Universidade Tecnológica Federal do Paraná – UTFPR, Campus Curitiba - Centro
Alunas: Amanda Jury Nakamura RA: 2582686 Curso: Sistemas de Informação
Julia Kamilly de Oliveira RA: 2588005 Curso: Sistemas de Informação
Disciplina: ICSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu, Profa. Leyza Baldo Dorini, Prof. Daniel Fernando Pigatto

Relatório - Projeto 02

O segundo projeto do período teve como objetivo a criação de funções para manipulação de áudio. A primeira etapa foi a leitura individual das instruções por cada uma das integrantes, que fizeram anotações e considerações. Após essa etapa, a dupla se juntou e discutiu os pontos relevantes que cada uma havia separado, bem como metodologias e horários de encontros para trabalhar neste projeto.

A ideia inicial era realizar um projeto conjunto utilizando o Git para que as duas conseguissem trabalhar no código remotamente. Porém, como a habilidade das integrantes em relação ao Git não é muito profunda e o trabalho é curto e de certa forma simples, julgou-se desnecessário o uso da ferramenta. Tudo foi feito em conjunto através do Discord, onde eram trocadas ideias, discutidos problemas, e feita a revisão do código.

O código inteiro e o relatório foram feitos 100% sincronamente pela dupla ao compartilhar tela. O resultado do projeto é produto de ideias de ambas as participantes que iam compartilhando ideias, desenhos e "rabiscos" conforme o trabalho se desenvolvia.

Alguns colegas da turma compartilharam o software de áudio "Audacity" para testar e tornar possível a visualização das ondas, o que foi muito interessante para confirmar as alterações feitas por cada função.

O primeiro passo em todas as funções foi separar os dados mais importantes, destacados em tópicos dentro da descrição de cada função.

Função 01 - void mudaGanho (double* dados, int n_amostras, double ganho)

- Recebe: Vetor de dados, tamanho do vetor, ganho (dado *double*).
- Objetivo: Modificar o ganho do sinal.
- Saída: Função *void,* as alterações são feitas *in place*.

Esta função, após a identificação dos dados recebidos foi simples. O próprio texto continha o comando a ser realizado, bastando apenas interpretá-lo para código em C. Isso foi feito através de um *loop* que percorre o vetor dos dados e multiplica cada um deles pelo ganho, informação recebida pela função.

Função 02 - int contaSaturacoes (double* dados, int n_amostras)

- Recebe: vetor de dados, tamanho do vetor.
- Objetivo: Contar o número de amostras que saturam.
- Saída: Número de amostras que saturam.

Assim como a função anterior, a dupla concordou que seria necessário percorrer o vetor através de um *loop*. Dessa vez, a cada iteração era verificado se o valor estava fora do limite, e em caso positivo, através de uma variável contadora, guardava-se o número de amostras saturadas.

Para o *if*, a ideia inicial era utilizar o operador "ou" considerando casos acima do limite ou casos abaixo do limite. Uma sugestão da aluna Amanda, foi o uso do "valor absoluto" (abs) da biblioteca "math.h", porém, após compilar e rodar o código percebeu-se a aparição de *warnings*. Então, a aluna Julia resolveu construir uma função com esse fim, e isso foi implementado no restante do código.

Função 03 - int hardClipping (double* dados, int n_amostras, double limite)

- Recebe: Vetor com dados, tamanho do vetor, variável limite (formato *double*)
- Objetivo: Simular a saturação – verificar amostras fora do limite e ajustá-las para terem magnitude igual ao limite.
- Saída: Número de amostras alteradas.

Através das imagens ficou claro o objetivo desta função. Para analisar os dados, novamente, foi necessário o uso de um loop, junto com uma estrutura condicional *if* para verificar se a amostra estava fora do limite. Nessa etapa surgiu o questionamento: "Como esse loop e condicional se repetem da mesma forma que na função 02, seria possível utilizar dados da função anterior?". Apesar disso, a dupla não encontrou uma forma eficiente que solucionasse o questionamento.

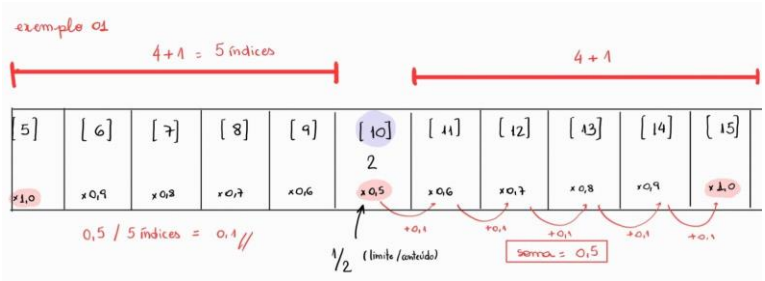
Diferente da função anterior, não se utilizou a função módulo, uma vez que era necessário diferenciar casos que passavam do limite inferior (que receberam o mesmo valor do limite com sinal negativo) e superior (receberam o mesmo valor do limite).

Função 04 - void limitaSinal (double* dados, int n_amostras, int n_passos)

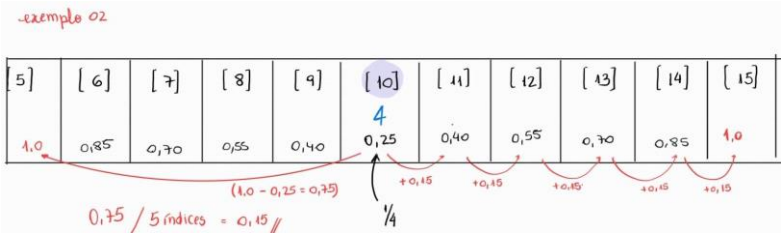
- Recebe: vetor com dados, tamanho do vetor, "número de vizinhos".
- Objetivo: Percorrer posições do vetor, atenuar amostras fora do limite e seus vizinhos, com ganho progressivamente maiores. Necessário descobrir o ganho!
- Saída: Função *void*, as alterações são feitas *in place*.

Reconhecer o objetivo da função foi simples, mas ao analisar o exemplo, foi necessário descobrir o motivo do ganho usado ser 0.5 para o índice 10 e 0.5+0.1 para os vizinhos. Para isso usou-se bastante da abstração, imaginando os índices como caixinhas.

Primeiramente, foi feito o esboço com o exemplo dado e a partir daí foram encontrados certos padrões: 0.5 é o quanto o valor deve ser alterado para ficar no limite (limite/vetor[i]); a caixa do "vizinho+1" não tem o ganho alterado, portanto, é multiplicado por 1; para que o ganho vá de 0.5 até 1 em 5 casas, é necessário que aumente em 0.1 a cada casa.



Para confirmar as teorias obtidas, foi feito outro exemplo, com o mesmo índice (10) acima do limite, porém com valor 4, n_passos = 4. Com a mesma lógica obtida anteriormente, os resultados foram satisfatórios, confirmando as hipóteses.



Para o código, utilizou-se novamente o *loop* e a condicional. Quando era identificado um valor que extrapolava o limite, dentro do condicional *if*, eram calculados "ganho" a partir do limite e do quanto o valor extrapolava esse limite; e a "diferença de ganho" (constante que aumenta a cada vizinho). Um *loop* aninhado foi utilizado para percorrer e ajustar o valor dos vizinhos, levando em conta o fato de que os vizinhos a esquerda não podem ter índice menor que 0, uma vez que levaria ao acesso indevido de memória. (Essa ideia surgiu após tentarmos corrigir outro erro).

Função 05 - void geraOndaQuadrada (double* dados, int n_amostras, int taxa, double freq.)

- Recebe: vetor com dados, tamanho do vetor (período), taxa de amostragem e frequência.
- Objetivo: Transformar o vetor em uma onde quadrada
- Saída: Função *void*, as alterações são feitas *in place*.

Para atingir o objetivo dessa função foi necessário que a dupla estudasse brevemente sobre ondas, frequência, amplitude e amostragem. Através da imagem fornecida e das pesquisar deu-se que um ciclo corresponde a um pico e um vale, a amplitude é a "altura" das ondas, e a frequência é a quantidade de vezes que o ciclo se repete em um segundo.

A lógica utilizada após entender as informações recebidas pela função foi descobrir a origem de cada dado do exemplo. Descobriu-se então que o valor de amostras para cada meio período do sinal corresponde à metade da taxa de amostragem, e que a quantidade de amostras para determinada frequência é o meio período dividido pela frequência.

Como dito no texto, o período do sinal pode não ser um número inteiro e por isso deve-se considerar o erro acumulado, que consiste em pegar as "sobras" (decimais), somá-las e depois utilizá-las no próximo período caso esse valor passe para um inteiro, ou seja, maior que 0.

No código foram feitos esses cálculos para encontrar a quantidade de amostras por meio período pela frequência e sua parte inteira e decimal. O erro acumulado foi inicializado com o valor decimal do número de amostras, uma vez que o primeiro ciclo já se inicia com essa "sobra".

Foi utilizado um *loop* para percorrer e atribuir o valor para os dados do vetor até que o contador atingisse a parte inteira do número de amostras. Ao atingir esse número que corresponde ao meio período, o programa entra na condicional que tem os comandos de inverter sinal e zerar o contador, além de incrementar o erro acumulado. Caso o erro acumulado seja maior que 1, entra-se na segunda condicional que faz com que ocorra uma amostra a mais com o valor invertido.

Considerações finais

O trabalho sendo realizado em dupla foi muito proveitoso. Poder compartilhar melhor as ideias e dúvidas com um colega tornou a experiência mais simples uma vez que ocorreram diversas ocasiões em que uma das integrantes sanava a dúvida da outra e vice-versa. Foi interessante também notar a forma que cada uma das integrantes percebia e processava o problema de tal forma que completava ou complementava as ideias e soluções.

A parte do código em si foi considerada simples pela dupla. Nenhum dos comandos utilizados nas funções exigia conhecimento externo, era necessário somente compreender bem as informações obtidas durante o semestre.

Colegas de outros grupos também foram importantes na troca de ideias e principalmente nos testes. Inicialmente testávamos somente pelo áudio gerado e depois nos compartilharam o software Audacity que nos permitiu visualizar as ondas modificadas, a visualização é melhor ao aproximar as ondas. As ondas obtidas estão nas imagens a seguir:

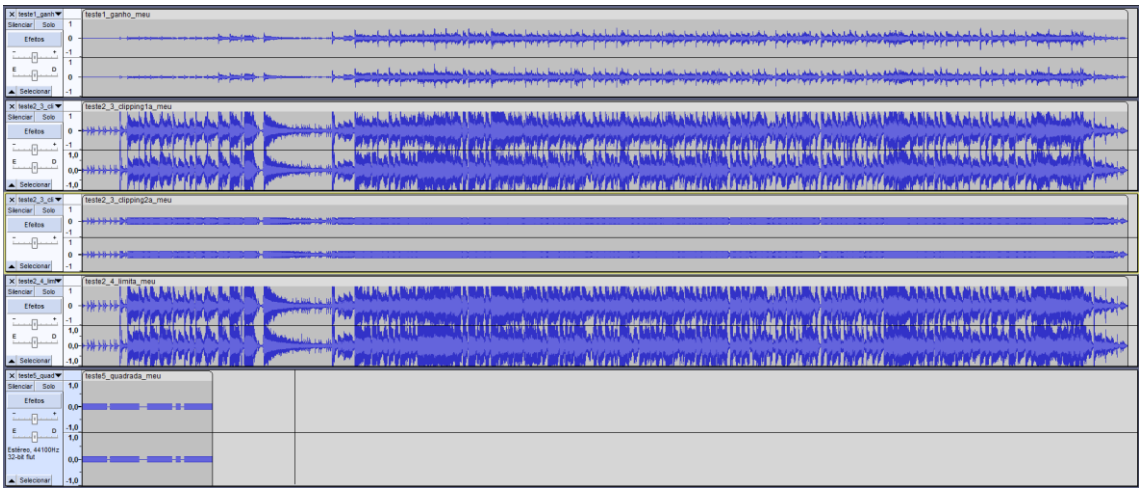


Figura 1. Ondas inteiras obtidas nos testes 1, 2, 3, 4 e 5 respectivamente.

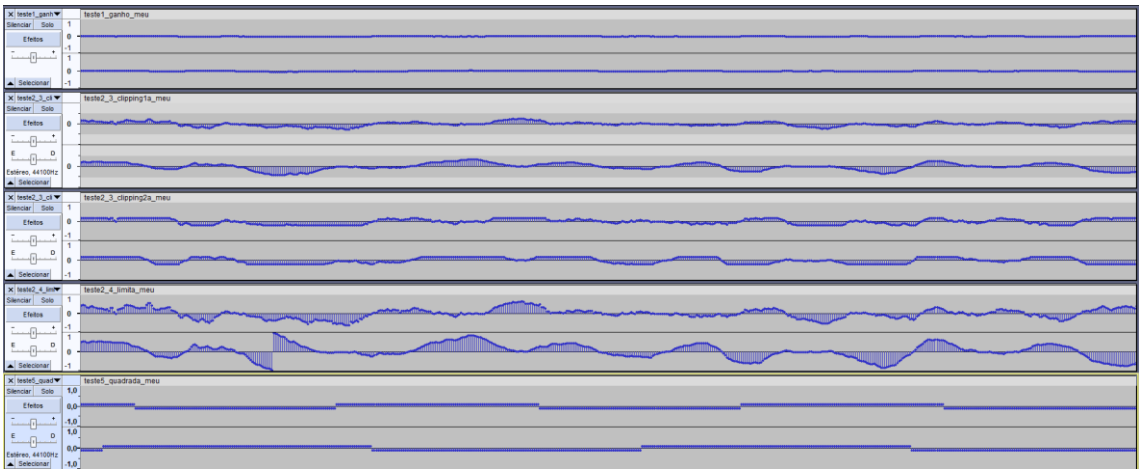


Figura 2. Fragmento das ondas nos testes 1, 2, 3, 4 e 5 respectivamente.