
WS(1) = 1 | WS = mémoire de la section tous pôles de W(z)
For K = 2, 3, ..., IDIM, do the following
    A0 = 0
    A1 = 0
    A2 = 0
    For I = K, K – 1, ..., 3, 2, do the next 5 lines
        TEMP(I) = TEMP(I – 1)
        WS(I) = WS(I – 1) |
        A0 = A0 – A(I) * TEMP(I) | Filtrage
        A1 = A1 + AWZ(I) * TEMP(I) |
        A2 = A2 – AWP(I) * WS(I)

    TEMP(1) = A0
    WS(1) = A0 + A1 + A2
```

Repeat the above indented section for the next K

```
ITMP = IDIM + 1 | Obtenir h(n) en inversant l'ordre de la mémoire
For K = 1, 2, ..., IDIM, do the next line | de la section tous pôles de W(z)
    H(K) = WS(ITMP – K) |
```

Les valeurs des coefficients du prédicteur, A(), AWZ() et AWP() sont toutes stockées en format Q14. Dans le pseudo-code à virgule fixe spécifié ci-après, seuls sont indiqués deux accumulateurs de 32 bits, AA0 et AA1. Dans le pseudo-code à virgule flottante, les accumulateurs A1 et A2 étaient combinés. Les bits de garde sont inutiles. La table de sortie, H(), est stockée en format Q13. Ci-après, le pseudo-code à virgule fixe.

```
TEMP(1) = 8192 | TEMP = mémoire du filtre de synthèse
WS(1) = 8192 | WS = mémoire de la section tous pôles de W(z)
| WS & TEMP sont des mots de 16 bits en Q13

For K = 2, 3, ..., IDIM, do the following
    AA0 = 0
    AA1 = 0
    For I = K, K – 1, ..., 3, 2, do the next 5 lines
        TEMP(I) = TEMP(I – 1)
        WS(I) = WS(I – 1) |
        AA0 = AA0 – A(I) * TEMP(I) | Filtrage
        AA1 = AA1 + AWZ(I) * TEMP(I) |
        AA1 = AA1 – AWP(I) * WS(I)
```

```

AA1 = AA0 + AA1
AA0 = AA0 >> 14
AA1 = AA1 >> 14
TEMP(1) = AA0
WS(1) = AA1

```

| >> 14 parce que A(), AWZ() et
| AWP() étaient en Q14

Repeat the above indented section for the next K

```

ITMP = IDIM + 1
For K = 1, 2, ..., IDIM, do the next line
    H(K) = WS(ITMP - K)

```

| Obtenir h(n) en inversant l'ordre de la
| mémoire de la section tous pôles de W(z)
|

G.3.6 Bloc 13 – Convolution à inversion temporelle

Ce module effectue la convolution en inversion temporelle, pour préparer la recherche dans le répertoire d'excitation. Le pseudo-code de départ à virgule flottante était le suivant:

```

For K = 1, 2, ..., IDIM, do the following
    K1 = K - 1
    PN(K) = 0
    For J = K, K + 1, ..., IDIM, do the next line
        PN(K) = PN(K) + TARGET(J) * H(J - K1)

```

Repeat the above for the next K

Dans la version à virgule fixe, H() est représenté en format Q13 et TARGET est représenté avec virgule flottante de bloc. NLSTARGET est déterminé en bloc 16. NLSPN est fixé à 7.

```

For K = 1, 2, ..., IDIM, do the following
    K1 = K - 1
    AA0 = 0
    For J = K, K + 1, ..., IDIM, do the next 2 lines
        P = TARGET(J) * H(J - K1)
        AA0 = AA0 + P
    AA0 = AA0 >> 13 + (NLSTARGET - 7)
    If AA0 > 32767, set AA0 = 32767
    If AA0 < -32768, set AA0 = -32768
    PN(K) = AA0

```

| Mise à zéro de l'accumulateur
| Décalage à droite pour faire Q7
| Ecrêter AA0 à 16 bits parce que PN
| sera une entrée du multiplicateur
| AA0 est en mode saturation

Repeat the above for the next K

G.3.7 Bloc 14 – Convolution des vecteurs code et calcul de l'énergie

Ci-après, le pseudo-code pour le calcul d'énergie des vecteurs code (blocs 14 et 15).

```

For J = 1, 2, ..., NCWD, do the following
    J1 = (J - 1) * IDIM
    For K = 1, 2, ..., IDIM, do the next 4 lines
        K1 = J1 + K + 1
        TEMP(K) = 0
        For I = 1, 2, ..., K, do the next line
            TEMP(K) = TEMP(K) + H(I) * Y(K1 - I)

```

| Un seul vecteur code par boucle
| Convolution

Repeat the above 4 lines for the next K

```

Y2(J) = 0
For K = 1, 2, ..., IDIM, do the next line
    Y2(J) = Y2(J) + TEMP(K) * TEMP(K)

```

| Calcul de l'énergie

Repeat the above for the next J

Dans le pseudo-code à virgule fixe, $H()$ est représenté en format Q13 et $Y()$ en format Q11. On a découvert empiriquement que, après convolution de $H()$ et $Y()$ il n'y a pas de débordements dans l'accumulateur (résultat d'amplitude supérieure à 2^{*31}). En conséquence, le pseudo-code suivant ne prévoit pas de test de débordement dans l'accumulateur. Cela permet un déroulement plus rapide du code DSP correspondant. Le décalage subséquent de 14 bits est nécessaire pour permettre la représentation de $TEMP()$ en format Q10. On a constaté que les représentations utilisées fonctionnent avec une variété de fichiers. $NLSY2 = (NLSH + NLSY - 14) * 2 - 15$. Puisque $NLSY = 11$ et $NLSH = 13$, on a $NLSY2 = 5$.

```

For J = 1, 2, ..., NCWD, do the following | Un seul vecteur code par boucle
  J1 = (J - 1) * IDIM
  For K = 1, 2, ..., IDIM, do the next 7 lines
    K1 = J1 + K + 1
    AA0 = 0
    For I = 1, 2, ..., K, do the next 2 lines |
      P = H(I) * Y(K1 - I) | Convolution
      AA0 = AA0 + P |

    AA0 = AA0 >> 14
    TEMP(K) = AA0 | Seulement les 16 bits inférieurs

Repeat the above 7 lines for the next K

AA0 = 0
For K = 1, 2, ..., IDIM, do the next 2 lines
  P = TEMP(K) * TEMP(K) | Calcul de l'énergie
  AA0 = AA0 + P
AA0 = AA0 >> 15
Y2(J) = AA0 | Seulement les 16 bits inférieurs

```

Repeat the above for the next J

G.3.8 Bloc 16 – Normalisation du vecteur cible VQ

Ci-après, le pseudo-code à virgule flottante pour ce module.

```

TMP = 1./GAIN
For K = 1, 2, ..., IDIM, do the next line
  TARGET(K) = TARGET(K) * TMP

```

Pour le pseudo-code à virgule fixe, il faut considérer le NLS du gain et le NLS de TARGET. A l'entrée, on a $NLSTARGET = 2$. En cours de processus, on créera le NLS pour TMP.

```

| Numérateur pour la division = 16384
| NLS = 14 pour le numérateur
| NLS pour le dénominateur = NLSGAIN
| NLSGAIN est déterminé dans le bloc 46

```

Call DIVIDE(16384, 14, GAIN, NLSGAIN, TMP, NLSTMP)

```

For K = 1, 2, ..., IDIM, do the next 2 lines
  AA0 = TMP * TARGET(K) | AA0 se compose de 32 bits
  TARGET(K) = AA0 >> 15 | Conserver seulement les 16 bits inférieurs
| TARGET est en Q2 à ce point

```

```

NLSTARGET = 2 + NLSTMP - 15 | Positionner TARGET pour bloquer la virgule
| flottante

```

```

Call VSCALE(TARGET, IDIM, IDIM, 14, TARGET, NLS)
NLSTARGET = NLSTARGET + NLS | NLS a été changé en VSCALE

```

G.3.9 Bloc 17 – Calculateur d'erreur sur la recherche de VQ et sélecteur du meilleur index du répertoire d'excitation

Ci-après, le pseudo-code à virgule flottante pour le calculateur d'erreur et le sélecteur du meilleur index du répertoire d'excitation (blocs 17 et 18).

```

Initialize DISTM to the largest number representable in the hardware
N1 = NG/2
For J = 1, 2, ..., NCWD, do the following
  J1 = (J - 1) * IDIM
  COR = 0
  For K = 1, 2, ..., IDIM, do the next line
    COR = COR + PN(K) * Y(J1 + K) | Calculer le produit interne Pj

  If COR > 0, then do the next 5 lines
    IDXG = N1
    For K = 1, 2, ..., N1 - 1, do the next "if" statement
      If COR < GB(K) * Y2(J), do the next 2 lines
        IDXG = K | Meilleur gain positif trouvé
        GO TO LABEL

  If COR ≤ 0, then do the next 5 lines
    IDXG = NG
    For K = N1 + 1, N1 + 2, ..., NG - 1, do the next "if" statement
      If COR > GB(K) * Y2(J), do the next 2 lines
        IDXG = K | Meilleur gain négatif trouvé
        GO TO LABEL

LABEL: D = -G2(IDXG) * COR + GSQ(IDXG) * Y2(J) | Calcul de la distorsion  $\hat{D}$ 
  If D < DISTM, do the next 3 lines
    DISTM = D | Sauvegarde de la plus faible distorsion
    IG = IDXG | et des meilleurs index du répertoire
    IS = J | d'excitation trouvés jusqu'à maintenant

Repeat the above inserted section for the next J

ICHAN = (IS - 1) * NG + (IG - 1)

```

On trouvera ci-après le pseudo-code à virgule fixe pour le calculateur d'erreur et le sélecteur du meilleur index du répertoire d'excitation (blocs 17 et 18). Ce code a été écrit de telle sorte que la valeur absolue de la corrélation avec la valeur cible – AA0 ci-dessous – soit utilisée pour la recherche du gain, après quoi il y a nouvelle détermination du signe de la corrélation. Il peut sembler que cela crée un surcroît de travail, mais on évite ainsi d'avoir un embranchement au milieu de la boucle de recherche. Pour la plupart des formes de mise en œuvre du traitement numérique des signaux ou traitement DSP (*digital signal processing*), le fait d'éviter la création d'un embranchement permet d'économiser des instructions. Une autre solution consiste à sauvegarder le signe et à faire l'économie d'un nouveau calcul. Mais cette solution se paie généralement par une instruction supplémentaire dans la boucle de recherche.

```

DISTM = 2147483647
For J = 1, 2, ..., NCWD, do the following
  J1 = (J - 1) * IDIM
  AA0 = 0
  For K = 1, 2, ..., IDIM, do the next 2 lines
    P = PN(K) * Y(J1 + K) | Calculer le produit interne Pj
    AA0 = AA0 + P | Le NLS pour AA0 est 7 + 11 = 18
  If AA0 < 0, set AA0 = - AA0 | Prendre la valeur absolue

  IDXG = 1
  P = GB(1) * Y2(J) | Le NLS pour P est 13 + 5 = 18
  If AA0 ≥ P, set IDXG = IDXG + 1
  P = GB(2) * Y2(J)
  If AA0 ≥ P, set IDXG = IDXG + 1
  P = GB(3) * Y2(J)
  If AA0 ≥ P, set IDXG = IDXG + 1

```

AA0 = AA0 >> 14	Le NLS pour AA0 = 4
If AA0 > 32767, set AA0 = 32767	Ecrêter AA0; AA0 est en mode saturation
AA1 = GSQ(IDXG) * Y2(J)	NLSGSQ = 11, NLSY2 = 5, d'où NLSAA1 = 16
P = G2(IDXG) * AA0	NLSG2 = 12, NLSAA0 = 4, d'où NLSP = 16

AA1 = AA1 - P	
If AA1 < DISTM, do the next 3 lines	
DISTM = AA1	DISTM à double précision
IG = IDXG	
IS = J	

Repeat the above inserted section for the next J

AA0 = 0	Trouver maintenant le bit de signe
J1 = (IS - 1) * IDIM	
For K = 1, 2, ..., IDIM, do the next 2 lines	
P = PN(K) * Y(J1 + K)	Calculer le produit interne
AA0 = AA0 + P	
If AA0 ≤ 0, set IG = IG + 4	

ICHAN = (IS - 1) * NG + (IG - 1)

Dans le code ci-dessus, on a utilisé les quatre lignes suivantes:

AA0 = AA0 >> 14	NLS pour AA0 = 4
If AA0 > 32767, set AA0 = 32767	Ecrêter AA0
AA1 = GSQ(IDXG) * Y2(J)	NLSGSQ = 11, NLSY2 = 5, d'où NLSAA1 = 16
P = G2(IDXG) * AA0	NLSG2 = 12, NLSAA0 = 4, d'où NLSP = 16

Dans les puces DSP qui ont une fonction d'«écrêtage», ces lignes peuvent être remplacées par le code suivant, qui donne exactement les mêmes résultats:

AA0 = AA0 << 2	NLS pour AA0 = 20
AA0 = CLIP(AA0)	AA0 est en mode saturation
AA0 = AA0 >> 16	Prendre le mot supérieur; NLS pour AA0 = 4
AA1 = GSQ(IDXG) * Y2(J)	NLSGSQ = 11, NLSY2 = 5, d'où NLSAA1 = 16
P = G2(IDXG) * AA0	NLSG2 = 12, NLSAA0 = 4, d'où NLSP = 16

La fonction CLIP et le mode saturation correspondent au principe selon lequel AA0 ne doit pas déborder pendant l'opération << 2. Au lieu de déborder, AA0 est positionné sur le nombre positif ou négatif maximal, selon son signe au départ. Dans ce cas, AA0 est toujours positif. Cette solution, qui dépend du traitement DSP, peut nécessiter plus qu'un accumulateur à 32 bits. La solution mettant en œuvre le pseudo-code principal peut toujours être mise en œuvre.

G.3.10 Bloc 19 – Répertoire des vecteurs d'excitation quantifiés et bloc 21 – Module de normalisation du gain

Ci-après, la version à virgule flottante du pseudo-code pour le bloc 19 (répertoire des vecteurs d'excitation):

```

NN = (IS - 1) * IDIM
For K = 1, 2, ..., IDIM, do the next line
    YN(K) = GQ(IG) * Y(NN + K)

```

Ci-après, la version à virgule flottante du pseudo-code pour le bloc 21 (module de normalisation du gain):

```

For K = 1, 2, ..., IDIM, do the next line
    ET(K) = GAIN * YN(K)

```

Pour le pseudo-code à virgule fixe, on associe les deux blocs 19 et 21 pour former un seul module. Y et GQ ont des formats Q fixes, à savoir respectivement Q11 et Q13. NLSGAIN est associé à la valeur de GAIN. Pour obtenir la précision maximale, on normalise le produit $GQ(IG) * GAIN$ à 32 bits avant d'arrondir aux 16 bits supérieurs. Posons: $NNGQ(I) = [1 + \text{le nombre de décalages à gauche nécessaires pour normaliser le } Q13 \text{ } GQ(I)]$. On a alors: $NNGQ(I) = 3$ pour $I = 1, 2, 5, 6$, $NNGQ(I) = 2$ pour $I = 3, 7$, et $NNGQ(I) = 1$ pour $I = 4, 8$. Cela étant, on peut écrire le pseudo-code comme suit:

```

AA0 = GQ(IG) * GAIN
AA0 = AA0 << NNGQ(IG)
TMP = RND(AA0)
NLSAA0 = 13 + NLSGAIN
NLSTMP = NLSAA0 + NNGQ(IG) - 16
NN = (IS - 1) * IDIM
Call VSCALE(Y(NN + 1), IDIM, IDIM, 14, TEMP, NLS)
For K = 1, 2, ..., IDIM, do the next 2 lines
    AA0 = TMP * TEMP(K)
    ET(K) = RND(AA0)
NLSET = NLSTMP + 11 + NLS - 16

```

| AA0 contient NNGQ(IG) avec des zéros en tête
| Décalage à gauche des bits de NNGQ(IG) pour normaliser AA0
| Arrondir aux 16 bits supérieurs et attribuer à TMP
| Format Q du produit GQ(IG) * GAIN
| Format Q de TMP, parce que AA0 est décalé à gauche du nombre de bits de NNGQ(IG); ensuite, arrondir et prendre les 16 bits supérieurs
| Normaliser le vecteur code de forme choisi à 16 bits; ensuite, mettre dans TEMP
| TMP et TEMP sont normalisés à 16 bits; le produit a donc 1 zéro en tête. En arrondissant directement au mot supérieur, on obtient une table ET de 15 bits
| Calculer le NLS pour ET

G.3.11 Bloc 32 – Filtre de synthèse du décodeur

On trouvera ci-après le pseudo-code à virgule flottante, pour le bloc 32 (filtre de synthèse du décodeur).

```

For K = 1, 2, ..., IDIM, do the next 6 lines
    TEMP(K) = 0
    For J = LPC, LPC - 1, ..., 3, 2, do the next 2 lines
        TEMP(K) = TEMP(K) - STATELPC(J) * A(J + 1)
        STATELPC(J) = STATELPC(J - 1)
    TEMP(K) = TEMP(K) - STATELPC(1) * A(2)
    STATELPC(1) = TEMP(K)
Repeat the above for the next K
TEMP(1) = ET(1)
For K = 2, 3, ..., IDIM, do the next 5 lines
    A0 = ET(K)
    For I = K, K - 1, ..., 2, do the next 2 lines
        TEMP(I) = TEMP(I - 1)
        A0 = A0 - A(I) * TEMP(I)
    TEMP(1) = A0
Repeat the above 5 lines for the next K
For K = 1, 2, ..., IDIM, do the next 3 lines
    STATELPC(K) = STATELPC(K) + TEMP(K)
    If STATELPC(K) > MAX, set STATELPC(K) = MAX
    If STATELPC(K) < MIN, set STATELPC(K) = MIN
I = IDIM + 1
For K = 1, 2, ..., IDIM, do the next line
    ST(K) = STATELPC(I - K)

```

| Réponse à entrée nulle
| Traiter le dernier vecteur différemment
| Maintenant mettre à jour la mémoire en ajoutant les réponses d'état zéro aux réponses à entrée nulle
| ZIR + ZSR
| Limitation de la plage
| Obtention du vecteur de parole quantifié par inversion de l'ordre de la mémoire du filtre de synthèse

La méthodologie suivie pour le bloc 9 est aussi appliquée pour le pseudo-code à virgule fixe du bloc 32, à la différence près qu'il n'y a pas de mise à jour de la mémoire pour le filtre de pondération perceptive.

```

NLSSTATE(11) = NLSSTATE(1)
For K = 2, 3, 4, ..., 10, do the next line | Trouver le NLSSTATE minimal
  If NLSSTATE(K) < NLSSTATE(11), set NLSSTATE(11) = NLSSTATE(K)

For K = 1, 2, ..., IDIM, do the following
  I = 1
  L = 6 - K
  J = LPC
  AA0 = 0
  For LL = 1, ..., L, do the next 3 lines
    AA0 = AA0 - STATELPC(J) * A(J + 1) | Multiplier – ajouter
    STATELPC(J) = STATELPC(J - 1) | Décalage mémoire
    J = J - 1
  NLS = NLSSTATE(I) - NLSSTATE(11)
  AA1 = AA0 >> NLS

  For I = 2, ..., 10, do the next 8 lines
    AA0 = 0
    For LL = 1, 2, ..., IDIM, do the next 3 lines
      AA0 = AA0 - STATELPC(J) * A(J + 1)
      STATELPC(J) = STATELPC(J - 1) | STATELPC(0) = parasites si J = 1; c'est OK
      J = J - 1
    NLS = NLSSTATE(I) - NLSSTATE(11)
    AA0 = AA0 >> NLS | Décaler pour aligner
    AA1 = AA1 + AA0

  If K = 1, go to SHIFT2
  L = K - 1
  AA0 = 0
  For LL = 1, 2, ..., L, do the next 3 lines
    AA0 = AA0 - STATELPC(J) * A(J + 1)
    STATELPC(J) = STATELPC(J - 1) | STATELPC(0) = parasites si J = 1; c'est OK
    J = J - 1
  AA1 = AA1 + AA0 | Aucun décalage nécessaire pour cette fois

SHIFT2: AA1 = AA1 >> 14 | A() était en Q14, NLS de AA1
 | est maintenant NLSSTATE(11)
  If AA1 > 32767, set AA1 = 32767 | Ecrêter à 16 bits si nécessaire, puisque
  If AA1 < -32768, set AA1 = -32768 | STATELPC(1) sera l'entrée du multiplicateur

  STATELPC(1) = AA1 | Sauvegarder le mot inférieur de 16 bits
  IR = NLSSTATE(11) - 2 | pour STATELPC
  If IR > 0, set AA1 = AA1 >> IR | Mettre TEMP en format Q2
  If IR < 0, set AA1 = AA1 << -IR |
  TEMP(K) = AA1

Repeat the above for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 13, STATELPC, NLS)
NLSSTATE(11) = NLSSTATE(11) + NLS | Renormaliser le nouveau STATELPC à 15 bits

For L = 1, 2, ..., 10, do the next line | Mise à jour de NLSSTATE
  NLSSTATE(L) = NLSSTATE(L + 1)

LABEL1: TEMP(1) = ET(1) | Calculer d'abord la réponse d'état zéro
 | du filtre de synthèse LPC
For K = 2, 3, ..., IDIM, do the following indented lines
  AA0 = ET(K) << 14 | Parce que A(1) = 1 en Q14 = 16384
  For I = K, K - 1, ..., 2, do the next 3 lines
    TEMP(I) = TEMP(I - 1)
    P = A(I) * TEMP(I) | Multiplication de Q14
    AA0 = AA0 - P | Calculer les réponses d'état zéro

```

AA1 = AA0 << 3	
If AA1 overflowed above, do the next 4 lines	S'assurer que, après AA0 >> 14,
For I = 1, 2, ..., IDIM, do the next line	le résultat ne dépasse pas 15 bits.
ET(I) = ET(I) >> 1	En cas de dépassement, faire ET >> 1
NLSET = NLSET - 1	et répéter le calcul jusqu'à ce que
GO TO LABEL1	le résultat tienne dans 15 bits
AA0 = AA0 >> 14	Compenser le fait que A() est en Q14
TEMP(1) = AA0	Conserver les 16 bits inférieurs
Repeat the above indented section for the next K	
If NLSET = NLSSTATE(10), go to LABEL2	Aucune modification n'est nécessaire
If NLSET < NLSSTATE(10), do the next 5 lines	Perte de précision sur STATELPC
NLSD = NLSSTATE(10) - NLSET	Perte = nombre de bits de NLSD
For K = 1, 2, ..., IDIM, do the next line	
STATELPC(K) = STATELPC(K) >> NLSD	
NLSSTATE(10) = NLSET	
go to LABEL2	Seul cas restant: NLSET > NLSSTATE
NLSD = NLSET - NLSSTATE(10)	Perte de précision sur TEMP;
For K = 1, 2, ..., IDIM, do the next line	perte = nombre de bits de NLSD
TEMP(K) = TEMP(K) >> NLSD	
LABEL2:	Maintenant nous sommes prêts
AA1 = 4095	4095 = niveau d'écrêtage de STATELPC
If NLSSTATE(10) ≥ 0, set AA1 = AA1 << NLSSTATE(10)	Décaler le niveau d'écrêtage pour
If NLSSTATE(10) < 0, set AA1 = AA1 >> -NLSSTATE(10)	aligner avec STATELPC
For K = 1, 2, ..., IDIM, do the following indented lines	
AA0 = STATELPC(K) + TEMP(K)	Mettre à jour la mémoire du filtre LPC
If AA0 > AA1, set AA0 = AA1	Si nécessaire, effectuer l'écrêtage comme
If AA0 < -AA1, set AA0 = -AA1	indiqué dans G.728 pour la virgule flottante.
	A noter que ces valeurs ont été normalisées
	Ainsi, si 32767 > AA0 < AA1, il faut écrêter
If AA0 > 32767, set AA0 = 32767	AA0 à 16 bits, parce que STATELPC(K)
If AA0 < -32768, set AA0 = -32768	sera ultérieurement une entrée à 16 bits
STATELPC(K) = AA0	du multiplicateur
Repeat the above indented section for the next K	
Call VSCALE(STATELPC, IDIM, IDIM, 12, STATELPC, NLS)	Normaliser STATELPC à 14 bits,
NLSSTATE(10) = NLSSTATE(10) + NLS	pour prévenir un débordement dans
	le calcul ultérieur de la réponse en entrée nulle
I = IDIM + 1	
For K = 1, 2, ..., IDIM, do the next line	Obtention du vecteur de parole quantifié par
ST(K) = STATELPC(I - K)	inversion de l'ordre des 5 positions supérieures
NLSST = NLSSTATE(10)	de la mémoire du filtre de synthèse
	NLSST est utilisé plus tard dans le décodeur

G.3.12 Bloc 36 – Pseudo-code pour le module de fenêtrage hybride

On trouvera dans le présent paragraphe le pseudo-code à virgule flottante et le pseudo-code à virgule fixe pour le bloc 36. Tout d'abord, le pseudo-code à virgule flottante:

N1 = LPCW + NFRSZ	Calculer quelques constantes (qui pourront être
N2 = LPCW + NONRW	précalculées et mises en mémoire)
N3 = LPCW + NFRSZ + NONRW	
For N = 1, 2, ..., N2, do the next line	
SBW(N) = SBW(N + NFRSZ)	Décaler le tampon de l'ancien signal
For N = 1, 2, ..., NFRSZ, do the next line	
SBW(N2 + N) = STMP(N)	Introduire par décalage le nouveau signal
	SBW(N3) est le plus récent échantillon

```

K = 1
For N = N3, N3 - 1, ..., 3, 2, 1, do the next 2 lines
    WS(N) = SBW(N) * WNRW(K) | Multiplier par la fonction de fenêtrage
    K = K + 1

For I = 1, 2, ..., LPCW + 1, do the next 4 lines
    TMP = 0
    For N = LPCW + 1, LPCW + 2, ..., N1, do the next line
        TMP = TMP + WS(N) * WS(N + 1 - I)
    REXPW(I) = (1/2) * REXPW(I) + TMP | Mettre à jour la composante récurrente

For I = 1, 2, ..., LPCW + 1, do the next 3 lines
    R(I) = REXPW(I)
    For N = N1 + 1, N1 + 2, ..., N3, do the next line
        R(I) = R(I) + WS(N) * WS(N + 1 - I) | Ajouter la composante non récurrente

R(1) = R(1) * WNCF | Correction par bruit blanc

```

Ci-dessus, la version à virgule fixe pour le même module. Dans ce code, on a ajouté plusieurs variables nouvelles. NLSREXPW est une variable globale qui renferme le nombre de décalages à gauche nécessaires pour normaliser REXPW. Cette variable est initialisée avec la valeur 31.

```

N1 = LPCW + NFRSZ (= 10 + 20) | Calculer quelques constantes (qui pourront
N2 = LPCW + NONRW (= 10 + 30) | être précalculées et mises en mémoire)
N3 = LPCW + NFRSZ + NONRW (= 10 + 20 + 30)

For N = 1, 2, ..., N2, do the next line
    SBW(N) = SBW(N + NFRSZ) | Décaler le tampon de l'ancien signal
For N = 1, 2, ..., NFRSZ, do the next line
    SBW(N2 + N) = STMP(N) | SBW(N3) est le plus récent échantillon
    | Tous les SBW sont en Q2 et représentés avec
    | une précision de 15 bits

Call FINDNLS(SBW, N3, N3, 14, NLS) | Trouver le nombre de décalages à gauche
    | nécessaires dans la boucle suivante pour
    | obtenir une marge de 2 bits. Il n'est pas
    | vraiment nécessaire de faire la normalisation
    | On utilise simplement NLS

NLSTMP = NLS - 1
K = 1
For N = 60, 59, ..., 1, do the next 4 lines
    P = SBW(N) * WNRW(K) | WNRW est en Q15. Avec un décalage à
    | gauche égal au nombre de bits de NLSWS,
    AA0 = P << NLSWS | le plus grand élément de WS(N) deviendra
    WS(N) = RND(AA0) | un nombre de 14 bits (2 bits de marge pour
    K = K + 1 | accumulation ultérieure)

NLSATTW = 15
Call
HWMCORE(LPCW, N1, N3, NLSATTW, WS, NLSTMP, REXPW, NLSREXPW, R, ILLCONDW)

If NLSREXPW > 41, set NLSREXPW = 41 | Pour éviter une diminution de la précision sur
    | REXPW() et R() dans les cas où le signal
    | d'entrée est nul pendant de longues périodes

```

Le sous-programme HWMCORE est décrit au G.3.18.

Dans le code ci-dessus, un appel à FINDNLS déclenche une recherche dans tout le tampon SBW (60 échantillons). Toutefois, il est possible d'abrégier ce calcul en ayant recours à un substitut exactement identique en bits, qui utilise 2 mots de mémoire supplémentaires. SBW contiendra toujours 40 anciens échantillons et 20 nouveaux. On peut diviser ces échantillons en trois vecteurs de 20 échantillons chacun. On vérifie NLS pour chacun des trois vecteurs et on choisit celui qui a la valeur minimale pour application de la fenêtre hybride. Comme deux des vecteurs sont composés d'anciens échantillons, on connaît déjà leurs NLS respectifs. Il suffit de contrôler le vecteur le plus récent pour trouver son nombre NLS. On met ensuite en mémoire le NLS relatif au vecteur le plus récent et au plus récent des deux anciens vecteurs, pour le prochain calcul. Cette méthode conduit à un NLS qui est exactement le même que dans la procédure indiquée dans le pseudo-code spécifié plus haut.

Pour plus de commodité, on trouvera dans le tableau suivant la liste de toutes les variables du pseudo-code spécifié ci-dessus, avec indication de leur format de représentation et de leur taille. Le tableau indique si chaque variable est 0temporaire (temp) – auquel cas il est inutile de la mettre en mémoire après l'exécution du module – ou permanente (perm), auquel cas on aura encore besoin de la valeur après le calcul en cours. Le tableau indique également les variables qui ne figureraient pas dans le pseudo-code précédent à virgule flottante (ancienne/nouvelle).

Variable	Format	Taille	Temp/perm	Ancienne/nouvelle
NLS	entier	1	temp	nouvelle
NLSREXPW	entier	1	perm	nouvelle
NLSTMP	entier	1	temp	nouvelle
REXPW	BFL	11	perm	ancienne
R	BFL	1	perm	ancienne
SBW	Q2	60	perm	ancienne
STMP	Q2	20	perm	ancienne
WS	BFL	60	temp	ancienne
BFL	Virgule flottante de bloc			
entier	Entier de 16 bits			

G.3.13 Bloc 38 – Calculateur des coefficients du filtre de pondération

Ci-après, le pseudo-code à virgule flottante pour ce bloc.

If ICOUNT \neq 3, skip the execution of this block
 Otherwise, do the following

For I = 2, 3, ..., 11, do the next line

AWP(I) = WPCFV(I) * AWZTMP(I) | Normalisation des coefficients du dénominateur

For I = 2, 3, ..., 11, do the next line

AWZ(I) = WZCFV(I) * AWZTMP(I) | Normalisation des coefficients du numérateur

Dans le pseudo-code à virgule fixe, il faut envisager les éventualités suivantes: soit l'existence d'un défaut de conditionnement dans la récurrence de Durbin, soit l'impossibilité même d'exprimer AWZTMP en Q13. (On n'a jamais observé que Q13 fut insuffisant, mais cette éventualité doit néanmoins être envisagée.) La variable ILLCONDW est un fanion émanant du bloc 37 qui indique si les résultats de ce bloc sont valables ou non. Dans la Recommandation G.728, il y a une hypothèse implicite selon laquelle les résultats de Durbin ne seront pas utilisés si ILLCONDW est vrai. Cela signifie que AWZ et AWP ne seront pas mis à jour à partir de AWZTMP. La même hypothèse est réitérée ici. Si ILLCONDW est vrai, il n'y a pas de mise à jour de AWP ni de AWZ. Cela est inutile car on continuera à utiliser les valeurs précédentes.

Il faut ensuite envisager l'éventualité que les coefficients AWZTMP() de la récurrence de Durbin soient au format Q13, Q14 ou Q15. NLSAWZTMP représente le nombre de décalages à gauche de AWZTMP. Il faut que les coefficients du numérateur et du dénominateur, AWZ et AWP, soient en format Q14 pour la sortie. Il peut y avoir impossibilité de représenter AWZ en Q14. Si tel est le cas, on ne met pas à jour AWZ ni AWP. Ci-après, le pseudo-code à virgule fixe.

If ICOUNT \neq 3, skip the execution of this block
 Otherwise, do the following

| Voir tout d'abord si ILLCONDW est vrai

If ILLCONDW = .TRUE., skip the execution of this block
 Otherwise, do the following

| Déterminer ensuite les coefficients du
 | numérateur, s'ils débordent pour Q14,
 | pas de mise à jour de AWZ ni de AWP
 | La table temporaire WS est utilisée en cas
 | de débordement, de manière à conserver AWZ

For I = 2, 3, ..., 7, do the next 6 lines

AA0 = WZCFV(I) * AWZTMP(I)
 If NLSAWZTMP = 13, AA0 = AA0 << 3
 If NLSAWZTMP = 14, AA0 = AA0 << 2
 If NLSAWZTMP = 15, AA0 = AA0 << 1
 If AA0 overflowed above, go to LABEL
 WS(I) = RND(AA0)

| WZCFV est en Q14,
 | AA0 = 14 + NLSAWZTMP. Mettre AA0
 | en Q30 pour les 3 cas en effectuant
 | le nombre approprié de décalages
 | Si cela est vrai, il y aura débordement
 | de Q14; arrondir sur le mot supérieur
 | pour WS. Il ne peut pas y avoir de
 | débordement dans les autres cas
 | Si le programme parvient à ce point,
 | continuer sans les contrôler
 | Copier ensuite WS sur AWZ

For I = 8, 9, 10, 11, do the next 5 lines

AA0 = WZCFV(I) * AWZTMP(I)
 If NLSAWZTMP = 13, AA0 = AA0 << 3
 If NLSAWZTMP = 14, AA0 = AA0 << 2
 If NLSAWZTMP = 15, AA0 = AA0 << 1
 WS(I) = RND(AA0)

For I = 2, 3, ..., 11, do the next line

AWZ(I) = WS(I)

| Pas de débordements, donc copier
 | WS sur AWZ

| Déterminer maintenant les coefficients
 | du dénominateur
 | Si le numérateur n'a pas débordé, le dénominateur
 | ne peut pas déborder non plus

For I = 2, 3, ..., 11, do the next 5 lines

AA0 = WPCFV(I) * AWZTMP(I)
 If NLSAWZTMP = 13, AA0 = AA0 << 3
 If NLSAWZTMP = 14, AA0 = AA0 << 2
 If NLSAWZTMP = 15, AA0 = AA0 << 1
 AWP(I) = RND(AA0)

| WPCFV est en Q14; AA0 = 14 + NLSAWZTMP
 | Mettre AA0 en Q30 pour les 3 cas
 | en effectuant le nombre approprié de décalages
 |
 | Arrondir au mot supérieur pour AWP

Exit this subroutine

LABEL:

| Si le programme parvient à ce point, on aura
 | un débordement si on essaie de représenter AWZ
 | en Q14. Dans ce cas, on ne met pas à jour
 | les coefficients du filtre de pondération
 | (en d'autres termes, on continue d'utiliser les
 | coefficients du filtre du précédent cycle d'adaptation).

G.3.14 Bloc 43 – Module de fenêtrage hybride

On trouvera dans ce paragraphe le pseudo-code à virgule flottante et le pseudo-code à virgule fixe pour le bloc 43. Tout d'abord, le pseudo-code à virgule flottante:

N1 = LPCLG + NUPDATE
 N2 = LPCLG + NONRLG
 N3 = LPCLG + NUPDATE + NONRLG

| Calculer quelques constantes (qui peuvent
 | être précalculées et mises en mémoire)

For N = 1, 2, ..., N2, do the next line

SBLG(N) = SBLG(N + NUPDATE)

| Décaler le tampon de l'ancien signal

For N = 1, 2, ..., NUPDATE, do the next line

SBLG(N2 + N) = GTMP(N)

| Introduire par décalage le nouveau signal
 | SBW(N3) est le plus récent échantillon

```

K = 1
For N = N3, N3 - 1, ..., 3, 2, 1, do the next 2 lines
    WS(N) = SBLG(N) * WNRLG(K)           | Multiplier par la fonction de fenêtrage
    K = K + 1

For I = 1, 2, ..., LPCLG + 1, do the next 4 lines
    TMP = 0
    For N = LPCLG + 1, LPCLG + 2, ..., N1, do the next line
        TMP = TMP + WS(N) * WS(N + 1 - I)
    REXPLG(I) = (3/4) * REXPLG(I) + TMP   | Mettre à jour la composante récurrente

For I = 1, 2, ..., LPCLG + 1, do the next 3 lines
    R(I) = REXPLG(I)
    For N = N1 + 1, N1 + 2, ..., N3, do the next line
        R(I) = R(I) + WS(N) * WS(N + 1 - I) | Ajouter la composante non récurrente

R(1) = R(1) * WNCF                       | Correction par bruit blanc

```

A noter que, avant l'appel de ce programme, GTMP() est affecté comme suit:

```

GTMP(1) = GSTATE(4)
GTMP(2) = GSTATE(3)
GTMP(3) = GSTATE(2)
GTMP(4) = GSTATE(1)

```

et les valeurs initiales de GSTATE() sont -32 en virgule flottante, ce qui correspond à -16384 en virgule fixe format Q9. On trouvera ci-après la version du même module en virgule fixe. Plusieurs variables nouvelles ont été ajoutées dans ce code. NLSREXPLG est une variable globale qui renferme le nombre de décalages à gauche nécessaires pour normaliser REXPLG. Cette variable est initialisée avec la valeur 31.

```

N1 = LPCLG + NUPDATE (= 10 + 4)           | Calculer quelques constantes (qui peuvent
N2 = LPCLG + NONRLG (= 10 + 20)           | être précalculées et mises en mémoire)
N3 = LPCLG + NUPDATE + NONRLG (= 10 + 4 + 20)

For N = 1, 2, ..., N2, do the next line
    SBLG(N) = SBLG(N + NUPDATE)           | Décaler le tampon de l'ancien signal
For N = 1, 2, ..., NUPDATE, do the next line
    SBLG(N2 + N) = GTMP(N)                | SBLG(N3) est l'échantillon le plus récent
                                           | Tous les SBLG sont en Q9 et représentés
                                           | avec une précision de 16 bits
Call FINDNLS(SBLG, N3, N3, 14, NLS)      | Trouver le nombre de décalages à gauche
NLSTMP = NLS - 1                          | nécessaires dans la boucle suivante pour
                                           | obtenir plus tard une marge de 2 bits

K = 1
For N = 34, 33, ..., 1, do the next 5 lines
    P = SBLG(N) * WNRLG(K)                | WNRLG est en Q15
    If NLSTMP = -1, set AA0 = P >> 1
    If NLSTMP > -1, set AA0 = P << NLSTMP
    WS(N) = RND(AA0)                       | WS(N) se compose de 14 bits, ou moins
    K = K + 1

NLSATTLG = 14
Call HWMCORE(LPCLG, N1, N3, NLSATTLG, WS, NLSTMP, REXPLG, NLSREXPLG, R, ILLCONDG)

```

Le sous-programme HWMCORE est décrit au G.3.18.

Pour plus de commodité, on trouvera dans le tableau suivant la liste de toutes les variables du pseudo-code spécifié ci-dessus, avec indication de leur taille et de leur format de représentation. Le tableau indique si chaque variable est temporaire (temp) – auquel cas il est inutile de la mettre en mémoire après l'exécution du module – ou permanente (perm), auquel cas on aura encore besoin de la valeur après le calcul en cours. Le tableau indique également quelles variables ne figuraient pas dans le pseudo-code précédent à virgule flottante (ancienne/nouvelle).

Variable	Format	Taille	Temp/perm	Ancienne/nouvelle
GTMP	Q9	4	perm	ancienne
NLS	entier	1	temp	nouvelle
NLSREXPLG	entier	1	perm	nouvelle
NLSTMP	entier	1	temp	nouvelle
REXPLG	BFL	11	perm	ancienne
R	BFL	11	perm	ancienne
SBLG	Q9	34	perm	ancienne
WS	BFL	34	temp	ancienne
BFL	Virgule flottante de bloc			
entier	Entier de 16 bits			

G.3.15 Bloc 45 – Module d'extension de la largeur de bande

Ci-après, le pseudo-code à virgule flottante pour le bloc 45 (module d'extension de la largeur de bande).

```

If ICOUNT ≠ 2, skip the execution of this block
For I = 2, 3, ..., LPCLG + 1, do the next line
    GP(I) = FACGPV(I) * GPTMP(I)                | Normalisation des coefficients

```

Les tables de FACGPV sont établies en format Q14, comme celles des autres coefficients d'extension de la largeur de bande. Les valeurs de la table des GPTMP d'entrée sont exprimées en format Q13, Q14 ou Q15. Comme indiqué dans la description précédente du module de récurrence de Levinson-Durbin à virgule fixe, NLSGPTMP est donné par ce module pour spécifier le format utilisé pour GPTMP. Après la multiplication $FACGPV(I) * GPTMP(I)$, on a besoin du nombre requis correspondant de décalages à gauche.

Les valeurs finales de GP sont toujours représentées en format Q14. On a établi empiriquement que les tables des coefficients de sortie du bloc 45 ne sont jamais trop grandes pour pouvoir être représentées en Q14 (c'est-à-dire qu'elles nécessitent le format Q13 ou moins). Cependant, par souci de sécurité, il faut être prêt à gérer le cas, peu probable, d'un débordement en Q14 à la sortie des blocs d'extension de la largeur de bande. Le pseudo-code développé plus loin recherche l'éventualité d'un tel débordement. Si cette éventualité se produit et est décelée, on procède un peu comme dans le cas des modules de récurrence de Levinson-Durbin: on ne met pas à jour les coefficients du prédicteur et on continue à utiliser les anciens coefficients du cycle d'adaptation précédent. En théorie, on pourrait avoir recours à un format commutable Q14/Q13, avec un fanion pour signaler aux modules de filtrage quel est celui des deux formats Q possibles qui est utilisé. Mais cette solution accroîtrait inutilement la complexité du code DSP ainsi que la durée d'exécution. Comme on n'a jamais observé de débordement en Q14 à la sortie des modules d'extension de la largeur de bande, il suffit d'appliquer un contrôle de sécurité simple comme celui décrit ci-dessous.

Ci-après, le pseudo-code à virgule fixe pour le bloc 45.

```

If ICOUNT ≠ 2, skip the execution of this block
Otherwise, do the following
    | Vérifier d'abord si ILLCONDG est vrai

If ILLCONDG = .TRUE., skip the execution of this block
Otherwise, do the following
    GPTMP(1) = 16384
    For I = 2, 3, 4, ..., LPCLG + 1, do the next 6 lines
        AA0 = FACGPV(I) * GPTMP(I)                | AA0 est en Q27, Q28 ou Q29
        If NLSGPTMP = 13, AA0 = AA0 << 3          | Mettre AA0 en Q30 pour les 3 cas,
        If NLSGPTMP = 14, AA0 = AA0 << 2          | en appliquant le nombre approprié
        If NLSGPTMP = 15, AA0 = AA0 << 1          | de décalages
        If AA0 overflowed above, go to LABEL        | Si cette relation n'est pas vérifiée,
        GPTMP(I) = RND(AA0)                        | arrondir au mot supérieur pour GP

```

For I = 2, 3, 4, ..., LPCLG + 1, do the next line

GP(I) = GPTMP(I)

Exit this program

| Tout est normal, copier GPTMP

| sur GP, puis sortir

LABEL:

| Si le programme parvient à ce point, on aura un
| débordement si on essaie de représenter GP en Q14.

| Dans ce cas, ne pas mettre à jour les coefficients
| du prédicteur de gain logarithmique (c'est-à-dire
| utiliser les coefficients du prédicteur du précédent
| cycle d'adaptation).

G.3.16 Bloc 46 – Prédicteur linéaire de gain logarithmique

Ci-après, le pseudo-code à virgule flottante pour le prédicteur linéaire de gain logarithmique, bloc 46.

LOGGAIN = 0

For I = LPCLG, LPCLG - 1, ..., 3, 2, do the next 2 lines

LOGGAIN = LOGGAIN - GP(I + 1) * GSTATE(I)

GSTATE(I) = GSTATE(I - 1)

LOGGAIN = LOGGAIN - GP(2) * GSTATE(1)

LOGGAIN et GSTATE sont représentés en format Q9 dans tout le codeur. GP est représenté en format Q14. Ci-après, le pseudo-code à virgule fixe.

AA0 = 0

For I = LPCLG, LPCLG - 1, ..., 3, 2, do the next 3 lines

P = GP(I + 1) * GSTATE(I)

AA0 = AA0 - P

GSTATE(I) = GSTATE(I - 1)

P = GP(2) * GSTATE(1)

AA0 = AA0 - P

AA0 = AA0 >> 14

LOGGAIN = AA0

Ci-après, le pseudo-code à virgule flottante pour le bloc 98 (limiteur de gain logarithmique). Comme ce code est fondé sur des modifications apportées pour la virgule fixe, il ne figure pas dans la Recommandation G.728. Il est spécifié aux fins de comparaison avec le pseudo-code à virgule fixe qui suit.

If LOGGAIN > 28., set LOGGAIN = 28

If LOGGAIN < -32., set LOGGAIN = -32

Comme LOGGAIN est représenté en format Q9, les seuils maximal et minimal sont multipliés par 512. Ces valeurs sont utilisées dans le pseudo-code à virgule fixe spécifié ci-après.

If LOGGAIN > 14336, set LOGGAIN = 14336

If LOGGAIN < -16384, set LOGGAIN = -16384

Ci-après, le pseudo-code à virgule flottante pour l'additionneur d'écart de gain logarithmique (bloc 99).

Z = LOGGAIN + GOFF

En virgule flottante, GOFF a la valeur 32, et en virgule fixe la valeur 16384, qui correspond à 512 * 32. Comme la plage de LOGGAIN va de -32 à +28, celle de Z va de 0 à 60. Le code en virgule fixe est identique au code en virgule flottante.

Ci-après, le code à virgule flottante du bloc 48 (module d'exponentiation).

GAIN = 10^(Z/20)

La valeur complète recherchée peut être exprimée en fonction de l'antilogarithme de 2, soit:

$$10^{0,05 Z} = 2^{0,05 \log_2(10) Z} = 2^{0,1660964 Z}$$

Posons: $X = 0,1660964 Z$, avec une plage allant de 0 à 9,97. Posons également $X = [X] + x$, où $[X]$ est le plus grand entier inférieur ou égal à X et x est la partie fractionnaire. La valeur de $2^{[X]}$ est exacte et il suffit de la représenter par son exposant. Il reste maintenant à résoudre le problème suivant: calculer la valeur de la partie fractionnaire.

Pour le calcul de X , représentons $0,1660964$ en format Q21. Cela correspond à un nombre qui peut être représenté par 10 dans les 16 bits supérieurs et par 20649 dans les 15 bits inférieurs. Multiplions Z par les deux parties séparément, afin d'obtenir une précision satisfaisante pour X . Séparons ensuite $[X]$ et x . Dans le calcul de l'exponentielle pour la partie fractionnaire, on sait que $0 < x < 1$, donc $1 < 2^x < 2$. Il est donc possible d'utiliser les représentations fixes suivantes: x est en Q15 et 2^x en Q14. On a recours à un développement en série de Taylor pour calculer 2^x :

$$2^x = \left((c_4 x + c_3) x + c_2 \right) x + c_1 \Big|_x + c_0$$

$$= c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

Les valeurs c , mémorisées en Q14 et Q15, sont données par:

$$c_4 = 323 = 0,0098571 \text{ en Q15}$$

$$c_3 = 1874 = 0,0571899 \text{ en Q15}$$

$$c_2 = 7866 = 0,2400512 \text{ en Q15}$$

$$c_1 = 22702 = 0,6928100 \text{ en Q15}$$

$$c_0 = 16384 = 1,0 \text{ en Q14}$$

On trouvera ci-après le pseudo-code servant à calculer $10^{0,05 \text{ GAIN}}$ sur un DSP de 16 bits avec deux accumulateurs de 32 bits. On admet par hypothèse que GAIN est en format Q9 et que l'écart de 32 dB lui a déjà été ajouté.

AA0 = 10 * Z	Z est en Q9 et 10 en Q6, donc AA0 est en Q15
AA1 = 20649 * Z	20649 est en Q21, donc AA1 est en Q30
AA1 = AA1 << 1	Mettre AA1 en Q31
AA1 = RND(AA1)	Arrondir AA1 pour le mettre en Q15 dans
	le mot inférieur
AA0 = AA0 + AA1	AA0 = [X] + x en Q15
AA1 = AA0 >> 15	Nécessite [X] en Q0 et x en Q15
NLS = AA1	NLS = [X]
AA1 = AA1 << 15	
x = AA0 - AA1	x est la partie fractionnaire, en Q15
	Calculer 2 ** x
AA0 = c4 * x	Q15 * Q15 => AA0 est en Q30
AA0 = AA0 << 1	AA0 est maintenant en Q31
AA1 = c << 16	AA1 est en Q31
AA0 = AA0 + AA1	
TMP = RND(AA0)	TMP est en Q15, appliquer l'arrondi
AA0 = TMP * x	Q15 * Q15 => AA0 est en Q30
AA0 = AA0 << 1	AA0 est maintenant en Q31
AA1 = c2 << 16	AA1 est en Q31
AA0 = AA0 + AA1	
TMP = RND(AA0)	TMP est en Q15, appliquer l'arrondi
AA0 = TMP * x	Q15 * Q15 => AA0 est en Q30
AA0 = AA0 << 1	AA0 est maintenant en Q31
AA1 = c1 << 16	AA1 est en Q31
AA0 = AA0 + AA1	
TMP = RND(AA0)	TMP est en Q15, appliquer l'arrondi
AA0 = TMP * x	Q15 * Q15 => AA0 est en Q30
	Pas de décalage à gauche cette fois!!
AA1 = c0 << 16	AA1 est en Q30
AA0 = AA0 + AA1	
GAIN = RND(AA0)	GAIN est en Q14 et contient 2 ** x
	NLSGAIN est égal à 14 - NLS pour 2 ** X
NLSGAIN = 14 - NLS	Facteur Q comme résultat

On trouvera ci-après le pseudo-code à virgule fixe pour les blocs 96 et 97, calcul de la nouvelle valeur de GSTATE(1). Ce bloc ne figure pas dans la Recommandation G.728; il est spécifié ici aux fins de comparaison avec le pseudo-code à virgule fixe qui sera décrit immédiatement après. Les variables d'entrée sont les suivantes: la valeur de GAIN, fournie à la sortie du bloc 98; GCBLG, valeur en dB de l'entrée du répertoire de gain, sélectionné pour le vecteur d'excitation précédent; et SHAPELG, valeur en dB de l'entrée du répertoire de forme, sélectionné pour le vecteur d'excitation précédent. Ces valeurs sont consignées dans les Tableaux G.3 et G.4 respectivement, qui indiquent, pour lesdites valeurs, les représentations avec virgule flottante et avec virgule fixe. Les représentations avec virgule fixe sont en format Q11. Le pseudo-code à virgule flottante est le suivant:

GSTATE(1) = LOGGAIN + GCBLG(IG) + SHAPELG(IS)
 If GSTATE(1) < -32., set GSTATE(1) = -32.

Ci-après, le pseudo-code à virgule fixe.

AA0 = LOGGAIN << 7	Aligner les virgules à la frontière entre
AA0 = AA0 + (GCBLG(IG) << 5)	le mot supérieur et le mot inférieur
AA0 = AA0 + (SHAPELG(IS) << 5)	de l'accumulateur
AA0 = AA0 >> 7	Décalage à droite pour revenir au format Q9
IF AA0 < -16384, set AA0 = -16384	Vérifier la limite inférieure
GSTATE(1) = AA0	Le mot inférieur de 16 bits est sauvegardé

G.3.17 Bloc 49 – Module de fenêtrage hybride pour le filtre de synthèse

Tout d'abord, le pseudo-code à virgule flottante pour ce module.

N1 = LPC + NFRSZ	Calculer quelques constantes (qui pourront
N2 = LPC + NONR	être précalculées et mises en mémoire)
N3 = LPC + NFRSZ + NONR	
For N = 1, 2, ..., N2, do the next line	
SB(N) = SB(N + NFRSZ)	Décaler le tampon de l'ancien signal
For N = 1, 2, ..., NFRSZ, do the next line	
SB(N2 + N) = STTMP(N)	Introduire par décalage le nouveau signal
	SB(N3) est le plus récent échantillon
K = 1	
For N = N3, N3 - 1, ..., 3, 2, 1, do the next 2 lines	
WS(N) = SB(N) * WNR(K)	Multiplier par la fonction de fenêtrage
K = K + 1	
For I = 1, 2, ..., LPC + 1, do the next 4 lines	
TMP = 0	
For N = LPC + 1, LPC + 2, ..., N1, do the next line	
TMP = TMP + WS(N) * WS(N + 1 - I)	
REXP(I) = (3/4) * REXP(I) + TMP	Mettre à jour la composante récurrente
For I = 1, 2, ..., LPC + 1, do the next 3 lines	
RTMP(I) = REXP(I)	
For N = N1 + 1, N1 + 2, ..., N3, do the next line	
RTMP(I) = RTMP(I) + WS(N) * WS(N + 1 - I)	Ajouter la composante non récurrente
RTMP(1) = RTMP(1) * WNCF	Correction par bruit blanc

Le pseudo-code à virgule fixe pour le module de fenêtrage hybride (bloc 49) est beaucoup plus compliqué que la version à virgule flottante. Cela s'explique par le traitement spécial appliqué au format avec virgule flottante de bloc segmenté ou format SBFL (*segmental block floating point*), nécessaire pour garantir une précision numérique suffisante.

La table STTMP contient 4 vecteurs de parole quantifiés du cycle d'adaptation précédent. Lorsque chacun de ces 4 vecteurs de parole quantifiés (table ST) a été calculé, il était représenté en format BFL avec précision de 14 bits. En général, les nombres de décalages à gauche (nombres NLS) seront différents pour les 4 vecteurs. Pour cette raison, on dit que la table STTMP est stockée en format SBFL; en effet, elle est le résultat de la concaténation de 4 vecteurs BFL ST. La table SB résulte de la concaténation de 21 vecteurs BFL ST. C'est la raison pour laquelle la table SB est stockée, elle aussi, en format SBFL avec précision de 14 bits. A chacun des 4 vecteurs qui composent STTMP est associée une valeur de NLS. Ces valeurs sont mémorisées dans la table NLSSTTMP(). Pour les 21 vecteurs qui composent SB, les valeurs de NLS sont mémorisées dans la table NLSSB().

On trouvera ci-après le pseudo-code à virgule fixe pour le module de fenêtrage hybride.

```

N1 = LPC + NFRSZ (= 70) | Calculer quelques constantes (qui peuvent
N2 = LPC + NONR (= 85) | être précalculées et mises en mémoire)
N3 = LPC + NFRSZ + NONR (= 105)
N4 = N3/IDIM (= 21)
N5 = NFRSZ/IDIM (= 4)
N6 = N4 - N5 (= 17)

For N = 1, 2, ..., N2, do the next line
  SB(N) = SB(N + NFRSZ) | Décaler l'ancienne partie du tampon SB
For N = 1, 2, ..., N6, do the next line
  NLSSB(N) = NLSSB(N + N5) | Décaler l'ancien NLSSB
For N = 1, 2, ..., NFRSZ, do the next line
  SB(N2 + N) = STTMP(N) | Introduire par décalage la nouvelle partie de SB
For N = 1, 2, ..., N5, do the next line
  NLSSB(N6 + N) = NLSSTMP(N) | Introduire par décalage le nouveau NLSSB

| Maintenant, trouver le NLSSB minimal,
| qui détermine NLSWS
NLSTMP = Min{NLSSB(1), NLSSB(2), ..., NLSSB(N4)}

K = 1 | Maintenant, multiplier SB par la
N = N3 | fonction de fenêtrage hybride
For J = 1, 2, ..., N4, do the next 8 lines
  NRSH = NLSSB(J) - NLSTMP - 1 | -1 pour compenser la multiplication Q15
  For M = 1, 2, ..., IDIM, do the next 6 lines
    P = SB(K) * WNR(N) | WNR: multiplication Q15
    If NRSH = -1, set AA0 = P << 1
    If NRSH > -1, set AA0 = P >> NRSH
    WS(K) = RND(AA0) | Arrondir le mot supérieur et stocker dans WS
    N = N - 1
    K = K + 1

NLSATT50 = 14
Call HWMCORE(LPC, N1, N3, NLSATT50, WS, NLSTMP, REXP, NLSREXP, RTMP, ILLCOND)

```

NOTE – Pour plus de commodité, on trouvera dans le tableau suivant la liste de toutes les variables de ce pseudo-code, avec indication de leur format de représentation et de leur taille. Le tableau indique si chaque variable est temporaire (temp) – auquel cas il est inutile de la mettre en mémoire après l'exécution du module – ou permanente (perm), auquel cas on aura encore besoin de la valeur après le calcul en cours. Le tableau indique également quelles variables ne figuraient pas dans le pseudo-code précédent à virgule flottante (ancienne/nouvelle).

Variable	Format	Taille	Temp/perm	Ancienne/nouvelle
NLSSB	entier	21	perm	nouvelle
NLSREXP	entier	1	perm	nouvelle
NLSSTTMP	entier	4	perm	nouvelle
NLSTMP	entier	1	temp	nouvelle
NRSH	entier	1	temp	nouvelle
REXP	BFL	51	perm	ancienne
RTMP	BFL	51	perm	ancienne
SB	SBFL	105	perm	ancienne
STTMP	SBFL	20	perm	ancienne
WS	BFL	105	temp	ancienne
BFL	Virgule flottante de bloc			
entier	Entier de 16 bits			
SBFL	Virgule flottante de bloc segmenté avec précision de 14 bits			
WS	Est en format BFL avec précision de 14 bits			
REXP et RTMP	Ont une précision de 16 bits			

G.3.18 HWMCORE – Noyau du module de fenêtrage hybride

Ce module complète le calcul de fenêtrage hybride pour les blocs 36, 43 et 49. Chacun de ces blocs a sa propre partie initiale. Les variables circulent de ces blocs à destination de ce module. Pour éviter toute confusion, on a changé la désignation de certaines variables afin que les dénominations utilisées dans ce pseudo-code soient sans rapport avec l'un quelconque des trois blocs précités. Le tableau ci-après indique la correspondance entre les dénominations adoptées dans ce module et celles des blocs 36, 43 et 49.

Variable	Bloc 36	Bloc 43	Bloc 49
LPO	LPCW (=10)	LPCLG (=10)	LPC (=50)
NLSATT	NLSATTW	NLSATTLG	NLSATT50
NLSRREC	NLSREXPW	NLSREXPPLG	NLSREXP
N1	30	14	70
N3	60	34	105
R	R	R	RTMP
RREC	REXPW	REXPPLG	REXP

En plus de ces variables, la table de travail WS et le nombre correspondant de décalages, NLSTMP, circulent aussi de ces blocs à destination de ce module.

On trouvera ci-après le pseudo-code à virgule fixe pour ce module.

```

SUBROUTINE HWMCORE(LPO, N1, N3, NLSATT, WS, NLSTMP, RREC, NLSRREC, R, ILLCOND)
NLSAA0 = 2 * NLSTMP
AA0 = 0 | Calculer la partie récurrente de RREC(1)
For N = LPO + 1, ..., N1, do the next 2 lines
  P = WS(N) * WS(N) | WS a une marge de 2 bits
  AA0 = AA0 + P | AA0 aura une marge de 5 bits
  | pour le calcul de l'énergie

| Cas 1: NLSRREC > NLSAA0
If NLSRREC > NLSAA0, do the next 22 lines
  AA0 = AA0 >> 1
  IR = NLSRREC - NLSAA0 + 1
  AA1 = RREC(1) << NLSATT | Cela peut être obtenu par multiplication
  AA1 = -AA1 + (RREC(1) << 16) | Normaliser RREC par le facteur d'atténuation
  AA1 = AA1 >> IR | Aligner AA0 et AA1
  AA0 = AA0 + AA1
  Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE) | Trouver NLS pour RREC
  RREC(1) = RND(AA0) | Les 16 bits supérieurs de AA1 sont sauvegardés
  NLSRREC = NLSAA0 - 1 + NLSRE
  For I = 1, 2, ..., LPO, do the next 11 lines
    AA0 = 0 | Calculer la partie récurrente de RREC(I + 1)
    For N = LPO + 1, ..., N1, do the next 2 lines
      P = WS(N) * WS(N - I)
      AA0 = AA0 + P
    AA0 = AA0 >> 1
    AA1 = RREC(I + 1) << NLSATT | Normaliser RREC par 3/4 ou 1/2
    AA1 = AA1 + (RREC(I + 1) << 16) |
    AA1 = AA1 >> IR
    AA0 = AA0 + AA1
    AA0 = AA0 << NLSRE
    RREC(I + 1) = RND(AA0) | Les 16 bits supérieurs de AA0 sont sauvegardés
  Go to FIN_RECUR

| Cas 2: NLSRREC = NLSAA0
If NLSRREC = NLSAA0, do the next 21 lines
  AA1 = RREC(1) << NLSATT | Normaliser RREC par 3/4 ou 1/2
  AA1 = -AA1 + (RREC(1) << 16) |
  AA0 = AA0 >> 1
  AA1 = AA1 >> 1
  AA0 = AA0 + AA1
  Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE) | Trouver NLS pour RREC
  RREC(1) = RND(AA0) | Les 16 bits supérieurs de AA1 sont sauvegardés
  NLSRREC = NLSRREC - 1 + NLSRE
  For I = 1, 2, ..., LPO, do the next 11 lines
    AA0 = 0 | Calculer la partie récurrente de RREC(I + 1)
    For N = LPO + 1, ..., N1, do the next 2 lines
      P = WS(N) * WS(N - I)
      AA0 = AA0 + P
    AA0 = AA0 >> 1
    AA1 = RREC(I + 1) << NLSATT | Normaliser RREC par 3/4 ou 1/2
    AA1 = -AA1 + (RREC(I + 1) << 16) |
    AA1 = AA1 >> 1
    AA0 = AA0 + AA1
    AA0 = AA0 << NLSRE
    RREC(I + 1) = RND(AA0) | Les 16 bits supérieurs de AA0 sont sauvegardés
  Go to FIN_RECUR

```

	Cas 3: NLSRREC < NLSAA0
If NLSRREC < NLSAA0, do the next 21 lines	
IR = NLSAA0 – NLSRREC + 1	
AA0 = AA0 >> IR	
AA1 = RREC(1) << NLSATT	Normaliser RREC par 3/4 ou 1/2
AA1 = –AA1 + (RREC(1) << 16)	
AA1 = AA1 >> 1	
AA0 = AA0 + AA1	
Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE)	
RREC(1) = RND(AA0)	Les 16 bits supérieurs de AA1 sont sauvegardés
NLSRREC = NLSRREC – 1 + NLSRE	
For I = 1, 2, ..., LPO, do the next 11 lines	
AA0 = 0	Calculer la partie récurrente de RREC(I + 1)
For N = LPO + 1, ..., N1, do the next 2 lines	
P = WS(N) * WS(N – I)	
AA0 = AA0 + P	
AA0 = AA0 >> IR	
AA1 = RREC(I + 1) << NLSATT	Normaliser RREC par 3/4 ou 1/2
AA1 = –AA1 + (RREC(I + 1) << 16)	
AA1 = AA1 >> 1	
AA0 = AA0 + AA1	
AA0 = AA0 << NLSRE	
RREC(I + 1) = RND(AA0)	Les 16 bits supérieurs de AA0 sont sauvegardés
FIN_RECUR:	Lorsqu'on est parvenu à ce point, la composante récurrente a été calculée
AA0 = 0	Calculer la partie non récurrente de R(1)
For N = N1 + 1, ..., N3, do the next 2 lines	
P = WS(N) * WS(N)	
AA0 = AA0 + P	
	Cas 1: NLSRREC > NLSAA0
If NLSRREC > NLSAA0, do the next 21 lines	
IR = NLSRREC – NLSAA0 + 1	
AA1 = RREC(1) << 16	
AA1 = AA1 >> IR	
AA0 = AA0 >> 1	
AA1 = AA0 + AA1	
AA0 = AA1 >> 8	Appliquer le facteur de correction par bruit blanc
AA1 = AA1 + AA0	
Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR)	
R(1) = RND(AA1)	Les 16 bits supérieurs de AA1 sont sauvegardés
For I = 1, 2, ..., LPO, do the next 10 lines	
AA0 = 0	Calculer la partie non récurrente de R(I + 1)
For N = N1 + 1, ..., N3, do the next 2 lines	
P = WS(N) * WS(N – I)	
AA0 = AA0 + P	
AA0 = AA0 >> 1	
AA1 = RREC(I + 1) << 16	
AA1 = AA1 >> IR	
AA1 = AA0 + AA1	
AA1 = AA1 << NLSRR	
R(I + 1) = RND(AA1)	Sauvegarder les 16 bits supérieurs
Go to END	

<pre> If NLSRREC = NLSAA0, do the next 18 lines AA0 = AA0 >> 1 AA1 = RREC(1) << 15 AA1 = AA0 + AA1 AA0 = AA1 >> 8 AA1 = AA1 + AA0 Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR) R(1) = RND(AA1) For I = 1, 2, ..., LPO, do the next 9 lines AA0 = 0 For N = N1 + 1, ..., N3, do the next 2 lines P = WS(N) * WS(N - I) AA0 = AA0 + P AA0 = AA0 >> 1 AA1 = RREC(I + 1) << 15 AA1 = AA0 + AA1 AA1 = AA1 << NLSRR R(I + 1) = RND(AA1) Go to END </pre>	<pre> Cas 2: NLSRREC = NLSAA0 Cela peut être obtenu par multiplication Appliquer le facteur de correction par bruit blanc Les 16 bits supérieurs de AA1 sont sauvegardés Calculer la partie non récurrente de R(I + 1) Sauvegarder les 16 bits supérieurs </pre>
<pre> If NLSRREC < NLSAA0, do the next 18 lines IR = NLSAA0 - NLSRREC + 1 AA0 = AA0 >> IR AA1 = RREC(1) << 15 AA1 = AA0 + AA1 AA0 = AA1 >> 8 AA1 = AA1 + AA0 Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR) R(1) = RND(AA1) For I = 1, 2, ..., LPO, do the next 9 lines AA0 = 0 For N = N1 + 1, ..., N3, do the next 2 lines P = WS(N) * WS(N - I) AA0 = AA0 + P AA0 = AA0 >> IR AA1 = RREC(I + 1) << 15 AA1 = AA0 + AA1 AA1 = AA1 << NLSRR R(I + 1) = RND(AA1) </pre>	<pre> Cas 3: NLSRREC < NLSAA0 Cela peut être obtenu par multiplication Appliquer le facteur de correction par bruit blanc Les 16 bits supérieurs de AA1 sont sauvegardés Calculer la partie non récurrente de R(I + 1) Sauvegarder les 16 bits supérieurs </pre>
<pre> END: ILLCOND = .FALSE. If AA1 = 0, set ILLCOND = .TRUE </pre>	<pre> Dernière opération: rechercher un défaut de conditionnement éventuel AA1 contient encore R(LPO + 1) de 32 bits </pre>

NOTE – Pour plus de commodité, on trouvera dans le tableau suivant la liste de toutes les variables du pseudo-code spécifié ci-dessous, avec indication de leur format de représentation et de leur taille. Le tableau indique si chaque variable est temporaire (temp) – auquel cas il est inutile de la mettre en mémoire après l'exécution du module – ou permanente (perm), auquel cas on aura encore besoin de la valeur après le calcul en cours. Le tableau indique également quelles variables ne figuraient pas dans le pseudo-code précédent à virgule flottante (ancienne/nouvelle).

Variable	Format	Taille	Temp/perm	Ancienne/nouvelle
NLSRE	entier	1	temp	nouvelle
NLSRR	entier	1	temp	nouvelle
NLSRREC	entier	1	perm	nouvelle
NLSTMP	entier	1	temp	nouvelle
RREC	BFL	51	perm	ancienne
R	BFL	51	perm	ancienne
WS	BFL	105	temp	ancienne
BFL	Virgule flottante de bloc			
entier	Entier de 16 bits			
SBFL	Virgule flottante de bloc segmenté, précision de 14 bits			
WS	Est en format BFL avec précision de 14 bits			
RREC et R	Ont une précision de 16 bits			
RREC	Représente REXP, REXPW ou REXPLG, selon que ce module a été appelé depuis le bloc 49, 36 ou 43			

G.3.19 Bloc 51 – Module d'extension de la largeur de bande

Ci-après, le pseudo-code à virgule flottante pour le bloc 51, module d'extension de la largeur de bande. Le même code existe pour le bloc 45, module d'extension de la largeur de bande du prédicteur linéaire de gain logarithmique. Dans le cas présent, on utilise une table différente et le nombre des coefficients du filtre est plus grand.

```

For I = 2, 3, ..., LPC + 1, do the next line
    ATMP(I) = FACV(I) * ATMP(I)                | Normalisation des coefficients

Wait until ICOUNT = 3, then
for I = 2, 3, ..., LPC + 1, do the next line
    A(I) = ATMP(I)

```

Les tables de valeurs de FACV sont établies en format Q14 pour les autres coefficients d'extension de la largeur de bande. Les valeurs des entrées de la table ATMP sont en format Q13, Q14 ou Q15. Comme déjà indiqué dans la description du module de récurrence de Levinson-Durbin avec virgule fixe, NLSATMP est fourni par ce module pour spécifier le format utilisé pour ATMP. Après la multiplication $FACV(I) * ATMP(I)$, on a besoin de connaître le nombre correspondant de décalages à gauche.

Les valeurs finales pour ATMP sont toujours représentées en format Q14. On a établi empiriquement que les valeurs de ATMP ne sont jamais trop grandes pour pouvoir être représentées en Q14 (c'est-à-dire qu'elles nécessitent le format Q13 ou moins). Cependant, par souci de sécurité, il faut être prêt à gérer le cas, peu probable, d'un débordement en Q14 à la sortie du module d'extension de la largeur de bande. Le pseudo-code développé plus loin recherche l'éventualité d'un tel débordement. Si cette éventualité se produit et est décelée, on procède un peu comme dans le cas des modules de récurrence de Levinson-Durbin: on ne met pas à jour les coefficients du prédicteur et on continue à utiliser les anciens coefficients du cycle d'adaptation précédent. En théorie, on pourrait avoir recours à un format commutable Q14/Q13, avec un fanion pour signaler aux modules de filtrage quel est celui des deux formats Q possibles qui est utilisé. Mais cette solution accroîtrait inutilement la complexité du code DSP ainsi que la durée d'exécution. Comme on n'a jamais observé de débordement en Q14 à la sortie des modules d'extension de la largeur de bande, il suffit d'appliquer un contrôle de sécurité simple comme celui décrit ci-dessous.

Ci-après, le pseudo-code à virgule fixe pour le bloc 51.

If ICOUNT \neq 3, skip the following
 Otherwise, do the following

| Vérifier d'abord si ILLCOND est vrai

If ILLCOND = .TRUE., skip the execution of this block
 Otherwise, do the following

ATMP(1) = 16384

For I = 2, 3, 4, ..., LPC + 1, do the next 6 lines

AA0 = FACV(I) * ATMP(I)

| AA0 est en Q27, Q28 ou Q29

If NLSATMP = 13, AA0 = AA0 << 3

| Mettre AA0 en Q30 en effectuant

If NLSATMP = 14, AA0 = AA0 << 2

| le nombre approprié de décalages

If NLSATMP = 15, AA0 = AA0 << 1

If AA0 overflowed above, go to LABEL

| Si cette relation n'est pas vérifiée,

ATMP(I) = RND(AA0)

| arrondir au mot supérieur pour ATMP

For I = 2, 3, ..., LPC + 1, do the next line

A(I) = ATMP(I)

Exit this module

LABEL:

| Si le programme parvient à ce point, on aura un
 | débordement si on essaie de représenter A en Q14.
 | Dans ce cas, on ne met pas à jour les coefficients du
 | filtre de synthèse (en d'autres termes on continue
 | d'utiliser les coefficients du filtre de synthèse du
 | précédent cycle d'adaptation).

G.3.20 Blocs 71 et 72 – Postfiltres à long terme et à court terme

Les blocs 71 et 72 sont combinés, afin de protéger la précision de la variable intermédiaire TEMP qui a circulé entre eux dans le pseudo-code à virgule flottante. Ce pseudo-code à virgule flottante, pour ces deux blocs, est détaillé ci-après.

For K = 1, 2, ..., IDIM, do the next line

TEMP(K) = GL * (ST(K) + B * ST(K - KP))

| Postfiltrage à long terme

For K = -NPWSZ - KPMAX + 1, ..., -2, -1, 0, do the next line

ST(K) = ST(K + IDIM)

| Décaler le registre de vecteurs
 | de parole décodés

For K = 1, 2, ..., IDIM, do the following

TEMP = TEMP(K)

For J = 10, 9, ..., 3, 2, do the next 2 lines

TEMP(K) = TEMP(K) + STPFIR(J) * AZ(J + 1)

| Section tous zéros

STPFIR(J) = STPFIR(J - 1)

| du filtre

TEMP(K) = TEMP(K) + STPFIR(1) * AZ(2)

| Dernier multiplicateur

STPFIR(1) = TMP

For J = 10, 9, ..., 3, 2, do the next 2 lines

TEMP(K) = TEMP(K) - STPFIIR(J) * AP(J + 1)

| Section tous pôles

STPFIIR(J) = STPFIIR(J - 1)

| du filtre

TEMP(K) = TEMP(K) - STPFIIR(1) * AP(2)

| Dernier multiplicateur

STPFIIR(1) = TEMP(K)

TEMP(K) = TEMP(K) + STPFIIR(2) * TILTZ

| Filtre de compensation
 | d'écart de niveau spectral

Ci-après, le pseudo-code à virgule fixe. Dans l'ensemble de ce code, les variables STPFIR et STPFIIR sont en Q2.

For K = 1, 2, ..., IDIM, do the following indented lines

AA0 = GL * SST(K)

| Traiter d'abord le postfiltre à long terme

AA0 = AA0 + GLB * SST(K - KP)

| GL est en Q14, SST(1:5) est en Q2

| GLB est en Q16, SST(-239:0) est en Q0

AA1 = AA0	Traiter ensuite le postfiltre à court terme
	Réaliser la partie FIR du filtre
For J = 10, 9, ..., 3, 2, do the next 2 lines	
AA1 = AA1 + STPFIR(J) * AZ(J + 1)	AZ est en Q14, STPFIR(J) est en Q2
STPFIR(J) = STPFIR(J - 1)	
AA1 = AA1 + STPFIR(1) * AZ(2)	
AA0 = AA0 << 2	
STPFIR(1) = RND(AA0)	Q2 pour STPFIR
	Réaliser maintenant la partie IIR du filtre
For J = 10, 9, ..., 3, 2, do the next 2 lines	
AA1 = AA1 - STPIIR(J) * AP(J + 1)	AP est en Q14, STPIIR(J) est en Q2
STPIIR(J) = STPIIR(J - 1)	
AA1 = AA1 - STPIIR(1) * AP(2)	
AA0 = AA0 >> 14	
	Maintenant, faire un contrôle de saturation
If AA0 > 32767, set AA0 = 32767	
If AA0 < -32768, set AA0 = -32768	
STPIIR(1) = AA0	
	Maintenant, mettre en œuvre le filtre de
	compensation d'écart de niveau spectral
AA1 = AA1 + STPIIR(2) * TILTZ	TILTZ est en Q14
AA1 = AA1 >> 14	
If AA1 > 32767, set AA1 = 32767	
If AA1 < -32768, set AA1 = -32768	
TEMP(K) = AA1	
	Maintenant, décaler le tampon du postfiltre
	à long terme
For K = -NPWSZ - KPMAX + 1, ..., -7, -6, -5, do the next line	
SST(K) = SST(K + IDIM)	Décaler le registre de vecteurs
	de parole décodés
For K = -4, -3, ..., 0, do the next line	
SST(K) = SST(K + IDIM) >> 2	Décaler le registre de vecteurs
	de parole décodés et remplacer Q2 par Q0

G.3.21 Blocs 73 et 74 – Calculateurs de somme de valeurs absolues

Les blocs 73 et 74 sont très semblables. Leurs résultats sont conservés en double précision. Comme indiqué ici, il est inutile que ces résultats soient mémorisés avant l'intervention du bloc 75. On trouvera ci-après le pseudo-code à virgule flottante pour le bloc 73. A noter que, dans ce code, le nom de variable ST a été remplacé par SST, cela afin d'assurer la cohérence avec le code à virgule fixe donné plus loin.

On se rappellera que SST(1:5) est représenté en Q2.

```
SUMUNFIL = 0
For K = 1, 2, ..., IDIM, do the next line
    SUMUNFIL = SUMUNFIL + | SST(K) |
```

Ci-après, le pseudo-code pour le bloc 74.

```
SUMFIL = 0
For K = 1, 2, ..., IDIM, do the next line
    SUMFIL = SUMFIL + absolute value of TEMP(K)
```

Ci-après, le pseudo-code à virgule fixe pour ces deux blocs.

```

AA1 = 0
AA0 = 0
For K = 1, 2, ..., IDIM, do the next 2 lines
    AA0 = AA0 + | SST(K) |           | Ajouter la valeur absolue de SST(K)
    AA1 = AA1 + | TEMP(K) |        | Ajouter la valeur absolue de TEMP(K)
                                     | AA0 = SUMUNFIL
                                     | AA1 = SUMFIL
                                     | SST et TEMP sont en Q2, donc
                                     | AA0 et AA1 sont aussi en Q2
                                     | AA0 et AA1 seront utilisés dans le bloc 75

```

G.3.22 Bloc 75 – Calculateur de facteur de normalisation

Le bloc 75 calcule le rapport SUMUNFIL/SUMFIL et le résultat est stocké dans SCALE avec la précision NLSSCALE. SUMUNFIL(AA0) et SUMFIL(AA1) viennent respectivement des blocs 73 et 74. Ci-après, le pseudo-code à virgule flottante.

```

If SUMFIL > 1, set SCALE = SUMUNFIL / SUMFIL
Otherwise, set SCALE = 1

```

Ci-après, le pseudo-code à virgule fixe.

```

If AA1 > 4, do the following indented lines
    Call VSCALE(AA1, 1, 1, 30, AA1, NLSDEN)
    DEN = RND(AA1)
    Call VSCALE(AA0, 1, 1, 30, AA0, NLSNUM)
    NUM = RND(AA0)                               | NLSNUM et NLSDEN sont tous deux
                                                | décalés de 16, d'où annulation
    Call DIVIDE(NUM, NLSNUM, DEN, NLSDEN, SCALE, NLSSCALE)
Otherwise, set SCALE = 16384 and NLSSCALE = 14

```

G.3.23 Bloc 76 – Filtre passe-bas du premier ordre, et bloc 77 – Module de normalisation du gain de sortie

On trouvera ci-après le pseudo-code à virgule flottante pour ces deux blocs.

```

For K = 1, 2, ..., IDIM, do the following
    SCALEFIL = AGCFAC * SCALEFIL + (1 - AGCFAC) * SCALE | Filtrage passe-bas
    SPF(K) = SCALEFIL * TEMP(K)                        | Normalisation de la sortie

```

Dans le pseudo-code à virgule fixe, on calcule d'abord le second terme et on l'ajoute dans chaque itération, afin de sauvegarder la soustraction et la multiplication à l'intérieur de la boucle. Ci-dessous, le pseudo-code à virgule fixe:

```

AA1 = AGCFAC1 * SCALE           | AGCFAC1 = 20972 en Q21 = 0,010000228
NRS = NLSSCALE - 14 + (21 - 14) | Calculer le décalage à droite
If NRS ≥ 0, AA1 = AA1 >> NRS    | AA1 doit être en Q28
If NRS < 0, AA1 = AA1 << -NRS  | Décalage à gauche si NRS est négatif

```

```

For K = 1, 2, ..., IDIM, do the following
    AA0 = AA1 + AGCFAC * SCALEFIL | Filtrage passe-bas
                                     | AGCFAC = 16220 en Q14 et SCALEFIL
                                     | est en Q14
    AA0 = AA0 << 2                | Mettre SCALEFIL en Q14
    SCALEFIL = RND(AA0)
                                     | Normalisation de la sortie
    AA0 = SCALEFIL * TEMP(K)       | TEMP(K) est en Q2
    AA0 = AA0 << 2
    SPF(K) = RND(AA0)              | SPF(K) est en Q2

```

G.3.24 Bloc 81 – Filtre LPC inverse du 10^e ordre

Ci-après, la version à virgule flottante du pseudo-code pour le bloc 81 (filtre LPC inverse du 10^e ordre).

```
If IP = NPWSZ, then set IP = NPWSZ – NFRSZ | Vérifier et mettre IP à jour

For K = 1, 2, ..., IDIM, do the next 7 lines
  ITMP = IP + K
  D(ITMP) = ST(K)
  For J = 10, 9, ..., 3, 2, do the next 2 lines
    D(ITMP) = D(ITMP) + STLPCI(J) * APF(J + 1) | Filtrage de FIR
    STLPCI(J) = STLPCI(J – 1) | Introduire le signal d'entrée par décalage
  D(ITMP) = D(ITMP) + STLPCI(1) * APF(2) | Traiter le dernier
  STLPCI(1) = ST(K) | Introduire le signal d'entrée par décalage

IP = IP + IDIM | Mettre IP à jour
```

Dans le code à virgule fixe, il faut tout d'abord convertir ST de la virgule flottante de bloc au format Q2 fixe, puis écrire la version Q2 de ST sur le tampon du postfiltre à long terme, pour utilisation ultérieure par ce postfiltre. A noter que ce tampon avait précédemment la dénomination ST dans le pseudo-code à virgule flottante. ST symbolise la virgule flottante de bloc et le tampon est en Q2. Pour éviter toute confusion, il a été nécessaire de donner une nouvelle désignation, SST, au tampon. A noter que les coefficients, APF, du filtre LPC sont représentés en Q13.

```
NLS = 16 – NLSST + 2 | Calculer l'amplitude du décalage à gauche pour Q2
For K = 1, 2, ..., IDIM, do the next 2 lines
  AA0 = ST(K) << NLS
  SST(K) = RND(AA0) | SST est le nouveau tampon du
  | postfiltre à long terme
If IP = NPWSZ, then set IP = NFRSZ | Vérifier et mettre IP à jour
  | Déclencher le filtrage LPC inverse

For K = 1, 2, ..., IDIM, do the next 10 lines
  AA0 = SST(K)
  AA0 = AA0 << 13
  For J = 10, 9, ..., 3, 2, do the next 2 lines
    AA0 = AA0 + STLPCI(J) * APF(J + 1)
    STLPCI(J) = STLPCI(J – 1)
  AA0 = AA0 + STLPCI(1) * APF(2)
  STLPCI(1) = SST(K)
  ITMP = IP + K
  AA0 = AA0 << 2
  D(ITMP) = RND(AA0) | D(ITMP) est en Q1

IP = IP + IDIM
```

G.3.25 Bloc 82 – Module d'extraction de la période fondamentale (tonie)

Ci-après, tout d'abord, la version à virgule flottante du pseudo-code pour le bloc 82 (module d'extraction de la période fondamentale).

```
IF ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following | Filtrage passe-bas et décimation de rapport 4:1

For K = NPWSZ – NFRSZ + 1, ..., NPWSZ, do the next 7 lines
  TMP = D(K) – STLPF(1) * AL(1) – STLPF(2) * AL(2) – STLPF(3) * AL(3) | Filtre IIR
  If K is divisible by 4, do the next 2 lines
    N = K/4 | N'effectuer le filtrage FIR que si nécessaire
    DEC(N) = TEMP * BL(1) + STLPF(1) * BL(2) + STLPF(2) * BL(3) + STLPF(3) * BL(4)

  STLPF(3) = STLPF(2)
  STLPF(2) = STLPF(1) | Décaler la mémoire du filtre passe-bas
  STLPF(1) = TMP
```

M1 = KPMIN/4	Commencer la recherche des pics de corrélation
M2 = KPMAX/4	dans le domaine des résidus LPC décimés
CORMAX = most negative number of the machine	
For J = M1, M1 + 1, ..., M2, do the next 6 lines	
TMP = 0	
For N = 1, 2, ..., NPWSZ/4, do the next line	
TMP = TMP + DEC(N) * DEC(N - J)	TMP = corrélation dans le domaine décimé
If TMP > CORMAX, do the next 2 lines	
CORMAX = TMP	Trouver le coefficient de corrélation maximal
KMAX = J	et le décalage temporel correspondant
For N = -M2 + 1, -M2 + 2, ..., (NPWSZ - NFRSZ)/4, do the next line	
DEC(N) = DEC(N + IDIM)	Décaler le tampon des résidus LPC décimés
M1 = 4 * KMAX - 3	Commencer la recherche des pics de corrélation
	dans le domaine non décimé
M2 = 4 * KMAX + 3	
If M1 < KPMIN, set M1 = KPMIN	Vérifier si M1 est hors plage
If M2 > KPMAX, set M2 = KPMAX	Vérifier si M2 est hors plage
CORMAX = most negative number of the machine	
For J = M1, M1 + 1, ..., M2, do the next 6 lines	
TMP = 0	
For K = 1, 2, ..., NPWSZ, do the next line	
TMP = TMP + D(K) * D(K - J)	Corrélation dans le domaine non décimé
If TMP > CORMAX, do the next 2 lines	
CORMAX = TMP	Trouver la corrélation maximale et le
KP = J	décalage temporel correspondant
M1 = KP1 - KPDELTA	Déterminer la plage de recherche autour de la
M2 = KP1 + KPDELTA	période fondamentale de la trame précédente
If KP < M2 + 1, go to LABEL	Si cette relation est vérifiée, KP ne peut pas
	être un multiple de la période fondamentale
	Vérifier si M1 est hors plage
If M1 < KPMIN, set M1 = KPMIN	
CMAX = most negative number of the machine	
For J = M1, M1 + 1, ..., M2, do the next 6 lines	
TMP = 0	
For K = 1, 2, ..., NPWSZ, do the next line	
TMP = TMP + D(K) * D(K - J)	Corrélation dans le domaine non décimé
If TMP > CMAX, do the next 2 lines	
CMAX = TMP	Trouver la corrélation maximale et le
KPTMP = J	décalage temporel correspondant
SUM = 0	
TMP = 0	Commencer à calculer les poids des
	coefficients des filtres de prédiction de tonie
For K = 1, 2, ..., NPWSZ, do the next 2 lines	
SUM = SUM + D(K - KP) * D(K - KP)	
TMP = TMP + D(K - KPTMP) * D(K - KPTMP)	
If SUM = 0, set TAP = 0; otherwise, set TAP = CORMAX/SUM	
If TMP = 0., set TAP1 = 0.; otherwise, set TAP1 = CMAX/TMP	
If TAP > 1, set TAP = 1	Clamper TAP entre 0 et 1
If TAP < 0, set TAP = 0	
If TAP1 > 1, set TAP1 = 1	Clamper TAP1 entre 0 et 1
If TAP1 < 0, set TAP1 = 0	
	Remplacer KP par la période fondamentale
	si TAP1 est assez grand
If TAP1 > TAPTH * TAP, then set KP = KPTMP	
LABEL: KP1 = KP	Mettre à jour la période fondamentale de la
	trame précédente
For K = -KPMAX + 1, -KPMAX + 2, ..., NPWSZ - NFRSZ, do the next line	
D(K) = D(K + NFRSZ)	Décaler le tampon des résidus LPC

Dans la version à virgule fixe de ce bloc, la table D et les variables d'état du filtre passe-bas sont représentées en Q1. On procède ainsi pour empêcher des débordements dans les calculs de corrélation et d'énergie. On trouvera ci-après le pseudo-code à virgule fixe.

If ICOUNT \neq 3, skip the execution of this block
Otherwise, do the following

For K = NPWSZ – NFRSZ + 1, ..., NPWSZ, do the next 17 lines

AA0 = D(K) * BL(0)	Traiter tout d'abord la partie FIR
AA0 = AA0 + LPFFIR(1) * BL(1)	D(K) est en Q1, BL() en Q19
AA0 = AA0 + LPFFIR(2) * BL(2)	BL(0) = 18721, BL(1) = -3668
AA0 = AA0 + LPFFIR(3) * BL(3)	BL(2) = -3668, BL(3) = 18721
LPFFIR(3) = LPFFIR(2)	
LPFFIR(2) = LPFFIR(1)	
LPFFIR(1) = D(K)	LPFFIR est en Q1
AA0 = AA0 >> 6	Maintenant, traiter la partie IIR
AA0 = AA0 – LPFIIR(1) * AL(1)	AL() sont en Q13, LPFIIR() en Q1
AA0 = AA0 – LPFIIR(2) * AL(2)	AL(1) = -19172, AL(2) = 16481
AA0 = AA0 – LPFIIR(3) * AL(3)	AL(3) = -5031
LPFIIR(3) = LPFIIR(2)	
LPFIIR(2) = LPFIIR(1)	
AA0 = AA0 << 3	
LPFIIR(1) = RND(AA0)	LPFIIR est en Q1
N = (K >> 2)	
If K = (N << 2), set DEC(N) = LPFIIR(1)	DEC(N) est en Q1

M1 = KPMIN/4	Commencer la recherche des pics de corrélation
M2 = KPMAX/4	dans le domaine des résidus LPC décimés
AA1 = -2147483648	= -2^{31}

For J = M1, M1 + 1, ..., M2, do the next 6 lines

AA0 = 0	
For N = 1, 2, ..., NPWSZ/4, do the next line	
AA0 = AA0 + DEC(N) * DEC(N – J)	
If AA0 > AA1, do the next 2 lines	
AA1 = AA0	Trouver la corrélation maximale et
KMAX = J	l'écart temporel correspondant

For N = -M2 + 1, -M2 + 2, ..., (NPWSZ – NFRSZ)/4, do the next line

DEC(N) = DEC(N + IDIM)

M1 = 4 * KMAX – 3	Commencer la recherche des pics de corrélation
	dans le domaine non décimé

M2 = 4 * KMAX + 3

If M1 < KPMIN, set M1 = KPMIN | Vérifier si M1 est hors plage

If M2 > KPMAX, set M2 = KPMAX | Vérifier si M2 est hors plage

AA1 = -2147483648 | = -2^{31}

For J = M1, M1 + 1, ..., M2, do the next 6 lines

AA0 = 0	
For K = 1, 2, ..., NPWSZ, do the next line	
AA0 = AA0 + D(K) * DEC(K – J)	Corrélation dans le domaine non décimé
If AA0 > AA1, do the next 2 lines	
AA1 = AA0	
KP = J	

CORMAX = AA1 | Sauvegarde avec double précision pour CORMAX

M1 = KP1 – KPDELTA | Déterminer la plage de recherche autour de la

M2 = KP1 + KPDELTA | période fondamentale de la trame précédente

If KP < M2 + 1, go to LABEL | Si cette relation est vérifiée, KP ne peut pas

| être un multiple de la période fondamentale

If M1 < KPMIN, set M1 = KPMIN | Vérifier si M1 est hors plage

If M2 > KPMAX, set M2 = KPMAX | Vérifier si M2 est hors plage

| Cette dernière relation n'est pas en

| virgule flottante

AA1 = -2147483648 | = -2^{31}

```

For J = M1, M1 + 1, ..., M2, do the next 6 lines
  AA0 = 0
  For K = 1, 2, ..., NPWSZ, do the next line
    AA0 = AA0 + D(K) * D(K - J)
  If AA0 > AA1, do the next 2 lines
    AA1 = AA0
    KPTMP = J
  CMAX = AA1

```

| Corrélation dans le domaine non décimé

| Trouver la corrélation maximale et l'écart temporel correspondant

| Sauvegarde avec double précision pour CMAX

```

AA0 = 0
AA1 = 0
For K = 1, 2, ..., NPWSZ, do the next 2 lines
  AA0 = AA0 + D(K - KP) * D(K - KP)
  AA1 = AA1 + D(K - KPTMP) * D(K - KPTMP)

```

| Trouver TAP

| Si nécessaire, limiter les coefficients du filtre de prédiction de tonie

```

If AA0 = 0, set CORMAX = 0
If AA1 = 0, set CMAX = 0
If CORMAX > AA0, set CORMAX = AA0
If CORMAX < 0, set CORMAX = 0
If CMAX > AA1, set CMAX = AA1
If CMAX < 0, set CMAX = 0

```

```

If AA0 > AA1, do the next 2 lines
  call VSCALE(AA0, 1, 1, 30, AA0, NLS)
  AA1 = AA1 << NLS
otherwise do the next 2 lines
  call VSCALE(AA1, 1, 1, 30, AA1, NLS)
  AA0 = AA0 << NLS

```

```

SUM = AA0 >> 16
TMP = AA1 >> 16
AA0 = CORMAX << NLS
CORMAX = AA0 >> 16
AA0 = CMAX << NLS
CMAX = AA0 >> 16
AA1 = CORMAX * TMP
AA1 = AA1 >> 16
AA1 = AA1 * ITAPTH
AA0 = CMAX * SUM
If AA0 > AA1, set KP = KPTMP

```

| ITAPTH = 26214 en Q16

```

LABEL: KP1 = KP
For K = -KPMAX + 1, -KPMAX + 2, ..., NPWSZ - NFRSZ, do the next line
  D(K) = D(K + NFRSZ)

```

| Mettre KP1 à jour et décaler le résidu LPC

| Décaler le tampon des résidus LPC

G.3.26 Bloc 83 – Calculateur des coefficients de prédicteur de tonie

On trouvera tout d'abord ci-après la version à virgule flottante du pseudo-code. On utilise SST au lieu de ST comme dénomination du tampon du postfiltre à long terme.

```

If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following
SUM = 0
TMP = 0
For K = -NPWSZ + 1, -NPWSZ + 2, ..., 0, do the next 2 lines
  SUM = SUM + SST(K - KP) * SST(K - KP)
  TMP = TMP + SST(K) * SST(K - KP)
If SUM = 0, set PTAP = 0; otherwise, set PTAP = TMP/SUM

```

Ci-après, le pseudo-code à virgule fixe. A noter que SST() est en Q0 à 13 bits. Lorsqu'on effectue les corrélations, la multiplication de SST par lui-même, ou un échantillon retardé, donne un résultat qui est en Q0 à 25 bits.

If ICOUNT \neq 3, skip the execution of this block

Otherwise, do the following

AA0 = 0

AA1 = 0

For K = -NPWSZ + 1, -NPWSZ + 2, ..., 0, do the next 4 lines

P = SST(K - KP) * SST(K - KP)

AA0 = AA0 + P

P = SST(K) * SST(K - KP)

AA1 = AA1 + P

If AA0 = 0, set PTAP = 0 and return to calling program

If AA1 \leq 0, set PTAP = 0 and return to calling program

If AA1 \geq AA0, set PTAP = 16384 | NLSPTAP = 14

Otherwise, do the following

Call VSCALE(AA0, 1, 1, 30, AA0, NLSDEN)

Call VSCALE(AA1, 1, 1, 30, AA1, NLSNUM)

NUM = RND(AA1)

DEN = RND(AA0)

Call DIVIDE(NUM, NLSNUM, DEN, NLSDEN, PTAP, NLSPTAP)

NRS = NLSPTAP - 14

PTAP = PTAP >> NRS | NLSPTAP = 14

G.3.27 Bloc 84 – Calculateur des coefficients du postfiltre à long terme

Tout d'abord, le pseudo-code à virgule flottante pour le bloc 84:

If ICOUNT \neq 3, skip the execution of this block

Otherwise, do the following

If PTAP > 1, set PTAP = 1 | Clamper PTAP à 1

If PTAP < PPFTH, set PTAP = 0 | Désactiver le postfiltre fondamental

| si PTAP est inférieur au seuil

B = PPFZCF * PTAP

GL = 1/(1 + B)

Ci-après, le pseudo-code à virgule fixe. On définit une variable supplémentaire, GLB, produit de GL par B. Cela permet par la suite d'économiser des multiplications. B et GLB sortent en Q16 et GL en Q14.

| Noter que PTAP est < 16385,

| en provenance du bloc 83

If ICOUNT \neq 3, skip the execution of this block

Otherwise, do the following

If PTAP < PPFTH, set PTAP = 0 | PPFTH = 9830 en Q14

AA0 = PPFZCF * PTAP | PPFZCF = 9830 en Q16, PTAP est en Q14

B = AA0 >> 14 | Sauvegarder B en Q16

AA0 = AA0 >> 16 | AA0 = B en Q14

AA0 = AA0 + 16384

DEN = AA0 | DEN est en Q14

Call DIVIDE(16384, 14, DEN, 14, GL, NLS)

AA0 = GL * B | NLS = 14 ou 15, NLS de B = 16

GLB = AA0 >> NLS | GLB = GL * B, précalculé ici en Q16

| pour le bloc 71

NRS = NLS - 14

If NRS > 0, SET GL = GL >> NRS | Résultat: GL est en Q14

G.3.28 Bloc 85 – Calculateur des coefficients du postfiltre à long terme

Tout d'abord, le pseudo-code à virgule flottante pour ce bloc:

```
If ICOUNT ≠ 1, skip the execution of this block
Otherwise, do the following
For I = 2, 3, ..., 11, do the next 2 lines
    AP(I) = SPFPCFV(I) * APF(I)           | Normaliser les coefficients du dénominateur
    AZ(I) = SPFZCFV(I) * APF(I)         | Normaliser les coefficients du numérateur
TILTZ = TILTF * RC1                     | Coefficients du filtre de compensation
                                         | d'écart de niveau spectral
```

Dans le pseudo-code à virgule fixe, il faut envisager les possibilités suivantes: soit l'existence d'un défaut de conditionnement dans la récurrence de Durbin, soit l'impossibilité même d'exprimer les coefficients de prédiction en Q13 (on n'a jamais observé que Q13 fut insuffisant, mais cette éventualité doit néanmoins être envisagée). La variable ILLCONDP est un fanion émanant du bloc 50 qui indique si les résultats de ce bloc sont valables ou non. Dans la Recommandation G.728, il y a une hypothèse implicite selon laquelle les résultats de Durbin ne seront pas utilisés si ILLCONDP est vrai. Cela signifie que ATMP ne sera pas copié sur APF au terme de la récurrence du 10^e ordre. La même hypothèse est réitérée ici. Si ILLCONDP est vrai, il n'y a pas de mise à jour de AP, AZ ou TILTZ.

Il faut ensuite envisager l'éventualité selon laquelle les coefficients APF() de la récurrence de Durbin peuvent être en Q13, Q14 ou Q15. NLSAPF représente le nombre de décalages à gauche de APF. A la sortie, on souhaite aussi sauvegarder APF() en Q13, pour utilisation ultérieure dans l'opération de filtrage LPC inverse. Il faut que les coefficients du numérateur et du dénominateur, AP() et AZ(), soient en format Q14 pour la sortie. TILTZ sort en Q14. Il peut y avoir impossibilité de représenter AP en Q14. Si tel est le cas, on ne met pas à jour AP, AZ ni TILTZ, mais on pourra utiliser les nouvelles valeurs pour APF. Ces valeurs devraient déjà se trouver en format Q13. Ci-après, le pseudo-code à virgule fixe.

```
If ICOUNT ≠ 1, skip the execution of this block
Otherwise, do the following
                                         | Vérifier d'abord si ILLCONDP est vrai

If ILLCONDP = .TRUE., skip the execution of this block
otherwise, do the following
                                         | Traiter ensuite les coefficients du dénominateur
                                         | S'ils débordent pour Q14, ne pas
                                         | mettre à jour AP, AZ ni TILTZ
                                         | La table temporaire WS est utilisée en cas
                                         | de débordement, ce qui permet de préserver AP

For I = 2 and 3, do the next 6 lines
    AA0 = SPFPCFV(I) * APF(I)           | SPFPCFV est en Q14, AA0 = 14 + NLSAPF
    If NLSAPF = 13, AA0 = AA0 << 3     | Mettre AA0 en Q30 pour les 3 cas, en
    If NLSAPF = 14, AA0 = AA0 << 2     | appliquant le nombre approprié de décalages
    If NLSAPF = 15, AA0 = AA0 << 1
    If AA0 overflowed above, go to LABEL
    WS(I) = RND(AA0)                   | Arrondir au mot supérieur pour WS
                                         | Un débordement ne peut se produire que
                                         | pour 2 ou 3; en conséquence, copier ces résultats
                                         | sur AP et continuer

For I = 2 and 3, do the next line
    AP(I) = WS(I)

                                         | Maintenant, faire le reste

For I = 4, 5, ..., 11, do the next 5 lines
    AA0 = SPFPCFV(I) * APF(I)           | SPFPCFV est en Q14, AA0 = 14 + NLSAPF
    If NLSAPF = 13, AA0 = AA0 << 3     | Mettre AA0 en Q30 pour les 3 cas, en
    If NLSAPF = 14, AA0 = AA0 << 2     | appliquant le nombre approprié de décalages
    If NLSAPF = 15, AA0 = AA0 << 1
    AP(I) = RND(AA0)                   | Arrondir au mot supérieur pour AP
                                         | Traiter maintenant les coefficients du numérateur
                                         | Si le dénominateur n'a pas débordé,
                                         | le numérateur ne peut pas déborder non plus
```

For I = 2, 3, ..., 11, do the next 5 lines

AA0 = SPZCFV(I) * APF(I) | SPZCFV est en Q14, AA0 = 14 + NLSAPF

If NLSAPF = 13, AA0 = AA0 << 3 | Mettre AA0 en Q30 pour les 3 cas, en

If NLSAPF = 14, AA0 = AA0 << 2 | appliquant le nombre approprié de décalages

If NLSAPF = 15, AA0 = AA0 << 1

AZ(I) = RND(AA0) | Arrondir au mot supérieur pour AZ

AA0 = TILTF * RC1 | Maintenant, mettre à jour TILTZ

TILTZ = RND(AA0) | TILTZ = 4915 en Q15

| RC1 est en Q15

| TILTZ est en Q14

LABEL: | Sauvegarder APF() en Q3 pour filtrage

| LPC inverse ultérieur

| Cas 1: NLSAPF = 13, ne rien faire

If NLSAPF = 14, do the next 3 lines | Cas 2: NLSAPF = 14, décalage à 15,

For I = 2, 3, 4, ..., 11, do the next 2 lines | arrondi

AA0 = APF(I) << 15

APF(I) = RND(AA0)

If NLSAPF = 15, do the next 3 lines | Cas 3: NLSAPF = 15, décalage à 14,

For I = 2, 3, 4, ..., 11, do the next 2 lines | arrondi

AA0 = APF(I) << 14

APF(I) = RND(AA0)

On notera ce qui suit à propos de ce code: les boucles «For» contenant trois instructions «If NLSAPF = ...» peuvent être supprimées si le code est réécrit en totalité pour chacune des trois valeurs possibles de NLSAPF. Ce code plus long donnera exactement les mêmes résultats, mais il fonctionnera plus rapidement sur la plupart des dispositifs programmables.

G.4 Représentations des variables de traitement interne LD-CELP

On trouvera dans le présent paragraphe, une mise à jour des Tableaux 1/G.728 et 2/G.728. Le Tableau G.1 est une version abrégée du Tableau 1/G.728. Il donne seulement la liste des constantes qui ne sont pas intrinsèquement des nombres entiers et qui ne figurent pas par ailleurs dans la spécification. Dans le Tableau 1/G.728, on a supprimé les colonnes Symbole équivalent et Valeur (initiale), afin de faire de la place pour le format avec virgule fixe et la représentation nécessaires pour chaque variable. Le Tableau G.2 reprend intégralement le Tableau 2/G.728. Ici aussi, on a supprimé la même colonne du Tableau 2/G.728, afin de présenter le format avec virgule fixe. Par ailleurs, on a introduit un certain nombre de variables nouvelles qui se rattachent exclusivement à la spécification avec virgule fixe.

TABLEAU G.1/G.728

Paramètres de base du codeur qui ne sont pas intrinsèquement des entiers et qui ne figurent pas par ailleurs

Nom	Valeur avec virgule flottante	Valeur avec virgule fixe	Format Q	Description
AGCFAC	0,99	16220	Q14	Facteur de commande de la rapidité d'adaptation du contrôle automatique de gain
AGCFAC1	0,01	20972	Q21	Valeur de (1 – AGCFAC)
GOFF	32	16384	Q9	Valeur d'écart du gain logarithmique
PPFTH	0,6	9830	Q14	Seuil pour désactiver le postfiltre fondamental
PPFZCF	0,15	9830	Q16	Facteur de commande des zéros du postfiltre fondamental
TAPTH	0,4	26214	Q16	Seuil de recalage de la période fondamentale
TILTF	0,15	4915	Q15	Facteur de commande de compensation du niveau spectral

TABLEAU G.2/G.728

Variables de traitement interne LD-CELP

Nom	Plage d'index de table	Format avec virgule fixe	Description
A	1 à LPC + 1	Q14	Coefficients du filtre de synthèse
AL	1 à 3	Q13	Coefficients du dénominateur de filtre passe-bas à 1 kHz
AP	1 à 11	Q14	Coefficients du dénominateur de postfiltre à court terme
APF	1 à 11	Q13	Coefficients de filtre LPC du 10 ^e ordre
ATMP	1 à LPC + 1	Q13/Q14/Q15	Tampon temporaire pour les coefficients du filtre de synthèse
AWP	1 à LPCW + 1	Q14	Coefficients du dénominateur de filtre de pondération perceptive
AWZ	1 à LPCW + 1	Q14	Coefficients du numérateur de filtre de pondération perceptive
AWZTMP	1 à LPCW + 1	Q13/Q14/Q15	Tampon temporaire pour les coefficients du filtre de pondération
AZ	1 à 11	Q14	Coefficients du numérateur de postfiltre à court terme
B	1	Q16	Coefficients du postfiltre à long terme
BL	1 à 4	Q19	Coefficients du numérateur de filtre passe-bas à 1 kHz
D	-139 à 100	Q1	Résidu de prédiction LPC
DEC	-34 à 25	Q1	Résidu de prédiction LPC, décimé à 4:1
ET	1 à IDIM	15b BFL	Vecteur d'excitation normalisé en gain
FACV	1 à LPC + 1	Q14	Vecteur d'extension de la largeur de bande du filtre de synthèse
FACGPV	1 à LPCLG + 1	Q14	Vecteur d'extension de la largeur de bande du prédicteur de gain
G2	1 à NG	Q12	Deux fois les niveaux de gain dans le répertoire de gain
GAIN	1	SFL	Gain d'excitation linéaire
GB	1 à NG - 1	Q13	Point médian entre niveaux de gain adjacents
GL	1	Q14	Facteur de normalisation du postfiltre à long terme
GLB	1	Q16	Terme de produit du postfiltre à long terme
GP	1 à LPCLG + 1	Q14	Coefficient du prédicteur de gain logarithmique, valeur initiale = 16384, -16384, 0, ..., 0
GPTMP	1 à LPCLG + 1	Q13/Q14/Q15	Table temporaire de coefficient du prédicteur linéaire de gain logarithmique
GQ	1 à NG	Q13	Niveaux de gain dans le répertoire de gain
GSQ	1 à NG	Q11	Carrés des niveaux de gain dans le répertoire de gain
GSTATE	1 à LPCLG	Q9	Registre du prédicteur linéaire de gain logarithmique, valeur initiale = -16384
GTMP	1 à 4	Q9	Tampon temporaire de gain logarithmique, valeur initiale = -16384
H	1 à IDIM	Q13	Vecteur de réponse impulsionnelle de F(z) W(z)
ICHAN	1	Q0	Meilleur index de répertoire d'excitation à transmettre
ICOUNT	1	Q0	Compteur de vecteurs de parole (indexé de 1 à 4)
IG	1	Q0	Meilleur index à 3 bits du répertoire de gain
ILLCOND	1	Q0	Fanion de défaut de conditionnement pour le filtre de synthèse
ILLCONDG	1	Q0	Fanion de défaut de conditionnement pour le prédicteur de gain logarithmique
ILLCONDP	1	Q0	Fanion de défaut de conditionnement pour le postfiltre
ILLCONDW	1	Q0	Fanion de défaut de conditionnement pour le filtre de pondération
IP	1	Q0	Pointeur d'adresse vers le résidu de prédiction LPC
IS	1	Q0	Meilleur index à 7 bits du répertoire de forme
KP	1	Q0	Période fondamentale de la trame courante
KP1	1	Q0	Période fondamentale de la trame précédente

TABLEAU G.2/G.728 (suite)

Variables de traitement interne LD-CELP

Nom	Plage d'index de table	Format avec virgule fixe	Description
LOGGAIN	1	Q9	Logarithme du gain d'excitation
LPIIFIR	3	Q1	Mémoire FIR du filtre passe-bas
LPIIIR	3	Q1	Mémoire IIR du filtre passe-bas
NLSATMP	1	Q0	Fanion de précision de la récurrence de Durbin pour ATMP
NLSAWZTMP	1	Q0	Fanion de précision de la récurrence de Durbin pour AWZTMP
NLSGPTMP	1	Q0	Fanion de précision de la récurrence de Durbin pour GPTMP
NLSET	1	Q0	NLS pour ET
NLSGAIN	1	Q0	NLS pour GAIN linéaire
NLSREXP	1	Q0	NLS pour REXP, valeur initiale = 31
NLSREXPPLG	1	Q0	NLS pour REXPPLG, valeur initiale = 31
NLSREXPW	1	Q0	NLS pour REXPW, valeur initiale = 31
NLSSB	21	Q0	NLS pour SB, valeur initiale = 16
NLSST	1	Q0	NLS pour ST dans le décodeur
NLSSTATE	11	Q0	NLS pour STATELPC, valeur initiale = 16
NLSSTTMP	4	Q0	NLS pour STTMP, valeur initiale = 16
PN	1 à IDIM	Q7	Vecteur de corrélation pour la recherche dans le répertoire
PTAP	1	Q14	Pondérateur de prédicteur tonal calculé par le bloc 83
R	1 à 11	BFL	Coefficients d'autocorrélation
RC	1	Q15	Facteur de réflexion
RC1	1	Q15	Tampon temporaire du premier facteur de réflexion
REXP	1 à LPC + 1	BFL	Partie récurrente de l'autocorrélation – Filtre de synthèse
REXPPLG	1 à LPCLG + 1	BFL	Partie récurrente de l'autocorrélation – Prédicteur de gain logarithmique
REXPW	1 à LPCW + 1	BFL	Partie récurrente de l'autocorrélation – Filtre de pondération
RTMP	1 à LPC + 1	BFL	Tampon temporaire pour coefficients d'autocorrélation
S	1 à IDIM	15b Q2	Vecteur de parole d'entrée MIC uniforme
SB	1 à 105	14b BFL	Tampon pour signaux de parole préalablement quantifiés
SBLG	1 à 34	Q9	Tampon pour le gain logarithmique précédent
SBW	1 à 60	Q2	Tampon pour le signal de parole d'entrée précédent
SCALE	1	SFL	Facteur de normalisation de postfiltre non filtré
SCALEFIL	1	Q14	Facteur de normalisation de postfiltre filtré en passe-bas, valeur initiale = 16384
SD	1 à IDIM	Q0	Tampon de parole décodé
SPF	1 à IDIM	Q2	Vecteur de parole postfiltré
SPFPCFV	1 à 11	Q14	Vecteur de commande des pôles du postfiltre à court terme
SPFZCFV	1 à 11	Q14	Vecteur de commande des zéros du postfiltre à court terme
SO	1	octet	Echantillon de parole d'entrée MIC loi A ou loi μ
SST (past)	-239 à 0	13b Q0	Tampon de signaux de parole quantifiés
SST (current)	1 à IDIM	15b Q2	Tampon de signaux de parole quantifiés
ST	1 à IDIM	14b BFL	Vecteur de parole quantifié
STATELPC	1 à LPC	14b SBFL	Mémoire du filtre de synthèse
STLPCI	1 à 10	Q2	Mémoire du filtre inverse LPC

TABLEAU G.2/G.728 (fin)

Variables de traitement interne LD-CELP

Nom	Plage d'index de table	Format avec virgule fixe	Description
STMP	1 à 4 * IDIM	15b Q2	Tampon pour fenêtre hybride du filtre de pondération perceptive
STTMP	1 à 4 * IDIM	14b SBFL	Tampon pour fenêtre hybride du filtre de synthèse
STPFIR	1 à 10	Q2	Mémoire du postfiltre à court terme – section tous zéros
STPFIR	1 à 10	Q2	Mémoire du postfiltre à court terme – section tous pôles
SU	1	Q2	Echantillon de parole d'entrée MIC uniforme
SUMFIL	1	Q2	Somme des valeurs absolues du signal de parole postfiltré
SUMUNFIL	1	Q2	Somme des valeurs absolues du signal de parole décodé
SW	1 à IDIM	Q2	Vecteur de parole issu du filtre de pondération perceptive
TARGET	1 à IDIM	BFL	Vecteur VQ cible (normalisé en gain)
TEMP	1 à IDIM	*	Table de travail pour espace de travail temporaire
TILTZ	1	Q14	Coefficient de compensation d'écart de niveau du postfiltre à court terme
WFIR	1 à LPCW	Q2	Mémoire du filtre de pondération 4 – section tous zéros
WIIR	1 à LPCW	Q2	Mémoire du filtre de pondération 4 – section tous pôles
WNR	1 à 105	Q15	Fonction de fenêtrage pour filtre de synthèse
WNRLG	1 à 34	Q15	Fonction de fenêtrage pour prédicteur de gain logarithmique
WNRW	1 à 60	Q15	Fonction de fenêtrage pour filtre de pondération
WPCFV	1 à LPCW + 1	Q14	Vecteur de commande des pôles du filtre de pondération perceptive
WS	1 à 105	#	Table d'espace de travail pour variables intermédiaires
WZCFV	1 à LPCW + 1	Q14	Vecteur de commande des zéros du filtre de pondération perceptive
Y	1 à IDIM * NCWD	Q11	Table du répertoire de forme
Y2	1 à NCWD	Q5	Energie du vecteur code de forme convolué
ZIR	1 à IDIM	15b Q2	Réponse en entrée nulle
ZIRWFIR	1 à LPCW	15b Q2	Mémoire du filtre de pondération 10 – section tous zéros
ZIRWIIR	1 à LPCW	15b Q2	Mémoire du filtre de pondération 10 – section tous pôles
SFL	Virgule flottante scalaire (<i>scalar floating point</i>)		
BFL	Virgule flottante de bloc (<i>block floating point</i>)		
SBFL	Virgule flottante de bloc segmenté (<i>segmented block floating point</i>)		
Qx	Format Qx		
14b ou 15b	Indique une précision de 14 ou 15 bits; dans tous les autres cas, on admet par hypothèse la précision totale de 16 bits		
#	Défini par l'usage; ce peut être BFL ou Q fixe, car il s'agit d'une mémoire de travail		
*	TEMP est une table temporaire de travail qui est utilisée dans plusieurs blocs; son format Q peut changer d'un bloc à l'autre.		

G.5 Tables de gain logarithmique pour les vecteurs des répertoires de gain et de forme codés

Voir les Tableaux G.3 et G.4.

TABLEAU G.3/G.728

Gain en virgule flottante (dB) et représentation en virgule fixe, format Q11, pour les vecteurs de répertoire de gain codé

Index		dB	Virgule fixe
0	4	-5,7534180	-11783
1	5	-0,8925781	-1828
2	6	3,9682620	8127
3	7	8,8291020	18082
NOTE – Pour obtenir la valeur en virgule fixe, multiplier la valeur en virgule flottante par 2048 = 2 ¹¹ .			

TABLEAU G.4/G.728

**Gain en virgule flottante (dB) et représentation en virgule fixe,
format Q11, pour les vecteurs de répertoire de forme codé**

Index	dB	Virgule fixe	Index	dB	Virgule fixe	Index	dB	Virgule fixe
0	-0,1108398	-227	43	1,1064450	2266	85	1,7133790	3509
1	5,0332030	10308	44	7,0932620	14527	86	0,4252930	871
2	3,1977540	6549	45	9,1738280	18788	87	1,0693360	2190
3	3,7856450	7753	46	6,3623050	13030	88	2,7080080	5546
4	3,7094730	7597	47	3,0458980	6238	89	7,4887700	15337
5	8,0874020	16563	48	0,8911133	1825	90	1,8105470	3708
6	3,1279300	6406	49	4,4384770	9090	91	1,1748050	2406
7	5,8266600	11933	50	0,1030273	211	92	2,8076170	5750
8	6,6254880	13569	51	0,9218750	1888	93	3,6806640	7538
9	5,1606450	10569	52	8,8320310	18088	94	1,9101560	3912
10	7,9726560	16328	53	11,0141600	22557	95	1,7299800	3543
11	3,1914060	6536	54	5,3188480	10893	96	-4,9335940	-10104
12	7,7163090	15803	55	8,8652340	18156	97	0,1479492	303
13	5,6997070	11673	56	1,6728520	3426	98	-3,0083010	-6161
14	10,4091800	21318	57	6,5429690	13400	99	-0,5576172	-1142
15	4,4433590	9100	58	-2,1362300	-4375	100	1,8881840	3867
16	5,9790040	12245	59	3,8916020	7970	101	2,8979490	5935
17	5,8681640	12018	60	3,7861330	7754	102	-3,5161130	-7201
18	1,2221680	2503	61	12,3388700	25270	103	-0,3706055	-759
19	7,1728520	14690	62	2,5942380	5313	104	-1,0219730	-2093
20	8,8818360	18190	63	7,6245120	15 615	105	-1,3979490	-2863
21	14,0629900	28801	64	-3,0742190	-6296	106	1,0825200	2217
22	8,2045900	16803	65	2,2021480	4510	107	-1,5834960	-3243
23	9,9272460	20331	66	1,0751950	2202	108	3,0083010	6161
24	8,7983400	18019	67	-3,5297850	-7229	109	2,8579100	5853
25	12,1679700	24920	68	1,5361330	3146	110	3,7104490	7599
26	7,8901370	16159	69	-1,3759770	-2818	111	3,2944340	6747
27	8,6025390	17618	70	-1,3056640	-2674	112	-0,9770508	-2001
28	11,2656300	23072	71	-0,7651367	-1567	113	4,9892580	10218
29	13,7085000	28075	72	0,8989258	1841	114	-0,0263672	-54
30	9,3598630	19169	73	2,8334960	5803	115	0,9335938	1912
31	12,5600600	25723	74	3,8203130	7824	116	5,6127930	11495
32	4,2333980	8670	75	0,1557617	319	117	5,1635740	10575
33	4,9165040	10069	76	0,8862305	1815	118	2,2055660	4517
34	0,2456055	503	77	0,8618164	1765	119	2,0893550	4279
35	4,2221680	8647	78	3,3930660	6949	120	0,8852539	1813
36	5,4516600	11165	79	1,2128910	2484	121	0,2763672	566
37	9,0073240	18447	80	1,3710940	2808	122	2,2309570	4569
38	2,0820310	4264	81	4,7431640	9714	123	2,0278320	4153
39	8,4868160	17381	82	-2,0581050	-4215	124	1,6445310	3368
40	1,7241210	3531	83	3,2607420	6678	125	5,4584960	11179
41	5,1479490	10543	84	1,2861330	2634	126	0,8271484	1694
42	-1,1679690	-2392				127	0,3715820	761

NOTE – Pour obtenir la valeur en virgule fixe, multiplier la valeur en virgule flottante par 2¹¹.

G.6 Valeurs entières des tables relatives au répertoire de gain codé

On trouvera ci-dessous les valeurs entières équivalentes pour le tableau de valeurs avec virgule flottante figurant dans l'Annexe B/G.728 (voir le Tableau G.5).

TABLEAU G.5/G.728

Valeurs entières des tables relatives au répertoire de gain codé

Index de table	1	2	3	4	5	6	7	8
GQ (Q13)	4224	7392	12936	22638	-4224	-7392	-12936	-22638
GB (Q13)	5808	10164	17787	*	-5808	-10164	-17787	*
G2 (Q12)	4224	7392	12936	22638	-4224	-7392	-12936	-22638
GSQ (Q11)	545	1668	5107	15640	545	1668	5107	15640

* Peut être une valeur arbitraire quelconque (non utilisé).

G.7 Pseudo-codes pour le programme principal du codeur et du décodeur

On trouvera dans ce paragraphe les pseudo-codes relatifs respectivement au programme principal du codeur et au programme principal du décodeur. Il s'agit essentiellement d'indiquer l'ordre dans lequel les divers blocs sont exécutés. En conséquence, seule est spécifiée la séquence d'exécution des blocs, à l'exclusion de renseignements banals sur le transfert des paramètres. A noter que la séquence admissible d'exécution n'est pas unique. Il existe de nombreux ordres d'exécution différents qui donnent tous un résultat exact pour les bits. Les deux pseudo-codes développés ci-dessus ne sont que des exemples. Toutefois, en cas d'utilisation d'un ordre différent pour l'exécution des blocs, le responsable de la mise en œuvre devra vérifier qu'il obtient des résultats exacts en bits.

Ci-après, le pseudo-code pour le programme principal du codeur.

Initialize all encoder variables to their initial values.

Initialize $y_2()$ by executing blocks 14 and 15 with $h = [8192, 0, 0, 0, 0]$

ILLCOND = .FALSE.

ILLCONDW = .FALSE.

ILLCONDG = .FALSE.

ICOUNT = 0

VEC_LOOP:

```

    If ICOUNT = 4, set ICOUNT = 0                | Initialiser le compteur de vecteurs
    ICOUNT = ICOUNT + 1                          | Mettre à jour le compteur de vecteurs
    Get one vector of input speech from the input buffer
    Convert input speech vector to the range [-16384, +16383],
    then assign to S()                            | Représentation en Q2 de [-4096, +4095,75]

                                                | Vérifier, si nécessaire mettre à jour les
                                                | coefficients du filtre

    If ICOUNT = 3, do the next 4 lines
        If ILLCOND = .FALSE., do block 51
        If ILLCONDW = .FALSE., do block 38
        do block 12
        do blocks 14 and 15

    If ICOUNT = 2 and ILLCONDG = .FALSE., then do block 45

                                                | Commencer le traitement «une fois par vecteur»
    do blocks 46, 98, 99, and 48                  | Déterminer le gain adapté en boucle
                                                | GSTATE(1:9) est décalé d'une position vers le bas

    do "blockzir" (blocks 9 and 10 during zero-input response calculation)
    do block 4                                    | Filtre de pondération perceptive
    do block 11                                  | Calcul du vecteur cible VQ

```

do block 16	Normalisation du vecteur cible VQ
do block 13	Convolution à inversion temporelle
do blocks 17 and 18	Recherche dans le répertoire d'excitation
put out ICHAN to the communication channel	
do blocks 19 and 21	Normaliser le vecteur code d'excitation choisi
do blocks 9 and 10 during filter memory update	Déterminer ST()
do blocks 93, 94, 96, and 97	Mettre à jour le gain logarithmique; à noter
	que les 3 unités de délai de l'adaptateur de
	gain interviennent naturellement dans le
	processus de mise en boucle et qu'il est
	inutile de les mettre en œuvre explicitement
	Mettre à jour la mémoire du prédicteur de gain
	I = adresse de départ de STTMP()
	Mettre à jour STTMP()
GSTATE(1) = output of block 97	
I = (ICOUNT - 1) * IDIM	
copy ST(1:5) to STTMP(I + 1:I + 5)	
NLSSTTMP(ICOUNT) = NLSST	
I = (ICOUNT - 3) * IDIM	
If ICOUNT < 3, set I = I + 20	I = adresse de départ de STMP()
copy S(1:5) to STMP(I + 1:I + 5)	Mettre à jour STMP()
	Fin du traitement «une fois par vecteur»
	Début du traitement «une fois par trame»
If ICOUNT = 4, do the next 2 lines	
do block 49	Fanion de défaut de conditionnement
	de sortie = ILLCOND
do block 50	Coeff. du prédicteur de sortie = ATMP()
	Fanion de défaut de conditionnement
	de sortie = ILLCOND
If ICOUNT = 2, do the next 2 lines	
do block 36	Fanion de défaut de conditionnement
	de sortie = ILLCONDW
do block 37	Coeff. du prédicteur de sortie = AWZTMP()
	Fanion de défaut de conditionnement
	de sortie = ILLCONDW
If ICOUNT = 1, do the next 6 lines	
GTMP(1) = GSTATE(4)	Mettre à jour GTMP() en une seule fois
GTMP(2) = GSTATE(3)	
GTMP(3) = GSTATE(2)	
GTMP(4) = GSTATE(1)	
do block 43	Fanion de défaut de conditionnement
	de sortie = ILLCONDG
do block 44	Coeff. du prédicteur de sortie = GPTMP()
	Fanion de défaut de conditionnement
	de sortie = ILLCONDG
	Fin du traitement «une fois par trame»

Go to VEC_LOOP

Ci-après, maintenant, le pseudo-code pour le programme principal du décodeur. Ici encore, on a spécifié uniquement la séquence d'exécution des blocs, sans renseignements banals sur le transfert des paramètres.

Initialize all decoder variables to their initial values.

```
ILLCOND = .FALSE.
ILLCONDG = .FALSE.
ILLCONDP = .FALSE.
ICOUNT = 0
```

VEC_LOOP:

If ICOUNT = 4, set ICOUNT = 0	Réinitialisation du compteur de vecteurs
ICOUNT = ICOUNT + 1	Mise à jour du compteur de vecteurs
Get ICHAN of the current vector from the input buffer	
Obtain the shape index IS and gain index IG from ICHAN	
	Vérifier, si nécessaire mettre à jour les
	coefficients du filtre

<p>If ICOUNT = 3, do the next line If ILLCOND = .FALSE., do block 51</p> <p>If ICOUNT = 2 and ILLCONDG = .FALSE., then do block 45</p> <p>do blocks 46, 98, 99, and 48</p> <p>do blocks 19 and 21 do block 32</p> <p>If ICOUNT = 1, do block 85</p> <p>do block 81</p> <p>If ICOUNT = 3, do the next 3 lines do block 82 do block 83</p> <p> do block 84</p> <p>do block 71 do block 72 do blocks 73 and 74 do block 75 do block 76 do block 77</p> <p>do blocks 93, 94, 96, and 97</p> <p>GSTATE(1) = output of block 97 I = (ICOUNT - 1) * IDIM copy ST(1:5) to STTMP(I + 1:I + 5) NLSSTTMP(ICOUNT) = NLSST</p> <p>If ICOUNT = 4, do the next 5 lines do block 49</p> <p> do block 50, order 1 to 10</p> <p> NLSAPF = NLSATMP copy ATMP(2:11) to APF(2:11) continue block 50, order 11 to 50</p> <p>If ICOUNT = 1, do the next 6 lines GTMP(1) = GSTATE(4) GTMP(2) = GSTATE(3) GTMP(3) = GSTATE(2) GTMP(4) = GSTATE(1) do block 43</p> <p> do block 44</p>	<p> Commencer le traitement «une fois par vecteur» Déterminer le gain adapté en boucle GSTATE(1:9) est décalé d'une position vers le bas Normaliser le vecteur code d'excitation choisi</p> <p> Mettre à jour les coefficients du postfiltre à court terme</p> <p> Extraction de la période fondamentale Calculer les poids des coefficients du prédicteur de tonie Mettre à jour les coefficients du postfiltre à long terme Postfiltre à long terme Postfiltre à court terme Calculer les sommes des valeurs absolues Rapport des sommes des valeurs absolues Filtre passe-bas du facteur de pondération Commande de gain de la sortie du postfiltre</p> <p> Mise à jour du gain logarithmique; à noter que les 3 unités de délai de l'adaptateur de gain interviennent naturellement dans le processus de mise en boucle et qu'il est inutile de les mettre en œuvre explicitement Mettre à jour la mémoire du prédicteur de gain I = adresse de départ de STTMP() Mettre à jour STTMP() Fin du traitement «une fois par vecteur»</p> <p> Commencer le traitement «une fois par trame»</p> <p> Fanion de défaut de conditionnement de sortie = ILLCOND</p> <p> Coeff. du prédicteur de sortie = ATMP() avec NLSATMP Fanion de défaut de conditionnement de sortie = ILLCONDP Sauvegarder le prédicteur du 10^e ordre pour utilisation ultérieure du postfiltre Continuer jusqu'à la fin du bloc 50 Coeff. du prédicteur de sortie = ATMP() avec NLSATMP Fanion de défaut de conditionnement de sortie = ILLCOND</p> <p> Mettre à jour GTMP() en une seule fois</p> <p> Fanion de défaut de conditionnement de sortie = ILLCONDG Coeff. du prédicteur de sortie = GPTMP() Fanion de défaut de conditionnement de sortie = ILLCONDG Fin du traitement «une fois par trame»</p>
--	--

Go to VEC_LOOP