# Decomposição LU para Matrizes Tridiagonais Cíclicas

Exercício Programa 01

Relatório Final


Natanael Magalhães Cardoso, 8914122

Valber Marcelino Filho, 11353165


Professora: Cláudia Peixoto

Turma: 03


**Universidade de São Paulo**

Escola Politécnica


MAP3121 - Métodos Numéricos e Aplicações

**UNIVERSIDADE DE SÃO PAULO**

**ESCOLA POLITÉCNICA**

MAP3121 - Métodos Numéricos e Aplicações


# Decomposição LU para Matrizes Tridiagonais Cíclicas

Natanael Magalhães Cardoso*, Valber Marcelino Filho†

## 1. INTRODUÇÃO

Uma matriz tridiagonal é uma matriz em banda que possui termos diferentes de zeros apenas na diagonal principal, na subdiagonal (abaixo da diagonal principal) e na supradiagonal (acima da diagonal principal). Como existem aplicações onde se é necessário resolver sistemas lineares com esse tipo de matrizes, uma técnica numérica de resolução desses sitemas é a fatoração LU (Banachiewicz, 1938a,b), que consiste na decomposição da matriz dos coeficientes do sistema em uma matriz triangular superior em outra inferior, que podem ser interpretadas como as matrizes provenientes da eliminação gaussiana. A triangular superior com a informação do resultado da eliminação e a triangular inferior com a informação dos multiplicadores. Além de sistemas lineares, a decomposição LU também é um passo fundamental para inversão de matrizes e para o cálculo de determinantes.

Sistemas tridiagonais são encontrados em problemas em diversas áreas, como em engenharia elétrica, na resolução de certos tipos de circuitos RLC (Yarlagadda, 1968), em economia, na criação de modelos estatísticos que utilizam a autoregressão de primeira ordem para induzir dependência na estrutura de covariância (Tan, 2019), e em oceanografia, na modelagem da propagação do som no oceano (Ivansson, 2017).

Nesse trabalho, implementamos o algoritmo para a decomposição LU de uma matriz tridiagonal e para a resolução de um sistema linear tridiagonal cíclico ou não cíclico usando a decomposição LU da matriz dos coeficientes do sistema.

Na Seção 2, descrevemos o funcionamento do algorítmo implementado, incluindo instruções de execução, e na Seção 3, comparamos o resultado do nosso algorítmo com outra biblioteca de álgebra linear e, também, analisamos ordem do algorítmo implementado. Por fim, concluímos que

*nUSP: 8914122, Turma: 03
†nUSP: 11353165, Turma: 03

é possível implementar um algorítmo de solução de sistemas lineares de ordem $\mathcal{O}(n)$ considerando a tridiagonalidade da matriz dos coeficientes do sistema, que é consideravelmente menor que a ordem $\mathcal{O}(n^3)$ da eliminação gaussiana.

# 2. MÉTODOS

Este programa foi implementado usando a linguagem de programação Python (Van Rossum and Drake, 2009) e o pacote de computação científica Numpy (Harris et al., 2020), ambos de código aberto.

A descrição da implementação é dividida em duas partes: *backend* (Seção 2.1) e *frontend* (Seção 2.2). A primeira descreve o functionamento do algorítmo de solução de sistemas tridiagonais cíclicos e a segunda descreve o funcionamento da interface do programa com o usuário.

## 2.1. BACKEND

O objetivo deste programa é encontrar a solução do sistema $Ax = d$, em que $A$ tem a forma descrita pela eq. (1). Isto é, encontrar os valores da matriz coluna $x$.

$$A = \begin{bmatrix} b_1 & c_1 & & & & & & a_1 \\ a_2 & b_2 & c_2 & & & & & \\ & a_3 & b_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_i & b_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & & & & & & a_n & b_n \end{bmatrix}_{n\times n} \tag{1}$$

Uma matriz $A$ desta forma é chamada matriz tridiagonal em banda. Sua característica é que apenas os elementos das diagonais principal, subdiagonal e supradiagonal são diferentes de zero. Ela também pode ser dividida em cíclica, caso $a_1 \neq 0$ e $c_n \neq 0$, ou não cíclica, caso $a_1 = 0$ e $c_n = 0$.

Somente $3n$ dos elementos da matriz são, de fato, relevantes nos cálculos da solução do sistema, sendo $3n - 2$ elementos diferentes de zero para o caso não cíclico. Então, para otimizar os recursos computacionais gastos, apenas os elementos da banda tridiagonal e das bordas são armazenados no programa, ao invés da matriz $A$ inteira. Dessa forma, a matriz $A$ é representada por três vetores: $\mathbf{a} = [a_i]_n$, $\mathbf{b} = [b_i]_n$ e $\mathbf{c} = [c_i]_n$, ambos de dimensão $n$. Para o caso não cíclico, os elementos $a_1$ e $c_n$ valem zero.

Para encontrar a solução de um sistema tridiagonal, o programa calcula a decomposição LU da matriz dos coeficientes $A$ e, então, são calculadas as soluções para os sistemas $Ly = d$ e $Ux = y$, onde $x$ é a solução do sistema inicial $Ax = d$.

E, quando a matriz dos coeficientes $A$ é tridiagonal cíclica, o problema é separado em partes menores. É criada uma submatriz de $A$ de dimensão $m \times m$, com $m = n - 1$, denominada $A'$, que é tridiagonal não cíclica. Com isso, são resolvidos dois sistemas tridiagonais não cíclicos: $A'y' = d'$ e $A'z' = v'$, onde $d' = (d_1, d_2, ..., d_m)^T$ e $v' = (c_n, 0, ..., 0, a_n)^T$.

Como dois sistemas tridiagonais com a mesma fatoração LU da matriz $A$ precisam ser resolvidos na implementação da solução do sistema tridiagonal cíclio, reutilizamos a fatoração LU para otimizar a performance do programa. A Tabela 1 mostra uma breve descrição para cada uma das três funções implementadas no *backend*.

**Tabela 1:** Descrição dos argumentos da Interface de Linha de Comando

| Função | Descrição |
|---:|---|
| `decomp_lu` | Recebe a matriz $A$ na forma dos três vetores **a**, **b** e **c**, calcula a decomposição LU da matriz e retorna dois vetores, **l** e **u**, que representam a matriz L e U da decomposição, respectivamente. |
| `solve_tridiagonal` | Recebe a decomposição LU da matriz dos coeficientes $A$ e o vetor **d** com os termos idependentes, resolve os sistemas $Ly = d$ e $Ux = y$ e retorna a solução do segundo sistema, um vetor **x**, que é, também, a solução do sistema $Ax = d$ ou $(LU)x = d$. |
| `solve_cyclic` | Recebe a matriz $A$ na forma dos três vetores **a**, **b** e **c** e o vetor **d** com os termos idependentes, executa o algorítmo de solução do sistema para o caso cíclico através de duas chamadas à função `solve_tridiagonal` e retorna um vetor **x**, que representa a solução do sistema. |

## 2.2. FRONTEND

O *frontend* consiste na implementação de uma Interface de Linha de Comando (CLI), por onde o usuário final interage com o programa. Através dela, o usuário obtêm o resultado de acordo com os argumentos definidos no momento da chamada do programa. Mudando estes argumentos, o usuário consegue escolher o tipo de sistema (cíclico ou não-cíclico), ajustar o tamanho da matriz dos coeficientes, a quantidade de casas decimais exibidas ou, até mesmo, o formato de saída. A Tabela 2 descreve cada um desses argumentos.

### 2.2.1 Exemplos de chamadas do programa

1. `python3 ep1.py -a`
2. `python3 ep1.py -a -n 5`
3. `python3 ep1.py -a -n 250 -d 10`
4. `python3 ep1.py -c`
5. `python3 ep1.py -c -n 5`
6. `python3 ep1.py -c -n 250 -d 10`

As chamadas (1), (2) e (3), que possuem o argumento `-a`, configuram o programa para resolver um problema com matriz tridiagonal não-cíclica. Enquanto que as chamadas (4), (5) e (6), que possuem o argumento `-c`, configuram o programa para resolver um sistema com matriz tridiagonal cíclica. Além disso, as chamadas (1) e (4) executam o programa com os valores padrões de `-n = 20` e `-d = 4`, já esses argumentos não foram especificados. As chamadas (2) e (5) executam o programa com uma matriz dos coeficientes de dimensão $5 \times 5$ (`-n 5`). E, as chamadas (3) e (6) executam o programa com uma matriz dos coeficientes de dimensão $250 \times 250$ (`-n 250`) e os resultados são exibidos com 10 casas decimais (`-d 10`). As Tabelas 3 e 4 da Seção 3 mostram a saída do programa para as chamadas (2) e (6), respectivamente.

**Tabela 2:** Descrição dos argumentos da Interface de Linha de Comando

| Argumento | Descrição |
|---|---|
| `-h` | *Opcional.* Exibe uma mensagem de ajuda com breve descrição dos argumentos do programa. |
| `-a` | *Obrigatório*.* Configura o programa para resolver um sistema tridiagonal não-cíclico (ou acíclico). Isto é, a matriz dos coeficientes carregada possui $a_1 = 0$ e $c_n = 0$ e o algorítmo usado para resolver este sistema é o de matrizes tridiagonais não cíclicas. |
| `-c` | *Obrigatório*.* Configura o programa para resolver um sistema tridiagonal cíclico. A matriz dos coeficientes gerada possui $a_1 \neq 0$ e $c_n \neq 0$ e o algorítmo usado é o que considera matrizes cíclicas. **Importante:** (*) Os argumentos `-a` e `-c` são obrigatórios, mas não podem ser usado em conjunto. Uma chamada do programa deve ter o parâmetro `-a` ou o parâmetro `-c`, mas não ambos e nem a ausência de ambos. |
| `-n <N>` | *Opcional.* Este argumento possui um parâmetro $N$ que deve ser um número inteiro que representa a dimensão $N \times N$ da matriz dos coeficientes. Valor padrão: `20` |
| `-d <D>` | *Opcional.* Este argumento possui um parâmetro $D$ que configura o número de dígitos decimais a serem mostrados na resposta do programa. Valor padrão: `4` |
| `--csv` | *Opcional.* Configura a saída do programa para ser impressa no formato CSV. Este parâmetro foi criado por propósito de desenvolvimento, mas pode ser usado pelo usuário final. Valor padrão: `False` |

# 3. RESULTADOS

## 3.1. SAÍDA DO PROGRAMA

A saída do programa é uma tabela com 6 colunas e o número de linhas definido pelo argumento `-n` na chamada do programa, ou 20 caso não haja especificação do argumento `-n`. As colunas da saída do programa são as mesmas das Tabelas 3 e 4, que são exemplos de saída para diferentes valores de `-n` e `-d`. No primeiro, `-n=5` e `-d` não foi especificado e, no segundo, `-n=250` e `-d=10`.

**Tabela 3:** Exemplo de saída do programa para resolução do problema cíclico executado com os argumentos `-n=5` e `-d` não foi especificado

| $i$ | $a_i$ | $b_i$ | $c_i$ | $d_i$ | $x_i$ |
|---|---|---|---|---|---|
| 1 | **0.0000** | 2.0000 | 0.7500 | 0.9686 | 0.3827 |
| 2 | 0.3750 | 2.0000 | 0.6250 | 0.5358 | 0.2709 |
| 3 | 0.4167 | 2.0000 | 0.5833 | -0.6374 | -0.2392 |
| 4 | 0.4375 | 2.0000 | 0.5625 | -0.6374 | -0.4660 |
| 5 | 0.9000 | 2.0000 | **0.0000** | 1.0000 | 0.7097 |

**Tabela 4:** Exemplo de saída do programa para resolucão do problema cíclico executado com os argumentos `-n=250` e `-d=10`

| $i$ | $a_i$ | $b_i$ | $c_i$ | $d_i$ | $x_i$ |
|---|---|---|---|---|---|
| 1 | 0.2500000000 | 2.0000000000 | 0.7500000000 | 0.9917900138 | 0.3086835218 |
| 2 | 0.3750000000 | 2.0000000000 | 0.6250000000 | 0.8713187041 | 0.3221924507 |
| 3 | 0.4166666667 | 2.0000000000 | 0.5833333333 | 0.4047833431 | 0.1778839713 |
| ... | ... | ... | ... | ... | ... |
| 248 | 0.4989919355 | 2.0000000000 | 0.5010080645 | 0.9949912944 | 0.3317989975 |
| 249 | 0.4989959839 | 2.0000000000 | 0.5010040161 | 0.9987420027 | 0.3330519735 |
| 250 | 0.9980000000 | 2.0000000000 | 0.0020000000 | 1.0000000000 | 0.3334737509 |

## 3.2. TESTES

A validacão das solucões foi feita comparando o valor da solucão do sistema dado pelo algorítmo implementado com o valor dado pelo módulo de álgebra linear do Numpy, mostrada na Tabela 5. Nela é mostrada as entradas $a_i$, $b_i$, $c_i$ e $d_i$ junto com as solucões do sistema não cíclico e do sistema cíclico de $n = 20$ incógnitas dadas por esse algorítmo e pelo Numpy (coluna np).

**Tabela 5:** Comparacão dos valores obtidos com o programa implementado com o numpy para matrizes diagonais cíclicas e não cíclicas. Os valores em negrito ($a_1$ e $c_{20}$) são diferentes para solucão cíclica e não cíclica, estes mostrados são para solucão cíclica

| | Entradas | | | | Sol. (não cíclico) | | Sol. (cíclico) | |
|---|---|---|---|---|---|---|---|---|
| $i$ | $a_i$ | $b_i$ | $c_i$ | $d_i$ | $x_i$ | $x_i$ (np) | $x_i$ | $x_i$ (np) |
| 1 | **0.250000** | 2.000000 | 0.750000 | 0.999877 | 0.378440 | 0.378440 | 0.330315 | 0.330315 |
| 2 | 0.375000 | 2.000000 | 0.625000 | 0.998027 | 0.323996 | 0.323996 | 0.333698 | 0.333698 |
| 3 | 0.416667 | 2.000000 | 0.583333 | 0.990024 | 0.332990 | 0.332990 | 0.330821 | 0.330821 |
| 4 | 0.437500 | 2.000000 | 0.562500 | 0.968583 | 0.324077 | 0.324077 | 0.324586 | 0.324586 |
| 5 | 0.450000 | 2.000000 | 0.550000 | 0.923880 | 0.310661 | 0.310661 | 0.310538 | 0.310538 |
| 6 | 0.458333 | 2.000000 | 0.541667 | 0.844328 | 0.284951 | 0.284951 | 0.284981 | 0.284981 |
| 7 | 0.464286 | 2.000000 | 0.535714 | 0.718126 | 0.243765 | 0.243765 | 0.243757 | 0.243757 |
| 8 | 0.468750 | 2.000000 | 0.531250 | 0.535827 | 0.183489 | 0.183489 | 0.183491 | 0.183491 |
| 9 | 0.472222 | 2.000000 | 0.527778 | 0.294040 | 0.102745 | 0.102745 | 0.102744 | 0.102744 |
| 10 | 0.475000 | 2.000000 | 0.525000 | 0.000000 | 0.003606 | 0.003606 | 0.003606 | 0.003606 |
| 11 | 0.477273 | 2.000000 | 0.522727 | -0.323917 | -0.106697 | -0.106697 | -0.106697 | -0.106697 |
| 12 | 0.479167 | 2.000000 | 0.520833 | -0.637424 | -0.214728 | -0.214728 | -0.214728 | -0.214728 |
| 13 | 0.480769 | 2.000000 | 0.519231 | -0.883766 | -0.301138 | -0.301138 | -0.301137 | -0.301137 |
| 14 | 0.482143 | 2.000000 | 0.517857 | -0.998027 | -0.343306 | -0.343306 | -0.343308 | -0.343308 |
| 15 | 0.483333 | 2.000000 | 0.516667 | -0.923880 | -0.320983 | -0.320983 | -0.320975 | -0.320975 |
| 16 | 0.484375 | 2.000000 | 0.515625 | -0.637424 | -0.224483 | -0.224483 | -0.224511 | -0.224511 |
| 17 | 0.485294 | 2.000000 | 0.514706 | -0.171929 | -0.063964 | -0.063964 | -0.063864 | -0.063864 |
| 18 | 0.486111 | 2.000000 | 0.513889 | 0.368125 | 0.126168 | 0.126168 | 0.125807 | 0.125807 |
| 19 | 0.486842 | 2.000000 | 0.513158 | 0.818150 | 0.285825 | 0.285825 | 0.287136 | 0.287136 |
| 20 | 0.975000 | 2.000000 | **0.025000** | 1.000000 | 0.360660 | 0.360660 | 0.355892 | 0.355892 |

## 3.3. CONTAGEM DAS OPERACÕES

**Tabela 6:** Contagem de operacões para cada uma das funcões listadas na Tabela 1

| Funcão | # mul/div | # adi/sub | total |
|---|---|---|---|
| `decomp_lu` | $2n-2$ | $n-1$ | $3n-3$ |
| `solve_tridiagonal` | $3n-2$ | $2n-2$ | $5n-4$ |
| `solve_cyclic` | $9n-11$ | $6n-8$ | $15n-19$ |

A Tabela 6 mostra a contagem de operacões feitas pelo algorítmo. Além disso, foi feito um experimento para cálcular o tempo de execucão do algorítmo de solucão de sistemas cíclicos em fucão do número $n$ de incógnitas. Este experimento foi feito em uma máquina com as seguintes especificacões de software: S.O. Ubuntu 20.04 Kernel GNU Linux 5.13.0-39, Python 3.8.10 (CPython compilado com GCC 9.4.0 no Linux), Numpy 1.21.4 e as especificacões de hardware mostradas na Figura 2. O resultado desse experimento é mostrado na Figura 1, onde cada ponto representa o tempo médio de 200 execucões para cada valor de $n$, a área sombreada respresenta a banda de $\pm 1\sigma$ em torno da média e a reta vermelha é uma regressão linear do tempo médio, que tem a expressão

dada na eq. 2 e coeficiente de determinacão $R^2 \approx 0.99689$.

$$T(n) = 0.0040619n - 0.2301847 \quad [\text{ms}] \tag{2}$$

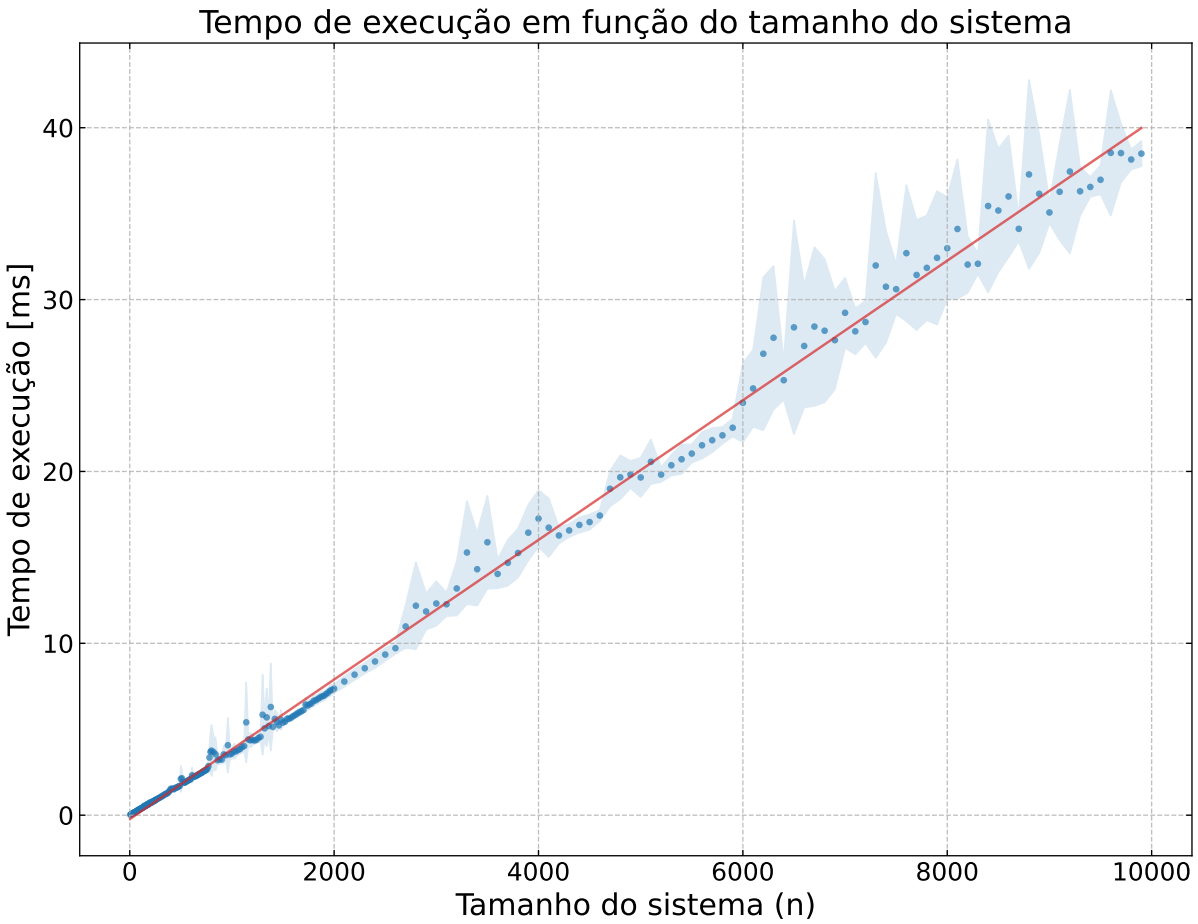


**Figura 1:** Tempo médio de execucão em funcão do tamanho de sistemas cíclicos. Cada ponto representa a média de 200 execucões, a área sombreada representa $\pm 1\sigma$ da distribuicão de execucões para cada $n$ e a reta vermelha é a regressão linear do tempo médio.

## 4. Discussão e Conclusão

Pela Tabela 6, notamos que a contagem de operacões do algorítmo de solucão de sistemas cíclicos segue um polinômio de grau 1. A Figura 1, que mostra o resultado experimental, confirma os cálculos teóricos. A partir da eq. (2) notamos que o tempo de execucão possui escalamento linear em relacão ao aumento do tamanho do sistema, $T(n) \in \mathcal{O}(n)$.

Com isso, concluímos que é possível implementar um algorítmo de complexidade $\mathcal{O}(n)$ em relacão ao tempo para resolver um sistema tridiagonal cíclico usando decomposicão LU. Este resultado representa uma vantagem significativa em relacão aos métodos que não consideram a tridiagonalidade da matriz. O método da Eliminacão Gaussiana, por exemplo, possui complexidade em relacão ao tempo $\mathcal{O}(n^3)$ se não considerada a tridiagonalidade da matriz e $\mathcal{O}(n)$ se considerada (Ryaben'kii and Tsynkov, 2006).

# APÊNDICE

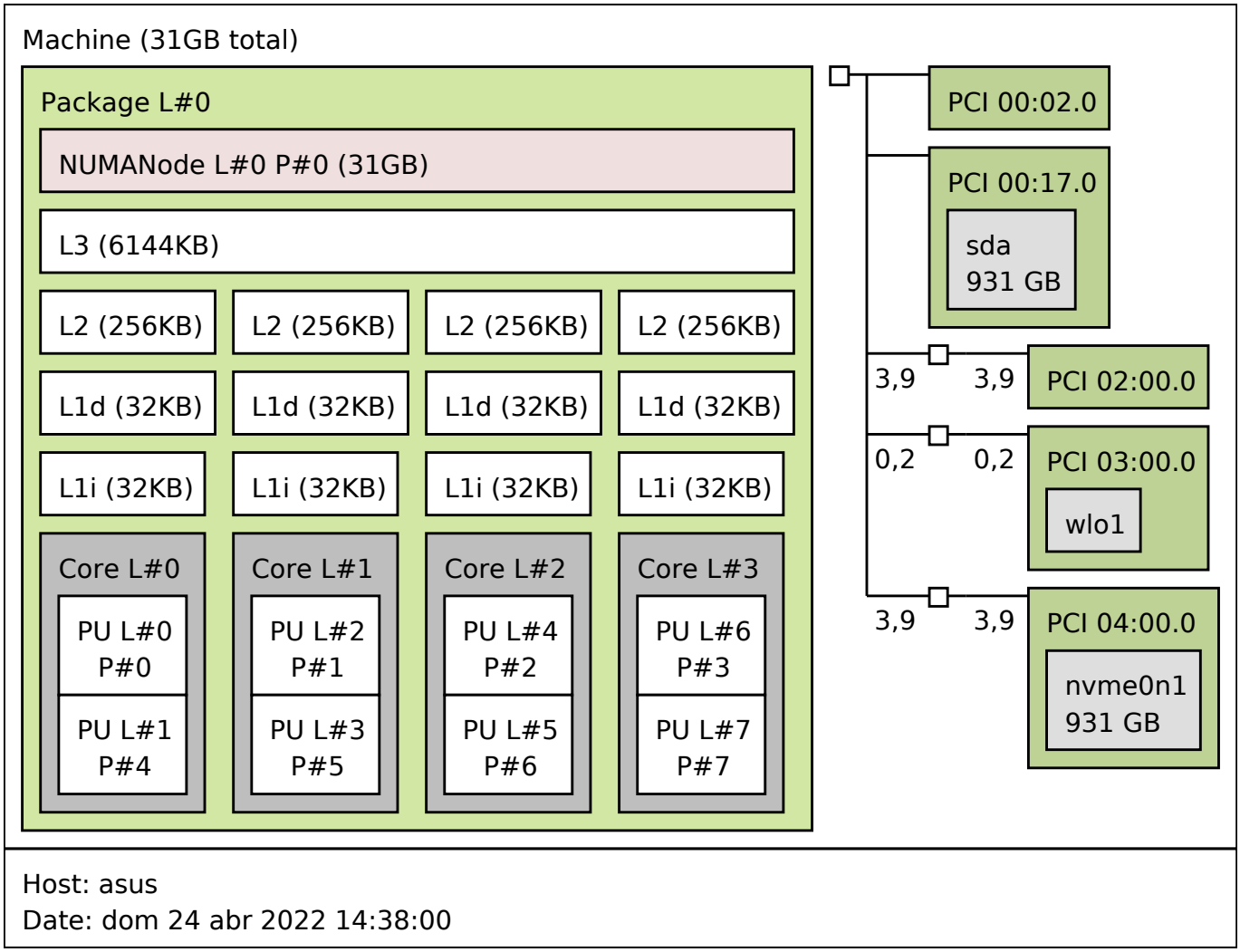


**Figura 2:** Topologia da máquina usada para testes experimentais criada pelo pacote open-source `hwloc` (https://www.open-mpi.org/projects/hwloc/).

# REFERÊNCIAS

Tadeusz Banachiewicz. Principes d'une nouvelle techinique de la méthhode des moindres carrés. *Bull. Internat. Acad. Polan. Sci. A.*, pages 134–135, 1938a.

Tadeusz Banachiewicz. Méthode de résolution numérique des équations linéaires, de calcul des déterminants et des inverses, et de réduction des formes quadratiques. *Bull. Internat. Acad. Polan. Sci. A.*, pages 393–404, 1938b.

Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

S. Ivansson. Chapter 3 - sound propagation modeling. In Thomas H. Neighbors and David Bradley, editors, *Applied Underwater Acoustics*, pages 185–272. Elsevier, 2017. ISBN 978-0-12-811240-3. doi: 10.1016/B978-0-12-811240-3.00003-5. URL https://www.sciencedirect.com/science/article/pii/B9780128112403000035.

Victor S Ryaben'kii and Semyon V Tsynkov. *A theoretical introduction to numerical analysis*. Chapman & Hall/CRC, Philadelphia, PA, November 2006.

Linda S L Tan. Explicit inverse of tridiagonal matrix with applications in autoregressive modelling. *IMA Journal of Applied Mathematics*, 84(4):679–695, 07 2019. ISSN 0272-4960. doi: 10.1093/imamat/hxz010. URL https://doi.org/10.1093/imamat/hxz010.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

R. Yarlagadda. An application of tridiagonal matrices to network synthesis. *SIAM Journal on Applied Mathematics*, 16(6):1146–1162, 1968. ISSN 00361399. URL http://www.jstor.org/stable/2099533.


**Universidade de São Paulo**

Escola Politécnica

MAP3121 - Métodos Numéricos e Aplicacões