Proposta de um padrão de implementação de sistemas de equações diferenciais ordinárias em linguagem R

Marcos Paulo Cardoso de Almeida[1]
Adauto Luiz Mancini[2]

A linguagem R é utilizada para cálculos e análise de dados estatísticos e gráficos e, portanto, é utilizada em diversas áreas de pesquisas. Foi proposto um padrão de codificação para modelos matemáticos baseados em sistemas de equações diferenciais ordinárias, a fim de melhorar a legibilidade, facilitar o aprendizado, o reúso, a verificação e integração com eventuais projetos que possam aparecer. Este pensamento surgiu baseado nas experiências obtidas por meio da codificação de modelos matemáticos desenvolvidos para o projeto Pecus.

O padrão de código foi projetado em linguagem R [v. 3.1.1] para ser compatível com a biblioteca de integração numérica DeSolve [v. 1.10-9] - usada para cálculo de equações diferenciais. A IDE RStudio [v. 0.98] foi adotada para se ter uma interface mais amigável com a linguagem R. Utilizou-se o compilador 32bits para o sistema operacional Windows. O sistema de equações diferenciais utilizados nos exemplos e testes foi o descrito por Oltjen et al. (1986), para simulação de crescimento de bovinos.

A programação do modelo, seguindo o padrão proposto, trata por encapsular em uma única função todo o processo de cálculo do modelo matemático. Dentro dessa função pode-se dividir sua estrutura em três partes:

1) A função principal da simulação, denominada Simulator.

2) A função interna, denominada de *Expressions*.

[1] Faculdade de Tecnologia de Americana (Fatec Americana)
[2] Embrapa Informática Agropecuária

3) *ODE*, função que irá calcular as equações diferenciais.

A primeira parte consiste no cabeçalho da função Simulator, mostrada na Figura 1. Os dados de simulação são passados por argumento – tempo de execução, valores dos estados, valores de entrada e variáveis de parâmetros –, para o interior da função Simulator, que irá tratar esses valores em cálculos internos. Os argumentos passados irão determinar quais os tipos de valores serão aceitos na hora do uso da função, e também quais os valores que serão assumidos como padrão, caso o usuário não forneça seus próprios dados a simulação.

```
  time = seq(0, 365, 0.5), State = c(St_DNA = 200, St_PROT = 40, St_GORD = 40),
  Inputs = list(In_DMI = 6.5, In_MEC = 3, In_NEm = 1.8, In_Neg = 1.2),
  Pars = list(Par_Beta0_IMEref = 0.438, Par_Beta1_IMEref = 0.2615,Par_EBWM = 900,
              Par_Beta0_Nut1 = -0.7, Par_Beta1_Nut1 = 1.7, Par_Beta0_Nut2 = 0.83,
              Par_Beta1_Nut2 = 0.2, Par_Beta2_Nut2 = 0.15, Par_k1 = 0.00429,
              Par_k2 = 0.0461, Par_k3 = 0.143, Par_FRAME = 1.2,
              Par_DNAmax = 438, Par_alfa = 0.086)
                                )
```

Figura 1. Cabeçalho da função Simulator.

A segunda parte consiste nos cálculos internos da função Simulator, sendo estes feitos dentro da função Expressions, exemplificado na Figura 2. Esta função interna é necessária para tratar os dados de simulação, passados anteriormente como argumento da função Simulator – ela irá fazer os cálculos dos dados recebidos e, converter o tipo destes dados para os tipos aceitos pela função *ODE*. Também é importante mencionar que a sequência de dados inseridos na declaração/chamada da função Simulator, deve ser idêntica a sequência dos estados retornadas pela função Expressions.

```
Expressions <- function(time, y , parms, inputs, constants){
  with(as.list(c(y, parms, inputs, constants)),  {
    Aux_EBW     = St_PROT/c_LEAN_PROT + St_GORD
    Aux_IMEnorm = (Par_Beta0_IMEref - Par_Beta1_IMEref * Aux_EBW/Par_EBWM)*Aux_EBW^(0.73)
    Aux_IME     = In_DMI*In_MEC
    Aux_P       = Aux_IME/Aux_IMEnorm
    Aux_Nut1    = Par_Beta0_Nut1 + Par_Beta1_Nut1 * Aux_P
    Aux_Nut2    = Par_Beta0_Nut2 + Par_Beta1_Nut2*Aux_P/(Par_Beta2_Nut2 + Aux_P)
    d_DNA  = Par_k1*(Par_DNAmax - St_DNA)*Aux_Nut1
    d_PROT = Aux_Nut2* Par_k2* St_DNA^(0.73) - Par_k3* St_PROT^(0.73)
    d_GORD = ((In_DMI - Par_alfa *Aux_EBW^0.73/ In_NEm)* In_Neg - d_PROT * c_E_PROT)/c_E_FAT
    return(list(c(d_DNA, d_PROT, d_GORD)))
  })
}
StateTrajectory <- ode(y = State, times = time, func = Oltjen, parms = Pars, inputs = Inputs,
                       constants = Constants, method = "rk4")
return(as.data.frame(StateTrajectory))
```

Figura 2. Estrutura da função *Expressions*.

A terceira parte é onde se encontra a chamada da função *ODE*, que soluciona equações diferenciais ordinárias, pertencente ao pacote DeSolve mencionado anteriormente. Nesta parte acontece a junção das partes anteriores, já que para o uso da função *ODE* é necessário utilizar todos os dados tratados na segunda parte, proveniente dos argumentos da primeira parte, como segue exemplificado na Figura 2. Ao passar os argumentos necessários, a função *ODE* retornará todos os dados calculados ao longo do tempo de simulação em uma matriz, logo, para carregar este retorno é necessário atribuir seu retorno a uma variável que irá armazenar a matriz.

Como resultado da implementação do padrão, foram obtidas diversas vantagens, como a restrição de manipulação na estrutura de cálculo interna do modelo, por exemplo, exportando essa função a outro projeto, impediria de alterar as expressões do modelo e eventualmente efetuar alguma ação comprometedora à estrutura de cálculos. Também há a facilidade de manipulação dos dados da simulação, como exemplificado na Figura 3 – os parâmetros da função possuem tipos e valores padronizados, mas a própria linguagem permite que sejam redefinidos cada um dos dados presentes nos argumentos – possibilitando alterar o tempo de execução ou valores iniciais de estados para chamadas distintas, e assim efetuar diversas simulações. Outra vantagem é a formalização de um padrão para versionamento e organização de código para projetos, que facilita a conversão em outras linguagens de programação.

```
State <- OltjenSimulator() #chamada da funçao com valor padrão
State <- OltjenSimulator(#chamada da funçao com valores redefinidos
    time = seq(0, 2000, 20), State = c(St_DNA = 100, St_PROT = 28, St_GORD = 15),
    Inputs = list(In_DMI = DMI,I n_MEC = 3, In_NEm = 1.8, In_Neg = 1.2 )
```

Figura 3. Exemplo do uso da função Simulator.

Palavras-chave: Linguagem R, simulação, padronização.

Referências

OLTJEN, J. W.; BYWATER, A. C.; BALDWIN, R. L.; GARRETT , W. N. Development of a Dynamic Model of Beef Cattle Growth and composition. **Journal of Animal Science**, v. 62, p. 86-97, 1986.