Desenvolvimento de um software científico para a modelagem e simulação computacional

Dissertação de Mestrado

Mestrado em Ciência da Computação

Brenno Lemos Melquiades dos Santos

Orientador: Prof. Alexandre B. Pigozzo

25 de março de 2024

Universidade Federal de São João del-Rei

UFSJ

Sumário

1 Introdução

Introdução

- O desenvolvimento de modelos computacionais requer um conjunto de etapas: estudo do problema, formulação de hipóteses, construção, implementação e simulação do modelo.
- Uma das etapas mais desafiadoras é a implementação.
  - Requer o conhecimento de programação, estrutura de dados, bibliotecas, etc;
  - Um erro na implementação pode comprometer todo o trabalho.
- Questão científica:
  - É possível que ferramentas de software automatizem etapas do processo de modelagem computacional?

- Para responder a questão anterior, foi desenvolvido um software para auxiliar a implementação e simulação de modelos de Equações Diferenciais Ordinárias (EDOs) com o objetivo de automatizar certas etapas do processo de modelagem.
- A partir de uma representação visual de um modelo, o software é capaz de gerar o código que implementa o modelo, simulá-lo e até exportar gráficos com os resultados.

UFSJ

Sumário

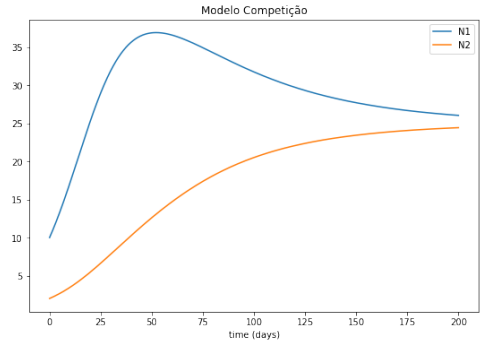
2 Referencial teórico

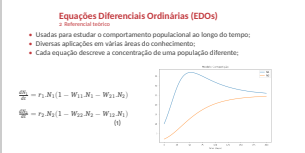
Equações Diferenciais Ordinárias (EDOs)

- Usadas para estudar o comportamento populacional ao longo do tempo;
- Diversas aplicações em várias áreas do conhecimento;
- Cada equação descreve a concentração de uma população diferente;

$$\frac{dN_1}{dt} = r_1.N_1(1 - W_{11}.N_1 - W_{21}.N_2)$$

$$\frac{dN_2}{dt} = r_2.N_2(1 - W_{22}.N_2 - W_{12}.N_1) \quad (1)$$





1. Exemplos de uso de EDOs: medicina, neurociência, estudo do câncer.

EDO — Modelo Predador-Presa

2 Referencial teórico

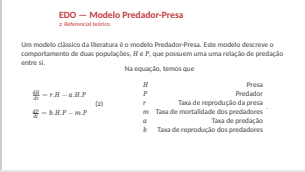
Um modelo clássico da literatura é o modelo Predador-Presa. Este modelo descreve o comportamento de duas populações, H e P, que possuem uma uma relação de predação entre si.

$$\begin{aligned} \frac{dH}{dt} &= r.H - a.H.P \\ \frac{dP}{dt} &= b.H.P - m.P \end{aligned} \tag{2}$$

Na equação, temos que

H	Presa
P	Predador
r	Taxa de reprodução da presa
m	Taxa de mortalidade dos predadores
a	Taxa de predação
b	Taxa de reprodução dos predadores

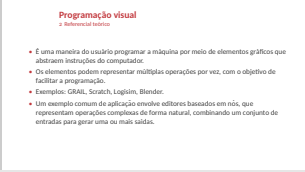
.



1. Neste modelo, temos os seguintes processos sendo modelados:
   - Reprodução das presas ($r.H$);
   - Predação ($a.H.P$);
   - Reprodução dos predadores ($b.H.P$);
   - Morte dos predadores ($m.P$).
2. Termos de replicação, predação e morte como os vistos neste modelo são muito comuns. Estes termos são construídos com base no princípio da Lei de Ação de Massas, que diz que "O número de interações entre duas partículas depende da concentração de ambas."

Programação visual

- É uma maneira do usuário programar a máquina por meio de elementos gráficos que abstraem instruções do computador.
- Os elementos podem representar múltiplas operações por vez, com o objetivo de facilitar a programação.
- Exemplos: GRAIL, Scratch, Logisim, Blender.
- Um exemplo comum de aplicação envolve editores baseados em nós, que representam operações complexas de forma natural, combinando um conjunto de entradas para gerar uma ou mais saídas.



1. A linguagem GRAIL foi desenvolvida junto a uma tela sensível ao toque e uma *stylus*

- GRAIL — 1968:

Geração de código

- Recebe como entrada uma Representação Intermediária (RI) e gera como saída um código na linguagem alvo (por exemplo, Python).
- Aplicações da RI:
  - Separar o *front-end* do compilador do *back-end*;
  - Permitir que sejam realizadas otimizações independente de máquina ou otimizações independente da linguagem alvo;
  - Facilitar a tradução e geração do código alvo.
- Exemplo: LLVM IR, usada originalmente para compilar códigos em C/C++ pelo Clang, e agora também usada na compilação de códigos em Rust.

Geração de código baseada em *templates*

- Um *template* é um esqueleto (uma estrutura) que serve de referência para todo o processo de geração de código.
- Em sua essência, *templates* são arquivos com marcadores especiais que podem ser substituídos por outros valores dinamicamente.
- A presença de estruturas de controle de fluxo como condicionais e laços de repetição permitem a escrita de *templates* legíveis e de fácil manutenção.

Geração de código baseada em *templates*


```
def system(t: np.float64, y: np.ndarray, *constants) -> np.ndarray:
    {% for arg in populations -%}
        {{- arg.name }}, {%- endfor %} = y

    {%- if constants %}
    {% for arg in constants -%}
        {{- arg.name }}, {%- endfor %} = constants
    {% endif -%}

    {% for pop in populations %}
    {%- set comp = model.arguments[equations[pop.name].argument] %}
    d{{ pop.name }}_dt = {{ display_composite(comp) }}
    {%- endfor %}

    # ...
```

UFSJ

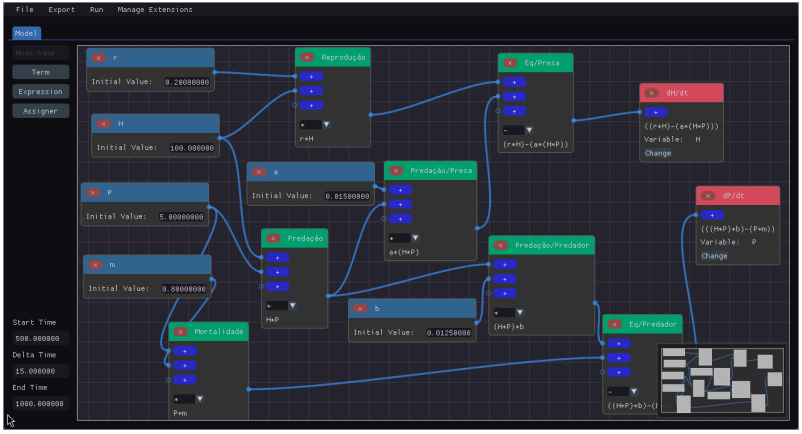
Sumário

3 Software para modelagem e simulação

Visão geral do software

3 Software para modelagem e simulação

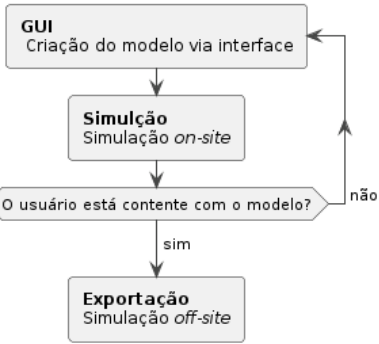
- O software desenvolvido apresenta uma GUI contendo um editor baseado em nós.
- Os nós representam partes das equações, como parâmetros, populações e expressões



Visão geral do software

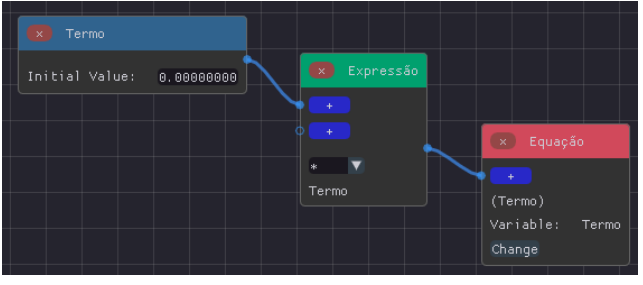
3 Software para modelagem e simulação

Após construído o modelo, o usuário poderá simulá-lo diretamente pela GUI, exportar um PDF dos resultados, ou exportar um código de Python equivalente para o modelo desenhado



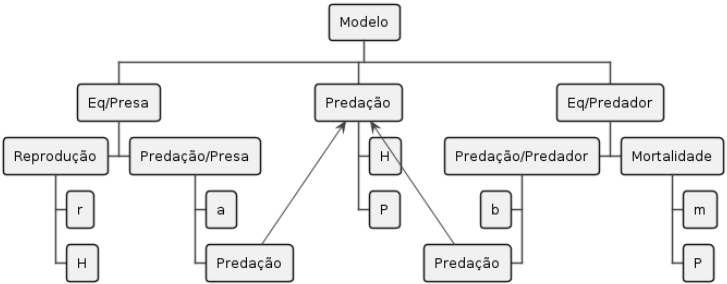
Representação visual

Tipo de nó	Usado para representar	Exemplo
Termo	Variáveis, parâmetros e constantes.	H, P, a, b, r, m
Expressão	Expressões matemáticas que compõem as equações.	$r.H, a.H.P, r.H - a.H.P$
Equação	O lado direito de uma EDO.	$\frac{dH}{dt} = \ldots$



Árvore de expressões

- Os nós de Expressão e de Equação carregam consigo árvores de expressões para representar as expressões que constroem.
- As árvores são copiadas e enviadas para os nós relacionados quando algum destes eventos ocorrem na interface:
  - Mudança de sinal;
  - Alteração de nome de nós ‘Termos’;
  - Adições ou remoções de ligações;



Representação intermediária

- Uma representação intermediária foi desenvolvida para simplificar as transformações dos modelos.
- A GUI apenas precisa transformar seus modelos na RI. A partir disto, a biblioteca de RI pode gerar códigos em Python e gerar a representação para armazenamento em disco.
- A RI foi projetada para ser humanamente legível, possibilitando a recuperação do modelo construído na ausência do software.

Geração de código, simulação interativa e exportação de resultados

- Os modelos gerados na interface podem ser convertidos para o código em Python equivalente que implementa a simulação.
- Este mesmo código também é utilizado pela GUI para gerar os dados necessários para a plotagem dos resultados diretamente na interface.
- Similarmente, PDFs podem ser exportados com os mesmos dados.
- A utilização das mesmas funções para todas estas exportações e visualizações garante consistência nos resultados obtidos.

Geração de código, simulação interativa e exportação de resultados


```
def initial_values() -> np.ndarray:
    H_0, P_0 = 100.0, 5.0
    return np.array(( H_0, P_0, ))

def constants() -> list:
    a, b, m, r = 0.015, 0.0125, 0.8, 0.2
    return [ a, b, m, r, ]

def variable_names() -> list[str]:
    return [ "H", "P", ]

def system(t: np.float64, y: np.ndarray, *constants) -> np.ndarray:
    H,P, = y
    a,b,m,r, = constants
    dH_dt = (r * H ) - (a * (H * P ) )
    dP_dt = ((H * P ) * b ) - (P * m )
    return np.array([dH_dt,dP_dt])
```

Estensibilidade

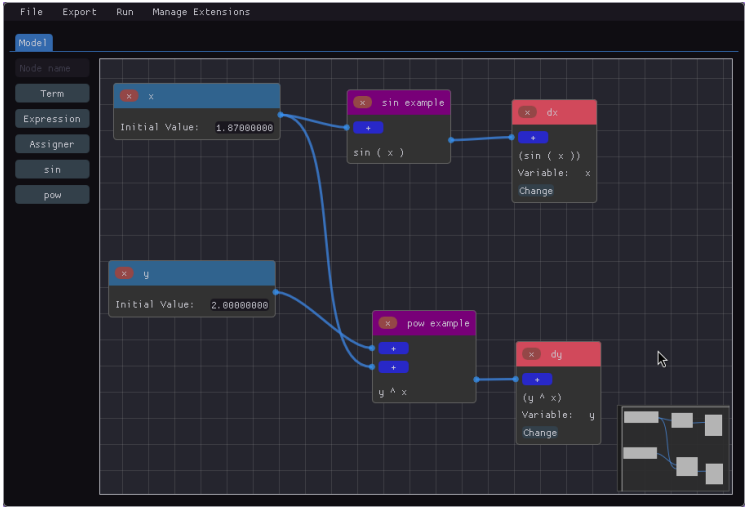
- É improvável que apenas as operações básicas sejam o suficiente para construir qualquer sistema de EDOs.
- Paralelamente, é impensável construir um software que possui todas as operações matemáticas existentes.
- Portanto, uma funcionalidade de extensões foi desenvolvida. O usuário pode utilizar Python para definir nós e as expressões que eles constroem.
  - Sendo escrito em Python, o código é apropriado para ser usado nas etapas mencionadas anteriormente, não prejudicando a usabilidade do software.
  - O software utiliza de trechos de códigos especiais para injetar e inspecionar o código dos usuários a fim de determinar quais funções deveriam representar nós customizados.

Estensibilidade

3 Software para modelagem e simulação

```
import math
@node
def sin(x):
    return math.sin(x)

@node(format='$1 ^ $2')
def pow(x, y):
    return x ** y
```



Distribuição do Software

- Como mencionado anteriormente, o software possui dependência do Python, assim como algumas bibliotecas da linguagem.
- Visto que o público-alvo do software não é necessariamente técnico, a maneira como seria distribuído o software se tornou um ponto chave.
- Portanto, foram desenvolvidas automações capazes de compilar e empacotar o software e todas as suas dependências.
  - Para Linux, o uso de AppImages provê um arquivo único executável.
  - Para Windows, SO que não possui uma ferramenta equivalente, uma pasta compactada é distribuída. A pasta contém o executável principal e as dependências necessárias.

UFSJ

Sumário

4 Aplicações do software

Modelo SIRS

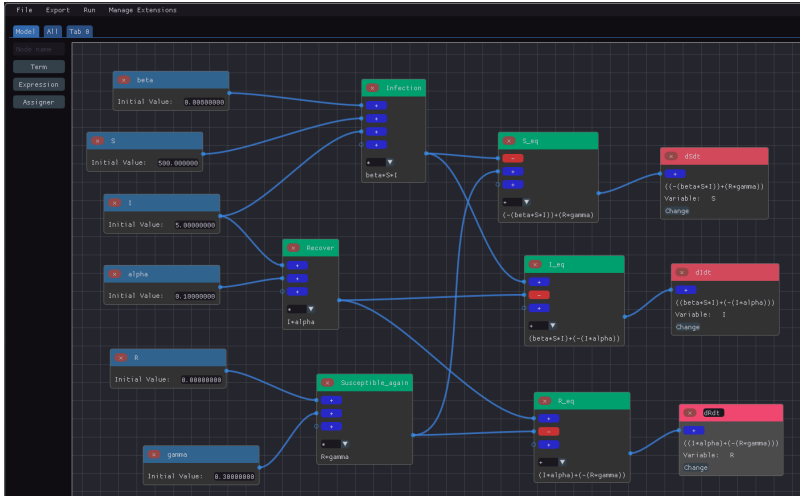
4 Aplicações do software

O modelo SIRS divide a população em indivíduos suscetíveis (S) à doença, infectados (I) e recuperados (R).
Uma parte da população de suscetíveis pode se infectar (termo $\beta.S.I$). Os indivíduos infectados podem se recuperar com uma determinada probabilidade (termo $\alpha.I$) e os indivíduos recuperados podem voltar a ser suscetíveis à doença (termo $\gamma.R$).

$$\begin{aligned} \frac{dS}{dt} &= -\beta.S.I + \gamma.R \\ \frac{dI}{dt} &= \beta.S.I - \alpha.I \\ \frac{dR}{dt} &= \alpha.I - \gamma.R \end{aligned} \tag{3}$$

Modelo SIRS - Representação no Software

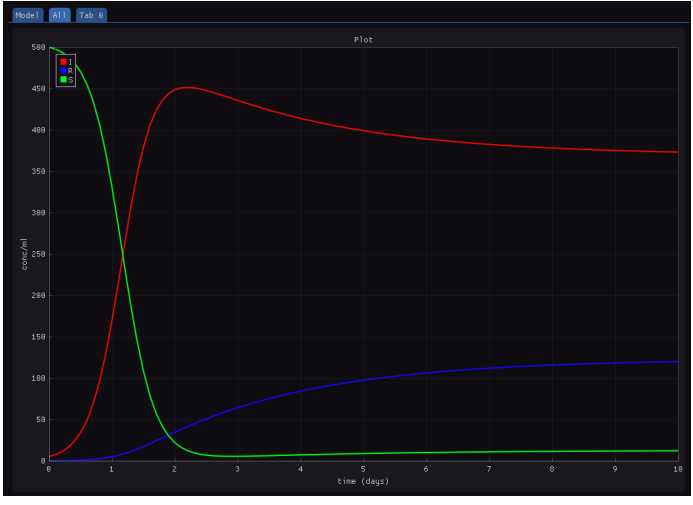
4 Aplicações do software



Modelo SIRS - Resultados

4 Aplicações do software

$$
\begin{aligned}
S_0 &= 500 \\
I_0 &= 5 \\
R_0 &= 0 \\
\alpha &= 0.1 \\
\beta &= 0.008 \\
\gamma &= 0.3
\end{aligned}
$$



UFSJ

Sumário

5 Conclusões e trabalhos futuros

Conclusões

5 Conclusões e trabalhos futuros

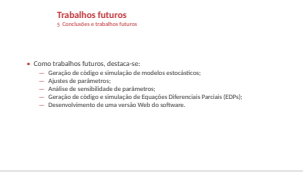
- Neste trabalho, foi desenvolvido um software para automatizar a implementação e simulação de modelos computacionais baseados em EDOs.
- A construção das equações do modelo matemático é auxiliada pela representação visual que foi criada permitindo que o usuário acompanhe a construção de todas as expressões e como elas estão sendo combinadas para formar o sistema de EDOs.
- Através da GUI, é possível ver as entradas e operações de cada expressão, os sinais de cada entrada, quais expressões fazem parte de uma determinada EDO, entre outras coisas.

Conclusões

- Como limitações do trabalho, destaca-se:
  - A representação visual apresenta uma limitação na qual tornar-se mais difícil entender um modelo complexo com muitos nós e ligações.
  - Não foi realizada uma avaliação de usabilidade do software.

Trabalhos futuros

- Como trabalhos futuros, destaca-se:
  - Geração de código e simulação de modelos estocásticos;
  - Ajustes de parâmetros;
  - Análise de sensibilidade de parâmetros;
  - Geração de código e simulação de Equações Diferenciais Parciais (EDPs);
  - Desenvolvimento de uma versão Web do software.



1. Modelos estocásticos: utilizando o algoritmo de Gillespie, o mesmo editor de nós e um novo template para a simulação
2. Ajustes de parâmetros: fornecendo recursos para o carregamento de dados experimentais, escolha dos parâmetros a serem ajustados e plotagens comparativas
3. Web: Rust e as tecnologias usadas na interface gráfica possuem suporte nativo à web (via WebAssebmly). Existem distribuições de Python suportadas no navegador, mas uma solução poderia envolver a execução das simulações do lado do servidor ao invés do cliente.

UFSJ

Desenvolvimento de um software científico para a modelagem e simulação computacional