Elementos do grupo:

* Diogo Fonseca nº 79858
* Tomás Teodoro nº 80044
* Diogo Silva nº 79828
* Tiago Granja nº 79845

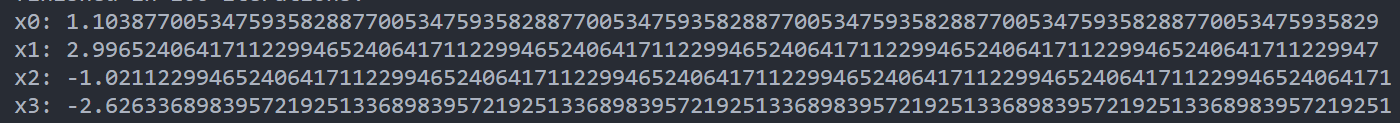
Resumo

O programa é capaz de aproximar o integral de uma função, de x0 a x1 dado que a função é contínua dentro desses pontos. Isto é feito usando o método de Newton-Cotes Aberto e Fechado para n = 4. O programa calcula também os erros relativos das duas aproximações.

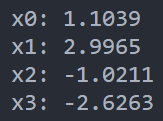
Instruções de Utilização

Nota: pressione *Ctrl+C* a qualquer momento durante a execução para parar o programa.

* **Precision:**  – O número (inteiro) de algarismos significativos a ser usado internamente pelo programa. ()

Exemplo: ()

Exemplo: ()



* **Expression:**  – A função matemática à qual se pretende integrar.
  + **Variáveis**: ‘x’.
  + **Operadores aritméticos:** ‘+’, ‘-‘, ‘/’, ‘\*’.
  + **Funções Trigonométricas:** ’’, ‘’, ‘’, ‘’, ‘’, ‘’, ‘acos(x)’, ‘asin(x)’, ‘atan(x)’, ‘asec(x)’, ‘acsc(x)’, ‘acot(x)’, ‘cosh(x)’, ‘sinh(x)’, ‘tanh(x)’, ‘sech(x)’, ‘csch(x)’, ‘coth(x)’, ‘acosh(x)’, ‘asinh(x)’, ‘atanh(x)’, ‘asech(x)’, ‘acsch(x)’, ‘acoth(x)’.
  + **Expoentes:** ‘x\*\*y’ (ou) ‘x^y’ (x levantado a y).
  + **Logaritmo:** ‘’ (logaritmo de x, base n).
  + **Constantes:** ‘E’ (número de Euler), ‘pi’.
  + **Raízes:** ‘’ (para raiz quadrada ou elevar um número a 1/raiz)
  + **Fatoriais:** ‘’ (AVISO: O uso de fatoriais pode tornar o programa bastante ineficiente)

Exemplo:

Exemplo:

* 
* **Initial Range:**  – O número real (incluí constantes e aritmética) ao qual o integral vai integrar de.
* **Final Range:**  – O número real (incluí constantes e aritmética) ao qual o integral vai integrar até. ()

Execução

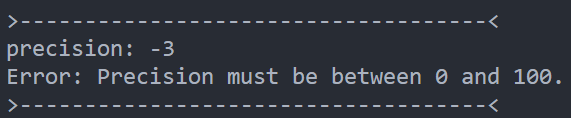
Cada ficheiro aqui posto deve ser colocado no seu ficheiro particular com o mesmo nome que a sua classe, este projeto usa também o auxílio da biblioteca matemática SymPy (<https://www.sympy.org>). Para a sua execução é necessário instalar a mesma para o seu funcionamento correto. Com python instalado no computador, isto pode ser feito através do terminal com o comando “pip3 install sympy”.

Observações

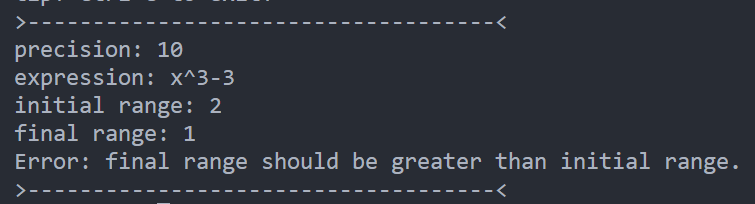
Os integrais aproximados, dão valores bastante aproximados quando a função se porta minimamente bem dentro do intervalo. Outra observação a fazer é que quando os limites de integração são muito grandes, o resultado perde qualidade, tendo um erro maior. É de notar também que de modo geral o método fechado tem menores erros que o método aberto.

O programa está preparado para obter qualquer tipo de input, mesmo que este viole as especificações previamente mencionadas, gerando uma mensagem de erro apropriada, mas continuando a execução.

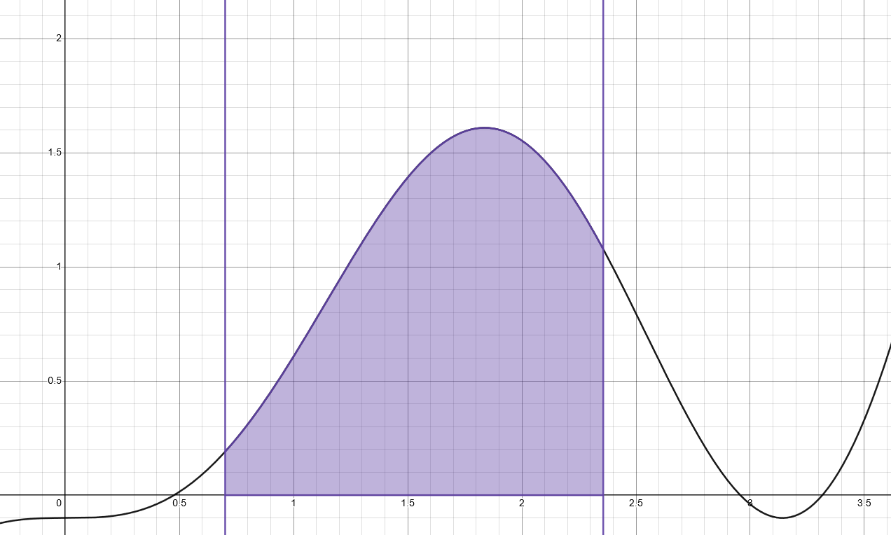
Exemplo:

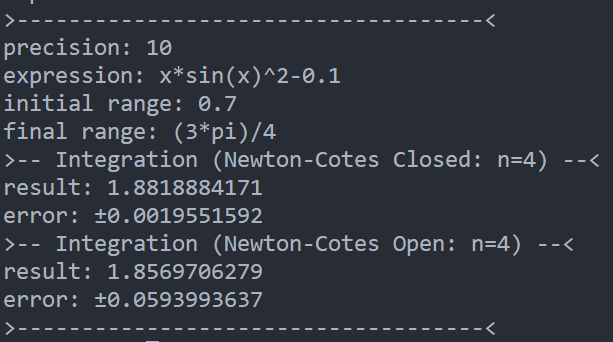


Exemplo:

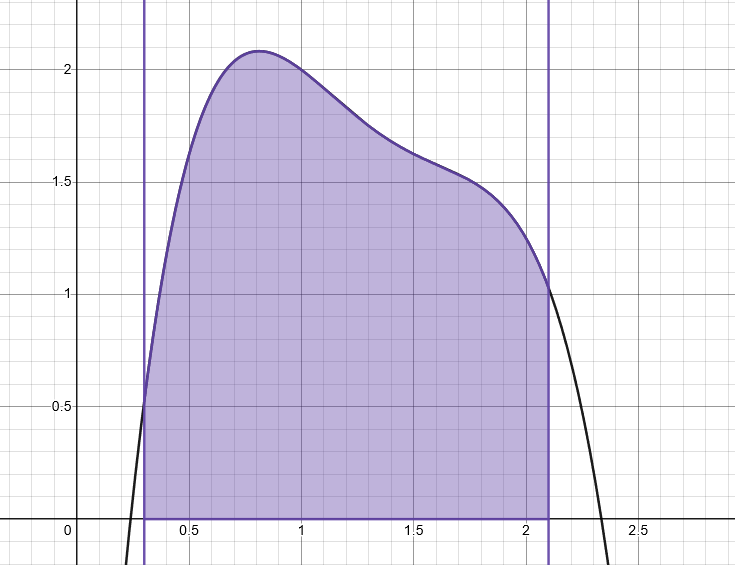


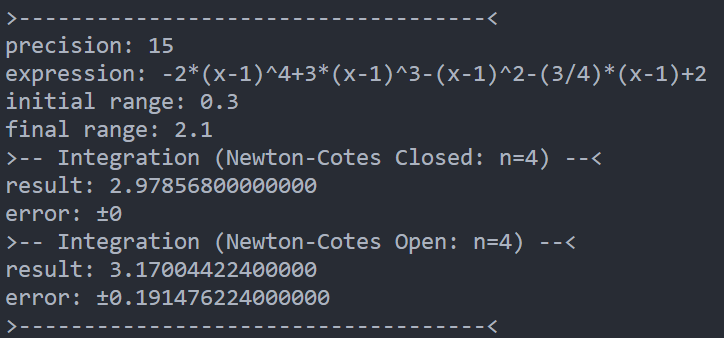
Screenshots



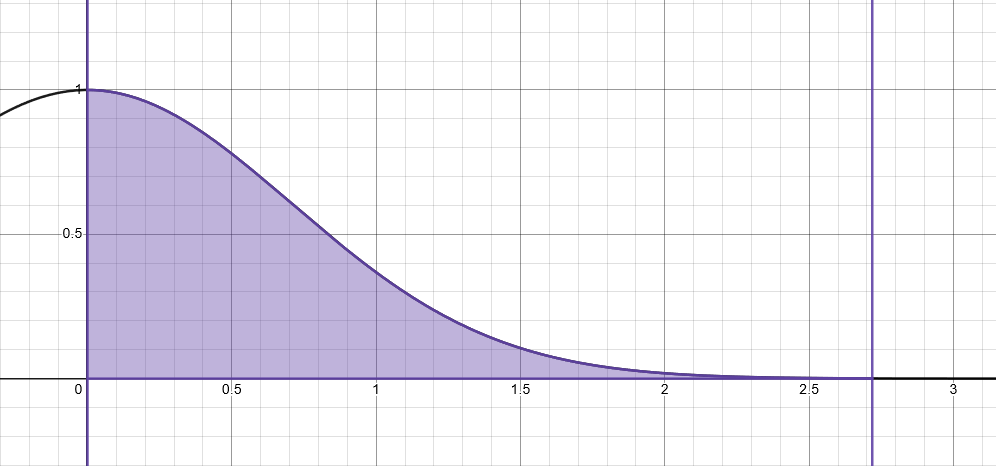


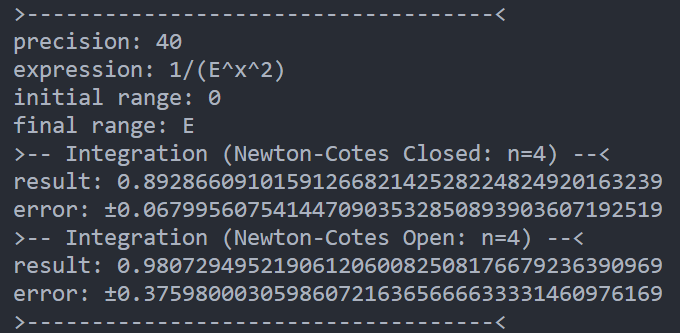
Valor real: 1.88254188804





Valor real: 2.978568





Valor real: 0.886119753109

Código

main.py

import math\_input

import nc\_closed

import nc\_open

from sympy import \*

exit = false

print("tip: Ctrl+C to exit!")

while exit == false:

    print(">------------------------------------<")

    try:

        precision = math\_input.math\_input.get\_precision()

        expression = math\_input.math\_input.get\_expression(precision)

        range0 = math\_input.math\_input.get\_range0(precision)

        range1 = math\_input.math\_input.get\_range1(range0, precision)

        try:

            print(">-- Integration (Newton-Cotes Closed: n=4) --<")

            result = nc\_closed.nc\_closed.solve(expression, range0, range1)

            error = nc\_closed.nc\_closed.calc\_error(expression, range0, range1)

            print(f"result: {round(result, precision)}")

            print(f"error: ±{round(error, precision)}")

        except Exception as e:

            print("Newton-Cotes Closed method failed.")

            print(str(e))

        try:

            print(">-- Integration (Newton-Cotes Open: n=4) --<")

            result = nc\_open.nc\_open.solve(expression, range0, range1)

            error = nc\_open.nc\_open.calc\_error(expression, range0, range1)

            print(f"result: {round(result, precision)}")

            print(f"error: ±{round(error, precision)}")

        except Exception as e:

            print("Newton-Cotes Open method failed.")

            print(str(e))

    except KeyboardInterrupt:

        exit = true

    except Exception as e:

        print(str(e))

print("")

print("Exiting...")

math\_input.py

from sympy import \*

class math\_input:

    @staticmethod

    def get\_precision():

        this\_input = input("precision: ")

        try:

            try:

                this\_input = int(this\_input)

            except:

                raise Exception("Error: precision must be an integer.")

            if not ask(Q.integer(this\_input)):

                raise Exception("Error: precision must be an integer.")

            if this\_input < 0 or this\_input > 100:

                raise Exception("Error: Precision must be between 0 and 100.")

            return int(this\_input)

        except Exception as e:

            raise e

    @staticmethod

    def get\_expression(*precision*):

        this\_input = input("expression: ")

        try:

            return N(this\_input, *precision*)

        except:

            raise Exception("Error: Invalid expression: '" + this\_input + "'.")

    @staticmethod

    def get\_range0(*precision*):

        this\_input = input("initial range: ")

        try:

            this\_input = N(this\_input, *precision*)

            if not ask(Q.real(this\_input)):

                raise Exception()

            return this\_input

        except:

            raise Exception("Error: Invalid range.")

    @staticmethod

    def get\_range1(*range0*, *precision*):

        this\_input = input("final range: ")

        try:

            this\_input = N(this\_input, *precision*)

        except:

            raise Exception("Error: Invalid range.")

        try:

            if not ask(Q.real(this\_input)):

                raise Exception("Error: Invalid range.")

            elif *range0* > this\_input:

                raise Exception(

                    "Error: final range should be greater than initial range."

                )

            return this\_input

        except Exception as e:

            raise e

nc\_open.py

from sympy import \*

class nc\_open:

    @staticmethod

    def solve(*expression*, *range0*, *range1*):

        h = nc\_open.calc\_h(*range0*, *range1*)

        return (5 / 24) \* h \* (

            11 \* *expression*.subs(Symbol("x"), nc\_open.calc\_x(*range0*, 0, h)) +

*expression*.subs(Symbol("x"), nc\_open.calc\_x(*range0*, 1, h)) +

*expression*.subs(Symbol("x"), nc\_open.calc\_x(*range0*, 2, h)) +

            11 \* *expression*.subs(Symbol("x"), nc\_open.calc\_x(*range0*, 3, h)))

    @staticmethod

    def calc\_h(*range0*, *range1*):

        return (*range1* - *range0*) / 5

    @staticmethod

    def calc\_x(*range0*, *i*, *h*):

        return *range0* + (*i* + 1) \* *h*

    @staticmethod

    def calc\_error(*expression*, *range0*, *range1*):

        h = nc\_open.calc\_h(*range0*, *range1*)

        fourth\_derivative = diff(*expression*, Symbol("x"), 4)

        max\_diff\_error = Abs(fourth\_derivative.subs(Symbol("x"), nc\_open.calc\_x(*range0*, -1, h)))

        for i in range(5):

            max\_diff\_error = Max(max\_diff\_error, Abs(fourth\_derivative.subs(Symbol("x"), nc\_open.calc\_x(*range0*, i, h))))

        return (95 / 144) \* Pow(h, 5) \* max\_diff\_error

nc\_closed.py

from sympy import \*

class nc\_closed:

    @staticmethod

    def solve(*expression*, *range0*, *range1*):

        h = nc\_closed.calc\_h(*range0*, *range1*)

        return (2 / 45) \* h \* (

            7 \* *expression*.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 0, h)) +

            32 \* *expression*.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 1, h)) +

            12 \* *expression*.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 2, h)) +

            32 \* *expression*.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 3, h)) +

            7 \* *expression*.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 4, h)))

    @staticmethod

    def calc\_h(*range0*, *range1*):

        return (*range1* - *range0*) / 4

    @staticmethod

    def calc\_x(*range0*, *i*, *h*):

        return *range0* + *i* \* *h*

    @staticmethod

    def calc\_error(*expression*, *range0*, *range1*):

        h = nc\_closed.calc\_h(*range0*, *range1*)

        sixth\_derivative = diff(*expression*, Symbol("x"), 6)

        max\_diff\_error = Abs(sixth\_derivative.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, 0, h)))

        for i in range(4):

            max\_diff\_error = Max(max\_diff\_error, Abs(sixth\_derivative.subs(Symbol("x"), nc\_closed.calc\_x(*range0*, i + 1, h))))

        return (8 / 945) \* Pow(h, 7) \* max\_diff\_error