notebook3

April 4, 2016

1 Sumário

Teste da transformada rápida de Fourier (FFT)

Nessa parte do trabalho iremos testar como o cálculo da FFT irá se comportar com as bibliotecas nativa do julia e depois comparar com o myFFT

$$y[k] = \sum_{n=0}^{\text{N-1}} W_n x[n]$$

$$W_n = e^{-2\pi j \frac{km}{N}}$$

Iremos calcular a FFT de um vetor com 4096 valores usando as bibliotecas propostas e assim iremos analisar o tempo de compilação de cada.

Resolução da FFT usando a função nativa

```
In [33]: # Aqui iremos implementar o  o cálcula da FFT usando julia
         vetorFftJulia=[1:1:4096];
         @time fft(vetorFftJulia);
```

```
0.000165 seconds (66 allocations: 131.141 KB)
```

Resolução da FFT usando myFFT

```
In [34]: # criando a função myFFT, proposta para o trabalho
         function myfft(x)
             N=4096
             for i=1:N
                 x[i]= cos(2*pi*i/N)-1im*sin(2*pi*i/N)
                 end
         end
```

```
Out[34]: myfft (generic function with 1 method)
```

```
In [35]: N=4096
         vetorFftMy=Complex32[1:1:N];
         @time myfft(vetorFftMy)

0.008397 seconds (3.94 k allocations: 181.251 KB)
```

Teste da transformada rápida de Fourier inversa (IFFT)

Nessa parte do trabalho iremos testar como o cálculo da IFFT irá se comportar com as bibliotecas nativa do julia e depois comparar com o myIFFT

$$x[n] = \frac{1}{N} \times \sum_{n=0}^{\text{N-1}} W_n y[k]$$

$$W_n = e^{2\pi j \frac{km}{N}}$$

Iremos calcular a FFT de um vetor com 4096 valores usando as bibliotecas propostas e assim iremos analisar o tempo de compilação de cada.

Resolução da IFFT usando a função nativa

```
In [36]: # Aqui iremos implementar o  o cálcula da FFT usando julia
         vetorIfftJulia=[1:1:4096];
         @time ifft(vetorIfftJulia);

0.000211 seconds (76 allocations: 131.688 KB)
```

Resolução da IFFT usando myIFFT

```
In [42]: # criando a função myIFFT, proposta para o trabalho
         function myifft(x)
             N=4096
             for i=1:N
                 x[i]=1/N*cos(2*pi*i/N)-1im*sin(2*pi*i/N)
                 end
         end

Out[42]: myifft (generic function with 1 method)

In [43]: N=4096
         vetorIfftMy=Complex32[1:1:N];
         @time myifft(vetorIfftMy)

0.012006 seconds (4.31 k allocations: 196.286 KB)
```

1.1 # Conclusão

Ao termino do trabalho podemos concluir que assim como no Python(notebook 2), as bibliotecas ou funções nativas(no caso do julia,que já nos fornece uma gama de funções), tiveram um tempo de compilação menor,para as funções FFT() e IFFT(),já para as funções myFFT() e myIFFT() - que foram criadas no decorrer do trabalho- tiveram um tempo de compilação maior.

1.2 # Referências

www.docs.julialang.org/en/release-0.4/manual/

www.en.wikibooks.org/wiki/Introducing_Julia/