



UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO INTERDISCIPLINAR EM CIÊNCIA E TECNOLOGIA

MICHELE EDNEIDE DA COSTA LEITÃO

**ÁLGEBRA LINEAR E LINGUAGEM DE PROGRAMAÇÃO PYTHON
NUMA PERSPECTIVA INTERDISCIPLINAR**

MOSSORÓ
2021

MICHELE EDNEIDE DA COSTA LEITÃO

**ÁLGEBRA LINEAR E LINGUAGEM DE PROGRAMAÇÃO PYTHON NUMA PERSPECTIVA
INTERDISCIPLINAR**

Trabalho de conclusão de curso do Curso de Graduação em
Interdisciplinar em Ciência e Tecnologia do Centro de Ciências
Exatas e Naturais da Universidade Federal do Semi-Árido.

Orientador: Prof. Dr. Fabricio de Figueiredo Oliveira

MOSSORÓ
2021

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei nº 9.279/1996 e Direitos Autorais: Lei nº 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

L533Á Leitão, Michele Edneide da Costa.
Álgebra Linear e Linguagem de Programação
Python numa Perspectiva Interdisciplinar /
Michele Edneide da Costa Leitão. - 2021.
99 f. : il.

Orientador: Fabricio de Figueiredo Oliveira.
Monografia (graduação) - Universidade Federal
Rural do Semi-árido, Curso de , 2021.

1. Interdisciplinaridade. 2. Python. 3.
Matriz. 4. Operações. 5. Algoritmo. I. Oliveira,
Fabricio de Figueiredo, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade
com AACR2 e os dados fornecidos pelo(a) autor(a).

Biblioteca Campus Mossoró / Setor de Informação e Referência
Bibliotecária: Keina Cristina Santos Sousa e Silva
CRB: 15/120

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC's) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

MICHELE EDNEIDE DA COSTA LEITÃO

**ÁLGEBRA LINEAR E LINGUAGEM DE PROGRAMAÇÃO
PYTHON NUMA PERSPECTIVA INTERDISCIPLINAR**

Monografia apresentada à Universidade Federal Rural do Semi-Árido – UFERSA como requisito para obtenção do título de Bacharel em Interdisciplinar em Ciência e Tecnologia.

Defendida em: 18/11/2021.

BANCA EXAMINADORA

FABRICIO DE FIGUEREDO Assinado de forma digital por FABRICIO
OLIVEIRA:64905896304 DE FIGUEREDO OLIVEIRA:64905896304
Dados: 2021.11.18 08:26:11 -03'00'

Prof. Dr. Fabricio de Figueredo Oliveira (UFERSA)
Presidente

IVAN Assinado de forma digital por IVAN
MEZZOMO:72569891053 MEZZOMO:72569891053
Dados: 2021.11.18 10:46:34 -03'00'

Prof. Dr. Ivan Mezzomo (UFERSA)
Membro Examinador

Rafael Jorge Pontes Diógenes
Prof. Dr. Rafael Jorge Pontes Diógenes (UNILAB)
Membro Examinador

RESUMO

Durante o desenvolvimento da aprendizagem é possível abordar métodos diversos para alcançar os objetivos esperados nos estudos. O deferido trabalho busca a ideia de unir as disciplinas de Álgebra Linear e Programação para aprimorar a maneira de aprender, dessa forma foi adotado o conceito da interdisciplinaridade, que busca a unificação de disciplinas com a intenção de ajudar no desenvolvimento uma da outra. O estudo a seguir possui como objetivo apresentar ao leitor, de maneira didática, como utilizar a linguagem Python para solucionar problemas de Álgebra Linear que envolvam os conceitos, definições de matrizes e suas operações, para que em seguida possam ser realizadas aplicações na Engenharia. Inicialmente é dada a definição de matrizes, para que em seguida seja definida as operações (soma, subtração e multiplicação), utilizando Python serão estruturados algoritmos com base nas definições, conceitos e propriedades das operações. O mesmo é realizado para matriz transposta, anti-simétrica, simétrica e inversa de modo que ao desenvolver os algoritmos não seja necessário nenhuma restrição no código, uma vez que o leitor terá os devidos conhecimentos para implementá-lo. O determinante é introduzido na análise de sistemas de equações lineares, verificando se os mesmos possuem solução, infinitas soluções ou não apresente nenhuma, e caso o sistema possua solução única é aplicado o método da eliminação de Gauss para solucionar as equações. Como esse projeto está voltado para iniciantes em Python, a metodologia busca desde a instalação do Python 3 até a execução de um algoritmo na IDE PyCharm, que foi escolhida por possuir fácil acesso, ser um software gratuito e apresentar uma interface limpa.

Palavras-chaves: Interdisciplinaridade. Python. Matriz. Operações. Algoritmo.

ABSTRACT

During the development of learning, it is possible to approach different methods to achieve the expected objectives in the studies. This work seeks the idea of uniting the disciplines of Linear Algebra and Programming to improve the way of learning. In this way, the concept of interdisciplinarity was adopted, which seeks to unify disciplines with the intention of helping each other to develop. The following study aims to show the reader, in a didactic way, how to use the Python language to solve Linear Algebra problems involving the concepts, definitions of matrices and their operations, so that applications can then be carried out in Engineering. Initially, the definition of matrices is given, so that the operations (addition, subtraction and multiplication) are then defined. Using Python, algorithms will be structured based on the definitions, concepts and properties of the operations. The same is done for transposed, anti-symmetric, symmetric and inverse matrix so that when developing the algorithms no restriction in the code is necessary, since the reader will have the necessary knowledge to implement it. The determinant is introduced in the analysis of systems of linear equations, checking if they have a solution, infinite solutions or none, and if the system has a single solution, the Gauss elimination method is applied to solve the equations. As this project is aimed at Python beginners, the methodology seeks from the installation of Python 3 to the execution of an algorithm in the PyCharm IDE, which was chosen because of its easy access, being free software and presenting a clean interface.

Keywords: Interdisciplinarity. Python. Matrix. Operations. Algorithm.

LISTA DE FIGURAS

| | | |
|-------------|---|----|
| Figura 1 – | Site oficial do Python. | 11 |
| Figura 2 – | Download para Windows. | 12 |
| Figura 3 – | Versões Python. | 12 |
| Figura 4 – | Dowload Python 3. | 13 |
| Figura 5 – | Instalação Python 3. | 13 |
| Figura 6 – | Fim da instalação do Python 3. | 13 |
| Figura 7 – | Interpretador Python 3. | 14 |
| Figura 8 – | Instalação PyCharm. | 15 |
| Figura 9 – | Passos para instalação do PyCharm. | 15 |
| Figura 10 – | Configurações do PyCharm. | 16 |
| Figura 11 – | Fim da instalação do PyCharm. | 16 |
| Figura 12 – | Criando projeto no PyCharm. | 17 |
| Figura 13 – | Criando arquivo. | 17 |
| Figura 14 – | Nomeando o arquivo. | 18 |
| Figura 15 – | Run'teste001'. | 18 |
| Figura 16 – | Ambiente Colaboratory. | 19 |
| Figura 17 – | Inserir matrizes. | 27 |
| Figura 18 – | Soma de matrizes. | 28 |
| Figura 19 – | Run 'Soma de matrizes'. | 29 |
| Figura 20 – | Soma de matrizes de ordem 2×3 . | 29 |
| Figura 21 – | Soma de matrizes. | 33 |
| Figura 22 – | Soma de matrizes. | 35 |
| Figura 23 – | Soma de matrizes 4×2 . | 35 |
| Figura 24 – | Multiplicação de matrizes. | 44 |
| Figura 25 – | Run' Multiplicação de matrizes'. | 44 |
| Figura 26 – | Laço de repetição While(). | 45 |
| Figura 27 – | Algoritmo matriz transposta. | 48 |
| Figura 28 – | Run 'Algoritmo matriz transposta'. | 48 |
| Figura 29 – | Algoritmo Determinante | 64 |
| Figura 30 – | Run 'Algoritmo Determinante' | 65 |
| Figura 31 – | Run 'Algoritmo Determinante 4×4 '. | 65 |
| Figura 32 – | Instalando biblioteca NumPy | 66 |
| Figura 33 – | Install | 66 |
| Figura 34 – | Determinante Numpy | 68 |
| Figura 35 – | Run 'Determinante Numpy' | 68 |
| Figura 36 – | Algoritmo verificação de matrizes | 72 |
| Figura 37 – | Run 'matriz simétrica' | 72 |
| Figura 38 – | Run 'matriz anti-simétrica' | 73 |
| Figura 39 – | Run 'verificação de matrizes' | 73 |
| Figura 40 – | Algoritmo inversa de matrizes | 78 |
| Figura 41 – | Run 'inversa de matrizes' | 78 |
| Figura 42 – | Algoritmo solucionando | 87 |
| Figura 43 – | Algoritmo solucionando sistema | 87 |
| Figura 44 – | Run 'Algoritmo solucionando sistema' | 88 |
| Figura 45 – | Eliminação de Gauss | 91 |
| Figura 46 – | Run 'Eliminação de Gauss' | 92 |

| | |
|--|----|
| Figura 47 – Treliça | 93 |
| Figura 48 – Forças nos membros de uma treliça. | 95 |

SUMÁRIO

| | |
|---|-----------|
| 1 INTRODUÇÃO | 9 |
| 2 METODOLOGIA ABORDADA | 10 |
| 2.1 Python e Álgebra Linear | 10 |
| 2.2 Instalação do Python e a IDE PyCharm | 11 |
| 2.3 Execução do PyCharm | 17 |
| 3 MATRIZES | 19 |
| 3.1 Definição de Matriz | 19 |
| 3.2 Matriz Linha e Coluna | 20 |
| 3.3 Matriz Retangular | 20 |
| 3.4 Matriz Quadrada | 20 |
| 3.5 Matriz Diagonal e Escalar | 21 |
| 3.6 Matriz Triangular Superior e Inferior | 21 |
| 3.7 Matriz Unitária e Nula | 22 |
| 4 OPERAÇÕES COM MATRIZES E COMO AS INSERIR EM PYTHON | 23 |
| 4.1 Igualdade de Matrizes | 23 |
| 4.2 Adição | 23 |
| 4.3 Soma de Matrizes em Python | 24 |
| 4.3.1 Soma de matrizes com a ordem $m \times n$ definida | 24 |
| 4.3.2 Soma de matrizes de ordem $m \times n$ definida pelo usuário | 30 |
| 4.4 Subtração | 36 |
| 4.5 Subtração de Matrizes em Python | 36 |
| 4.5.1 Subtração de matrizes com a ordem $m \times n$ definida | 36 |
| 4.5.2 Subtração de matrizes de ordem $m \times n$ definida pelo usuário | 36 |
| 4.6 Multiplicação de Matrizes | 37 |
| 4.6.1 Multiplicação de Matrizes por um Escalar | 39 |
| 4.6.2 Multiplicações de matrizes em Python | 40 |
| 5 MATRIZ TRANSPOSTA | 45 |
| 5.1 Matriz Transposta em Python | 46 |
| 6 DETERMINANTE | 48 |
| 6.1 Determinante pelo método de Sarrus | 50 |
| 6.2 Teorema de Laplace | 52 |
| 6.3 Eliminação de Gauss | 55 |
| 6.4 Calculando determinantes em Python | 59 |
| 7 MATRIZES SIMÉTRICAS E ANTI-SIMÉTRICAS | 68 |
| 7.1 Identificando em Python Matrizes Simétricas e Anti-simétricas | 69 |
| 8 MATRIZ INVERSA | 74 |
| 8.1 Matriz Inversa em Python | 76 |
| 9 SISTEMA DE EQUAÇÕES LINEARES | 78 |
| 9.1 Solução de Sistemas Lineares | 81 |

| | |
|---|-----------|
| 9.2 Solução de Sistemas Lineares em Python | 83 |
| 9.2.1 Algoritmo de substituições em Python | 85 |
| 9.2.2 Algoritmo eliminação de Gauss em Python | 88 |
| 10 APLICAÇÃO | 92 |
| 11 CONCLUSÃO | 95 |
| REFERÊNCIAS | 97 |

1 INTRODUÇÃO

Na sociedade atual, as Universidades tem necessidade de dar ênfase à questão da interdisciplinaridade. Tradicionalmente os alunos tendem a aprender disciplinadas separadas, ou seja, sem fazer a interligação de determinados conteúdos para aprimorar o conhecimento ou até mesmo otimizar uma determinada atividade no âmbito acadêmico ou profissional. Sendo assim, a idéia de aprender se torna limitada ao conhecimento individual de determinada disciplina, fazendo com que o aluno se torne refém desse modo de aprendizagem, realizando processos trabalhosos para que a atividade ou o problema seja solucionado de maneira mais complicada.

De acordo com Guimarães (2016), a interdisciplinaridade se apresenta como a inserção de métodos de cooperação entre as disciplinas e outras atividades educacionais, visando o combate à fragmentação do conhecimento, uma vez que, limitada a forma de adquirir conhecimento o estudante realiza processos mais trabalhosos para se encontrar as soluções desejadas ou deixa de absorver o conhecimento de forma mais prática. Sendo assim, nesse trabalho será abordado a idéia de aprender de modo interdisciplinar utilizando duas disciplinas, Álgebra Linear e a Linguagem de Programação, com o intuito de solucionar e otimizar problemas ou atividades que envolvam matrizes, além de realizar aplicações no campo da engenharia.

No decorrer do desse trabalho, têm-se que na seção 2 é mostrado como instalar e executar tanto o Python quanto o Ambiente de Desenvolvimento Integrado PyCharm. Na seção 3 é mostrado os tipos de matrizes, em seguida é dada as operações com matrizes e como inseri-las em Python, bem como desenvolver códigos que realizem as operações. Na seção 5 é dada a definição de matriz transposta e criado o código para que, dada uma matriz sua transposta seja exibida. Na seção 6 é dada a definição de determinante, mostrado os métodos de se calcular o determinante de uma matriz quadrada e desenvolvido o algoritmo utilizando o método de Laplace para encontrar o número associado a uma dada matriz de ordem n . Na seção 7 é dada a definição de matriz simétrica e anti-simétrica para que em seguida seja desenvolvido o algoritmo que identifique em qual desses dois tipos se enquadra uma dada matriz. Na seção 8 é dada a definição de matriz inversa e em seguida utilizando a biblioteca NumPy será criado um código que calcule a inversa de uma dada matriz. Na seção 9 é dada a

definição de sistemas de equações lineares, como classificá-los e utilizada a notação matricial para encontrar a solução de um dado sistema de equações lineares. Ainda na seção 9 é desenvolvido um algoritmo que utiliza substituições e o método da eliminação de Gauss para encontrar o vetor solução de um dado sistema de equações lineares. Por fim, é mostrada uma aplicação no campo da engenharia utilizando a notação matricial num sistemas de equações, onde, para encontrar a solução é utilizado o algoritmo das substituições e eliminação de Gauss desenvolvido.

Tendo em vista que existem inúmeros softwares que envolvem linguagem de programação, nesse trabalho será utilizada a linguagem Python. A linguagem de programação Python, nos últimos anos vem sendo cada vez mais utilizada, uma vez que a mesma apresenta uma linguagem de programação simples e de fácil compreensão, além do próprio software ser leve e possuir fácil manuseio, podendo ser utilizado até mesmo pelo celular.

Para o referido trabalho será dada as definições e conceitos de matrizes para que em seguida utilizando a lógica em programação as matrizes possam ser inseridas em Python. Será desenvolvido códigos que realizem operações com matrizes e possam ser utilizados no dia a dia para solucionar problemas de modo rápido e eficaz. O objetivo geral do projeto é que o aluno utilize a interdisciplinaridade para desenvolver e aprimorar a maneira de adquirir conhecimento e solucionar problemas e o objetivo específico de trabalho é que o estudante compreenda as disciplinas de Álgebra Linear e Linguagem de programação, de modo que uma auxilie no desenvolvimento do conhecimento da outra.

2 METODOLOGIA ABORDADA

2.1 Python e Álgebra Linear

Este trabalho busca a interação de duas disciplinas, Álgebra Linear e Linguagem de Programação, com a intenção de compreender uma com o apoio da outra. Python foi escolhido por ser um Software gratuito e que está disponível em diversas plataformas, com fácil manuseio, por possuir uma sintaxe simples até mesmo para iniciantes. Python possibilita um fácil entendimento por parte do usuário, apresenta um desenvol-

vimento rápido em aplicações além de possuir disponibilidade de bibliotecas aos usuários, as quais são soluções padronizadas para diversos problemas cotidianos (BANIN, 2018). o papel da Linguagem de Programação em Python é desenvolver algoritmos que otimizem o cotidiano de seus usuários, sem que o indivíduo seja obrigado a fazer várias etapas para solucionar seus problemas.

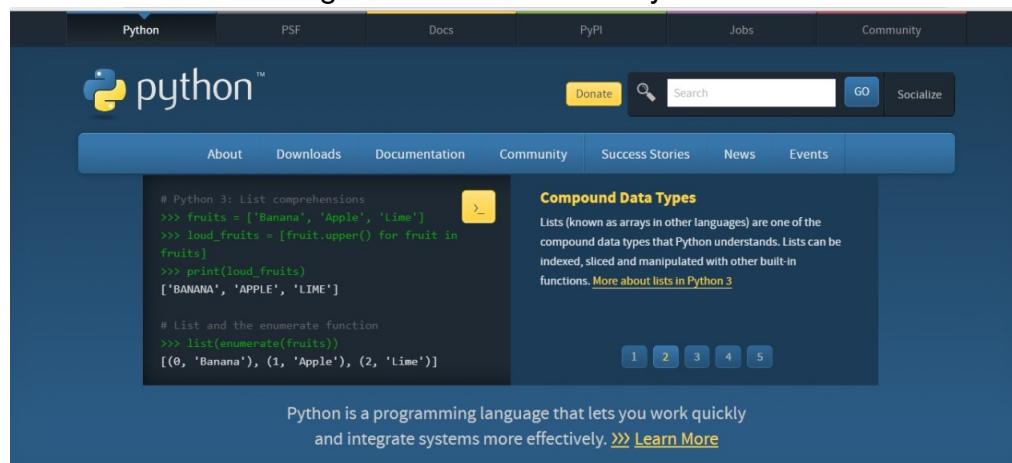
O conteúdo abordado irá tratar especificamente de problemas que envolvam matrizes, buscará desde a compreensão dos conceitos de matrizes até aplicações que as envolvam, mas para que isso ocorra é necessário fazer o download e a instalação do Software, Python.

2.2 Instalação do Python e a IDE PyCharm

Para esse trabalho será apenas explicitado a instalação para o Windows, uma vez que, para os usuários Linux e Mac OS o Python já está pré-instalado.

1º Passo: Digitar no navegador **Python.org**, em seguida clicar no botão **Enter**, logo após irá aparecer o página do site, como na figura abaixo.

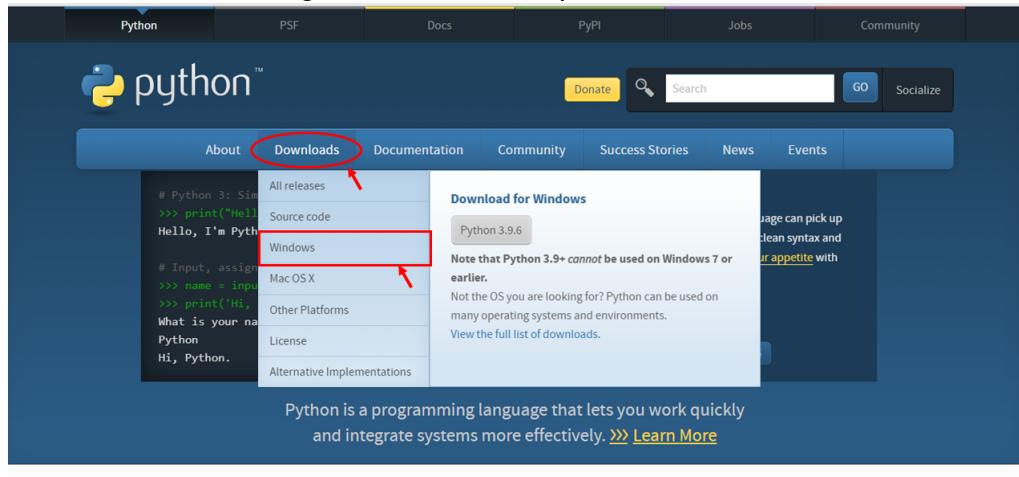
Figura 1: Site oficial do Python.



Fonte: Autoria própria (2021).

2º Passo: Passar o cursor do mouse no nome **Downloads** e em seguida selecionar a opção **Windows**.

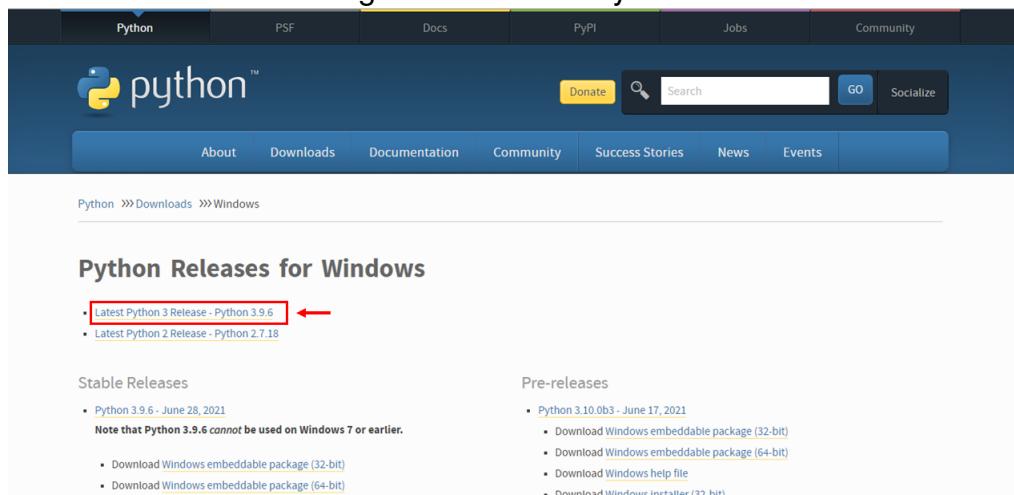
Figura 2: Download para Windows.



Fonte: Autoria própria (2021).

3º Passo: Após clicar em **Windows**, será aberta uma página onde estará disponível várias versões para download e mais acima da página estará as opções de Python 2 e Python 3. Para esse trabalho será feito o download da versão mais atualizada de Python 3.

Figura 3: Versões Python.



Fonte: Autoria própria (2021).

4º Passo: Após selecionar a versão mais atual do Python 3, será aberta uma página onde estará as opções para fazer o download de acordo com as configurações do seu notebook ou computador. Verifique nas configurações do seu aparelho se o sistema operacional é de 64 bits ou 32 bits, no caso será feito o download para o sistema operacional de 64 bits.

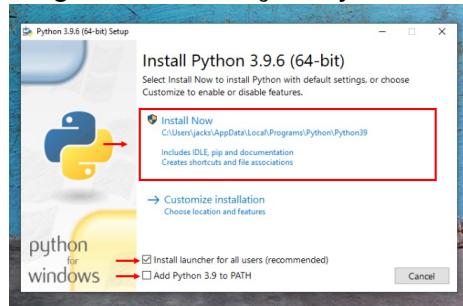
Figura 4: Download Python 3.

| Version | Operating System | Description | MD5 Sum | File Size | GPG |
|-------------------------------------|------------------|--|----------------------------------|-----------|-----|
| Gzipped source tarball | Source release | | 79889d3e866e190f6fe32203c4c560fa | 25640094 | SIG |
| XZ compressed source tarball | Source release | | ecc25a7688f6e550d29db2ee66cf80 | 19051972 | SIG |
| macOS 64-bit Intel installer | Mac OS X | for macOS 10.9 and later | d714923985e0303b9e9b037ef7af815 | 29950653 | SIG |
| macOS 64-bit universal2 installer | Mac OS X | for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental) | 93a208545863d1b9c1a45c8823e034d | 38033506 | SIG |
| Windows embeddable package (32-bit) | Windows | | 5b9693f74979e86a9d463cf73bf0c2ab | 7599619 | SIG |
| Windows embeddable package (64-bit) | Windows | | 89980d3e54160c10554b01f2b9f0a03b | 8448277 | SIG |
| Windows help file | Windows | | 91482c82390caa02acfdacbcbaabf18 | 6501645 | SIG |
| Windows installer (32-bit) | Windows | | 90987973d91d4e2cd8864ee0a54ba7e | 24931328 | SIG |
| Windows installer (64-bit) | Windows | Recommended | ac25cf7f710bf31601ed067cd07deb | 26037888 | SIG |

Fonte: Autoria própria (2021).

5º Passo: Após selecionar a opção anterior, será feito o download do Python 3. Quando o download estiver totalmente concluído, clique no arquivo com o botão direito e depois em **Abrir**. Em seguida selecionar as duas opções, após essa seleção clicar em **Install Now**.

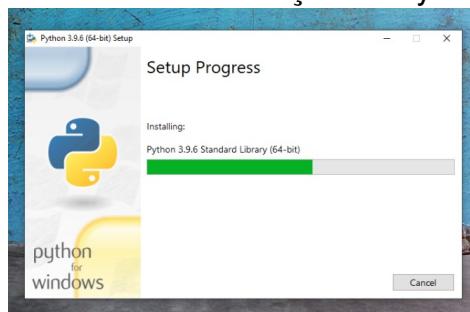
Figura 5: Instalação Python 3.



Fonte: Autoria própria (2021).

6º Passo: Ao clicar em **Install Now**, será pedida a autorização para fazer a instalação, selecionar a opção **Sim**. Após a autorização será feita a instalação.

Figura 6: Fim da instalação do Python 3.



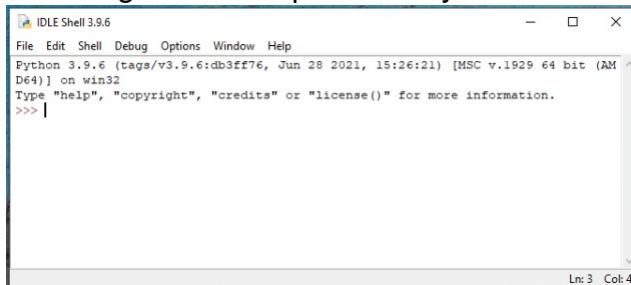
Fonte: Autoria própria (2021).

Finalizada a Instalação recomenda-se clicar na opção **Disable path length limit**, a qual fica a cargo do usuário se deseja desabilitar ou não, em seguida clicar em **Close**.

Ao concluir o passo a passo, estará instalado no Windows o Python 3 e o IDLE. O IDLE é um Ambiente de Desenvolvimento Integrado do Python, nesse ambiente é possível utilizar o modo interativo e todas as instruções do Python, ou seja, testar as instruções sem que seja necessário desenvolver um programa.

Verificando se o IDLE foi instalado adequadamente, clique em **Iniciar** em seguida localize a pasta **Python**, ao abrir a pasta localize o IDLE, que deve possuir tal aparência.

Figura 7: Interpretador Python 3.



Fonte: Autoria própria (2021).

O IDLE é bem simples de ser trabalhado e será utilizado apenas para verificar operações e testar instruções.

Agora, será realizada a instalação da IDE, um Ambiente de Desenvolvimento Integrado no qual estará disponível todas as ferramentas para desenvolver, efetuar e solucionar problemas. Existem várias IDEs e fica a critério do leitor qual escolher para realizar a instalação.

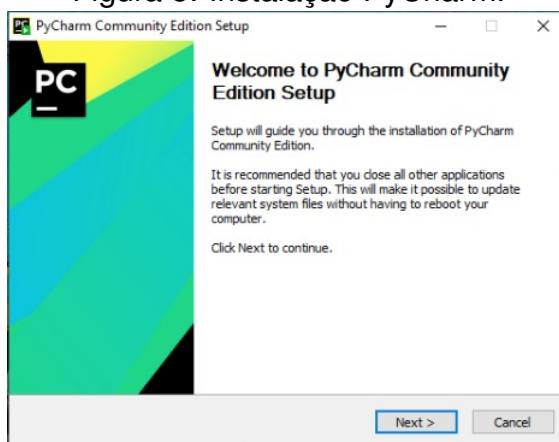
- Spyder
- PyCharm
- Thonny
- Atom
- Jupyter Notebook

Para esse projeto será utilizada o PyCharm por possuir uma interface simples, limpa, personalizável e ser uma ótima IDE para iniciantes em Python.

1º Passo: Abrir o navegador e pesquisar o site oficial do PyCharm ([jetbrains.com/pt-br/pycharm/](https://www.jetbrains.com/pt-br/pycharm/)), em seguida clicar no ícone **Baixar**. Logo após será aberta uma página com duas opções de efetuar o download. A versão a ser baixada é a **Community**, pois é gratuita.

2º Passo: Assim que o download estiver concluído, clicar em cima do arquivo com o botão direito do mouse e em seguida **Abrir**. Após abrir o aplicativo é possível que o dispositivo peça permissão para ser utilizado. Caso o dispositivo peça permissão clicar em **Sim**, ao aceitar a permissão de usar o aplicativo deve aparecer a seguinte guia.

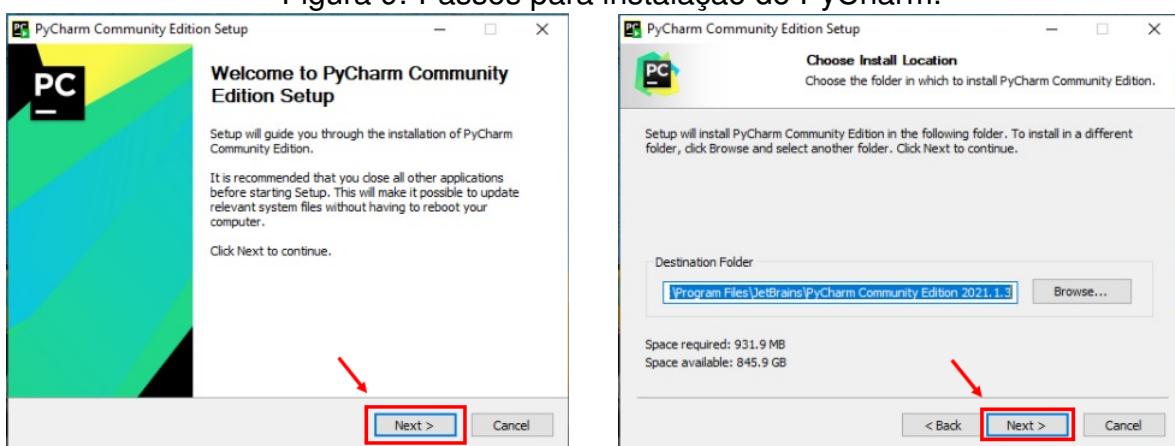
Figura 8: Instalação PyCharm.



Fonte: Autoria própria (2021).

3º Passo: Clicar em **Next**, em seguida será pedido o local para realizar a instalação, clicar em **Next** novamente.

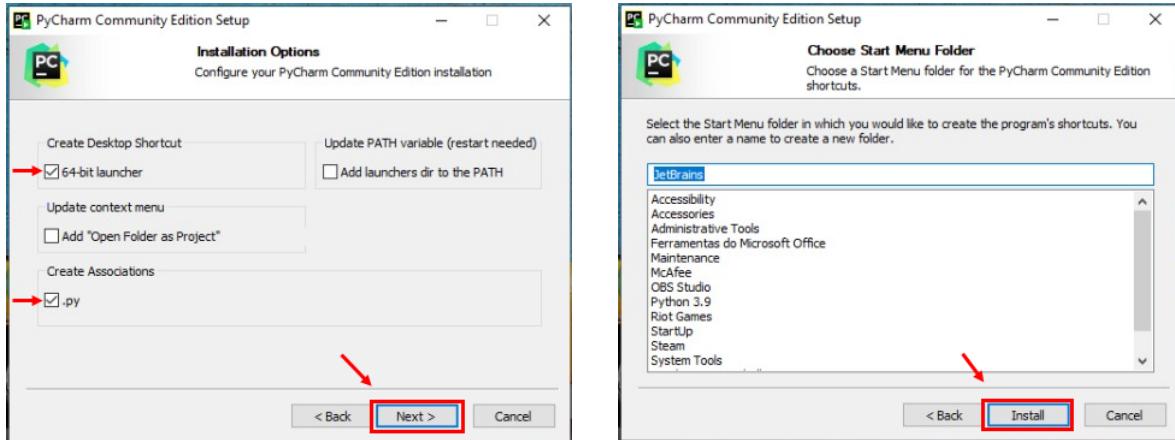
Figura 9: Passos para instalação do PyCharm.



Fonte: Autoria própria (2021).

4º Passo: Será selecionada a opção de 32 bits ou 64 bits, essa opção será selecionada de acordo com o sistema operacional disponível, além disso deve-se selecionar a opção de criar arquivos .py, em seguida clicar em **Next** e depois em **Install**.

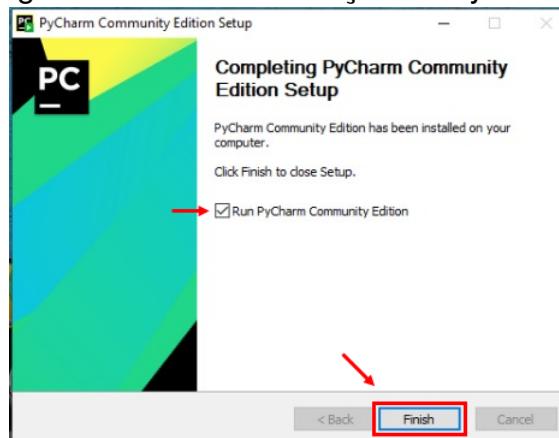
Figura 10: Configurações do PyCharm.



Fonte: Autoria própria (2021).

5º Passo: Finalizada a instalação, selecionar a opção **Run PyCharm Community Edition** e em seguida clicar em **Finish**

Figura 11: Fim da instalação do PyCharm.



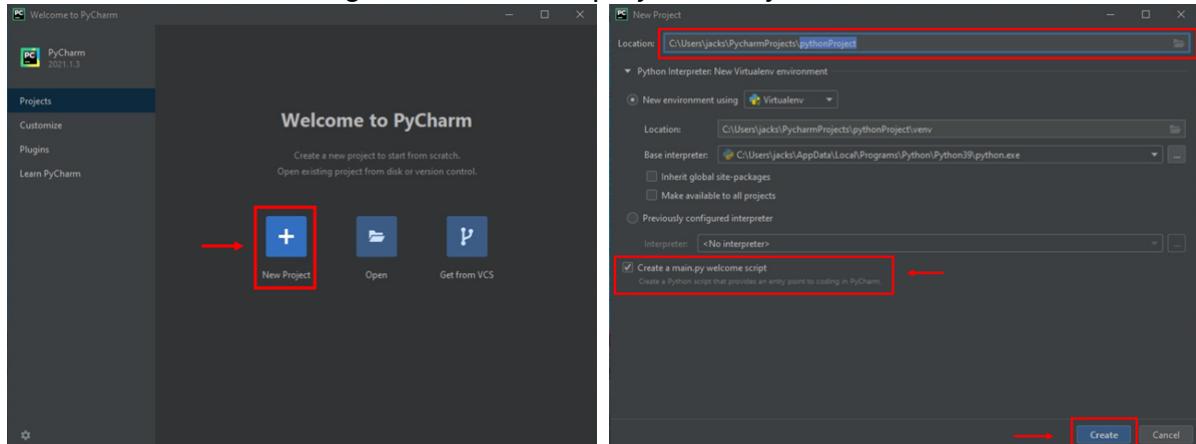
Fonte: Autoria própria (2021).

Finalizado o passo a passo, basta apenas aceitar os termos de uso do PyCharm e selecionar a opção que deseja ou não ajudar a empresa desenvolvedora do Software, após isso o aplicativo irá ser aberto.

2.3 Execução do PyCharm

Ao abrir o PyCharm deve-se criar um novo projeto, e em seguida o nomear. Após escolher o local para salvar o projeto, desmarcar a opção **Create a main.py welcome script** e em seguida clicar em **Create**.

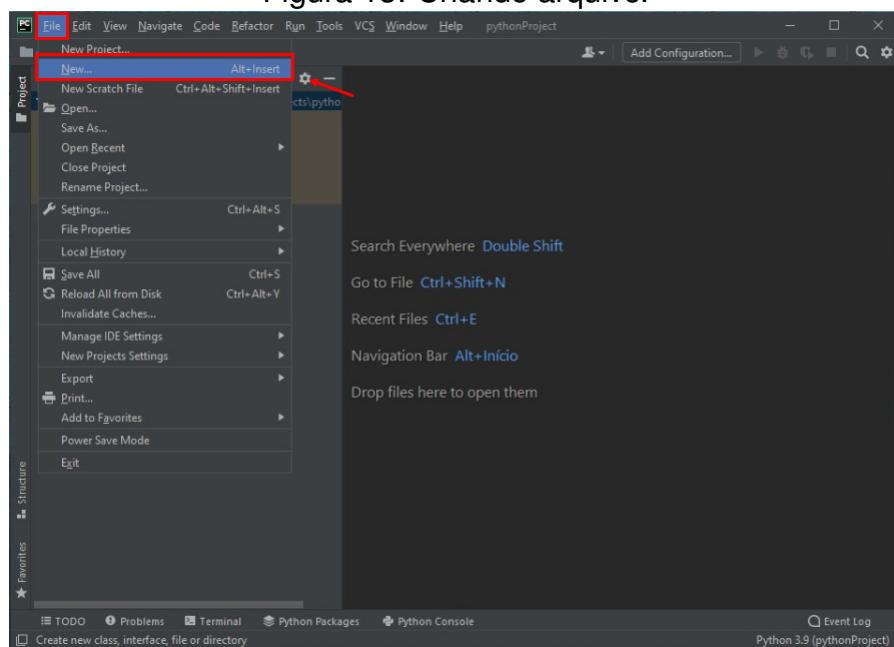
Figura 12: Criando projeto no PyCharm.



Fonte: Autoria própria (2021).

Logo após clicar em **Close**. Para que o leitor possa desenvolver atividades é necessário criar um ou mais arquivos. Sendo assim, o usuário deve clicar em **File** e em seguida **New...**

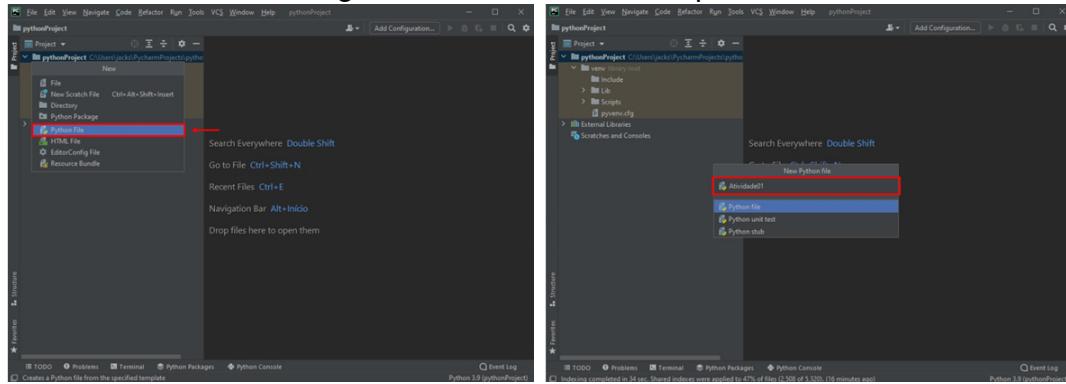
Figura 13: Criando arquivo.



Fonte: Autoria própria (2021).

Em seguida clicar em **Python File**, ao clicar será aberta uma pequena página para inserir o nome do arquivo e para que o arquivo seja salvo e comece a ser utilizado basta clicar em **Enter**.

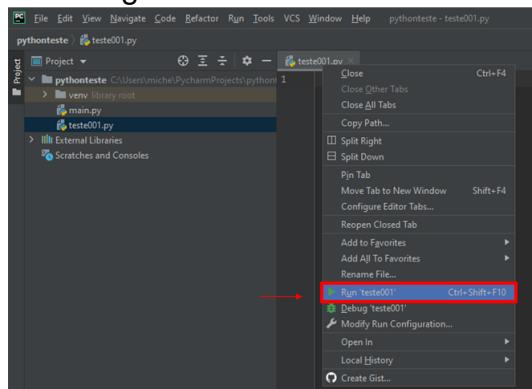
Figura 14: Nomeando o arquivo.



Fonte: Autoria própria (2021).

Finalizado a criação do arquivo, o PyCharm está pronto para criar os códigos. Para executar o código deve-se clicar em cima do arquivo criado com o botão direito do mouse em em seguida clicar em **Run**.

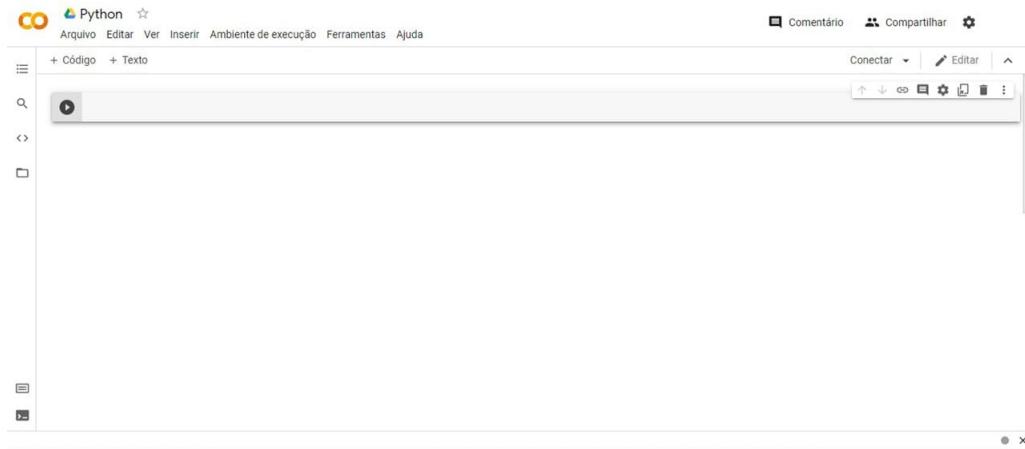
Figura 15: Run'teste001'.



Fonte: Autoria própria (2021).

Outra opção para realizar o desenvolvimento de códigos é o **Google Colab** (colab.research.google.com), o qual basta apenas o usuário possuir uma conta no Google para poder utilizá-lo.

Figura 16: Ambiente Colaboratory.



Fonte: Autoria própria (2021).

3 MATRIZES

3.1 Definição de Matriz

De acordo com Steinbruch (1987) uma matriz equivale a uma tabela disposta em linhas e colunas. A ordem da matriz é a quantidade destas linhas e colunas, uma tabela com m linhas e n colunas terá ordem m por n ($m \times n$). A representação de cada elemento da matriz é dada por a_{ij} , onde i indica a linha e j indica a coluna em que o elemento se encontra. Para uma matriz $m \times n$ temos a seguinte representação:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \cdots & a_{mn} \end{bmatrix}.$$

Os elementos presentes na matriz A podem ser números, funções, polinômios entre outros e como dito anteriormente, a representação dos elementos de uma matriz é dada por a_{ij} , para que o leitor entenda melhor essa nomenclatura será analisado a posição de alguns elementos na matriz A .

O elemento a_{12} : os índices indicam que ele se encontra na linha 1 e coluna 2.

O elemento a_{24} : os índices indicam que ele se encontra na linha 2 e coluna 4.

O elemento a_{43} : os índices indicam que ele se encontra na linha 4 e coluna 3.

Tem-se que a matriz A pode ser representada de forma abreviada por $A = [A_{ij}]$,

onde i está variando de 1 até m e j está variando de 1 até n . A matriz A pode ser escrita como $A_{(m,n)}$, onde m indica a quantidade de linhas e n a quantidade de colunas. Logo para uma matriz $B_{(2,3)}$, temos o seguinte:

$$B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}.$$

3.2 Matriz Linha e Coluna

Uma matriz linha apresenta uma ordem $1 \times n$, onde n irá variar de 1 até n , logo:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \end{bmatrix}.$$

. Uma matriz coluna apresenta uma ordem $m \times 1$, onde m irá variar de 1 até m , logo:

$$B = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \\ \vdots \\ a_{m1} \end{bmatrix}.$$

3.3 Matriz Retangular

Uma matriz retangular possui ordem $m \times n$, onde $m \neq n$, dessa forma um exemplo de matriz retangular $A_{(4,3)}$:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}.$$

3.4 Matriz Quadrada

Uma matriz quadrada possui ordem $n \times n$, assim pode-se dizer apenas matriz de ordem n .

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}.$$

Uma matriz quadrada apresenta uma diagonal principal, que são os elementos a_{ij} em que $i = j$. Assim, temos que os elementos da diagonal principal em uma matriz $n \times n$ são:

$$a_{11}, a_{22}, a_{33}, a_{44}, \dots, a_{nn}.$$

Uma matriz quadrada $A = [a_{ij}]$, também apresenta uma diagonal secundária, onde os elementos que constituem a diagonal secundária em uma matriz quadrada de ordem $n \times n$, são aqueles em que $i + j = n + 1$, ou seja:

$$a_{1n}, a_{2n-1}, a_{3n-2}, a_{4n-3}, \dots, a_{n1}.$$

3.5 Matriz Diagonal e Escalar

Uma matriz quadrada $A = [a_{ij}]$ em que os elementos a_{ij} são iguais a zero para $i \neq j$, é chamada de matriz diagonal.

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & 0 & \cdots & 0 \\ 0 & 0 & 0 & a_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix}.$$

Uma matriz escalar $B = [b_{ij}]$ consiste numa matriz quadrada, na qual os elementos a_{ij} são iguais. Um exemplo é a matriz escalar $B_{(4 \times 4)}$ abaixo:

$$B = \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}.$$

3.6 Matriz Triangular Superior e Inferior

Uma matriz triangular superior possui todos os elementos a_{ij} abaixo da diagonal principal iguais a zero, ou seja, nulos.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & a_{34} & \cdots & a_{3n} \\ 0 & 0 & 0 & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix}.$$

Uma matriz triangular inferior possui todos os elementos a_{ij} acima da diagonal principal iguais a zero, ou seja, nulos.

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}.$$

3.7 Matriz Unitária e Nula

Uma matriz quadrada unitária possui os elementos a_{ij} da diagonal principal iguais a 1 e os demais elementos nulos, logo se for analisado um pouco mais, têm-se que, uma matriz escalar apresentar os elementos a_{ij} de sua diagonal principal também iguais, assim a matriz unitária é uma matriz escalar. Uma matriz unitária é representada por I_n ou apenas I .

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Uma matriz nula Z possui todos os elementos iguais a zero, ou seja nulos.

$$Z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Uma matriz quadrada nula também pode ser classificada como uma matriz diagonal, uma vez que, a definição para matriz diagonal diz que todos os elementos de uma matriz em que $i \neq j$ devem ser iguais a zero. Assim, observando a matriz nula Z os elementos que não compõem a diagonal principal são iguais a zero, ou seja, também classificando uma matriz nula em uma matriz diagonal.

Após estudar os tipos de matrizes, será analisado alguns exemplos. O primeiro exemplo é a matriz $D_{(3,2)}$, dada por:

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

A matriz D é considerada uma matriz retangular e nula, pois $m \neq n$ e todos os seus elementos são zero.

O segundo exemplo é a matriz I_4 , dada por:

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

A matriz I_4 é considerada uma matriz diagonal, unitária e escalar, pois os elementos $i \neq j$ são nulos e a diagonal principal possui os elementos iguais a 1.

4 OPERAÇÕES COM MATRIZES E COMO AS INSERIR EM PYTHON

4.1 Igualdade de Matrizes

Considerando duas matrizes de ordem $m \times n$, uma sendo a matriz $A = [a_{ij}]$ e a outra sendo $B = [b_{ij}]$, tais matrizes são iguais se e somente se $a_{ij} = b_{ij}$. Considerando as matrizes A e B :

$$A = \begin{bmatrix} 2 & 5 & 4 & 9 \\ 3 & 1 & 6 & 8 \\ 5 & 7 & 0 & 1 \end{bmatrix} \text{ e } B = \begin{bmatrix} 2 & 5 & 4 & 9 \\ 3 & 1 & 6 & 8 \\ 5 & 7 & 0 & 1 \end{bmatrix}$$

Analisando as matrizes A e B , temos que $A = B$.

4.2 Adição

Considerando duas matrizes de ordem $m \times n$, uma sendo $A = [a_{ij}]$ e outra matriz $B = [b_{ij}]$, a soma dessas matrizes deve resultar em uma matriz $C = [c_{ij}]$, ou seja, $c_{ij} = a_{ij} + b_{ij}$. Para que tais matrizes possam ser somadas elas devem apresentar a

mesma ordem (STEINBRUCH, 1987).

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \text{ e } B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

As matrizes A e B acima possuem a mesma ordem $m \times n$, logo $c_{ij} = a_{ij} + b_{ij}$

$$C = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \\ a_{31} + b_{31} & a_{32} + b_{32} \end{bmatrix}.$$

PROPRIEDADES DA ADIÇÃO DE MATRIZES

- I. $A + B = B + A$
- II. $A + (B + C) = (A + B) + C$
- III. $A + 0 = 0 + A = A$

4.3 Soma de Matrizes em Python

Matrizes podem ser introduzidas no Python através de listas, mas o que são essas listas?

Listas no Python são sequências ordenadas de valores, as quais podem ser inseridas com dois colchetes, onde os valores que compõem essas listas são chamados de elementos ou itens. Para inserir matrizes em Python será seguida uma sequência de passos ordenados e bem definidos para se chegar a solução. Além disso o usuário também irá desenvolver seus conhecimentos em lógica em programação, pois para que o leitor chegue a solução requer a escolha de "caminhos" para efetuar o passo a passo de maneira mais simples.

4.3.1 Soma de matrizes com a ordem $m \times n$ definida

Utilizando o PyCharm para desenvolver um algoritmo simples que efetue a soma de duas matrizes. Abra o PyCharm e em seguida um projeto, crie um arquivo como mostrado na seção 2.3 e será dado inicio ao desenvolvimento do algoritmo.

Tendo em vista que, um algoritmo deve ter seu desenvolvimento organizado, iremos começar por um exemplo simples. Considere que um usuário deseja realizar uma soma de matrizes, onde o mesmo já irá trabalhar com a ordem que deseja, ou seja, o usuário deixará explícito no código a ordem das matrizes. Uma vez que o usuário

tem a noção que para realizar a soma de duas matrizes A e B de mesma ordem, não será necessário uma limitação no código para que o usuário execute-o, tendo em vista que se caso o usuário inserir matrizes de diferentes ordens o algoritmo não funcionará. Sendo assim, considere duas matrizes de ordem 3×2 , onde a soma também irá resultar numa matriz de ordem 3×2 .

Uma vez que já se tem a ordem da matriz é necessário inseri-lá no algoritmo como uma variável.

```
A = [[0, 0], [0, 0], [0, 0]]
```

Note que a matriz A é formada por três listas que estão dentro de uma lista geral, as quais apresentam elementos nulos para que o próprio usuário possa inseri-los mais a frente. Em seguida, será percorrido cada linha e coluna da matriz para que a mesma possa ser construída e apresente uma forma de tabela. Para construir as linhas e as colunas dessa matriz será empregado a estrutura de repetição **for ... in ...** e a função **range()** a qual retorna uma série numérica no intervalo determinado, ou seja, é ela que irá determinar a quantidade de elementos nas linhas e nas colunas. Percorrendo primeiro as linhas temos a seguinte instrução.

```
for i in range(0, 3):
```

significa que em cada linha retorne a quantidade de elementos contidos no intervalo determinado. Note que as linhas vão de 0 até 3, porém o último número que é o 3 não está dentro do intervalo, fazendo com que a matriz A possua três linhas (0, 1 e 2). Agora, será percorrido as colunas com a seguinte instrução.

```
for j in range(0, 2):
```

significa que em cada coluna retorne a quantidade de elementos contidos no intervalo determinado. Assim, temos duas colunas para a matriz A , pois o último número da função **range()** não entra no intervalo. A seguinte representação irá mostrar como os elementos da matriz estão dispostos com a função **range()**.

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix} \begin{array}{c} 0 \\ 1 \\ 2 \\ 0 \quad 1 \end{array} \quad (1)$$

Uma vez que temos os elementos que formam a matriz A e percorremos as linhas e as colunas, agora será pedido ao usuário o valor dos elementos que irão ocupar as determinadas posições já existentes.

```
A[i][j] = int(input('Digite o valor do elemento da linha {} e '
                     'coluna {}: '.format(i+1, j+1)))
```

Dessa forma foi pedido ao usuário o valor do elemento que ocupa cada posição da matriz. Por se tratar de uma soma o tipo de variável é o inteiro (**int**) para que possa ser efetuada a soma dos elementos, a função **input()** pede ao usuário os dados, já a função **.format()** irá retornar as posições dos elementos **i+1** e **j+1** que foram pedidos dentro das chaves.

```
A[i][j] = int(input('Digite o valor do elemento da linha {i+1} e '
                     'coluna {j+1}: '.format(i+1, j+1)))
```

Como mostrado anteriormente a função **range()** faz com que os elementos da matriz possuam as seguintes numerações de acordo com a equação 1. Agora analisando com a seguinte formatação $i+1$ e $j+1$, que está dentro do parênteses da função **.format()**, teremos a seguinte representação para os elementos.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 1 & 2 \end{matrix}$$

Após o usuário determinar os valores dos elementos, de que forma será exibida a matriz?

Como o objetivo para esse estudo é a representação de matrizes, a forma com que a matriz A deve ser exibida é a de tabela. Para isso, utilizar a função **print()** não será o suficiente para representar a matriz, dessa forma será utilizado o comando **def** que é uma função criada pelo usuário que se ajusta a necessidade do mesmo, assim será criada a seguinte função para exibir a matriz.

```
def exibir_matriz(A):
    for i in A:
        print(i)
exibir_matriz(A)
```

A função **def exibir_matriz(A)** diz que para cada linha i na matriz A mostre (print) a linha e após a instrução a função é finalizada com **exibir_matriz(A)**, dessa forma a matriz A será exibida na forma de tabela. Para inserir a matriz B no Pycharm deve-se seguir o mesmo procedimento, criar a matriz com os elementos nulos, percorrer as linhas e as colunas para estruturar a matriz, pedir ao usuário o valor dos elementos de entrada e em seguida criar uma função para exibi-la. Logo, o algoritmo para inserir as matrizes A e B no PyCharm deve apresentar a mesma lógica mostrada na figura abaixo.

Figura 17: Inserir matrizes.

```

1  A = [[0, 0], [0, 0], [0, 0]]
2  print('Para a Matriz 1:\n')
3  for i in range(0, 3):
4      for j in range(0, 2):
5          A[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
6  print('Matriz 1\n')
7  def exibir_matriz(A):
8      for i in A:
9          print(i)
10     exibir_matriz(A)
11
12    print('\n')
13
14  B = [[0, 0], [0, 0], [0, 0]]
15  print('Para a Matriz 2:\n')
16
17  for i in range(0, 3):
18      for j in range(0, 2):
19          B[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
20  print('Matriz 2\n')
21  exibir_matriz(B)
22  print('\n')

```

Fonte: Autoria própria (2021).

No algoritmo foi utilizada a função **print()** para exibir imprimir textos. Essa função foi inserida no código apenas para mantê-lo mais organizado esteticamente quando for executado.

Para realizar a soma de matrizes o procedimento não muda muito, é necessário criar uma lista com os elementos nulos que irá compor a matriz M , onde os elementos que compõem a matriz M será o resultado da soma das matrizes A e B . Assim, criando a lista para a matriz M .

$M = [[0, 0], [0, 0], [0, 0]]$

Note que a matriz M é composta por listas e os elementos nulos que a compõem serão substituídos mais a frente pela soma dos elementos i e j das matrizes A e B . Em seguida será percorrida as linhas e as colunas para que a mesma apresente a forma de tabela, dessa forma será utilizada a estrutura de repetição **for ... in ...** juntamente com a função **range()**.

```

for i in range(0, 3):
    for j in range(0, 2):

```

Assim, temos que para cada linha i e coluna j retorne a quantidade de elementos contidos no intervalo determinado, após isso será realizada a soma dos elementos de suas respectivas posições.

$$M[i][j] = A[i][j] + B[i][j]$$

Dessa forma efetuamos a soma das matrizes A e B , porém ainda falta exibi-la. Como foi definida anteriormente o comando **def exibir_matriz()**, basta apenas pedir para exibir a matriz M .

```
exibir_matriz(M)
```

A figura abaixo mostra o código completo, com algumas linhas a mais de instruções apenas para manter uma estética agradável.

Figura 18: Soma de matrizes.

```

1  A = [[0, 0], [0, 0], [0, 0]]
2  print('Para a Matriz 1:\n')
3  for i in range(0, 3):
4      for j in range(0, 2):
5          A[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
6  print('Matriz 1:\n')
7  def exibir_matriz(A):
8      for i in A:
9          print(i)
10 exibir_matriz(A)
11 print('\n')
12
13 B = [[0, 0], [0, 0], [0, 0]]
14 print('Para a Matriz 2:\n')
15
16 for i in range(0, 3):
17     for j in range(0, 2):
18         B[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
19 print('Matriz 2:\n')
20 exibir_matriz(B)
21 print('\n')
22
23 print('Soma de matrizes:\n')
24
25 M=[[0, 0], [0, 0], [0, 0]]
26 for i in range(0, 3):
27     for j in range(0, 2):
28         M[i][j] = A[i][j] + B[i][j]
29
30 exibir_matriz(M)

```

Fonte: Autoria própria (2021).

Ao executar o código, a seguinte aba será aberta para solicitar ao usuário os valores dos elementos que ocupam as respectivas posições, além de exibir as matrizes e efetuar a soma das matrizes A e B .

Figura 19: Run 'Soma de matrizes'.

```

Para a Matriz 1:
Digite o valor do elemento da linha 1 e coluna 1: 3
Digite o valor do elemento da linha 1 e coluna 2: 4
Digite o valor do elemento da linha 2 e coluna 1: 5
Digite o valor do elemento da linha 2 e coluna 2: 6
Digite o valor do elemento da linha 3 e coluna 1: 7
Digite o valor do elemento da linha 3 e coluna 2: 8
Matriz 1
[3, 8]
[5, 6]
[7, 9]

Para a Matriz 2:
Digite o valor do elemento da linha 1 e coluna 1: 8
Digite o valor do elemento da linha 1 e coluna 2: 7
Digite o valor do elemento da linha 2 e coluna 1: 9
Digite o valor do elemento da linha 2 e coluna 2: -3
Digite o valor do elemento da linha 3 e coluna 1: 4
Digite o valor do elemento da linha 3 e coluna 2: 6
Matriz 2
[8, 7]
[9, -3]
[4, 6]

Soma de matrizes
[11, 15]
[5, -2]
[4, 12]

```

Fonte: Autoria própria (2021).

Note que o código é bem simples e para que o usuário o utilize corretamente é necessário ter em mente que as matrizes devem possuir a mesma ordem, além disso a quantidade de elementos para as linhas e colunas já estarão definidas no código, logo para que o usuário possa efetuar a soma de matrizes em diferentes ordens é necessário alterar o código, especificamente as listas e o intervalo determinado na função **range()**. Como por exemplo, para efetuar a soma de duas matrizes C e D de ordem 2×3 que irá resultar numa matriz $N_{(2,3)}$.

Figura 20: Soma de matrizes de ordem 2×3 .

```

1  C = [[0, 0, 0], [0, 0, 0]]
2  print('Para a Matriz 1:\n')
3  for i in range(0, 2):
4      for j in range(0, 3):
5          C[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
6  print('Matriz 1\n')
7  def exibir_matriz(C):
8      for i in C:
9          print(i)
10 exibir_matriz(C)
11 
12 print('\n')
13 
14 D = [[0, 0, 0], [0, 0, 0]]
15 print('Para a Matriz 2:\n')
16 
17 for i in range(0, 2):
18     for j in range(0, 3):
19         D[i][j] = int(input('Digite o valor do elemento da linha {} e coluna {}: '.format(i+1, j+1)))
20 print('Matriz 2\n')
21 exibir_matriz(D)
22 print('\n')
23 
24 print('Soma de matrizes\n')
25 
26 N = [[0, 0, 0], [0, 0, 0]]
27 for i in range(0, 2):
28     for j in range(0, 3):
29         N[i][j] = C[i][j] + D[i][j]
30 exibir_matriz(N)

```

Fonte: Autoria própria (2021).

Note que para as matrizes C , D e N , temos a seguinte representação de lista.

Tomando a matriz C , dada por:

```
C = [[0, 0, 0], [0, 0, 0]]
```

As linhas são formadas pela quantidade de listas, para esse exemplo temos duas listas dentro de uma lista geral, logo, temos duas linhas. Já a quantidade de colunas corresponde a quantidade de elementos que uma linha apresenta, ou seja, três colunas.

A função **range()**, dará forma a matriz fazendo com que seja retornada a quantidade determinada de elementos dentro do intervalo e para solicitar ao usuário os valores dos elementos será empregada a função **input()**. Para que a posição de cada elemento seja retornada na forma que estudamos em Álgebra Linear, somamos 1 a posição de cada elemento ($i+1, j+1$) determinado pela função **range()**, assim teremos a seguinte representação para a posição.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 1 & 2 & 3 \end{matrix}$$

Dessa forma, para realizar a soma de matrizes no código que já está definida a ordem das mesmas, basta modificar as listas e o intervalo da função **range()**.

4.3.2 Soma de matrizes de ordem $m \times n$ definida pelo usuário

A lógica em programação considerada para esse tópico foi abordada por Paz (2019). No exemplo anterior a ordem das matrizes já era definida no algoritmo, porém, caso o usuário queira deixar o algoritmo aberto para que seja inserida a quantidade de linhas e colunas que desejar, sem que seja necessário alterar o código, como deve ser a ordem de instruções a ser seguida?

Para esse segundo algoritmo será solicitado ao usuário a quantidade de linhas e colunas que desejar para efetuar a soma, tendo em mente que as matrizes devem possuir a mesma ordem. Criando as variáveis para as linhas e colunas da matriz, temos:

```
m = int(input('Digite a quantidade de linhas: '))
n = int(input('Digite a quantidade de colunas: '))
```

Dessa forma será pedido ao usuário a quantidade de linhas e colunas tanto para a matriz A quanto para a matriz B , o tipo da variável é inteiro (**int()**) para que possa ser efetuada a soma das matrizes mais a frente e a função **input()** é responsável por receber e armazenar os valores digitados pelo usuário.

```
A = []
```

Note que para esse exemplo não foi definido a quantidade de elementos neutros para serem substituídos mais a frente, apenas foi criada a lista que será construída a matriz A . Após criada a lista será percorrida as linhas e as colunas para construir a matriz A , para isso será utilizada a estrutura de repetição **for...in ...** e a função **range()**.

```
for i in range(m):
```

Com isso temos que para cada linha i , retorne a quantidade de elementos no intervalo da função **range()**, ou seja, a quantidade de linhas dada pelo usuário. Porém, como dito anteriormente não temos a matriz A formada por elementos nulos para serem substituídos, sendo assim se torna necessário criar uma lista para as linhas da matriz.

```
linha = []
```

Lembre-se que a matriz A é composta de listas, dessa forma temos uma lista para as linhas dentro de uma lista geral A , com isso percorremos as linhas, agora será percorrida as colunas.

```
for j in range(n):
```

Temos que para cada coluna j retorne a quantidade de elementos no intervalo da função **range()**, que é a quantidade de colunas digitada pelo usuário. Não será necessário criar uma lista para as colunas, pois estamos considerando que a matriz A será formada por listas, ou seja, linhas uma em baixo da outra. A seguir será solicitado ao usuário os valores para cada elemento a_{ij} .

```
E = int(input('Elemento da Matriz A: linha {} e coluna {} é: '
              ''.format(i + 1, j + 1)))
```

Assim, foi solicitado ao usuário o valor do elemento que ocupa cada posição da matriz A . Por se tratar de uma soma o tipo de variável é o inteiro (**int**) para que possa ser efetuada a soma dos elementos mais a frente, a função **input()** pede ao usuário os dados, já a função **.format()** irá retornar as posições dos elementos $i+1$ e $j+1$ que serão pedidos dentro das chaves.

```
linha.append(E)
A.append(linha)
```

Os elementos pedidos ao usuário devem ser adicionados a lista, pois na linha 5 do algoritmo foi apenas criada a linha = [] que irá formar a matriz e não foram definidos os elementos nulos para serem substituídos depois. Dessa forma os valores dos elementos pedidos ao usuário serão adicionados a linha = [] e em seguida a linha = [] será adicionada a matriz A com a função **.append()** que possui a finalidade de adicionar elementos a listas.

Finalizado esses passos a matriz A está construída, porém ela precisar ser exibida na forma de uma tabela. Para exibir a matriz A , será utilizada a função **def()** que é uma função criada pelo usuário que se ajusta a necessidade do mesmo, assim será criada a seguinte função para exibir a matriz.

```
def exibir_matriz(A):
    for linha in A:
        print(linha)
exibir_matriz(A)
```

Assim temos que para cada linha na matriz mostre (print) a linha e finalizamos a função com **exibir_matriz(A)**. Para construir a matiz B deve-se seguir os mesmos passos, porém não é necessário pedir novamente ao usuário a quantidade de linhas e colunas que a matriz irá possuir, pois no começo do algoritmo já foi solicitado e armazenado esses dados. Sendo assim, para a matriz B , basta começar a construção da matriz.

Crie a lista para a matriz B , percorra as linhas e as colunas, peça ao usuário os valores dos elementos da matriz, adicione os elementos as linhas e em seguida adicione as linhas a matriz B , após isso peça para exibir a matriz.

```
B = []
```

```

for i in range(m):
    linha = []
    for j in range(n):
        e = int(input('Elemento da Matriz B: linha {} e '
                      'coluna {} é: '.format(i + 1, j + 1)))
        linha.append(e)
    B.append(linha)
print('\nMatriz B\n')
exibir_matriz(B)

```

Finalizado o passo a passo, o algoritmo para inserir as matrizes A e B no PyCharm deve apresentar a mesma lógica mostrada na figura abaixo.

Figura 21: Soma de matrizes.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProjectTCC - Soma_de_matrizes2.py
pythonProjectTCC teste001.py teste002.py teste003.py teste004.py teste005.py teste006.py Soma_de_matrizes.py Soma_de_m
Project
1 print("SOMA DE MATRIZES")
2 m = int(input("Digite o número de linhas: "))
3 n = int(input("Digite o número de colunas: "))
4 print("Para a matriz A\n")
5 A = []
6 for i in range(m):
7     linha = []
8     for j in range(n):
9         E = int(input("Elemento da Matriz A: linha {} e coluna {} é: ".format(i + 1, j + 1)))
10        linha.append(E)
11    A.append(linha)
12
13 def exibir_matriz(A):
14     for linha in A:
15         print(linha)
16     print("\nMatriz A\n")
17     exibir_matriz(A)
18
19 print("\nPara a matriz B\n")
20
21 B = []
22 for i in range(m):
23     linha = []
24     for j in range(n):
25         e = int(input("Elemento da Matriz B: linha {} e coluna {} é: ".format(i + 1, j + 1)))
26         linha.append(e)
27     B.append(linha)
28     print("\nMatriz B\n")
29     exibir_matriz(B)

```

Fonte: Autoria própria (2021).

A soma das matrizes A e B resultará em uma matriz C , onde primeiro é necessário criar a lista que irá conte-lá.

$C = []$

Após criar a lista para a matriz C , note que a mesma não possui elementos nulos para substitui-los mais a frente pela soma dos elementos das matrizes A e B . Agora será percorrida a quantidade de linhas que irá construir a matriz M .

```

for i in range(m):
    linha = [0] * n

```

```
C.append(linha)
```

Temos que para cada linha i , retorne a quantidade de elementos m que está determinado dentro do intervalo da função **range()**, ou seja, será retornada a quantidade de linhas que o usuário inseriu no inicio do algoritmo. Como a matriz C também será constituída por linhas uma a baixo da outra, será determinado os elementos em cada linha, pois os elementos que irão ocupar essas posições não serão definidos pelo usuário e sim pela soma dos elementos das matriz A e B . Com isso, cada linha i será formada por elementos nulos, onde a quantidade de elementos dessas linhas será determinada pela quantidade de colunas n . Agora será percorrida as colunas da matriz C .

```
for j in range(n):
```

Considerando que a matriz C , será composta por linhas não é necessário criar uma lista para as colunas. Após percorrer as colunas será realizada a soma dos elementos $a_{ij} + b_{ij}$.

$$C[i][j] = A[i][j] + B[i][j]$$

Com isso será determinado cada elemento c_{ij} da matriz C . Com a matriz C construída basta pedir para exibi-lá.

```
exibir_matriz(C)
```

A figura abaixo mostra o código completo, com algumas linhas a mais de instruções apenas para manter uma estética agradável.

Figura 22: Soma de matrizes.

```

1  print('SOMA DE MATRIZES')
2  m = int(input('Digite o número de linhas: '))
3  n = int(input('Digite o número de colunas: '))
4  print('Para a matriz A\n')
5  A = []
6  for i in range(m):
7      linha = []
8      for j in range(n):
9          E = int(input('Elemento da Matriz A: linha {} e coluna {} é: '.format(i + 1, j + 1)))
10         linha.append(E)
11     A.append(linha)
12
13 def exibir_matriz(A):
14     for linha in A:
15         print(linha)
16     print('\nMatriz A\n')
17     exibir_matriz(A)
18
19 print('\nPara a matriz B\n')
20
21 B = []
22 for i in range(m):
23     linha = []
24     for j in range(n):
25         e = int(input('Elemento da Matriz B: linha {} e coluna {} é: '.format(i + 1, j + 1)))
26         linha.append(e)
27     B.append(linha)
28 print('\nMatriz B\n')
29 exibir_matriz(B)
30
31 C = []
32 for i in range(m):
33     linha = [0] * n
34     C.append(linha)
35     for j in range(n):
36         C[i][j] = A[i][j] + B[i][j]
37 print('-' * 30)
38 print("Resultado da soma:\n")
39 exibir_matriz(C)

```

Fonte: Autoria própria (2021).

Ao executar o código, a seguinte aba será aberta para solicitar ao usuário a quantidade de linhas e colunas das matrizes e os valores dos elementos que ocupam as respectivas posições, além de exibir as matrizes e efetuar a soma das matrizes A e B . Na figura abaixo será exibida a soma de duas matrizes A e B de ordem 4×2 .

Figura 23: Soma de matrizes 4×2 .

| Matriz A | Matriz B | Resultado da soma: |
|-----------|-----------|--------------------|
| $[2, 4]$ | $[3, 5]$ | $[5, 9]$ |
| $[7, -1]$ | $[7, 0]$ | $[14, -1]$ |
| $[4, 6]$ | $[2, -2]$ | $[6, 4]$ |
| $[5, -9]$ | $[3, 6]$ | $[8, -3]$ |

Fonte: Autoria própria (2021).

Assim para efetuar a soma de matrizes com diversas ordens basta apenas escolher a quantidade de linhas e colunas, além de inserir os elementos das matrizes.

4.4 Subtração

Para realizar a subtração de duas matrizes de ordem 4×2 considere a matriz $A = [a_{ij}]$ e a matriz $B = [b_{ij}]$, onde a subtração dessas matrizes deve resultar numa matriz $C = [c_{ij}]$ de ordem 4×2 , ou seja, $c_{ij} = a_{ij} - b_{ij}$. Lembrando que as matrizes devem apresentar a mesma ordem para realizar a subtração.

$$A = \begin{bmatrix} 3 & -8 \\ 7 & 9 \\ -2 & 0 \\ 4 & -5 \end{bmatrix} \text{ e } B = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 6 & 9 \\ 8 & 2 \end{bmatrix}$$

Assim a matriz $C = [c_{ij}]$, será igual a:

$$C = \begin{bmatrix} (3 - 1) & (-8 - 3) \\ (7 - 2) & (9 - 5) \\ (-2 - 6) & (0 - 9) \\ (4 - 8) & (-5 - 2) \end{bmatrix} = \begin{bmatrix} 2 & -11 \\ 5 & 4 \\ -8 & -9 \\ -4 & -7 \end{bmatrix}$$

4.5 Subtração de Matrizes em Python

A subtração de matrizes não difere muito da soma, pois a única mudança que será feita para os códigos mostrados anteriormente na soma de matrizes será o sinal da operação.

4.5.1 Subtração de matrizes com a ordem $m \times n$ definida

Considerando que o algoritmo da soma de matrizes já está construído, basta mudar o sinal da soma pelo sinal de subtração ao pedir os elementos para a matriz M .

$$M[i][j] = A[i][j] - B[i][j]$$

Da mesma forma que é para soma também será para a subtração, ou seja, para alterar a ordem das matrizes é necessário modificar as listas e ajustar o intervalo da função `range()` de acordo com a ordem das matrizes.

4.5.2 Subtração de matrizes de ordem $m \times n$ definida pelo usuário

Como o algoritmo da soma de matrizes que pede ao usuário a ordem das mesmas já está construído, basta mudar o sinal da soma pelo sinal de subtração ao pedir os

elementos para a matriz C .

$$C[i][j] = A[i][j] - B[i][j]$$

As funções utilizadas para o algoritmo da subtração de matrizes possuem a mesma finalidade das funções do algoritmo da soma de matrizes.

4.6 Multiplicação de Matrizes

Para efetuar a multiplicação de duas matrizes A e B que apresentem ordem $m \times n$ e $p \times q$ respectivamente, a quantidade de colunas da matriz A deve ser igual a quantidade de linhas da matriz B , isto é, $n=p$.

$$A_{(\underline{m},\underline{n})} \times B_{(\underline{p},\underline{q})}$$

Já a matriz resultante irá apresentar ordem de acordo com a quantidade de linhas da matriz A e a quantidade de colunas da matriz B .

$$A_{(\underline{m},\underline{n})} \times B_{(\underline{p},\underline{q})}$$

EXEMPLO Para realizar a multiplicação considere as matrizes $A_{(3,2)}$ e a matriz $B_{(2,2)}$, dadas por:

$$A = \begin{bmatrix} 5 & -3 \\ 7 & 2 \\ -9 & 0 \end{bmatrix} \text{ e } B = \begin{bmatrix} 1 & -2 \\ 4 & 7 \end{bmatrix}$$

Analizando a ordem das matrizes é possível notar que a quantidade de colunas da matriz A é igual a quantidade de linhas da matriz B , dessa forma a multiplicação de matrizes pode ser realizada e a matriz resultante C , irá possuir ordem 3×2 .

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix} = \begin{bmatrix} 5 \times 1 + (-3) \times 4 & 5 \times (-2) + (-3) \times 7 \\ 7 \times 1 + 2 \times 4 & 7 \times (-2) + 2 \times 7 \\ (-9) \times 1 + 0 \times 4 & (-9) \times (-2) + 0 \times 7 \end{bmatrix}.$$

A matriz resultante C é obtida pela multiplicação de vetores, os quais correspondem a posição de cada elemento.

- O elemento c_{11} é resultado da multiplicação da 1ª linha da matriz A e da 1ª coluna da matriz B .

- O elemento c_{12} é resultado da multiplicação da 1^a linha da matriz A e da 2^o coluna da matriz B .
- O elemento c_{21} é resultado da multiplicação da 2^a linha da matriz A e da 1^a coluna da matriz B .
- O elemento c_{22} é resultado da multiplicação da 2^a linha da matriz A e da 2^a coluna da matriz B .
- O elemento c_{31} é resultado da multiplicação da 3^a linha da matriz A e da 1^a coluna da matriz B .
- O elemento c_{32} é resultado da multiplicação da 3^a linha da matriz A e da 2^a coluna da matriz B .

Assim temos a matriz:

$$C = \begin{bmatrix} -7 & -31 \\ 15 & 0 \\ -9 & 18 \end{bmatrix}.$$

PROPRIEDADES DA MULTIPLICAÇÃO DE UMA MATRIZ POR OUTRA

- I. Considerando as matrizes $A_{(m,n)}$, $B_{(n,p)}$ e $C_{(p,r)}$, temos a seguinte igualdade:
 $(AB)C = A(BC)$.
- II. Considerando as matrizes $A_{(m,n)}$, $B_{(m,n)}$ e $C_{(n,p)}$, temos a seguinte igualdade:
 $(A + B)C = AC + BC$.
- III. Considerando as matrizes $A_{(n,p)}$, $B_{(n,p)}$ e $C_{(m,n)}$, temos a seguinte igualdade:
 $C(A + B) = CA + CB$.
- IV. Considerando a matriz $A_{(m,n)}$, tem-se a seguinte igualdade: $I_m A = A I_n = A$.
- V. Considerando as matrizes $A_{(m,n)}$ e $B_{(n,p)}$, tem-se que para qualquer escalar λ é válida a igualdade: $(\lambda A)B = A(\lambda B) = \lambda(AB)$.
- VI. A multiplicação entre duas matrizes não é, em geral, comutativa.

Exemplo

Considere as matrizes $F_{(3,4)}$ e $G_{(4,3)}$, dadas por:

$$F = \begin{bmatrix} -2 & 6 & 8 & 4 \\ 9 & 4 & -1 & -7 \\ 7 & -3 & 4 & 3 \end{bmatrix} \text{ e } G = \begin{bmatrix} 2 & 10 & 3 \\ 1 & -8 & 0 \\ -4 & 5 & 4 \\ 3 & 6 & -2 \end{bmatrix}.$$

Realizando a multiplicação das matrizes F e G , respectivamente:

$$F \cdot G = \begin{bmatrix} -2 & 6 & 8 & 4 \\ 9 & 4 & -1 & -7 \\ 7 & -3 & 4 & 3 \end{bmatrix} \begin{bmatrix} 2 & 10 & 3 \\ 1 & -8 & 0 \\ -4 & 5 & 4 \\ 3 & 6 & -2 \end{bmatrix} = \begin{bmatrix} -18 & -4 & 18 \\ 5 & 11 & 37 \\ 4 & 132 & 31 \end{bmatrix}.$$

Realizando agora a multiplicação da matriz G e F , respectivamente:

$$G \cdot F = \begin{bmatrix} 2 & 10 & 3 \\ 1 & -8 & 0 \\ -4 & 5 & 4 \\ 3 & 6 & -2 \end{bmatrix} \begin{bmatrix} -2 & 6 & 8 & 4 \\ 9 & 4 & -1 & -7 \\ 7 & -3 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 107 & 43 & 18 & -53 \\ -74 & -26 & 16 & 60 \\ 81 & -16 & -21 & -39 \\ 34 & 48 & 10 & -36 \end{bmatrix}.$$

Observe que a comutatividade de matrizes na multiplicação nem sempre é verdadeira, porém existem casos em que a comutatividade de matrizes na multiplicação pode existir, como por exemplo a multiplicação da matriz E_3 pela matriz I_3 :

$$E \cdot I = \begin{bmatrix} 5 & 2 & 6 \\ 3 & 9 & 1 \\ 7 & -2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 6 \\ 3 & 9 & 1 \\ 7 & -2 & 4 \end{bmatrix}.$$

Realizando a comutatividade na multiplicação de matrizes temos:

$$I \cdot E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 2 & 6 \\ 3 & 9 & 1 \\ 7 & -2 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 6 \\ 3 & 9 & 1 \\ 7 & -2 & 4 \end{bmatrix}.$$

Analizando os resultados obtemos a seguinte igualdade:

$$EI = IE = E$$

4.6.1 Multiplicação de Matrizes por um Escalar

Dada a matriz A de ordem 3×2 e um escalar λ , temos o seguinte produto:

$$B = \lambda \cdot \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \\ e_{31} & e_{32} \end{bmatrix}$$

Dessa forma, tem-se que a matriz resultante C será a multiplicação de todos os elementos da matriz E pelo escalar λ .

$$C = \begin{bmatrix} \lambda \cdot e_{11} & \lambda \cdot e_{12} \\ \lambda \cdot e_{21} & \lambda \cdot e_{22} \\ \lambda \cdot e_{31} & \lambda \cdot e_{32} \end{bmatrix}$$

Exemplo de Multiplicação de Matriz por um Escalar

Considere a matriz D de ordem 2×4 e um escala de valor 2.

$$D = \begin{bmatrix} -2 & 11 & 0 & 4 \\ -9 & 1 & 7 & 3 \end{bmatrix}$$

A multiplicação de cada elemento da matriz D pelo escalar irá resultar numa matriz E também de ordem 2×4 .

$$E = 2 \cdot \begin{bmatrix} -2 & 11 & 0 & 4 \\ -9 & 1 & 7 & 3 \end{bmatrix} = \begin{bmatrix} 2 \cdot (-2) & 2 \cdot 11 & 2 \cdot 0 & 2 \cdot 4 \\ 2 \cdot (-9) & 2 \cdot 1 & 2 \cdot 7 & 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} -4 & 22 & 0 & 8 \\ -18 & 2 & 14 & 6 \end{bmatrix}$$

PROPRIEDADES DA MULTIPLICAÇÃO DE MATRIZES POR UM ESCALAR

- I. $(\lambda\mu)A = \lambda(\mu A)$
- II. $(\lambda + \mu)A = \lambda A + \mu A$
- III. $\lambda(A + B) = \lambda A + \lambda B$
- IV. $1A = A$

4.6.2 Multiplicações de matrizes em Python

Considere duas matrizes $D_{(m,n)}$ e $E_{(p,q)}$, em que $n = p$ e $F_{(m,q)}$ é a matriz resultante do produto de D e E .

Para realizar o produto de matrizes será levado em considerações uma ordem lógica de passos e restrições. Tendo em vista que o usuário deseja solicitar a quantidade de linhas e colunas das matrizes, é necessário criar variáveis para armazenar as informações, sendo assim:

```
m = int(input('Digite a quantidade de linhas para a matriz D: '))
n = int(input('Digite a quantidade de colunas para a matriz D: '))
p = int(input('Digite a quantidade de linhas para a matriz E: '))
q = int(input('Digite a quantidade de colunas para a matriz E: '))
```

Dessa forma, utilizando o comando **input()** foi solicitado ao usuário a quantidade de linhas e colunas tanto para a matriz D quanto para a matriz E . Para que essa

restrição seja atendida será utilizado um laço de repetição **While ()**, o qual irá verificar se $n \neq p$, caso essa condição seja verdadeira o algoritmo retornara novamente ao usuário pedindo a quantidade de linhas e colunas para ambas as matrizes.

```
while (n != p):
    print('A quantidade de colunas da matriz D é diferente da quantidade '
          'de linhas da matriz E!')
    print('Digite novamente a ordem das matrizes!')
    m = int(input('Digite a quantidade de linhas para a matriz D: '))
    n = int(input('Digite a quantidade de colunas para a matriz D: '))
    p = int(input('Digite a quantidade de linhas para a matriz E: '))
    q = int(input('Digite a quantidade de colunas para a matriz E: '))
```

Com isso, caso o usuário digite valores para as linhas e as colunas que não satisfazem a condição da função **While ()**, será solicitado novamente as ordens das matrizes e para encerrar o laço de repetição **While ()**, basta retirar uma endentação, ou seja, retirar um espaço na próxima linha do código. Como feito para a soma de matrizes em Python, será criada as variáveis que irão armazenar as listas para cada matriz, em seguida será percorrida as linhas e as colunas para que as matrizes apresentem forma de tabela, depois será solicitado ao usuário os valores dos elementos d_{ij} e por último esses elementos serão adicionados a linha e esta será adicionada a matriz. Primeiramente construindo a matriz D .

```
D = []
```

Pecorrendo as linhas e as colunas da matriz D para construi-lá.

```
for i in range(m):
    linha = []
    for j in range(n):
        e = int(input('Elemento da Matriz D: linha {} e coluna {} é: '
                      ''.format(i + 1, j + 1)))
        linha.append(e)
    D.append(linha)

def exibir_matriz(D):
```

```

for linha in D:
    print(linha)
exibir_matriz(D)

```

Para construir e exibir a matriz D , será seguido o mesmo procedimento que foi mostrado na seção 4.3.2. Como a matriz D já está construída, agora será construída agora a matriz E .

```

E = []
for i in range(p):
    linha = []
    for j in range(q):
        el = int(input('Elemento da Matriz E: linha {} e coluna {} é: '
                       .format(i + 1, j + 1)))
        linha.append(el)
    E.append(linha)
exibir_matriz(E)

```

O mesmo procedimento para criar a matriz E foi explicado na seção 4.3.2. Construída as matrizes, será realizada a multiplicação das mesmas para se obter a matriz F , de ordem $m \times q$. Primeiro será criada a variável que irá armazenar a lista para a matriz F .

```
F = []
```

Após criar a matriz será percorrida as linhas e as colunas para que ao exibir a matriz ela possa apresentar uma forma de tabela.

```

for i in range(m):
    F.append([])

```

Temos que, para cada linha i retorne a quantidade de elementos no intervalo da função **range()**, ou seja, a quantidade de linhas dada no início do algoritmo pelo usuário, em seguida será adicionado a matriz F uma lista, pois, como na soma as matrizes serão formadas de listas dentro de uma lista geral, o mesmo será feito para a multiplicação. Agora percorrendo as colunas.

```
for j in range(q):
    F[i].append(0)
```

Com isso, para cada coluna j retorne a quantidade elementos no intervalo da função **range()**, ou seja, a quantidade de colunas dada pelo usuário no início do algoritmo. Como se trata de uma soma de multiplicações, foi adicionado a linha os elementos nulos para que possa ser efetuada a soma cumulativa das multiplicações e mais a frente serem substituídos pelos elementos resultantes do produto vetorial. Como se trata de um produto vetorial é necessário mais um laço de repetição para realizar a multiplicação dos elementos.

```
for k in range(n):
    F[i][j] += D[i][k] * E[k][j]
print('Matriz F\n')
```

Como se trata de uma soma cumulativa, é utilizado o sinal de soma (+) antes da igualdade e para exibir a matriz F basta escrever a função **exibir_matriz(F)**, uma vez que a função já foi definida anteriormente. A figura abaixo mostra o algoritmo da multiplicação completo e organizado.

Figura 24: Multiplicação de matrizes.

```

1 m = int(input('Digite a quantidade de linhas para a matriz D: '))
2 n = int(input('Digite a quantidade de colunas para a matriz D: '))
3 p = int(input('Digite a quantidade de linhas para a matriz E: '))
4 q = int(input('Digite a quantidade de colunas para a matriz E: '))
5
6 while (n != p):
7     print('A quantidade de colunas da matriz D é diferente da quantidade de linhas da matriz E!')
8     print('Digite novamente a ordem das matrizes!')
9     m = int(input('Digite a quantidade de linhas para a matriz D: '))
10    n = int(input('Digite a quantidade de colunas para a matriz D: '))
11    p = int(input('Digite a quantidade de linhas para a matriz E: '))
12    q = int(input('Digite a quantidade de colunas para a matriz E: '))
13
14    print('A matriz D irá possuir ordem {} x {}'.format(m, n))
15    print('A matriz E irá possuir ordem {} x {}'.format(p,q))
16    print('A matriz resultante (F) da multiplicação irá possuir ordem {} x {}'.format(m,q))
17    print('\n')
18    D = []
19    for i in range(m):
20        linha = []
21        for j in range(n):
22            e = int(input('Elemento da Matriz D: linha {} e coluna {} é: '.format(i + 1, j + 1)))
23            linha.append(e)
24        D.append(linha)
25    def exibir_matriz(D):
26        for linha in D:
27            print(linha)
28    print('Matriz D\n')
29    exibir_matriz(D)
30    print('\n')
31    E = []
32    for i in range(p):
33        linha = []
34        for j in range(q):
35            el = int(input('Elemento da Matriz E: linha {} e coluna {} é: '.format(i + 1, j + 1)))
36            linha.append(el)
37        E.append(linha)
38    print('Matriz E\n')
39    exibir_matriz(E)
40    print('\n')
41    F = []
42    for i in range(m):
43        F.append([])
44        for j in range(q):
45            F[i].append(0)
46            for k in range(n):
47                F[i][j] += D[i][k] * E[k][j]
48    print('Matriz F\n')
49    exibir_matriz(F)

```

Fonte: Autoria própria (2021).

Uma vez o algoritmo da multiplicação de matrizes desenvolvido, o mesmo será testado o código para duas matrizes D e E de ordem 3×2 e 2×3 , respectivamente. Ao executar o código será solicitado ao usuário os valores dos elementos de ambas as matrizes.

Figura 25: Run' Multiplicação de matrizes'.

```

Digite a quantidade de linhas para a matriz D: 3
Digite a quantidade de colunas para a matriz D: 2
Digite a quantidade de linhas para a matriz E: 2
Digite a quantidade de colunas para a matriz E: 3
A matriz D irá possuir ordem 3 x 2.
A matriz E irá possuir ordem 2 x 3.
A matriz resultante (F) da multiplicação irá possuir ordem 3 x 3.

Elemento da Matriz D: linha 1 e coluna 1 é: 2
Elemento da Matriz D: linha 1 e coluna 2 é: 0
Elemento da Matriz D: linha 2 e coluna 1 é: -2
Elemento da Matriz D: linha 2 e coluna 2 é: 3
Elemento da Matriz D: linha 3 e coluna 1 é: 0
Elemento da Matriz D: linha 3 e coluna 2 é: 1
Matriz D
[2, 0]
[-2, 3]
[0, 1]

Elemento da Matriz E: linha 1 e coluna 1 é: 9
Elemento da Matriz E: linha 1 e coluna 2 é: 3
Elemento da Matriz E: linha 1 e coluna 3 é: -2
Elemento da Matriz E: linha 2 e coluna 1 é: 4
Elemento da Matriz E: linha 2 e coluna 2 é: 1
Elemento da Matriz E: linha 2 e coluna 3 é: 0
Matriz E
[9, 3, -2]
[4, 1, 0]

Matriz F
[50, 14, -4]
[-6, -3, 4]
[49, 16, -10]

Process finished with exit code 0

```

Fonte: Autoria própria (2021).

Para esse algoritmo foi dada uma restrição, a qual diz que a quantidade de colunas

(n) da matriz D deve ser igual a quantidade de linha (p) da matriz E , para verificar se o laço de repetição **while()** está funcionando corretamente, será testado adicionar a ordem de duas matrizes em que $n \neq p$.

Figura 26: Laço de repetição While().

```

Digite a quantidade de linhas para a matriz D: 3
Digite a quantidade de colunas para a matriz D: 1
Digite a quantidade de linhas para a matriz E: 2
Digite a quantidade de colunas para a matriz E: 3
A quantidade de colunas da matriz D é diferente da quantidade de linhas da matriz E!
Digite novamente a ordem das matrizes!
Digite a quantidade de linhas para a matriz D: |

```

Fonte: Autoria própria (2021).

Note que foi inserido no algoritmo duas matrizes em que $n \neq q$, dessa forma o laço de repetição **While()** irá solicitar ao usuário a ordem das matrizes até que a quantidade de colunas n da matriz D seja igual a quantidade de linhas p da matriz E .

5 MATRIZ TRANSPOSTA

Dada a matriz B de ordem 3×2 , tem-se que a matriz transposta B^T irá possuir ordem 2×3 , pois as colunas da matriz B passam a ser as linhas da matriz B^T e as linhas da matriz B passam a ser as colunas da matriz B^T .

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \Rightarrow B^T = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \end{bmatrix}.$$

Exemplo

Considere a matriz C de ordem 4×3 , dada por:

$$C = \begin{bmatrix} -2 & 1 & 10 \\ 9 & -7 & 2 \\ 0 & 4 & 5 \\ -11 & 3 & 8 \end{bmatrix}$$

A matriz transposta C^T apresenta ordem 3×4 , uma vez que, as linhas e as colunas da matriz C passam a ser, respectivamente, as colunas e as linhas da matriz transposta C^T .

$$C^T = \begin{bmatrix} -2 & 9 & 0 & -11 \\ 1 & -7 & 4 & 3 \\ 10 & 2 & 5 & 8 \end{bmatrix}.$$

PROPRIEDADES DA MATRIZ TRANSPOSTA

I. $(A + B)^T = A^T + B^T$.

II. $(\lambda A)^T = \lambda A^T$.

III. $(A^T)^T = A$.

IV. $(AB)^T = B^T A^T$.

5.1 Matriz Transposta em Python

Considere que deseja-se encontrar a matriz transposta de M que possui ordem $r \times s$, sendo assim, a matriz que será obtida T irá apresentar ordem $s \times r$. A idéia do algoritmo é solicitar ao usuário a quantidade de linhas e colunas, deixando o código em aberto para que o usuário possa trabalhar com qualquer ordem de matrizes. De início será criada as variáveis para as linhas e as colunas das matrizes.

```
r = int(input('Digite a quantidade de linhas para a matriz M: '))
s = int(input('Digite a quantidade de colunas para a matriz M: '))
```

Com isso, foi solicitado ao usuário a quantidade de linhas e colunas para a matriz, em seguida será criada a lista para a matriz M e percorrida as linhas e as colunas para construir a matriz em forma de tabela.

```
M = []
for i in range(r):
    linha = []
    for j in range(s):
```

Após percorrer as linhas e as colunas será solicitado ao usuário os valores para os elementos m_{ij} , em seguida esses elementos serão adicionados a linha e a linha adicionada a matriz.

```
e = int(input('Elemento da Matriz M: linha {} e coluna {} é: '
              ''.format(i + 1, j + 1)))
linha.append(e)
M.append(linha)
```

Dessa forma a matriz M de ordem $r \times s$ já está construída, porém, falta exibi-lá. Para exibir a matriz será utilizada a função **def()**.

```
def exibir_matriz(M):
    for linha in M:
        print(linha)
exibir_matriz(M)
```

Após exibir a matriz M será construída a matriz transposta T . A matriz T também será formada por listas, mas não será necessário solicitar novamente ao usuário a quantidade de linhas e colunas, pois a ordem da matriz M já foi dada. Sendo assim, se a ordem da matriz M é $r \times s$ a ordem da matriz T será $s \times r$.

```
T = []
for i in range(s):
    linha =[0]*r
    T.append(linha)
    for j in range(r):
        T[i][j] = M[j][i]
exibir_matriz(T)
```

Dessa forma foi percorrida as linhas e as colunas da matriz transposta T . Considerando que para se obter a matriz transposta de M basta trocar as linhas pelas colunas, podemos escrever a seguinte igualdade no PyCharm:

$$T[i][j] = M[j][i].$$

Por fim, para exibir a matriz transposta basta escrever a função **exibir_matriz(T)**. A figura a baixo mostra o algoritmo completo e organizado de forma lógica.

Figura 27: Algoritmo matriz transposta.

```

1 r = int(input('Digite a quantidade de linhas para a matriz M: '))
2 s = int(input('Digite a quantidade de colunas para a matriz M: '))
3
4 print('A matriz M irá possuir ordem {} x {}'.format(r, s))
5 print('A matriz transposta de M irá possuir ordem {} x {}'.format(s, r))
6 M = []
7 for i in range(r):
8     linha = []
9     for j in range(s):
10         e = int(input('Elemento da Matriz M: linha {} e coluna {} é: '.format(i + 1, j + 1)))
11         linha.append(e)
12     M.append(linha)
13 def exibir_matriz(M):
14     for linha in M:
15         print(linha)
16     print('Matriz M')
17     exibir_matriz(M)
18     print('\n')
19     print('Matriz transposta de M')
20 T = []
21 for i in range(s):
22     linha = [0]*r
23     T.append(linha)
24     for j in range(r):
25         T[i][j] = M[j][i]
26 exibir_matriz(T)
27 print('\n')

```

Fonte: Autoria própria (2021).

Executando o algoritmo para uma matriz M de ordem 2×4 .

Figura 28: Run 'Algoritmo matriz transposta'.

```

Digite a quantidade de linhas para a matriz M: 2
Digite a quantidade de colunas para a matriz M: 4
A matriz M irá possuir ordem 2 x 4.
A matriz transposta de M irá possuir ordem 4 x 2
Elemento da Matriz M: linha 1 e coluna 1 é: 2
Elemento da Matriz M: linha 1 e coluna 2 é: 5
Elemento da Matriz M: linha 1 e coluna 3 é: -1
Elemento da Matriz M: linha 1 e coluna 4 é: 6
Elemento da Matriz M: linha 2 e coluna 1 é: 9
Elemento da Matriz M: linha 2 e coluna 2 é: 7
Elemento da Matriz M: linha 2 e coluna 3 é: 0
Elemento da Matriz M: linha 2 e coluna 4 é: -8
Matriz M
[2, 5, -1, 6]
[9, 7, 0, -8]

Matriz transposta de M
[2, 9]
[5, 7]
[-1, 0]
[6, -8]

```

Fonte: Autoria própria (2021).

6 DETERMINANTE

O Determinante é um número associado a uma matriz quadrada $A = (a_{ij})$. Porém, antes de adentrarmos no assunto de determinantes é necessário dar ênfase em alguns termos e conceitos (BOLDRINI, 1984).

TERMO PRINCIPAL

Considere a matriz $A_{(n,n)}$, tem-se que o produto dos elementos da diagonal principal ($i = j$) recebem o nome de termo principal, ou seja:

$$a_1 \cdot a_2 \cdot a_3 \dots a_n.$$

TERMO SECUNDÁRIO

Dada a matriz $A_{(n,n)}$, tem-se que o produto dos elementos da diagonal secundária ($i + j = n + 1$) recebem o nome de termo secundário, ou seja:

$$a_1 \cdot a_{n-1} \cdot a_{n-2} \dots a_n.$$

Com isso, o determinante de uma matriz $A_{(n,n)}$ é a soma dos produtos obtidos efetuando todas as permutações dos segundos índices do termo principal, mantendo fixos os primeiros índices e adicionando antes dos produtos o sinal positivo (+) ou negativo (-) de acordo com a quantidade de permutações (BOLDRINI, 1984). Porém, como analisar as permutações e os sinais?

Considere a seguinte ordem dos elementos:

$$x \ y \ z.$$

Alterando as posições dos elementos X e Y temos a seguinte permutação:

$$y \ x \ z.$$

A permutação desses elementos (X e Y) é considerada uma permutação inversa, pois realizando novamente a permutação desses elementos é obtida a permutação principal.

$$x \ y \ z.$$

As permutações podem ser classificadas em **par** ou **ímpar**, dependerá do número de inversões. Uma permutação par recebe o sinal positivo (+) e ímpar negativo (-).

Tomando a seguinte sequência como a permutação principal:

$$a \ b \ c.$$

A permutação **a c b** é da classe ímpar por possuir apenas uma inversão e receberá o sinal negativo (-).

A permutação **c a b** é da classe par por possuir duas inversões e receberá o sinal positivo (+).

REPRESENTAÇÃO DE UM DETERMINANTE

Considere a matriz $A = [a_{ij}]$ de ordem n :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}.$$

O determinante da matriz A é denotado por $\det A$ e é colocada entre duas barras verticais, assim:

$$\det A = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{vmatrix}.$$

A ordem de um determinante é equivalente a ordem da matriz que se deseja associar um número, ou seja, se a matriz possuir ordem 2 o determinante também irá possuir ordem 2.

6.1 Determinante pelo método de Sarrus

Para uma matriz A , de ordem 1.

$$A = \begin{bmatrix} 4 \end{bmatrix} \Rightarrow \det A = \begin{vmatrix} 4 \end{vmatrix} = 4.$$

O determinante de uma matriz que possui apenas um elemento (a_{11}) é ele mesmo.

Para uma matriz B , de ordem 2.

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \Rightarrow \det B = \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} = b_{11} \cdot b_{22} - b_{12} \cdot b_{21}$$

Realizando a multiplicação dos elementos da diagonal principal temos:

$$b_{11} \cdot b_{22}$$

Observe que os primeiros índices i da multiplicação do termo principal é 1 e 2.

agora analisando a multiplicação dos elementos da diagonal secundária é obtido:

$$b_{12} \cdot b_{21}.$$

Os índices i para a multiplicação do termo secundário também são 1 e 2, já os índices (j) para a multiplicação do termo principal e secundário são respectivamente 12 e 21, note que o termo secundário é obtido através de uma inversão, a qual recebe o sinal negativo por ser uma inversão ímpar. Dessa forma temos o determinante.

$$\det B = b_{11} \cdot b_{22} - b_{12} \cdot b_{21}.$$

Exemplo

Considere a matriz B , de ordem 2.

$$B = \begin{bmatrix} 5 & -1 \\ 3 & 2 \end{bmatrix} \Rightarrow \det B = \begin{vmatrix} 5 & -1 \\ 3 & 2 \end{vmatrix} = 5 \cdot 2 - [(-1) \cdot 3] = 13$$

Para uma matriz C , de ordem 3 o determinante se dá de uma forma diferente. Para calcular o determinante da matriz C é necessário repetir novamente a frente da matriz as duas primeiras colunas.

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$\det C = \begin{vmatrix} c_{11} & c_{12} & c_{13} & | & c_{11} & c_{12} \\ c_{21} & c_{22} & c_{23} & | & c_{21} & c_{22} \\ c_{31} & c_{32} & c_{33} & | & c_{31} & c_{32} \end{vmatrix}.$$

Analizando o sinal dos produtos de acordo com as permutações dos elementos.

- O 1º produto $(c_{11} \cdot c_{22} \cdot c_{33})$ irá receber sinal positivo, pois a permutação 123 é par.
- O 2º produto $(c_{12} \cdot c_{23} \cdot c_{31})$ irá receber sinal positivo, pois a permutação 231 é par.
- O 3º produto $(c_{13} \cdot c_{21} \cdot c_{32})$ irá receber sinal positivo, pois a permutação 312 é par.
- O 4º produto $(c_{12} \cdot c_{21} \cdot c_{33})$ irá receber sinal negativo, pois a permutação 213 é ímpar.
- O 5º produto $(c_{11} \cdot c_{23} \cdot c_{32})$ irá receber sinal negativo, pois a permutação 132 é ímpar.

- O 6º produto ($c_{13} \cdot c_{22} \cdot c_{31}$) irá receber sinal negativo, pois a permutação 321 é ímpar.

Realizando a soma dos produtos com seus respectivos sinais temos:

$$\det C = +c_{11} \cdot c_{22} \cdot c_{33} + c_{12} \cdot c_{23} \cdot c_{31} + c_{13} \cdot c_{21} \cdot c_{32} - c_{12} \cdot c_{21} \cdot c_{33} - c_{11} \cdot c_{23} \cdot c_{32} - c_{13} \cdot c_{22} \cdot c_{31}.$$

Observe que quando realizamos o produto de cima para baixo dos elementos da esquerda para a direita todos recebem o sinal positivo, já o produto dos elementos de cima para baixo da direita para a esquerda recebem o sinal negativo.

Considere a matriz C , de ordem 3, dada por:

$$C = \begin{bmatrix} 7 & 2 & 1 \\ -6 & -3 & 1 \\ 4 & 5 & 9 \end{bmatrix}$$

$$\det C = \left| \begin{array}{ccc|cc} 7 & 2 & 1 & 7 & 2 \\ -6 & -3 & 1 & -6 & -3 \\ 4 & 5 & 9 & 4 & 5 \end{array} \right|$$

$$\det C = [7 \cdot (-3) \cdot 9] + (2 \cdot 1 \cdot 4) + [1 \cdot (-6) \cdot 5] - [2 \cdot (-6) \cdot 9] - (7 \cdot 1 \cdot 5) - [1 \cdot (-3) \cdot 4]$$

$$\det C = -189 + 8 - 30 + 108 - 35 + 12 = -126.$$

O determinante de uma matriz de ordem 3×3 também pode ser encontrado pelo teorema de Laplace, porém para matrizes que apresentem ordem superior a 3 o método explicado até o momento não é válido.

6.2 Teorema de Laplace

Quando uma matriz quadrada apresenta uma ordem maior ou igual a 3 pode-se utilizar o teorema de Laplace para calcular o determinante. O teorema de Laplace consiste em selecionar uma fila (linha ou coluna) e em seguida somar os produtos de cada elemento pertencente a fila pelo seu respectivo cofator (STEINBRUCH, 1987).

Considere a matriz D de ordem 3, dada por:

$$D = \begin{bmatrix} 3 & -1 & 2 \\ 5 & 4 & -2 \\ 0 & 7 & 1 \end{bmatrix}.$$

Escolhendo a primeira linha da matriz, tem-se que de acordo com o teorema de Laplace o determinante da matriz D será dado por:

$$\det D = d_{11} \cdot \Delta_{11} + d_{12} \cdot \Delta_{12} + d_{13} \cdot \Delta_{13}$$

Sendo que Δ_{ij} corresponde ao cofator relacionado ao elemento d_{ij} .

$$\Delta_{ij} = (-1)^{i+j} \cdot \det D_{ij}$$

Se D é uma matriz de ordem 3×3 , definimos a submatriz (D_{ij}) associado ao elemento d_{ij} , retirando a linha i e a coluna j correspondente a posição do elemento, dessa forma será obtida uma matriz de ordem 2×2 .

Encontrando a submatriz D_{11} associado ao elemento d_{11} , ou seja, eliminando a 1^a linha e a 1^a coluna da matriz D :

$$D_{11} = \begin{bmatrix} 4 & -2 \\ 7 & 1 \end{bmatrix}.$$

Encontrando a submatriz D_{12} associado ao elemento d_{12} , ou seja, eliminando a 1^a linha e a 2^a coluna da matriz D :

$$D_{12} = \begin{bmatrix} 5 & -2 \\ 0 & 1 \end{bmatrix}.$$

Encontrando a submatriz D_{13} associado ao elemento d_{13} , ou seja, eliminando a 1^a linha e a 3^a coluna da matriz D :

$$D_{13} = \begin{bmatrix} 5 & 4 \\ 0 & 7 \end{bmatrix}.$$

Será definido ainda o cofator associado ao elemento d_{ij} , que é determinado por:

$$\Delta_{ij} = (-1)^{i+j} \cdot \det D_{ij}.$$

Encontrando o cofator para o elemento d_{11} que está associado a submatriz D_{11} .

$$\Delta_{11} = (-1)^{1+1} \cdot (18)$$

$$\Delta_{11} = 1 \cdot 18 = 18.$$

Encontrando o cofator para o elemento d_{12} que está associado a submatriz D_{12} .

$$\Delta_{12} = (-1)^{1+2} \cdot (5)$$

$$\Delta_{12} = -1 \cdot 5 = -5.$$

Encontrando o cofator para o elemento d_{13} que está associado a submatriz D_{13} .

$$\Delta_{13} = (-1)^{1+3} \cdot (35)$$

$$\Delta_{13} = 1 \cdot 35 = 35.$$

De acordo com o teorema de Laplace, o determinante da matriz D de ordem 3, será dado por:

$$\det D = d_{11} \cdot \Delta_{11} + d_{12} \cdot \Delta_{12} + d_{13} \cdot \Delta_{13}$$

$$\det D = 3 \cdot 18 + (-1) \cdot (-5) + 2 \cdot 35 = 129.$$

Dessa forma encontramos o determinante da matriz D pelo Teorema de Laplace.

PROPRIEDADES DO DETERMINANTE

- I.** O determinante de uma matriz A não é alterado quando se troca as linhas pelas colunas.
- II.** Se a matriz possuir uma linha ou uma coluna constituída apenas por elementos nulos, o determinante dessa matriz é igual a zero.
- III.** Caso a matriz possua duas linhas ou duas colunas iguais, o determinante dessa matriz é zero.
- IV.** Se uma matriz apresenta duas linhas ou duas colunas que possuem elementos correspondentes proporcionais, o determinante dessa matriz é zero.
- V.** Se em uma matriz A cada elemento de uma linha ou coluna é resultado da soma de duas parcelas, o determinante de A pode ser obtido pela soma dos determinantes de duas matrizes.
- VI.** O determinante de uma matriz triangular superior ou inferior pode ser obtido pela multiplicação dos elementos da diagonal principal.
- VII.** Ao trocar duas linhas ou duas colunas entre si, o determinante muda de sinal, ou seja, fica multiplicado por (-1) .
- VIII.** Quando os elementos de uma linha ou uma coluna são multiplicados por um número real k , o determinante da matriz também deve ser multiplicado pelo número real k .

IX. O determinante de uma matriz não se altera quando se somam os elementos de uma linha ou coluna aos elementos correspondentes de outra linha ou coluna previamente multiplicados por um número diferente de zero.

6.3 Eliminação de Gauss

É possível encontrar o determinante de uma matriz de ordem n através do processo de triangulação, o qual consiste em transformar uma matriz quadrada em uma matriz triangular superior ou inferior (STEINBRUCH, 1987). Para transformar uma matriz quadrada em uma matriz triangular superior ou inferior será utilizada operações elementares entre as linhas.

As operações elementares são as seguintes:

- Trocar a ordem de duas linhas da matriz.
- Multiplicar uma linha da matriz por uma constante diferente de zero.
- A substituição de uma linha da matriz por sua soma com outra linha multiplicada por uma constante diferente de zero.

Para esse trabalho será construída apenas a matriz triangular superior. Lembre-se que em uma matriz triangular superior os elementos que estão abaixo da diagonal principal devem ser nulos. Para o leitor compreender melhor esse método, considere a matriz a seguir de ordem n .

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} & \cdots & e_{1n} \\ e_{21} & e_{22} & e_{23} & e_{24} & \cdots & e_{2n} \\ e_{31} & e_{32} & e_{33} & e_{34} & \cdots & e_{3n} \\ e_{41} & e_{42} & e_{43} & e_{44} & \cdots & e_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & e_{n3} & e_{n4} & \cdots & e_{nn} \end{bmatrix}$$

A primeira linha será manipulada para que o elemento e_{11} seja igual a 1 e através das operações elementares os demais elementos da coluna 1 serão iguais a zero, o mesmo processo será realizado para o elemento e_{22} , a segunda linha será manipulada para que e_{22} seja igual a 1, e os demais elementos que estão abaixo na coluna 2, através de operações elementares serão iguais a zero, e assim por diante.

Diante disso, existem três condições para os elementos da diagonal principal.

1. Caso o elemento seja igual a zero é necessário realizar a troca de linhas e em seguida multiplicar o $\det E$ por -1 , para compensar a troca de linhas e manter o valor do determinante.
2. Considere que o elemento é igual a k , nesse caso deve-se multiplicar todos os elementos que pertencem a mesma linha de k por $\frac{1}{k}$, dessa forma será obtido o número 1 como elemento da diagonal principal dessa linha. Porém, para que o determinante seja conservado deve-se multiplicar o $\det E$ pelo inverso de $\frac{1}{k}$, que é k .
3. Caso o elemento seja igual a 1 não é necessário fazer nada em relação à diagonal principal.

O método da eliminação de Gauss possui muitas etapas, porém é eficiente para matrizes que possuam uma ordem superior a 3. Diante a quantidades de processos repetitivos a serem realizados, será feito um exemplo para uma matriz de ordem 3×3 , para que o leitor possa entender o processo da triangulação de uma matriz.

Considere a seguinte matriz:

$$E = \begin{bmatrix} 5 & 9 & 10 \\ 2 & 6 & 6 \\ 4 & 1 & 9 \end{bmatrix}.$$

De acordo com o algoritmo de calcular o determinante, $\det E = 74$. Agora encontrando o determinante da matriz E pelo método da triangulação, fazendo uso das instruções dadas anteriormente.

1. Multiplicando a 1º linha por $\frac{1}{5}$.

$$\det E = \left| \begin{array}{ccc} 5 & 9 & 10 \\ 2 & 6 & 6 \\ 4 & 1 & 9 \end{array} \right| \longrightarrow L_1 = \left(\frac{1}{5} \right) L_1 \longrightarrow \det E = 5 \cdot \left| \begin{array}{ccc} 1 & \frac{9}{5} & 2 \\ 2 & 6 & 6 \\ 4 & 1 & 9 \end{array} \right|$$

2. Multiplicando a 1º linha por -2 e em seguida somando a 2º linha.

$$det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 2 & 6 & 6 \\ 4 & 1 & 9 \end{vmatrix} \longrightarrow L_2 = L_1(-2) + L_2 \longrightarrow det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 0 & \frac{12}{5} & 2 \\ 4 & 1 & 9 \end{vmatrix}$$

3. Multiplicando a 1º linha por -4 e em seguida somando a 3º linha.

$$det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 0 & \frac{12}{5} & 2 \\ 4 & 1 & 9 \end{vmatrix} \longrightarrow L_3 = L_1(-4) + L_3 \longrightarrow det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 0 & \frac{12}{5} & 2 \\ 0 & -\frac{31}{5} & 1 \end{vmatrix}$$

4. Multiplicando a 2º linha por $\frac{31}{12}$ e em seguida somando a 3º linha.

$$det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 0 & \frac{12}{5} & 2 \\ 0 & -\frac{31}{5} & 1 \end{vmatrix} \longrightarrow L_3 = L_2\left(\frac{31}{12}\right) + L_3 \longrightarrow det E = 5 \cdot \begin{vmatrix} 1 & \frac{9}{5} & 2 \\ 0 & \frac{12}{5} & 2 \\ 0 & 0 & \frac{37}{6} \end{vmatrix}$$

Dessa forma temos que o determinante da matriz E , pelo método da triangulação é:

$$det E = 5 \cdot 1 \cdot \frac{12}{5} \cdot \frac{37}{6} \\ det E = 74.$$

Esse método de triangulação também é conhecido como eliminação de Gauss e para o primeiro elemento não nulo de cada linha da-se o nome de pivô. Observe que não necessariamente os termos da diagonal principal devem ser iguais a 1 para se obter o determinante, basta realizar as operações elementares para que os elementos abaixo da diagonal principal sejam iguais a zero.

A triangulação da matriz E , sem que os elementos da diagonal principal sejam iguais a 1 é:

$$E = \begin{bmatrix} 5 & 9 & 10 \\ 0 & \frac{12}{5} & 2 \\ 0 & 0 & \frac{37}{6} \end{bmatrix}.$$

Onde, o determinante é apenas a multiplicação dos termos da diagonal principal.

$$\det E = 5 \cdot \frac{12}{5} \cdot \frac{37}{6} = 74.$$

MATRIZES EQUIVALENTE

Considerando duas matrizes A e B , têm-se que B será equivalente a A se possuir o mesmo determinante que B e se por meio de finitas operações elementares é possível transformar A em B (STEINBRUCH, 1987).

Considere a matriz F a seguir, dada por:

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 10 & 4 & 9 \\ 5 & 9 & -2 \end{bmatrix}.$$

O determinante da matriz A será:

$$\det A = \begin{vmatrix} 5 & 1 & 2 \\ 10 & 4 & 9 \\ 5 & 9 & -2 \end{vmatrix} = 5 \cdot 4 \cdot (-2) + 1 \cdot 9 \cdot 5 + 2 \cdot 10 \cdot 9 - [1 \cdot 10 \cdot (-2)] - (5 \cdot 9 \cdot 9) - (2 \cdot 4 \cdot 5) = -240.$$

Realizando a triangulação da matriz F para analisar o seu determinante pelo método de Sarrus e Laplace.

1. A 2º linha será substituída pela seguinte operação elementar: $L_1 \cdot (-1) + L_2$.

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 0 & 2 & 5 \\ 5 & 9 & -2 \end{bmatrix}.$$

2. A 3º linha será substituída pela seguinte operação elementar: $L_1 \cdot (-1) + L_3$.

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 8 & -4 \end{bmatrix}.$$

3. A 3º linha será substituída pela seguinte operação elementar: $L_2 \cdot (-4) + L_3$.

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -24 \end{bmatrix}.$$

Observe que não foram utilizadas operações elementares que alterassem o valor do determinante, dessa forma realizando a multiplicação dos termos da diagonal principal é encontrado o valor de -240 para o determinante.

Encontrando o determinante da matriz A escalonada pelo método de **Sarrus**.

$$A = \begin{vmatrix} 5 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -24 \end{vmatrix}$$

$$\det A = 5 \cdot 2 \cdot (-24) + 1 \cdot 5 \cdot 0 + 2 \cdot 0 \cdot 0 - [1 \cdot 0 \cdot (-24)] - (5 \cdot 5 \cdot 0) - (2 \cdot 2 \cdot 0) = -240.$$

O único produto que não possui zero é o termo principal, em todos os outros produtos irá aparecer pelo menos um zero.

Encontrando agora o determinante pelo **Teorema de Laplace**, lembre-se que para encontrar o determinante por Laplace deve-se utilizar a linha ou a coluna que possuir mais zeros, dessa forma o cálculo ficará mais simples.

Escolhendo a 1ª coluna para obter o determinante.

$$A = \begin{vmatrix} 5 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -24 \end{vmatrix} = 5 \cdot \begin{vmatrix} 2 & 5 \\ 0 & -24 \end{vmatrix} - 0 \cdot \begin{vmatrix} 1 & 2 \\ 0 & -24 \end{vmatrix} + 0 \cdot \begin{vmatrix} 1 & 2 \\ 2 & 5 \end{vmatrix} = -240.$$

Note que no Teorema de Laplace os elementos a_{12} e a_{13} são zeros, ou seja o determinante da matriz A irá se reduzir ao seguinte produto:

$$\text{Det } A = a_{11} \cdot \Delta_{11}$$

Onde, $\Delta_{11} = 1^{1+1} \cdot \det a_{11}$.

6.4 Calculando determinantes em Python

Dada uma matriz de ordem n , deseja-se calcular o determinante dessa matriz e para isso deve escolher um método (Álgebra e Linguagem de Programação) que

possa ser utilizado para qualquer ordem. Será utilizado o Teorema de Laplace para solucionar esse problema, dessa forma começando a escrita do código temos:

```

n = int(input('Digite a ordem da matriz: '))
A = []
for i in range(n):
    linha = []
    for j in range(n):
        E = int(input('Elemento da Matriz A: linha {} e coluna {} é: '
                      .format(i + 1, j + 1)))
        linha.append(E)
    A.append(linha)

def exibir_matriz(A):
    for linha in A:
        print(linha)
print('\nMatriz A\n')
exibir_matriz(A)

```

Dessa forma foi solicitada a ordem n para a matriz quadrada A , em seguida foi criada a lista **A=[]**. Com isso, foi percorrida as linhas e as colunas, solicitado os valores dos elementos que irão compor a matriz ao usuário e por fim, exibida a matriz A (PAZ, 2018). O algoritmo irá ser efetuado de acordo com o Teorema de Laplace, porém iremos fazer a análise primeiro para uma matriz de ordem 2×2 . Lembre-se que a função **range()** começa a contagem dos elementos a partir do zero, ou seja, se a matriz possuir ordem 2×2 , a função irá realizar a contagem 0 e 1.

Dada a matriz A :

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$$

tem-se que o determinante da matriz A de ordem 2, pode ser dado da seguinte forma:

$$\det A = \begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} = a_{00} \cdot a_{11} - a_{01} \cdot a_{10}.$$

Esse método é conhecido como Regra de Sarrus, e pode ser usado para encontrar o determinante de matrizes que possuam ordem 2 ou 3, para matrizes que possuam ordem superior a 3 essa regra não é válida, já o Teorema de Laplace permite encontrar o determinante para matrizes que possuam ordem igual ou superior a 3. Com isso será criada uma função **def determinante(A)**, para que no final possa ser retornado o determinante da matriz, em seguida será definido uma variável nula para o determinante, para que o mesmo possa ser somado a outros valores, pois, o determinante é resultado da soma de produtos. Dessa forma o algoritmo irá possuir condições, uma para quando a matriz possuir ordem igual a 2 e outra para quando a matriz possuir ordem superior a 2. Se a matriz possuir ordem igual a 2×2 , deve-se criar um laço de repetição **if**, que verifica se a condição **len(A) == 2** é verdadeira. Caso a condição seja verdadeira, será executado o comando para calcular o determinante da matriz 2×2 , a condição **len (A)** é responsável por confirmar o "tamanho" da matriz.

```
def determinante(A):
    det = 0
    if len(A) == 2:
        det = (A[0][0] * A[1][1]) - (A[1][0] * A[0][1])
    return det
```

Concluída a primeira condição, será construída a segunda condição **if** para quando a matriz tiver ordem superior a 2×2 .

```
if len(A) > 2:
    for j in range(len(A)):
        det += A[0][j] * (-1)**(0+j) * determinante(remover(A,0,j))
    return det
```

Nessa segunda condição **len(A)> 2**, será efetuado o determinante pelo Teorema de Laplace e para que o leitor possa compreender essa segunda condição, considere a seguinte matriz *A* de ordem 3×3 .

$$A = \begin{bmatrix} -1 & 4 & 9 \\ 7 & -5 & -2 \\ 0 & 3 & 6 \end{bmatrix}$$

Para a segunda condição do algoritmo **if len(A)>2**, a linha que será sempre retirada para obter a submatriz é a zero, o que irá variar será a coluna *j*.

Encontrando o cofator Δ_{00} :

$$B = \begin{bmatrix} -1 & 4 & 9 \\ 7 & -5 & -2 \\ 0 & 3 & 6 \end{bmatrix}$$

$$\Delta_{00} = (-1)^{0+0} * \det A'_{00}$$

$$\Delta_{00} = 1 * (-24)$$

$$\Delta_{00} = -24.$$

Encontrando o cofator Δ_{01} :

$$B = \begin{bmatrix} -1 & 4 & 9 \\ 7 & -5 & -2 \\ 0 & 3 & 6 \end{bmatrix}$$

$$\Delta_{01} = (-1)^{0+1} * \det A_{01}$$

$$\Delta_{01} = -1 * (42)$$

$$\Delta_{01} = -42.$$

Encontrando o cofator Δ_{02} :

$$B = \begin{bmatrix} -1 & 4 & 9 \\ 7 & -5 & -2 \\ 0 & 3 & 6 \end{bmatrix}$$

$$\Delta_{02} = (-1)^{0+2} * \det A_{02}$$

$$\Delta_{02} = 1 * (21)$$

$$\Delta_{02} = 21.$$

O determinante da matriz A é dada pela soma dos produtos dos elementos da primeira linha pelos seus respectivos cofatores, logo:

$$\det A = a_{11} \cdot \Delta_{11} + a_{12} \cdot \Delta_{12} + a_{13} \cdot \Delta_{13}$$

$$\det A = (-1) \cdot (-24) + 4 \cdot (-42) \cdot 9 \cdot 21 = 45$$

Porém, como será obtida as submatrizes dentro do algoritmo?

Será criada uma nova função **def** para armazenar apenas os elementos que irão pertencer a submatriz.

```
def remover(A,remover_i,remover_j):
    B = [[int(0) for i in range(len(A)-1)] for j in range(len(A)-1)]
```

Assim, temos uma nova matriz **B** composta por elementos inteiros nulos com uma ordem a menos que a matriz dada inicialmente, ou seja, se a matriz possuir ordem 3×3 , a submatriz irá possuir ordem 2×2 .

```

I = 0
for i in range(len(A)):
    J = 0
    for j in range(len(A)):

```

A posição de cada elemento para a linha I da nova matriz B irá começar em zero e cada linha da matriz antiga possuirá o mesmo tamanho **len(A)**. A posição de cada elemento para a coluna J da nova matriz B irá começar em zero e as colunas da matriz antiga possuiram o mesmo tamanho **len(A)**. Dessa forma I e J irão percorrer a matriz nova e i e j irão percorrer a matriz antiga.

```

if j != remover_j:
    B[I][J] = A[i][j]
    J += 1

```

Para a condição **j!=remover_j**, iremos fazer a seguinte analogia.

Lembre-se que para esse algoritmo a linha 0 é fixa, ou seja, não varia. Já a posição da coluna não é fixa, significa dizer que quando **j = 0**, os elementos que pertencem a coluna 0 não serão gravados, dessa forma será somado 1 a posição 0, armazenando assim apenas aqueles elementos que compõem a primeira submatriz a ser encontrada. O próximo valor a entrar na condição **j!=remover_j** será **j = 1**, sendo assim, os elementos que pertencem a coluna 1 não serão gravados e a esses elementos será adicionado 1 a sua posição. Dessa forma, serão gravados apenas aqueles elementos que compõem a segunda submatriz a ser encontrada e por fim, **j = 2** significa que os elementos que pertencem a coluna 2 não serão gravados e a esses elementos será adicionado 1 a sua posição, armazenando assim apenas os elementos que compõem a terceira submatriz a ser encontrada.

```

if i != remover_i:
    I += 1
return B

```

Após ser retornada a nova matriz B , a mesma irá entrar no laço de repetição **if len(A) == 2**, caso ela possua ordem igual a dois será calculado o determinante da submatriz, e em seguida será calculado o cofator para o elemento que corresponde a linha e a coluna retirada. Dessa forma, será calculado os seguintes cofatores:

```

A[0][0] * (-1) **(0 + 0) * determinante(remover(A, 0, 0))
A[0][1] * (-1) **(0 + 1) * determinante(remover(A, 0, 1))
A[0][2] * (-1) **(0 + 2) * determinante(remover(A, 0, 2))

```

Após ser calculado os três cofatores o algoritmo irá soma-los e em seguida irá exibir o determinante.

```
print("Determinante : ", determinante(A))
```

Após exibir o determinante o algoritmo é encerrado. A figura abaixo irá mostrar o algoritmo completo.

Figura 29: Algoritmo Determinante

```

1  n = int(input('Digite a ordem da matriz: '))
2  A = []
3  for i in range(n):
4      linha = []
5      for j in range(n):
6          E = int(input('Elemento da Matriz A: linha {} e coluna {} é: '.format(i + 1, j + 1)))
7          linha.append(E)
8      A.append(linha)
9
10 def exibir_matriz(A):
11     for linha in A:
12         print(linha)
13     print('\nMatriz A\n')
14 exibir_matriz(A)
15
16 def determinante(A):
17     det = 0
18     if len(A) == 2:
19         det = (A[0][0] * A[1][1]) - (A[1][0] * A[0][1])
20         return det
21     if len(A) > 2:
22         for j in range(len(A)):
23             det += A[0][j] * (-1)**(0+j) * determinante(remover(A, 0, j))
24         return det
25
26 def remover(A, remover_i, remover_j):
27     B = [[int(0) for j in range(len(A)-1)] for i in range(len(A)-1)]
28     I = 0
29     for i in range(len(A)):
30         J = 0
31         for j in range(len(A)):
32             if j != remover_j:
33                 B[I][J] = A[i][j]
34                 J += 1
35             if i != remover_i:
36                 I += 1
37     return B
38
39 print()
39 print("Determinante : ", determinante(A))

```

Fonte: Autoria própria (2021).

Note que caso seja inserida uma matriz de ordem 4×4 , o algoritmo irá rodar até o momento em que a submatriz possua ordem 2×2 , para que seja calculado os cofatores e em seguida somados.

Executando o algoritmo para a matriz A de ordem 3×3 .

Figura 30: Run 'Algoritmo Determinante'

```

Digitte a ordem da matriz: 3
Elemento da Matriz A: linha 1 e coluna 1 é: -1
Elemento da Matriz A: linha 1 e coluna 2 é: 4
Elemento da Matriz A: linha 1 e coluna 3 é: 9
Elemento da Matriz A: linha 2 e coluna 1 é: 7
Elemento da Matriz A: linha 2 e coluna 2 é: -5
Elemento da Matriz A: linha 2 e coluna 3 é: -2
Elemento da Matriz A: linha 3 e coluna 1 é: 0
Elemento da Matriz A: linha 3 e coluna 2 é: 3
Elemento da Matriz A: linha 3 e coluna 3 é: 6

Matriz A

[-1, 4, 9]
[7, -5, -2]
[0, 3, 6]

Determinante : 45

Process finished with exit code 0

```

Fonte: Autoria própria (2021).

Executando o algoritmo para uma matriz de ordem 4×4

Figura 31: Run 'Algoritmo Determinante 4×4 '

```

Digitte a ordem da matriz: 4
Elemento da Matriz A: linha 1 e coluna 1 é: -2
Elemento da Matriz A: linha 1 e coluna 2 é: 0
Elemento da Matriz A: linha 1 e coluna 3 é: 7
Elemento da Matriz A: linha 1 e coluna 4 é: 1
Elemento da Matriz A: linha 2 e coluna 1 é: 0
Elemento da Matriz A: linha 2 e coluna 2 é: -1
Elemento da Matriz A: linha 2 e coluna 3 é: 2
Elemento da Matriz A: linha 2 e coluna 4 é: 9
Elemento da Matriz A: linha 3 e coluna 1 é: 4
Elemento da Matriz A: linha 3 e coluna 2 é: -3
Elemento da Matriz A: linha 3 e coluna 3 é: 1
Elemento da Matriz A: linha 3 e coluna 4 é: 7
Elemento da Matriz A: linha 4 e coluna 1 é: 0
Elemento da Matriz A: linha 4 e coluna 2 é: -6
Elemento da Matriz A: linha 4 e coluna 3 é: 2
Elemento da Matriz A: linha 4 e coluna 4 é: 1

Matriz A

[-2, 0, 7, 1]
[0, -1, 2, 9]
[4, -3, 1, 7]
[0, -6, 2, 1]

Determinante : 328

Process finished with exit code 0

```

Fonte: Autoria própria (2021).

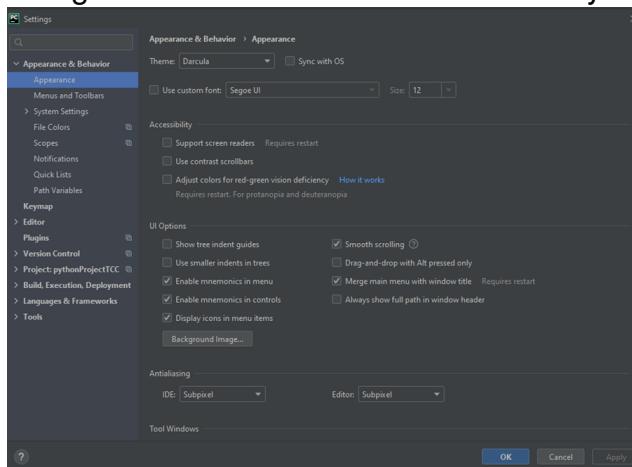
Como observado, o algoritmo para calcular determinante possui uma sequência lógica de passos a ser seguidos, contudo existem outras formas de calcular o determinante de modo mais prática utilizando a biblioteca **NumPy**. Bibliotecas em Python são coleções de módulos de script acessíveis, utilizadas para simplificar o processo de programação sem que seja necessário reescrever os comandos mais usados. A biblioteca NumPy é considerada uma das bibliotecas mais importantes em Python, ela é capaz de realizar cálculos altamente complexos em Álgebra Linear, é utilizada na

Computação Científica e Numérica, além de fornecer um tipo de **Array** de alta performance, conhecido como **ndarray** que significa array de n dimensões.

Array em NumPy é uma tabela multidimensional de elementos do mesmo tipo, por exemplo *int* e *float*, dessa forma um array é uma estrutura de dados é semelhante ás listas do Python. A biblioteca **NumPy** é base para inúmeras bibliotecas importantes no Python, as quais envolvem computação quantica, computação estatística, processamento de sinal, gráficos e redes, análise matemática entre outros.

Para instalar a biblioteca **NumPy** no PyCharm clique em **File**, em seguida **Settings** e será aberta a seguinte guia.

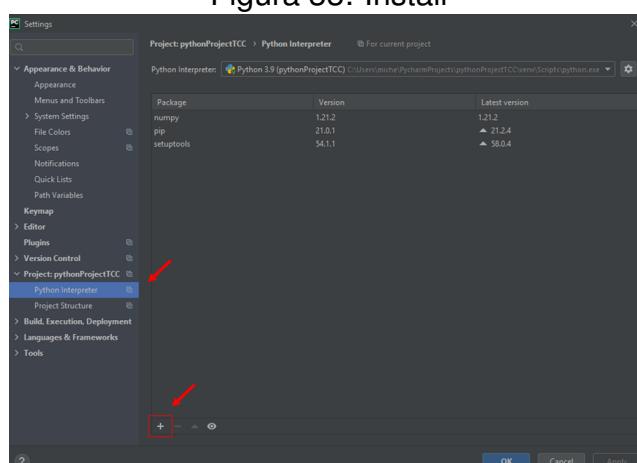
Figura 32: Instalando biblioteca NumPy



Fonte: Autoria própria (2021).

Após abrir a aba, clicar no projeto que deseja realizar a instalação, depois clicar em **Python Interpreter** e por em **Install**.

Figura 33: Install



Fonte: Autoria própria (2021).

Em seguida digitar o nome da biblioteca que deseja ser instalada, nesse caso é a biblioteca **NumPy** e depois clicar em **Install Package**, finalizada a instalar, a biblioteca estará pronta para ser utilizada.

Utilizando a biblioteca **NumPy** para calcular o determinante de uma matriz de ordem $n \times n$. Para começar o algoritmo a biblioteca que se deseja trabalhar deve ser importada ao utilizar o comando **import...as...**, dessa forma estamos considerando o caminho mais completo para o arquivo Python que apresente o módulo que se deseja importar.

```
import numpy as np
```

Após importar a biblioteca será criada a lista que irá conter a matriz B , solicitado ao usuário a ordem da mesma, percorrida as linhas e as colunas, solicitado ao usuário o valor dos elementos que a compõem e exibida a matriz.

```
n = int(input('Informe o ordem da matriz: '))
B = []
for i in range(n):
    linha = []
    for j in range(n):
        E = int(input('Elemento da Matriz B: linha {} e coluna {} é: '
                      .format(i + 1, j + 1)))
        linha.append(E)
    B.append(linha)
def exibir_matriz(B):
    for linha in B:
        print(linha)
    print('\nMatriz B\n')
exibir_matriz(B)
```

De acordo com a biblioteca NumPy, o comando que irá calcular o determinante da matriz B é:

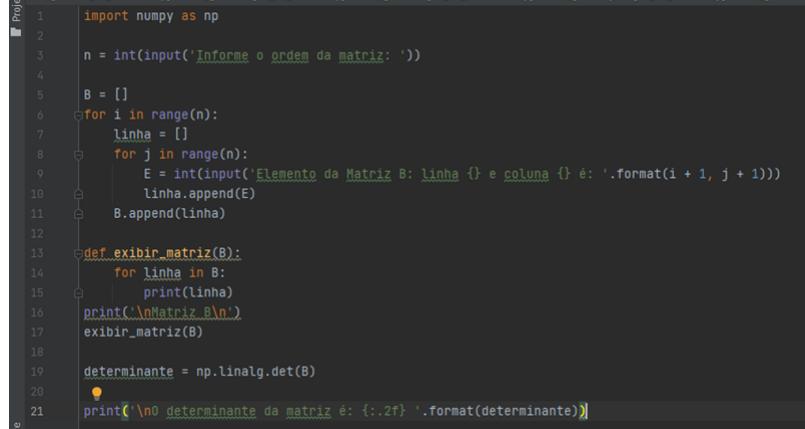
```
determinante = np.linalg.det(B)
```

Por fim, será mostrado o resultado do determinante.

```
print('\nO determinante da matriz é: {:.2f} '.format(determinante))
```

A figura abaixo irá mostrar o código completo.

Figura 34: Determinante Numpy

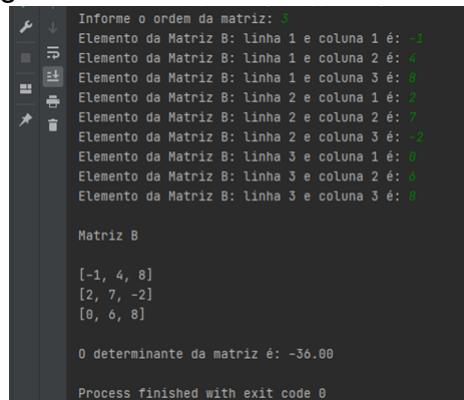


```
Proj
1 import numpy as np
2
3 n = int(input('Informe o ordem da matriz: '))
4
5 B = []
6 for i in range(n):
7     linha = []
8     for j in range(n):
9         E = int(input('Elemento da Matriz B: linha {} e coluna {} é: '.format(i + 1, j + 1)))
10        linha.append(E)
11    B.append(linha)
12
13 def exibir_matriz(B):
14     for linha in B:
15         print(linha)
16     print('\nMatriz B\n')
17 exibir_matriz(B)
18
19 determinante = np.linalg.det(B)
20
21 print('O determinante da matriz é: {:.2f} '.format(determinante))
```

Fonte: Autoria própria (2021).

Executando o algoritmo para uma matriz de ordem 3×3 .

Figura 35: Run 'Determinante Numpy'



```
Informe o ordem da matriz: 3
Elemento da Matriz B: linha 1 e coluna 1 é: 1
Elemento da Matriz B: linha 1 e coluna 2 é: 4
Elemento da Matriz B: linha 1 e coluna 3 é: 8
Elemento da Matriz B: linha 2 e coluna 1 é: 2
Elemento da Matriz B: linha 2 e coluna 2 é: 7
Elemento da Matriz B: linha 2 e coluna 3 é: 3
Elemento da Matriz B: linha 3 e coluna 1 é: 0
Elemento da Matriz B: linha 3 e coluna 2 é: 6
Elemento da Matriz B: linha 3 e coluna 3 é: 0

Matriz B

[-1, 4, 8]
[2, 7, -2]
[0, 6, 8]

O determinante da matriz é: -36.00
Process finished with exit code 0
```

Fonte: Autoria própria (2021).

Dessa forma é possível calcular o determinante de qualquer matriz $n \times n$.

7 MATRIZES SIMÉTRICAS E ANTI-SIMÉTRICAS

Uma matriz C de ordem $n \times n$ é considerada simétrica quando a transposta (C^T) é igual a própria matriz C . Por exemplo, considere a matriz dada por:

$$C = \begin{bmatrix} 4 & 9 & -2 \\ 9 & -1 & 3 \\ -2 & 3 & 1 \end{bmatrix}.$$

A matriz C_3 é considerada simétrica, pois $C^T = C$. Têm-se que os elementos que pertencem a diagonal principal ($i = j$) permanecem os mesmos, dessa forma podemos dizer que a diagonal principal em uma matriz simétrica pode ser observada como um espelho, pois os elementos que estão acima da diagonal principal são iguais aos elementos que estão abaixo da diagonal principal.

Uma matriz D de ordem $n \times n$ é considerada anti-simétrica quando a transposta (D^T) é igual a $-D$. Por exemplo, considere a seguinte matriz.

$$D = \begin{bmatrix} 0 & 6 & -1 \\ -6 & 0 & -7 \\ 1 & 7 & 0 \end{bmatrix}.$$

A matriz D é considerada Anti-simétricas, pois:

$$D^T = \begin{bmatrix} 0 & -6 & 1 \\ 6 & 0 & 7 \\ -1 & -7 & 0 \end{bmatrix} = -D.$$

Note que os elementos da diagonal principal em uma matriz anti-simétrica são iguais a zero.

7.1 Identificando em Python Matrizes Simétricas e Anti-simétricas

Para identificar se uma matriz é simétrica ou anti-simétrica será construído um algoritmo que utilize como base o código da matriz transposta desenvolvida anteriormente na seção 5.1.

```
r = int(input('Digite a ordem da matriz: '))

print('A matriz M irá possuir ordem {} x {}'.format(r, r))
M = []
for i in range(r):
    linha = []
    for j in range(r):
        e = int(input('Elemento da Matriz M: linha {} e coluna {} é: '
                      ''.format(i + 1, j + 1)))
        linha.append(e)
    M.append(linha)
```

```

def exibir_matriz(M):
    for linha in M:
        print(linha)
print('Matriz M')
exibir_matriz(M)
print('\n')

```

Dessa forma, foi solicitado ao usuário a ordem da matriz, criada a lista geral que irá conte-la, criada as linhas e as colunas, solicitado ao usuário os elementos e por fim, será exibida a matriz. Agora será criada a matriz transposta de M , para que possa ser verificada as condições de matriz simétrica ($M = M^T$) e matriz anti-simétrica ($M^T = -M$).

```

T = []
for i in range(r):
    linha =[0]*r
    T.append(linha)
    for j in range(r):
        T[i][j] = M[j][i]

```

A variável T irá conter a matriz transposta, em seguida será percorrida as linhas e as colunas, note que para as linhas foi criada uma lista, a qual irá possuir a quantidade de elementos que foi solicitada ao usuário no inicio do algoritmo, lembre-se que a matriz T é composta por listas uma sobre a outra, por isso não é necessário criar uma lista para as colunas. Para obter a matriz transposta temos que trocar as linhas pelas colunas, logo:

$$T[i][j] = M[j][i].$$

A matriz T está recebendo a matriz M , porém com a posição das linhas e colunas alteradas. Agora será verificado se a matriz M pode ser simétrica, de acordo com a seguinte condição:

```

if T == M:
    print('A Matriz M é SIMÉTRICA, pois a matriz M é igual a sua '
          'transposta (T).')
    exibir_matriz(T)

```

Com a condição **if** será verificada se a matriz M é igual a matriz T , caso a condição seja verdadeira será exibida uma mensagem dizendo que a matriz M é simétrica e em seguida será exibida a matriz transposta T . Agora, caso a matriz não seja classificada como matriz simétrica é necessário verificar se ela é anti-simétrica, para isso deve-se ter em mente a definição de matriz anti-simétrica.

```
N = []
for i in range(r):
    linha = [0]*r
    N.append(linha)
    for j in range(r):
        N[i][j] = -M[i][j]
```

De acordo com a definição de matriz anti-simétrica temos que a transposta da matriz M deve ser igual a $-M$. Dessa forma, a matriz anti-simétrica foi construída para comparar com a matriz transposta T .

```
if T == N:
    print('A matriz é ANTI-SIMETRICA, pois a transposta de M (T) é igual '
          'a matriz m (-M) \n')
    exibir_matriz(N)
if T != M and T != N:
    print('A matriz não é simétrica e nem anti-simétrica')
```

Com a condição **if** será verificada se a matriz N é igual a matriz T , caso a condição seja verdadeira será exibida uma mensagem dizendo que a matriz N é anti-simétrica e em seguida será exibida a matriz N . Caso a condição seja falsa irá aparecer uma mensagem dizendo que a matriz não é simétrica e nem anti-simétrica, finalizando assim o código.

A figura abaixo irá mostrar o algoritmo completo com algumas instruções a mais apenas para manter a estética do código ao ser executado.

Figura 36: Algoritmo verificação de matrizes

```

Proj
1   r = int(input('Digite a ordem da matriz: '))
2
3   print('A matriz M irá possuir ordem {} x {}'.format(r, r))
4   M = []
5   for i in range(r):
6       linha = []
7       for j in range(r):
8           e = int(input('Elemento da Matriz G: linha {} e coluna {} é: '.format(i + 1, j + 1)))
9           linha.append(e)
10      M.append(linha)
11  def exibir_matriz(M):
12      for linha in M:
13          print(linha)
14  print('Matriz M')
15  exibir_matriz(M)
16  print('\n')
17  T = []
18  for i in range(r):
19      linha =[0]*r
20      T.append(linha)
21      for j in range(r):
22          T[i][j] = M[j][i]
23  if T == M:
24      print('A Matriz M é SIMÉTRICA, pois a matriz M é igual a sua transposta (T).')
25  exibir_matriz(T)
26
27  N = []
28  for i in range(r):
29      linha = [0]*r
30      N.append(linha)
31      for j in range(r):
32          N[i][j] = -M[i][j]
33  if T == N:
34      print('A matriz é ANTI-SIMETRICA, pois a transposta de M (T) é igual a matriz m (-M) \n')
35  exibir_matriz(N)
36  if T != M and T != N:
37      print('A matriz não é simétrica e nem anti-simétrica')

```

Fonte: Autoria própria (2021).

Executando o algoritmo para uma matriz simétrica de ordem 3×3 .

Figura 37: Run 'matriz simétrica'

```

Digite a ordem da matriz: 3
A matriz M irá possuir ordem 3 x 3.
Elemento da Matriz G: linha 1 e coluna 1 é: -2
Elemento da Matriz G: linha 1 e coluna 2 é: 7
Elemento da Matriz G: linha 1 e coluna 3 é: 1
Elemento da Matriz G: linha 2 e coluna 1 é: 7
Elemento da Matriz G: linha 2 e coluna 2 é: 0
Elemento da Matriz G: linha 2 e coluna 3 é: 6
Elemento da Matriz G: linha 3 e coluna 1 é: 1
Elemento da Matriz G: linha 3 e coluna 2 é: 6
Elemento da Matriz G: linha 3 e coluna 3 é: 5
Matriz M
[-2, 7, 1]
[7, 0, 6]
[1, 6, 5]

A Matriz M é SIMÉTRICA, pois a matriz M é igual a sua transposta (T).
[-2, 7, 1]
[7, 0, 6]
[1, 6, 5]

Process finished with exit code 0

```

Fonte: Autoria própria (2021).

Observe que é exibida uma mensagem justificando o porquê da matriz ser simétrica.

trica e em seguida a matriz transposta de \mathbf{M} é exibida para que possam ser comparadas as matrizes.

Executando o algoritmo para uma matriz anti-simétrica de ordem 4×4 .

Figura 38: Run 'matriz anti-simétrica'

```
Digite a ordem da matriz: 4
A matriz M irá possuir ordem 4 x 4.
Elemento da Matriz G: linha 1 e coluna 1 é: 0
Elemento da Matriz G: linha 1 e coluna 2 é: 7
Elemento da Matriz G: linha 1 e coluna 3 é: -3
Elemento da Matriz G: linha 1 e coluna 4 é: 3
Elemento da Matriz G: linha 2 e coluna 1 é: -7
Elemento da Matriz G: linha 2 e coluna 2 é: 0
Elemento da Matriz G: linha 2 e coluna 3 é: 9
Elemento da Matriz G: linha 2 e coluna 4 é: 1
Elemento da Matriz G: linha 3 e coluna 1 é: 3
Elemento da Matriz G: linha 3 e coluna 2 é: -9
Elemento da Matriz G: linha 3 e coluna 3 é: 0
Elemento da Matriz G: linha 3 e coluna 4 é: 5
Elemento da Matriz G: linha 4 e coluna 1 é: -3
Elemento da Matriz G: linha 4 e coluna 2 é: -1
Elemento da Matriz G: linha 4 e coluna 3 é: -5
Elemento da Matriz G: linha 4 e coluna 4 é: 0
Matriz M
[0, 7, -3, 3]
[-7, 0, 9, 1]
[3, -9, 0, 5]
[-3, -1, -5, 0]

A matriz é ANTI-SIMETRICA, pois a transposta de M (T) é igual a matriz m (-T)

[0, -7, 1, -3]
[7, 0, -9, -1]
[-1, 9, 0, -5]
[3, 1, 5, 0]

Process finished with exit code 0
```

Fonte: Autoria própria (2021).

Observe que é exibida uma mensagem justificando o porquê da matriz ser anti-simétrica e em seguida a matriz $-M$ é exibida para que possam ser comparadas as matrizes.

Executando o algoritmo para uma matriz 3×3 que não se classifique em simétrica e anti-simétrica.

Figura 39: Run 'verificação de matrizes'

```
Digite a ordem da matriz: 3
A matriz M irá possuir ordem 3 x 3.
Elemento da Matriz G: linha 1 e coluna 1 é: -1
Elemento da Matriz G: linha 1 e coluna 2 é: 6
Elemento da Matriz G: linha 1 e coluna 3 é: 2
Elemento da Matriz G: linha 2 e coluna 1 é: 9
Elemento da Matriz G: linha 2 e coluna 2 é: -4
Elemento da Matriz G: linha 2 e coluna 3 é: 7
Elemento da Matriz G: linha 3 e coluna 1 é: 0
Elemento da Matriz G: linha 3 e coluna 2 é: 1
Elemento da Matriz G: linha 3 e coluna 3 é: 4
Matriz M
[-1, 6, 2]
[9, -4, 7]
[0, 1, 4]

A matriz não é simétrica e nem anti-simétrica

Process finished with exit code 0
```

Fonte: Autoria própria (2021).

Note que para matrizes que possuam uma ordem pequena o algoritmo se torna desnecessário, porém para matrizes que possuam uma ordem grande o algoritmo se torna eficiente.

8 MATRIZ INVERSA

De acordo com Steinbruch (1987), dada uma matriz quadrada A , de ordem n , se existir uma matriz quadrada B , de mesma ordem, que satisfaça à seguinte condição:

$$AB = BA = I$$

B é inversa de A e se representa por A^{-1} :

$$AA^{-1} = A^{-1}A = I$$

Dada a definição, considere a matriz E de ordem 2×2 , dada por:

$$E = \begin{bmatrix} 2 & -6 \\ 7 & 3 \end{bmatrix}. \quad (2)$$

A matriz E_2 será multiplicada por outra matriz (F_2) de tal forma que a multiplicação resulte em uma matriz identidade (I_2).

$$EF = FE = I.$$

A matriz F também é chamada de inversa de E e pode ser representada por E^{-1} , isto é:

$$EE^{-1} = E^{-1}E = I.$$

Dada a matriz E , sua matriz inversa só é possível ser encontrada se o determinante for diferente de zero. Para matrizes que possuem o determinante igual a zero se da o nome de *matriz singular*.

$$\det E = \begin{vmatrix} 2 & -6 \\ 7 & 3 \end{vmatrix} = 2 \cdot 3 - (-6 \cdot 7) = 48.$$

Logo a matriz E possui inversa:

$$\begin{bmatrix} 2 & -6 \\ 7 & 3 \end{bmatrix} \cdot \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

A inversa será calculada através da inversão de matrizes por meio de finitas operações elementares. Para calcular a matriz inversa coloca-se ao lado da matriz E a matriz identidade I , separada por um traço na vertical e por meio das operações elementares transforma-se a matriz E na matriz identidade, as mesmas operações elementares realizados na matriz E também serão realizados na matriz I .

$$\left[\begin{array}{cc|cc} 2 & -6 & 1 & 0 \\ 7 & 3 & 0 & 1 \end{array} \right]$$

Realizando as operações elementares.

1. Substituindo a 1º linha de ambas as matrizes pela seguinte operação elementar:

$$L_1 = \frac{1}{2} \cdot L_1.$$

$$\left[\begin{array}{cc|cc} 1 & -3 & \frac{1}{2} & 0 \\ 7 & 3 & 0 & 1 \end{array} \right].$$

2. Substituindo a 2º linha de ambas as matrizes pela seguinte operação elementar:

$$L_2 = L_1 \cdot (-7) + L_2.$$

$$\left[\begin{array}{cc|cc} 1 & -3 & \frac{1}{2} & 0 \\ 0 & 24 & -\frac{7}{2} & 1 \end{array} \right].$$

3. Substituindo a 2º linha de ambas as matrizes pela seguinte operação elementar:

$$L_2 = \frac{1}{24} \cdot L_2.$$

$$\left[\begin{array}{cc|cc} 1 & -3 & \frac{1}{2} & 0 \\ 0 & 1 & -\frac{7}{48} & \frac{1}{24} \end{array} \right].$$

4. Substituindo a 1º linha de ambas as matrizes pela seguinte operação elementar:

$$L_1 = 3 \cdot L_2 + L_1.$$

$$\left[\begin{array}{cc|cc} 1 & 0 & \frac{1}{16} & \frac{1}{8} \\ 0 & 1 & \frac{-7}{48} & \frac{1}{24} \end{array} \right].$$

Dessa forma foi encontrada a matriz inversa de E .

$$E^{-1} = \left[\begin{array}{cc} \frac{1}{16} & \frac{1}{8} \\ \frac{-7}{48} & \frac{1}{24} \end{array} \right].$$

PROPRIEDADES DA MATRIZ INVERSA

- I. Se o determinante da matriz A for diferente de zero, a matriz irá possuir inversa, a qual é única.
- II. A matriz inversa de A^{-1} é A . Seja a matriz A não-singular, sua inversa A^{-1} também será não-singular.
- III. A matriz identidade I possui determinante igual a 1, ou seja, ela admite inversa. A inversa da matriz identidade é ela mesma, $I = I^{-1}$.
- IV. Sendo A não-singular, sua matriz transposta também será. A matriz inversa de A^T é $(A^{-1})^T$.
- V. Considerando duas matrizes A e B não-singulares e de mesma ordem, o produto entre elas também resultará em uma matriz não-singular. A inversa do produto AB é a a matriz $B^{-1}A^{-1}$.

8.1 Matriz Inversa em Python

Deseja-se calcular a inversa de matriz B , de tal forma que seja solicitado ao usuário os elementos e a ordem da matriz. Começando a escrita do código, temos:

```
import numpy as np
```

Assim, será importada a biblioteca **numpy** para que o algoritmo não possua muitas linhas, implementando assim o código e em seguida será construída a matriz B .

```
n = int(input('Informe o ordem da matriz: '))
B = []
for i in range(n):
    linha = []
```

```

for j in range(n):
    e = int(input('Elemento da Matriz B: linha {} e coluna {} é: '
                  .format(i + 1, j + 1)))
    linha.append(e)
B.append(linha)

def exibir_matriz(B):
    for linha in B:
        print(linha)
print('\nMatriz B\n')
exibir_matriz(B)

```

Dessa forma a matriz B foi construída, agora será encontrado o determinante para verificar se a matriz é singular ou não.

```

det = np.linalg.det(B)
print('\nO determinante da matriz é: {:.2f}'.format(det))

```

Após o determinante ser encontrado, será verificado se o mesmo é diferente de zero.

```

if det != 0:
    matriz_inv = np.linalg.inv(B)
    print('Matriz inversa')
    print(matriz_inv)
else:
    print('A matriz não possui inversa!')

```

Caso a matriz possua determinante diferente de zero será calculada a inversa da matriz B , do contrário será exibida uma mensagem de texto dizendo que a matriz não possui inversa. Os valores retornados por esse algoritmo são decimais.

A figura abaixo mostra o algoritmo completo.

Figura 40: Algoritmo inversa de matrizes

```

1 import numpy as np
2 n = int(input('Informe o ordem da matriz: '))
3 B = []
4 for i in range(n):
5     linha = []
6     for j in range(n):
7         E = int(input('Elemento da Matriz B: linha {} e coluna {} é: '.format(i + 1, j + 1)))
8         linha.append(E)
9     B.append(linha)
10
11 def exibir_matriz(B):
12     for linha in B:
13         print(linha)
14     print('\nMatriz B\n')
15 exibir_matriz(B)
16
17 det = np.linalg.det(B)
18
19 print('\nO determinante da matriz é: {:.2f} '.format(det))
20
21 if det != 0:
22     matriz_inv = np.linalg.inv(B)
23     print('Matriz inversa')
24     print(matriz_inv)
25 else:
26     print('A matriz não possui inversa!')

```

Fonte: Autoria própria (2021).

Encontrando a inversa da matriz B_3 .

Figura 41: Run 'inversa de matrizes'

```

Informe o ordem da matriz: 3
Elemento da Matriz B: linha 1 e coluna 1 é: -1
Elemento da Matriz B: linha 1 e coluna 2 é: 0
Elemento da Matriz B: linha 1 e coluna 3 é: 3
Elemento da Matriz B: linha 2 e coluna 1 é: 9
Elemento da Matriz B: linha 2 e coluna 2 é: 4
Elemento da Matriz B: linha 2 e coluna 3 é: 6
Elemento da Matriz B: linha 3 e coluna 1 é: 1
Elemento da Matriz B: linha 3 e coluna 2 é: -2
Elemento da Matriz B: linha 3 e coluna 3 é: 5

Matriz B

[-1, 0, 3]
[9, 4, 6]
[1, -2, 5]

O determinante da matriz é: -98.00
Matriz inversa
[[-0.32653061  0.06122449  0.12244898]
 [ 0.39795918  0.08163265 -0.33673469]
 [ 0.2244898   0.02040816  0.04081633]]

```

Fonte: Autoria própria (2021).

9 SISTEMA DE EQUAÇÕES LINEARES

Uma equação linear pode ser representada da seguinte forma:

$$a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n = b_n.$$

onde:

 $a_1, a_2, a_3, \dots, a_n$ são os coeficientes das variáveis.

$x_1, x_2, x_3, \dots, x_n$ são as variáveis.

b_n é o termo independente.

Um sistema de equações lineares com m equações e n variáveis pode ser representado da seguinte forma:

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \dots + a_{3n}x_n = b_3 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \ddots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \dots + a_{mn}x_n = b_m \end{array} \right.$$

onde:

i irá variar de 1, 2, 3, ..., m.

j irá variar de 1, 2, 3, ..., n.

x_j são as variáveis.

a_{ij} são os coeficientes das variáveis.

b_i são os termos independentes.

Porém, como transformar esse sistema de equações em notação matricial?

Note que no sistema de equações temos uma matriz A para os coeficientes das variáveis, a qual possui ordem $m \times n$, assim temos A_{mn} . Para as variáveis temos uma matriz de n linhas e uma coluna, ou seja, X_{n1} e para os termos independentes temos m linhas e uma coluna, dessa forma B_{m1} . Com isso, temos a seguinte notação matricial para um sistema de equações lineares (SANTOS, 2020).

$$A_{mn} \times X_{n1} = B_{m1}$$

Que pode ser representado por matrizes da seguinte forma:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{31} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{n1} \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ \vdots \\ b_{m1} \end{bmatrix}$$

DEPENDÊNCIA LINEAR

De acordo com Boldrini (1984) um conjunto de vetores é dito **Linearmente Independente (LI)** se nenhum dos seus vetores puder ser escrito como combinação linear

dos demais, dessa forma temos:

$$x_1v_1 + x_2v_2 + x_3v_3 + \dots + x_nv_n = 0.$$

A igualdade acima só pode ser verificada se:

$$x_1, x_2, x_3, \dots, x_n = 0.$$

Para um vetor **Linearmente Dependente (LD)** existem escalares $x_1, x_2, x_3, \dots, x_n$ não todos nulos tais que:

$$x_1v_1 + x_2v_2 + x_3v_3 + \dots + x_nv_n = 0$$

onde, pelo menos um $x_i \neq 0$.

POSTO DE UMA MATRIZ

Considerando uma matriz, quadrada ou não, o seu posto será dado pelo número máximo de vetores linhas ou colunas que são linearmente independentes. Caso a matriz apresente linhas ou colunas linearmente dependentes, as mesmas irão zerar ao realizar o escalonamento da matriz e o posto também poderá ser dado pela quantidade de linhas ou colunas não nulas.

EXEMPLO

Considere a matriz A de ordem 3×3 .

$$A = \begin{bmatrix} 2 & -5 & 7 \\ 8 & 1 & 3 \\ -4 & -11 & 11 \end{bmatrix}$$

Observe que na matriz A a 3º linha é combinação linear da 1º linha multiplicada por 2 e em seguida subtraída da 2º linha, logo, a matriz A irá possuir posto igual a 2 e caso essa matriz venha ser escalonada utilizando o método da eliminação de Gauss, a 3º linha irá zerar. Mas qual a finalidade de descobrir o posto de uma matriz?

A matriz dos coeficientes A pode ser classificada de acordo com o valor do determinante de uma matriz quadrada e a forma da matriz A_{mn} , ou seja, a quantidade de linhas, colunas e seu posto (SANTOS, 2020).

Um sistema é dito **Sobredeterminado** quando:

$$m \geq n \text{ e } P(A) = n$$

Dessa forma haverá mais equações do que incógnitas, fazendo com que o sistema possua uma única solução ou não apresente nenhuma solução.

Um sistema é dito **Subdeterminado** quando:

$$m < n \text{ e } P(A) = m$$

Dessa forma o sistema irá apresentar mais incógnitas do que equações, fazendo com que o sistema não possua solução ou apresente infinitas soluções.

CLASSIFICANDO UMA MATRIZ DE ACORDO COM O VALOR DO DETERMINANTE

Dada uma matriz quadrada para os coeficientes das variáveis, é possível verificar se o sistema de equações lineares possui solução através do cálculo do determinante. O sistema de equações lineares pode ser classificado das seguintes formas:

1. **Sistema possível Determinado:** $\det A \neq 0$.

Um sistema é classificado como determinado se possuir uma solução única que satisfaça todas as m equações.

2. **Sistema possível Indeterminado ou Incompatível:** $\det A = 0$.

Um sistema é classificado como indeterminado quando o mesmo possui infinitas soluções e classificado como incompatível quando não é admita nenhuma solução.

9.1 Solução de Sistemas Lineares

Dada a seguinte notação matricial:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{31} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{n1} \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ \vdots \\ b_{m1} \end{bmatrix}.$$

Será utilizado o método da **Eliminação de Gauss** para obter uma matriz triangular superior e equivalente a A , de modo que as mesmas operações elementares efetuadas para a matriz dos coeficientes das variáveis também sejam efetuadas para o vetor

constante B . Para realizar o escalonamento será colocada a matriz A ao lado do vetor constante B separadas por um traço na vertical.

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_{11} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_{21} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} & b_{31} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} & b_{m1} \end{array} \right]$$

Após realizar as operações elementares, o sistema de equações irá possuir a seguinte representação.

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2 \\ a_{33}x_3 + \dots + a_{3n}x_n = b_3 \\ \vdots \quad \vdots \\ a_{nn}x_n = b_n \end{array} \right.$$

De forma que a partir de substituições inversas seja possível encontrar os valores para as variáveis x_i .

EXEMPLO

Considere o seguinte sistema de equações:

$$\left\{ \begin{array}{l} x_1 + 2x_2 - 3x_3 = 8 \\ 3x_1 - 2x_2 + 5x_3 = 4 \\ 2x_1 + 6x_3 = 1 \end{array} \right.$$

Representando o sistema acima em notação matricial, temos:

$$\left[\begin{array}{ccc} 1 & 2 & -3 \\ 3 & -2 & 5 \\ 2 & 0 & 6 \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right] = \left[\begin{array}{c} 8 \\ 4 \\ 1 \end{array} \right].$$

Escalonando as matrizes dos coeficientes das variáveis (A) e do vetor constante (B).

$$\left[\begin{array}{ccc|c} 1 & 2 & -3 & 8 \\ 3 & -2 & 5 & 4 \\ 2 & 0 & 6 & 1 \end{array} \right].$$

1. Substituindo a 2º linha em ambas as matrizes pela seguinte operação elementar:

$$L_2 = (-3)L_1 + L_2,$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -3 & 8 \\ 0 & -8 & 14 & -20 \\ 2 & 0 & 6 & 1 \end{array} \right].$$

2. Substituindo a 3º linha em ambas as matrizes pela seguinte operação elementar:

$$L_3 = (-2)L_1 + L_3.$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -3 & 8 \\ 0 & -8 & 14 & -20 \\ 0 & -4 & 12 & -15 \end{array} \right].$$

3. Substituindo a 3º linha em ambas as matrizes pela seguinte operação elementar:

$$L_3 = \frac{-1}{2} \cdot L_2 + L_3.$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -3 & 8 \\ 0 & -8 & 14 & -20 \\ 0 & 0 & 5 & -5 \end{array} \right].$$

Retornando para a notação matricial, temos:

$$\left[\begin{array}{ccc} 1 & 2 & -3 \\ 0 & -8 & 14 \\ 0 & 0 & 5 \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right] = \left[\begin{array}{c} 8 \\ -20 \\ -5 \end{array} \right].$$

logo o sistema de equações dado inicialmente se reduz a:

$$\begin{cases} x_1 + 2x_2 - 3x_3 = 8 \\ -8x_2 + 14x_3 = -20 \\ 5x_3 = -5 \end{cases}.$$

Encontrando as soluções para as variáveis a partir de substituições na ordem inversa para sistemas.

$$5x_3 = -5 \Rightarrow x_3 = -1$$

$$-8x_2 + 14 \cdot (-1) = -20 \Rightarrow x_2 = \frac{17}{4}$$

$$x_1 + 2 \cdot \frac{17}{4} - 3 \cdot (-1) = 8 \Rightarrow x_1 = \frac{-9}{2}.$$

9.2 Solução de Sistemas Lineares em Python

Dada a seguinte notação matricial:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{n1} \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ \vdots \\ b_{m1} \end{bmatrix}.$$

Após realizar a triangulação da matriz A , o sistema de equações será:

$$\left\{ \begin{array}{l} \bar{a}_{11}x_1 + \bar{a}_{12}x_2 + \bar{a}_{13}x_3 + \dots + \bar{a}_{1n}x_n = \bar{b}_1 \\ \bar{a}_{22}x_2 + \bar{a}_{23}x_3 + \dots + \bar{a}_{2n}x_n = \bar{b}_2 \\ \bar{a}_{33}x_3 + \dots + \bar{a}_{3n}x_n = \bar{b}_3 \\ \vdots \\ \bar{a}_{mn}x_n = \bar{b}_m \end{array} \right..$$

Encontrando a solução para x_n , temos:

$$x_n = \frac{\bar{b}_m}{\bar{a}_{mn}}.$$

Encontrando a solução para x_{n-1} , temos:

$$x_{n-1} = \frac{\bar{b}_{n-1} - \bar{a}_{n-1,n}x_n}{\bar{a}_{n-1,n-1}}.$$

Esse processo irá se repetir até encontrar a solução para x_1 , dessa forma temos:

$$x_1 = \frac{\bar{b}_1 - \bar{a}_{12}x_2 - \bar{a}_{13}x_3 - \bar{a}_{14}x_4 - \dots - \bar{a}_{1n}x_n}{\bar{a}_{11}}.$$

De forma geral encontramos a seguinte igualdade:

$$x_i = \frac{\bar{b}_i - \sum_{j=i+1}^n \bar{a}_{ij}x_j}{\bar{a}_{ii}}. \quad (3)$$

onde, $i = n, n-1, \dots, 1$

A equação 3 será utilizada para encontrar a solução a partir de substituições inversas para a ordem da matriz, contudo essa equação só pode ser utilizada para sistemas de equações já escalonados.

9.2.1 Algoritmo de substituições em Python

Para esse tópico foi considerada a lógica em programação abordada por Santos (2020). Considerando que o algoritmo será utilizado para encontrar uma solução, caso exista, será digitado no próprio código os elementos das matrizes A e B . Pois, caso a matriz dos coeficiente e vetor constante apresente por exemplo 7, o usuário deverá inserir os 49 elementos da matriz A e os 7 elementos que compõem a matri B de forma repetitiva, sendo que ainda existe a possibilidade de inserir os valores erroneamente depois que o algoritmo for executado.

Inicialmente será construída duas matrizes nulas, uma para os coeficientes e outra para as constantes independentes.

```
A = [[0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0]]
```

```
B = [0, 0, 0]
```

Dessa forma foram inseridas as matrizes compostas de elementos nulos, para que o usuário após finalizar o código possa retornar e inserir os valores que desejar ou modificar a ordem $m \times n$, desde que a matriz inserida seja de ordem quadrada. Agora será dado início a solução a partir de substituições retroativas de acordo com a equação a seguir:

$$x_i = \frac{\bar{b}_i - \sum_{j=i+1}^n \bar{a}_{ij}x_j}{\bar{a}_{ii}} \quad (4)$$

onde, $i = n, n-1, \dots, 1$.

Será definida uma função para realizar as substituições e encontrar o vetor das variáveis que satisfação as m equações.

```
def solucionando (A, B):
    n = len(A)
    x = n * [0]
```

Dessa forma, n irá receber o tamanho da matriz A e x é o vetor solução que irá possuir a quantidade de colunas que a matriz A apresenta.

```
for i in range(n-1, -1, -1):
    S = 0
```

Lembre-se que em Python os índices começam a contar do zero, considerando que a solução para os sistemas será dada a partir da enésima linha da matriz até a primeira, temos que o intervalo para as linhas i será dado por $n - 1$, que em Python é a última linha da matriz, até a linha zero e para que essa linha entre no intervalo da função **range()** deve-se colocar o intervalo até -1 , como se trata de substituições decrescentes o passo será -1 .

Como será realizado o somatório no *for...in* para as colunas j , será definido ainda no *for...in* para as linhas i o somatório inicializando em zero. Após definir o somatório inicializando em zero, será construído o intervalo para as colunas.

```
for j in range(i+1, n):
    S = S + A[i][j] * x[j]
    x[i] = (B[i] - S)/A[i][i]

return x

x = solucionando(A, B)
print('velor solução {}'.format(x))
```

O intervalo para as colunas j será de $i + 1$ até n , e em seguida será desenvolvido o somatório presente na equação 3,o qual no algoritmo será dado por:

$$S = S + A[i][j] * x[j].$$

Concluído o somatório, os valores para as variáveis serão calculados, em seguida retornados e por fim exibidos. A figura abaixo mostra o algoritmo completo.

Figura 42: Algoritmo solucionando

```

Proj
1   A = [[0, 0, 0, 0],
2       [0, 0, 0, 0],
3       [0, 0, 0, 0],
4       [0, 0, 0, 0]]
5
6   B = [0, 0, 0, 0]
7
8   def solucionando(A, B):
9       n = len(A)
10      x = n * [0]
11      for i in range(n-1, -1, -1):
12          S = 0
13          for j in range(i+1, n):
14              S = S + A[i][j] * x[j]
15          x[i] = (B[i] - S)/A[i][i]
16
17      return x
18
19 x = solucionando(A, B)
20 print('vetor solução {}'.format(x))

```

Fonte: Autoria própria (2021).

Encontrando o vetor solução para o seguinte sistema de equações já escalonado.

$$\begin{cases} x_1 + 7x_3 + 2x_4 = 41 \\ x_2 + 3x_3 + x_4 = 17 \\ -5x_3 + 8x_3 = 4 \\ 3x_4 = 9 \end{cases} .$$

Primeiro, deve-se substituir os elementos nulos das matrizes A e B no algoritmo.

Figura 43: Algoritmo solucionando sistema

```

Proj
1   A = [[1, 0, 7, 2],
2       [0, 2, 3, 1],
3       [0, 0, -5, 8],
4       [0, 0, 0, 3]]
5
6   B = [41, 17, 4, 9]
7
8   def solucionando(A, B):
9       n = len(A)
10      x = n * [0]
11      for i in range(n-1, -1, -1):
12          S = 0
13          for j in range(i+1, n):
14              S = S + A[i][j] * x[j]
15          x[i] = (B[i] - S)/A[i][i]
16
17      return x
18
19 x = solucionando(A, B)
20 print('vetor solução {}'.format(x))

```

Fonte: Autoria própria (2021).

Em seguida basta executar o algoritmo para que seja exibido apenas o vetor solução.

Figura 44: Run 'Algoritmo solucionando sistema'

```
valor solução [7.0, 1.0, 4.0, 3.0]
Process finished with exit code 0
```

Fonte: Autoria própria (2021).

9.2.2 Algoritmo eliminação de Gauss em Python

Considerando a lógica em programação abordada por Santos (2020), o algoritmo para o método de eliminação de Gauss será a continuação do método das substituições mostrado anteriormente.

```
A = [[0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0]]

B = [0, 0, 0, 0]

def solucionando (A, B):
    n = len(A)
    x = n * [0]
    for i in range(n-1, -1, -1):
        S = 0
        for j in range(i+1, n):
            S = S + A[i][j] * x[j]
        x[i] = (B[i] - S)/A[i][i]

    return x
```

Em seguida será criada uma função **def** para o método da eliminação de Gauss.

```
def Gauss (A, B):
    n = len(A)
    for k in range(0, n-1):
```

Para cada coluna será realizada uma etapa k , ou seja, será eliminado os elementos que estão abaixo do elemento principal a_{ij} , em que $i = j$.

```
for i in range(k+1, n):
    F = -A[i][k] / A[k][k]
```

O elemento da diagonal principal está indexado na posição kk , logo para começar na primeira linha a baixo da diagonal principal o intervalo da função **range()** começará em $k + 1$ e irá terminar na última linha que é n , para que o último valor do intervalo seja $n - 1$. Para os elementos que estão a baixo da diagonal principal será calculado por linha o fator que irá zera-los, em seguida será atualizada cada linha da matriz A percorrendo as colunas.

```
for j in range(k+1, n):
    A[i][j] = F * A[k][j] + A[i][j]
```

O intervalo para as colunas será de $k + 1$ até n , de modo que o último valor a entrar no intervalo da função **range()** seja $n - 1$. Após definir o intervalo para as colunas cada elemento da matriz será igual ao fator (F) que multiplica o elemento pivô que está na linha k e coluna j somado ao elemento da linha i e coluna j . Ao realizar a atualização dos elementos da matriz A , deve-se realizar as mesmas operações para o vetor B .

```
B[i] = F * B[k] + B[i]
A[i][k] = 0
```

Dessa forma, cada elemento do vetor B será igual ao fator (F) que multiplica o elemento pivô ,localizado na linha k , somado ao elemento atual da linha i . Atualizada a matriz A e o vetor das constantes B , será zerado os elementos que estão na linha i e coluna k . Após escalar a matriz A será calculado o determinante para verificar se a mesma possui solução.

```
det = 1
for i in range(n):
    det = det * A[i][i]
```

Inicialmente o determinante admitirá valor igual a 1, para que ao entrar no laço de repetição **for** possa ser efetuado a multiplicação dos elementos da diagonal principal.

Após isso, deverá ser verificado se o sistema possui solução única ou não possui solução.

```

x = substituicoes_somatoria(A, B)
return (x,det)

else:
    print('A matriz dos coeficientes é singular, ou seja, o '
          'determinante é igual a zero!')
    return ([] , det)

```

Caso o determinante seja diferente de zero será retornado o vetor solução e o valor do determinante, já se o determinante for igual a zero será exibida uma mensagem dizendo que a matriz dos coeficientes é singular e será retornado um vetor nulo para as variáveis e o valor do determinante será exibido.

```
(x, det) = Gauss(A, B)
```

O vetor solução e o determinantes serão retornados da função `Gauss(A, B)` e para que a matriz *A* e o vetor *B* sejam exibidos, foi criada uma função `def exibir_matriz()`.

```

def exibir_matriz(A):
    for i in A:
        print(i)
print('Matriz A escalonada')
exibir_matriz(A)
print('\n')
print('Vetor B equivalente')
exibir_matriz(B)

```

Dessa forma será exibida a matriz dos coeficientes escalonada e o vetor *B* equivalente, por fim será exibido o valor do determinante e o vetor solução.

```
print('O determinante da matriz é {:.2f} e velor solução {}'.format(det, x)
      )
```

A figura abaixo mostra o algoritmo completo.

Figura 45: Eliminação de Gauss

```

1  A = [[2, 4, -2],
2    [4, -1, 2],
3    [2, 1, -3]]
4
5  B = [1, 17, 12]
6
7  def substituicoes_somatoria(A, B):
8      n = len(A)
9      x = n * [0]
10     for i in range(n-1, -1, -1):
11         S = 0
12         for j in range(i+1, n):
13             S = S + A[i][j] * x[j]
14         x[i] = (B[i] - S)/A[i][i]
15     return x
16
17 def Gauss_(A, B):
18     n = len(A)
19     for k in range(0, n-1):
20         for i in range(k+1, n):
21             F = -A[i][k]/A[k][k]
22             for j in range(k+1, n):
23                 A[i][j] = F * A[k][j] + A[i][j]
24             B[i] = F * B[k] + B[i]
25             A[i][k] = 0
26     det = 1
27     for i in range(n):
28         det = det * A[i][i]
29     if det != 0:
30         x = substituicoes_somatoria(A, B)
31         return (x, det)
32     else:
33         print('A matriz dos coeficientes é singular, ou seja, o determinante é igual a zero!')
34         return ([], det)
35
36 (x, det) = Gauss_(A, B)
37
38 def exibir_matriz(A):
39     for i in A:
40         print(i)
41     print('Matriz A escalonada')
42     exibir_matriz(A)
43     print('\n')
44     print('Vetor B equivalente')
45     exibir_matriz(B)
46     print('\n')
47     print('A matriz possui determinante igual a {} e valor solução {}'.format(det, x))

```

Fonte: Autoria própria (2021).

Utilizando o algoritmo Eliminação de Gauss para solucionar o sistema a seguir:

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 1 \\ 4x_1 - x_2 + 2x_3 = 17 \\ 2x_1 + x_2 - 3x_3 = 12 \end{cases}.$$

Primeiro deve-se alterar os elementos nulos da matriz dos coeficientes A e do vetor constante B por:

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & -1 & 2 \\ 2 & 1 & -3 \end{bmatrix} \text{ e } B = \begin{bmatrix} 1 & 17 & 12 \end{bmatrix}.$$

Após inserir as matrizes o algoritmo será executado.

Figura 46: Run 'Eliminação de Gauss'

```

Matriz A escalonada
[2, 4, -2]
[0, -9.0, 6.0]
[0, 0, -3.0]

Vetor B equivalente
1
15.0
6.0

A matriz possui determinante igual a 54.0 e valor solução [4.5, -3.0, -2.0]

Process finished with exit code 0

```

Fonte: Autoria própria (2021).

O algoritmo do método da eliminação de Gauss apresenta vantagens, tais como:

- Realiza as operações elementares para o escalonamento da matriz A .
- Encontra a solução para o sistema de equações.
- Calcula o determinante da matriz A .
- Exibe a matriz A escalonada e o vetor B equivalente.

Porém, o algoritmo também apresenta desvantagens:

- Arredondamento de valores. Ao realizar uma operação elementar entre duas linhas é possível obter alguns valores com infinitas casas decimais, os quais ao realizar outra operação elementar podem acumular erros por se tratar de arredondamentos.
- Não representar frações.
- Caso um dos elementos que esteja na posição a_{ij} em que $i = j$ seja zero, a posição entre as linhas deve ser alterada de modo que nenhum elemento na posição de pivô seja igual a zero.

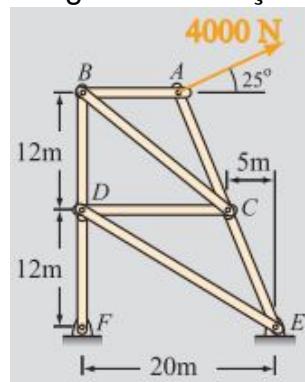
10 APLICAÇÃO

Álgebra Linear pode ser aplicada em diversas áreas para facilitar o processo de resolução de problemas, as matrizes por exemplo são utilizadas para solucionar problemas que envolvam sistemas de equações considerados complexos. Na Engenharia

Mecânica é possível aplicar sistema de equações em análise estrutural, a qual determina as forças nos membros de uma treliça utilizando o método dos nós e o método das seções. Treliças são estruturas esbeltas conectadas entre si nas extremidades e normalmente são usadas em construções tais como: pontes, telhados, viadutos e estruturas de máquinas.

Considere a treliça a seguir:

Figura 47: Treliça



Fonte: GILAT; SUBRAMANIAN (2008, p. 114)

Encontrando as equações das forças nos membros da treliça.

- $0,9231F_{AC} = 1690;$
- $F_{AB} - 0,7809F_{BC} = 0;$
- $F_{CD} + 0,8575F_{DE} = 0;$
- $0,3846F_{CE} - 0,3846F_{AC} - 0,7809F_{BC} - F_{CD} = 0;$
- $0,9231F_{AC} + 0,6247F_{BC} - 0,9231F_{CE} = 0;$
- $-F_{AB} - 0,3846F_{AC} = 3625;$
- $0,6247F_{BC} - F_{BD} = 0;$
- $F_{BD} - 0,5145F_{DE} - F_{DF} = 0.$

Dessa forma, o sistema de equações acima pode ser representado em notação matricial da seguinte forma:

$$\left[\begin{array}{ccccccc} 0 & 0,9231 & 0 & 0 & 0 & 0 & 0 \\ -1 & -0,3846 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0,8575 \\ 1 & 0 & -0,7809 & 0 & 0 & 0 & 0 \\ 0 & -0,3846 & -0,7809 & 0 & -1 & 0,3846 & 0 \\ 0 & 0,9231 & 0,6247 & 0 & 0 & -0,9231 & 0 \\ 0 & 0 & 0,6247 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -0,5145 \end{array} \right] = \left[\begin{array}{c} F_{AB} \\ F_{AC} \\ F_{BC} \\ F_{BD} \\ F_{CD} \\ F_{CE} \\ F_{DE} \\ F_{DF} \end{array} \right] = \left[\begin{array}{c} 1690 \\ 3625 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right].$$

Observe que na matriz dos coeficientes das variáveis existem elementos nulos na posição a_{ij} em que $i = j$, para poder inserir a matriz dos coeficientes no algoritmo *Eliminação de Gauss* é necessário alterar as linhas de modo que os elementos que estejam na posição a_{ij} em que $i = j$ não sejam nulos, pois o algoritmo não realiza a troca de linhas e o fator que será calculado para zerar os elementos abaixo da diagonal principal é encontrado a partir dos elementos que ocupam as posições em que $i = j$, logo:

$$\left[\begin{array}{ccccccc} -1 & -0,3846 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0,9231 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -0,7809 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0,6247 & -1 & 0 & 0 & 0 \\ 0 & -0,3846 & -0,7809 & 0 & -1 & 0,3846 & 0 \\ 0 & 0,9231 & 0,6247 & 0 & 0 & -0,9231 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0,8575 \\ 0 & 0 & 0 & 1 & 0 & 0 & -0,5145 \end{array} \right] = \left[\begin{array}{c} F_{AB} \\ F_{AC} \\ F_{BC} \\ F_{BD} \\ F_{CD} \\ F_{CE} \\ F_{DE} \\ F_{DF} \end{array} \right] = \left[\begin{array}{c} 3625 \\ 1690 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right].$$

Após alterar a posição das linhas da matriz dos coeficientes das variáveis e do vetor constante, as matrizes são inseridas do algoritmo.

$$A = \left[\begin{array}{ccccccc} -1 & -0,3846 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0,9231 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -0,7809 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0,6247 & -1 & 0 & 0 & 0 \\ 0 & -0,3846 & -0,7809 & 0 & -1 & 0,3846 & 0 \\ 0 & 0,9231 & 0,6247 & 0 & 0 & -0,9231 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0,8575 \\ 0 & 0 & 0 & 1 & 0 & 0 & -0,5145 \end{array} \right] \text{ e } B = \left[\begin{array}{c} 3625 \\ 1690 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right].$$

Ao inserir as matrizes e executar o algoritmo será mostrada a matriz dos coeficientes escalonada, o vetor das constantes equivalente, o valor do determinante e o vetor solução.

Figura 48: Forças nos membros de uma treliça.

```

Matriz A escalonada
[-1, -0.3846, 0, 0, 0, 0, 0, 0]
[0, 0.9231, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0, 0, -0.7809, 0.0, 0.0, 0.0, 0.0, 0.0]
[0, 0, 0, -1.0, 0.0, 0.0, 0.0, 0.0]
[0, 0, 0, 0, -1.0, 0.3846, 0.0, 0.0]
[0, 0, 0, 0, 0, -0.9231, 0.0, 0.0]
[0, 0, 0, 0, 0, 0, 0.8575, 0.0]
[0, 0, 0, 0, 0, 0, 0, -1.0]

Vetor B equivalente
3625
1690.0
4329.120896977575
3463.185842415023
-3625.0
1773.1858424150232
-2886.2205882430744
1731.4534894691785

A matriz possui determinante igual a 0.57 e valor solução [-4329.120896977575, 1830.7875636442423, -5543.758351872935,
-3463.185842415023, 2886.2205882430744, -1920.9033067002742, -3365.8549134030022, -1731.4534894691785]

```

Fonte: Autoria própria (2021).

Dessa forma temos que as forças que atuam sobre os membros da treliça são:

$$\begin{aligned}
F_{AB} &= -4329.120896977575 \text{ N;} \\
F_{AC} &= 1830.7875636442423 \text{ N;} \\
F_{BC} &= -5543.758351872935 \text{ N;} \\
F_{BD} &= -3463.185842415023 \text{ N;} \\
F_{CD} &= 2886.2205882430744 \text{ N;} \\
F_{CE} &= -1920.9033067002742 \text{ N;} \\
F_{DE} &= -3365.8549134030022 \text{ N;} \\
F_{DF} &= -1731.4534894691785 \text{ N;}
\end{aligned}$$

11 CONCLUSÃO

Nesse trabalho é abordada a ideia da interdisciplinaridade, unindo componentes curriculares para uma auxiliar no aprendizado da outra. Dessa forma, a metodologia abordada desenvolve o conhecimento nas disciplinas de Álgebra Linear e Programação. As definições e operações de matrizes foram mostradas e desenvolvidas em Python para facilitar a criação dos códigos e a lógica em programação foi utilizada para gerar os algoritmos da forma mais compreensível. Os sistemas de equações lineares foram transformados em notação matricial para que fosse verificado se o mesmo possui solução e caso existir fosse encontrada facilmente através do método de Gauss,

a linguagem de programação é empregada para solucionar o problema rapidamente, sendo que é possível utilizar para matrizes de diversas ordem, ou seja, para um sistema de equações considerado grande. Uma vez que os sistemas foram simplificados pelos conceitos de matrizes e utilizando Python a resolução se tornou mais eficiente para sistemas com várias equações lineares, é possível realizar aplicações no campo da engenharia. Algumas das aplicações possíveis são em análise estrutural como mostrado anteriormente e em circuitos elétricos para encontrar as correntes presentes, onde o algoritmo para solucionar sistemas de equações só é eficaz para circuitos que apresentem diversas malhas, de forma que para encontrar as correntes é necessário construir um sistema com diversas equações. Dessa forma, conclui-se que a interligação das disciplinas de Álgebra Linear e Programação facilita na aprendizagem em ambas as disciplinas e torna viável a construção de soluções para determinados problemas, otimizando assim o tempo.

REFERÊNCIAS

BANIN, Sérgio Luiz. **Python 3**. Saraiva Educação SA, 2018. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788536530253/>. Acesso em: 07 maio 2021.

BOLDRINI, José Luiz et al. **Álgebra Linear** 3º Edição, 1984. São Paulo: Editora Harbra – Harper & Row do Brasil.

GUIMARÃES, Patricia Baldow; MAGALHÃES, Antônio Pádua. A importância da interdisciplinaridade no ensino superior universitário no contexto da sociedade do conhecimento. **Revista Científica Vozes dos Vales**, Diamantina, n. 9. Disponível em: <http://site.ufvjm.edu.br/revistamultidisciplinar/files/2016/06/Patricia.pdf>. Acesso em: 20 maio 2021.

PAZ, Hallison. Soma e Subtração de Matrizes | Matemática com Python #5. Youtube, 25 de fev. 2019. Disponível em: <https://www.youtube.com/watch?v=rk5KM9rfWs4>. Acesso em: 16 abril 2021.

PAZ, Hallison. Representação de Matrizes | Matemática com Python. Youtube, 24 de mar. 2018. Disponível em: <https://www.youtube.com/watch?v=O5JoebBvxbM>. Acesso em: 06 maio 2021.

SANTOS, Emanuele. Ideia Geral do Método da Eliminação de Gauss. Youtube, 11 de set. 2020. Disponível em: <https://cutt.ly/6TteKLr>. Acesso em: 23 jun. 2021.

SANTOS, Emanuele. Implementação do Método da Eliminação de Gauus. Youtube, 11 de set. 2020. Disponível em: <https://cutt.ly/iTtrfgg>. Acesso em: 05 jul. 2021.

SANTOS, Emanuele. Introdução a Sistemas de Equações Lineares. Youtube, 07 de set. 2020. Disponível em: <https://www.youtube.com/watch?v=KGFrnTTyOn8>. Acesso em: 10 jun. 2021.

STEINBRUCH, Alfredo; WINTERLE, Paulo. **Álgebra Linear**. 2º Edição, 1987, Makron Books.

Índice Remissivo

- adicionar, 32
- Adição, 23
- algoritmo, 24
- alterar, 29
- aplicações, 11
- aprendizagem, 9
- arquivos, 17
- bibliotecas, 11
- classificadas, 49
- cofator, 53
- comutativa, 38
- criada, 26
- código, 25
- dados, 26
- Dependente, 80
- dependência, 79
- Determinante, 48
- diagonal, 21
- download, 11
- elementares, 55
- eliminação, 57
- equivalente, 58
- escalar, 39
- estrutural, 93
- ferramentas, 14
- função, 25
- Incompatível, 81
- Independente, 79
- Indeterminado, 81
- instalação, 13
- instruções, 14
- Integrado, 14
- interdisciplinaridade, 9
- interligação, 9
- intervalo, 25
- introduzidas, 24
- inversa, 74
- inversão, 75
- Laplace, 52
- Lineares, 81
- listas, 24
- lógica, 24
- manipulada, 55
- matricial, 79
- matriz, 19
- modificar, 36
- multiplicação, 37
- método, 59
- nomenclatura, 19
- numerações, 26
- NumPy, 65
- não-singulares, 76
- operações, 55
- performance, 66
- permutações, 49
- posto, 80
- previamente, 55
- procedimento, 27
- proporcionais, 54
- Propriedades, 24
- PyCharm, 17
- Python 3, 12
- repetição, 25
- representação, 19
- secundária, 21
- sequências, 24
- singular, 74
- sintaxe, 10
- sistemas, 79
- Sobredeterminado, 80
- softwares, 10
- solicitado, 30
- Subdeterminado, 81
- submatriz, 53
- Substituindo, 75
- subtração, 36
- teorema, 52
- transposta, 45
- triangulação, 55
- troca, 54
- unitária, 22

variável, 25

índices, 49
única, 76