Prova 2

Processamento de Sinais

Pedro Henrique dos Santos Cunha

Universidade Federal Fluminense
Instituto de Ciências Exatas
Departamento de Física

Conteúdo

1 Introdução à Filtragem de Sinais

No estudo de processamento de sinais uma área de bastante interesse tanto acadêmico quanto industrial é a área de filtragem de sinais. A filtragem de sinais possui uma ampla e diversa aplicabilidade tecnológica no mundo contemporâneo, possibilitando a realização de melhorias, criação de novos receptores de sinais, entre outros.

Tais filtros são responsáveis pela filtragem do sinal recebido pelo aparelho com o intuito de realizar alguma ação com este sinal. Exemplo de filtros comuns são os filtros de sinal digital para a renderização de fotografias.

A representação de um filtro pode ser feita a partir de uma resposta a um impulso, uma resposta a uma frequência ou a um degrau. Neste trabalho utilizaremos a descrição a partir da resposta a uma frequência.

2 Filtros de Frequência

Os filtros de frequência são responsáveis por filtrar as frequências e as dividir em bandas de frequência, resultando em duas bandas principais:

- Bandas de passagem: São as bandas cujo filtro permite a passagem.
- Bandas de rejeição: São as bandas cujo filtro não permite a passagem.

Os filtros de resposta a frequencia podem ser classificados em filtros passa-baixa, passa-alta, passa-faixa e rejeita-faixa. O filtro passa-baixa é capaz de permitir a passagem apenas de sinais de baixas frequências de entrada, filtrando a partir de uma frequencia pre-determinada. Já o filtro passa-alta funciona de forma exatamente contrária. No caso dos filtros passa-faixa, é selecionado uma faixa de frequências cujo o filtro permite a passagem, enquanto que no filtro rejeita-faixa, é selecionado uma faixa de frequências cujo o filtro não permite a passagem.

Além disso, um dos conceitos fundamentais para a construção de um filtro de frequência é o conceito de roll-of, que se refere a largura da faixa de frequencia. Em suma, o conceito de roll-off nos diz que quanto maior a faixa de frequencia, pior os resultados do filtro. Sendo assim, idealmente, devemos considerar uma faixa de frequência de transição pequena.

3 Filtros de Frequência FIR

No nosso trabalho, utilizaremos o modelo de filtro FIR - Finite Impulse Response - para a construção do nosso filtro. Intuitivamente, o filtro FIR representa respostas finitas aos impulsos.

O filtro FIR apresenta diversas vantagens, uma delas é o fato do filtro FIR ser um filtro de estrutura regular e sua implementação pode ser realizada a partir de diferenças entre equações sem dependência recursiva.

A resposta de um filtro FIR é modulada por:

$$y(n) = \sum_{i=0}^{R-1} d_i x[n-i] \tag{1}$$

Onde R é o comprimento do filtro e d_i os coeficientes do filtro.

4 Discrete Fourier Transform

Quando aplicamos uma transformação integral em uma função, essa transformação integral constrói uma continuação analítica levando nossa função definida em um espaço, em outra função correspondente à esta original, porém em outro espaço. Com a transformada de Fourier não é diferente. No nosso contexto, ela transforma nossa função definida no espaço das frequências em uma função definida no espaço do tempo.

Como estamos lidando com um código de computador, precisamos realizar uma discretização. Por conta disso, utilizamos a técnica da DFT - Discrete Fourier Transform - no nosso trabalho. Sendo mais específico, utilizamos a implementação da FFT - Fast Fourier Transform -.

Em síntese, modelamos a transformada direta como:

$$t(\omega) = \sum_{i=0}^{n-1} \omega[i] \left\{ \cos\left(\frac{2\pi i\omega}{N}\right) - \sin\left(\frac{2\pi i\omega}{N}\right) \right\} \tag{2}$$

Em contrapartida, a transformada inversa é modelada como:

$$\omega(t) = \sum_{i=0}^{N-1} \frac{1}{n} t[i] \left\{ \cos\left(\frac{2\pi i\omega}{N}\right) + \sin\left(\frac{2\pi i\omega}{N}\right) \right\} \tag{3}$$

5 Objetivos

Nosso objetivo nessa atividade é implementar filtro digital que reproduza a resposta em frequência definida em um arquivo endereçado a cada aluno. No meu caso, o arquivo é a frequência:

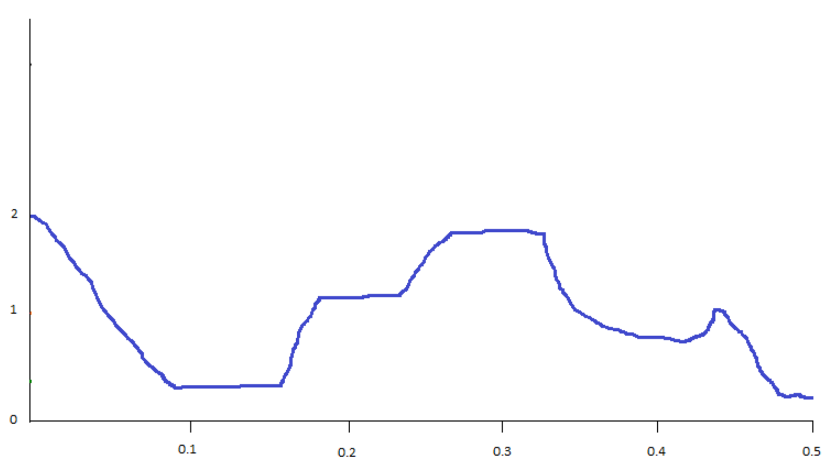

Figura 1: Frequência a ser reproduzida.

6 Metodologia e Resultados

Implementamos o código para um kernel de tamanho $M = 30$, $M = 70$ e $M = 150$ e utilisamos como dados iniciais os dados da figura de referência. Após aplicado a transformada de Fourier, aplicamos a técnica de janelamento e então aplicamos novamente a transformada inversa para determinar a frequência de saída para a comparação com a original. Obtemos os seguintes resultados:

- Para $M = 30$:

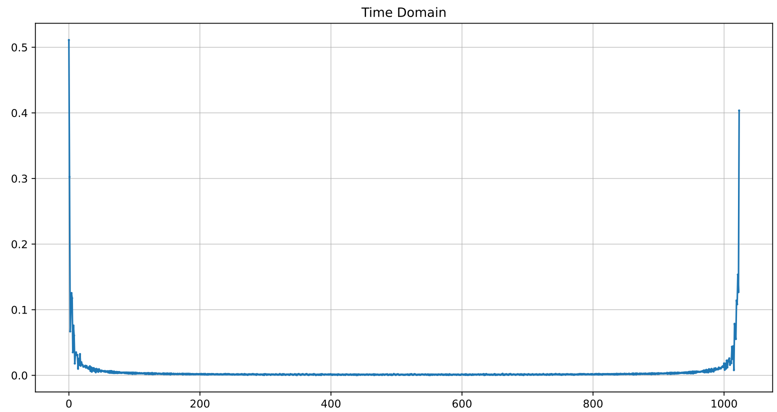

Figura 2: Resultado após aplicar a transformada de Fourier nos dados da figura de referência.

Aplicando a técnica de janelamento, obtemos a transformada de fourier da frequencia original:

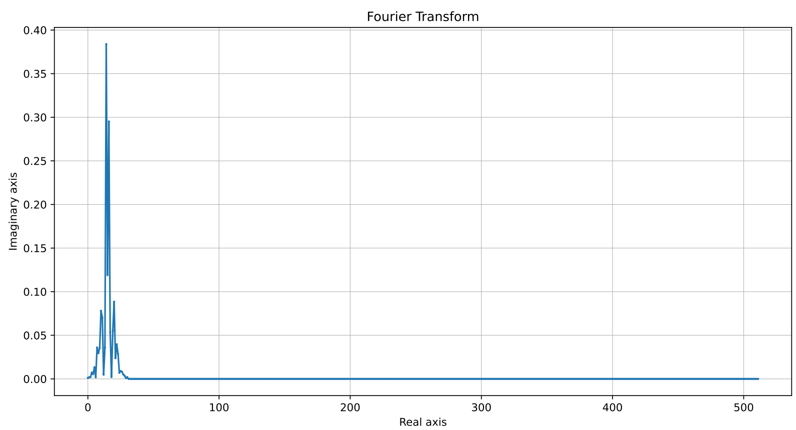

Figura 3: Resultado após aplicar a técnica de janelamento.

Aplicando a transformada inversa, recuperamos a frequência:

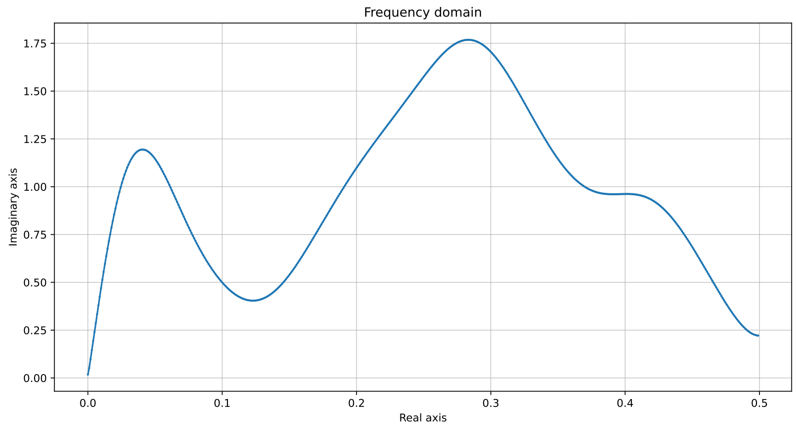

Figura 4: Resultado após aplicar a transformada inversa, isto é, frequência de saída.

- Para $M = 70$: Repetimos o mesmo procedimento, os resultados foram:

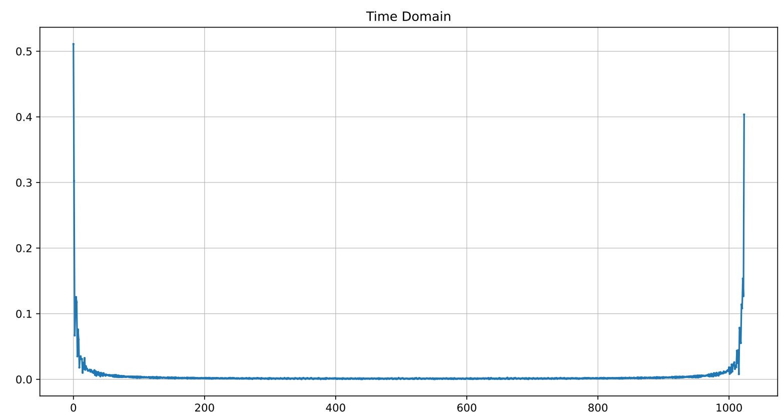

Figura 5: Resultado após aplicar a transformada de Fourier nos dados da figura de referência.

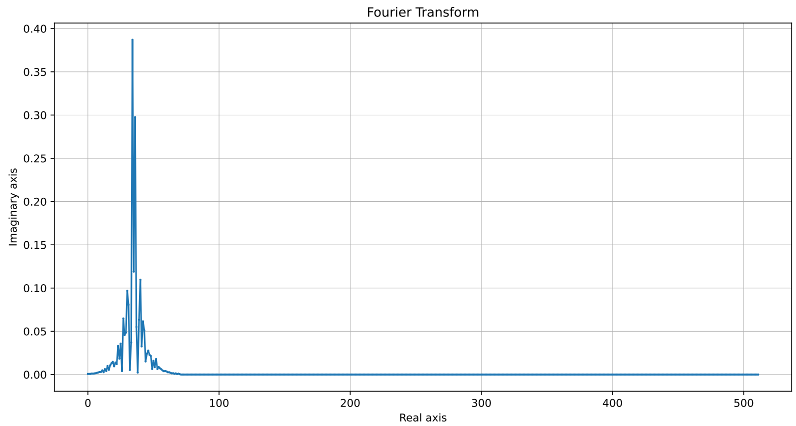

Figura 6: Resultado após aplicar a técnica de janelamento.

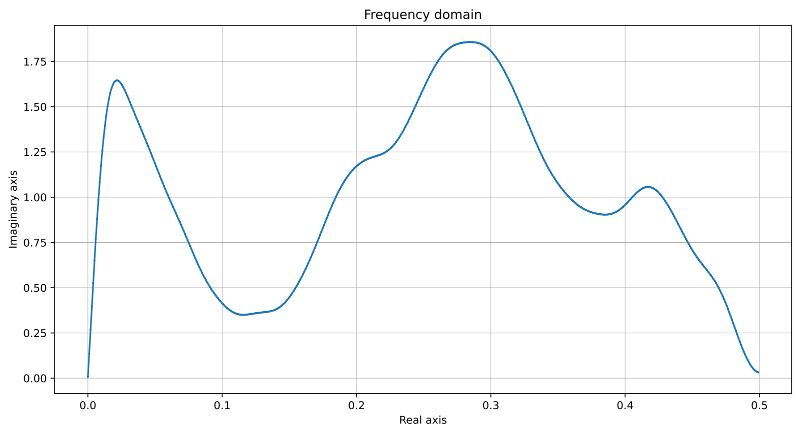

Figura 7: Resultado após aplicar a transformada inversa, isto é, frequência de saída.

- Para $M = 150$: Repetimos o mesmo procedimento, os resultados foram:

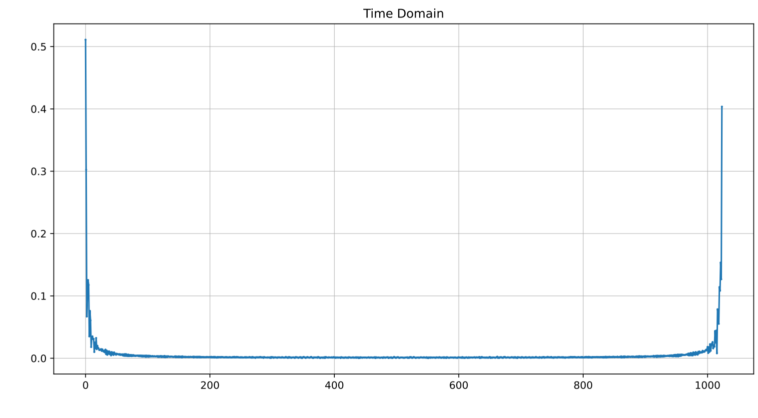

Figura 8: Resultado após aplicar a transformada de Fourier nos dados da figura de referência.

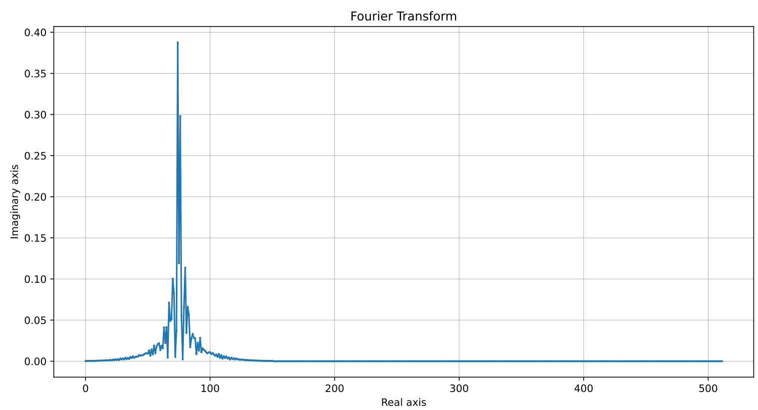

Figura 9: Resultado após aplicar a técnica de janelamento.

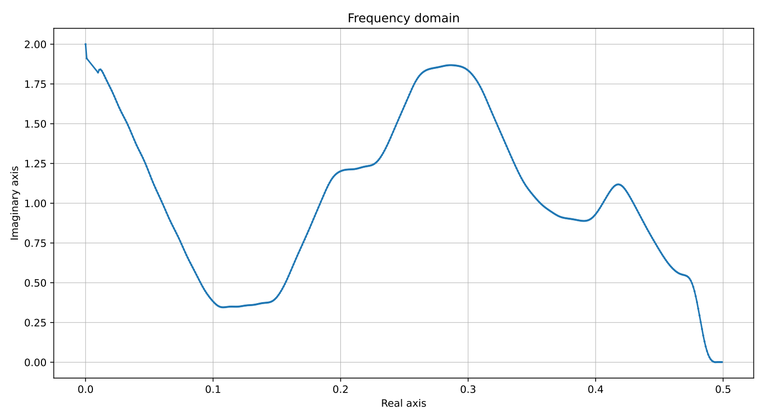

Figura 10: Resultado após aplicar a transformada inversa, isto é, frequência de saída.

Repare que os resultados ficam cada vez mais semelhantes a medida que aumentamos o valor do kernel. Isto é, comparando a figura 10 com a figura original 1, ja notamos grande semelhança.

Nesse sentido, portanto, conseguimos implementar o filtro com sucesso.

7 Teste dos filtros

De acordo com o slide do ultimo modulo do curso de processamento de sinais, podemos aplicar a convolução para testar o filtro. O algoritmo foi dado no slide. Nesse sentido, para testar os filtros, selecionamos uma faixa de frequencias para visualizar como o filtro se comporta quando utilizado tais frequencias. O gráfico abaixo mostra o comportamento do nosso filtro:

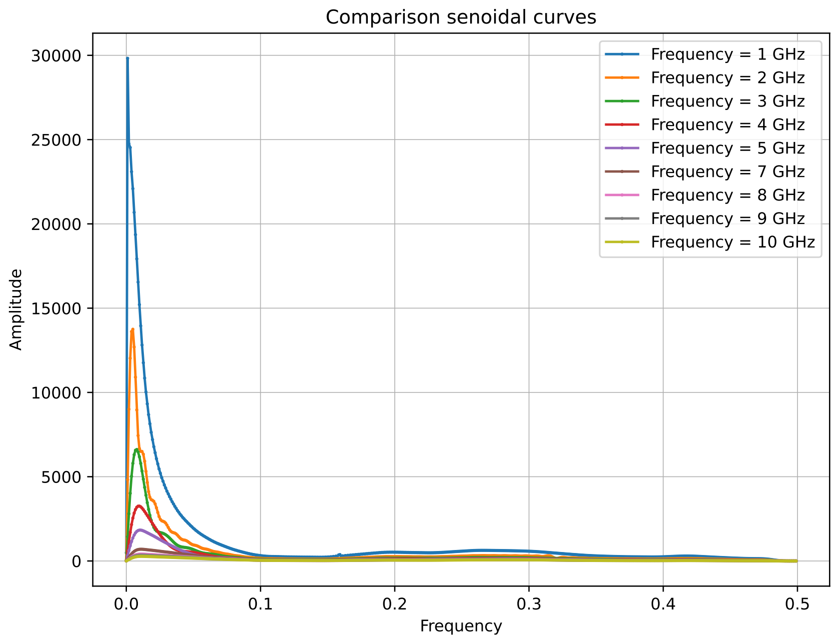

Figura 11: Teste do filtro para uma faixa de frequências

8 Conclusões

Por fim, podemos concluir que a medida que o valor do kernel se torna suficientemente grande, retomamos a frequência original utilizado. Portanto, conseguimos construir um filtro eficiente para o nosso problema, além de desenvolver meios para a recuperação da frequencia inicial recebida.

Além disso, reforçamos a importância e a eficiencia da utilização do método da transformada de Fourier discreta (DFT) possui para a construção dos filtros, pois, sem ela, não seria possível essa implementação.

9 Apêndice - Códigos utilizados

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

#define TAM 1024

void windowing(int kernel, double *RealAxis, double *ImaginaryAxis) {

    int i, j, k, index;
  double *time;

  time = (double*) malloc( TAM*sizeof(double));

  for( i=0; i < TAM; i++) {

      index = i + kernel/2;

        if(index > TAM-1){
        index = index - TAM;
    }

    time[index] = RealAxis[i];
    RealAxis[i] = 0;
    ImaginaryAxis[i] = 0;
    }

    for( i=0; i < TAM; i++) {
        RealAxis[i] = time[i];
    }

  for( i=0; i < TAM; i++) {
        if( i <= kernel) {
            RealAxis[i] = RealAxis[i]*(0.54 - 0.46*cos((2.*M_PI*i)/kernel));
    }
    else {
            RealAxis[i] = 0;
            ImaginaryAxis[i] = 0;
    }
  }
}

void fourierTransform(double *RealAxis, double *ImaginaryAxis) {

    int i, j, k, m, l, le, le2, aux;
    double  tr, ti, ur, ui, sr, si;

    m = (int)(log(TAM)/log(2));

    j = TAM/2;

    for( i=1; i < TAM-2; i++)
    {
        if(i < j)
```

```
        {
            tr = RealAxis[j];
            ti = ImaginaryAxis[j];
            RealAxis[j] = RealAxis[i];
            ImaginaryAxis[j] = ImaginaryAxis[i];
            RealAxis[i] = tr;
            ImaginaryAxis[i] = ti;
        }

        k = TAM/2;

        while(k <= j)
        {
            j = j - k;
            k = k/2;
        }

        j = j + k;
    }

    for( l=1; l<=m; l++)
    {
        le = (int)pow(2,l);
        le2 = le/2;
        ur = 1;
        ui = 0;
        sr = cos(M_PI/(double)le2);
        si = (-1)*sin(M_PI/(double)le2);

        for( j=1; j <= le2; j++)
        {
            for( i=j-1; i <= TAM-1; i=i+le)
            {
                aux = i + le2;
                tr = RealAxis[aux]*ur - ImaginaryAxis[aux]*ui;
                ti = RealAxis[aux]*ui + ImaginaryAxis[aux]*ur;
                RealAxis[aux] = RealAxis[i] - tr;
                ImaginaryAxis[aux] = ImaginaryAxis[i] - ti;
                RealAxis[i] = RealAxis[i] + tr;
                ImaginaryAxis[i] = ImaginaryAxis[i] + ti;
            }

            tr = ur;
            ur = tr*sr - ui*si;
            ui = tr*si + ui*sr;
        }
    }
}

void main() {

  int i, j, kernel = 500;
  double *RealAxis, *ImaginaryAxis, *MX;
  double *time;

  RealAxis = (double *)calloc(TAM, sizeof(double));
```

```
  ImaginaryAxis = (double *)calloc(TAM, sizeof(double));
  MX = (double *)calloc(TAM, sizeof(double));
  time = (double *)calloc(TAM, sizeof(double));

  FILE *inputFile, *fourierFile, *kernelFile, *outputFile;

  inputFile = fopen("seno5.txt", "r+");
  fourierFile = fopen("fourier.dat", "w+");
  kernelFile = fopen("kernel.dat", "w+");
  outputFile = fopen("MX.dat", "w+");

  fprintf(fourierFile, "RX\tIX\n");
  fprintf(kernelFile, "RX\tIX\n");
  fprintf(outputFile, "RX\tIX\n");

  for(i = 0; i < TAM/2; i++)
    fscanf(inputFile, "%lf\t%lf", &RealAxis[i], &ImaginaryAxis[i]);

  fourierTransform(RealAxis, ImaginaryAxis);

  for(i = 0; i < TAM; i++){

    RealAxis[i] /= TAM;
    ImaginaryAxis[i] /= -1*TAM;
  }

  for(i = 0; i < TAM; i++)
    fprintf(fourierFile, "%d %lf\n", i, sqrt(pow(RealAxis[i],2) + pow(ImaginaryAxis[i
    ],2)));


  windowing(kernel, RealAxis, ImaginaryAxis);

  for(i = 0; i < TAM/2; i++)
    fprintf(kernelFile, "%d %lf\n", i, sqrt(pow(RealAxis[i],2) + pow(ImaginaryAxis[i],2)
    ));

  fourierTransform(RealAxis, ImaginaryAxis);

  for(i = 0; i < TAM/2; i++){

    MX[i] = 2*sqrt(pow(RealAxis[i],2) + pow(ImaginaryAxis[i],2));
    time[i] = (double) i/(double) TAM;
    fprintf(outputFile, "%lf %lf\n", time[i],MX[i]);
  }
}
```

Listing 1: Filtro implementado.

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

array_file = pd.read_csv("seno5.txt", header = 0, sep='\s+')

fig = plt.figure(figsize=(12,6))

x=np.arange(0,4,1)

axes = fig.add_axes([0.1,0.1,0.8,0.8])

axes.plot(array_file['i'], array_file['seno'], ls='-', marker='o',markersize=1)


axes.set_xlabel('Real axis')
axes.set_ylabel("Imaginary axis")
axes.set_title("Frequency domain")
#axes.legend(loc='upper right')

#plt.show()
plt.grid(linestyle='-', linewidth=0.5)
plt.savefig("plotFrequency.pdf",dpi=600)

#########

array_file = pd.read_csv("fourier.dat", header = 0, sep='\s+')

fig = plt.figure(figsize=(12,6))

x=np.arange(0,4,1)

axes = fig.add_axes([0.1,0.1,0.8,0.8])

axes.plot(array_file['RX'], array_file['IX'], ls='-', marker='o',markersize=1)


axes.set_title("Time Domain")
#axes.legend(loc='upper right')

#plt.show()
plt.grid(linestyle='-', linewidth=0.5)
plt.savefig("plotTime.pdf",dpi=600)


#######



array_file = pd.read_csv("kernel.dat", header = 0, sep='\s+')

fig = plt.figure(figsize=(12,6))

x=np.arange(0,4,1)

axes = fig.add_axes([0.1,0.1,0.8,0.8])
```

```python

axes.plot(array_file['RX'], array_file['IX'], ls='-', marker='o',markersize=1)


axes.set_xlabel('Real axis')
axes.set_ylabel("Imaginary axis")
axes.set_title("Fourier Transform")
#axes.legend(loc='upper right')

#plt.show()
plt.grid(linestyle='-', linewidth=0.5)
plt.savefig("plotFourier.pdf",dpi=600)
```

Listing 2: Visualização dos dados.