UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JORGE RICARDO JAÚ JUNIOR

REDES DE SENSORES E INTERPOLAÇÃO DE DADOS: UM CASO DE USO PRÁTICO

RIO DE JANEIRO
2022

JORGE RICARDO JAÚ JUNIOR

REDES DE SENSORES E INTERPOLAÇÃO DE DADOS: UM CASO DE USO PRÁTICO

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profa. Silvana Rossetto
Co-orientador: Prof. Daniel Gregorio Alfaro Vigo

RIO DE JANEIRO
2022

CIP - Catalogação na Publicação

J41r

Jaú Junior, Jorge Ricardo
Redes de sensores e interpolação de dados: um caso de uso prático / Jorge Ricardo Jaú Junior. -- Rio de Janeiro, 2022.
53 f.

Orientadora: Silvana Rossetto.
Coorientador: Daniel Gregorio Alfaro Vigo.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2022.

1. Redes de sensores sem fio. 2. Interpolação linear. 3. Interpolação spline cúbico. 4. Python. I. Rossetto, Silvana, orient. II. Vigo, Daniel Gregorio Alfaro, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

JORGE RICARDO JAÚ JUNIOR

REDES DE SENSORES E INTERPOLAÇÃO DE DADOS: UM CASO DE USO PRÁTICO

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 03 de outubro de 2022

BANCA EXAMINADORA:

Silvana Rossetto
Professora Doutora (Universidade Federal do Rio de Janeiro)

Daniel Gregorio Alfaro Vigo
Professor Doutor (Universidade Federal do Rio de Janeiro)

João Carlos Pereira Da Silva
Professor Doutor (Universidade Federal do Rio de Janeiro)

Angela Maria Silva Gonçalves, Professora Doutora (Universidade Federal do Rio de Janeiro)

Dedico este trabalho a todos os professores, aos que são eternos estudantes na missão de trazer luz aos que buscam por conhecimento.

AGRADECIMENTOS

Agradeço a Deus, pois Ele foi a força e a base que eu precisava quando não tinha nenhuma. Sou extremamente grato à minha esposa, que me ajudou nos momentos em que eu não acreditava em meu potencial e que sempre esteve ao meu lado. Agradeço aos meus pais por todos os degraus que construíram na escada para meu crescimento acadêmico. Agradeço aos meus pastores que me apoiaram incondicionalmente e que estiveram dispostos a me ouvir e me incentivar. Agradeço aos ausentes, que por meio de suas vidas expressaram o desejo de ver este resultado. Agradeço profundamente aos meus professoes orientadores, que em tempos difíceis creram em mim e foram meus guias para a conclusão desta obra.

"A persistência é o menor caminho do êxito."

Albert Einstein

RESUMO

Este trabalho mostra uma forma didática de aplicar métodos numéricos de interpolação linear e de interpolação por spline cúbico a uma massa de dados de medições de temperatura, coletada a partir de um *testbed* de uma rede de sensores sem fio. O objetivo dessa aplicação é auxiliar os professores da cadeira de Cálculo Numérico no ensino destes tópicos, mostrando a aplicação dos modelos matemáticos de forma prática. O trabalho compreende a coleta dos dados a partir da rede de sensores sem fio e o envio desses dados até o computador final, onde os métodos de interpolação são aplicados aos dados e são geradas as massas de dados interpolados. O trabalho faz uso da linguagem de programação Python e de algumas de suas bibliotecas para leitura da massa de dados, aplicação dos métodos numéricos mencionados e apresentação gráfica dos resultados. Este trabalho oferece aos alunos e professores de Cálculo Numérico uma forma de verificação prática dos métodos de interpolação. Seu intuito é servir como uma ferramenta para a avaliação dos métodos e seus aspectos de forma clara e objetiva.

Palavras-chave: Redes de sensores sem fio; Interpolação spline cúbico e linear; Python

ABSTRACT

This paper shows a didatic way of applying numerical methods of linear interpolation and cubic spline interpolation to a mass of temperature measurement data collected from a *testbed* of a wireless sensors network. The main purpose for this application is to assist numerical calculus professors in teaching these topics, showing the application of mathematical models in a practical way. This work comprises the collection of data from the wireless sensors network and the sending of the data to the final computer, where the interpolation methods are applied. This work uses Python as its programming language and some of its frameworks for reading the data, applying the numerical methods mentioned and the graphically presenting the results. This work offers to students and professors of Numerical Calculus a way to check the interpolation methods. Its purpose is to serve as a tool to evaluate the methods and their aspects in a clear and objective way.

Keywords: Wireless Sensors Network; Linear and Cubic Spline Interpolation; Python

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Segundo (BURDEN; FAIRES, 1989), a interpolação é uma técnica matemática para geração de um conjunto de valores a partir de um conjunto de dados já conhecido. Um dos fatores importantes para o uso dessa ferramenta é a necessidade de estimar valores intermediários dentro da massa de dados.

Uma possível aplicação é estimar a quantidade de pessoas acometidas de uma doença contagiosa em um intervalo de tempo menor do que o intervalo entre medições. Usando os métodos de interpolação é possível gerar uma forma de estimar valores fora dos intervalos, porém próximos ao conjunto de dados previamente conhecidos.

Geralmente a apresentação dos métodos de interpolação aos estudantes nos cursos de Ensino Superior foca na parte teórica, com exemplos pouco expressivos para o dia a dia do aluno, de forma que a compreensão do tema pelo aluno fica comprometida. A forma mais usada no ensino destes métodos envolve a explicação de como o método funciona e do uso de exemplos teóricos e aplicação de exercícios. Este modelo por vezes não atrai a atenção de todos os alunos, por não ser aparentemente prático ou real.

Este trabalho tem como objetivo apresentar uma ferramenta para apresentar um caso de uso de dois métodos de interpolação em conjunto com dados sensoriados, apresentando dados reais a partir de grandezas físicas. Esses valores podem ser medidos em intervalos diversos e armazenados para verificação futura. A comparação entre os valores gerados pelo método de interpolação e os dados reais permite atestar o quanto essa aproximação é válida.

Grandezas físicas podem variar de forma lenta durante um intervalo de tempo ou de forma mais rápida. Podemos exemplificar a variação lenta como a mudança da temperatura de uma área ao longo de um dia, assim como a variação rápida na medição da luminosidade de um ambiente onde podemos acender um foco de luz em um ambiente com pouca luminosidade. A mensuração desses dados pode ser feita através de uma rede de sensores sem fio (RSSF)(AKYILDIZ et al., 2002) previamente configurada para medir as grandezas físicas que são nossos alvos.

As RSSFs são redes compostas por dispositivos com microcontroladores, capazes de fazer o sensoriamento de alguma grandeza física no ambiente onde são dispostas (por exemplo, iluminação, temperatura ou pressão) e transmitir os dados coletados entre os componentes da rede de forma a chegar no componente responsável pela entrega desses dados à aplicação final.

Os componentes das RSSFs podem ser distribuídos de formas diversas (*ad hoc* ou de maneira ordenada). Cada componente é provido de uma fonte de energia limitada (baterias ou pilhas) e por isso precisa efetuar o sensoriamento e transmissão dos dados com o mínimo de gasto de energia.

Os métodos de interpolação permitem extrapolar a base de dados previamente armazenada. Do ponto de vista da RSSFs, isso permite que possamos diminuir a quantidade de coletas de dados, de forma a prolongar a vida da RSSF (consumindo menos energia da pilha ou bateria de cada nó da rede).

Para facilitar a realização de experimentos que usam RSSF, existem ambientes controlados chamados de *testbeds*. Neste trabalho, faremos uso desses ambientes para realizar o sensoriamento das grandezas físicas, e, em seguida aplicar os métodos de interpolação para a extrapolação dos dados coletados. Nossa proposta é usar um testbed de RSSF como fonte de coleta de dados reais em diferentes intervalos de medições.

Uma RSSF é configurada no *testbed* para coletar e transmitir os dados coletadas até chegarem em um nó sensor que está diretamente ligado a um computador. Desse computador, os dados são transmitidos até o computador do professor (via Internet) onde apresentamos em uma interface gráfica os dados coletados e aplicamos os métodos de interpolação linear e de Spline Cúbico nos valores medidos em frequências definidas.

Usando dados de temperatutra, fizemos uma comparação entre os dados coletados, os dados interpolados e o intervalo de confiança dos dados interpolados e, conforme visto, os dados estão próximos com uma taxa de acerto de 85% .

O restante deste trabalho está dividido em cinco capítulos. No Capítulo 2 são apresentados conceitos básicos sobre redes de sensores sem fio e os métodos de interpolação estudados. A implementação dos algoritmos dos nós sensores, da interface gráfica e da geração dos gráficos estão no Capítulo 3. A avaliação do experimento, em conjunto com as métricas usadas está no Capítulo 4. A conclusão e trabalhos futuros estão no Capítulo 5.

2 CONCEITOS BÁSICOS

Neste capítulo abordamos os conceitos alvos deste trabalho. Apresentamos na primeira seção os métodos de interpolação, especificamente o método de interpolação linear e o método de interpolação por Spline Cúbico (seu uso e como se aplica neste trabalho). Na segunda seção, apresentamos as bibliotecas da linguagem de programação Python utilizadas na manipulação dos dados e apresentação dos resultados, as bibliotecas SciPy e as sub-bibliotecas embarcadas NumPy, Matplotlib e TkAgg. Por fim, na terceira seção, descrevemos as redes de sensores sem fio e seus componentes principais.

2.1 MÉTODOS DE INTERPOLAÇÃO

A interpolação é um método de construção de uma função que gera um conjunto de valores a partir de um conjunto de pontos ordenados conhecidos. Este procedimento é tomado quando há um conjunto baixo de amostras e se deseja extrapolar esse conjunto, seja na intenção de conhecer possíveis valores entre os dados medidos ou fazer projeções fora do espaço amostral coletado.

Neste trabalho vamos abordar os métodos de interpolação linear e Spline Cúbico. O objetivo do uso destes métodos é avaliar o seu comportamento quando o conjunto de pontos de entrada são grandezas físicas (por exemplo, a temperatura de um ambiente), dadas algumas métricas estatísticas para avaliação e, sendo assim, se eles são relevantes para extrapolar uma análise dos pontos não cobertos pelos dados coletados.

2.1.1 Interpolação Linear

De acordo com (BURDEN; FAIRES, 1989), a interpolação linear é a criação de uma função que retorna um conjunto de dados baseado nos dados que obtivemos dentro de um espaço amostral. A interpolação linear particulariza a interpolação polinomial, onde o alvo do método é definir uma função interpoladora de grau 1.

Dados dois pontos $A\ (x_a, y_a)$ e $B\ (x_b, y_b)$, é possível encontrar a função interpoladora $f(x_n)$ no intervalo entre x_a e x_b.

Podemos estabelecer a seguinte relação:

$$\frac{y - y_a}{y_b - y_a} = \frac{x - x_a}{x_b - x_a}$$

Após manipulação algébrica chegaremos em

$$f(x) = y = y_a - (y_b - y_a)\frac{x - x_a}{x_b - x_a}$$

que é um polinômio de primeiro grau, dependente apenas dos valores de x, dado que conhecemos os valores de x_a, x_b, y_a e y_b.

Vale ressaltar que a interpolação linear gera uma função interpoladora para cada subintervalo do conjunto de dados, ou seja, para subintervalos diferentes, haverá polinômios interpoladores diferentes.

2.1.2 Interpolação de Spline Cúbico

O método de interpolação linear é, como uma primeira aproximação, uma estimativa válida para geração de dados. Porém é necessário considerar que esta técnica possui uma deficiência importante. Ao observarmos os pontos extremos dos subintervalos usados para a geração do polinômio gerador, notamos que estes pontos não possuem uma segunda derivada suave, devido à natureza das funções lineares.

Uma das formas de aplicarmos uma função interpoladora contínua e suave é aproximar por polinômios contínuos nos intervalos entre os pontos escolhidos dentro do espaço amostral dos dados colhidos (também conhecido por aproximação por polinômios seccionados).

Temos então a seguinte formalização:

Seja uma função f dentro do intervalo $[A, B]$ com imagem em $\mathbb{R}$, com um conjunto de nós $A = x_0 < x_1 < \cdots < x_n = B$.

Um Spline Cúbico é uma função $S : [A, B] \to \mathbb{R}$, onde

a) $S(x)$ é um polinômio cúbico, indicado por $S_j(x)$ no intervalo entre x_j e x_{j+1} para cada $0 \leq j \leq n-1$;

b) $S(x_j) = f(x_j)$, para cada $0 \leq j \leq n$;

c) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$, para cada $0 \leq j \leq n-2$;

d) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, para cada $0 \leq j \leq n-2$;

e) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, para cada $0 \leq j \leq n-2$;

f) Uma das propriedades é válida:

- $S''(x_0) = S''(x_n) = 0$, também conhecido como Spline natural ou de contorno livre.
- $S'(x_0) = f'(x_0)$ e $S'(x_n) = f'(x_n)$, também conhecido como Spline restrito.

O termo contorno livre se deve à característica visual da Spline se parecer com hastes que passam forçadamente pelos pontos de interpolação (inclusive os extremos), sem imposição de condições aos pontos.

A interpolação por Spline Cúbico aproxima os intervalos entre os pontos por polinômios de grau três (para garantir a continuidade da função em suas primeiras e segundas derivadas — incluindo nos pontos extremos do intervalo). A partir disso, há a necessidade de calcular quatro constantes para cada um dos n intervalos.

Os coeficientes da Spline são calculados da forma a seguir, conforme (BURDEN; FAIRES, 1989):

$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$

Isso vale para cada $j = 0, 1, \ldots, n - 1$.

O termo $S_j(x_j) = a_j = f(x_j)$, então pela aplicação da condição **c** obtemos:

$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$

Usaremos a convenção a seguir para melhor simplificação da demonstração.

$q_j = x_{j+1} - x_j$

Se $a_j = f(x_j)$, então é possível afirmar que

$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j q_j + c_j q_j^2 + d_j q_j^3$

para cada intervalo $j = 0, 1, ..., n - 1$.

Podemos definir também que

$b_n = S'(x_n)$

e podemos deduzir que

$S'_j(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2$

Se $b_j = S'_j(x_j)$

para cada $j = 0, 1, ..., n - 1$.

Aplicando a condição **d** temos:

$b_{j+1} = b_j + 2c_j q_j + 3d_j q_j^2$ para cada $j = 0, 1, ..., n - 1$.

É possível deduzir a relação de mais uma das constantes de S_j por definir $c_j = \frac{S''(x_j)}{2}$

Aplicando a condição **e**, temos, para cada $j = 0, 1, ..., n - 1$:

$c_{j+1} = c_j + 3d_j q_j$

Se isolarmos o termo d_j e fizermos as substituições, teremos:

$a_{j+1} = a_j + b_j q_j + \frac{q_j^2}{3}(2c_j + c_{j+1})$

$b_{j+1} = b_j + q_j(c_j + c_{j+1})$

Podemos também por d_j em evidência e substituir acima, obtendo

$a_{j+1} = a_j + b_j q_j + \frac{q_j^2}{3}(2c_j + c_{j+1})$

$b_{j+1} = b_j + q_j(c_j + c_{j+1})$

Evidenciando b_j, temos:

$b_j = \frac{1}{q_j}(a_{j+1} - a_j) - \frac{q_j}{3}(2c_j + c_{j+1})$

Se trocarmos o termo j pelo termo $j - 1$, teremos

$b_{j-1} = \frac{1}{q_{j-1}}(a_j - a_{j-1}) - \frac{q_{j-1}}{3}(2c_{j-1} + c_j)$

Ao substituirmos os termos na equação gerada para b_{j+1} e usarmos o termo $j - 1$, temos

$q_{j-1}c_{j-1} + 2(q_{j-1} + q_j)c_{j+1} = \frac{3}{q_j}(a_{j+1} - a_j) - \frac{3}{q_{j-1}}(a_j - a_{j-1})$ **(a)** para cada $j = 1, 2, ..., n - 1$.

De acordo com (BURDEN; FAIRES, 1989), temos o seguinte teorema sobre Splines naturais.

Teorema: *Se f é definida para $a = x_0 < x_1 < x_2 < \cdots < x_n = b$, então f possui um único polinômio interpolador S nos pontos $x_0, x_1, \ldots, x_n$, uma função interpoladora do tipo Spline que satisfaz as condições de contorno $S"(a) = 0$ e $S"(b) = 0$.*

Com essas condições de contorno, podemos dizer que $c_n = \frac{S"(x_n)}{2} = 0$ e que $0 = S"(x_0) = 2c_0 + 6d_0(x_0 - x_0)$ então $c_0 = 0$.

As definições $c_0 = 0$ e $c_n = 0$, junto com a equação **(a)** formam um sistema linear descrito pela equação vetorial $A\mathbf{x} = \mathbf{b}$ onde A é uma matriz $(n+1) \times (n+1)$, $mathbfb$ é um vetor de $(n+1)$ linhas, ambos demonstrados em (BURDEN; FAIRES, 1989) e $\mathbf{x}$ é o vetor com os coeficientes $(C_0, C_1, ..., C_n)$, os coeficientes desejados.

2.2 PYTHON E BIBLIOTECAS UTILIZADAS

Neste trabalho optamos por usar a linguagem de programação Python. Esta decisão teve por base a maior familiaridade do autor com esta linguagem, bem como as várias bibliotecas com foco em desenvolvimento científico disponíveis.

Nesta seção apresentamos os métodos usados da biblioteca `Scipy` (VIRTANEN et al., 2020), uma biblioteca focada em programação científica; e também os métodos usados da biblioteca `Numpy` (OLIPHANT, 2006), uma bilbioteca pertencente a `Scipy`, com foco em programação para cálculos numéricos de grande escala de dados. Por fim, apresentamos os métodos usados da biblioteca `Matplotlib` (HUNTER, 2007) para visualização dos dados e dos resultados obtidos, em conjunto com a bilbioteca `Tkinter` (LUNDH, 1999), uma biblioteca de geração de janelas.

2.2.1 Biblioteca SciPy

A biblioteca SciPy é um ecossistema formada de vários subpacotes de algoritmos matemáticos para computação científica. As classes e métodos desta biblioteca facilitam a manipulação e visualização de dados. Esta biblioteca foi escolhida para este trabalho por conter o subpacote *scipy.interpolation*, que contém os métodos usados para gerar a interpolação dos dados obtidos.

2.2.2 Biblioteca NumPy

A biblioteca NumPy começou a ser desenvolvida por Travis Oliphant (OLIPHANT, 2006). Ela foi baseada em versões anteriores de bibliotecas para operações numéricas e operações de vetores multidimensionais de forma mais eficiente. Hoje essa biblioteca é mantida por uma comunidade científica que zela pelo seu conteúdo. Esta biblioteca é utilizada em vários tipos de projetos, como modelos de aprendizado de máquina e processamento de imagens em computação gráfica. A biblioteca também é usada para cálculos matemáticos, como integração e derivação de funções. O objetivo do uso desta biblioteca neste trabalho é o de manipular dados coletados por sensores de uma rede de

sensores sem fio, como horário de detecção/medição e o valor medido, de forma fácil e colaborativa com outras bibliotecas científicas, como a SciPy.

2.2.3 A interpolação linear na biblioteca SciPy

Usamos o subpacote *interpolate* da biblioteca SciPy para gerar a função interpoladora linear deste trabalho. Nele é possível criar um objeto do tipo *interp1d*, que é a função interpoladora. São necessários alguns parâmetros para criar o objeto, dois vetores unidimensionais do tipo `narray` e a especificação para o tipo de interpolação que queremos gerar.

Os vetores usados para a criação do objeto foram o de tempo da medição (primeiro parâmetro) e o valor medido (no caso, a temperatura) (segundo parâmentro) neste trabalho. Estes foram gerados a partir do vetor de pontos (tempo, temperatura). Cada vetor possui uma quantidade de elementos menor que a quantidade total de pontos, uma vez que desejamos criar novos pontos usando a função interpoladora. O objetivo disso é verificar se nossa função interpoladora geraria pontos exatos ou próximos do intervalo de confiança dos pontos originais. Mostramos no capítulo quatro a métrica usada.

A classe *scipy.interpolate.interp1d* cria uma função interpoladora. Usamos essa função com os valores das abscissas (os valores de tempo) para gerar os valores das ordenadas (temperaturas).

A classe, no método de inicialização de um objeto, verifica os vetores passados como parâmetros e cria internamente uma cópia de cada. O método de inicialização ordena cada vetor seguindo a orientação do primeiro parâmetro (o tempo da medição, neste trabalho). Ajustamos o terceiro parâmetro para a interpolação linear, por isso o método verifica se as condições são aceitas:

1. se os elementos dos vetores são do tipo real;
2. se os vetores têm mais de um elemento; e
3. se não tenta usar nenhum ponto fora dos limites de interpolação.

Quando todos os requisitos são cumpridos, o método calcula os valores interpolados para cada ponto através de uma chamada de um método de classe que calcula a inclinação (a razão entre as diferenças das temperaturas pela diferença entre os tempos) entre pontos imediatamente vizinhos, conforme cálculo mostrado na seção Interpolação Linear, onde y é a temperatura interpolada, x o tempo medido, y_b e y_a são os valores para as temperaturas entre os pontos b e a, assim como x_b e x_a são os valores dos tempos das medições nos pontos b e a, respectivamente. O método retorna um vetor com os valores de temperatura interpolados.

2.2.4 A interpolação Spline Cúbico na biblioteca SciPy

Usamos o subpacote *interpolate* da biblioteca SciPy para gerar a função interpoladora de Spline Cúbico neste trabalho. Criamos um objeto do tipo CubicSpline para gerar a função interpoladora de Spline Cúbico. A criação do objeto recebe dois parâmetros, que neste trabalho foram os vetores com os tempos e as temperaturas escolhidas.

A classe verifica inicialmente as condições de contorno do vetor passado como segundo parâmetro. Caso essas condições estejam favoráveis, o método retorna o vetor do segundo parâmetro.

O objeto, em sua criação, calcula a primeira e a segunda derivadas dos vetores através de produtos matriciais vistos na demonstração do método de interpolação Spline Cúbico. São calculados os coeficientes para a spline que passa pelos pontos de temperatura informados pelo experimento.

2.2.5 Biblioteca Matplotlib

A biblioteca Matplotlib é uma biblioteca voltada para criação de gráficos em duas dimensões, como mencionado em (HUNTER, 2007).

Neste trabalho usamos a sub-biblioteca `pyplot` para a geração dos histogramas, do gráfico dos valores medidos em conjunto com os valores interpolados e o gráfico de estatísticas.

2.2.6 Módulo Matplotlib.TkAgg

O módulo TkAgg serve para a geração da imagem dos gráficos em cada um dos elementos deste trabalho. Os dois histogramas e o gráfico com as medições das temperaturas distribuídas no tempo, além das temperaturas interpoladas pelos diferentes métodos são gerados através deste módulo.

O módulo serve como *backend*, permitindo que o usuário possa interagir com o gráfico (aproximação, gravação da imagem do gráfico, entre outras permissões). Usamos este módulo porque é o recomendado para a geração de gráficos que são inseridos em janelas feitas pela biblioteca TkInter.

2.2.7 Biblioteca TkInter

A biblioteca TkInter é a biblioteca padrão para a geração de interfaces gráficas de usuários. Ela possui vários elementos gráficos, chamados da biblioteca de *widgets*, para a composição de telas e menus de forma rápida, sucinta e amigável pelo desenvolvedor.

2.3 SENSORIAMENTO EM REDE

Nesta seção apresentamos algumas formas de sensoriar grandezas em uma determinada área. Isso compreente o uso de dispositivos sensores (móveis ou não) para medir grandezas físicas. Também tratamos como esses dispositivos podem se conectar por protocolos sem cabeamento, formar uma rede e transmitir informações através dessa rede até que cheguem ao dispositivo destino para devido tratamento dos dados.

2.3.1 Rede de sensores sem fio

Redes de sensores sem fio podem ser usadas para vários tipos de finalidades. Elas podem ser usadas para monitoramento de locais de difícil acesso ao homem (medições de temperatura e abalos sísmicos em vulcões, por exemplo) (CASTILLO-EFFER et al., 2004)

Estas redes são formadas por dispositivos com capacidade de sensoriamento, processamento e comunicação de dados, chamados neste trabalho de *nós sensores* ou *nós*. Estes nós são constituídos de microcontroladores, sensores de finalidades diversas que podem ser acoplados dado o objetivo da monitoração (umidade, calor, luminosidade, distância, entre outros) e uma fonte de energia. A fonte de energia do nó pode ser uma bateria instalada ou energia coletada do meio ambiente por meio de células fotovoltaicas, motores que captem energia através da movimentação dos nós ou de moinhos hidráulicos ou eólicos ou ainda dispositivos que visitem e reabasteçam a carga energética dos nós, entre outros(AKYILDIZ et al., 2002).

Os nós podem ser classificados em dois tipos, quanto à função que desempenham na rede: nós de sensoreamento e estação base. O primeiro serve para o sensoriamento, possuindo um sistema operacional simplificado, especializado em coletar amostras e transmitir os dados através da rede, com baixa taxa de processamento, além da transferência dos dados entre os nós até as estações base. Os nós estação base servem para a recepção dos dados transmitidos entre os nós e transferência desses dados para o computador com a aplicação em execução.

A figura 1 ilustra um modelo de uma rede de sensores sem fio. Os pontos vermelhos dentro do círculo são os nós de sensoriamento da rede, que coletam os dados. Os nós diretamente ligados ao nó externo ao círculo pontilhado são os nós que servem de saída da rede, também chamados de *gateways*. O nó sensor conectado à Internet é a estação base, o nó responsável em transmitir os dados recebidos da rede através da Internet até o computador, chamado de terminal neste trabalho, que faz uso dos dados coletados.

A forma de distribuição e organização da rede depende de seu objetivo. Pode-se usar uma rede com algumas dezenas de nós ou milhares, dada a área a ser coberta pela rede. A dispersão dos nós pode ser de maneira controlada ou de forma aleatória, dependendo da

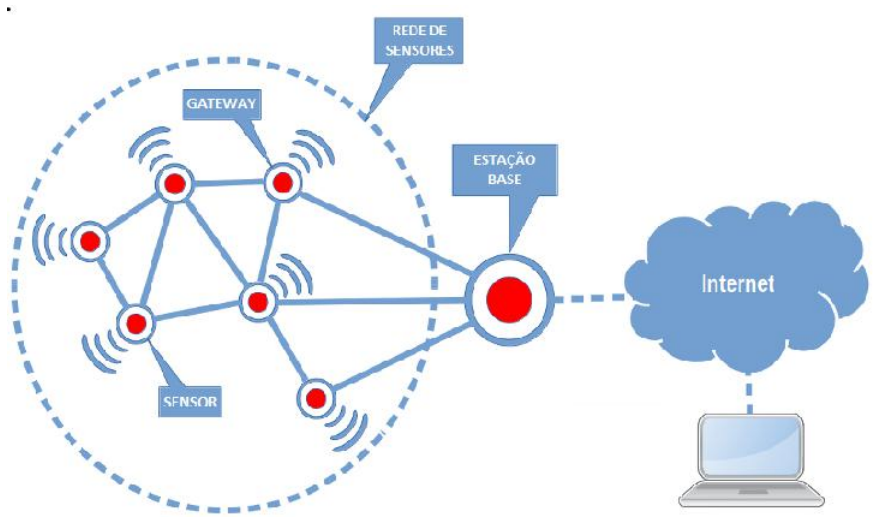

Figura 1 – Exemplo de rede de sensores sem fio Fonte: <https://www.gta.ufrj.br/ensino/eel878/redes1-2018-1/trabalhos-vf/rssf/>

possibilidade de entrada de pessoal para posicionar os nós ou se os nós serão distribuídos, por exemplo, a partir de aviões para cobrir uma área extensa.

É desejável que as redes de sensores sem fio administrem seus recursos para manterem-se ativas, dada a finalidade das mesmas. Elas precisam autogerenciar-se, ou seja, no caso de falha em um nó de transmissão, os outros nós precisam saber reorganizar as rotas de transmissão dos pacotes de dados sem o nó parado para otimizar energia, por exemplo.

Cada nó tem como recurso mais importante, para a manutenção da rede, a quantidade de energia armazenada. É importante implementar protocolos e métodos de coleta e envio dos dados a partir dos nós de medição com o menor gasto de energia possível (AKYILDIZ et al., 2002).

3 PROPOSTA DO TRABALHO

Este trabalho tem a intenção de apresentar ao leitor um uso de rede de sensores sem fio em conjunto com uma aplicação de dois métodos de interpolação. O foco principal é oferecer aos professores da disciplina de Cálculo Numérico uma ferramenta que permita chamar a atenção e despertar a curiosidade dos alunos sobre o tema.

A proposta divide-se em duas vertentes. A primeira é o uso de um *testbed* de rede de sensores sem fio com a intenção de gerar uma massa de dados a partir da coleta de medidas de temperatura de um ambiente, em intervalos periódicos, durante um longo tempo de exposição. Escolhemos a temperatura por ser uma grandeza que oscila lentamente, permitindo assim verificarmos variações suaves na aplicação dos métodos de interpolação.

A segunda é o uso dos dados em um script que, após a leitura da massa de dados, gera dois conjuntos de dados interpolados por dois métodos diferentes, com métricas que nos fornecem um parâmetro para verificar o quão confiáveis são os dados gerados a partir dos métodos de interpolação.

Este capítulo discorre sobre alguns tópicos importantes deste trabalho. Nele explicitamos como é feita a coleta de dados na rede de sensores e como realizamos a sua implementação. Também apresentamos o desenvolvimento da solução para uso pelo professor (*scripts* para a manipulação e visualização dos dados).

3.1 TESTBED DE REDES DE SENSORES

Um testbed de rede de sensores sem fio permite a realização de experimentos em dispositivos reais de forma controlada. Ele é acessível em qualquer momento para a realização da coleta de dados de grandezas reais como temperaturas, pressão atmosférica, luminosidade, entre outras. É possível programar a coleta dos dados a partir de sensores de forma simultânea ou de forma escalonada.

Escolhemos o testbed FIT-IoT (ADJIH et al., 2015) para este trabalho devido à gama diversificada de plataformas de sensores à disposição, assim como a permissão de acesso público aos corpos docente e discente de qualquer univesidade, não apenas para os laboratórios e universidades que os mantém. O FIT-IoT é um conjunto de laboratórios, distribuídos entre alguns *campi* universitários em cidades na França.

Conseguimos inserir o script para os nós sensores a partir da plataforma web do FIT-IoT. Escolhemos o laboratório, o tipo de nó sensor, os nós desejados, o horário da obtenção dos dados, e o intervalo entre as coletas. A proposta é, com as medições, verificar a variação das grandezas em um longo período de tomada de dados para, então, comparar com os dados gerados a partir das implementações de interpolação que aplicamos neste trabalho.

3.2 IMPLEMENTAÇÃO

Nesta seção, descrevemos os programas implementamos para coleta e transmissão dos dados medidos pelos sensores, leitura do arquivo de dados gerado por cada nó sensor, assim como os programas de geração da interface de usuário, análise dos dados e representação gráfica dos resultados.

3.2.1 Programas para coleta dos dados sensoreados

Escrevemos um método para inicialização do nó sensor, onde apenas os componentes necessários para nosso experimento são ativados. Neste trabalho focamos em inicializar apenas os nós, os sensores de temperatura e o serviço de transmissão dos dados para o terminal principal, que armazenaria os dados dos nós.

O nó sensor usado em nosso trabalho — WSN430[1] — permite a execução de métodos de inicialização do nó, captura de dados e gravação dos mesmos em intervalos definidos, e transmissão dos dados coletados por um meio de comunicação sem fio.

Os nós sensores contam com uma biblioteca de funções escritas em C, permitindo a programação de um ou diversos dispositivos de sensoriamento. Eles podem operar cooperativamente, transmitindo os pacotes de dados de um para o outro até que os dados cheguem ao nó destino; ou simplesmente sensoreando e enviando os dados diretamente para o dispositivo conectado ao terminal que irá processá-los[2].

Tomamos como base neste trabalho o arquivo do exemplo de inicialização dos nós sensores fornecido pelo FIT-IoT para a experimentação básica do *testbed*. Seu *script* é explicado no decorrer desta seção.

Destacamos o uso das bibliotecas a seguir:

```
1 #include <io.h>
2 #include <signal.h>
3 #include <stdio.h>
4 #include <string.h>
5
6 #include "clock.h"
7 #include "leds.h"
8 #include "uart0.h"
9 #include "tsl2550.h"
10 #include "ds1722.h"
11 #include "ds2411.h"
12 #include "timerA.h"
13
14 #include "mac.h"
```

Código 3.1 – Bibliotecas utilizadas no sensor

[1] como descrito em <https://www.iot-lab.info/legacy/hardware/wsn430/index.html>
[2] A biblioteca pode ser acessada em <https://github.com/iot-lab/wsn430>

A biblioteca *<stdio.h>* (código 3.1) é uma biblioteca padrão de entrada e saída de dados para a linguagem C. A biblioteca *uart0.h* atua na recepção e transmissão assíncrona dos dados. A biblioteca *ds1722* é a responsável pelo sensor de temperatura. A biblioteca *ds2411.h* atua no controle da interface serial de cada nó sensor. Por fim, a biblioteca *timerA* serve para o controle do tempo corrido dentro do *script*.

As seguintes constantes são usadas:

```
1 #define CHANNEL 17
2 #define TIMER1SEC 10000000
3 volatile int16_t contagemMinutos = 0;
```

Código 3.2 – Constantes e variável de contagem de minutos

Escolhemos fazer as transmissões pelo canal 17 (código 3.2). Há a possibilidade de escolha entre os canais 0 a 20, totalizando 21 canais disponíveis.

Estabelecemos uma constante chamada TIMER1SEC, de valor 10000000. Essa constante serviu para termos o intervalo de um segundo, usado no decorrer do experimento.

Definimos também uma variável chamada *contagemMinutos* que servirá para armazenar a quantidade de minutos que se passaram desde o início da coleta de dados. Ela começa zerada e será incrementada a cada coleta de dados. Todas as medições serão feitas entre intervalos de um minuto.

```
1 static void serial_number()
2 {
3     uint16_t node_addr = (((uint16_t)ds2411_id.serial1)<<8) + (ds2411_id
    .serial0);
4     printf("Unique identifier: %x\n", node_addr);
5 }
```

Código 3.3 – Método de identificação do sensor

O método *serial_number* (código 3.3) recupera dos dados designados a partir da biblioteca *ds2411*, o identificador serial daquele nó e imprime o resultado do mesmo. O número de série do nó sensor é importante para identificar o nó que fez a medição. Esse método é usado no momento de armazenar o valor da grandeza sensoriada no arquivo de medições.

```
1 static void temperature_sensor()
2 {
3     int16_t value_0, value_1;
4
5     value_0 = ds1722_read_MSB();
6     value_1 = ds1722_read_LSB();
7     value_1 >>= 5;
8
9
10    printf("%i|%i.%i\n",contagemMinutos, value_0, value_1);
```

```
11      contagemMinutos++;
12  }
```

Código 3.4 – Leitura da temperatura

O método *temperature_sensor* (código 3.4) faz a leitura do sensor de temperatura DS1722 (DALLAS SEMICONDUCTORS, MAXIM INTEGRATED, 2007). Os bytes mais significativos resultam no valor absoluto da temperatura, enquanto que os bytes menos significativos fornecem os dados para a parte decimal da temperatura. Um detalhe importante é que os bytes menos significativos sofrem um deslocamento de 5 unidades para que apenas o valor da primeira casa decimal apareça na leitura. Após a leitura, a função registra o valor da variável *contagemMinutos* e o valor da temperatura, com os valores absoluto e decimal. A variável *contagemMinutos* é incrementada de uma unidade em seguida.

```
1  static void hardware_init()
2  {
3      // para o timer
4      WDTCTL = WDTPW + WDTHOLD;
5
6      // Ajuste de MCLK a 8MHz e SMCLK a 1MHz
7      set_mcu_speed_xt2_mclk_8MHz_smclk_1MHz();
8      set_aclk_div(8); // ACKL is at 4096Hz
9
10
11     // Inicializacao dos sensores
12     ds1722_init();
13     ds1722_set_res(12);
14     ds1722_sample_cont();
15
16
17     // Inicializa o serial
18     ds2411_init();
19
20     // Inicializa o CSMA da camada de radio
21     mac_init(CHANNEL);
22     mac_set_rx_cb(mac_rx_isr);
23     mac_set_sent_cb(mac_tx_done_isr);
24     mac_set_error_cb(mac_tx_fail_isr);
25
26     // Inicializa o UART0
27     uart0_init(UART0_CONFIG_1MHZ_115200);
28
29     // Habilita as interrupcoes
30     eint();
31
32
```

```
33 }
```

Código 3.5 – Método de inicialização do hardware do sensor

O método *hardware_init* (código 3.5) faz a inicialização do nó sensor. Neste método o timer do nó é configurado, o processador é ajustado para a frequência de 1MHz, os leds são inicializados, os sensores de temperatura são inicializados, assim como os radiotransmissores, o dispositivo de radiotransmissão assíncrona UART0 e os timers dos leds do nó sensor.

```
1  int main( void )
2  {
3      hardware_init();
4      __delay_cycles(TIMER1SEC*60);
5
6      while (1) {
7          temperature_sensor();
8          __delay_cycles(TIMER1SEC*60);
9
10     }
11     return 0;
12 }
```

Código 3.6 – Método principal do sensor

No fim temos um método *main* (código 3.6), responsável por disparar a execução do sensor. Primeiro inicializamos o nó, usando um método de espera chamado *__delay_cycles()*, que recebe como parâmetro a quantidade de segundos que queremos que o nó sensor aguarde de forma ociosa. Neste trabalho usamos a constante TIMER1SEC e multiplicamos por 60, a fim de termos o intervalo de 1 minuto. Em seguida imprimimos uma mensagem para estabelecer o início das medições das temperaturas e, de forma cíclica, o nó sensor deveria medir a temperatura e mais uma vez executar o método *__delay_cycles()*, usando o intervalo de TIMER1SEC multiplicado por 60.

A função *temperature_sensor()* (código 3.4) realiza as medições das temperaturas. Ela faz as medições dos valores absolutos e decimal da temperatura, dadas as variáveis ligadas aos sensores.

O valor absoluto da temperatura é lido pelo sensor usando o comando *ds1722_read_MSB()*, que está na bilbioteca *ds1722.h*[3], incluída no projeto do nó sensor. Já o valor decimal da temperatura é lido pelo sensor através do comando *ds1722_read_LSB()*, que faz a leitura do byte menos significativo, também incluído na biblioteca acima citada. Precisamos ajustar o valor do byte menos significativo para mostrar o valor da medição em decimal de forma correta, por isso fizemos o *right shift* de cinco casas decimais no código.

[3] Conforme em <https://senstools.gforge.inria.fr/doxygen/ds1722_8h.html>

A função então imprime o valor dos minutos decorridos desde o início do experimento e o valor da temperatura (absoluto e decimal). Acrescentamos um dígito na variável de contagem de minutos e o método termina.

O nó faz a medição e envia para o terminal (estação base) o número de série do nó na rede, o tempo passado desde o início das medições (estabelecido no método principal) e a medida da temperatura naquele momento. O nó sensor aguarda um minuto após cada medição para, em seguida, fazer a próxima medição e enviar ao terminal. Usamos o método *_delay_ cycles(parametro)*, pertencente à biblioteca do nó sensor, para criar um intervalo entre as medições.

Essa instrução deixa o nó sensor em espera pela quantidade de ciclos definida no parâmetro. O valor do parâmetro é arbitrário, porém como usamos um nó sensor com um clock de 10MHz, usamos a constante **TIMER1SEC** (código 3.2), que, combinada ao uso deste método, nos forneceram o intervalo de 1 segundo.

Inicialmente decidimos que faríamos as medições a cada segundo, porém os experimentos mostraram que não conseguiríamos reproduzir um intervalo prolongado de medições. Dado isto, preferimos fazer as medições a cada minuto. Com isso temos, no método principal, a função *_delay_ cycles()* que recebe como argumento o produto da constante **TIMER1SEC** por 60 (código 3.6), resultando no tempo de um minuto por medição.

Após a inicialização do nó sensor, na interface de comandos do testbed, inserimos um comando de direcionamento da saída do que seria impresso no shell para um arquivo de texto, com o nome do nó sensor e a data das leituras.

Usamos o comando a seguir para a plataforma do FIT-IoT:

```
nc wsn430-228.strasbourg.iot-lab.info 20000 >> 201911262300.tat
```

Transferimos a saída do nó sensor pela porta 20000 para o arquivo em questão com este comando.

Cada linha do arquivo é composta do nome do nó sensor, o tempo em minutos a partir do início do experimento e a medição da temperatura naquele momento. Um exemplo ilustrativo é mostrado abaixo:

```
1575335916.909697;wsn430-228;5|28.0
```

Os dois números iniciais são o identificador do nó, a palavra entre ponto e vírgula é o nome do nó sensor, o número após o nome do nó é o momento da detecção e o número após o caracter '|' é a temperatura medida.

3.2.2 Programas para a interface gráfica com o usuário

Optamos pela biblioteca Tkinter para a criação da interface de saída do programa, de forma que nela podemos inserir os elementos gráficos para facilitar a interação do usuário (professor e alunos).

Criamos, para cada método de interpolação, um histograma de forma a permitir a análise dos dados gerados pela interpolação dos dados. Criamos também uma janela para

inserir estes histogramas e o gráfico das interpolações e as medições reais. Inserimos cada elemento da tela diretamente na janela principal, com exceção do gráfico de interpolações.

3.2.3 Programa para manipulação dos dados

Criamos duas classes para trabalhar com os vetores de tempo e temperatura. Inserimos os valores dos dados medidos em cada objeto, de forma que eles se mantivessem organizados pelo índice dos vetores.

```
1 class Tempo:
2     #define a classe tempo
3 
4     def __init__(self):
5 
6         self.x = []
7         self.step = []
8         self.limitadoLinear = []
```

Código 3.7 – Classe Tempo

Desenvolvemos uma classe chamada Tempo. Esta classe conta com os seguintes atributos: o vetor com os horários das medições, o valor do intervalo entre as medições para a interpolação (aqui chamado de "passo de interpolação") e um vetor para armazenar os horários das medições de forma limitada, ou seja, todos os horários que estiverem dentro do último passo. Por exemplo, podemos ilustrar como dez medições de um segundo cada. Se o passo for três, teremos no vetor *limitadoLinear* os horários da segunda, quinta e oitava medição. Se o passo for quarto, teremos os horários da terceira e da sétima medição.

O passo é um intervalo maior que aquele que fizemos as medições. Pode ser 3, 4, 10 60 minutos. Este passo serve para que usemos os métodos de interpolação e encontremos as funções interpoladoras, que passam pelos pontos especificados ao definirmos o passo.

```
1 from scipy import interpolate
2 import numpy as np
3 class Temperatura:
4     #define a classe tempo
5 
6     def __init__(self):
7 
8         self.y = []
9         self.step = []
10        self.limitadoLinear = []
11        self.interpolada = []
12        self.spline = []
13 
14 
15    def calculaDiferencaSpline(self):
```

```
16          return (self.spline - self.y)
17
18      def calculaDiferencaLinear(self):
19          return (self.interpolada - self.limitadoLinear)
20
21      def setTemperaturasPorStep(self,step):
22          for k in range(0,len(self.y),int(step)):
23              self.step.append(self.y[k])
24
25      def getDesvioPadraoLinear(self):
26          return self.calculaDiferencaLinear().std()
27
28      def getDesvioPadraoSpline(self):
29          return self.calculaDiferencaSpline().std()
30
31      def setTemperaturasInterpoladas(self,tempo):
32
33          #cria a funcao interpoladora linear apenas nos pontos de Step
34          f=interpolate.interp1d(tempo.step,self.step,kind='linear')
35
36          #cria a funcao interpoladora Spline cubico para cada ponto, a
        cada step segundos
37          f2=interpolate.CubicSpline(tempo.step,self.step)
38
39          #array de temperaturas interpolado, a step segundos,
        interpolacao linear
40          self.interpolada=f(tempo.limitadoLinear)
41
42          #temperaturas interpoladas para todos os pontos, ou seja, (tempo
        ,temperaturaInterpolada)
43          self.spline=f2(tempo.x)
44          #print("Y spline cubico: ",len(temperatura.spline))
45
46      def retornaMetricas(self,differences,step):
47      #retorna dicionario com as metricas necessarias (media, mediana,
        amplitude, desvio padrao e variancia)
48
49          metricas={}
50          metricas["passo"]=step
51          metricas["media"]=np.mean(differences)
52          metricas["mediana"]=np.median(differences)
53          metricas["amplitude"]=differences.max()-differences.min()
54          metricas["desvioPadrao"]=differences.std()
55          metricas["variancia"]=differences.var()
56
57          return metricas
```

Código 3.8 – Classe Temperatura

Criamos a classe Temperatura, que possui como atributos o vetor de temperaturas medidas, o valor do passo, o vetor das temperaturas separadas pelo passo, o vetor das temperaturas interpoladas pela interpolação linear e o vetor das temperaturas interpoladas usando Spline Cúbico. A classe possui métodos que nos ajudam no processo de geração dos vetores para as diferentes interpolações.

O método *calculaDiferençaSpline* retorna um vetor de diferenças entre os valores das temperaturas interpoladas por Spline Cúbico e o vetor das temperaturas medidas. Neste vetor teremos o valor zero em todos os pontos escolhidos para a criação da função interpoladora.

O método *calculaDiferencaLinear* retorna um vetor com as diferenças entre os valores das temperaturas interpoladas pela função interpoladora linear e o vetor das temperaturas medidas. De forma similar ao método anterior, os valores serão zerados nos pontos que usamos para gerar a função interpoladora.

```
1  def importaArquivo():
2      data=open(filedialog.askopenfilename(),"r")
3
4      #criacao da lista para inserir os segundos e as temperaturas
5      pointVec=[]
6      tempoX=[]
7      temperaturaY=[]
8
9      #Recupera o tempo em segundos e a temperatura, cria um objeto do
       tipo Point e adiciona a
10     #pointVec a cada linha lida.
11     for line in data:
12         temp1,temp2=line[line.find("228;")+4:len(line)-1].split("|")
13
14         tempoX.append(float(temp1))
15         temperaturaY.append(float(temp2))
16     data.close()
17     tempoX=np.array(tempoX)
18     temperaturaY=np.array(temperaturaY)
19
20     return tempoX,temperaturaY
```

Código 3.9 – Método importaArquivo

Desenvolvemos uma função que faz a coleta dos dados a partir do arquivo informado pelo usuário na inicialização do programa. A função percorre o arquivo, linha por linha, até a última linha do arquivo. Cada linha contém o nome do nó sensor, o momento da medição e a temperatura medida. A função grava em duas listas distintas o tempo da

medição e a temperatura detectada pelo nó sensor. Em seguida, após fechar o arquivo, as listas são convertidas em vetores do tipo *narray* e então retornadas pela função.

```
1 tmp_tempoX, tmp_temperaturaY = importaArquivo()
2
3 #passo para criar os pontos de um unico passo
4 step = novoPasso()
5
6 #criacao dos objetos de tempo e temperatura
7 tempo = Tempo()
8 temperatura = Temperatura()
```

Código 3.10 – Criação de instâncias de Tempo e Temperatura

O programa principal cria dois objetos, um da classe Tempo e um da classe Temperatura, após a leitura dos dados do arquivo. O vetor com as medidas de tempo é copiado para o atributo x do objeto do tipo Tempo e o vetor das temperaturas é passado para o atributo y do objeto do tipo Temperatura.

```
1 tempo.x = tmp_tempoX
2 temperatura.y = tmp_temperaturaY
3
4 #pega os intervalos limitados
5 tempo,temperatura=criaBaseLimitada(tempo,temperatura,step)
6
7 #cria um array para adicionar os valores das temperaturas a cada step
8 temperatura.setTemperaturasPorStep(step)
9
10 #cria as tempertaturas interpoladas
11 temperatura.setTemperaturasInterpoladas(tempo)
12 setDadosEstatisticas(tempo, temperatura)
```

Código 3.11 – Geração das temperaturas interpoladas

Notamos que surgiu um erro ao usar a função de interpolação linear quando tentamos interpolar em todos os pontos da amostra. Não era possível criar a função interpoladora com pontos além dos limites dos pontos. A solução foi criar um método que armazene, dado o salto, os valores de tempo e temperatura nos objetos. O método busca o índice da última temperatura armazenada dado o salto no vetor de tempo e usa esse índice para limitar a quantidade de temperaturas coletadas no vetor de temperaturas original. Os valores para a o tempo, dado o salto, são armazenados no atributo *limitadoLinear* do objeto do tipo Tempo e as temperaturas são armazenadas no atributo *limitadoLinear* no objeto do tipo Temperatura. Chamamos esse método de *criabaselimitada*. A interpolação linear é feita nesses pontos, porém pode ser estendida para pontos entre e além dos pontos escolhidos. O método retorna os objetos do tipo Tempo e Temperatura modificados, com os atributos para a criação dos objetos interpoladores preenchidos.

O script principal segue com o uso do método *setTemperaturaPorStep*, que coleta, a partir da primeira temperatura até o final do vetor, temperaturas a cada salto. Por exemplo, o usuário pode escolher coletar os dados a cada intervalo de 5 minutos. Então serão coletados os dados da primeira, da sexta, da décima primeira medições e assim por diante. O novo vetor é armazenado no atributo *step*.

Com os objetos dos tipos Tempo e Temperatura com os atributos *limitadoLinear* e *step* preenchidos, seguimos com o método *setTemperaturasInterpoladas*, que faz a criação dos objetos interpoladores e atribui ao objeto do tipo Temperatura dois vetores, um com os valores das temperaturas interpoladas linearmente e outro com os valores das temperaturas interpoladas por Spline Cúbico. Os valores interpolados são calculados inclusive nos pontos que passamos para criação do objeto. Como esperado, os valores das temperaturas interpoladas, nos pontos usados para a criação das funções, são coincidentes com os valores medidos.

Segue o pseudocódigo do script principal.

```
1 vetorTemporarioTempo,vetorTemporarioTemperatura = importaDados()
2 step = recebeOPassoDoUsuario()
3 tempo.x = vetorTemporarioTempo
4 temperatura.y = vetorTemporarioTemperatura
5 tempo, temperatura = criaVetoresLimitadosParaInterpLinear(tempo,
    temperatura)
6 temperatura.recuperaTempertarurasPorStep(step)
7 temperatura.recuperaTempertarurasInterpoladas(tempo, temperatura)
8 janela = criaJanela()
9 janela = criaMenus()
10 janela = criaHistogramaInterpolacaoSplineCubico()
11 janela = criaGraficoDeInterpolacoes()
```

Código 3.12 – Pseudo Código do Script Principal

3.2.4 Programas para geração de gráficos

Criamos uma classe para a geração dos gráficos na tela principal. Nomeamos a classe como *Grafico.py* e ela é chamada para a criação do objeto que gera os histogramas e o gráfico das interpolações.

Foram criados objetos do tipo *Grafico* para cada elemento gráfico da janela no código principal, um para o histograma de interpolação linear, um para o histograma de interpolação de Spline Cúbico e um para o gráfico com os valores interpolados. Eles foram inseridos na janela principal logo após as suas criações através de métodos de geração dos histogramas e do gráfico principal.

Usamos a biblioteca *Tkinter* para a criação das janelas e a biblioteca *Matplotlib* (HUNTER, 2007) para a criação dos gráficos. Cada janela funcionou como um *container* para os gráficos. A biblioteca *Matplotlib* tem como finalidade criar gráficos 2D e para isso ela

oferece um módulo chamado *pyplot*. Consideramos o uso deste módulo pela facilidade de interação com o módulo NumPy da biblioteca SciPy.

Criamos um objeto da classe *matplot.pyplot.figure* para a inserção dos pontos interpolados e dos pontos reais, para a geração do gráfico desejado. Criamos também as seguintes representações no gráfico: os pontos reais, os pontos interpolados pelo método de interpolação linear, uma curva representando o método de interpolação linear, os pontos interpolados pelo método de interpolação de Spline Cúbico, uma curva representando o método de interpolação de Spline Cúbico e os pontos escolhidos para serem os pontos entre os saltos.

Usamos para a inserção dos pontos no gráfico — valores interpolados e valores reais — um objeto da classe *matplotlib.pyplot.errorbar*. A classe recebe para a inicialização do objeto os seguintes parâmetros: o vetor das abscissas, o vetor das ordenadas, o valor para o erro das abscissas (este não foi utilizado neste trabalho), o valor para o erro das ordenadas, o formato do gráfico e o rótulo do gráfico.

Usamos neste trabalho o desvio padrão da diferença entre os valores reais de temperatura e os valores interpolados das temperaturas como o nosso valor de erro das ordenadas. Essa medida foi calculada apenas para os pontos, de forma manter o gráfico o mais limpo possível.

Decidimos inserir nosso gráfico de interpolações em um *backend* chamado *FigureCanvasTkAgg*. Cada gráfico da janela é inserido em um objeto deste tipo. Nossa proposta foi de criar uma janela interativa de forma que o ususário possa alterar os gráficos de forma intuitiva, além de permitir a visualização e navegação em cada um dos gráficos. Existe para cada elemento gráfico um menu, que permite, além de ampliação ou distanciamento do gráfico, salvar em arquivo a imagem ampliada ou distanciada.

3.2.5 Programas para geração de histograma

Criamos para cada elemento gráfico da tela um objeto da subclasse *Figure*. Este tipo de objeto atua como container na janela dos histogramas.

Após a criação do objeto do tipo *Figure* incluímos o histograma no objeto através do comando *add_ subplot(arg).hist(parâmetro 1,parâmetro 2)*, onde a variável *arg* corresponde aos valores na coluna, linha e índice onde foi acrescentado o gráfico. No caso dos histogramas, foram inseridos na primeira linha, primeira coluna, primeiro índice. O valor da variável é ajustado para 1,1,1.

Neste comando também foi realizada a inserção do histograma, onde o parâmetro 1 corresponde à fonte dos dados e o parâmetro 2 corresponde ao agrupamento de valores. A fonte dos dados para o histograma de interpolação linear é o vetor com os valores gerados pela função de interpolação linear. O vetor com os valores interpolados pela função de interpoalação de Spline Cúbico foi usado para a criação do histograma para a interpolação

de Spline Cúbico. O agrupamento entre valores foi ajustado para o automático, sem a necessidade de limitações.

Cada objeto do tipo *Figure* foi inserido em um *backend* do tipo *FigureCanvasTkAgg*. Esta classe serve para inserir gráficos gerados pela biblioteca *pyplot*, permitindo também a criação de um menu de navegação. É possível ampliar ou distanciar o gráfico, além de gerar um arquivo de imagem com o estado momentâneo do gráfico. Os parâmetros passados para o objeto foram o gráfico fonte (neste caso os objetos onde foram criados os histogramas) e o local onde ficaria alocado este objeto, neste trabalho, o objeto da janela. Após a inserção do título de cada histograma, usamos o método *draw()* para desenhar o histograma no objeto do tipo *FigureCanvasTkAgg*.

Por fim, usamos o método *get _tk _widget().pack(parametros)*. Este método cria um *widget* para a janela, de forma a gerar um *container* para o objeto do tipo *FigureCanvasTkAgg*, que contém todos os objetos componentes do histograma. Ajustamos os parâmetros do método *pack()* para inserir o *widget* do lado esquerdo da janela, se ajustando à tela conforme necessite.

3.2.6 Scripts para criação dos menus

Escrevemos o código dos menus diretamente no programa principal. Acrescentamos os menus logo após a criação da janela, antes da criação dos gráficos.

Foram inseridos os seguintes Menus: **Carregar novo arquivo**, que abre uma janela para buscar pelo novo arquivo para a interpolação usando um navegador de arquivos; **Novo passo**, que permite a troca do salto entre os valores para interpolação; **Estatísticas**, que permite a geração do gráfico que mostra as comparações entre as métricas para as interpolações linear e de Spline Cúbico para os saltos informados; e **Sair**, que fecha todas as janelas.

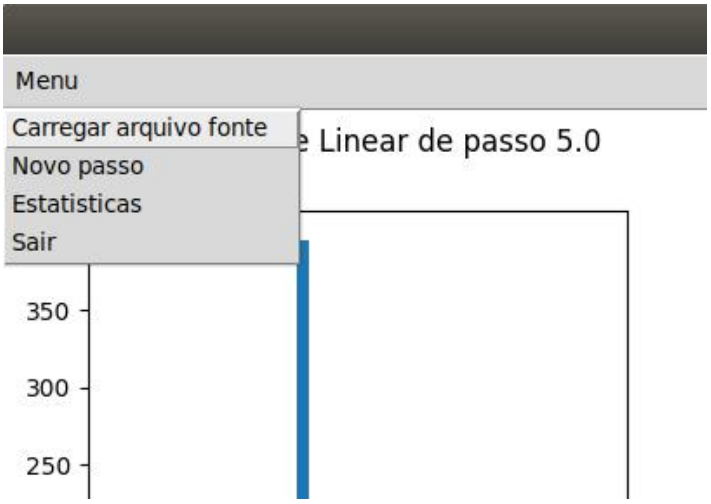

Figura 2 – Menu da aplicação

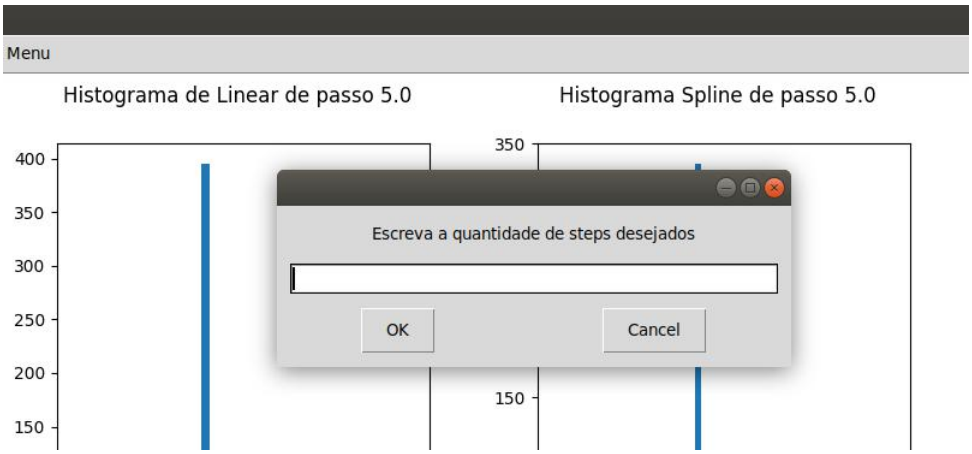

Figura 3 – Execução do programa após seleção do menu *Estatísticas*

3.3 CÓDIGOS

Os *scripts* utilizados para este trabalho podem ser encontrados na URL <https://github.com/jorgejaujr/tccFinal> .

4 AVALIAÇÃO

Neste capítulo avaliamos o projeto com um conjunto de métricas no propósito de validá-lo ou verificar se há alguma abordagem melhor para o experimento.

4.1 DESCRIÇÃO DOS EXPERIMENTOS

Fizemos medições de temperaturas nos intervalos de um segundo, um minuto e dez minutos. Dentro do intervalo de leituras a cada segundo foi possível fazer 5037 medições. Quando fizemos uma leitura a cada um minuto, obtivemos 1152 leituras. A cada 10 minutos forma feitas 10 leituras.

A cada leitura o nó sensor enviou um texto para a instância do terminal que foi aberta no início das leituras. Usamos um comando que redirecionava a saída dos valores enviados ao terminal a um arquivo texto, onde cada linha continha as seguintes informações, separadas por ponto e vírgula: o nome de descrição do sensor na rede e o tempo corrido desde o início do experimento (ou seja, o momento da medição), seguido da temperatura medida, separados pelo caractere |.

O nome de cada arquivo texto extraído do servidor começa com o padrão AAAAMMDDHHHH.txt, onde AAAA são os dígitos do ano, MM são os dois dígitos correspondentes ao mês, DD são os dois dígitos correspondentes ao dia e HHHH são os dígitos correspondentes aos dígitos de horas e minutos do início das medições, começando sempre em 00 segundos.

4.2 DESCRIÇÃO DAS MÉTRICAS

Utilizamos cinco métricas para verificar o quanto os resultados interpolados se aproximavam aos valores medidos. Estas métricas foram: **média** das amostras, a **mediana**, a **variância**, o **desvio padrão** e a **amplitude**. Utilizamos essas métricas por serem importantes na avaliação de quão estáveis estão os dados interpolados (amplitude), o erro médio por se usar a média ao representarmos um dos valores interpolados (desvio padrão), o quanto os dados se afastam da média (variância) e o erro de se usar a média central dos dados interpolados (mediana).

Elas são obtidas a partir de um vetor dos módulos das diferenças entre os valores medidos e os valores interpolados. Os métodos usados foram *numpy.mean()*, *numpy.median()*, *vetor.var()* e *vetor.std()*, respectivamente.

Utilizamos os métodos estatísticos da biblioteca *Numpy* para gerar essas métricas, com exceção a amplitude, onde fizemos a diferença entre o maior e o menor valor do vetor.

Exceto pela mediana dos dados lidos e interpolados, como pode ser visto em 6, os valores interpolados e reais se comportam de forma próxima, o que nos encoraja a concluir que os métodos de inferpolação são boas aproximações para os dados reais.

Conforme o script armazenado no repositório apontado na seção 3.3, temos os seguintes métodos: O método *quantidadeDeSteps()*, dentro do arquivo *molule.py*, solicita a quantidade de novos saltos a serem usados para a geração das métricas. O usuário digita a quantidade e esse número é retornado pelo método.

O método *verificaStep(step)* verifica se o usuário digitou os valores 0, 1 ou 2 e retorna verdadeiro se não for nenhum deles ou falso se for um deles. Optamos pela criação desse método para evitar saltos que não funcionam para a criação das funções interpoladoras.

O método *setDadosEstatisticas(tempo,temperatura)* possui dois parâmetros. Estes parâmetros são os objetos do tipo *Tempo* e do tipo *Temperatura*. Estes objetos são armazenados em um dicionário dentro do módulo *module.py*. Esta função é chamada após a geração dos dados interpolados.

O método *getDadosEstatisticas()* retorna o dicionário *dados*, um dicionário global para armazenar os dados a partir do script principal para a geração das estatísticas.

O método *novosPassos(quantidade)* recebe o parâmetro que contém a quantidade de passos a serem armazenados. Ele possui um vetor chamado *steps*, no qual serão armazenados todos os saltos digitados individualmente pelo usuário. Optamos por não permitir saltos iguais, uma vez que os resultados seriam coincidentes. Usamos o método *verificaStep* a fim de evitar que o usuário digite um salto que não permitiria a interpolação.

O método *estatisticas()* faz uma chamada ao método *novosPassos()*, tendo como argumento uma chamada ao método *quantidadeDeSteps()*. O vetor com os passos é passado para variável *steps*. Na sequência recuperamos o dicionário com os objetos de tempo e temperatura e após criamos referências a esses objetos. São criadas listas para armazenar os dados de média, mediana, amplitude, variância e desvio padrão, para a interpolação linear e para a interpolação de Spline Cúbico. Após isso fazemos, para cada um dos itens no vetor de passos *steps*, mantemos apenas os vetores com os dados originais nos objetos Tempo e Temperatura. Fazemos o processo de geração de base limitada usando o passo do vetor *steps*, a criação dos objetos das funções interpoladoras, a criação dos dados interpolados para os métodos de interpolação linear e Spline Cúbico.

O método segue com os métodos do objeto Temperatura chamados *calculaDiferencaLinear* e *calculaDiferencaSpline*. Essas diferenças são usadas para criar dicionários de métricas, um para cada método de interpolação, que são inseridos nas listas *metricasLinear* e *metricasSpline*, um dicionário por step. Por fim, para cada dicionário dentro da lista de metricasSpline e metricasLinear, recuperamos e inserimos nas listas correspondentes os valores de média, mediana, amplitude, desvio padrão e variância para cada um dos métodos.

Fazemos em seguida a criação de uma janela para a exibição de um gráfico usando

como eixo de abscissas para todos os gráficos o vetor dos saltos e como ordenadas, respectivamente, os valores de média, mediana, amplitude, desvio padrão e variância, primeiro para os métodos de interpolação de Spline Cúbico e depois para o método de interpolação linear.

Finalmente exibimos a janela com o gráfico dos dados gerados.

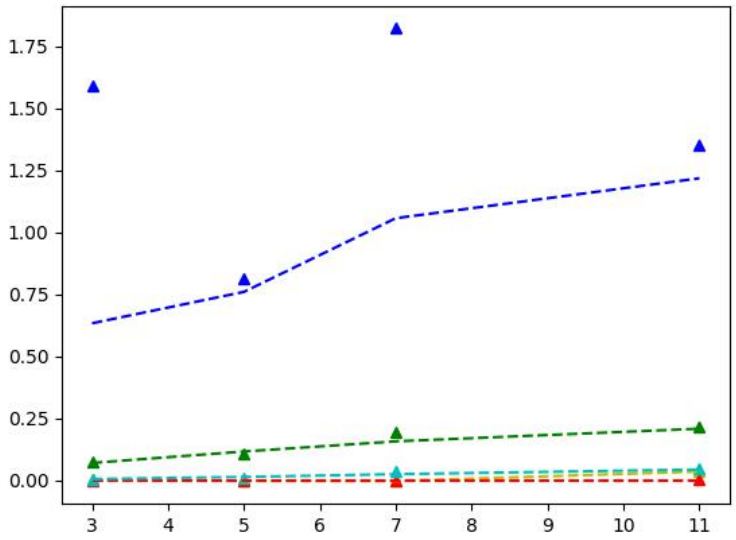

Figura 4 – Exemplo de gráfico gerado a partir dos passos 3, 5, 7 e 11, comparando os métodos.

Na figura 4 mostramos os resultados estatísticos para os passos de valores 3, 5, 7 e 11. Os gráficos em triângulos correspondem às métricas para a interpolação Spline Cúbico, enquanto que os gráficos em linha tracejada correspondem aos resultados para a interpolação linear. Os resultados estão distintos nas cores *amarelo* para a *média*, *vermelho* para a *mediana*, *azul* para a *amplitude*, *verde* para o *desvio padrão* e *ciano* para a *variância*.

Na figura 6 comparamos os resultados estatísticos para os passos de valores 3, 5, 7 e 11 em paralelo com os dados reais obtidos. Assim como no gráfico anterior, os dados em triângulos correspondem às métricas para a interpolação Spline Cúbico, os gráficos em linha tracejada correspondem aos resultados para a interpolação linear e, neste gráfico, as circunferências são correspondentes aos dados reais. Os resultados estão distintos nas cores *amarelo* para a *média*, *vermelho* para a *mediana*, *azul* para a *amplitude*, *verde* para o *desvio padrão* e *ciano* para a *variância*.

Na figura 8 vemos um exemplo de histograma gerado para os dados interpolados por Spline Cúbico com o passo 5. Temos uma distribiução muito grande das medidas cerca do ponto central, porém se afasta do modelo normal de distribuição.

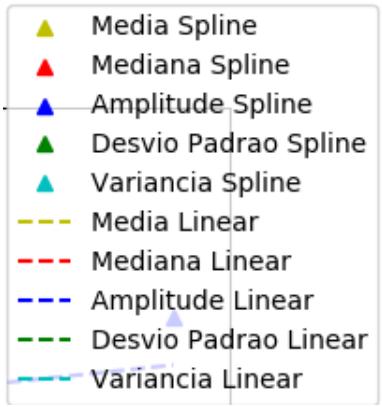

Figura 5 – Legenda para o gráfico gerado a partir dos passos 3, 5, 7 e 11, comparando os metodos.

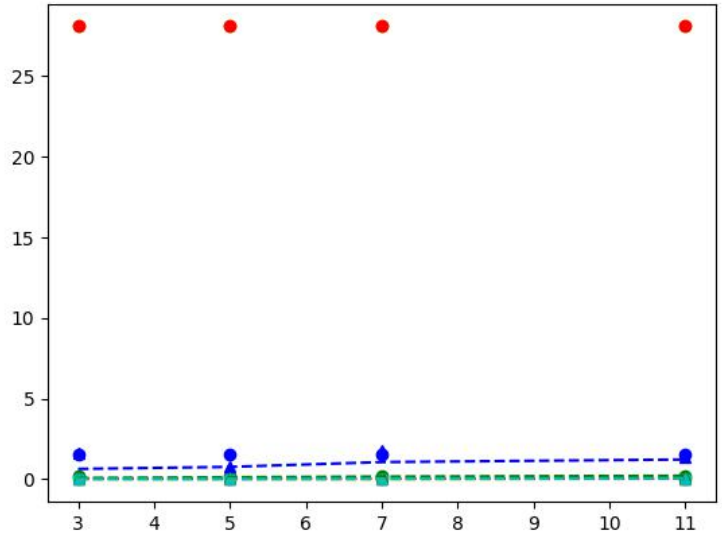

Figura 6 – Exemplo de gráfico gerado a partir dos passos 3, 5, 7 e 11, comparando os resultados dos métodos de interpolação com os valores reais.

4.3 DESCRIÇÃO DO AMBIENTE DE EXPERIMENTAÇÃO

Os dados coletados neste trabalho foram medições de temperaturas. A RSSF utilizada está em um *testbed* chamado ICube, laboratório pertencente à Universidade de Strasbourg (França), associada ao FIT IoT-LAB.

4.3.1 Tedtest - IoT-Lab

O FIT IoT-Lab (uma parte do *FIT - Future Internet of Things*) <https://www.iot-lab.info/> é um banco de ensaio acadêmico e científico, um conjunto de 6 laboratórios distribuídos em cidades na França. Ele conta com 1786 sensores de baixo consumo em rede,

Figura 7 – Legenda para o gráfico gerado a partir dos passos 3, 5, 7 e 11, comparando os resultados dos métodos de interpolação com os valores reais.

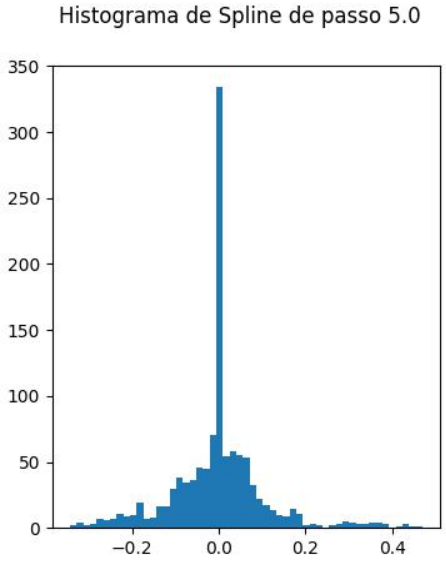

Figura 8 – Exemplo de histograma gerado

dispostos em diferentes tipos de distribuições, de forma a monitorar e analisar diferentes medições de dados, como pressão, temperatura, luminosidade, umidade, entre outros. É possível fazer avaliações nos dados coletados de forma rápida e eficaz.

4.3.2 O banco de ensaio ICube

Este banco de ensaio, também conhecido como Laboratório ICube, está localizado na Universidade de Strasbourg. (ADJIH et al., 2015) <https://www.iot-lab.info/legacy/

deployment/strasbourg/index.html>. A escolha por este laboratório se deu pelo fato de todos os sensores do tipo WSN430 estarem exclusivamente nele. A instalação possui os sensores dispostos em formato de cubo, onde cada nó (exceto dez nós móveis) fica alocado dentro do volume do cubo.

Figura 9 – Foto do interior do ICube com sensores WSN430 Fonte: <https://www.iot-lab.info/legacy/deployment/strasbourg/index.html>

4.3.3 O nó sensor WSN430

Utilizamos em todos os experimentos o nó sensor WSN430 [1]. Trata-se de um equipamento de baixo gasto de energia que pode ser disposto de maneira ad hoc em um ambiente, podendo permanecer em atividade por tempo prolongado.

Sua arquitetura é baseada na plataforma MSP430F1611 (TEXAS INSTRUMENTS, 2002). Essa plataforma conta com um microcontrolador MSP430F1611, com 48kb de armazenamento em memória flash e 10kb de memória RAM. O sensor possui também um conjunto de sensores, como o de temperatura (o sensor DS1722), luminosidade (um sensor do tipo TSL2550) e de áudio. A conexão deste dispositivo em rede sem fio pode ser feita através de uma antena. As duas versões do sensor se diferem pela configuração da antena. A versão 1.3b possui uma interface de rádio CC1100, que se comunica na frequência de 868MHz. A versão 1.4 possui uma interface de rádio CC2420 no padrão IEEE802.15.4, que se comunica na faixa de 2.4GHz.

4.4 MÉTODO DE AQUISIÇÃO DE DADOS

Neste trabalho cada nó sensor mede a temperatura do ambiente, a cada intervalo entre as medições. Esses parâmetros são estabelecidos na programação enviada ao nó sensor.

[1] Conforme descrição do dia 19 de outubro de 2019 em https://github.com/iot-lab/iot-lab/wiki/Hardware-Wsn430-node

Após cada medição, o nó sensor envia à estação base a temperatura, que é gravada em um arquivo para processamento posterior.

A aquisição dos dados se fez a partir dos nós sensores e o *firmware* escrito para sua execução. Uma vez que o nome do nó, o tempo de medição e a temperatura são coletados, estes dados são enviados ao computador, que registra em um arquivo de texto. Criamos um arquivo individual para a recepção das medições por nó sensor.

Usamos o serviço de envio de arquivos transfer.sh <https://transfer.sh/> para a transferência dos arquivos com as medições das temperaturas. Este serviço fez o envio dos dados entre o terminal linux do *testbed* e um repositório acessado por http, permitindo o *download* dos arquivos. Por sua vez, o pesquisador pode baixar os arquivos para o computador local com os scripts de geração dos gráficos e análise dos dados. Escolhemos usar essa plataforma pela necessidade de enviar os arquivos com os dados do servidor do banco de ensaio, a partir de uma interface de shell, para o terminal de recepção.

4.5 PROCESSAMENTO DOS DADOS

Os dados, após coleta pelo WSN430, são enviados ao terminal, que os armazena em um arquivo por nó sensor. Os dados são armazenados em um vetor de objetos da classe *Point*. Após esse armazenamento os valores para o horário de medição e a temperatura medida são separados em dois vetores. Em seguida criamos dois novos vetores onde, a partir da primeira medição, armazenamos os valores para o horário de medição e o valor da temperatura, saltando os pontos dado um valor de salto entre pontos, escolhido pelo usuário. Estes novos vetores servem para a criação das funções interpoladoras. As funções interpoladoras geram novos pares (horário de medição, temperatura interpolada), que são usados para gerar o gráfico de valores interpolados na interface gráfica.

4.6 RESULTADOS E AVALIAÇÃO

Nesta seção analisamos os dados do gráfico 6 e verificamos o quanto nossos dados interpolados se aproximam dos valores das leituras, dadas as métricas estabelecidas neste trabalho. Fizemos 1156 leituras dentro do intervalo de exposição apresentado neste trabalho.

Na figura 19 temos uma visão geral dos dados medidos, dos dados interpolados pelos dois métodos de interpolação deste trabalho, as curvas geradas pelos métodos e as margens de erro dos valores interpolados. Pode-se verificar cada método isoladamente nas figuras 21 (método de interpolação linear) e 23 (método de spline cúbico).

Pode ser visto nas figuras 20 e 24 uma visão recortada dos dados reais, dos dados interpolados pelos métodos de interpolação linear e spline cúbico, respectivamente, da curva que cada método gera e das margens de erro dos dados interpolados.

Verificamos, conforme mostram as visões dos gráficos, que as médias e medianas possuem valores constantes para os valores reais, independente dos passos. Notamos que os valores de média e mediana para os valores interpolados se mostram bastante inferiores aos valores medidos. Isso faz sentido quando consideramos que temos uma menor massa de dados para calcular as métricas, conforme o aumento no valor do salto.

Podemos observar na figura 11 que a amplitude spline está mais próxima da amplitude dos dados reais quando comparada à amplitude linear.

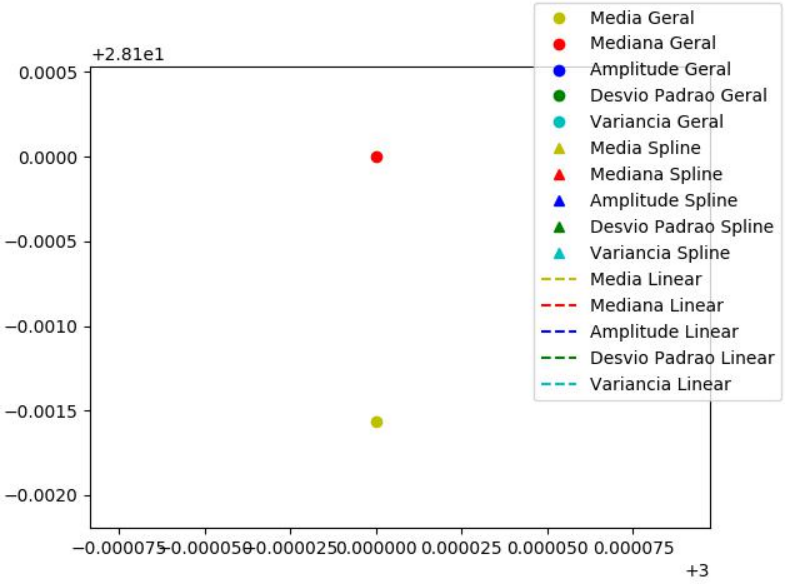

Figura 10 – Gráfico da média e Mediana geral no passo 3

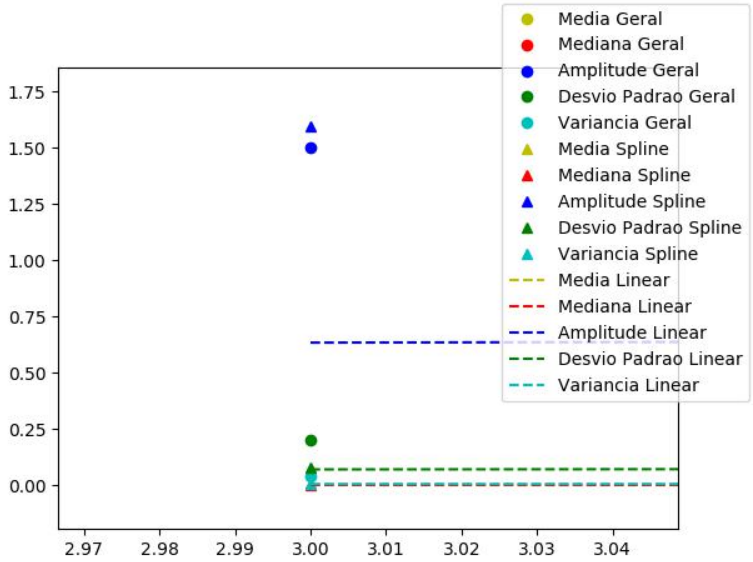

Figura 11 – Gráfico de amplitude, desvio Padrão e variância para os valores reais e interpolados para o salto 3

Conforme a figura 12, o desvio padrão, a amplitude, a mediana e a média dos valores interpolados se aproximam bastante dos valores reais para o passo 3.

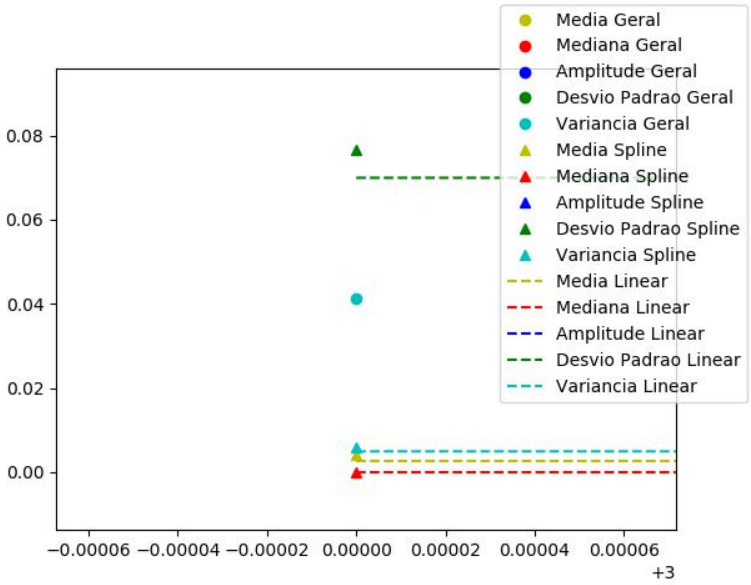

Figura 12 – Gráfico do desvio Padrão para dados interpolados, variância de dados medidos, média e mediana interpolados no salto 3

A figura 13 ilustra a proximidade dos valores das métricas para os dados interpolados, independente de método. Por exemplo, pode-se verificar que o valor da amplitude para os dados mensurados se diferencia em menos de 0,8 para os dados interpolados pela interpolação linear com o passo 5. Essa diferença se torna ainda menor visto pelo método de interpolação Spline Cúbico.

É possível observar na figura 14 a proximidade dos valores para o desvio padrão, tanto dos dados medidos como dos dados interpolados, onde a diferença é inferior a 0,1.

A diferença entre os valores de desvio padrão entre os dados reais e interpolados de forma linear é inferior a 0,05 para o caso do salto 7. Essa diferença se aproxima a 0,01 quando comparamos os dados reais com a interpolação spline no mesmo salto. Isso fica evidenciado na figura 15. Notamos também essa proximidade maior dos dados reais com os valores interpolados (com menor distância para os dados interpolados por spline) para a variância no salto 7.

Analisando os resultados das métricas de amplitude para o valor de salto 11, notamos que a variância para os dados medidos e os dados interpolados por spline têm diferença inferior a 0,05. Essa diferença é um pouco maior comparando a amplitude dos dados reais com os dados interpolados pelo método linear, como indica a figura 16

Conforme a figura 17, a diferença entre o desvio padrão dos dados medidos e dos dados interpolados por spline é inferior a 0,02. Quando os dados medidos foram comparados aos dados interpolados pelo método linear, a diferença é inferior a 0,01.

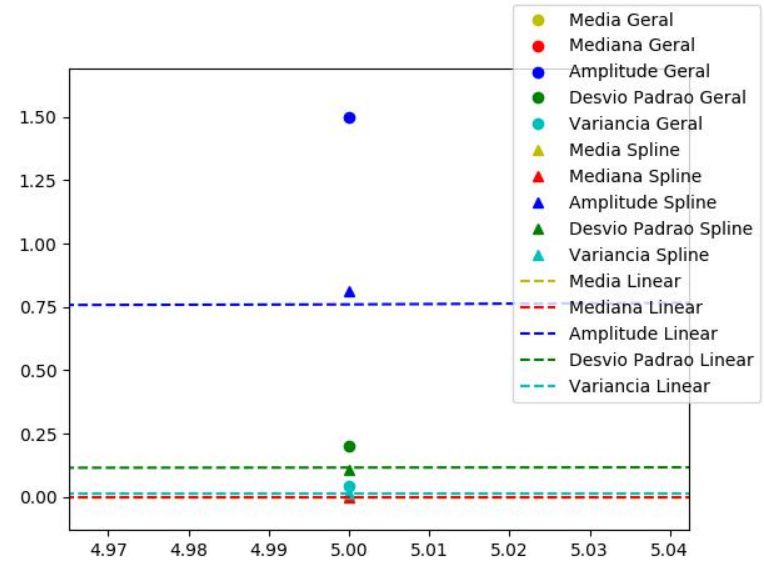

Figura 13 – Gráfico da comparação entre a amplitude para os dados medidos e interpolados, desvio adrão dos dados medidos, variância dos valores medidos e interpolados, assim como a média e a mediana intepolados no salto 5

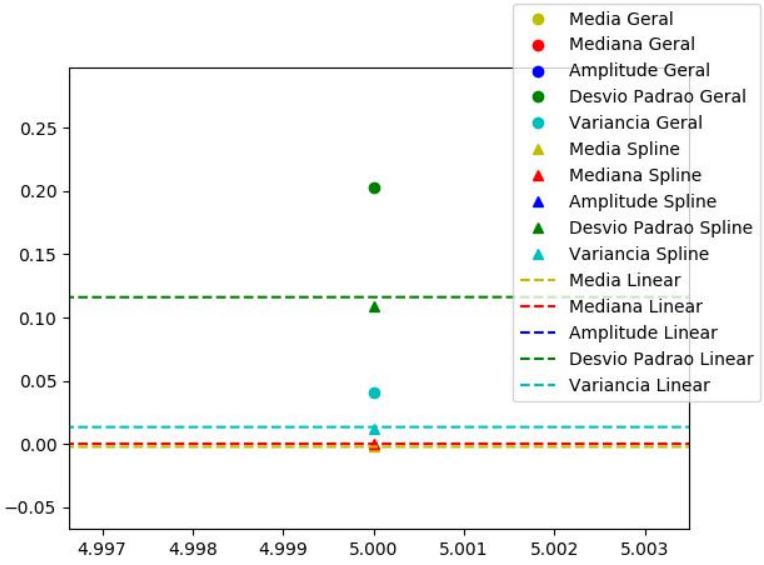

Figura 14 – Gráfico dos valores para o desvio padrão medido e interpolados, variância dos valores medidos e interpolados, assim como a média e a mediana intepolados no salto 5

Por último, quando analisamos os dados da figura 18, percebemos o quão próximo dos valores reais ficam nossos métodos de interpolação. Suas médias e medianas também possuem uma diferença muito pequena entre si.

Podemos concluir que os dados que geramos através dos métodos de interpolação se aproximam bastante aos dados reais. Consideramos para esta conclusão a análise

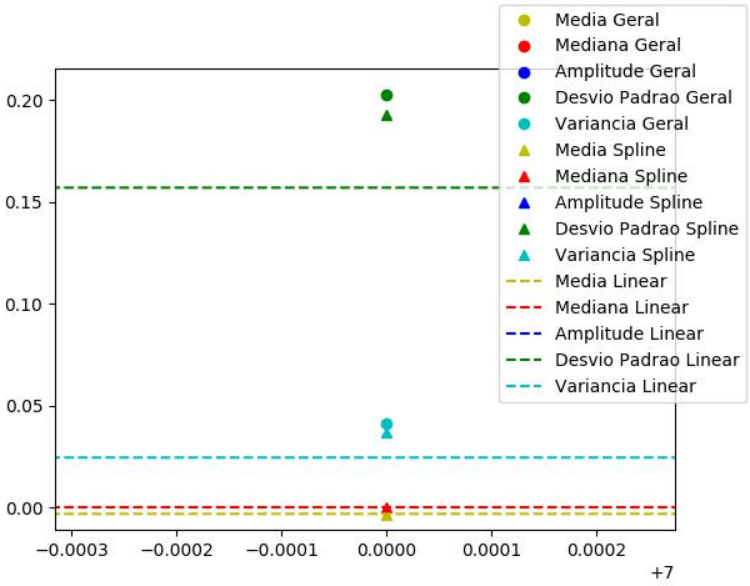

Figura 15 – Gráfico dos valores para o desvio padrão medido e interpolados, variância dos valores medidos e interpolados, assim como a média e a mediana intepolados no salto 7

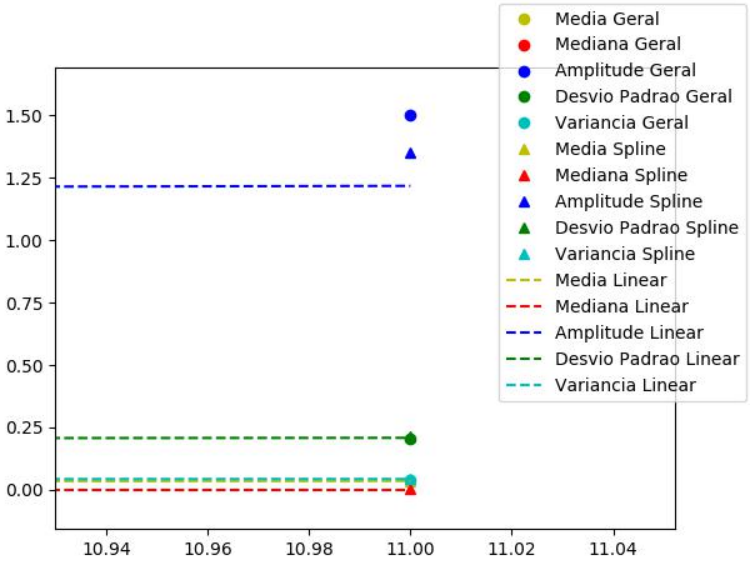

Figura 16 – Gráfico dos valores para a amplitude, o desvio padrão e a variância dos valores medidos e dos valores interpolados, assim como a média e a mediana intepolados no salto 11

nos dados nos pontos de saltos, a medida que o desvio padrão e a variância dos dados interpolados se aproximam dos resultados dos valores de temperaturas medidos.

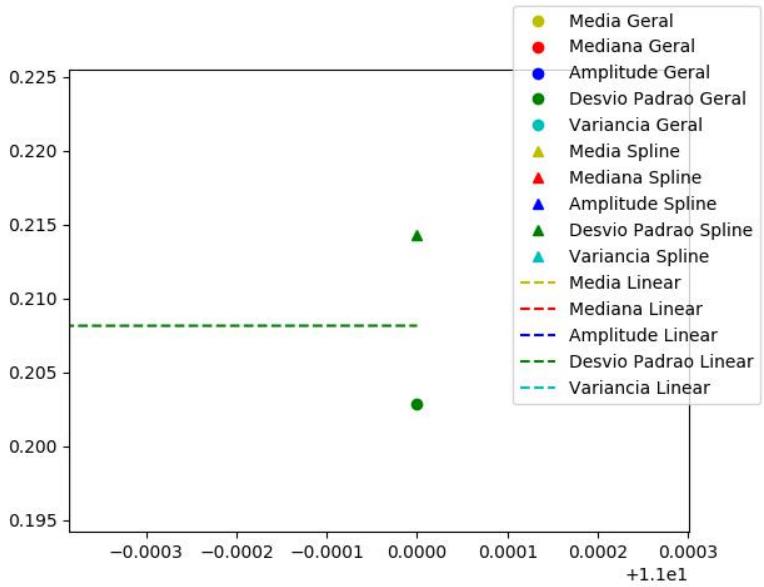

Figura 17 – Gráfico dos valores ampliados do desvio padrão dos valores medidos e dos valores interpolados no salto 11

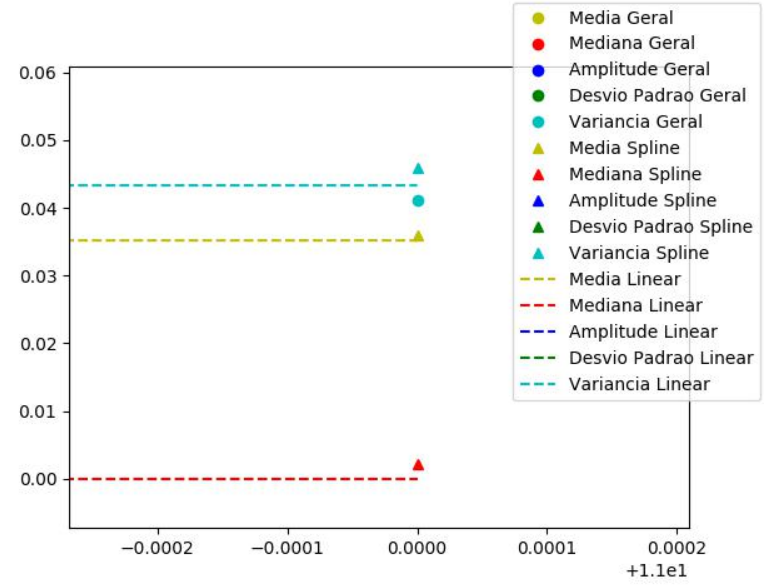

Figura 18 – Gráfico dos valores para a variância dos valores medidos e dos valores interpolados, assim como a média e a mediana intepolados no salto 11

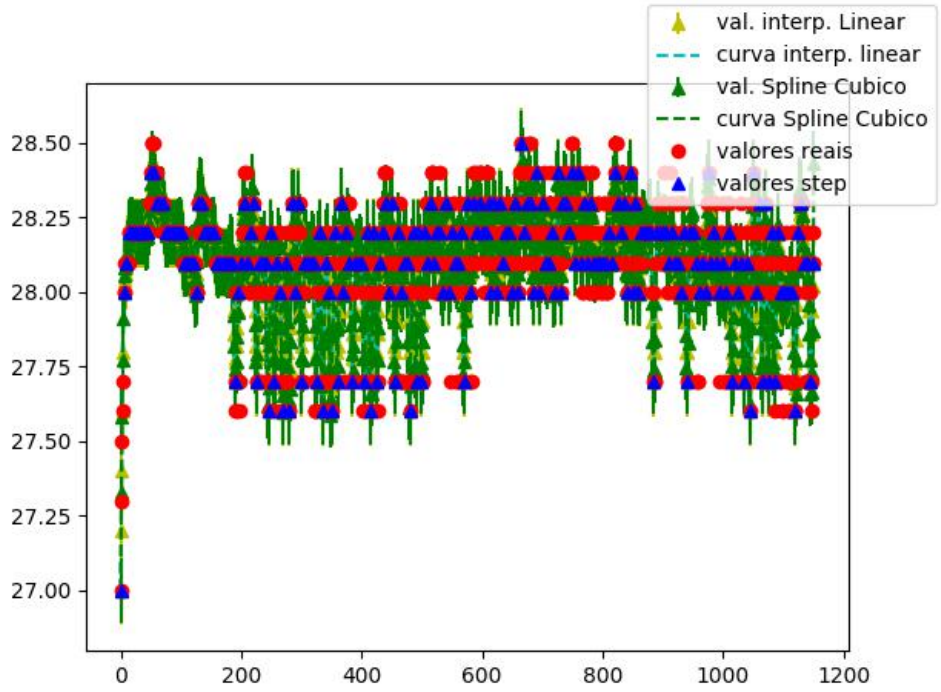

Figura 19 – Gráfico gerado pela interpolação do conjunto de dados de medições de 1 em 1 minuto em 29 de outubro de 2019, a partir de 02h40min GMT+1

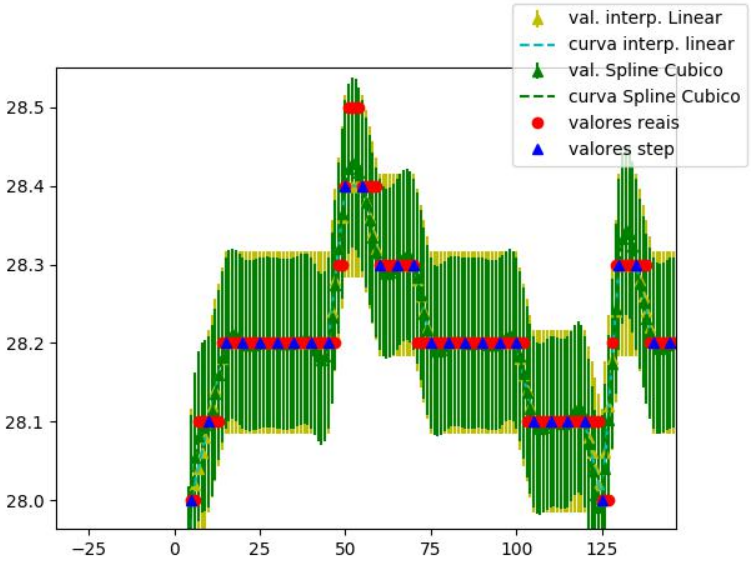

Figura 20 – Gráfico dos valores medidos, dos valores interpolados e dos intervalos de erro de cada interpolação gerados pelo script main.py

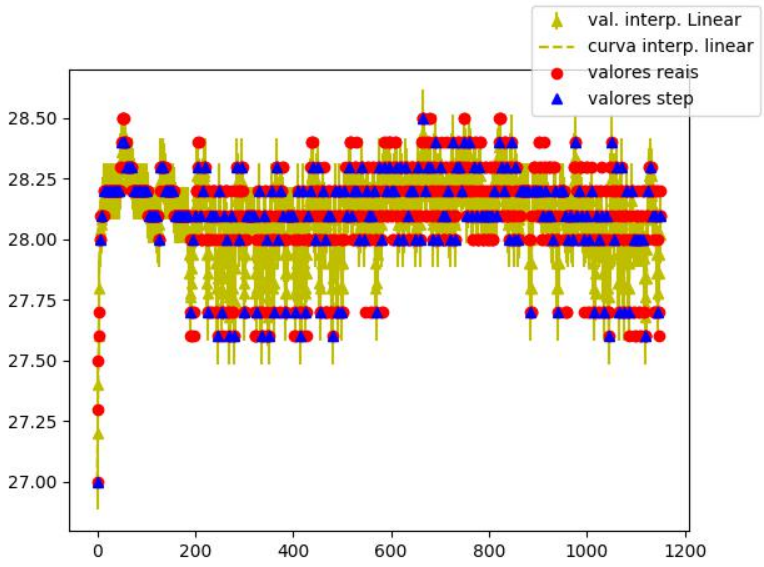

Figura 21 – Gráfico dos valores medidos e dos valores interpolados pelo método de Interpolação Linear, com os intervalos de erro gerados pelo script main.py

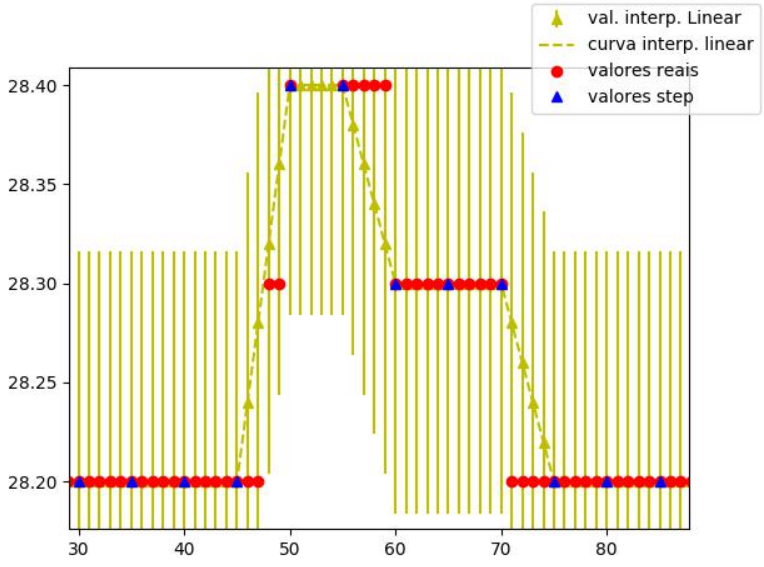

Figura 22 – Gráfico aumentado em região definida dos valores medidos e dos valores interpolados pelo método de Interpolação Linear, com os intervalos de erro gerados pelo script main.py

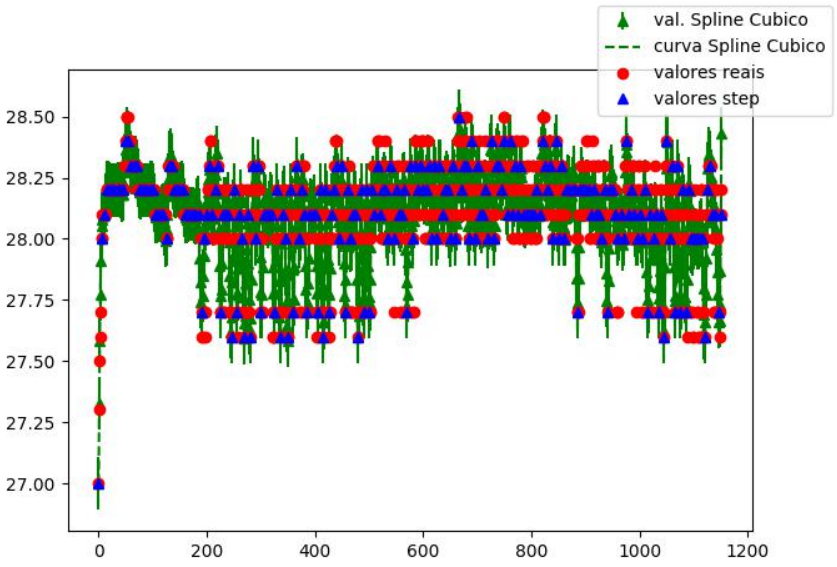

Figura 23 – Gráfico dos valores medidos e dos valores interpolados pelo método de Interpolação Spline Cúbico, com os intervalos de erro gerados pelo script main.py

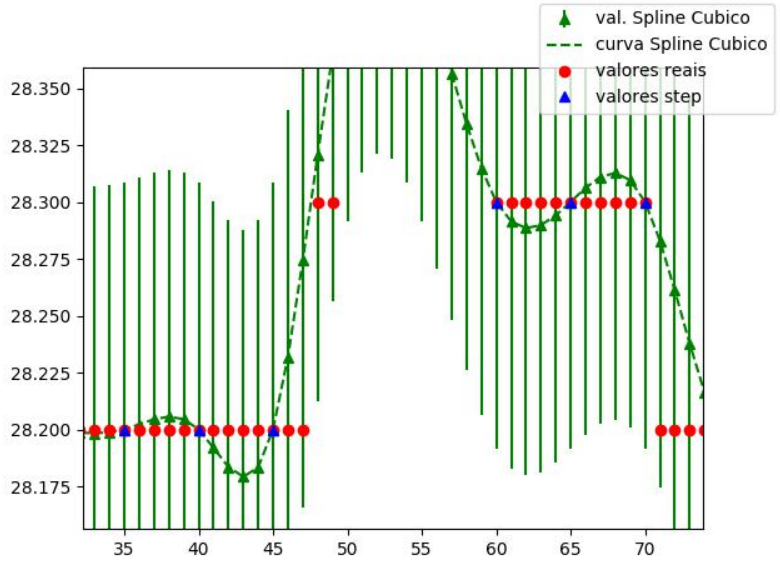

Figura 24 – Gráfico aumentado em região definida dos valores medidos e dos valores interpolados pelo método de Interpolação Spline Cúbico, com os intervalos de erro gerados pelo script main.py

5 CONCLUSÃO

Este trabalho apresentou uma ferramenta para auxiliar professores da disciplina de *Cálculo Numérico* no ensino de métodos de interpolação.

A proposta é permitir uma análise comparativa entre as leituras reais feitas por sensores de temperatura e os dados gerados pela interpolação de um subconjunto dessas leituras, usando os métodos de interpolação Linear e interpolação de Spline Cúbico.

Esses resultados são gerados de forma gráfica para ilustração dos métodos em sala de aula, tanto das métricas aplicadas a esses métodos como aos dados reais.

Concluímos neste trabalho que os valores das temperaturas interpoladas, tanto pelo método de interpolação Linear como pelo método de interpolação de Spline Cúbico se aproximam de forma considerável dos valores medidos. Notamos que houve pouca variação entre os resultados dos métodos de interpolação e os resultados medidos, porém o método de interpolação Linear se mostrou mais suave na geração da curva gráfica, como esperado.

Esta ferramenta pode auxiliar ao professor de cálculo numérico de forma positiva. Alunos que já tenham familiaridade com desenvolvimento de scripts em Python (ou qualquer outra linguagem) terão facilidade na leitura e compreensão do código, assim como da visualização de métodos de interpolação de forma gráfica.

Há a possibilidade de estender este trabalho, inserindo outras técnicas de interpolação, além de outros métodos de avaliação dos resultados. Os métodos de interpolação polinomial de Lagrange e de Bernstein podem ser boas aproximações, uma vez que ambos métodos são gerados a partir de polinômios. Esses métodos podem mostrar possíveis interpolações intermediárias entre os métodos estudados neste trabalho.

REFERÊNCIAS

ADJIH, C. et al. Fit iot-lab: A large scale open experimental iot testbed. In: . Milan, Italy: [s.n.], 2015. Disponível em: <https://hal.inria.fr/hal-01213938>.

AKYILDIZ, I. et al. Wireless sensor networks: a survey. **Computer Networks**, v. 38, n. 6, p. 393–422, 2002.

BURDEN, R. L.; FAIRES, J. D. **Numerical Analysis**. Fourth. Boston: PWS-Kent Publishing Company, 1989. (The Prindle, Weber and Schmidt Series in Mathematics).

CASTILLO-EFFER, M. et al. Wireless sensor networks for flash-flood alerting. **Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits and Systems, 2004.**, v. 1, p. 142–146, 2004.

DALLAS SEMICONDUCTORS, MAXIM INTEGRATED. **Digital Thermometer with SPI/3-Wire Interface**. [S.l.], 2007. Revisado em outubro de 2008.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

LUNDH, F. An introduction to tkinter. **URL: www. pythonware. com/library/tkinter/introduction/index. htm**, 1999.

OLIPHANT, T. E. **A guide to NumPy**. [S.l.]: Trelgol Publishing USA, 2006. v. 1.

TEXAS INSTRUMENTS. **MSP430F15x, MSP430F16x, MSP430F161x MIXED SIGNAL MICROCONTROLLER**. [S.l.], 2002. Revisado em outubro de 2011.

VIRTANEN, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020.