Vetores de números complexos, matrizes bidimensionais de números complexos, convolução complexa, FFT (*Fast Fourier Transform*).

Centro de Tecnologia – Departamento de Eletrônica e Computação

Engenharia de Telecomunicações

O BÁSICO DE LINGUAGEM C PARA PROCESSAMENTO DE SINAL

Prof. Fernando DeCastro

```c
#include <stdio.h>
int main() {
  printf("Hello World\n");
  return 0;
}
```

Vetores de números complexos

Números complexos são de particular importância em processamento digital de sinais para sistemas de comunicações porque o TX de um sistema de comunicação digital que utilize modulação QAM (*quadrature amplitude modulation*) ou modulação PSK (*phase shift keying*) ou qualquer outra modulação digital com símbolos em forma de fasor, converte previamente cada palavra binária a ser transmitida através do canal de transmissão em um respectivo número complexo $\mathrm{z} = \mathrm{I} + j\mathrm{Q} = \mathrm{Re}(\mathrm{z}) + j\mathrm{Im}(\mathrm{z})$, denominado **símbolo IQ**.

Cada número complexo $\mathrm{I} + j\mathrm{Q}$ é denominado de símbolo IQ porque a magnitude $A = |\mathrm{I} + j\mathrm{Q}|$ e a fase $\varphi = \angle\{\mathrm{I} + j\mathrm{Q}\}$ representam (i.e., simbolizam) respectivamente a amplitude e a fase da onda eletromagnética (onda EM) que o TX transmite e se propaga no canal de transmissão, onda que transporta as palavras binárias até o RX na forma de "*wavepackets*" de frequência f, com amplitude A e fase φ associada à respectiva palavra binária transmitida (ver slides 2 e 3 de https://www.fccdecastro.com.br/pdf/SCD1_CapIV.pdf).

Conforme vimos no código-fonte https://www.fccdecastro.com.br/CursoC&C++/C/CpxOpRef.c , mostrado nos slides 125 a 133 do Cap II.6, uma declaração **typedef** foi usada para definir o tipo de dado denominado **CPX_R**, a partir de uma estrutura representativa de números complexos cujo **tag** é **ComplexRect**. Os membros da **struct ComplexRect** são variáveis **double Re** e **Im**, e representam respectivamente a parte real e a parte imaginária do número complexo.

Neste contexto, e de maneira similar ao Cap II.6, vamos convencionar que, ao longo do Capítulo III, todo número complexo será representado conforme o tipo de dado definido abaixo pela **struct** com **tag** de nome **Complex**. Os membros da **struct Complex** são variáveis **float Re** e **Im**, e representam respectivamente a parte real e a parte imaginária do número complexo. O tipo de dado definido pela **struct Complex** através da declaração **typedef** é denominado **COMPLEX.**

```
typedef struct Complex{
float Re; //  parte real da variável do tipo COMPLEX
float Im; //  parte imaginária da variável do tipo COMPLEX
} COMPLEX;
```

A declaração abaixo, por exemplo, define a variável **Impedancia** como uma variável de valor complexo:

```
COMPLEX Impedancia;
```

E as declarações abaixo, por exemplo, definem as partes real (resistência) e imaginária (reatância) da variável **Impedancia**:

```
Impedancia.Re = 3; // resistência = 3 ohms
Impedancia.Im = 4; // reatância = 4 ohms
```

Vetores de números complexos

O código https://www.fccdecastro.com.br/CursoC&C++/C/CpxVectAlloc.c, por exemplo, define um ponteiro **COMPLEX *ZDat** e aloca memória no ***heap*** para o vetor **ZDat[N]** de N =NUM_ELEM = 12 elementos de valor complexo, conforme mostrado abaixo. A seguir o programa inicializa o vetor **ZDat** com valores complexos $10e^{j\frac{2\pi n}{N}}$, cuja parte real e imaginária são dadas pela fórmula de Euler $e^{j\frac{2\pi n}{N}} = \cos\left(2\pi\frac{n}{N}\right) + j\sin\left(2\pi\frac{n}{N}\right)$, com $n = 0,1\cdots N-1$ e $j = \sqrt{-1}$ (ver https://en.wikipedia.org/wiki/Euler%27s_formula). Finalizando, o programa imprime o vetor **ZDat[N]** na tela do console no formato retangular {real, imaginário} e no formato polar {módulo, ângulo}.

```
/*****************************************************************
* Declara um ponteiro COMPLEX *ZDat e aloca memoria no heap para o vetor ZDat[N],
* de N=NUM_ELEM = 12 elementos de valor complexo. A seguir o programa inicializa
* o vetor ZDat com valores complexos 10e^(j2*pi*n/N), cuja parte real e imaginaria
* sao dadas pela formula de Euler e^(j2*pi*n/N) =cos(2*pi*n/N)+j*sin(2*pi*n/N),com
* n=0,1...N-1. Finalizando, o programa imprime o vetor ZDat[N]  na tela do console
* no formato retangular {real, imaginario} e no formato polar {modulo, angulo}.
*****************************************************************/
/*****************************************************************
* HEADERS:
*****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<conio.h>
```

Vetores de números complexos

```
17 /*****************************************************************************
18 * DATA TYPE STRUCTURES:
19 *****************************************************************************/
20 typedef struct Complex{
21 float Re;
22 float Im;
23 }COMPLEX;
24 /*****************************************************************************
25 * MACROS:
26 *****************************************************************************/
27 static double sqrarg;
28 #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
29
30 static COMPLEX zarg;
31 /* COMPLEX modulus */
32 #define ZMOD(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 :
          sqrt(SQR(zarg.Re)+SQR(zarg.Im)))
33
34 /*****************************************************************************
35 * PROGRAM DEFINITIONS:
36 *****************************************************************************/
37 #define NUM_ELEM 12
38 /*****************************************************************************
39 * FUNCTION PROTOTYPES:
40 *****************************************************************************/
41 void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem);
42 void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem);
43 double ZAng(COMPLEX Z);
```

Vetores de números complexos

```
44  /*****************************************************************************
45  * main():
46  *****************************************************************************/
47  int main(void){
48
49  COMPLEX *ZDat;
50
51  unsigned register n;
52
53  /* aloca memoria p/ vetor ZDat[] */
54  ZDat=(COMPLEX *)malloc(NUM_ELEM*sizeof(COMPLEX));
55  if(!ZDat){puts("Nao ha memoria para ZDat!");
56  return 1;
57  }
```

Vetores de números complexos

```
58
59  /* atribui valores p/ os elementos do vetor ZDat[] */
60  for(n=0;n<NUM_ELEM;n++){
61  ZDat[n].Re = 10*cos(2*M_PI*n/NUM_ELEM);
62  ZDat[n].Im = 10*sin(2*M_PI*n/NUM_ELEM);
63  }
64
65  /* imprime vetor ZDat[] na tela do console no formato retangular {real,imag} */
66  printf("Re{ZDat}\tIm{ZDat}\n");
67  PrintCpxVec(ZDat, NUM_ELEM);
68
69  /* imprime vetor ZDat[] na tela do console no formato polar {modulo,fase} */
70  printf("\n|ZDat|\t\tang{ZDat}\n");
71  PrintPolarCpxVec (ZDat, NUM_ELEM);
72  }
```

```
73  /*****************************************************************************
74  * FUNC: void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
75  *
76  * DESC: Imprime vetor ZVec[] na tela do console no formato retangular {real,imag}
77  *****************************************************************************/
78  void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
79  unsigned register n;
80
81  for(n=0;n<NumElem;n++){
82  printf("%f\t%f\n",ZVec[n].Re,ZVec[n].Im);
83  }
84  }
```

Vetores de números complexos

```
85  /*************************************************************************
86  * FUNC: void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
87  *
88  * DESC: Imprime vetor ZVec[] na tela do console no formato polar {modulo,fase}
89  *************************************************************************/
90  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
91  unsigned register n;
92
93  for(n=0;n<NumElem;n++){
94  printf("%f\t%lf\n",ZMOD(ZVec[n]),ZAng(ZVec[n])); /* ZMOD(Z) calcula |Z| e
95                                                      eh definida em "MACROS:" acima */
96  }
97  }
98  /*************************************************************************
99  * FUNC: double ZAng(COMPLEX Z)
100 *
101 * DESC: Retorna a fase (angulo) Theta em [graus] da variavel
102 *       COMPLEX Z=Re{Z}+j*Im{Z}=|Z|exp(j*Theta), no intervalo
103 *       -pi < Theta <= pi, com |Z|=sqrt((Z.Re)^2+(Z.Im)^2).
104 *************************************************************************/
105 double ZAng(COMPLEX Z){
106 return (180/M_PI)*atan2(Z.Im,Z.Re);
107 }
108
```

Vetores de números complexos

Execução na tela do console:

```
Re{ZDat}        Im{ZDat}
10.000000       0.000000
8.660254        5.000000
5.000000        8.660254
0.000000        10.000000
-5.000000       8.660254
-8.660254       5.000000
-10.000000      0.000000
-8.660254       -5.000000
-5.000000       -8.660254
-0.000000       -10.000000
5.000000        -8.660254
8.660254        -5.000000

|ZDat|          ang{ZDat}
10.000000       0.000000
10.000000       29.999999
10.000000       60.000001
10.000000       90.000000
10.000000       119.999999
10.000000       150.000001
10.000000       180.000000
10.000000       -150.000001
10.000000       -119.999999
10.000000       -90.000000
10.000000       -60.000001
10.000000       -29.999999
```

Vetores de números complexos

Observe que se definirmos o tipo COMPLEX com partes real e imaginária do tipo **double**, conforme mostrado abaixo

```
typedef struct Complex{
double Re;
double Im;
}COMPLEX;
```

verifica-se uma melhora da conformidade da precisão numérica do resultado em formato polar **ang{Zdat}**, resultado este mostrado na execução na tela do console ao lado. No entanto, o uso de variáveis **double** no lugar de variáveis **float** tem o efeito indesejado de dobrar o uso de memoria. Por esta razão, as **operações** entre números complexos devem ser preferencialmente efetuadas com os números expressos no formato retangular {real,imag} e não no formato polar {modulo, angulo}.

A abordagem numérica e a verificação de conformidade de resultados para operações com números complexos que foi adotada neste exemplo – e que adotaremos ao longo de todo o Capítulo III – é baseada nas seguintes referências:

[1]https://www.fccdecastro.com.br/CursoC&C++/C++%20algorithms%20for%20DSP%20-%20Embree.pdf

[2] https://www.fccdecastro.com.br/CursoC&C++/C Algorithms for real time DSP - Embree.pdf

[3]https://www.fccdecastro.com.br/CursoC&C++/Numerical%20Recipes%20in%20C%202nd%20-%20%20Press.pdf

[3A]https://www.fccdecastro.com.br/CursoC&C++/NRC206H_HLP/index.html

[4]https://www.fccdecastro.com.br/CursoC&C++/MathcadUsersGuide.pdf


```
Re{ZDat}        Im{ZDat}
10.000000       0.000000
8.660254        5.000000
5.000000        8.660254
0.000000        10.000000
-5.000000       8.660254
-8.660254       5.000000
-10.000000      0.000000
-8.660254       -5.000000
-5.000000       -8.660254
-0.000000       -10.000000
5.000000        -8.660254
8.660254        -5.000000

|ZDat|          ang{ZDat}
10.000000       0.000000
10.000000       30.000000
10.000000       60.000000
10.000000       90.000000
10.000000       120.000000
10.000000       150.000000
10.000000       180.000000
10.000000       -150.000000
10.000000       -120.000000
10.000000       -90.000000
10.000000       -60.000000
10.000000       -30.000000
```

Vetores de números complexos

Alternativamente, a alocação de memória no ***heap*** para o código-fonte do slide 3 pode ser feita da seguinte maneira: Declara-se um ponteiro float *Dat e aloca-se memoria no *heap* para o vetor Dat[2*N], de 2*N=2*NUM_ELEM = 24 elementos de valor real. A seguir o programa faz o cast de *float Dat para *COMPLEX ZDat e inicializa o vetor ZDat com N valores complexos 10e^(j2*pi*n/N), conforme segue (https://www.fccdecastro.com.br/CursoC&C++/C/CpxVectAlloc1.c):

```
/*****************************************************************
* Declara um ponteiro float *Dat e aloca memoria no heap para o vetor Dat[2*N],
* de 2*N=2*NUM_ELEM = 24 elementos de valor real. A seguir o programa faz o cast
* de *float Dat para *COMPLEX ZDat e inicializa o vetor ZDat com N valores compl
* 10e^(j2*pi*n/N), cuja parte real e imaginaria sao dadas pela formula de
* Euler e^(j2*pi*n/N) =cos(2*pi*n/N)+j*sin(2*pi*n/N),com n=0,1...N-1.
* Finalizando, o programa imprime o vetor ZDat[N]  na tela do console
* no formato retangular {real, imaginario} e no formato polar {modulo, angulo}.
*****************************************************************/
/*****************************************************************
* HEADERS:
*****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<conio.h>
/*****************************************************************
* DATA TYPE STRUCTURES:
*****************************************************************/
typedef struct Complex{
float Re;
float Im;
}COMPLEX;
```

Vetores de números complexos

```
25  /*****************************************************************************
26  * MACROS:
27  *****************************************************************************/
28  static double sqrarg;
29  #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
30
31  static COMPLEX zarg;
32  /* COMPLEX modulus */
33  #define ZMOD(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 :
                     sqrt(SQR(zarg.Re)+SQR(zarg.Im)))
34
35  /*****************************************************************************
36  * PROGRAM DEFINITIONS:
37  *****************************************************************************/
38  #define NUM_ELEM 12 // numero de elementos complexos no vetor ZDat
39  /*****************************************************************************
40  * FUNCTION PROTOTYPES:
41  *****************************************************************************/
42  void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem);
43  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem);
44  double ZAng(COMPLEX Z);
```

Vetores de números complexos

```
45  /*************************************************************************
46  * main():
47  *************************************************************************/
48  int main(void){
49
50  COMPLEX *ZDat;
51  float *Dat;
52
53  unsigned register n;
54
55  /* aloca memoria p/ vetor float Dat[] em que os elementos
56  de indice par armazenam a parte real e os elementos de indice
57   impar armazenam a parte imaginaria */
58  Dat=(float *)malloc(NUM_ELEM*sizeof(COMPLEX));
59  if(!Dat){puts("Nao ha memoria para Dat!");
60  return 1;
61  }
62
63  ZDat=(COMPLEX *)Dat; /* cast de *float Dat para *COMPLEX ZDat. O cast
64  OBRIGATORIAMENTE deve ser feito apos malloc() atribuir a Dat o endereço
65  inicial da area de armazenamento no heap */
66
67  /* atribui valores p/ os elementos do vetor ZDat[] */
68  for(n=0;n<NUM_ELEM;n++){
69  ZDat[n].Re = 10*cos(2*M_PI*n/NUM_ELEM);
70  ZDat[n].Im = 10*sin(2*M_PI*n/NUM_ELEM);
71  }
```

Vetores de números complexos

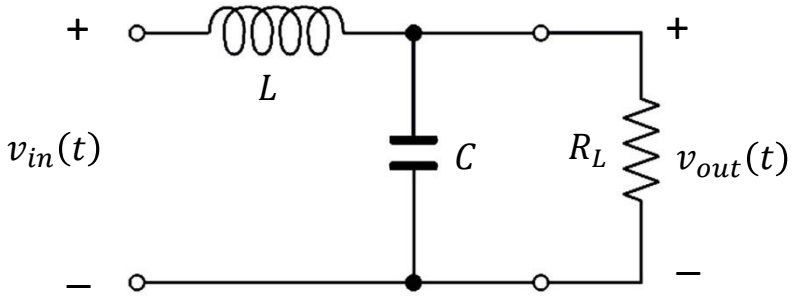
```
72
73  /* imprime vetor ZDat[] na tela do console no formato retangular {real,imag} */
74  printf("Re{ZDat}\tIm{ZDat}\n");
75  PrintCpxVec(ZDat, NUM_ELEM);
76
77  /* imprime vetor ZDat[] na tela do console no formato polar {modulo,fase} */
78  printf("\n|ZDat|\t\tang{ZDat}\n");
79  PrintPolarCpxVec (ZDat, NUM_ELEM);
80  }
81  /*****************************************************************************
82  * FUNC: void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
83  *
84  * DESC: Imprime vetor ZVec[] na tela do console no formato retangular {real,imag}
85  *****************************************************************************/
86  void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
87  unsigned register n;
88
89  for(n=0;n<NumElem;n++){
90  printf("%f\t%f\n",ZVec[n].Re,ZVec[n].Im);
91  }
92  }
```

Vetores de números complexos

```
93  /*************************************************************************
94  * FUNC: void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
95  *
96  * DESC: Imprime vetor ZVec[] na tela do console no formato polar {modulo,fase}
97  *************************************************************************/
98  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
99  unsigned register n;
100
101 for(n=0;n<NumElem;n++){
102 printf("%f\t%lf\n",ZMOD(ZVec[n]),ZAng(ZVec[n])); /* ZMOD(Z) calcula |Z| e
103                                                    eh definida em "MACROS:" acima */
104 }
105 }
106 /*************************************************************************
107 * FUNC: double ZAng(COMPLEX Z)
108 *
109 * DESC: Retorna a fase (angulo) Theta em [graus] da variavel
110 *       COMPLEX Z=Re{Z}+j*Im{Z}=|Z|exp(j*Theta), no intervalo
111 *       -pi < Theta <= pi, com |Z|=sqrt((Z.Re)^2+(Z.Im)^2).
112 *************************************************************************/
113 double ZAng(COMPLEX Z){
114 return (180/M_PI)*atan2(Z.Im,Z.Re);
115 }
116
```

Vetores de números complexos

Exemplo 1: O diagrama abaixo mostra um filtro passa-baixa LC cuja resistência de carga R_L representa a resistência de entrada do bloco funcional que segue o filtro.



A função de transferência $H(f) = V_{out}(f)/V_{in}(f)$ deste filtro é dada por (ver slides 2 a 4 de https://www.fccdecastro.com.br/pdf/SS_Aula5&6_26032020.pdf):

$$H(f) = \frac{1}{(1 - LC(2\pi f)^2) + j\frac{2\pi fL}{R_L}}$$

onde f varia no intervalo $fmin < f < fmax$.

Pede-se: Escreva o código fonte C para um programa que determine os valores complexos da $H(f)$ acima no intervalo $fmin < f < fmax$ e que atenda às especificações abaixo. Valide os resultados com o software MathCad.

(a) Os dados de entrada do programa são lidos da linha de comando. Os 7 argumentos da linha de comando do programa são (nesta ordem) os valores de **L [µH]**, **C[pF]**, $\boldsymbol{R_L}$**[Ω]**, $\boldsymbol{fmin}$ **[MHz]**, $\boldsymbol{fmax}$ **[MHz]**, o número de pontos **Np** em que $H(f)$ é determinada no intervalo $fmin < f < fmax$ e o nome do arquivo texto **"OutFile"** em que os valores de f , $\mathrm{Re}\{H(f)\}$ e $\mathrm{Im}\{H(f)\}$ serão gravados linha a linha no referido arquivo texto, cada linha correspondendo a um dos Np valores de f.
(b) Quando argc resultar diferente de 8 (argv[0] é o nome do executável) para a entrada de argumentos na linha de comando o programa imprime na tela do console um texto de "help" especificando e descrevendo o que faz o programa e quais são os argumentos da linha de comando.
(c) Determinar $H(f)$ conforme equação acima em Np pontos uniformemente distribuídos no intervalo $fmin < f < fmax$ e imprimir na tela do console no formato polar.
(d) Gravar $H(f)$ determinada em (c) no arquivo texto cujo nome é especificado no argumento da linha de comando "OutFile". O arquivo deve ter 3 colunas, cada coluna respectivamente representando f , $\mathrm{Re}\{H(f)\}$ e $\mathrm{Im}\{H(f)\}$.

Vetores de números complexos

Solução: O código fonte https://www.fccdecastro.com.br/CursoC&C++/C/FuncTransf.c, listado abaixo, é uma possível solução para o problema posto no enunciado no slide anterior.

```
1  /*************************************************************************
2  * Determina os Np valores complexos de H(f)=1/((1-L*C*(2*pi*f)^2)+j(2*pi*f*L/RL))
3  * no intervalo fmin<f<fmax e grava em arquivo texto de nome dado por argv[7].
4  *************************************************************************/
5  /*************************************************************************
6  * HEADERS:
7  *************************************************************************/
8  #include<stdio.h>
9  #include<stdlib.h>
10 #include<ctype.h>
11 #include<math.h>
12 #include<conio.h>
13 /*************************************************************************
14 * DATA TYPE STRUCTURES:
15 *************************************************************************/
16 typedef struct Complex{
17 float Re;
18 float Im;
19 }COMPLEX;
20 /*************************************************************************
21 * MACROS:
22 *************************************************************************/
23 static double sqrarg;
24 #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
25
```

Vetores de números complexos

```
26  static COMPLEX zarg;
27  /* COMPLEX modulus |a| */
28  #define ZMOD(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 :
    sqrt(SQR(zarg.Re)+SQR(zarg.Im)))
29
30  /* COMPLEX norm */
31  #define ZNRM(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 :
                (SQR(zarg.Re)+SQR(zarg.Im)))
32
33  static COMPLEX zarg0,zarg1;
34  /* Multiply two COMPLEX values z0 and z1. Put the result in zR. */
35  #define CMPY(z0,z1,zR) {zR.Re =(zarg0.Re=(z0.Re))* (zarg1.Re=(z1.Re))-
                        (zarg0.Im=(z0.Im))* (zarg1.Im=(z1.Im));
                        zR.Im =(zarg0.Re)* (zarg1.Im)+ (zarg0.Im)*(zarg1.Re);}
```

Vetores de números complexos

```
36
37
38  /*************************************************************************
39  * PROGRAM DEFINITIONS:
40  *************************************************************************/
41  #define BUFSIZE 0x8000 /* in&out file buffer size */
42  #define MAXLINESIZE 65536 /* max line size for  GetSheetNumLines() and NumberOfFloatsInLine() */
43
44  /*************************************************************************
45  * FUNCTION PROTOTYPES:
46  *************************************************************************/
47  void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem);
48  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem);
49  double ZAng(COMPLEX Z);
50  COMPLEX CAdd(COMPLEX A, COMPLEX B);
51  COMPLEX CSub(COMPLEX A, COMPLEX B);
52  COMPLEX CMul(COMPLEX A, COMPLEX B);
53  COMPLEX Cpx(float Real, float Imag);
54  COMPLEX Conj(COMPLEX Z);
55  COMPLEX CDiv(COMPLEX A, COMPLEX B);
56  float CAbs(COMPLEX Z);
57  COMPLEX CSqrt(COMPLEX Z);
58  COMPLEX RCMul(float x, COMPLEX A);
59  void Hf(COMPLEX *H_f,float L,float C,float RL,float fmin, float fmax,unsigned Np);
60  void FPrintfH_f(COMPLEX *H_f, char *Buffer,float fmin, float fmax, unsigned Np, char *OutFile);
```

Vetores de números complexos

```
61  /*************************************************************************
62  * main():
63  *************************************************************************/
64  void main(int argc, char *argv[]){
65
66
67  COMPLEX *H_f;
68  float L,C,RL,fmin,fmax;
69  unsigned Np;
70  char *OutFile;
71  char *Buffer;
72
73  unsigned register n;
74
75  /* Help da linha de comando */
76  if(argc!=8){
77
78  puts("\nDetermina os Np valores complexos de H(f)=1/((1-L*C*(2*pi*f)^2)+j(2*pi*f*L/RL))");
79  puts("no intervalo fmin<f<fmax e grava em arquivo texto de nome dado pelo argumento");
80  puts("'OutFile'da linha de comando abaixo especificada.");
81  /*    0    1     2     3       4         5         6  7   */
82  puts("Uso: L[uH] C[pF] RL[ohm] fmin[MHz] fmax[MHz] Np OutFile");
83  exit(1);
84  }
85
86  /* Atribuicao dos argumentos da linha de comando 'as respctivas variaveis */
87  L=atof(argv[1])*1E-6; // indutancia L convertida de uH p/ H
88  C=atof(argv[2])*1E-12;// capacitancia C convertida de pF p/ F
89  RL=atof(argv[3]);// resistencia RL em ohms
90  fmin=atof(argv[4])*1E6;// frequencia minima convertida de MHz p/ Hz
91  fmax=atof(argv[5])*1E6;// frequencia maxima convertida de MHz p/ Hz
92  Np=atoi(argv[6]); /* número de pontos Np em que H(f) eh determinada
93                       no intervalo fmin<f<fmax */
94  OutFile=argv[7]; /* nome do arquivo texto de 3 colunas em que os
95                      valores de f, Re{H(f)} e Im{H(f)} serao gravados */
```

Vetores de números complexos

```
 96
 97  //printf("L=%8.6g\tC=%8.6g\tRL=%8.6f\tfmin=%8.6f\tfmax=%8.6f\tNp=%8u\tOutFile=%s\n",L,C
 98
 99  /* alloc I/O file buffer */
100  Buffer=(char *)malloc(BUFSIZE*sizeof(char));
101  if(!Buffer){puts("Nao ha memoria para Buffer!");exit(1);}
102
103  /* aloca memoria p/ vetor COMPLEX H_f[] de Np elementos*/
104  H_f=(COMPLEX *)malloc(Np*sizeof(COMPLEX));
105  if(!H_f){puts("Nao ha memoria para H_f!");exit(1);}
106
107  /* determina H_f[] */
108  Hf(H_f,L,C,RL,fmin,fmax,Np);
109
110  #if(0)
111  /* imprime vetor H_f[] na tela do console no formato retangular {real,imag} */
112  printf("Re{H(f)}\tIm{H(f)}\n");
113  PrintCpxVec(H_f, Np);
114  #endif
115
116  /* imprime vetor H_f[] na tela do console no formato polar {modulo,fase} */
117  printf("\n|H(f)|\t\tang{H(f)}\n");
118  PrintPolarCpxVec (H_f, Np);
119
120  /* Grava linha a linha no arquivo texto de 3 colunas OutFile os valores
121  respectivos de f, Re{H(f)} e Im{H(f)},cada linha correspondendo a um dos Np
122  valores de f no intervalo fmin <= f <= fmax */
123  FPrintfH_f(H_f,Buffer,fmin,fmax,Np,OutFile);
124
125  }
```

Vetores de números complexos

```
126 /*************************************************************************
127 * FUNC: void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
128 *
129 * DESC: Imprime vetor ZVec[] na tela do console no formato retangular {real,imag}
130 *************************************************************************/
131 void PrintCpxVec (COMPLEX *ZVec, unsigned NumElem){
132 unsigned register n;
133
134 for(n=0;n<NumElem;n++){
135 printf("%f\t%f\n",ZVec[n].Re,ZVec[n].Im);
136 }
137 }
138 /*************************************************************************
139 * FUNC: void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
140 *
141 * DESC: Imprime vetor ZVec[] na tela do console no formato polar {modulo,fase}
142 *************************************************************************/
143 void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
144 unsigned register n;
145
146 for(n=0;n<NumElem;n++){
147 printf("%f\t%lf\n",ZMOD(ZVec[n]),ZAng(ZVec[n])); /* ZMOD(Z) calcula |Z| e
148                                                      eh definida em "MACROS:" acima */
149 }
150 }
```

Vetores de números complexos

```
151 /*************************************************************************
152 * FUNC: double ZAng(COMPLEX Z)
153 *
154 * DESC: Retorna a fase (angulo) Theta em [graus] da variavel
155 *       COMPLEX Z=Re{Z}+j*Im{Z}=|Z|exp(j*Theta), no intervalo
156 *       -pi < Theta <= pi, com |Z|=sqrt((Z.Re)^2+(Z.Im)^2).
157 *************************************************************************/
158 double ZAng(COMPLEX Z){
159 return (180/M_PI)*atan2(Z.Im,Z.Re);
160 }
161 /*************************************************************************
162 * FUNC: COMPLEX CAdd(COMPLEX A, COMPLEX B)
163 *
164 * DESC: Retorna A + B.
165 *************************************************************************/
166  COMPLEX CAdd(COMPLEX A, COMPLEX B)
167  {
168  COMPLEX C;
169  C.Re=A.Re+B.Re;
170  C.Im=A.Im+B.Im;
171  return C;
172  }
```

Vetores de números complexos

```
173 /*************************************************************************
174 * FUNC: COMPLEX CSub(COMPLEX A, COMPLEX B)
175 *
176 * DESC: Retorna A - B.
177 *************************************************************************/
178  COMPLEX CSub(COMPLEX A, COMPLEX B)
179  {
180  COMPLEX C;
181  C.Re=A.Re-B.Re;
182  C.Im=A.Im-B.Im;
183  return C;
184  }
185 /*************************************************************************
186 * FUNC: COMPLEX CMul(COMPLEX A, COMPLEX B)
187 *
188 * DESC: Retorna A * B.
189 *************************************************************************/
190  COMPLEX CMul(COMPLEX A, COMPLEX B)
191  {
192  COMPLEX C;
193  C.Re=A.Re*B.Re-A.Im*B.Im;
194  C.Im=A.Im*B.Re+A.Re*B.Im;
195  return C;
196  }
```

Vetores de números complexos

```
197 /*****************************************************************************
198 * FUNC: COMPLEX Cpx(float Real, float Imag)
199 *
200 * DESC: Retorna Real+j*Imag.
201 *****************************************************************************/
202  COMPLEX Cpx(float Real, float Imag)
203  {
204  COMPLEX C;
205  C.Re=Real;
206  C.Im=Imag;
207  return C;
208  }
209
210 /*****************************************************************************
211 * FUNC: COMPLEX Conj(COMPLEX Z)
212 *
213 * DESC: Retorna o conjugado do valor de Z.
214 *****************************************************************************/
215  COMPLEX Conj(COMPLEX Z)
216  {
217  COMPLEX C;
218  C.Re=Z.Re;
219  C.Im =-Z.Im;
220  return C;
221  }
```

Vetores de números complexos

```
222 /*************************************************************************
223 * FUNC: COMPLEX CDiv(COMPLEX A, COMPLEX B)
224 *
225 * DESC: Retorna A/B.
226 *************************************************************************/
227  COMPLEX CDiv(COMPLEX A, COMPLEX B)
228  {
229  COMPLEX C;
230  float r,den;
231  if (fabs(B.Re) >= fabs(B.Im)) {
232  r=B.Im/B.Re;
233  den=B.Re+r*B.Im;
234  C.Re=(A.Re+r*A.Im)/den;
235  C.Im=(A.Im-r*A.Re)/den;
236  } else {
237  r=B.Re/B.Im;
238  den=B.Im+r*B.Re;
239  C.Re=(A.Re*r+A.Im)/den;
240  C.Im=(A.Im*r-A.Re)/den;
241  }
242  return C;
243  }
244
```

Vetores de números complexos

```
245 /*************************************************************************
246 * FUNC: float CAbs(COMPLEX Z)
247 *
248 * DESC: Retorna |Z|.
249 *************************************************************************/
250  float CAbs(COMPLEX Z)
251  {
252  float x,y,ans,temp;
253  x=fabs(Z.Re);
254  y=fabs(Z.Im);
255  if (x == 0.0)
256  ans=y;
257  else if (y == 0.0)
258  ans=x;
259  else if (x > y) {
260  temp=y/x;
261  ans=x*sqrt(1.0+temp*temp);
262  } else {
263  temp=x/y;
264  ans=y*sqrt(1.0+temp*temp);
265  }
266  return ans;
267  }
```

Vetores de números complexos

```
/*****************************************************************************
* FUNC: COMPLEX CSqrt(COMPLEX Z)
*
* DESC: Retorna sqrt(Z).
*****************************************************************************/
 COMPLEX CSqrt(COMPLEX Z)
 {
 COMPLEX C;
 float x,y,w,r;
 if ((Z.Re == 0.0) && (Z.Im == 0.0)) {
 C.Re=0.0;
 C.Im=0.0;
 return C;
 } else {
 x=fabs(Z.Re);
 y=fabs(Z.Im);
 if (x >= y) {
 r=y/x;
 w=sqrt(x)*sqrt(0.5*(1.0+sqrt(1.0+r*r)));
 } else {
 r=x/y;
 w=sqrt(y)*sqrt(0.5*(r+sqrt(1.0+r*r)));
 }
 if (Z.Re >= 0.0) {
 C.Re=w;
 C.Im=Z.Im/(2.0*w);
 } else {
 C.Im=(Z.Im >= 0) ? w :-w;
 C.Re=Z.Im/(2.0*C.Im);
 }
 return C;
 }
 }
```

Vetores de números complexos

```
301 /*************************************************************************
302  * FUNC: COMPLEX RCMul(float x, COMPLEX A)
303  *
304  * DESC: Retorna x*A.
305  *************************************************************************/
306 COMPLEX RCMul(float x, COMPLEX A)
307  {
308  COMPLEX C;
309  C.Re=x*A.Re;
310  C.Im=x*A.Im;
311  return C;
312  }
```

Vetores de números complexos

```
313 /*************************************************************************
314 * FUNC: void Hf(float *H_f,float L,float C,float RL,float fmin, float fmax,unsigned Np)
315 *
316 * DESC: Retorna Np valores de H(f)=1/((1-L*C*(2*pi*f)^2)+j(2*pi*f*L/RL))
317 *       no intervalo fmin<f<fmax.
318 *************************************************************************/
319 void Hf(COMPLEX *H_f,float L,float C,float RL,float fmin, float fmax,unsigned Np)
320  {
321
322  float Delta_f;
323  float f;
324  unsigned register n;
325  COMPLEX Num;
326  COMPLEX Den;
327
328 Num=Cpx(1,0); // numerador = 1+j0
329 Delta_f=(fmax-fmin)/(Np-1); // intervalo entre dois pontos no dominio f
330
331  for(n=0;n<Np;n++){
332     f=fmin+n*Delta_f;// determina o valor de f para o n-esimo ponto
333     Den=Cpx((1-L*C*SQR(2*M_PI*f)),(2*M_PI*f*L/RL)); // denominador
334     H_f[n]= CDiv(Num,Den); // calcula H(f)
335     }
336
337  }
```

Vetores de números complexos

```
/*************************************************************************
* FUNC: void FPrintfH_f(COMPLEX *H_f, char *Buffer,float fmin, float fmax, unsigned Np, char *OutFile)
*
* DESC: Grava linha a linha no arquivo texto de 3 colunas OutFile os valores
* respectivos de f, Re{H(f)} e Im{H(f)},cada linha correspondendo a um dos Np
* valores de f no intervalo fmin <= f <= fmax.
*************************************************************************/
void FPrintfH_f(COMPLEX *H_f, char *Buffer,float fmin, float fmax, unsigned Np, char *OutFile){

FILE *Fp;
unsigned register n;
float f, Delta_f;

/* abre o stream Fp p/ o arquivo de saida OutFile */
if((Fp=fopen(OutFile,"wt"))==NULL)
{printf("Nao consigo abrir %s!",OutFile);exit(1);}

/* seta buffer de I/O p/ o stream Fp */
if (setvbuf(Fp,Buffer, _IOFBF, BUFSIZE) != 0)
{printf("Nao consigo setar o buffer de I/O para %s!",OutFile);exit(1);}

Delta_f=(fmax-fmin)/(Np-1); // intervalo entre dois pontos no dominio f

/* escreve f, Re{H(f)} e Im{H(f)} em OutFile */
for(n=0;n<Np;n++){
    f=fmin+n*Delta_f;// determina o valor de f para o n-esimo ponto
    fprintf(Fp,"%8.6g\t%8.6g\t%8.6g\n",f,H_f[n].Re,H_f[n].Im);
    }

/* fecha o stream Fp */
fclose(Fp);
}
```

Vetores de números complexos

Execução na tela do console – tela de "help" resultante ao se digitar o nome do executável (FuncTransf.exe) com a linha de comando sem argumentos :

```
Determina os Np valores complexos de H(f)=1/((1-L*C*(2*pi*f)^2)+j(2*pi*f*L/RL))
no intervalo fmin<f<fmax e grava em arquivo texto de nome dado pelo argumento
'OutFile'da linha de comando abaixo especificada.
Uso: L[uH] C[pF] RL[ohm] fmin[MHz] fmax[MHz] Np OutFile
```

Execução na tela do console – linha de comando conforme especificada abaixo:

FuncTransf 120 410 777 0.1 2.5 12 tmp.txt

Resultado na tela do console :

```
|H(f))|          ang{H(f)}
1.014851         -5.651574
1.161915         -21.023348
1.465580         -49.711387
1.351709         -98.225901
0.792320         -131.592986
0.475941         -146.632190
0.315759         -154.421177
0.225528         -159.137693
0.169673         -162.311098
0.132585         -164.603206
0.106629         -166.343490
0.087714         -167.714182
```

Arquivo tmp.txt gerado na execução de FuncTransf.exe e aberto no aplicativo bloco de notas (notepad.exe) :

```
  100000          1.00992          -0.0999412
  318182          1.08457          -0.416835
  536364           0.9477          -1.11794
  754545         -0.193398           -1.3378
  972727         -0.525969          -0.592559
1.19091e+006     -0.397486          -0.261773
1.40909e+006     -0.284812          -0.13633
1.62727e+006     -0.210742          -0.0803157
1.84545e+006     -0.161651          -0.051555
2.06364e+006     -0.127827          -0.0352016
2.28182e+006     -0.103615          -0.0251752
2.5e+006         -0.085705          -0.0186645
```

Vetores de números complexos

Para validar os resultados da execução do programa FuncTransf.exe com o software MathCad vamos utilizar o *script* em https://www.fccdecastro.com.br/ZIP/TesteH(f).zip, conforme segue:

Gerando os valores complexos de H(f) definida pela equação (1) abaixo com parâmetros dados por

$L := 120\mu H \quad C := 410pF \quad RL := 777\Omega \quad fmin := 100KHz \quad fmax := 2.5MHz \quad Np := 12 \quad OutFile := "tmp.txt"$

$$H(f) := \frac{1}{\left[1 - L \cdot C \cdot (2 \cdot \pi \cdot f)^2\right] + j \cdot \frac{2 \cdot \pi \cdot f \cdot L}{RL}} \qquad (1)$$

notando que o parâmetros acima correspondem à linha de comando do programa FuncTransf.exe especificada por **FuncTransf 120 410 777 0.1 2.5 12 tmp.txt**

Definindo os valores de frequência f_n para determinação de $H(f_n)$, com $n := 0, 1 .. Np - 1$:

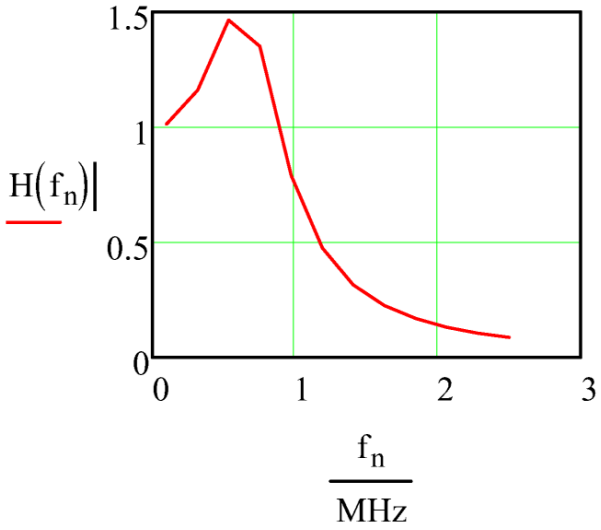
$$Delta\_f := \frac{(fmax - fmin)}{Np - 1} \qquad f_n := fmin + n \cdot Delta\_f$$

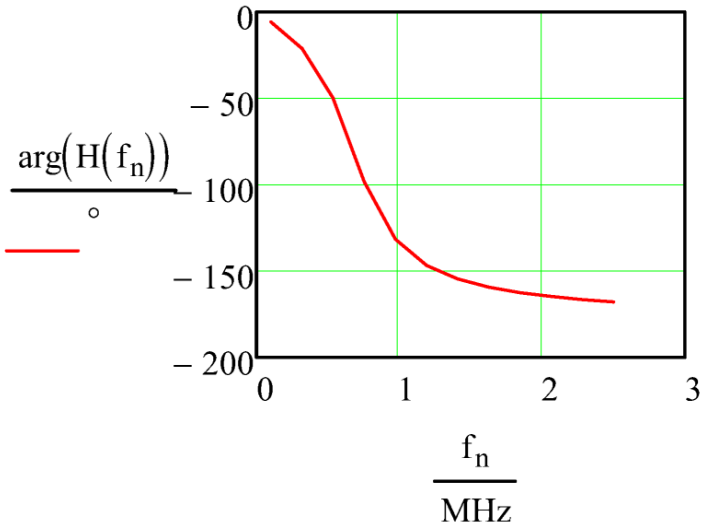
Vetores de números complexos

Determinando $H(f_n)$ e plotando $|H(f_n)|$ e $\arg(H(f_n))°$:

$H(f_n) =$

	0
0	1.01-0.1i
1	1.085-0.417i
2	0.948-1.118i
3	-0.193-1.338i
4	-0.526-0.593i
5	-0.397-0.262i
6	-0.285-0.136i
7	-0.211-0.08i
8	-0.162-0.052i
9	-0.128-0.035i
10	-0.104-0.025i
11	-0.086-0.019i





Vetores de números complexos

A seguir, o comando Dat := READPRN(OutFile) lê o arquivo de 3 colunas tmp.txt gerado por FuncTransf.exe, com linha de comando **FuncTransf 120 410 777 0.1 2.5 12 tmp.txt.** As 3 colunas de tmp.txt representam respectivamente f, Re{H(f)} e Im{H(f)}. Os valores lidos são atribuídos à matriz Dat., que resulta conforme segue.

Dat =

	0	1	2
0	$1 \cdot 10^5$	1.01	-0.1
1	$3.182 \cdot 10^5$	1.085	-0.417
2	$5.364 \cdot 10^5$	0.948	-1.118
3	$7.545 \cdot 10^5$	-0.193	-1.338
4	$9.727 \cdot 10^5$	-0.526	-0.593
5	$1.191 \cdot 10^6$	-0.397	-0.262
6	$1.409 \cdot 10^6$	-0.285	-0.136
7	$1.627 \cdot 10^6$	-0.211	-0.08
8	$1.845 \cdot 10^6$	-0.162	-0.052
9	$2.064 \cdot 10^6$	-0.128	-0.035
10	$2.282 \cdot 10^6$	-0.104	-0.025
11	$2.5 \cdot 10^6$	-0.086	-0.019

Vetores de números complexos

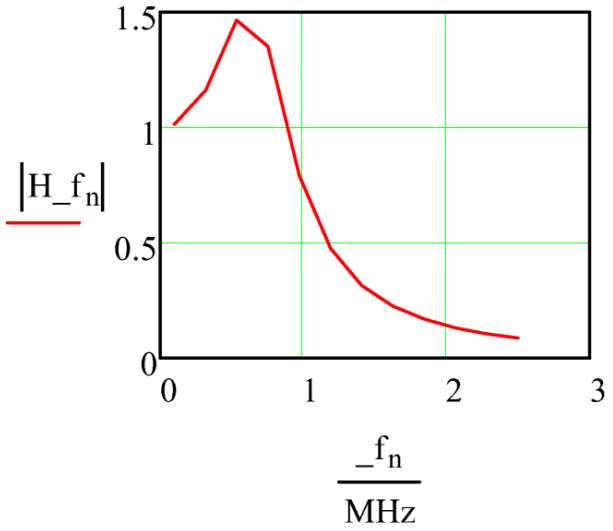
O próximo passo consiste na declaração $H\_f := Dat^{\langle 1 \rangle} + j \cdot Dat^{\langle 2 \rangle}$, que atribui à cada elemento do vetor H_f os valores das colunas 1 e 2 na respectiva linha da matriz Dat convertidos para valor complexo.

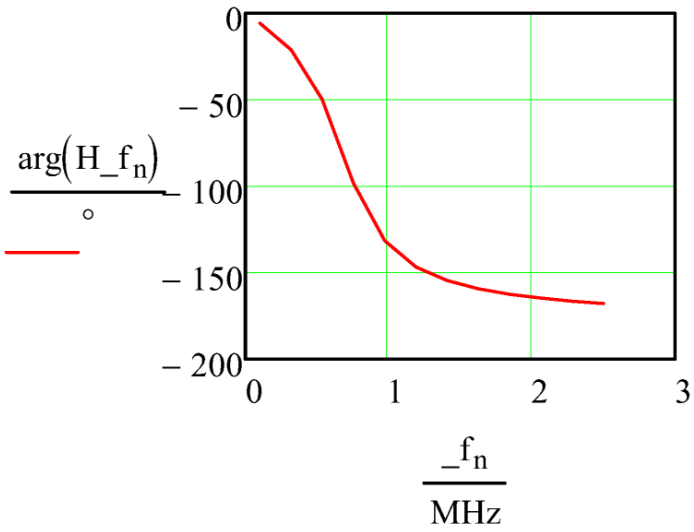
E, finalizando as operações, efetua-se a atribuição $\_f := Dat^{\langle 0 \rangle}\ Hz$, que atribui a cada elemento do vetor $\_f$ o valor da coluna 0 na respectiva linha da matriz Dat, e insere a unidade Hz. O índice n para H_f_n passa a ser definido por $n := 0, 1 .. length(\_f) - 1$.

Explicitando e plotando H_f_n gerada por FuncTransf.exe :

$H\_f_n =$

	0
0	1.01-0.1i
1	1.085-0.417i
2	0.948-1.118i
3	-0.193-1.338i
4	-0.526-0.593i
5	-0.397-0.262i
6	-0.285-0.136i
7	-0.211-0.08i
8	-0.162-0.052i
9	-0.128-0.035i
10	-0.104-0.025i
11	-0.086-0.019i

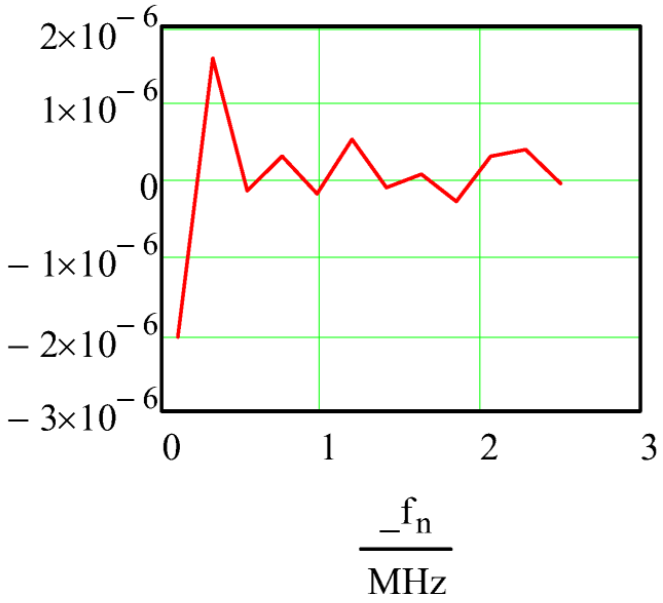




Vetores de números complexos

Subtraindo da parte real de $H(f_n)$ a parte real de H_f_n e plotando para efeito de verificar a conformidade entre o resultado da H(f) gerada neste script MathCad ($H(f_n)$) e o resultado da H(f) gerada pelo programa FuncTransf.exe (H_f_n):

$\underline{Re(H(f_n))} - Re(H\_f_n)$

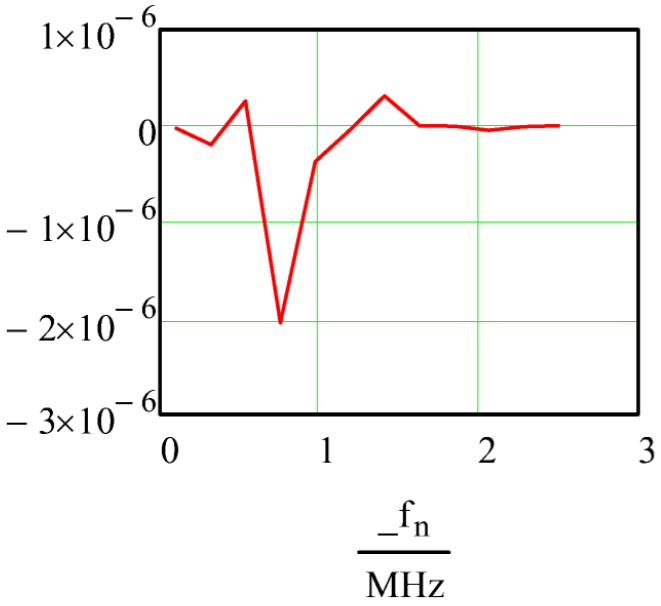


Observa-se que há a conformidade entre as partes reais.

Vetores de números complexos

Subtraindo da parte imaginária de $H(f_n)$ a parte imaginária de H_f_n e plotando para efeito de verificar a conformidade entre o resultado da H(f) gerada neste script MathCad ($H(f_n)$) e o resultado da H(f) gerada pelo programa FuncTransf.exe (H_f_n):

$\underline{Im(H(f_n))} - Im(H\_f_n)$



Observa-se que há a conformidade entre as partes imaginárias.

Dado que há conformidade tanto entre as partes reais como entre as partes imaginárias, considera-se que o código fonte FuncTransf.c está validado.

Vetores de números complexos

Exemplo 2: O *script* MathCad em https://www.fccdecastro.com.br/ZIP/GeraEuler.zip mostrado a seguir gera $Np = 12$ valores complexos a partir da expressão $Euler_n = Ae^{j2\pi\frac{n}{Np}}$, onde $n = 0,1 \cdots Np - 1$ e $A = 10$. Os valores complexos gerados são gravados no formato retangular no arquivo texto "euler.txt" de duas colunas numéricas {real imag}.

Parâmetros de entrada:

$A := 10 \quad Np := 12 \quad OutFile := "euler.txt"$

Determinando os Np elementos do vetor $Euler_n$ e mostrando no formato retangular e polar :

$n := 0, 1 .. Np - 1 \quad Euler_n := A \cdot e^{j \cdot 2 \cdot \pi \cdot \frac{n}{Np}}$

Euler =

	0
0	10
1	8.66+5i
2	5+8.66i
3	10i
4	-5+8.66i
5	-8.66+5i
6	-10
7	-8.66-5i
8	-5-8.66i
9	-10i
10	5-8.66i
11	8.66-5i

$|Euler_n| = \quad \frac{\arg(Euler_n)}{\circ} =$

10	0
10	30
10	60
10	90
10	120
10	150
10	180
10	-150
10	-120
10	-90
10	-60
10	-30

Vetores de números complexos

Atribuindo os Np valores da parte real do vetor $Euler_n$ à coluna 0 da matriz Dat[Np][2]: $Dat^{\langle 0 \rangle} := Re(Euler)$

Atribuindo os Np valores da parte imaginária do vetor $Euler_n$ à coluna 1 da matriz Dat[Np][2]: $Dat^{\langle 1 \rangle} := Im(Euler)$

Dat =

	0	1
0	10	0
1	8.66	5
2	5	8.66
3	0	10
4	-5	8.66
5	-8.66	5
6	-10	0
7	-8.66	-5
8	-5	-8.66
9	0	-10
10	5	-8.66
11	8.66	-5

Especificando o formato de gravação de cada uma das 2 colunas do arquivo de saída $OutFile = "euler.txt"$:

$PRNPRECISION := 6$ → representa o número de dígitos significativos a serem usados quando se escreve para um arquivo texto com a função WRITEPRN().

$PRNCOLWIDTH := 18$ → representa a largura de cada coluna quando se escreve para um arquivo texto com a função WRITEPRN().

Grava a matriz Dat no arquivo texto de nome dado por $OutFile = "euler.txt"$:

$WRITEPRN(OutFile) := Dat$

Arquivo “euler.txt” gerado na execução do *script* “GeraEuler.xmcd” e aberto no aplicativo bloco de notas (notepad.exe) :

```
                10                 0
           8.66025                 5
                 5           8.66025
  6.12303e-016                10
                -5           8.66025
          -8.66025                 5
               -10  1.22461e-015
          -8.66025                -5
                -5          -8.66025
 -1.83691e-015                -10
                 5          -8.66025
           8.66025                -5
```

Vetores de números complexos

Pede-se: Escreva o código fonte C para um programa que leia os valores complexos de um arquivo texto de duas colunas numéricas {real imag} cujo nome é especificado no argumento "InpFile" da linha de comando e, a seguir, imprima em formato polar {módulo, ângulo} na tela do console os valores complexos lidos. Teste e valide o programa com o arquivo "euler.txt" gerado pelo script MathCad referido no slide 38. O programa deve atender às especificações abaixo:

(a) Quando argc resultar diferente de 2 para a entrada de argumentos na linha de comando o programa imprime na tela do console um texto de "help" especificando e descrevendo o que faz o programa e quais são os argumentos da linha de comando (no caso, há somente um argumento da linha de comando "InpFile", correspondendo a argv[1]).

(b) O programa determina automaticamente o numero de valores complexos armazenados no arquivo cujo nome é especificado no argumento da linha de comando "InpFile".

(c) O programa usa alocação dinâmica para alocar memória para o conjunto de valores complexos lidos de "InpFile".

(d) O programa testa a conformidade de cada número complexo lido de "InpFile", e acusa erro de leitura quando o dado lido não corresponder a um numero complexo no formato retangular {real imag}.

Solução: O código fonte https://www.fccdecastro.com.br/CursoC&C++/C/LeCpxVec.c, listado no próximo side, é uma possível solução para o problema posto no enunciado acima.

Vetores de números complexos

```c
/*****************************************************************************
* Le um vetor de elementos COMPLEX armazenado em um arquivo texto de
* 2 colunas {real imag} de nome dado por argv[1] e imprime os elementos
* do vetor na tela do console em formato polar.
*****************************************************************************/
/*****************************************************************************
* HEADERS:
*****************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<conio.h>
#include<string.h>
/*****************************************************************************
* DATA TYPE STRUCTURES:
*****************************************************************************/
typedef struct Complex{
float Re;
float Im;
}COMPLEX;
```

Vetores de números complexos

```
22  /*****************************************************************************
23  * MACROS:
24  *****************************************************************************/
25  static double sqrarg;
26  #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
27
28  static COMPLEX zarg;
29  /* COMPLEX modulus |a| */
30  #define ZMOD(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 : sqrt(SQR(z
31
32  /* COMPLEX norm */
33  #define ZNRM(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 : (SQR(zarg.
34
35  static COMPLEX zarg0,zarg1;
36  /* Multiply two COMPLEX values z0 and z1. Put the result in zR. */
37  #define CMPY(z0,z1,zR) {zR.Re =(zarg0.Re=(z0.Re))* (zarg1.Re=(z1.Re))- (zarg0.Im=(z0.Im))
38  /*****************************************************************************
39  * PROGRAM DEFINITIONS:
40  *****************************************************************************/
41  #define BUFSIZE 0x8000 /* in&out file buffer size */
42  #define MAXLINESIZE 65536 /* max line size for GetFileNumLines(FILE *Fp) */
43  /*****************************************************************************
44  * FUNCTION PROTOTYPES:
45  *****************************************************************************/
46  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem);
47  double ZAng(COMPLEX Z);
48  COMPLEX *ReadCpxVect(char *Buffer, char *InpFile, unsigned long *NumElem);
```

Vetores de números complexos

```
49 /*****************************************************************************
50 * main():
51 *****************************************************************************/
52 void main(int argc, char *argv[]){
53
54 COMPLEX *Vect;
55 unsigned long NumElem;
56 char *InpFile;
57 char *Buffer;
58
59 unsigned register n;
60
61 /* Help da linha de comando */
62 if(argc!=2){
63
64 puts("\nLe um vetor de elementos de valor complexo de um arquivo texto de duas colunas {real imag}");
65 puts("de nome dado pelo argumento 'InpFile' e imprime na tela do console em formato polar.");
66
67 /*    0    1        */
68 puts("Uso: InpFile");
69 exit(1);
70 }
71
72 /* Atribuicao do argumento argv[1] da linha de comando 'a variavel InpFile */
73 InpFile=argv[1]; /* nome do arquivo texto de duas colunas que armazena o vetor de elementos
74                     de valor complexo, cada coluna respectivamente representando {real imag} */
```

Vetores de números complexos

```
75
76  /* alloc I/O file buffer */
77  Buffer=(char *)malloc(BUFSIZE*sizeof(char));
78  if(!Buffer){puts("Nao ha memoria para Buffer!");exit(1);}
79
80  /* Determina o numero NumElem de elementos COMPLEX que existem no arquivo texto de 2 colunas {real imag}
81   de nome dado pelo argumento 'InpFile', aloca memoria p/ o vetor COMPLEX Vect[] de NumElem elementos,
82   Le os NumElem COMPLEX de 'InpFile' e armazena no vetor COMPLEX Vect[] */
83  Vect=ReadCpxVect(Buffer, InpFile, &NumElem);
84
85  /* imprime o vetor COMPLEX Vect[] na tela do console no formato polar {modulo,fase} */
86  printf("\n|Vect|\t\tang{Vect}\n");
87  PrintPolarCpxVec (Vect, NumElem);
88  }
89  /*****************************************************************************
90  * FUNC: void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
91  *
92  * DESC: Imprime vetor ZVec[] na tela do console no formato polar {modulo,fase}
93  *****************************************************************************/
94  void PrintPolarCpxVec (COMPLEX *ZVec, unsigned NumElem){
95  unsigned register n;
96
97  for(n=0;n<NumElem;n++){
98  printf("%f\t%lf\n",ZMOD(ZVec[n]),ZAng(ZVec[n])); /* ZMOD(Z) calcula |Z| e
99                                                      eh definida em "MACROS:" acima */
100 }
101 }
```

Vetores de números complexos

```
/*****************************************************************************
* FUNC: double ZAng(COMPLEX Z)
*
* DESC: Retorna a fase (angulo) Theta em [graus] da variavel
*       COMPLEX Z=Re{Z}+j*Im{Z}=|Z|exp(j*Theta), no intervalo
*       -pi < Theta <= pi, com |Z|=sqrt((Z.Re)^2+(Z.Im)^2).
*****************************************************************************/
double ZAng(COMPLEX Z){
return (180/M_PI)*atan2(Z.Im,Z.Re);
}
/*****************************************************************************
* FUNC: unsigned long GetFileNumLines(FILE *Fp)
*
* DESC: Retorna o numero de linhas no arquivo texto apontado pelo stream Fp
*****************************************************************************/
unsigned long GetFileNumLines(FILE *Fp)
{
  char line[MAXLINESIZE]; // buffer de linha
  unsigned long register Count;

  Count=0;
  while (fgets(line,MAXLINESIZE-2,Fp)){// le linha a linha o arquivo apontado por Fp
  Count++; // incrementa a cada linha lida
  }

  if(strlen(line)==1)Count--;/* p/ descontar o ultimo '\n' no arquivo no
                                caso em que o arquivo tenha sido eventualmente editado
                                e o usuario tenha teclado <enter> no final do arquivo */
  rewind(Fp);
  return Count; // retorna o numero de linhas do arquivo
}
```

Vetores de números complexos

```
133 /*************************************************************************
134 * FUNC: COMPLEX *ReadCpxVect(char *Buffer, char *InpFile, unsigned long *NumElem)
135 *
136 * DESC: (1) Determina o numero NumElem de elementos COMPLEX que existem no arquivo
137 * texto de 2 colunas {real imag} de nome dado pelo argumento 'InpFile'.
138 * (2) Aloca memoria p/ o vetor  COMPLEX Vect[] de NumElem elementos.
139 * (3) Le os NumElem COMPLEX de 'InpFile' e armazena no vetor COMPLEX Vect[].
140 * (4) Retorna &Vect[0] para a funcao chamadora.
141 *************************************************************************/
142 COMPLEX *ReadCpxVect(char *Buffer, char *InpFile, unsigned long *NumElem)
143 {
144 COMPLEX *Vect; // vetor COMPLEX Vect[]
145 FILE *Fp;
146 unsigned register n;
147
148 /* abre o stream Fp p/ o arquivo de entrada InpFile */
149 if((Fp=fopen(InpFile,"rt"))==NULL)
150 {printf("Nao consigo abrir %s!",InpFile);exit(1);}
151
152 /* seta buffer de I/O p/ o stream Fp */
153 if (setvbuf(Fp,Buffer, _IOFBF, BUFSIZE) != 0)
154 {printf("Nao consigo setar o buffer de I/O para %s!",InpFile);exit(1);}
155
156  /* obtem o numero de elementos COMPLEX no arquivo InpFile */
157 *NumElem=GetFileNumLines(Fp);
158
159 /* aloca memoria p/ vetor COMPLEX Vect[] de *NumElem elementos*/
160 Vect=(COMPLEX *)malloc((*NumElem)*sizeof(COMPLEX));
161 if(!Vect){puts("Nao ha memoria para vetor Vect!");exit(1);}
162
```

Vetores de números complexos

```
163  /* Le vetor Vect[] do arquivo de entrada InpFile cujo stream eh Fp */
164  for(n=0;n<*NumElem;n++)
165  if((fscanf(Fp,"%f%f",&Vect[n].Re,&Vect[n].Im))!=2)/* if fscanf()!=2 ->
166                                                     erro de leitura */
167  {printf("Erro lendo arquivo %s!",InpFile);exit(1);}
168
169  /* fecha o stream Fp */
170  fclose(Fp);
171
172  /* retorna &Vect[0] */
173  return Vect;
174  }
```

Vetores de números complexos

Execução na tela do console – tela de "help" resultante ao se digitar o nome do executável (LeCpxVec.exe) com a linha de comando sem argumentos :

```
Le um vetor de elementos de valor complexo de um arquivo texto de duas colunas {real imag}
de nome dado pelo argumento 'InpFile' e imprime na tela do console em formato polar.
Uso: InpFile
```

Execução na tela do console – linha de comando conforme especificada abaixo:

LeCpxVec euler.txt

Resultado na tela do console :

```
|Vect|          ang{Vect}
10.000000       0.000000
9.999996        30.000012
9.999996        59.999988
10.000000       90.000000
9.999996        120.000012
9.999996        149.999988
10.000000       180.000000
9.999996        -149.999988
9.999996        -120.000012
10.000000       -90.000000
9.999996        -59.999988
9.999996        -30.000012
```

Dado que há conformidade do módulo e do ângulo dos valores complexos impressos na tela do console (ao lado) com o módulo e o ângulo dos valores complexos gerados pelo *script* "GeraEuler.xmcd" (slide 38), considera-se que o código fonte LeCpxVec.c está validado.

Matrizes de números complexos

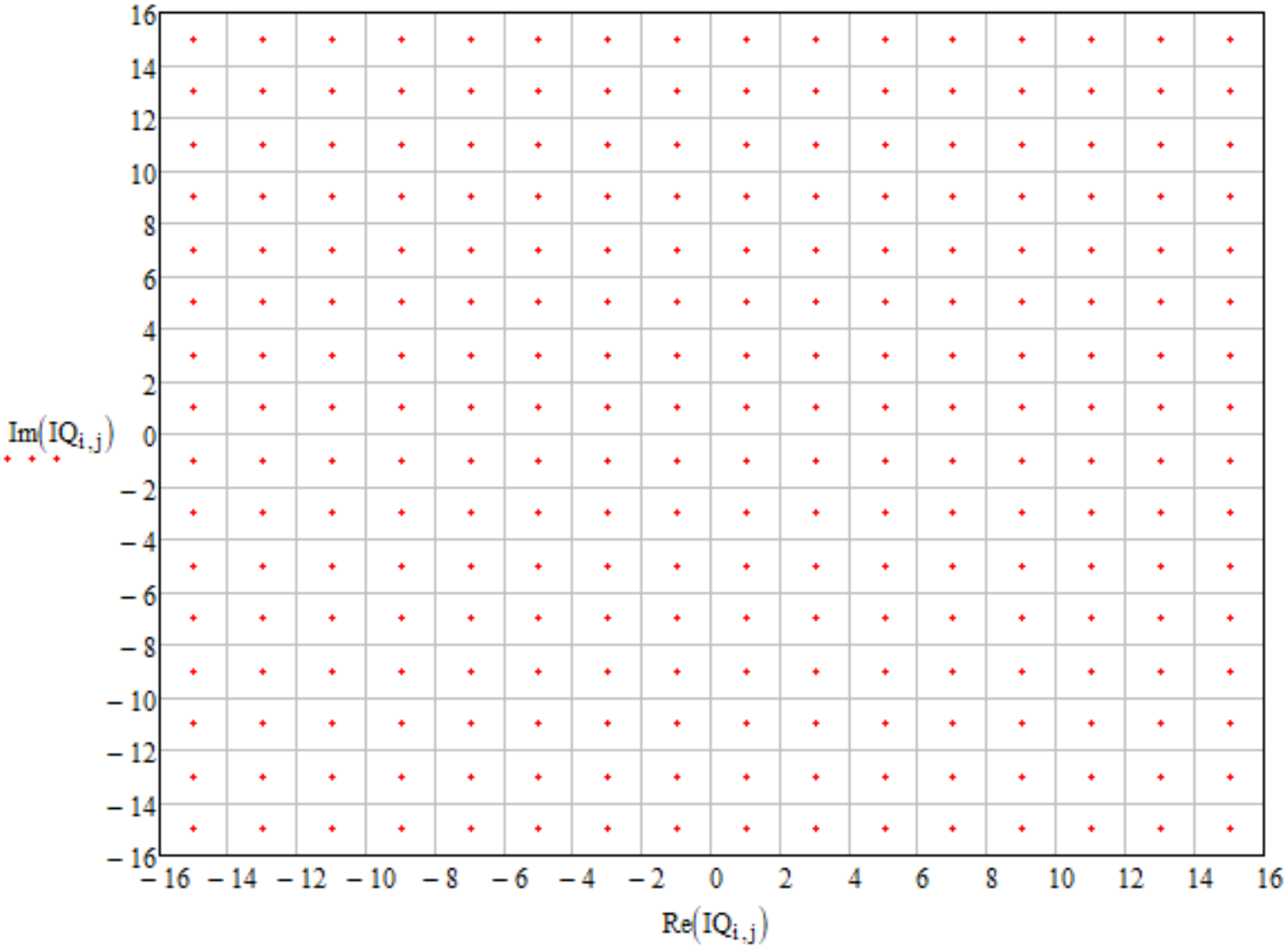
Matrizes de números complexos são usuais em processamento digital de sinais para sistemas de comunicações.

Por exemplo, abaixo é mostrado a matriz de símbolos IQ para a modulação digital 256-QAM (ver slide 22 de https://www.fccdecastro.com.br/pdf/SCD1_CapIV.pdf) :

$$IQ = \begin{pmatrix}
-15+15j & -13+15j & -11+15j & -9+15j & -7+15j & -5+15j & -3+15j & -1+15j & 1+15j & 3+15j & 5+15j & 7+15j & 9+15j & 11+15j & 13+15j & 15+15j \\
-15+13j & -13+13j & -11+13j & -9+13j & -7+13j & -5+13j & -3+13j & -1+13j & 1+13j & 3+13j & 5+13j & 7+13j & 9+13j & 11+13j & 13+13j & 15+13j \\
-15+11j & -13+11j & -11+11j & -9+11j & -7+11j & -5+11j & -3+11j & -1+11j & 1+11j & 3+11j & 5+11j & 7+11j & 9+11j & 11+11j & 13+11j & 15+11j \\
-15+9j & -13+9j & -11+9j & -9+9j & -7+9j & -5+9j & -3+9j & -1+9j & 1+9j & 3+9j & 5+9j & 7+9j & 9+9j & 11+9j & 13+9j & 15+9j \\
-15+7j & -13+7j & -11+7j & -9+7j & -7+7j & -5+7j & -3+7j & -1+7j & 1+7j & 3+7j & 5+7j & 7+7j & 9+7j & 11+7j & 13+7j & 15+7j \\
-15+5j & -13+5j & -11+5j & -9+5j & -7+5j & -5+5j & -3+5j & -1+5j & 1+5j & 3+5j & 5+5j & 7+5j & 9+5j & 11+5j & 13+5j & 15+5j \\
-15+3j & -13+3j & -11+3j & -9+3j & -7+3j & -5+3j & -3+3j & -1+3j & 1+3j & 3+3j & 5+3j & 7+3j & 9+3j & 11+3j & 13+3j & 15+3j \\
-15+j & -13+j & -11+j & -9+j & -7+j & -5+j & -3+j & -1+j & 1+j & 3+j & 5+j & 7+j & 9+j & 11+j & 13+j & 15+j \\
-15-j & -13-j & -11-j & -9-j & -7-j & -5-j & -3-j & -1-j & 1-j & 3-j & 5-j & 7-j & 9-j & 11-j & 13-j & 15-j \\
-15-3j & -13-3j & -11-3j & -9-3j & -7-3j & -5-3j & -3-3j & -1-3j & 1-3j & 3-3j & 5-3j & 7-3j & 9-3j & 11-3j & 13-3j & 15-3j \\
-15-5j & -13-5j & -11-5j & -9-5j & -7-5j & -5-5j & -3-5j & -1-5j & 1-5j & 3-5j & 5-5j & 7-5j & 9-5j & 11-5j & 13-5j & 15-5j \\
-15-7j & -13-7j & -11-7j & -9-7j & -7-7j & -5-7j & -3-7j & -1-7j & 1-7j & 3-7j & 5-7j & 7-7j & 9-7j & 11-7j & 13-7j & 15-7j \\
-15-9j & -13-9j & -11-9j & -9-9j & -7-9j & -5-9j & -3-9j & -1-9j & 1-9j & 3-9j & 5-9j & 7-9j & 9-9j & 11-9j & 13-9j & 15-9j \\
-15-11j & -13-11j & -11-11j & -9-11j & -7-11j & -5-11j & -3-11j & -1-11j & 1-11j & 3-11j & 5-11j & 7-11j & 9-11j & 11-11j & 13-11j & 15-11j \\
-15-13j & -13-13j & -11-13j & -9-13j & -7-13j & -5-13j & -3-13j & -1-13j & 1-13j & 3-13j & 5-13j & 7-13j & 9-13j & 11-13j & 13-13j & 15-13j \\
-15-15j & -13-15j & -11-15j & -9-15j & -7-15j & -5-15j & -3-15j & -1-15j & 1-15j & 3-15j & 5-15j & 7-15j & 9-15j & 11-15j & 13-15j & 15-15j
\end{pmatrix}$$

Matrizes de números complexos

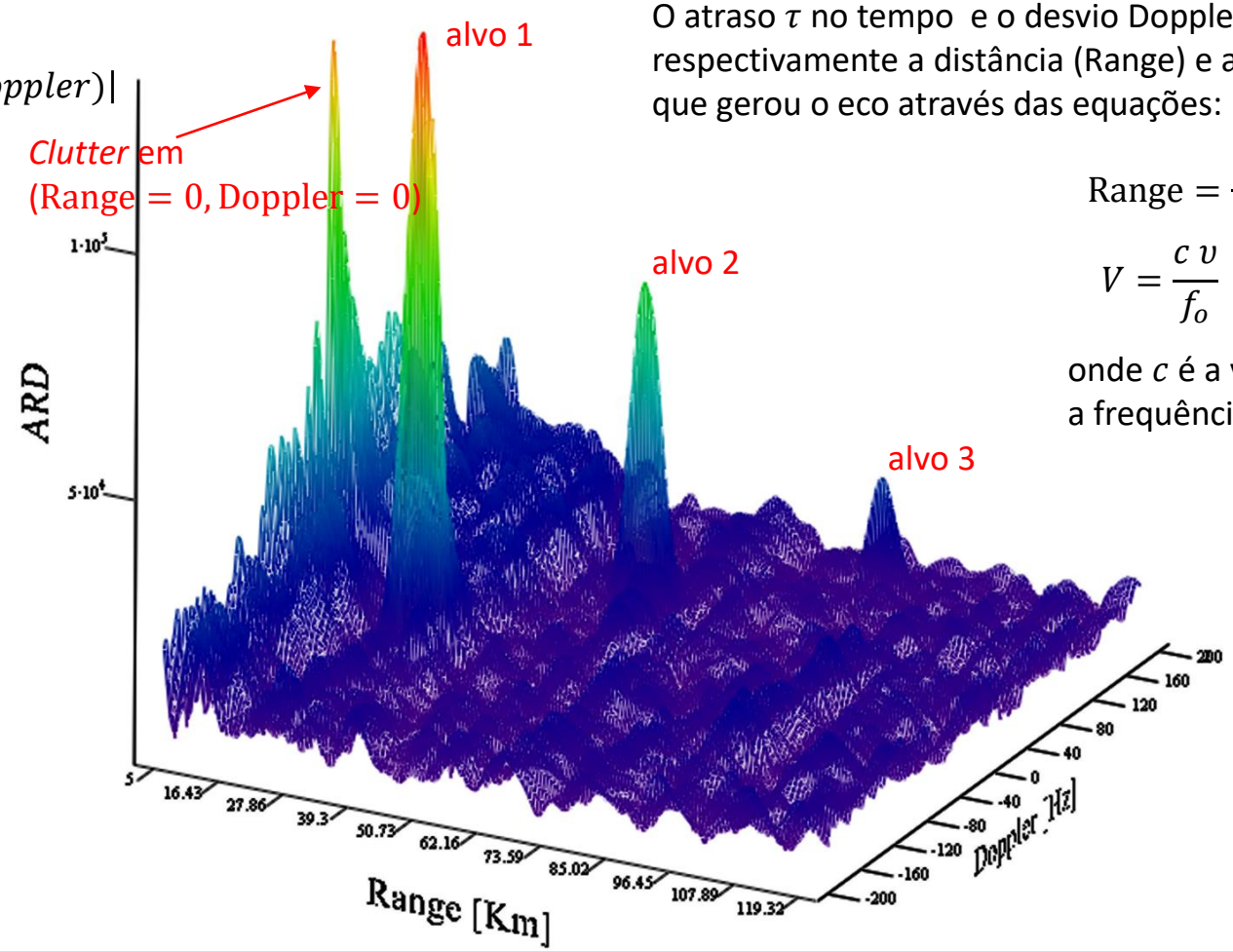
A matriz de símbolos IQ no slide anterior define o denominado gráfico da **constelação** 256-QAM, conforme abaixo, em que cada ponto da constelação representa um símbolo $I + jQ$ definido na matriz IQ no slide anterior. No caso da modulação 256-QAM, cada ponto da constelação representa uma palavra binária de 8 bits a ser transmitida pelo TX digital, conforme slide 22 de https://www.fccdecastro.com.br/pdf/SCD1_CapIV.pdf .



Matrizes de números complexos

Um outro uso de matrizes de números complexos é no processamento *Range*-Doppler de radares *pulse*-Doppler. Uma superfície de valores complexos $\Psi(Range, Doppler)$ denominada de **função de ambiguidade**, cujo módulo $|\Psi(Range, Doppler)|$ é denominado **superfície ARD** (**A**mplitude-**R**ange-**D**oppler) conforme gráfico abaixo, é obtida da correlação no domínio tempo e no domínio frequência entre a sequência de valores $\mathrm{I} + j\mathrm{Q}$ do sinal do pulso emitido pelo TX do radar e a sequência de valores $\mathrm{I} + j\mathrm{Q}$ do sinal recebido no RX do radar. A sequencia de valores $\mathrm{I} + j\mathrm{Q}$ do sinal recebido no RX constitui uma superposição dos ecos de cada respectivo alvo, possibilitando identificar por correlação com o sinal do TX o atraso τ no tempo e o desvio Doppler υ do eco (ver https://www.fccdecastro.com.br/pdf/ADA615308.pdf).

$|\Psi(Range, Doppler)|$



O atraso τ no tempo e o desvio Doppler υ do eco determinam respectivamente a distância (Range) e a velocidade V do alvo que gerou o eco através das equações:

$$\text{Range} = \frac{1}{2}\, c\tau$$

$$V = \frac{c\,\upsilon}{f_o}$$

onde c é a velocidade da luz e f_o é a frequência do sinal do TX radar.

Matrizes de números complexos

O módulo $|\Psi(Range, Doppler)|$ da superfície de valores complexos, exemplificado no gráfico abaixo, apresenta picos para valores $(Range_i, Doppler_i)$ em que há ocorrência de um alvo i. $Doppler_i = (V_i/c)f_o$ é o desvio Doppler do sinal do eco do i-ésimo alvo em relação à frequência f_o do sinal do TX, alvo que se move com velocidade V_i. $Range_i = \frac{1}{2}\, c\tau_i$ é a distância do i-ésimo alvo, onde τ_i é o atraso do sinal do eco do i-ésimo alvo em relação ao sinal emitido pelo TX do radar. Para o caso de radar passivo, ver slides 38 a 41 de https://www.fccdecastro.com.br/pdf/TR_CapIII.pdf .

$|\Psi(Range, Doppler)|$

Clutter em $(\text{Range} = 0, \text{Doppler} = 0)$

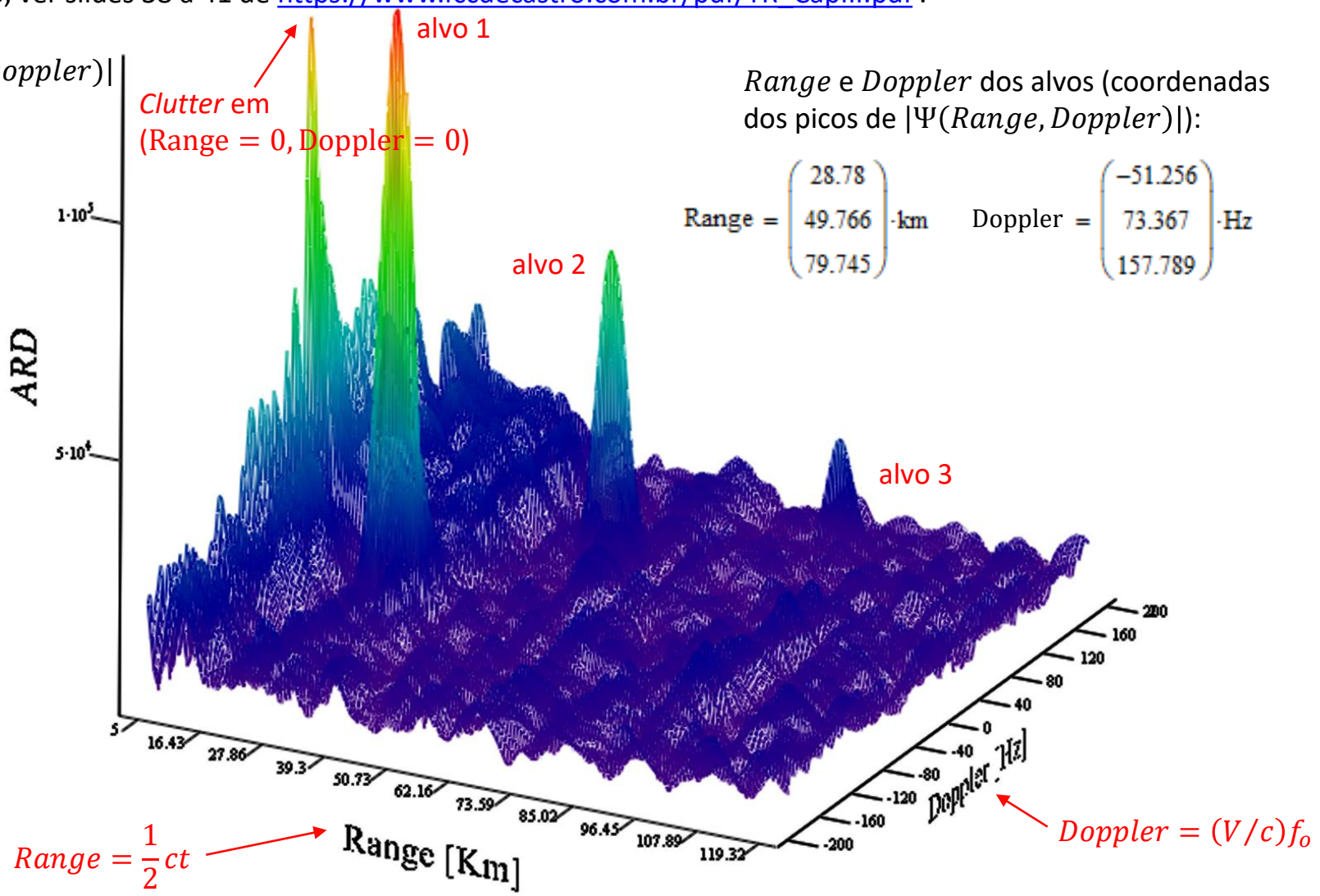
alvo 1

alvo 2

alvo 3

$Range$ e $Doppler$ dos alvos (coordenadas dos picos de $|\Psi(Range, Doppler)|$):

$$\text{Range} = \begin{pmatrix} 28.78 \\ 49.766 \\ 79.745 \end{pmatrix} \cdot \text{km} \qquad \text{Doppler} = \begin{pmatrix} -51.256 \\ 73.367 \\ 157.789 \end{pmatrix} \cdot \text{Hz}$$



$Range = \frac{1}{2}ct$

$Doppler = (V/c)f_o$

Matrizes de números complexos

Exemplo 3: O *script* MathCad em https://www.fccdecastro.com.br/ZIP/QAMmatrix.zip mostrado a seguir gera a matriz complexa de MxM símbolos IQ da modulação digital M-QAM e grava no arquivo texto de nome armazenado na string 'OutFile':

$M := 256$ $\quad$ $OutFile := "IQ\_256QAM.txt"$

$i := 0 .. \sqrt{M} - 1$ $\quad$ $A_i := \sqrt{M} - 1 - 2 \cdot i$

$A^T = (15 \quad 13 \quad 11 \quad 9 \quad 7 \quad 5 \quad 3 \quad 1 \quad -1 \quad -3 \quad -5 \quad -7 \quad -9 \quad -11 \quad -13 \quad -15)$

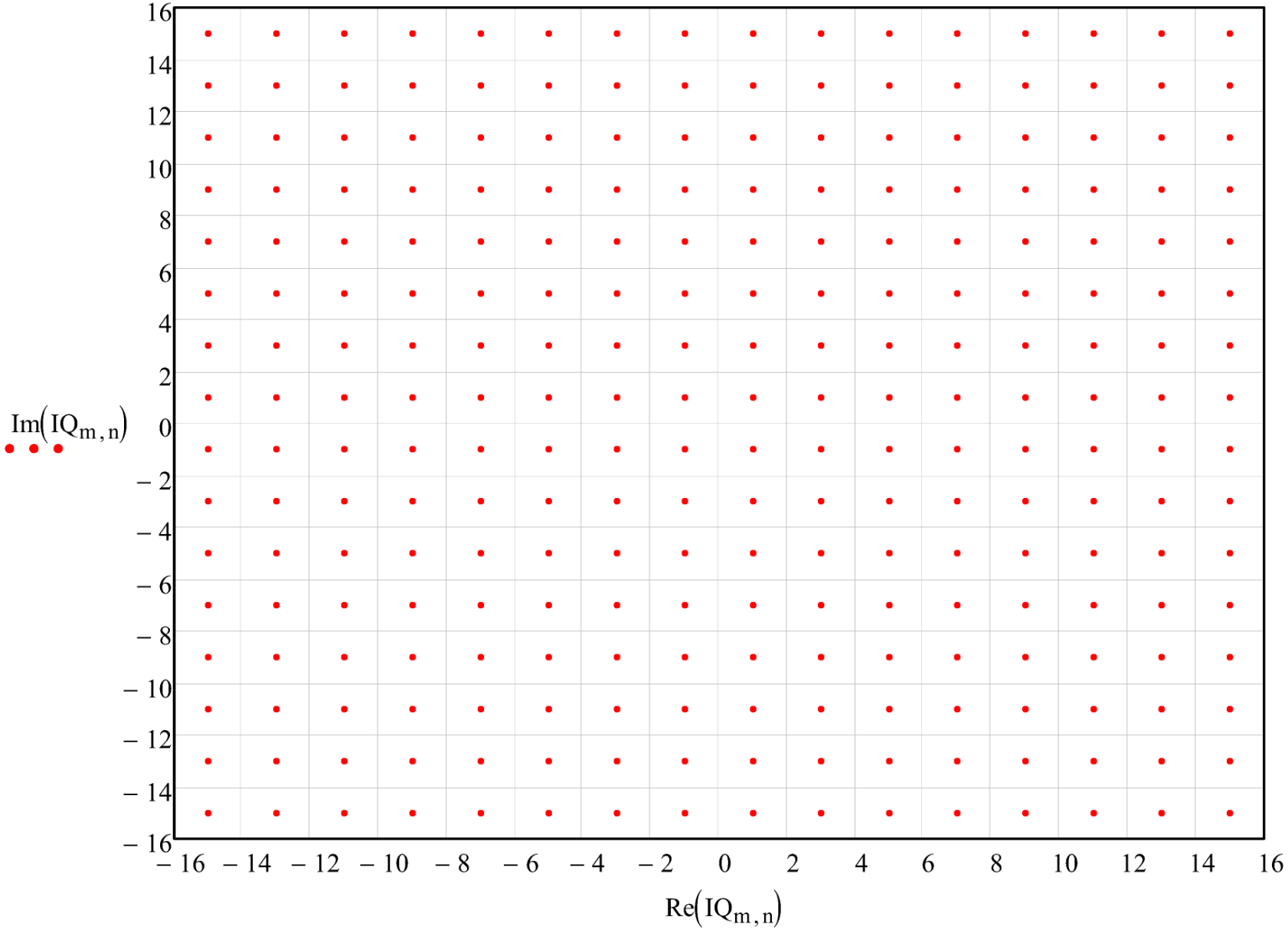
$m := 0 .. length(A) - 1$ $\quad$ $n := 0 .. length(A) - 1$

$IQ_{m,n} := A_{length(A)-1-n} + j \cdot A_m$

Matrizes de números complexos

$$IQ = \begin{pmatrix}
-15+15j & -13+15j & -11+15j & -9+15j & -7+15j & -5+15j & -3+15j & -1+15j & 1+15j & 3+15j & 5+15j & 7+15j & 9+15j & 11+15j & 13+15j & 15+15j \\
-15+13j & -13+13j & -11+13j & -9+13j & -7+13j & -5+13j & -3+13j & -1+13j & 1+13j & 3+13j & 5+13j & 7+13j & 9+13j & 11+13j & 13+13j & 15+13j \\
-15+11j & -13+11j & -11+11j & -9+11j & -7+11j & -5+11j & -3+11j & -1+11j & 1+11j & 3+11j & 5+11j & 7+11j & 9+11j & 11+11j & 13+11j & 15+11j \\
-15+9j & -13+9j & -11+9j & -9+9j & -7+9j & -5+9j & -3+9j & -1+9j & 1+9j & 3+9j & 5+9j & 7+9j & 9+9j & 11+9j & 13+9j & 15+9j \\
-15+7j & -13+7j & -11+7j & -9+7j & -7+7j & -5+7j & -3+7j & -1+7j & 1+7j & 3+7j & 5+7j & 7+7j & 9+7j & 11+7j & 13+7j & 15+7j \\
-15+5j & -13+5j & -11+5j & -9+5j & -7+5j & -5+5j & -3+5j & -1+5j & 1+5j & 3+5j & 5+5j & 7+5j & 9+5j & 11+5j & 13+5j & 15+5j \\
-15+3j & -13+3j & -11+3j & -9+3j & -7+3j & -5+3j & -3+3j & -1+3j & 1+3j & 3+3j & 5+3j & 7+3j & 9+3j & 11+3j & 13+3j & 15+3j \\
-15+j & -13+j & -11+j & -9+j & -7+j & -5+j & -3+j & -1+j & 1+j & 3+j & 5+j & 7+j & 9+j & 11+j & 13+j & 15+j \\
-15-j & -13-j & -11-j & -9-j & -7-j & -5-j & -3-j & -1-j & 1-j & 3-j & 5-j & 7-j & 9-j & 11-j & 13-j & 15-j \\
-15-3j & -13-3j & -11-3j & -9-3j & -7-3j & -5-3j & -3-3j & -1-3j & 1-3j & 3-3j & 5-3j & 7-3j & 9-3j & 11-3j & 13-3j & 15-3j \\
-15-5j & -13-5j & -11-5j & -9-5j & -7-5j & -5-5j & -3-5j & -1-5j & 1-5j & 3-5j & 5-5j & 7-5j & 9-5j & 11-5j & 13-5j & 15-5j \\
-15-7j & -13-7j & -11-7j & -9-7j & -7-7j & -5-7j & -3-7j & -1-7j & 1-7j & 3-7j & 5-7j & 7-7j & 9-7j & 11-7j & 13-7j & 15-7j \\
-15-9j & -13-9j & -11-9j & -9-9j & -7-9j & -5-9j & -3-9j & -1-9j & 1-9j & 3-9j & 5-9j & 7-9j & 9-9j & 11-9j & 13-9j & 15-9j \\
-15-11j & -13-11j & -11-11j & -9-11j & -7-11j & -5-11j & -3-11j & -1-11j & 1-11j & 3-11j & 5-11j & 7-11j & 9-11j & 11-11j & 13-11j & 15-11j \\
-15-13j & -13-13j & -11-13j & -9-13j & -7-13j & -5-13j & -3-13j & -1-13j & 1-13j & 3-13j & 5-13j & 7-13j & 9-13j & 11-13j & 13-13j & 15-13j \\
-15-15j & -13-15j & -11-15j & -9-15j & -7-15j & -5-15j & -3-15j & -1-15j & 1-15j & 3-15j & 5-15j & 7-15j & 9-15j & 11-15j & 13-15j & 15-15j
\end{pmatrix}$$

Matrizes de números complexos



Matrizes de números complexos

Índices i e j das linhas e colunas da matriz complexa IQ:

$$i := 0 .. rows(IQ) - 1 \quad j := 0 .. cols(IQ) - 1$$

Converte a matriz complexa IQ para a matriz em ponto flutuante IQOut, em que, em cada linha, as colunas de índice par correspondem a parte real do respectivo símbolo IQ e as colunas de índice ímpar correspondem a parte imaginária do respectivo símbolo IQ:

$$IQOut^{\langle 2\cdot j\rangle} := Re(IQ)^{\langle j\rangle}$$

$$IQOut^{\langle 2\cdot j+1\rangle} := Im(IQ)^{\langle j\rangle}$$

$IQOut =$

	0	1	2	3	4	5	6	7	8	9	10	11
0	-15	15	-13	15	-11	15	-9	15	-7	15	-5	15
1	-15	13	-13	13	-11	13	-9	13	-7	13	-5	13
2	-15	11	-13	11	-11	11	-9	11	-7	11	-5	11
3	-15	9	-13	9	-11	9	-9	9	-7	9	-5	9
4	-15	7	-13	7	-11	7	-9	7	-7	7	-5	7
5	-15	5	-13	5	-11	5	-9	5	-7	5	-5	5
6	-15	3	-13	3	-11	3	-9	3	-7	3	-5	3
7	-15	1	-13	1	-11	1	-9	1	-7	1	-5	1
8	-15	-1	-13	-1	-11	-1	-9	-1	-7	-1	-5	-1
9	-15	-3	-13	-3	-11	-3	-9	-3	-7	-3	-5	-3
10	-15	-5	-13	-5	-11	-5	-9	-5	-7	-5	-5	...

Matrizes de números complexos

Especificando o formato de gravação de cada uma das 2 colunas do arquivo de saída OutFile = "IQ_256QAM.txt" :

PRNPRECISION:= 6 → representa o número de dígitos significativos a serem usados quando se escreve para um arquivo texto com a função WRITEPRN().

PRNCOLWIDTH:= 18 → representa a largura de cada coluna quando se escreve para um arquivo texto com a função WRITEPRN().

Grava a matriz Dat no arquivo texto de nome dado por OutFile = "IQ_256QAM.txt" :

WRITEPRN(OutFile) := IQOut

Matrizes de números complexos

Pede-se: Escreva o código fonte C para um programa que lê uma matriz COMPLEX 2D do arquivo texto de entrada cujo nome é dado pelo argumento de linha de comando "InputMatrix", armazena os valores lidos na matriz COMPLEX CMat alocada na memória *heap*, conjuga os elementos do CMat e escreve CMat conjugado no arquivo texto de saída cujo nome é dado pelo argumento da linha de comando "OutputMatrix". Teste e valide o programa com o arquivo "IQ_256QAM.txt" gerado pelo script MathCad referido no slide 53. O programa deve atender às especificações abaixo:

(a) Quando argc resultar diferente de 3 para a entrada de argumentos na linha de comando o programa imprime na tela do console um texto de "help" especificando e descrevendo o que faz o programa e quais são os argumentos da linha de comando. No caso, os argumentos da linha de comando são "InputMatrix" correspondendo a argv[1], e "OutputMatrix" correspondendo a argv[2] .

(b) O programa determina automaticamente o numero de valores complexos armazenados no arquivo cujo nome é especificado no argumento da linha de comando "InputMatrix".

(c) O programa usa alocação dinâmica para alocar memória para o conjunto de valores complexos lidos de "InputMatrix".

(d) O programa testa a conformidade da matriz complexa lida de "InputMatrix", e acusa erro quando ocorrer qualquer tipo de inconformidade.

Solução: O código fonte https://www.fccdecastro.com.br/CursoC&C++/C/ConjCpxMat.C, listado no próximo slide, é uma possível solução para o problema posto no enunciado acima.

Matrizes de números complexos

```c
/* ConjCpxMat.c: Reads COMPLEX 2D matrix from input text file whose name is
* given by argument argv[1], stores read values in COMPLEX matrix CMat allocated
* in heap memory, conjugate the elements of CMat puts and writes conjugated CMat
* to output text file whose name is given by argument argv[2].
*
*                    By FCCDeCastro - April 2024
*
* Note: This C source code should be compiled with DevC++ compiler.
*********************************************************************************/

/*********************************************************************************
* HEADERS:
*********************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<process.h>
#include<math.h>
#include<float.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#include<time.h>
#include<dos.h>
/*********************************************************************************
* PROGRAM DEFINITIONS:
*********************************************************************************/
#define BUFSIZE 0x8000 /* in&out file buffer size */
```

Matrizes de números complexos

```
28  /*****************************************************************************
29  * COMPILING DIRECTIVES:
30  *****************************************************************************/
31  #pragma GCC diagnostic ignored "-Wwrite-strings"
32  /*****************************************************************************
33  * CONSTANTS:
34  *****************************************************************************/
35
36  /*****************************************************************************
37  * DATA TYPE STRUCTURES:
38  *****************************************************************************/
39  typedef enum {
40      NO_ERROR,           /*  " ", */
41      MEMORY_ERR,         /*  "Not enough memory", */
42      READ_OPEN_ERR,      /*  "I can't find the file", */
43      READ_ERR,           /*  "Error reading file", */
44      WRITE_OPEN_ERR,     /*  "I can't open the file", */
45      WRITE_ERR,          /*  "I can't write the file", */
46      BUFFER_ERR,         /*  "I can't attach buffer to file" */
47      SIZE_ERR,           /*  "Specified size doesn't match data size in file", */
48      MAXIT_ERR,          /*  "Exceeded max number of iterations in function" */
49      LOGIC_ERR,          /*  "Logic error in function" */
50      CMDLINE_ERR,        /*  "Wrong command line argument" */
51      MTXDIM_ERR,         /*  "Invalid matrix dimensions in file" */
52      NROWS_ERR,          /*  "Not compatible number of rows between input matrices" */
53      NCOLS_ERR,          /*  "Not compatible number of columns between input matrices" */
54    /* ------------------------------------------------*/
55  }ERR;
56
57  typedef struct Complex{
58  float Re;
59  float Im;
60  }COMPLEX;
```

Matrizes de números complexos

```
61  /*****************************************************************************
62  * FUNCTION PROTOTYPES:
63  *****************************************************************************/
64  float **FloatMatCAlloc(unsigned NRows,unsigned NCols);
65  void FloatMatFree(float **M);
66  void Quit(ERR err, char *name);
67  unsigned long GetFileNumLines(FILE *Fp);
68  long GetFileNumSamples(FILE *Fp);
69  float  **FloatMatRead(float **M,char *file,char *buffer,unsigned *NRows,unsigned *NCols, char AllocFlag);
70  void FloatMatWrite(float **M,char *file,char *buffer,unsigned NRows,unsigned NCols);
71  void CpxMatWrite(COMPLEX **M,char *file,char *buffer,unsigned NRows,unsigned NCols);
72  void CpxMatElementConjugate(COMPLEX **Mat,unsigned NRows,unsigned NCols);
73  /*****************************************************************************
74  * GLOBAL VARIABLES:
75  *****************************************************************************/
76
77  /*****************************************************************************
78  * MACROS:
79  *****************************************************************************/
80  #define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
81
82  static double steparg;
83  #define STEP(a) ((steparg=(a))<= 0.0 ? 0.0 : steparg)
84
85  static float maxarg1,maxarg2;
86  #define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
87      (maxarg1) : (maxarg2))
88
89  static int iminarg1,iminarg2;
90  #define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2) ?\
91      (iminarg1) : (iminarg2))
92
93  static double sqrarg;
94  #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
95
```

Matrizes de números complexos

```
 96  #define SWAP(a,b) temp=(a);(a)=(b);(b)=temp;
 97
 98  float *tempp;
 99  #define SWAPP(a,b) tempp=(a);(a)=(b);(b)=tempp;
100
101  static COMPLEX zarg;
102  /* COMPLEX modulus */
103  #define ZMOD(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 : sqrt(SQR(zarg.Re)+SQR(zarg.Im)))
104  /* COMPLEX norm */
105  #define ZNRM(a) ((((zarg.Re=(a.Re)) == 0.0)*((zarg.Im=(a.Im)) == 0.0)) ? 0.0 : (SQR(zarg.Re)+SQR(zarg.Im)))
106
107  static COMPLEX zarg0,zarg1;
108  /* Multiply two COMPLEX values z0 and z1. Put the result in zR. */
109  #define CMPY(z0,z1,zR) {zR.Re =(zarg0.Re=(z0.Re))* (zarg1.Re=(z1.Re))- (zarg0.Im=(z0.Im))* (zarg1.Im=(z1.Im));  z
110
111  /* Multiply two COMPLEX values (NOT EXPRESSIONS -> slow code!)  z0 and z1. Accumulate the result in zA. */
112  #define CMPA(z0,z1,zA) {zA.Re+=(z0.Re)*(z1.Re)-(z0.Im)*(z1.Im); zA.Im+=(z0.Re)*(z1.Im)+(z0.Im)*(z1.Re);}
113
114  /* Multiply two COMPLEX values (NOT EXPRESSIONS -> slow code!) z0 and conj{z1). Accumulate the result in zA. */
115  #define CMPCA(z0,z1,zA) {zA.Re+=(z0.Re)*(z1.Re)-(z0.Im)*(-z1.Im); zA.Im+=(z0.Re)*(-z1.Im)+(z0.Im)*(z1.Re);}
116
117  /* Multiply two COMPLEX values (NOT EXPRESSIONS -> slow code!) z0 and conj{z1). Put the result in zA. */
118  #define CMPYC(z0,z1,zA) {zA.Re=(z0.Re)*(z1.Re)-(z0.Im)*(-z1.Im); zA.Im=(z0.Re)*(-z1.Im)+(z0.Im)*(z1.Re);}
119
120  /* Multiply two COMPLEX pointers values z0 and z1. Put the result in zR. */
121  #define CMPP(z0,z1,zR) {zR->Re=(z0->Re)*(z1->Re)-(z0->Im)*(z1->Im); zR->Im=(z0->Re)*(z1->Im)+(z0->Im)*(z1->Re);}
122
123  /* Multiply two COMPLEX pointers values z0 and conj{z1}. Put the result in zR. */
124  #define CMPPC(z0,z1,zR) {zR->Re=(z0->Re)*(z1->Re)-(z0->Im)*(-z1->Im); zR->Im=(z0->Re)*(-z1->Im)+(z0->Im)*(z1->Re);}
125
126  /* Multiply two COMPLEX pointers values z0 and z1. Accumulate the result in zA. */
127  #define CMPPA(z0,z1,zA) {zA->Re+=(z0->Re)*(z1->Re)-(z0->Im)*(z1->Im); zA->Im+=(z0->Re)*(z1->Im)+(z0->Im)*(z1->Re);}
```

Matrizes de números complexos

```
128 /*****************************************************************************
129 *               MAIN():
130 *****************************************************************************/
131 int main(int argc, char *argv[])
132 {
133 int test;
134 char *Buffer;
135 unsigned register i,j;
136 unsigned NRows,NCols;
137 float **Mat;
138 COMPLEX **CMat;
139 char *InpFile, *OutFile;
140
141
142 /* Command Line & help */
143 if(argc!=3){
144 puts("\nReads COMPLEX 2D matrix from input text file whose name is given by argument InputMatrix,");
145 puts("stores read values in COMPLEX matrix CMat allocated in heap memory, conjugate the elements of CMat");
146 puts("and writes conjugated CMat to output text file whose name is given by argument OutputMatrix.");
147       /* 0          1           2*/
148 puts("\nUse: ConjCpxMat InputMatrix OutputMatrix\n");
149 exit(1);
150 }
151 InpFile=argv[1];
152 OutFile=argv[2];
153
154 /* alloc I/O file buffer */
155 Buffer=(char *)calloc(BUFSIZE,sizeof(char));
156 if(!Buffer)Quit(MEMORY_ERR,"");
157
158 /* read Mat from file InpFile */
159 Mat= FloatMatRead(Mat,InpFile, Buffer, &NRows, &NCols,1);
160
161 if(NCols%2)Quit(MTXDIM_ERR,argv[1]);/* check valid matrix dimensions: a COMPLEX matrix */
162                                     /* MUST have an even number of columns */
```

Matrizes de números complexos

```
163
164  /* convert float Mat to COMPLEX CMat */
165  CMat=(COMPLEX **)Mat;
166  NCols=NCols/2;
167
168  /* conjugate the elements of CMat */
169  CpxMatElementConjugate(CMat,NRows,NCols);
170
171  /* write CMat to file whose name is given by OutFile */
172  CpxMatWrite(CMat,OutFile,Buffer,NRows,NCols);
173
174  return 0;
175  }
176
```

Matrizes de números complexos

```
177 /*****************************************************************************
178 * FUNC: float **FloatMatCAlloc(unsigned NRows,unsigned NCols)
179 *
180 * DESC: Allocates memory for a NRows x NCols  float matrix.
181 *****************************************************************************/
182 float **FloatMatCAlloc(unsigned NRows,unsigned NCols)
183 {
184 float  *x;
185 float  **Mx;
186 unsigned register i;
187
188 x = (float *)calloc((unsigned long)(NRows*NCols),sizeof(float));
189 if(!x)Quit(MEMORY_ERR,"");
190
191 Mx=(float  **)calloc((unsigned long)NRows,sizeof(float *));
192 if(!Mx)Quit(MEMORY_ERR,"");
193
194 for(i=0;i<NRows;i++){
195 Mx[i]=(float *)&x[i*NCols];
196 }
197
198 return Mx;
199 }
200 /*****************************************************************************
201 * FUNC: void FloatMatFree(float **M)
202 *
203 * DESC: Frees memory for a float matrix.
204 *****************************************************************************/
205 void FloatMatFree(float **M)
206 {
207 free(M[0]);
208 free(M);
209 }
```

Matrizes de números complexos

```
210 /*************************************************************************
211  * FUNC: void Quit(int err, char *name)
212  *
213  * DESC: Prints Error message and quit.
214  *************************************************************************/
215 void Quit(ERR err, char *name)
216 {
217     static char *ErrMess[] = {
218     " ",
219     "Not enough memory",
220     "I can't find the file",
221     "Error reading file",
222     "I can't open the file",
223     "I can't write the file",
224     "I can't attach buffer to file",
225     "Specified size doesn't match data size in file",
226      "Exceeded max number of iterations in function",
227     "Logic error in function",
228     "Wrong command line argument",
229     "Invalid matrix dimensions in file",
230     "Not compatible number of rows between input matrices",
231     "Not compatible number of columns between input matrices",
232     /* ----------------------------------------------------*/
233     };
```

Matrizes de números complexos

```
234
235     system("cls");
236
237     if (err != NO_ERROR) {
238
239         if(name[0]){
240         fprintf (stderr,"%s %s!\n", ErrMess[err], name);
241         putch(7);
242         exit (err);
243         }
244         else{
245         fprintf (stderr,"%s!\n", ErrMess[err]);
246         putch(7);
247         exit (err);
248         }
249     }
250     exit (0);
251 }
```

Matrizes de números complexos

```
252 /*****************************************************************************
253 * FUNC: unsigned long GetFileNumLines(FILE *Fp)
254 *
255 * DESC: Return number of lines in the TEXT file pointed by Fp.
256 *****************************************************************************/
257 #define MAXLINESIZE 8192
258 unsigned long GetFileNumLines(FILE *Fp)
259 {
260
261   char line[MAXLINESIZE];
262   unsigned long register Count;
263
264
265   Count=0;
266   while (fgets(line,MAXLINESIZE-2,Fp)){
267   Count++;
268   }
269
270   if(strlen(line)==1)Count--;/* to discount the last \n in file
271                                 for the case the user has typed it */
272   rewind(Fp);
273   return Count;
274 }
```

Matrizes de números complexos

```
275 /*************************************************************************
276 * FUNC: long GetFileNumSamples(FILE *Fp)
277 *
278 * DESC: Return number of numeric samples in file pointed by Fp. Returns -1
279 *       on read error.
280 *************************************************************************/
281 long GetFileNumSamples(FILE *Fp)
282 {
283
284 long register Count;
285 float x;
286 int test;
287
288 Count=0;
289
290 while((test=fscanf(Fp,"%f",&x))!=-1){
291 if(test!=1)return -1;
292 Count++;
293 }
294
295 rewind(Fp);
296 return Count;
297 }
```

Matrizes de números complexos

```
298 /*****************************************************************************
299 * FUNC: float  **FloatMatRead(float **M,char *file,char *buffer,unsigned *NRows,unsigned *NCols, char Al
300 *
301 * DESC: Reads an ascii float matrix from 'file' and store it in matrix
302 *       M[NRows][NCols]. If AllocFlag==1 the function allocates memory
303 *       for M and the arguments *NRows and *NCols identify the size of M.
304 *****************************************************************************/
305 float  **FloatMatRead(float **M,char *file,char *buffer,unsigned *NRows,unsigned *NCols, char AllocFlag)
306 {
307 unsigned register i,j;
308 FILE *in;
309 long NumSamples;
310 unsigned long NumLines;
311
312 if((in=fopen(file,"rt"))==NULL)Quit(READ_OPEN_ERR,file);
313 if (setvbuf(in, buffer, _IOFBF, BUFSIZE) != 0)Quit(BUFFER_ERR,file);
314
315 NumSamples=GetFileNumSamples(in);
316 if(NumSamples==-1)Quit(READ_ERR,file);
317 NumLines=GetFileNumLines(in);
318
319 if(NumSamples%NumLines)Quit(MTXDIM_ERR,file); /* check valid matrix dimensions */
320
321 *NRows=NumLines;
322 *NCols=NumSamples/NumLines;
323
```

Matrizes de números complexos

```
324  //printf("NRows=%d\tNCols=%d\n",*NRows,*NCols);
325
326  if(AllocFlag==1){
327  M=FloatMatCAlloc(*NRows,*NCols);
328  }
329
330  for(i=0;i<*NRows;i++){
331  for(j=0;j<*NCols;j++){
332  fscanf(in,"%f",&M[i][j]);
333  }
334  }
335
336  fclose(in);
337
338  return M;
339  }
```

Matrizes de números complexos

```
340 /*****************************************************************************
341 * FUNC: void FloatMatWrite(float **M,char *file,char *buffer,unsigned NRows,unsigned NCols)
342 *
343 * DESC: Writes a float NRows x NCols matrix to an ascii file. If file=NULL, it
344 *       writes to stdout. If buffer=NULL, no buffer is attached to the file.
345 *****************************************************************************/
346 void FloatMatWrite(float **M,char *file,char *buffer,unsigned NRows,unsigned NCols)
347 {
348 unsigned register i,j;
349 FILE *out;
350 int test;
351
352 if(file!=NULL){
353 if((out=fopen(file,"wt"))==NULL)Quit(WRITE_OPEN_ERR,file);
354 if(buffer!=NULL)if (setvbuf(out, buffer, _IOFBF, BUFSIZE) != 0)Quit(BUFFER_ERR,file);
355 }
356 else out=stdout;
357
358 for(i=0;i<NRows;i++){
359 for(j=0;j<NCols;j++){
360
361 fprintf(out,"%8.6g\t",M[i][j]);
362 }
363
364 test=fprintf(out,"\n");
365 if(test!=sizeof(char))Quit(WRITE_ERR,file);
366 }
367
368 if(file!=NULL)fclose(out);
369 }
```

Matrizes de números complexos

```
370 /*****************************************************************************
371 * FUNC: void CpxMatWrite(COMPLEX **M,char *file,char *buffer,unsigned NRows,unsigned NCols)
372 *
373 * DESC: Writes a COMPLEX NRows x NCols matrix to an ascii file. If file=NULL, it
374 *       writes to stdout. If buffer=NULL, no buffer is attached to the file.
375 *****************************************************************************/
376 void CpxMatWrite(COMPLEX **M,char *file,char *buffer,unsigned NRows,unsigned NCols)
377 {
378
379 unsigned register i,j;
380 FILE *out;
381 int test;
382
383 if(file!=NULL){
384 if((out=fopen(file,"wt"))==NULL)Quit(WRITE_OPEN_ERR,file);
385 if(buffer!=NULL)if (setvbuf(out, buffer, _IOFBF, BUFSIZE) != 0)Quit(BUFFER_ERR,file);
386 }
387 else out=stdout;
388
389 for(i=0;i<NRows;i++){
390 for(j=0;j<NCols;j++){
391 fprintf(out,"%8.6g\t%8.6g\t",M[i][j].Re,M[i][j].Im);
392 }
393
394 test=fprintf(out,"\n");
395 if(test!=sizeof(char))Quit(WRITE_ERR,file);
396 }
397
398 if(file!=NULL)fclose(out);
399 }
```

Matrizes de números complexos

```
400 /*****************************************************************************
401 * FUNC: void CpxMatElementConjugate(COMPLEX **Mat,unsigned NRows,unsigned NCols)
402 *
403 * DESC: Conjugate all elements of NRows x NCols Mat COMPLEX matrix.
404 *****************************************************************************/
405 void CpxMatElementConjugate(COMPLEX **Mat,unsigned NRows,unsigned NCols)
406 {
407 unsigned register i,j;
408
409 for(i=0;i<NRows;i++){
410 for(j=0;j<NCols;j++){
411
412     Mat[i][j].Im=-Mat[i][j].Im;
413
414 }
415 }
416
417 }
```

Matrizes de números complexos

Execução na tela do console – tela de "help" resultante ao se digitar o nome do executável (ConjCpxMat.exe) com a linha de comando sem argumentos :

```
Reads COMPLEX 2D matrix from input text file whose name is given by argument InputMatrix,
stores read values in COMPLEX matrix CMat allocated in heap memory, conjugate the elements of CMat
and writes conjugated CMat to output text file whose name is given by argument OutputMatrix.

Use: ConjCpxMat InputMatrix OutputMatrix
```

Execução na tela do console – linha de comando conforme especificada abaixo:

ConjCpxMat IQ_256QAM.txt ConjIQ_256QAM.txt

Para validar os resultados da execução do programa ConjCpxMat.exe com o software MathCad vamos utilizar o *script* em https://www.fccdecastro.com.br/ZIP/TestaConjCpxMat.zip, conforme mostrado no próximo slide.

Matrizes de números complexos

Este script testa a conformidade do codigo fonte ConjCpxMat.c (slide 58 do Cap III.2 das notas de aula de "Linguagem C p/ processamento de sinais"):

Lê o arquivo gerado pelo script MathCad QAMmatrix.xmcd e armazena na matriz Mat_M:

$$\text{InpFileM} := \text{"IQ\_256QAM.txt"}$$

$$\text{Mat\_M} := \text{READPRN}(\text{InpFileM})$$

Número de linhas e colunas da matriz Mat_M:

$$\text{NRows\_M} := \text{rows}(\text{Mat\_M}) = 16$$

$$\text{NCols\_M} := \text{cols}(\text{Mat\_M}) = 32$$

Divide número de colunas por 2 p/ a representação de Mat_M em forma complexa:

$$\text{NCols\_M} := \frac{\text{NCols\_M}}{2}$$

Índices i e j das linhas e colunas da representação de Mat_M em forma complexa:

$$i := 0 .. \text{NRows\_M} - 1 \qquad j := 0 .. \text{NCols\_M} - 1$$

Converte Mat_M para a representação em forma complexa:

$$\text{CMat\_M}^{\langle j \rangle} := \text{Mat\_M}^{\langle 2 \cdot j \rangle} + j \cdot \text{Mat\_M}^{\langle 2 \cdot j + 1 \rangle}$$

Lê o arquivo gerado pelo executavel compilado e linkado a partir do fonte ConjCpxMat.c e armazena na matriz Mat_C:

$$\text{InpFileC} := \text{"ConjIQ\_256QAM.txt"}$$

$$\text{Mat\_C} := \text{READPRN}(\text{InpFileC})$$

Número de linhas e colunas da matriz Mat_C:

$$\text{NRows\_C} := \text{rows}(\text{Mat\_C}) = 16$$

$$\text{NCols\_C} := \text{cols}(\text{Mat\_C}) = 32$$

Divide número de colunas por 2 p/ a representação de Mat_C em forma complexa:

$$\text{NCols\_C} := \frac{\text{NCols\_C}}{2}$$

Índices i_ e j_ das linhas e colunas da representação de Mat_C em forma complexa:

$$i\_ := 0 .. \text{NRows\_C} - 1 \qquad j\_ := 0 .. \text{NCols\_C} - 1$$

Converte Mat_C para a representação em forma complexa:

$$\text{CMat\_C}^{\langle j\_ \rangle} := \text{Mat\_C}^{\langle 2 \cdot j\_ \rangle} + j \cdot \text{Mat\_C}^{\langle 2 \cdot j\_ + 1 \rangle}$$

Matrizes de números complexos

CMat_M =

	0	1	2	3	4	5	6	7	8	9
0	-15+15j	-13+15j	-11+15j	-9+15j	-7+15j	-5+15j	-3+15j	-1+15j	1+15j	3+15j
1	-15+13j	-13+13j	-11+13j	-9+13j	-7+13j	-5+13j	-3+13j	-1+13j	1+13j	3+13j
2	-15+11j	-13+11j	-11+11j	-9+11j	-7+11j	-5+11j	-3+11j	-1+11j	1+11j	3+11j
3	-15+9j	-13+9j	-11+9j	-9+9j	-7+9j	-5+9j	-3+9j	-1+9j	1+9j	3+9j
4	-15+7j	-13+7j	-11+7j	-9+7j	-7+7j	-5+7j	-3+7j	-1+7j	1+7j	3+7j
5	-15+5j	-13+5j	-11+5j	-9+5j	-7+5j	-5+5j	-3+5j	-1+5j	1+5j	3+5j
6	-15+3j	-13+3j	-11+3j	-9+3j	-7+3j	-5+3j	-3+3j	-1+3j	1+3j	3+3j
7	-15+j	-13+j	-11+j	-9+j	-7+j	-5+j	-3+j	-1+j	1+j	3+j
8	-15-j	-13-j	-11-j	-9-j	-7-j	-5-j	-3-j	-1-j	1-j	3-j
9	-15-3j	-13-3j	-11-3j	-9-3j	-7-3j	-5-3j	-3-3j	-1-3j	1-3j	3-3j
10	-15-5j	-13-5j	-11-5j	-9-5j	-7-5j	-5-5j	-3-5j	-1-5j	1-5j	3-5j
11	-15-7j	-13-7j	-11-7j	-9-7j	-7-7j	-5-7j	-3-7j	-1-7j	1-7j	3-7j
12	-15-9j	-13-9j	-11-9j	-9-9j	-7-9j	-5-9j	-3-9j	-1-9j	1-9j	3-9j
13	-15-11j	-13-11j	-11-11j	-9-11j	-7-11j	-5-11j	-3-11j	-1-11j	1-11j	3-11j
14	-15-13j	-13-13j	-11-13j	-9-13j	-7-13j	-5-13j	-3-13j	-1-13j	1-13j	3-13j
15	-15-15j	-13-15j	-11-15j	-9-15j	-7-15j	-5-15j	-3-15j	-1-15j	1-15j	...

Matrizes de números complexos

CMat_C =

	0	1	2	3	4	5	6	7	8	9
0	-15-15j	-13-15j	-11-15j	-9-15j	-7-15j	-5-15j	-3-15j	-1-15j	1-15j	3-15j
1	-15-13j	-13-13j	-11-13j	-9-13j	-7-13j	-5-13j	-3-13j	-1-13j	1-13j	3-13j
2	-15-11j	-13-11j	-11-11j	-9-11j	-7-11j	-5-11j	-3-11j	-1-11j	1-11j	3-11j
3	-15-9j	-13-9j	-11-9j	-9-9j	-7-9j	-5-9j	-3-9j	-1-9j	1-9j	3-9j
4	-15-7j	-13-7j	-11-7j	-9-7j	-7-7j	-5-7j	-3-7j	-1-7j	1-7j	3-7j
5	-15-5j	-13-5j	-11-5j	-9-5j	-7-5j	-5-5j	-3-5j	-1-5j	1-5j	3-5j
6	-15-3j	-13-3j	-11-3j	-9-3j	-7-3j	-5-3j	-3-3j	-1-3j	1-3j	3-3j
7	-15-j	-13-j	-11-j	-9-j	-7-j	-5-j	-3-j	-1-j	1-j	3-j
8	-15+j	-13+j	-11+j	-9+j	-7+j	-5+j	-3+j	-1+j	1+j	3+j
9	-15+3j	-13+3j	-11+3j	-9+3j	-7+3j	-5+3j	-3+3j	-1+3j	1+3j	3+3j
10	-15+5j	-13+5j	-11+5j	-9+5j	-7+5j	-5+5j	-3+5j	-1+5j	1+5j	3+5j
11	-15+7j	-13+7j	-11+7j	-9+7j	-7+7j	-5+7j	-3+7j	-1+7j	1+7j	3+7j
12	-15+9j	-13+9j	-11+9j	-9+9j	-7+9j	-5+9j	-3+9j	-1+9j	1+9j	3+9j
13	-15+11j	-13+11j	-11+11j	-9+11j	-7+11j	-5+11j	-3+11j	-1+11j	1+11j	3+11j
14	-15+13j	-13+13j	-11+13j	-9+13j	-7+13j	-5+13j	-3+13j	-1+13j	1+13j	3+13j
15	-15+15j	-13+15j	-11+15j	-9+15j	-7+15j	-5+15j	-3+15j	-1+15j	1+15j	...

Matrizes de números complexos

Conjuga os elementos da matriz complexa CMat_C : $\mathrm{CMat\_C}_{i\_,j\_} := \overline{\mathrm{CMat\_C}_{i\_,j\_}}$

$\mathrm{CMat\_M} - \mathrm{CMat\_C} =$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Testa se a média dos elementos de $\mathrm{CMat\_M} - \mathrm{CMat\_C}$ resulta zero, indicando conformidade do fonte ConjCpxMat.c:

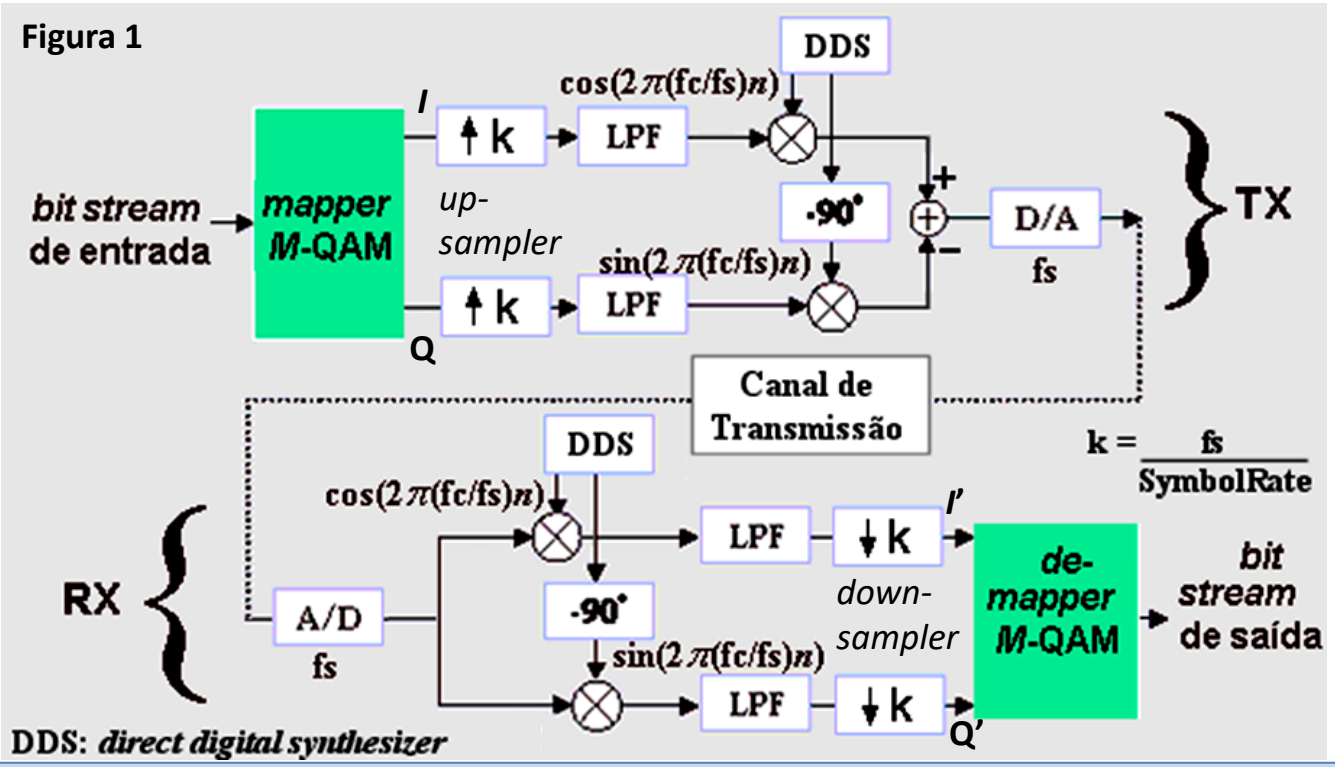
$$\mathrm{mean}\left[\overrightarrow{\left(\left|\mathrm{CMat\_M} - \mathrm{CMat\_C}\right|\right)}\right] = 0$$

Convolução entre duas sequências de valores complexos

A Figura 1 abaixo mostra o diagrama de blocos simplificado de um sistema de comunicação sem fio que utiliza modulação digital 16-QAM. O transmissor (TX) converte a informação nas palavras binárias do *bit stream* de entrada em um *stream* de números complexos I+*j*Q denominado de IQ *stream.* O IQ *stream* é obtido a partir do *bit stream* de entrada através da operação de mapeamento mostrada na Figura 2, operação que é realizada no *mapper* do TX. O IQ *stream* é então enviado através do canal de transmissão, i.e, através do espaço entra a antena do TX e a antena do RX. O RX recebe o IQ *stream* I'+*j*Q' do canal de transmissão e, idealmente, recupera o *bit stream* original através da operação de mapeamento inverso mostrada na Figura 2, operação que é realizada no *de-mapper* do RX. Detalhes do processo de modulação e demodulação digital podem ser encontrados nos slides 2 a 20 de https://www.fccdecastro.com.br/pdf/SCD1_CapIV.pdf .

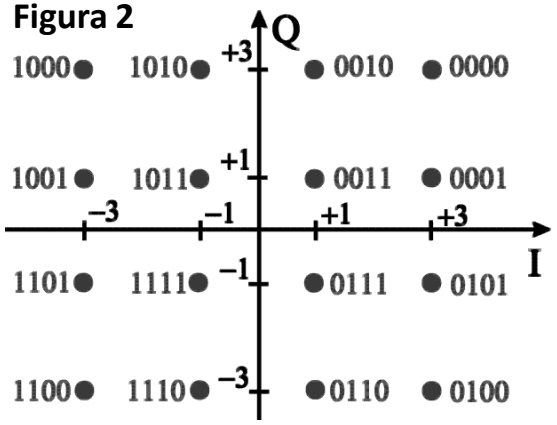
Ocorre que todo canal de transmissão, em particular o canal sem fio, degrada a inteligibilidade do IQ *stream* que por ele trafega em função das múltiplas reflexões da onda eletromagnética (EM) que transporta o IQ *stream*. Esta degradação por múltiplas reflexões da onda EM é denominada de ***multipath*** (multipercurso). O *multipath* dá origem à múltiplos ecos no IQ *stream* recebido no RX, degradando sua inteligibilidade (ver https://www.fccdecastro.com.br/pdf/SCD2_Cap%20II.pdf).

Figura 1



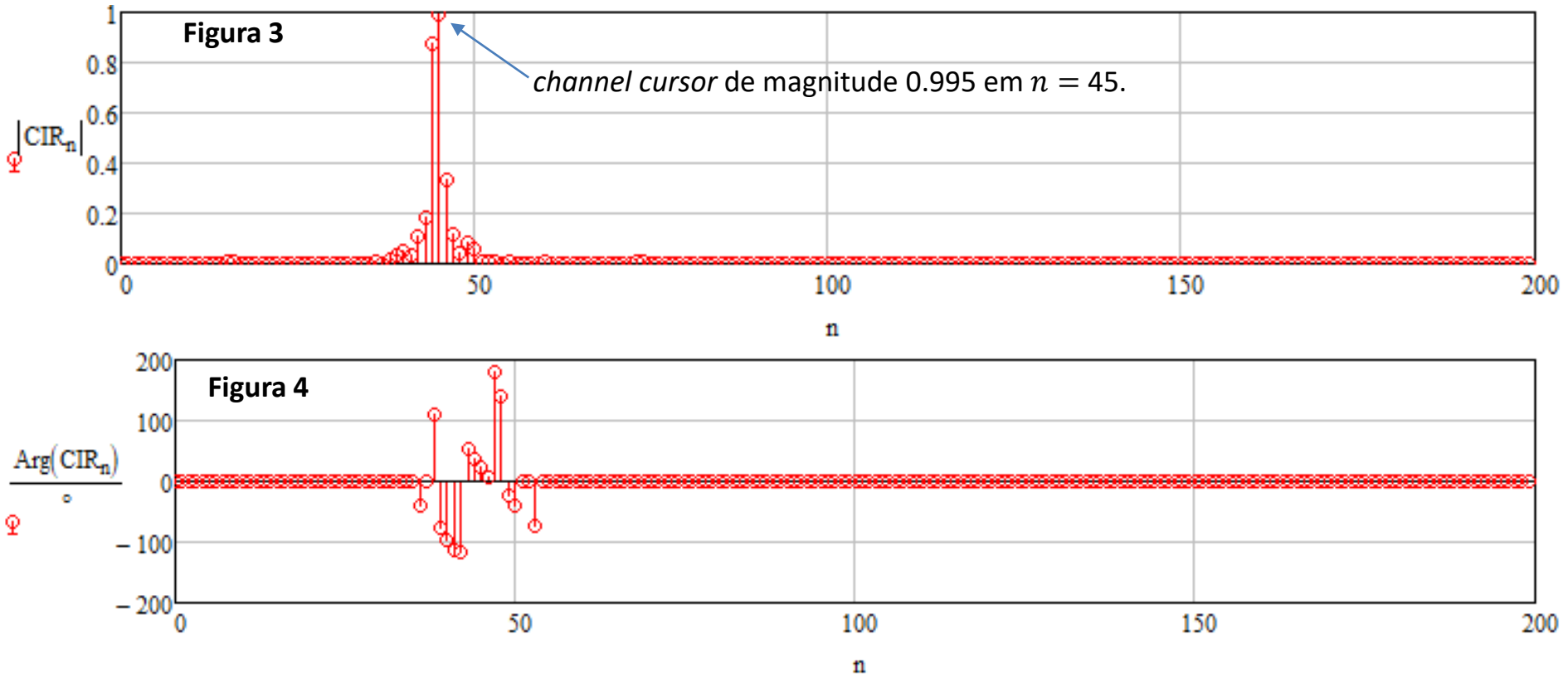
Estes ecos se fazem presentes na **resposta ao impulso do canal de transmissão**, cada eco constituindo um impulso na resposta discreta ao impulso do canal, conforme veremos no próximo slide.

Figura 2



Convolução entre duas sequências de valores complexos

Por exemplo, as figuras 3 e 4 abaixo mostram respectivamente os *plots* do módulo e fase da CIR (*channel impulse response*) discreta do canal de microondas "chan 13", disponível em http://spib.linse.ufsc.br/microwave.html. O arquivo "chan13.mat" disponível no referido *link* foi convertido no Matlab para o arquivo texto chan13.txt e foi plotado usando o *script* MathCad "Microwave CIR.xmcd". O arquivos "chan13.txt" e "Microwave CIR.xmcd" estão disponíveis em https://www.fccdecastro.com.br/ZIP/CpxConv.zip .



Note na Figura 3 que o ***channel cursor*** (cursor do canal = eco de maior amplitude) tem uma amplitude de 0.995 e ocorre no índice $n = 45$. Note que antes e depois do *channel cursor* em $n = 45$ ocorrem pré-ecos e pós-ecos de amplitude menor, característica que é típica de cenários de *multipath* urbano.

Consideremos, então, que o sistema 16-QAM descrito pelas figuras 1 e 2 transmite o IQ *stream* mostrado nas figuras 5 e 6 do próximo slide e cuja constelação de símbolos IQ correspondentes é mostrado na Figura 7.

As figuras 5,6 e 7 abaixo são plotadas a partir do arquivo "IQstream.txt" que define a sequência de símbolos IQ transmitidos pelo TX e a partir do *script* MathCad "IQ StreamOut.xmcd", ambos disponíveis em https://www.fccdecastro.com.br/ZIP/CpxConv.zip .

Figura 5

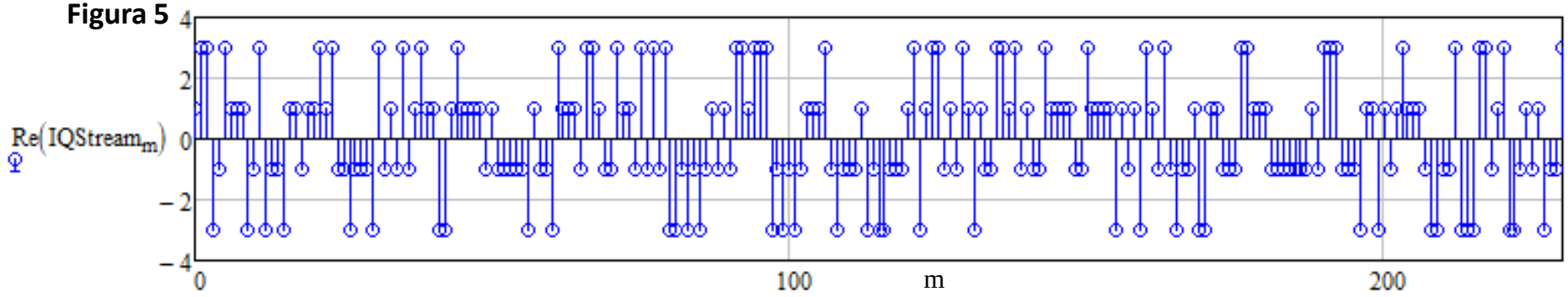


Figura 6

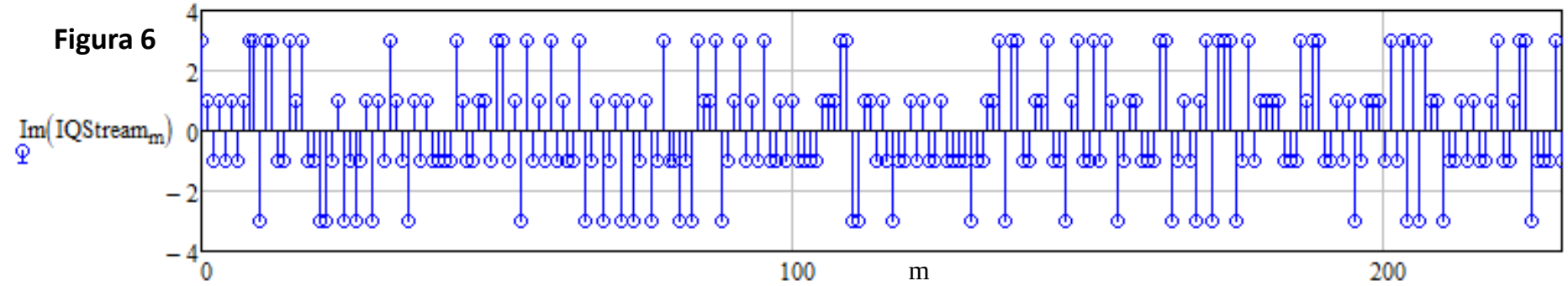
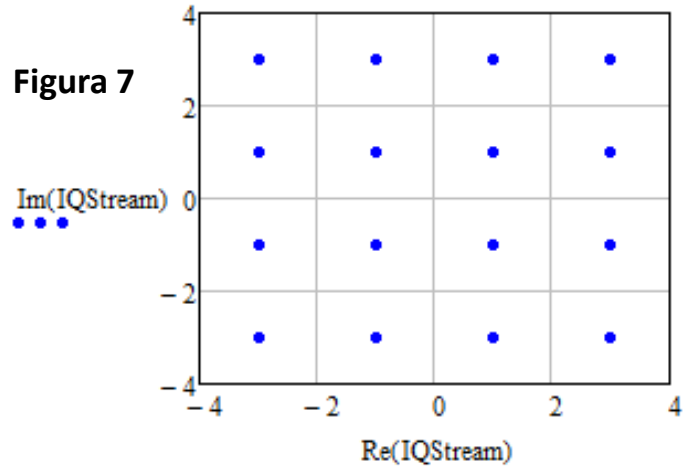


Figura 7



Da Figura 1 no slide 80 observa-se que o *stream* de símbolos IQ na saída do canal é o *stream* de símbolos IQ recebido pelo RX. Este *stream* de símbolos IQ recebido pelo RX, que denominaremos IQStreamOut, é resultante da **convolução** entre o *stream* de símbolos IQStream transmitido pelo TX e o *channel impulse response* CIR (ver slides 22 a 49 de https://www.fccdecastro.com.br/pdf/SS_Aula5&6_26032020.pdf) para relembrar o conceito de convolução). O sinal IQStreamOut é mostrado nas figuras 8 e 9 do próximo slide e cuja constelação de símbolos IQ correspondentes é mostrado na Figura 10.

As figuras 8,9 e 10 abaixo são plotadas a partir do *script* MathCad "IQ StreamOut.xmcd", disponível em https://www.fccdecastro.com.br/ZIP/CpxConv.zip .

Figura 8

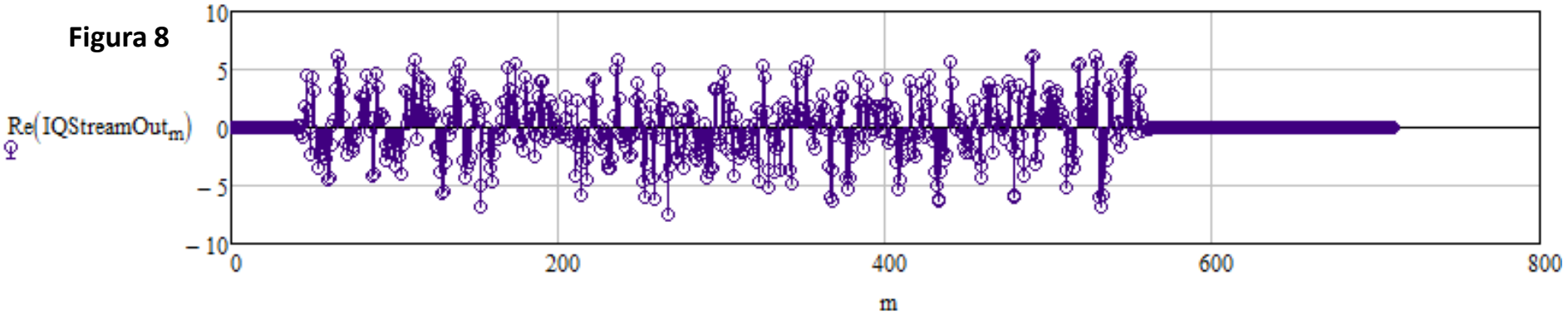


Figura 9

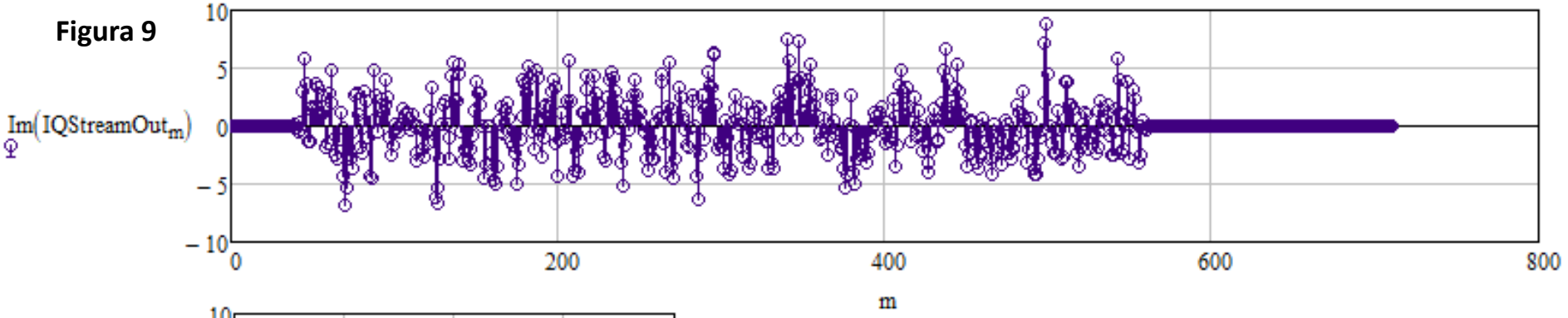
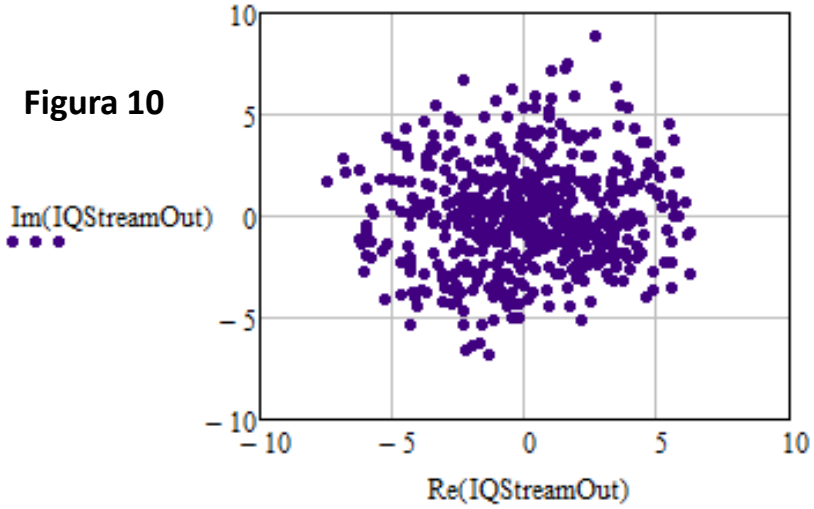


Figura 10



Comparando as figuras 7 e 10 se observa que a inteligibilidade da constelação do sinal recebido no RX (IQStreamOut) foi totalmente degradada pelo *multipath* do canal, não mantendo qualquer relação de semelhança com a constelação de símbolos IQ transmitida pelo TX (IQStream). Portanto, o *de-mapper* do RX efetuará inúmeras decisões erradas e o resultado será um grande número de palavras binárias de-mapeadas erroneamente, aumentando significativamente a BER (*bit error rate* – taxa de erro de bits) no *bit stream* de saída do *de-mapper*.

Homework - no contexto do que foi discutido nos slides anteriores, **pede-se**:

(a) Compile no compilador DevC++ e gere o executável do código fonte C "CpxConv_R1.c", disponível em em https://www.fccdecastro.com.br/ZIP/CpxConv.zip. O referido código fonte efetua a convolução entre duas sequências complexas e é baseado na seção 4.6 "Complex Filters" na página 255 da referência [1] no slide 9 do Cap III.1. Os argumentos da linha de comando do executável cpxconv.exe gerado são conforme abaixo:

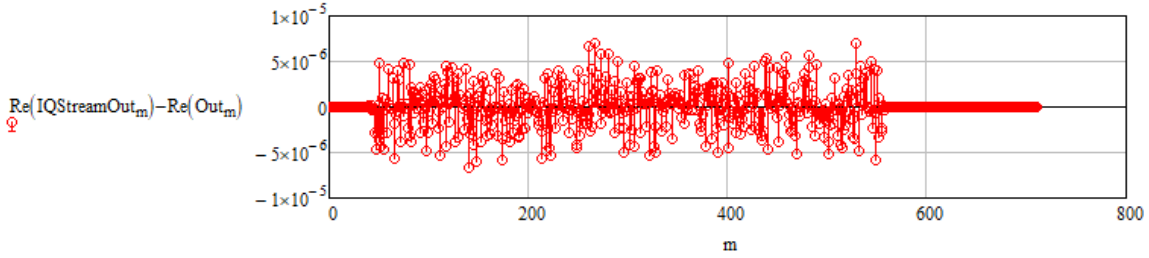
D:\C_DSP\CpxConv>cpxconv

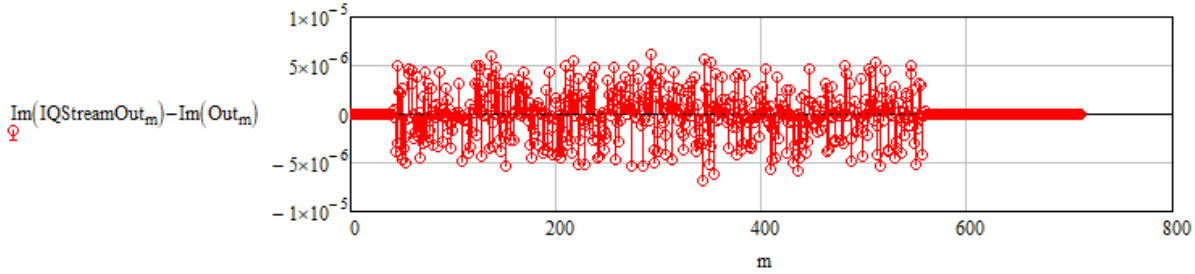
ComplexConvolution: InputSignal1_File InputSignal2_File OutputSignal_File

Note: Usually InputSignal1_File stores a complex signal to be filtered
and InputSignal2_File stores the impulse response (coefficients)
of a complex FIR filter. OutputSignal_File stores the filtered signal.
Denoting #{.} as the number_of_samples operator, we have:
#{OutputSignal}=#{InputSignal1}+#{InputSignal1}-1

(b) Utilize o executável cpxconv.exe para efetuar a convolução entre a CIR definida no arquivo "chan13.txt" e a sequência de símbolos IQ transmitida pelo TX 16-QAM definida no arquivo "IQstream.txt". Gravar o resultado da convolução no arquivo "Out.txt".

(c) Modifique e adicione os comandos necessários no *script* MathCad "IQ StreamOut.xmcd" de modo que o mesmo leia o arquivo "Out.txt" obtido em (b). Neste mesmo *script* MathCad modificado, plote e compare graficamente as partes real e imaginária das duas sequências de amostras resultantes da convolução complexa – a sequência complexa IQStreamOut resultante da função convol() nativa do MathCad e a sequência complexa armazenada em "Out.txt" (já lida), que é resultante do executável cpxconv.exe. Verifique e valide a consistência dos resultados plotando em um gráfico a diferença entre a parte real das duas sequências e plotando em outro gráfico a diferença entre a parte imaginária das duas sequências. Para efeito de validação, a diferença deve resultar pequena, conforme a seguir exemplificado:





(d) Modifique e adicione os comandos necessários ao código fonte C "CpxConv_R1.c" de modo que, após a compilação no DevC++, o novo e modificado executável cpxconv.exe identifique a magnitude (=amplitude) e o índice do *channel cursor* na CIR (ver slide 3), bem como identifique a máxima magnitude e o respectivo índice de ocorrência da máxima magnitude nas outras duas sequências complexas. Para tanto, conceber e codificar em C uma função com o seguinte protótipo:

```
/*************************************************************************
* FUNC:  float MaxOfCpxVecAtIndex(COMPLEX *Vec, unsigned VecSize, unsigned *IndexOfMax)
*
* DESC:  Return the maximum magnitude value in COMPLEX vector Vec and its index.
*************************************************************************/
float MaxOfCpxVecAtIndex(COMPLEX *Vec, unsigned VecSize, unsigned *IndexOfMax)
```

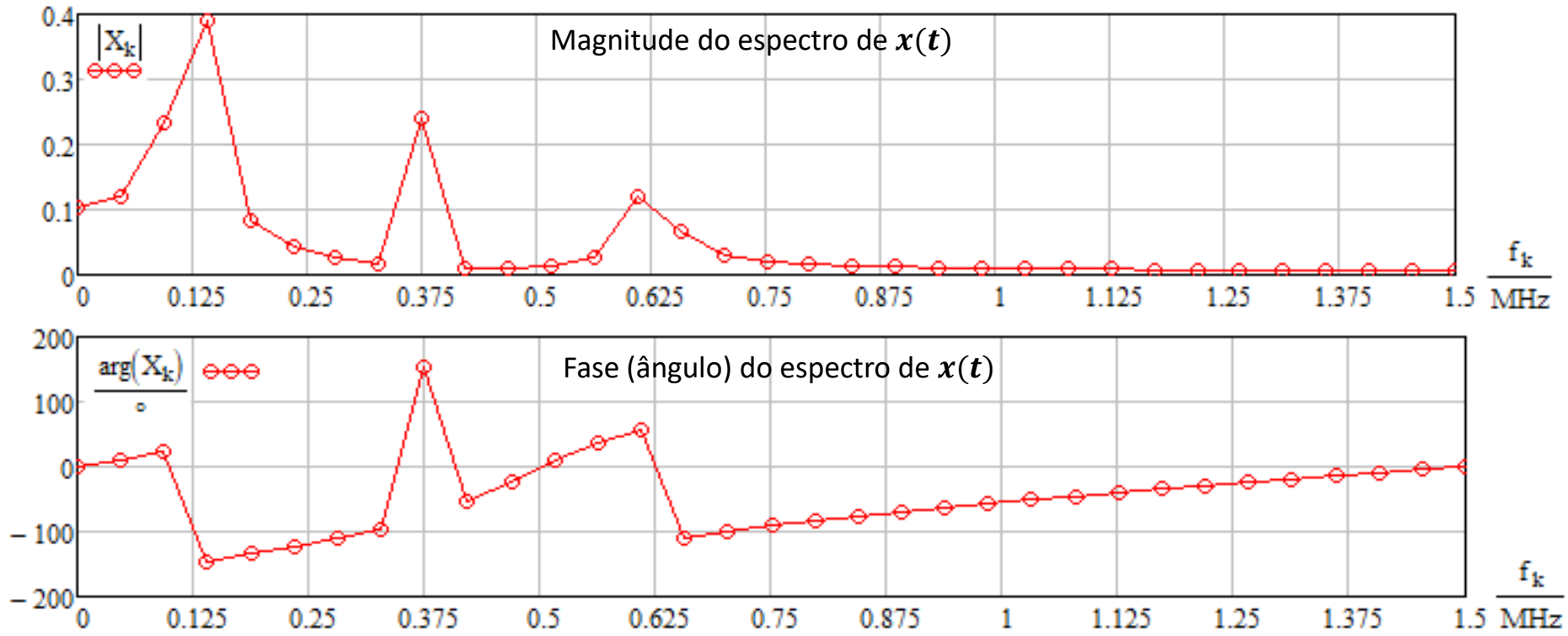
O resultado ao rodar o novo e modificado executável cpxconv.exe no *command prompt* do Windows deve ser conforme abaixo para o exemplo em questão:

```
D:\C_DSP\CpxConv> cpxconv chan13.txt IQstream.txt Out.txt
Max magnitude of chan13.txt is 0.995 at index 45.
Max magnitude of IQstream.txt is 4.243 at index 9.
Max magnitude of Out.txt is 9.216 at index 499.
```

FFT - *Fast Fourier Transform*

O **espectro** de um sinal $\boldsymbol{x(t)}$ que varia no tempo $\boldsymbol{t}$ é a representação matemática definida por uma função $\boldsymbol{X(f)}$ de valor complexo, cujo módulo $|\boldsymbol{X(f)}|$ expressa a **magnitude** (= intensidade) com que um particular sinal de frequência $\boldsymbol{f}$ "influencia" na variação no tempo do sinal $x(t)$ e cujo **ângulo** $\angle \boldsymbol{X(f)}$ expressa o deslocamento no tempo dentro do período $T = 1/f$ com que o sinal de frequência f "influencia" a variação no tempo do sinal $x(t)$, conforme exemplificado nos gráficos abaixo. Note que esta é uma interpretação informal. Note também que os 3 picos de maior magnitude no gráfico de $|X(f)|$ abaixo indicam que o sinal $x(t)$ é composto por 3 frequências de "maior influência".

Em uma interpretação mais específica e menos informal, o gráfico de $|X(f)|$ poderia expressar, por exemplo, a tela de um analisador de espectro quando o sinal $x(t)$ aplicado à sua porta de entrada é originado por uma antena receptora. Nesta situação, os picos em $|X(f)|$ representam as frequências de maior amplitude do sinal $x(t)$ captado pela antena, indicando que há 3 transmissores de rádio transmitindo nestas 3 respectivas frequências (ver slides 32 a 45 https://www.fccdecastro.com.br/pdf/SS_Aula2_12032020.pdf).



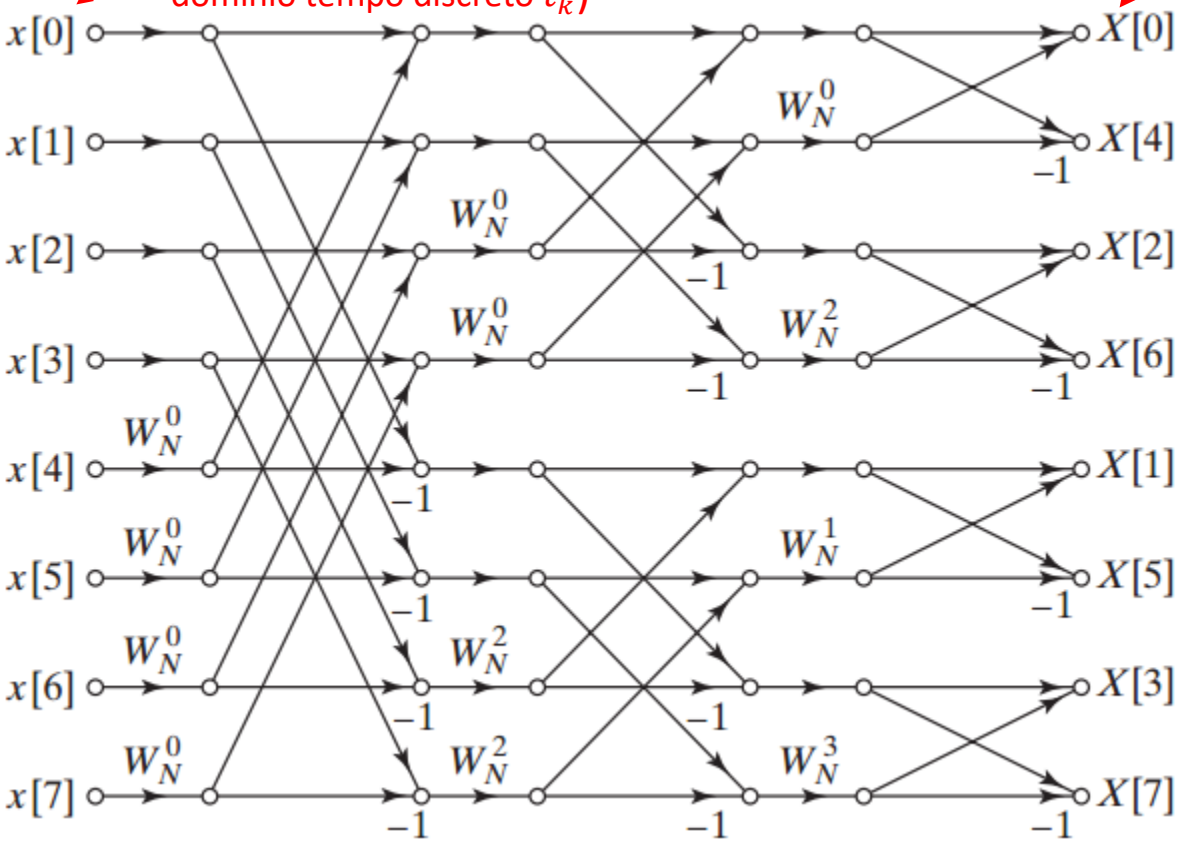
FFT - *Fast Fourier Transform*

Há diversas maneiras de determinar o espectro $X(f)$ de um sinal $x(t)$. Uma maneira usual em processamento de sinais é usar a FFT (*Fast Fourier Transform*), que é a implementação em forma de árvore binária da DFT (*Discrete Fourier Transform*) – ver https://www.fccdecastro.com.br/pdf/SS_aula27a29_06072020.pdf . Assim como a DFT, a FFT é representada por um operador da forma $X(f) = \text{FFT}\{x(t)\}$, operador que retorna o espectro $X(f)$ do argumento $x(t)$, onde $x(t)$ é um sinal do domínio tempo.

Por exemplo, o grafo de fluxo de sinal (https://en.wikipedia.org/wiki/Signal-flow_graph) abaixo mostra a operação $X(f_k) = \text{FFT}\{x(t_k)\}$ para $k = 0,1 \cdots N-1$ e $N = 8$.

$x[k] = x(t_k)$ (sinal amostrado no domínio tempo discreto t_k)

$\text{X}[k] = X(f_k)$ (espectro amostrado no domínio frequência discreta f_k)



A operação $X(f_k) = \text{FFT}\{x(t_k)\}$ converte o sinal $x[k] = x(t_k)$ com $N = 8$ amostras no domínio tempo discreto $t_k = k/f_s$ no seu espectro $\text{X}[k] = X(f_k)$ com mesmo número de amostras N no domínio frequência discreta $f_k = \frac{k}{N} f_s$.

A frequência f_s é a frequência de amostragem do conversor A/D que previamente converteu o sinal $x(t)$ no domínio tempo contínuo t no sinal $x(t_k)$ no domínio tempo discreto t_k.

A transmitância W_N^k nos ramos do grafo é dada por $W_N^k = e^{-j\frac{2\pi}{N}k}$. Ramos sem indicação de transmitância significam ramos com transmitância 1. Sinais que chegam em um nó do grafo são somados no nó e transmitidos adiante na(s) saída(s) do nó.

FFT - *Fast Fourier Transform*

O código fonte https://www.fccdecastro.com.br/CursoC&C++/C/cfft_di.c implementa a FFT direta e inversa (IFFT) de sequências de valores complexos, i.e, o programa implementa $X(f_k) = \mathrm{FFT}\{x(t_k)\}$ e implementa $x(t_k) = \mathrm{IFFT}\{X(f_k)\}$, onde $x(t_k)$ é uma sequência de valores complexos.

O "Help" da linha de comando de cfft_di.exe é conforme abaixo:

cfft_di: InpSignal_File OutSignal_File Mode[d/i]
Note: Mode='d' -> FFT, Mode='i' -> IFFT

No modo 'd' cfft_di.exe implementa $X(f_k) = \mathrm{FFT}\{x(t_k)\}$, onde a sequência de entrada $x(t_k)$ está armazenada no arquivo de nome especificado pelo argumento '**InpSignal_File'** e a sequência do espectro $X(f_k)$ resultante da operação é armazenada no arquivo de nome especificado pelo argumento '**OutSignal_File'**.

No modo 'i' cfft_di.exe implementa $x(t_k) = \mathrm{IFFT}\{X(f_k)\}$, onde a sequência de entrada referente ao espectro $X(f_k)$ está armazenada no arquivo de nome especificado pelo argumento '**InpSignal_File'** e a sequência do sinal $x(t_k)$ recuperado no tempo através da operação inversa é armazenada no arquivo de nome especificado pelo argumento '**OutSignal_File'**.

Os arquivos de nomes especificados pelos argumentos '**InpSignal_File'** e '**OutSignal_File'** são ambos arquivos texto de 2 colunas numéricas, cada coluna respectivamente representando a parte real e a parte imaginária da amostra de valor complexo referenciada em cada linha do arquivo. Cada linha do arquivo representa, portanto, o valor da respectiva amostra complexa na sequência de amostras complexas armazenada no arquivo.

Para efeito de verificar a conformidade dos resultados da execução do programa cfft_di.exe vamos utilizar o *script* MathCad em https://www.fccdecastro.com.br/ZIP/TestaCpxFFT.zip, conforme mostrado nos próximos slides.

FFT - *Fast Fourier Transform*

Este script testa a conformidade do codigo fonte cfft_di.c (link no slide 88 do Cap III.3 das notas de aula de "Linguagem C p/ processamento de sinais"):

Linha de comando de **cfft_di.exe** após executar uma vez este *script* (Ctrl<F9>) para efeito de gerar o arquivo "**x(t).txt**":

cfft_di x(t).txt X(f).txt d → Executa X(f)=FFT{x(t)}

cfft_di X(f).txt _x(t).txt i → Executa _x(t)=IFFT{X(f)}

"Help" da linha de comando de cfft_di.exe:

cfft_di: InpSignal_File OutSignal_File Mode[d/i]
Note: Mode='d' -> FFT, Mode='i' -> IFFT

Parâmetros de configuração deste *script*:

$N := 2^6 = 64$ → Número de pontos da FFT/IFFT (sempre uma potência inteira de 2 !!!)

$OutFile\_t := "x(t).txt"$ → Arquivo das amostras geradas neste *script* para o sinal complexo $x(t)$ no dominio tempo discreto t_k (ver definição de $x(t)$ e de x_k abaixo). O referido arquivo é argumento na linha de comando **cfft_di x(t).txt X(f).txt d**, notando que **X(f)=FFT{x(t)}**.

$InpFile\_f := "X(f).txt"$ → Arquivo das amostras do espectro complexo **X(f)** no domínio frequência discreto f_k, amostras que são geradas por cfft_di.exe. O referido arquivo é argumento na linha de comando **cfft_di x(t).txt X(f).txt d**, notando que **X(f)=FFT{x(t)}**.

FFT - *Fast Fourier Transform*

$InpFile\_t := "\_x(t).txt"$ → Arquivo das amostras do sinal complexo **_x(t)** no dominio tempo discreto t_k, amostras que são geradas por cfft_di.exe. O referido arquivo é argumento na linha de comando **cfft_di X(f).txt _x(t).txt i**, notando que **_x(t)=IFFT{X(f)}**.

$f_s := 3MHz$ → Frequência de amostragem com que o sinal complexo $x(t)$ no domínio tempo contínuo (ver definição de $x(t)$ abaixo) é convertido em um sinal amostrado x_k no domínio tempo discreto.

$T_s := \frac{1}{f_s} = 0.333 \cdot \mu s$ → Intervalo de tempo entre as amostras do sinal amostrado x_k no domínio tempo discreto.

$f_o := 0.125MHz$ → Frequência da 1ª harmônica (fundamental) do sinal complexo $x(t)$ no domínio tempo contínuo.

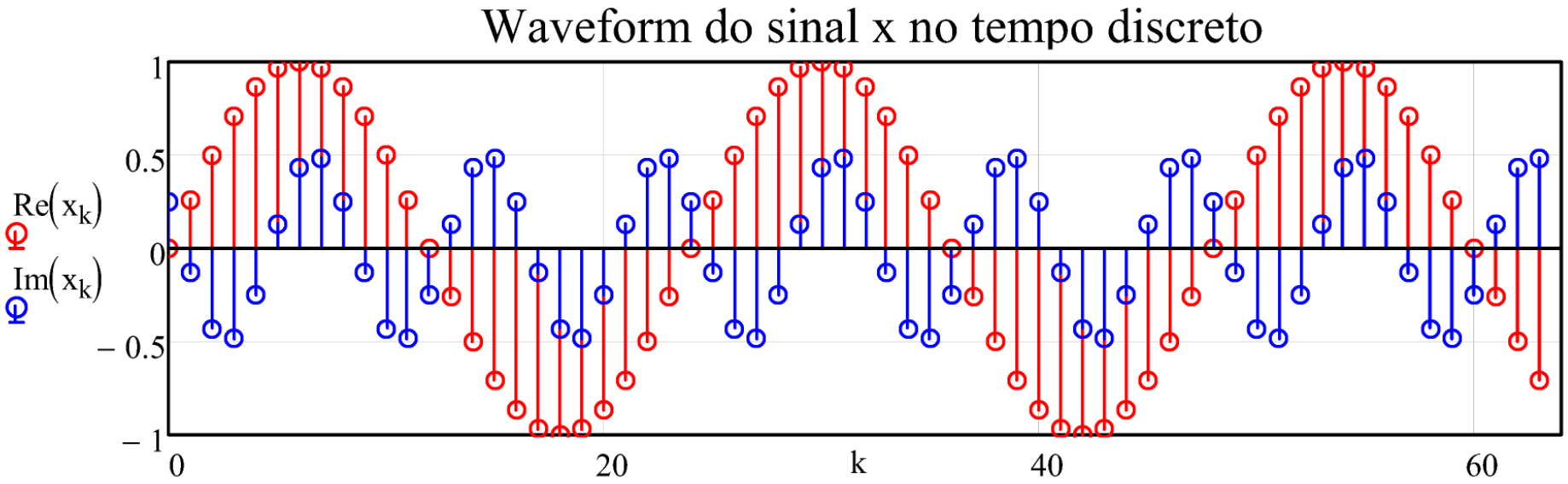
$x(t) := \sin\left(2 \cdot \pi \cdot f_o \cdot t\right) + j \cdot 0.5 \cdot \sin\left(2 \cdot \pi \cdot 3 \cdot f_o \cdot t + 150°\right)$ → Definição do sinal complexo $x(t)$ no domínio tempo contínuo.

$k := 0 .. N - 1$ → Índice das amostras nos domínios tempo discreto t_k e frequência discreta f_k.

$f_k := \frac{k}{N} \cdot f_s$ → Definição do domínio frequência discreta f_k.

$t_k := T_s \cdot k$ → Definição do domínio tempo discreto t_k.

$x_k := x\left(t_k\right)$ → Amostragem do sinal complexo $x(t)$ no domínio tempo contínuo, convertendo $x(t)$ em um sinal complexo x_k no domínio tempo discreto t_k .



PRNPRECISION:= 6 → representa o número de dígitos significativos a serem usados quando se escreve para um arquivo texto com a função WRITEPRN().

PRNCOLWIDTH:= 18 → representa a largura de cada coluna quando se escreve para um arquivo texto com a função WRITEPRN().

Atribuindo os N valores da parte real do vetor x_k e os N valores da parte imaginária do vetor x_k respectivamente à coluna 0 e à coluna 1 da matriz Dat[N][2]:

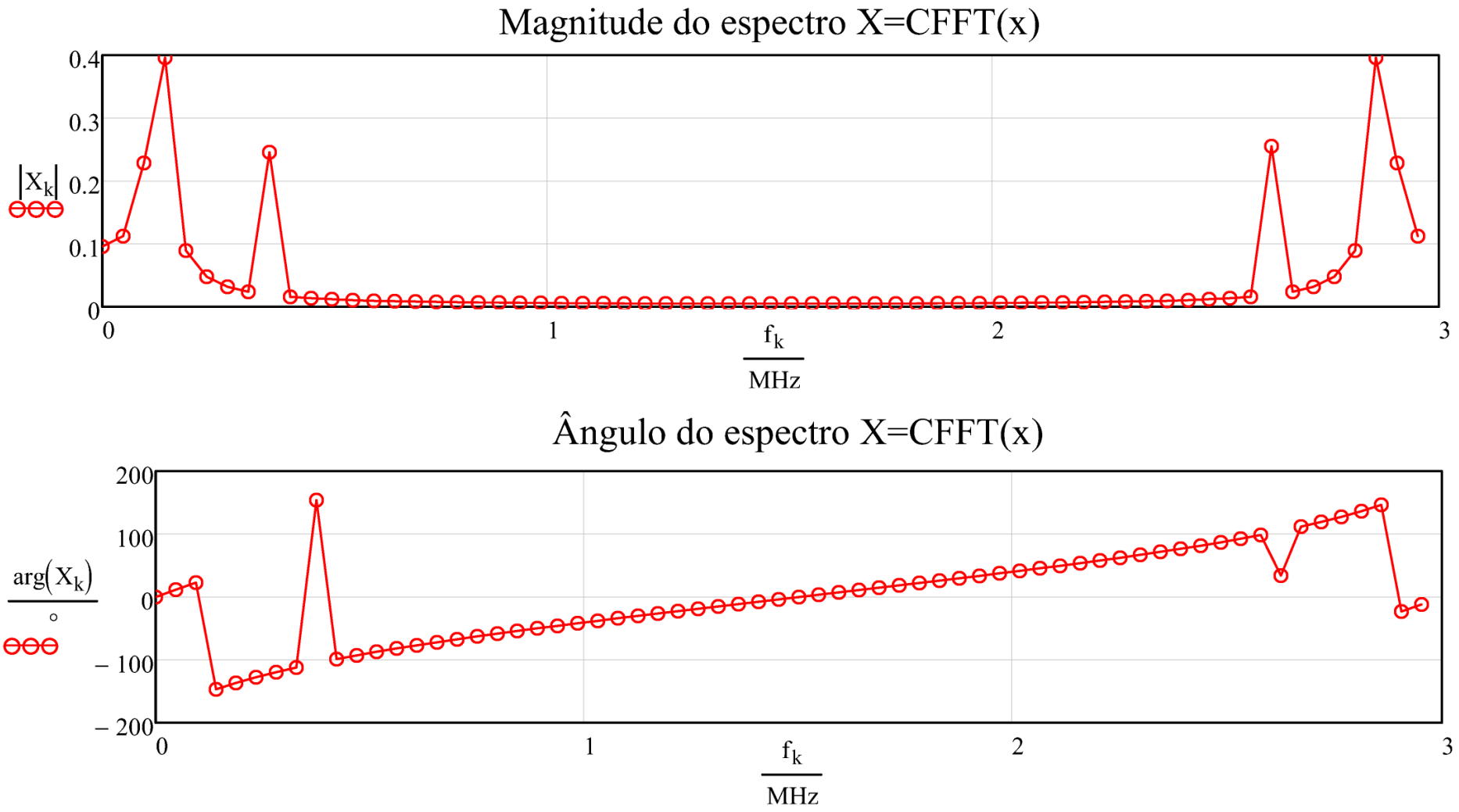
$$Dat^{\langle 0 \rangle} := Re(x) \qquad Dat^{\langle 1 \rangle} := Im(x)$$

Grava a matriz Dat no arquivo texto de nome dado por $OutFile\_t = "x(t).txt"$:

$$WRITEPRN(OutFile\_t) := Dat$$

FFT - *Fast Fourier Transform*

$X := CFFT(x)$ → Determina o espectro **X(f)=FFT{x(t)}**

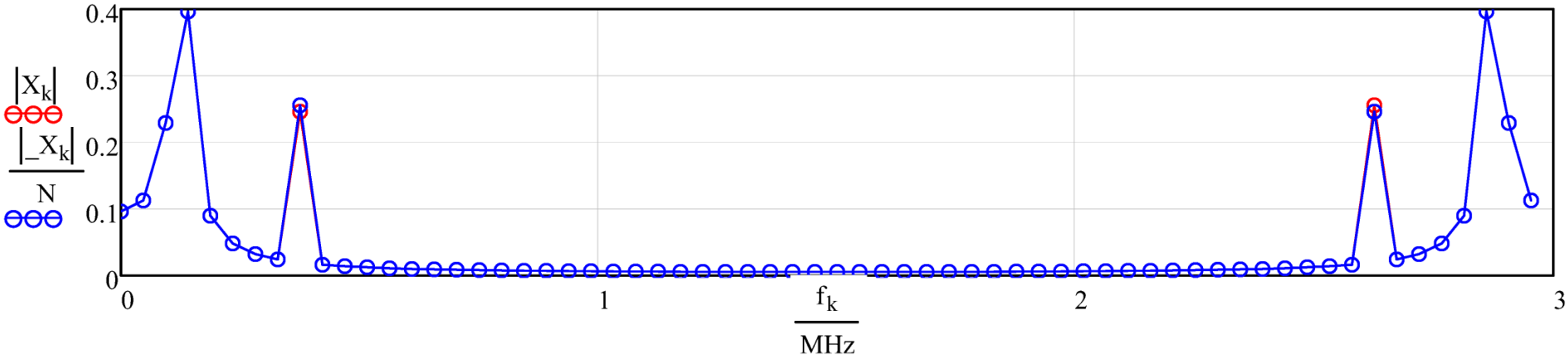


FFT - *Fast Fourier Transform*

$Dat := READPRN(InpFile\_f)$ → Lê o arquivo $InpFile\_f = "X(f).txt"$ contendo as amostras do espectro complexo **X(f)=FFT{x(t)}** no domínio frequência discreto $freq_k$, gerado por **cfft_di x(t).txt X(f).txt d**.

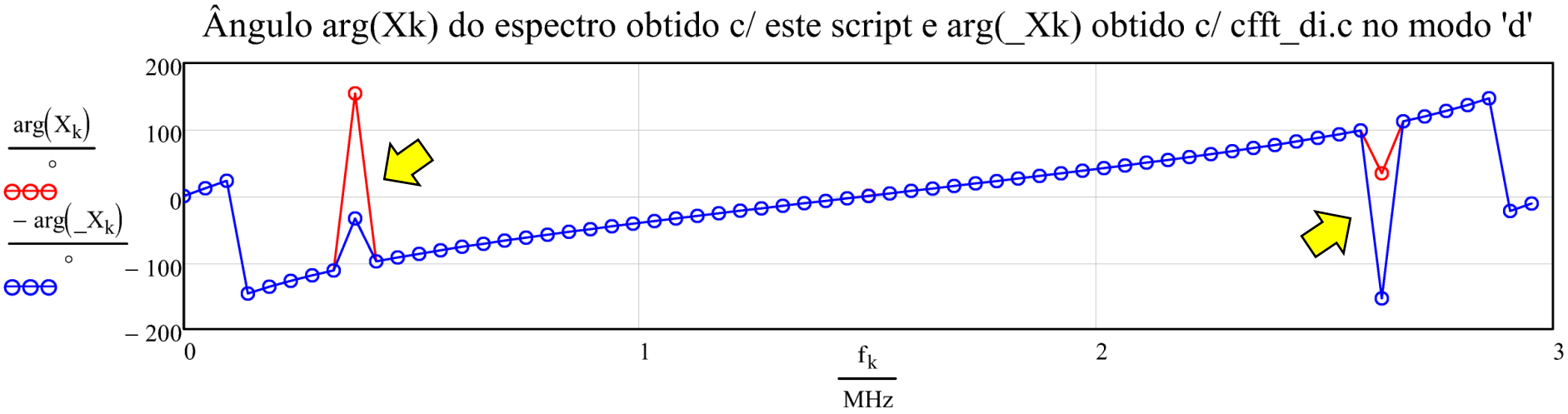
$\_X := Dat^{\langle 0 \rangle} + j \cdot Dat^{\langle 1 \rangle}$ → Atribui à cada elemento do vetor $\_X$ os valores das colunas 1 e 2 na respectiva linha da matriz Dat convertidos para valor complexo.

Magnitude |Xk| do espectro obtido c/ este script e |_Xk| obtido c/ cfft_di.c no modo 'd'



→ Note no gráfico acima que o módulo do espectro X_k gerado neste script é N vezes menor do que o módulo do espectro $\_X_k$ gerado por **cfft_di x(t).txt X(f).txt d**.

Dado que, a menos do fator de escala N, há conformidade do módulo do espectro $|X_k|$ determinado neste script com o módulo do espectro $|\_X_k|$ determinado através de cfft_di.exe no modo 'd', considera-se que cfft_di.exe está validado para este modo quanto à magnitude do espectro.



→ Note no gráfico acima que o ângulo do espectro X_k gerado neste script tem o sinal contrário do sinal do ângulo do espectro $\_X_k$ gerado por **cfft_di x(t).txt X(f).txt d**.

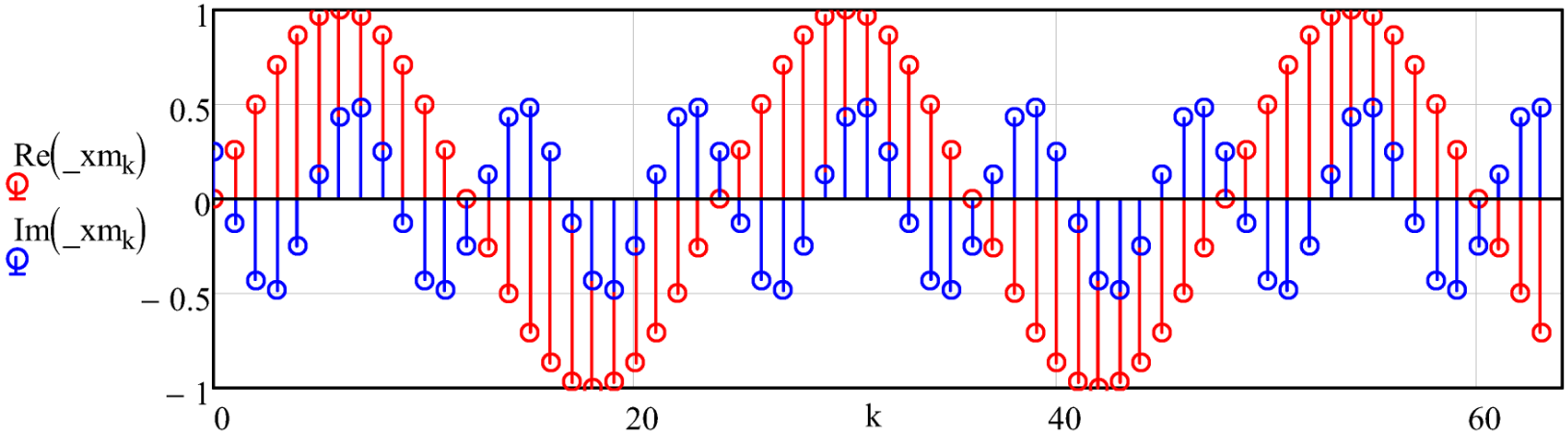
Dado que, a menos de uma inversão de sinal, há conformidade do ângulo do espectro $\arg(X_k)$ determinado neste script com o ângulo do espectro $-\arg(\_X_k)$ determinado através de cfft_di.exe no modo 'd', considera-se que cfft_di.exe está validado para este modo quanto ao ângulo do espectro.

A discrepância de valores em $f_k = 0.375\ \text{MHz}$ e em $f_k = 2.675\ \text{MHz}$ (setas amarelas no gráfico acima) são esperadas dados que estamos usando variáveis float para o cálculo da FFT em cfft_di.c enquanto o MathCad utiliza variáveis do tipo double.

FFT - *Fast Fourier Transform*

$\_xm := ICFFT(X)$ → Recupera o sinal sinal complexo $\_xm_k$ no domínio tempo discreto t_k através de **_xm(t)=IFFT{X(f)}**

Waveform recuperada no tempo atraves de _xm=ICFFT(X)

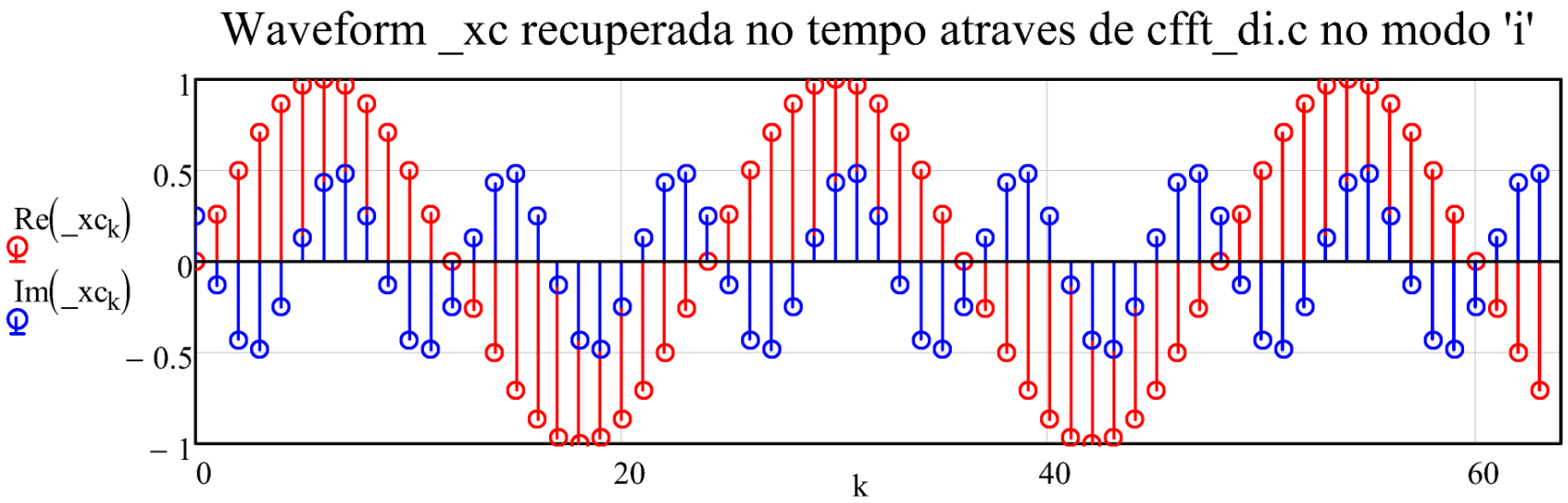


$Dif_k := x_k - \_xm_k$ $mean(Dif) = 0$ → Se resulta zero, então x_k = $\_xm_k$.

FFT - *Fast Fourier Transform*

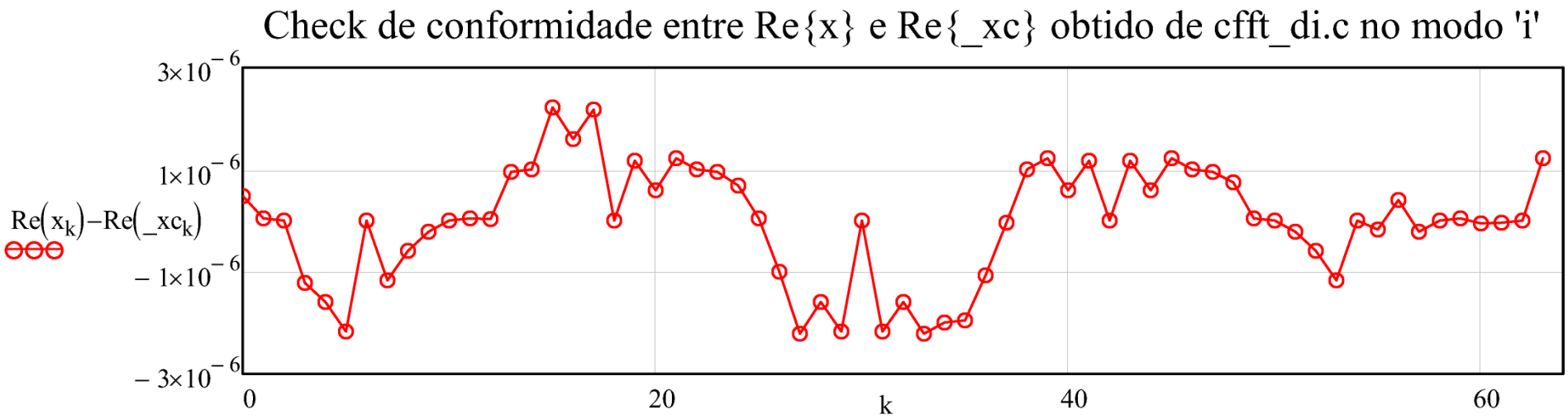
$Dat := READPRN(InpFile\_t)$ → Lê o arquivo $InpFile\_t = "\_x(t).txt"$ contendo as amostras do sinal complexo $\_xc_k$ no domínio tempo discreto t_k, gerado por **cfft_di X(f).txt _x(t).txt i.**

$\_xc := Dat^{\langle 0 \rangle} + j \cdot Dat^{\langle 1 \rangle}$ → Atribui à cada elemento do vetor $\_xc$ os valores das colunas 1 e 2 na respectiva linha da matriz Dat convertidos para valor complexo.

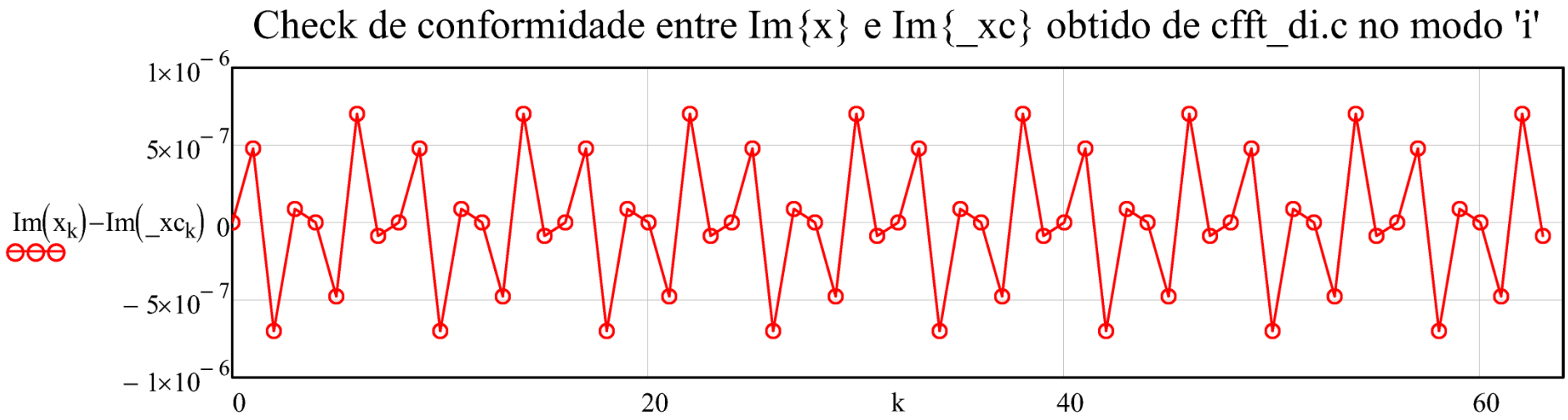


FFT - *Fast Fourier Transform*

Verificando a conformidade entre o sinal x gerado neste *script* e o sinal _xc gerado por **cfft_di.c** no modo 'i', i.e., gerado através da linha de comando **cfft_di X(f).txt _x(t).txt i**:



Observa-se no gráfico acima que há a conformidade entre as partes reais.



Observa-se no gráfico acima que há a conformidade entre as partes imaginárias.

Dado que há conformidade das partes real e imaginária entre o sinal original x_k gerado neste script e o sinal $\_xc_k$ recuperado no tempo através de cfft_di.exe no modo 'i', considera-se que cfft_di.exe está validado para este modo.