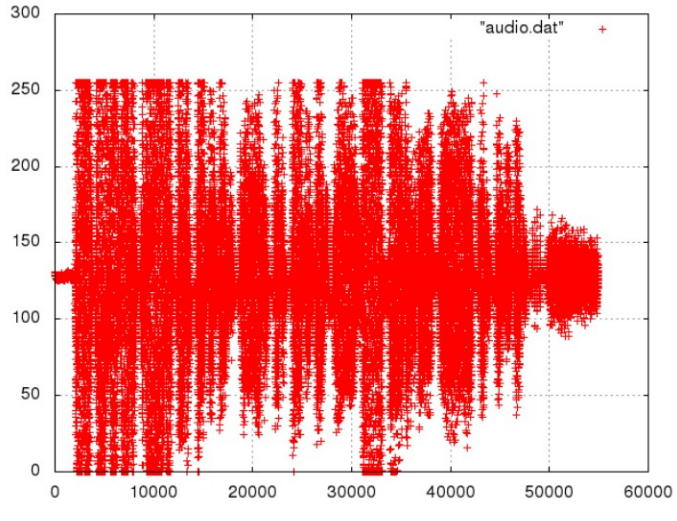
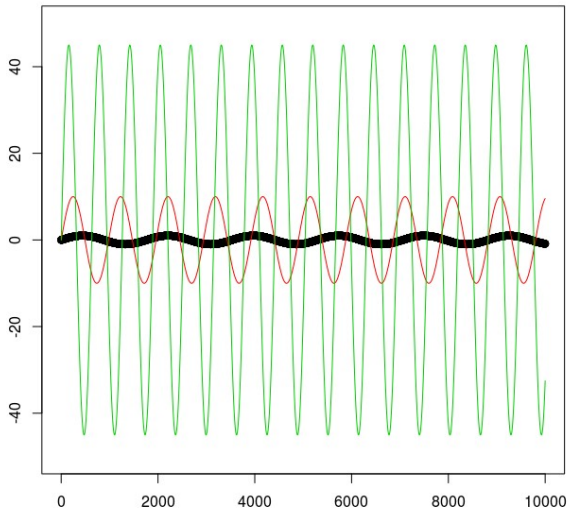
Trabalho 2

Descrição

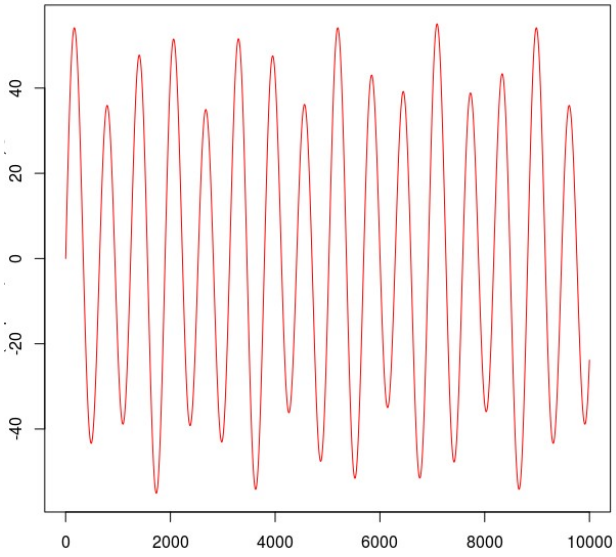
Uma das ferramentas mais úteis na análise e processamento de sinais como áudio é a Transformada Discreta de Fourier. Essa transformada permite obter um coeficiente (que é um par de valores) para cada observação (amplitude do diafragma do microfone) do sinal de áudio original. De forma grosseira, esses coeficientes representam a quantidade de uma determinada onda senoidal presente no sinal, desde aquelas de menor frequência (sons mais graves), até as de maior frequência (sons mais agudos).

Para exemplificar, considere o sinal de áudio a seguir. Observe que ao obter várias senóides podemos, por meio de sua soma, reconstruir o sinal original. A figura à direita apresenta algumas senóides para exemplificar.





Após somar as várias senóides que compõem um sinal, o reconstruimos, veja o exemplo da soma das senóides acima:



Para obter a Transformada Discreta de Fourier de um sinal, usa-se a equação abaixo:

$$C_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{k}{N} n}$$

Em que:

a) um conjunto de valores x_0, x_1, …, x_{n-1} é aplicado como entrada;

b) k é o índice do coeficiente produzido pela transformada, ou seja, se k = 1, então produzimos o primeiro coeficiente C_k e assim sucessivamente. Em geral k = 1, 2, 3, … N.

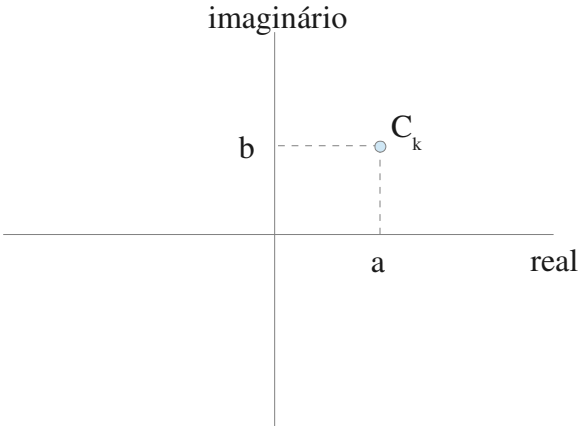
c) N é o número máximo de coeficientes que serão produzidos pela transformada

d) O índice n é utilizado no somatório a fim de produzir um coeficiente C_k

e) O termo ***e*** representa a aplicação da função cexp(valor) na linguagem C, ou seja, é o exponencial de certo valor

f) O termo i indica imaginário, ou seja, calculamos a função cexp(valor) para um valor que é dado por um número complexo

Por consequência do uso do número complexo na exponencial, cada coeficiente C_k produzido por essa transformada é composto por uma parte real e outra imaginária, ou seja, um número complexo. A maneira mais simples de compreender um número complexo é considerando um espaço de coordenadas polares na forma:



O ponto C_k formado nesse espaço é definido pelo valor real a e pelo valor imaginário b na forma:

$$C_k = a + b\,i$$

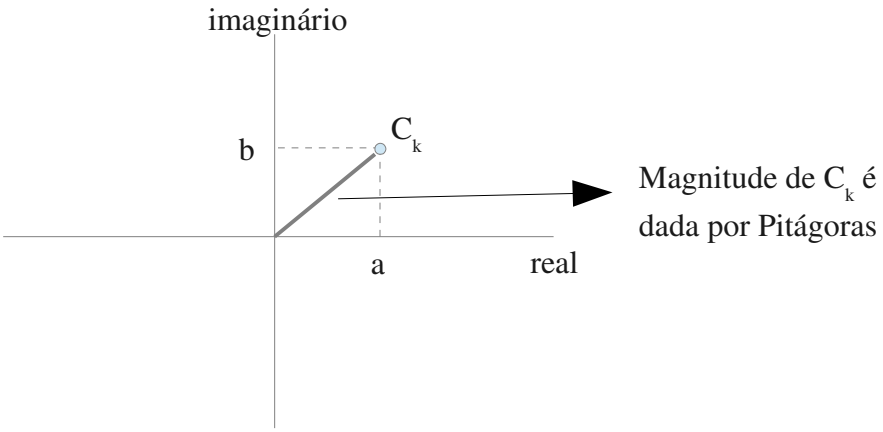
em que i indica a parte imaginária do número complexo. Após aplicar a transformada sobre os bytes que compõem um áudio, iremos obter vários pontos nesse espaço de coordenadas polares. Cada ponto desses representa um componente do áudio. Podemos medir a magnitude desses componentes. A magnitude indica a importância desse componente no áudio, ou seja, quanto maior a magnitude, mais

relevante esse componente é para o áudio em questão.

Para medir a magnitude computamos a equação:

$$M_k = \text{sqrt}(a^2 + b^2)$$

Observando o espaço de coordenadas polares, ao calcularmos a magnitude do componente C_k temos:



Uma das propriedades importantes de uma Transformada é a possibilidade de invertê-la. Nesse caso, a partir dos coeficientes podemos obter o sinal (áudio) novamente. Com isso é possível realizar o seguinte processamento do áudio:

1. Transformar o sinal x em uma série de coeficientes complexos C
2. Modificar (todos ou em parte) os coeficientes C
3. Realizar a transformada inversa em C, obtendo um sinal y, que é o sinal x processado.

Vamos utilizar essa idéia nesse trabalho para eliminar parte dos coeficientes. Essa eliminação é uma forma de representar o sinal por uma menor quantidade de dados ou ainda de melhorar a capacidade de compressão dos dados.

A Inversa da Transformada Discreta de Fourier pode ser obtida pela função contida no seguinte código fonte (salvar num arquivo inverse.c):

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

unsigned char* inverseFourier(double complex *coefficients, int N) {
	// N é o total de coeficientes
	// M_PI é uma macro que define o valor do PI (math.h)

	double complex *audio = (double complex *) calloc(N, sizeof(double complex));
	unsigned char *audio8ubits = (unsigned char *) calloc(N, sizeof(unsigned char));
	int n, k;

	for (n = 0; n < N; n++) {
		for (k = 0; k < N; k++) {
			audio[n] = audio[n] + coefficients[k] * cexp( ( 2.0 * M_PI * ( ((k+1) *
```

```
n * 1.0) / ( N * 1.0 ) ) ) * _Complex_I );
                }
                audio[n] = audio[n] / (N * 1.0);
                audio8ubits[n] = (int) __real__ audio[n];
        }
        free(audio);
        return audio8ubits;
}
```

Para compilar apenas esse código use:

gcc -c inverse.c -lm

A opção -lm é necessária para termos acesso à libmath, essa biblioteca contém a função cexp() dentre outras.

Caso você inclua uma função main() que utilize essa função, será possível gerar um executável, compilando com:

gcc -o inverse inverse.c -lm

Código exemplo para definir um número imaginário usando linguagem C:

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

int main(int argc, char *argv[]) {
        double complex numero_complexo = 3.0 + 4.0 * _Complex_I;

        double parte_real = __real__ numero_complexo;
        double parte_imaginaria = __imag__ numero_complexo;

        printf("Parte real: %lf\n", parte_real);
        printf("Parte imaginaria: %lf\n", parte_imaginaria);

        return 0;
}
```

A teoria da Transformada de Fourier diz que é possível reconstruir o áudio original a partir da representação desse sinal em somas de funções senoidais, ou seja, conseguimos reconstruir o sinal de áudio a partir da soma de senos e cossenos (representados na equação pela exponencial complexa).

O interessante dessa técnica é que, em geral, os vetores de coeficientes (após a Transformada) são esparsos (possuem muitos valores nulos, ou muito próximos de zero). Dessa forma, perceba que, se precisarmos transferir o áudio para outro computador, podemos, apenas, transferir os *T* coeficientes mais importantes e suas posições originais no vetor (ou seja, antes de ordená-lo), além, é claro, do número total de coeficientes. No outro computador, poderíamos recuperar essas informações, criar um vetor para conter os coeficientes selecionados e aqueles que terão valor igual a zero. Em seguida, aplicaríamos a Transformada Inversa de Fourier e obteríamos uma aproximação do sinal de áudio.

Perceba que assim, transferiríamos uma quantidade muito menor de dados para o computador

destino e poderíamos, ainda, reconstituir o áudio e ouví-lo.

Para saber mais sobre a Transformada Discreta de Fourier, seu funcionamento e aplicações procure em:

http://en.wikipedia.org/wiki/Discrete_Fourier_transform
http://mathworld.wolfram.com/DiscreteFourierTransform.html

Tarefa

Considerando um arquivo de áudio gravado em unsigned char, ou seja, cada valor de variação do diafragma do microfone é dado por um inteiro de 8 bits sem sinal:

1) Leia do teclado o nome do arquivo de áudio a ser aberto. Abra esse arquivo em modo leitura de arquivo binário.
2) Leia do teclado o número de coeficientes que devem ser utilizados. Esse número inteiro T será indicado pelo usuário.
3) Leia todo seu conteúdo e armazene em um vetor dinamicamente alocado (memória heap).
4) Em seguida, considere esse vetor como entrada para a Transformada Discreta de Fourier, ou seja, esse vetor definirá o conjunto x_0, x_1, …, x_{n-1}.

Aplique a Transformada gerando o mesmo número N de coeficientes que o número N de observações contidas do arquivo de áudio. Entenda por observação cada inteiro de 8 bits sem sinal contido no arquivo de áudio. Dessa maneira, serão produzidos os seguintes coeficientes após aplicar a transformada: C_1, C_2, …, C_N, ou seja, o índice dos coeficientes varia de 1 a N, sendo N o número total de observações no arquivo de áudio.

3) Armazene os coeficientes em um vetor dinamicamente alocado.
4) Calcule a magnitude para esses coeficientes. Em seguida ordene o vetor de coeficientes de maneira **decrescente** segundo suas magnitudes, ou seja, o coeficiente de maior magnitude será o primeiro do vetor, em seguida o segundo de maior magnitude e assim sucessivamente. Será preciso, no entanto, guardar qual é a posição original de cada coeficiente, pois essa informação será usada no passo 6.
5) Atribua zero aos coeficientes das posições maiores do que T. Devem ser mantidos no vetor de coeficientes somente os valores dos T primeiros coeficientes.
6) Em seguida, volte os coeficientes para suas posições originais, ou seja, para as posições que eles tinham antes da ordenação.
7) Utilizando como entrada os coeficientes processados (obtidos no passo 6), aplique a Transformada Inversa de Fourier. Note que a Inversa poderá gerar valores de ponto flutuante, que deverão, na hora da escrita, serem convertidos para inteiros.
8) Salve o resultado dessa Transformada Inversa em um arquivo binário de áudio em que as amplitudes do diafragma são formadas por 8 bits sem sinal.
9) O programa deverá mostrar como saída (na tela do computador):

- o número de observações lidas do arquivo de áudio
- o número de valores menores ou iguais a zero tante na parte real quanto na parte imaginária no vetor dos coeficientes original (**antes** da atribuição de 0 para os valores de

posições maiores que T)
- a sequência ordenada das magnitudes dos coeficientes das posições de 1 a T, convertidas para um inteiro (*casting* para *int*).

Para demonstrar uma das aplicações do procedimento realizado, toque o arquivo de entrada e de saída e perceba que, devido ao corte de frequências, o áudio é suavizado, eliminando ruído e chiados.

Exemplo de Caso de Teste (entrada e saída)

Entrada:

audio01.raw
8

Saída:

4000**\n**
1251**\n**
428300_12003_8020_6011_4003_3999_3803_3003_**\n**

onde _ significa um espaço em branco e **\n** uma quebra de linha no arquivo de saída.

Serão oferecidos 4 arquivos de áudio .raw e 2 casos de entrada e saída para exemplificar.

Informações adicionais importantes

- Sobre o sistema de submissão (SQTPM):

1. Seu código deverá ser submetido num arquivo fonte .c. Esse arquivo deverá **obrigatoriamente** conter no início um comentário com seu nome, número USP, turma e data da entrega do trabalho

2. A compilação do código é feita pelo comando:
   ```
   gcc -o prog proc.c -lm -Wall
   ```

3. A saída do seu programa deve ser exatamente igual à saída esperada, incluindo: espaços em branco, quebras de linha e precisão decimal

4. Há um limite de 90 segundos para a execução dos 10 casos de teste e um limite de 18.5MB de memória total para ser utilizada. Você deverá gerenciar bem o tempo de execução e o espaço ocupado para que seu programa fique dentro desses limites. O erro de “Violação de Memória” significa que seu programa está usando mais do que 18.5MB.

5. Como os algoritmos envolvidos são relativamente lentos, o servidor poderá demorar para

mostrar os casos de teste, após o envio. Aguarde ao menos 90 segundos, sem recarregar a página, para obter a resposta.

- Sobre a avaliação

1. Você deverá desenvolver seu trabalho sem copiar trechos de código de outros alunos nem realizar codificação em conjunto. Portanto, compartilhem idéias, soluções, modos de resolver o problema, mas não o código. Qualquer dúvida entrem em contato com o professor.

2. Todos os códigos fontes serão comparados por um sistema de detecção de plágio (MOSS): http://theory.stanford.edu/~aiken/moss/, e **os trabalhos com similaridade detectada terão suas notas zeradas**, tanto aqueles relativos ao código de origem quanto do código copiado.

3. A avaliação incluirá a porcentagem de acertos verificada pelo SQTPM e também a análise do seu código, incluindo endentação, comentários, bom uso da memória e práticas de programação.