UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

ESTUDO DE PSICOACÚSTICA E SUAS APLICAÇÕES

Autor: Luiz Fabiano Nogueira de Moura

Orientador: Luiz Wagner Pereira Biscainho, D.Sc.

Examinador: Eduardo A. B. da Silva, Ph.D.

Examinador: Sidney C. de Lucena, D.Sc.

DEL
Abril/2006

Agradecimentos

Esta é a parte do trabalho mais importante para o autor e talvez a mais entediante para o leitor, porém sem essas pessoas este trabalho e toda a minha trajetória na Universidade não teriam sido realizados.

Começo agradecendo aos meus pais, que me deram a base de caráter e dignidade e que me incutiram desde muito cedo a importância do conhecimento e por terem acreditado no meu sonho e segurado a barra destes anos. Agradeço a minha família pela convivência, carinho e apoio. Aos meus amigos(as) que acompanharam esta trajetória, mesmo os que muitas vezes acabei negligênciando devido às horas que tive que dedicar aos estudos e aos trabalhos, em especial aos meus amigos e companheiros da banda Ventania; sem as horas de ensaio para relaxar tudo teria sido mais difícil. Agradeço aos meus colegas de faculdade pelo convívio que tivemos nestes anos todos, em especial as colegas Gustavo Barbosa e Lucas Lopes pelos grupos de estudo e pelos trabalhos em conjunto e sobretudo pela amizade que se formou entre nós. Também devo agradecer ao grupo de áudio do LPS, em especial aos colegas Bruno Carluccio, Fábio Freeland e Cristiano Santos, aprendi muito com todos vocês. Como não poderia deixar de ser agradeço ao meu Orientador e Mestre Luiz Wagner, aquele que mesmo sem saber me deu a mão quando pensei em desistir de tudo e que acreditou na minha capacidade mesmo quando eu não acreditava. A ele agradeço a dedicação como professor, a amizade do amigo e os conselhos, elogios e broncas como orientador.

Finalmente, e de um modo egoísta, agradeço à música, à ciência e à tecnologia, as minhas três maiores paixões, que carrego no peito desde a infância e que sempre serviram como norte da minha bússola e foram determinantes em todas as grandes escolhas da minha vida.

Resumo

A psicoacústica, ciência que estuda o modo como o nosso cérebro percebe os sons, tem mostrado grande importância em aplicações tais como restauração de som, áudio 3D, algoritmos de codificação e compressão de arquivos de áudio, entre outros. Tais possibilidades só foram atingidas graças aos estudos e testes desenvolvidos desde o século XVIII por diversos pesquisadores. Há diversos efeitos observados à luz da psicoacústica, como, por exemplo, o mascaramento, o limiar de audição e a percepção de *pitch*. Com o objetivo de formar uma base de estudos psicoacústicos, foram idealizadas duas frentes de trabalhos. A primeira é um sistema de *scripts* feitos em MatLab© que reproduz experimentos já consagrados em diversas pesquisas associadas à percepção *pitch* de modo a possibilitar sua comprovação e a geração de estatísticas próprias. A segunda frente se insere no contexto do limiar de audição e do mascaramento, e consiste na realização de um codificador de áudio MPEG-1 - Layer I.

Palavras chave

Áudio, psicoacústica, percepção de *pitch*, mascaramento, MPEG-1.

Sumário

Capítulo 1

Introdução

Efeitos de ilusão de ótica são bastante conhecidos por boa parte das pessoas, e mesmo os que não sabem explicar como ocorrem alguns destes efeitos geralmente os conhecem. Entretanto, da mesma forma que o nosso sistema visual processa os sinais recebidos por nossos olhos de um modo que nos permite até "brincar" com a nossa visão, viabilizando, assim, o cinema e a televisão, entre outras mídias, o sistema auditivo humano também possui características bastante peculiares que nos permitem afirmar que nem tudo que se ouve foi produzido por uma determinada fonte sonora, assim como nem tudo que é gerado por tal fonte será ouvido.

Este fato é mais do que suficiente para justificar o estudo de tais efeitos e as suas aplicações, e à luz disso surgiu o interesse de pesquisar sobre a psicoacústica. Dividimos o trabalho em duas partes, percepção de *pitch* e mascaramento, passando assim por dois ramos, talvez os mais importantes, desta área.

Para a primeira parte resolvemos construir um conjunto de programas com interface amigável ao usuário e com parâmetros flexíveis que reproduzisse alguns dos testes específicos da percepção de *pitch* - para sua melhor compreensão, para gerar uma base de dados estatísticos ou ainda para permitir realização de outros testes com a finalidade de se tirar ou confirmar alguma conclusão.

Já a segunda parte consiste da construção de um codificador de áudio de acordo com o padrão MPEG-1 *Layer* I, que, como veremos mais tarde, tem a sua etapa mais importante para a codificação do áudio baseada nos princípios do mascaramento. O *Layer* 3 é o mais conhecido deste padrão, vulgarmente chamado de *mp3*, porém a escolha pelo *Layer* I foi feita por motivos didáticos, devido a sua simplicidade quando comparado ao *mp3*, por não acharmos codificadores e/ou decodificadores com a mesma facilidade que o fazemos com o mesmo *mp3*, e também porque devido à sua semelhança com o *Layer* 2 é possível, posteriormente, acrescentar este *Layer* ao codificador que construímos com uma facilidade razoável.

Capítulo 2

Sistema para Avaliação de Percepção de *Pitch*

2.1 Percepção de *Pitch* - Fundamentos Teóricos e Práticos

2.1.1 Definição de *Pitch*

"*Pitch* é um atributo da sensação auditiva segundo o qual um som pode ser ordenado em uma escala estendendo-se de baixo a alto. O *pitch* depende principalmente das freqüências contidas no estímulo sonoro, mas também depende da pressão sonora e da forma de onda do som."(ANSI Standard, 1994)

A ênfase nas freqüências contidas distingue a sensação de *pitch* da sensação de intensidade sonora, ou audibilidade (*loudness*). Essa definição não é exatamente o que a psicoacústica quer dizer quando fala em *pitch*. Normalmente a palavra *pitch* é limitada a se referir em quão baixo ou alto um determinado som se encontra em uma escala musical. Isto não significa que a determinação de *pitch* é restrita a sons musicais. Significa, sim, que a dimensão psicológica que o termo *pitch* implica é a mesma dimensão psicológica da melodia. A distinção feita por esta restrição adicional é o que diferencia *pitch* de timbre. Timbres são informações sonoras adicionais que nos permitem distinguir um piano de um violão, por exemplo, ou se um som tem mais "brilho" do que outro etc.

Uma definição operacional de *pitch* seria dizer que um som tem um certo *pitch* se pudermos ajustar a freqüência de uma senóide pura, com uma amplitude qualquer, até "igualarmos" suas alturas percebidas.

Há duas grandes classes de modelos para percepção de *pitch*, o modelo de lugar e o modelo de tempo. Um modelo de percepção de *pitch* é uma tentativa de entendimento dos elementos fundamentais que caracterizam o *pitch*. Ele tenta explicar por que um sinal com certas características físicas gera uma sensação de *pitch* em particular.

2.1.2 Modelo do Lugar

Esta teoria começa com o mecanismo interno de audição (Fig. 2.1). Podemos resumir o processo da audição da seguinte forma: as ondas sonoras entram pelo pavilhão auricular (orelha), provocam a oscilação da membrana do tímpano, que por sua vez causa a vibração da cadeia ossicular (composta pelos ossos estribo, martelo e bigorna) e passam pelas células ciliadas da membrana basilar na cóclea (Fig. 2.2), fazendo que elas se dobrem e, com isso, liberem uma substância química. Essa substância química dispara um impulso nervoso até o cérebro.

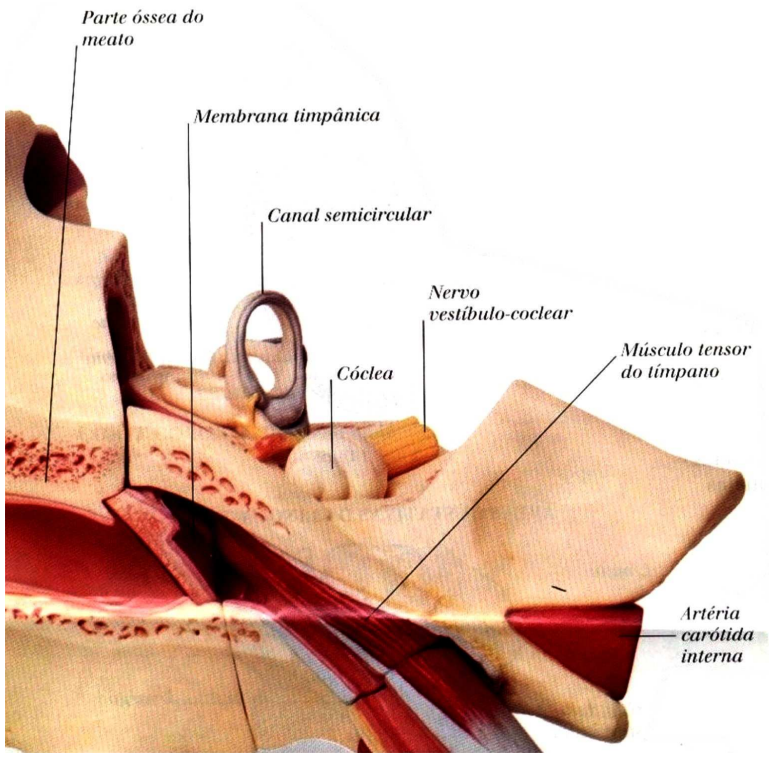

Figura 2.1: Vista em corte do sistema auditivo.

A região mais importante para a teoria do lugar é a das células ciliadas. Essas células são distribuídas ao longo do canal coclear (35 mm) e são distribuídas tonotopicamente, ou seja, indo da recepção de sons graves aos agudos (Tabela 2.1). Entretanto, o movimento destas células em diferentes lugares ao longo da membrana basilar indica uma excitação de neurônios diferentes no nervo auditivo. Assim a freqüência de um tom é representada em um código baseado em quais neurônios estão ativos e quais estão em repouso.

Nós podemos definir o padrão de excitação causado por um tom senoidal de uma dada freqüência e intensidade. Um padrão de excitação representa a atividade dos neurônios como função de uma variável tonotópica como o lugar de origem da membrana basilar. Um padrão de excitação hipotético é mostrado na Fig. 2.3 [1].

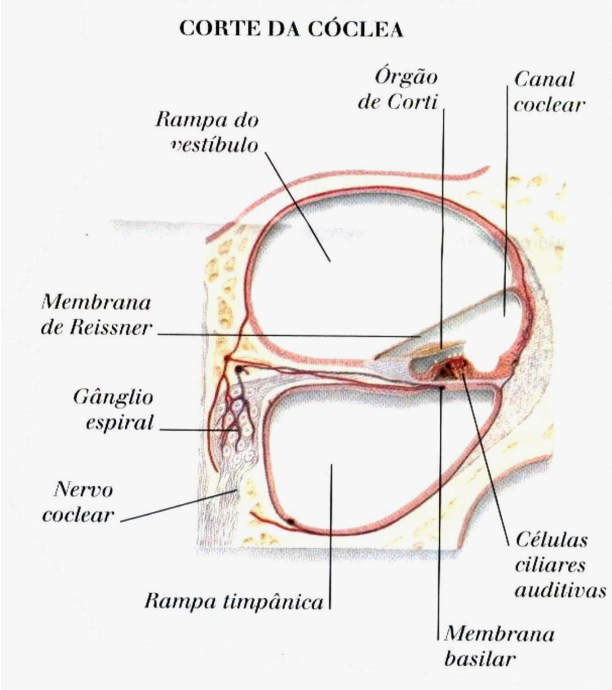

Figura 2.2: Vista em corte da Cóclea.

Uma das evidências associadas ao modelo do lugar vem da maleabilidade da sensação de *pitch*. Os *pitches* de tons senoidais mudam de um modo que poderia ser explicado por mudanças no padrão de excitação periférico. Por exemplo, o *pitch* de uma senóide é ligeiramente diferente em cada um dos ouvidos; esse efeito é chamado de diplacusia, e ocorre em todos os ouvidos normais devido a irregularidades nas cócleas individuais.

Outra evidência é que o *pitch* de uma senóide pode ser alterado mudando-se a intensidade desta. Este efeito pode ser observado em um experimento que procede do seguinte modo: O ouvinte escuta um

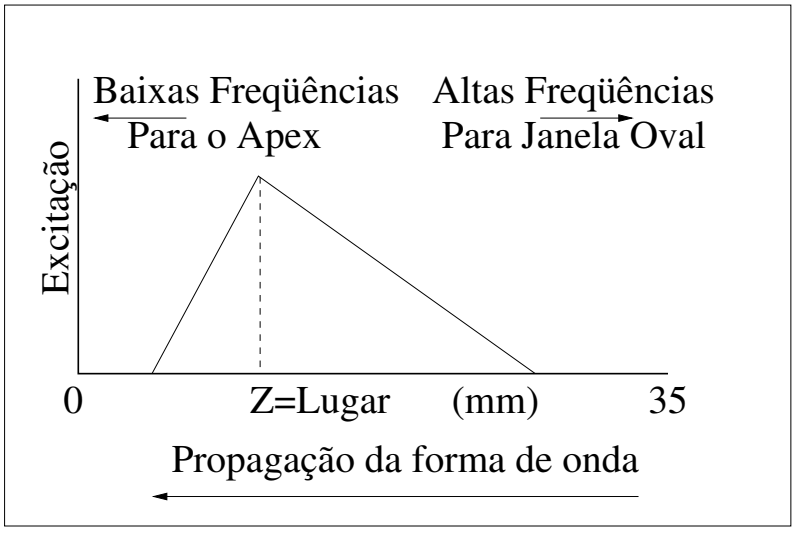

Figura 2.3: Hipotético padrão de excitação de um tom senoidal ao longo da membrana basilar.

Lugar(mm)	Freqüência (Hz)
3,00	85
8,00	333
13,00	829
18,00	1818
23,00	3790
28,00	7725
33,00	15575

Tabela 2.1: Freqüência referente à região excitada.

tom senoidal padrão com 40 dB de pressão sonora; posteriormente, ele tenta ajustar a freqüência de um segundo tom senoidal de potência sonora mais alta até que ele considere que o *pitch* percebido entre os tons seja igual. Posteriormente eleva-se a potência sonora do segundo tom e o processo se repete. A função *pitch* versus intensidade para freqüências particulares de senóides e ouvintes distintos mostra diferenças individuais devido a pequenas diferenças na cóclea. Mas a média sobre uma região de freqüências possui um padrão de ordenamento que é conhecido como regra de Stevens.

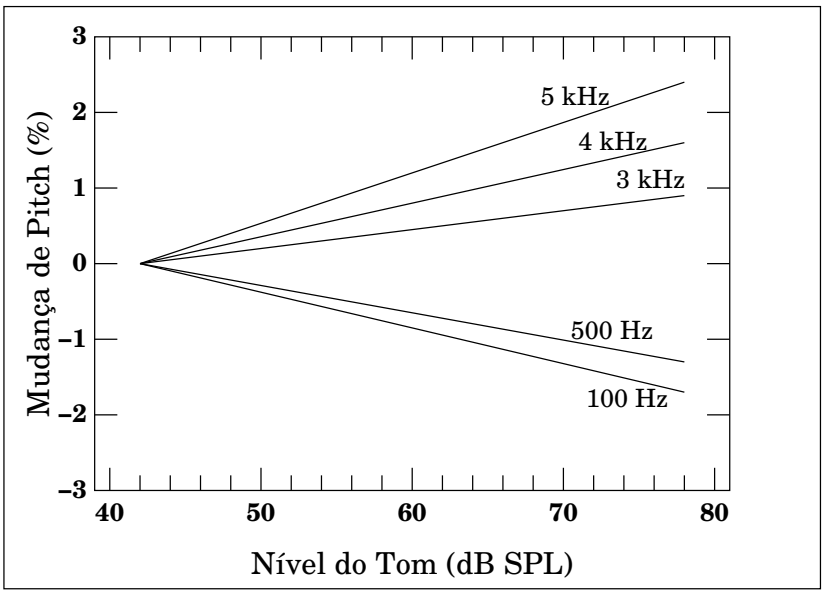

Figura 2.4: Nível do Tom x Mudança de *Pitch*.

A regra de Stevens indica que quando a intensidade aumenta o *pitch* de tons baixos decresce, e o *pitch* de tons altos sobe (Fig. 2.4). Isso ocorre porque a membrana basilar é um sistema neuromecânico não linear, e o padrão de excitação se move ao longo da membrana basilar com o aumento da intensidade [1]. Segundo observações fisiológicas, o padrão de movimento quanto ao sentido concorda com a regra de Stevens, ao menos para freqüências altas. Entretanto há um problema quanto ao pico do padrão de excitação, que é muito alterado (Fig. 2.5). Pode ocorrer uma alteração de lugar de excitação da membrana basilar que corresponde a até uma oitava, enquanto na verdade o *pitch* só é alterado por

uma pequena porcentagem [1].

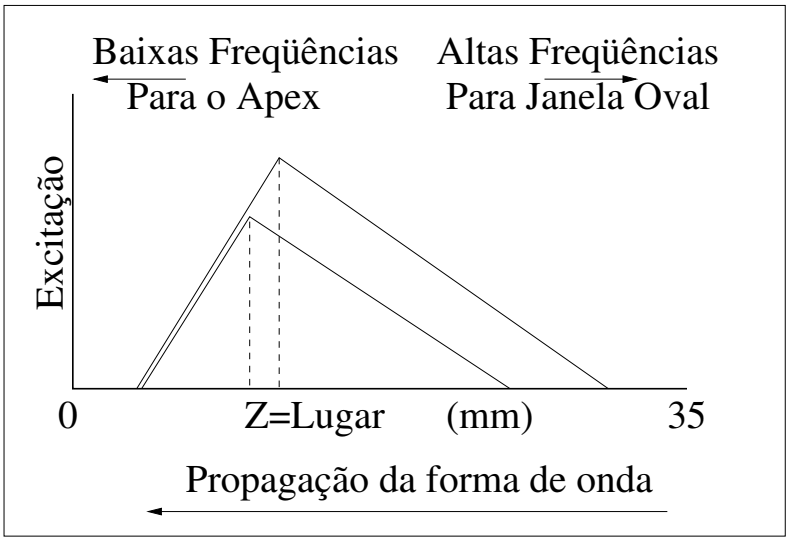

Figura 2.5: Padrão de excitação para dois tons com a mesma freqüência e níveis de intensidade diferentes.

Uma outra aparente conexão entre a fisiologia da cóclea e o *pitch* é a possibilidade de se mudar a freqüência pela pressurização da cadeia ossicular, que impõe uma pressão estática à janela oval.

2.1.3 Modelo do Tempo

Em adição ao princípio do lugar, no qual a freqüência de um tom é codificada tonotopicamente, o padrão de tempo dos pulsos neurais também pode codificar freqüências. Esse padrão pode ser observado no nervo auditivo conforme a intensidade de um tom aumenta.

Antes de haver um sinal, os neurônios no nervo auditivo já são estimulados espontaneamente, podendo atingir 100 picos por segundo, vindo em momentos aleatórios. À medida que o nível aumenta, os picos vão se ordenando no tempo e se sincronizando com o período do sinal. Conforme a intensidade do tom aumenta esses picos tornam-se mais numerosos, e o sincronismo entre os picos e o estímulo fica em torno de 90%. Este sincronismo depende fortemente da freqüência do tom [1].

O modelo do tempo assume que a sensação de *pitch* é determinada por este sincronismo neural. É possível perceber que há informações suficientes no padrão de tempo para estimar o nosso senso de *pitch*, mas, por causa de efeitos de alteração de *pitch* como a diplacusia e o efeito de intensidade, nós sabemos que o *pitch* não pode ser determinado somente pelo período de um trem de picos neurais. Este período está fortemente ligado ao período dos estímulos, para ser possível acomodar os efeitos de alteração de *pitch* já citados.

Algumas das mais fortes evidências do modelo do tempo vêm das dificuldades encontradas pelo

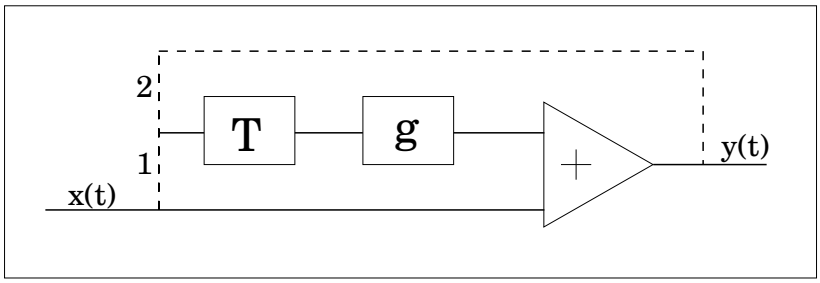

Figura 2.6: Diagrama em bloco de um sistema *delay-and-add.*

modelo do lugar. A teoria do lugar é baseada no padrão de excitação estabelecido por elementos neurais sintonizados, e compartilha algumas das limitações dos filtros sintonizados [1].

A teoria do tempo pode ser aplicada ao fenômeno conhecido como repetição de *pitch* (Fig. 2.6). Nesta figura x(t) representa um ruído gaussiano. Esta entrada x(t) é atrasada por T e atenuada pelo ganho g pelo caminho "1" ou pelo caminho "2". Quando um ruído branco é adicionado a sua versão atrasada, o resultado é uma sensação de *pitch* inversamente proporcional a esse atraso T.

Um outro caso interessante é quando modulamos um ruído branco por uma onda quadrada e alternamos a freqüência desta modulação com valores entre 150 e 300 Hz, uma vez por segundo. Ouvintes próximos ao alto-falante e mantidos no campo sonoro direto podem detectar uma mudança de *pitch* [1].

Neste caso específico, os estudiosos da psicoacústica têm dificuldades em decidir qual das teorias é aplicável. Parece que os dois mecanismos são envolvidos, ainda que o mecanismo de tempo tenda a dominar para as baixas freqüências e o de lugar para as altas.

2.1.4 *Pitch* de Tons Complexos

Há duas classes de tons complexos, os periódicos e os não-periódicos. O primeiro é característico de instrumentos musicais em geral, assim como das vogais da fala humana.

Tons complexos periódicos têm harmônicos, e seu *pitch* corresponde à freqüência fundamental, ou seja, se criamos um sinal que seja um somatório de senóides de 200 Hz e seus harmônicos (Fig. 2.7), o *pitch* encontrado é de 200 Hz. Isto cria um problema para a teoria do lugar, já que diferentes regiões das células ciliares serão excitadas, e não fica claro qual lugar poderia codificar o *pitch*. Por outro lado, isto não cria problemas para teoria do tempo, já que o período do tom é igual ao da fundamental, em concordância com o *pitch*.

A teoria do lugar poderia escapar deste problema com uma regra arbitrária que diria que para estes casos, a menor freqüência venceria; entretanto, se tirarmos do sinal anterior a fundamental (Fig. 2.8) e o

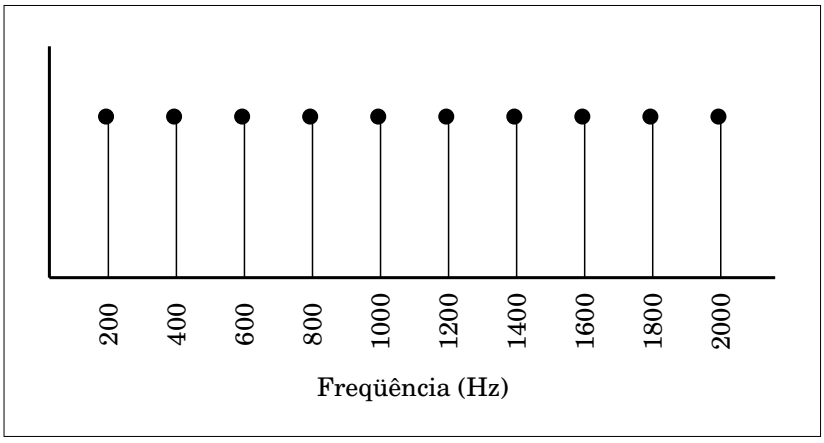

Figura 2.7: $\sum_{n=1}^{10} sen(200.n.2.\pi.t)$.

segundo harmônico, os testes indicam que os ouvintes ainda ouvem um *pitch* de 200 Hz. Isto é coerente com a teoria do tempo, já que o período ainda é de 1/200 s, o que está descrito nos detalhes de picos e vales da forma de onda resultante, ou ainda pelo envelope da estrutura total.

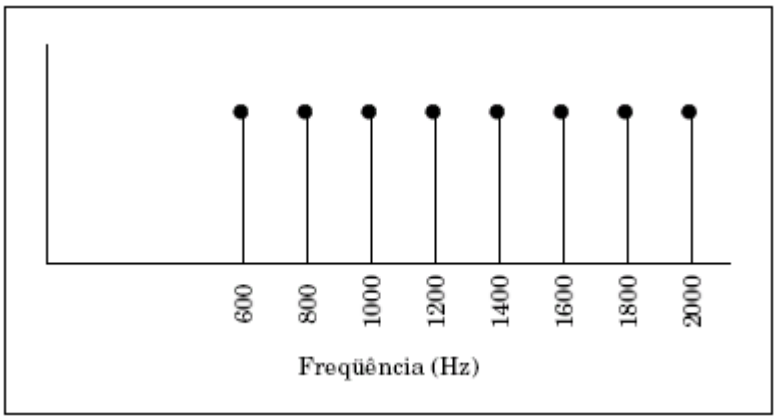

Figura 2.8: $\sum_{n=3}^{10} sen(200.n.2.\pi.t)$.

No caso do modelo de picos e vales, as formas de onda dependem das fases dos seus componentes. Estas fases mudam conforme nos movimentamos de um lugar para outro dentro de uma sala, e ainda que o ouvinte receba esse sinal por fones de ouvido, ao alterarmos a fase de componentes do sinal enviado o *pitch* não é alterado. Logo, a idéia do envelope se torna mais atraente. Ainda assim, podemos criar problemas para esse modelo se aumentarmos todas as componentes do sinal por um Δf comum a todas (Fig. 2.9). O envelope do tom modificado ainda tem período de 1/200 s, porque o envelope depende somente do espaçamento dos harmônicos. Além disso, o próprio modelo prediz que o *pitch* deverá ser imutável por uma mudança como a que propusemos. O fato é que experimentos práticos indicam que se fizermos o teste com um acréscimo igual em todos os harmônicos de um Δf comum, o *pitch* do sinal resultante aumenta.

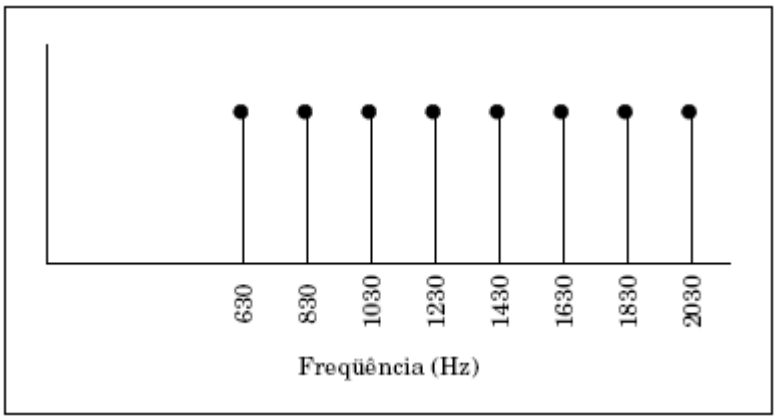

Figura 2.9: $\sum_{n=3}^{10} sen((200.n + 30).2.\pi.t)$.

A falha no modelo de envelope para se estimar o *pitch* de um espectro deslocado em freqüência significa que precisamos procurar outra informação para chegarmos a uma teoria do tempo válida. A escolha mais comum é um modelo baseado na autocorrelação, e este modelo é muito bem sucedido, especialmente em estímulos de banda larga como o de repetição de *pitch* [2]. Ainda assim, este modelo não é eficaz quando trata de tons de baixa freqüência. Um autocorrelator que pode registrar um tom de 100 Hz precisaria ter um atraso neural de ao menos 20 ms de duração, e atrasos desta duração não têm sido encontrados na fisiologia auditiva. Então, este modelo também tem que lutar de alguma forma com a realidade do mecanismo auditivo.

Uma explicação alternativa para o *pitch* de tons complexos enfatiza o papel dos filtros auditivos, ou seja, o *pitch* seria gerado a partir da análise espectral do sinal. Esta análise poderia estar sob a luz da teoria do lugar ou do tempo, mas o papel mais importante a ser desempenhado para resolver os espectros harmônicos encontra naturalmente a representação no domínio do lugar. Esta aproximação está particularmente associada com o tópico de organização auditiva.

2.1.5 *Pitch* e Organização Auditiva

O conceito de organização auditiva surge do fato irrefutável de que o sistema auditivo periférico analisa sinais complexos em diferentes bandas de freqüência. Os processadores nos locais mais centrais são responsáveis por reagrupar estes sinais de um modo significativo. Este processo de reagrupamento é que chamamos de organização auditiva, e o *pitch* é fundamental para esta organização.

Esta conexão entre *pitch* e organização auditiva começa com a observação de que a percepção de *pitch* para tons complexos assemelha-se a um processo de "casar" este tom a um padrão, e isto é feito pelo sistema auditivo central. A idéia é que o processador central tenta ajustar um modelo harmônico

a um padrão de componentes harmônicos já pré-estabelecido ao longo da nossa vida pelo nosso cérebro. Para cada sinal de entrada haveria algum modelo harmônico, definido pela freqüência fundamental, que indicaria um melhor ajuste. É esta freqüência indicativa do melhor ajuste que corresponde ao *pitch* percebido. No caso de haver dois modelos com diferentes freqüências fundamentais indicando igualmente ajustes bons, podemos esperar um *pitch* ambíguo, ou que ambos sejam ouvidos [1].

Este modelo de ajuste que indica o *pitch* de um tom complexo é tido como o responsável para a determinação do *pitch* do carrilhão, sendo este *pitch* a freqüência de um tom senoidal que a maioria dos ouvintes dizem que "casa" com o tom do carrilhão. Gerando o tom do carrilhão digitalmente com sete componentes adicionadas juntas (Fig. 2.10), temos estes componentes representando as vibrações livres das barras do carrilhão, e eles não são harmônicos. O interessante é que o *pitch* não é encontrado pelos modos do carrilhão, e sim é sintetizado pelos terceiro e quarto modos, que aparecem agindo como segundo e terceiro harmônicos de uma fundamental ausente [1]. Os números indicados acima de cada barra nesta figura indicam a fração entre a freqüência do *pitch* encontrado e a freqüência da componente indicada.

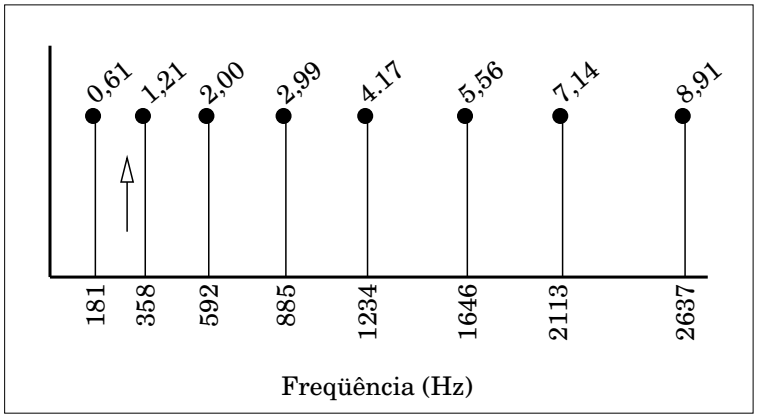

Figura 2.10: Espectro do tom de um carrilhão. O que os ouvintes dizem ouvir é um *pitch* de 296 Hz.

De acordo com o modelo do padrão, os harmônicos de um tom complexo caminham juntos para sintetizar o *pitch*. Só que o modo como esses harmônicos são combinados é a grande questão de interesse. Geralmente o processo de ajuste com um padrão harmônico dá um peso maior para os harmônicos de 3 a 6 [1]. Tais harmônicos seriam dominantes no processo de ajuste de *pitch*, ou seja, eles estariam associados aos harmônicos resolvidos e são estes, em particular, que precisam ser reagrupados no processo de organização auditiva.

O conceito de organização auditiva enfatiza que um estímulo pode ser ouvido de diferentes maneiras, de acordo com as diferentes alocações possíveis de suas componentes. Por exemplo, os harmônicos individuais de um tom complexo não são necessariamente combinados para fazer um *pitch* de tom baixo. Em circunstâncias especiais eles podem ser ouvidos individualmente.

Se construirmos um tom complexo de 200 Hz com dez harmônicos e, enquanto este estiver soando, removermos e reinserirmos por diversas vezes o quinto harmônico, os ouvintes ouvem separadamente este

harmônico do resto do processo, pois o processo chamou a atenção do ouvinte para isto.

O que tudo isto diz a respeito do *pitch* de um tom complexo é que o *pitch* é um grande consolidador. Em uma situação onde há uma grande quantidade de harmônicos, o processador do *pitch* reduz a complexidade por integrá-los a uma única entidade, caracterizada por um *pitch*. Entretanto, no caso de uma componente que afirma sua independência por uma conduta temporal anômala, como no caso anterior, o processador irá apontar para um *pitch* próprio [1]. A organização auditiva envolvida na percepção de *pitch* é parte essencial do nosso senso dos sons do mundo.

Há limites para esta integração do processador de *pitch*. Se uma componente não estiver afinada com relação às demais, então não é realizado o seu ajuste do modelo, e o processador a separa do resto.

No caso de termos um tom complexo de 200 Hz com dez harmônicos e desafinarmos o quinto harmônico em torno de 7%, este harmônico será ouvido separadamente do resto do conjunto.

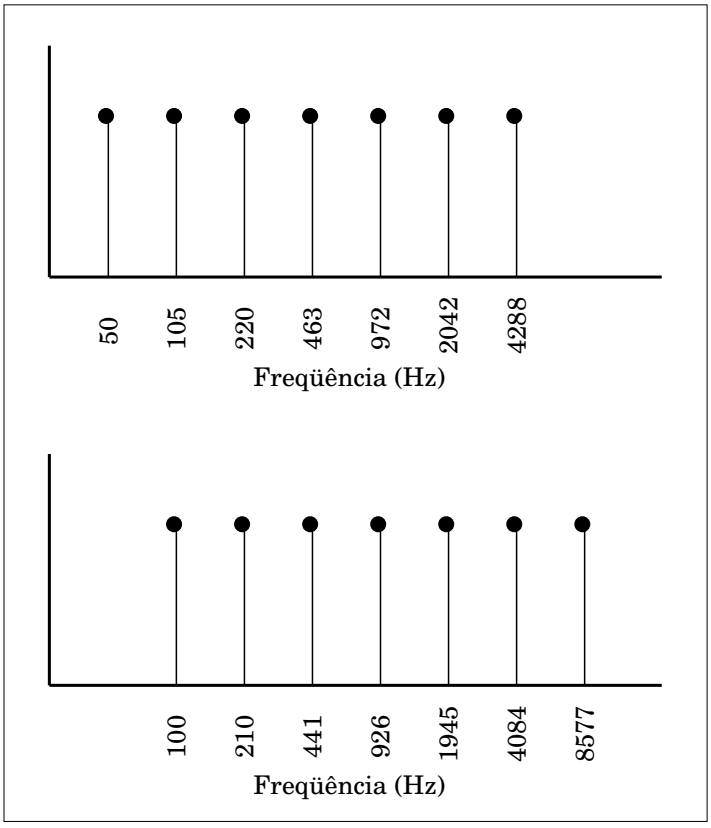

Figura 2.11: Espectro de um Tom de Shepard ($f_n = f_0 2{,}1^n$).

Normalmente, um tom com componentes harmônicos é ouvido como uma única entidade; entretanto, uma componente não-harmônica pode quebrá-lo em componentes individuais. Um tom não-harmônico como no caso anterior é representante do que acontece quando há duas ou mais fontes fora de fase. O problema de separar diversas fontes é normalmente um importante elemento na pesquisa de

inteligência artificial.

Para muitos casos importantes, nos quais o sinal é organizado de acordo com o *pitch*, há alguns efeitos que não são explicados com esse modelo padrão. Um deles é o *pitch* de um tom de *Shepard* estendido. Este tom (sem extensão) é um tom complexo composto apenas por componentes em oitavas. As freqüências das componentes são dadas por $f_n = f_0 2^n$, onde f_0 é uma constante. No tom de *Shepard* estendido as freqüências são dadas por $f_n = f_0 2{,}1^n$. Se imaginarmos que este tom é gravado em uma fita e a tocarmos com o dobro de velocidade, naturalmente todas as freqüências são dobradas. O resultado é mostrado na Fig. 2.11. O efeito curioso é que se dobrarmos a freqüência, o *pitch* decresce.

Este caso pode ser explicado se olharmos com atenção ambos os sinais. Se fizermos a correspondência entre a segunda componente do primeiro sinal e a primeira componente do segundo sinal e assim sucessivamente até a sétima componente do primeiro com a sexta do segundo, vemos claramente que o primeiro tom possui freqüências mais altas. Podemos ver também que os intervalos entre as freqüências de componentes sucessivos se tornam menores quando a velocidade é dobrada (segundo sinal). O modelo tradicional não poderia explicar este efeito, porque o modelo de harmônicos consecutivos não se aplica a esse caso incomum de estrutura. Entretanto, fica reforçada a idéia de que, para a formação do *pitch*, os harmônicos entre o terceiro e sexto são os mais importantes nesta formação [1]; e neste caso, fica claro que se eliminarmos o primeiro "harmônico" do primeiro sinal, as freqüências de suas componentes individuais são mais altas do que no segundo caso.

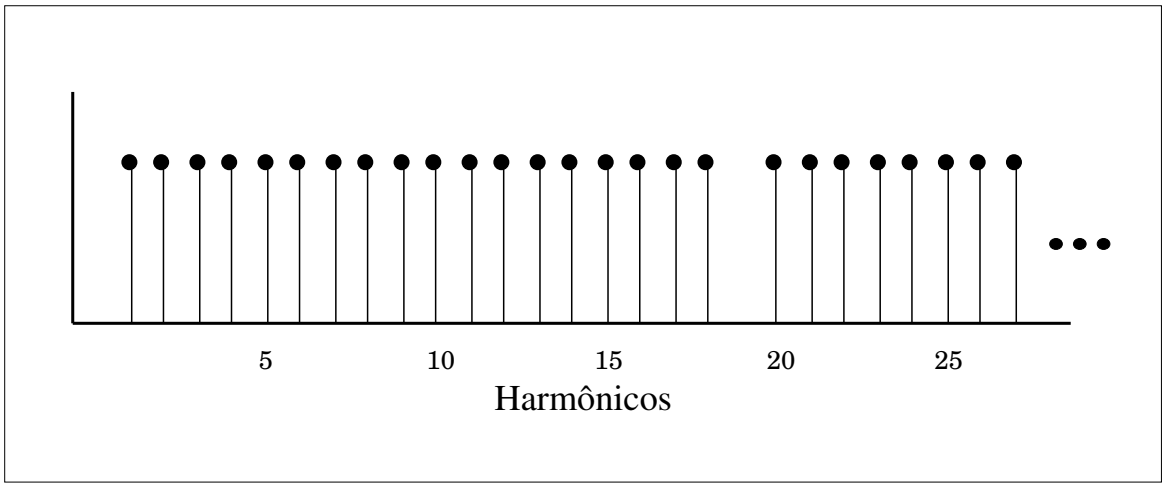

Figura 2.12: Amplitude de um espectro que produz um *Duifhuis pitch*.

Um último caso interessante é o *Duifhuis pitch*, o *pitch* do tom que não está lá. O espectro deste estímulo está mostrada na Fig. 2.12. Consiste de um alto número de harmônicos de uma freqüência baixa, todos em fase. Todos os harmônicos estão presentes exceto um, neste exemplo o décimo-nono. Os

ouvintes relatam que ouvem dois tons, um é o tom de baixa freqüência e o outro é um tom puro com um *pitch* correspondente ao harmônico ausente.

Embora esse resultado seja surpreendente, pois o que é ouvido é justamente o que não está presente no espectro, podemos analisar este efeito da seguinte forma: Se pensarmos no estímulo como um trem de impulsos na freqüência ao qual é adicionada uma senóide que está em oposição exata de fase com a componente a ele correspondente, então esta componente será cancelada. O fato é que se ouve o tom do seno adicionado ao tom composto completo "original" [1].

2.2 Percepção de *Pitch* - Sistema de Scripts

Com bases nesses dados resolvemos escolher alguns destes testes e implementá-los de modo a poder constatar os resultados encontrados e demonstrar tais efeitos com fins didáticos. O material utilizado para a realização desta parte do projeto foi o MatLab© 6.0, e seu pacote auxiliar para a construção de interfaces gráficas GUIDE. O método empregado consistiu em determinar quais testes seriam reproduzidos, e posteriormente construir algumas funções auxiliares que têm como função gerar os vetores de áudio que fazem parte do programa. Após essa etapa, foram discutidos quais parâmetros o usuário poderia modificar e a partir disto foi feita a interface do programa e a sua integração com as funções auxiliares.

2.2.1 *Differencial Limen*

O primeiro programa trata do *Differencial Limen* (DL.m). No primeiro instante o usuário encontrará uma tela (Fig. 2.13) permitindo apenas a entrada dos parâmetros ou a finalização do programa. Os parâmetros (Fig. 2.14) que o usuário pode controlar são: a freqüência do sinal padrão do teste; a duração deste sinal em segundos; o tempo de intervalo de silêncio entre o sinal de teste e o sinal de comparação; e, finalmente, o índice, que será explicado mais tarde.

Após a entrada de parâmetros, o usuário clicará no botão de início (Fig. 2.15) e ouvirá uma senóide pura na freqüência especificada por ele; na seqüência, ouvirá uma outra senóide com uma freqüência ligeiramente maior ou menor que o sinal anterior. O objetivo é que o ouvinte diga se este segundo sinal é mais agudo ou mais grave que o seu antecessor. Testes já realizados por pesquisadores indicam que uma diferença de até 0,2% é identificada em 75% da vezes. Note que 50% corresponde a uma escolha aleatória, não caracterizando uma identificação de fato.

O sinal de áudio é construído em um *script* auxiliar (DLimen.m) que toma os parâmetros ajustados pelo ouvinte para construír a senóide de referência e utiliza um vetor já pré-escolhido que determinará o percentual de diferença entre a freqüência da senóide a ser testada e a de referência. Há cinco senóides a serem testadas abaixo da freqüência do sinal de teste, cinco freqüências acima desta e uma igual. Estes

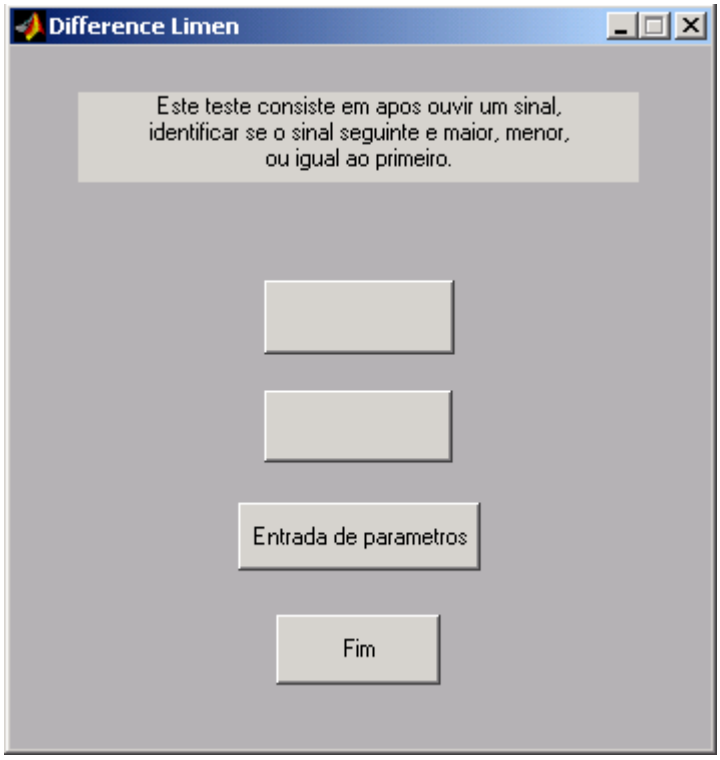

Figura 2.13: Interface gráfica inicial do programa DL.m.

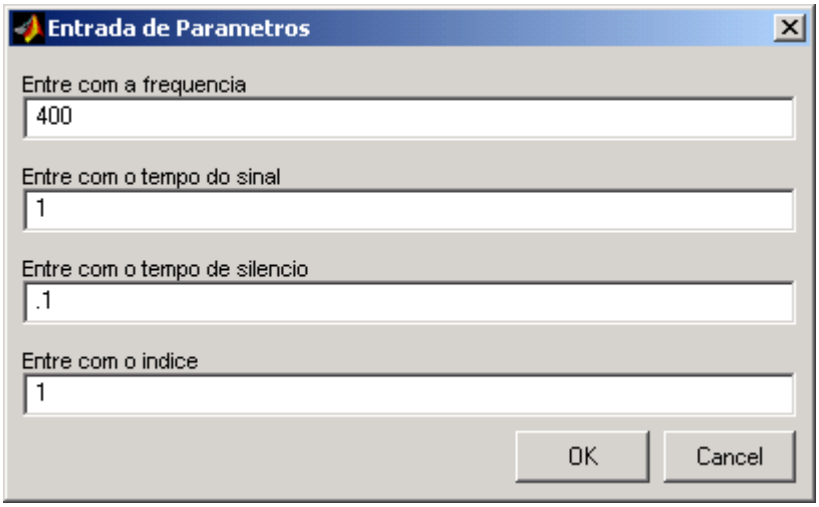

Figura 2.14: Tela de entrada de parâmetros do programa DL.m.

sinais estão dispostos fora de ordem e o parâmetro índice, citado anteriormente, determina justamente com qual destas senóides o teste iniciará.

Esse vetor para o ouvinte é pseudo-aleatório, já que a sua seqüencia é desconhecida; porém, é importante que a pessoa que esteja levantando a estatística de reconhecimento da diferença de freqüência entre os sinais conheça este vetor.

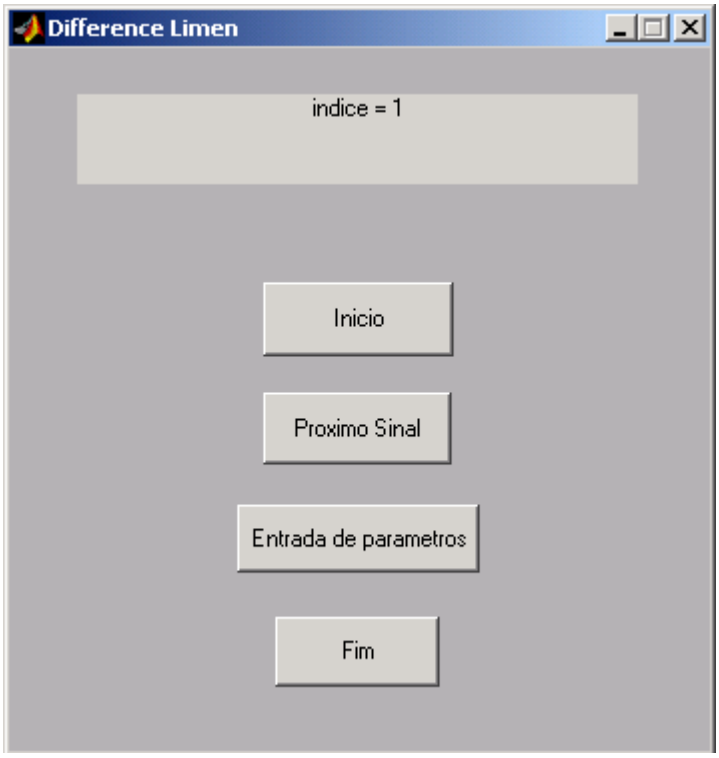

Figura 2.15: Interface gráfica do programa DL.m.

2.2.2 *Match*

O próximo programa trata da identificação do *pitch* de tons complexos (MATCH.m). Este teste é uma reprodução da experiência citada na subseção 2.1.4, e tem como objetivo demonstrar que um sinal do tipo da Fig. 2.8 possui um *pitch* igual, ou próximo, ao do primeiro harmônico, mesmo quando este e o segundo harmônico não estão presentes.

Da mesma forma que no teste anterior, as únicas opções a que o usuário tem acesso são a entrada de parâmetros e a finalização do programa (Fig. 2.16). É possível configurar pela entrada de parâmetros (Fig. 2.17): a freqüência fundamental do sinal; o tempo de duração do sinal; o intervalo de silêncio entre o sinal de referência e a senóide a ser ajustada, como veremos mais tarde; e o número de harmônicos que comporão o sinal de referência, lembrando que o primeiro e segundo harmônicos não participarão do sinal.

Após este ajuste, ao pressionar o botão "Início" (Fig. 2.18), o usuário ouvirá um sinal composto pelo somatório de senóides com o número de harmônicos que foi ajustado previamente; na seqüência, ouvirá um sinal com uma senóide pura com uma freqüência que ele deve ajustar até julgar que o *pitch* está igual ao do primeiro sinal. O usuário tem a opção de parar o teste ao pressionar o botão "Parar" bastando clicar novamente em "Início" para repetir o processo.

Utilizamos dois *scripts* auxiliares (senowav.m e senowav2.m) para a construção dos sinais de áudio. O primeiro, que também será utilizado no próximo programa a ser apresentado, tem como função con-

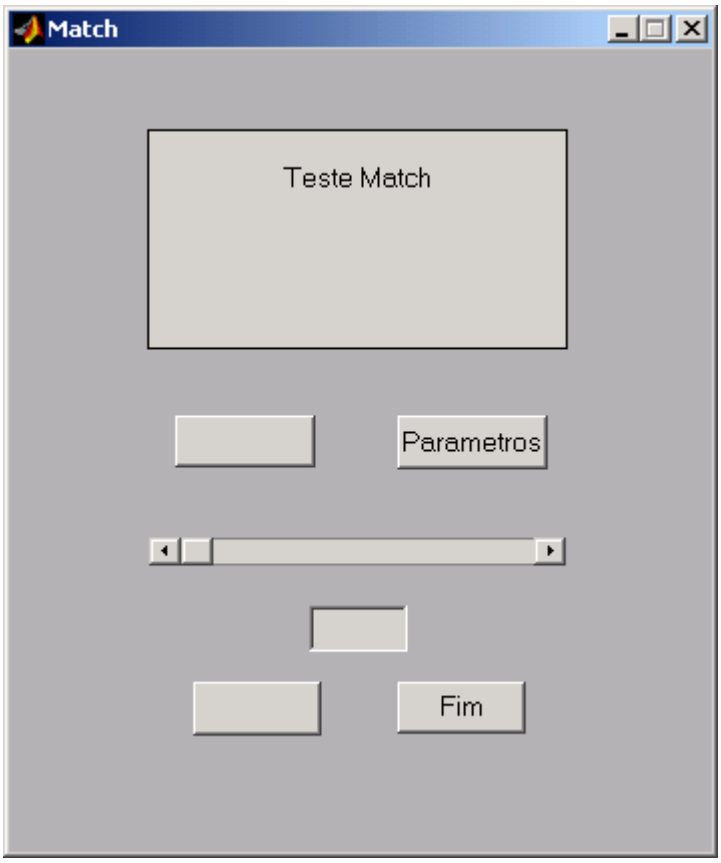

Figura 2.16: Interface gráfica inicial do programa MATCH.m.

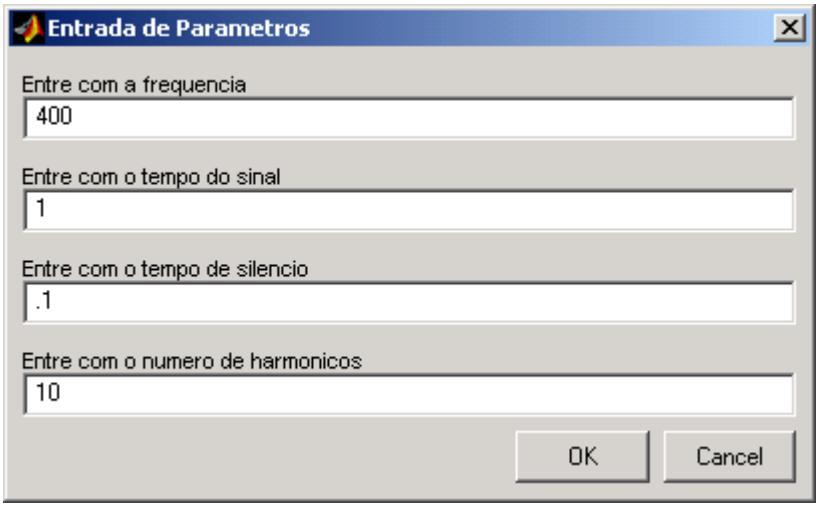

Figura 2.17: Tela de entrada de parâmetros do programa MATCH.m.

struir uma senóide pura com a freqüência a ser ajustada pelo ouvinte ao longo do teste. Já o segundo *script* é utilizado para montar o tom complexo (Fig. 2.8), objeto da identificação do *pitch*.

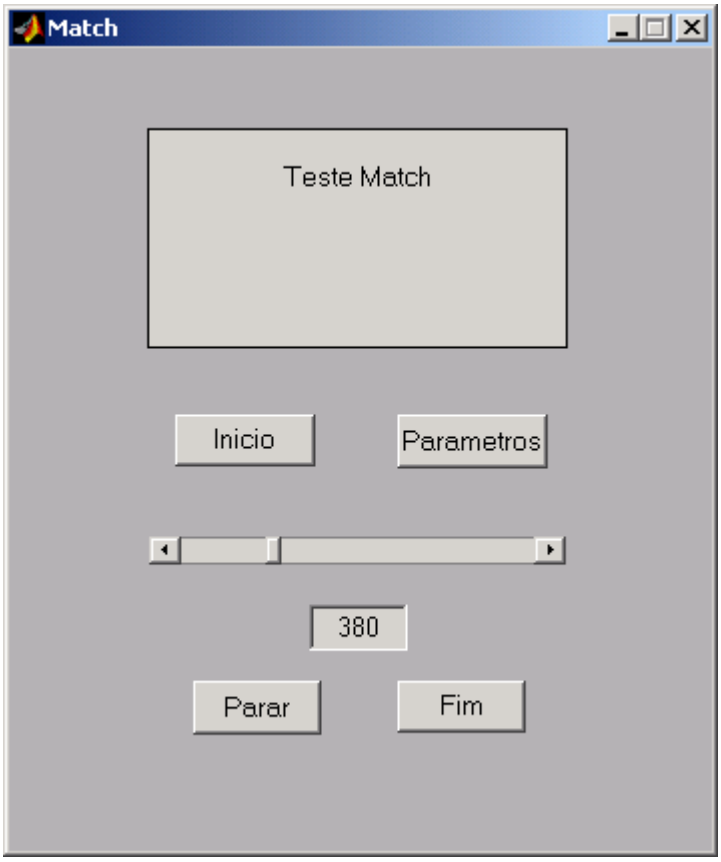

Figura 2.18: Interface gráfica do programa MATCH.m.

2.2.3 *Distuned*

Este programa (Fig. 2.19) trata de um sinal complexo com um dos seus harmônicos desafinados (DISTUNED.m), como o discutido na subseção 2.1.5 na página 14. Assim como no programa anterior, o usuário também deve identificar o *pitch* deste sinal.

O usuário poderá escolher na tela de entrada de parâmetros (Fig. 2.20): a freqüência fundamental do sinal; o tempo de duração deste sinal; o tempo de silêncio entre o sinal de referência e a senóide a ser ajustada; o número de harmônicos que constituirá o sinal de referência; o harmônico que será "desafinado" do sinal de referência; e o percentual deste "desafino".

Uma vez realizada a entrada de parâmetros, pelo usuário, os botões "Inicio" e "Parar" estarão habilitados. Ao clicar no primeiro, o usuário ouvirá o tom complexo com o seu harmônico desafinado e, após o intervalo de silêncio ajustado, ouvirá a senóide que deverá ter a sua freqüência ajustada até o usuário considerar que houve o casamento do *pitch*.

Como já foi explicado, em um caso como esse, o ouvinte ouvirá esse harmônico separadamente do resto do conjunto; entretanto, podemos, com base nas respostas dadas pelo usuário, avaliar o quanto esse desafino prejudica a identificação do *pitch*, comparando os resultados encontrados com os do teste

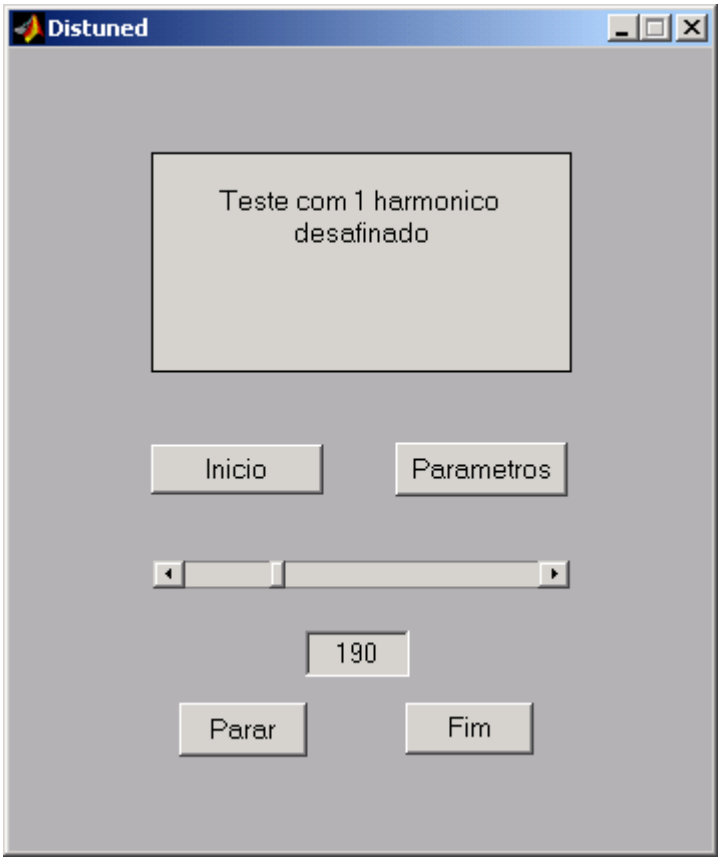

Figura 2.19: Interface gráfica do programa DISTUNED.m.

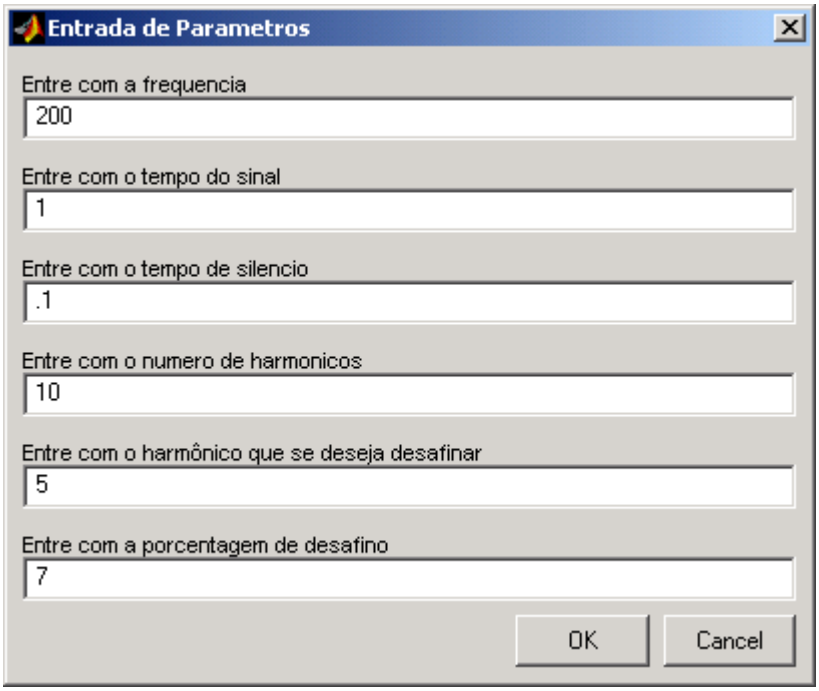

Figura 2.20: Tela de entrada de parâmetros do programa DISTUNED.m.

anterior (subseção 2.1.4).

Além do *script* "senowav.m" utilizamos outro *script* (Distuned1.m) para montar o tom complexo com uma das suas componentes harmônicas "desafinada". Este *script* é uma variação do "senowav2.m", com a diferença de ocorrer a distorção da freqüencia do harmônico escolhido pelo ouvinte, durante a entrada de parâmetros, com o percentual também escolhido.

2.2.4 *Shepard*

Também foi construído um programa (Fig. 2.22) para mostrar os efeitos do tom de *Shepard*, como os explicados na subseção 2.1.5, pág. 15 (SHEPARD.m).

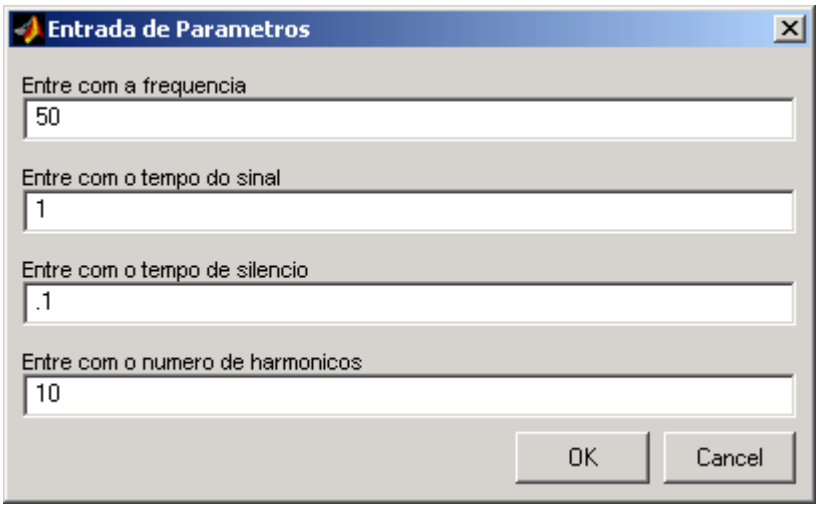

Figura 2.21: Tela de entrada de parâmetros do programa SHEPARD.m.

Antes de utilizar o programa o usuário deverá configurar os seguintes parâmetros: a freqüência fundamental do primeiro sinal; o tempo de duração dos sinais; o intervalo de silêncio entre os sinais; e o número de harmônicos que comporão os sinais.

Uma vez realizada a entrada de parâmetros, o usuário ouvirá um tom de *Shepard* com a freqüência que ele escolheu, e posteriormente ouvirá outro tom de *Shepard* com a sua fundamental duplicada, como o da Fig. 2.11, por exemplo. O ouvinte poderá identificar que o segundo sinal é mais grave que o primeiro, apesar de parecer o contrário a priori, e assim confirmar os testes já realizados por outros pesquisadores.

Foi utilizado o *script* auxiliar "Shepard1.m" para a construção do tom de *Shepard*(Fig 2.11). Este *script* monta um tom de *Shepard* com a fundamental especificada pelo ouvinte na entrada de parâmetros seguindo a equação $f_n = f_0 2{,}1^n$, e posteriormente constrói um outro vetor de áudio com a equação $f_n = 2.f_0 2{,}1^n$, o que equivale a dobrar todas as freqüências dos harmônicos de *Shepard*.

Figura 2.22: Interface gráfica do programa SHEPARD.m.

Capítulo 3

Aplicação de Mascaramento - Codificação MPEG-1 *Layer* I

3.1 Breve Histórico

Após a introdução das tecnologias de vídeo digital e do CD nos anos oitenta, uma enxurrada de aplicações envolvendo áudio/vídeo digital e tecnologias multimídia começaram a surgir. A necessidade de interoperabilidade, imagens de alta qualidade acompanhadas de áudio com qualidade de CD, com baixa taxa de dados, e tudo isto em um único formato de arquivo, exigiu a instituição de um novo grupo de padronização com o *Joint Technical Committee on Information Technology* (JTC 1) supervisionado pelo *International Organization for Standardization* (ISO) e o *International Electrotechnical Commission* (IEC). Este grupo, o *Moving Picture Experts Group* (MPEG), foi formado ao fim dos anos oitenta com o objetivo de desenvolver um padrão para codificação de vídeo associado a áudio e suas combinações [3].

O MPEG-1 é o primeiro padrão internacional que especifica o formado digital para áudio de alta qualidade, onde o objetivo é reduzir a taxa de dados mantendo a qualidade do CD [3]. O sucesso deste padrão permitiu a adoção de áudio em alta qualidade em uma grande faixa de aplicações, desde *digital broadcasting* a aplicações para internet. A introdução do MPEG áudio mudou radicalmente a perspectiva da distribuição de música digital, tocando diversos aspectos dela, incluindo proteção dos direitos autorais, e modelos de negócio, entre outros.

3.2 Redução da Taxa de Bit (*bit rate*)

A principal motivação para reduzir a taxa de bits de codificação é a necessidade de minimizar o custo de transmissão ou tornar o custo de armazenamento mais eficiente, a demanda para transmitir vários canais de capacidade limitada como os canais de rádio móvel e suportar a taxa variável de codi-

ficação em redes orientada a pacotes [4].

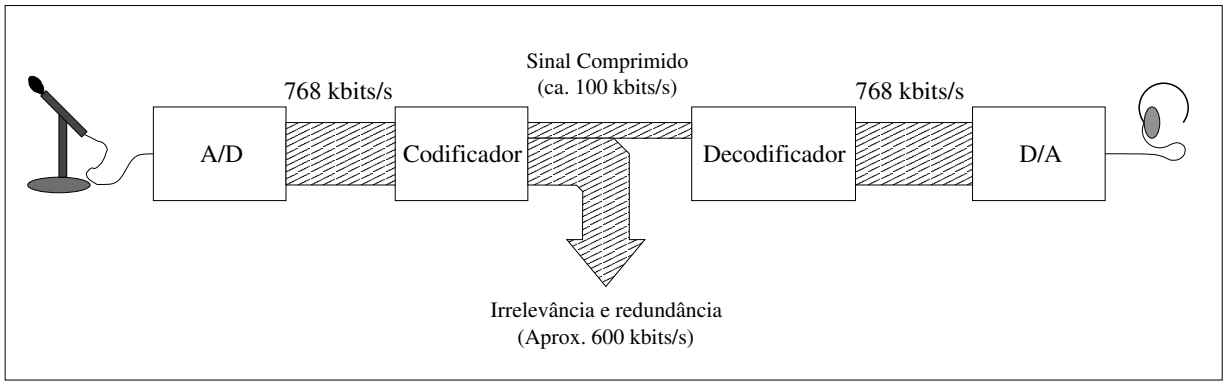

Figura 3.1: Princípio da redução da taxa de bits para sinais de áudio.

A Fig. 3.1 mostra o diagrama em blocos de uma transmissão de informação com redução da taxa de bits. A fonte de informação fornece um sinal de áudio analógico que é transformado em um sinal digital por meio de amostragem e quantização. Com taxa de amostragem em 48 kHz e uma quantização de 16 bits, resulta uma taxa de bit de 768 kbits/s para um canal monofônico. Para transmitir este sinal com a menor energia de entrada possível, a taxa de bits é melhor reduzida no codificador da fonte. O decodificador no fim da recepção converte o sinal com taxa de bits reduzida de volta em um sinal PCM, que é então transformado novamente em um sinal analógico, amplificado e usado para alimentar um alto-falante [5].

A arte de codificar a fonte implica que a redução na taxa de bit seja realizada de modo que o objeto da informação (no caso de codificação de áudio, o ouvido humano) não possa detectar qualquer deterioração na qualidade (do som) ou, ao menos, que os efeitos negativos causados por esta redução sejam minimizados. Na literatura de tecnologia da informação, há dois tratamentos básicos: redução da redundância e redução da irrelevância.

A redundância de um sinal é uma medida da preditibilidade deste sinal. Para reduzir a redundância, é necessário conhecer a estatística da fonte de informação. Porções redundantes do sinal podem ser reduzidas no codificador e restauradas no receptor de maneira que o sinal de áudio possa ser completamente reconstituído. Porém, sinais de áudio genéricos (com exceções específicas, como os sinais de fala) podem conter a quantidade de correlação estatística necessária para obtermos altas taxas de compressão por meio de uma simples redução da redundância [5].

Uma redução de irrelevância, por outro lado, tira vantagem da nossa capacidade limitada de audição para fazer a compressão dos dados [5]. No caso de sinais de áudio, a porção do sinal que o ouvido humano não perceba na amplitude, no tempo e no espectro, será eliminada no codificador. Diferentemente da redução de redundância, a redução da irrelevância é um processo irreversível. Entretanto,

desde que esta redução não seja percebida pela audição humana, o ouvido está livre da degeneração do sinal. Como uma redução de irrelevância tira vantagem da capacidade limitada do receptor, este tipo de compressão de dados de áudio requer um profundo entendimento dos limites da percepção do ouvido humano. Por esta razão, a seguir discutiremos alguns princípios básicos da psicoacústica que são utilizados para a redução da irrelevância na codificação do áudio MPEG.

3.3 Princípios Básicos da psicoacústica

Discutiremos aqui princípios básicos da psicoacústica que possuem um papel fundamental na redução da irrelevância na codificação MPEG-1 *Layer* I.

3.3.1 Limiar de Audibilidade e Área de Sensação Auditiva

O limiar de audibilidade (Fig. 3.2) é o nível de pressão sonora mínimo de um tom senoidal que pode ser percebido. Esta função é dependente da freqüência do tom [5]. Valores de pressão sonora relevantes para aplicações do áudio variam entre 10^{-5} Pa, que é o limite inferior da nossa audição nas freqüências mais sensibilizadoras, e 10^{2} Pa, que corresponde ao limiar da dor [3].

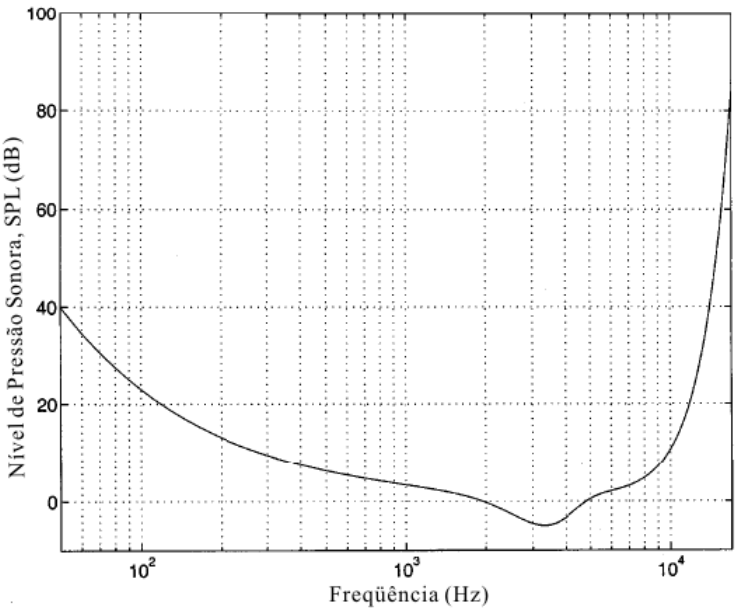

Figura 3.2: Limiar de audibilidade.

Para descrever uma ampla faixa de pressões sonoras relevantes, nós usualmente escolhemos trabalhar em unidade logarítmica e definimos o nível de pressão sonora, SPL (*Sound Pressure Level*), em unidades de dB como:

$$SPL = 10 \log_{10}(p/p_0)^2, \tag{3.1}$$

onde $p_0 = 20$ μPa é igual à pressão sonora no limiar de audição para tons com freqüência em torno de 2 kHz.

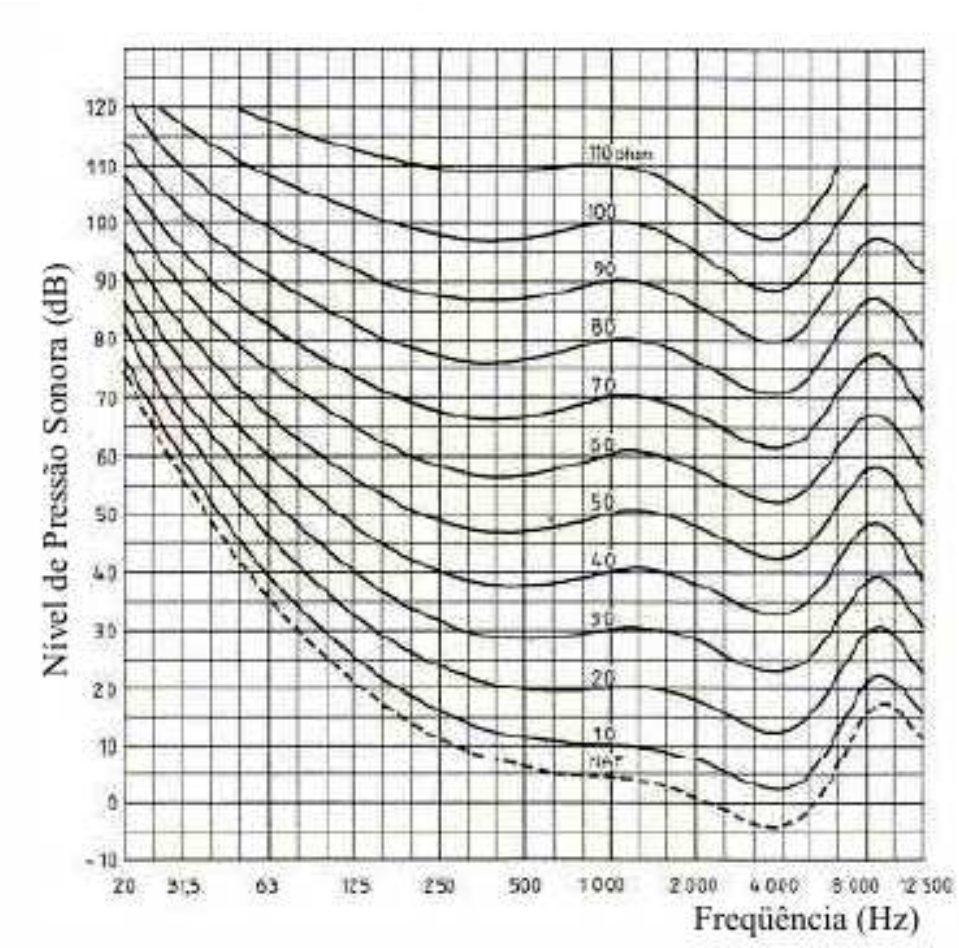

Figura 3.3: Curvas de audibilidade.

A determinação do limiar de audibilidade foi realizada através de testes subjetivos, onde os ouvintes baixavam o nível sonoro de fontes senoidais até estas se tornarem inaudíveis. Este nível de pressão mínimo foi determinado para várias freqüências e tirou-se uma média de um número grande de sujeitos testados.

Este nível forma o limiar de audibilidade, Fig. 3.2. O limiar da dor foi detectado em níveis de pressão sonora de 130 dB. A faixa entre limiar de audibilidade e limiar da dor é chamado de área de sensação auditiva (Fig. 3.4). Curvas de mesma percepção sonora, *loudness*, são vistas na Fig. 3.3. Estas curvas foram igualmente extraídas de testes subjetivos, os ouvintes ouviam um tom padrão de 1 kHz e posteriormente ouviam um segundo tom, em uma outra freqüência qualquer, e ajustavam o nível de pressão sonora deste até afirmar que ambos possuíam o mesmo volume. Este processo foi realizado com

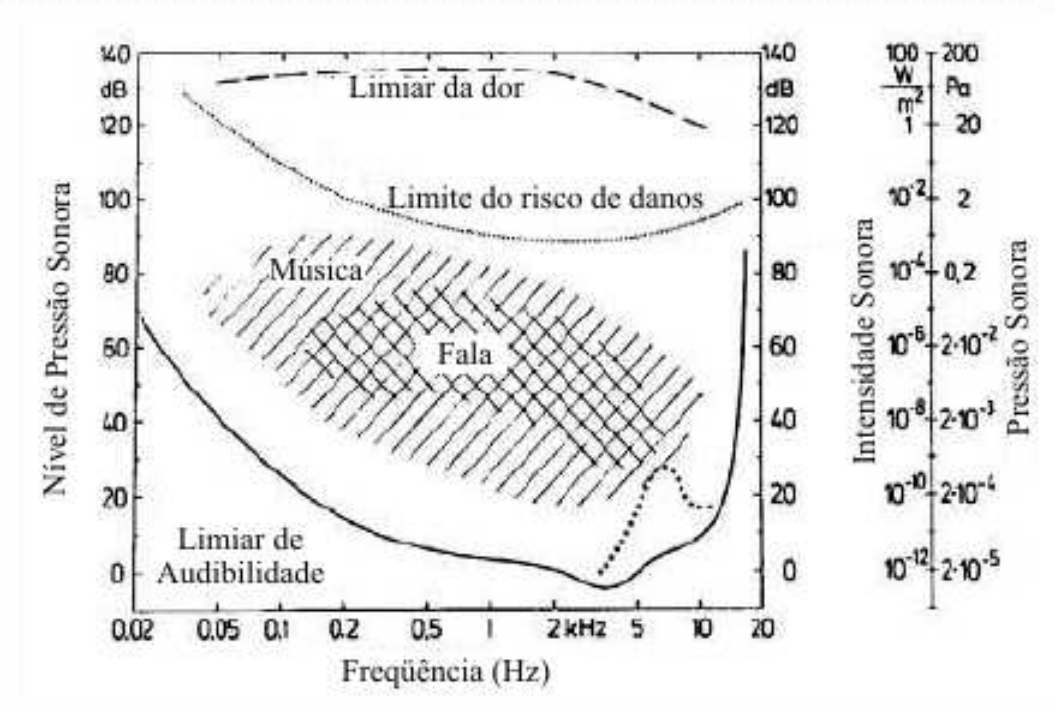

Figura 3.4: Área de sensação auditiva.

diversas freqüências diferentes, em diversos níveis de pressão diferentes, e as curvas mostradas representam uma média dos resultados encontrados com os inúmeros sujeitos testados. Estas curvas mostram que a percepção sonora e o limiar de audibilidade são fortemente dependentes da freqüência, que é uma indicação de que para a codificação do áudio é recomendável um tratamento diferenciado entre espectros diferentes [5].

3.3.2 Banda Crítica

A estrutura da banda crítica e a escala relacionada a sua taxa podem ser usadas para descrever muitos aspectos do funcionamento do nosso sistema auditivo. A banda crítica tem uma relação perceptual e física com o sistema auditivo. Uma definição básica de banda crítica é a "banda fora da qual respostas subjetivas mudam mais abruptamente". Outro modo de vermos a banda crítica é que ela representa uma primeira aproximação da habilidade auditiva em separar sons de diferentes freqüências. A taxa da banda crítica (CBR) é uma medida de localização na membrana basilar, assim como a freqüência dá uma medida de localização espectral. A unidade da taxa da banda crítica é o Bark.

A banda crítica, que representa 1 Bark na escala CBR, representa o comprimento de 1,3 mm longitudinalmente à membrana basilar (Fig. 2.1), e 150 células ciliadas em cada coluna de células ciliadas que atravessam a membrana basilar. Experimentos demonstram que podem-se definir 25 bandas críticas contíguas dentro da faixa de freqüências da audição humana. Observando a Tabela 3.1, é nítido que as bandas críticas têm largura de banda constante para freqüências centrais de 100 Hz a 500 Hz, e a partir

Banda (Bark)	Limite Inferior (Hz)	Centro (Hz)	Limite Superior (Hz)	Banda (Bark)	Limite Inferior (Hz)	Centro (Hz)	Limite Superior (Hz)
1	0	50	100	14	2000	2150	2320
2	100	150	200	15	2320	2500	2700
3	200	250	300	16	2700	2900	3150
4	300	350	400	17	3150	3400	3700
5	400	450	510	18	3700	4000	4400
6	510	570	630	19	4400	4800	5300
7	630	700	770	20	5300	5800	6400
8	770	840	920	21	6400	7000	7700
9	920	1000	1080	22	7700	8500	9500
10	1080	1170	1270	23	9500	10500	12000
11	1270	1370	1480	24	12000	13500	15500
12	1480	1600	1720	25	15500	19500	
13	1720	1850	2000				

Tabela 3.1: Freqüências centrais e limítrofes das bandas críticas.

daí a largura da banda aumenta conforme a freqüência central aumenta. O CBR de uma determinada freqüência pode ser obtido por:

$$z = 13\text{arctg}(0{,}76f) + 3{,}5\text{arctg}[(f/7{,}5)^2], \tag{3.2}$$

onde z é o CBR em Bark e f é a freqüência em kHz. A taxa da banda crítica é uma função de freqüência contínua, e há uma largura de banda crítica para cada freqüência, que é obtida por:

$$\Delta f_G = 25 + 75(1 + 1{,}4f^2)^{0{,}69}, \tag{3.3}$$

onde Δf_G é a largura da banda crítica na freqüência f, ainda expressa em kHz. A relação entre freqüência, taxa da banda crítica e o local correspondente na membrana basilar é mostrada na Fig 3.5.

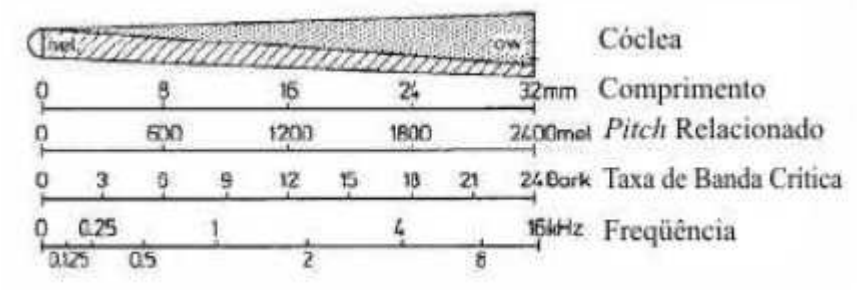

Figura 3.5: Relação entre taxa de *pitch*, taxa de banda crítica, freqüência e localização na membrana basilar.

A escala Bark é mais útil do que a escala de freqüências para analizar o sistema auditivo, pois o CBR indica o local de excitação da membrana basilar, fazendo, assim, uma escala linear apropriada para a representação de eventos ocorridos nestes locais. Como o local de excitação da membrana basilar tem uma relação quase linear com a escala de freqüência em freqüências baixas, porém uma relação logaritmíca para freqüências mais altas, a escala de freqüências demonstra ser inadequada para a representação do sistema auditivo. Este fato será considerado no projeto do modelo psicoacústico do MPEG 1 *Layer* I.

3.3.3 Mascaramento

O mascaramento é um efeito conhecido do cotidiano; por exemplo, um evento sonoro, como uma música, que pode ser facilmente ouvida em um ambiente silencioso pode tornar-se imperceptível na presença de ruído, como o causado por um motor. Para que a música permaneça perceptível na presença do ruído, é necessário que o volume daquele seja consideravelmente maior do que no ambiente silencioso. O conceito de "limiar de mascaramento" é usado para descrever quantitativamente os efeitos do mascaramento. Um limiar de mascaramento é definido como o nível de pressão sonora ao qual um som de teste, como regra um tom senoidal, deve ser reduzido para se tornar inaudível na presença de um sinal mascarador. A natureza do efeito do mascaramento pode ser dividida em dois itens que discutiremos a seguir.

a) Mascaramento por Sons Estacionários

Antes de mais nada iremos discutir o mascaramento de sons cuja estrutura do tempo pode ser desconsiderada. A estrutura temporal de um som pode ser desconsiderada se a duração do som não for menor do que 200 ms. Assim sendo, para definirmos o limiar de mascaramento causado por um ruído de banda estreita, os ouvintes são expostos a um tom senoidal adicionado a este ruído. O nível de pressão sonora é progressivamente reduzido até tornar-se inaudível. Este procedimento é repetido, sendo algumas centenas de ouvintes testados para uma faixa de freqüência entre 20 Hz e 20 kHz [5]. Os níveis de pressão sonora, retirados da média do número de ouvintes, constituem o limiar de mascaramento.

A Fig. 3.6 mostra o limiar de mascaramento por um ruído de banda estreita mascarador de 60 dB, nas freqüências centrais de 250 Hz, 1 kHz e 4 kHz com largura de banda de 100 Hz, 160 Hz e 700 Hz respectivamente. Todos os tons que se encontram abaixo deste limite são mascarados por estes ruídos e são imperceptíveis para o ouvido humano. Como podemos ver, a forma destes limiares de mascaramento é altamente dependente da freqüência central do ruído de banda estreita, e a distância entre o máximo do mascaramento e a faixa de 60 dB também aumenta conforme aumenta a freqüência. Porém, a freqüência do mascarador não é o único parâmetro que influência a forma do limiar de mascaramento. Outro parâmetro é o nível do mascarador. O limiar de mascaramento causado por um ruído de banda estreita de diferentes níveis de pressão sonora L_g, com freqüência central de 1 kHz e uma largura de banda de 160 Hz, é mostrado na Fig. 3.7. Além da freqüência

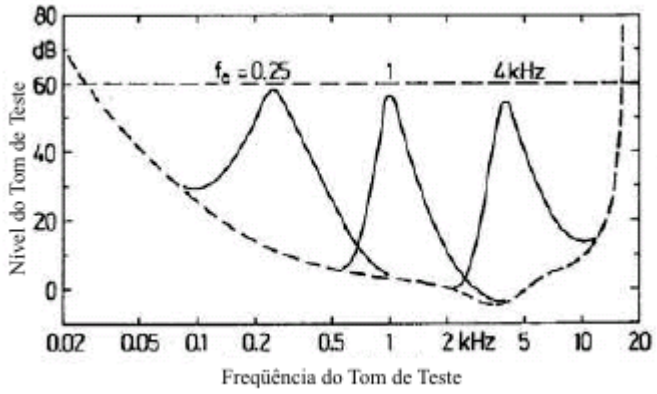

Figura 3.6: Limiar de mascaramento causado por ruído de banda estreita de freqüência central variável com um nível de pressão sonora igual a 60 dB.

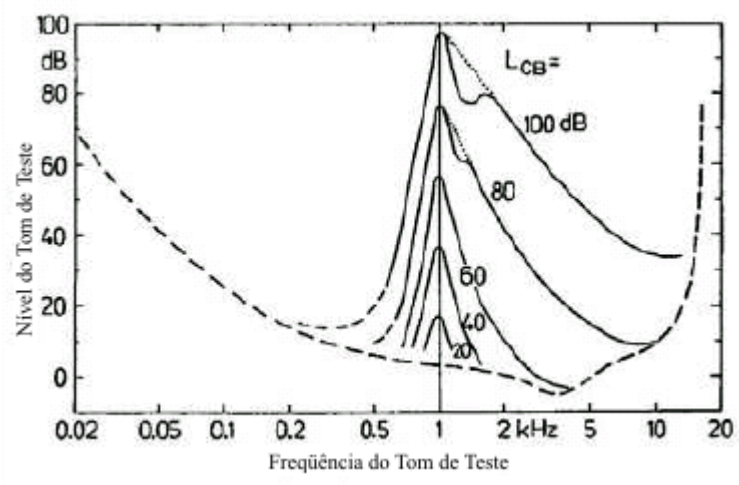

Figura 3.7: Limiar de mascaramento causado por ruído de banda estreita com diferentes níveis de pressão sonora.

central, e do nível de pressão sonora do mascarador, o terceiro parâmetro relevante é o tipo do mascarador. Se o mascarador for um tom senoidal, por exemplo, o limiar de mascaramento resultante será diferente do mostrado na Fig. 3.7.

b) Efeitos de Mascaramento Dependentes do Tempo

Os efeitos de mascaramento dependentes do tempo são resultantes de eventos sonoros de curtíssima duração. Para a determinação deste efeito, ouvintes foram expostos a um pulso de ruído branco gaussiano agindo como mascarador. Este ruído possui amplitude de L_{WN} e uma duração de 0,5s. Após este pulso, seguindo de um intervalo de tempo t_v, segue-se um segundo pulso gaussiano de 20 μs de duração (Fig. 3.8). O ouvinte tinha a tarefa de ajustar a amplitude deste segundo pulso, até este se tornar imperceptível.

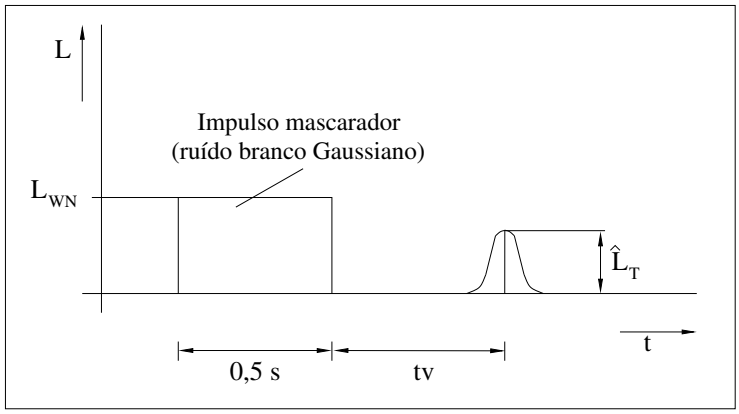

Figura 3.8: Medida do limiar do pós-mascaramento.

Os resultados deste teste subjetivo são mostrados na Fig. 3.9. O limiar de mascaramento permanece com, aproximadamente, o mesmo nível após um pouco mais de 10 ms do ruído mascarador ter sido desligado. Ou seja, durante esse tempo de até 10 ms após o fim do ruído mascarador o efeito é quase o mesmo de se apresentar o ruído gaussiano e o impulso gaussiano simultaneamente. Somente após estes 10 ms o limiar de mascaramento cai, chegando ao limiar de audibilidade após aproximadamente 150 ms.

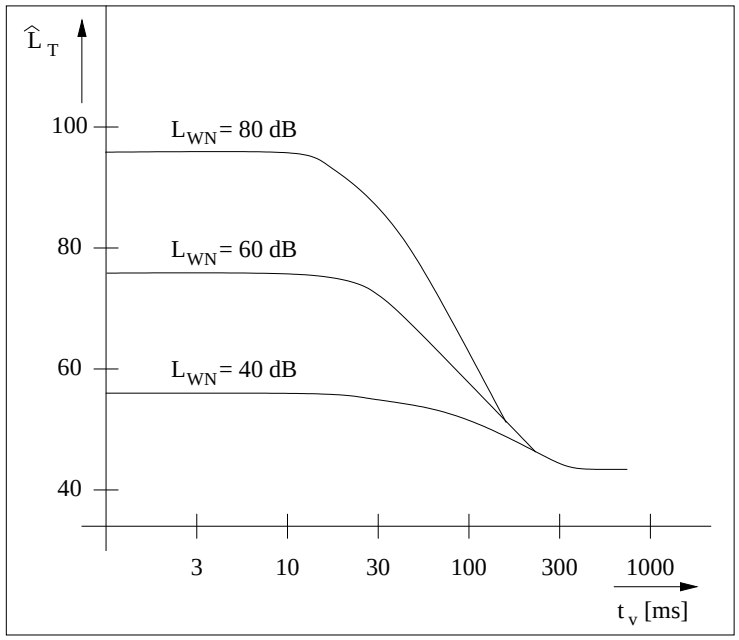

Figura 3.9: Limiar de mascaramento com amplitudes diferentes de mascarador e distâncias diferentes ao mascarador.

Entretanto, o mascaramento não ocorre somente quando o pulso gaussiano é ligado durante ou após a ocorrência do mascarador, mas também quando este pulso ocorre antes do ruído gaussiano. Este efeito, descrito na literatura como pré-mascaramento, só é percebido quando o intervalo de tempo entre o pulso gaussiano e o acionamento do ruído mascarador não é maior do que 20 ms.

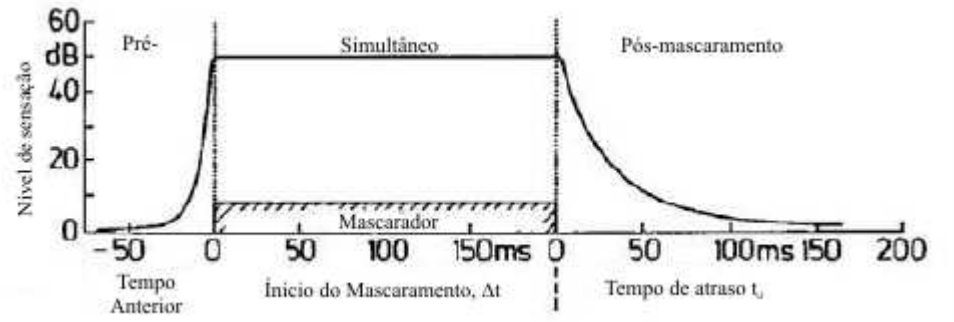

Figura 3.10: Resumo do efeito do mascaramento.

Os efeitos de mascaramento são resumidos na Fig. 3.10. O mascaramento de sons que são apresentados ao ouvido humano ao mesmo tempo que um mascarador é conhecido como 'mascaramento simultâneo'. Entretanto, o efeito de mascaramento que permanece após o mascarador ter sido desligado, e desaparece aproximadamente após 200 ms, é conhecido como 'pós-mascaramento'. Já o efeito do som que é mascarado antes do ruído de mascaramento, como já citamos, é chamado de 'pré-mascaramento'. Porém este efeito pemanece somente por uns 5 ms [5].

3.3.4 Adição de Mascaramentos

Na subseção anterior descrevemos o efeito de mascaramento de um único mascarador. Consideraremos agora o efeito de dois ou mais mascaradores. Se os limiares de mascaramento de dois mascaradores A e B forem denotados como M_A e M_B, e o limiar de mascaramento resultante da combinação de A e B de M_{AB}, então um modelo de adição de mascaramento tem a seguinte forma:

$$J(M_{AB}) = J(M_A) + J(M_B). \tag{3.4}$$

Assim, o efeito combinado dos mascaradores A e B é a transformada inversa da soma linear da função transformada de mascaramento, onde J indica esta transformação.

O modelo linear da soma do mascaramento simplesmente indica que a soma de dois mascaradores, de igual potência sonora, produz 3 dB de mascaramento adicional, correspondente a dobrar o efeito de mascaramento. O modelo de mascaramento linear é expresso como:

$$J(M_X) = \sum_{1}^{n=N} (10^{(M_n/10)}), \tag{3.5}$$

onde M_X é o nível de mascaramento de sinais M_n combinados, denotado por X. Entretanto este modelo é inconsistente com observações experimentais, apesar de ser usado em alguns modelos psicoacústicos.

Os resultados de experimentos de mascaramento indicam que o limiar de mascaramento se combina de uma maneira não linear. O modelo de lei de potência modificado (MPL), leva em conta esta não-linearidade e ainda inclui o efeito do limiar de audibilidade, que é assumido como o resultado de ruídos internos. O MPL é expresso como:

$$J(M_{TX}) = \sum_{1}^{Tn=N} \left((10^{(M_{Tn}/10)})^p \right) - (10^{(QT/10)})^p, \tag{3.6}$$

onde M_{TX} é o limiar de mascaramento do efeito combinado dos mascaradores M_{Tn}, QT é o limiar de audibilidade, e p pode assumir valores entre $0,1$ e $0,3$, sendo uma variável determinada de modo empírico e que está relacionada com a mínima sobreposição espectral entre os mascaradores. Neste modelo, assume-se que os efeitos de ruídos internos e do sinal de mascaramento são comprimidos antes de se combinarem. O modelo MPL, sem os ajustes descritos, não sobrepõe os mascaradores temporais. Quando o mascaramento se sobrepõe no tempo, a supressão entre os mascaradores pode contribuir para o limiar de mascaramento global. Para computar a supressão, ou p é modificado ou a soma das supressões é calculada e então usada para atenuar o limiar de mascaramento apropriadamente, antes de eles entrarem no modelo MPL.

3.4 Estrutura Básica do Codificador

A Fig. 3.11 mostra o diagrama em blocos de um codificador de áudio de acordo com MPEG-1 *Layer* I. A cada *loop* de codificação entram neste sistema 384 novas amostras de áudio PCM e é gerado como saída um *frame* que contém todas as informações necessárias para a decodificação do arquivo codificado. Como podemos observar, há dois processos que correm em paralelo e que estão interligados.

Em um dos caminhos encontramos o banco de filtros que divide o sinal em 32 sub-bandas, com a mesma largura de banda cada. Este banco é um banco de filtros polifásicos e seus coeficientes são encontrados na recomendação [6]. Em cada um desses 32 canais o sinal é criticamente sub-amostrado, o que siginifica que a taxa de amostragem é reduzida à trigésima segunda parte da taxa de amostragem original; assim, cada um dos 32 canais possui 12 amostras (32 sub-bandas $*$ 12 amostras por sub-banda = 384 amostras). Podemos encontrar os detalhes desta implementação na seção 3.6.

Posteriormente, é determinado o valor máximo - fator de escala - de cada sub-banda. Este fator de escala é quantizado e transmitido com uma resolução de amplitude correspondente a 6 bits. As 12 amostras de cada sub-banda, resultantes da saída do banco de filtros, são então divididas pelo fator de escala, e o resultado será quantizado de acordo com o resultado da alocação dinâmica de bits. Para computar o número de bits utilizados para a quantização das amostras de cada sub-banda, é preciso aplicar o limiar de mascaramento obtido no modelo psicoacústico. Todas as 12 amostras de cada sub-banda são então sujeitas à mesma quantização.

Em paralelo, à análise em sub-bandas, o sinal é submetido a uma transformada de Fourier com 512

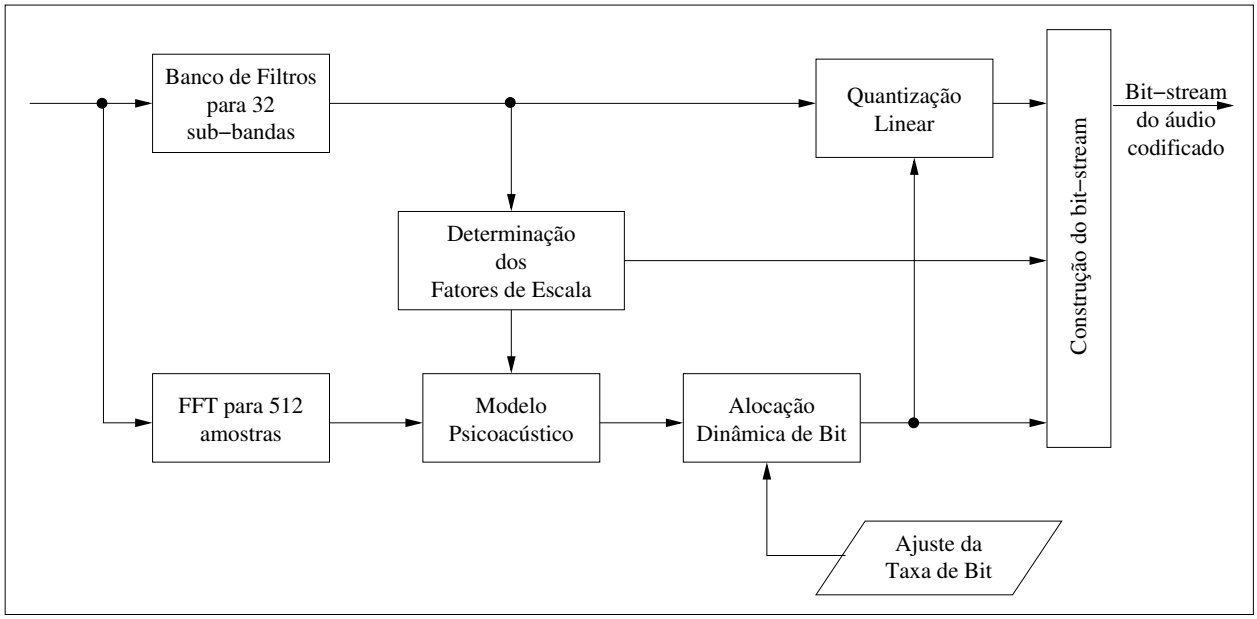

Figura 3.11: Diagrama em bloco do codificador MPEG-1 *Layer* I.

amostras. Os máximos locais deste espectro são, então, determinados, e a vizinhança destes máximos é calculada de modo a determinar quais deles são componentes tonais ou não-tonais do sinal. Este cálculo é necessário pois a tonalidade do sinal tem um impacto relevante no formato do limiar de mascaramento. Desta maneira é possível determinar, do limiar de mascaramento, o nível sonoro máximo de cada sub-banda. Este processo é mais comumente conhecido como Modelo Psicoacústico.

Uma vez obtido o limiar de mascaramento, e dada a taxa de bits desejada, o número de passos de quantização pode ser determinado de modo a maximizar a margem entre o limiar de mascaramento e o ruído de quantização. As amostras quantizadas são combinadas com os fatores de escala e a alocação de bits para formar um *bitstream*; a alocação de bits indica o número de bits usados para quantizar as amostras de cada sub-banda.

Cabeçalho	Código Corretor de Erro	Alocação de Bits	Fatores de Escala	Amostras	Dados Auxiliares
12 bits de sincronismo e 20 bits de informação	16 bits (opcional)	4 bits cada	6 bits cada	de 2 a 15 bits cada	

Figura 3.12: *Bitstream* do áudio de acordo com MPEG-1 *Layer* I.

Estes dados são precedidos por um cabeçalho que contém informações para o controle do decodificador. O *frame* de dados é definido como parte de um *stream* que contém todas as informações relevantes necessárias para a decodificação. No início do *frame* estão os 32 bits do cabeçalho, que inicia com uma

palavra de sincronismo e contém na seqüência informações que descrevem o sinal de áudio em detalhes. Posteriormente é colocado um código opcional, CRC, de 16 bits como proteção de erro, e este código serve para proteger a maioria dos dados importantes, a alocação de bits e parte do cabeçalho contra erros de transmissão. Depois há os bits que representam a alocação de bits para cada sub-banda, e os fatores de escala, que são quantizados com 6 bits cada. Finalmente, temos as amostras, que podem ser quantizadas com um número entre 2 e 15 bits, de acordo com o ruído de quantização permitido.

3.5 Modelo Psicoacústico I

O codificador MPEG 1 *Layer* I utiliza os príncipios do mascaramento, discutidos na seção 3.3, para reduzir a taxa de bits. Esta codificação é realizada com redução da irrelevância. A idéia básica é: o que não pode ser ouvido não precisa ser transmitido. Como vimos na referida seção, um sinal pode se tornar inaudível na presença de outro sinal de amplitude maior, desde que o primeiro sinal esteja abaixo do limiar de mascaramento (Fig. 3.7).

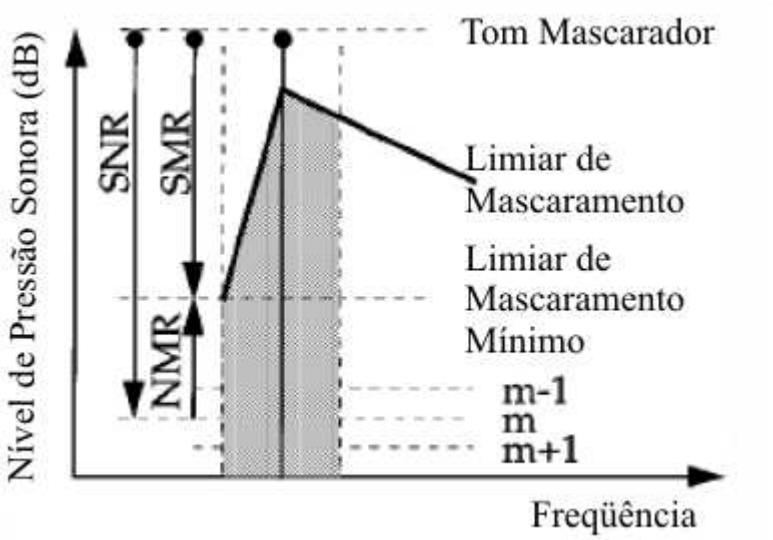

Figura 3.13: Representação esquemática de um mascaramento simultâneo.

Vamos definir a distância entre o nível do mascarador e o limiar de mascaramento mínimo, dentro de uma banda de freqüência, como razão sinal-máscara (*signal-to-mask ratio - SMR* (Fig. 3.13)). Assumindo a quantização do sinal PCM original como m bits, dentro desta faixa de freqüência (sub-banda), o ruído de quantização não será audível enquanto a razão sinal-ruído (*signal-to-noise ratio - SNR*) for maior que a SMR.

Definindo SNR(m) como a SNR resultante de uma quantização de m-bits, a distorção percebida em uma dada sub-banda é medida pela razão ruído-máscara (*noise-to-mask ratio - NMR*).

$$NMR(m) = SMR - SNR(m), \text{ em dB} \tag{3.7}$$

O NMR(m) é a diferença entre o nível de quantização do ruído e o nível em que uma distorção se torna audível, em uma dada sub-banda. Em uma sub-banda, o ruído codificado será inaudível enquanto o NMR(m) for negativo.

Assim, cabe ao modelo psicoacústico determinar o SMR de cada sub-banda, e será a diferença entre SMR e NMR que determinará quantos bits serão utilizados para quantizar o sinal dentro de cada sub-banda.

Descreveremos a seguir as etapas que formam o modelo psicoacústico.

3.5.1 Cálculo da FFT

O limiar de mascaramento é derivado de uma estimativa do espectro de densidade de potência que é calculado por uma FFT de 512 pontos, para o *Layer* I. A FFT é calculada diretamente do sinal de entrada PCM, 384 novas amostras mais as 128 últimas amostras a cada *loop* de codificação, janelado por uma janela de Hann.

Para sincronizar a alocação de bits e a amostra da sub-banda correspondente, as amostras PCM que entram na FFT têm que ser atrasadas:

a) O atraso do banco de filtros é de 256 amostras, corespondente a 5,8 ms na taxa de amostragem de 44,100 kHz. Uma janela deslocada de 256 amostras é necessária para compensar o atraso do banco de filtros.

b) A janela de Hann tem que coincidir com a amostra da sub-banda do *frame*. Para o *Layer* I isto implica um deslocamento adicional de 64 amostras.

Dados técnicos da FFT:
- Janela de Hann, $h(i)$:

$$h(i) = \sqrt{8/3}.\ 0{,}5.(1 - \cos[2.\pi.(i)/N]) \tag{3.8}$$

-Espectro da densidade de potência $X(k)$:

$$X(k) = 10.\log_{10}\left|\frac{1}{N}\sum_{l=0}^{N-1} h(l).s(l).e^{(-j.k.l.2.\pi/N)}\right|^2 \text{ dB } , \; k = 0...N/2, \tag{3.9}$$

onde $s(l)$ é o sinal de entrada.

Uma normalização ao nível de referência de 96 dB SPL tem que ser feita no espectro $X(k)$ de modo que o valor máximo corresponda a 96 dB.

3.5.2 Determinação do Nível de Pressão Sonora

O nível de pressão sonora L_{sb} na sub-banda n é computado por:

$$L_{sb}(n) = \text{MÁX}\left[X(k),\ 20.\log(scf_{max}(n).32768) - 10\right] \text{ dB}, \tag{3.10}$$

onde $X(k)$ é o nível de pressão sonora da linha espectral com índice k da FFT com a maior amplitude no domínio da freqüência dentro da sub-banda n. A expressão $scf_{max}(n)$ é o fator de escala no *Layer* I. O termo "-10 dB" corrige a diferença entre níveis de pico e RMS. O nível de pressão sonora $L_{sb}(n)$ é computado para cada sub-banda n.

3.5.3 Determinação do Limiar Auditivo no Silêncio

O limiar no silêncio $LT_q(k)$, também chamado de limiar absoluto ou limiar de audição, e está disponível nas tabelas "*Frequencies, critical band rates and absolute threshold*", tabelas D.1a, D.1b, D.1c para *Layer* I [6]. Estas tabelas dependem da taxa de amostragem do sinal de entrada PCM. Os valores são disponíveis para cada amostra no domínio da freqüência, onde os limiares de mascaramento são calculados. Um *offset* é usado para o limiar absoluto, dependendo da taxa de bits utilizada. Este *offset* é de -12 dB para taxas de bits $\geq$ 96 kbits/s e 0 dB para taxas de bits $<$ 96 kbits/s por canal.

3.5.4 Busca das Componentes Tonais e Não-Tonais

A tonalidade de um componente mascarador tem uma influência no limiar de mascaramento. Por esta razão, é conveniente a distinção entre componentes tonais e não-tonais. Para o cálculo do limiar global de mascaramento é necessário derivar os componetes tonais e não-tonais do espectro da FFT. Entende-se como componente mascarador tonal o mascarador oriundo de um sinal tonal com potência muito alta; já o mascarador não-tonal é, por exemplo, um ruído de banda estreita. Deve-se salientar que as componentes não-tonais têm um efeito mascarador mais intenso que os tonais, daí a importância da sua distinção.

Esta etapa se inicia com a determinação dos máximos locais; então, extraímos as componentes tonais (senoidais) e calculamos a intensidade das componentes não-tonais com uma largura de banda de uma banda crítica. Os limites das bandas críticas são dados nas tabelas "*Critcal band boundaries*", tabelas D.2a, D.2b, D.2c para *Layer* I [6].

A largura das bandas críticas varia com a freqüência central, indo de uma largura de banda em torno de 0,1 kHz em baixas freqüências, até uma largura de banda em torno de 4 kHz em altas freqüências. É conhecido dos experimentos psicoacústicos que o ouvido tem uma melhor resolução para as freqüências baixas do que para as freqüências altas. Para determinar se um máximo local pode ser um componente tonal, uma faixa de freqüência *df* em torno do máximo local é examinada. A faixa de freqüência *df* é dada por:

Taxa de amostragem de 32 kHz:

$$\begin{aligned} df &= 125\ Hz & 0\ kHz < f \leq 4,0\ kHz \\ df &= 187,5\ Hz & 4,0\ kHz < f \leq 8,0\ kHz \\ df &= 375\ Hz & 8,0\ kHz < f \leq 15,0\ kHz. \end{aligned}$$

Taxa de amostragem de 44,1 kHz:

$$\begin{aligned} df &= 172,266\ Hz & 0\ kHz < f \leq 5,512\ kHz \\ df &= 281,25\ Hz & 5,512\ kHz < f \leq 11,024\ kHz \\ df &= 562,50\ Hz & 11,024\ kHz < f \leq 19,982\ kHz. \end{aligned}$$

Taxa de amostragem de 48 kHz:

$$\begin{aligned} df &= 187,50\ Hz & 0\ kHz < f \leq 6,0\ kHz \\ df &= 281,25\ Hz & 6,0\ kHz < f \leq 12,0\ kHz \\ df &= 562,50\ Hz & 12,0\ kHz < f \leq 24,0\ kHz. \end{aligned}$$

Para fazer a lista das linhas espectrais $X(k)$ que são tonais ou não-tonais, as três operações seguintes são realizadas:

a) Classificação dos máximos locais

Uma linha espectral $X(k)$ é classificada como um máximo local se:

$$X(k) > X(k-1) \text{ e } X(k) \geq X(k+1). \tag{3.11}$$

b) Listagem das componentes tonais e cálculo do nível de pressão sonora

Um máximo local é colocado na lista de componentes locais se:

$$X(k) - X(k+j) \geq 7 \text{ dB}, \tag{3.12}$$

onde j é escolhido de acordo com:

$$
\begin{aligned}
&j = -2, +2 && \text{para} \quad 2 < k < 63 \\
&j = -3, -2, +2, +3 && \text{para} \quad 63 \leq k < 127 \\
&j = -6, ..., -2, +2, ..., +12 && \text{para } 127 \leq k \leq 250.
\end{aligned}
$$

Se $X(k)$ é classificado como um componente tonal, então os seguintes parâmetros são listados:

- Número do índice k da linha espectral.
- Nível de pressão sonora

$$X_{tm}(k) = 10.\log_{10}\left\{10^{\frac{X(k-1)}{10}} + 10^{\frac{X(k)}{10}} + 10^{\frac{X(k+1)}{10}}\right\}, \text{ em dB} \tag{3.13}$$

- *Tonal flag.*

Posteriormente, todas as linhas espectrais dentro da faixa de freqüência examinada, serão colocadas em $-\infty$ dB.

c) Listagem das componentes não-tonais e cálculo da potência

As componentes não-tonais (ruído) são calculados das linhas espectrais restantes. Para calcular as componentes não-tonais destas linhas espectrais $X(k)$, as bandas críticas $z(k)$ são determinadas usandos as tabelas "*Critcal band boundaries*", tabelas D.2a, D.2b, D.2c para *Layer* I [6]. No *Layer* I, 23 bandas críticas são usadas para a taxa de amostragem de 32 kHz, 24 bandas críticas para 44,1 kHz e 25 bandas críticas são usadas para 48 kHz. Em cada banda crítica, as potências das linhas espectrais (que restaram após as componentes tonais terem sido zerados) são somadas para formar o nível de pressão sonora das novas componentes não-tonais $X_{nm}(k)$ correspondente àquela banda crítica.

Os seguintes parâmetros são listados:

- Índice de número k da linha espectral mais próxima da média geométrica da banda crítica.
- Nível de pressão sonora $X_{nm}(k)$ em dB.
- *Flag não-tonal.*

3.5.5 Decimação dos Mascaradores

Decimação é um procedimento que é usado para reduzir o número de mascaradores que são considerados para o cálculo do limiar de mascaramento global.

a) As componentes tonais $X_{tm}(k)$ e não-tonais $X_{nm}(k)$ são considerados para o cálculo do limiar de mascaramento somente se:

$$X_{tm} \geq LT_q(k) \text{ ou } X_{nm} \geq LT_q(k). \tag{3.14}$$

Nesta expressão $LT_q(k)$ é o limiar absoluto, ou limiar em silêncio, na freqüência de índice k. Estes valores são dados nas tabelas D.1a, D.1b, D.1c para *Layer* I [6].

b) Decimação de dois ou mais componentes tonais $X_{tm}(k)$ com uma distância menor do que 0,5 Bark: Mantém as componentes $X_{tm}(k)$ com a maior potência, e remove as componentes $X_{tm}(k)$ de menor potência da lista de componentes tonais. Para esta operação, é usada uma janela deslocada no domínio da banda crítica com largura de 0,5 Bark.

Na próxima etapa, o índice j é usado para indicar as componentes tonais e não-tonais relevantes da lista decimada combinada.

3.5.6 Cálculo do Limiares Individuais de Mascaramento

Das $N/2$ amostras originais do domínio de freqüência, indexadas por k, só um sub-conjunto das amostras indexadas por i é considerado para o cálculo do limiar de mascaramento global. As amostras usadas são encontradas nas tabelas D.1a, D.1b, D.1c para *Layer* I [6].

No *Layer* I, para as linhas de freqüência correspondentes à região de freqüência que está encoberta pelas primeiras seis sub-bandas, nenhuma sub-amostragem é usada. Para a região de freqüência correspondente às próximas seis sub-bandas, cada duas linhas espectrais são consideradas. Finalmente, nos casos de taxas de amostragem de 44,1 kHz e 48 kHz, nas regiões de freqüência correspondentes às sub-bandas restantes, cada quatro linhas espectrais são consideradas acima dos 20 kHz. No caso de taxa de amostragem de 32 kHz, na região de freqüência correspondente às sub-bandas restantes, cada oito linhas espectrais são consideradas acima dos 15 kHz. (Ver tabelas D.1a, D.1b, D.1c para *Layer* I [6]).

O número de amostras, n, no domínio de freqüência sub-amostrado é diferente, dependendo da taxa de amostragem e do *layer*.

Taxa de amostragem de 32 kHz:	n=108 para *Layer* I
Taxa de amostragem de 44,1 kHz:	n=106 para *Layer* I
Taxa de amostragem de 48 kHz:	n=102 para *Layer* I.

Para cada componente tonal e não-tonal é apontado o valor do índice i mais próximo correspondente à freqüência da linha espectral original $X(k)$. Este índice i é dado na tabela D.1a, D.1b, D.1c para o *Layer* I [6].

Os limiares de mascaramento individuais das componentes tonais e não-tonais são dados pela seguinte expressão:

$$LT_{tm}[z(j), z(i)] = X_{tm}[z(j)] + av_{tm}[z(j)] + vf[z(j), z(i)] \text{ dB} \quad (3.15)$$

$$LT_{nm}[z(j), z(i)] = X_{nm}[z(j)] + av_{nm}[z(j)] + vf[z(j), z(i)] \text{ dB} \quad (3.16)$$

Nessa fórmula, LT_{tm} e LT_{nm} são os limiares de mascaramento individual na taxa de banda crítica z, em Bark, do componente mascarado na taxa de banda crítica do mascarador z_m em Bark. Os valores em dB podem ser tanto positivos quanto negativos. O termo $X_{tm}[z(j)]$ é o nível de pressão sonora do componente de mascaramento com o índice de número j na taxa da banda crítica correspondente $z(j)$. O termo av é chamado de índice de mascaramento e vf de função de mascaramento do componente de mascaramento $X_{tm}[z(j)]$. O índice de mascaramento av é diferente para mascaradores tonais e não-tonais (av_{tm} e av_{nm}, respectivamente).

Para mascaradores tonais:

$$av_{tm} = -1,525 - 0,275.z(j) - 4,5 \text{ dB}. \quad (3.17)$$

Para mascaradores não-tonais:

$$av_{nm} = -1,525 - 0,175.z(j) - 0,5 \text{ dB}. \quad (3.18)$$

A função de mascaramento vf de um mascarador é caracterizada por inclinações diferentes, que dependem da distância em Bark $dz = z(i) - z(j)$ para o mascarador. Nesta expressão, i é o índice da linha espectral na qual a função de mascaramento é calculada e j é o índice do mascarador. A taxa da banda crítica $z(i)$ e $z(j)$ pode ser encontrada nas tabelas D.1a, D.1b, D.1c para *Layer* I [6]. A função de mascaramento, que é a mesma para componentes tonais e não-tonais, é dada por:

$$\begin{aligned}
vf[z(j), z(i)] = &\ 17.(dz + 1) - (0,4.X[z(j)] + 6) \text{ dB} && \text{para} && -3 \leq \text{dz} < -1 && \text{Bark} \\
vf[z(j), z(i)] = &\ (0,4.X[z(j)] + 6).dz \text{ dB} && \text{para} && -1 \leq \text{dz} < -0 && \text{Bark} \\
vf[z(j), z(i)] = &\ 17.dz \text{ dB} && \text{para} && 0 \leq \text{dz} < 1 && \text{Bark} \\
vf[z(j), z(i)] = &\ -(dz - 1).(17 - 0,15.X[z(j)]) - 17 \text{ dB} && \text{para} && 1 \leq \text{dz} < 8 && \text{Bark.}
\end{aligned}$$

Nestas expressões $X[z(j)]$ é o nível de pressão sonora do j-ésimo componente de mascaramento em dB. Para não aumentar a complexidade de implementação, o mascaramento não é considerado para "grandes" distâncias (LT_{tm} e LT_{nm} são colocadas em $-\infty$ fora desta faixa), isto é, se $dz < -3$ Bark, ou $dz \geq 8$ Bark.

3.5.7 Determinação do Limiar Global de Mascaramento

O limiar global de mascaramento $LT_g(i)$ na i-ésima amostra da freqüência é extraído das inclinações dos limiares de mascaramento individual de cada j-ésimo componente tonal e não-tonal e do

limiar em silêncio $LT_q(i)$. Isto também é dado pelas tabelas D.1a, D.1b, D.1c para *Layer* I [6]. O limiar global de mascaramento é encontrado pela soma das potências correspondentes aos limiares de mascaramento individual e ao limiar em silêncio.

$$LT_g(i) = 10.\log_{10}\left(10^{LT_q(i)/10} + \sum_{j=1}^{m} 10^{LT_{tm}(z(j),z(i))/10} + \sum_{j=1}^{n} 10^{LT_{nm}(z(j),z(i))/10}\right). \tag{3.19}$$

O número total de mascaradores tonais é dado por m, e o número total de mascaradores não tonais é dado por n. Para um dado i, a distância de j pode ser reduzida limitando-se os componentes de mascaramento que estão entre -8 e +3 Bark de i. Fora deste limite LT_{tm} e LT_{nm} valem $-\infty$ dB.

3.5.8 Determinação do Limiar Mínimo de Mascaramento por Sub-Banda

O limiar mínimo de mascaramento $LT_{min}(n)$ na sub-banda n é determinado pela seguinte expressão:

$$LT_{min}(n) = \text{MÍN}[LT_g(i)] \text{ dB}, \tag{3.20}$$

$$f(i) \text{ na sub-banda } n, \tag{3.21}$$

onde $f(i)$ é a freqüência da i-ésima amostra de freqüência. As $f(i)$ são encontradas nas tabelas D.1a, D.1b, D.1c para *Layer* I [6]. Um limiar mínimo de mascaramento $LT_{min}(n)$ é computado para cada sub-banda.

3.5.9 Cálculo do SMR em Cada Sub-Banda

A razão sinal-máscara

$$SMR_{sb} = L_{sb}(n) - LT_{min}(n) \text{ dB} \tag{3.22}$$

é computada para cada sub-banda n.

Com o valor do SMR_{sb} é possível determinar o número de bits que serão alocados por sub-banda (seção 3.8).

3.6 Banco de Filtros - PQMF (*Analysis Subband filter*)

Todo codificador de áudio utiliza-se de um bloco de mapeamento tempo-freqüência com o objetivo de extrair do domínio do tempo do sinal de entrada um conjunto de parâmetros que amenizem os efeitos de ruído de quantização e codificação, seguindo uma métrica de distorção perceptual. A ferramenta mais comumente usada para este tipo de mapeamento é o banco de filtros, que é um conjunto de

filtros passa-faixa em paralelo cobrindo toda a faixa espectral. O banco de filtros divide o espectro do sinal em sub-bandas de freqüência e gera uma série de coeficientes indexados no tempo, representando a potência do sinal em cada sub-banda. Para fornecer a informação explícita da distribuição do sinal e conseqüentemente a potência de mascaramento sobre o plano tempo-freqüência, o banco de filtros tem um papel essencial na identificação das irrelevâncias perceptuais, quando usado em conjunto com o modelo psicoacústico. Ao mesmo tempo, os parâmetros de tempo-freqüencia gerados pelo banco de filtros provê um mapeamento do sinal, que é convenientemente manipulado para modelar a distorção de codificação de modo a casar a distribuição tempo-freqüência com a potência de mascaramento. Em outras palavras, o banco de filtros facilita a análise psicoacústica, assim como o modelamento do ruído percebido. Adicionalmente, por decompor o sinal em componentes divididas por sub-bandas, o banco de filtros também auxilia na redução de redundâncias estatísticas [7].

O MPEG 1 *Layer I* utiliza um banco de filtros conhecido como *Pseudo - Quadrature Mirror Filters* (PQMF) de 32 sub-bandas. O PQMF é caracterizado pelas seguintes propriedades:

- Facilidade de projeto, um único protótipo de filtro FIR.
- Fase linear, e conseqüentemente atraso de grupo constante.
- Algoritmo rápido.
- Uniformidade, resposta de fase linear por canal.
- Baixa complexidade, um filtro mais modulação.
- Sub-amostragem crítica.

O banco de filtros PQMF consiste de K canais, cada um dos quais é um filtro passa-baixa $h[n]$ modulado por um cosseno. O banco de filtros utilizado neste codificador emprega o seguinte filtro de análise $h[n]$ [7].

$$h_k[n] = h[n] \cos\left[\left(k + \frac{1}{2}\right)(n-16)\frac{\pi}{32}\right], \quad k = 0, 1, ..., 31 \ \text{ e } \ n = 0, 1, ..., 511, \tag{3.23}$$

onde k é o índice de freqüência e n é o índice de tempo. Os coeficientes do filtro que descrevem o protótipo do filtro $h[n]$ são encontrados na tabela C.1 do anexo C da recomendação [6].

Após algumas manipulações algébricas podemos chegar no seguinte resultado:

$$h_k[n] = h[n] \cos\left[\left(k + \frac{1}{2}\right)\left(n - \frac{(N-1)}{2}\right)\frac{\pi}{32} + \phi_k\right], \tag{3.24}$$

para $N = 513$.

Podemos notar pela forma geral do banco de filtros PQMF que a fase ϕ_k no MPEG 1 é igual a:

$$\phi_k = \frac{\pi}{2}\left(\frac{N-1-K}{K}\right)\left(k+\frac{1}{2}\right), \tag{3.25}$$

que satisfaz para os valores de $N = 513$ e $K = 32$ o cancelamento de *aliasing* na vizinhança da sub-banda, que requer que $\phi_k - \phi_{k-1}$ seja igual a um multiplo ímpar de $\pi/2$ (Fig. 3.14).

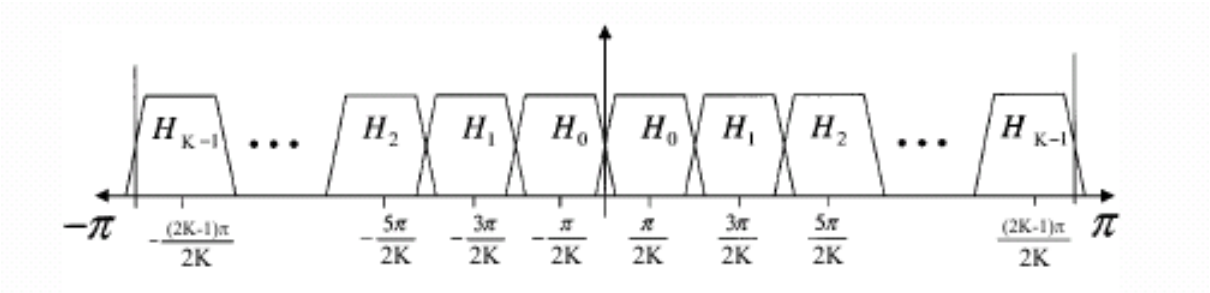

Figura 3.14: Resposta de magnitude, banco de filtros de K-bandas.

Cada filtro $h_k[n]$ é o protótipo $h[n]$ modulado por um cosseno equação (3.23). Portanto, a resposta na freqüência do filtro $h_k[n]$ é o filtro protótipo $h[n]$ deslocado tanto para a esquerda quanto para a direita por:

$$f_k = \pm\frac{F_s}{2K}\left(k+\frac{1}{2}\right) \tag{3.26}$$

para $k = 0, 1, 2, ..., 31$, com cada cópia tendo uma largura de banda nominal de $F_s/64$.

Na implementação do banco de filtros de análise PQMF, segundo a recomendação [6], a cada passagem do filtro temos um *buffer* de entrada $x[n]$ de 512 amostras de áudio que são multiplicadas pelos coeficientes do filtro, resultando em uma amostra por sub-banda na saída do filtro. Uma vez que o último coeficiente do filtro é igual a zero, o banco de filtro é implementado como se o comprimento do filtro fosse de $N = 512$. Como o PQMF é criticamente sub-amostrado, o banco de filtros pode ser implementado como um bloco que transforma 512 amostras em uma amostra por sub-banda na saída do filtro, e que a cada vez recebe mais 32 amostras novas [3].

Uma implementação direta de um banco de filtros PQMF resulta em $512*32 = 16384$ multiplicações e $511 * 32 = 16352$ somas para cada novo conjunto de 32 amostras (ou 512 multiplicações e adições por amostra). Na recomendação [6] é descrita uma implementação de complexidade média, que envolve 80 multiplicações e adições por amostra (Fig. 3.15), dadas por:

$$y_m[k] = \sum_{r=0}^{63} M[k,r].\sum_{p=0}^{7}[C[r+64p\ ]],\ \text{para todo } m \text{ e } k = 0, ..., 31, \tag{3.27}$$

onde $M[k,r] = \cos\left(\left(k+\frac{1}{2}\right)(r-16)\frac{\pi}{32}\right)$ e $C[n]$ são os coeficientes encontrados na tabela C.1 da recomendação [6].

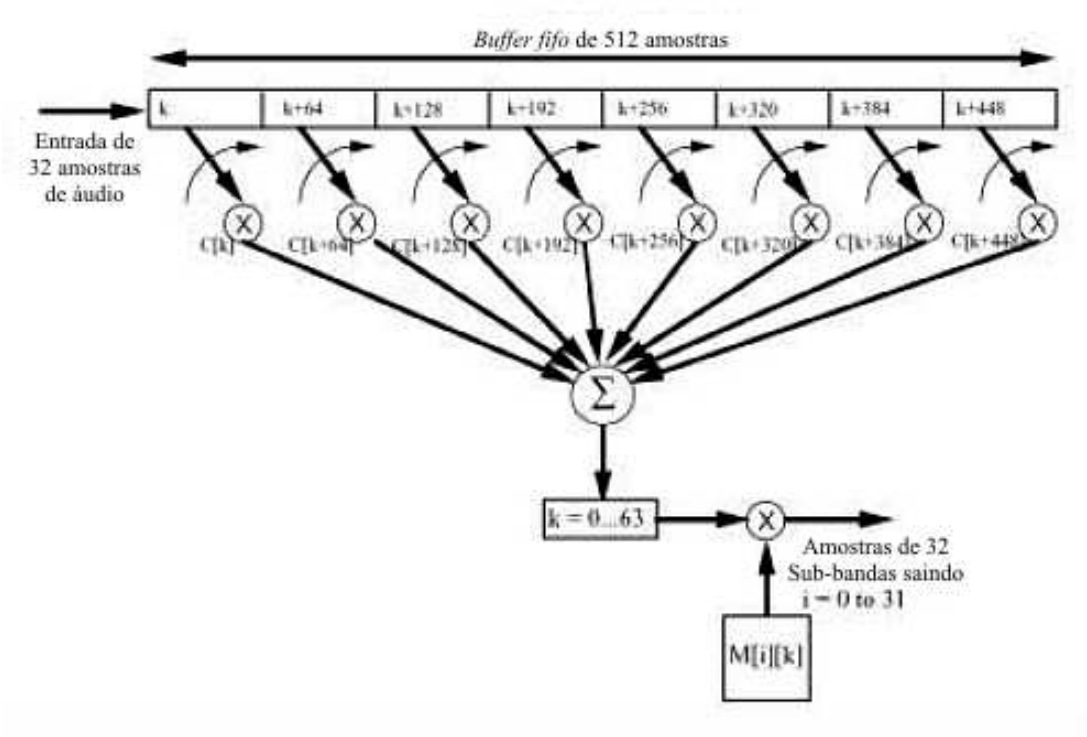

Figura 3.15: Visualização do Banco de Filtros.

Nestas equações, k é o índice de freqüência, $y_m[k]$ é a saída do k-ésimo filtro de análise após o processamento do m-ésimo bloco de 32 novas amostras e $x_m[n]$ representa um bloco de 512 amostras de entrada com o tempo reverso, ou seja, $x_m[n] = x[32.(m+1) - 1 - n]$ [3].

Em resumo, no primeiro *loop* de codificação começamos com o *buffer* de entrada do filtro igual a zero, exceto as 32 primeiras posições, que recebem as 32 primeiras amostras de áudio. Após esse *buffer* passar pelo PQMF temos como resultado 32 amostras na saída, uma para cada sub-banda. Essas 32 amostras de áudio da entrada são deslocadas, e são recebidas mais 32 novas amostras de áudio, resultando assim em mais 32 amostras na saída do processo. Este processo se repete por 12 vezes, ou seja, até completarmos 12 amostras por sub-banda. Assim sendo, a cada *loop* de codificação entramos com 384 novas amostras de áudio, resultando em 12 amostras por 32 sub-bandas (384 amostras) na saída do filtro.

Para melhor compreensão do algoritmo sugerido na recomendação do MPEG 1 *Layer* I [6], podemos ver o diagrama de blocos da Fig. 3.16. Este fluxograma é extraído da figura C.4 da recomendação [6] e representa um *loop* de codificação.

3.7 Cálculo do Fator de Escala (*Scalefactor*)

O cálculo do fator de escala para cada sub-banda é efetuado em cada 12 amostras da sub-banda. O valor máximo absoluto destas 12 amostras é determinado. O menor valor na tabela B.1 [6], "*Layer*

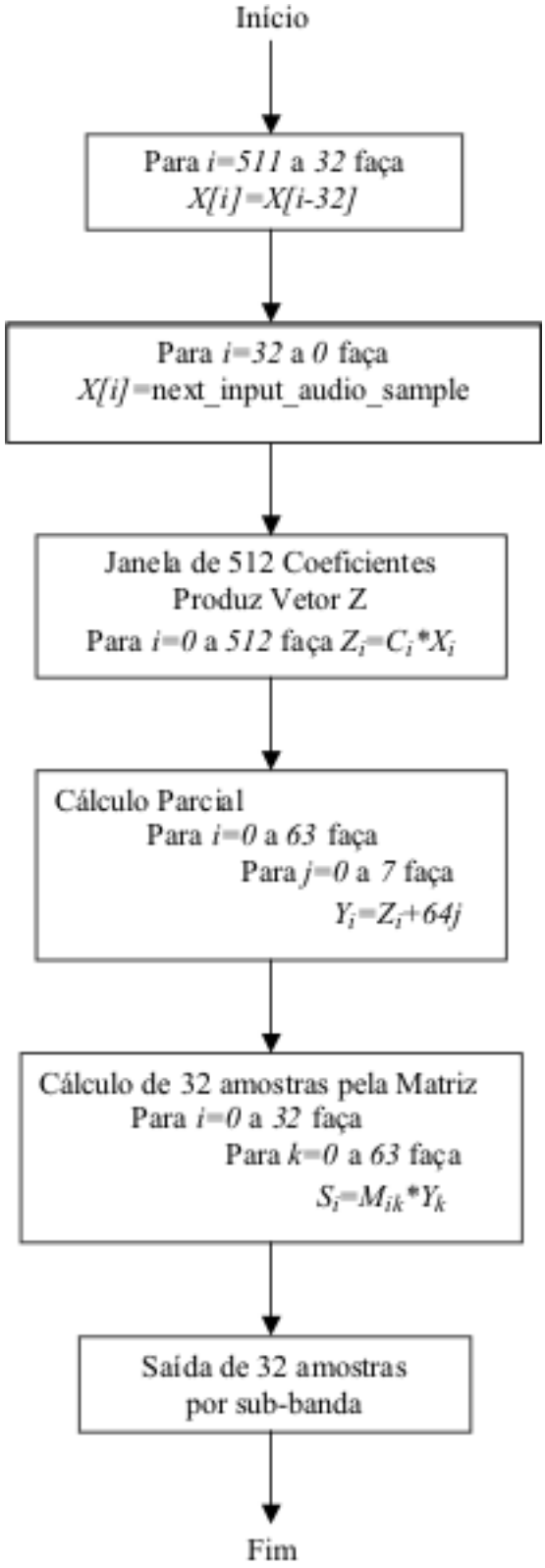

Figura 3.16: Fluxograma do Banco de Filtros.

I, II Scalefactors" imediatamente maior do que a amostra máxima da sub-banda é usado como *scalefactor*.

3.8 Alocação de Bits

Antes de ajustar uma taxa de bits fixa, o número de bits disponíveis para a codificação das amostras e dos *scalefactors* deve ser determinado. Este número pode ser obtido subtraíndo-se do número total de bits disponível *cb*, o número de bits necessários para o cabeçalho *bhdr* (32 bits), o CRC *bcrc*, se usado (16 bits), a alocação de bits *bbal* e o número de bits necessários para dados auxiliares *banc*:

$$adb = cb - (bhdr + bcrc + bbal + banc)$$

O total de bits restantes pode ser usado para codificar as amostras por sub-banda e os *scalefactors*. O princípio usado no procedimento de alocação é a minimização da razão ruído-mascaramento total do *frame*, considerando que o número de bits utilizados não exceda o número de bits permitidos para este *frame*. O número possível de bits alocados por amostra pode ser encontrado na tabela 2.4.2.5 da parte principal da recomendação (*Audio data, Layer I*)[6]; esta faixa é de 0 a 15 bits, excluindo a alocação de 1 bit.

O procedimento de alocação é um método iterativo onde, em cada passo de iteração, o número de níveis das amostras da sub-banda de maior relevância é aumentado.

Primeiro, a razão máscara-ruído "MNR" para cada sub-banda é calculada subtraindo-se da razão sinal-ruído "SNR" a razão sinal-máscara "SMR":

$$MNR = SNR - SMR.$$

A razão sinal-ruído pode ser encontrada na tabela C.2 "*Layer I Signal-to-Noie Ratio*"[6]. A razão sinal-máscara é a saída do modelo psicoacústico.

Então, zero bit é alocado para as amostras e os *scalefactors*. O número de bits para as amostras *bspl* e o número de bits para os *scalefactors bscf* são colocados em zero. Posteriormente, um processo iterativo é iniciado. Cada *loop* de iteração contém os seguintes passos:

- Determina-se o MNR mínimo de todas as sub-bandas.
- A acurácia da quantização da sub-banda com o mínimo MNR é aumentada para usar o próximo número de bits mais alto.
- O novo MNR desta sub-banda é calculada.
- O *bspl* é atualizado de acordo com o número adicional de bits necessários. Se um número diferente de zero de bits é determinado para uma sub-banda pela primeira vez, *bscf* tem o seu valor incrementado em 6 bits. Então *adb* é calculado usando-se a fórmula:

$$adb = cb - (bhdr + bcrc + bbal + bscf + bspl + banc).$$

O processo iterativo é repetido enquanto adb é maior do que qualquer acréscimo possível de *bspl* e *bscf* com apenas um *loop*.

3.9 Quantização e Codificação das Amostras de Cada Sub-Banda

Um quantizador linear com uma representação simétrica em zero é usada para quantizar as amostras das sub-bandas. Esta representação previne pequenas mudanças de valores em torno do zero dos diferentes níveis de quantização. Cada amostra da sub-banda é normalizada, dividindo-se o seu valor pelo fator de escala para obter X, e quantizada na seguinte seqüência:

- Calcular $AX + B$.
- Tomar os N bits mais significativos.
- Inverter o Bit Mais Significativo (MSB).

A e B podem ser encontrados na tabela C.3, "*Layer I Quantization Coefficients*"[6]. N representa o número necessário de bits para codificar o número de passos. A inversão do bit mais significativo (MSB) é feita de modo a evitar que o código seja representado apenas por '1', porque o código todo igual a '1' é usado como a palavra de sincronização.

3.10 Sintaxe do *Bitstream*

3.10.1 Seqüência de Áudio

Sintaxe	No. de bits	Mnemônico
Audio sequence()		
{		
while (nextbits()==syncword){		
frame()		
}		
}		

3.10.2 *Frame* do Áudio

Sintaxe	No. de bits	Mnemônico
frame()		
{		
header()		
error_ check()		
audio_ data()		
ancilary_ data()		
}		

3.10.3 Cabeçalho

Sintaxe	No. de bits	Mnemônico
header()		
{		
syncword	12	bslbf
ID	1	bslbf
layer	2	bslbf
protection_ bit	1	bslbf
bitrate_ index	4	bslbf
sampling_ frequency	2	bslbf
padding_ bit	1	bslbf
private_ bit	1	bslbf
mode	2	bslbf
mode_ extension	2	bslbf
copyright	1	bslbf
original/copy	1	bslbf
emphasis	2	bslbf
}		

3.10.4 Checagem de Erro

Sintaxe	No. de bits	Mnemônico
error_ check()		
{		
if(protection_ bit==0){		
crc_ check	16	rpchof
}		
}		

3.10.5 Dados de Áudio, *Layer* I

Sintaxe	No. de bits	Mnemônico
audio_ data()		
{		
for (sb=0; sb<bound; sb++)		
for (ch=0; ch<nch; ch++)		
allocation[ch][sb]	4	uimsbf
for (sb=bound; sb<32; sb++) {		
allocation[0][sb]	4	uimsbf
allocation[1][sb]= allocation[0][sb]	4	uimsbf
}		
for (sb=0; sb<32; sb++)		
for (ch=0; ch<nch; ch++)		
if(allocation[ch][sb]!=0)		
scalefactor[ch][sb]	6	uimsbf
for (s=0; s<12; s++)		
for (sb=0; sb<bound; sb++)		
for (ch=0; ch<nch; ch++)		
if(allocation[ch][sb]!=0)		
sample[ch][sb][s]	2..15	uimsbf
for (sb=bound; sb<32; sb++)		
if(allocation[0][sb]!=0)		
sample[0][sb][s]	2..15	uimsbf
}		
}		

3.11 Descrição dos Itens do *Bitstream*

3.11.1 Seqüência geral do áudio

- *frame* – Parte do *bitstream* que é decodificável por si só. No *Layer* I o *frame* contém informação para 384 amostras. Ele começa comum *syncword*, e termina ligeiramente antes do próximo *syncword*. Isto significa um número inteiro de *slots* (1 *slot* = quatro bytes, no *Layer* I).

3.11.2 *Frame* áudio

- **header** – Parte do *bitstream* contendo informação de sincronização e estado.
- **error_ check** – Parte do *bitstream* contendo informação para detecção de erro.
- **audio_ data** – Parte do *bitstream* contendo informação de amostras de áudio.
- **ancillary_ data** – Parte do *bitstream* que pode ser usada como dado auxiliar.

3.11.3 Header

- **syncword** – A palavra '1111 1111 1111'.
- **ID** – Um bit para indicar o ID do algoritmo. Igual a '1' para áudio ISO/IEC 11172-3, '0' é reservado.
- **protection_ bit** – Um bit para indicar aonde a redundância tem sido adicionada no *bitstream* do áudio para facilitar detecção de erro e encobri-los. Igual a '1' se não houver redundância adicionada, '0' caso contrário.
- **bitrate_ index** – Indica a taxa de bits. O valor igual a zero indica a condição de 'formato livre', no qual uma taxa de bits fixa que não precisa estar na lista pode ser usada. Taxa de bits fixa significa que um *frame* contém sempre N ou N+1 *slots*, dependendo do valor do *padding bit*. O *bitrate_ index* é um índice para uma tabela, que é diferente para diferentes *layers*.

  O *bitrate_ index* indica que a taxa de bits total independente do modo (*stereo, joint_ stereo, dual_ channel, single_ channel*).

bitrate_ index	taxa de bits especificada (kbits/s) *Layer* I
‘0000’	livre
‘0001’	32
‘0010’	64
‘0011’	96
‘0100’	128
‘0101’	160
‘0110’	192
‘0111’	224
‘1000’	256
‘1001’	288
‘1010’	320
‘1011’	352
‘1100’	384
‘1101’	416
‘1110’	448
‘1111’	proibido

De modo a permitir uma pequena possibilidade de atraso e complexidade, não é necessário que o decodificador dê suporte contínuo a taxas de bits variáveis no *layer* I. Entretanto no formato livre uma taxa de bits fixa, naturalmente diferente da tabelada, se faz necessária. O decodificador também não é obrigado a suportar taxas de bits maiores do que 448 kbits/s no *layer* I quando em formato livre.

- **sampling_ frequency** – Indica a freqüência de amostragem de acordo com a tabela seguinte:

sampling_ frequency	Freqüência especificada (kHz)
‘00’	44,1
‘01’	48
‘10’	32
‘11’	reservado

Uma reinicialização do decodificador de audio pode ser necessária no caso de uma mudança de taxa de amostragem.

- **padding_ bit** – Se este bit for igual a ‘1’, o *frame* contém um *slot* adicional para ajustar a taxa de bits média para a freqüência de amostragem; caso contrário, este bit será igual a ‘0’. *Padding* é necessário com a freqüência de amostragem de 44,1 kHz. *Padding* também pode ser necessário no

formato livre.

O *padding* deve ser aplicado à taxa de bits de modo que o comprimento acumulado dos *frames* codificados, após um certo número de *frames* de áudio não desviar mais do que (+0, -1 slot) do seguinte valor computado:

$$\text{comprimento do } \textit{frame} \text{ acumulado} = \sum_{\text{primeiro } \textit{frame}}^{\textit{frame} \text{ atual}} \left(\frac{\text{tamanho do } \textit{frame} * \text{taxa de bit}}{\text{freqüência de amostragem}} \right), \quad (3.28)$$

onde comprimento do *frame* = 384 para *Layer* I.

O seguinte método pode ser usado para determinar onde ou não usar o *padding*.

• para o primeiro *frame* de áudio:

resto = 0

padding = não

• para os *frames* de áudio subseqüentes:

if(Layer == 1) dif = (12*bitrate)% sampling_ frequency;
else dif = (144 * bitrate) % sampling_ frequency;
resto = resto - dif;
if(resto < 0){
padding = sim;
resto = resto + sampling_ frequency;
}
else padding = não;

- **private_ bit** – Bit para uso privado. Este bit não será usado no futuro pela ISO/IEC.

- **mode** – Indica o modo de acordo com a tabela seguinte. No *Layer* I o modo *joint_ stereo* é o modo *intensity_ stereo*.

mode	modo especificado
'00'	*stereo*
'01'	*joint_ stereo*
'10'	*dual_channel*
'11'	*single_channel*

No *Layer* I, em todos os modos exceto no *joint_ stereo*, o valor do limite (*bound*) é igual a 32. No modo *joint_ stereo* o limite é determinado pelo *mode_ extension*.

O modo *joint_ stereo* explora redundâncias e irrelevâncias espaciais para reduzir a taxa de dados da codificação do áudio. O modo *single_channel* significa que o sinal está apenas no canal direito, e o *dual_channel* significa que o áudio do canal esquerdo é uma cópia exata do canal direito, ou seja, não utiliza a localização espacial do som como no modo *stereo*.

- **mode_ extension** – Estes bits são usados em conjunto com o modo *joint_ stereo*. No *Layer* I ele indica que sub-bandas estão em *intensity_ stereo*. Todas as outras sub-bandas são codificadas em estéreo.

mode_ extension	
'00'	sub-bandas 4-31 em *intensity_ stereo*, limite=4
'01'	sub-bandas 8-31 em *intensity_ stereo*, limite=8
'10'	sub-bandas 12-31 em *intensity_ stereo*, limite=12
'11'	sub-bandas 16-31 em *intensity_ stereo*, limite=16

Note que o modo estéreo é usado se o modo de bits especificar *stereo* ou, equivalentemente, se o modo de bits especificar *joint_ stereo*, e o *mode_ extension* especificar *intensity_ stereo* "off" e *ms_ stereo* "off".

- **copyright** – Se este bit é igual a '0', não há *copyright* na taxa de bit no ISO/IEC 11172-3, '1' significa *copyright* protegido.

- **original/copy** – Este bit é igual a '0' se o *bitstream* é uma cópia, e igual a '1' se for original.

- **emphasis** – Indica o tipo de de-ênfase que deve ser utilizada.

emphasis	ênfase especificada
'00'	nenhuma
'01'	50/15 microsegundos
'10'	reservado
'11'	CCITT J.17

3.11.4 Error check

- **crc_ check** – Uma palavra de 16 bits de checagem de paridade que pode ser usada opcionalmente para detecção de erro com o *bitstream* codificado.

3.11.5 Dados do áudio, *Layer* I

- **allocation[ch][sb]** – Indica o número de bits usados para codificar as amostras na sub-banda *sb* do canal *ch*. Para sub-bandas no modo intensity_ stereo, o *bitstream* contém somente uma alocação do dado por sub-banda.

allocation[ch][sb]	bits por amostra
'0000'	0
'0001'	2
'0010'	3
'0011'	4
'0100'	5
'0101'	6
'0110'	7
'0111'	8
'1000'	9
'1001'	10
'1010'	11
'1011'	12
'1100'	13
'1101'	14
'1110'	15
'1111'	proibido

Nota: Para o código '0000' nenhuma amostra é transferida.

- **scalefactor[ch][sb]** – Indica o fator da sub-banda *sb* de canal *ch* pelo qual as amostras requantizadas da sub-banda *sb* no canal *ch* será multiplicado. Os seis bits constituem um inteiro sem sinal, vide tabela B.1 "*Layer I scalefactors*".

- **sample[ch][sb][s]** – Representação codificada da *s*-ésima amostra na sub-banda *sb* do canal *ch*. Para sub-bandas no modo *intensity_ stereo* a representação codificada das amostras é válida para ambos os canais.

3.12 MPLF_GPA1 - O Codificador

Baseado na recomendação [6] e nos conceitos anteriormente apresentados, construímos o nosso codificador para MPEG-1 *Layer* I. O programa foi desenvolvido utilizando a linguagem de programação

C com o auxílio das bibliotecas *fftw-3.0.1* e *libsndfile-1.0.4* utilizadas para a realização da FFT e da leitura do arquivo em formato *wav*, respectivamente.

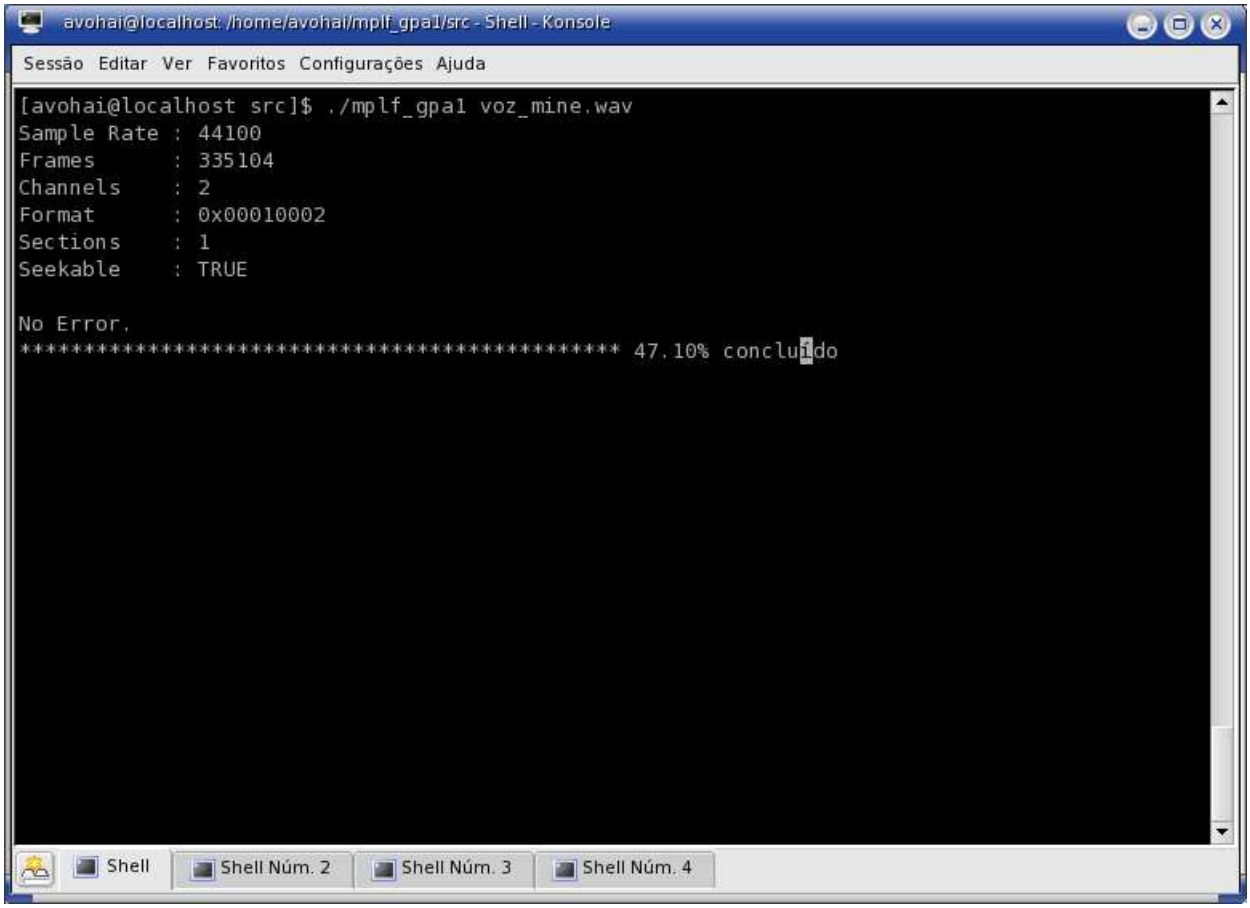

Figura 3.17: Interface do programa mplf_ gpa1.

O programa foi denominado "mplf_ gpa1" e é chamado via linha de comando, como podemos ver na Fig. 3.17. O usuário utiliza o programa digitando no *prompt* de comando o nome "mplf_ gpa1" seguido pelo nome do arquivo *wav* que ele deseja codificar. Após uma leitura inicial do cabeçalho do arquivo *wav*, se não houver erros, o arquivo é codificado e a saída gerada pelo programa é um arquivo com o mesmo nome do arquivo de entrada, porém com a extensão *mp1*.

Atualmente o programa codifica apenas arquivos *wav* com taxas de amostragem de 44,1 kHz ou de 48 kHz, podendo ser mono ou estéreo. Além disso, o usuário ainda não tem liberdade de escolha na taxa de bits utilizada para a escrita do arquivo *mp1*; este parâmetro pode ser modificado no arquivo "common.h", sendo necessário a recompilação do programa para a utilização do novo valor.

Outras limitações do programa são parâmetros que o usuário não tem liberdade de escolher, por exemplo: o CRC, que não foi implementado; o "*copyright*", que é fixo em '0'; o "*original/copy*", que é fixo em original; o "*emphasis*", fixo em nenhuma ênfase; e finalmente o "*mode_extension*", fixo em '00'.

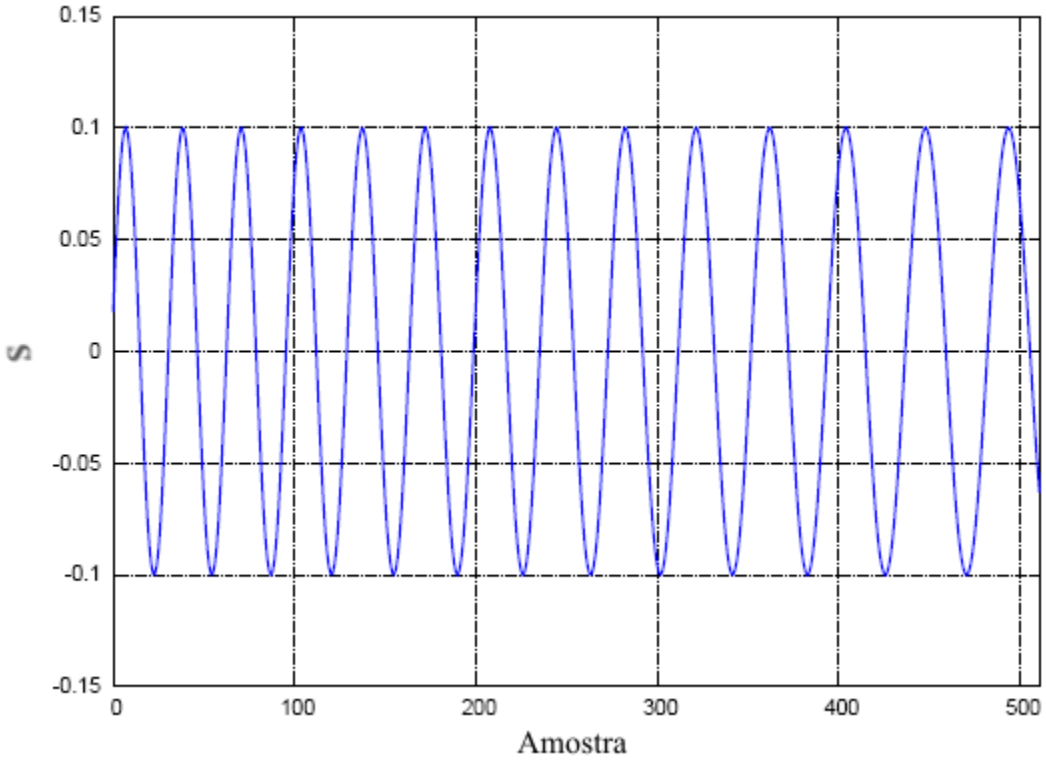

Figura 3.18: Trecho do arquivo fl3.wav.

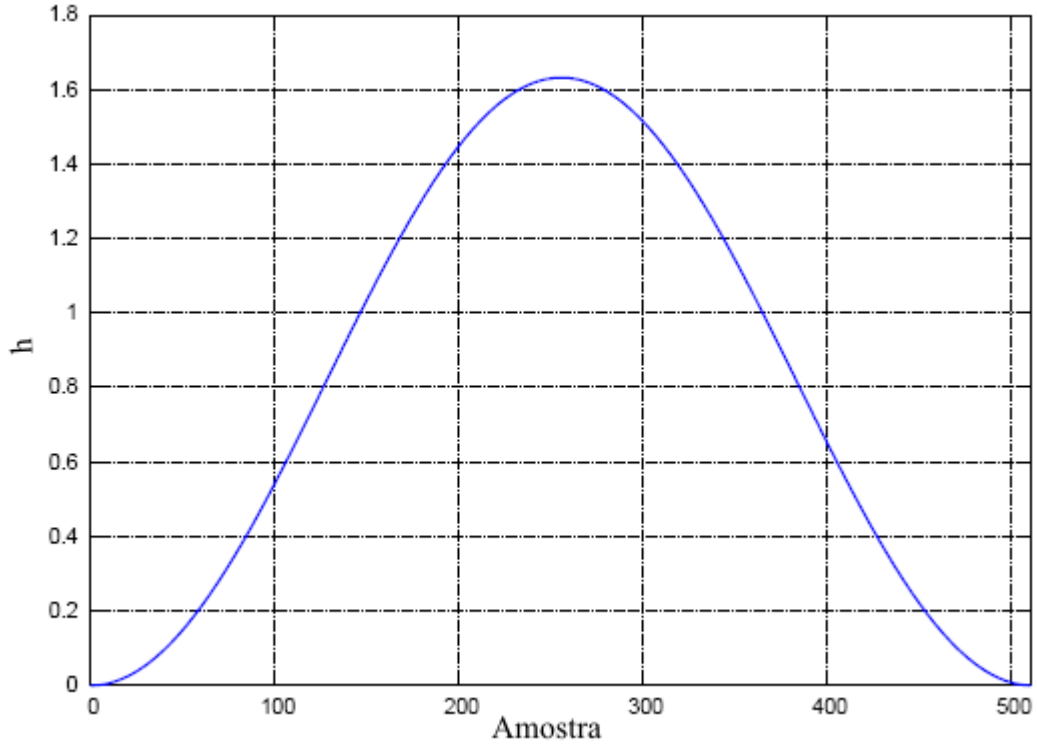

Figura 3.19: Janela de Hann.

Para a realização dos testes foram escolhidos alguns arquivos de tom de teste e trechos de músicas, sempre objetivando avaliar alguma característica especial do codificador. Na tabela 3.2 vemos os arquivos escolhidos e a peculiaridade de cada um.

Arquivo *wav*	Música - Artista	Objetivo
fl3.wav	Tom de Teste fl3	Verificar como o codificador se comporta com um sinal de freqüência variável.
fl5.wav	Tom de Teste fl5	Verificar como o codificador se comporta com um tom senoidal, de freqüências distintas, em cada canal.
fl7.wav	Tom de Teste fl7	Verificar como o codificador se comporta com um tom impulsivo semelhante ao ataque de um xilofone
allegro.wav	Allegro Scherzando Impromptu Opus 142,2 - Schubert	Verificar como o codificador se comporta na presença de um único instrumento, medindo a resposta ao ataque das teclas de piano.
op16.wav	Concerto para piano e orquestra Opus 16, Allegro moderato - Grieg	Verificar como o codificador se comporta na transição entre um trecho suave de cordas, passando pelo ataque de um piano, somado posteriormente ao ataque de instrumentos de cordas e sopro.
Berimbau.wav	Berimbau - Toquinho	Verificar como o codificador se comporta com relação ao grave de um violão, e também analisar o seu comportamento para sons de percussão.
Divagacoes.wav	Divagações Sobre o Tempo - Ventania	Verificar como o codificador se comporta com uma música *pop*, estilo mais comumente codificado, em trecho de voz e instrumental.
Divagacoes2.wav	Divagações Sobre o Tempo - Ventania	Verificar como o codificador se comporta com uma música *pop*, estilo mais comumente codificado, em trecho apenas instrumental.
Kamaitachi.wav	Kamaitachi - Sepultura	Verificar como o codificador se comporta na presença de ataques rápidos de bateria e sons distorcidos de contrabaixo e guitarra.
voz_mine.wav	Arquivo de voz	Verificar como o codificador se comporta com um arquivo de voz com ruído ambiente.

Tabela 3.2: Tabela com os arquivos utilizados para testar o codificador MPEG-1 *Layer* I mplf_ gpa1.

Os arquivos citados nesta tabela foram codificados com taxas de 384 e 448 kbits/s e tiveram a sua qualidade avaliada comparando-se auditivamente o arquivo original a estes arquivos. Para a realização

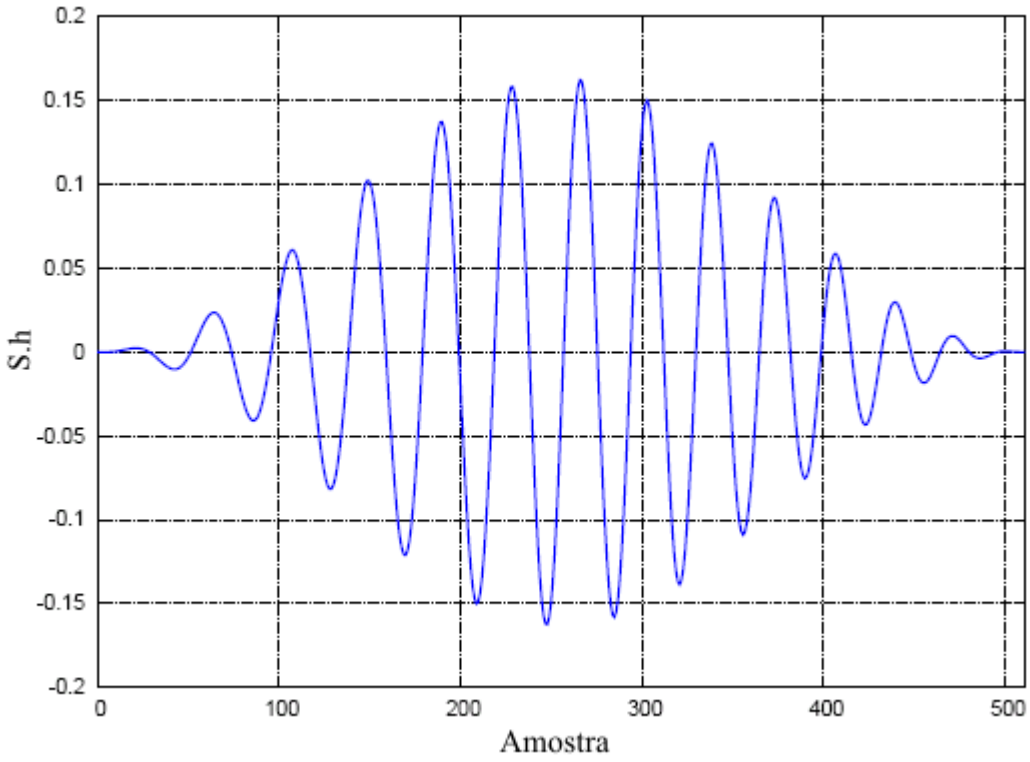

Figura 3.20: Trecho do arquivo fl3.wav janelado.

desta comparação, os arquivos *mp1* foram decodificados, de volta para o formato *wav*, utilizando o programa mpg123. Este programa foi escolhido devido à dificuldade em encontrar decodificadores *mp1* e por ter demonstrado um bom desempenho na decodificação de alguns arquivos de teste encontrados na *internet*.

A codificação destes arquivos foi feita tal qual a recomendação [6], e para demonstrarmos este processo, acompanharemos o processo de codificação de um *frame* do arquivo fl3.wav. Na Fig. 3.18 vemos um trecho do arquivo com as 512 amostras usadas no cálculo do modelo psicoacústico para um *frame*.

Essas amostras serão janeladas por uma janela de Hann, como na equação 3.8 (Fig. 3.19), será computada a FFT deste resultado (Fig. 3.20), e posteriormente será cálculado o nível de pressão sonora deste sinal, como na equação (3.9). Este resultado será normalizado para 96 dB (Fig. 3.21).

O nível de pressão sonora deste *frame* será a entrada do modelo psicoacústico. Neste modelo calcularemos os máximos locais tonais e não-tonais do *frame* e os seus limiares de mascaramento, vistos em vermelho e verde na Fig. 3.22, respectivamente. Com esses parâmetros é calculado o limiar global de mascaramento equação (3.19), visto em preto na mesma figura.

Ainda no modelo calcularemos o nível de pressão sonora por sub-banda (Fig. 3.23), equação (3.10), e o limiar mínimo de mascaramento (Fig. 3.24), equação (3.20). Com este dois parâmetros podemos calcu-

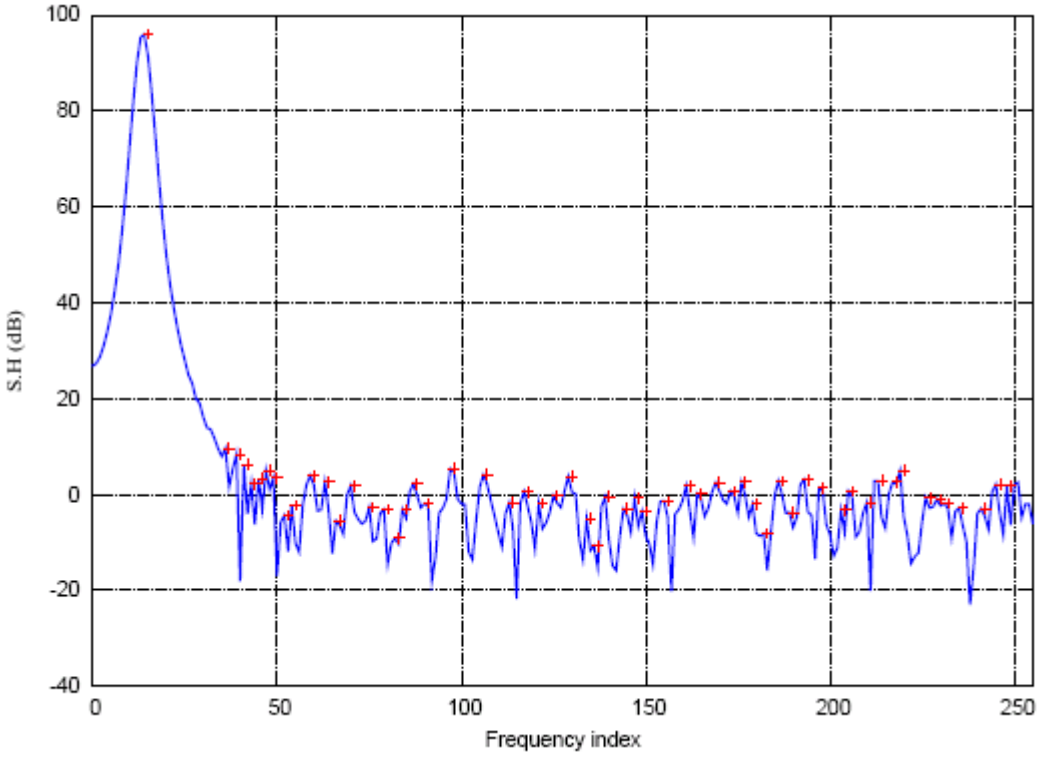

Figura 3.21: FFT do sinal janelado e seus máximos locais.

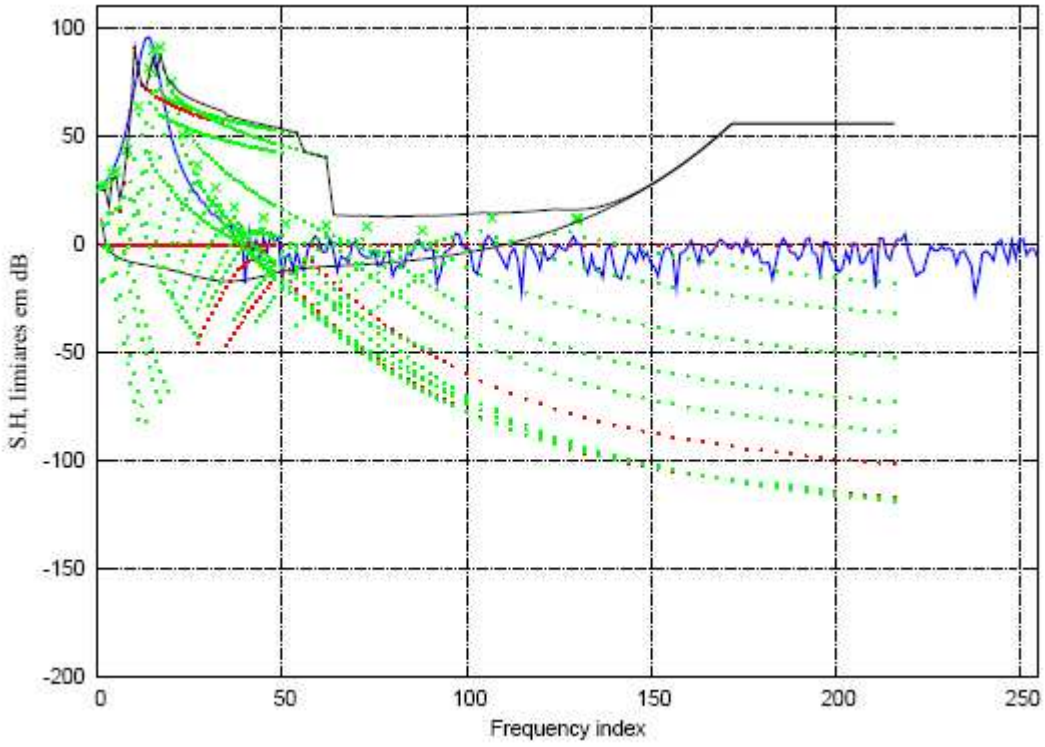

Figura 3.22: FFT do sinal janelado com seus limiares de mascaramento.

lar a razão sinal-máscara do *frame* (Fig. 3.25), equação (3.22).

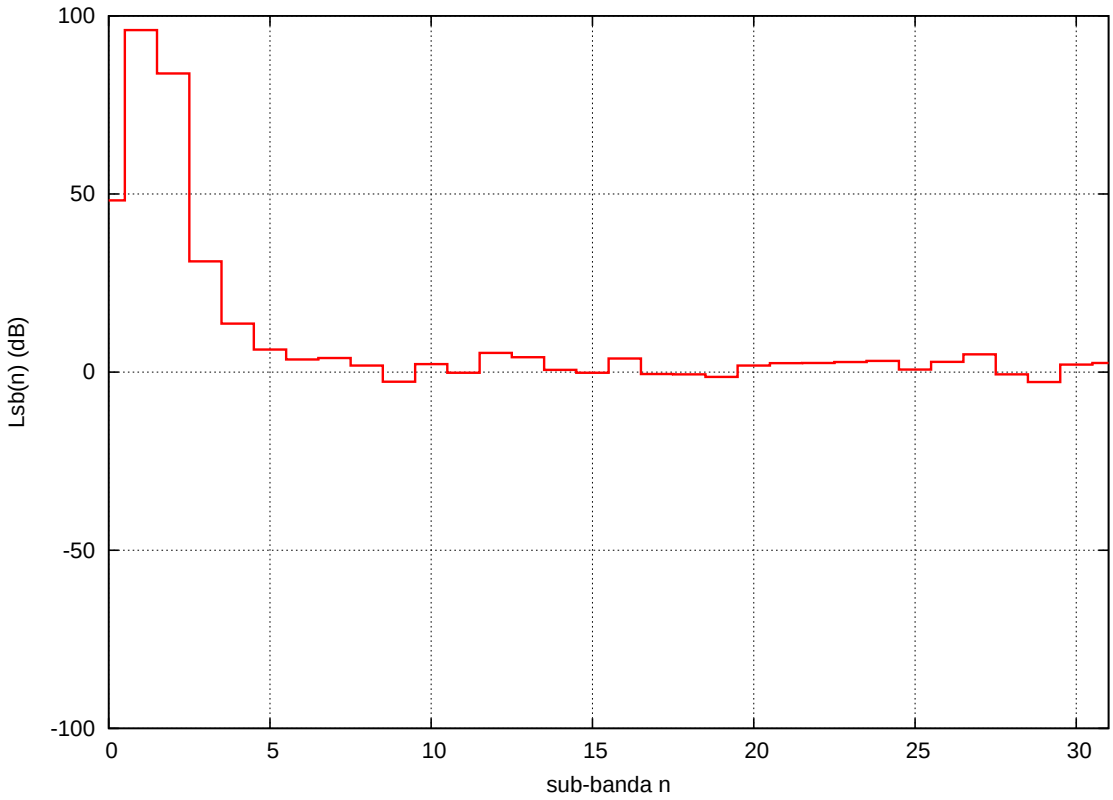

Figura 3.23: Nível de pressão sonora por sub-banda.

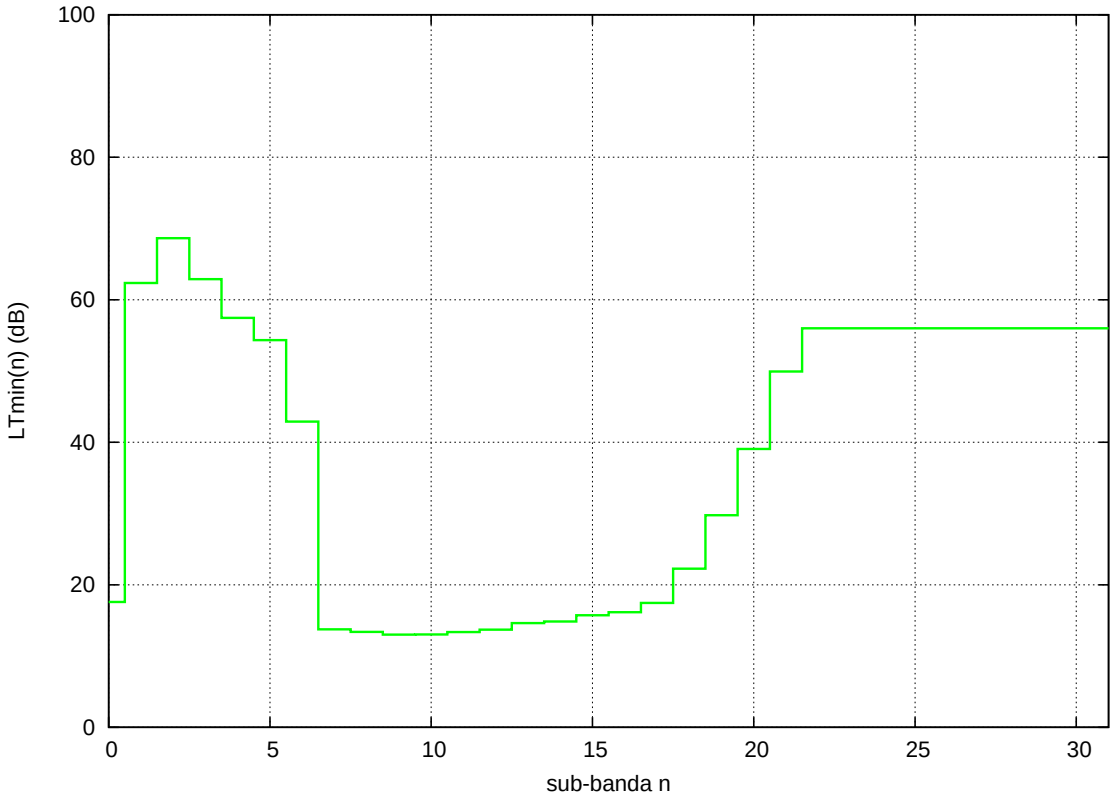

Figura 3.24: Limiar de mascaramento mínimo por sub-banda.

Até aqui o processo de codificação independe da taxa de bits que se deseja utilizar para codificação. A partir do processo de alocação de bits, seção 3.8, o processo torna-se dependente do *bitrate*, e conseqüentemente a razão sinal-ruído e a razão sinal máscara do sinal será diferente para taxas de bits diferentes, além do número de bits alocados, naturalmente.

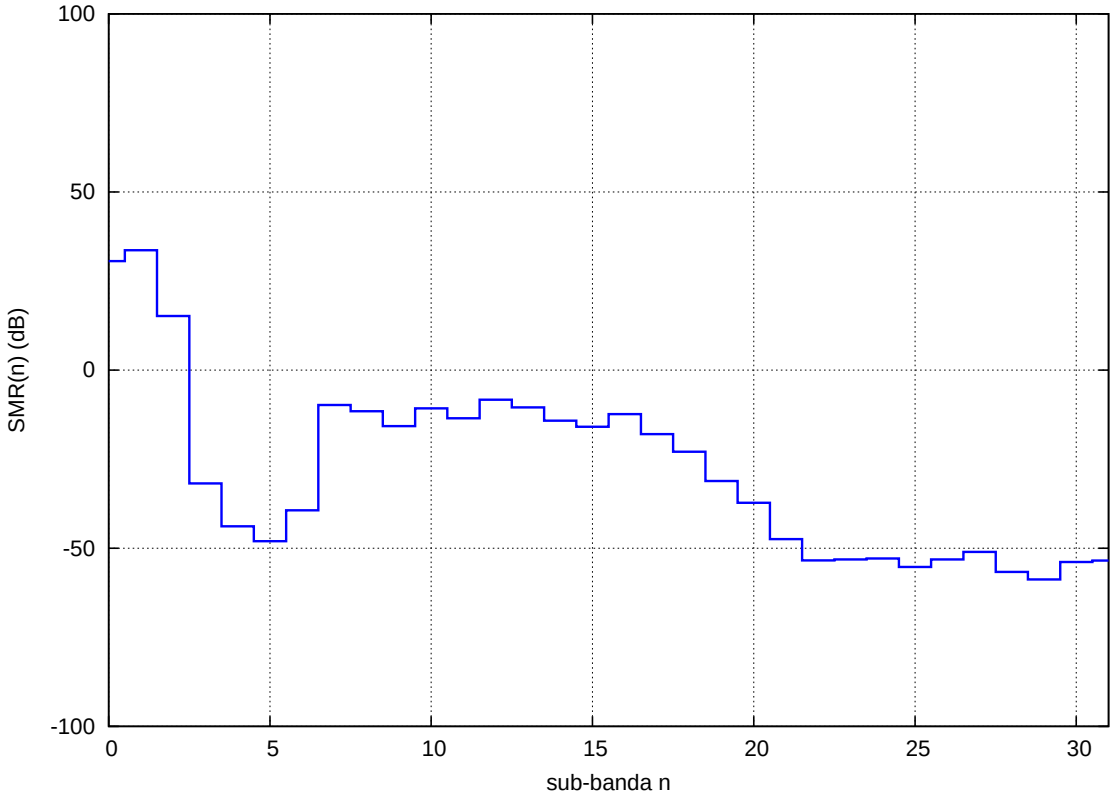

Figura 3.25: Razão sinal-máscara.

Nas próximas figuras vemos a SNR, a MNR e o número de bits alocados por sub-banda para a entrada vista na Fig. 3.18, para taxas de bits de 384, 416 e 448 kbits/s respectivamente. Nota-se que nas figuras relativas aos bits alocados por sub-banda, Fig. 3.32, 3.33 e 3.34, que conforme a taxa de bits aumenta as "curvas" tornam-se mais definidas, ficando mais próximas do nível de pressão sonora por sub-banda, ou seja, mais próximas do arquivo original.

Os arquivos de teste codificados forneceram dados suficientes para avaliarmos a qualidade do codificador mplf_ gpa1. A seguir comentaremos os resultados encontrados após análise utilizando uma placa de som *on-board*, com *chipset* AC97, de computador de 16 bits e fones de ouvido com resposta em freqüência plana, ±3 dB, de 100 Hz a 20 kHz e sensibilidade de 107 dB ± 3 dB, com potência máxima de 30 mW RMS. Vale ressaltar que esta é uma configuração comumente utilizada pela maioria dos usuários de computadores pessoais, e portanto, é relevante o desempenho do codificador nestas condições.

- Arquivos fl3, fl5 e fl7 – Estes arquivos tiveram excelentes resultados tanto para taxas de alocação

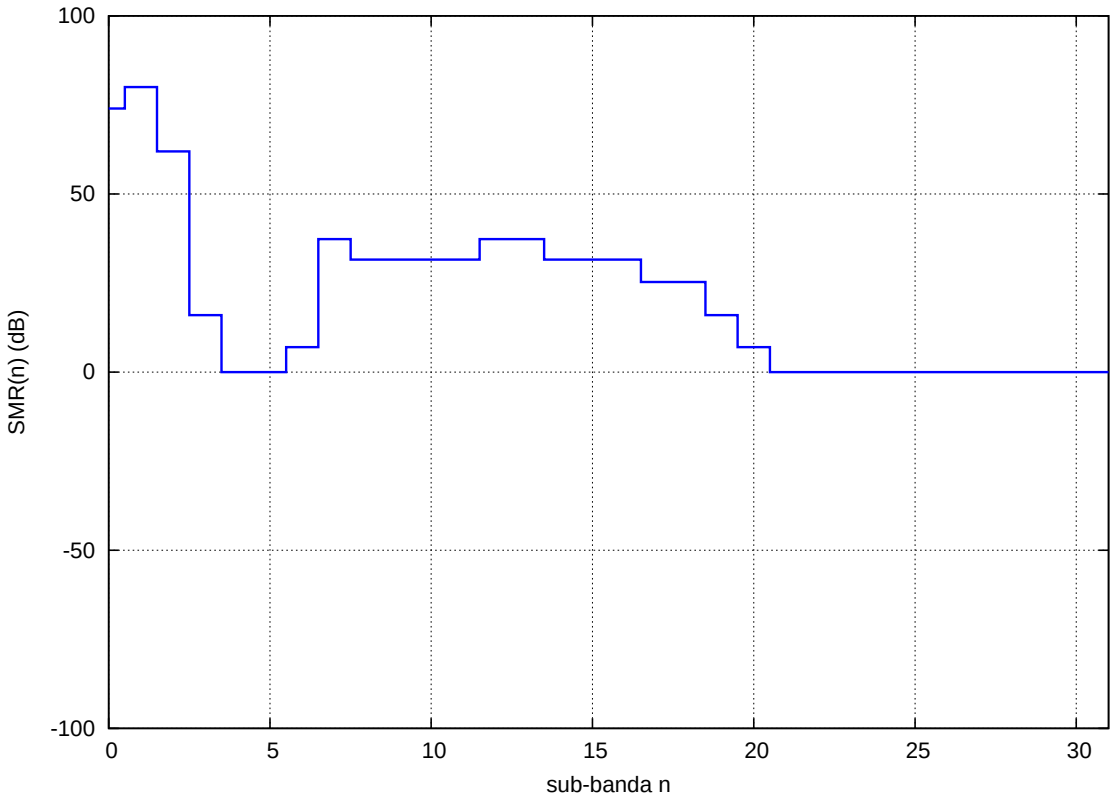

Figura 3.26: Razão sinal-ruído com *bitrate* de 384 kb/s.

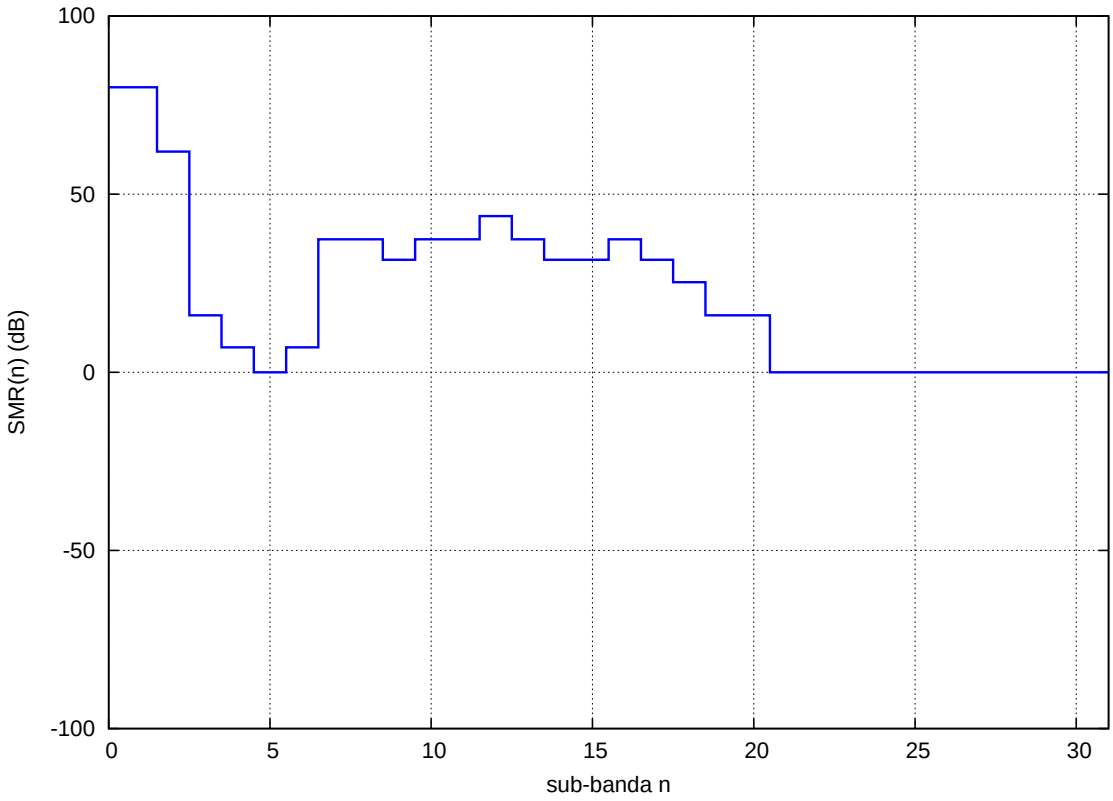

Figura 3.27: Razão sinal-ruído com *bitrate* de 416 kb/s.

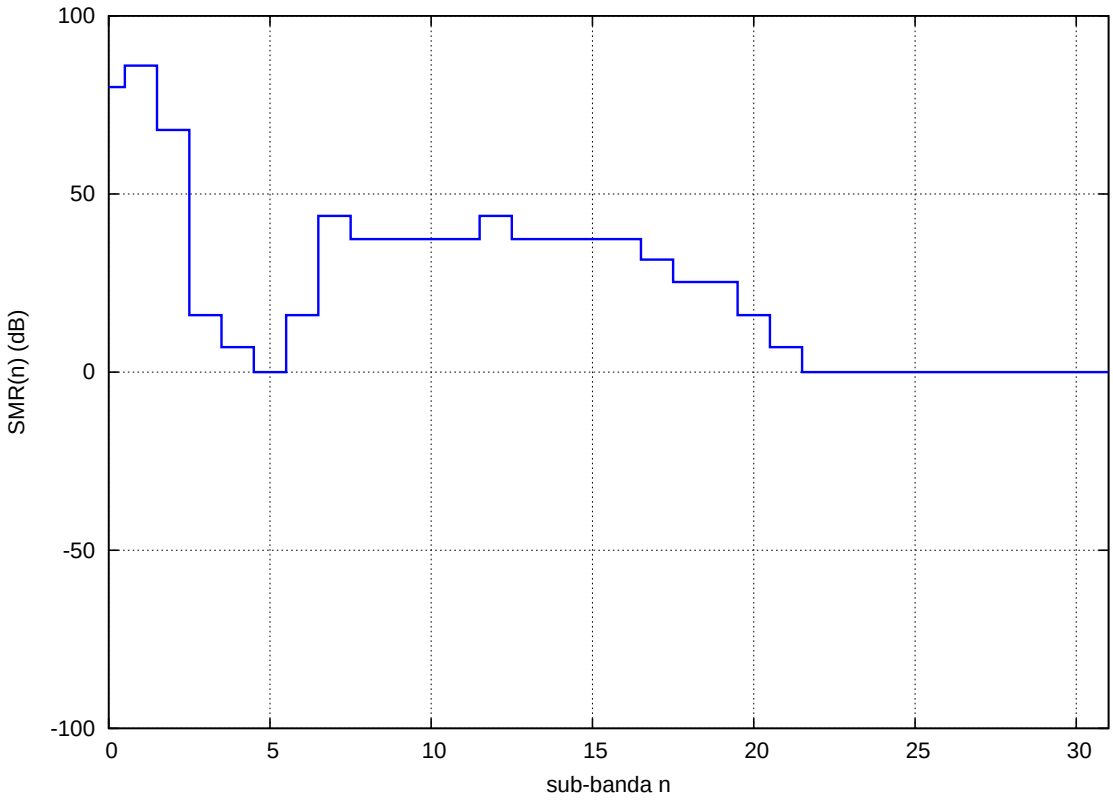

Figura 3.28: Razão sinal-ruído com *bitrate* de 448 kb/s.

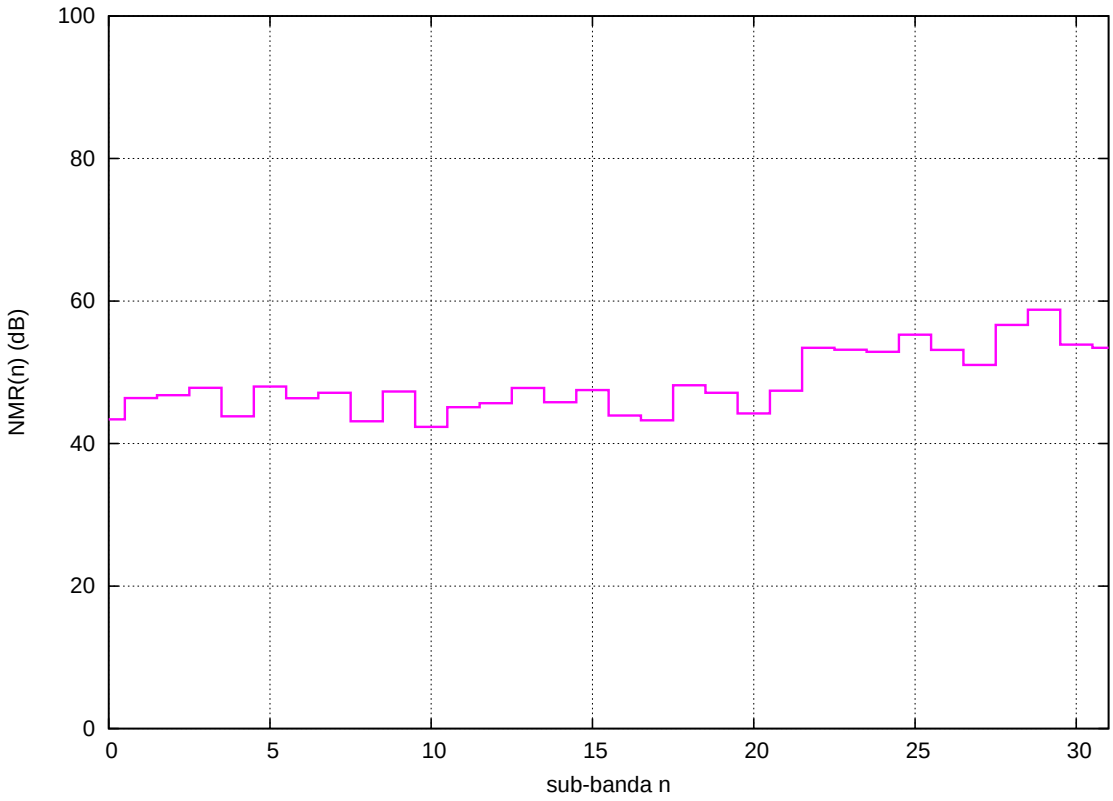

Figura 3.29: Razão máscara-ruído com *bitrate* de 384 kb/s.

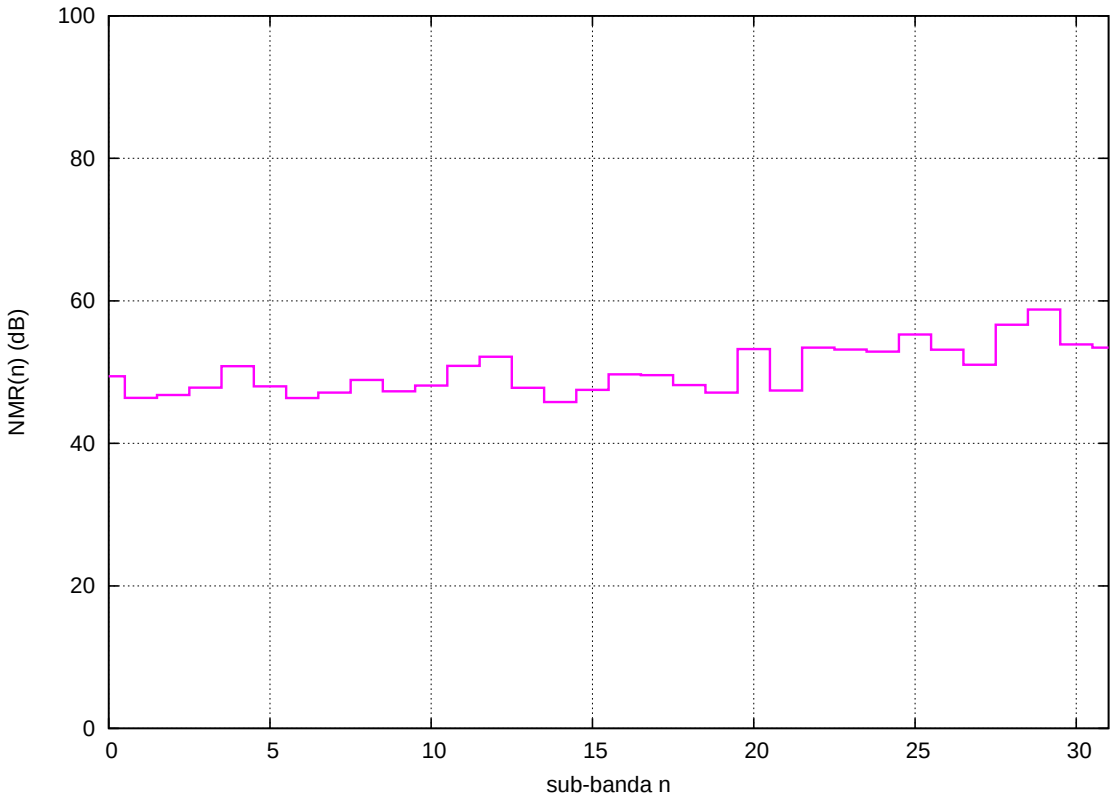

Figura 3.30: Razão máscara-ruído com *bitrate* de 416 kb/s.

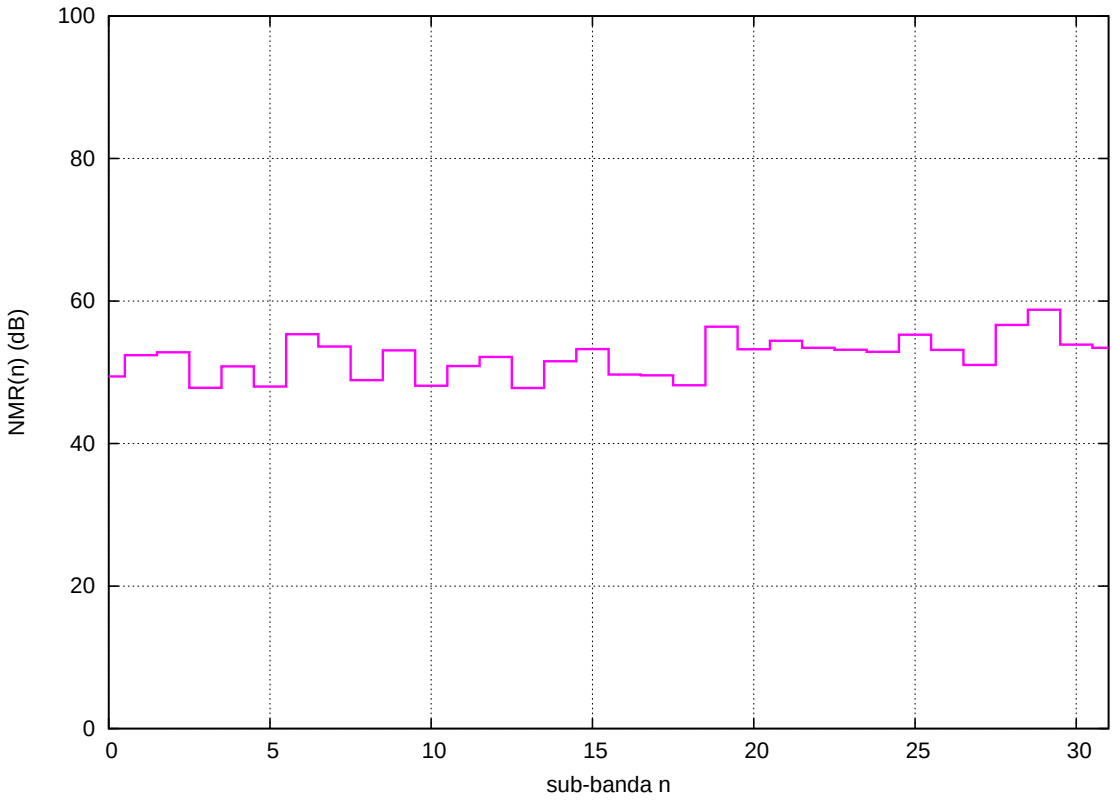

Figura 3.31: Razão máscara-ruído com *bitrate* de 448 kb/s.

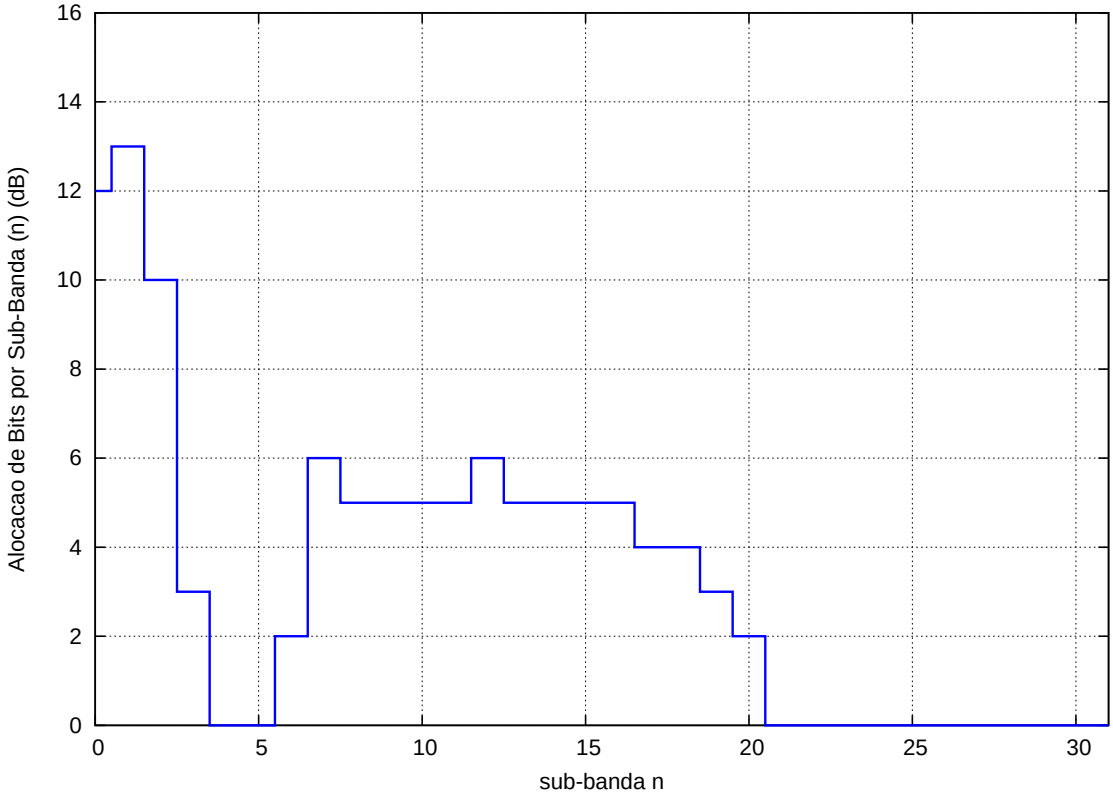

Figura 3.32: Bits alocados por sub-banda com *bitrate* de 384 kb/s.

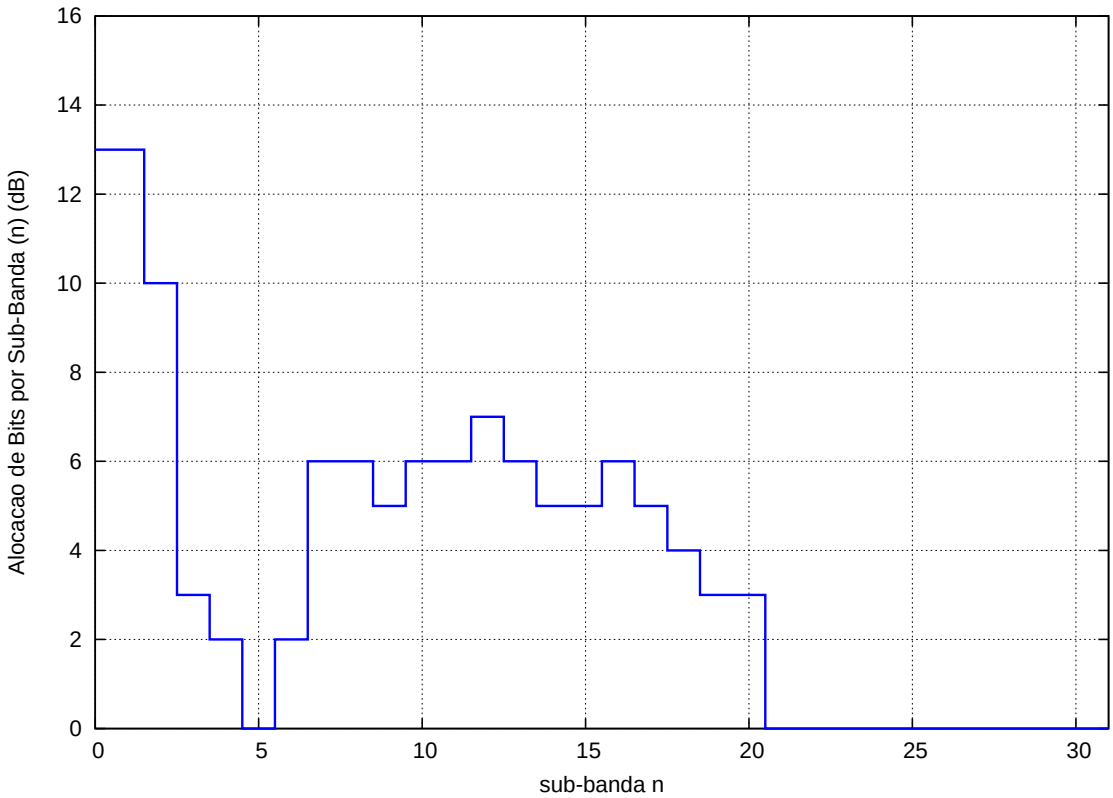

Figura 3.33: Bits alocados por sub-banda com *bitrate* de 416 kb/s.

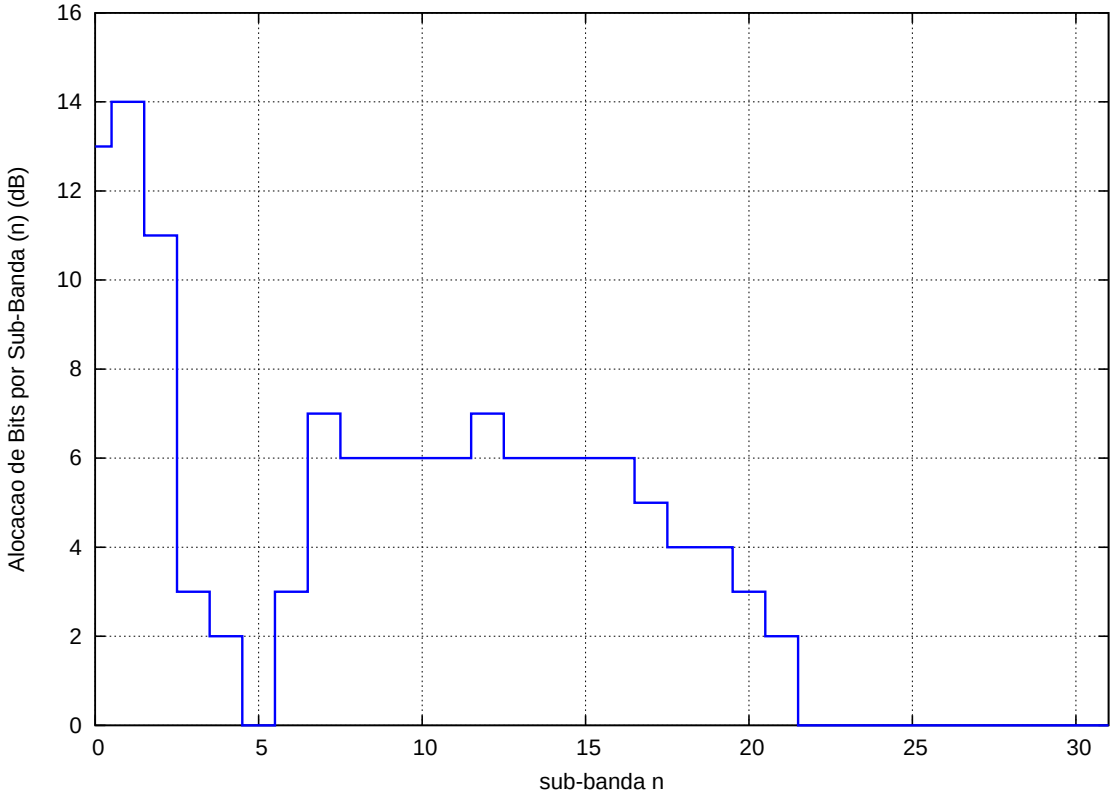

Figura 3.34: Bits alocados por sub-banda com *bitrate* de 448 kb/s.

de 384 kbits/s quanto para 448 kbits/s. Os arquivos, mesmo quando comparados com os originais, são indistingüíveis.

- Arquivo allegro – Para a taxa de alocação de 384 kbits/s demonstrou ruídos, onomatopaicamente traduzido por "brrrrr", nos trechos mais agudos, onde o ataque ao piano era mais rápido e forte. Para a taxa 448 kbits/s este efeito é menos perceptível, parece haver um espalhamento menor do som. Em ambos os arquivos esses problemas podem passar quase despercebidos pelos ouvintes medianos, se não comparados com os arquivos originais.

- Arquivo Berimbau – Este arquivo parece ter tido um ganho nos médios, talvez o que tenha acontecido na realidade foi um ganho menor que 1 nos graves, problema só percebido ao ser comparado com o arquivo original.

- Arquivo Divagacao – Apresentou borbulhos ao longo do arquivo, aparentemente o codificador não responde bem a batidas constantes no contra-tempo da bateria, problema que pode ter sido aumentado com o efeito de *chorus* de uma das guitarras-base.

- Arquivo Divagacao2 – Também apresentou borbulhos ao longo do arquivo; este efeito indesejado aparece no momento em que a bateria começa a ser executada.

- Arquivo voz_mine – Inseriu uma rouquidão, e voz ligeiramente metalizada, no final da frase gravada. No arquivo original há uma pequena saturação do áudio neste trecho da gravação.

- Arquivo op16 – Defeitos quase imperceptíveis, boa resposta ao ataque do piano, apesar de leve borbulho, notado sobretudo se comparado com o arquivo original.

- Arquivo Kamaitachi – Este arquivo apresentou borbulhos nos ataques dos ton-tons da bateria, com persistência dos borbulhos após o último ataque da bateria, onde há a adição de uma flauta bem aguda, que parece que prolongou o borbulho, mesmo após o decaimento do som da bateria. Ficou também a impressão de que as distorções utilizadas na guitarra e no contrabaixo elétrico intensificaram o problema.

Em todos os casos, o desempenho dos arquivos codificados com taxa de bits de 448 kbits/s foi melhor, em outras palavras, os efeitos indesejados são menos perceptíveis. Também foi notado que arquivos "bem-comportados", ou seja, arquivos de música mais suave, com pouca ou nenhuma percussão, são melhor codificados; a perda, ou diferença, da codificação é percebida somente se ele é comparado diretamente com o arquivo original.

Capítulo 4

Conclusão

O trabalho apresentado é representativo de dois grandes ramos da psicoacústica e trata-se de um passo inicial para trabalhos posteriores. É cumprida a proposta inicial de formar uma base de dados e de testes que podem servir tanto para fins didáticos, quanto como base de apoio a outros projetos correlacionados com a psicoacústica.

Ainda há melhorias e aperfeiçoamentos a serem feitos em trabalhos futuros. Com relação ao sistema de *scripts*, que reproduz alguns testes clássicos da perceção de *pitch*, apresentado na primeira parte do projeto, é sugerido que venha a ser escrito em alguma linguagem de programação, como C ou C++ (deixando assim a dependência do MatLab©, podendo o programa rodar em qualquer máquina), também utilizando uma interface gráfica amigável, integrando os diversos testes em um único sistema, acrescentando outros testes que foram excluídos, e, idealmente, que tenha o código portável para máquinas que rodem sistemas operacionais Windows© e Linux.

Já quanto ao codificador para MPEG-1 *Layer* I, mplf_ gpa1, o projeto já prevê essa portabilidade, já que foi escrito em C ANSI e foram utilizadas bibliotecas portáveis para ambos os sistemas. Além disso é possível integrar com certa facilidade a codificação para MPEG-1 *Layer* II, já que ambos utilizam o mesmo modelo psicoacústico.

Com relação aos problemas de ruídos e borbulhos apresentados em alguns arquivos codificados, tudo indica que seja um problema proveniente na forma de tratamento da região dos agudos, acima de 6/7 kHz, pelo modelo psicoacústico. Esta conclusão deve-se ao fato de problemas terem surgido principalmente quando o codificador se deparou com ataques rápidos de instrumentos percussivos. Estes ataques podem ser interpretados, ou aproximados, como um impulso, e portanto, possuem um espectro de freqüência largo. Além disso, também foi notado o problema em instrumentos que usam distorção eletrônica. Esta distorção, ou saturação, significa o aumento de harmônicos ímpares, atingindo freqüências suficientemente altas para prejudicar o desempenho do codificador. O mesmo efeito pôde ser observado no caso em que houve leve saturação da voz.

A solução para estes problemas pode ser encontrada dentro do modelo psicoacústico, já que o modelo apresentado é uma sugestão da recomendação [6], e portanto pode ser modificado, desde que seja respeitado o formato de escrita do *frame*. Deve-se salientar que o modelo sugerido pela recomendação [6] é bastante simples e que o próprio codificador/decodificador é razoavelmente simples quando comparado a outros codificadores/decodificadores, portanto é esperado que haja alguma perda percebida pelos ouvintes ao ser feita uma implementação sem o auxílio de outras referências e sugestões de modificações deste modelo.

Além disso também poderiam ter sido realizados testes com algoritmos de avaliação subjetiva como o da recomendaçao [8], porém não foram realizados pois o programa que tinhamos à disposição só avaliava arquivos com taxas de amostragem de 48 kHz e tivemos dificuldades em converter os arquivos de teste para esta taxa mantendo a qualidade original do som.

Finalmente os testes de comparação deveriam ter sido realizados com mais ouvintes e mais decodificadores. No caso dos testes com ouvintes não houve tempo hábil para a sua realização. Já quanto a outros decodificadores de arquivos *mp1* estes não foram encontrados pelo autor, sendo este um dos motivos para a realização deste projeto.

Referências Bibliográficas

[1] W. M. Hartmann. Pitch, Periodicity, and Auditory Organization, volume 100.

[2] Ray Meddis and Michael J. Hewitt. Virtual pitch and phase sensitivity of a computer model of the auditory periphery. I: Pitch identification, volume 89.

[3] Marina Bosi and Richard E. Goldberg. Introduction to Digital audio Coding and Standards. 2002.

[4] Peter Noll. MPEG Digital Audio Coding. 1997.

[5] Ulrich Reimers. Digital Video Broadcasting - The International Standard for Digital Television. 2001.

[6] ISO/IEC JTCI/SC29. Information of Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s - IS 11172 (Part 3, Audio). 1993.

[7] Ted Painter and Andreas Spanias. Perceptual Coding of Digital Audio, volume 88. 2000.

[8] ITU-R BS.1387. International Telecommunications Union, Radiocommunication Sector BS.1387 - Method for the Objective Measurements of Perceived Audio Quality. 1997.

Apêndice A

Código fonte dos scripts de teste da percepção de *pitch*

A.1 DL.m

```
% Script para testar a Differencial Limen.
% Este teste consiste em tocar um tom padrao e na sequencia tocar um outro tom
% com uma frequencia muito proxima, cabe ao ouvinte dizer se o segundo tom
% tem a frequencia maior, menor ou igual ao do tom de teste.

function varargout = DL(varargin)
% DL Application M-file for DL.fig
%    FIG = DL launch DL GUI.
%    DL('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Jul-2004 08:55:14

if nargin == 0  % LAUNCH GUI

fig = openfig(mfilename,'reuse');


% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if nargout > 0
varargout{1} = fig;
```

```
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
catch
disp(lasterr);
end

end

% --------------------------------------------------------------------
function varargout = begin_Callback(h, eventdata, handles, varargin)
% Botao de Inicio

if(size(get(handles.begin,'String'))==size('Inicio'))

    param=get(handles.text,'Value');
    [signal,indice]=DLimen(param(1), param(2), param(3), param(4)); % Este script
                    % monta o sinal de audio do teste
    wavplay(signal,44100,'sync'); % Toca o Sinal de teste
    pause(.1);
    set(handles.begin,'String','Repetir');
    set(handles.next,'Value',signal);
    set(handles.val,'Value',indice);

elseif(size(get(handles.begin,'String'))==size('Repetir'))
  param=get(handles.text,'Value');
  signal=get(handles.next,'Value');
  wavplay(signal,44100,'sync');
  pause(.1);
else
    text='Inicie o programa atraves do botao "Entrada de parametros"';
    DLGNAME='Iniciar';
```

```
    helpdlg(text,DLGNAME);
end

% --------------------------------------------------------------------
function varargout = next_Callback(h, eventdata, handles, varargin)
% Botao Proximo

if(size(get(handles.next,'String'))==size('Proximo Sinal'))

    indice=get(handles.val,'Value');
    n=get(handles.text,'Value');
    n(4)=n(4)+1;
    if(n(4)<indice+1)
        set(handles.text,'String',['Indice = ' num2str(n(4))])
        set(handles.text,'Value',n)
        set(handles.begin,'String','Inicio')
    else
        set(handles.begin,'String','');
        set(handles.next,'String','');
        set(handles.text,'String',['Este teste consiste em apos ouvir um sinal, '...
                    'identificar se o sinal seguinte e maior, menor, '...
                    'ou igual ao primeiro.']);
        set(handles.text,'Value',0);
    end
else
    text='Inicie o programa atraves do botao "Entrada de parametros"';
    DLGNAME='Iniciar';
    helpdlg(text,DLGNAME);
end

% --------------------------------------------------------------------
function varargout = val_Callback(h, eventdata, handles, varargin)
% Botao Entrada de Parametros
prompt  = {'Entre com a frequencia','Entre com o tempo do sinal',...
        'Entre com o tempo de silencio', 'Entre com o indice'};
    title   = 'Entrada de Parametros';
    lines= 1;
    def     = {'400','1','.1','1'};
    answer  = inputdlg(prompt,title,lines,def);
```

```
    w=str2num(answer{1}); % Frequencia do tom de padrao
    x=str2num(answer{2}); % Tempo de duraçao do sinal de teste
    y=str2num(answer{3}); % Tempo de intervalo entre o tom padrao e o tom de teste
    z=str2num(answer{4}); % Indice de inicio. O teste possui 10 frequencias de tom de teste
                          % pre ajustadas, comecar com um indice igual a 4, por exemplo,
                          % significa nao utilizar os 4 primeiros tons de teste
    set(handles.text,'Value',[w x y z]);
    set(handles.text,'String',['indice = ' num2str(z)])
    set(handles.begin,'String','Inicio');
    set(handles.next,'String','Proximo Sinal');

% --------------------------------------------------------------------
function varargout = closed_Callback(h, eventdata, handles, varargin)
close
```

A.2 MATCH.m

```
% Script para o usuario "encontrar" o pitch de um tom padrao, composto
% por uma senoide com seus n harmonicos seguido por um tom de teste
% de uma senoide pura, a qual o usuario tem a liberdade de ajustar
% a sua frequencia ate julgar que o pitch do tom de teste se igualou
% ao do tom padrao.

function varargout = match(varargin)
% MATCH Application M-file for match.fig
%    FIG = MATCH launch match GUI.
%    MATCH('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Jul-2004 09:18:10

if nargin == 0  % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if nargout > 0
```

```
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
catch
disp(lasterr);
end

end

% --------------------------------------------------------------------
function varargout = begin_Callback(h, eventdata, handles, varargin)
% Botao Inicio

flag=get(handles.text,'Value');
if(flag==0)
   msgbox('Entre com os parametros')

else
   signal1=flag(5:length(flag));
   flag=flag(1:4);
   set(handles.par,'Value',0);
   w=get(handles.freq_slider,'Value');

   % Monta o Sinal de teste de acordo com a frequencia ajustada pelo ouvinte
   signal2=senowav(w, flag(2));
   signal=[signal1 signal2];

   while(get(handles.par,'Value')==0 ) % enquanto o ouvinte nao parar
          if(str2num(get(handles.freq,'String'))==w)
              wavplay(signal,44100,'sync');
              pause(.2);
```

```
            else % Se o ouvinte mudar a frequencia do tom de teste
                 % o sinal e´ atualizado.
                w=(str2num(get(handles.freq,'String')));
                signal2=senowav(w, flag(2));
                signal=[signal1 signal2]; % Concatena o sina padrao com o sinal de teste
                wavplay(signal,44100,'sync'); % Toca o Sinal de teste
                pause(.2);
            end;
    end;
end

% --------------------------------------------------------------------
function varargout = par_Callback(h, eventdata, handles, varargin)
% Botao Entrada de Parametros

prompt  = {'Entre com a frequencia','Entre com o tempo do sinal',...
        'Entre com o tempo de silencio', 'Entre com o numero de harmonicos'};
    title   = 'Entrada de Parametros';
    lines= 1;
    def     = {'400','1','.1','10'};
    answer  = inputdlg(prompt,title,lines,def);
    w=str2num(answer{1}); % Frequencia do principal do tom de padrao
    x=str2num(answer{2}); % Tempo de duraçao do sinal de teste
    y=str2num(answer{3}); % Tempo de intervalo entre o tom padrao e o tom de teste
    z=str2num(answer{4}); % N° de harmonicos desejados no tom padrao
    set(handles.endless,'Value',[w x y z]);
    signal=senowav2(w,x,y,z); % Monta o sinal padrao
    f=w*.95; % normaliza a amplitude do sinal padrao
    set(handles.begin,'String','Inicio');
    set(handles.freq,'String',num2str(f));

    % Prepara o 'slide' para ficar 10% proximo do 1° harm. do tom padrao
    Xmax=w*1.1;
    Xmin=w*.9;
    slider_step(1)=0.05/(Xmax-Xmin);
    slider_step(2)=0.1/(Xmax-Xmin);
    set(handles.freq_slider,'sliderstep',slider_step,...
    'max',Xmax,'min',Xmin,'Value',f);
```

```
    set(handles.text,'Value',[w x y z signal]);
    set(handles.begin,'String','Inicio');
    set(handles.parar,'String','Parar');

% --------------------------------------------------------------------
function varargout = freq_slider_Callback(h, eventdata, handles, varargin)
% Slide

% Colhe a frequencia ajustada pelo ouvinte
f=get(handles.freq_slider,'Value');
set(handles.freq,'String',num2str(f));

% --------------------------------------------------------------------
function varargout = freq_Callback(h, eventdata, handles, varargin)
% Caixa de Texto

% Reajusta o slide para 10% da frequencia ajustada pelo ouvinte
x=str2num(get(handles.freq,'String'));
Xmax=x*1.1;
Xmin=x*.9;
slider_step(1)=0.05/(Xmax-Xmin);
slider_step(2)=0.1/(Xmax-Xmin);
set(handles.freq_slider,'Sliderstep',slider_step,...
      'Max',Xmax,'Min',Xmin,'Value',x);

% --------------------------------------------------------------------
function varargout = endless_Callback(h, eventdata, handles, varargin)
    set(handles.par,'Value',1)
% Botao Fim

clear all
clc
close all

% --------------------------------------------------------------------
function varargout = parar_Callback(h, eventdata, handles, varargin)
% Botao Parar

flag=get(handles.text,'Value');
```

```
if(flag==0)
   msgbox('Entre com os parametros')
else
   set(handles.par,'Value',1);
end
```

A.3 DISTUNED.m

```
% Script para mostrar que um sinal composto por uma senoide
% com n harmonicos, porem com um destes harmonicos "desafinados",
% indica ao ouvinte dois pitchs, um do tom composto e outro
% do harmonico "desafinado". Na realidade este harmonico chama a
% atençao do ouvinte para ele, ja que ele destoa do conjunto.
% O script tambem permite que o usuario tente encontrar o pitch
% do sinal padrao ajustando a frequencia de um sinal de teste composto
% por uma senoide pura.

function varargout = distuned(varargin)
% DISTUNED Application M-file for match.fig
%    FIG = DISTUNED launch match GUI.
%    DISTUNED('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Jul-2004 09:01:01

if nargin == 0  % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if nargout > 0
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
```

```
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
catch
disp(lasterr);
end

end

% --------------------------------------------------------------------
function varargout = begin_Callback(h, eventdata, handles, varargin)
% Botao Inicio

flag=get(handles.text,'Value');
if(flag==0)
   msgbox('Entre com os parametros')

else
   signal1=flag(3:length(flag));
   flag=flag(1:2);
   set(handles.par,'Value',0);
   w=get(handles.freq_slider,'Value');

   % Monta o Sinal de teste de acordo com a frequencia ajustada pelo ouvinte
   signal2=senowav(w, flag(2));
   signal=[signal1 signal2];

   while(get(handles.par,'Value')==0 )% enquanto o ouvinte nao parar
          if(str2num(get(handles.freq,'String'))==w)
              wavplay(signal,44100,'sync');
              pause(.2);
          else % Se o ouvinte mudar a frequencia do tom de teste
               % o sinal e´ atualizado.
              w=(str2num(get(handles.freq,'String')));
              signal2=senowav(w, flag(2));
              signal=[signal1 signal2]; % Concatena o sina padrao com o sinal de teste
              wavplay(signal,44100,'sync'); % Toca o Sinal de teste
```

```
            pause(.2);
        end;
    end;
end

% --------------------------------------------------------------------
function varargout = par_Callback(h, eventdata, handles, varargin)
% Botao Entrada de Parametros
prompt  = {'Entre com a frequencia','Entre com o tempo do sinal',...
        'Entre com o tempo de silencio', 'Entre com o numero de harmonicos',...
        'Entre com o harmônico que se deseja desafinar',...
        'Entre com a porcentagem de desafino'};
    title   = 'Entrada de Parametros';
    lines= 1;
    def     = {'200','1','.1','10','5','7'};
    answer  = inputdlg(prompt,title,lines,def);
    w=str2num(answer{1}); % Frequencia do principal do tom de padrao
    x=str2num(answer{2}); % Tempo de duraçao do sinal de teste
    y=str2num(answer{3}); % Tempo de intervalo entre o tom padrao e o tom de teste
    z=str2num(answer{4}); % N° de harmonicos desejados no tom padrao
    d=str2num(answer{5}); % Harmonico que se deseja "desafinar"
    p=str2num(answer{6}); % Percentual deste "desafino"
    set(handles.endless,'Value',[w x y z d p]);
    signal=Distuned1(w,x,y,z,d,p); % Monta o sinal padrao
    f=w*.95; % normaliza a amplitude do sinal padrao
    set(handles.begin,'String','Inicio');
    set(handles.freq,'String',num2str(f));

    % Prepara o 'slide' para ficar 10% proximo do 1° harm. do tom padrao
    Xmax=w*1.1;
    Xmin=w*.9;
    slider_step(1)=0.05/(Xmax-Xmin);
    slider_step(2)=0.1/(Xmax-Xmin);
    set(handles.freq_slider,'sliderstep',slider_step,...
        'max',Xmax,'min',Xmin,'Value',f);
    set(handles.text,'Value',[w x signal]);
    set(handles.begin,'String','Inicio');
    set(handles.parar,'String','Parar');
```

```
% --------------------------------------------------------------------
function varargout = freq_slider_Callback(h, eventdata, handles, varargin)
% Slide

% Colhe a frequencia ajustada pelo ouvinte
f=get(handles.freq_slider,'Value');
set(handles.freq,'String',num2str(f));

% --------------------------------------------------------------------
function varargout = freq_Callback(h, eventdata, handles, varargin)
% Caixa de Texto

% Reajusta o slide para 10% da frequencia ajustada pelo ouvinte
x=str2num(get(handles.freq,'String'));
Xmax=x*1.1;
Xmin=x*.9;
slider_step(1)=0.05/(Xmax-Xmin);
slider_step(2)=0.1/(Xmax-Xmin);
set(handles.freq_slider,'Sliderstep',slider_step,...
      'Max',Xmax,'Min',Xmin,'Value',x);

% --------------------------------------------------------------------
function varargout = endless_Callback(h, eventdata, handles, varargin)
    set(handles.par,'Value',1)
% Botao Fim

clear all
clc
close all

% --------------------------------------------------------------------
function varargout = parar_Callback(h, eventdata, handles, varargin)
% Botao Parar

flag=get(handles.text,'Value');
if(flag==0)
   msgbox('Entre com os parametros')
else
   set(handles.par,'Value',1);
```

```
end
```

A.4 SHEPARD.m

```
% Script para mostrar os efeitos do Tom de Shepard.
% Este tom responde a seguinte equaçao: somatorio de Fn=Fo*2.1^n.
% Ao dobrarmos todas as frequencias de um sinal como este
% curiosamente o pitch identificado pelos ouvintes decresce,
% como pode ser comprovado por este script.
% O ouvinte tem a liberdade de escolher a frequencia fundamental
% do tom padrao e o tempo de duraçao dos sinais padrao e de teste.

function varargout = shepard(varargin)
% SHEPARD Application M-file for match.fig
%    FIG = SHEPARD launch match GUI.
%    SHEPARD('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Jul-2004 09:26:58

if nargin == 0  % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if nargout > 0
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
```

```
catch
disp(lasterr);
end

end

% --------------------------------------------------------------------
function varargout = begin_Callback(h, eventdata, handles, varargin)
% Botao Inicio

flag=get(handles.text,'Value');
if(flag==0)
   msgbox('Entre com os parametros')

else
   signal=flag;
   while(get(handles.par,'Value')==0 )
               wavplay(signal,44100,'sync'); % Toca o Sinal de teste
               pause(.2);

   end;
end

% --------------------------------------------------------------------
function varargout = par_Callback(h, eventdata, handles, varargin)
% Botao Entrada de Parametros

prompt  = {'Entre com a frequencia','Entre com o tempo do sinal',...
        'Entre com o tempo de silencio', 'Entre com o numero de harmonicos'};
    title   = 'Entrada de Parametros';
    lines= 1;
    def     = {'50','1','.1','10'};
    answer  = inputdlg(prompt,title,lines,def);
    w=str2num(answer{1}); % Frequencia do principal do tom de padrao
    x=str2num(answer{2}); % Tempo de duraçao do sinal de teste
    y=str2num(answer{3}); % Tempo de intervalo entre o tom padrao e o tom de teste
    z=str2num(answer{4}); % N° de harmonicos desejados no tom padrao
    set(handles.endless,'Value',[w x y z]);
    signal=Shepard1(w,x,y,z); % Monta o sinal composto por dois tons de shepard.
```

```
    set(handles.begin,'String','Inicio');
    set(handles.text,'Value',[signal]);
    set(handles.begin,'String','Inicio');
    set(handles.parar,'String','Parar');

% --------------------------------------------------------------------
function varargout = endless_Callback(h, eventdata, handles, varargin)
    set(handles.par,'Value',1)
% Botao Fim

clear all
clc
close all

% --------------------------------------------------------------------
function varargout = parar_Callback(h, eventdata, handles, varargin)
% Botao Parar

flag=get(handles.text,'Value');
if(flag==0)
   msgbox('Entre com os parametros')
else
   set(handles.par,'Value',1);
end
```

Apêndice B

Código fonte dos scripts auxiliares do teste da percepção de *pitch*

B.1 DLimen.m

```
function[signal,na]=DLimen(fixedFreq, TimeSignal, TimeSilence,n)
TA=44100;
% Pega o inteiro mais proximo do tempo de duracao dos sinais pela Taxa de amostragem
t1=round(TimeSignal*TA);
% Pega o inteiro mais proximo do tempo entre os sinais pela Taxa de amostragem
t2=round(TimeSilence*TA);

% Gera os vetores de tempo austado para taxa de amostragem
timeSig=0:t1;
timeSil=0:t2;

% Multiplica a frequencia por 3%
DeltaOfFreq=fixedFreq*.03;

% Calcula o intervlo entre as frequencias de ajuste
DeltaOfDelta=DeltaOfFreq/10;

% Vetor "randomico"
x=[4,2,-2,5,-6,-4,3,-5,1,-3,6];
na=length(x);

% Gera vetor sinal com freq. fixa + tempo de silencio + sinal com freq "randomica"
signal=[sin(2*timeSig*fixedFreq*pi/TA) zeros(1,t2) sin(2*timeSig*(fixedFreq+(x(n)*DeltaOfDelta))*p
```

B.2 senowav.m

```
function[signal]=senowav(fixedFreq, TimeSignal)
TA=44100;
% Pega o inteiro mais proximo do tempo de duracao dos sinais pela Taxa de amostragem
t1=round(TimeSignal*TA);

% Gera o vetor de tempo e ajusta para taxa de amostragem
timeSig=0:t1;
A=2*pi*fixedFreq*timeSig/TA;

%Monta o sinal de audio
signal=sin(A);

% Normaliza a sua amplitude
signal=0.95*signal/(max(abs(signal)));
```

B.3 senowav2.m

```
function[signal]=senowav2(fixedFreq, TimeSignal, TimeSilence, harm)
TA=44100;
% Pega o inteiro mais proximo do tempo de duracao dos sinais pela Taxa de amostragem
t1=round(TimeSignal*TA);
% Pega o inteiro mais proximo do tempo entre os sinais pela Taxa de amostragem
t2=round(TimeSilence*TA);

% Gera o vetor de tempo e ajusta para taxa de amostragem
timeSig=0:t1;
A=2*pi*fixedFreq*timeSig/TA;

% Constroi o sinal com os harmonicos de 3 em diante
signal=[zeros(1,length(timeSig))];
for n=3:harm
    signal=signal+sin(n*A);
end

% Normaliza a amplitude do sinal para 1
signal=signal/(max(abs(signal)));
signal=[signal zeros(1,t2)];
```

B.4 Distuned1.m

```
function[signal]=Distuned(fixedFreq, TimeSignal, TimeSilence, harm, dist, perc)
TA=44100;
% Pega o inteiro mais proximo do tempo de duracao dos sinais pela Taxa de amostragem
t1=round(TimeSignal*TA);
% Pega o inteiro mais proximo do tempo entre os sinais pela Taxa de amostragem
t2=round(TimeSilence*TA);

% Gera o vetor de tempo e ajusta para taxa de amostragem
timeSig=0:t1;
A=2*pi*fixedFreq*timeSig/TA;

% Transforma o valor percentual de distorçao em fator multiplicativo equivalente
if(perc==0)
   p=1;
elseif(perc>0)
   p=(100+perc)/100;
else
   p=(100-perc)/100;
end

% Constroi o sinal com os harmonicos
signal=[zeros(1,length(timeSig))];
for n=1:harm
    if(n==dist)
        % multiplica a frequencia do harmonico n
        % pelo percentual de distorçao desejado
       signal=signal+sin(n*A*p);
    else
       signal=signal+sin(n*A);
    end
end

% Normaliza a amplitude do sinal para 1
signal=signal/(max(abs(signal)));
signal=[signal zeros(1,t2)];
```

B.5 Shepard1.m

```
function[signal]=Shepard(fixedFreq, TimeSignal, TimeSilence, harm)
TA=44100;
% Pega o inteiro mais proximo do tempo de duracao dos sinais pela Taxa de amostragem
t1=round(TimeSignal*TA);
% Pega o inteiro mais proximo do tempo entre os sinais pela Taxa de amostragem
t2=round(TimeSilence*TA);

% Gera o vetor de tempo e ajusta para taxa de amostragem
timeSig=0:t1;
A=2*pi*fixedFreq*timeSig/TA;

% Constroi o sinal com os harmonicos
signal1=[zeros(1,length(timeSig))];
signal2=[zeros(1,length(timeSig))];

for n=0:(harm-1)
    signal1=signal1+sin((2.1^n)*A);
    signal2=signal2+sin(2*(2.1^n)*A);
end

% Normaliza a amplitude do sinal para 0.95
signal=.95*signal1/(max(abs(signal1)));
signal2=.95*signal2/(max(abs(signal2)));
signal=[signal1 zeros(1,t2) signal2];
```

Apêndice C

Códigos fonte do programa MPEG-1 - *Layer* I

C.1 MPEG1 encoder

```
//mplf_gpa1.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "./lib/fftw-3.0.1/api/fftw3.h"
#include "./lib/libsndfile-1.0.4/src/sndfile.h"
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "matb.h"
#include "common.h"
#include "mpeg_fun.h"


int main(int argc, char *argv[]){

  double SNR[]={ 0.0, 7.0, 7.0, 16.0, 25.28, 31.59, 37.35, 43.84,
   49.89, 55.93, 61.96, 67.98, 74.01, 80.03, 86.05, 92.01};
  double A[]={0.0, 0.0, 0.75, 0.875, 0.9375, 0.96875, 0.984375,
   0.9921875, 0.99609375, 0.998046875, 0.999023438, 0.999511719,
   0.999755859, 0.99987793, 0.999938965, 0.999969482};
  double B[]={0.0, 0.0, -0.25, -0.125, -0.0625, -0.03125, -0.015625,
```

```
-0.0078125, -0.00390625, -0.001953125, -0.000976563, -0.000488281,
-0.000244141, -0.00012207, -0.000061035, -0.000030518};

FILE *fpt;
SNDFILE *infile;
SF_INFO sfinfo;
unsigned short int sample[2][12][N_SUBBAND];
unsigned short int *buffer_sample;
int temp_sample;
int count_byte, count_buf, count_slot, limit_stream;
double *double_buffer;
int readcount;
char *outfilename;
int  i, j, k, n, nb, nby, ch, sb, s;
long int OFFSET;
float rest, dif;
char padding, sampling_frequency;
double S[2][12][N_SUBBAND];
double SQ[2][12][N_SUBBAND];
double *scf[2];
short int *scf_index[2];
double *y;
double *X[2];
double *Lsb[2];
double *SMRsb[2];
double *MNRsb[2];
unsigned short int step, *allocation[2];
table tables;
tonal tonals[2];
lt lts[2];
double xAux[2][FFT_SIZE];
int remaining_bits, ind, *inds, mode;
double slask;


sfinfo.format = 0 ;

/* Verifica se o usuário entrou com a linha de comando correta */
if(argc != 2){
```

```
  printf("Usage: %s <coded_file_in>\n", argv[0]);
  exit(1);
}

/* Abre o arquivo wav e verifica se há erro de leitura */
if (! (infile = sf_open (argv[1], SFM_READ, &sfinfo))){
  puts (sf_strerror (NULL)) ;
  exit(1);
}
  sf_close(infile);

/* Recolhe informações do arquivo wav e imprime na tela */
if((infile = sf_open (argv[1], SFM_READ, &sfinfo))){
  printf ("Sample Rate : %d\n", sfinfo.samplerate) ;
  printf ("Frames      : %ld\n", (long) sfinfo.frames) ;
  printf ("Channels    : %d\n", sfinfo.channels) ;
  printf ("Format      : 0x%08X\n", sfinfo.format) ;
  printf ("Sections    : %d\n", sfinfo.sections) ;
  printf ("Seekable    : %s\n\n", (sfinfo.seekable ? "TRUE" : "FALSE")) ;
  puts(sf_strerror(infile));
}

/* Inicialização do arquivo codificado  */
memset(outfilename, 0, strlen(argv[1])*sizeof(char));
strncpy(outfilename, argv[1], strlen(argv[1])-4);
strcat(outfilename, ".mp1");


/* Prepara informações para o buffer de escrita e para o padding_bit */
ch=0;
rest=0.0;
padding=no;
//IF(LAY==I)
if(sfinfo.samplerate==44100){
  dif=(12000*BITRATE)%sfinfo.samplerate; //
  nb=(floor(BITRATE*12000*(N_SUBBAND)/sfinfo.samplerate)); // n. de bites por frame
  nby=(floor(BITRATE*12000/sfinfo.samplerate))*32/8; // n. de bytes por frame
  sampling_frequency=frequency441;
}
```

```
else if(sfinfo.samplerate==48000){
  dif=(12000*BITRATE)%sfinfo.samplerate; //
  nb=(floor(BITRATE*12000*(N_SUBBAND)/sfinfo.samplerate)); // n. de bites por frame
  nby=(floor(BITRATE*12000/sfinfo.samplerate))*32/8; // n. de bytes por frame
  sampling_frequency=frequency48;
}


if(sfinfo.channels==2)
  mode=0x00;
else
  mode=0xC0;

/* Inicialização do buffer de escrita */
double_buffer=malloc(sfinfo.channels*BUFFER_LEN*sizeof(double));

/* Seta a forma de leitura do arquivo wav */
j=sf_command (infile, SFC_SET_NORM_DOUBLE, NULL, SF_TRUE) ;


/* aloca e inicializa algumas variáveis */
for(ch=0; ch<sfinfo.channels; ch++){
  allocation[ch]=malloc(N_SUBBAND*sizeof(short int));
  memset(allocation[ch], 0, sizeof(int));
  lts[ch].LTg=malloc(255*sizeof(double));
  lts[ch].LTmin=malloc((N_SUBBAND+1)*sizeof(double));
  tonals[ch].Flags=malloc((FFT_SIZE/2)*sizeof(int));
  Lsb[ch]=malloc((N_SUBBAND+1)*sizeof(double));
  X[ch]=malloc(512*sizeof(double));
  SMRsb[ch]=malloc(N_SUBBAND*sizeof(double));
  MNRsb[ch]=malloc(N_SUBBAND*sizeof(double));
  scf[ch]=malloc((N_SUBBAND+1)*sizeof(double));
  scf_index[ch]=malloc(N_SUBBAND*sizeof(short int));
  memset(xAux[ch], 0, BUFFER_INPUT*sizeof(double));
  memset(lts[ch].LTmin, 0, (N_SUBBAND+1)*sizeof(double));
  memset(scf[ch], 0, (N_SUBBAND+1)*sizeof(double));
  memset(tonals[ch].Flags,0,(FFT_SIZE/2)*sizeof(int));
}
```

```
  buffer_sample=malloc((nby+1)*sizeof(unsigned short int));
  tables.Map=malloc(255*sizeof(int));
  tables.LTq=malloc(255*sizeof(double));
  tables.CB=malloc(26*sizeof(int));
  y=malloc(N_SUBBAND*sizeof(double));
  tables.C=malloc(513*sizeof(double));
  inds=malloc(N_SUBBAND*sfinfo.channels*sizeof(double));


  /* Carrega as tabelas e testa-as */
  tables=Table_absolute_threshold(1, sfinfo.samplerate, BITRATE, tables);
  if(tables.erro==1) exit(0);
  Table_critical_band_boundaries(1, sfinfo.samplerate, tables);
  if(tables.erro==1) exit(1);
  Table_analysis_window(tables);


  memset(y ,0, N_SUBBAND*sizeof(double));

  /* Abre o arquivo de escrita */
  fpt=fopen(outfilename, "wb");
  fclose(fpt);

  /* Carrega os coeficientes do Banco de Filtros */
  for(i=0;i<32;i++){
    for(k=0;k<64; k++){
      m[i][k]= cos((2 * i + 1) * (k - 16) * pi / 64);
    }
  }

  OFFSET=1;
  k=n=0;

  /* Lê as primeiras 448 amostras do arquivo wav */
  readcount = sf_readf_double (infile, double_buffer, BUFFER_LEN);
  do{

/****************** Banco de Filtros e modelo psicoacústico ************************/
    for(ch=0; ch<sfinfo.channels; ch++){
```

```
j=0;

/* Análise do Banco de Filtros */
for(s=0;s<12;s++){
  /* Prepara o vetor para o Banco de Filtros */
  for(i=511; i>31; i--){
    xAux[ch][i]=xAux[ch][i-32];
  }
  for(i=31; i>-1; i--){
    if(sfinfo.channels>1)
      xAux[ch][31-i]=double_buffer[(2*i+ch)+64*(s)];
    else
      xAux[ch][31-i]=double_buffer[(i)+32*(s)];
  }

  /* Faz a filtragem  [1, pp. 67]*/
  y=Analysis_subband_filter(xAux[ch], N_SUBBAND *
                    (s+1), tables.C, FFT_SIZE, y);

  /* Carrega as novas 32 amostras por sub-banda */
  for(sb=0;sb<32;sb++){
    S[ch][s][sb]=y[sb];
    j++;
  }
}

/* Cálculo dos fatores de escala [1, pp. 70]*/
scf[ch]=Scale_factors(S[ch], scf[ch], scf_index[ch]);

/* Calcula a FFT para a conversão de tempo em freqüência  [1, pp. 110].*/
X[ch]=FFT_Analysis(xAux[ch], 0, X[ch]);

/* Determina o nível de pressão sonora em cada sub-banda [1, pp. 110].*/
Lsb[ch]=Sound_pressure_level(X[ch], scf[ch], Lsb[ch], 512, scf[ch][0]);

/* Encontra os componentes tonais e não-tonais do sinal [1, pp. 111--113]*/
tonals[ch]=Find_tonal_components(X[ch], tables, 512, tonals[ch]);

/* Decima os mascaradores, elimina todos os mascaradores não relevantes
```

```
      [1, pp. 113]*/
    tonals[ch]=Decimation(tables, tonals[ch]);

    /* Calcula os limiares de mascaramento individuais [1, pp. 113--114]*/
    lts[ch]=Individual_masking_thresholds(X[ch],tonals[ch], tables, lts[ch]);

    /* Calcula o limiar de mascaramento global [1, pp. 114]*/
    lts[ch]=Global_masking_threshold(tables, lts[ch]);

    /* Determina o limiar de mascaramento mínimo para cada sub-banda
      [1, pp. 114]*/
    lts[ch]=Minimum_masking_threshold(lts[ch], tables);

    /* Calcula o SMR (Signal-to-Mask-Ratio) [1, pp. 115]*/
    for(sb=0; sb<N_SUBBAND; sb++){
      SMRsb[ch][sb]=Lsb[ch][sb]-lts[0].LTmin[sb+1];
    }

    memset(allocation[ch], 0, N_SUBBAND*sizeof(short int));

    /* Determinação do NMR (Noise-to-Mask-Ratio) inicial */
    for(sb=0; sb<N_SUBBAND; sb++){
      MNRsb[ch][sb]=SNR[allocation[ch][sb]]-SMRsb[ch][sb];
    }

    /* Inicializa o número de bits que restam para o frame */
    remaining_bits=(nb-16-4*N_SUBBAND*2)/sfinfo.channels;

    if(padding==no)
      remaining_bits-=16;
/*****************Inicio da quantização [1, pp. 70--71]*******************/

    /* Enquanto houver mais de 12 bits para serem alocados */
    while(remaining_bits>=12){
      /* Busca a sub-banda com o menor MNR */
      slask=Min_ind(MNRsb[ch], N_SUBBAND, &ind);

      /* Se já foi alocado mais de 15 bits nesta sub-banda */
      if(allocation[ch][ind]==15){
```

```
  MNRsb[ch][ind]=200.0;
  /* Busca a sub-banda com o próximo MNR menor */
  slask=find_minor_value(MNRsb[ch], N_SUBBAND, allocation[ch], 15, &ind);
}

/* Se é a primeira alocação da sub-banda e restam menos
  de 30 bits para serem alocados */
if (allocation[ch][ind]==0 && remaining_bits<30){
  /* Busca a sub-banda com o próximo MNR menor e que
    já tenha alocado mais de 2 bits */
  slask=find_value_between(MNRsb[ch], N_SUBBAND, allocation[ch], 0, 15, &ind);
}

/* Se não há bits alocados para esta sub-banda */
if(allocation[ch][ind]==0){
  /* Se ainda há mais de 30 bits para serem alocados */
  if(remaining_bits>=30){
    /* Aloca 2 bits para cada amostra da sub-banda */
    allocation[ch][ind]=2;
    /* Bits restantes é igual ao anterior menos 6 bits do Scale Factor
      e menos 2 bits vezes 12 amostras da sub-banda */
    remaining_bits=remaining_bits-2*12-6;
  }
}
/* Se já há bits alocados nesta sub-banda */
else{
  /* Se ainda há mais 12 bits para serem alocados */
  if(remaining_bits>=12){
    /* Se ainda não foram alocados 15 bits nesta sub-banda */
    if(allocation[ch][ind]<15)
      /* Aloca mais 1 bit por amostra da sub-banda */
      allocation[ch][ind]++;
    }
    /* Bits restantes é igual ao anterior menos 1 bit vezes
      12 amostras da sub-banda */
    remaining_bits=remaining_bits-12;
  }
}
```

```
        /* Atualiza MNR da sub-banda */
        MNRsb[ch][ind]=SNR[allocation[ch][ind]]-SMRsb[ch][ind];
      }/* Fim dos bits restantes */
    }// end for ch


      /* Quantiza cada amostra com o número de bits calculados
        na etapa anterior [1, pp. 71]*/
      for(sb=0; sb<N_SUBBAND; sb++){
        for(ch=0; ch<sfinfo.channels; ch++){
          if(allocation[ch][sb]!=0){
            step = 0xffff << (16-allocation[ch][sb]);

            for(s=0; s<12; s++){
              SQ[ch][s][sb]=A[allocation[ch][sb]]*
                        S[ch][s][sb]/scf[ch][sb+1]+B[allocation[ch][sb]];
              sample[ch][s][sb]= (unsigned short int)
                (floor(((double)SQ[ch][s][sb]+1)/(2-pow(2, -(allocation[ch][sb]-2)))
                * (double)(pow(2, 8*sizeof(short int))-1)));
              sample[ch][s][sb]=sample[ch][s][sb]&step;
            }
          }
        }
      }
/*****************  Fim da quantização *********************/


/*****************  Início do bitstream *********************/
      /* Escreve o novo frame do arquivo codificado */
      if( (fpt=fopen(outfilename, "ab"))==NULL){
        printf("erro de abertura de arquivo\n");
        exit(1);
      }

      //Header() [1, pp. 14]
      buffer_sample[0]=0xFFFF;
      fwrite(&buffer_sample[0], sizeof(unsigned short int), 1,fpt);
      fflush(fpt);
      buffer_sample[0]=SWAP(((bitrate_index|sampling_frequency|padding)<<8)^(mode)^0x04);
```

```
fwrite(&buffer_sample[0], sizeof(unsigned short int), 1,fpt);
fflush(fpt);

count_byte=0;
count_buf=0;
count_slot=0;
memset(buffer_sample, 0, (nby+4)*sizeof(unsigned short int));

//audio_data() [1, pp. 15]
/* Escreve o número de bits alocados por sub-banda */
for(sb=0; sb<N_SUBBAND; sb++){
  for(ch=0; ch<sfinfo.channels; ch++){

    if(allocation[ch][sb]!=0)
      temp_sample=( (allocation[ch][sb]-1)<<0xC);
    else
      temp_sample=0;
    buffer_sample[count_buf]=buffer_sample[count_buf]|
                    (temp_sample>>count_byte);
    count_byte+=0x04;
    if(count_byte>15){
      buffer_sample[count_buf]=SWAP(buffer_sample[count_buf]);
      fwrite(&buffer_sample[count_buf], sizeof(unsigned short int), 1,fpt);
      fflush(fpt);
      count_buf++;
      count_byte=0;
    }
  }
}

/* Escreve os Scale Factor por sub-banda */
for(sb=0; sb<N_SUBBAND; sb++){
  for(ch=0; ch<sfinfo.channels; ch++){
    if(allocation[ch][sb]>1){
    temp_sample=(scf_index[ch][sb]<<10);

    buffer_sample[count_buf]=buffer_sample[count_buf]|
                    (temp_sample>>count_byte);
    count_byte+=0x06;
```

```
    if(count_byte>15){
      buffer_sample[count_buf]=SWAP(buffer_sample[count_buf]);
      fwrite(&buffer_sample[count_buf], sizeof(unsigned short int), 1,fpt);
      fflush(fpt);
        count_buf++;
        temp_sample=scf_index[ch][sb]<<(16-count_byte);
        buffer_sample[count_buf]=temp_sample>>16;
        count_byte=count_byte-16;
      }
    }
  }
}

/* Escreve as amostras quantizadas */
for(s=0; s<12; s++){
  for(sb=0; sb<N_SUBBAND; sb++){
    for(ch=0; ch<sfinfo.channels; ch++){
      if(allocation[ch][sb]>1){
        temp_sample=sample[ch][s][sb];
        buffer_sample[count_buf]=buffer_sample[count_buf]|
                        (temp_sample>>(count_byte));
        count_byte+=allocation[ch][sb];
        if(count_byte>15){
          buffer_sample[count_buf]=SWAP(buffer_sample[count_buf]);
          fwrite(&buffer_sample[count_buf],
               sizeof(unsigned short int), 1,fpt);
          fflush(fpt);
          count_buf++;
          buffer_sample[count_buf]=sample[ch][s][sb]<<
                          (15-(count_byte-16));
          count_byte=count_byte-16;
        }
      }
    }
  }
}

/* Calcula o espaço restante do frame de acordo com o padding_bit
  [1, pp. 22]*/
```

```
      if(padding==no)
        limit_stream=2*floor(12000*BITRATE/sfinfo.samplerate);
      else
        limit_stream=2*floor(12000*BITRATE/sfinfo.samplerate)+2;
      /* Preenche o espaço restante do Frame com zero */
      for(i=count_buf; i<limit_stream-2; i++){
        buffer_sample[i]=0;
        fwrite(&buffer_sample[i], sizeof(unsigned short int), 1,fpt);
        fflush(fpt);
      }

      /* Calcula o padding_bit, útil para taxa de amostragem = 44,1 kHz */
      rest=rest-dif;
      if(rest<0){
        padding=yes;
        rest+=sfinfo.samplerate/1000;
      }
      else padding=no;


      fclose(fpt);
/******************  Fim do bitstream  *********************/


    /* Incrementa o número de amostras já lidas */
    OFFSET=OFFSET+384;

    /* Imprime a porcentagem da codificação concluída */
    printf("********************************************** %3.2f%% concluído\r",
    ((float)OFFSET/(float)sfinfo.frames)*100);

  }while ((readcount = sf_readf_double (infile, double_buffer, BUFFER_LEN)));
  /* Fica em loop enquanto não acabar a leitura do arquivo wav */

  printf("********************************************** 100%% concluído    \r\n");

  /* Desaloca o espaço da memória */
  sf_close(infile);
  free(double_buffer);
```

```
  free(y);
  free(tables.LTq);
  free(tables.CB);
  free(tables.C);
  free(inds);


  for(ch=0; ch<sfinfo.channels; ch++){
    free(lts[ch].LTg);
    free(tonals[ch].Flags);
    free(lts[ch].LTmin);
    free(Lsb[ch]);
    free(X[ch]);
    free(SMRsb[ch]);
    free(MNRsb[ch]);
    free(scf[ch]);
    free(scf_index[ch]);
  }
  exit(0);
}
```

C.2 Constantes Comuns

```
//common.h

#define pi 3.141592653589793

#define FFT_SHIFT    384
#define FFT_SIZE     512
#define FFT_OVERLAP  (FFT_SIZE - FFT_SHIFT) / 2

#define N_SUBBAND  32
#define BUFFER_LEN 384
#define BUFFER_INPUT BUFFER_LEN+64
#define BITRATE  448
#define bitrate_index 0xE0
#define frequency441  0x00
#define frequency48  0x04
```

```
#ifndef FALSE
#define         FALSE                   0
#endif

#ifndef TRUE
#define         TRUE                    1
#endif

#define  yes  0x02
#define  no  0x00


#include "./lib/fftw-3.0.1/api/fftw3.h"

// Flags for tonal analysis
#define NOT_EXAMINED  0
#define TONAL         1
#define NON_TONAL     2
#define IRRELEVANT    3

#define MIN_POWER  -296

// Indices used in tables like the threshold table
// or in the list of tonal and non-tonal components.
#define INDEX  0
#define BARK   1
#define SPL    1
#define ATH    2


float m[32][64];
```

C.3 Declarações das Funções Principais

```
//mpeg_fun.h
```

```
typedef struct Threshold {double TH[107][3]; int *Map; double *LTq;
          int *CB; float *C; int erro;} table;

typedef struct Tonal {int *Flags; double Tonal_list[257][2];
        double Non_tonal_list[26][2];} tonal;

typedef struct LTs {double LTt[100][106]; double LTn[100][106];
          double *LTg; double *LTmin;} lt;


table Table_absolute_threshold(int Layer, int fs, int bitrate, table);

void Table_critical_band_boundaries(int Layer, int fs, table);

void Table_analysis_window(table tables);

double * Analysis_subband_filter(double *Input, int n, float *C, int len, double *s);

double * Scale_factors(double S[12][32], double *scf, short int *scf_index);

double * FFT_Analysis(double *Input, int n, double *X);

double * Sound_pressure_level(double *X, double *scf, double *Lsb, int lenX, int lenScf);

tonal Find_tonal_components(double *X, table tables, int lenX, tonal tonals);

tonal Decimation(table tables, tonal tonals);

int SWAP(unsigned short int);

lt Individual_masking_thresholds(double *, tonal, table, lt);

lt Global_masking_threshold(table tables, lt lts);

lt Minimum_masking_threshold(lt lts, table tables);
```

C.4 Declarações das Funções Auxiliares

```
// matlab.h

void Abs(double *, int);
double Max(double *, int);
double MaxAbs(double *, int);
double Min(double *, int);
double MinAbs(double *, int);
double Mean(double *, int);
int Round(double);
double Min_ind(double *, int, int *);
double find_minor_value(double *, int, unsigned short int *, int, int *);
double find_value_between(double *, int , unsigned short int *, int , int , int *);
```

C.5 Table Absolute Threshold

```
//Table_absolute_threshold(Layer, fs, bitrate, *TH, *Map, *LTq)



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

table Table_absolute_threshold(int Layer, int fs, int bitrate, table tables){

  int N, i,j;
  double k[3];

  if(Layer ==1){
    if(fs == 44100){
    /*  Frequencia | Taxa da Banda Crítica | Limiar Absoluto */
      double  TH_1[][3] ={
      {  106.0 , 106.0,   106.0},
      {   86.13,   0.850,  25.87}, {   172.27,   1.694,  14.85},
```

```
{  258.40,   2.525,  10.72}, {   344.53,   3.337,   8.50},
{  430.66,   4.124,   7.10}, {   516.80,   4.882,   6.11},
{  602.93,   5.608,   5.37}, {   689.06,   6.301,   4.79},
{  775.20,   6.959,   4.32}, {   861.33,   7.581,   3.92},
{  947.46,   8.169,   3.57}, {  1033.59,   8.723,   3.25},
{ 1119.73,   9.244,   2.95} ,{  1205.86,   9.734,   2.67},
{ 1291.99,  10.195,   2.39} ,{  1378.13,  10.629,   2.11},
{ 1464.26,  11.037,   1.83} ,{  1550.39,  11.421,   1.53},
{ 1636.52,  11.783,   1.23} ,{  1722.66,  12.125,   0.90},
{ 1808.79,  12.448,   0.56} ,{  1894.92,  12.753,   0.21},
{ 1981.05,  13.042,  -0.17} ,{  2067.19,  13.317,  -0.56},
{ 2153.32,  13.578,  -0.96} ,{  2239.45,  13.826,  -1.38},
{ 2325.59,  14.062,  -1.79} ,{  2411.72,  14.288,  -2.21},
{ 2497.85,  14.504,  -2.63} ,{  2583.98,  14.711,  -3.03},
{ 2670.12,  14.909,  -3.41} ,{  2756.25,  15.100,  -3.77},
{ 2842.38,  15.284,  -4.09} ,{  2928.52,  15.460,  -4.37},
{ 3014.65,  15.631,  -4.60} ,{  3100.78,  15.796,  -4.78},
{ 3186.91,  15.955,  -4.91} ,{  3273.05,  16.110,  -4.97},
{ 3359.18,  16.260,  -4.98} ,{  3445.31,  16.406,  -4.92},
{ 3531.45,  16.547,  -4.81} ,{  3617.58,  16.685,  -4.65},
{ 3703.71,  16.820,  -4.43} ,{  3789.84,  16.951,  -4.17},
{ 3875.98,  17.079,  -3.87} ,{  3962.11,  17.205,  -3.54},
{ 4048.24,  17.327,  -3.19} ,{  4134.38,  17.447,  -2.82},
{ 4306.64,  17.680,  -2.06} ,{  4478.91,  17.905,  -1.32},
{ 4651.17,  18.121,  -0.64} ,{  4823.44,  18.331,  -0.04},
{ 4995.70,  18.534,   0.47} ,{  5167.97,  18.731,   0.89},
{ 5340.23,  18.922,   1.23} ,{  5512.50,  19.108,   1.51},
{ 5684.77,  19.289,   1.74} ,{  5857.03,  19.464,   1.93},
{ 6029.30,  19.635,   2.11} ,{  6201.56,  19.801,   2.28},
{ 6373.83,  19.963,   2.46} ,{  6546.09,  20.120,   2.63},
{ 6718.36,  20.273,   2.82} ,{  6890.63,  20.421,   3.03},
{ 7062.89,  20.565,   3.25} ,{  7235.16,  20.705,   3.49},
{ 7407.42,  20.840,   3.74} ,{  7579.69,  20.972,   4.02},
{ 7751.95,  21.099,   4.32} ,{  7924.22,  21.222,   4.64},
{ 8096.48,  21.342,   4.98} ,{  8268.75,  21.457,   5.35},
{ 8613.28,  21.677,   6.15} ,{  8957.81,  21.882,   7.07},
{ 9302.34,  22.074,   8.10} ,{  9646.88,  22.253,   9.25},
{ 9991.41,  22.420,  10.54} ,{ 10335.94,  22.576,  11.97},
{10680.47,  22.721,  13.56} ,{ 11025.00,  22.857,  15.31},
```

```
{11369.53,  22.984,  17.23} ,{ 11714.06,  23.102,  19.34},
{12058.59,  23.213,  21.64} ,{ 12403.13,  23.317,  24.15},
{12747.66,  23.415,  26.88} ,{ 13092.19,  23.506,  29.84},
{13436.72,  23.592,  33.05} ,{ 13781.25,  23.673,  36.52},
{14125.78,  23.749,  40.25} ,{ 14470.31,  23.821,  44.27},
{14814.84,  23.888,  48.59} ,{ 15159.38,  23.952,  53.22},
{15503.91,  24.013,  58.18} ,{ 15848.44,  24.070,  63.49},
{16192.97,  24.125,  68.00} ,{ 16537.50,  24.176,  68.00},
{16882.03,  24.225,  68.00} ,{ 17226.56,  24.271,  68.00},
{17571.09,  24.316,  68.00} ,{ 17915.63,  24.358,  68.00},
{18260.16,  24.398,  68.00} ,{ 18604.69,  24.436,  68.00},
{18949.22,  24.473,  68.00} ,{ 19293.75,  24.508,  68.00},
{19638.28,  24.542,  68.00} ,{ 19982.81,  24.574,  68.00}
};

N=TH_1[0][0]+1;
i=j=0;
for(i=0;i<N;i++){
  for(j=0;j<3;j++){
    tables.TH[i][j]=TH_1[i][j];
  }
}
/* Converte as freqüências para o índice das amostras */
for(i=1;i<N;i++){
  tables.TH[i][INDEX] = Round(tables.TH[i][INDEX] / 44100 * 512);
}

/* Gera um mapeamento entre as FFT_SIZE/2 amostras do sinal de entrada
  e os N coeficientes da tabela do limiar absoluto de audição */

/* Bordas */
for(i=1; i<tables.TH[1][INDEX]+1; i++){
  tables.Map[i]=1;
}
k[0]=i;
for(i=(int)(tables.TH[N-1][INDEX]); i<(FFT_SIZE/2); i++){
  tables.Map[i]=N-1;
}
k[1]=i;
```

```
  /* Todos os outros da tabela */
  for(i=2; i<N; i++){
    for(j=(int)tables.TH[i][INDEX]; j<(int)tables.TH[i+1][INDEX]+1; j++){
      tables.Map[j]=i;
    }
  }
  k[2]=j;
  tables.Map[0]=(int)Max(k,3);

  /* Dependendo da taxa de bits utilizada para a codificação, é adicionado um
    offset ao limiar absoluto de audição. Este offset  */
  if(bitrate>95){
    for(i=1; i<N; i++){
      tables.TH[i][ATH]=tables.TH[i][ATH]-12;
    }
  }
  for(j=1; j<N; j++){
    tables.LTq[j]=tables.TH[j][ATH];
  }
  tables.LTq[0]=j;
  tables.erro=0;
}/* fim fs=44100 */

else if(fs==48000){
/*  Frequencia | Taxa da Banda Crítica | Limiar Absoluto */
  double  TH_1[][3] ={
  {  102.0,  102.0,   102.0 },
  {   93.75,   0.925,  24.17} ,{   187.50,   1.842,  13.87},
  {  281.25,   2.742,  10.01} ,{   375.00,   3.618,   7.94},
  {  468.75,   4.463,   6.62} ,{   532.50,   5.272,   5.70},
  {  656.25,   6.041,   5.00} ,{   750.00,   6.770,   4.45},
  {  843.75,   7.457,   4.00} ,{   937.50,   8.103,   3.61},
  { 1031.25,   8.708,   3.26} ,{  1125.00,   9.275,   2.93},
  { 1218.75,   9.805,   2.63} ,{  1312.50,  10.301,   2.32},
  { 1406.25,  10.765,   2.02} ,{  1500.00,  11.199,   1.71},
  { 1593.75,  11.606,   1.38} ,{  1687.50,  11.988,   1.04},
  { 1781.25,  12.347,   0.67} ,{  1875.00,  12.684,   0.29},
  { 1968.75,  13.002,  -0.11} ,{  2062.50,  13.302,  -0.54},
  { 2156.25,  13.586,  -0.97} ,{  2250.00,  13.855,  -1.43},
```

```
{ 2343.75,  14.111,  -1.88} ,{  2437.50,  14.354,  -2.34},
{ 2531.25,  14.585,  -2.79} ,{  2625.00,  14.807,  -3.22},
{ 2718.75,  15.018,  -3.62} ,{  2812.50,  15.221,  -3.98},
{ 2906.25,  15.415,  -4.30} ,{  3000.00,  15.602,  -4.57},
{ 3093.75,  15.783,  -4.77} ,{  3187.50,  15.956,  -4.91},
{ 3281.25,  16.124,  -4.98} ,{  3375.00,  16.287,  -4.97},
{ 3468.75,  16.445,  -4.90} ,{  3562.50,  16.598,  -4.76},
{ 3656.25,  16.746,  -4.55} ,{  3750.00,  16.891,  -4.29},
{ 3843.75,  17.032,  -3.99} ,{  3937.50,  17.169,  -3.64},
{ 4031.25,  17.303,  -3.26} ,{  4125.00,  17.434,  -2.86},
{ 4218.75,  17.563,  -2.45} ,{  4312.50,  17.688,  -2.04},
{ 4406.25,  17.811,  -1.63} ,{  4500.00,  17.932,  -1.24},
{ 4687.50,  18.166,  -0.51} ,{  4875.00,  18.392,   0.12},
{ 5062.50,  18.611,   0.64} ,{  5250.00,  18.823,   1.06},
{ 5437.50,  19.028,   1.39} ,{  5625.00,  19.226,   1.66},
{ 5812.50,  19.419,   1.88} ,{  6000.00,  19.606,   2.08},
{ 6187.50,  19.788,   2.27} ,{  6375.00,  19.964,   2.46},
{ 6562.50,  20.135,   2.65} ,{  6750.00,  20.300,   2.86},
{ 6937.50,  20.461,   3.09} ,{  7125.00,  20.616,   3.33},
{ 7312.50,  20.766,   3.60} ,{  7500.00,  20.912,   3.89},
{ 7687.50,  21.052,   4.20} ,{  7875.00,  21.188,   4.54},
{ 8062.50,  21.318,   4.91} ,{  8250.00,  21.445,   5.31},
{ 8437.50,  21.567,   5.73} ,{  8625.00,  21.684,   6.18},
{ 8812.50,  21.797,   6.67} ,{  9000.00,  21.906,   7.19},
{ 9375.00,  22.113,   8.33} ,{  9750.00,  22.304,   9.63},
{10125.00,  22.482,  11.08} ,{ 10500.00,  22.646,  12.71},
{10875.00,  22.799,  14.53} ,{ 11250.00,  22.941,  16.54},
{11625.00,  23.072,  18.77} ,{ 12000.00,  23.195,  21.23},
{12375.00,  23.309,  23.94} ,{ 12750.00,  23.415,  26.90},
{13125.00,  23.515,  30.14} ,{ 13500.00,  23.607,  33.67},
{13875.00,  23.694,  37.51} ,{ 14250.00,  23.775,  41.67},
{14625.00,  23.852,  46.17} ,{ 15000.00,  23.923,  51.04},
{15375.00,  23.991,  56.29} ,{ 15750.00,  24.054,  61.94},
{16125.00,  24.114,  68.00} ,{ 16500.00,  24.171,  68.00},
{16875.00,  24.224,  68.00} ,{ 17250.00,  24.275,  68.00},
{17625.00,  24.322,  68.00} ,{ 18000.00,  24.368,  68.00},
{18375.00,  24.411,  68.00} ,{ 18750.00,  24.452,  68.00},
{19125.00,  24.491,  68.00} ,{ 19500.00,  24.528,  68.00},
{19875.00,  24.564,  68.00} ,{ 20250.00,  24.597,  68.00},
```

```
};

N=TH_1[0][0]+1;
i=j=0;
for(i=0;i<N;i++){
  for(j=0;j<3;j++){
    tables.TH[i][j]=TH_1[i][j];
  }
}
/* Converte as freqüências para o índice das amostras */
for(i=1;i<N;i++){
  tables.TH[i][INDEX] = Round(tables.TH[i][INDEX] / 48000 * 512);
}

/* Gera um mapeamento entre as FFT_SIZE/2 amostras do sinal de entrada
  e os N coeficientes da tabela do limiar absoluto de audição */

/* Bordas */
for(i=1; i<tables.TH[1][INDEX]+1; i++){
  tables.Map[i]=1;
}
k[0]=i;
for(i=(int)(tables.TH[N-1][INDEX]); i<(FFT_SIZE/2); i++){
  tables.Map[i]=N-1;
}
k[1]=i;

/* Todos os outros da tabela */
for(i=2; i<N; i++){
  for(j=(int)tables.TH[i][INDEX]; j<(int)tables.TH[i+1][INDEX]+1; j++){
    tables.Map[j]=i;
  }
}
k[2]=j;
tables.Map[0]=(int)Max(k,3);

/* Dependendo da taxa de bits utilizada para a codificação, é adicionado um
  offset ao limiar absoluto de audição. Este offset  */
```

```
      if(bitrate>95){
        for(i=1; i<N; i++){
          tables.TH[i][ATH]=tables.TH[i][ATH]-12;
        }
      }
      for(j=1; j<N; j++){
        tables.LTq[j]=tables.TH[j][ATH];
      }
      tables.LTq[0]=j;
      tables.erro=0;
    }/* fim fs=48000 */

    else{
      printf("Freqüência não suportada");
      tables.erro=1;
    }
  }
  else{
    printf("Layer não suportado");
    tables.erro=1;
  }
  return tables;
}
```

C.6 Table Critical Band Boundaries

```
//CB = Table_critical_band_boundaries(Layer, fs)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

void Table_critical_band_boundaries(int Layer, int fs, table tables){
  int cb44100[26]={26,1, 2, 3, 5, 6, 8, 9, 11, 13, 15, 17, 20, 23,
```

```
            27, 32, 37, 45, 50, 55, 61, 68, 75, 81, 93, 106};
  int cb48000[27]={27,1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 14, 16, 19, 21,
            25, 29, 35, 41, 49, 53, 58, 65, 73, 79, 89, 102};
  int i;


  /* Carrega a tabela de Banda Críticas de acordo com a taxa de amostragem */
  if(Layer ==1){
    if(fs == 44100){

      for(i=1;i<cb44100[0];i++){
        tables.CB[i]=cb44100[i];
      }
      tables.CB[0]=cb44100[0];
      tables.erro=0;
    }
    else if(fs == 48000){

      for(i=1;i<cb48000[0];i++){
        tables.CB[i]=cb44100[i];
      }
      tables.CB[0]=cb48000[0];
      tables.erro=0;
    }
    else{
      tables.erro=1;
    }
  }
  else{
    tables.erro=1;
  }
}
```

C.7 Table Analysis Window

```
//C = Table_analysis_window

#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"


void Table_analysis_window(table tables){

  int i;
  float c[513]={ 513.0,
   0.0        , -0.000000477, -0.000000477, -0.000000477 ,
  -0.000000477, -0.000000477, -0.000000477, -0.000000954 ,
  -0.000000954, -0.000000954, -0.000000954, -0.000001431 ,
  -0.000001431, -0.000001907, -0.000001907, -0.000002384 ,
  -0.000002384, -0.000002861, -0.000003338, -0.000003338 ,
  -0.000003815, -0.000004292, -0.000004768, -0.000005245 ,
  -0.000006199, -0.000006676, -0.000007629, -0.000008106 ,
  -0.000009060, -0.000010014, -0.000011444, -0.000012398 ,
  -0.000013828, -0.000014782, -0.000016689, -0.000018120 ,
  -0.000019550, -0.000021458, -0.000023365, -0.000025272 ,
  -0.000027657, -0.000030041, -0.000032425, -0.000034809 ,
  -0.000037670, -0.000040531, -0.000043392, -0.000046253 ,
  -0.000049591, -0.000052929, -0.000055790, -0.000059605 ,
  -0.000062943, -0.000066280, -0.000070095, -0.000073433 ,
  -0.000076771, -0.000080585, -0.000083923, -0.000087261 ,
  -0.000090599, -0.000093460, -0.000096321, -0.000099182 ,
   0.000101566,  0.000103951,  0.000105858,  0.000107288 ,
   0.000108242,  0.000108719,  0.000108719,  0.000108242 ,
   0.000106812,  0.000105381,  0.000102520,  0.000099182 ,
   0.000095367,  0.000090122,  0.000084400,  0.000077724 ,
   0.000069618,  0.000060558,  0.000050545,  0.000039577 ,
   0.000027180,  0.000013828, -0.000000954, -0.000017166 ,
  -0.000034332, -0.000052929, -0.000072956, -0.000093937 ,
  -0.000116348, -0.000140190, -0.000165462, -0.000191212 ,
  -0.000218868, -0.000247478, -0.000277042, -0.000307560 ,
  -0.000339031, -0.000371456, -0.000404358, -0.000438213 ,
  -0.000472546, -0.000507355, -0.000542164, -0.000576973 ,
```

```
-0.000611782, -0.000646591, -0.000680923, -0.000714302 ,
-0.000747204, -0.000779152, -0.000809669, -0.000838757 ,
-0.000866413, -0.000891685, -0.000915051, -0.000935555 ,
-0.000954151, -0.000968933, -0.000980854, -0.000989437 ,
-0.000994205, -0.000995159, -0.000991821, -0.000983715 ,
 0.000971317,  0.000953674,  0.000930786,  0.000902653 ,
 0.000868797,  0.000829220,  0.000783920,  0.000731945 ,
 0.000674248,  0.000610352,  0.000539303,  0.000462532 ,
 0.000378609,  0.000288486,  0.000191689,  0.000088215 ,
-0.000021458, -0.000137329, -0.000259876, -0.000388145 ,
-0.000522137, -0.000661850, -0.000806808, -0.000956535 ,
-0.001111031, -0.001269817, -0.001432419, -0.001597881 ,
-0.001766682, -0.001937389, -0.002110004, -0.002283096 ,
-0.002457142, -0.002630711, -0.002803326, -0.002974033 ,
-0.003141880, -0.003306866, -0.003467083, -0.003622532 ,
-0.003771782, -0.003914356, -0.004048824, -0.004174709 ,
-0.004290581, -0.004395962, -0.004489899, -0.004570484 ,
-0.004638195, -0.004691124, -0.004728317, -0.004748821 ,
-0.004752159, -0.004737377, -0.004703045, -0.004649162 ,
-0.004573822, -0.004477024, -0.004357815, -0.004215240 ,
-0.004049301, -0.003858566, -0.003643036, -0.003401756 ,
 0.003134727,  0.002841473,  0.002521515,  0.002174854 ,
 0.001800537,  0.001399517,  0.000971317,  0.000515938 ,
 0.000033379, -0.000475883, -0.001011848, -0.001573563 ,
-0.002161503, -0.002774239, -0.003411293, -0.004072189 ,
-0.004756451, -0.005462170, -0.006189346, -0.006937027 ,
-0.007703304, -0.008487225, -0.009287834, -0.010103703 ,
-0.010933399, -0.011775017, -0.012627602, -0.013489246 ,
-0.014358521, -0.015233517, -0.016112804, -0.016994476 ,
-0.017876148, -0.018756866, -0.019634247, -0.020506859 ,
-0.021372318, -0.022228718, -0.023074150, -0.023907185 ,
-0.024725437, -0.025527000, -0.026310921, -0.027073860 ,
-0.027815342, -0.028532982, -0.029224873, -0.029890060 ,
-0.030526638, -0.031132698, -0.031706810, -0.032248020 ,
-0.032754898, -0.033225536, -0.033659935, -0.034055710 ,
-0.034412861, -0.034730434, -0.035007000, -0.035242081 ,
-0.035435200, -0.035586357, -0.035694122, -0.035758972 ,
 0.035780907,  0.035758972,  0.035694122,  0.035586357 ,
 0.035435200,  0.035242081,  0.035007000,  0.034730434 ,
```

```
 0.034412861,  0.034055710,  0.033659935,  0.033225536 ,
 0.032754898,  0.032248020,  0.031706810,  0.031132698 ,
 0.030526638,  0.029890060,  0.029224873,  0.028532982 ,
 0.027815342,  0.027073860,  0.026310921,  0.025527000 ,
 0.024725437,  0.023907185,  0.023074150,  0.022228718 ,
 0.021372318,  0.020506859,  0.019634247,  0.018756866 ,
 0.017876148,  0.016994476,  0.016112804,  0.015233517 ,
 0.014358521,  0.013489246,  0.012627602,  0.011775017 ,
 0.010933399,  0.010103703,  0.009287834,  0.008487225 ,
 0.007703304,  0.006937027,  0.006189346,  0.005462170 ,
 0.004756451,  0.004072189,  0.003411293,  0.002774239 ,
 0.002161503,  0.001573563,  0.001011848,  0.000475883 ,
-0.000033379, -0.000515938, -0.000971317, -0.001399517 ,
-0.001800537, -0.002174854, -0.002521515, -0.002841473 ,
 0.003134727,  0.003401756,  0.003643036,  0.003858566 ,
 0.004049301,  0.004215240,  0.004357815,  0.004477024 ,
 0.004573822,  0.004649162,  0.004703045,  0.004737377 ,
 0.004752159,  0.004748821,  0.004728317,  0.004691124 ,
 0.004638195,  0.004570484,  0.004489899,  0.004395962 ,
 0.004290581,  0.004174709,  0.004048824,  0.003914356 ,
 0.003771782,  0.003622532,  0.003467083,  0.003306866 ,
 0.003141880,  0.002974033,  0.002803326,  0.002630711 ,
 0.002457142,  0.002283096,  0.002110004,  0.001937389 ,
 0.001766682,  0.001597881,  0.001432419,  0.001269817 ,
 0.001111031,  0.000956535,  0.000806808,  0.000661850 ,
 0.000522137,  0.000388145,  0.000259876,  0.000137329 ,
 0.000021458, -0.000088215, -0.000191689, -0.000288486 ,
-0.000378609, -0.000462532, -0.000539303, -0.000610352 ,
-0.000674248, -0.000731945, -0.000783920, -0.000829220 ,
-0.000868797, -0.000902653, -0.000930786, -0.000953674 ,
 0.000971317,  0.000983715,  0.000991821,  0.000995159 ,
 0.000994205,  0.000989437,  0.000980854,  0.000968933 ,
 0.000954151,  0.000935555,  0.000915051,  0.000891685 ,
 0.000866413,  0.000838757,  0.000809669,  0.000779152 ,
 0.000747204,  0.000714302,  0.000680923,  0.000646591 ,
 0.000611782,  0.000576973,  0.000542164,  0.000507355 ,
 0.000472546,  0.000438213,  0.000404358,  0.000371456 ,
 0.000339031,  0.000307560,  0.000277042,  0.000247478 ,
 0.000218868,  0.000191212,  0.000165462,  0.000140190 ,
```

```
   0.000116348,  0.000093937,  0.000072956,  0.000052929 ,
   0.000034332,  0.000017166,  0.000000954, -0.000013828 ,
  -0.000027180, -0.000039577, -0.000050545, -0.000060558 ,
  -0.000069618, -0.000077724, -0.000084400, -0.000090122 ,
  -0.000095367, -0.000099182, -0.000102520, -0.000105381 ,
  -0.000106812, -0.000108242, -0.000108719, -0.000108719 ,
  -0.000108242, -0.000107288, -0.000105858, -0.000103951 ,
   0.000101566,  0.000099182,  0.000096321,  0.000093460 ,
   0.000090599,  0.000087261,  0.000083923,  0.000080585 ,
   0.000076771,  0.000073433,  0.000070095,  0.000066280 ,
   0.000062943,  0.000059605,  0.000055790,  0.000052929 ,
   0.000049591,  0.000046253,  0.000043392,  0.000040531 ,
   0.000037670,  0.000034809,  0.000032425,  0.000030041 ,
   0.000027657,  0.000025272,  0.000023365,  0.000021458 ,
   0.000019550,  0.000018120,  0.000016689,  0.000014782 ,
   0.000013828,  0.000012398,  0.000011444,  0.000010014 ,
   0.000009060,  0.000008106,  0.000007629,  0.000006676 ,
   0.000006199,  0.000005245,  0.000004768,  0.000004292 ,
   0.000003815,  0.000003338,  0.000003338,  0.000002861 ,
   0.000002384,  0.000002384,  0.000001907,  0.000001907 ,
   0.000001431,  0.000001431,  0.000000954,  0.000000954 ,
   0.000000954,  0.000000954,  0.000000477,  0.000000477 ,
   0.000000477,  0.000000477,  0.000000477,  0.000000477  };

   tables.C[0]=c[0];

  /* Carrega os coeficientes da Janela Ci do Banco de Filtros */
  for(i=1;i<513;i++){
    tables.C[i]=c[i];
  }
}
```

C.8 Analysis Subband Filter

```
//double * Analysis_subband_filter(double Input, int n, double C);

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

double * Analysis_subband_filter(double *Input, int n, float *C, int len, double *s){

  int nmax, i, j, k;
  float z[512];
  float y[64];

  nmax=len;

  if(((n+31)>nmax) | (n<1)){
    for(i=0;i<32;i++){
    s[i]=0;
    }
    return s;
  }

  j=0;

  /* Janela o vetor de entrada pelo vetor C. Isto produz o buffer Z */
  for(i=0;i<512;i++){
    z[i]=Input[i]*C[i+1];
  }

  /* Cálculo parcial: 64 coeficientes Yi */
  for(i=0;i<64;i++){
    y[i]=0;
    for(j=0;j<8; j++){
      y[i]=y[i]+z[i + 64 * j];
    }
  }

  /* Calcula as 32 amostras Si das sub-bandas */
  for(i=0;i<32;i++){
    s[i]=0;
```

```
    for(k=0;k<64; k++){
      s[i]=s[i]+m[i][k]*y[k];
    }
  }
  for(i=0;i<32;i++){
  }
  return (s);

}
```

C.9 Scale Factors

```
//Scale_factors(S).c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"



double * Scale_factors(double S[12][32], double *scf, short int *scf_index){

  int i, j, k, N;
  double si_min;
  double aux[12];
  double Table_scf[64]={
  2.00000000000000, 1.58740105196820, 1.25992104989487, 1.00000000000000,
  0.79370052598410, 0.62996052494744, 0.50000000000000, 0.39685026299205,
  0.31498026247372, 0.25000000000000, 0.19842513149602, 0.15749013123686,
  0.12500000000000, 0.09921256574801, 0.07874506561843, 0.06250000000000,
  0.04960628287401, 0.03937253280921, 0.03125000000000, 0.02480314143700,
  0.01968626640461, 0.01562500000000, 0.01240157071850, 0.00984313320230,
  0.00781250000000, 0.00620078535925, 0.00492156660115, 0.00390625000000,
  0.00310039267963, 0.00246078330058, 0.00195312500000, 0.00155019633981,
  0.00123039165029, 0.00097656250000, 0.00077509816991, 0.00061519582514,
```

```
  0.00048828125000, 0.00038754908495, 0.00030759791257, 0.00024414062500,
  0.00019377454248, 0.00015379895629, 0.00012207031250, 0.00009688727124,
  0.00007689947814, 0.00006103515625, 0.00004844363562, 0.00003844973907,
  0.00003051757813, 0.00002422181781, 0.00001922486954, 0.00001525878906,
  0.00001211090890, 0.00000961243477, 0.00000762939453, 0.00000605545445,
  0.00000480621738, 0.00000381469727, 0.00000302772723, 0.00000240310869,
  0.00000190734863, 0.00000151386361, 0.00000120155435, 1E-20};

    N=64;

    /* Busca a maior amostra de cada sub-banda, o Scale factor será o primeiro valor
    da tabela maior que esta amostra*/
    for(i=0;i<32;i++){
      for(k=0;k<12;k++){
        aux[k]=S[k][i];
      }
      si_min=MaxAbs(aux,k);

      /* Se o módulo de alguma amostra for maior do que 2 */
      if(si_min>Table_scf[0]){
        printf("Aviso: não é possível encontrar o Fator de Escala\n");
        scf[i]=Table_scf[0];
      }
      else{
        j=0;
        while(( j < N )& (si_min > Table_scf[N - j - 1])){
          j++;
        }
        scf[i+1]=Table_scf[N-j-1];
        scf_index[i]=N-j-1;

        if(scf_index[i]>62){
          scf_index[i]=62 ;
        }
      }
    }
    scf[0]=i+1;
    return scf;
}
```

C.10 FFT Analysis

```
//[X, Delta]  = FFT_Analysis(Input, n)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./lib/fftw-3.0.1/api/fftw3.h"
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

double * FFT_Analysis(double *Input, int n, double *X){

  double x[2];
  double y, Delta;
  int i;

  fftw_complex *in, *out;
  fftw_plan p;

  /* Aloca espaço para o vetor de entrada no formato da biblioteca FFTW */
  in = fftw_malloc(sizeof(fftw_complex) * 512);

  /* Multiplica o vetor de entrada pela janela de Hanning */
  for(i=0;i<FFT_SIZE;i++){
    in[i][0]=Input[FFT_SIZE-1-i]*sqrt(8.0/3.0)*0.5*(1.0-cos(2*pi*i/512));
    in[i][1]=0.0;
  }

  /* Aloca espaço para o vetor de saída no formato da biblioteca FFTW */
  out = fftw_malloc(sizeof(fftw_complex) * 512);

  /* Calcula a FFT do sinal janelado */
  p = fftw_plan_dft_1d(512, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
  fftw_execute(p);
  fftw_destroy_plan(p);
```

```
  /* Calcula a densidade de potência do espectro e normaliza com relação a 96 dB */
  for(i=0;i<FFT_SIZE/2;i++){
    y=sqrt( pow((out[i][0]), 2) + pow((out[i][1]),2) );
    x[1]=MIN_POWER;
    x[0]=20 * (log10(y/FFT_SIZE));
    X[i]=Max(x,2);
  }

  Delta=96-Max(X,FFT_SIZE);
  for(i=0;i<FFT_SIZE;i++){
    X[i]=X[i]+Delta;
  }
  /* Desaloca os vetores de entrada e saída da FFTW */
  fftw_free(in);
  fftw_free(out);
  return X;
}
```

C.11 Sound Pressure Level

```
//Lsb = Sound_pressure_level(X, scf)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

double * Sound_pressure_level(double *X, double *scf, double *Lsb, int lenX, int lenScf){

  double Xmin, local_max;
  double aux[2];
  int i, j, n;

  /* Checa os parâmtros de entrada */
```

```
   if(lenX != FFT_SIZE){
    printf("Tamanho do Espectro da Densidade de Potência inesperado\n");
  }
  if((lenScf-1)!= N_SUBBAND){
    printf("Inesperado número de Fatores de Escala\n");
  }

  /* Busca o maior nível de potência espectral de cada sub-banda
     e calcula o nível de pressão sonora Lsb */
  Xmin=Min(X, lenX);
  n = FFT_SIZE / 2 / N_SUBBAND;
  for(i=0; i<N_SUBBAND; i++){
    local_max = Xmin;
    for(j=0; j<n; j++){
      aux[0]=X[(i*n)+j];
      aux[1]=local_max;
      local_max=Max(aux,2);
    }
    aux[0]=(20 * log10(scf[i+1] * 32768) )- 10;
    if(aux[0]>local_max){
      Lsb[i]=aux[0];
    }
    else{
      Lsb[i]=local_max;
    }
  }
  return Lsb;
}
```

C.12 Find Tonal Components

```
//function [Flags, Tonal_list, Non_tonal_list] = ...
//   Find_tonal_components(X, TH, Map, CB)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
```

```
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"



tonal Find_tonal_components(double *FFTin, table tables, int lenX, tonal tonals){

  float local_max_list[( (FFT_SIZE/2) )][2];
  double power, weight, aux1, aux2[2];
  int i, j, k, aux, is_tonal, index, counter, J[12];

  j=i=k=aux=index=counter=0;

  for(i=0; i<257; i++){
    tonals.Tonal_list[i][0]=0;
    tonals.Tonal_list[i][1]=MIN_POWER;
  }
  for(i=0; i<26; i++){
    tonals.Non_tonal_list[i][0]=0;
    tonals.Non_tonal_list[i][1]=MIN_POWER;
  }

  /* Checa os parâmtros de entrada */
  if(lenX != FFT_SIZE){
    printf("Tamanho do Espectro da Densidade de Potência inesperado\n");
  }

  /* Lista os Flags para todas as linhas de freqüência */
  for(i=0; i<FFT_SIZE/2; i++){
    tonals.Flags[i]=NOT_EXAMINED;
  }

  /* Busca os máximos locais */
  local_max_list[0][0]=0;
  counter=1;
  for(k=0; k<FFT_SIZE/2; k++){
    if ( (FFTin[k] > FFTin[k-1]) && (FFTin[k] >= FFTin[k+1]) && (k > 2) && (k <= 250) ){
         local_max_list[counter][INDEX] = k+1;
```

```
    local_max_list[counter][SPL] = FFTin[k];
    counter = counter + 1;
  }
}

local_max_list[0][0] = counter;
local_max_list[0][1] = counter;

counter=0;
tonals.Tonal_list[0][0]=counter;
tonals.Tonal_list[0][1]=counter;

/* Lista os componentes tonais e calcula o nível de pressão sonora */
if(local_max_list[0][0]>0){

  for(i=1;i<(int)local_max_list[0][0]; i++){
      k = (int)local_max_list[i][INDEX];
    is_tonal = 1;

    /* Examina as freqüências vizinhas */
    if (2 < k && k < 63){
      J[0]=3;
      J[1]=-2;
      J[2]=2;
    }
    else if(63 <= k && k < 127){
      J[0]=5;
      J[1]=-3;
      J[2]=-2;
      J[3]=2;
      J[4]=3;
    }
    else if (127 <= k && k < 250){
      J[0]=11;
      J[1]=-6;
      J[2]=-5;
      J[3]=-4;
      J[4]=-3;
      J[5]=-2;
```

```
  J[6]=2;
  J[7]=3;
  J[8]=4;
  J[9]=5;
  J[10]=6;
}
else{
  is_tonal = 0;
  J[0]=0;
}

for(j=1;j<J[0]; j++){
  if((FFTin[k-1] - FFTin[k-1 + J[j]]) >= 7){
    aux=1;
  }
  else{
    aux=0;
  }
  is_tonal = is_tonal && aux;
}

/*   Se FFTin(k) é um componente tonal então é listado
    - O index k da linha espectral
    - Nível de pressão sonora
    - Seta o flag tonal */
if(is_tonal){
  counter++;
  tonals.Tonal_list[counter][INDEX]=k;
  tonals.Tonal_list[counter][SPL]=10*log10( pow(10,FFTin[k-2]/10) +
                        pow(10,FFTin[k-1]/10) + pow(10,FFTin[k]/10));
  tonals.Flags[k-1]=TONAL;
  J[J[0]]=-1;
  J[J[0]+1]=1;
  J[0]=J[0]+2;
  for(j=1;j<J[0]; j++){
    tonals.Flags[k+J[j]-1]=IRRELEVANT;
  }
}
j=0;
```

```
  }
}

tonals.Tonal_list[0][0]=counter;
tonals.Tonal_list[0][1]=counter;

/* Lista todos os componentes não-tonais e calcula a sua potência
  Todas as linhas espectrais que não foram examindas na busca anterior
  são somadas para formar os componentes não-tonais */
for(i=1; i<tables.CB[0]-1;i++){
  /* Para cada banda crítica, calcula a potência dos componentes não-tonais */
  power  = MIN_POWER; /* Soma parcial */
  weight = 0; /* Usado para calcular a média geométrica de cada banda crítica */
  for(k = (int)tables.TH[tables.CB[i]][INDEX];
        k < (int)ceil(tables.TH[tables.CB[i+1]][INDEX]);
        k++){
    if(tonals.Flags[k-1]==0/*NOT_EXAMINED*/){
      power=10*log10( pow(10,power/10) + pow(10,FFTin[k-1]/10) );
      aux1=pow(10,FFTin[k-1] / 10) * (tables.TH[tables.Map[k]][BARK] -i);
      weight   = weight + aux1;
      tonals.Flags[k-1]=IRRELEVANT;
    }
  }

  /* O index dos componentes não tonais é o index mais próximo
    da média geométrica da banda crítica */
  if (power <= MIN_POWER){
    aux2[0]=tables.TH[tables.CB[i]][INDEX];
    aux2[1]=tables.TH[tables.CB[i+1]][INDEX];
    index = Round( Mean(aux2,2));
  }
  else{
    index = (int)tables.TH[tables.CB[i]][INDEX] +
          Round( weight / pow(10, power / 10) *
           (tables.TH[tables.CB[i+1]][INDEX] -
          tables.TH[tables.CB[i]][INDEX]) );
  }
  if (index < 1){
    index = 1;
```

```
    }
    if (index > (FFT_SIZE/2)){
      index = (FFT_SIZE/2);
    }
    if (tonals.Flags[index-1] == TONAL){
      index = index + 1; /* Não pode haver dois componentes tonais consecutivos */
    }

    /* Para cada sub-banda
      - Index dos componentes não-tonais
      - Nível de pressão sonora deste componente */
    tonals.Non_tonal_list[i][INDEX] = index;
    tonals.Non_tonal_list[i][SPL] = power;
    tonals.Flags[index-1] = NON_TONAL;
  }

  tonals.Non_tonal_list[0][INDEX] = i;
  tonals.Non_tonal_list[0][SPL] = i;
  return(tonals);
}
```

C.13 Decimation

```
// function [DFlags, DTonal_list, DNon_tonal_list] = ...
// Decimation(X, Tonal_list, Non_tonal_list, Flags, TH, Map)



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fftw3.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

tonal Decimation(table tables, tonal tonals){
```

```
int i, k, k_next, j, l;
tonal ton;

ton=tonals;
j=0;
k=0;

for(i=0; i<257; i++){
  ton.Tonal_list[i][0]=MIN_POWER;
  ton.Tonal_list[i][1]=MIN_POWER;
}
for(i=0; i<27; i++){
  ton.Non_tonal_list[i][0]=MIN_POWER;
  ton.Non_tonal_list[i][1]=MIN_POWER;
}


/* Caso não-tonal */
if(tonals.Non_tonal_list!=0){
  for(i=1; i<(int)tonals.Non_tonal_list[0][0]; i++){
    k=tonals.Non_tonal_list[i][INDEX];
    /* Se k > Comprimento(Map)
      Flag= Irrelevante
      Senão */
    if(tonals.Non_tonal_list[i][SPL]<tables.TH[tables.Map[k]][ATH]){
      ton.Flags[k-1]=IRRELEVANT;
    }
    else{
      j++;
      ton.Non_tonal_list[j][0]=tonals.Non_tonal_list[i][0];
      ton.Non_tonal_list[j][1]=tonals.Non_tonal_list[i][1];
    }
  }

  ton.Non_tonal_list[0][0]=j;
  ton.Non_tonal_list[0][1]=j;
  j=0;
}
```

```
/* Caso Tonal */
if(tonals.Tonal_list!=0){
  for(i=1; i<(int)tonals.Tonal_list[0][0]+1; i++){
    k=tonals.Tonal_list[i][INDEX];
    /* Se k > Comprimento(Map)
      Flag= Irrelevante
      Senão */
    if(tonals.Tonal_list[i][SPL]<tables.TH[tables.Map[k]][ATH]){
      ton.Flags[k-1]=IRRELEVANT;
    }
    else{
      j++;
      ton.Tonal_list[j][0]=tonals.Tonal_list[i][0];
      ton.Tonal_list[j][1]=tonals.Tonal_list[i][1];
    }
  }
  ton.Tonal_list[0][0]=j;
  ton.Tonal_list[0][1]=j;
}
 i=j=1;
l=1;

/* Eliminar as componentes tonais que está a menos
  da metade do comprimento da banda crítica da componente
  vizinha */
if(ton.Tonal_list!=0){
  while(i<ton.Tonal_list[0][0]){
    k=ton.Tonal_list[i][INDEX];
    k_next=ton.Tonal_list[i+1][INDEX];
    if ( (tables.TH[tables.Map[k_next]][BARK] - tables.TH[tables.Map[k]][BARK]) < 0.5){
      if (ton.Tonal_list[i][SPL] < ton.Tonal_list[i + 1][SPL]){
        for(j=1; j<i-1;j++){
          ton.Tonal_list[l][0]=ton.Tonal_list[j][0];
          ton.Tonal_list[l][1]=ton.Tonal_list[j][1];
          l++;
        }
        for(j=i+1; j<(int)ton.Tonal_list[0][0]; j++){
          ton.Tonal_list[l][0]=ton.Tonal_list[j][0];
          ton.Tonal_list[l][1]=ton.Tonal_list[j][1];
```

```
            l++;
          }
          ton.Flags[k-1]=IRRELEVANT;
          l=1;
        }
        else{
          for(j=1; j<i;j++){
            ton.Tonal_list[l][0]=ton.Tonal_list[j][0];
            ton.Tonal_list[l][1]=ton.Tonal_list[j][1];
            l++;
          }
          for(j=i+2; j<(int)ton.Tonal_list[0][0]; j++){
            ton.Tonal_list[l][0]=ton.Tonal_list[j][0];
            ton.Tonal_list[l][1]=ton.Tonal_list[j][1];
            l++;
          }
          ton.Flags[k_next-1]=IRRELEVANT;
          l=1;
        }
        i++;
      }
      else i++;
    }
  }
  tonals = ton;
  return(tonals);
}
```

C.14 Individual Masking Thresholds

```
//function  [LTt, LTn] = Individual_masking_thresholds(X, Tonal_list, ...
//   Non_tonal_list, TH, Map)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
```

```
#include "common.h"
#include "mpeg_fun.h"

lt Individual_masking_thresholds(double *X,tonal tonals, table tables, lt lts){

  int i, k, j;
  double zi, zj, dz, avtm, avnm, vf;

  i=vf=k=0;

  /* O limiar de mascaramento individual para componentes tonais e não-tonais
    são colocados em - infinito se a função de mascaramento tenha atenuação
    infinita fora do limite de -3 e 8 barks, ou seja, a componente não
    tem efeito de mascaramento fora desses limites */
  if(tonals.Tonal_list[0][0]!=0){
    for(i=0; i<tonals.Tonal_list[0][0]+1; i++){
      for(j=0; j<tables.TH[0][0]+1; j++){
        lts.LTt[i][j]=MIN_POWER;
      }
    }
  }

  for(i=0; i<tonals.Non_tonal_list[0][0]+1; i++){
    for(j=0; j< tables.TH[0][0]+1; j++){
      lts.LTn[i][j]=MIN_POWER;
    }
  }

  /* Só um subconjunto de amostras são consideradas para o cálculo
    do limiar de mascaramento global. o número destas amostras dependem
    da taxa de amostragem. As informações necessárias encontram-se em TH
    que contém as freqüências, as taxas da banda crítica e o limiar absoluto */
  for(i=1; i<tables.TH[0][0]+1; i++){
    zi=tables.TH[i][BARK]; /* Taxa da banda crítica da freqüência considerada */
    if(tonals.Tonal_list[0][0]!=0){
      for(k=1; k<tonals.Tonal_list[0][0]+1; k++){
        /* Para cada componente tonal */
        j=tonals.Tonal_list[k][INDEX];
        zj=tables.TH[tables.Map[j]][BARK];/* Taxa da banda crítica do mascarador */
```

```
    dz=zi-zj;                /* Distância em Bark ao mascarador */

    if (dz >= -3 && dz < 8){

      /* Index do mascaramento */
      avtm = -1.525 - 0.275 * zj - 4.5;

      /* Função de mascaramento */
      if (-3 <= dz && dz < -1){
        vf = (17 * (dz + 1)) - (0.4 * X[j-1] + 6);
      }
      else if (-1 <= dz && dz < 0){
        vf = ( (0.4 * X[j-1]) + 6) * dz;
      }
      else if (0 <= dz && dz < 1){
        vf = -17 * dz;
      }
      else if (1 <= dz && dz < 8){
        vf = ((- (dz - 1)) * (17 - 0.15 * X[j-1])) - 17;
      }
      lts.LTt[k][i] = tonals.Tonal_list[k][SPL] + avtm + vf;
    }
  }
}

lts.LTt[0][0]=k;

for(k=1; k<tonals.Non_tonal_list[0][0]+1; k++){
  /* Para cada componente tonal */
  j=tonals.Non_tonal_list[k][INDEX];
  zj=tables.TH[tables.Map[j]][BARK];/* Taxa da banda crítica do mascarador */
  dz=zi-zj;                /* Distância em Bark ao mascarador */

  if (dz >= -3 && dz < 8){
    /* Index do mascaramento */
    avnm = -1.525 - 0.175 * zj - 0.5;

    /* Função de mascaramento */ /*  */
    if (-3 <= dz && dz < -1){
```

```
          vf = (17 * (dz + 1)) - (0.4 * X[j-1] + 6);
        }
        else if (-1 <= dz && dz < 0){
          vf = ( (0.4 * X[j-1]) + 6) * dz;
        }
        else if (0 <= dz && dz < 1){
          vf = -17 * dz;
        }
        else if (1 <= dz && dz < 8){
          vf = ((- (dz - 1)) * (17 - 0.15 * X[j-1])) - 17;
        }
        lts.LTn[k][i] = tonals.Non_tonal_list[k][SPL] + avnm + vf;
      }
    }
  }

  lts.LTn[0][0]=k;
  return(lts);
}
```

C.15 Global Masking Threshold

```
  //function  LTg = Global_masking_threshold(LTq, LTt, LTn)



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

lt Global_masking_threshold(table tables, lt lts){

  int i, j, m, n, N;
  double temp;
```

```
  m=0;
  N= tables.LTq[0];
  if (lts.LTt[0][0]!=0)
  m = (int)lts.LTt[0][0];
  n = (int)lts.LTn[0][0];


  /* O limiar de mascaramento global é calculado para um subconjunto
    de freqüências definidos em [1, Table 1.b]. Este limiar é a soma
    das potências dos limiares de mascaramento individuais (LTt e LTn) e do
    limiar em silêncio (LTq) */
  for(i=1; i<N; i++){
    /* Limiar em silêncio */
    temp = pow(10, tables.LTq[i] / 10);

    /* Contribuição das componentes tonais */
    if(lts.LTt[0][0]!=0){
      for(j=1; j<m+1; j++){
        temp = temp + pow(10,lts.LTt[j][i]/10);
      }
    }

    /* Contribuição das componentes não-tonais */
    for(j=1; j<n+1; j++){
      temp = temp + pow(10,lts.LTn[j][i] / 10);
    }
    lts.LTg[i] = 10 * log10(temp);
  }
  lts.LTg[0]=i;
  return(lts);
}
```

C.16 Minimum Masking Threshold

```
//function  LTmin = Minimum_masking_threshold(LTg, Map)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "./numerical/nr.h"
#include "./numerical/nrutil.h"
#include "./numerical/matb.h"
#include "common.h"
#include "mpeg_fun.h"

lt Minimum_masking_threshold(lt lts, table tables){

  int j, n, Subband_size;

  Subband_size = FFT_SIZE / 2 / N_SUBBAND; /* Para cada sub-banda */
  for(n=1; n<N_SUBBAND+1; n++){
    lts.LTmin[n] = lts.LTg[tables.Map[(n - 1) * Subband_size + 1]];

    /* Tenta para todas as amostras em cada sub-banda */
    for(j=2; j<Subband_size+1; j++){
      if(lts.LTg[tables.Map[(n - 1) * Subband_size + j]] < lts.LTmin[n]){
        lts.LTmin[n]=lts.LTg[tables.Map[(n - 1) * Subband_size + j]];
      }
    }
  }
  lts.LTmin[0]=n;
  return(lts);
}
```