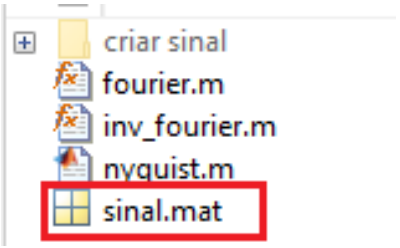
Teorema da Amostragem

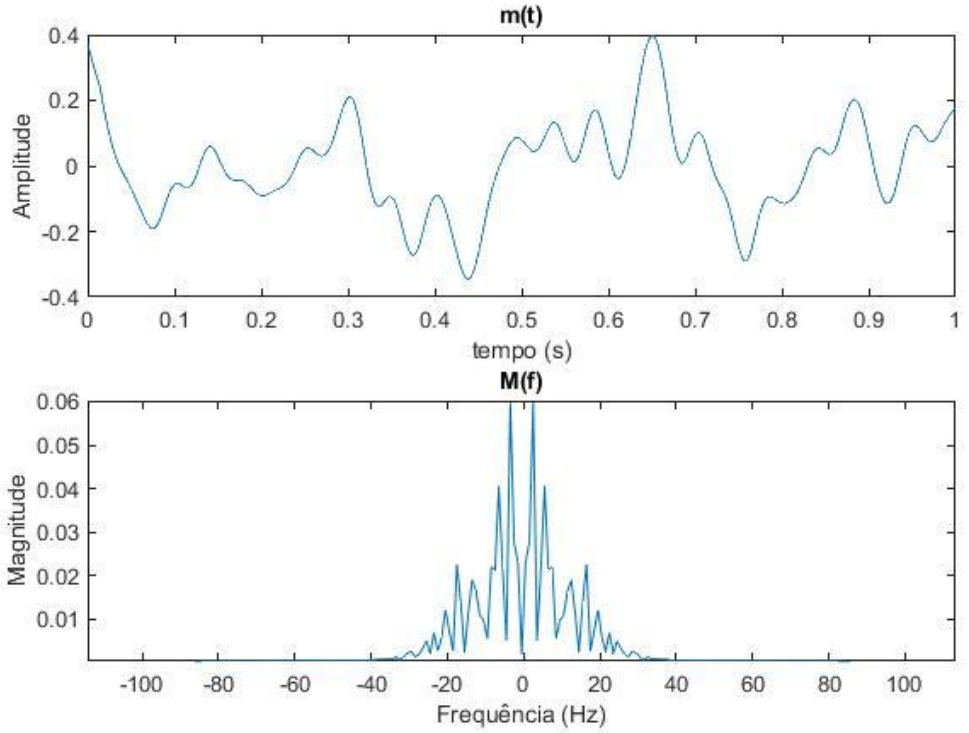
Desta vez, para fazer nosso sinal de mensagem **`m(t)`**, utilizaremos dados que já estão no arquivo **`sinal.mat`**.No Matlab é possível fazer uma simulação e salvar dados desta simulação para uso posterior. (Explicado no tópico extra).



Para importar esse sinal, basta utilizar o comando load (tendo o arquivo **`sinal.mat`** no mesmo diretório da simulação):

```
% carregar as variáveis s, t e fs
% s é o sinal de mensagem
% t é o vetor de tempo
% fs é a frequência de amostragem em que ele foi simulado
load sinal.mat
```

A seguir está um plot do sinal e seu espectro:

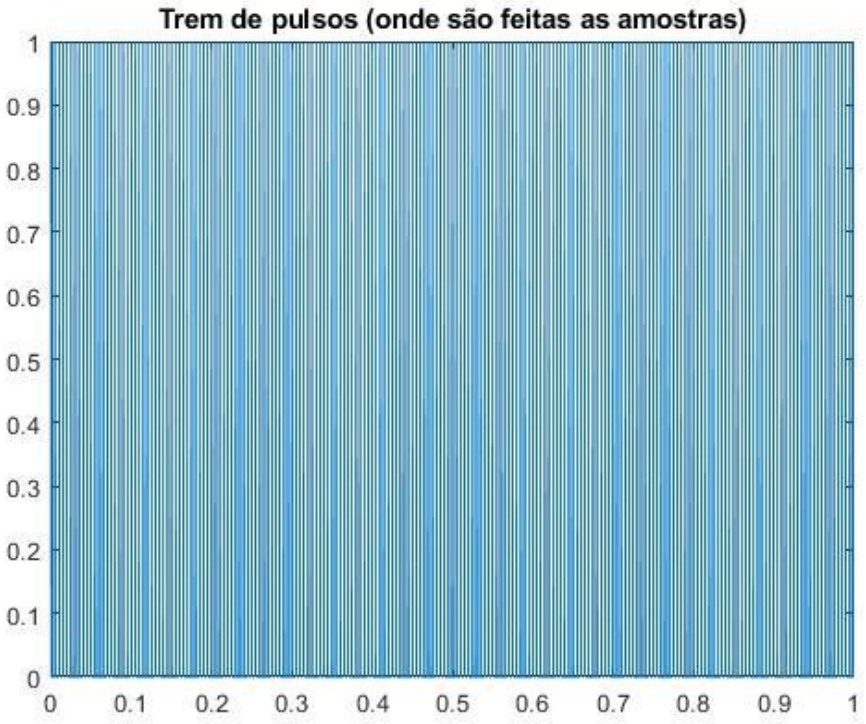


Observa-se pelo espectro que o sinal possui uma frequência máxima em torno de 50Hz (é bom dar uma folga, você vai ver porque).

Vamos agora criar uma versão amostrada do sinal e para isso, simplesmente faremos a multiplicação do sinal por um trem de impulsos. Teoricamente, deveríamos utilizar deltas de Dirac, que têm valor máximo infinito e área 1. Mas para a nossa simulação, podemos simplesmente utilizar impulsos com altura de 1. O nosso vetor trem de impulsos se parecerá com algo como:

```
[0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0]
```

É importante que os impulsos sejam periódicos, já que vamos simular uma **taxa de amostragem**.
O gráfico disso no Matlab se parece com:



Para criar esse trem de impulsos, utilizaremos a função pulstrain do pacote Signal Processing Toolbox:

```
% Frequência de amostragem do amostrador
FS = 200;

% Criação do trem de impulsos
d = 0 : 1/FS : 1;
y = pulstran(t,d,@rectpuls,1e-5);
```

Veja mais sobre esse comando na documentação:
https://www.mathworks.com/help/signal/ref/pulstran.html

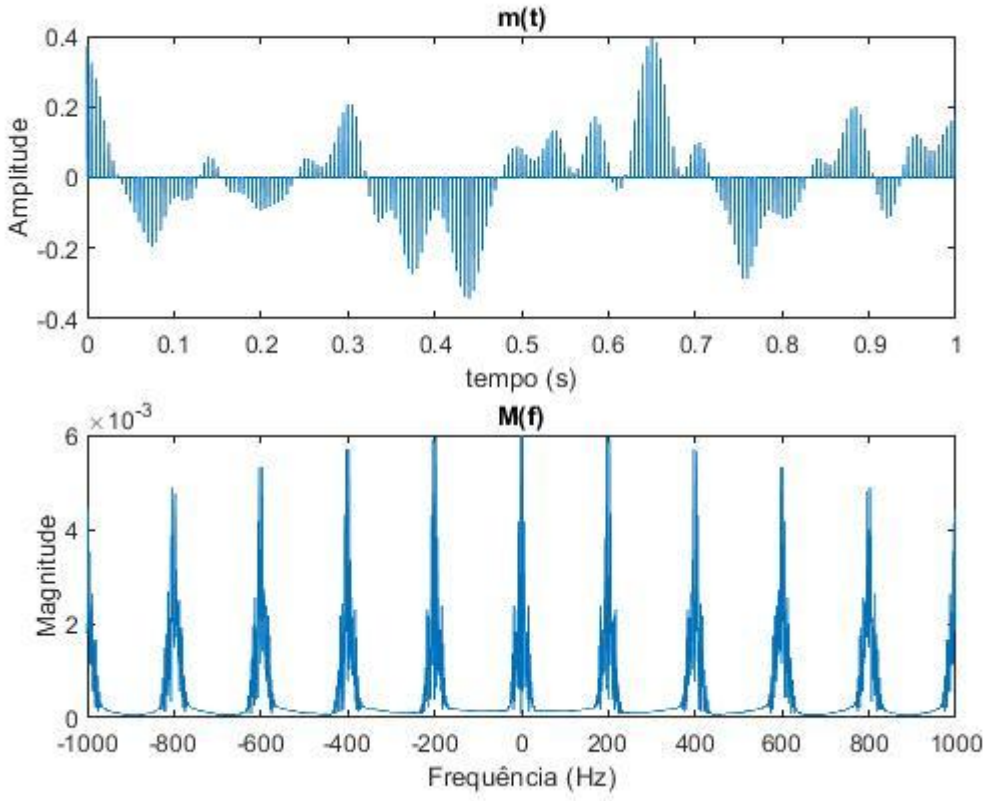
Em seguida, fazemos a multiplicação do nosso sinal de mensagem pelo trem de impulsos para mostrá-lo. Do ponto de vista dos vetores, estamos fazendo algo parecido com isso:

```
[2 3 4 5 5 6 7 8 8 7 6 5 5 4 3 2 1 1 2 3 4 5 6 7 8 8 8 7 6 6 5 4
3]
                                  X
[0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0
0]
                                  =
[0 0 0 0 0 6 0 0 0 0 0 5 0 0 0 0 0 1 0 0 0 0 0 7 0 0 0 0 0 6 0 0
0]
```

Ou seja, pegamos **amostras** do sinal e **zeramos** o sinal fora dessas amostras.

Lembre-se! Fazemos a multiplicação elemento-a-elemento no matlab utilizando o operador “ponto-asterisco”: **.***

A seguir, temos o resultado dessa amostragem no tempo (note os impulsos traçando o contorno do sinal) e na frequência.



Observe o fenômeno interessante na frequência de que surgem cópias periódicas do espectro do sinal como resultado da multiplicação dele pelo trem de impulsos.
Isso é resultado transformada de Fourier de um trem de impulsos:

$$\sum_{n=-\infty}^{\infty} g(nT_s)\delta(t - nT_s)$$

$$\Updownarrow$$

$$\sum_{n=-\infty}^{\infty} g(nT_s)e^{-j2\pi nT_s f} = f_s \sum_{m=-\infty}^{\infty} G(f - mf_s)$$

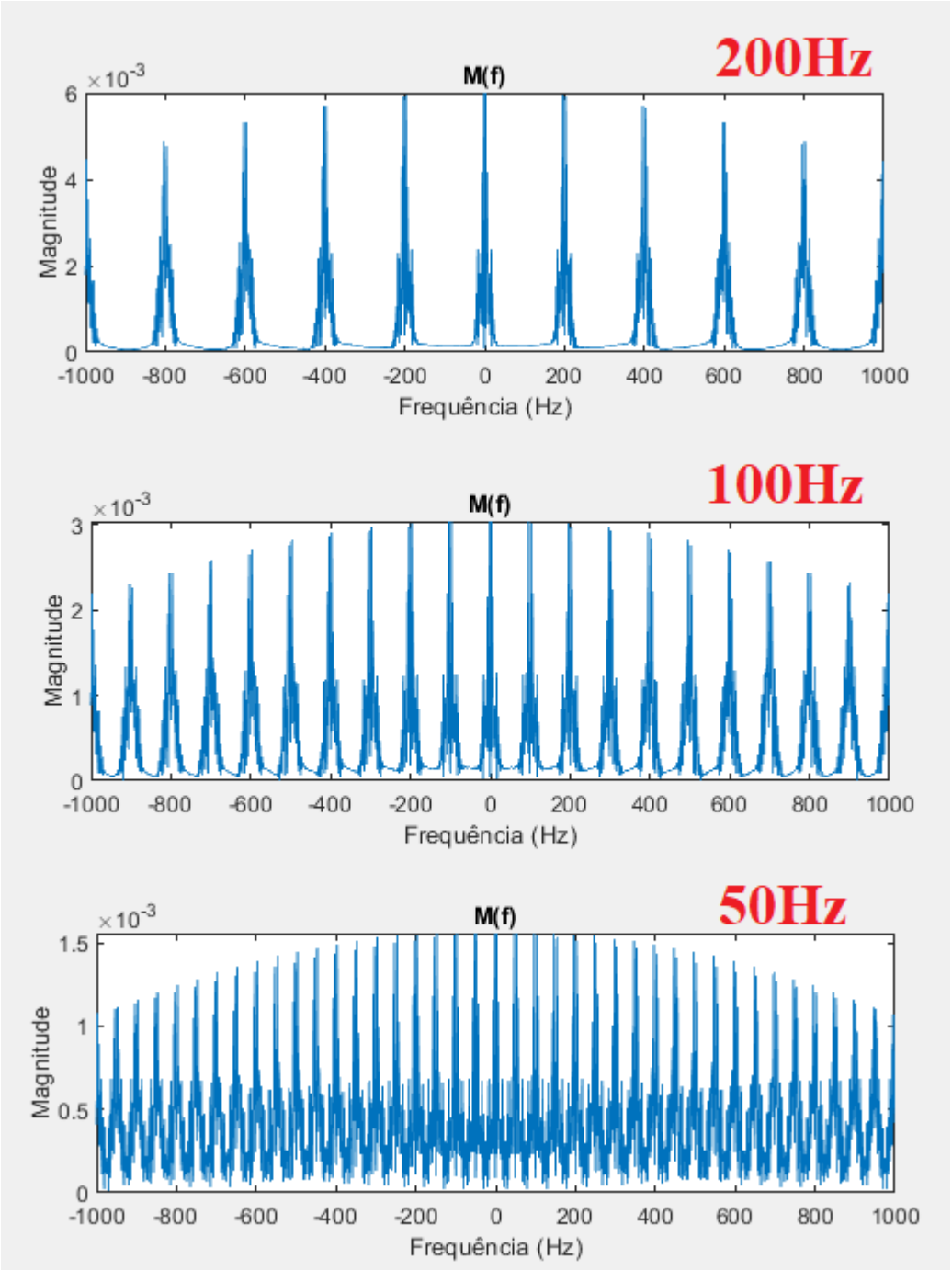
Mas o importante a se lembrar é que:

> Quando fazemos a amostragem, ou seja, multiplicamos um sinal por um trem de impulsos com **taxa de amostragem** f_s;
>
> O **espectro** do sinal resultante se tornará periódico, **se repetindo a cada** f_s.

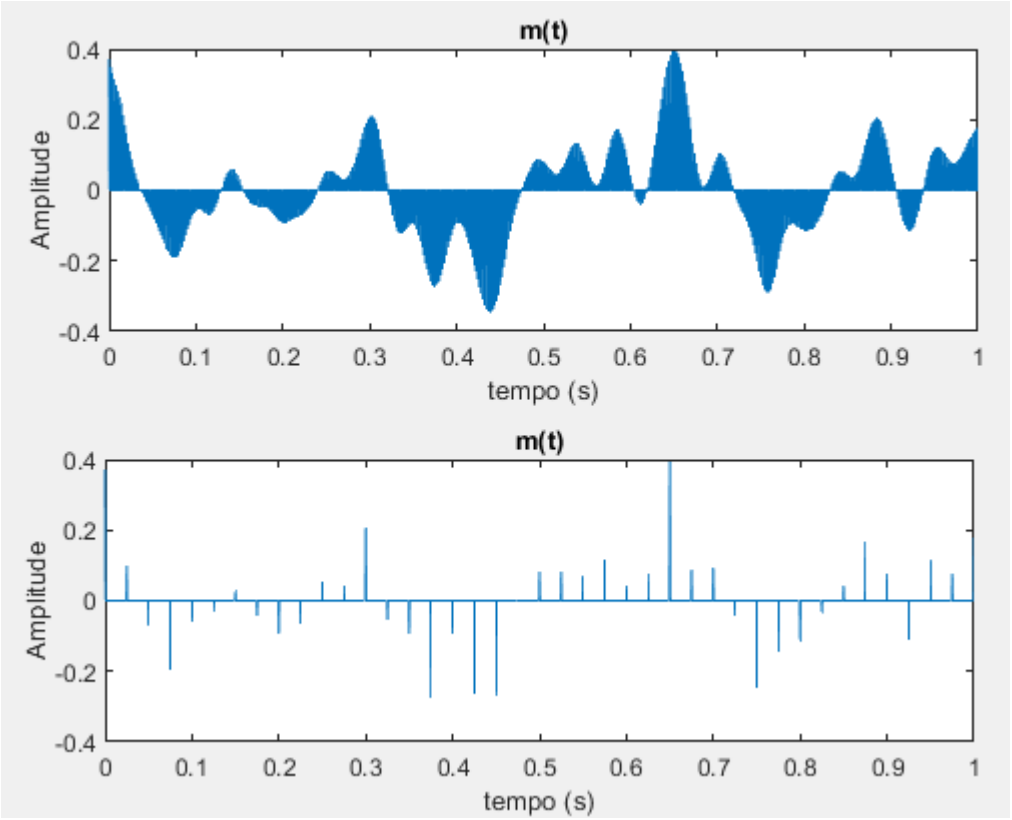
É o que podemos ver graficamente na figura anterior. O sinal foi amostrado 200 vezes por segundo, ou seja, com um trem de impulsos de 200Hz e no espectro de frequência ele passa a se repetir a cada 200Hz.

E daí surge o teorema da amostragem. Caso queiramos amostrar o sinal com taxas de amostragem cada vez menores, teremos os espectros cada vez mais perto até o momento em que eles se sobrepõem, como pode ser visto alterando a variável FS na simulação:

```
27      %% Criando o trem de impulsos para samplear o sinal
28
29      % Frequência de amostragem do amostrador
30 -    FS = 200;
31
32      % Criação do trem de impulsos
33 -    d = 0 : 1/FS : 1;
34 -    y = pulstran(t,d,@rectpuls,1e-5);
35
```



Essa mistura no espectro causa **perda de informação**. No domínio do tempo, podemos ver que se torna mais difícil recuperar o sinal original. Veja a comparação entre o mesmo sinal amostrado a 500Hz e a 40 Hz:



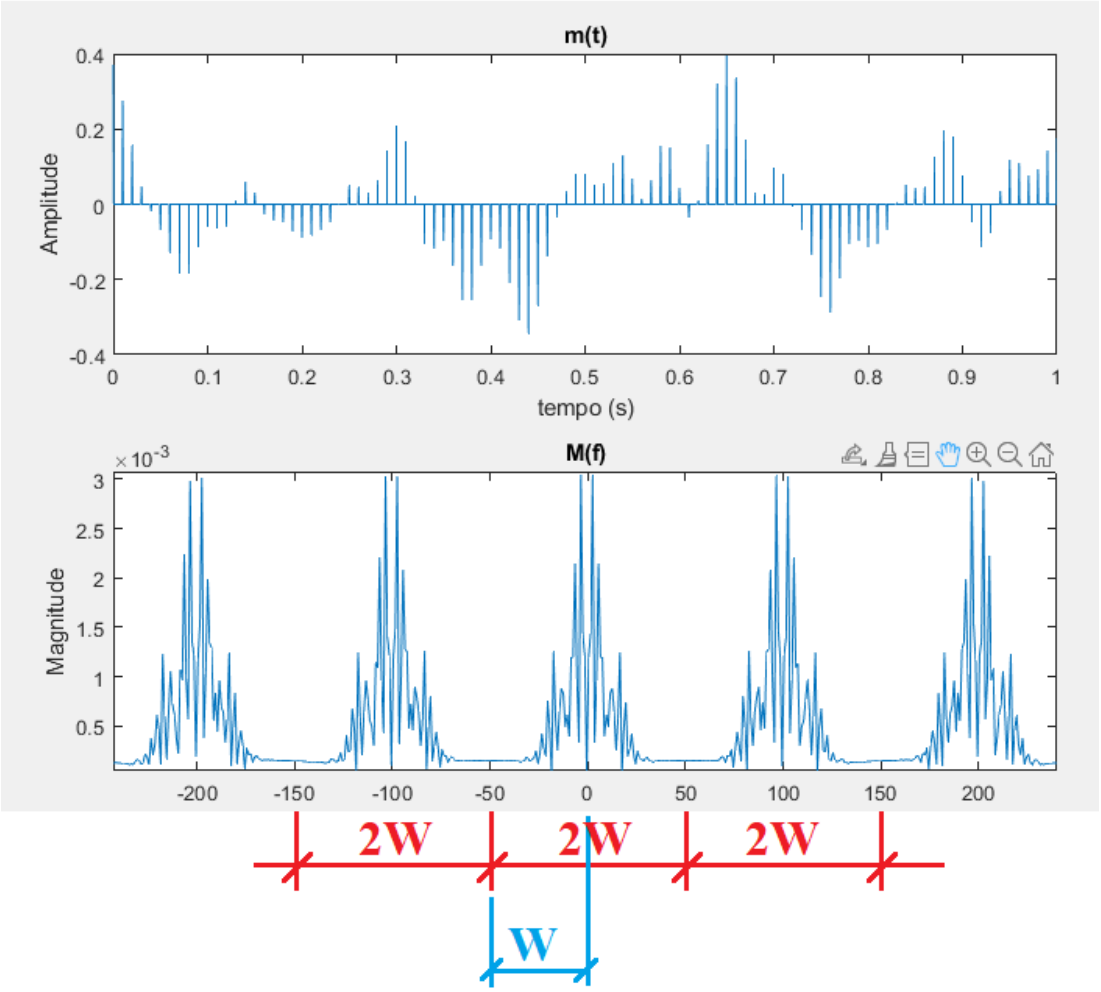
Veja que se tentarmos recuperar o sinal original somente olhando a segunda imagem, já não temos certeza sobre o formato dele e é impossível pegar alguns detalhes.

Como o sinal se repete a cada f_s (frequência de amostragem), a frequência de amostragem mínima para que não haja perda de informação depende da largura de banda W do sinal:

$$f_{s\,MÍNIMA} = 2\,W$$

Este é o teorema da amostragem, ou teorema de Nyquist!

É claro que existem demonstrações matemáticas mais formais, mas podemos ver o teorema graficamente.
Como vimos no início, o nosso sinal tem uma frequência máxima de aproximadamente 50Hz, ou seja, sua largura de banda que vai de 0Hz a 50Hz, ou seja, W = 50Hz.
A frequência de amostragem mínima é então 100Hz. Veja o que acontece quando amostramos com 100 Hz:



Observe como estamos (com alguma folga) no limite para que os espectros não comecem a se sobrepor e comecemos a perder informação. Caso a frequência de amostragem diminua, **geometricamente** os espectros começarão a se sobrepor.

Apesar de parecer que podemos “apertar” um pouco mais o sinal no espectro, realmente já estamos no limite. Tente utilizar taxas de amostragem como 90Hz. Há também o efeito da amostragem utilizada para fazer a própria simulação (passos no vetor do tempo). De qualquer forma, na prática, geralmente utilizamos uma frequência de amostragem bem maior do que a mínima para ter sistemas confiáveis.

Extra: gerando o sinal de mensagem

Dessa vez, em vez de uma simples senoide, utilizamos como sinal de mensagem um sinal mais parecido com o que existe em situações práticas. Esse sinal poderia ter sido utilizado nas outras simulações e pode ser um exercício interessante fazer isso caso você queira treinar um pouco de Matlab. Este texto não faz parte da ementa de princípios de comunicação e é apenas uma explicação sobre uma forma que eu encontrei (com certeza existem formas melhores) para gerar um sinal que eu considero satisfatório como sinal de mensagem. Ainda assim, acho uma discussão interessante e serve como experiência.

Para simular o sinal de mensagem, foi utilizada a função **`rand`**, que gera números aleatórios entre 0 e 1. Podemos também gerar uma matriz de números aleatórios passando o número de linhas e de colunas para a função, por exemplo:

```
rand(5,5)
```

Gera algo parecido com:

```
0.8147    0.0975    0.1576    0.1419    0.6557
0.9058    0.2785    0.9706    0.4218    0.0357
0.1270    0.5469    0.9572    0.9157    0.8491
0.9134    0.9575    0.4854    0.7922    0.9340
0.6324    0.9649    0.8003    0.9595    0.6787
```

Para gerar nosso sinal de mensagem, começamos com uma sequência de números aleatórios do tamanho do vetor de tempo:

```
rand(1,length(t))
```

Esse sinal, porém, estará sempre entre 0 e 1. Surgiu então a ideia de multiplicá-lo por uma amplitude A e deslocá-lo para cruzar o 0. Temos então:

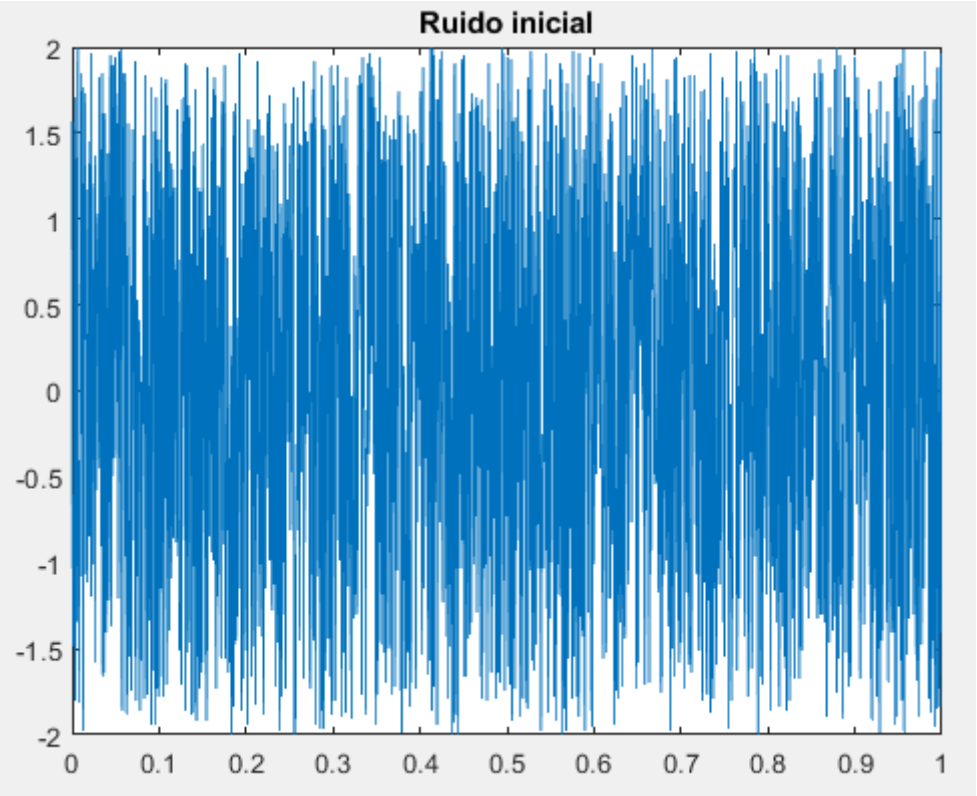
```
% Frequência de amostragem
fs = 2000;

% Vetor de tempo
t = (0:1/fs:1);

% Máximo e mínimo
smax =  2;
smin = -2;

% Ruído inicial]
A = (smax-smin);
m = smin + A*rand(1,length(t));
```

O sinal agora se parece com isso:

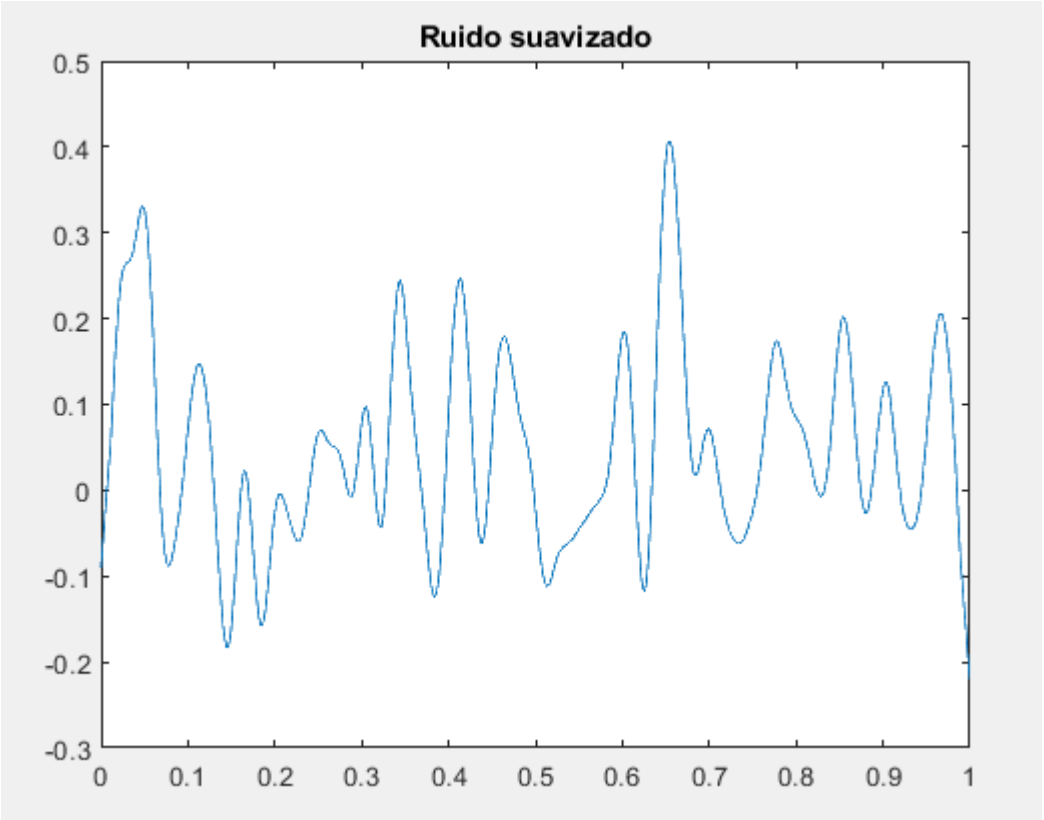


Um pouco “aleatório” de mais. Na verdade, um valor do sinal é totalmente descorrelacionado com seus vizinhos. Acabamos de criar um **ruído branco**, muito estudado em livros de sinais e estatística e também na disciplina de sinais aleatórios da UFABC.

Mas na nossa simulação queremos um sinal de mensagem mais “suave”. Para isso, encontrei em fóruns e na documentação a função **`smoothdata`**, uma função que faz exatamente isso com o sinal. Após aplicar algumas vezes a função no sinal:

```
% Suavizando o sinal de ruído
% (Fui aplicando e visualizando até achar que estava bom)
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
m = smoothdata(m);
```

Temos algo parecido com isso:



Finalmente, vamos algumas variáveis para poder utilizar em outros scripts com o comando **`load`**. Vamos salvar as seguintes variáveis:

- Sinal de mensagem - m - o principal!
- Frequência de amostragem - fs - para compatibilidade na hora de gerar outros sinais em outros arquivos
- Vetor de tempo - t - também por compatibilidade.

```
save('sinal.mat', 'm', 'fs', 't');
```

É gerado então um arquivo .mat, que contém os dados salvos: