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 x_0 a x_1 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:** d – O número (inteiro) de algarismos significativos a ser usado internamente pelo programa. ($0 < x \leq 100$)

  Exemplo: ($x = 100$)

```
1.10387700534759358288770053475935828877005347593582887700534759358288770053475935828877005347593582887700534759358288770053475935829
```

  Exemplo: ($x = 5$)

```
1.1039
```

- **Expression:** $\boldsymbol{f}(x)$ – A função matemática à qual se pretende integrar.
  - **Variáveis**: 'x'.
  - **Operadores aritméticos:** '+', '-', '/', '*'.
  - **Funções Trigonométricas:** '$cos(x)$', '$sin(x)$', '$tan(x)$', '$cot(x)$', '$sec(x)$', '$csc(x)$', '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:** '$log(x, n)$' (logaritmo de x, base n).
  - **Constantes:** 'E' (número de Euler), 'pi'.
  - **Raízes:** '$sqrt(x)$' (para raiz quadrada ou elevar um número a 1/raiz)
  - **Fatoriais:** '$x!$' (AVISO: O uso de fatoriais pode tornar o programa bastante ineficiente)

  Exemplo: $e^{\frac{1}{x}} - 5x^3 + 2x^2 - 2$

```
expression: E^(1/x)-5*x^3+2*x**2-2
```

  Exemplo: $x^5 + \frac{2}{3}x^2 + e^x - log_{10}(727x) + xlog_3(x) + \tan(x^2) - \frac{3}{2x} + \sqrt{2x^3}$

```
expression: x^5+(2/3)*x^2+E^x-log(727*x,10)+x*log(x,3)+tan(x^2)-3/(2*x)+sqrt(2*x^3)
```

- **Initial Range:** x_0 – O número real (incluí constantes e aritmética) ao qual o integral vai integrar de.
- **Final Range:** x_1 – O número real (incluí constantes e aritmética) ao qual o integral vai integrar até. ($x_0 \leq x_1$)

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:

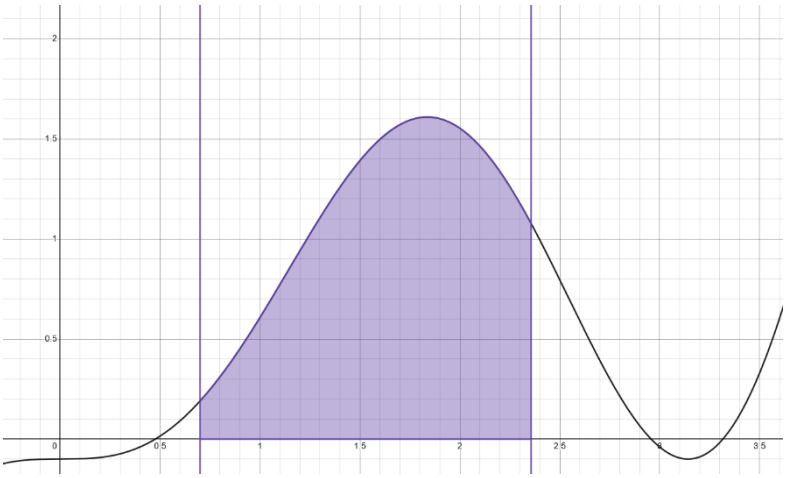
```
>------------------------------------<
precision: -3
Error: Precision must be between 0 and 100.
>------------------------------------<
```

Exemplo:

```
>------------------------------------<
precision: 10
expression: x^3-3
initial range: 2
final range: 1
Error: final range should be greater than initial range.
>------------------------------------<
```

Screenshots

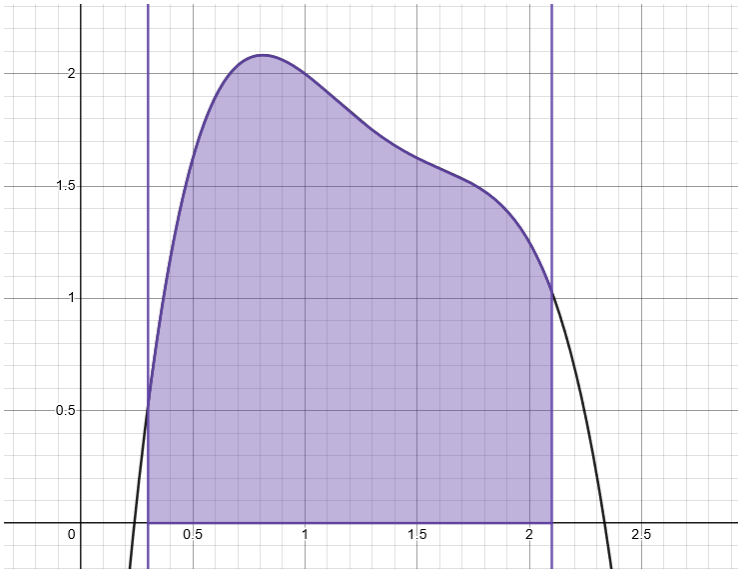
$$\int_{0.7}^{\frac{3\pi}{4}} x\sin^2(x) - 0.1$$



```
>-------------------------------------<
precision: 10
expression: x*sin(x)^2-0.1
initial range: 0.7
final range: (3*pi)/4
>-- Integration (Newton-Cotes Closed: n=4) --<
result: 1.8818884171
error: ±0.0019551592
>-- Integration (Newton-Cotes Open: n=4) --<
result: 1.8569706279
error: ±0.0593993637
>-------------------------------------<
```

Valor real: 1.88254188804

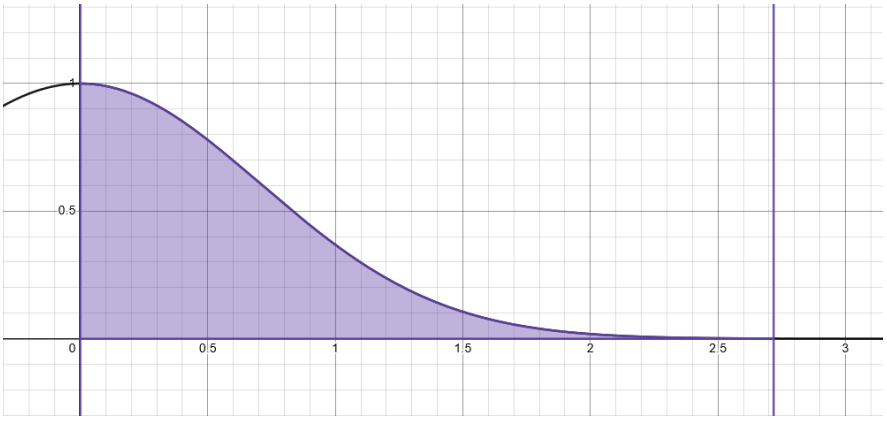
$$\int_{0.3}^{2.1} -2(x-1)^4 + 3(x-1)^3 - (x-1)^2 - \frac{3}{4}(x-1) + 2$$




```
>-----------------------------------<
precision: 15
expression: -2*(x-1)^4+3*(x-1)^3-(x-1)^2-(3/4)*(x-1)+2
initial range: 0.3
final range: 2.1
>-- Integration (Newton-Cotes Closed: n=4) --<
result: 2.97856800000000
error: ±0
>-- Integration (Newton-Cotes Open: n=4) --<
result: 3.17004422400000
error: ±0.191476224000000
>-----------------------------------<
```

Valor real: 2.978568

$$\int_0^e \frac{1}{e^{x^2}}$$




```
>-----------------------------------<
precision: 40
expression: 1/(E^x^2)
initial range: 0
final range: E
>-- Integration (Newton-Cotes Closed: n=4) --<
result: 0.89286609101591266821425282248249201632 39
error: ±0.06799560754144709035328508939036071925 19
>-- Integration (Newton-Cotes Open: n=4) --<
result: 0.98072949521906120600825081766792363909 69
error: ±0.37598000305986072163656666333314609761 69
>-----------------------------------<
```

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
```