Universidade Federal de Minas Gerais Departamento de Ciência da Computação

Análise Numérica: Trabalho Prático II

Recuperação de Arquivos de Áudio Usando Splines Cúbicos

Autor:
Gustavo Campos Ferreira Guimarães

Matrícula:
2005041291

Documentação feita em LaTeX

19 de junho de 2007

Sumário

1 Introdução

1.1 Áudio Digital

Desde o aparecimento dos primeiros computadores de tamanho e preço reduzidos, compatíveis com o poder aquisitivo da população mediana, em especial dos países desenvolvidos, suas utilidades para o entretenimento começaram a ser exploradas. Jogos, vídeos, música e várias outras aplicações neste que é um dos ramos mais lucrativos do planeta possuem suas versões digitais, aplicadas de uma forma ou de outra a algum tipo de computador pessoal, dos tão comuns PC's e consoles de video-game até os novatos Media Centers e Home Theaters.

O áudio digital foi um dos pioneiros nessa nova onda de entretenimento digital. Primeiramente com os primeiros e rudimentares sons sendo gerados por 'tiques' em auto-falantes simplórios. Os primeirs sons de computador comumente eram loops do tipo *for* ou *while*, gerando em intervalos de tempo certos valores numéricos, que eram convertidos em sinais elétricos e enviados aos auto-falantes. Um método bastante trabalhoso para qualquer um que se aventurasse no áudio digital.

Com o tempo, sobretudo após os computadores se tornarem mais robustos, as técnicas de amostragem de áudio surgiram e possibilitaram o boom do áudio digital, que culminou na década de 90 com o aparecimento do Compact Disc, que revolucionou a forma das pessoas ouvirem música em casa. Seguido pouco depois pelo MP3, que na verdade teve o efeito contrário, o CD foi o responsável por uma melhoria significativa na qualidade do áudio acessível ás pessoas, acostumadas ao discos de vinil, que, apesar de usarem áudio analógico, sofriam muito facilmente os efeitos das intempéries de mau uso e do tempo, já que exigiam o atrito entre as superfícies do disco e da agulha.

1.2 O Problema

Apesar da aparente melhoria na qualidade de som, o áudio digital mascara uma muitas vezes considerável perda nessa qualidade. Arquivos de áudio digital na verdade são amostragens dos sons reais, pontos discretos de uma onda sonora contínua.

Basicamente, o processo de amostragem de som consiste em capturar as ondas sonoras com um dispositivo de entrada, usualmente um microfone, e em certos intervalos de tempo gravar o valor numérico da amplitude da onda, numa determinada escala. No fim, tem-se uma amostragem de pontos que se aproximam da onda original. Quanto mais pontos, ou seja, quanto maior a taxa de amostragem, melhor a qualidade do som, em teoria.

Em geral uma taxa de 44100Hz (44100 pontos por segundo) é a utilizada na indústria fonográfica para realizar a amostragem de som digital. Essa é a qualidade padrão dos CD's de áudio e dos arquivos de MP3 (a perda de qualidade destes se deve a outros fatores, que fogem do escopo desta análise). Isso gera um som bastante fiel, indiferenciável para a maioria dos ouvidos humanos (há sempre quem diga que pode diferenciar o som original do digital).

Apesar de o áudio digital gerado não ser 100% fiel à onda sonora original, a qualidade obtida nos CD's é bem superior, em geral, á obtida em discos de vinil (existem discos especiais, mais resistentes a risco, e uma nova tecnologia de leitura de discos de vinil que não necessita do contato entre o dispositivo de leitura e o disco), já que a leitura é feita através de um laser, que não toca a superfície do disco, como fazia-se com as antigas vitrolas, evitando assim ruídos devido ao atrito.

O problema real nasce quando por algum motivo, a taxa de amostragem do arquivo de áudio é inferior à um certo valor (aproximadamente 90% dos 44100Hz). A partir daí a degradação do áudio começa a ser notável pelas pessoas comuns, o que gera insatisfação de várias formas. Há ainda aqueles sedentos pelo áudio perfeito, chamados de audiófilos, que estão sempre em busca de áudio de maior pureza e fidelidade. Para esses casos foi desenvolvida a técnica da interpolação de áudio.

A maioria dos CD-players de qualidade hoje possui em seu hardware um chip que faz a interpolação do áudio de CD's e arquivos de áudio digital em geral. Não tive acesso às técnicas utilizadas por esses aparelhos, por isso resolvi implementar minha própria técnica, como forma de experimentar o poder da interpolação por *Splines Cúbicos*, e avaliar o resultado criticamente.

2 Abordagem de Solução

A idéia básica por trás da interpolação de áudio é bastante simples: utilizar técnicas de interpolação de curvas para, de posse dos pontos amostrados da onda original, tentar aproximar outros pontos não presentes

na amostragem, de forma a compor uma nova onda, mais próxima da onda original, tentando assim melhorar a fidelidade do som.

É possível usar várias técnicas de interpolação para isso, que não é uma tarefa fácil. Ondas sonoras possuem uma quantidade elevadíssima de picos e vales, o que diminui consideravelmente a eficiência de qualquer algoritmo de interpolação, afinal normalmente esses algoritmos em geral usam um processo de *ligar os pontos* para poder conectar os pontos de amostragem e formar uma curva.

Escolhi o método de splines cúbicos pois sua característica principal é criar curvas mais suaves, partindo de um número maior de pontos de amostragem. A minha intenção com isso era conseguir tentar recuperar os picos da onda sonora. Para isso escolhi a implementação de splines extrapolados, pois sua característica de extrapolação poderia ser útil para obter esses picos e vales.

2.1 Splines Cúbicos

Os *splines cúbicos* são uma técnica de interpolação que consiste em utilizar uma série de polinômios interpoladores cúbicos para aproximar os pontos interpolados, a característica principal desta técnica é gerar curvas bastante suaves, com um relativamente baixo custo computacional. os *splines* são o algoritmo padrão de interpolação de diversos softwares matemáticos, a citar o *MatLab* e o usado neste trabalho: *GNUplot*.

As características dos *splines* advêm das suas condições de existência, basicamente duas:

- Dois *splines* consecutivos devem, obrigatoriamente, passar pelo mesmo ponto
- As derivadas de dois *splines* consecutivos devem ser iguais

Dessa forma garante-se a continuidade dos *splines* bem como a máxima aproximação de suas inclinações e curvaturas. Existem vários tipos de *splines*, dentre os quais foram estudados os naturais e extrapolados. Para esta implementação, optei pelos *splines extrapolados* na esperança de que a geração de valores ligeiramente excessivos nas bordas me ajudasse a conseguir melhores aproximações para os picos e vales das ondas sonoras. Certamente o ideal seria comparar vários métodos de interpolação, não apenas *splines*, nem tão somente apenas *splines extrapolados*, mas as limitações de tempo inerentes à condição de estudante impediram tal extrapolação.

3 Modelagem Matemática

Sendo um método que não envolve matemática avançada, a interpolação não requer uma modelagem matemática muito complexa. Descreverei brevemente, quase que em citação do livro didático adotado em sala de aula, a técnica usada especificamente pelos *splines extrapolados* para obter suas aproximações. Excluindo-se as provas envolvidas, vamos às fórmulas:

Um *spline cúbico* é essencialmente um polinômio da forma:

$$\boxed{s_i = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, i = 0, 1, 2, ..., n - 1}$$

Lembrando que os valores dos *splines* nos extremos de cada intervalo devem ser iguais, bem como suas derivadas primeira e segunda nesses mesmos pontos. Partindo desse pressuposto obtemos as fórmulas para os coeficientes *a,b,c* e *d* de cada *spline*:

$$\boxed{a_i = \frac{s^n_{i+1}(x_{i+1}) - s^n_i(x_i)}{6h_i}, i = 0, 1, 2, ..., n - 1}$$

$$\boxed{b_i = \frac{s^n_i(x_i)}{2}, i = 0, 1, 2, ..., n - 1}$$

$$\boxed{c_i = \Delta y_i - \frac{s^n_{i+1}(x_{i+1}) + 2s^n_i(x_i)}{6} h_i, i = 0, 1, 2, ..., n - 1}$$

$$\boxed{d_i = y_i, i = 0, 1, 2, ..., n - 1}$$

onde $\boxed{h_i = (x_{i+1}) - x_i}$, e $\boxed{\Delta y_i = \frac{y_{i+1} - y_i}{h_i}}$

Que nos dá um sistema linear subdeterminado com quatro incógnitas para cada spline, o que é um problema, visto que com isso falta-nos duas equações para poder determinar completamente o sistema.

A estratégia utilizada no *spline cúbico extrapolado* é igualar as derivadas terceiras dos extremos dos *splines*, eliminando assim duas variáveis e possibilitando sua solução. Como o sistema linear resultante é relativamente simples, só possuindo a diagonal principal e as sub-diagonais imediatamente inferior e superior a ela, o esforço computacional necessário para sua solução é relativamente pequeno, se comparado à solução de sistemas lineares convencionais equivalentes, tornando viável a solução de

4 Implementação

A implementação so método supracitado foi feita utilizando Fortran90, por ser mais flexível e eficiente que o Fortran77. Além disso, os compiladores de Fortran atuais possuem um suporte parcial ao Fortran77, podendo acarretar em problemas. O código foi compilado tanto usando o compilador livre *gfortran*, parte do pacote de compiladores livres *gcc*, quanto na plataforma fechada *Intel Fortran Compiler*, distribuida gratuitamente para uso não comercial.

Não posso avaliar com precisão o desempenho de ambos os compiladores. O compilador da Intel gerou um código ligeiramente mais eficiente na maiora dos casos, mas o *gfortran* foi superior em algumas ocasiões, sobretudo na maior massa de dados que eu dispunha, com cerca de 800Kb.

A seguir seguem-se os detalhes da implementação.

4.1 Decisões de Implementação

Devido a alguns problemas encontrados, algumas decisões precisaram ser tomadas para possibilitar a implementação do programa. Abaixo defino as decisões que considero mais relevantes:

4.1.1 Scripts Auxiliares

Para conseguir manipular as massas de dados, foi necessária a criação de diversos scripts, de forma a automatizar o processo de geração, compilação e plotagem dos dados. São scripts relativamente simples, usando linguagem bash e os softwares gnuplot e sox, comentados à frente. O principal deles é o script *geradados.sh* que consiste no código a seguir.

Uma das funções do script é "degradar"um amostra de áudio de 44100hz, gerando versões com menor amostragem, de forma que eu possa aplicar a interpolação nas amostras degradadas e avaliar sua aproximação à amostra original.

```
#!/bin/bash

# Este script gera as versoes degradadas dos arquivos de som, para isso
# os arquivos sao renomeados para identificar as suas sample-rate e entao
# e usado o software 'sox' para fazer a conversao

freq0=44100
freq1=22050
freq2=11025

cd ../samples/original

for arquivo in *.wav
do

        # gera wav's de diversos sample rates
        nome0=`echo $arquivo |cut -d '.' -f 1`"-$freq0.wav"
        nome1=`echo $arquivo |cut -d '.' -f 1`"-$freq1.wav"
        nome2=`echo $arquivo |cut -d '.' -f 1`"-$freq2.wav"

        cp $arquivo "../$nome0"
        sox "../$nome0" -r $freq1 "../$nome1"
        sox "../$nome0" -r $freq2 "../$nome2"

        # gera arquivos de dados
        nome3=`echo $nome0 |cut -d '.' -f 1`".dat"
        nome4=`echo $nome1 |cut -d '.' -f 1`".dat"
        nome5=`echo $nome2 |cut -d '.' -f 1`".dat"
        sox "../$nome0" -t dat "../../dados/$nome3"
        sox "../$nome1" -t dat "../../dados/$nome4"
        sox "../$nome2" -t dat "../../dados/$nome5"
done

cd "../../dados"

# conta as linhas dos arquivos e prepara os arquivos para input
for arquivo in *.dat
do
        sed -e '1d' "$arquivo" > "$arquivo.temp"
        cat "$arquivo.temp" | wc -l > "$arquivo"
        cat "$arquivo.temp" >> "$arquivo"
        rm *.temp
done


# reconstitui os originais
for arquivo in *-44100.dat
do
        sed -e '1d' "$arquivo" > "$arquivo.temp"
        cp "$arquivo.temp" "$arquivo"
done

# gera dados do seno

cd ..
./seno
mv "./seno.dat" "./dados/."
cd dados
```

```
cp "seno.dat" "seno.temp"
cat "seno.temp"| wc -l > "seno.dat"
cat "seno.temp" >> "seno.dat"
rm "seno.temp"
```

4.1.2 Alocação Estática

A princípio, como pode ser visto no código comentado, utilizei a estrutura `ALLOCATABLE` do Fortran, que permite alocar dinamicamente memória. Isso me possibilitava ler primeiramente do arquivo a quantidade de linhas a serem lidas, e em seguida alocar a quantidade de memória exata para a leitura, usando o comando `ALLOCATE`. Infelizmente a quantidade de problemas ocorridos, em especial devido a acessos indevidos à memória, me fez optar pela forma clássica de alocação estática de arrays, estipulei então um valor teto de 40000 espaços, suficiente para analisar alguns segundos de áudio.

4.1.3 Argumentos em Linha de Comando

Como faria testes com arquivos diferentes várias vezes, e pretendia automatizar o processo através de scripts, decidi passar os arquivos de entrada e saída como parâmetros de linha e comando. Isso é facilmente obtido através do uso da função `GETARG`, em conjunto com a função `IARGC`.

4.1.4 Seno

Para melhor ilustrar o funcionamento do programa, decidi rodá-lo sobre uma função simples para que os resultados fossem melhor visualizados. Escolhi a função $\sin 30x$, pois possui uma periodicidade razoável e é regular o suficiente para deixar os gráficos plotados claros e visíveis. Para gerar os pontos, implementei um pequeno programa em fortran:

```
program seno

    ! gera pontos de uma senoide para comparacao

    character (len=8) :: outfile = "seno.dat"
    real*8 i,y

            open(unit=11,file=outfile,status='replace',action='write')

    do i=0,2,0.05
        y = sin(30*i)
        write(11,100) i,y
    enddo

    100 format (f16.13,2x,f16.13)

    close(11)
end program seno
```

4.2 Softwares Auxiliares

Dois softwares de terceiros foram necessários para a implementação do projeto (estou excluindo obviamente editores de texto, compiladores e editores latex). Estes e suas funções são descritos abaixo:

4.2.1 SoX

O Sox é um projeto de software livre para conversão de arquivos de áudio digital. Fiquei muito satisfeito ao descobrir que ele é poderoso o suficiente para, a partir de um arquivo de áudio, gerar um arquivo de dados consistindo da amplitude da onda pelo tempo, extamente o que eu precisava para extrair dados numéricos de arquivos de áudio. Como também pode fazer o caminho contrário, o Sox me possibilitou verificar na prática, de forma auditiva, a eficiência do meu método.

4.2.2 GNUPlot

O GNUplot é um software livre renomado para plotagem de dados matemáticos tanto em 2d quanto em 3d. Ele é capaz de gerar diversos tipos de imagens e arquivos, a partir de dados experimentais, massas de dados numéricos e funções predefinidas. É ainda capaz de utilizar vários métodos numéricos, entre eles a própria interpolação por *splines cúbicos*. Todos os gráficos usados foram feitos com ajuda do GNUplot.

4.3 Formatos de Dados

Formatos de dados bastante simples foram usados. Como amostragem, escolhi alguns arquivos WAV, que tratam-se de áudio digital sem compressão, para fazer os testes. A princípio havia tentado com pequenos trechos de músicas, mas tendo em vista que a quantidade de dados contida era imensa, preferi optar por *samples* de instrumentos, como notas únicas de bateria e teclado. Como o resultado é bastante similar para todas, estou incluindo apenas dois exemplos nesta documentação.

4.3.1 Entrada e Saída de Dados

Tanto a entrada como a saída do programa consistem, basicamente, de arquivos de texto em duas colunas, sendo a primeira uma lista de abcissas, e a segunda uma lista de ordenadas correspondentes. Nestes arquivos, as abcissas equivalem ao tempo, enquanto as ordenadas à amplitude da onda. Este formato é comum tanto ao GNUplot quanto ao meu programa, e são gerados automaticamente pelo Sox a partir de arquivos de áudio.

No caso da entrada do meu programa, precisava de uma primeira linha que me dissesse quantos pontos devo ler. inclui e exclui essa linha conforme necessidade usando o comando `sed` do Linux, como pode ser observado em meus scripts.

4.4 Sintaxe de Execução

A sintaxe de execução do programa é bastante simples e consiste da forma:

```
cubicsplines arquivo-de-entrada arquivo-de-saida
```

Sendo os argumentos auto descritivos.

5 Resultados

Abaixo uma descrição e comentários dos resultados obtidos.

5.1 Testes Realizados

Escolhi arbitrariamente dois arquivos de áudio, baseado na diferença audível entre a amostra degradada. Primeiramente temos uma comparação dos resultados aplicados a uma função simples: $\sin 30x$, em seguida aplicadas aos arquivos de áudio. Vale lembrar que foi preciso aumentar consideravelmente o zoom nos casos dos arquivos de áudio para possibilitar alguma diferenciação visual.

Extrai as legendas incluidas automaticamente pelo GNUplot, pois confundiam a visualização, devido à impressão monocromática.

5.2 Resultados Obtidos

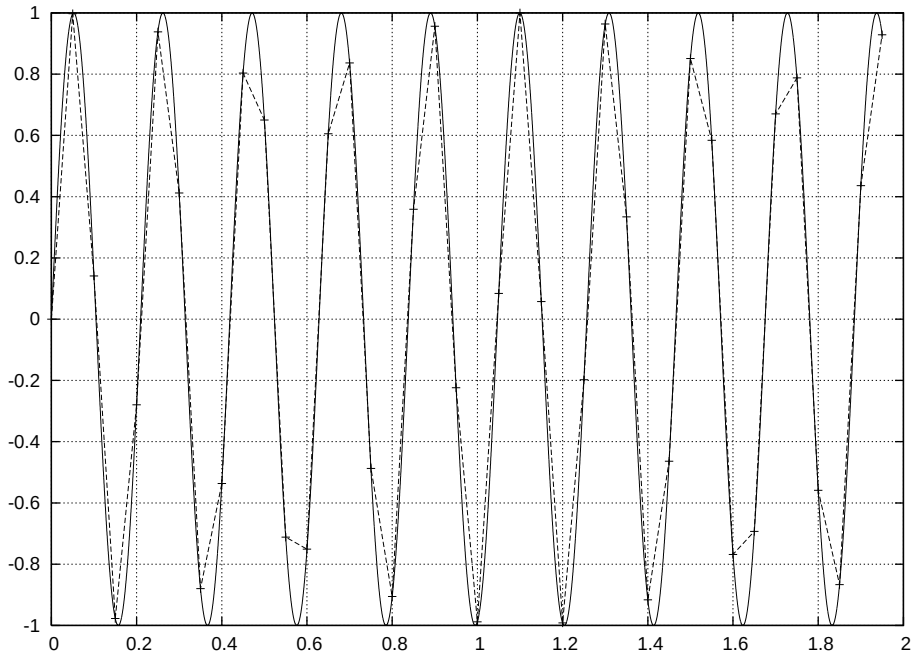

Figura 1: Neste gráfico, a linha pontilhada é a amostra degradada do seno, comparada à função seno original

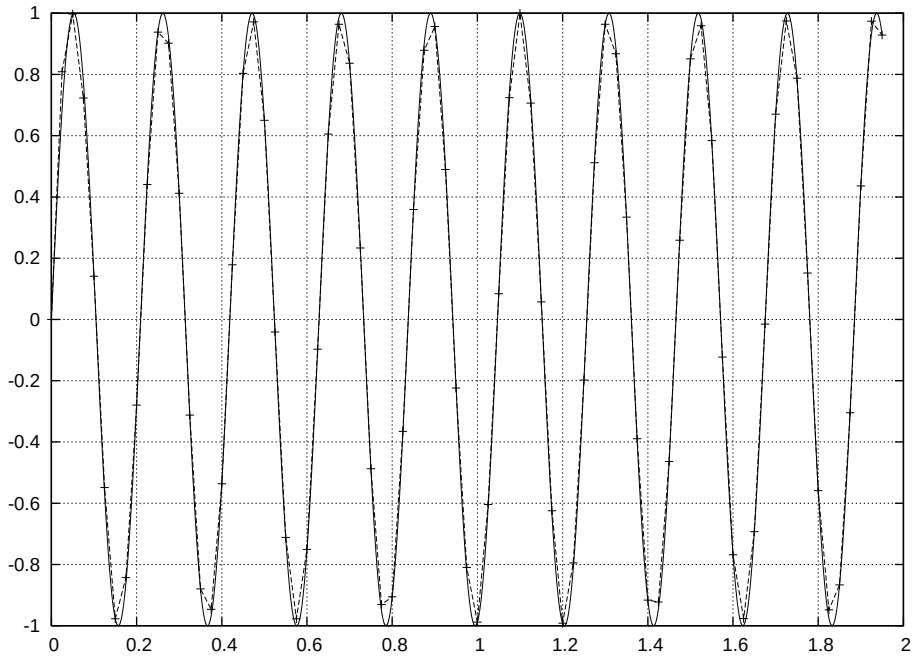

Figura 2: Neste gráfico, a linha pontilhada é a amostra interpolada do seno, comparada à função seno original

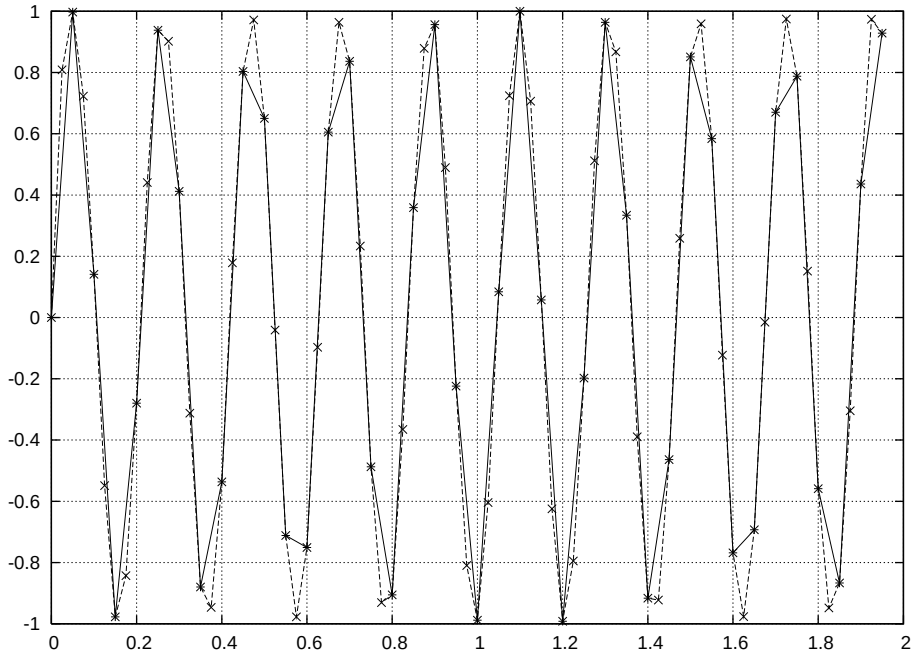

Figura 3: Neste gráfico, a linha pontilhada é a amostra interpolada do seno, comparada à amostra degradada

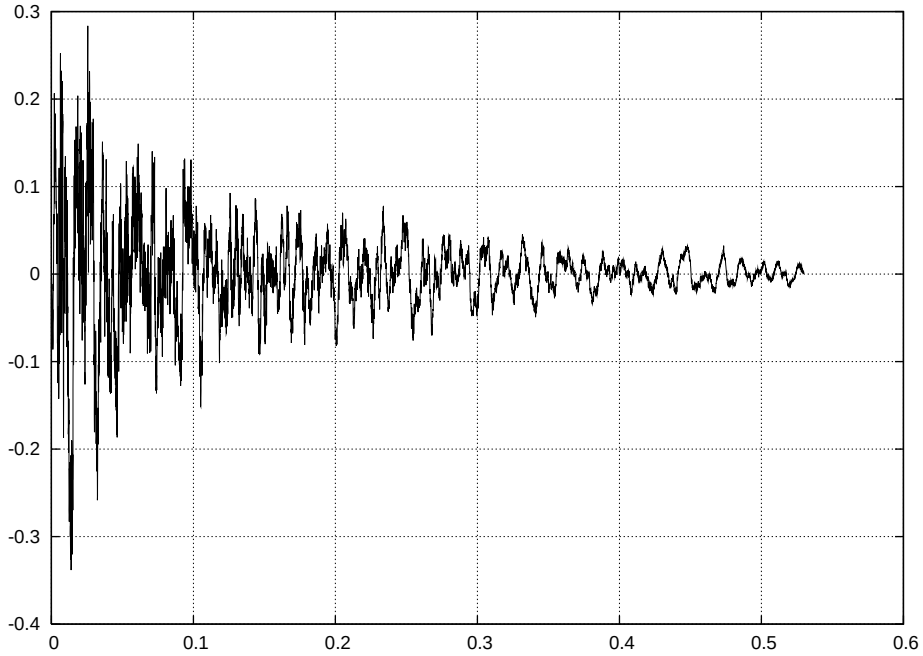

Figura 4: Amostra degradada de uma bateria a 22050Hz, é difícil notar alguma coisa, pois é muita informação

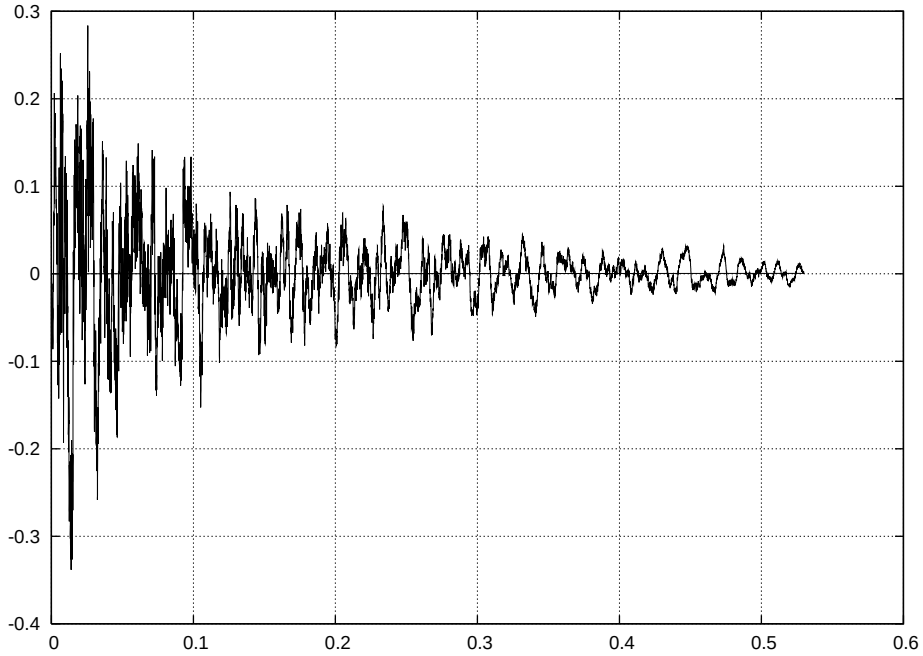

Figura 5: Amostra interpolada a partir do arquivo acima, aparentemente são iguais

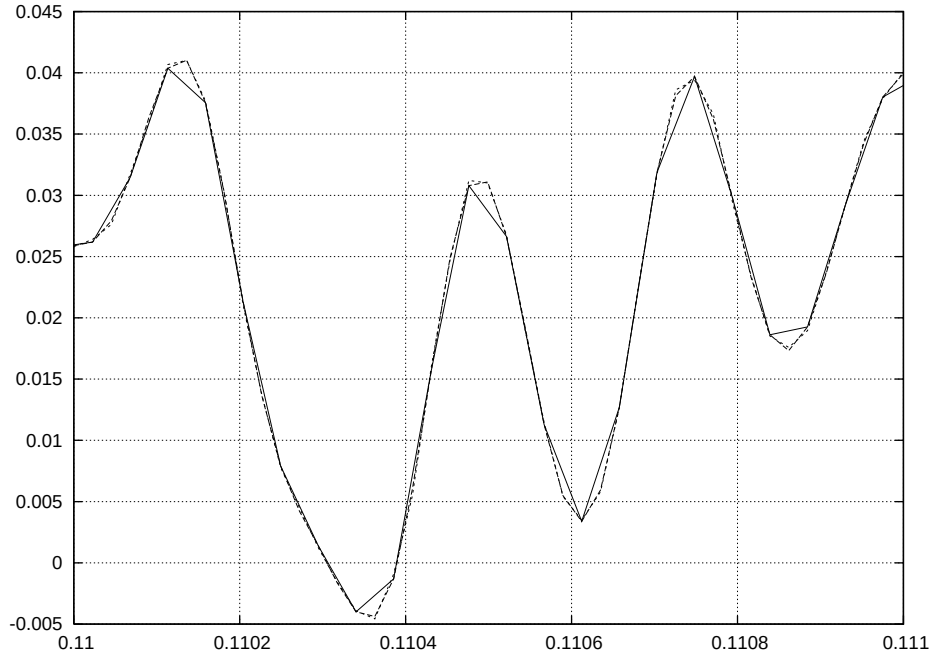

Figura 6: Fazendo um zoom vemos a diferença. A linha tracejada (interpolada) e a pontilhada (original) estão quase coincidentes, e relativamente espaçadas da amostra degradada (linha contínua)

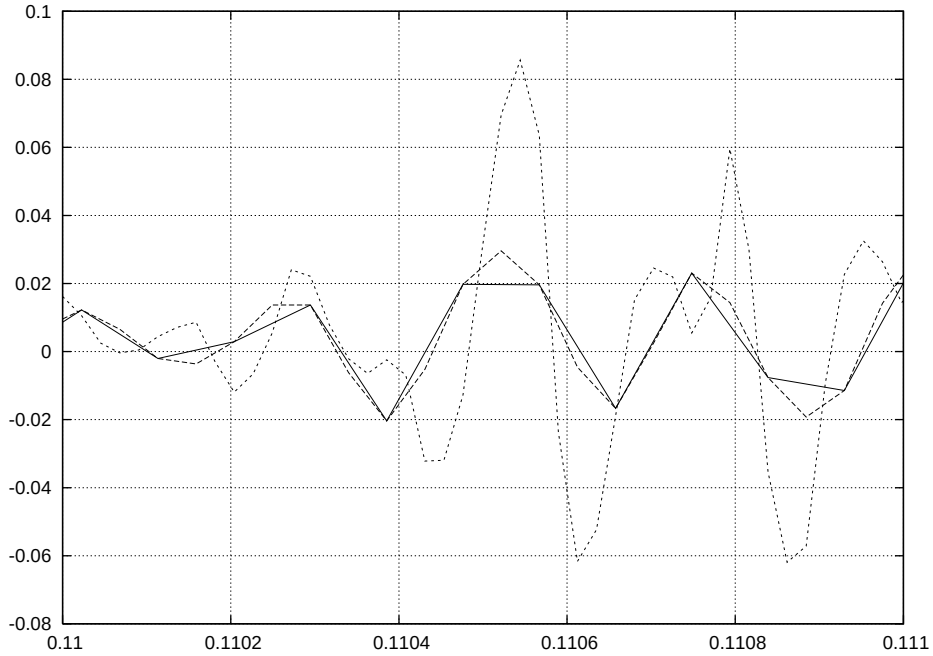

Figura 7: Amostra de um outro áudio, dessa vez mais degradada - a 11025Hz - nessa amostra tantos dados foram perdidos, que a melhoria da linha interpolada (tracejada) deixa muito a desejar à original (pontilhada) mesmo estando razoavelmente melhor que a versão degradada (linha contínua)

5.3 Comentários

Podemos notar claramente em todos os exemplos que a amostra interpolada possui uma fidelidade maior com a amostra real. Isso é claramente visível na função seno, mas podemos inferir também a mesma conclusão para as amostras sonoras, uma vez que, mesmo não estando de posse da onda original, a amostra reconstituida se aproximou da amostra não degradada.

6 Conclusão

Analisando visualmente os dados, podemos concluir que a interpolação por *splines cúbicos* dos arquivos de áudio foi um sucesso, e foi capaz de reconstituir uma parte considerável das ondas originais. Talvez se mais interpolações fossem feitas, uma sobre a outra, os resultados fossem melhores, mas infelizmente não houve tempo hábil para fazê-lo.

Infelizmente porém, fui incapaz de perceber qualquer diferença notável no som gerado pelas amostras degradadas e pelas amostras reconstituidas. Muito provavelmente este fato está conectado à incapacidade de se gerar os picos e vales mais extremos que foram perdidos na degradação da amostra de áudio. Tendo em vista que essa degradação chegou a 75% em alguns dos meus testes, era de se esperar que fosse difícil reconstituí-los.

O resultado final porém foi bastante satisfatório, sobretudo como forma de testar, avaliar e compreender melhor o método de interpolação por *splines cúbicos*, bem como de visualizar os resultados de sua aplicação numa situação real.

7 Código Fonte

Estão anexos os códigos fontes implementados em Fortran90

Referências

[1] *LaTeX: Alpha II - Programming Information: Audio Interpolation*
http://www.alpha-ii.com/Info/AudioInt.html

[2] *Gnuplot Homepage*
http://www.gnuplot.info

[3] *Sox Project*
http://sox.sourceforge.net/

[4] *Gnuplot: Not so frequently asked questions*
http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

[5] Frederico Ferreira Campos filho; *Algoritmos Numéricos*. LTC, Segunda Edição, 2006.