

Met. Numéricos Computacionais

Aula 1-2 Projeto em Python

Prof. Dr. Stefan Michael Blawid
sblawid@cin.ufpe.br

Slides baseados em material suplementar para o livro de texto "A primer on scientific programming with Python" de Hans Petter Langtangen

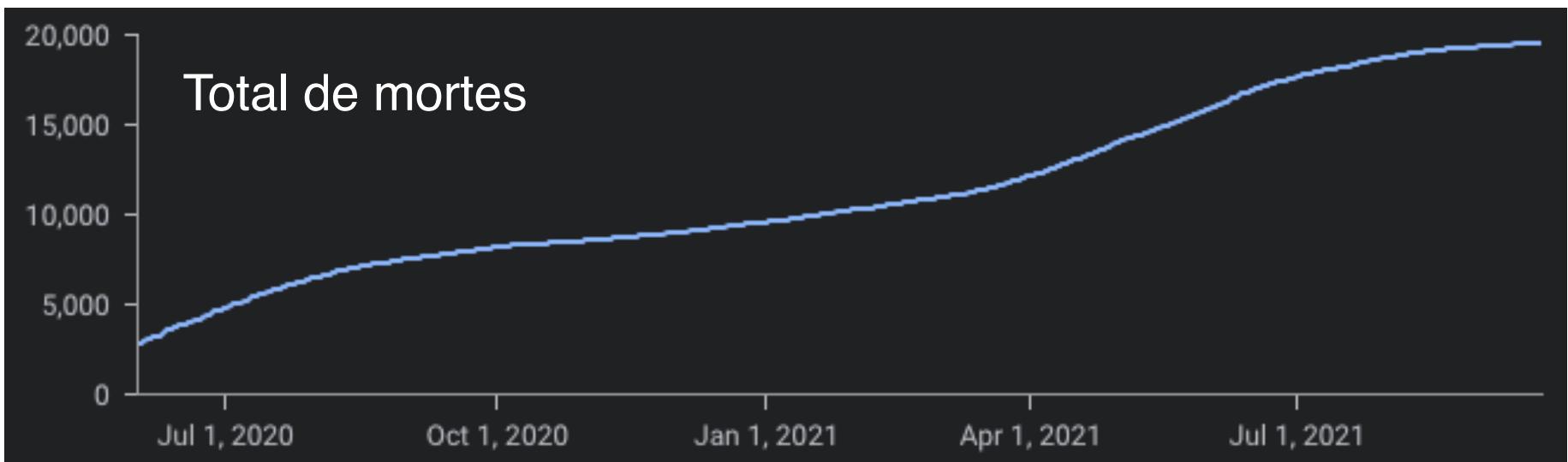
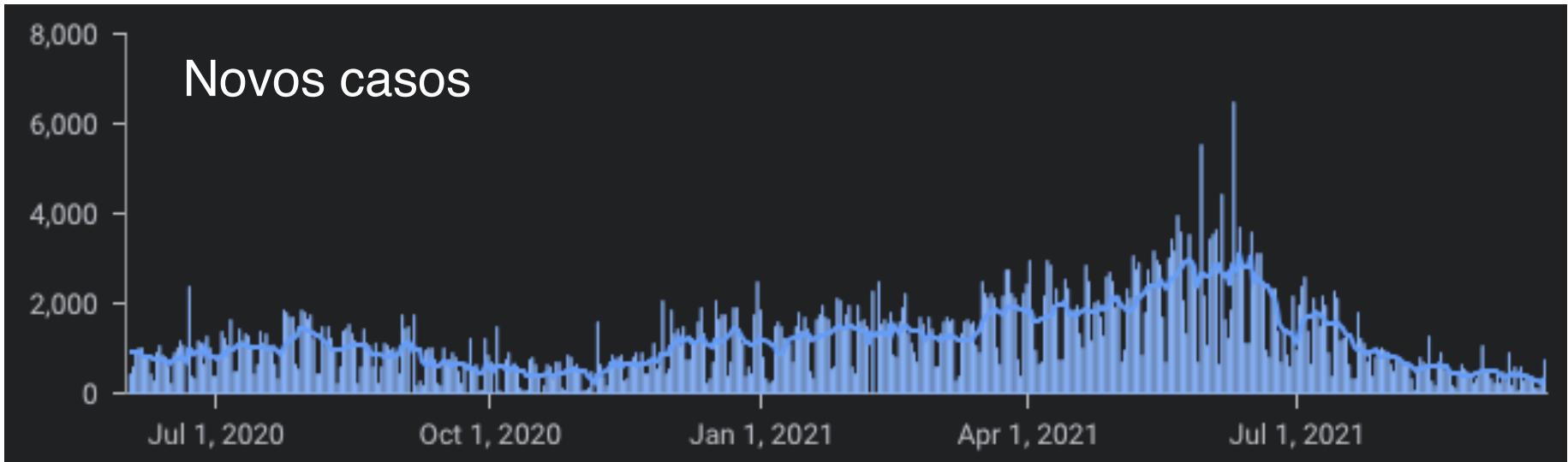
Tópicos

- Descrição do Projeto
- Métodos Numéricos
- Intro Python: Computando com Fórmulas

Tópicos

- Descrição do Projeto
- Métodos Numéricos
- Intro Python: Computando com Fórmulas

Covid-19 em Pernambuco



Tarefa

Sua tarefa é **explicar (01/07/2020 - 31/10/2021) e prever (até 28/02/2022)** a progressão da epidemia da doença coronavírus (Covid-19) no estado de Pernambuco. O sistema de equações diferenciais empregado deve ser baseado em **uma variante apropriada do modelo SIR** para modelar o surto de **Covid-19**. Uma coleção de estudos semelhantes está disponível em nosso "Google Classroom".

O sistema de equações diferenciais deve ser resolvido com o auxílio de módulos implementados na linguagem de programação **Python**. Você pode escrever seus próprios solucionadores, mas também pode optar por usar módulos amplamente disponíveis, como o módulo "integrate" do SciPy.

Você deve documentar seu projeto na forma de um **Notebook Jupyter** e enviar ambos, o arquivo original .ipynb e o arquivo .pdf correspondente.

O Modelo SIR

Um modelo SIR é um modelo epidemiológico que calcula o número teórico de pessoas infectadas por uma doença contagiosa em uma população fechada ao longo do tempo. O nome desta classe de modelos deriva do fato de envolverem equações acopladas relacionando o número de pessoas suscetíveis $S(t)$, o número de pessoas infectadas $I(t)$ e o número de pessoas que recuperaram $R(t)$. Um dos modelos SIR mais simples é o modelo Kermack-McKendrick.

Modelo Kermack-McKendrick

O modelo foi proposto para explicar o rápido aumento e queda no número de pacientes infectados observados em epidemias como a peste (Londres 1665-1666, Bombaim 1906) e cólera (Londres 1865).

$$\frac{dS}{dt} = -\beta SI$$

$$\frac{dI}{dt} = \beta SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- Parâmetro β : Taxa de contato efetiva;
- Parâmetro γ : Taxa média de recuperação

Preste atenção às dimensões! A normalização $s = S/N$, $i = I/N$, $r = R/N$ leva a $\beta \rightarrow \beta/N$, ou seja, β deve ser diminuído conforme $1/N$

Resolvendo o Sistema de ODEs

```
# The SIR model differential equations
def deriv(y, t, N, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt
```

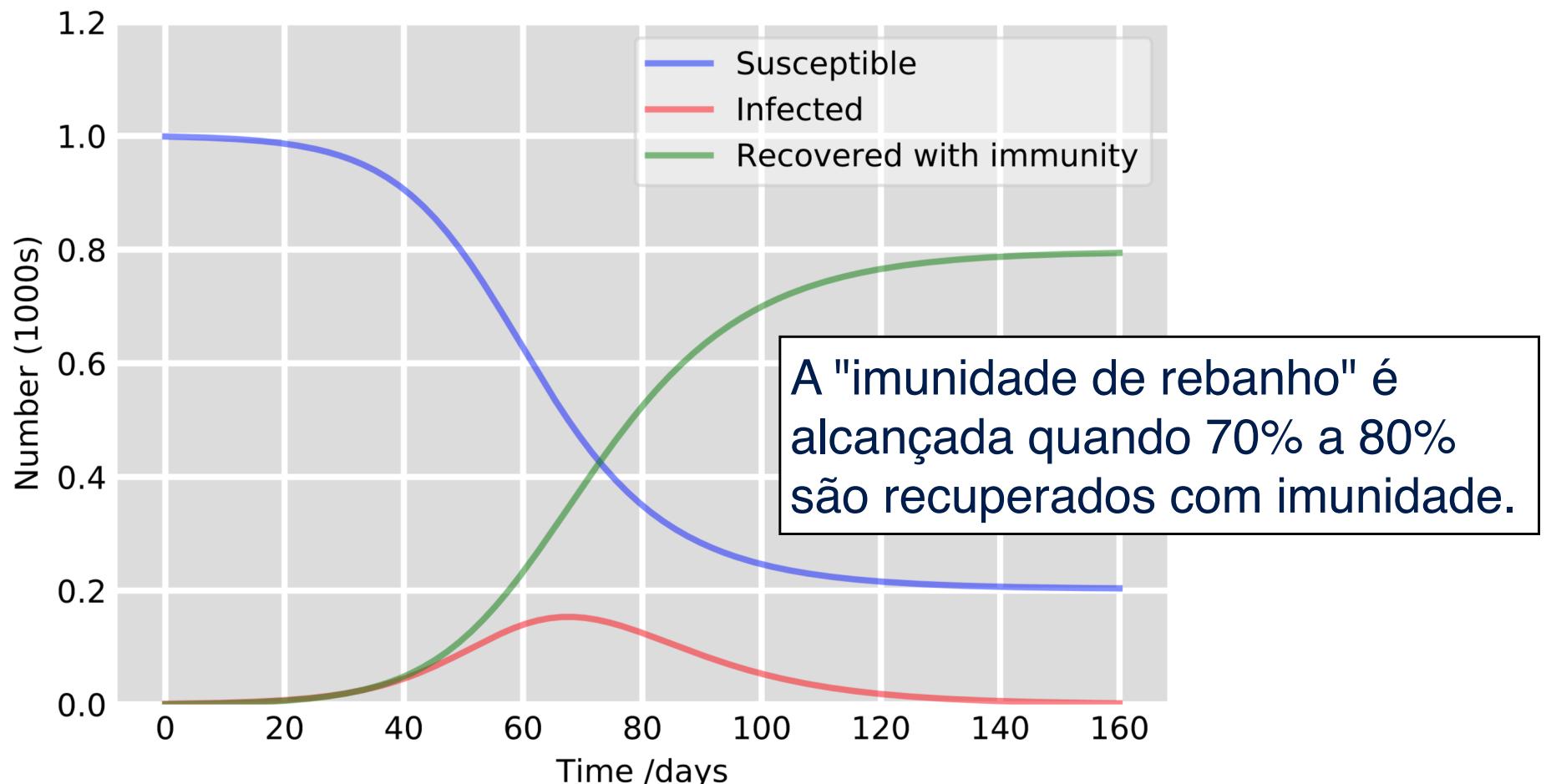
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

```
# Initial conditions vector
y0 = S0, I0, R0
# Integrate the SIR equations over the time grid, t.
ret = odeint(deriv, y0, t, args=(N, beta, gamma))
S, I, R = ret.T
```

Exemplo

o $\beta = 0.2/\text{dia}/N$, $\gamma = 0.1/\text{dia}$, $N = S+I+R = 1000$

o $I(t=0) = 1$, $R(0) = 0 \Rightarrow S(0) = 999$



Observações

O modelo faz várias suposições:

- O tamanho da população é fixo (ou seja, sem nascimentos, mortes devido a doenças ou mortes por causas naturais).
- O período de incubação do agente infeccioso é instantâneo.
- A duração da infecciosidade é igual à duração da doença.
- Também pressupõe uma população completamente homogênea, sem idade, estrutura espacial ou social.

→ Muitas melhorias possíveis!

Comece com uma pergunta

Algumas ideias da turma 2020.3:

- Impacto do Natal e Ano Novo em tempos de pandemia contínua e consequências para o Carnaval
- Boas políticas de saúde durante o período de vacinação da Covid-19
- O efeito da desigualdade social na dinâmica da pandemia de Covid-19
- Simulador Covid-19 interativo
 - ▶ Página da web
 - ▶ Exemplo: <https://shiny.covid-simulator.com/covidsim/>
- Realidades alternativas
- ...

Estenda o modelo

- Modelo SIRASD: Indivíduos suscetíveis-infectados-recuperados para assintomáticos-sintomáticos e mortos
- Modelar novos fenômenos
 - ▶ Disponibilidade limitada de UTIs
 - ▶ Estruturada por idade
 - ▶ Distanciamento Social
 - ▶ Estratégias de desaceleração
 - ▶ Impacto da vacinação
 - ▶ Mutações aleatórias
 - ▶ ...
- Novos fenômenos podem afetar os parâmetros do modelo ou exigir equações diferenciais adicionais

Consulte os artigos disponíveis para inspiração

Discuta e analise

Parâmetros de modelo

- Estimativas de valores

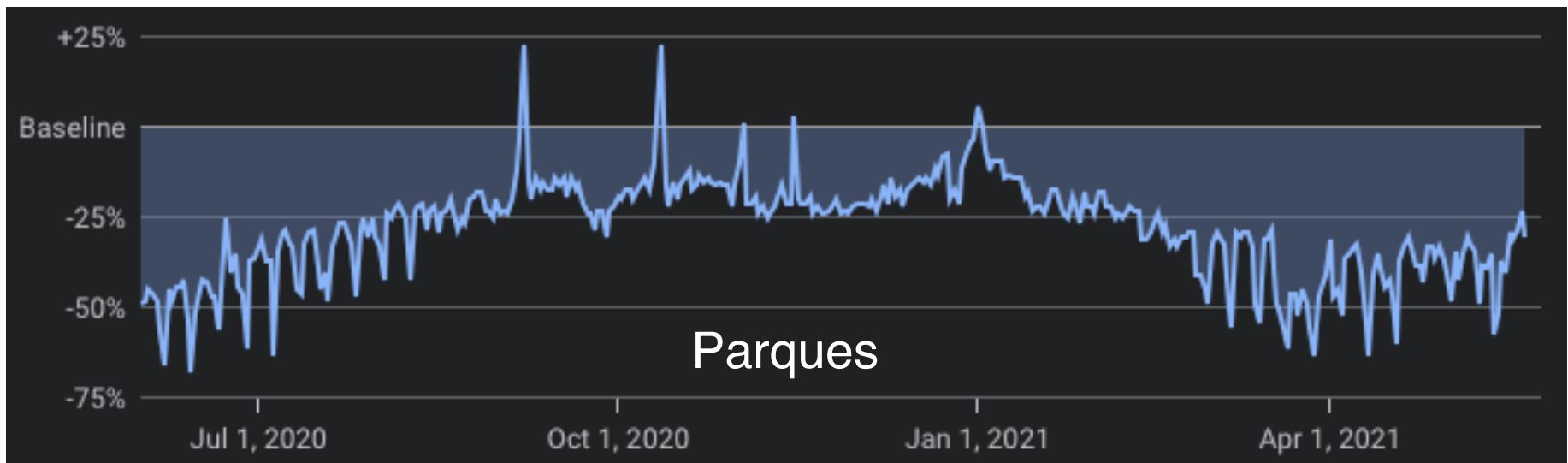
- ▶ Calibrar um modelo mínimo
- ▶ Desafiadora, Faça o melhor que puder
- ▶ Considere as dificuldades

- Os parâmetros podem ser dependentes do tempo!

Consequências:

- Pico de casos ativos
- Imunidade de rebanho
- Estados estáveis do modelo

Outros dados (Google News)



Opcional: Inclui Aspectos Técnicos

- Usar algoritmos de solução diferentes
 - ▶ Runge-Kutta de ordens diferentes
 - ▶ Implementações próprias versus pacotes disponíveis
 - ▶ ...
- Mudar parâmetros numéricos como tamanho dos passos
- Avaliar comparativa (Benchmarking)
- Desenvolver cenários de teste
 - ▶ Soluções analíticas
 - ▶ Quantidades conservadas
- Investigar
 - ▶ Erros numéricos
 - ▶ Estabilidade de longo tempo
- Análise computacional de dados

Quatro partes

- P1: Mapa Mental do seu projeto
 - ▶ Individual, Entrega 31/10/2021
- P2: Exercícios curtos para se familiarizar com a solução numericamente de EDOs com Python
 - ▶ Individual, Entrega 07/11/2021
- Desenvolvimento do projeto em duas etapas (P3 e P4)
 - ▶ Em grupos (3 - 4 alunos, Decisão após a conclusão do P1)
 - ▶ Entrega do relatório parcial (P3): 14/11/2021
 - ▶ Entrega do relatório final (P4): 19/12/2021

Relatório em Formato .ipynb

- Resumo - No início do relatório (0.5 ponto)
- Introdução (1 ponto)
- Métodos (4 pontos)
 - ▶ Equações resolvidas
 - ▶ Métodos numéricos
 - ▶ Extração de Parâmetros
 - ▶ Lista de códigos
- Resultados (4 pontos)
 - ▶ Visualização
 - ▶ Discussão
 - ▶ Avaliação crítica
 - ▶ Impacto/Importância do projeto
- Conclusão - No final do relatório (0.5 ponto)

Tópicos

- Descrição do Projeto
- **Métodos Numéricos**
- Intro Python: Computando com Fórmulas

Vamos falar sobre ...

Solução Numérica de Equações diferenciais.

- Grande tópico; vamos cobrir em partes
- Não é difícil, muito útil

Tipos de Equações Diferenciais

- Ordem = grau de derivado

- Primeira ordem ODE:

$$\frac{dy}{dt} = f(t, y)$$

- A função derivada ou força $f(t, y)$ no RHS é arbitrária, por exemplo:

$$f(t, y) = -3t^2y + t^9 + y^7$$

- Uma segunda ordem ODE (por exemplo, a lei de Newton):

$$m \frac{d^2y}{dt^2} = -3t^2 \left(\frac{dy}{dt} \right)^4 + t^9 y(t)$$

- Variável independente t = tempo

- Variável dependente $y(t)$ = posição

Mais palavras: Ordinaria, Parcial, Inicial, Limiar

- Equação Diferencial Ordinária (= ODE) contém apenas uma variável independente.
- Equação Diferencial Parcial (= PDE) contém várias variáveis independentes
- A solução geral de um ODE de ordem n contém n constantes arbitrárias. Seus valores são fixados pelas condições iniciais.
- Uma maneira diferente de selecionar uma solução específica de equações diferenciais é definir condições de contorno que limitam a solução a ter valores fixos nos limites do espaço da solução. Problemas desse tipo são chamados de problemas de autovalores.

ODEs lineares e não lineares

- ODE linear:

$$\frac{dy}{dt} = g^3(t)y(t)$$

- ODE não linear:

$$\frac{dy}{dt} = \lambda y(t) - \lambda^2 y^2(t)$$

- Parte da liberação da ciência computacional é que não estamos mais limitados a resolver equações lineares.
- Uma propriedade importante das equações lineares é a lei da superposição linear que nos permite adicionar soluções para formar novas.

Forma padrão de ODEs

Uma forma padrão para ODEs é expressar ODEs de qualquer ordem como N ODEs de primeira ordem simultâneos.

$$\begin{aligned}
 \frac{dy^{(0)}}{dt} &= f^{(0)}(t, y^{(i)}) \\
 \frac{dy^{(1)}}{dt} &= f^{(1)}(t, y^{(i)}) \\
 &\vdots = \vdots \\
 \frac{dy^{(N-1)}}{dt} &= f^{(N-1)}(t, y^{(i)}) \qquad \Rightarrow \qquad \frac{d\vec{y}}{dt} = \vec{f}(t, \vec{y})
 \end{aligned}$$

Essas equações podem ser expressas de forma mais sucinta pelo uso dos vetores N-dimensionais.

$$\vec{y} = \begin{pmatrix} y^{(0)}(t) \\ y^{(1)}(t) \\ \ddots \\ y^{(N-1)}(t) \end{pmatrix}, \quad \vec{f} = \begin{pmatrix} f^{(0)}(t, y) \\ f^{(1)}(t, y) \\ \ddots \\ f^{(N-1)}(t, y) \end{pmatrix}$$

Forma dinâmica de Lei de Newton

Exemplo importante:

$$\frac{d^2x}{dt^2} = \frac{1}{m} F\left(t, x, \frac{dx}{dt}\right)$$

A regra para converter para a forma padrão é que o RHS não pode conter quaisquer derivados explícitos, embora componentes individuais de $y^{(i)}$ possam representar derivativos. Definir:

$$y^{(0)}(t) := x(t), \quad y^{(1)}(t) := \frac{dx}{dt}$$

O ODE de segunda ordem agora se torna dois ODEs de primeira ordem simultâneos:

$$\frac{dy^{(0)}}{dt} = y^{(1)}(t), \quad \frac{dy^{(1)}}{dt} = \frac{1}{m} F(t, y^{(0)}, y^{(1)})$$

Em outras palavras:

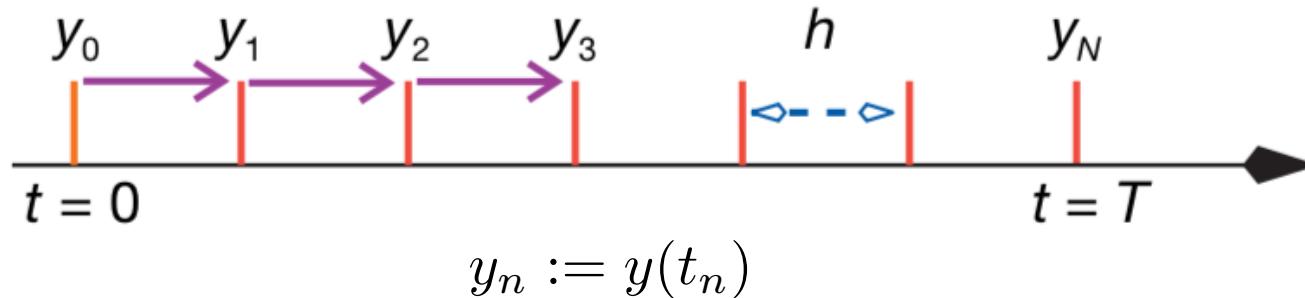
$$f^{(0)} = y^{(1)}(t), \quad f^{(1)} = \frac{1}{m} F(t, y^{(0)}, y^{(1)})$$

Algoritmos ODE

- "time" stepping (integração) =

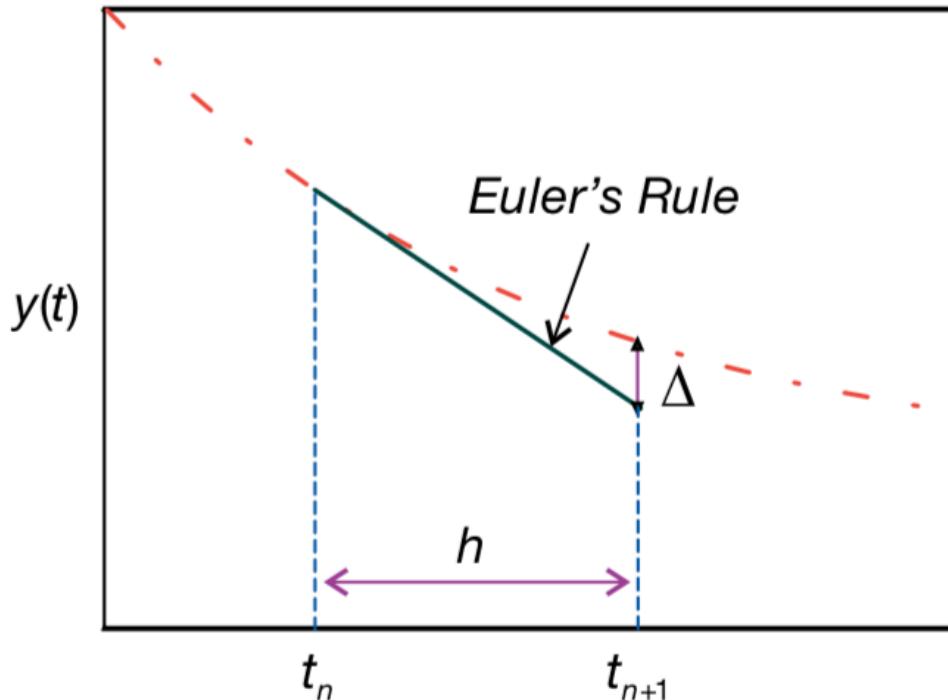


= saltar =



- Métodos, que requerem um h muito pequeno para integração, são propensos a erros de arredondamento

Algoritmo muito simples: Euler



$$y_{n+1} \approx y_n + h f(t_n, y_n)$$

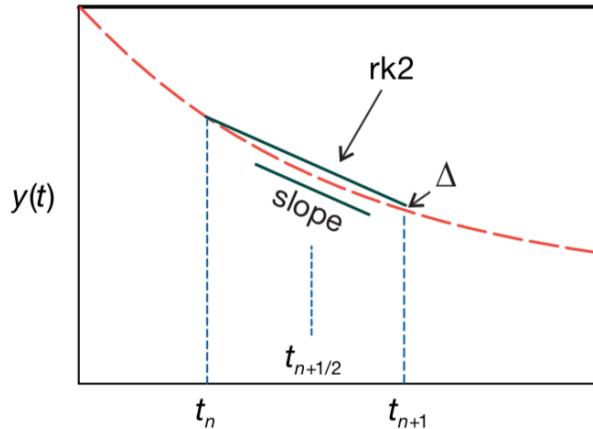
- Geralmente y e f são vetores
- Vamos cobrir mais detalhes em outra aula

Melhor Algoritmo: Runge-Kutta

Baseado na integral formal (exata) do ODE:

$$y_{n+1} \approx y_n + \int_{t_n}^{t_{n+1}} f(t, y) dt$$

Use forma aproximada para $f(t,y)$ (!)



$$\begin{aligned}
 f(t, y) &\approx f(t_{n+1/2}, y_{n+1/2}) \\
 &+ (t - t_{n+1/2}) \left. \frac{df}{dt} \right|_{t_{n+1/2}} \\
 &+ \mathcal{O}(h^2)
 \end{aligned}$$

A integral sobre o segundo termo desaparece. Portanto:

$$y_{n+1} \approx y_n + h f(t_{n+1/2}, y_{n+1/2}) + \mathcal{O}(h^3)$$

Use o algoritmo de Euler para determinar $y_{n+1/2}$. Em forma de vetor:

$$\vec{y}_{n+1} \approx \vec{y}_n + \vec{k}_2,$$

$$\vec{k}_2 = h \vec{f} \left(t_n + h/2, \vec{y}_n + \vec{k}_1/2 \right), \quad \vec{k}_1 = h \vec{f}(t_n, \vec{y}_n)$$

Exemplo

Equação de movimento: $m \frac{d^2x}{dt^2} = F_k(x)$

As equações rk2 lidas para n=0:

$$\begin{aligned} y_1^{(0)} &= y_0^{(0)} + h f^{(0)} \left(\frac{h}{2}, y_0^{(0)} + \frac{h}{2} f^{(0)}, y_0^{(1)} + \frac{h}{2} f^{(1)} \right) \\ &= x_0 + h \left(y_0^{(1)} + \frac{h}{2} f^{(1)} \right) = x_0 + h \left[v_0 + \frac{h}{2m} F_k(x_0) \right] \end{aligned}$$

$$y_1^{(1)} = v_0 + \frac{h}{m} F_k \left(y_0^{(0)} + \frac{h}{2} f^{(0)} \right) = v_0 + \frac{h}{m} F_k \left(x_0 + \frac{h}{2} v_0 \right)$$

Aqui usamos as condições iniciais: $y_0^{(0)} = x_0, \quad y_0^{(1)} = v_0$

rk4 requer a avaliação de quatro declives intermediários, e estes são aproximados com o algoritmo de Euler (Press et al., 1994)

$$\vec{y}_{n+1} \approx \vec{y}_n + \frac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4),$$

$$\vec{k}_1 = h\vec{f}(t_n, \vec{y}_n), \quad \vec{k}_2 = h\vec{f}\left(t_n + \frac{h}{2}, \vec{y}_n + \frac{\vec{k}_1}{2}\right)$$

$$\vec{k}_3 = h\vec{f}\left(t_n + \frac{h}{2}, \vec{y}_n + \frac{\vec{k}_2}{2}\right), \quad \vec{k}_4 = h\vec{f}(t_n + h, \vec{y}_n + \vec{k}_3)$$

Tópicos

- Descrição do Projeto
- Métodos Numéricos
- **Intro Python: Computando com Fórmulas**
 - **Python-P1.ipynb**

Avaliar Fórmulas

- Todo mundo entende o problema
- Muitos conceitos fundamentais são introduzidos
 - Variáveis
 - Expressões aritméticas
 - Objetos
 - Impressão de texto e números

Aritmética é fundamental ...

Um físico, um biólogo e um matemático estavam em um café quando do outro lado da rua, duas pessoas entraram em uma casa. Momentos depois, três pessoas saíram. O físico disse, "Hmm, que deve ser um erro de medição". O biólogo se perguntou: "deve ser a reprodução!" E o matemático disse: "se alguém entrar na casa, estará vazio novamente".

Avaliando uma Fórmula Matemática

Altura de uma bola em movimento vertical:

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

- y é a altura (posição) como função do tempo t
- v_0 é a velocidade inicial em $t = 0$
- g é a aceleração da gravidade

Tarefa: dado v_0 , g e t , calcula y .

Use uma Calculadora?

- Um programa é muito mais poderoso!
- O que é um programa?
 - Uma seqüência de instruções para o computador, escrita em uma linguagem de programação, algo como o inglês, mas muito mais simples - e muito mais rigoroso.
- Este curso ensina a linguagem Python.
- Nosso primeiro exemplo de programa: Avalie $y(t)$ para $v=5$, $g=9.81$ e $t=0.6$

O programa completo do Python:

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Feb 13 16:08:30 2018
5
6 @author: sblawid
7 """
8
9 print 5*0.6 - 0.5*9.81*0.6**2
```

A Sintaxe printf

- Dá grande exatidão na formatação de texto com números
- A saída dos cálculos geralmente contém texto e números, por exemplo, Em $t=0.6\text{s}$, y é 1.23m .
- Queremos controlar a formatação de números: número de decimais, estilo: 0.6 vs 6E-01 ou 6.0e-01. A chamada formatação printf é útil para este propósito:

```
9 t = 0.6; y = 1.2342
10 print 'At t=%gs, the height of the ball is %.2fm' %(t, y)
```

- O formato printf possui slots onde as variáveis listadas no final são:
 $\%g \leftarrow t$, $\%.2f \leftarrow y$

Exemplos

<code>%g</code>	formatação mais compacta de um número real
<code>%f</code>	notação decimal (-34.674)
<code>%10.3f</code>	notação decimal, 3 decimais, largura de campo 10
<code>%.3f</code>	decimal notation, 3 decimals, minimum width
<code>%e ou %E</code>	notação científica (1.42e-02 ou 1.42E-02)
<code>%9.2e</code>	notação científica, 2 decimais, largura do campo 9
<code>%d</code>	número inteiro
<code>%5d</code>	inteiro em um campo de largura 5 caracteres
<code>%s</code>	string (texto)
<code>%20-s</code>	string, largura do campo 20, ajustada à esquerda
<code>%%</code>	o sinal percentual % próprio

Usando a Formatação printf

- As strings de quebra de linha tripla ("'''") podem ser usadas para a saída de várias linhas, e aqui combinamos essa string com a formatação do printf:

```
 9 # Program for computing the height of a ball in vertical motion
10 vo = 5          # initial velocity
11 g = 9.81        # acceleration of gravity
12 t = 0.6         # time
13 y = vo*t - 0.5*g*t**2    # vertical position
14
15 print """
16 At t=%fs a ball with
17 initial velocity v0=%.3em/s
18 is located at the height %.2fm
19 """ % (t, vo, y)
```

Blanks para formatar bem o Código

- Blanks (espaços em branco) pode ou não ser importante nos programas Python. Essas declarações são equivalentes (os espaços em branco não importam):

```
8
9 vo=3
10 vo = 3
11 vo= 3
12 vo = 3
13
```

- Aqui os espaços em branco são importantes:

```
9
10 counter = 1
11 while counter <= 4:
12     counter = counter + 1      # correct (4 leading blanks)
13
14 while counter <= 4:
15 counter = counter + 1        # invalid syntax
16
```

Avaliando mais uma Fórmula

- Conversão de temperatura: Dado C como temperatura em graus Celsius, calcula os graus Fahrenheit correspondentes F

$$F = \frac{9}{5}C + 32$$

- Programa:

```
9
10 C = 21
11 F = (9/5)*C + 32
12 print F
13
14
```

- Execução:

```
In [1]: runfile('/Users/sblawid/MyDocs/02_Places/01_UnB/03_Aulas/CCcP/02_Python/
Exemplos/temperature.py', wdir='/Users/sblawid/MyDocs/02_Places/01_UnB/03_Aulas/
CCcP/02_Python/Exemplos')
53
```

Nós sempre devemos verificar ...

... que um novo programa calcula a resposta certa.

Usando uma calculadora: $9/5$ vezes 21 mais 32 é $69,8$, e não 53 .

O Erro é causado por ...

... (não intencional) divisão inteira

- $9/5$ não é 1.8 mas 1 na maioria das linguagens de computador (!)
- Se a e b são inteiros, a/b implica divisão inteira: o maior inteiro c tal que $cb \leq a$
- Exemplos: $1/5 = 0$, $2/5 = 0$, $7/5 = 1$, $12 / 5 = 2$
- Em matemática, $9/5$ é um número real (1.8) - isso é chamado de divisão de flutuadores em Python e é a divisão que queremos
- Um dos operandos (a ou b) em a / b deve ser um número real ("flutuador") para obter divisão de flutuadores
- Um flutuador em Python tem um ponto (ou decimais): 9.0 ou $9.$ são flutuadores
- Nenhum ponto implica número inteiro: 9 é um número inteiro
- $9.0/5$ produz 1.8 , $9/5.$ produz 1.8 mas $9/5$ rendimentos 1

Tudo em Python é um Objeto

As variáveis referem-se a objetos:

```
8
9 a=5 # a refers to an integer (int) object
10 b=9 # b refers to an integer (int) object
11 c=9.0 # c refers to a real number (float) object
12 d=b/a # d refers to an int/int => int object
13 e=c/a # e refers to float/int => float object
14 s = 'b/a=%g' % (b/a) # s is a string/text (str) object
15
```

Podemos converter entre tipos de objeto:

```
8
9 a = 3                      # a is int
10 b = float(a)                # b is float 3.0
11 c = 3.9                     # c is float
12 d = int(c)                  # d is int 3
13 d = round(c)                # d is float 4.0
14 d = int(round(c))           # d is int 4
15 d = str(c)                  # d is str '3.9'
16 e = '-4.2'                   # e is str
17 f = float(e)                 # f is float -4.2
18
```

As Expressões Aritméticas ...

... são avaliados como você aprendeu em matemática

- Exemplo: $5/9 + 2a^4/2$, em Python escrito como `5/9 + 2*a**4/2`
- As mesmas regras que em matemática: prosseguem termo por termo (adições / subtrações) a partir da esquerda, calculem potências em primeiro lugar, em seguida, multiplicação e divisão, em cada termo
 - $r1 = 5/9 (= 0)$
 - $r2 = a^{**4}$
 - $r3 = 2 * r2$
 - $r4 = r3/2$
 - $r5 = r1 + r4$
- Use parênteses para substituir essas regras padrão - ou use parênteses para indicar explicitamente como as regras funcionam:
 - `(5/9) + (2*(a**4))/2`

Funções Matemáticas Padrão ...

... são encontrados no módulo de matemática

- Exemplos: Se precisarmos calcular $\sin(x)$, $\cos(x)$, $\ln(x)$, etc. em um programa?
- Tais funções estão disponíveis no módulo `math` de Python
- Em geral: muita funcionalidade útil no Python está disponível em módulos - mas os módulos devem ser importados em nossos programas

Calcule $\sqrt{2}$ usando a função `sqrt` no módulo de matemática:

```
8
9 import math
10 r = math.sqrt(2)
11 print r
12
```

```
8
9 from math import sqrt
10 r = sqrt(2)
11 print r
12
```

```
8
9 from math import *
10 r = sqrt(2)
11 print r
12
```

Outro Exemplo

Avalie $Q = \sin(x) \cos(x) + 4 \ln(x)$ para $x = 1.2$.

```
8
9 from math import sin, cos, log
10 x = 1.2
11 Q = sin(x)*cos(x) + 4*log(x)    # log is ln (base e)
12 print Q
13
```

Computadores têm Aritmética Inexata ...

... por causa de erros de arredondamento

Deixe-nos calcular $1/49$ vezes 49 e $1/51$ vezes 51:

```
8
9 v1 = 1/49.0*49
10 v2 = 1/51.0*51
11 print '%.16f %.16f' % (v1, v2)
12
```

```
In [9]: runfile('/Users/sblawid/MyDocs/
Exemplos/rounding.py', wdir='/Users/sbl
02_Python/Exemplos')
0.9999999999999999 1.0000000000000000
```

- A maioria dos números reais são representados inexatamente em um computador (17 dígitos)
- Nem $1/49$ nem $1/51$ são representados exatamente, o erro é tipicamente 10^{-16}
- Às vezes, tais erros pequenos se propagam para a resposta final, às vezes não, e às vezes os pequenos erros se acumulam através de muitas operações matemáticas
- Lição aprendida: números reais em um computador e os resultados de cálculos matemáticos são apenas aproximados

Teste que um Cálculo é correto

O que está impresso?

```
8  
9 a = 1; b = 2;  
10 computed = a + b  
11 expected = 3  
12 correct = computed == expected  
13 print 'Correct:', correct  
14
```

Mude para $a = 0.1$ e $b = 0.2$ (esperado = 0.3). O que está agora impresso? Por quê? Como a comparação pode ser realizada?

Use Teste de Igualdade com Tolerância!

```
8
9 a = 0.1; b = 0.2;
10 computed = a + b
11 expected = 0.3
12 diff = abs(computed - expected)
13 tol = 1e-15
14 correct = diff < tol
15 print 'Correct:', correct
16
```

Outro Exemplo

A função $\sinh(x)$ é definida como:

$$\sinh(x) = \frac{1}{2} [\exp(x) - \exp(-x)]$$

Podemos avaliar esta função de três maneiras:

- `math.sinh`
- combinação de dois
`math.exp`
- combinação de dois
`math.e`

```
8
9 from math import sinh, exp, e, pi
10 x = 2*pi
11 r1 = sinh(x)
12 r2 = 0.5*(exp(x) - exp(-x))
13 r3 = 0.5*(e**x - e**(-x))
14 print '%.16f %.16f %.16f' % (r1, r2, r3)
15
```

```
In [13]: runfile('/Users/sblawid/MyDocs/02_Places/01_UnB/03_Aula
Exemplos/sinhyper.py', wdir='/Users/sblawid/MyDocs/02_Places/01_
02_Python/Exemplos')
267.7448940410164369 267.7448940410164369 267.7448940410163232
```

Suporte para Números Complexos

$2 + 3i$ em matemática é escrito como $2 + 3j$ em Python

```
8
9 a=-2
10 b=0.5
11 s = complex(a,b)      # make complex from variables
12 print s
13 w=4.5+3j;
14 print s*w              # complex*complex
15 print s/w               # complex/complex
16 print s.real
17 print s.imag
18
```

```
In [7]: runfile('/Users/sblawid/My
Exemplos/temp.py', wdir='/Users/sb
02_Python/Exemplos')
(-2+0.5j)
(-10.5-3.75j)
(-0.25641025641+0.282051282051j)
-2.0
0.5
```

Computação Simbólica

- Computação numérica: computação com números
- Computação simbólica: trabalhe com fórmulas (como em matemática tradicional)

```
9 from sympy import *
10 t, vo, g = symbols('t vo g')
11 y = vo*t - Rational(1,2)*g*t**2
12 dydt = diff(y,t)
13 print 'velocity:', dydt
14 print 'acelleration:', diff(y,t,t)
15 print 'position:', integrate(dydt, t)
16
```

```
In [9]: runfile('/Users/sbla/Exemplos/symbmat.py', wdir='02_Python/Exemplos')
velocity: -g*t + vo
acelleration: -g
position: -g*t**2/2 + t*vo
```

SymPy pode fazer Matemáticas Tradicionais

```
8
9 from sympy import *
10 t, vo, g = symbols('t vo g')
11 y = vo*t - Rational(1,2)*g*t**2
12 roots = solve(y, t)    # solve y=0 wrt t
13 print roots
14 x, y = symbols('x y')
15 f = -sin(x)*sin(y) + cos(x)*cos(y)
16 print simplify(f)
17 print expand(sin(x+y), trig=True)    # requires a trigonometric hint
18
```

```
In [10]: runfile('/Users/sblawid/Exemplos/symbmat2.py', wdir='/Users/sblawid/02_Python/Exemplos')
[0, 2*vo/g]
cos(x + y)
sin(x)*cos(y) + sin(y)*cos(x)
```

Resumo

- Os programas devem ser precisos!
- Variáveis são nomes para objetos
- Conhecemos diferentes tipos de objetos: `int`, `float`, `str`
- Escolha nomes de variáveis próximos aos símbolos matemáticos no problema que está sendo resolvido
- Operações aritméticas em Python: termo por termo (+/-) da esquerda para a direita, potência antes de * e / - como em matemática; use parênteses quando houver alguma dúvida
- Cuidado com a divisão inteira não intencional!
- Funções matemáticas devem ser importadas do módulo `math`
- Use a sintaxe `printf` para obter o controle total da saída de texto e números!
- Termos importantes: objeto, variável, algoritmo, declaração, atribuição, implementação, verificação, depuração

A Programação é desafiadora

- Você acha que sabe quando pode aprender, tem mais certeza quando pode escrever, ainda mais quando pode ensinar, mas é certo quando pode programar
- Dentro de um computador, a linguagem natural não é natural
- Para entender um programa você deve se tornar a máquina e o programa

Alan Perlis, computer scientist, 1922-1990

Problema: Jogando uma Bola

Jogamos uma bola com a velocidade v_0 , num ângulo θ com a horizontal, a partir do ponto ($x = 0$, $y = y_0$). A trajetória da bola é uma parábola (negligenciamos a resistência do ar):

$$y = x \tan(\theta) - \frac{1}{2v_0^2} \frac{gx^2}{\cos^2(\theta)} + y_0$$

- Tarefas do programa:
 - inicializar dados de entrada (v_0 , g , θ , y_0)
 - importação de **math**
 - calcular y
- Nós damos x , y e y_0 em m, $g = 9,81$ m/s², v_0 em km/h e θ em graus - isso requer conversão de v_0 para m/s e θ para radianos

Exemplo

```
In [43]: runfile('/Users/sblawid/LocDoc/02_Places/01_UnB/03_Aulas/CCcP/  
201901/02_Python/21_aula/PROBLEM.py', wdir='/Users/sblawid/LocDoc/02_Places/  
01_UnB/03_Aulas/CCcP/201901/02_Python/21_aula')  
v0 = 15.0km/h  
theta = 60degrees  
y0 = 1.0m  
x = 0.5m  
  
y = 1.6m
```

Solução

```
9
10 g = 9.81      # m/s**2
11 v0 = 15        # km/h
12 theta = 60     # degrees
13 x = 0.5
14 y0 = 1
15
16 print """v0 = %.1f km/h
17 theta = %d degrees
18 y0 = %.1f m
19 x = %.1f m"""\ % (v0, theta, y0, x)
20
21 # convert v0 to m/s and theta to radians:
22 v0 = v0/3.6
23 from math import pi, tan, cos
24 theta = theta*pi/180
25
26 y = x*tan(theta) - 1/(2*v0**2)*g*x**2/((cos(theta))**2) + y0
27 print 'y = %.1f m' % y
28
```



»Wissen schafft Brücken.«

