UPF / Lab. De Processamento Digital de Sinais Prof. Fernando Passold

Lab 1: Introdução à Sinais.

Exercícios:

1. Defina o que é “ruído (Gaussiano) branco”?
2. Defina o que é “ruído rosa”?
3. Defina o que é “ruído marrom”?

Obs.: Para fins mais práticos, as definições acima são esperadas para um sinal na faixa de adio (20Hz à 20KHz). Espera-se um texto, seguido de um diagrama espectral do sinal (diagrama de Bode/Magnitude), além de definições matemáticas associadas para cada sinal envolvendo valor médio do sinal, desvio padrão. Em seguida, use alguma ferramenta matemática capaz de simular estes 3 tipos de ruído. Apresente o código e o resultado (resposta temporal do sinal).

Dicas:

1. No caso do **Matlab**, a função ‘wgn’ permite gerar um ruído branco Gaussiano, onde se especifica o tamanho do vetor numérico à ser gerado e potência do ruído em dBW. Lembrado que 1 dBW = 10 $\log_{10}$(P). Obs.: a função ‘wgn’ do Matlab, requer o “**Communications Toolbox**” — buscar por: https://www.mathworks.com/help/comm/ref/wgn.html#bu204xm
2. No caso do **Octave**, a função ‘awgn’ — buscar por: https://octave.sourceforge.io/communications/function/awgn.html
3. No caso do **Octave**, a função ‘sgram’ permite gerar os 3 tipos de ruído. Por exemplo: ‘sgram(noise(5000,’white’),’dynarange’,70)’ ou ‘sgram(noise(5000,’pink'),'dynrange',70)’ ou ‘sgram(noise(5000,’brown’),'dynrange',70)’. — buscar por: https://octave.sourceforge.io/ltfat/function/noise.html e recomenda-se também: http://ltfat.org/doc/signals/noise.html

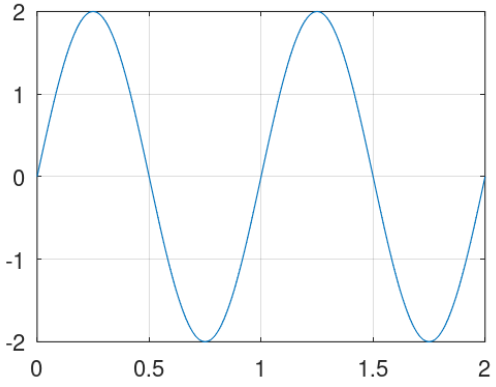
Primeiras simulações:

No **Octave/Matlab**:

1. Primeiro é necessário criar um vetor contendo o sinal sobre o qual será sobreposto o ruído:

```
>> t=0:1/200:2; % cria vetor tempo, período de 0 à 2 segundos; 200 amostras/segundo
>> y=2*sin(2*pi*1*t); % cria onda senoidal de 1 Hz usando vetor t anterior, amplitude de 2 Vpp.
>> plot(t,y)
```

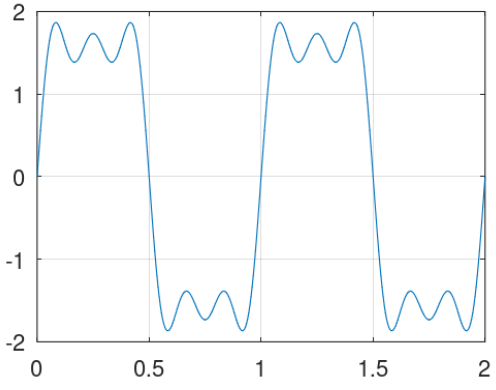
Estes comandos devem gerar a seguinte saída gráfica (figura ao lado).



2. Supondo que se queira sintetizar um sinal um pouco mais “rico” em harmônicas, lembrando da transformada de Fourier de uma Onda Quadrada, podemos fazer:

```
>> y2=2*sin(2*pi*1*t)+2/3*sin(6*pi*t)
+2/5*sin(10*pi*t);
>> figure;
>> plot(t,y2)
```

Desta vez, a seguinte saída deve ter sido gerada (figura inferior, ao lado).



Ruído Branco:

O ruído branco é um conceito que se origina de processamento e estatísticas de sinais. Refere-se a um tipo de sinal aleatório que contém energia igual em todas as frequências. Em termos simples, é um sinal aleatório que soa como estática ou aquele ruído que se pode ouvir em um rádio quando nenhuma estação está sintonizada. O nome "ruído branco" vem da analogia com a luz branca, que contém todas as cores visíveis em igual intensidade.

Espectro e Matemática:

Para entender melhor o ruído branco, vamos mergulhar em seu espectro e características matemáticas:

1. **Espectro**: Se você fosse analisar o ruído branco no domínio de frequência, seu espectro seria plano. Isso significa que tem potência igual em todas as frequências. Em termos matemáticos, a densidade espectral de energia (PSD) do ruído branco é constante. É por isso que é chamado de ruído "branco" - apenas como a luz branca contém todas as cores, o ruído branco contém todas as frequências.

2. **Matemática**: O ruído branco pode ser representado matematicamente como uma sequência de valores aleatórios, tipicamente desenhado de uma distribuição gaussiana com uma média de 0 e um certo desvio padrão (variância). Cada valor na sequência é independente dos outros. A fórmula matemática para o ruído branco é geralmente denotada como n(t), onde t representa o tempo.

De forma discreta, você pode representá-lo como:
$n[n] = \eta[n]$,
Onde $\eta[n]$ é uma sequência de variáveis aleatórias independentes desenhadas de uma distribuição Gaussiana.

Códigos Exemplo:

Em **Python**:
Aqui está um exemplo simples de código Python usando a biblioteca NumPy para gerar e visualizar ruído branco e a biblioteca MatPlotLib para gerar gráficos ilustrando o resultado:

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
duration = 1.0  # Duration in seconds
sampling_rate = 44100  # Number of samples per second
amplitude = 0.5  # Adjust to control the intensity

# Generate white noise
num_samples = int(sampling_rate * duration)
white_noise = amplitude * np.random.randn(num_samples)

# Time values
time = np.arange(0, duration, 1/sampling_rate)
```

```
# Plot the white noise waveform
plt.plot(time, white_noise)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('White Noise')
plt.grid()
plt.show()
```

Neste exemplo, a função `np.random.randn()` gera valores aleatórios usando distribuição Gaussiana (isto é, média nula e desvio padrão igual a 1). Podemos ajustar a amplitude do sinal gerado variando o desvio.

Código exemplo similar usando **Octave**:

```
% noise_white1.m
% Parameters
duration = 1.0;         % Duration in seconds
sampling_rate = 44100;  % Number of samples per second
amplitude = 0.5;        % Adjust to control the intensity

% User-defined standard deviation
std_deviation = input('Enter the standard deviation: ');

% Generate white noise
num_samples = int32(sampling_rate * duration);
white_noise = amplitude * randn(1, num_samples) * std_deviation;

% Time values
time = 0 : 1/sampling_rate : duration - 1/sampling_rate;

% Plot the white noise waveform
plot(time, white_noise);
xlabel('Time (s)');
ylabel('Amplitude');
title('White Noise');
grid on;
```

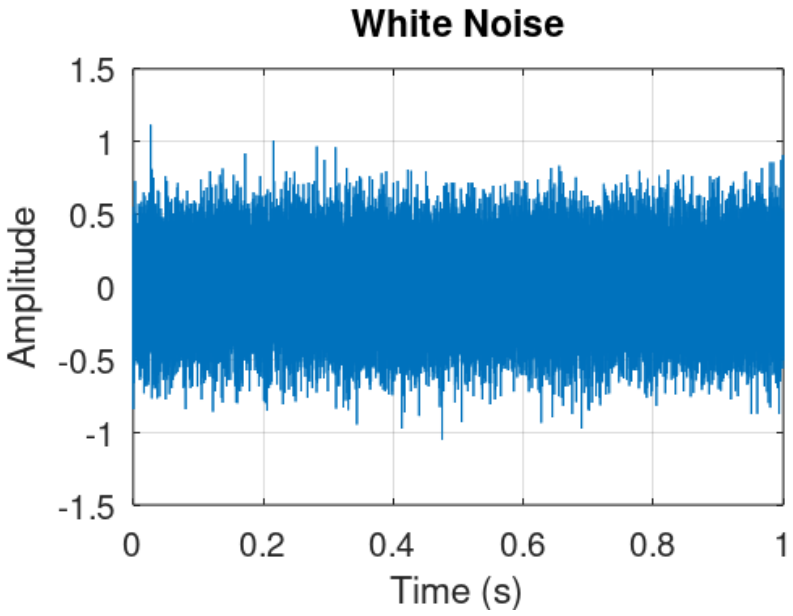
Salve este código num arquivo texto .m e execute-o usando o Octave. Este código exige que se informe o desvio padrão antes de tentar gerar uma sequencia de pontos e depois gera um gráfico do ruído gerado.

Este código quando executado rende:

```
>> noise_white1
Enter the standard deviation: 0.5
>> mean(white_noise) % valor médio
ans = -9.9206e-04
>> max(white_noise) % valor máximo gerado
ans = 1.1119
>> min(white_noise) % menor valor gerado
ans = -1.0479
```

>>

Com o seguinte gráfico:

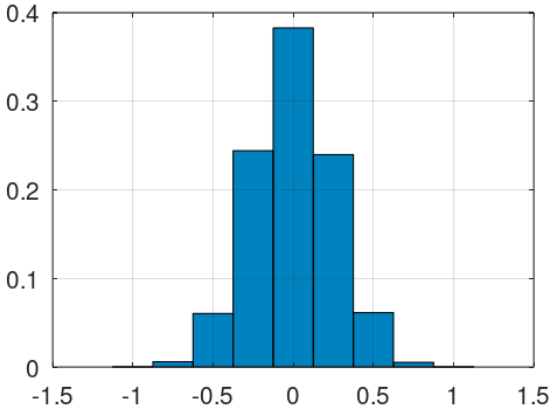


Se quisermos verificar a distribuição dos valores aleatórios gerados, podemos usar a função 'hist()` do Octave para gerar um histograma dos valores gerados, por exemplo:

```
>> x=-1:0.25:1
x =
   -1.0000   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500    1.0000

>> figure; hist(white_noise,x,1)
>> colormap(winter) % especifica uma paleta de cores para o gráfico, ver:
>> % https://octave.sourceforge.io/octave/function/colormap.html
```

Obs.: o vetor `x` gerado no código anterior, corresponde ao segundo argumento que indica que valores centrais devem ser adotadas para as barras do histograma. E o terceiro argumento, *norm*, possibilita "normalizar" o gráfico gerado, isto é, o histograma é normalizado de forma que a soma das barras seja igual à *norm*.

O que gera o gráfico:



Questão: Varie o desvio padrão para se certificar que o mesmo modifica a intensidade (amplitudes) do sinal gerado.

Lembre-se de que o ruído branco é apenas um tipo de sinal aleatório. Existem outros tipos de ruído colorido, como **ruído rosa** (onde a potência diminui pela metade do dobro da frequência) e **ruído marrom** (onde a potência diminui por 3 dB por oitava). Cada um desses tipos tem suas próprias características únicas em termos de conteúdo espectral e propriedades matemáticas.

Distribuição Gaussiana

Uma distribuição Gaussiana, também conhecida como distribuição normal, é uma distribuição de probabilidade simétrica caracterizada por sua curva em forma de sino. É comumente usado para modelar uma ampla variedade de fenômenos naturais em áreas como estatística, física e ciências sociais. A distribuição gaussiana é definida por dois parâmetros: a média (μ) e o desvio padrão (σ).

Matematicamente:

A equação abaixo modela a distribuição de probabilidade da função gaussiana:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Onde:

- x = variável de interesse (pode ser sintetizada ou observada);
- μ = média da distribuição, que representa o valor central da distribuição;
- σ = desvio padrão, que mede a dispersão da coleção de pontos;
- $\exp = e$ = base do logaritmo natural;
- π = constante matemática (aproximadamente 3.14159).

O formato da curva de distribuição gaussiana é determinada pelos valores de μ e σ. Como estes valores afetam a curva de distribuição:

- **Média**, μ, representa o valor central da distribuição. Determina onde está localizado o pico da curva. Deslocar a média para a esquerda ou para a direita fará com que toda a curva se mova ao longo do eixo x sem alterar sua forma. Matematicamente, modificar μ acaba deslocando toda a função para a esquerda ou para a direita, mas a largura da curva permanece a mesma.
- **Desvio padrão**, σ: controla a dispersão dos pontos de dados. Um desvio padrão menor resulta em uma curva mais estreita, enquanto um desvio padrão maior leva a uma curva mais larga. A dispersão da curva é tal que aproximadamente 68% dos dados se enquadram na faixa dentro de $\mu \pm \sigma$; 95% se enquadra dentro da faixa de $\mu \pm 2\sigma$, e 99,7% dentro da faixa de $\mu \pm 3\sigma$.

Códigos exemplo para exemplificar os efeitos da média (μ) e do desvio padrão (σ):

Código 1: variando a média mas mantendo o desvio padrão constante:

Em **Python**:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(-10, 10, 1000)
mu_values = [0, 2, -3]
sigma = 1

for mu in mu_values:
    y = norm.pdf(x, mu, sigma)
    plt.plot(x, y, label=f"μ = {mu}")

plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Gaussian Distribution with Varying Mean')
plt.legend()
plt.show()
```

Código 2: variando o desvio padrão, mas mantendo a média constante:

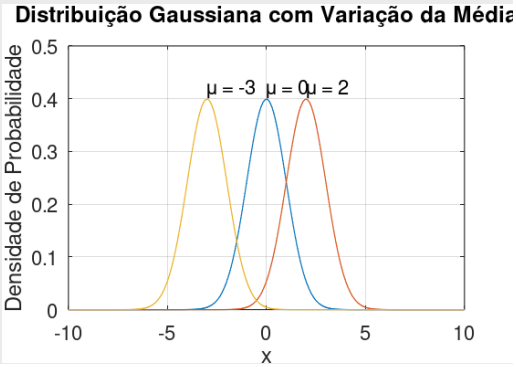
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(-10, 10, 1000)
mu = 0
sigma_values = [0.5, 1, 2]

for sigma in sigma_values:
    y = norm.pdf(x, mu, sigma)
    plt.plot(x, y, label=f"σ = {sigma}")

plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Gaussian Distribution with Varying Standard Deviation')
plt.legend()
plt.show()
```

No gráfico gerado pelo primeiro código, note como as curvas se deslocam horizontalmente enquanto mantêm a mesma forma que a média (μ) varia.
No gráfico gerado pelo segundo código, observe como as curvas se alargam ou estreitam enquanto a mesma posição do desvio padrão (σ) é mantida constante.
Isso demonstra como a média e o desvio padrão afetam a curva de sinal da distribuição Gaussiana.

Os mesmos códigos anteriores, mas para **Octave**:

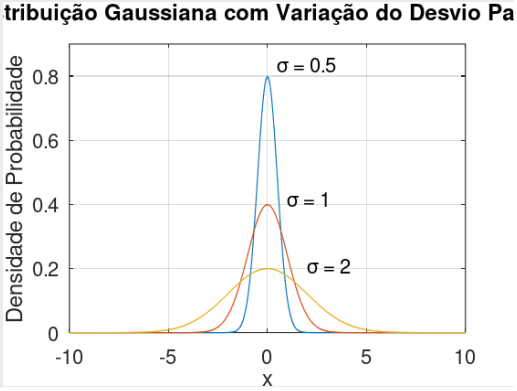
Código 1 [gaussiana1.m]: Variação da Média (μ) com Desvio Padrão Constante (σ):

```
% gaussiana1.m
pkg load statistics

x = linspace(-10, 10, 1000);
mu_values = [0, 2, -3];
sigma = 1;

cont=0;
for mu = mu_values
    y = normpdf(x, mu, sigma);
    plot(x, y) % , "labels", sprintf("μ = %d", mu));
    % acrescenta texto no topo da curva com informação do mu:
    [maior,posicao]=max(y);
    texto=["μ = " num2str(mu)];
    text(mu,maior*1.05,texto);
    cont=cont+1;
    legenda(cont,1:length(texto))=texto;
    hold on;
end

xlabel('x');
ylabel('Densidade de Probabilidade');
title('Distribuição Gaussiana com Variação da Média');
% legend(legenda);
axis([-10 10 0 0.5])
hold off;
```



Código 2 [gaussiana2.m]: Variação do Desvio Padrão (σ) com Média Constante (μ)

```
% gaussiana2.m
pkg load statistics

x = linspace(-10, 10, 1000);
mu = 0;
sigma_values = [0.5, 1, 2];

for sigma = sigma_values
    y = normpdf(x, mu, sigma);
    plot(x, y) %, "label", sprintf("σ = %0.1f", sigma));
    % acrescenta texto no topo da curva com
informação do mu:
    [maior,posicao]=max(y);
    texto=["σ = " num2str(sigma)];
    text(sigma,maior*1.05,texto);
    cont=cont+1;
    legenda(cont,1:length(texto))=texto;
    hold on;
end

xlabel('x');
```



```
ylabel('Densidade de Probabilidade');
title('Distribuição Gaussiana com Variação do Desvio Padrão');
% legend();
axis([-10 10 0 0.9])
hold off;
```

Obs.: No caso do Octave: certifique-se de ter o pacote de estatísticas (`statistics`) instalado no Octave para usar a função `normpdf` para calcular a função de densidade de probabilidade da distribuição normal. Você pode instalar o pacote com o comando `pkg install -forge statistics`. Lembre-se de salvar esses códigos em arquivos separados e executá-los no ambiente do Octave para visualizar os gráficos.

Média e Desvio Padrão

- **Média** (*Mean*): é uma medida de tendência central que representa o valor médio de um conjunto de dados. É calculada somando todos os valores do conjunto e dividindo pelo número total de valores. A fórmula para a média (μ) de um conjunto de n valores é dado por:

  $$\mu = \frac{1}{b}\sum_{i=1}^{n} x_i$$

  Onde: n = quantidade de valores dentro do conjunto x de dados.
- **Desvio Padrão** (*Standard Deviation*): é uma medida de dispersão que indica o quanto os valores de um conjunto estão afastados da média. Ele fornece uma estimativa da variação ou *spread* dos dados. Um desvio padrão maior indica maior dispersão dos valores em relação à média, enquanto um desvio padrão menor indica menor dispersão. O desvio padrão é calculado usando a seguinte fórmula:

  $$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2}$$
- **Variância** (*Variance*): é uma medida relacionada ao desvio padrão e indica o quão dispersos os valores estão em relação à média. Ela é a média dos quadrados das diferenças entre cada valor e a média. A variância (σ^2) é calculada pela equação:

  $$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2$$

  ou seja, A raiz quadrada da variância é o desvio padrão (σ).

Derivada da Função Gaussiana e Valor Máximo

A função Gaussiana é caracterizada pela expressão:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Sua derivada primeira rende:

$$f'(x) = -\frac{x-\mu}{\sigma^3\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

O valor máximo da função Gaussiana, $f(x)$, ocorre no ponto em que sua derivada primeira se anula. Fazendo $f'(x) = 0$ e resolvendo em relação à x, obtemos:

$$-\frac{x-\mu}{\sigma^3\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} = 0$$

Notamos que $f'(x) = 0$ quando $x = \mu$, ou seja, o valor máximo da função Gaussiana ocorre no ponto $x = \mu$ que corresponde ao seu valor médio. Então, seu valor máximo corresponde à:

$$f(\mu) = \frac{1}{\sigma\sqrt{2\pi}}$$

Note, depois de executada a rotina `noise_white1.m` poderíamos comprovar a distribuição dos valores gerados acrescentando os seguintes comandos (no Octave):

```
>> mean(white_noise) % comprovando valor médio gerado
ans = -9.9206e-04 % praticamente zero (nulo)
>> std(white_noise)
ans = 0.2494  % σ = 1/4 = 0,25
>> 1/(0.25*sqrt(2*pi)) % Valor máximo esperado para a distribuição gaussiana
ans = 1.5958
```

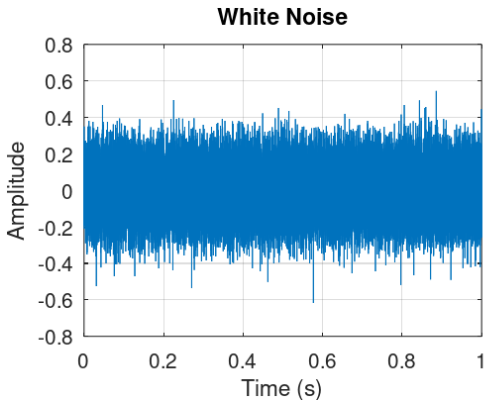
Poderíamos sobrepor a curva de distribuição da função Gaussiana sobre o gráfico do histograma gerado anteriormente:

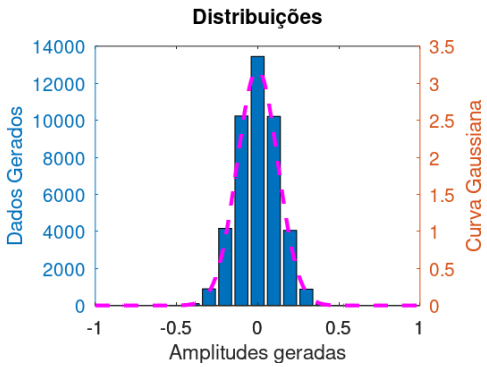
```
% white_noise2.m
% Parameters
duration = 1.0;         % Duration in seconds
sampling_rate = 44100;  % Number of samples per second
amplitude = 0.5;        % Adjust to control the intensity

% User-defined standard deviation
% std_deviation = input('Enter the standard deviation: ');
std_deviation = 0.25;

% Generate white noise
num_samples = int32(sampling_rate * duration);
white_noise = amplitude * randn(1, num_samples) * std_deviation;

% Time values
time = 0 : 1/sampling_rate : duration - 1/sampling_rate;
```

```
% Plot the white noise waveform
plot(time, white_noise);
xlabel('Time (s)');
ylabel('Amplitude');
title('White Noise');
grid on;

% Gerando algumas estatísticas:
media = mean(white_noise)
sigma = std(white_noise)
x=-1:0.1:1; % valores centrais das barras do histograma, variação de 1/4
[y_bars,x_bars] = hist(white_noise,x); % cria o histograma

% gera curva de distribuição gaussiana com base nos dados gerados
xx = -1:1/100:1; % eixo X da curva gaussiana
yy = normpdf(xx, media, sigma);

figure;
[hax, h1, h2] = plotyy (x_bars,y_bars, xx,yy, @bar, @plot);
xlabel ("Amplitudes geradas");
ylabel (hax(1), "Dados Gerados");
ylabel (hax(2), "Curva Gaussiana");
title("Distribuições");

% Modificando corres dos gráficos
lcolor = get (gca, "ColorOrder")(1,:);
rcolor = get (gca, "ColorOrder")(2,:);
set ([h2], "linestyle", "--");
set ([h2], 'color','m');
set ([h2], "linewidth", 2);
```

Esta rotina gera resultados como:

```
>> white_noise2

media = -8.6031e-04
sigma = 0.1252
>>
```

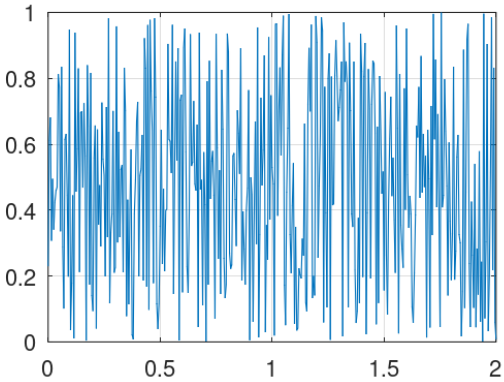




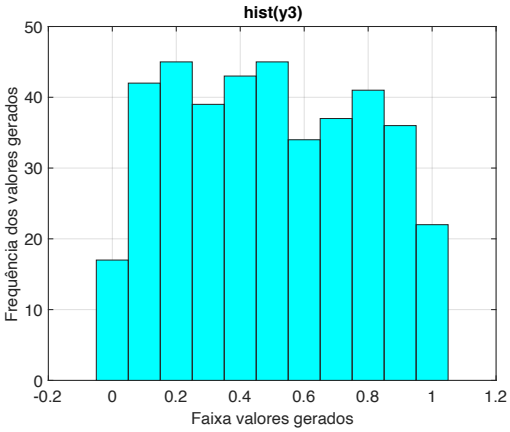
Outros exemplos de código

Existe a função `rand()` no Octave, Matlab e outras linguagens de programação; um gerador de números pseudo-aleatórios. Note um detalhe:

```
>> t=0:1/200:2; % cria vetor tempo, período
de 0 à 2 segundos; 200 amostras/segundo
>> L=length(t)
L = 401
>> y3=rand(L,1);
>> figure; plot(t,y3)
>> max(y3)
ans = 0.9990
>> min(y3)
ans = 2.3084e-03
>> mean(y3) % calcula valor médio
ans = 0.4802
>> std(y3) % calcula desvio padrão
ans = 0.2923
```

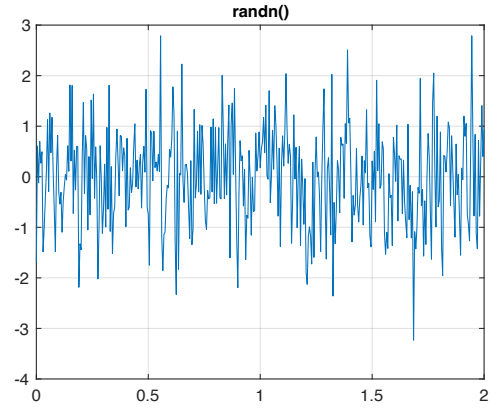


Note que o valor médio está próximo do esperado, aproximadamente 0,5; já que a função rand() da forma como foi utilizada antes, gera números no intervalo: (0: 1), excluindo os valores 0 e 1. Mas isto não é tudo, note que esta função não é um bom gerador de números aleatórios. Um histograma sobre os valores gerados revela sua distribuição de valores:

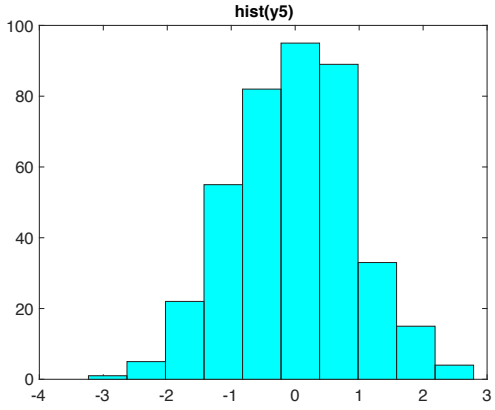



```
>> x=0:0.1:1; % centros das barras para
histograma
>> figure; hist(y3,x);
```

Note a diferença para a função `randn()` no MATLAB:




```
>> y5=randn(L,1);
>> plot(t,y5)
>> max(y5)
ans =
       2.7891
>> min(y5)
ans =
       -3.232
>> mean(y5)
ans =
    -0.022846
>> std(y5)
ans =
      0.95313
>> figure; hist(y5)
```



Note, se tivesse sido usada a função `rand()` da linguagem C obteríamos o seguinte resultado:

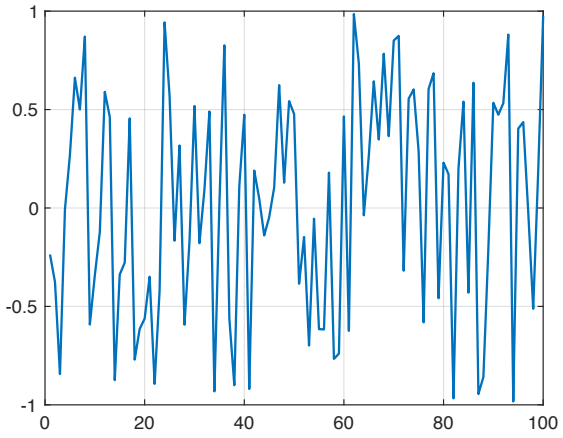
```
// numeros.c
// Fernando Passold, em 25/08/2023
// Gera numeros aleatÃ³rios (-1, +1)
// Linguagem GNU C
// compilar com: % gcc numeros.c -o numeros

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

int main(){
  int i,n;
  double random_value, maior, menor, sum, variance, mean, std;

  srand( time(NULL) ); // inicializa semente de numeros aleatÃ³rios
  printf("\n\nEntre com quantidade de numeros aleatorios: ? ");
  scanf("%i", &n);

  // Aloca memÃ³ria dinamicamente para o vetor de nÃºmeros aleatÃ³rios
  double *number = (double *)malloc(n * sizeof(double));
  if (number == NULL) {
    printf("Ops... Problemas de alocaÃ§Ã£o de memÃ³ria.\n\n");
    return 1;
  }

  // Criar e escrever os nÃºmeros aleatÃ³rios em um arquivo de texto
  FILE *file = fopen("numeros.txt", "w");
  if (file == NULL) {
    printf("Erro ao criar o arquivo.\n");
    return 1;
  }

  printf("\n");
  sum = 0;
  for(i=0; i<n; i++){
    // rand() gera numeros aleatorios entre 0 e RAND_MAX (macro predefinida em stdlib.h)
    random_value = -1.0 + 2.0*(rand()/(double)RAND_MAX); // gera numero float aleatorio
entre -1 a  +1

    //printf("%2i) %7.4f ", i+1, random_value);
    // Exibir os nÃºmeros formatados na tela, 10 numeros por linha de texto
    printf("%7.4f ", random_value);
    if ((i + 1) % 10 == 0) {
      printf("\n");
    }

    sum = sum + random_value;
    number[i] = random_value; // guarda numero no vetor
    if (i == 0){
      menor = random_value;
      maior = random_value;
    }
    if (random_value < menor){
      menor = random_value;
    }
    if (random_value > maior){
      maior = random_value;
    }

    // gravando os dados em arquivo texto
    fprintf(file, "%f\n", random_value);
  }

  fclose(file);

  mean = sum/n;
  variance = 0;
  for(i=0; i<n; i++){
    variance = variance + pow(number[i] - mean, 2);
```

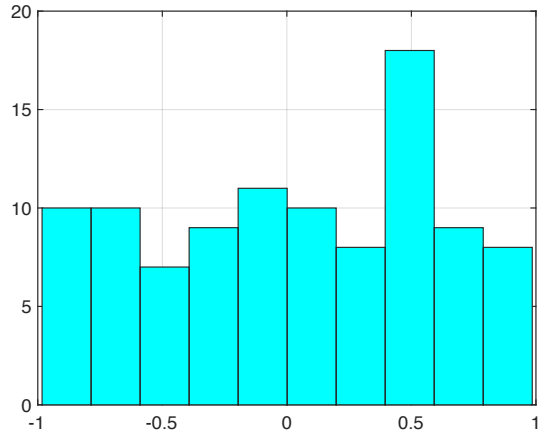
```
  }
  variance = variance / n;
  std = sqrt(variance);
  printf("\nMenor Valor gerado: %7.4f\n", menor);
  printf(  "Maior Valor gerado: %7.4f\n", maior);
  printf(  "Valor MÃ©dio gerado: %7.4f\n", mean);
  printf(  "Desvio padrÃ£o: %7.4f\n", std);
  printf("\nGerado arquivo <<numeros.txt>>\n\n");

  // liberando memoria alocada para o vetor
  free(number);
  return 0; // finalizado sem problemas
}
```

Quando este programa é executado é gerado algo como:

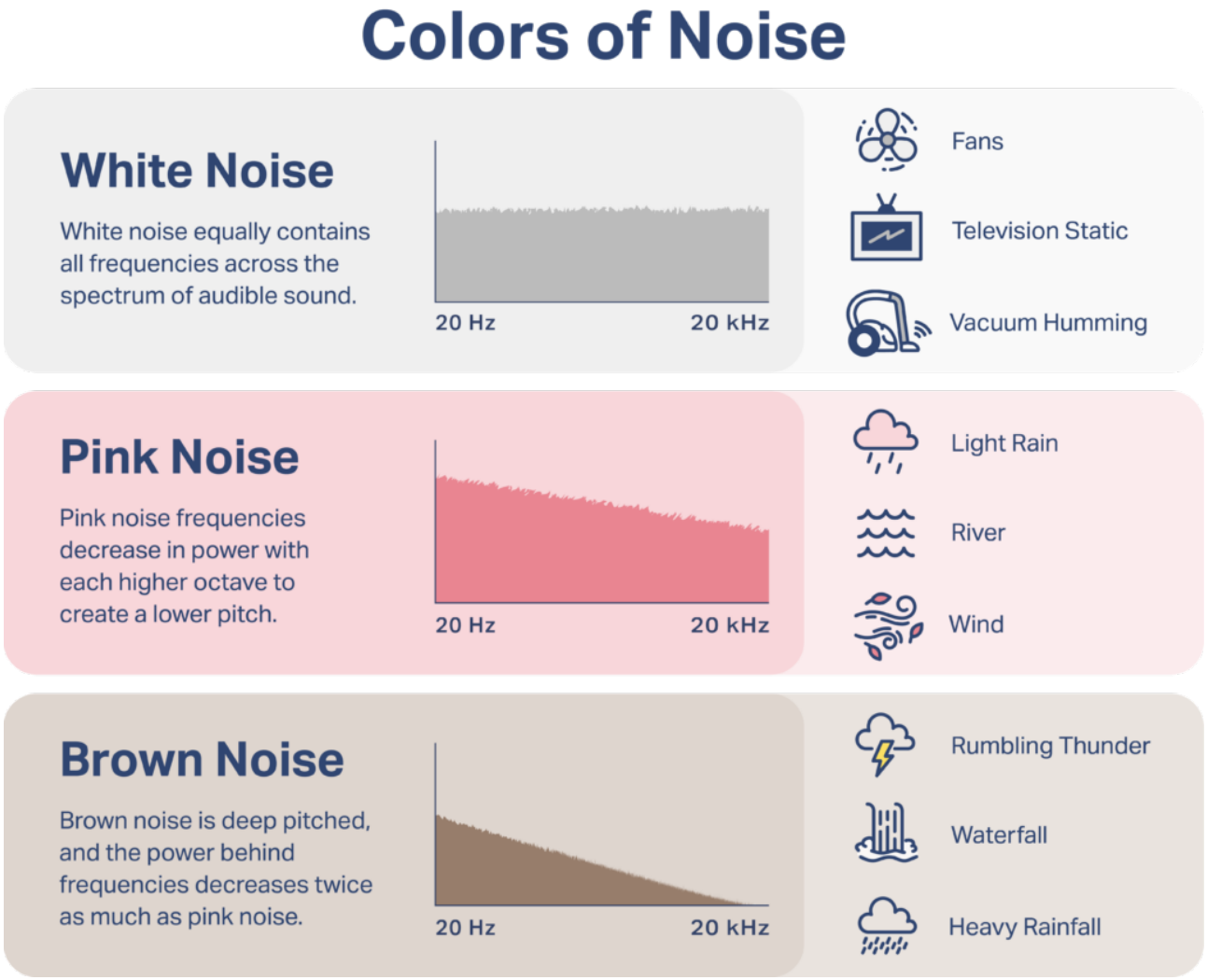
```
Entre com quantidade de numeros aleatorios: ? 100

-0.2361 -0.3761 -0.8436 -0.0068  0.2704  0.6617  0.5004  0.8708 -0.5923 -0.3414
-0.1216  0.5901  0.4626 -0.8744 -0.3377 -0.2776  0.4551 -0.7702 -0.6147 -0.5622
-0.3493 -0.8935 -0.4198  0.9429  0.5709 -0.1660  0.3175 -0.5929 -0.1706  0.5180
-0.1788  0.0927  0.4900 -0.9315  0.0125  0.8263 -0.5607 -0.9002  0.1107  0.4737
-0.9194  0.1900  0.0477 -0.1395 -0.0466  0.1038  0.6240  0.1286  0.5428  0.4775
-0.3848 -0.1472 -0.6986 -0.0547 -0.6157 -0.6168  0.1795 -0.7664 -0.7399  0.4650
-0.6243  0.9852  0.7330 -0.0371  0.2653  0.6434  0.3478  0.7835  0.3648  0.8503
 0.8741 -0.3186  0.5552  0.6024  0.2877 -0.5809  0.6047  0.6844 -0.4585  0.2302
 0.1718 -0.9670  0.2013  0.5402 -0.4300  0.6362 -0.9450 -0.8587 -0.2056  0.5344
 0.4739  0.5321  0.8808 -0.9828  0.4033  0.4361 -0.0192 -0.5120  0.1841  0.9749

Menor Valor gerado: -0.9828
Maior Valor gerado:  0.9852
Valor Médio gerado:  0.0252
Desvio padrão:  0.5614

Gerado arquivo <<numeros.txt>>
```

O gráfico e histograma mostrando distribuição dos números gerados, aparece à seguir:





Resumo



Mais material de consulta

- **White Noise Time Series with Python**, Jason Brownlee, March 6, 2017; URL: https://machinelearningmastery.com/white-noise-time-series-python/ - Esta página faz parte de uma série de artigos na área (ver: https://machinelearningmastery.com/category/time-series/) - Acessados em 23/08/2023.
- Vídeo no YouTube (em inglês): "**White Gaussian Noise**", da "Fun Cepts", URL: https://www.youtube.com/watch?v=iQJkOkIMP6M - Acessado em 23/08/2023