Boletim Técnico da Escola Politécnica da USP

Departamento de Engenharia Eletrônica

BT/PEE/94-13

Análise do Ruído Sonoro em uma Sala de Aquisição de Amostras de Som com Microcomputador

Fábio Luís Romão
Reinaldo Silveira
Euvaldo Cabral Jr.

São Paulo - 1994

Romão, Fábio Luís
Análise do ruído sonoro em uma sala de aquisição de amostras de som com microcomputador / F.L. Romão, R. Silveira, E. Cabral Jr. -- São Paulo : EPUSP, 1994.
23p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia Eletrônica, BT/PEE/94-13)

1. Ruídos 2. Sinais - Processamento digital 3. Filtros digitais I. Cabral Jr., Euvaldo II. Silveira, Reinaldo III. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Eletrônica IV. Título V. Série

CDU 621.391.82
621.391
621.372.54

ANÁLISE DO RUÍDO SONORO EM UMA SALA DE AQUISIÇÃO DE AMOSTRAS DE SOM COM MICROCOMPUTADOR

Fábio Luís Romão
Reinaldo Silveira
Lab. de Sistemas Integrados (LSI), DEE, Escola Politécnica - USP
CAIXA POSTAL : 61548 CEP 05424-970 - São Paulo - SP
e-mail: romao@lsi.usp.br ou silveira@lsi.usp.br

Euvaldo Cabral Jr.
Lab. de Comunicações e Sinais (LCS), DEE, Escola Politécnica - USP
CAIXA POSTAL : 61548
CEP 05424-970 - São Paulo - SP
e-mail: euvaldo@lcs.usp.br

TEMA : Processamento Digital de Sinais

Sumário

Este trabalho tem por finalidade analisar as características do ruído encontrado numa amostragem de som. Obtida através de uma placa de aquisição de som para microcomputador pessoal, estas amostras estão geralmente submetidas a algumas condições que são inerentes do processo de gravação. Este trabalho tem por finalidade identificar, através de uma análise prática de uma destas amostras, quais são as fontes naturais de ruído e como podemos minimizá-los utilizando as técnicas do processamento digital de sinais.

Este trabalho será dividido em duas partes principais além de uma breve introdução. Na primeira parte é feita uma breve explanação da análise do sinal amostrado, e na segunda , é feito o tratamento da amostra através da filtragem conveniente do ruído identificado.

1. Introdução

Hoje tem-se disponível para os microcomputadores PC, uma variedade relativamente grande de placas de aquisição de som, a um custo bastante acessível se comparado com os equipamentos existentes a quatro ou cinco anos atrás. Estas placas são utilizadas basicamente como partes integrantes de sistemas multimídia, que integram as informações de som e imagem de alta definição em programas diversos como jogos, base de dados, teleconferência etc.

O usuário, tendo disponível estes equipamentos, pode incluir nas suas aplicações mensagens audíveis, efeitos sonoros, etc, ou ainda para os mais sofisticados implementar programas de reconhecimento de padrões e até mesmo da própria fala. Entretanto alguns problemas operacionais se fazem presentes.

Para a aquisição de sons, estas placas permitem selecionar várias taxas de amostragem a fim de selecionar o nível de fidelidade do som gravado, inclusive qualidade de CD. O usuário, desejando tal nível de fidelidade, não deve se esquecer de providenciar um ambiente adequado às gravações, um ambiente de estúdio ou coisa parecida. Porém na maioria das vezes isto não é possível e uma série de ruídos indesejáveis são introduzidos nas gravações. Os mais comuns são os referentes ao próprio ambiente onde se encontra a estação de trabalho, no caso o microprocessador. Mesmo sob silêncio "absoluto" alguns sons se destacam, podendo-se citar o ruído do ventilador do computador e eventualmente o som do aparelho condicionador de ar (quando houver). Pois bem este trabalho se propõe a analisar este tipo de ruído e eliminá-lo através de um pós-processsamento do sinal amostrado, utilizando as técnicas do processamento digital de sinais.

2. Análise da amostra

A amostra de som utilizado neste trabalho refere-se a uma pessoa pronunciando uma palavra, sendo precedida por um breve período de silêncio. No total têm-se pouco mais de dois segundos e meio gravados, a 'plotagem' desta função no tempo pode ser vista na figura 2.1.

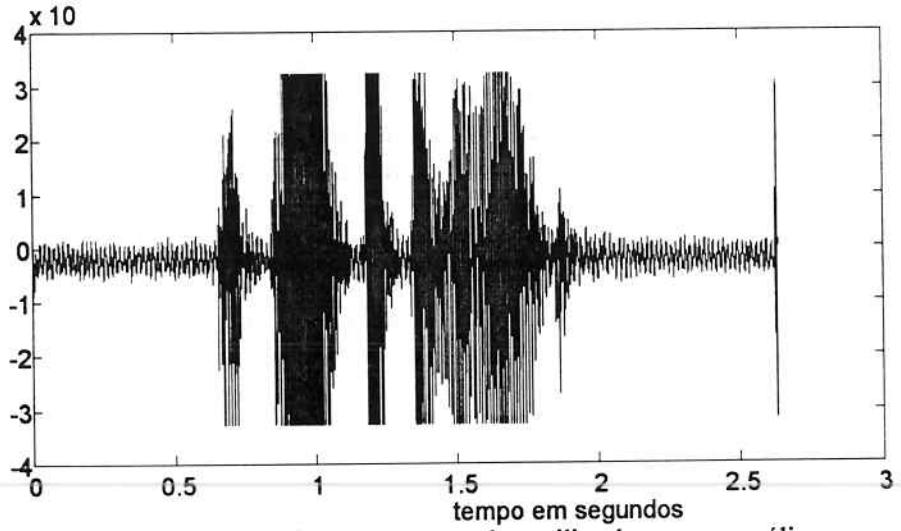

Figura 2.1: Sinal de voz amostrado, utilizado para a análise.

Utilizou-se o período que antecede a palavra como referência para a análise do silêncio. Inicialmente faremos uma verificação de como varia o espectro de freqüência do sinal, ao longo do tempo. Para isto foi considerada uma janela de amostragem correspondente a 1024 amostras balanceadas segundo o critério de Von Hann. Esta janela será deslocada ao longo do sinal amostrado a intervalos de 256 amostras correspondendo a aproximadamente 0,0232 segundos. Faz-se assim uma 'plotagem' tridimensional de como varia o espectro ao longo da amostra, a fim de se determinar:

1°) Existe alguma freqüência que se destaca no período de silêncio, a qual pode se classificar como ruído.

2°) Se o ruído identificado continua presente em todo amostra, caracterizando-se como um fenômeno intermitente, passível de ser eliminado sem trazer conseqüências indesejáveis para o sinal amostrado.

Na figura 2.2 pode ser visto a variação do espectro da amostragem de acordo com esta janela, nela vê-se quatro gráficos divididos de acordo com intervalos de tempo convenientes na amostra. O primeiro gráfico (acima à esquerda) corresponde ao primeiro meio segundo descontado 300 amostras de transitório, o segundo (à direita) corresponde ao segundo meio segundo, o terceiro (abaixo e a esquerda) ao terceiro, o quarto e último ao período de 1,5 a 2 segundos. a evolução do espectro do intervalo de 2 a 2,5 segundos é idêntico ao primeiro por isso não foi 'plotado'.

Pode-se ver que no primeiro intervalo existe uma composição de freqüências que permanece constante durante toda a análise. Deve ser lembrado que existe também uma componente contínua que é mais pronunciada que as demais componentes, o que pode ser um pouco mais difícil de ser constatado neste gráfico.

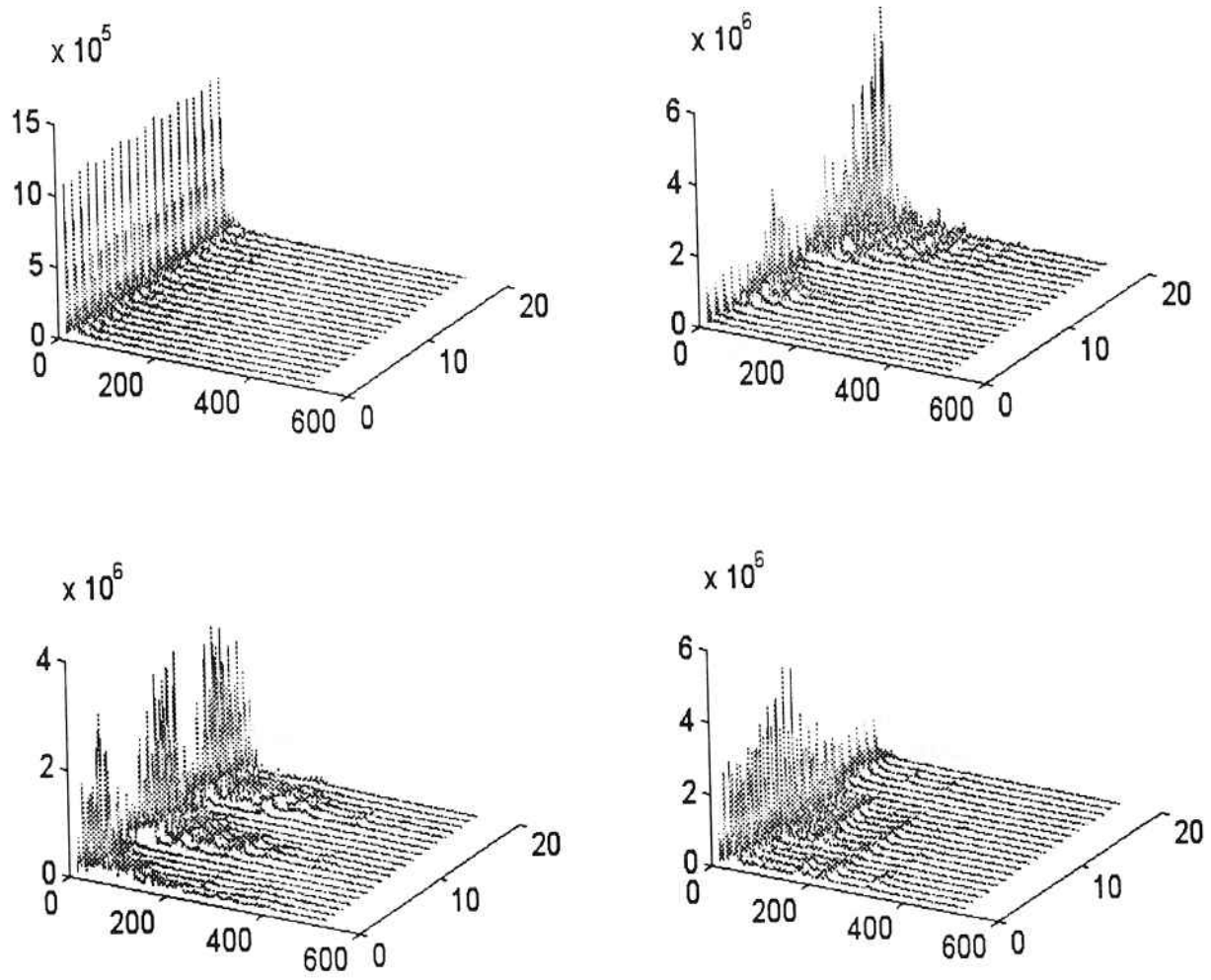

Figura 2.2: Evolução do espectro em freqüência da amostra.

Nos demais gráficos pode ser notado o aparecimento das outras freqüências em relação ao tempo. Deve-se tomar a precaução de notar as escalas dos vários gráficos e perceber que um é a continuação do outro.

A fim de identificar melhor as freqüências espúrias do período de silêncio é conveniente observar a figura 2.3 que é a média dos espectros da figura 2.2.a.

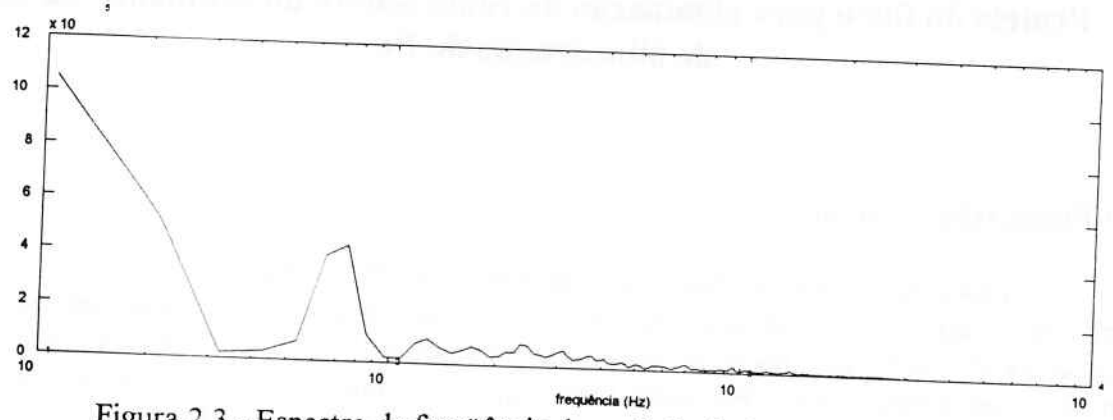

Figura 2.3 - Espectro de freqüência do período de 0,1 a 0,6 segundos.

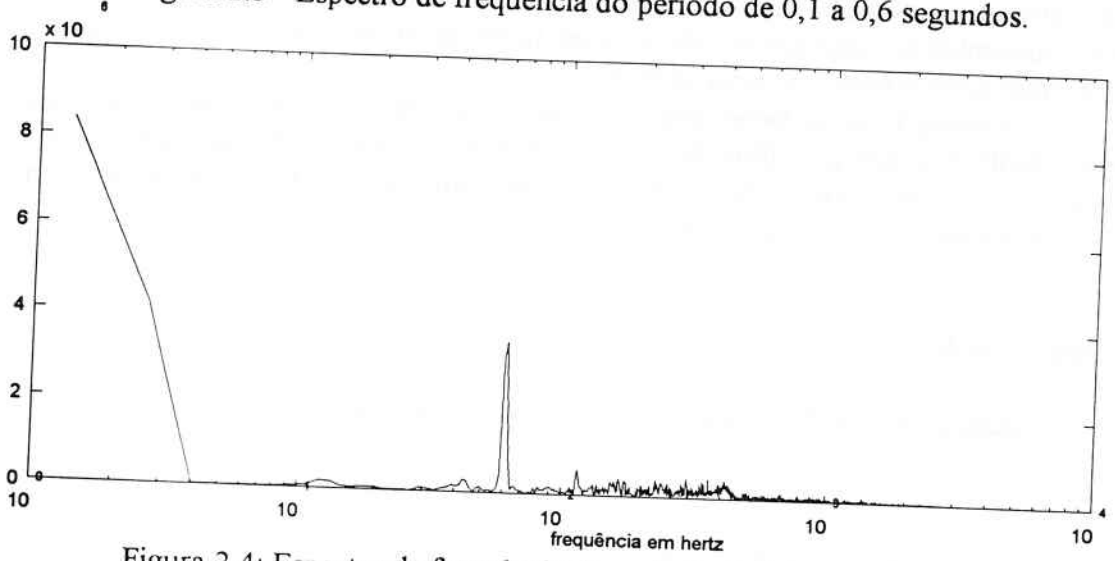

Figura 2.4: Espectro de freqüência do período de 0,1 a 0,6 segundos (FFT com grande número de pontos).

A figura 2.4 mostra o mesmo espectro da figura 2.3 som um número maior de amostras no domínio da freqüência, isto corresponde à transformada discreta de Fourier de uma janela maior que a usada na análise anterior. Desta vez foi usada uma janela de 8192 amostras, também balanceadas segundo Von Hann.

Neste segundo espectro pode-se ver nitidamente a freqüência de 60 Hz como fonte principal do ruído no intervalo de silêncio, donde pode-se concluir que a eliminação desta componente melhorará consideravelmente o qualidade do sinal amostrado. Esta componente tem origem, obviamente, na influência da rede elétrica, seja por indução eletromagnética ou pelo fato dos ventiladores produzirem um barulho surdo com 60 Hz de componente principal, de qualquer forma foi possível identificar a componente principal do ruído podendo, e com isto, trabalhar no sentido de eliminá-lo.

3. Projeto do filtro para eliminação do ruído sonoro do ventilador da fonte de alimentação do PC

3.1 Eliminação do ruído

Através da análise do ruído do ventilador do PC, explicitado na seção anterior, verificou-se que a principal componente espectral do mesmo se encontra em torno da freqüência de 60Hz. Isto pode ser explicado pelo fato do motor do ventilador do PC ser síncrono com a rede elétrica. Por outro lado, verifica-se também que a voz humana apresenta componentes de muito baixa intensidade em torno de freqüência de 60 Hz. Assim, eliminando-se as componentes do sinal sonoro situadas em torno de 60 Hz, elimina-se a maior parte do ruído do ventilador presente no sinal sonoro.

Desta forma torna-se necessário o projeto de um filtro rejeita-banda estreita, também chamado de filtro *Notch*. Este filtro deve ser muito bem projetado, a fim de rejeitar uma banda suficientemente estreita em torno de 60 Hz. Por outro lado, não deve afetar as demais componentes espectrais do sinal sonoro.

3.2. Filtro *Notch*

A equação de transferência de um filtro *Notch* é dada pela expressão abaixo:

$$H(z) = H_0 \frac{\prod_{i=1}^{p} (z - z_i)(z - z_i^*)}{\prod_{i=1}^{p} (z - p_i)(z - p_i^*)} \qquad (3.1)$$

onde $z_i = re^{j\omega i}$ e $p_i = \alpha re^{j\omega i}$

ω_i freqüência que se deseja eliminar;
$r \leq 1$ para garantir a estabilidade do filtro;
$\alpha \rightarrow 1$ para aproximar ao máximo o polo p_i do zero z_i.

Um filtro construído segundo (3.1) [1][2][3] possui as características desejadas neste trabalho. Assim foi projetado inicialmente um filtro *Notch* com as características acima e com $\omega_i = 2\pi 60$ para eliminar as freqüências em torno de 60 Hz.

O filtro implementado desta forma apresentou características de ganho e de rejeição de banda inferiores ao filtro eliptico que é apresentado na seção seguinte.

3.3. ELLIP: Programa para projeto automático de Filtros IRR elipticos

O projeto de um filtro IRR eliptico pode ser realizado seguindo métodos analíticos muito bem definidos e explicitados (de forma incompleta, porém) em [4]. Um estudo detalhado de [4] associado a uma análise matemática das propriedades dos filtros elipticos permitiu a sistematização do projeto de filtros elipticos a partir dos parâmetros básicos do filtro desejado. Foi implementado então um programa em C++ que "roda" em microcomputadores PC que realiza o trabalho de projeto de filtros digitais elipticos. Não faz parte do escopo deste trabalho apresentar detalhadamente este programa, mas sim mostrar como ele foi utilizado para projetar e filtrar o sinal sonoro afetado de ruído. No anexo 2 é mostrada a listagem do arquivo *ellip.h*

3.3.1. Características do programa para projeto de filtros elipticos

O programa desenvolvido calcula coeficientes reais para filtros digitais elipticos a partir da especificação de Ap (ripple da banda de passagem, em dB), Aa (máxima atenuação da banda de rejeição, em dB) , $\Omega p1$ (freqüência limite da banda de passagem), $\Omega p2$ (outra freqüência limite da banda de passagem, caso o filtro seja passa-banda ou rejeita-banda, ou seja, possua duas regiões de transição), $\Omega a1$ (freqüência limite da banda de rejeição), $\Omega a2$ (outra freqüência limite da banda de rejeição) e ω_s (freqüência de amostragem). Todas as freqüências devem ser fornecidas em Hz. O programa é capaz de gerar de forma automática filtros digitais passa-alta, passa-baixa, passa-banda ou rejeita-banda de qualquer ordem (ou pelo menos com valores de n bem grandes, caso seja necessário). O programa também realiza a filtragem de sinais com os filtros obtidos, além de fornecer o traçado da curva de transferência do filtro digital projetado e um relatório do filtro projetado.

O programa gera os filtros na configuração cascata-paralelo, onde cada componente cascateado apresenta no máximo ordem 2. Assim um filtro de ordem 8 pode ser projetado pelo programa com quatro "subfiltros" em cascata, em paralelo ou não com igual configuração. A configuração paralela apresenta somente duas derivações, e apenas está presente em alguns filtros especiais (como os filtros *Notch*es, por exemplo).

O programa gera filtros digitais com coeficientes reais, podendo portanto ser utilizado para o projeto de realização de filtros em hardware.

3.4. O Projeto do filtro *Notch* eliptico.

Quando diante da especificação do um filtro rejeita banda estrita, o programa não consegue atingir as especificações dadas baseado no seu modelo interno de filtros deste tipo, torna-se necessário considerar configurações compostas de filtros em paralelo. Verifica-se que o modelo intrínsico de filtro elíptico rejeita banda acaba se degenerando para faixas estreitas de freqüências. O programa pode projetar filtros de banda larga com freqüências Ωa e Ωp bem próximas, ou seja, com transição de banda abrupta. Assim, um filtro *Notch* pode ser obtido através da colocação em paralelo de um filtro passa-baixa e um filtro passa-alta, ajustando as freqüências de forma que a faixa que se deseja eliminar seja correspondente a faixa de rejeição comum entre os filtros, conforme o esquema da figura 3.1.

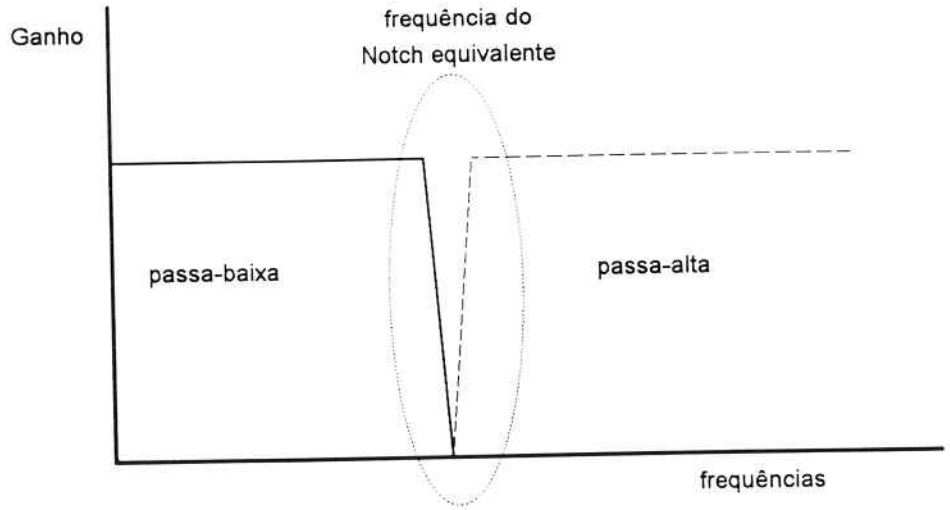

Figura 3.1: Filtro *Notch* obtido pelo programa

Assim, foi utilizado o ELLIP para projetar um filtro com as seguintes especificações:
Ap: 1 dB; Aa: 45 dB
$\Omega p1$: 58 Hz; $\Omega p2$: 62 Hz; $\Omega a1$: 60 Hz; $\Omega a2$; 60 Hz
ω_S: 11025 Hz
obtém-se assim um filtro com a configuração próxima a dada pela figura 3.2.

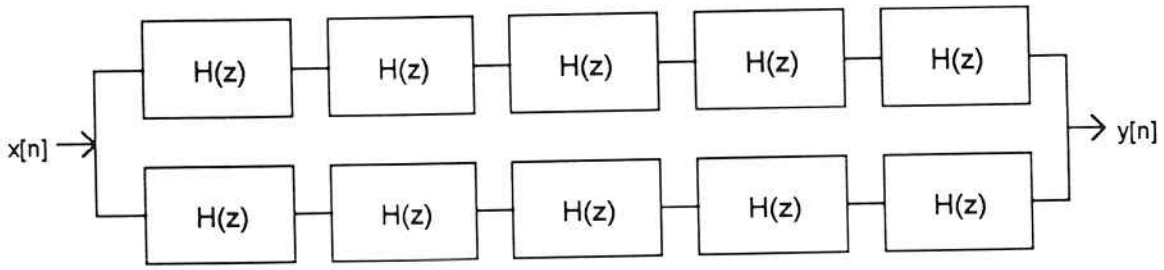

Figura 3.2: Configuração do filtro *Notch*

3.5. Resultados obtidos

O filtro *Notch* projetado através do programa apresentou boa resposta em freqüência, como poderá ser observado nas figuras seguintes. O relatório fornecido pelo ELLIP, com todos os parâmetros do filtro projetado, pode ser visto no Anexo 1

A curva de transferência do filtro obtido, gerada através do ELLIP, pode ser observada na figura 3.3. Verifica-se a resposta plana do filtro em todas as freqüências, a boa atenuação em 60 Hz e a boa transição entre as bandas. A figura 3.4 mostra o detalhamento da curva de transferência em torno de 60 Hz.

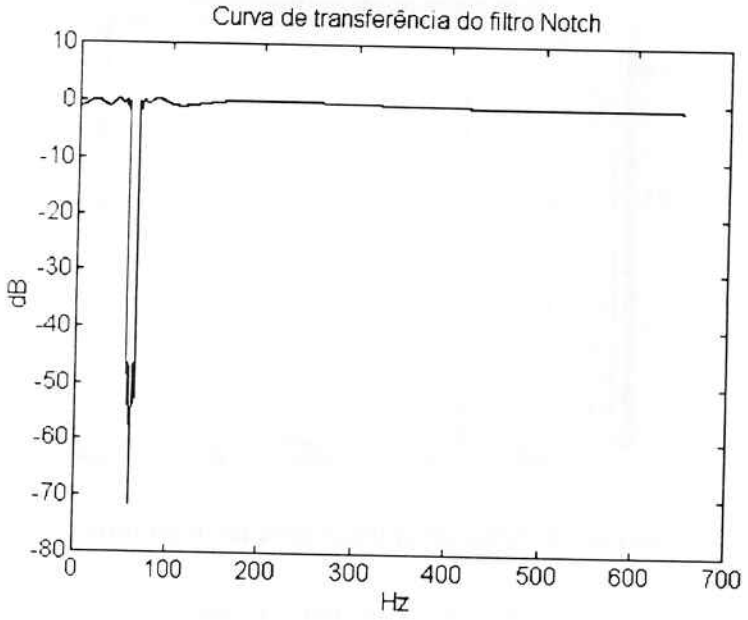

Figura 3.3: Curva de transferência do filtro

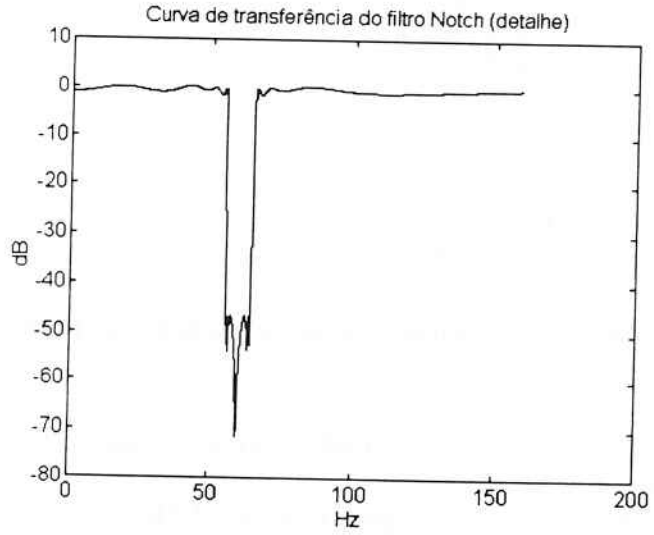

Figura 3.4: Detalhe da curva de transferência do filtro

Foi feito um levantamento prático da característica de transferência do filtro. Este levantamento procurou testar o filtro para algumas freqüências de entrada. Para testar o filtro fizemos passar por ele um sinal composto pela soma de várias senoides de mesma amplitude, cada uma com determinada freqüência. O espectro do sinal de entrada do teste do filtro pode ser visto na figuras 3.5 e 3.6.

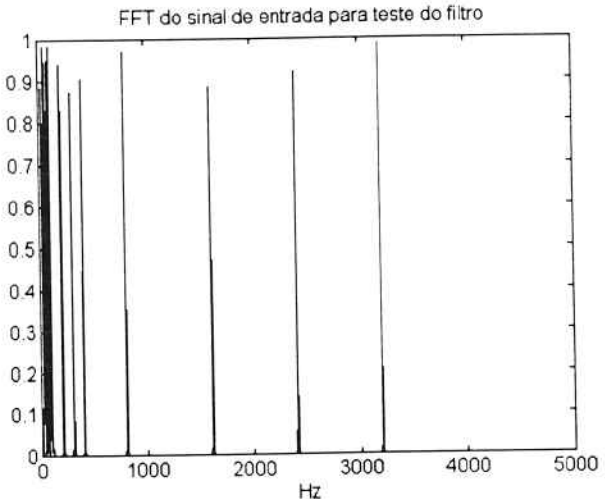

Figura 3.5: Sinal de entrada para teste do filtro.

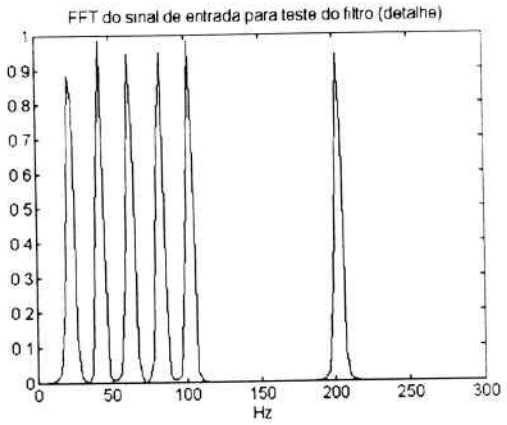

Figura 3.6: Detalhe do sinal de entrada para teste.

Passando o sinal das figuras 3.5 e 3.6 pelo filtro projetado obtivemos o sinal das figuras 3.7 e 3.8. Observamos nessas figuras como o filtro é eficiente no corte da freqüência de 60 Hz (não esquecer que a freqüência de amostragem é de 11025 Hz >> 60Hz). A precisão em torno do 60 Hz poderia ser melhor ajustada. No entanto observa-se que para altas freqüências ocorre uma atenuação razoável. Para aplicações onde o sinal filtrado é um sinal de voz humana, onde estas freqüências altas aparecem com componentes muito baixas, este filtro se mostra adequado.

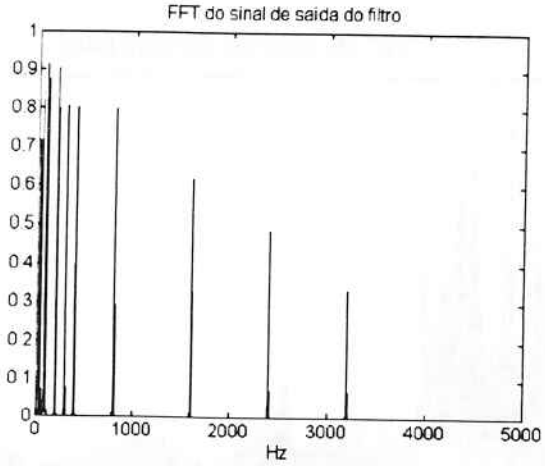

Figura 3.7: Sinal de teste filtrado.

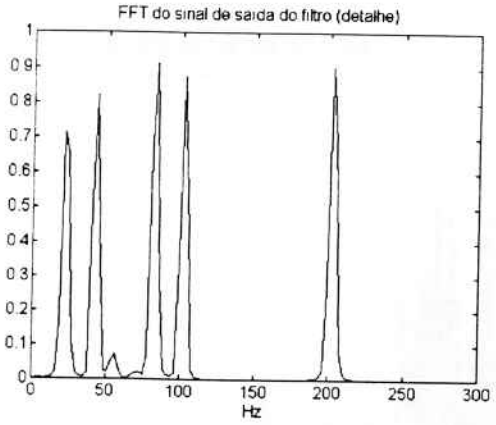

Figura 3.8: Detalhe do sinal de teste filtrado

3.5.1. Filtragem do sinal de voz

Verificada a adequação do filtro para aplicações que envolvam voz humana e o seu correto funcionamento para o corte das componentes em 60 Hz, este pode ser empregado para a filtragem do sinal de voz com o ruído do ventilador.

A título de demostração é mostrado aqui o resultado da filtragem de uma frase adquirida com a placa SoundBlaster, com freqüência de amostragem de 11025 Hz.

Na figura 3.9 observa-se o espectro do sinal antes e depois de filtrado. Nesta figura é interessante notar que o espectro praticamente não se altera (a menos em torno de 60 Hz), mostrando a boa característica de transferência do filtro em toda a faixa de freqüências relevantes.

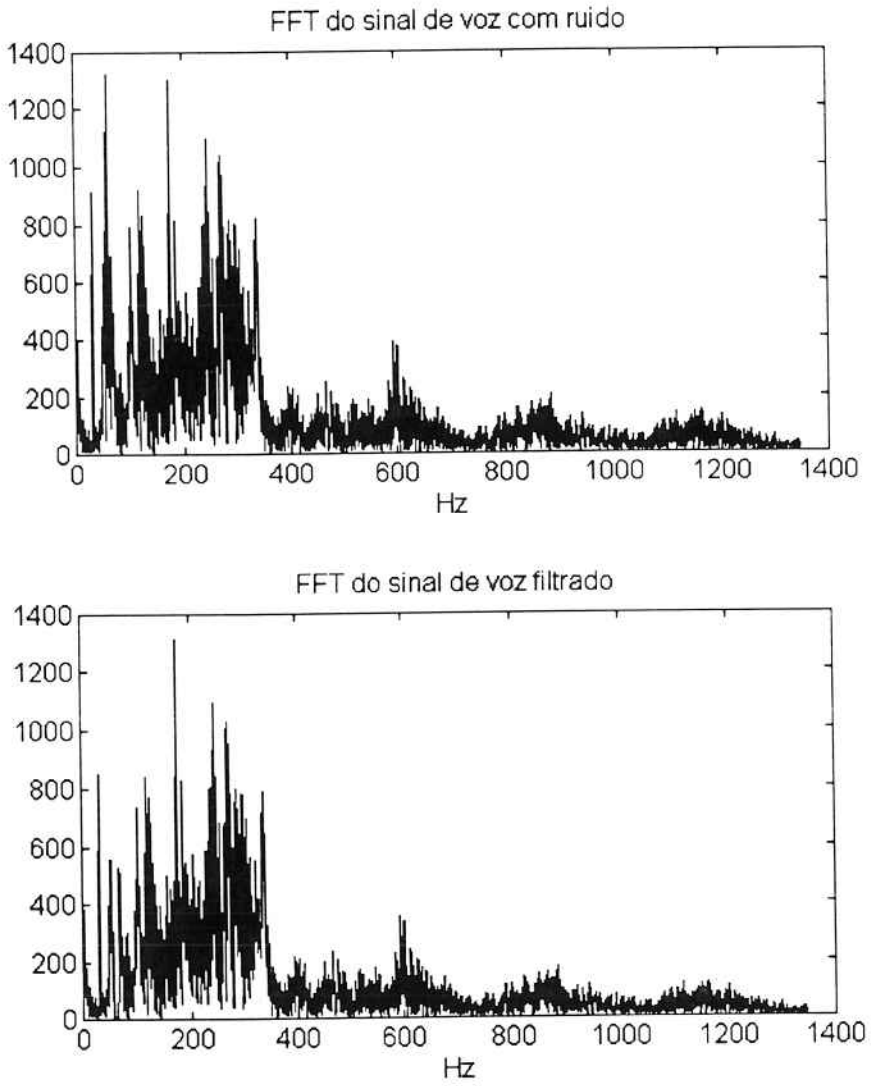

Figura 3.9: Comparação entre os sinais de voz antes e após a filtragem.

Na figura 3.10 pode-se observar o detalhamento da figura 3.9 em torno da freqüência de 60 Hz. Observa-se que o corte das componentes de 60 Hz no sinal de voz é praticamente total, dando ao sinal de saída do filtro o aspecto que que uma parte de seu espectro foi simplesmente eliminada. Observa-se também como uma faixa bem estreita foi eliminada de forma bem abrupta, caracterizando um filtro *Notch*. Observando em detalhe o espectro do sinal filtrado vê-se que outras freqüências que compõem o ruído poderiam também ser eliminadas, mas sem dúvida alguma a componente em 60 Hz representa a mais "comprometedora" do sinal de voz.

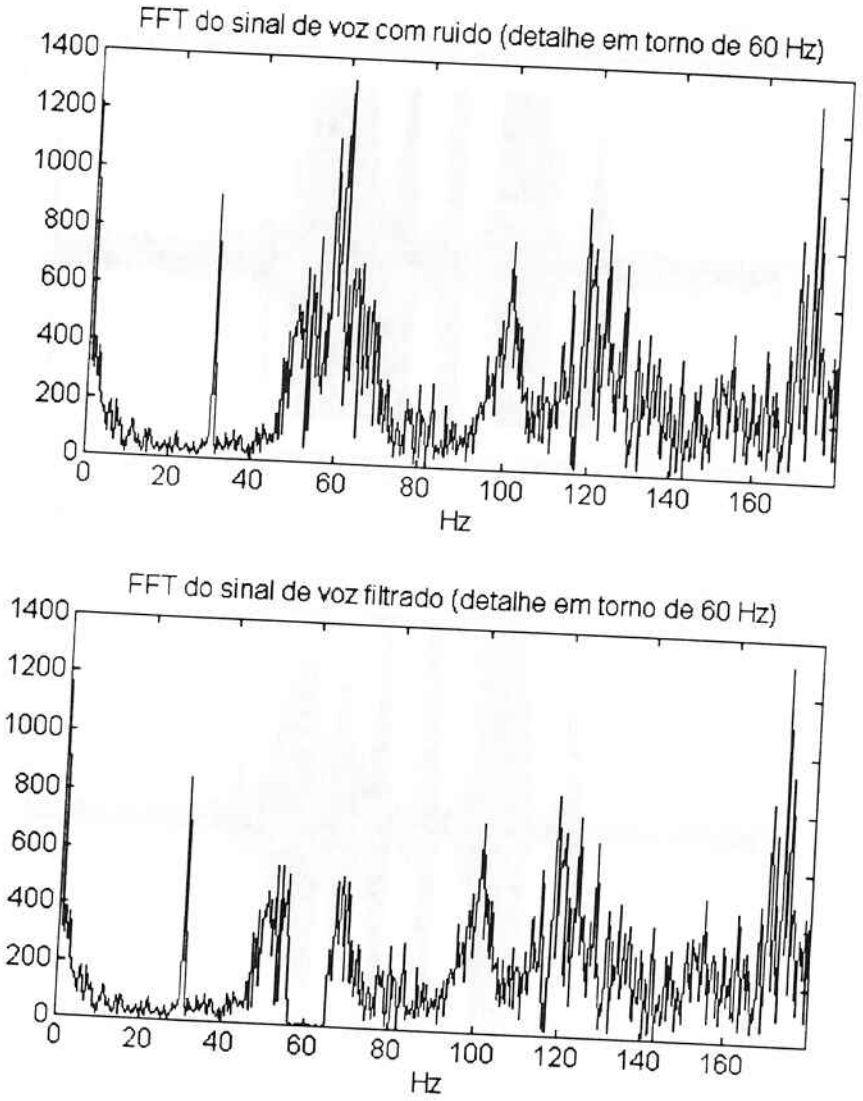

Figura 3.10: Detalhe da filtragem do sinal de voz.

Na figura 3.9 a componente em 60 Hz do ruído está com alto valor pois a FFT foi executada em todo o comprimento da amostra de voz (figura 3.11). Sendo assim, como grande parte do sinal contém apenas ruído, a partipação do mesmo no FFT é bem acentuada. Isto foi propositalmente feito de forma a destacar o funcionamento do filtro.

Na figura 3.11 é mostrado o sinal de voz no domínio do tempo, antes a após a filtragem. Devido a saturação do sinal de entrada, os sinais de entrada e saída apresentam envoltórias ligeiramente diferentes. No entanto nota-se visualmente que não houve grandes alterações no aspecto sonoro do sinal. Mas, sem dúvida, o aspecto mais interessante a ser observado na figura 3.11 é a diminuição do ruído nos instantes de silêncio do locutor, no início e no fim do tempo de aquisição do sinal. Vê-se claramente a influência do filtro sobre a intensidade do ruído remanescente da filtragem.

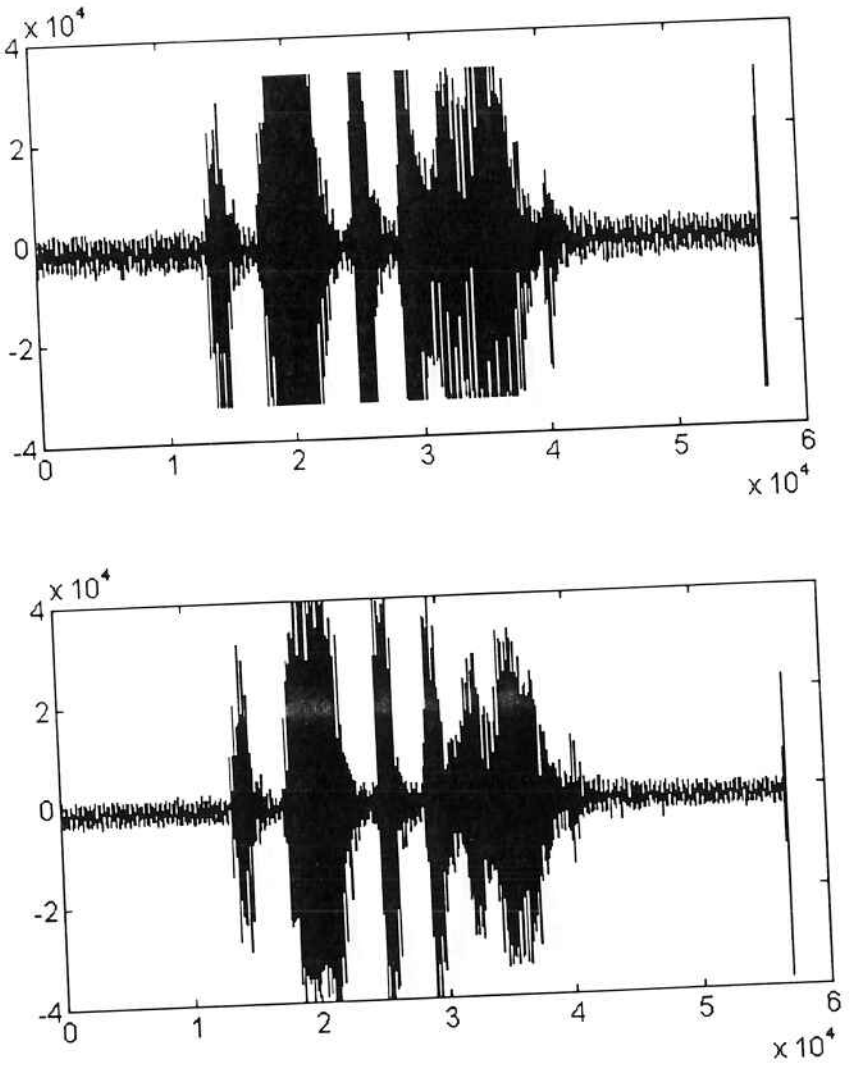

Figura 3.11: Sinal no domínio do tempo, antes e após a filtragem.

4. Conclusões

Deste trabalho pode-se tirar uma série de conclusões importantes, tanto no que que diz respeito a análise de ruídos sonoros, quanto ao projeto de filtros digitais. Dentre estas conclusões destacam-se:

- O ruído sonoro introduzido pelo ventilador apresenta uma componente principal a 60 Hz, devido provavelmente ao fato do ventilador girar a esta velocidade.
- A eliminação desta componente do ruído não o elimina completamente, mas o reduz consideravelmente.

- O projeto de um filtro *Notch* geralmente é mais complexo que suas considerações teóricas (expressão 3.1), havendo necessidade de busca algorítmica dos parâmetros de otimização destes filtros.

- Um filtro digital projetado através da aproximação eliptica pode ter um comportamento prático bem distinto da curva de transferência teórica levantada, havendo necessidade de se efetuar sempre o teste do filtro projetado e eventual ajuste de parâmetros.

- O programa desenvolvido para o projeto de filtros elipticos (ELLIP) pode ser uma ferramenta de grande utilidade na tarefa de projeto de filtros diversos.

- O filtro projetado se mostrou além das espectativas, reproduzindo o comportamento esperado.

A utilização do ELLIP pode ser didaticamente explorada, já que permite o projeto, de forma rápida, de filtros que possam ser testados para um melhor aprendizado da própria metodologia de projeto. A geração de um relatório sobre o filtro gerado e de sua curva de transferência apresenta também grande valor didático.

Por fim convém ressaltar que o estudo das técnicas de projetos de filtros digitais possibilitou conhecer aspectos práticos relacionados a esta tarefa e que não se encontram suficientementes explicitados na literatura relacionada.

5. Bibliografia

[1] Rao, D.V.B.; Kung, S-Y., *"Adaptative notch filtering for the retrieval of sinusoids in noise"*, IEEE Trans. Acoust. Speech, Signal Proc., Vol ASSP-32, pp. 791-802, Aug. 1984.

[2] Nehorai, A. *"A minimal parameter adaptative notch filter with constrained poles and zeros"*, IEEE Trans. Acoust. Speech, Signal Proc., Vol ASSP-33, pp. 983-996, Aug. 1985.

[3] Pei, S-C.; Tseng, C-C. *"Complex Adaptative IRR Notch Filter Algorithm and Its Applications"*, IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 41, No 2, pp 158-163, Feb. 1994.

[4] Antoniou, A. *"Digital filters:analysis and design"*, McGraw-Hill Series in Electrical Engineering, 1979.

Anexo 1: Relatório do filtro gerado pelo ELLIP

```
---------------- Filtro Digital projetado com Ellip ----------------
      (desen. por Fabio Luis Romao   e-mail:romao@lsi.usp.br)

Parametros do filtro:
          tipo          : corta-banda estreita
          Ripple pass.  : 1 dB
          Min atenuacao: 45 dB
          Freq amostr.  : 11025 Hz
          Freq Op1      : 58 Hz
          Freq Oa1      : 60 Hz
          Freq Op2      : 62 Hz
          Freq Oa2      : 60 Hz

Forma da expressao:
                                  A0 + A1*z^(-1) + A2*z^(-2)
          H(z)= H0 * ||  ----------------------------
                     j   B0 + B1*z^(-1) + B2*z^(-2)
 ( || = produtorio)


 Coeficientes em Z:

 H0: 0.003517
 ----j = 0 ----
 A0: 64.492241  A1: 77.279622  A2: 64.492241
 B0: 1.730153  B1: -3.429419  B2: 1.699613

 ----j = 1 ----
 A0: 12.961504  A1: -25.781854  A2: 12.961504
 B0: 401.891819  B1: 733.998073  B2: 338.965438

 ----j = 2 ----
 A0: 124.103749  A1: 226.708052  A2: 124.103749
 B0: 3.145525  B1: -6.2626  B2: 3.119185

 ----j = 3 ----
 A0: 5.377511  A1: -10.744423  A2: 5.377511
 B0: 213.751467  B1: 398.658984  B2: 197.434828

 ----j = 4 ----
 A0: 150.890499  A1: 284.420086  A2: 150.890499
 B0: 3.742497  B1: -7.471092  B2: 3.732173

 ----j = 5 ----
 A0: 4.341628  A1: -8.677656  A2: 4.341628
 B0: 175.943471  B1: 332.441457  B2: 171.443748

 ----j = 6 ----
 A0: 159.427845  A1: 302.487912  A2: 159.427845
 B0: 3.927383  B1: -7.847462  B2: 3.924224

 ----j = 7 ----
 A0: 4.093118  A1: -8.181541  A2: 4.093118
 B0: 166.430143  B1: 315.912778  B2: 165.181705

 ----j = 8 ----
 A0: 161.882234  A1: 307.660857  A2: 161.882234
 B0: 3.980007  B1: -7.954812  B2: 3.979121

 ----j = 9 ----
 A0: 4.027021  A1: -8.049569  A2: 4.027021
 B0: 163.866764  B1: 311.478774  B2: 163.52595

 ----j = 10 ----
 A0: 162.524566  A1: 309.013207  A2: 162.524566
 B0: 3.993694  B1: -7.982842  B2: 3.993511

 ----j = 11 ----
 A0: 4.010085  A1: -8.015754  A2: 4.010085
 B0: 163.190022  B1: 310.340185  B2: 163.120209


 coeficientes da implementacao paralela:

 H0: 0.003515
 ----j = 0 ----
```

```
A0: 0.011942  A1: 0.023768  A2: 0.011942
B0: 3.037247e-05  B1: -5.763365e-05  B2: 2.745629e-05

----j = 1 ----
A0: 2.886646e-05  A1: -5.772947e-05  A2: 2.886646e-05
B0: 0.000336  B1: 0.000364  B2: 0.000144

----j = 2 ----
A0: 0.002094  A1: 0.004072  A2: 0.002094
B0: 2.908936e-05  B1: -5.7702e-05  B2: 2.867105e-05

----j = 3 ----
A0: 2.887059e-05  A1: -5.772123e-05  A2: 2.887059e-05
B0: 0.00078  B1: 0.001352  B2: 0.000688

----j = 4 ----
A0: 0.001378  A1: 0.00264  A2: 0.001378
B0: 2.892444e-05  B1: -5.771064e-05  B2: 2.882733e-05

----j = 5 ----
A0: 2.887323e-05  A1: -5.771594e-05  A2: 2.887323e-05
B0: 0.001045  B1: 0.001945  B2: 0.001015

----j = 6 ----
A0: 0.001228  A1: 0.002341  A2: 0.001228
B0: 2.888776e-05  B1: -5.771255e-05  B2: 2.886208e-05

----j = 7 ----
A0: 2.887418e-05  A1: -5.771404e-05  A2: 2.887418e-05
B0: 0.001137  B1: 0.00215  B2: 0.001128

----j = 8 ----
A0: 0.00119  A1: 0.002265  A2: 0.00119
B0: 2.887813e-05  B1: -5.771305e-05  B2: 2.887122e-05

----j = 9 ----
A0: 2.887446e-05  A1: -5.771347e-05  A2: 2.887446e-05
B0: 0.001164  B1: 0.002211  B2: 0.001162

----j = 10 ----
A0: 0.00118  A1: 0.002245  A2: 0.00118
B0: 2.887532e-05  B1: -5.771318e-05  B2: 2.887391e-05

----j = 11 ----
A0: 2.887454e-05  A1: -5.771332e-05  A2: 2.887454e-05
B0: 0.001171  B1: 0.002227  B2: 0.001171
```

Anexo 2: Listagem de ellip.cpp (código C++)

```
/*************************************************************************
*                                                                        *
*                  FABIO LUIS ROMAO - LSI/DEE/EPUSP                      *
*                 Tel: 818-5589 / 818-5270 / 570-5751                    *
*                      e-mail romao@lsi.usp.br                           *
*                                                                        *
*************************************************************************
*
*  Este programa pode realizar as seguintes tarefas:
*  - Calcular o coeficiente de H(z) para filtros elipticos, a partir dos
*  parametros de ganho e frequencia;
*  - Filtrar um sinal, com o filtro H(z) projetado;
*  - Tracar a curva de transferencia do filtro projetado.
*
*  1. Utilizacao do programa (exclusivamente em ambientes C++)
*    - acrescente arquivo ellip.cpp em seu "projeto" e utilize
*    o header ellip.h nos seus arquivos .cpp; ou
*    - utilize diretamente o ellip.cpp como header de seu programa.
*
*    1.1. Palavras-chaves:
*         - declaracao dos objetos:
*
*         elliptic_filter <nome do obj.>(<tipo>,Ap, Aa, Fa, Op1, Oa1, Op2, Oa2);
*
*         onde: <tipo>: tipo do filtro:
*                       "LP" ou "lp": low-pass;
*                       "HP" ou "hp": high-pass;
*                       "BP" ou "bp": band-pass;
*                       "BS" ou "bs": band-stop;
*                       "NO" ou "no": notch (sharp band-stop);
*                  Ap: ripple da banda de passagem (dB);
*                  Aa: minima atenuacao da banda de corte (dB);
*                  Fa: freq. de amostragem (Hz);
*                  Op1: freq., na banda de passagem, do inicio (ou termino)
*                       da banda de transicao (Hz);
*                  Oa1: freq., na banda de corte, do inicio (ou termino) da
*                       banda de transicao (Hz);
*                  Op2: idem, Op1, caso existam duas bandas de transicao (Hz);
*                  Oa2; idem, Oa1, caso existam duas bandas de transicao (Hz).
*
*           Obs: Op2 e Oa2 dever ser iguais a 0 caso o filtro seja de uma unica
*                banda de transicao (filtros passa-baixa e passa-alta).
*
*          - chamada das funcao filtragem:
*
*         <nome obj.>.filtering(<arquivo de entrada>,<arquivo de saida>);
*
*
*          - tracado da curva:
*
*          <nome obj.>.trace(<arquivo de saida>, dB);
*                                  (dB=1:grafico em db; dB=0:grafico decimal)
*
*          - impressao dos coeficientes de H(z):
*
*          <nome obj.>.print_coef_Hz(<arquivo de saida>);
*
*
*     2. Informacoes gerais
*
*        O programa ellip.cpp contem a descricao de uma classe que determina,
*    para cada objeto, um filtro especifico. O filtro digital projetado segue
*    a metodologia desenvolvida em "Antoniou, Andreas; Digital Filters: Analysis
*    and Design, MacGraw-Hill, 1979". Os filtros gerados sao do tipo eliptico, usando a
aproximacao bilinear com pre-warping.
*
*/

#include <complex.h>
#include <fstream.h>
#include <math.h>
#define MCOEF 50
#define MAXSOMATORIA 50;
#define MAXLENGHTSAMPLES 500000
#define MAXN 100;

struct Hns_coef  // estrutura para conter os coefic. de Hn(s) - normalizado
```

```
                //          H0      r     s^2+A0
{               // Hn(s)=------- * M  -------------
   int r;   //         1+D0   i=1  s^2+B1*s+B0
   double H0;//         \____/ Ds=1: indica que existe este termo
   double A0[MCOEF], B1[MCOEF], B0[MCOEF];
   double D0;
   int Ds;
};

struct Hz_coef // estr. para conter os coefic. e H(z)
{             //
   int r;       //            z^2*F2+z*F1+F0    r  z^2*Zn[2]+z*Zn[1]+Zn[0]
   double H0;   //  H(z)=H0* -------------- * M ------------------------
   int Dz;      //            z^2*G2+z*G1+G0  i=1 z^2*Zd[2]+z*Zd[1]+Zd[0]
   double Zn[3][MCOEF];  //   \_____________/
   double Zd[3][MCOEF];  //        Dz=1:     indica que existe o primeiro termo
   double F2, F1, F0, G2, G1, G0;
};


class elliptic_filter    // classe que define os objetos do tipo filtro eliptico
{
public:
   Hns_coef Hns;
   Hz_coef Hz[2];
   int NH;     // NH+1 indica numero de Hz(z) em paralelo
   float Ap, Aa, T, ws, Op1, Op2, Oa1, Oa2, Op2no, Oa2no; // parametros do filtro
   float WS, OP1, OA1, OP2, OA2;
   int filter_type;  // indica de o filtro e' LP, HP, BP, BS ou NO

   elliptic_filter(char *tipo, float App, float Aaa, float wss, float Op11, float Oa11, float
Op22, float Oa22);
   void calc_Hns_coef(double k);
   void calc_Hz_coef_BS(double B, double wo2, int nH);
   void filter_BS(double& k, double& B, double& wo2);
   void calc_Hz_coef_BP(double B, double wo2, int nH);
   void filter_BP(double& k, double& B, double& wo2);

   void calc_Hz_coef_LP(double lambda, int nH);
   void filter_LP(double& k, double& lambda);
   void calc_Hz_coef_HP(double lambda, int nH);
   void filter_HP(double& k, double& lambda);

   void filtering(char *ent, char *sai);
   void trace(char *sai, short int dB);
   void print_coef_Hz(char *coef);
};


elliptic_filter::elliptic_filter(float App, float Aaa, float wss, float Op11, float Oa11,
float Op22=-1, float Oa22=-1)
{
   Ap=App; Aa=Aaa; Op1=Op11; Op2=Op22; Oa1=Oa11; Oa2=Oa22; ws=wss;
};

void elliptic_filter::design(char *tipo)
{
   if((tipo[0]=='B' || tipo[0]=='b') && (tipo[1]=='S' || tipo[1]=='s'))
   {
      double k, B, wo2;
      filter_BS(k, B, wo2);
      calc_Hns_coef(k);
      calc_Hz_coef_BS(B, wo2);

      cout<<"\nk:"<<k<<" B:"<<B<<" wo2:"<<wo2;
   }
}

void elliptic_filter::filtering(char *ent, char *sai)
{
   ifstream cent(ent);
   ofstream csai(sai);
   float x[5000], y[5000];
   x[0]=x[1]=x[2]=x[3]=0;
   y[0]=y[1]=y[2]=y[3]=0;
   int j,l;
   short int x_in_y=0;

   for(int i=4; i<104; i++)
```

```
    {
       cent>>x[i];
       if(Hz.Dz) y[i]=((Hz.F2*(float)x[i]+Hz.F1*(float)x[i-1]+Hz.F0*(float)x[i-2]-
                Hz.G1*(float)y[n-1]-Hz.G0*(float)y[n-2])/Hz.G2);
       if(Hz.Dz) csai<<y[i]<<' ';
    };
    float *px=x;
    float *py=y;
    if(Hz.Dz) x_in_y=1;

    double SomaX, SomaY;
    for(l=1; l<=Hz.r; l++)
    {
       csai<<"\n----------------------\n";
       if(x_in_y) {px=y; py=x; x_in_y=0;}
       else {px=x; py=y; x_in_y=1;}
       for(i=4; i<104; i++)
       {
          SomaX=(float)px[i]*Hz.Zn[4][l]; SomaY=0;
          for(j=1; j<=4; j++)
          {
             SomaX+=(float)px[i-j]*Hz.Zn[4-j][l];
             SomaY+=(float)py[i-j]*Hz.Zd[4-j][l];
          }
          py[i]=(float)((SomaX-SomaY)/Hz.Zd[4][l]);
          csai<<py[i]<<' ';
       }
    }
    csai<<"\n######################\n";
    for(i=4; i<104; i++)
    {
       csai<<(py[i]*Hz.H0)<<' ';
    }

}




void elliptic_filter::calc_Hns_coef(double k)
{
    double kli=sqrt(1-k*k);
    double sq_kli=sqrt(kli);
    double q0=(0.5)*(1-sq_kli)/(1+sq_kli);
    double q=q0+2*pow(q0,5)+15*pow(q0,9)+150*pow(q0,13);
    double D=(pow(10,0.1*Aa)-1)/(pow(10,0.1*Ap)-1);
    int n=ceil(log10(16*D)/log10(1/q));
    double Lambda=(1/(2*(double)n))*log((pow(10,0.05*Ap)+1)/(pow(10,0.05*Ap)-1));

    double somaux=1000;
    int m, cont;
    double md;
    double soma1=0;
    m=0; cont=MAXSOMATORIA;
    while (fabs(somaux)>pow(10,-12) && cont--)
    {
       md=(double)m;
       somaux=pow((-1),md)*pow(q,md*(md+1))*sinh((2*md+1)*Lambda);
       soma1+=somaux;
       m++;
    }
    double soma2=0;
    somaux=100000;
    m=1; cont=MAXSOMATORIA;
    while (fabs(somaux)>pow(10,-12) && cont--)
    {
       md=(double)m;
       somaux=pow((-1),md)*pow(q,md*md)*cosh(2*md*Lambda);
       soma2+=somaux;
       m++;
    }

    double sigma0=fabs((2*pow(q,0.25)*soma1)/(1+2*soma2));
    double W=sqrt((1+k*sigma0*sigma0)*(1+(sigma0*sigma0)/k));
    int r=(int)(n/2);

    double Omegai, Omi2, mu, Vi;
    float B0sA0=1;
    double mul2=fmod(n,2)/2-0.5;
```

```
//    mu12=fmod((double)n,2)/2-1/2;
  for (int i=1; i<=r; i++)
  {
     mu=i+mu12;

     soma1=0; somaux=100000;
     m=0; cont=MAXSOMATORIA;
     while (fabs(somaux)>pow(10,-12) && cont--)
     {
        md=(double)m;
        somaux=pow(-1,md)*pow(q,md*(md+1))*sin(((2*md+1)*M_PI*mu)/(double)n);
        soma1+=somaux;
        m++;
     }
     soma2=0; somaux=100000;
     m=1; cont=MAXSOMATORIA;
     while (fabs(somaux)>pow(10,-12) && cont--)
     {
        md=(double)m;
        somaux=pow(-1,md)*pow(q,md*md)*cos((2*md*M_PI*mu)/(double)n);
        soma2+=somaux;
        m++;
     }

     Omegai=(2*pow(q,0.25)*soma1)/(1+2*soma2);
     Omi2=Omegai*Omegai;
     Vi=sqrt((1-k*Omi2)*(1-Omi2/k));

     Hns.A0[i]=(float)(1/Omi2);
     Hns.B0[i]=(float)((pow(sigma0*Vi,2)+Omi2*W*W)/pow(1+sigma0*sigma0*Omi2, 2));
     Hns.B1[i]=(float)((2*sigma0*Vi)/(1+sigma0*sigma0*Omi2));

     B0sA0=B0sA0*(Hns.B0[i]/Hns.A0[i]);

     cout<<"\nA0"<<i<<":"<<Hns.A0[i]<<" ";
     cout<<"B0"<<i<<":"<<Hns.B0[i]<<" ";
     cout<<"B1"<<i<<":"<<Hns.B1[i]<<" ";
  }

  if (fmod(n,2))
  {
     Hns.H0=(float)(sigma0*B0sA0);
     Hns.Ds=1; Hns.D0=(float)sigma0;
  } else
  {
     Hns.H0=(float)(pow(10,-0.05*Ap)*B0sA0);
     Hns.Ds=0;
  }
  Hns.r=r;
  cout<<"\nH0:"<<Hns.H0<<" r:"<<Hns.r<<" ";
  cout<<"\nn="<<n<<" ";
};

void elliptic_filter::calc_Hz_coef_BS(double B, double wo2)
{
  double T=2*M_PI/ws;
  int r=Hns.r;
  Hz.r=r;
  double A0, B0, B1, T2=T*T, N0N1T3;
  for (int i=1; i<=r; i++)
  {
     A0=(double)Hns.A0[i];
     B0=(double)Hns.B0[i];
     B1=(double)Hns.B1[i];
     double M0, M2, M4, N0, N1, N2, N3, N4;
     M0=A0*wo2*wo2;
     M2=(2*A0*wo2+B*B);
     M4=A0;
     N0=B0*wo2*wo2;
     N1=B*B1*wo2;
     N2=B*B+2*B0*wo2;
     N3=B*B1;
     N4=B0;

     Hz.Zn[0][i]=(float)(4*(M2*T2+4*M4)+M0*T2*T2);
     Hz.Zn[1][i]=(float)(4*(M0*T2*T2-16*M4));
     Hz.Zn[2][i]=(float)(2*(3*M0*T2*T2+4*(12*M4-M2*T2)));
     Hz.Zn[3][i]=(float)(4*(M0*T2*T2-16*M4));
     Hz.Zn[4][i]=(float)(M0*T2*T2+4*(M2*T2+4*M4));
     N0N1T3=N0*N1*T2*T;
     Hz.Zd[0][i]=(float)(2*(2*(N2*T2+2*(2*N4-N3*T))-N0N1T3));
```

```
      Hz.Zd[1][i]=(float)(4*(N0N1T3+4*(4*N4-N3*T)));
      Hz.Zd[2][i]=(float)(8*(12*N4-N2*T2));
      Hz.Zd[3][i]=(float)(4*(N0N1T3-4*(N3*T+4*N4)));
      Hz.Zd[4][i]=(float)(2*(N0N1T3+2*(N2*T2+2*(N3*T+2*N4))));

      cout<<"\nZn0-"<<i<<":"<<Hz.Zn[0][i];
      cout<<" Zn1-"<<i<<":"<<Hz.Zn[1][i];
      cout<<" Zn2-"<<i<<":"<<Hz.Zn[2][i];
      cout<<" Zn3-"<<i<<":"<<Hz.Zn[3][i];
      cout<<" Zn4-"<<i<<":"<<Hz.Zn[4][i];
      cout<<" Zd0-"<<i<<":"<<Hz.Zd[0][i];
      cout<<" Zd1-"<<i<<":"<<Hz.Zd[1][i];
      cout<<" Zd2-"<<i<<":"<<Hz.Zd[2][i];
      cout<<" Zd3-"<<i<<":"<<Hz.Zd[3][i];
      cout<<" Zd4-"<<i<<":"<<Hz.Zd[4][i];
   }
   if (Hns.Ds)
   {
      Hz.Dz=1;
      double wo2T2=wo2*T2;
      double sig=(double)Hns.D0;
      Hz.F0=(float)(wo2T2+4);
      Hz.F1=(float)(wo2T2-4);
      Hz.F2=(float)(wo2T2+4);
      Hz.G0=(float)((-2)*B*T+sig*(wo2T2+4));
      Hz.G1=(float)(2*sig*(wo2T2-4));
      Hz.G2=(float)(2*B*T+sig*(wo2T2+4));
      Hz.H0=Hns.H0;

      cout<<"\nHz.F0:"<<Hz.F0;
      cout<<" Hz.F1:"<<Hz.F1;
      cout<<" Hz.F2:"<<Hz.F2;
      cout<<" Hz.G0:"<<Hz.G0;
      cout<<" Hz.G1:"<<Hz.G1;
      cout<<" Hz.G2:"<<Hz.G2;
      cout<<" Hz.H0:"<<Hz.H0;

   } else Hz.Dz=0;
};


void elliptic_filter::filter_BS(double& k, double& B, double& wo2)
{
   T=(2*M_PI)/ws;

   double KA=tan(Op2*T/2)-tan(Op1*T/2);
   double KB=tan(Op1*T/2)*tan(Op2*T/2);
   double KC=tan(Oa1*T/2)*tan(Oa2*T/2);
   double K1=KA*tan(Oa1*T/2)/(KB-pow(tan(Oa1*T/2),2));
   double K2=KA*tan(Oa2*T/2)/(pow(tan(Oa2*T/2),2)-KB);

   double wp;
   if(KC<KB)
   {
      k=1/K1; wp=1/sqrt(K1); //wa=wp*K1;
   }
   else
   {
      k=1/K2; wp=1/sqrt(K2); //wa=wp*K2;
   }

   wo2=pow(2*sqrt(KB)/T,2);
   B=2*KA*wp/T;
}

void elliptic_filter::trace(char *sai)
{
   ofstream csai(sai);
   int end=(int)ws;
   float prod;
   for(int i=0; i<end; i++)
   {
      prod=1.0;
      for(int j=1; j<=Hz.r; j++)
      {

   prod*=((Hz.Zn[4][j]*pow(i,4)+Hz.Zn[3][j]*pow(i,3)+Hz.Zn[2][j]*pow(i,2)+Hz.Zn[1][j]*i+Hz.Zn[
0][j])/

   (Hz.Zd[4][j]*pow(i,4)+Hz.Zd[3][j]*pow(i,3)+Hz.Zd[2][j]*pow(i,2)+Hz.Zd[1][j]*i+Hz.Zd[0][j]))
;
```

```
        }
    }
    if(Hz.Dz) prod*=((Hz.F2*i*i+Hz.F1*i+Hz.F0)/(Hz.G2*i*i+Hz.G1*i+Hz.G0));
    prod*=Hz.H0;
    csai<<prod<<' ';
    csai.close();
}
```

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PEE/93-01 - Oscilador a HEMT - 10 GHz - FÁTIMA S. CORRERA, EDMAR CAMARGO

BT/PEE/93-02 - Representação Senoidal da Voz através dos Polos do Filtro Preditor - MARCELO B. JOAQUIM, NORMONDS ALENS

BT/PEE/93-03 - Blindagens por Grades Condutoras: Cálculo do Campo Próximo - LUIZ CEZAR TRINTINALIA, ANTONIO ROBERTO PANICALI

BT/PEE/93-04 - Sistema de Otimização e Controle de Produção em Minas de Pequeno e Médio Porte - TSEN CHUNG KANG, VITOR MARQUES PINTO LEITE

BT/PEE/94-01 - Determinação das Frases de Aplicação Forense para o projeto NESPER e Tese de Mestrado IME/94, com Base em Estudos Fonéticos - MARCONI DOS REIS BEZERRA, EUVALDO F. CABRAL JUNIOR

BT/PEE/94-02 - Implementação e Teste de uma Rede Neural Artificial do Tlpo KSON (Kohonen Self-Organizing Network) com Entradas Bidimensionais - MARCELO YASSUNORI MATUDA, EUVALDO F. CABRAL JR.

BT/PEE/94-03 - Transformada de Walsh e Haar Aplicadas no Processamento de Voz - ALEXANDRE AUGUSTO OTTATI NOGUEIRA, THIAGO ANTONIO GRANDI DE TOLOSA, EUVALDO F. CABRAL JÚNIOR

BT/PEE/94-04 - Aplicação de Redes Neurais ao Problema de Reconhecimento de Padrões por um Sonar Ativo - ALEXANDRE RIBEIRO MORRONE, CRISTINA COELHO DE ABREU, EDUARDO KOITI KIUKAWA, EUVALDO F. CABRAL JR.

BT/PEE/94-05 - Tudo que se Precisa Saber sobre a Prática da FFT - Transformada Rápida de Fourier (Inclui Software) - ROGÉRIO CASAGRANDE, EUVALDO F. CABRAL JR.

BT/PEE/94-06 - A Survey on Speech Enhancement Techniques of Interest to Speaker Recognition - CELSO S. KURASHIMA, EUVALDO F. CABRAL JR.

BT/PEE/94-07 - Identificação de Pulsos Decádicos em Linhas Telefônicas - ANTONIO P. TIMOSZCZUK, MÁRCIO A. MATHIAS, EUVALDO F. CABRAL JR.

BT/PEE/94-08 - Implementação e Teste de Filtros do Tipo Adaptativo e "Notch" para a Remoção de Interferência de 60 Hz em Sinais de Eletrocardiograma - FLÁVIO ANTÔNIO MENEGOLA, JOSÉ AUGUSTO DE MATTOS, JOSÉ GOMES G. FILHO, SIDNEY SILVA VIANA, EUVALDO F. CABRAL JR.

BT/PEE/94-09 - Compressão de Sinais de Voz utilizando Transformadas de Karhunen-Loève, Fourier e Hadamard - IVAN LUIS VIEIRA, LUIZ FERNANDO STEIN WETZEL, EUVALDO F. CABRAL JR.

BT/PEE/94-10 - "Ray Tracing" Paralelo - EDUARDO TOLEDO SANTOS, JOÃO ANTONIO ZUFFO

BT/PEE/94-11 - Implementação de uma Ferramenta Posicionador para "Gate-Arrays" Tipo Mar de Portas - JORGE W. PERLAZA PRADO, WILHELMUS A. M. VAN NOIJE

BT/PEE/94-12 - Tudo que se Precisa Saber Sobre a Teoria da FFT - Transformada Rápida de Fourier - FÁBIO LUÍS ROMÃO, REINALDO SILVEIRA, ROGÉRIO CASAGRANDE, EUVALDO CABRAL JR.