Infelizmente, devido à uma confusão minha, acabei removendo o texto que tinha feito para a primeira e segunda etapa, juntamente com a pasta que tinha meu nome. (Precisei liberar espaço do gdrive "particular"). Tive que refazer então, o relatório 1 e 2, dessa vez, aproveitei para colocar mais detalhes e códigos com comparações sobre formas similares de fazer em outras linguagens

---

Etapa 1

Desenvolvi o algoritmo usado para resolver as equações diferenciais ordinárias, i.e., o algoritmo usando método de Euler e outro usando o Runge-Kutta de ordem 4 em python.

Com o *feedback* do professor, percebi realmente que poucos conceitos de OO seriam usados no projeto além de que o trabalho iria ser raso — teria pouco conteúdo, poucas linhas de programação — portanto, optei por mudar o paradigma para linguagem funcional e refazer os algoritmos.

O primeiro contato com a linguagem foi relativamente tranquilo, ela parecia uma mistura de python com MatLab/js. Ela não possui um caracter indicando fim de comandos, entretanto, tem as claves para indicar fim de um escopo. Trata-se de uma linguagem interpretada focada em desenvolvimento científico. Pelas pesquisas que havia feito, é amplamente utilizada no meio científico, pois, devido ao fato de ser simples, tornou-se uma excelente opção — é grátis, com bastante ferramentas estatísticas e matemáticas — quando levada em consideração ao MatLab, por exemplo.

Minha primeira dúvida era se haveria uma função **eval** para assim, tentar manter a base do algoritmo já feito. Uma vez que descobri que havia essa função, comecei a fazer as mudanças[1]. Observe que ainda faltava a substituição do caracter na string de entrada da função, para isso, precisava usar a função **gsub( )** que retorna a string modificada.

O primeiro obstáculo foi o uso das ***kwargs* — ***kwargs* são dicionários não declarados na função, de forma que quando chamo a função eu passo os valores indicando também o "campo" a que o valor pertence, pode ser visto como um *"json* interno da função" — que no **R** não havia algo tão explícito.

Exemplo de uso de uma ***kwarg*:

```
…
def odeEuler(self, **kwg):
…
n = CalcNum()
x = n.odeEuler(function='x - 2*y + 1',a=0,b=1,y0=1,m=10)#,VERBOSE=1)
```

Então pesquisando, descobri que havia algo similar — Optei por manter o código estruturado dessa forma por acreditar que diminui o erro do usuário, uma vez que ele precisa explicitamente dizer os parâmetros que quer alterar — que era o uso de lists "compactadas" com apelidos internos.[2]

Exemplo:

```
 …
odeEuler <- function(kwg, ...){    # Uma função é uma variável que recebe esta palavra
   # Em R, você pode os operadores <- = ou = -> para atribuir valor
```

```
    # Unpacking parameters
    args = list(...)
    x = args$a     #    x = 2
…
odeEuler(a=2)
```

Porém, ainda tinha um problema, em python havia o operador **in** para poder checar se havia sido passado um parâmetro ou não, i.e., se tinha uma *key* no dicionário. Precisava disso para poder verificar se o usuário queria a flag **VERBOSE** — flag usada no meu código para exibir o valor das variáveis a cada iteração — ou não. Descobri que em **R** era chamado de **.Names** e eram extraídos com a função **names( )**, com isso, eu poderia usar o operador **%in%**.[3]

Tive que pesquisar também como fazer uma exibição formatada, algo similar com o **printf( )** do **C** ou o **print("{}".format( ))** do **Python**. Em **R** você tem que fazer uso de duas funções, uma é apenas de exibição e a outra, faz essa formatação dos caracteres. Como o **R** também não permitia o recurso explícito de multiplicação por cadeia de caracteres na exibição — do **python** — tive que pesquisar, e descobri que através do **sprintf ( )** eu poderia fazer algo similar.

Exemplo:

```
…
cat(sprintf("----------", sep="", 1:value), sep="")  #  repete de 1 até a quantidade (value)
# sep é o fim da cadeia, pode ser o caractere '\n', por exemplo
# esse cat é similar ao MatLab
…
cat(sprintf("Function: \t%s\nTol:\t\t%.1E\nN0\t%d", args$func,tol, args$n0), sep = '\n')
…
```

Por fim, os últimos problemas dessa parte foi relacionado ao uso de vetores, em **R** um vetor é declarado usando a função **c(ELEMENTOS_DO_VETOR)** e para acessar os elementos, ele usa o índice 1, similar ao *MatLab*. Além disso, nos sites que estava usando como apoio, não falava sobre a característica do índice inicial, portanto, meu código não funcionava mas o **R não dizia o motivo**, em outras palavras, ele não me indicou o erro, o que me leva a crer que o **R** tem limitações para debug.

Uma observação importante é que, no caso das funções matemáticas, não precisei realizar um "import" ou algo do tipo, já estava de fácil acesso no ambiente. Outro ponto curioso é: como tudo é uma função, até mesmo o **return ( )**. Vale notar que o operador ternário do **R** é diferente do **python**.

Exemplo:

```
…
# Decimal places
DECIMAL = if ("DECIMAL" %in% names(args)) args$DECIMAL else self.DECIMAL
…
```

Etapa 2

Pelo *feedback* do professor, notei que era necessário acrescentar mais operações e funções ao meu código, portanto, comecei a desenvolver e colocar as funções passadas durante a aula de cálculo numérico no algoritmo, de forma que agora ele resolvesse não somente edo's, como também, raízes de polinômios.

Nessa parte, já pensava em como codificar baseando no que fiz até então.

Ainda nessa parte, descobri que para limpar uma variável, eu setava ela pra NULL.

Exemplo:

```
…
m = NULL
…
```

Etapa 3

Implementei mais 3 funções, similar à Etapa 2, entretanto, quando tentei usar a função de derivada do **R** ele me apresentou o seguinte erro:

> *Error: Due this msg: Error in deriv.default(a,'x') : invalid expression in 'FindSubexprs' you must pass funcDeriv also.*

Assim, tive que mudar a abordagem, obrigando o usuário, na função de iteração por método de newton, a entrar, não somente com a função base, como também com a sua derivada correspondente.

Nota: Não consegui achar nada que resolvesse o problema citado acima.

Pensei também em adicionar mais um parâmetro, para que assim, pudesse ser obtido os valores de cada iteração, i.e., o que era mostrado pelo parâmetro **VERBOSE** em forma de uma matriz. Adicionei então em cada função esse parâmetro.

Também adicionei os arquivos de documentação das funções que ainda não tinham, e atualizei os existentes.

Precisei pesquisar também, como fazer um "append" de vetores, formando uma matriz, que no **R** pode ser feito usando a função **rbind( )**.

Exemplo:

```
…
m = matrix(ncol=2)
m <- rbind(m, round(c(i+1, p), DECIMAL) )
…
```

Etapa 4

Resolvi mudar a organização de saída de matriz para dataframe, tentando melhorar a visualização dos dados — um **data.frame** em **R** é similar ao **dataframe** da biblioteca **pandas** no **python** —.

Para poder converter para dataframe, precisei de usar a técnica de *slice* — parecida também com o **python** —, porém, no **R** ele obriga a dizer o ponto de início e fim do corte, então precisei também da função **nrow( )**.

Exemplo:

```
    ...
    df = data.frame(Pn=m[2:nrow(m),2])
    return (df)
}
...
```

Atualizei a documentação.

Etapa 4

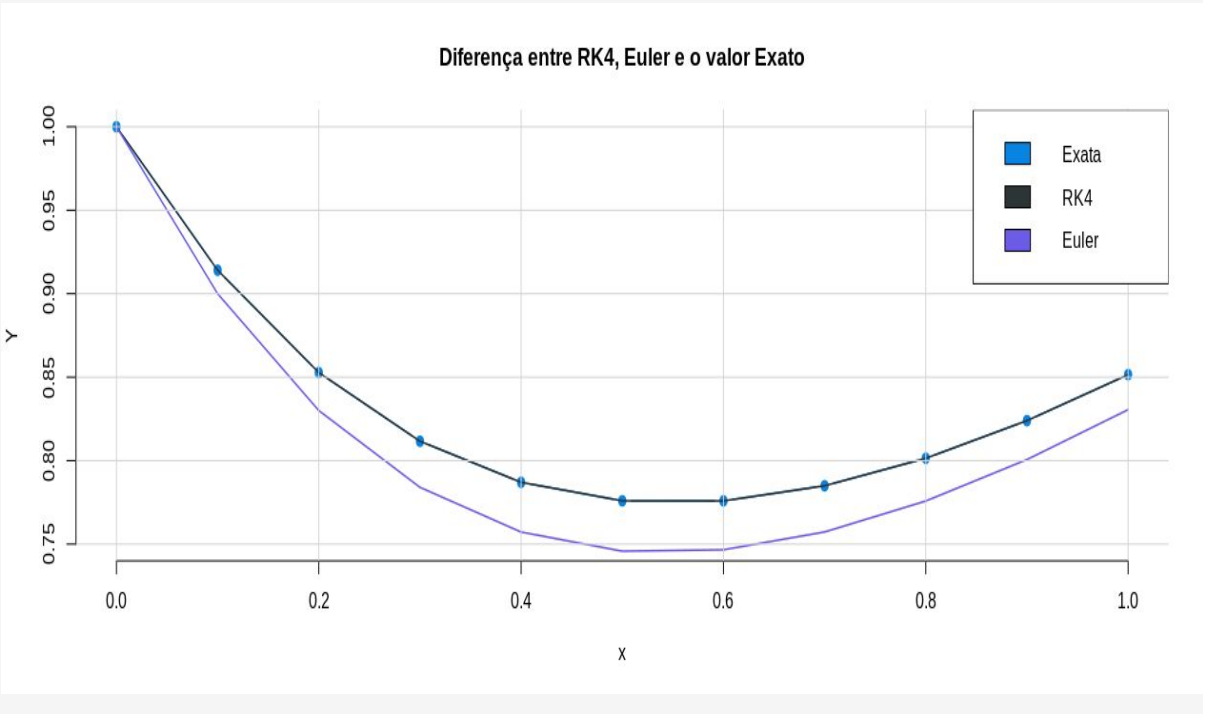
Fiz o plot usando a linguagem R

```
# Tests and analysis --------------------
euler <- odeEuler(func='x-2*y+1',a=0,b=1,y0=1,m=10,VECTOR=1)
rk4 <- odeRK4(func='x-2*y+1',a=0,b=1,y0=1,m=10,VECTOR=1)

x <- euler[1:11, 2]
y_exata <- (1/4)*(3*exp(-2*x)+2*x+1) # Função resolvida analiticamente
y_euler <- euler[1:11, 3]
y_rk4   <- rk4[1:11, 3]

DRACULA_ORCHID= "#2d3436"
WET_ASPHALT = "#34495e"
ORANGE_VILLE = "#e17055"
BRIGHT_YARROW = "#fdcb6e"
MINT_LEAF = "#00b894"
PRUNUS_AVIUM = "#e84393"
ELECTRON_BLUE = "#0984e3"
EXODUS_FRUIT = "#6c5ce7"
cor_exata = ELECTRON_BLUE
cor_rk4   = DRACULA_ORCHID
cor_euler = EXODUS_FRUIT

plot(frame = FALSE,
   ylim=c(0.75,1),
   lwd = 1,
   pch = 19,
   x, y_exata,
   xlab = "x",
   ylab = "Y",
   type = "b",
   col = cor_exata,
   main = "Diferença entre RK4, Euler e o valor Exato"
)
lines(x, y_rk4, col=cor_rk4, lwd = 1)
lines(x, y_euler, col=cor_euler, lwd = 1)
grid(lty="solid")#longdash")
legend(
  "topright",
  c("Exata","RK4","Euler"),
fill                    =                    c(cor_exata,cor_rk4,cor_euler)                    )
```


Diferença entre RK4, Euler e o valor Exato
Exata
RK4
Euler
Y
X
1.00
0.95
0.90
0.85
0.80
0.75
0.0
0.2
0.4
0.6
0.8
1.0

Referências:

1. <https://stackoverflow.com/questions/1743698/evaluate-expression-given-as-a-string>
2. <https://www.r-bloggers.com/a-new-r-trick-for-me-at-least/>
3. <https://stackoverflow.com/questions/19942405/in-r-how-can-i-check-whether-a-list-contains-a-particular-key>
4. <http://web.mit.edu/insong/www/pdf/rpackage_instructions.pdf>
5. <https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/sprintf>
6. <https://www.r-bloggers.com/how-to-write-the-first-for-loop-in-r/>
7. <https://www.guru99.com/r-matrix-tutorial.html>
8. <http://www.datasciencemadesimple.com/rbind-in-r/>
9. <https://www.datamentor.io/r-programming/data-frame/>