

# OdeShell: An Interactive Tool and a Specific Domain Language to Develop Models Based on Ordinary Differential Equations

Rafael Sachetto Oliveira(✉) and Carolina Ribeiro Xavier

Universidade Federal de São João del-Rei, São João del-Rei, MG, Brazil
{sachetto,carolinaxavier}@ufsj.edu.br

**Abstract.** ODEs are a useful mathematical tool for modeling dynamic systems in different fields, such as physics, engineering, biology, and economics. They can provide insights into the behavior of complex systems over time. However, creating ODE models can be difficult and requires expertise in the subject matter and mathematical techniques. This paper presents the OdeShell, a command line interface that enables users to build and simulate ODE models while examining their behavior under diverse circumstances. OdeShell is a valuable addition to the ODE modeling domain, with the capability to ease the development of intricate models in various fields. The tool accommodates novice and proficient modelers, giving them a flexible and user-friendly environment to build and test ODE models. We elaborate on the principal features and functionality of OdeShell and demonstrate its utility in developing ODE models through small examples. Additionally, we discuss the ODE language, emphasizing its syntax and meaning.

**Keywords:** ODEs · interactive · shell

## 1 Introduction

Ordinary Differential Equations (ODEs) are a powerful mathematical tool for modeling dynamic systems in various fields, from physics and engineering to biology and economics [12]. Ordinary Differential Equation (ODE) models can capture complex systems' underlying mechanisms and behaviors, providing insights into their dynamics over time. However, developing ODE models can be challenging, requiring expertise in the subject matter and the mathematical techniques involved.

To address this challenge, we introduce OdeShell[1], an interactive tool and specific domain language for developing ODE models. OdeShell provides a command line interface (CLI) that allows users to construct and simulate ODE models and analyze their behavior under various conditions. The tool is designed to

[1] Available at https://github.com/rsachetto/odecompiler.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
O. Gervasi et al. (Eds.): ICCSA 2023, LNCS 13957, pp. 385–392, 2023.
https://doi.org/10.1007/978-3-031-36808-0_25

support novice and expert modelers, providing a flexible and intuitive environment for developing and testing ODE models.

In this paper, we describe the key features and functionality of OdeShell and demonstrate its use in developing ODE models in small examples. We also describe the ODE language, highlighting its syntax and semantics. Overall, OdeShell represents a valuable contribution to the field of ODE modeling and has the potential to facilitate the development of complex models across several disciplines.

## 2 The ODE Language and the Source-to-Source Translator

ODE models can be implemented and solved using general-purpose programming languages like C/C++, Python, and Matlab. While these languages offer a high degree of flexibility and control over the model implementation, they often require significant manual coding and debugging efforts.

An alternative approach is describing ODE models using a definition language such as CellML [8]. CellML is an XML-based language that provides a standardized, machine-independent representation of mathematical models. By utilizing CellML, modelers can streamline the model development process, reduce the potential for errors, and enhance model reproducibility.

Once a model is defined in CellML, it can be translated into a general programming language using the CellML API or publicly available tools such as Agos[2] [2] or Myokit[3] [4]. These tools provide a convenient and efficient means of generating simulation code directly from the CellML model, allowing for rapid prototyping and efficient model testing.

### 2.1 ODE Development Language

The primary goal of our ODE representation language is to provide a clear and concise way to describe complex models. To achieve this, we have developed a syntax that focuses on the ODEs in the model but also allows for the use of *if* and *while* statements as well as function definitions. Our language is designed to be more approachable than CellML, while still being powerful as a general-purpose programming language. Researchers can easily communicate their findings and reproduce results by writing models in plain text using our language.

**Builtin Function.** All mathematical functions in the C math library[4] can be used with our ODE language. In addition, we have included four built-in utility functions: print(var), ode_get_value(ode_var, iteration), ode_get_time(ode_var, iteration), and ode_get_num_iterations(). These functions are intended for use within a particular function type called *endfn*, which will be explained in greater detail later in the text.

[2] Available at: https://github.com/FISIOCOMP-UFJF/agos.
[3] Available at: http://myokit.org.
[4] Available at: https://cplusplus.com/reference/cmath/.

**Examples.** Listing 1.1 (sir.ode) shows the implementation of the Susceptible, Infectious, or Recovered (SIR) model [1], described in Eq. 1, using the ODE language. The *ode* keyword is used to define an ordinary differential equation, and the ODE variable name on the left-hand side (LHS) needs to end with a ' character. For now, it is impossible to have mathematical expressions on the LHS; only the ODE variable is allowed. The *initial* keyword is used to set the initial condition of each defined ODE. Variables used in initial conditions must be marked as *global*. Global variables can be used anywhere in the code.

$$\begin{cases} \frac{dS}{dt} & = -\beta SI \\ \frac{dI}{dt} & = \beta SI - \gamma I \\ \frac{dR}{dt} & = \gamma I \end{cases} \tag{1}$$

```
1 global n = 1000
2 global init_i = 3
3 beta = 0.4/n
4 gamma = 0.04
5 #Variables used in initial conditions need to be global
6 initial S = n - init_i
7 initial I = init_i
8 initial R = 0
9 #ODE'S
10 ode S' = -beta*S*I
11 ode I' = beta*S*I - gamma*I
12 ode R' = gamma*I
```

**Listing 1.1.** SIR model implemented using the ODE language (sir.ode).

We can see a more complex model in Listings 1.2 (common.ode) and 1.3 (hh.ode). In the code described in Listing 1.2, we define two functions: *stim* and *calc_E_Na*. In the current version, functions can have Boolean or numeric parameters, and their return values must be numeric (internally, the numeric values are represented in double precision). Listing 1.3 shows the implementation of the Hodgkin-Huxley model [7]. In line 1, we have the *import* keyword. With this statement, the user can import all functions of a file, *common.ode* in this example, and use them to define the model. In the current version, the ODE independent variable is always named *time*, starts from 0, and is global.

```
1 fn stim(t) {
2     if(t > 10 and t < 10.5) { return 20; }
3     return 0;
4 }
5 fn calc_E_Na(E_R) {
6     return E_R + 115.0
7 }
```

**Listing 1.2.** Functions used in the Hodgkin-Huxley model (common.ode).

```
import "common.ode"
Cm    = 1.0
E_R   = -75.0
g_Na  = 120.0
g_K   = 36.0
g_L   = 0.3
E_Na     = calc_E_Na(E_R)
alpha_m  = ((-0.1)*(V + 50.0))/(exp((-(V + 50.0))/10.0) -
    1.0)
beta_m   = 4.0*exp((-(V + 75.0))/18.0)
alpha_h  = 0.07*exp((-(V + 75.0))/20.0)
beta_h   = 1.0/(exp((-(V + 45.0))/10.0) + 1.0)
E_K      = E_R - 12.0
alpha_n  = ((-0.01)*(V+65.0))/(exp((-(V+65.0))/10.0)-1.0)
beta_n   = 0.125*exp((V+75.0)/80.0)
E_L      = E_R + 10.613
i_Na     = g_Na*pow(m,3.0)*h*(V - E_Na)
i_K      = g_K*pow(n, 4.0)*(V - E_K)
i_L      = g_L*(V - E_L)
#Initial conditions
initial V = -7.5e+01
initial m = 5.0e-02
initial h = 6.0e-01
initial n = 3.25e-01
#ODEs
ode V' = -(-stim(time)) + i_Na + i_K + i_L)/Cm
ode m' = alpha_m*(1.0-m) - beta_m*m
ode h' = alpha_h*(1.0-h) - beta_h*h
ode n' = alpha_n*(1.0-n) - beta_n*n
```

**Listing 1.3.** Hodgkin-Huxley model implemented using the ODE language. In this example we show the usage of imports and functions (hh.ode).

### 2.2 The Source-to-Source Translator

After defining a model using the ODE language, it is possible to generate a solver using the *odec* (ODE compiler) software. The *odec* is an almost complete compiler with lexical, syntactic, and semantic analysis that translates the source model to a C solver (a python solver is also planned for future development) that can be compiled and executed to solve the developed model. In the current version, it is possible to generate a solver using a linear multistep method with Backward Differentiation Formulas (provided by the CVODE library [5]) or an adaptive time step method [3] that is implemented with no external dependencies. The current version of the shell and odec is currently available only on Linux and needs the GCC compiler installed.

Using a domain-specific language instead of a markup language to develop a model has several advantages. Firstly, it simplifies the parsing of code and the creation of an abstract syntax tree, making it easier to translate the original code

to a general-purpose language such as C or Python. Additionally, performing semantic and syntactic analysis of the code is more straightforward, providing a better means of discovering and correcting errors in the developed model than a description language, even with a visual inspection. In short, using a domain-specific language can lead to more efficient and accurate model development, with fewer errors and easier debugging.

The C solver can be generated from the SIR model code presented in Listing 1.1, using the command:

```
$ ./bin/odec -i sir.ode -o sir.c
```

If the solver was generated using the default adaptive time step method, the code will not have external dependencies, making it simple to compile using gcc:

```
$ gcc sir.c -o solver -lm
```

After the compilation, the solver can be executed, as described below:

```
$ ./solver 100 out.txt
```

where 100 is the simulation final time and *out.txt* is the output file (these are not fixed parameters, 100 and out.txt where only used as an example).

Listing 1.4 shows the first five lines of the outputted data after solving the sir model. The first line describes the order and the name of the variables, and the following lines are the value of each variable in each timestep.

```
#t, S, I, R
0.000000 997.000000 3.000000 0.000000
0.000001 996.999999 3.000001 0.000000
0.000002 996.988139 3.010671 0.001190
0.009915 996.976123 3.021482 0.002395
```

**Listing 1.4.** First five lines of the output file after solving the sir model.

## 3 The ODE Shell

The ODE Shell is an interactive command line interface (CLI) tool that can be used to solve, manipulate and analyze models developed using the ODE language presented before. The CLI tool was created to provide a flexible and intuitive environment for developing and testing ODE models, catering to novice and expert modelers.

### 3.1 Shell Commands

Several commands were implemented in the ODE shell to allow the developed models' manipulation, plotting, and solving. Following, we have an example usage of the ode shell, using as a model the *sir.ode* file (Listings 1.1):

```
$ /bin/ode_shell -w examples
Using qt as gnuplot terminal.
Current directory /odecompiler/examples
ode_shell> load sir
Model sir successfully loaded
ode_shell> solve 100
Model sir solved for 100.000000 steps.
ode_shell> plotvars
```

In the first line, we execute the ode_shell, passing the directory we want to be our "Work Directory" (-w option). After that, we can start writing commands to the shell. In this example, we first load the sir. model (the ode extension can be omitted), and then we solve the model for 100 steps using the *solve 100* command. After solving the model, we can plot all its variables over time using the *plotvars* command.

Figure 1 shows the output graph of the *plotvars* command as used in the example above. It is worth noting that all the plotting in the shell is performed using the Gnuplot software [10]. The *ode_shell* automatically checks if gnuplot is available in the user's machine and, if not, disables all plotting commands.

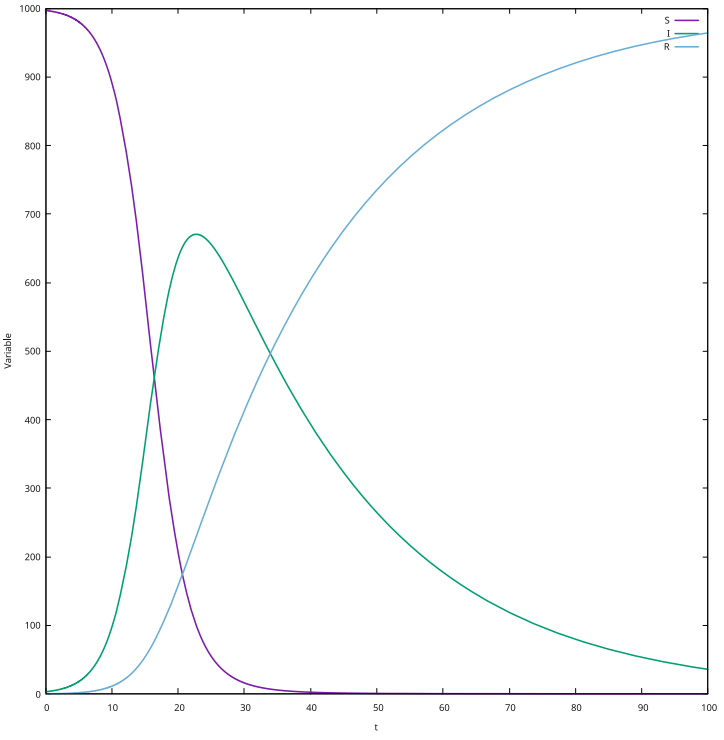


**Fig. 1.** Example output of the *plotvars* command for the sir model.

Table 1 list all the currently available commands in the ODE shell. The user can type 'help command', inside the shell, for more information about a specific command.

**Table 1.** Currently available commands for the ODE shell

| Available commands | | |
|---|---|---|
| cd | closeplot | converttoc |
| editmodel | getglobalvalue | getglobalvalues |
| getinitialvalue | getinitialvalues | getodevalue |
| getodevalues | getparamvalue | getparamvalues |
| getplotconfig | getruninfo | help |
| list | listruns | load |
| loadcmds | ls | odestolatex |
| plot | plottofile | plottoterm |
| plotvar | plotvars | printmodel |
| pwd | quit | replot |
| replottofile | replottoterm | replotvar |
| resetruns | savemodeloutput | saveplot |
| setautolreload | setcurrentmodel | setglobalreload |
| setglobalvalue | setinitialvalue | setodevalue |
| setparamvalue | setplotlegend | setplotx |
| setplotxlabel | setploty | setplotylabel |
| setshouldreload | solve | solveplot |
| unload | vars | |

## 4 Conclusions and Future Work

ODE models are a critical mathematical tool for understanding complex systems dynamics across various fields. However, developing ODE models can be challenging, requiring subject matter expertise and mathematical knowledge. In this paper, we presented OdeShell, an interactive tool, and specific domain language.

OdeShell is designed to support both novice and expert modelers. The described tool is an open-source project, freely available at https://github.com/rsachetto/odecompiler that provides a flexible and intuitive environment for constructing, simulating, and analyzing ODE models. By providing a streamlined and efficient means of generating simulation code, OdeShell has the potential to facilitate the development of complex models and enhance model reproducibility in a wide range of disciplines.

As for future work, the authors plan to incorporate additional analytical functionalities into the software, such as uncertainty quantification [9], sensitivity analysis [6], and stability analysis [11]. These complements would further enhance the capabilities of OdeShell and enable a more comprehensive analysis of ODE models.

## References


1. Anderson, R.M.: Discussion: the Kermack-McKendrick epidemic threshold theorem. Bull. Math. Biol. **53**, 1–32 (1991). https://doi.org/10.1007/BF02464422
2. Barbosa, C.B., et al.: A transformation tool for ODE based models. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 68–75. Springer, Heidelberg (2006). https://doi.org/10.1007/11758501_14
3. Campos, R.S., Lobosco, M., dos Santos, R.W.: Adaptive time step for cardiac myocyte models. Procedia Comput. Sci. **4**, 1092–1100 (2011)
4. Clerx, M., Collins, P., de Lange, E., Volders, P.G.: Myokit: a simple interface to cardiac cellular electrophysiology. Prog. Biophys. Mol. Biol. **120**(1), 100–114 (2016). https://doi.org/10.1016/j.pbiomolbio.2015.12.008. Recent Developments in Biophysics & Molecular Biology of Heart Rhythm
5. Cohen, S.D., Hindmarsh, A.C., Dubois, P.F.: CVODE, a stiff/nonstiff ODE solver in C. Comput. Phys. **10**(2), 138–143 (1996)
6. Dickinson, R.P., Gelinas, R.J.: Sensitivity analysis of ordinary differential equation systems-a direct method. J. Comput. Phys. **21**(2), 123–143 (1976)
7. Hodgkin, A.L., Huxley, A.F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. **117**(4), 500–544 (1952)
8. Miller, A.K., et al.: An overview of the CellML API and its implementation. BMC Bioinform. **11**(1), 1–12 (2010). https://doi.org/10.1186/1471-2105-11-178
9. Mitra, E.D., Hlavacek, W.S.: Parameter estimation and uncertainty quantification for systems biology models. Curr. Opin. Syst. Biol. **18**, 9–18 (2019)
10. Racine, J.: gnuplot 4.0: a portable interactive plotting utility (2006)
11. Roussel, M.R.: Stability analysis for ODEs. In: Nonlinear Dynamics: A Hands-on Introductory Survey. Morgan & Claypool Publishers (2019)
12. Teschl, G.: Ordinary Differential Equations and Dynamical Systems, vol. 140. American Mathematical Society (2012)