PROFESSOR DE MATEMÁTICA ONLINE
Revista eletrônica da Sociedade Brasileira de Matemática

PMO v.11, n.4, 2023
ISSN: 2319-023X
https://doi.org/10.21711/2319023x2023/pmo1133

Python™ com o Google Colab para entender conceitos básicos de Cálculo Diferencial e Integral

Gilcilene Sanchez de Paulo Caroline Viezel Isabella Lopes Corrêa

Resumo

Este texto traz uma proposta do uso da linguagem Python™ de programação no ensino do Cálculo Diferencial e Integral. Para muitos, quando se ouve falar em Python™ imediatamente já o associam à resolução de problemas que envolvam grandes conjuntos de dados. Entretanto, essa linguagem de programação tem utilidade inclusive, na área da Educação. Este texto traz propostas de atividades para uma disciplina do ensino superior, mas que podem ser reformuladas no contexto de outros níveis devido à facilidade de utilização do Python™ através do serviço de *notebook* Google Colaboratory, que é oferecido gratuitamente pelo Google sem necessidade de instalação ou de um computador. As atividades apresentadas aqui foram aplicadas em um minicurso no XV Simpósio de Matemática da FCT-Unesp, *campus* de Presidente Prudente/SP, e estão agrupadas em três módulos que abordam funções, limite e continuidade, derivadas, integrais e um experimento do pensamento computacional.

Palavras-chave: Python™ ; Google Colaboratory; Educação; Cálculo Diferencial e Integral; Pensamento Computacional.

Abstract

This text presents a proposal for using the programming language Python™ in teaching Differential and Integral Calculus. For many, when talking about Python™ , they immediately associate it to the problems that involve large data sets. However, this programming language is also useful in the area of Education. This text brings proposals for activities for a higher education discipline, but which can be reformulated in the context of other levels due to the ease manipulation of Python™ through the notebook Google Colaboratory, which is offered for free by Google, and it is not necessary any software installation process, nor a computer. The activities presented in this text were applied in a short course from XV Symposium of Mathematics of FCT/Unesp, campus from Presidente Prudente-SP, and are grouped into three modules that address functions, limit and continuity, derivatives, integrals and an experiment on computational thinking.

Keywords: Python™ ; Google Colaboratory; Education; Differential and Integral Calculus; Computational Thinking.

1. Introdução

Na atual Base Nacional Comum Curricular (BNCC) a palavra *tecnologia* aparece 259 vezes e a denominação mais recente na BNCC *pensamento computacional* já aparece 9 vezes. O termo *tecnologia* não aparece vinculado somente à área da Matemática, como na seção específica sobre "A área da Matemática e suas Tecnologias", mas também em seções como: "A área de Linguagens e suas Tecnologias", e "A área de Ciências da Natureza e suas Tecnologias". A BNCC traz com muita ênfase a necessidade de preparar as crianças e adolescentes brasileiros para os avanços da computação e das tecnologias digitais. O texto relaciona computação e tecnologias digitais a conhecimentos, habilidades, atitudes e valores através das dimensões bem definidas como: pensamento computacional, mundo digital e cultura digital. Tais dimensões vão além da simples utilização das ferramentas; elas visam o desenvolvimento de algoritmos, a codificação, armazenamento e proteção da informação, a compreensão dos impactos destes avanços na sociedade atual, a capacidade de expressar soluções e manifestar-se no âmbito cultural de forma contextualizada e crítica (ver detalhes em [2]).

Muitos cursos de ensino superior para formação de professores já apresentam em seus Projetos Políticos Pedagógicos (PPP) estratégias e disciplinas que atendem a esta forte demananda da BNCC. O presente texto visa oferecer uma sugestão de como utilizar a linguagem de programação Python™ na disciplina de Cálculo Diferencial e Integral ou em outra atividade complementar do curso, como o caso desta proposta que foi elaborada para um minicurso [8] no XV Simpósio de Matemática da FCT/Unesp, *campus* de Presidente Prudente-SP baseada no livro [1], porém utilizando o Google Colaboratory [4].

A linguagem Python™ foi escolhida por ser uma ferramenta muito potente, utilizada em pesquisas de alta *performance*, mas que pode ser introduzida em níveis básicos da Educação pela facilidade de utilização, gratuidade e ausência da necessidade de instalação ou até mesmo de um computador. Nesta proposta, a sugestão é iniciar o alunado na linguagem de programação Python™ através do serviço de *notebook* Google Colaboratory, ou Colab como é conhecido, que é totalmente gratuito e que pode ser utilizado em *smartphones*, *tablets* ou computadores ligados a uma rede de internet e conectados em uma conta Google (que também é gratuita). Como os códigos são executados no sistema em nuvem, em computadores do Google, os recursos de memória disponibilizados para o usuário, por exemplo, são variáveis, de modo a atender todos os interessados sem custo nenhum (ver mais detalhes em [4]).

Este texto está dividido da seguinte forma: a seção 2 traz um passo a passo de como utilizar o Google Colaboratory para aquelas pessoas que nunca tiveram contato com a ferramenta. A seção 3 restringe-se a descrever brevemente a biblioteca do Python™ que será utilizada nas atividades deste texto. As seções 4, 5 e 6, referem-se aos Módulos de atividades de Cálculo Diferencial e Integral utilizando o Colab, sendo, respectivamente, o Módulo I sobre funções, limite e continuidade, o Módulo II sobre derivadas e um experimento do pensamento computacional, e o Módulo III sobre integrais. E, por fim, todos os códigos com as implementações dos módulos são disponibilizados na última seção, seção 7, para que o leitor interessado possa fazer o *download*, imprimir ou copiar.

2. Como utilizar o Google Colaboratory?

Para iniciar a programação em Python™ utilizando o Google Colab não é necessário a instalação de nenhum *software* em seu computador, *smartphone* ou *tablet*.

Uma possível maneira de acessar o Google Colab é executando os seguintes passos:

Passo 1: Como o Google Colab é um produto oferecido pelo Google, o usuário deverá abrir uma conta Google, @gmail.com, gratuitamente clicando aqui, caso ainda não possua uma, ou acessando

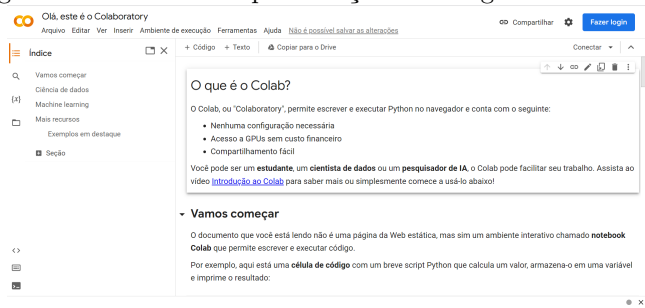
https://support.google.com/accounts/answer/27441?hl=pt

Passo 2: Acesse o endereço eletrônico

https://colab.research.google.com/notebooks/intro.ipynb

que levará a um *notebook* de apresentação como mostra a Figura 1.

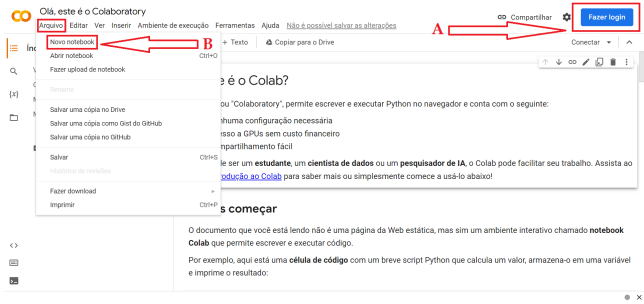
Figura 1: *Notebook* de apresentação do Google Colaboratory.



Fonte: Disponível em: https://colab.research.google.com/.
Acesso em: 10 fev. 2023 pelo navegador Google Chrome.

Passo 3: Caso ainda não tenha acessado o sistema, o usuário deverá efetuar a entrada clicando em "Fazer login" no canto superior direito, como mostra a Figura 2 (seguir seta com início no ponto A).

Figura 2: *Notebook* de apresentação do Google Colaboratory.



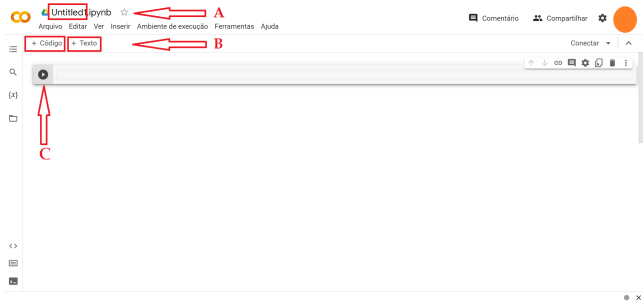
Fonte: Disponível em: https://colab.research.google.com/.
Acesso em: 10 fev. 2023 pelo navegador Google Chrome.

Passo 4: Para abrir o ambiente de programação em Python™ , o *notebook* Google Colab, o usuário deve acessar na barra horizontal de *menu* principal, no canto superior esquerdo, "Arquivo > Novo notebook", como mostra a Figura 2 (seguir seta com início no ponto B).

Passo 5: O usuário já está pronto para iniciar a programação e utilizar o Google Colab em problemas do Cálculo Diferencial e Integral, como será apresentado nas próximas seções deste texto. O

usuário pode escrever todo o seu código na caixa digitável apresentada na Figura 3 (seguir seta com início no ponto C) e, em seguida, executar os comandos implementados clicando no símbolo *play* ou, se preferir, o usuário pode documentar o seu código inserindo caixas de textos em "+ Texto", que inclusive aceitam textos LaTeX, e seguir intercalando a adição de novas caixas de código em "+ Código" (ver Figura 3 (seguir seta com início no ponto B)). O usuário pode atribuir um novo nome ao seu *notebook*, substituindo o nome "Untitled0" que aparece no canto superior esquerdo como mostra a Figura 3 (seguir seta com início no ponto A).

Figura 3: Novo *notebook* do Google Colaboratory. Pronto para iniciar a programação.



Fonte: Disponível em: https://colab.research.google.com/. Acesso em: 10 fev. 2023 pelo navegador Google Chrome.

3. A biblioteca SymPy no Cálculo Diferencial e Integral

SymPy é uma biblioteca do Python™ muito útil para manipulação de expressões algébricas. É um código aberto que pode ser estendido e/ou adaptado, totalmente gratuito sob a licença *Berkeley Software Distribution* (BSD).

Pode-se importar funções específicas da biblioteca SymPy. Entretanto, para este texto toda biblioteca SymPy será importada como: **from sympy import ***, pois, dessa forma, pode-se usar funções como **simplify**, **expand**, **solve**, diretamente, em vez de usar **sympy.simplify**, **sympy.expand**, **sympy.solve**. Para ir além deste texto, o leitor pode consultar o endereço eletrônico da SymPy [15] ou do prórpio Python™ [9], bem como livros para iniciantes, como, por exemplo, [7].

4. Módulo I

Neste primeiro módulo serão resolvidos em Python™ diversos exemplos percorrendo os seguintes tópicos: funções, gráficos, limites e continuidade de funções reais de uma variável real.

4.1. Funções

Como mencionado na Introdução deste texto, o objetivo desta seção será apresentar os comandos do Python™ para resolver de forma rápida e direta os problemas de Cálculo Diferencial e Integral. Se houver necessidade, o leitor deverá acompanhar o conteúdo conceitual do tópico em bibliografias de sua preferência (por exemplo, [1], [5], [7], [9], [14]).

Com o Exemplo 4.1, o leitor aprenderá definir uma função em uma variável simbólica, expressar um valor da imagem da função e, quando a biblioteca SymPy não retornar o valor real, no caso de funções transcendentais, também avaliar este valor da imagem como um número decimal.

Exemplo 4.1. Em Python™ , utilizando o Google Colab,

a) Defina f(x) = $\frac{\sin(x)}{x}$. b) Calcule f$\left(\frac{\pi}{2}\right)$. c) Dê uma aproximação decimal para f$\left(\frac{\pi}{2}\right)$.

Solução:

No *notebook* do Google Colab, na caixa de código, primeiramente, vamos importar toda biblioteca SymPy.

```
from sympy import *
```

A função **symbols** será utilizada para definir o objeto simbólico, a variável x do problema.

```
# gerando o objeto simbólico (a variável)
x = symbols ('x' )
```

Para textos mais explicativos pode-se inserir caixas de textos em "+ Texto", como mencionado no **Passo 5** da seção 2, mas também, pode-se inserir breves comentários na própria caixa de código precedendo o comentário com o símbolo #.

A função **pprint** (*pretty print*) imprime a expressão em modo matemático.

```
# 4.1a)
# Define a função na variável x
fx = sin(x) / x
pprint(fx)
```

saída: $\frac{\sin(x)}{x}$

A função **subs** fará a substituição da variável simbólica pelo valor numérico requerido.

```
# 4.1b)
# Calcula o valor da função para x = pi/2
fa = fx.subs(x,pi/2)
pprint(fa)
```

saída: $\frac{2}{\pi}$

Note que o procedimento adotado para resolver o Exemplo 4.1b) não retornou um valor numérico. Dessa forma, a função **evalf** fará essa avaliação numérica. Pode-se escolher o número de casas decimais n para o arredondamento com **evalf(**n**)**

```
# 4.1c)
# avalia uma expressão numérica
fa.evalf(3)
```

saída: 0.637

A função **evalf( )** também pode ser usada sem nenhum argumento. Para mais informações consulte [1], [7], [15], por exemplo.

4.1.1 Funções Definidas por Partes

As funções com diferentes sentenças para cada subintervalo de seu domínio estão muito presentes no Cálculo Diferencial e Integral. Portanto, no Exemplo 4.2 o leitor verá uma forma de definir estas funções em Python™ utilizando a função **Piecewise** e, que a função **subs** pode continuar sendo empregada como anteriormente.

Exemplo 4.2. Em Python™ , utilizando o Google Colab,

a) Defina a função $\phi(x) = \begin{cases} x^2 + 3, & x \leq 1 \ ; \\ x + 1, & x > 1 \ . \end{cases}$ b) Calcule $\phi(0)$, $\phi(1)$ e $\phi(5)$.

Solução:

```
from sympy import *
x, y1 , y2 = symbols ('x, y1 , y2' )
```

```
# 4.2a)
y1 = x**2 + 3
y2 = x + 1

phix = Piecewise(
      (y1, x <= 1),
      (y2, x > 1) )
```

```
# 4.2b)
# calcula o valor da função para x dado
phi0 = phix.subs(x,0)
phi1 = phix.subs(x,1)
phi5 = phix.subs(x,5)
```

```
# imprime os resultados
print('phi(0)= ', phi0)
print('phi(1)= ', phi1)
print('phi(5)= ', phi5)
```

saída: phi(0)= 3
phi(1)= 4
phi(5)= 6

4.2. Gráficos

Há muitas outras bibliotecas em Python™ para visualização de dados e esboço de gráficos, como Matplotlib e suas extensões. O leitor interessado nos recursos dessas bibliotecas pode consultar os endereços eletrônicos das bibliotecas Matplot [6] e Seaborn [11], por exemplo.

Neste texto, com o intuito mais simplicista, de motivar iniciantes, a biblioteca SymPy continuará sendo adotada para mostrar como esboçar gráficos em Python™ de expressões simbólicas.

Exemplo 4.3. Em Python™ , utilizando o Google Colab, esboce o gráfico da função

a) y = f(x) do Exemplo 4.1. b) y = ϕ(x) do Exemplo 4.2.

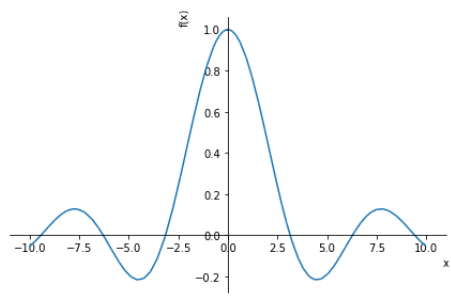
Solução:

```
from sympy import *
x, y1 , y2 = symbols ( 'x, y1 , y2' )
```

Utilizando simplesmente a função **plot** já é possível traçar gráficos de expressões simbólicas em Python™ .

```
# 4.3a)
plot(fx)
```

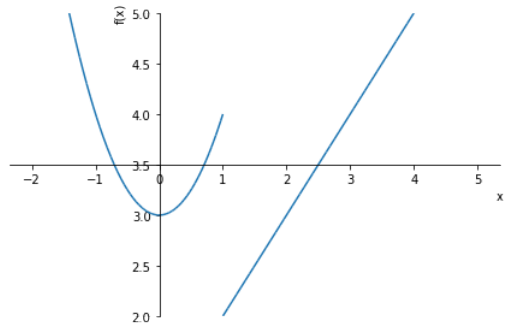
saída:



A função ϕ apresenta uma descontinuidade. Neste caso, a biblioteca SymPy não apresenta o esboço esperado quando se utiliza apenas **plot**(phix). Para excluir a linha vertical que aparece no ponto de descontinuidade, pode-se utilizar a mesma estratégia de esboçar gráficos de diversas funções em um mesmo sistema cartesiano, separando as sentenças. A informação de um intervalo que contém a imagem da função pela opção **ylim** também pode melhorar a apresentação do gráfico.

```
# 4.3b) - Melhorando a visualização
# plota duas sentenças no mesmo plano cartesiano
plot((y1, (x, -2, 1)), (y2, (x, 1, 5)), ylim = (2,5))
```

saída:



Outras opções para melhorar o esboço de gráficos em Python™ pela SymPy serão apresentadas no Módulo II. Quando o minicurso baseado neste texto foi ministrado, a ideia era aplicar alguns conceitos em um Módulo e retomá-los nos Módulos seguintes adicionando novos elementos.

4.3. Limites

O cálculo do limite é extremamente simples, faz-se por meio da função **limit**, sendo possível calcular todos os tipos de limites, limites laterais, limites no infinito e limites infinitos.

Por meio da sintaxe **limit(***expressão para* f(x)**,** *variável simbólica***,** *o valor de* x_0**)** a função **limit** calcula diretamente $\lim_{x \to x_0} f(x)$. Para mais detalhes, sugerimos [1], [5], [7], [9] e [14], por exemplo.

A seguir serão apresentados exemplos para cada tipo de limite.

4.3.1 Limites Laterais

Para o cálculo dos limites laterais, basta acrescentar os símbolos '+' ou '-', para limites laterais à direita ou à esquerda, respectivamente, como mais um argumento da função **limit**.

A seguir, exemplificamos as diversas situações, incluindo um *bug* que está presente na função **limit** para um determinado caso, e uma estratégia para contornar tal problema até que seja resolvido. Exemplos adicionais para este módulo podem ser encontrados no material complementar deste texto, seção 7.

Exemplo 4.4. Em Python™ , utilizando o Google Colab, calcule

a) $\lim_{x \to 0^+} f(x)$ e $\lim_{x \to 0^-} f(x)$ para f do Exemplo 4.1. b) $\lim_{x \to 1^+} \phi(x)$ e $\lim_{x \to 1^-} \phi(x)$ para ϕ do Exemplo 4.2.

Solução:

```
# 4.4a) Limite lateral à direita
print( limit(fx, x, 0, '+' ))
# Limite lateral à esquerda
print(limit(fx, x, 0, '-' ))
```

saída: 1
1

```
# 4.4b) Limite lateral à direita
print(limit(phix, x, 1, '+' ))
# Limite lateral à esquerda
print(limit(phix, x, 1, '-' ))
```

saída: 4
4

Portanto, $\lim_{x\to 0} f(x) = 1$. Entretanto, como mostrado neste Exemplo 4.4b), vemos que a biblioteca SymPy não está confiável para o cálculo de limites laterais no caso de funções definidas por partes, pois sabemos que não existe $\lim_{x\to 1} \phi(x)$. Funcionaria bem, caso a função ϕ não estivesse definida em x = 1, por exemplo.

ATENÇÃO: No caso de funções definidas por partes, recomendamos o uso do cálculo do limite sobre cada sentença separadamente, e concluir que o limite não existe quando os limites sobre cada setença forem distintos, ou seja, escreva:

```
# 4.4b) Para funções definidas por partes, recomenda-se
print(limit(x**2 + 3, x, 1))
print(limit(x + 1, x, 1))
```

saída: 4
2

Portanto, não existe $\lim_{x\to 1} \phi(x)$.

4.3.2 Limites no Infinito

A sintaxe para o uso da função **limit** no cálculo de limites no infinito é a mesma. A informação adicional aqui é quanto à representação do símbolo ∞ que é escrito com duas letras 'o' minúsculas: oo. Para mais sugestões de apresentação dos resultados por meio da biblioteca Sympy, no Exemplo 4.5 a seguir, atente-se à atribuição dos valores dos limites às novas variáveis, L_1 e L_2, forma de impressão dos resultados, inclusão de assíntotas horizontais e cores no gráfico.

Exemplo 4.5. Seja f(x) = arctan(x). Em Python™ , utilizando o Google Colab,

a) Calcule $\lim_{x\to+\infty} f(x)$ e $\lim_{x\to-\infty} f(x)$.

b) Esboce o gráfico de f.

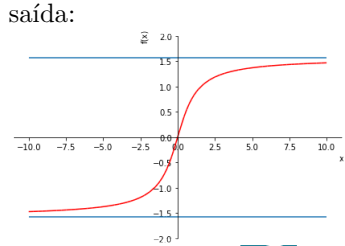
Solução:

```
# 4.5a)
expr4 = atan(x)
# Limite no infinito ( +oo )
L1 = limit(expr4, x, 'oo' )
# Limite no infinito ( –oo )
L2 = limit(expr4, x, '-oo' )
print('L1 =', L1, 'e L2 =', L2)
```

saída: L1 = pi/2 e L2 = -pi/2

```
# 4.5b) Gráfico de y = arctan(x) e das assíntotas horizontais
grafico_assintotas= plot((expr4,(x, -10, 10)), (pi/2, (x, -10, 10)),
(-pi/2, (x, -10, 10)), ylim = (-2,2), show=False)
grafico_assintotas[0].line_color =  'red'
grafico_assintotas.show()
```

saída:



O último parâmetro da função **plot** recebe False para que o gráfico não seja exibido antes da modificação da sua cor, para destacar o gráfico da função das assíntotas horizontais. Para isso armazenam-se os parâmetros da função **plot** na variável/vetor grafico_assintotas e por meio da função **line_color** altera-se a cor do gráfico para vermelho ('red'). Como o gráfico da função y = arctan(x) é o primeiro parâmetro da função **plot**, então atribui-se a cor para a primeira componente do vetor grafico_assintotas, que em Python™ inicia-se com o índice zero. Finalmente, após essa modificação desejada, exibe-se o gráfico utilizando a função **show**.

4.3.3 Limites Infinitos

Para encerrar as sugestões de atividades envolvendo todos os tipos de limites e o Python™ , o próximo exemplo traz a função $f(x) = \frac{1}{x}$, muito utilizada nos cursos de Cálculo. Observe, também, a utilização de outras opções de argumento da função **plot** que podem ser úteis para melhorar a apresentação gráfica, como as opções *title* e *size*, por exemplo. Para melhorar outras configurações dos gráficos , sugerimos a biblioteca SymPy [15] (ou especificamente, [16]).

Exemplo 4.6. Seja $f(x) = \frac{1}{x}$. Em Python™ , utilizando o Google Colab,

a) Calcule $\lim_{x\to 0^+} f(x)$ e $\lim_{x\to 0^-} f(x)$.

b) Esboce o gráfico de f.

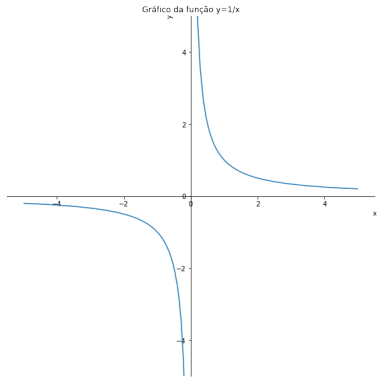
Solução:

```
# 4.6a)
expr5 = 1 / x
# Limite infinito (x -> p+)
print(limit(expr5, x, 0, '+' ))
# Limite infinito (x -> p-)
print(limit(expr5, x, 0, '-' ))
```

saída: ∞
$-\infty$

```
# 4.6b) Inserindo algumas opções no gráfico
#plot(expr5,(x, -1, 1), ylim=(-25,25))
y = symbols('y')
plot((expr5,(x, -5, 0)), (expr5,(x, 0, 5)), ylim=(-5,5), title =
'Gráfico da função y=1/x', ylabel=y, size=(8,8))
```

saída:



4.4. Continuidade

Nesta seção será exibida a implementação de um exemplo para discutir continuidade de funções, com os conhecimentos de Python™ adquiridos até aqui. Aos leitores interessados em utilizar este material para trabalhar continuidade com o Google Colab, recomenda-se acrescentar os exemplos adicionais que podem ser encontrados no material complementar deste texto, seção 7.

Considera-se que os conceitos teóricos de continuidade já foram ou serão abordados com os alunos.

Exemplo 4.7. Seja $f(x) = \begin{cases} x^2 + 3, & x < 2 \ ; \\ 9 - x, & x >= 2 \ . \end{cases}$

Em Python™ , utilizando o Google Colab,

a) Defina f(x). b) Calcule $\lim_{x \to 2^+} f(x)$. c) Calcule $\lim_{x \to 2^-} f(x)$.

d) Quais os valores de f(1), f(2) e f(3)? e) Faça o gráfico de f.

Solução:

```
# 4.7a)
y1 = x**2+3
y2 = 9 - x
fX = Piecewise(
(y1, x < 2),
(y2, x >= 2) )
```

```
# 4.7b)
# Limite lateral à direita
L1 = limit(fX, x, 2, '+')
# 4.7c)
# Limite lateral à esquerda
L2 = limit(fX, x, 2, '-')
```

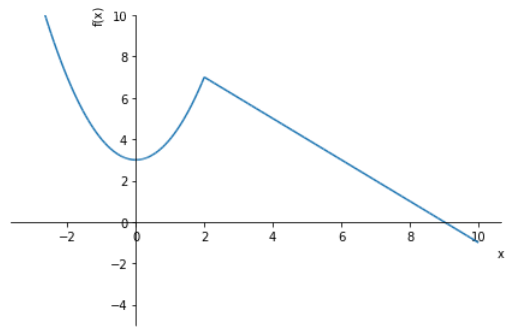
```
# 4.7d)
# Valor da função em p = 1, 2 e 3
f1 = fX.subs(x,1)
f2 = fX.subs(x,2)
f3 = fX.subs(x,3)
```

```
print('L1 =', L1, 'L2 =', L2, 'e f(2) =', f2)
print('f(1) =', f1, 'e f(3) =', f3)
```

saída: L1 = 7 L2 = 7 e f(2) = 7
f(1) = 4 e f(3) = 6

```
# 4.7e)
# Gráfico da função
plot((y1, (x, -3, 2)), (y2, (x, 2, 10)), ylim = (-5,10))
```

saída:



5. Módulo II

Este Módulo II traz uma proposta de utilizar a interdisciplinaridade, de iniciar o desenvolvimento do pensamento computacional dentro de uma disciplina mais teórica. Além de calcular derivadas utilizando a biblioteca SymPy, este módulo introduz a possibilidade de o aluno elaborar passos para resolver uma aplicação do Cálculo Diferencial e automatizar tal resolução utilizando a linguagem de programação Python™ . Dessa forma, o aluno será encorajado a desenvolver funções e não só utilizar as funções disponíveis na biblioteca SymPy.

5.1. Como implementar minha própria função?

Para que o aluno possa resolver problemas mais elaborados de aplicações do Cálculo utilizando o Google Colab, será muito útil que o aluno saiba como implementar suas próprias funções, de acordo com a sua linha de racíocinio. A seguir apresenta-se uma sintaxe possível do Python™ para criar funções utilizando a função **def** e **retur**.

def NomeDaFunção (argumento1, argumento2,⋯, argumentoN)**:**

pode conter instruções a serem executadas que geram resultado

return resultado

Não é necessário que haja instruções a serem executadas que geram resultados. Veja as linhas de código a seguir que criam a função **hello**, a qual simplesmente recebe um nome como parâmetro, ou seja, uma *string*, e imprime uma frase de saudação.

```
def hello(nome):
    print('Olá', nome)
```

Neste momento, a função **hello** já está pronta para ser utilizada, como descrita a seguir.

ATENÇÃO: A função **print** não pode ficar alinhada com a função **def**. O avanço na identação é mandatório no Python™ (ver [7], [9], por exemplo).

```
hello('Fulano')
```

saída: Olá Fulano

As próximas linhas de código já executam instruções e retornam um resultado. A função **SnPA** apresentada a seguir calcula a soma dos n primeiros termos de uma P.A. (progressão aritmética) de razão r, inciando do termo a_1.

```
def SnPA(a1, r, n):
    an = a1 + (n-1)*r
    sn = ((a1 + an)*n)/2
    return sn
```

Exemplo 5.1. Calcule a soma dos 100 primeiros termos da P.A. $(3, 5, 7, 9, 11, \dots)$ em Python™ , utilizando o Google Colab e a função **SnPA**.

Solução:

```
SnPA(3, 2, 100)
```

saída: 10200.0

5.2. Derivadas

Para o cálculo de derivadas , será utilizada a função **diff** do SymPy. Aproveita-se este momento para apresentar as funções **simplify** e **trigsimp** do Sympy, muito utilizadas para simplificar expressões matemáticas, sendo a segunda mais uma opção para o caso de expressões trigonométricas.

Exemplo 5.2. Calcule a derivada de f(x) = tan(x),

a) usando a definição por limite.

b) usando a função **diff** do SymPy.

Solução:

```
# gerando os objetos simbólicos (as variáveis)
x , y , z , h = symbols ( 'x , y , z , h' )
razao = (tan(x+h) - tan(x))/h
```

```
# 5.2a)
# limites laterais e o limite
derivadaD = limit(razao, h, 0, '+')
derivadaE = limit(razao, h, 0, '-')
derivada = limit(razao, h, 0)
print(derivadaD, derivadaE, derivada)
pprint(derivada)
```

saída: cos(x)**(-2) cos(x)**(-2) cos(x)**(-2)

$$\frac{1}{\cos^2(x)}$$

```
# 5.2b)
# Definindo a expressao da função
expr1 = tan(x)
# calcula a derivada
derivaexpr1 = diff(expr1,x)
pprint(derivaexpr1)
```

saída: $\tan^2(x) + 1$

```
# trigsimp: simplifica expressões trigonométricas
derivadaS = trigsimp(derivaexpr1)
pprint(derivadaS)
```

saída: $\frac{1}{\cos^2(x)}$

5.3. Reta Tangente

A interpretação geométrica da derivada de uma função não pode faltar em um curso de Cálculo. Com as funções do SymPy apresentadas neste texto já é possível pedir que o aluno exiba no Google Colab a expressão da reta tangente ao gráfico de f no ponto p e os gráficos para visualização. O próximo Exemplo 5.3 vem com esta proposta.

Exemplo 5.3. Seja f(x) = x^2 .

a) Determine $f'(x)$. b) Calcule f(1) e $f'(1)$.

c) Determine a expressão da reta tangente ao gráfico de f no ponto $(1, 1)$.

d) Faça os gráficos de f e da reta tangente passando pelo ponto $(1, 1)$.

Solução:

```
# Definindo f
expr5 = x**2
print('f(x) = ', expr5)
# 5.3a)
print('dydx = ', diff(expr5,x))
# 5.3b)
# calcula f(1)
print('f(1) = ', expr5.subs(x,1))
# calcula f'(1)
print('Coeficiente angular f\'(1) =', diff(expr5,x).subs(x,1) )
# 5.3c)
y = expr5.subs(x,1) + diff(expr5,x).subs(x,1)*(x - 1)
pprint(y)
```

saída:
f(x) = x**2
dydx = 2*x
f(1) = 1
Coeficiente angular f'(1) = 2
2·x - 1

Neste momento já é possível que o aluno automatize esse procedimento e crie sua própria função no Google Colab, utilizando Python™ , para retornar a reta tangente, dados a expressão da função e o ponto p. Uma ideia para isso é dada a seguir criando a função **retatangente**.

```
# Criando nossa própria função para retornar a reta tangente
def retatangente(f, p):
    x = symbols ('x')
    return f.subs(x,p) + diff(f,x).subs(x,p)*(x - p)
```

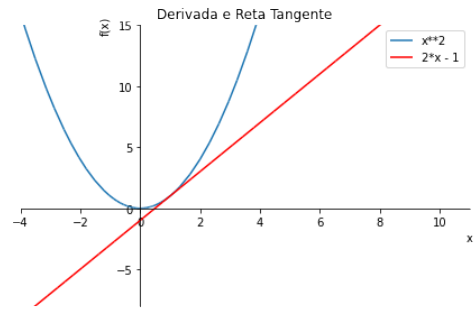
```
# 5.3c)
r = retatangente(x**2, 1)
print('A reta tangente ao gráfico de f em p é y = ' , r)
```

saída: A reta tangente ao gráfico de f em p é y = 2*x - 1

Veja no gráfico a seguir como incluir legendas, ou seja, como incluir o parâmetro de legenda na função **plot**.

```
# 5.3d)
p = plot(expr5, r, xlim=(-4,11), ylim = (-8,15), title = 'Derivada e Reta Tangente', legend = True, show=False)
p[1].line_color = 'red'
p.show()
```

saída:



Note que o recurso de modificar a cor de um gráfico também foi utilizado aqui, como foi explicado no Exemplo 4.5b). A diferença é que neste Exemplo 5.3b) a cor foi modificada para o segundo parâmetro da função **plot**, que corresponde ao índice 1.

5.4. Derivadas de Ordem Superior e Derivadas Parciais

Descrevendo a notação brevemente, seja y = f(x), as derivadas de f até ordem n denotam-se por:

$$\frac{dy}{dx} = f'(x),\ \frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = f''(x),\ \frac{d^3y}{dx^3} = \frac{d}{dx}\left(\frac{dy^2}{dx^2}\right) = f'''(x),\ \cdots,\ \frac{d^ny}{dx^n} = \frac{d}{dx}\left(\frac{dy^{n-1}}{dx^{n-1}}\right) = f^{(n)}(x).$$

A função **diff** possui o recurso de calcular estas derivadas de outras ordens por meio da seguinte sintaxe: **diff(***expressão para* f(x)**,** *variável simbólica***,** *ordem da derivada***)** ou, repetindo a variável de derivação o número de vezes da sua ordem:
diff(*expressão para* f(x)**,** *variável simbólica***,** *variável simbólica***, ...,** *variável simbólica***)**.

Exemplo 5.4. Seja $f(x) = -2x^3 + x^2 - 2x - 5$. Calcule todas as derivadas de f até a ordem 4.

Solução:

```
# 5.4)
expr6 = -2*x**3 + x**2 -2*x - 5
print( diff(expr6, x) )
print( diff(expr6, x, 2) )   # ou print( diff(expr6, x, x) )
print( diff(expr6, x, 3) )   # ou print( diff(expr6, x, x, x) )
print( diff(expr6, x, 4) )   # ou print( diff(expr6, x, x, x, x) )
```

saída: -6*x**2 + 2*x - 2
2*(1 - 6*x)
-12
0

Outra funcionalidade de **diff** é lidar com funções de várias variáveis reais e calcular as derivadas parciais de todas as ordens. Veja a proposta do Exemplo 5.5 a seguir.

Exemplo 5.5. Seja $f(x,y,z) = x^3y^4z^6$. Calcule $\dfrac{\partial^9 f(x)}{\partial x^2 \partial y^3 \partial z^4} = \dfrac{\partial^2}{\partial x^2}\left(\dfrac{\partial^3}{\partial y^3}\left(\dfrac{\partial^4 f(x)}{\partial z^4}\right)\right)$.

Solução:

```
# 5.5) expr7 = x**3 * y**4 * z**6
diff(expr7, x, 2, y, 3, z, 4)
```

saída: $51840xyz^2$

5.5. Aplicação: Taxas Relacionadas

Este texto traz uma sugestão de atividade envolvendo taxas relacionadas com o objetivo de expor o aluno ao raciocínio computacional. O aluno deverá entender o problema como uma aplicação da derivada e, em adição, deverá elaborar um algoritmo e passar estas instruções ao computador, por meio da linguagem Python™ , a fim de que a máquina resolva por ele todas as situações análogas.

Este desafio é lançado em termos do Exemplo 5.6.

Exemplo 5.6. Escadas deslizantes

Considere uma escada de comprimento Lm dado, apoiada em uma parede vertical.

A escada começa escorregar horizontalmente a uma velocidade v_x m/s e verticalmente a uma velocidade v_y m/s.

Desenvolva sua própria função em Python™ , utilizando o Google Colab, tal que:

a) relacione as distâncias horizontal[1] (d1) e vertical[2] (d2).

b) relacione as velocidades de deslizamento horizontal e vertical.

c) organizar as funções de a) e b) para automatizar as soluções dos problemas da escada deslizante como descritos:

Uma escada de 6 m está apoiada em uma parede vertical.

Se a base da escada começa a deslizar horizontalmente à razão de 0,6 m/s, com que velocidade o topo da escada percorre a parede quando está a 4 m de altura do solo?

d) Aplique o programa já desenvolvido para resolver o problema semelhante:

Uma escada de 8 m está apoiada em uma parede vertical.

Se o topo da escada começa a deslizar verticalmente à razão de 0,4 m/s, com que velocidade a base da escada percorre o solo quando está a 5 m de distância da parede?

Solução:

Pensamento computacional para implementação de 5.6a)

Sejam x e y as distâncias do pé da escada à parede e do topo da escada ao solo, respectivamente. Temos, pelo teorema de Pitágoras, $x^2 + y^2 = L^2$, então, $x = \sqrt{L^2 - y^2}$ ou $y = \sqrt{L^2 - x^2}$. Nota-se que

[1] distância horizontal: a distância da parede até a base da escada.
[2] distância vertical: a distância do solo até o topo escada.

o cálculo dos catetos segue o mesmo padrão. Dessa forma, basta desenvolver apenas uma função, nomeada aqui por **pitagorascateto**.

```
# 5.6a)
# Relacionar as grandezas distâncias
def pitagorascateto(a,c):
    b = (a**2 - c**2)**0.5
    return b
```

Pensamento computacional para implementação de 5.6b)

x = x(t) e y = y(t), t variável temporal. Então,

$x(t)^2 + y(t)^2 = L^2$, pela regra da cadeia, $2x(t)\frac{dx}{dt} + 2y(t)\frac{dy}{dt} = 0$.

Como $v_x = \frac{dx}{dt}$ e $v_y = \frac{dy}{dt}$, vem que: $v_x = -\frac{y(t)}{x(t)}v_y$, ou $v_y = -\frac{x(t)}{y(t)}v_x$.

Note que, a velocidade em uma direção é proporcional à velocidade na outra direção, cuja constante de proporcionalidade é dada pelo oposto da razão das distâncias em cada direção, e que é possível estabelecer um padrão na formação desta razão. Dessa forma, independentemente da velocidade que pretende-se calcular, na direção horizontal ou vertical, basta criar apenas uma função, nomeada aqui por **velslipstairs**.

```
# 5.6b)
# Relacionar as grandezas velocidades
def velslipstairs(v1, d1, d2):
    v2 = - ( v1 * d1 ) / d2
    return v2
```

Resolvendo 5.6c):

Lembrando que os códigos apresentados aqui são apenas sugestões.

Modo 1: Uma possibilidade, induzida pelos itens 5.6a) e 5.6b), é dada pela chamada das duas funções criadas anteriormente, **pitagorascateto** e **velslipstairs**.

```
# 5.6c)
catetox = pitagorascateto(6,4)
vy = velslipstairs(0.6, catetox, 4)

print('Quando a escada encontra-se à altura de 4 m do solo, \n a distância horizontal da escada até a parede é de', catetox, 'm.')

print('A velocidade com que o topo da escada percorre a parede \n quando a escada está a 4 m de altura do solo é de', vy, 'm/s.')
```

saída: Quando a escada encontra-se à altura de 4 m do solo,
distância horizontal da escada até a parede é de 4.47213595499958 m.

saída: A velocidade com que o topo da escada percorre a parede
quando a escada está a 4 m de altura do solo é de -0.6708203932499369 m/s.

Modo 2: Outra possibilidade é lançar o exercício de otimizar essa tarefa com a chamada de apenas uma função. Aqui, a sugestão é criar a função **VELescadadeslizante**.

```
# 5.6c)
def VELescadadeslizante(L, d2, v1):
    d1 = (L**2 - d2**2)**0.5
    v2 = - ( v1 * d1 ) / d2
    return v2, d1
```

Veja que neste momento foi apresentada uma forma de a função retornar dois valores, a velocidade requerida e a outra distância, horizontal ou vertical dependendo do problema. Desta forma, para solucionar o problema através da chamada de apenas uma função basta ter conhecimento da ordenação dos dados de entrada e também de saída.

```
# 5.6c)
velocidade, distancia = VELescadadeslizante(6, 4, 0.6)

print('A outra distância é de ', distancia, 'm.')
print('A velocidade procurada é de ', velocidade, 'm/s.')
```

saída: A outra distância é de 4.47213595499958 m.
A velocidade procurada é de -0.6708203932499369 m/s.

Neste momento, pode-se chamar atenção ao sistema de referencial, ou seja, ao sinal negativo da velocidade.

Resolvendo 5.6d):

Para este Exemplo 5.6d) basta lembrar do sinal negativo da velocidade indicando o deslizando da escada de cima para baixo.

```
# 5.6c)
velocidade2, distância2 = VELescadadeslizante(8, 5, -0.4)

print('A outra distância é de ', distância2, 'm.')
print('A velocidade procurada é de ', velocidade2, 'm/s.')
```

saída: A outra distância é de 6.244997998398398 m.
A velocidade procurada é de 0.49959983987187184 m/s.

6. Módulo III

Para completar as sugestões do uso do Google Colab em sala de aula, na disciplina de Cálculo Diferencial e Integral, este Módulo III é dedicado à Integração.

6.1. Integral

Nesta proposta serão apresentadas funções da biblioteca SymPy capazes de resolver integrais definidas e indefinidas de funções de uma variável, e integrais múltiplas para funções de várias variáveis.

Como nem sempre é possível exibir a primitiva de uma função integrável, este texto mostrará brevemente uma forma de obter o seu valor aproximado. Para isso, será necessário apresentar ao leitor uma outra biblioteca, SciPy, mais utilizada em cursos de Cálculo Numérico.

6.2. Integral Indefinida

Para o cálculo de integrais será utilizada a função **integrate** do SymPy, cuja sintaxe para integral indefinida é dada por **integrate(***expressão para* f**,** *variável simbólica***)**.

Exemplo 6.1. Em Python™ , utilizando o Google Colab, calcule a integral indefinida $\int \frac{x^5}{\sqrt{1-x^6}}dx$.

Solução:

```
# 6.1)
expr2 = x**5 / sqrt(1 - x**6)
integral2 = integrate(expr2,x)
# pretty print
pprint( integral2 )
```

saída: $-\frac{\sqrt{1-x^6}}{3}$

6.3. Integral Definida

Seja $f : [a, b] \to \mathbb{R}$ uma função integrável. A função **integrate** do SymPy também pode ser utilizada para o cálculo de integrais definidas com a seguinte sintaxe:

integrate(*expressão para* f, *variável simbólica x*, a, b).

Exemplo 6.2. Em Python™ , utilizando o Google Colab, calcule a integral definida $\int_{-\pi/2}^{1} \sin(x)\, dx$.

Solução:

```
# 6.2)
expr3 = sin(x)
integral3 = integrate(expr3, (x, -pi/2, 1))
pprint( integral3 )
integral3.evalf()
```

saída: $-\cos(1)$
-0.54030230586814

6.4. Integrais Múltiplas

Seja $f : [a_1, b_1] \times [a_2, b_2] \to \mathbb{R}$ uma função de duas variáveis reais a valores reais integrável. A função **integrate** do SymPy também pode ser utilizada para o cálculo de integrais múltiplas sobre regiões retangulares com a seguinte sintaxe:

integrate(*expressão para* f, (*variável simbólica x*, a_1, b_1), (*variável simbólica y*, a_2, b_2)).

No caso das funções de três variáveis reais a valores reais integráveis, para a integral tripa em um paralelepípedo $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$ basta adicionar mais uma terna de parâmetro com respeito a outra variável z e os limites inferior e superior de integração nesta direção: (*variável simbólica z*, a_3, b_3), ou seja, a sintaxe completa:

integrate(*expressão para* f, (*variável simbólica x*, a_1, b_1), (*variável simbólica y*, a_2, b_2), (*variável simbólica z*, a_3, b_3)).

Exemplo 6.3. Em Python™ , utilizando o Google Colab,

calcule a integral dupla $\int_{-1}^{1}\int_{0}^{3}(x^3 - y^2\cos(x))\, dx\, dy$.

Solução:

```
# 6.3)
expr4 = x**3 - y**2 * cos( x )
integral4 = integrate(expr4, (x, 0, 3), (y, -10, 1))
pprint( integral4 )
integral4.evalf()
```

saída: $\frac{891}{4} - \frac{1001 - \sin(3)}{3}$

175.662957310691

6.5. Primitiva Desconhecida

Uma leitura muito interessante sobre este tópico, que complementa os livros de Cálculo Diferencial e integral, é o artigo [3].

Apesar das funções e^{-x^2} e $\cos(x^5)$ serem contínuas para todos os reais, não é possível expressar suas primitivas em termos de "funções elementares" (ver por exemplo, o livro [12] p. 450-451, e o livro [13] p.687-688). Assim como em outros aplicativos, a função **integrate** do SymPy também retorna primitivas que não podem ser escritas por meio de funções elementares (ver [3] para entender o porquê!).

Exemplo 6.4. Em Python™ , utilizando o Google Colab, calcule a integral $\int_0^1 e^{-x^2}\,dx$.

Solução:

```
# 6.4)
integrate(exp(-x**2), (x, 0, 1))
```

saída: $\frac{\sqrt{\pi}\,\mathrm{erf}(1)}{2}$

onde só é possível explicar que, $\mathrm{erf}(1) = \frac{2}{\sqrt{\pi}} \int_0^1 e^{-x^2}\,dx$. A função erro, denotada por erf(x), é obtida integrando a função gaussiana. Para mais detalhes, veja [3].

Como trata-se de um texto que utiliza a linguagem Python™ , a próxima seção introduzirá ao leitor iniciante uma outra biblioteca, baseada em métodos numéricos, capaz de retornar um valor aproximado para as integrais de funções que não admitem primitivas elementares.

6.6. A biblioteca SciPy na Aproximação da Integral Definida

A biblioteca SciPy fornece comandos e classes de alto nível para manipulação e visualização de dados. Interessados, consultar o endereço eletrônico da biblioteca SciPy [10].

Para o objetivo desta seção, não será realizada a importação completa da biblioteca SciPy, como demonstrado anteriormente para o caso da SymPy.

A sintaxe para importar a função específica **quad**, que lida com fórmulas de quadratura, é:
from scipy.integrate import quad .

Dessa forma, para resolver aproximadamente o Exemplo 6.4, na caixa de código primeiramente importe a função **quad**:

```
from scipy.integrate import quad
```

Depois, para utilizar **quad** é necessário definir a função que terá o valor da sua integral definida aproximado. Ao invés de utilizar a função **def**, nesta oportunidade, será apresentada uma outra forma de definir funções em Python™ , mais rapidamente, por meio da função **lambda**.

```
# 6.4)
funcaointegrando = lambda x: exp(-x**2)
resultado, erro = quad(funcaointegrando, 0, 1)

print(resultado)
print(erro)
```

saída: 0.7468241328124271
8.291413475940725e-15

A função **quad** retorna dois valores, por isso que duas variáveis foram utilizadas, uma para armazenar o valor aproximado da integral (resultado) e a outra para armazenar uma estimativa do erro existente nesta aproximação.

7. Material complementar

Os códigos elaborados para cada módulo com atividades adicionais e os *slides* que foram utilizados no XV SMAT estão disponíveis nos seguintes endereços eletrônicos (basta clicar sobre o nome para iniciar o *download*): Módulo I Módulo II Módulo III *Slides*

8. Considerações Finais

O uso de uma ferramenta simples, porém potente, de programação desde os níveis mais básicos da Educação pode auxiliar a despertar jovens talentos em Matemática ou em áreas afins. Durante o curso de Licenciatura em Matemática, este uso serve de exemplo prático para os futuros professores em formação. Além disso, ao utilizar esta ferramenta nas séries iniciais de cursos de Matemática ou áreas afins, pode-se preparar melhor o aluno que for cursar a disciplina de Cálculo Numérico, por exemplo. Os professores das disciplinas de Cálculo Diferencial e Integral e Cálculo Numérico, podem trabalhar em conjunto a interdisciplinaridade de maneira mais simples e construindo o pensamento computacional.

O Google Colab veio como uma ferramenta de grande potencial para democratização e difusão da linguagem de programação Python™ na Educação e pode ser uma grande aliada da BNCC para "A área de Matemática e suas Tecnologias".

Referências

[1] Barbosa, A.C.C; Rojas, A.; Concordido, C.F.R.; Carvalhaes, G.G; *Cálculo Diferencial e Integral de Funções de uma Variável com Python*, Editora Ciência Moderna Ltda., Rio de Janeiro, 2019.

[2] BRASIL. Ministério da Educação. Base Nacional Comum Curricular. Brasília, 2018. Disponível em: <http://basenacionalcomum.mec.gov.br/>. Acesso em: 10 de fevereiro de 2023.

[3] De Morais Filho, D.C.; *Professor, qual a primitiva de* e^{x^2} *?*. Matemática Universitária, Brasil, v. 31, p. 143-161, 2001.

[4] Google Colaboratory. Disponível em: <https://colab.research.google.com/>. Acesso em: 10 de fevereiro de 2023.

[5] Guidorizzi, H. *Um curso de Cálculo*. Volume 1. 5ª edição. São Paulo: LTC, 1995.

[6] Matplotlib: Visualization with Python. Disponível em: <https://matplotlib.org/>. Acesso em: 10 de fevereiro de 2023.

[7] Menezes, N.N.C., *Introdução à Programação com Python: Algoritmos e Lógica de Programação Para Iniciantes*. 3ª ed. Novatec Editora, 2019.

[8] Paulo, G.S; Viezel, C; Corrêa, I.L. Minicurso. *In*: SMAT 2021. Disponível em: <https://www.fct.unesp.br/!/departamentos/matematica-e-computacao/docentes/gilcilene–sanchez-de-paulo/>. Acesso em: 10 de fevereiro de 2023.

[9] Python™ . Disponível em: <https://www.python.org/>. Acesso em: 10 de fevereiro de 2023.

[10] SciPy. Disponível em: <https://docs.scipy.org/doc/scipy-0.18.1/reference/index.html>. Acesso em: 10 de fevereiro de 2023.

[11] Seaborn: statistical data visualization. Disponível em: <https://seaborn.pydata.org/>. Acesso em: 10 de fevereiro de 2023.

[12] Stewart, J. *Cálculo*. 7ª ed. São Paulo: Cengage Learning, vol. 1, 2015.

[13] Stewart, J. *Cálculo*. 7ª ed. São Paulo: Cengage Learning, vol. 2, 2013.

[14] Swokowski, E.W. *Cálculo com Geometria Analítica*. 2ª ed. São Paulo: Makron Books, 1994.

[15] SymPy. Disponível em: <https://www.sympy.org/>. Acesso em: 10 de fevereiro de 2023.

[16] SymPy-Plotting. Disponível em: <https://docs.sympy.org/latest/modules/plotting.html>. Acesso em: 10 de fevereiro de 2023.

Gilcilene Sanchez de Paulo
Departamento de Matemática e Computação - DMC,
Universidade Estadual Paulista, UNESP, Presidente Prudente, SP
<gilcilene.sanchez@unesp.br>

Caroline Viezel
Doutora em Ciências da Computação e Matemática Computacional,
Universidade de São Paulo, USP, São Carlos, SP
<carol.viezel@gmail.com>

Isabella Lopes Corrêa
Mestre em Matemática Aplicada e Computacional,
Universidade Estadual Paulista, UNESP, Presidente Prudente, SP
<isabella.lopes@unesp.br>

Recebido: 02/04/2023
Publicado: 20/12/2023