Ana Julia Molinos Leite da Silva - 2582694
Nicolas Vieira Ferreira - 2603519

Relatório de progresso

Iniciamos o projeto discutindo, por meio de chamada, sobre como fazer e quando realizar todas as funções de maneira conjunta um ao outro, para evitar dúvidas quanto ao código durante a sua construção. Discutimos como ele funcionaria e de que forma a elaboração ideal do algoritmo seria definida. Com o plano de ação definido, decidimos começar pelas funções 1, 2 e 3, pois são consideravelmente mais simples. Optamos por fazer todas no mesmo dia, implementando a ideia e, em seguida, transcrevendo-a para o código. E com as funções 4 e 5 foi preciso optar por dedicar um tempo a mais por necessidade, para realizar a elaboração destas funções (4 e 5) foi preciso recorrer a monitoria para contestar as dúvidas que surgiram ao longo do planejamento do algoritmo.

Durante o desenvolvimento das funções, a função 4 (limitaSinal) foi a primeira a apresentar desafios. Entendemos que essa função atenua as amostras que ultrapassam o limite e que, ao mesmo tempo, atenua as amostras vizinhas. E chegamos à conclusão de que ela é feita para evitar a saturação que pode levar a distorção de áudio. Começamos definindo que o algoritmo precisava ter um laço de início para percorrer os dados do vetor, e uma condição para comparar se os dados são maiores que o limite ou menores que o limite negativo. Logo após esses primeiros passos, chegamos ao nosso primeiro entrave, como iríamos percorrer as casas vizinhas à direita e à esquerda das amostras que extrapolam o limite dado. Em um primeiro momento, pensamos que precisaria de dois loops para percorrer os passos à direita e à esquerda, mas, percebemos que só precisaria de um loop que realizaria a soma ou subtração ao índice da amostra atenuada. Em seguida chegamos em nosso segundo entrave, a definição da variável ganho, recorremos a monitoria para tirar a dúvida sobre qual cálculo a variável ganho deveria obedecer para que crescesse gradualmente conforme os n passos dados e com base na amplitude da amostra atenuada., O monitor esclareceu a importância de que o ganho acumulado ao final dos n passos não fosse maior que 1. Isso nos auxiliou a chegar no cálculo utilizado na implementação da função.

```
15
16  for(i=0; i<n_amostras; i++){
17
18      if(dados[i]>1 || dados[i]<-1)
19      {
20          dados[i]= alguma??? //atualiza o valor da posiçao que ex
21
22          for(j=1; j<n_passos; j++)//atualiza n_passos a direita
23          {
24              dados[i+j]=dados[i] + alguma coisa????
25          }
26
27          for(j=1; j<n_passos; j++)//atualiza n_passos a esquerda
28          {
29              dados[i+j]=dados[i] + alguma coisa????
30          }
31      }
32  }
```

```
48
49  for(i=0; i<n_amostras; i++)
50  {
51      if(dados[i]>LIM || dados[i]<-LIM)
52      {
53          if(dados[i]>0){
54              aux = 1.0/dados[i];
55           }else{
56              aux = 1.0/-(dados[i]);
57           }
58          dados[i] *= aux; //atenua p/ 1 o valor da posiçao que
59
60          for(j=1; j<=n_passos; j++) //percorre n_passos
61          {
62              //calcula o ganho baseado na intensidade da amostra
63              ganho = aux + (((0.9 - aux)/n_passos)*j);
64
65              dados[i+j] *= ganho;//atualiza n_passos a direita
66              dados[i-j] *= ganho;//atualiza n_passos a esquerda
```

Na função 5, acabamos por debater ideias e botar o código em prática para que pudéssemos nos localizar e propor melhorias. Nossa primeira dificuldade foi sobre como controlar o preenchimento de um meio-período com [1] ou [-1]. Inicialmente, cogitamos utilizar um segundo for para controlar o preenchimento do meio-período, tendo essa variável como limite, mas percebemos que não teríamos controle sobre o conteúdo que preencheria o vetor. Após observar um pouco mais, conseguimos chegar a conclusão que um contador atrelado a uma condicional seria mais prático. Assim, toda vez que o contador alcançasse o valor do meio-período, era necessário apenas zerá-lo e multiplicar o valor de preenchimento das amostras por -1. Isso garantiu a alternância entre -1 e 1 conforme o tamanho do meio-período. O segundo empecilho relacionado a essa função foi pensar em como adaptar esse algoritmo para os casos em que o meio-período fosse um número não inteiro e gerasse propagação de erro ao longo dos ciclos. O que ajudou a pensar na adaptação foi observar a relação entre o erro e o tamanho do meio-período por meio da tabela abaixo, construída com base no exemplo dado nas instruções do trabalho:

Meio-período = 2060,74766				
Ciclo	**[1]**	**Erro**	**[-1]**	**Erro**
1	2060	0,74766	2061	0,49532
2	2061	0,24298	2060	0,99064
3	2061	0,7383	2061	0,48596
4	2061	0,23362	2060	0,98128
5	2061	0,72894	2061	0,4766

Observamos que toda vez que o erro somado a parte decimal do meio-período resultava em um valor maior que 1, o meio-período seria acrescido de uma amostra. Isso nos ajudou a estruturar uma condicional que, junto de uma flag, define se o período a ser preenchido é correspondente ao meio-período ou meio-período+1, com base no erro calculado. Assim foi possível adaptar a ideia inicial para meio-períodos com valores não inteiros.