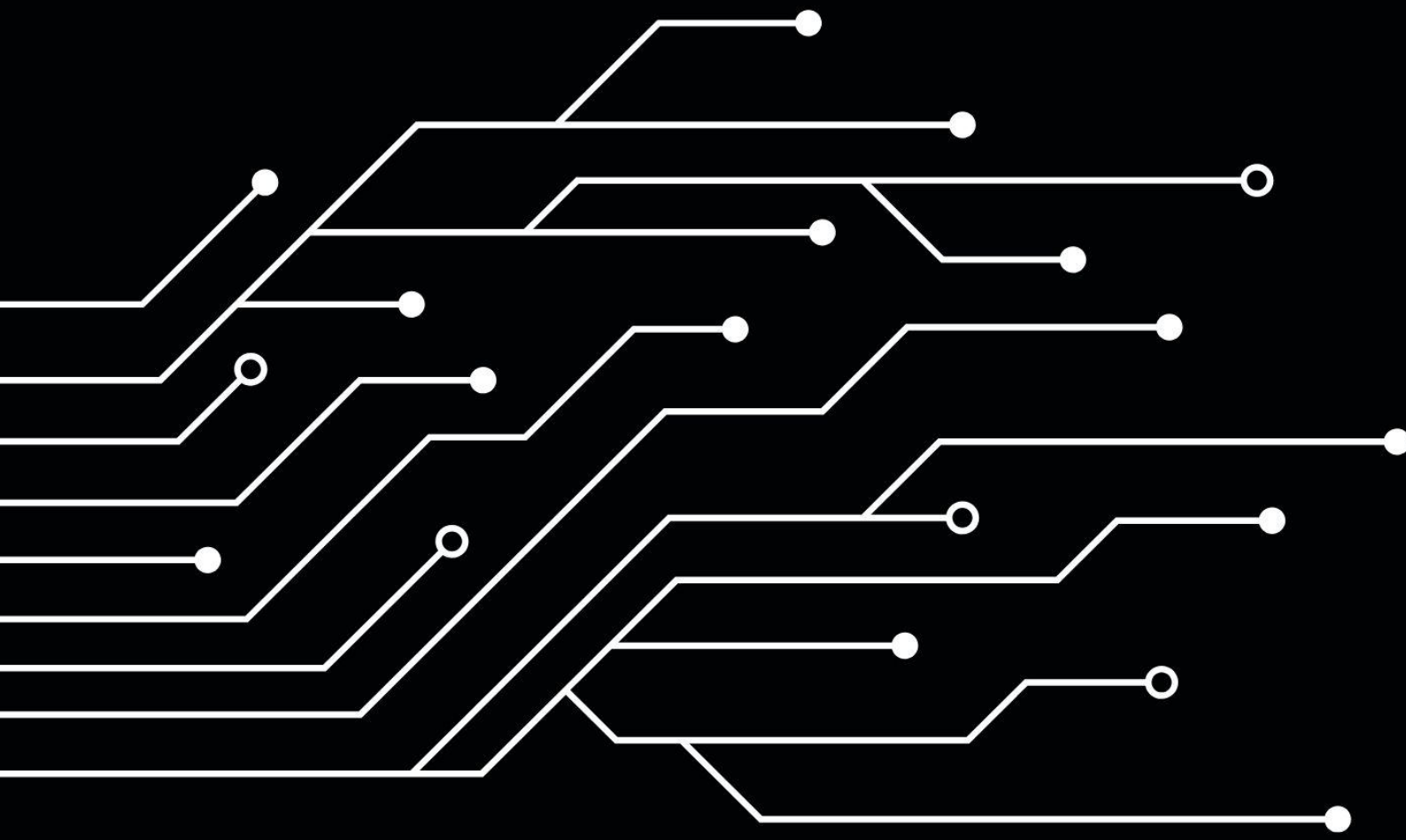
DSP

PARA COMUNIDADE MAKER



Paschoal Molinari Neto

DSP

PARA COMUNIDADE MAKER

1ªEdição

AUTOR

PASCHOAL MOLINARI NETO

DOI: 10.47538/AC-2025.61

ISBN: 978-6-55321-040-0

9 786553 210400

Ano 2025

DSP

PARA COMUNIDADE MAKER

1ªEdição

Catalogação da publicação na fonte

Molinari Neto, Paschoal.

DSP para comunidade *maker* [recurso eletrônico] / Paschoal Molinari Neto. – 1. ed. – Natal: Editora Amplamente, 2025.

PDF.
ISBN: 978-65-5321-040-0
DOI: 10.47538/AC-2025.61

1. Processamento digital de sinais. 2. Sistemas embarcados. 3. Comunidade *maker*. 4. Arduino. 5. Tecnologia aberta. I. Título.

CDU: 004.383
M722

Editora Amplamente
Empresarial Amplamente Ltda.
CNPJ: 35.719.570/0001-10
E-mail: publicacoes@editoraamplamente.com.br
www.amplamentecursos.com
Telefone: (84) 999707-2900
Caixa Postal: 3402
CEP: 59082-971
Natal- Rio Grande do Norte – Brasil

Declaração dos autores/ Declaração da Editora: disponível em:
https://www.amplamentecursos.com/politicas-editoriais
Editora-Chefe: Dayana Lúcia Rodrigues de Freitas
Assistentes Editoriais: Caroline Rodrigues de F. Fernandes; Margarete Freitas Baptista
Bibliotecária: Mônica Karina Santos Reis CRB-15/393
Projeto Gráfico, Edição de Arte e Diagramação: Luciano Luan Gomes Paiva; Caroline Rodrigues de F. Fernandes
Capa: Canva®/Freepik®
Parecer e Revisão por pares: Revisores

Ano 2025

Ano 2025

Agradecimentos

Em primeiro lugar agradeço ao IFPR pela oportunidade de conduzir a pesquisa que deu base a este livro. Agradeço também aos colegas professores e aos alunos, em especial ao Prof. Ronaldo Evaristo pelo incentivo. Finalmente agradeço a minha família e em particular a minha filha Giulia sempre presente na minha vida.

Ano 2025

Sumário

APRESENTAÇÃO

O livro *"DSP para Comunidade Maker"*, de autoria de Paschoal Molinari Neto, apresenta-se como uma contribuição inovadora e aplicada ao universo da tecnologia, voltada especialmente para a comunidade *maker* e para todos aqueles que se interessam pelo desenvolvimento de sistemas embarcados. A obra tem como objetivo difundir o uso do Processamento Digital de Sinais (DSP) em plataformas acessíveis, democratizando o acesso a ferramentas que, tradicionalmente, eram restritas a ambientes altamente especializados.

Com base sólida nos fundamentos do Processamento de Sinais Discretos no Tempo, o autor explora conceitos matemáticos e algoritmos essenciais; como sinais, convoluções, DFTs, FFTs e filtros; traduzindo-os em uma biblioteca de funções prática e funcional. Essa proposta permite que tais conceitos sejam não apenas estudados, mas efetivamente aplicados em projetos reais, ampliando as possibilidades de inovação tecnológica dentro e fora do ambiente acadêmico.

Um dos grandes diferenciais desta obra é a utilização de projetos de código aberto (*open-source*), reutilizáveis e adaptáveis. Essa escolha reforça o compromisso do autor com a democratização do conhecimento, abrindo caminhos para que a tecnologia DSP seja incorporada em projetos *makers*, didáticos ou profissionais, em áreas como controle, automação, robótica, telecomunicações, IoT e informática.

A metodologia adotada envolve a programação, teste e avaliação de desempenho da biblioteca em plataformas microcontroladas, com destaque para o uso do Arduino UNO R4 (32 bits ARM) e ambientes de desenvolvimento abertos. Esse aspecto torna o livro ainda mais acessível e prático, permitindo que tanto iniciantes quanto entusiastas experientes possam explorar o potencial do DSP em suas próprias criações.

Ao longo das páginas, o leitor encontrará uma ponte entre teoria e prática, construída com clareza e rigor técnico, mas também com a preocupação de tornar o conteúdo aplicável e estimulante. Dessa forma, *DSP para Comunidade Maker* se configura como um guia indispensável para estudantes, professores, pesquisadores e desenvolvedores que desejam ampliar seus horizontes no campo da tecnologia digital.

Esta primeira edição, ao integrar ciência, prática e inovação, reafirma a relevância do papel da comunidade *maker* como espaço de criatividade, colaboração e transformação tecnológica. Que esta obra inspire novas ideias, projetos e soluções, consolidando o DSP como ferramenta poderosa no cotidiano daqueles que ousam imaginar e construir o futuro.

Desejamos uma boa leitura!

Editora Amplamente

CAPÍTULO I

INTRODUÇÃO E MOTIVAÇÃO

Este trabalho é dedicado ao estudo prático do Processamento Digital de Sinais (*DSP - Digital Signal Processing*), através de uma biblioteca didática e aberta de funções DSP para a plataforma aberta Arduino UNO R4, indo da teoria matemática a programação na linguagem C++.

As plataformas Arduino são populares mundialmente para prototipagem de eletrônica e programação, tipicamente em linguagem C++. São versáteis e acessíveis a uma grande comunidade aberta de usuários *makers* tais como amadores, estudantes e professores do ciclo básico ao universitário, pesquisadores, projetistas e desenvolvedores de soluções. Os hardwares e softwares da Arduino são abertos (*open-source*) e bem documentados, disponíveis no site https://www.arduino.cc .

Figura 1 – Placas Arduino UNO R3

O bem-sucedido Arduino UNO R3 (Revisão 3) foi lançado em 2012 e estima-se mais de 10 milhões de usuários. Esta placa apresenta um microcontrolador

ATMEL (atualmente MICROCHIP) ATmega328P baseado em arquitetura de 8 bits. A placa possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, um ressonador cerâmico de 16 MHz (CSTCE16M0V53-R0), uma conexão USB, um conector de alimentação, um conector ICSP e um botão de reset. Pode ser alimentado pela USB (https://store.arduino.cc/products/arduino-uno-rev3)

O novo Arduino UNO R4 WiFi (Revisão 4) foi lançado em 2023, combina o poder de processamento e os poderosos periféricos do microcontrolador de 32 bits RA4M1 da empresa RENESAS (Japão), baseado na arquitetura Arm® Cortex®-M4F, com a versatilidade de conectividade sem fio (Wi-Fi® *and Bluetooth*®) do ESP32-S3 da *Espressif*.

Além disso, o UNO R4 WiFi tem na face superior da placa uma matriz de LEDs 12x8 on-board, conector Qwiic, VRTC e pino OFF, cobrindo muitas das necessidades potenciais que os desenvolvedores *Makers* e trabalhos interdisciplinares terão em projeto futuros.

O UNO R4 também introduz uma variedade de periféricos integrados, incluindo um DAC de 12 bits, CAN BUS e OP AMP, fornecendo recursos expandidos e flexibilidade de projeto. A conexão elétrica e de comunicação com computadores se faz por cabo padrão USB-C. A alimentação da fonte de energia (VIN) foi estendida para até 24v (6v – 24v), permitindo integração mais simples com outros componentes de projeto (motores, atuadores etc.). Existe também uma versão Arduino UNO R4 Minima que não dispõe de WiFi e de matriz de LEDs (8x12 LEDs) (https://store-usa.arduino.cc/products/uno-r4-wifi)

O microcontrolador de 32bits do Arduino UNO R4 tem no geral uma ordem de magnitude mais capacidade de processamento comparado ao do UNO R3, e elevada compatibilidade a nível de código na linguagem C++. Mantém também o mesmo formato, pinagem e tensão operacional de 5 V de seu antecessor, proporcionando uma transição adequada para shields (placas de expansão por conexão superposta), projetos eletrônicos e programas existentes.

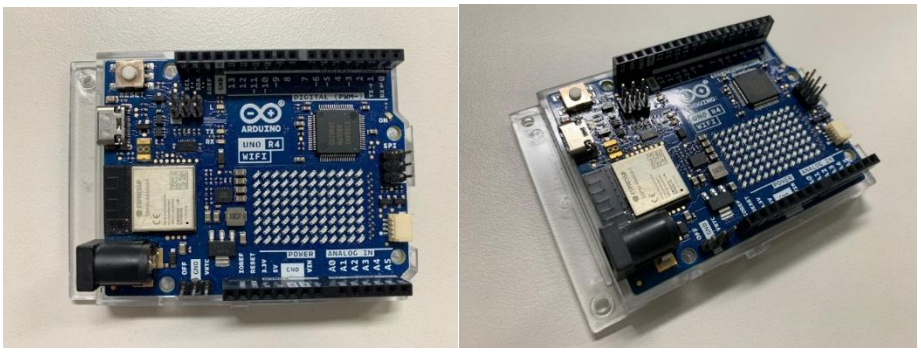


Figura 2 – Fotos placa Arduino UNO R4 WiFi (Fonte: autor/2025)

O microcontrolador RA4M1 tem uma unidade de ponto flutuante (FPU – Floating Point Unit) do Arm® Cortex®-M4F, oferecendo suporte total a operações de dados de ponto flutuante de precisão simples (32 bits), que incluem adição, subtração, multiplicação, divisão, multiplicação e acumulação (MAA) e operações de raiz quadrada, todas compatíveis com o padrão ANSI/IEEE Standard 754-2008.

Existe disponível também o Arduino UNO R4 Minima, que não tem Wi-Fi e a matriz de LEDs (https://store.arduino.cc/products/uno-r4-minima).

Com o Arduino UNO R4 a comunidade *Maker* poderá desenvolver projetos interdisciplinares mais avançados que incluam formas simples de:

- Processamento Digital de Sinais (*DSP – Digital Signal Processing*);
- Internet das Coisas (IoT);
- Aprendizagem de Máquinas (*Machine Learning*) para microcontroladores (TinyML);
- Jogos e Entretenimento;
- Robótica;
- Domótica;
- Controle e Automação.

Este trabalho também favorece as atividades interdisciplinares de uma forma geral pois envolve:

- Projetos práticos;
- Desenvolvimento de habilidades;
- Contextualização do conhecimento;
- Aprimoramento do processo de aprendizagem.

A facilidade de programação e prototipação do Arduino é um ganho no desenvolvimento de soluções simples e eficientes para a comunidade *Maker* e as atividades interdisciplinares. Isto nos motiva a implementar e testar uma biblioteca *open-source* (fonte aberta) de funções DSP e afins para o UNO R4, com vista a explorar as capacidades de processamento desta plataforma, e assim apoiar as atividades dos autodidatas, makers, acadêmicos e profissionais interessados no assunto.

Eventualmente esta biblioteca poderá ser migrada para outras plataformas. Por fim uma visão sobre a cultura *Maker* no Wikipedia:

> "A cultura *Maker* é uma subcultura contemporânea que representa uma extensão baseada em tecnologia da cultura DIY que se cruza com partes orientadas a hardware da cultura hacker e se deleita na criação de novos dispositivos, bem como na manipulação dos existentes. A cultura *Maker* em geral apoia hardware de código aberto. Os interesses típicos apreciados pela cultura *Maker* incluem atividades orientadas à engenharia, como eletrônica, robótica, impressão 3D e o uso de ferramentas de controle numérico computadorizado, bem como atividades mais tradicionais, como metalurgia, marcenaria e, principalmente, seu antecessor, artes e ofícios tradicionais.
>
> A subcultura enfatiza uma abordagem de cortar e colar para tecnologias padronizadas de hobby e incentiva a reutilização de livros de receitas de designs publicados em sites e publicações voltadas para o criador. Há um forte foco no uso e aprendizado de habilidades práticas e na aplicação delas em designs de referência" (https://en.wikipedia.org/wiki/Maker_culture).

DESENVOLVIMENTO E USO DA BIBLIOTECA DSP

A biblioteca de programas DSP e afins foi desenvolvida e testada, na sua forma final, com o ambiente de desenvolvimento integrado Arduino IDE (*Integrated Development Environment)* versão 2.3.6 do Windows e na plataforma Arduino UNO R4 WiFi.

O Arduino IDE pode ser executado em diferentes sistemas operacionais como Windows ou Linux, e pode ser obtido via download no endereço https://www.arduino.cc/en/software

Muitos programas geram avisos, informações e dados que são enviados pelo Arduino UNO R4, comunicação serial de 115.200 bps via USB-C, para o ambiente IDE do Arduino no computador. Atenção, esta velocidade de comunicação serial deverá ser selecionada também no Arduino IDE. A comunicação serial pode ser observada e copiada via copiar e colar (CTRL-C CTRL-V) na ferramenta "serial monitor" do Arduino IDE.

O Arduino UNO R4 executa programas escritos em um subconjunto da linguagem de programação C++ padrão (*Bjarne Stroustrup*), que é uma extensão da linguagem C (Kernighan e Ritchie,1978). A referência do conjunto de funções, variáveis e linguagem C++ do Arduino pode ser vista em https://www.arduino.cc/reference/en/

A referência do conjunto de bibliotecas do Arduino pode ser vista em https://www.arduino.cc/reference/en/libraries/

Ao longo deste livro a maior parte dos sinais a serem processados (geração, transformação, análise e filtragem) são discretos no formato inteiro de 16 bits da linguagem C++. Devido a necessidade de precisão do processamento dos algoritmos DSP, durante o processamento poderemos usar outros formatos internos as funções da biblioteca, como inteiro de 32 bits e ponto flutuante de 32 bits. Isto deve contemplar a maior parte das aplicações desta plataforma.

A documentação guia para placas Shield para o Arduino R4 pode ser vista em https://docs.arduino.cc/tutorials/uno-r4-minima/shield-guide/

Por fim observamos que a biblioteca DSP *Maker* contida neste livro é licenciada conforme a internacionalmente popular e reconhecida MIT *License* que segue conforme abaixo:

> Copyright © 2025 Paschoal Molinari Neto[1]
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Tradução livre para o português:

> Copyright © 2025 Paschoal Molinari Neto
>
> É concedida permissão, gratuita, a qualquer pessoa que obtenha uma cópia deste software e dos arquivos de documentação associados (o "Software"), para lidar com o Software sem restrições, incluindo, entre outras, os direitos de usar, copiar, modificar, mesclar, publicar, distribuir, sublicenciar e/ou vender cópias do Software, e para permitir que as pessoas a quem o Software é fornecido o façam, sujeito às seguintes condições:
>
> O aviso de direitos autorais acima e está permissão deverão ser incluídos em todas as cópias ou partes substanciais do Software.
>
> O SOFTWARE É FORNECIDO "NO ESTADO EM QUE SE ENCONTRA", SEM GARANTIA DE QUALQUER TIPO, EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS NÃO SE LIMITANDO ÀS GARANTIAS DE COMERCIALIZAÇÃO, ADEQUAÇÃO A UM DETERMINADO FIM E

[1] Para informações em português sobre esta licença: https://pt.wikipedia.org/wiki/Licença_MIT

NÃO VIOLAÇÃO. EM NENHUMA HIPÓTESE OS AUTORES OU TITULARES DOS DIREITOS AUTORAIS SERÃO RESPONSÁVEIS POR QUALQUER RECLAMAÇÃO, DANOS OU OUTRA RESPONSABILIDADE, SEJA EM UMA AÇÃO CONTRATUAL, ATO ILÍCITO OU DE OUTRA FORMA, DECORRENTE DE, DE OU EM CONEXÃO COM O SOFTWARE OU O USO OU OUTRAS NEGOCIAÇÕES NO SOFTWARE.

CAPÍTULO II

GERAÇÃO DE SINAIS E RUÍDOS

Neste capítulo é feito uma rápida revisão teórica da classificação e geração de sinais repetitivos ou não, pseudoaleatórios e ruídos, bem como, de forma didática, sua otimizada implementação e teste, em linguagem C++, no Arduino UNO R4.

RÁPIDA REVISÃO DOS TIPOS DE SINAIS

Sinais são funções de uma ou mais variáveis independentes, e contêm informações sobre o comportamento ou natureza de algum fenômeno. Exemplos de sinais são tensões e correntes como uma função do tempo em um circuito elétrico, ou dados amostrados vindos de sensores diversos tais como de som, imagem, pressão, vazão, temperatura, posição, velocidade, aceleração, vibração, luminosidade, antenas, radiação, sensor de partículas etc.

Formalmente em Processamento Digital de Sinais, sinal é uma função x(t), onde "t" é uma variável independente genérica, e a função pode ser determinística ou a realização de um processo estocástico. Os valores da variável independente podem ser contínuos ou discretos. Quando o sinal é discreto então é uma função x(n), onde "n" é a variável independente.

Especificamente:

- **Uma variável independente continua tem infinitos valores entre dois pontos**. Na figura 3 o exemplo é um sinal x(t) da variável independente continua t.
- **Uma variável independente discreta tem finitos valores entre dois pontos**. Na figura 3 o exemplo é um sinal x(n) da variável independente

discreta n. Neste exemplo x(n) é um sinal amostrado do sinal x(t), com 25 amostras.

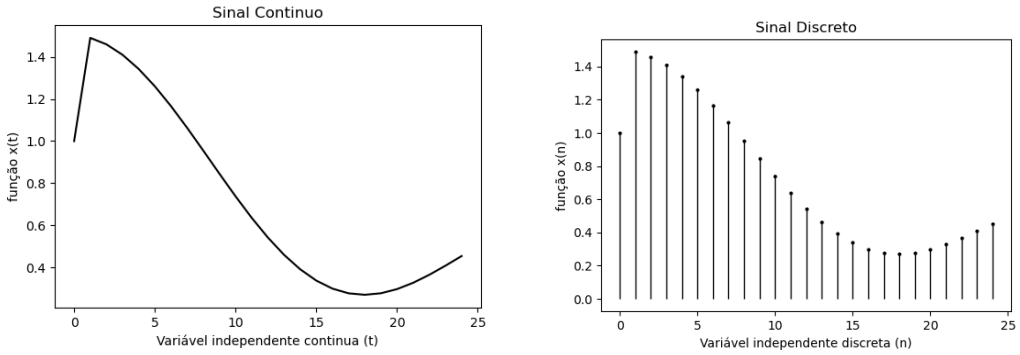


Figura 3 - Exemplos do mesmo sinal nas formas continua x(t) e discreta x(n)

No Processamento Digital de Sinais a variável independente é discreta, e não necessariamente é o tempo, podendo ser, por exemplo, o espaço no caso de pixels (pontos) em uma imagem, ou temperatura ao longo de uma superfície.

Sinais discretos podem ser obtidos por:

- **Amostragem de uma sequência de medições discretas**. Por exemplo, podem ser obtidos através de medições do conversor analógico-digital (ADC – *Analog Digital Converter*) de um microcontrolador. No caso do Arduino UNO R4 o ADC é de até 14 bits, podendo amostrar sinais na faixa de amplitude de 16.384 níveis (de zero a 16.383 em números inteiros).
- **Síntese de uma sequência de números discretos**. Geração por múltiplos algoritmos, como exemplo geração de senoide, formas quadradas, triangulares, sinais pseudoaleatórios diversos. Algumas funções de síntese de sinais discretos estão disponíveis na biblioteca DSP. O Arduino UNO R4 tem conversores digital-analógico (DAC – *Digital Analog Converter*) de 8 e 12 bits, podendo gerar sinais na faixa de amplitude de até 4.096 níveis (de zero a 4.095 em números inteiros). Adicionalmente o UNO R4 tem temporizadores PWM (*Pulse Width Modulation*) de 16 e 32 bits.

Geralmente os sinais discretos no Processamento Digital de Sinais são números inteiros ou ponto-flutuante obtidos por amostragem de ADCs, por dados informados através de rede local ou gerados internamente for funções geradoras de sinais.

SÉRIES DE FOURIER

As Séries de Fourier são um instrumento matemático importante para o entendimento e análise de sinais discretos. Elaborado no século 19 pelo matemático francês Jean-Baptiste Joseph Fourier, estas séries descrevem sinais periódicos como uma soma de funções trigonométricas. Sinais de funções periódicas seguem a formulação abaixo:

$$f(t) = f(t + nT)$$

Onde n é um número inteiro e T, o período da função.

O Teorema de Fourier implica que qualquer sinal periódico, de qualquer formato finito, pode ser expresso na forma de uma soma infinita de funções trigonométricas conforme a fórmula:

$$f(t) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(n\omega_0 t) + b_n sen(n\omega_0 t)]$$

Onde:

- a_0 é um valor médio continuo do sinal;
- a_n e b_n são os coeficientes de Fourier;
- $\omega_0 = 2\pi/T$ é a frequência fundamental em radianos por segundo.
- As funções trigonométricas seno e cosseno são as n-ésimas harmônicas do sinal.

As séries de Fourier nos permitem decompor sinais representados no domínio do tempo em sinais representados no domínio de frequência.

Nas figuras a seguir o exemplo de uma onda quadrada contínua de amplitude [-1,1] e suas 2 primeiras harmônicas somadas (pontilhado) e 5 primeiras harmônicas.

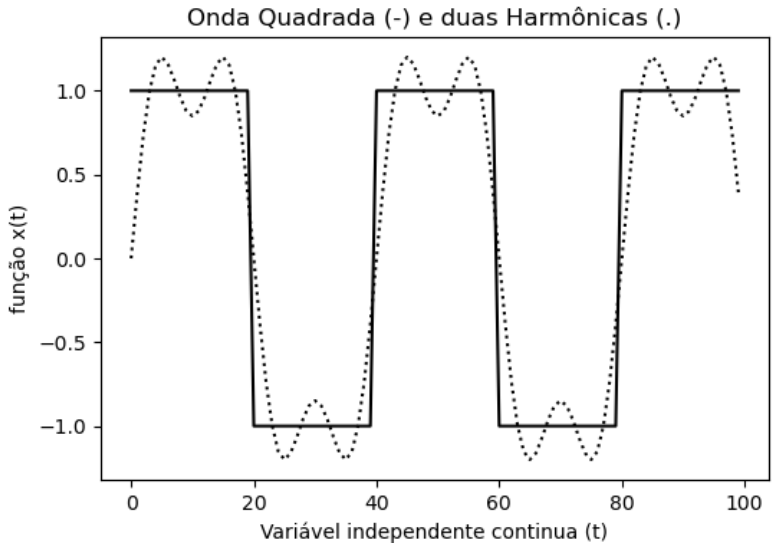


Figura 4 – Onda Quadrada e duas Harmônicas

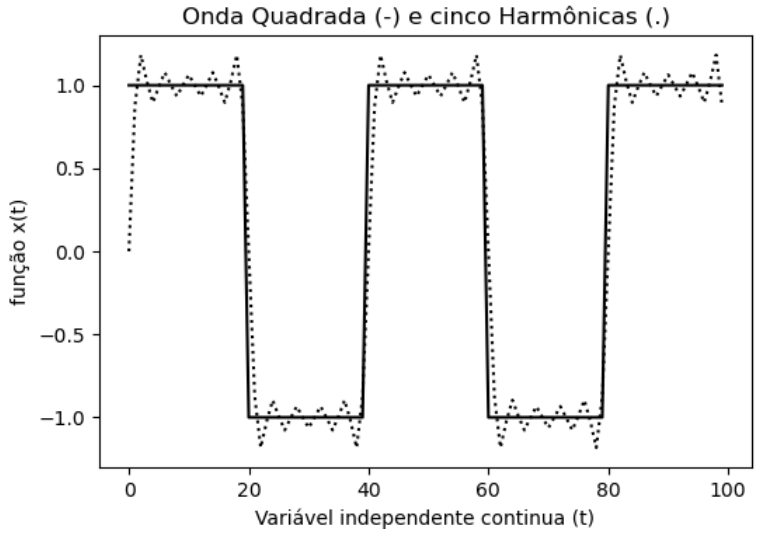


Figura 5 - Onda Quadrada e cinco Harmônicas

Podemos gerar um sinal suficientemente próximo, do ponto de vista prático, da onda quadrada usando um bom número de harmônicas. Segue figura como exemplo de quinze harmônicas.

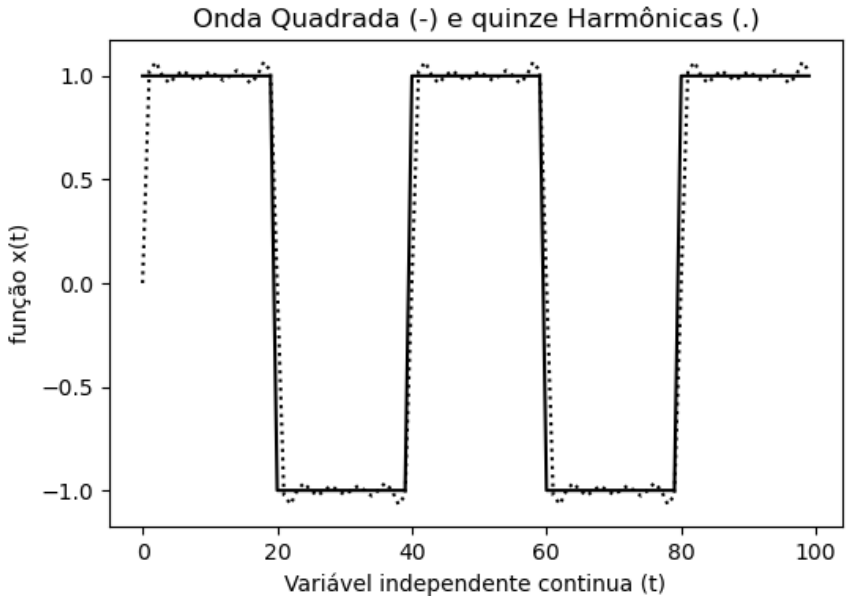


Figura 6 - Onda Quadrada e quinze Harmônicas

O que fizemos com a onda quadrada pode ser feito com qualquer outra forma de onda arbitraria e finita em amplitude.

Com o entendimento que a Série de Fourier nos traz sobre sinais nós podemos analisar, gerar, transformar e filtrar estes sinais, especialmente quando usamos as técnicas modernas do Processamento Digital de Sinais. Isto é o que faremos no restante desta obra.

RÁPIDA REVISÃO SOBRE RUÍDOS

Ruído, de uma forma geral, é um sinal indesejado ou prejudicial, considerado desagradável, inoportuno, alto, ofuscante, prejudicial à atividade em desenvolvimento ou distorcendo um sinal válido. Do ponto de vista da física, não há distinção entre ruído e o sinal desejado, pois ambos são alterações de valores de meio, como o ar, a água, ou meio eletromagnético. A diferença surge quando o cérebro, ou um equipamento, recebe e percebe ou tenta processar o ruído.

No campo do processamento digital de sinais, ruído é um termo geral para modificações indesejadas (e, na maioria das vezes, desconhecidas) que um sinal

pode sofrer durante as etapas de captura, armazenamento, transmissão, processamento ou conversão do sinal.

Podemos classificar os ruídos em quatro tipos principais:

- Ruído contínuo – Gerado por entidades, objetos ou máquinas sem interrupções.
- Ruído intermitente – ruído infrequente, ou seja, não contínuo e com variações de regularidade e intensidade.
- Ruído impulsivo – ruídos não regulares, que eventualmente podem acontecer de forma aleatória ou não.
- Ruído de baixa frequência – geralmente originado pelo meio ambiente na combinação de fontes diversas.

Existe uma maneira muito peculiar de classificar os ruídos sonoros fazendo analogia com cores. Esta classificação se baseia na densidade espectral dos sinais em função da frequência conforme gráfico abaixo (fonte: *wikipedia*). Existem definições para cada cor de ruído, geralmente a densidade espectral de potência por unidade de largura de banda proporcional a $\frac{1}{f^{\beta}}$ onde β tem valores diferentes para cada cor.

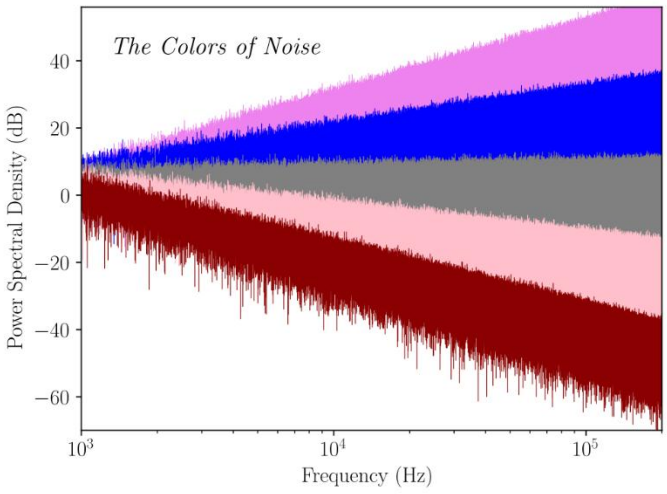


Figura 7 - Densidade espectral dos sinais em função da frequência (Fonte: Wikipedia)

No exemplo acima o ruído rosa (*pink noise*) aumenta a potência de forma mais intensa à medida que a frequência da componente do ruído aumenta. De forma inversa no ruído marrom a potência diminui com o aumento da frequência. Aparentemente isto é uma analogia vaga entre o espectro sonoro e o espectro da luz visível.

Particularmente importante é o conceito de ruído branco, associado a densidade igual para todas as frequências, em analogia a luz branca. O conceito de ruído branco (*white noise*) é utilizado genericamente em várias outras faixas espectrais de frequência, além da sonora e da luz visível.

De forma genérica para o processamento discreto (digital) de sinais o ruído branco é um sinal na forma de sequência serial discreta de valores não correlatos com média zero e variância finita. Em alguns casos exige-se características adicionais.

RÁPIDA REVISÃO DE GERAÇÃO DE NÚMEROS PSEUDOALEATÓRIOS

Uma maneira prática de gerar ruído é utilizar um gerador de números pseudoaleatórios (em inglês, PRNG - *Pseudo-Random Number Generator*), como o clássico Gerador Congruencial Linear (*Linear Congruential Generator* - LCG), cuja algoritmo para geração de uma sequência pseudoaleatória X é:

$$X_{n+1} = (aX_n + c) mod(m)$$

Onde:

- X é uma sequência pseudoaleatória de números inteiros
- X_0 é a semente (seed), ou valor inicial da sequência, sendo $0 \leq X_0 < m$
- X_n é o valor atual de *X* na sequência
- X_{n+1} é o próximo *X* a ser gerado na sequência
- a é o multiplicador inteiro, sendo $0 < a < m$
- c é o incremento inteiro, sendo $0 \leq c < m$
- m é o maior número inteiro que pode ser gerado, sendo $0 < m$
- mod é a função resto que retorna o resto depois da divisão de um número pelo divisor m

Como exemplo simplificado segue o cálculo de X_{17} com $X_{16} = 7$, a = 11 , c = 6 e m = 15 usando a fórmula acima:

$$X_{17} = (aX_{16} + c)mod(m) = (11\ x\ 7 + 6)mod(15) = 8$$

Em mais uma iteração calculando X_{18} temos:

$$X_{18} = (aX_{17} + c)mod(m) = (11\ x\ 8 + 6)mod(15) = 4$$

O algoritmo LCG é relativamente simples de escrever, entretanto exige muita atenção na sintonia dos parâmetros *a, c* e *m*, bem como cuidados nos detalhes de implementação. Observe que o valor médio de uma sequência de números assim gerada é aproximadamente $\frac{m}{2}$ para sequência com elevado número de elementos.

Vamos implementar, usando a linguagem C++, para o Arduino UNO R4 uma versão de LCG que foi originalmente desenvolvida para computadores caseiros (Sinclair ZX81), baseados nos microprocessadores Z80 de 8 bits no início da década de 1980. Os três parâmetros pré-sintonizados que serão utilizados são:

$$a = 75;\quad c = 74;\ m = 2^{16} + 1 = 65537$$

Com isto temos uma solução fácil e rápida para geração de números que são pseudoaleatórios na amplitude de faixa [0,65535], e também simulacros de ruído branco, para um sistema microcontrolado.

IMPLEMENTAÇÃO DO GERADOR DE NÚMEROS PSEUDOALEATÓRIO

Para testar o gerador de números pseudoaleatório, PRNG do tipo LCG, na plataforma Arduino UNO R4, desenvolvemos uma função em Linguagem C++ (dsp_prng()) e com uso da biblioteca de funções referência do Arduino.

A função do exemplo didático a seguir desenvolvido está escrito de forma simplificada. Não tem grandes otimizações de código e utiliza apenas o básico do IDE do Arduino.

O programa chama a função que gera um único número “X” pseudoaleatório por vez a cada fração de segundo. O número gerado é condicionado e enviados pelo Arduino, via comunicação serial, para o ambiente IDE do Arduino e pode ser observado na ferramenta “serial monitor” do IDE. O número X gerado é reutilizado como semente para gerar o próximo número da sequência.

O condicionamento feito é opcional e gera um número “x” com amplitude de faixa reduzida em 16 vezes para [0, 4095], como se fosse um sinal de amplitude de 12 bits armazenado no formato inteiro de 16 bits. Com isto o sinal condicionado pode ser armazenado com mais facilidade e enviado para o conversor digital-analógico (DAC) ou para temporizador PWM, sendo eventualmente utilizado externamente ao microcontrolador no mundo analógico. Outros condicionamentos são possíveis.

Durante a demonstração o LED do Arduino ficará piscando, indicando ao usuário que está ativo e com o programa de teste de geração de números em execução. A parte do programa que faz isto é simples e não bloqueia o programa principal.

```
// DEMONSTRAÇÃO DE FUNÇÃO GERADORA DE NÚMEROS
PSEUDOALEATÓRIOS - PRNG
// Gerador Congruencial Linear (Linear Congruential Generator - LCG)
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.6
// Data da última revisão: 2025-05-26
// Autor: Paschoal Molinari
// Licença: MIT License

// Declaração de variáveis globais
int i, j, linhas, colunas, x_12b;     // contadores de loop e variável condicionada
"x_12b"
int X = 1 ;   // variável dos números pseudoaleatórios (PRN) com semente inicial
(seed)

// função de setup inicial do ambiente
void setup() {
  Serial.begin(115200);        // seleção de velocidade de comunicação serial
  Serial.println("  ");        // Mensagem inicial do programa
```

```
  Serial.println("  ");
  Serial.println("  //  GERAÇÃO DE NÚMEROS PSEUDOALEATÓRIOS - PRNG  //");
  pinMode(LED_BUILTIN, OUTPUT);
  // configurando a quantidade e formato dos dados impressos por vez
  linhas = 5;
  colunas = 10;
}

// Função de geração de números pseudoaleatórios
// PRNG - Pseudo-random Number Generator
// Gera número inteiro 16 bits pseudoaleatório na faixa [0,65535]
int dsp_prng(int prn_seed){
  // prn é o número pseudoaleatório atual
  // valores iniciais da geração pseudoaleatória
  long a = 75,  c = 74;     // constantes a e c da função linear aX + c
  long m = 65537;          // constante divisor da função resto
  long prn;               // número pseudoaleatório
  // Geração do novo número pseudoaleatório (PRN)
  prn = long(prn_seed);
  prn = a*prn + c;         // função linear aX + c
  prn = prn - m*(prn/m);    // Resto da divisão de prn por m
  return(prn);            // retorna novo número pseudoaleatório (PRN)
}

// Função de loop infinito
void loop() {
  // Código principal do programa em loop infinito
  digitalWrite(LED_BUILTIN, HIGH);   // Liga o LED (HIGH is the voltage level)
  Serial.println(" LED PISCA ");
  delay(5000);                    // espera dois segundos
  digitalWrite(LED_BUILTIN, LOW);    // Desliga o LED
  delay(500);                     // espera meio segundo
  Serial.println(" 50 Números pseudoaleatórios gerados individualmente");

  // Loop para gerar 50 números pseudoaleatórios
  Serial.print("[");
  for(i=0; i< linhas ;i++){
    for(j=0; j< colunas ;j++){
    // função gerando um único número pseudoaleatório a partir do número
anterior
    X = dsp_prng(X);
    // Condicionamento da faixa para [0,4095], equivalente a número de 12 bits
    x_12b = int(X/16);
    // Impressão formatada do número na serial
    Serial.print(" ");
```

```
   Serial.print(x_12b);
   if(j < colunas-1)
    {Serial.print(",");}
   else
    {if(i < linhas-1)
     {Serial.print(",");}
    else
     {Serial.print("]");}
    }
   }
   Serial.println("  ");
 }
 Serial.println("  ");
}
```

Segue abaixo a tela da ferramenta Monitor do ambiente Arduino IDE. Os números pseudoaleatórios são apresentados na forma de um arranjo de 50 elementos separados por vírgula e delimitados por colchetes [] para serem facilmente transferidos por copiar-colar.

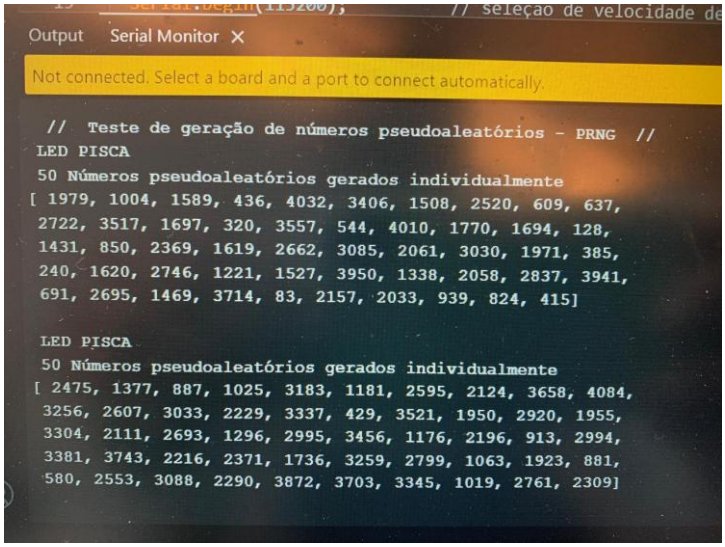


Figura 8 – Monitor do ambiente Arduino IDE

No programa exemplo a seguir temos uma função em C (dsp_prng_seq()) para geração de uma sequência de N números pseudoaleatórios no formato inteiro de 16 bits. Pode-se utilizar a sequência gerada como um simulacro de ruído branco.

Tem opcional de divisão para condicionamento dos números gerados. Este condicionamento não é aplicado a variável da semente de sequência.

```
// DEMONSTRAÇÃO DE FUNÇÃO GERADORA SEQUÊNCIA DE NÚMEROS PSEUDOALEATÓRIOS - PRNG
// Gerador Congruencial Linear (Linear Congruential Generator - LCG)
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.6
// Data da última revisão: 2025-05-26
// Autor: Paschoal Molinari
// Licença: MIT License

// Declaração de variáveis globais
int count=0;       // contador de números para impressão na serial
int x_comp = 50;    // comprimento da sequência de números pseudoaleatórios (PRN)
long prn_seed = 1;  // A semente de cada sequência de números
int x_signal[50];   // Sinal gerado com 50 números PRN em sequência

// função de setup inicial do ambiente
void setup() {
  Serial.begin(115200);     // seleção de velocidade de comunicação serial
  Serial.println("  ");     // Mensagem inicial do programa
  Serial.println("  ");
  Serial.println(" // GERAÇÃO SEQUENCIAL DE NÚMEROS PSEUDOALEATÓRIOS - PRN //");
  pinMode(LED_BUILTIN, OUTPUT);
}

// Função de geração de sequência de números pseudoaleatórios (PRN) inteiros
long dsp_prng_seq(int *x,int N, long prn, int xdiv, int xadd){
  // x é um arranjo ao qual se adiciona o PRN gerado
  // N é o comprimento do arranjo x
  // prn é a variável semente e também temporária da sequência pseudoaleatória
  // xdiv é um opcional de divisão/condicionamento dos PRN gerados
  // xadd é um opcional de adição (1) ou substituição (0) do arranjo x original

  // valores iniciais da geração pseudoaleatória
  long a = 75,  c = 74;  // constantes a e c da função linear aX + c
  long m = 65537;       // constante divisor da função resto
  int n;              // contador de loop da sequência
```

```
 // Geração de sinal X pseudoaleatório --- PRNG Pseudo-random Number
Generator
 for(n=0; n < N ;++n){
  prn = a*prn + c;              // cálculo da função linear aX + c
  prn = prn - m*(prn/m);        // PRN final como o resto da divisão de prn por m
  x[n] = x[n]*xadd + int(prn/xdiv);  // eventual adição do PRN condicionado a
sequência
  }
 return(prn);    // retorna pseudoaleatório semente para geração das próximas
sequências
}

// Função de loop infinito
void loop() {
 int i;

 // código principal que fica em loop infinito

 digitalWrite(LED_BUILTIN, HIGH);   // Ligando o LED (HIGH is the voltage level)
 delay(200);                  // esperando meio segundo
 digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
 delay(100);                  // esperando meio segundo

 if (count == 0) {
    Serial.println(" Números gerados sequencialmente");
 }

 // geração de uma sequência finita (x_signal) de números pseudoaleatórios
(PRN)
 prn_seed = dsp_prng_seq(x_signal, x_comp, prn_seed, 16, 0);

 // Impressão de números na serial em formato de arranjo
 Serial.print(" ");
 Serial.print("[");
 for(i=0;i<x_comp;i++){
  if ((i!=0)&&(i%10 == 0)){ Serial.println("  ");}
  Serial.print(" ");
  Serial.print(x_signal[i]);
  if(i < x_comp-1) {Serial.print(",");}
 }
  Serial.print("]");
 Serial.println("  ");
 Serial.println("  ");
 delay(6000);
}
```

AVALIAÇÃO DO SINAL PSEUDOALEATÓRIO

Seguem abaixo cinco visualizações com exemplos gráficos do funcionamento do Gerador de Números Pseudoaleatórios (PRNG) para uma avaliação visual do sinal obtido. Primeiro o sinal pseudoaleatório com 8192 pontos discretos de 16 bits de amplitude com sinal em sequência no domínio do tempo. Visualmente parece um ruído branco sem um padrão discernível, apenas um intenso ruido tipo “sal-e-pimenta”.

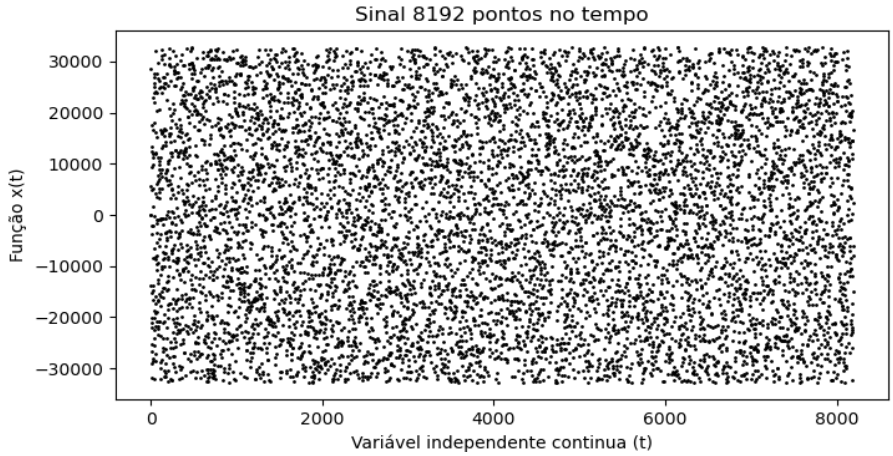


Figura 9 -Sinal Pseudoaleatório com 8192 pontos

Segunda visualização o sinal pseudoaleatório com 512 pontos próximos plotados com interpolação continua de 16 bits de amplitude com sinal em sequência no domínio do tempo. Novamente sem um padrão discernível.

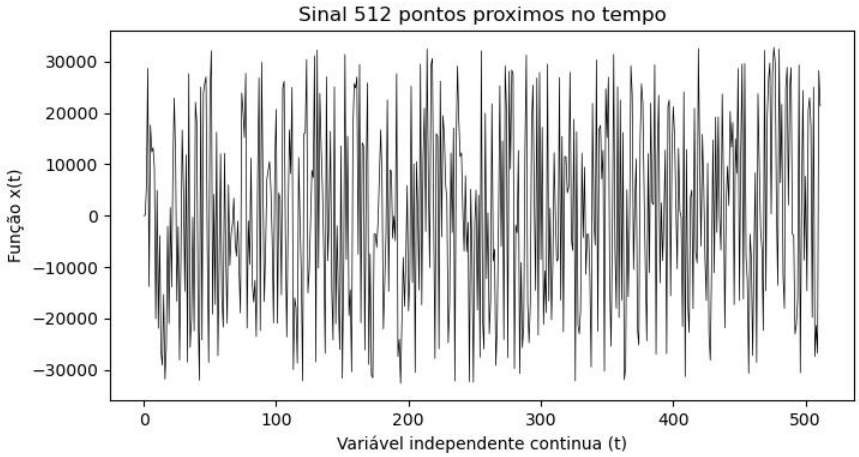


Figura 10 -Sinal Pseudoaleatório com 512 pontos interpolados

Terceira visualização o espectro de frequências através de FFT (*Fast Fourier Transform*) da mesma sequência de 8912 números pseudoaleatórios. Aparentemente sem predominância de faixas de frequências.

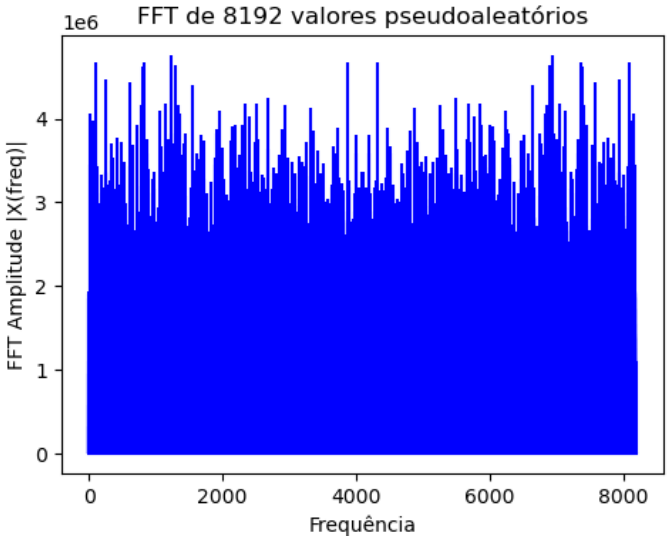


Figura 11 -Sinal Pseudoaleatório com 8192 pontos no domínio da frequência

Quarta visualização o histograma aproximado da distribuição de amplitude da referida geração de 8192 números pseudoaleatórios armazenados no formato de inteiros de 16 bits com sinal. Percebemos alguma tendência a distribuição uniforme com média pequena (0,3% da amplitude máxima).

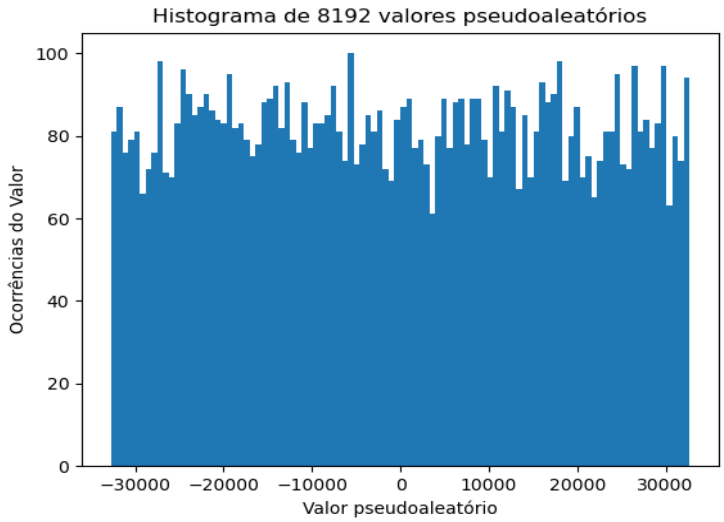


Figura 12 -Histograma de sinal Pseudoaleatório com 8192 pontos

A seguir o histograma da distribuição de amplitude da geração de 64K (2^{16}) números pseudoaleatórios de 16 bits com sinal em sequência. Percebemos uma distribuição quase perfeitamente uniforme, com média praticamente zero, quando a sequência de números gerados é longa.

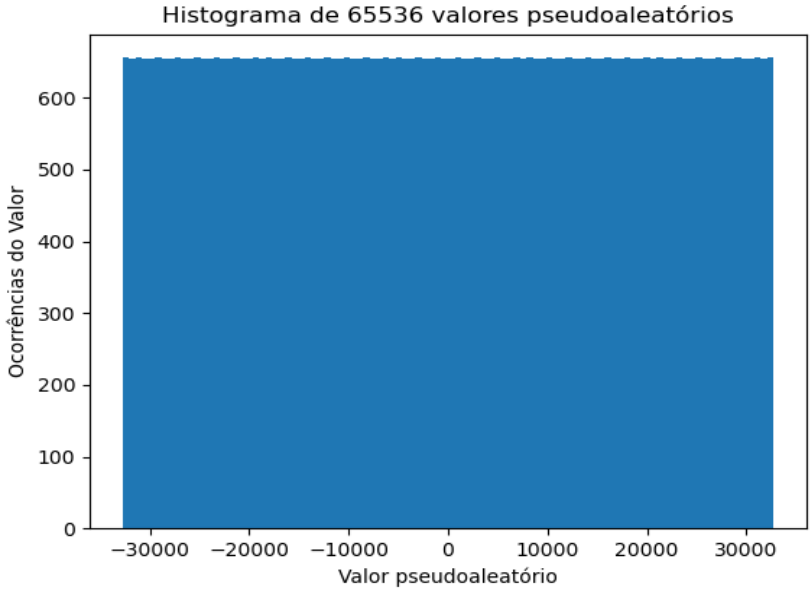


Figura 13 -Histograma de sinal Pseudoaleatório com 65536 pontos

O sinal pseudoaleatório visualmente é, portanto, satisfatório como simulacro de ruído branco embarcado em um microcontrolador para boa parte das aplicações.

CAPÍTULO III

TRANSFORMADA DISCRETA DE FOURIER – DFT

Este capítulo é dedicado a implementação e teste de função para cálculo numérico da Transformada Discreta de Fourier (DFT - *Discret Fourier Transform*) na plataforma de prototipagem Arduino UNO R4.

RÁPIDA REVISÃO DA TEORIA DA DFT

A Transformada Discreta de Fourier (*Discret Fourier Transform* - DFT) transforma uma sequência de N números $\{x_N\} = x_0, x_1, \dots, x_{N-1}$, geralmente ordenados no domínio do tempo, em uma sequência $\{X_K\}=X_0, X_1, \dots, X_{N-1}$, geralmente ordenada no domínio da frequência (Oppenheim, 2010).

Esta transformação é definida pela equação:

$$X[k] = \sum_{n=0}^{N-1} x[n].W_N^{kn} \qquad \because \qquad W_N^{kn} = e^{-i2\pi\frac{kn}{N}}$$

Onde:

- N é o número total de amostras (valor fixo)
- n é o índice da amostra sendo utilizada
- k é o índice da amplitude sendo calculada, variando de 0 a N-1
- $x[n]$ é a n-ésima amostra do sinal de entrada x_N
- $X[k]$ é a k-ésima amplitude do sinal de saída
- W_N^{kn} é a n-ésima parte do sinal referência W (número complexo) para a k-ésima saída X_k

Podemos interpretar a equação como uma correlação entre sinais, que é uma medida de similaridade entre o sinal de entrada amostrado $\{x_N\}$ com um sinal senoidal de referência {W} com frequência k e N amostras, obtendo as amplitudes não normalizadas X_k. Para normalizar basta dividir por $\sqrt{N}$.

A implementação simples de uma DFT como uma função de um programa com N amostras $\{x_N\}$ e N resultados $\{X_K\}$ implica em uma complexidade computacional com crescimento da ordem O(N^2) para o tempo de execução. Isto significa que o tempo de execução aumenta de forma quadrática com o tamanho N de amostras da DFT.

Utilizando a identidade de Euler na forma $e^{-iy} = \cos(y) - isen(y)$ obtemos a equação da DFT conforme abaixo.

$$X[k] = \sum_{n=0}^{N-1} x[n].\, e^{-i2\pi\frac{kn}{N}} =$$

$$= \sum_{n=0}^{N-1} x[n].\left[\cos\left(i2\pi\frac{kn}{N}\right) - isen\left(i2\pi\frac{kn}{N}\right)\right] =$$

$$= \sum_{n=0}^{N-1} x[n].\cos\left(i2\pi\frac{kn}{N}\right) - i\sum_{n=0}^{N-1} x[n].\,\mathrm{sen}\left(i2\pi\frac{kn}{N}\right) =$$

$$XR_K - iXI_K$$

No fim ficamos com $\boldsymbol{X_k = XR_k - iXI_k}$, aonde XR é a parte real e XI é a parte imaginária de X_k. Esta equação final nos permite calcular a parte real de forma numericamente independente da parte imaginária, que veremos adiante nos será muito útil.

Como exemplo imagine um sinal amostrado no tempo (128 amostras) conforme figura abaixo.

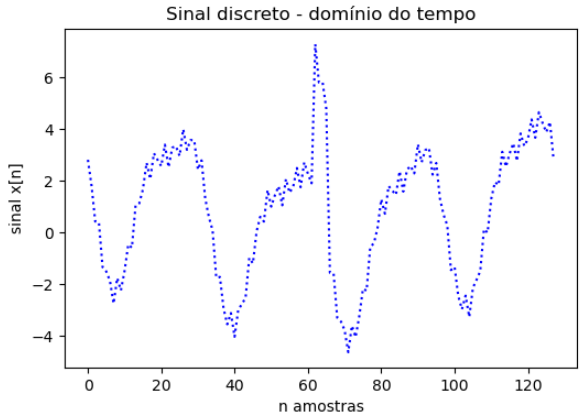


Figura 14 -Sinal discreto amostrado no domínio do tempo

Será muito difícil e impreciso analisar as componentes de frequência deste sinal utilizando apenas a análise matemática da Séries de Fourier. Se faz necessário um instrumento numérico eficaz e prático que transforme este sinal para o domínio da frequência. Daí vem a grande utilidade da Transformada Discreta de Fourier. No caso acima segue o sinal transformada na figura abaixo.

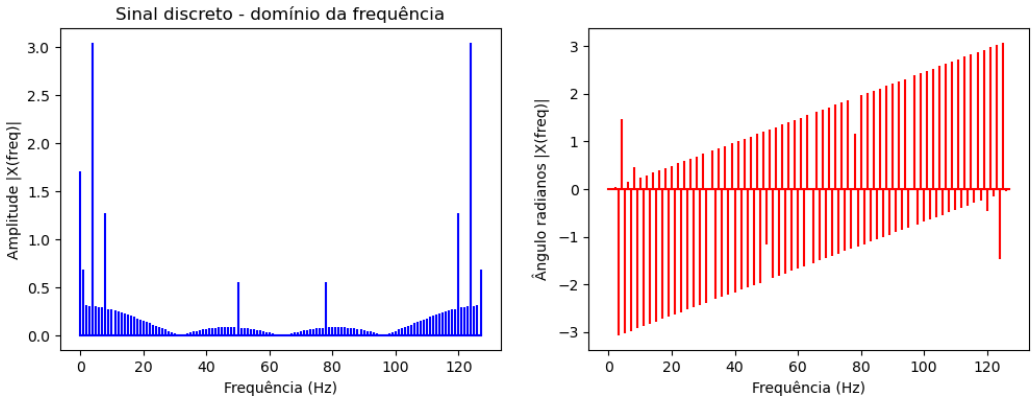


Figura 15 -Sinal discreto transformado para o domínio da frequência

Existe também a Transformada Inversa de Fourier (IDFT) definida pela equação (Oppenheim, 2010):

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k].e^{i2\pi\frac{kn}{N}}$$

A transformada inversa reconstitui o sinal no domínio do tempo $\{x_N\}$, por exemplo, a partir dos componentes dos componentes X_k no domínio da frequência.

Nestas transformadas as relações de domínios nem sempre são tempo x frequência, podendo ser espaço x frequência ou outras relações entre domínios. Isto fica conforme a necessidade da aplicação em questão.

IMPLEMENTAÇÃO DA DFT

Para testar as plataformas Arduino UNO R3 e R4 utilizaremos um programa em Linguagem C++ conforme a biblioteca referência do Arduino.

O programa de DFT calculará sete DFTs de tamanho variável, como 8,16,32,64,128,256 e 512 amostras, medindo o tempo gasto em microssegundos pelo microcontrolador para calcular as referidas DFT. O programa contém apenas variáveis globais e apenas três funções em C, duas padrão do Arduino (setup e loop) e uma da DFT. O resultado das medições de tempo será enviado pelo Arduino, via comunicação serial, para o ambiente IDE do Arduino e podem ser observadas no "serial monitor" do IDE.

O cálculo das DFTs é convenientemente feito com números reais utilizando a equação da DFT com cossenos e senos (identidade de Euler), calculando de forma independente, entretanto paralela, as partes real e imaginária da DFT. Fazemos isto pois o Arduino não manipula, na sua biblioteca básica, números complexos.

O programa exemplo didático desenvolvido tem apenas variáveis globais e está escrito de forma bem simples. Não tem otimizações de código ou algoritmo e utiliza apenas o básico do IDE do Arduino. Os dados do sinal de entrada x_N são gerados de forma randômica, ao invés de amostrados em um processo qualquer, apenas para fins de demonstração.

O LED do Arduino ficará piscando, indicando ao usuário que está ativo e com o programa de teste do DFT em execução. A parte do programa que faz isto é simples e bloqueia momentaneamente o programa.

```
// Demonstração da função DFT - Discret Fourier Transform
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.6
// Data da última revisão: 2025-05-26
// Autor: Paschoal Molinari
// Licença: MIT License

// Declaração de variáveis globais
unsigned long tempo_inicial, tempo_final;
const int tamanho_sinal = 512; // tamanho do sinal de entrada
int N = tamanho_sinal;
int x[tamanho_sinal]; // sinal de entrada
```

```
// partes real e imaginária do espectro do sinal no domínio da frequência
float Xr[tamanho_sinal], Xi[tamanho_sinal];

// função de setup inicial do ambiente
void setup() {
  // Mensagems iniciais do programa de teste de DFT
  Serial.begin(115200);
  delay(500);                  // esperando meio segundo
  Serial.println(" ");
  Serial.println(" Demonstração da DFT -Discret Fourier Transform");
  Serial.println(" versão basica");
  pinMode(LED_BUILTIN, OUTPUT);
}

// Função envia o arranjo "ARR" formatado (tipo inteiro e de tamanho N),
// para a linha serial, permitindo a monitoração
void dsp_print_int(int *ARR,int N){
  int n, ln;
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
    Serial.print(" ");
    Serial.print(ARR[n]);
    if (n < N - 1) {Serial.print(",");}
    if(++ln == 16) {
      ln = 0;
      Serial.println(" ");}
  }
  Serial.print(" ]");
  Serial.println(" ");
}

// Função envia o arranjo "ARR" formatado (tipo ponto flutuante e de tamanho
N),
// para a linha serial, permitindo a monitoração
void dsp_print_float(float *ARR,int N){
  int n, ln;
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
    Serial.print(" ");
    Serial.print(ARR[n],4);
    if (n < N - 1) {Serial.print(",");}
    if(++ln == 16) {
      ln = 0;
```

```
    Serial.println(" ");}
  }
  Serial.print(" ]");
  Serial.println(" ");
}

// Função de geração de números pseudoaleatórios
// PRNG - Pseudo-random Number Generator
// Gera número inteiro 16 bits pseudoaleatório na faixa [0,65535]
int dsp_prng(int prn_seed){
  // prn é o número pseudoaleatório a ser atualizado
  // valores iniciais da geração pseudoaleatória
  long a = 75,  c = 74;     // constantes a e c da função linear aX + c
  long m = 65537;          // constante divisor da função resto
  long prn = prn_seed;      // número pseudoaleatório
  // Geração do novo número pseudoaleatório (PRN)
  prn = a*prn + c;         // função linear aX + c
  prn = prn - m*(prn/m);    // Resto da divisão de prn por m
  return(prn);            // retorna novo número pseudoaleatório (PRN)
}

// função DFT - Discret Fourier Transform
// DFT básica
void dsp_dft_basic(int *x,float *Xr,float *Xi, int N) {
  float pi=3.14159265 , radianos;
  int k,n;   // k é a frequência , n é o n-ésimo valor do sinal amostrado
    // Loop do cálculo da DFT
    for(k=0; k<N; k++){
       Xr[k] = .0;    // zerando as variáveis da DFT para k atual
       Xi[k] = .0;
       // Calculando o somatorio de
       for(n=0; n<N; n++){
         radianos = (2*pi*k*n)/N;
         Xr[k] = Xr[k] + x[n]*cos(radianos);  // cálculo da parte real do X
         Xi[k] = Xi[k] - x[n]*sin(radianos);  // cálculo da parte imaginária do X
       }
    }
}

// Função de loop infinito
void loop() {
  // Parte principal do código que fica em loop indefinidamente
  int n;
  static int prng_seed= 1;
```

```
 Serial.println(" ");           // pula linha
 Serial.println("Piscando o LED - DFT basica");  // piscando o LED como
indicativo de funcionamento
 digitalWrite(LED_BUILTIN, HIGH);   // Ligando o LED (HIGH is the voltage level)
 delay(5000);                  // esperando um segundo
 digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
 delay(1000);                  // esperando um segundo

 // Gera um sinal de entrada x com N números pseudoaleatórios
 x[0] = dsp_prng(prng_seed);
 for(n=1; n<tamanho_sinal; n++){
    x[n] = dsp_prng(x[n-1]);
 }
 prng_seed = x[tamanho_sinal-1];

 // Loop de amostras (N) com N variando no conjunto {8,16,32,64,128,256,512}
 for(N = 8; N < tamanho_sinal+1; N = N*2){
  Serial.print("N= ");
  Serial.print(N);
  tempo_inicial = micros();

  dsp_dft_basic(x,Xr,Xi,N); // chamando a função DFT

  // Medição do tempo de execução da DFT em microsegundos
  tempo_final = micros() - tempo_inicial;
  Serial.print("  tempo DFT microsec ");
  Serial.println(tempo_final);
 }

 N = 128;
 dsp_dft_basic(x,Xr,Xi,N); // chamando a função DFT

 // Envia o sinal de entrada x para a linha serial apenas para teste
 Serial.println(" ");
 Serial.print(" Sinal de entrada - inteiro");
 Serial.println(N);
 dsp_print_int(x,N); // Função de impressão de N dados inteiros

 Serial.println(" ");
 Serial.print(" Sinal de saída - Real ");
 Serial.println(N);
 dsp_print_float(Xr,N); // Função de impressão de N dados inteiros

 Serial.println(" ");
 Serial.print(" Sinal de saída - Imaginário ");
```

```
 Serial.println(N);
 dsp_print_float(Xi,N); // Função de impressão de N dados inteiros
 Serial.println(" ");
}
```

IMPLEMENTAÇÃO OTIMIZADA DA DFT

Uma versão otimizada da DFT é apresentada abaixo para reduzir o tempo de execução. Basicamente o que se melhorou foram dois itens:

a) Limitar os valores máximos dos radianos em 2 PI (2π), pois as funções seno e cosseno se repetem a cada 2 Pi radianos, e como são geralmente calculadas por expansão de séries infinitas o seu tempo de execução cresce sensivelmente quanto maior for o número.

b) Calcular seno a partir do valor do cosseno usando sua relação trigonométrica básica que é $1 = sen^2 + cos^2$, portanto $sen = \sqrt{1 - cos^2}$.

Estas otimizações não intuitivas reduzem substancialmente o tempo de execução. No caso do Arduino UNO R4 de 1,5 a 12 vezes sem degradação perceptível da exatidão dos cálculos.

```
// Programa demonstração da função DFT - Discret Fourier Transform
// Para plataforma Arduino UNO R4
// versão otimizada de DFT
// Arduino IDE 2.3.6
// Data da última revisão: 2025-05-26
// Autor: Paschoal Molinari
// Licença: MIT License

// Declaração de variáveis globais
unsigned long tempo_inicial, tempo_final;
const int tamanho_sinal = 512; // tamanho do sinal de entrada
int N = tamanho_sinal;
int x[tamanho_sinal]; // sinal de entrada
// partes real e imaginária do espectro do sinalno domínio da frequência
float Xr[tamanho_sinal], Xi[tamanho_sinal];
```

```
// função de setup inicial do ambiente
void setup() {
  // Mensagems iniciais do programa de teste de DFT
  Serial.begin(115200);             // setando a velocidade da comunicação serial
  delay(500);                   // esperando meio segundo
  Serial.println(" ");
  Serial.println(" Demonstração da DFT - Discret Fourier Transform");
  Serial.println(" versão otimizada");
  pinMode(LED_BUILTIN, OUTPUT);
}

// Função envia o arranjo "ARR" formatado (tipo inteiro e de tamanho N),
// para a linha serial, permitindo a monitoração
void dsp_print_int(int *ARR,int N){
  int n, ln;
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
    Serial.print(" ");
    Serial.print(ARR[n]);
    if (n < N - 1) {Serial.print(",");}
    if(++ln == 16) {
      ln = 0;
      Serial.println(" ");}
  }
  Serial.print(" ]");
  Serial.println(" ");
}

// Função envia o arranjo "ARR" formatado (tipo ponto flutuante e de tamanho
N),
// para a linha serial, permitindo a monitoração
void dsp_print_float(float *ARR,int N){
  int n, ln;
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
    Serial.print(" ");
    Serial.print(ARR[n],4);
    if (n < N - 1) {Serial.print(",");}
    if(++ln == 16) {
      ln = 0;
      Serial.println(" ");}
  }
```

```
 Serial.print(" ]");
 Serial.println(" ");
}

// Função de geração de números pseudoaleatórios
// PRNG - Pseudo-random Number Generator
// Gera número inteiro 16 bits pseudoaleatório na faixa [0,65535]
int dsp_prng(int prn_seed){
 // prn é o número pseudoaleatório a ser atualizado
 // valores iniciais da geração pseudoaleatória
 long a = 75,  c = 74;     // constantes a e c da função linear aX + c
 long m = 65537;          // constante divisor da função resto
 long prn = prn_seed;      // número pseudoaleatório
 // Geração do novo número pseudoaleatório (PRN)
 prn = a*prn + c;         // função linear aX + c
 prn = prn - m*(prn/m);    // Resto da divisão de prn por m
 return(prn);            // retorna novo número pseudoaleatório (PRN)
}

// função DFT - Discret Fourier Transform
// Versão com duas otmizações: limitação de intervalo radiano [0,2PI),
//                    relação trigonométrica seno-cosseno
void dsp_dft_opt(int *x,float *Xr,float *Xi, int N) {
 float cosseno, seno, _2PI =  6.2831853 , pre_radianos, k_N;
 int k,n;

 // Loop do cálculo da DFT
 for(k=0; k<N; k++){
  Xr[k] = .0;                  // zerando as variaveis Xr e Xi da DFT
  Xi[k] = .0;
  k_N = float(k)/N;              // precalculando k/N
  for(n=0; n<N; n++){
   pre_radianos = k_N*n;
   pre_radianos -= long(pre_radianos); // limitando 0<= pre_radianos <1
   cosseno = cos(_2PI*pre_radianos);   // calculando radianos e o cosseno
   Xr[k] += x[n]*cosseno;          // cálculo da parte real do X
   seno  = sqrt(1-sq(cosseno));      // obtendo o seno a partir do cosseno
   if (pre_radianos < 0.5) {       // cálculo da parte imaginária do X
    Xi[k] -= x[n]*seno;}
   else {
    Xi[k] += x[n]*seno;}
  }
 }
}
```

```
// Função de loop infinito
void loop() {
 // Parte principal do código que fica em loop indefinidamente:
 int n;
 static int prng_seed= 1;

 Serial.println(" ");            // pula linha
 Serial.println("Piscando o LED - DFT otimizada");  // piscando o LED como
indicativo de funcionamento
 digitalWrite(LED_BUILTIN, HIGH);   // Ligando o LED (HIGH is the voltage level)
 delay(5000);                       // esperando um segundo
 digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
 delay(1000);                       // esperando um segundo

 // Gera um sinal de entrada x com N números pseudoaleatórios
 x[0] = dsp_prng(prng_seed);
 for(n=1; n<tamanho_sinal; n++){
    x[n] = dsp_prng(x[n-1]);
 }
 prng_seed = x[tamanho_sinal-1];

 // Loop de amostras (N) com N variando no conjunto {8,16,32,64,128,256,512}
 for(N = 8; N < tamanho_sinal+1; N = N*2){
  Serial.print("N= ");
  Serial.print(N);
  tempo_inicial = micros();

  dsp_dft_opt(x,Xr,Xi,N); // chamando a função DFT otimizada

  // Medição do tempo de execução da DFT em microsegundos
  tempo_final = micros() - tempo_inicial;
  Serial.print("  tempo DFT microsec ");
  Serial.println(tempo_final);
 }

  N = 128;
 dsp_dft_opt(x,Xr,Xi,N); // chamando a função DFT

 // Envia o sinal de entrada x para a linha serial apenas para teste
 Serial.println(" ");
 Serial.print(" Sinal de entrada - inteiro");
 Serial.println(N);
 dsp_print_int(x,N); // Função de impressão de N dados inteiros

 Serial.println(" ");
```

```
  Serial.print(" Sinal de saída - Real ");
  Serial.println(N);
  dsp_print_float(Xr,N); // Função de impressão de N dados inteiros

  Serial.println(" ");
  Serial.print(" Sinal de saída - Imaginário ");
  Serial.println(N);
  dsp_print_float(Xi,N); // Função de impressão de N dados inteiros
  Serial.println(" ");
}
```

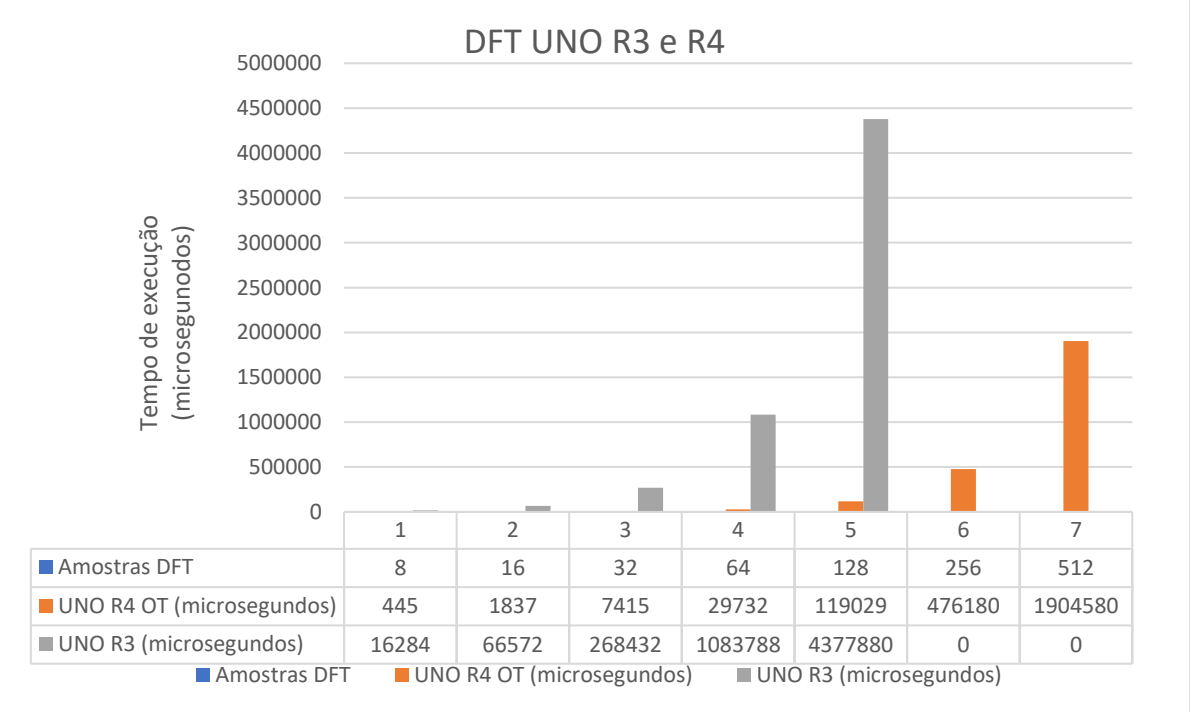
RESULTADOS DOS TESTES

Executamos o programa no Arduino UNO R3 e posteriormente no Arduino UNO R4 nas versões normal e otimizada, e obtivemos as medições de tempo abaixo no monitor do Arduino IDE. O Arduino UNO R3 não teve memória RAM suficiente para executar o programa com N = 256 amostras ou mais. Agregando estas medições de tempo em uma única tabela temos o resumo abaixo com tempo de execução em microssegundos para processamento de N amostras da DFT.

DFT (N)	Arduino UNO R3 (microsegundos)	Arduino UNO R4 (microsegundos)	Arduino UNO R4 OT Otimizado (microsegundos)	Ganho R3/R4 OT	Ganho R4/R4 OT
8	16.284	662	445	37	1,5
16	66.572	2.803	1.837	37	1,5
32	268.432	11.430	7.415	37	1,6
64	1.083.788	104.650	29.732	37	3,6
128	4.377.880	815.569	119.029	37	7
256	ND	4.598.355	476.180	ND	9,8
512	ND	22.185.840	1.904.580	ND	11,8

ND = Não Disponível

Pelos dados obtidos o Arduino UNO R4 foi 37 vezes mais rápido que o Arduino UNO R3 neste teste de DFT. Para o Arduino UNO R3 e UNO R4 temos o gráfico abaixo que mostra um crescimento quadrático do tempo (microssegundos) com o número de amostras.



	1	2	3	4	5	6	7
Amostras DFT	8	16	32	64	128	256	512
UNO R4 OT (microsegundos)	445	1837	7415	29732	119029	476180	1904580
UNO R3 (microsegundos)	16284	66572	268432	1083788	4377880	0	0

Figura 16 -Comparação dos tempos de execução de DFTs no UNO R3 e UNO R4

DISCUSSÃO E CONCLUSÃO

O comportamento de crescimento quadrático do tempo de execução dos Arduinos UNO R3 e R4 com o número de amostras N reflete a complexidade O(N^2) do algoritmo de DFT conforme previsto. O tempo de execução deve diminuir significativamente utilizando algoritmos otimizados como a Fast Fourier Transform (FFT), otimização de funções trigonométricas, series de Taylor, Chebyshev ou uso de tabelas, com especial atenção a necessidade de precisão da aplicação. Lembrando que mesmo algumas implementações da FFT necessitam utilizar as vezes no seu núcleo uma DFT, portanto é importante uma ótima implementação deste algoritmo.

O Arduino UNO R4 otimizado apresentou um ganho típico de desempenho de 37 vezes o desempenho do Arduino UNO R3 quando executando as DFTs, sem perda perceptível de precisão. Este é um ganho muito significativo e já esperado. A memória do Arduino UNO R4, por utilizar um microcontrolador avançado e com

mais memória, permite a execução de aplicações mais complexas e/ou com maior quantidade de dados como a DFT de 512 amostras (N).

A plataformas Arduino UNO R4, com as devidas otimizações, provavelmente podem embarcar algumas funções de processamento de sinais de baixa frequência comuns em aplicações de controle e automação. A facilidade de programação e prototipação é um ganho no desenvolvimento de soluções. Estas plataformas também têm um bom potencial didático para conceitos avançados devido a sua disseminação, interfaces e facilidade de uso.

Incentivamos ao leitor que altere o programa didático deste capítulo das seguintes formas:

- Colocar o LED para piscar sem bloquear o programa como um todo;
- Aplicar sinais de formatos e frequências diferentes em x e observar o resultado da DFT
- Fazer aquisição de dados e colocar na variável x o sinal de entrada;
- Tentar otimizar as operações da DFT.

CAPÍTULO IV

FFT - TRANSFORMADA RÁPIDA DE FOURIER

A Transformada Rápida de Fourier, mais conhecida como FFT (*Fast Fourier Transform*), é um algoritmo otimizado para o cálculo rápido da DFT (*Discret Fourier Transform*). Foi criada, na sua forma mais comum, em 1965 por James Cooley e John Tukey, mais conhecido como algoritmo *Cooley-Tukey*.

A FFT é um algoritmo no estilo "dividir para conquistar", que reescreve o algoritmo original da DFT de tamanho N, de tal forma que $N = N_1 N_2$, onde temos N_1 DFTs de tamanho N_2 calculadas recursivamente. Com isto reduz-se a complexidade computacional com crescimento da ordem O($NLogN$) para o tempo de execução, reduzindo assim enormemente este tempo e viabilizando o uso desta forma de DFT em uma série de aplicações no mundo moderno.

Como o algoritmo *Cooley-Tukey* divide a DFT em pequenas DFTs, podemos arbitrariamente combinar este algoritmo com outros algoritmos de DFT.

Para entendermos o algoritmo da FFT primeiro relembremos a equação da DFT original conforme segue.

$$X[k] = \sum_{n=0}^{N-1} x[n].\, e^{-i2\pi\frac{kn}{N}}$$

Onde:

- N é o número total de amostras (valor fixo)
- n é o índice da amostra sendo utilizada
- k é o índice da amplitude sendo calculada, variando de 0 a N-1
- $x[n]$ é a n-ésima amostra do sinal de entrada x_N
- $X[k]$ é a k-ésima amplitude do sinal de saída
- W_N^{kn} é a n-ésima parte do sinal referência W (número complexo) para a k-ésima saída X_k

Esta equação pode ser reescrita como a soma de dois somatórios de tamanho N/2, uma apenas para os índices n pares fazendo n = 2m, e outra apenas para índices n ímpares fazendo n = 2m+1. Portanto:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m].e^{-i2\pi\frac{k(2m)}{N}} + \sum_{m=0}^{N/2-1} x[2m+1].e^{-i2\pi\frac{k(2m+1)}{N}}$$

A parte impar da equação pode ter a sua exponencial expandida, e a parte não dependente da variável m pode ser reposicionada como um multiplicador do somatório, ficando:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m].e^{-i2\pi\frac{k(2m)}{N}} + e^{-i2\pi\frac{k}{N}}.\sum_{m=0}^{N/2-1} x[2m+1].e^{-i2\pi\frac{k(2m)}{N}}$$

Chamando o somatório de índices pares de $\boldsymbol{E}$ (Even), o somatório de índices ímpares de $\boldsymbol{O}$ (Odd), e a exponencial independente de m de $\boldsymbol{W_n^k}$ obtemos a configuração reduzida abaixo.

$$X[k] = E + W_n^k \,.O$$

A parte mais interessante vem agora. Devido a periodicidade da exponencial complexa podemos calcular um $X[k + N/2]$, ou seja, uma frequência k na metade superior do espectro, usando os mesmos três elementos E, O e W de $X[k]$, ficando a equação na forma:

$$X[k + N/2] = E - W_n^k \,.O$$

Com isto o conjunto de equações fica:

$$\boldsymbol{X[k] = E + W_n^k \,.\, O}$$

$$\boldsymbol{X[k + N/2] = E - W_n^k \,.\, O}$$

Onde:

$$E = \sum_{m=0}^{N/2-1} x[2m].\, e^{-i2\pi\frac{k(2m)}{N}}$$

$$O = \sum_{m=0}^{N/2-1} x[2m+1].\, e^{-i2\pi\frac{k(2m)}{N}}$$

$$W_n^k = e^{-i2\pi\frac{k}{N}}$$

Assim ao calcularmos $X[k]$ podemos rapidamente calcular $X[k + N/2]$ reaproveitando os termos E, W_n^k e O previamente calculados. Além disto as partes pares (E) e ímpares (O) podem, por sua vez, serem subdividas recursivamente em subpartes menores pares e ímpares. Isto resulta em um grande reaproveitamento de cálculos e consequentemente em uma redução extremamente significativa do tempo de execução do algoritmo FFT, pois a complexidade computacional do algoritmo FFT passa a ser O($NLogN$).

Existe uma representação gráfica bem difundida das equações da FFT na forma de uma borboleta (BT – Butterfly), mostrando o fluxo do processamento dos sinais da entrada x[n] para os sinais de saída X[k].

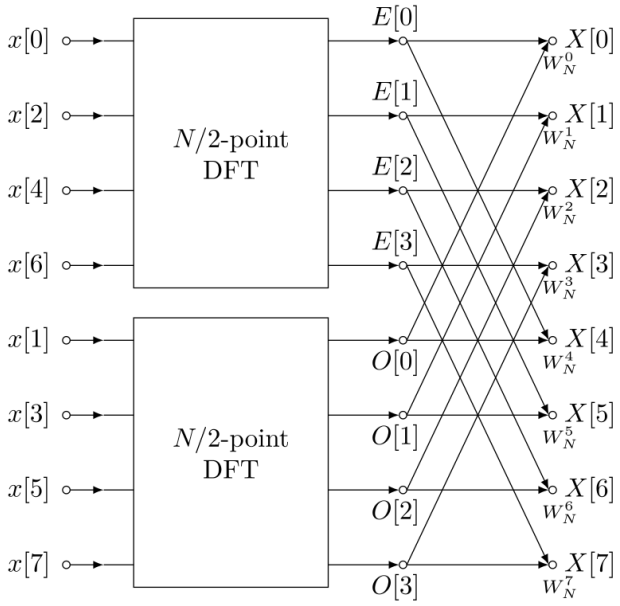


Figura 17 -Borboleta de FFT com oito saídas (Fonte: Wikipedia)

O impacto da FFT foi tão grande no mundo moderno, em aplicações práticas nas áreas mais diversas das Ciências e Engenharia, que foi incluído na lista dos 10 mais importantes Algoritmos do Século XX pela revista “IEEE Computing in Science & Engineering”.

No programa a seguir temos a implementação da FFT para o Arduino UNO R4.

Este programa é um exemplo didático e contém no seu núcleo uma DFT de 16 entradas x[n]. Isto mostra que a FFT é uma otimização algorítmica extremamente relevante e totalmente compatível com a DFT.

É utilizado o algoritmo de *bit-reverse permutation* para reordenar o sinal de entrada x[n] antes do processamento pela FFT, garantindo que a saída X[n] sai ordenada de forma crescente com a frequência.

```
// Demonstração da função FFT - Fast Fourier Transform
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.6
// Data da última revisão: 2025-05-26
```

```
// Autor: Paschoal Molinari
// Licença: MIT License

// Declaração de variáveis globais
unsigned long tempo_inicial, tempo_final;
const int tamanho_sinal = 1048; // tamanho maximo do sinal de entrada
int N = tamanho_sinal; // tamanho do sinal de entrada multiplo de 2
int x[tamanho_sinal]; // sinal de entrada
// partes real e imaginária do sinal no domínio da frequência
float Xr[tamanho_sinal], Xi[tamanho_sinal];

// função de setup inicial do ambiente
void setup() {
  // Mensagems iniciais do programa de teste de DFT
  Serial.begin(115200);           // setando a velocidade da comunicação serial
  delay(500);                     // esperando meio segundo
  Serial.println(" ");
  Serial.println(" ");
  Serial.println(" Demonstração de tempo - FFT - Fast Fourier Transform");
  Serial.println(" versão com Bit-Reverse e DFT");
  pinMode(LED_BUILTIN, OUTPUT);
}

// Função envia o arranjo "ARR" formatado (tipo inteiro e de tamanho N),
// para a linha serial, permitindo a monitoração
void dsp_print_int(int *ARR,int N){
  int n, ln;
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
    Serial.print(" ");
    Serial.print(ARR[n]);
    if (n < N - 1) {Serial.print(",");}
    if(++ln == 16) {
      ln = 0;
      Serial.println(" ");}
  }
  Serial.print(" ]");
  Serial.println(" ");
}

// Função envia o arranjo "ARR" formatado (tipo ponto flutuante e de tamanho N),
// para a linha serial, permitindo a monitoração
void dsp_print_float(float *ARR,int N){
  int n, ln;
```

```
  Serial.print("[ ");
  ln = 0;
  for(n=0;n<N;n++){
   Serial.print(" ");
   Serial.print(ARR[n],4);
   if (n < N - 1) {Serial.print(",");}
   if(++ln == 16) {
    ln = 0;
    Serial.println(" ");}
  }
  Serial.print(" ]");
  Serial.println(" ");
}

// Função de geração de números pseudoaleatórios
// PRNG - Pseudo-random Number Generator
// Gera número inteiro 16 bits pseudoaleatório na faixa [0,65535]
int dsp_prng(int prn_seed){
  // prn é o número pseudoaleatório a ser atualizado
  // valores iniciais da geração pseudoaleatória
  long a = 75,  c = 74;     // constantes a e c da função linear aX + c
  long m = 65537;          // constante divisor da função resto
  long prn = prn_seed;      // número pseudoaleatório
  // Geração do novo número pseudoaleatório (PRN)
  prn = a*prn + c;          // função linear aX + c
  prn = prn - m*(prn/m);    // Resto da divisão de prn por m
  return(prn);              // retorna novo número pseudoaleatório (PRN)
}

// Gerador de sinal senoidal
void dsp_sig_gen(int* x,int N_samples,float amplitude,float frequency,float
phase){
  float _2PI =  6.283185307, k;
  int n;

  k = _2PI*frequency/N_samples;
  for(n = 0; n < N_samples; n++){
    x[n] += int(amplitude*sin(k*n + phase));
  }
}

// função DFT (Discret Fourier Transform) com espaçamento
// Versão com duas otmizações: limitação de intervalo radiano [0,2PI),
//                    relação trigonométrica seno-cosseno
void dsp_DFT_space(int *x,float *Xr,float *Xi,int n_start,int n_space,int N,int kX) {
```

```
 float cosseno, seno, _2PI =  6.283185307 , pre_radianos, k_N;
 int k,n,n_x;

 // Loop do cálculo da DFT
 for(k=0; k<N; k++){
  Xr[kX] = .0;                  // zerando as variaveis Xr e Xi da DFT
  Xi[kX] = .0;
  n_x = n_start;
  k_N = float(k)/N;             // precalculando k/N
  for(n=0; n < N; n++) {
   pre_radianos = k_N*n;
   pre_radianos -= long(pre_radianos); // limitando pre_radianos no intervalo
[0.0,1.0)

   // Calculando cosseno e seno
   cosseno = cos(_2PI*pre_radianos);   // calculando radianos e o cosseno
   seno  = sqrt(1-sq(cosseno));      // obtendo o seno a partir do cosseno
   if (pre_radianos > 0.5) { seno = -seno;}  // ajuste do sinal do seno conforme o
radiano

   // Calculando o Xk
   Xr[kX] += x[n_x]*cosseno;          // cálculo da parte real Xr do X
   Xi[kX] -= x[n_x]*seno;             // cálculo da parte imaginária Xi do X
   n_x += n_space;
  }
  kX++;
 }
}

// função FFT - Fast Fourier Transform
// Versão com Bit-Reverse e DFTs com espaçamento
//
void dsp_FFT_br(int *x,float *Xr,float *Xi, int N) {
 float _2PI =  6.283185307 , pre_radianos;
 float x_even_r, x_even_i, x_odd_r, x_odd_i, Wk_N_r, Wk_N_i, Wx_odd_r,
Wx_odd_i;
 int k,n,kr,k_dist,space,levels;
 int ft_N=16; // minimal Fourier Transform size
 // Bit-Reverse list
 int BR_i, BR_list_size = 64;
 int BR_list[] = {0, 32, 16, 48,  8, 40, 24, 56,  4, 36, 20, 52, 12, 44, 28, 60,  2,
    34, 18, 50, 10, 42, 26, 58,  6, 38, 22, 54, 14, 46, 30, 62,  1, 33,
    17, 49, 9, 41, 25, 57,  5, 37, 21, 53, 13, 45, 29, 61,  3, 35, 19,
    51, 11, 43, 27, 59, 7, 39, 23, 55, 15, 47, 31, 63};
```

```
 if(N == ft_N){
  Serial.println(" DFT only ");
  dsp_DFT_space(x,Xr,Xi,0,1,ft_N,0); // Somente uma DFT - Fourier Transform
  return; }

 if(N > ft_N*BR_list_size){  // FFT maior que o possível para computar
  return; }

 // Calculando os níveis de borboletas FFT Butterfly após FTs iniciais
 space = N/ft_N;
 levels = space;
 for(n=0; levels > 1;){
  levels = levels/2;
  n++;}
 levels = n;

 for(k=0; k < space; k++){
  BR_i = k*(BR_list_size/space);
  dsp_DFT_space(x,Xr,Xi,BR_list[BR_i],space,ft_N,k*ft_N);  //   calculando DFTs
iniciais
 }

 // Loop do cálculo da FFT
 while(space > 1){
  space = space/2;
  k_dist = 0;
  // Calculating a BT level
  for(n=0; n < space; n++){
     for(k=0; k < ft_N; k++) {
      kr = k_dist + k;
      // Calculando cosseno (parte real) e seno (parte imaginária)
      pre_radianos = float(k)/(2*ft_N);
      pre_radianos -= long(pre_radianos);
      Wk_N_r = cos(_2PI*pre_radianos);     // calculando radianos e a parte real
(cosseno)
      Wk_N_i = sqrt(1-sq(Wk_N_r));         // calculando valor da parte imaginária
(seno) a partir da parte real (cosseno)
      if (pre_radianos < 0.5) { Wk_N_i = - Wk_N_i;}  // ajuste do sinal da parte
imaginária (seno) conforme o radiano
      x_even_r = Xr[kr];
      x_even_i = Xi[kr];
      x_odd_r = Xr[kr+ft_N];
      x_odd_i = Xi[kr+ft_N];
      Wx_odd_r = Wk_N_r*x_odd_r - Wk_N_i*x_odd_i;
      Wx_odd_i = Wk_N_r*x_odd_i + Wk_N_i*x_odd_r;
```

```
        Xr[kr] = x_even_r + Wx_odd_r;
        Xi[kr] = x_even_i + Wx_odd_i;
        Xr[kr+ft_N] = x_even_r - Wx_odd_r;
        Xi[kr+ft_N] = x_even_i - Wx_odd_i;
        }
     k_dist += 2*ft_N;
   }
   ft_N = 2*ft_N;
 }
}

// FUNÇÃO PRINCIPAL - LOOP INFINITO
// Parte principal do código que fica em loop indefinidamente
void loop() {
 int n, ln;
 static long prng_seed= 1;

 Serial.println(" ");            // pula linha
 Serial.println("Piscando o LED");   // piscando o LED como indicativo de funcionamento
 digitalWrite(LED_BUILTIN, HIGH);   // Ligando o LED (HIGH is the voltage level)
 delay(2000);                  // esperando um segundo
 digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
 delay(2000);                  // esperando um segundo

 /* Gera um sinal de entrada x com N números pseudoaleatórios
 for(n=0; n<tamanho_sinal; n++){
    x[n] = int(prng_seed - 32768);
    prng_seed = DSP_prng(prng_seed);
 } */

 // Zerando o sinal x[]
 for(n=0; n<tamanho_sinal; n++){
    x[n] = 0;
 }

 dsp_sig_gen(x,tamanho_sinal,20.0,10.0,1.5707963268);
 dsp_sig_gen(x,tamanho_sinal,50.0,5.0,-1.0);
 dsp_sig_gen(x,tamanho_sinal,5.0,35.0,-2.0);

 // TESTE DE TEMPO DE EXECUÇÃO DE DFT
 // Loop de DFTs com (N) amostras, N variando no conjunto {8,16,32,64,128,256,512,1024}
 for(N = 32; N < tamanho_sinal+1; N = N*2){
  Serial.print("N= ");
```

```
  Serial.print(N);
  tempo_inicial = micros();

  dsp_FFT_br(x, Xr, Xi, N); // chamando a função FFT

  // Medição do tempo de execução da DFT em microsegundos
  tempo_final = micros() - tempo_inicial;
  Serial.print("  tempo FFT microsec ");
  Serial.println(tempo_final);
 }

 // TESTE DE FUNCIONALIDADE
 N = 128;
 // Zerando o sinal x[]
 for(n=0; n<N; n++){
    x[n] = 0;
 }
 // gerando sinal de teste
 dsp_sig_gen(x,N,20.0,10.0,1.5707963268);
 dsp_sig_gen(x,N,50.0,5.0,-1.0);
 dsp_sig_gen(x,N,5.0,35.0,-2.0);

 // Teste de funcionalidade
 Serial.println(" ");
 Serial.println(" Demonstração de funcionalidade ");
 Serial.println(" ");

 dsp_FFT_br(x, Xr, Xi, N);
 // dsp_DFT_space(x,Xamp,Xfas,0,1,N,0); // chamando a função DFT
dsp_DFT_space(int *x,float *Xr,float *Xi,int nx,int n_space,int N,int kX)

 // Envia o sinal de entrada x para a linha serial apenas para teste
 Serial.println(" ");
 Serial.print(" Sinal de entrada - inteiro");
 Serial.println(N);
 dsp_print_int(x,N); // Função de impressão de N dados inteiros

 Serial.println(" ");
 Serial.print(" Sinal de saída - Real ");
 Serial.println(N);
 dsp_print_float(Xr,N); // Função de impressão de N dados inteiros

 Serial.println(" ");
 Serial.print(" Sinal de saída - Imaginário ");
 Serial.println(N);
```

```
  dsp_print_float(Xi,N); // Função de impressão de N dados inteiros
  Serial.println(" ");

  // retardo para vizualização
  delay(30000);
}
```

RESULTADOS DOS TESTES

Testes foram realizados com FFTs variando de 32 a 1024 amostras. Agregando estas medições de tempo em uma única tabela temos o resumo abaixo. O tempo de execução é em microssegundos para processamento de N amostras de DFT e FFT.

DFT (N)	DFT UNO R3 (µsec)	DFT UNO R4 (µsec)	DFT Otimizado UNO R4 (µsec)	FFT UNO R4 (µsec)	Ganho DFT R3/ FFT R4	Ganho DFT R4/FFT R4
8	16.284	662	445	ND	ND	ND
16	66.572	2.803	1.837	ND	ND	ND
32	268.432	11.430	7.415	3.340	80,4	3,4
64	1.083.788	104.650	29.732	6.825	158,8	15,3
128	4.377.880	815.569	119.029	14.009	312,5	58,2
256	ND	4.598.355	476.180	28.779	ND	159,8
512	ND	22.185.840	1.904.580	59.129	ND	375,2
1024	ND	ND	ND	121.460	ND	ND

ND = Não Disponível

Pelos dados obtidos a FFT no Arduino UNO R4 foi até 375 vezes mais rápido que a DFT na mesma plataforma.

DISCUSSÃO E CONCLUSÃO

O comportamento de crescimento da *Fast Fourier Transform* (FFT) foi ligeiramente acima do linear, refletindo complexidade O($NLog(N)$) do referido algoritmo. Isto é bem abaixo do crescimento quadrático do tempo de execução das DFTs nos Arduinos UNO R3 e R4 que refletem a complexidade O(N^2) do algoritmo de DFT.

O tempo de execução diminuiu significativamente com o uso da FFT, viabilizando sua aplicação prática no Arduino UNO R4 para a comunidade *Maker*. A FFT passa a ser o padrão ouro para o processamento de sinais nesta plataforma.

CAPÍTULO V

IMPLEMENTAÇÃO DE FILTRO MÉDIA MÓVEL

Neste capítulo é feito uma revisão teórica do filtro passa baixa Média Móvel, bem como, de forma didática, sua otimizada implementação e teste, em linguagem C++ no Arduino UNO R4.

INTRODUÇÃO E MOTIVAÇÃO

Este capítulo é dedicado a implementação e teste, de forma didática, de uma eficiente rotina de filtro FIR (*Finite Impulse Response*) passa baixa do tipo Média Móvel no Arduino UNO R4.

RÁPIDA REVISÃO DA TEORIA DA MÉDIA MÓVEL

Em estatística, uma Média Móvel (MM) é um Estimador para analisar pontos de dados criando uma série de médias de diferentes seleções do conjunto completo de dados. Em processamento digital de sinais a Média Móvel é um tipo de filtro de resposta ao impulso finito (FIR- *Finite Impulse Response*). As variações incluem: média móvel aritmética (MMA), média móvel acumulativa, e média móvel ponderada (MMP) entre outras.

A Média Móvel Aritmética de uma sequência de N elementos é definida pela equação discreta:

$$y[i] = \frac{1}{N} \sum_{j=0}^{N-1} x[i+j]$$

Onde:

- N é o número de amostras (valor fixo) a serem usados para calcular a média móvel
- x[i+j] é a j-ésima amostra do sinal de entrada x a partir da amostra atual x[i]
- y[i] é o i-ésima amplitude do sinal de saída

Como exemplo abaixo segue o cálculo de y[30] com N = 4 usando a equação da MM:

$$y[30] = \frac{1}{4}(x[30] + x[31] + x[32] + x[33])$$

A implementação simples de uma MM como uma função de um programa com N amostras x[i] e N resultados y[i] implica em uma complexidade O(N) para o tempo de execução. Isto significa que o tempo de execução aumenta de forma linear com o tamanho N da Média Móvel.

Existe uma otimização desta equação que pode eventualmente acelerar significativamente a computação do valor da MM para grandes valores de N.

$$y[i] = y[i-1] + \frac{x[i+N-1]}{N} - \frac{x[i-1]}{N}$$

Podemos também adequar a forma da equação para os sistemas computacionais trabalharem em tempo real com os dados de entrada atual x[i] e passado x[i-N].

$$y[i] = y[i-1] + \frac{x[i]}{N} - \frac{x[i-N]}{N}$$

Estas equações discretas otimizadas assumem que existe um y[i-1] previamente calculado. O ganho que se tem depende do tamanho de N, quanto maior o N maior o ganho no tempo da computação da MM. Adicionalmente a complexidade não aumenta com o tamanho de N. Esta última versão nos permite calcular y[30] com M = 8 da seguinte forma:

$$y[30] = y[29] + \frac{x[30]}{8} - \frac{x[22]}{8}$$

Não só a média móvel é uma solução equilibrada para uma ampla gama de aplicações, como é uma ótima solução para a presença de ruído branco mantendo resposta rápida ao degrau unitário. Pela rapidez de execução, a sua versão otimizada é adequada a microcontroladores que geralmente tem recursos limitados, liberando espaço para outras funções embarcadas, como por exemplo controle e automação, comunicação entre outros.

Segue abaixo figura com um exemplo gráfico da Média Móvel em ação. O sinal em azul é sinal de entrada não filtrado, a parte em vermelho é o sinal filtrado (fonte *wikipedia*).

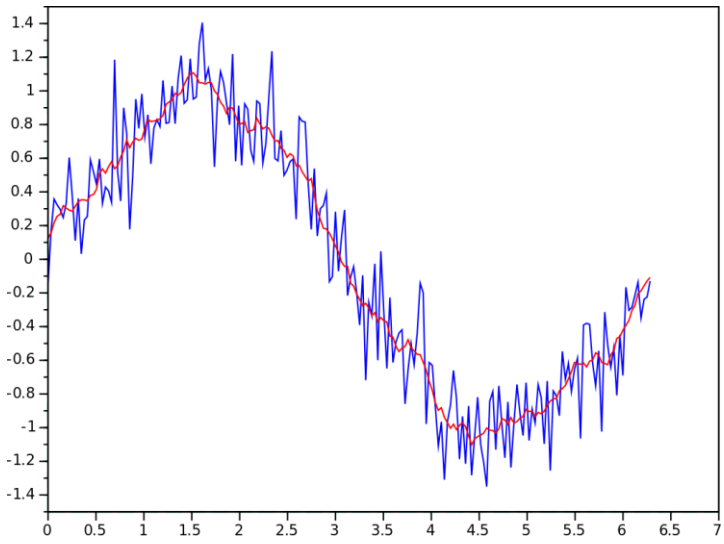


Figura 18 -Sinais filtrado (azul) e não filtrado (vermelho) por média móvel (Fonte: Wikipedia)

A resposta em frequência da Média Móvel é dada pela equação abaixo com f entre 0 e 0,5. Para f = 0 teremos H(f) = 1:

$$H(f) = \frac{sen(\pi f M)}{M sen(\pi f)}$$

Abaixo a representação gráfica da resposta em frequência para N=4 (azul), N=8 (verde), N=16 (preto) e dupla aplicação do N=16 (vermelho).

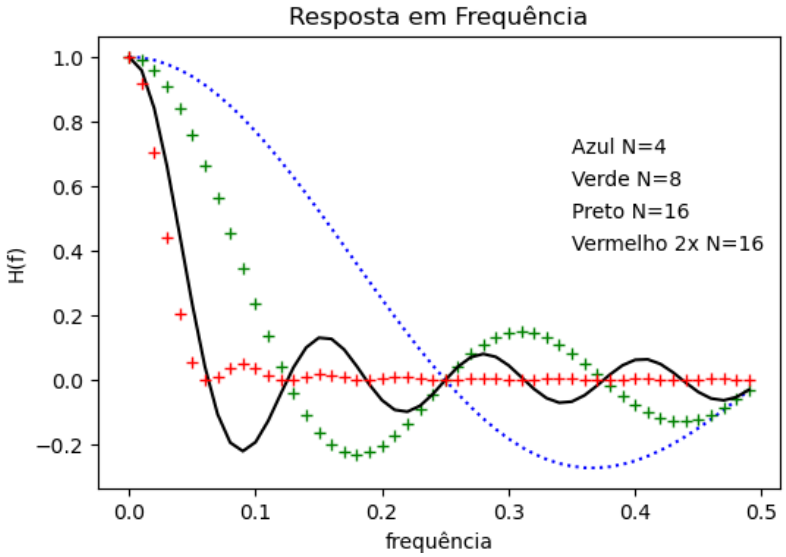


Figura 19 -Resposta em frequência da média móvel

A Média Móvel pode ser aplicada em múltiplas passagens, tipicamente duas a três vezes, melhorando significativamente a sua resposta em frequência, sim assim for necessário. Observa-se no gráfico acima um ótimo desempenho relativo da resposta em frequência com a dupla aplicação da Média Móvel com N=16.

IMPLEMENTAÇÃO DA MÉDIA MÓVEL

Para testar a Média Móvel nas plataformas Arduino UNO R4 desenvolvemos programa em Linguagem C++ utilizando a biblioteca referência e o ambiente IDE do Arduino.

O programa exemplo didático desenvolvido está escrito de forma simplificada. Não tem grandes otimizações de código e utiliza apenas o básico do IDE do Arduino. Os dados do sinal de entrada x[n] são obtidos através da interface Analógica do Arduino, apenas para fins de teste.

O programa calculará a Média Móvel com tamanho de N = 8 para uma taxa de 256 amostras por segundo. O programa contém apenas variáveis globais e apenas três funções em C++, duas padrão do Arduino (setup e loop) e uma da DFT. O resultado das medições de tempo será enviado pelo Arduino, via comunicação

serial, para o ambiente IDE do Arduino e podem ser observadas no “serial monitor” do IDE. O cálculo da Média Móvel será convenientemente feito com números inteiros utilizando a equação otimizada.

O LED do Arduino ficará piscando, indicando ao usuário que está ativo e com o programa de teste do Filtro MM em execução. A parte do programa que faz isto é simples e não bloqueia o programa principal.

```
// Programa para teste de Filtro Digital do tipo Média Móvel
// Obtêm o sinal x[n] na interface analógica gerando um sinal y[n] filtrado
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.2
// Agosto de 2024
// Autor: Paschoal Molinari
// Licença: MIT License

// Inicialização das variáveis globais
int M = 8;     // numero de amostras (x) na média
int xm[8] = {};  // buffer circular de entrada com sinal amostrado normalizado
x[n]/M
int x_pos = 0;  // posição do x mais antigo
int y = 0;     // sinal de saída filtrado
int conta_voltas = 0;  // contador auxiliar do LED piscando

void setup() {
  // Inicializa a comunicação serial a 115200 bits por segundo (bps):
  Serial.begin(115200);
  Serial.println(" ");
  Serial.println("Iniciando o teste do Filtro MM");
  // Define o LED interno para sinalizar atividade do Arduino
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  // controle do LED piscando
  ++conta_voltas;
  if(conta_voltas == 129){
    Serial.println("Piscando o LED");  // piscando o LED como indicativo de
funcionamento
    digitalWrite(LED_BUILTIN, HIGH); } // Ligando o LED (HIGH is the voltage level)
  if(conta_voltas == 257){
    digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
```

```
  conta_voltas = 0; }

 // Lê a entrada analógica do pin 0:
 // Aproximadamente 100 microsegundos de retardo no UNO R3
 int x_atual = analogRead(A0);

 // Calculo da Média Móvel com nova amostra
 // y[n] = y[n-1] + x[n]/M - (x[n-m])/M
 y = y - xm[x_pos];
 xm[x_pos] = x_atual/M;
 y = y + xm[x_pos];
 if (++x_pos == M) {x_pos = 0;}  // Calcula posição do último xm no buffer circular

 // Coloque aqui o código da sua aplicação
 //

 // Envia o sinal filtrado y para a linha serial apenas para teste
 Serial.print(" ");
 Serial.print(y);

 // retardo para ajustar aproximadamente 256 amostras/segundo
 delayMicroseconds(3800);
}
```

AVALIAÇÃO DO TEMPO DE EXECUÇÃO DA MÉDIA MÓVEL

Para obter o tempo utilizado na execução de uma única passagem de Média Móvel desenvolvemos o programa abaixo com alterações do programa exemplo anterior. A alteração é a inserção de um loop de 10.000 (dez mil) voltas, sendo que em cada volta são calculadas 10 MM, totalizando 100 mil MM calculadas por loop. A utilização de 10 MM por volta é para reduzir a influência do tempo do controle do loop no tempo total.

```
// Programa para teste de tempo de Filtro Digital do tipo Média Móvel - MM
// Obtêm o sinal x[n] na interface analógica gerando um sinal y[n] filtrado
// Para plataforma Arduino UNO R4
// Arduino IDE 2.3.2
// Agosto de 2024
// Autor: Paschoal Molinari
```

```
// Licença: MIT License

// Inicialização das variáveis globais
int M = 8;     // numero de amostras (x) na média
int xm[8] = {};  // buffer circular de entrada com sinal amostrado normalizado
x[n]/M
int x_pos = 0;  // posição do x mais antigo
int y = 0;     // sinal de saída filtrado
int conta_voltas = 0;  // contador auxiliar do LED piscando
int cont_num;       // contador auxiliar do envio de tempo de execução via serial

int i; // Para uso no loop de avaliação de tempo
long tempo;

void setup() {
 // Inicializa a comunicação serial a 115200 bits por segundo (bps):
 Serial.begin(115200);
 Serial.println(" ");
 Serial.println("Iniciando o teste de tempo do Filtro MM");
 // Define o LED interno para sinalizar atividade do Arduino
 pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
 // controle do LED piscando
 ++conta_voltas;
 if(conta_voltas == 129){
  Serial.println(" ");
  Serial.println("Piscando o LED");  // piscando o LED como indicativo de
funcionamento
  digitalWrite(LED_BUILTIN, HIGH); } // Ligando o LED (HIGH is the voltage level)
 if(conta_voltas == 257){
  digitalWrite(LED_BUILTIN, LOW);    // Desligando o LED
  conta_voltas = 0; }

 // Lê a entrada analógica do pin 0:
 // Aproximadamente 100 microsegundos de retardo no UNO R3
 int x_atual = analogRead(A0);

 // Loop para avaliação do tempo de execução do cálculo da Média Movel
 // total de 100 mil MM serão calculadas
 tempo = micros();
 for(i = 0; i < 10000; i++){
  // Calculo da Média Móvel com nova amostra
  // y[n] = y[n-1] + x[n]/M - (x[n-m])/M
```

```
   // 10 linhas iguais para diminuir o peso do loop de controle na medição do
tempo
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   y -= xm[x_pos]; xm[x_pos] = x_atual/M; y += xm[x_pos]; if(++x_pos == M){x_pos
= 0;}
   }
  tempo = micros() - tempo; // calculo do tempo de execução do loop

  // Envia os tempos de execução para a serial
  Serial.print(" ");
  Serial.print(tempo);
  if (++cont_num == 10) // pula a linha a cada 10 numeros
   {cont_num=0;
   Serial.println(" ");}

  // retardo para permitir a leitura do tempo
  delay(1000);
}
```

RESULTADOS DOS TESTES

Agregando estas medições em uma única tabela temos o resumo abaixo com tempo de execução em microssegundos para a execução de 100 mil MM.

Arduino UNO R3	Arduino UNO R4	Ganho R3/R4
161.430	18.385	8,8

Pelos dados obtidos o Arduino UNO R4 foi 8,8 vezes mais rápido que o Arduino UNO R3 neste teste. O tempo de execução de uma MM é de aproximadamente 1,6 microsegundos no Arduino UNO R3, e 0,18 microsegundos no Arduino UNO R4 mostrando ser este tipo de filtro viável para uma ampla gama de aplicações.

DISCUSSÃO E CONCLUSÃO

O comportamento fixo do tempo de execução dos Arduinos UNO R3 e UNO R4 com o número de amostras M é uma característica da implementação da equação otimizada da Média Móvel. O Filtro é bem leve e pode ser utilizado em microcontroladores de 8 bits com certa tranquilidade, deixando espaço para a implementação de aplicações como por exemplo de controle, automação e comunicação.

O Arduino UNO R4 apresentou um ganho típico de desempenho de 8,8 vezes o desempenho do Arduino UNO R3 quando executando a Média Móvel. Este é um ganho significativo e já esperado. A vantagem de uma ordem de magnitude do Arduino UNO R4 na execução da MM se deve a combinação do seu relógio (clock) superior e do fato de ser capaz de tratar cálculos com inteiros de 16 bits com uma eficiência muito superior.

A plataforma Arduino UNO R4, com as devidas otimizações, provavelmente podem embarcar algumas funções de processamento de sinais de baixa

frequência comuns em aplicações de controle e automação. A facilidade de programação e prototipação é um ganho no desenvolvimento de soluções.

Incentivamos ao leitor que altere o programa didático deste capítulo das seguintes formas: Colocar o sinal filtrado na saída PWM do Arduino; Implementar e testar múltiplas passagens da MM.

REFERÊNCIAS BIBLIOGRÁFICAS

ACOUSTIBLOK. **What are the different types of noise?**. [S. l.], 2024. Disponível em: https://www.acoustiblok.co.uk/what-are-the-different-types-of-noise/. Acesso em: março de 2025.

ARDUINO. [**S. l.**]: Arduino, [2025?]. Disponível em: https://www.arduino.cc/. Acesso em: março de 2025.

ARDUINO. **Arduino Libraries**. [S. l.], [2025?]. Disponível em: https://www.arduino.cc/reference/en/libraries/. Acesso em: março de 2025.

ARDUINO. **Arduino Reference**. [S. l.], [2025?]. Disponível em: https://www.arduino.cc/reference/en/. Acesso em: março de 2025.

ARDUINO. **Arduino UNO R3**. [S. l.], [2025?]. Disponível em: https://store.arduino.cc/products/arduino-uno-rev3. Acesso em: março de 2025.

ARDUINO. **Arduino UNO R4**. [S. l.], [2025?]. Disponível em: https://store.arduino.cc/products/uno-r4-wifi. Acesso em: março de 2025.

CIRRUS RESEARCH. **The Colours of Noise: What are they and what do they mean?**. [S. l.], 2023. Disponível em: https://www.cirrusresearch.co.uk/blog/2023/04/the-colours-of-noise/. Acesso em: março de 2025.

IEEE XPLORE. **ARM® Floating Point Unit (FPU)**. [S. l.], 2015. Disponível em: https://ieeexplore.ieee.org/document/7394685. Acesso em: março de 2025.

KERNIGHAN, B.; RITCHIE, D. **C - A Linguagem de Programação**. [S. l.]: Elsevier, 1989.

LYONS, Richard G. **Understanding Digital Signal Processing**. 3. ed. [S. l.]: Pearson, 2011.

MIT. **Exploring the MIT Open Source License: A Comprehensive Guide**. [S. l.], [2025?]. Disponível em: https://tlo.mit.edu/understand-ip/exploring-mit-open-source-license-comprehensive-guide. Acesso em: março de 2025.

OLIVEIRA, C. L. V. et al. **Aprenda Arduino**: uma abordagem prática. Duque de Caxias: Katzen Editora, 2018.

OPPENHEIM, Allan V.; SCHAFER, Ronald W. **Discrete-Time Signal Processing**. 3. ed. [S. l.]: Pearson, 2010.

OPPENHEIM, Allan V.; WILLSKY, Alan S. **Sinais e Sistemas**. 2. ed. [S. l.]: Pearson, 2010.

RENESAS. **RA Series 32-bit MCUs with Arm Cortex-M Core**. [S. l.], [2024?]. Disponível em: https://www.renesas.com/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus. Acesso em: março de 2025.

SMITH, Steven W. **The Scientist and Engineer's Guide to Digital Signal Processing**. 2. ed. Califórnia: California Technical Publishing, 1999.

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES (OEIS). **A357907**. [S. l.], [2024?]. Disponível em: https://oeis.org/A357907. Acesso em: março de 2025.

WIKIPEDIA. **Arduino Uno**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Arduino_Uno. Acesso em: março de 2025.

WIKIPEDIA. **Arduino**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Arduino. Acesso em: março de 2025.

WIKIPEDIA. **ARM Cortex-M**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/ARM_Cortex-M. Acesso em: março de 2025.

WIKIPEDIA. **Big O notation**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Big_O_notation. Acesso em: março de 2025.

WIKIPEDIA. **Butterfly diagram**. [S. l.], [2025?]. Disponível em: https://en.wikipedia.org/wiki/Butterfly_diagram. Acesso em: março de 2025.

WIKIPEDIA. **C (linguagem de programação)**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/C_(linguagem_de_programação. Acesso em: março de 2025.

WIKIPEDIA. **C++**. [S. l.], [2025?]. Disponível em: https://pt.wikipedia.org/wiki/C++. Acesso em: março de 2025.

WIKIPEDIA. **Colors of noise**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Colors_of_noise. Acesso em: março de 2025.

WIKIPEDIA. **Cooley–Tukey FFT algorithm**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm. Acesso em: março de 2025.

WIKIPEDIA. **Cultura Maker**. [S. l.], [2025?]. Disponível em: https://en.wikipedia.org/wiki/Maker_culture. Acesso em: março de 2025.

WIKIPEDIA. **Discrete Fourier transform**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Discrete_Fourier_transform. Acesso em: março de 2025.

WIKIPEDIA. **Fast Fourier transform**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Fast_Fourier_transform. Acesso em: março de 2025.

WIKIPEDIA. **Gerador de números pseudoaleatórios**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/Gerador_de_numeros_pseudoaleatorios. Acesso em: março de 2025.

WIKIPEDIA. **Geradores congruentes lineares**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/Geradores_congruentes_lineares. Acesso em: março de 2025.

WIKIPEDIA. **Identidade de Euler**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/Identidade_de_Euler. Acesso em: março de 2025.

WIKIPEDIA. **Licença MIT**. [S. l.], [2025?]. Disponível em: https://pt.wikipedia.org/wiki/Licença_MIT. Acesso em: março de 2025.

WIKIPEDIA. **Média Móvel**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/Média_móvel. Acesso em: março de 2025.

WIKIPEDIA. **Moving Average**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Moving_average. Acesso em: março de 2025.

WIKIPEDIA. **Noise (signal processing)**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Noise_(signal_processing. Acesso em: março de 2025.

WIKIPEDIA. **Noise (spectral phenomenon)**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Noise_(spectral_phenomenon. Acesso em: março de 2025.

WIKIPEDIA. **Noise**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Noise. Acesso em: março de 2025.

WIKIPEDIA. **Pesquisa aplicada**. [S. l.], [2025?]. Disponível em: https://pt.wikipedia.org/wiki/Pesquisa_aplicada. Acesso em: março de 2025.

WIKIPEDIA. **Ruído branco**. [S. l.], [2024?]. Disponível em: https://pt.wikipedia.org/wiki/Ruído_branco. Acesso em: março de 2025.

WIKIPEDIA. **Salt-and-pepper noise**. [S. l.], [2025?]. Disponível em: https://en.wikipedia.org/wiki/Salt-and-pepper_noise. Acesso em: março de 2025.

WIKIPEDIA. **Séries de Fourier**. [S. l.], [2025?]. Disponível em: https://en.wikipedia.org/wiki/Fourier_series. Acesso em: março de 2025.

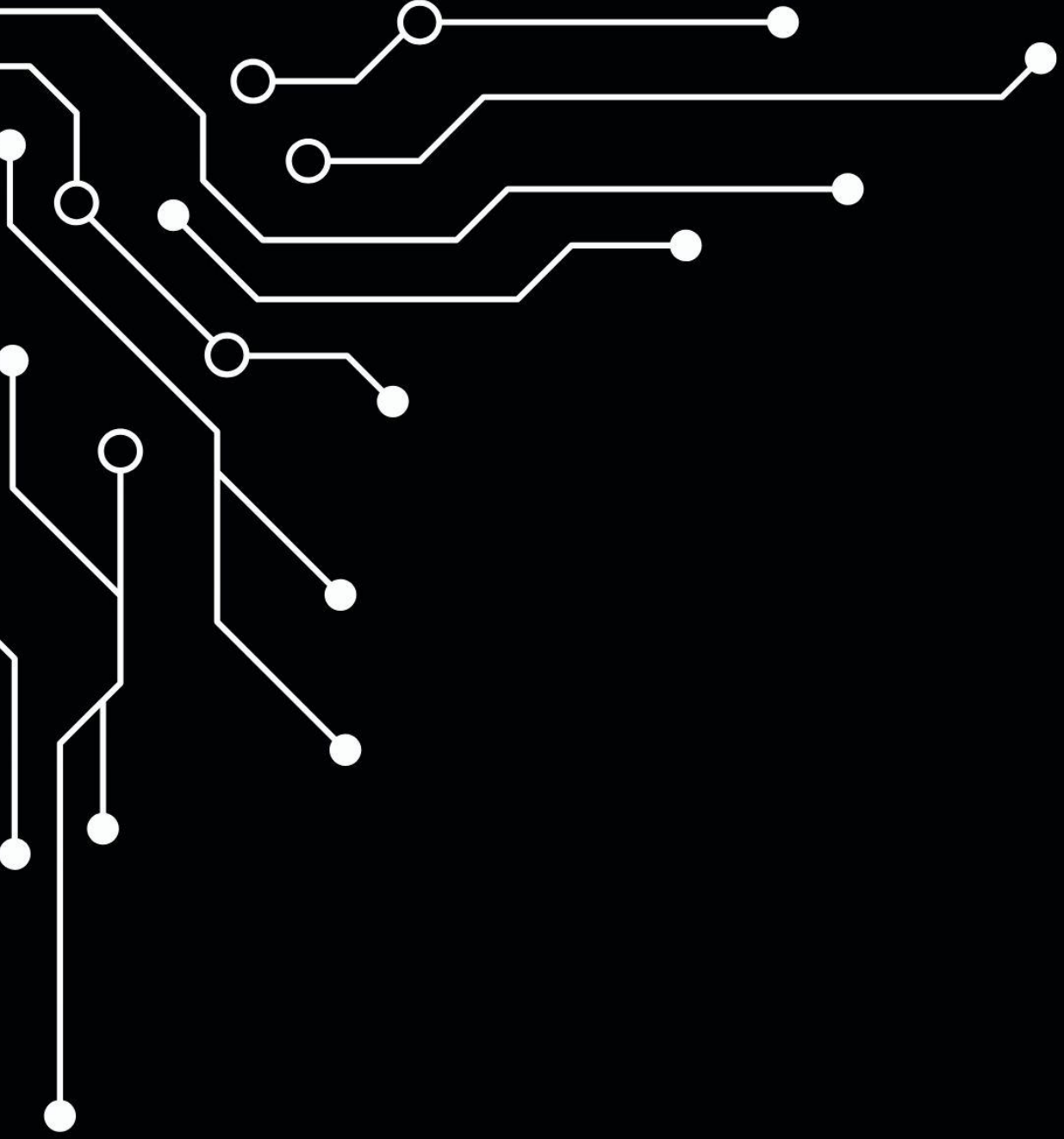
WIKIPEDIA. **Signal processing**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/Signal_processing. Acesso em: março de 2025.

WIKIPEDIA. **The C Programming Language**. [S. l.], [2025?]. Disponível em: https://pt.wikipedia.org/wiki/The_C_Programming_Language. Acesso em: março de 2025.

WIKIPEDIA. **White noise**. [S. l.], [2024?]. Disponível em: https://en.wikipedia.org/wiki/White_noise. Acesso em: março de 2025.

SOBRE O AUTOR

Paschoal Molinari Neto é um brasileiro, formado em Engenharia Elétrica pela UFBa e com Mestrado em Engenharia Eletrônica pelo ITA. Tem muitos anos de experiência nas áreas de Informática voltado as Telecomunicações e na Educação de Engenharia. Atualmente é Professor do IFPR. Tem atuado nas áreas de Eletrônica, Microcontroladores e Aprendizagem de Máquina.



ISBN: 978-6-55321-040-0