



gotranx: General ODE translator

Henrik Finsberg ¹¶ and Johan Hake²

¹ Simula Research Laboratory, Oslo, Norway ² Oslo Katedralskole, Oslo, Norway ¶ Corresponding author

DOI: [10.21105/joss.07063](https://doi.org/10.21105/joss.07063)

Software

- [Review](#) 
- [Repository](#)
- [Archive](#) 

Editor: [Rohit Goswami](#)  

Reviewers:

- [@benlansdell](#)
- [@ayush9pandey](#)
- [@SunnyXu](#)

Submitted: 01 July 2024

Published: 17 October 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

We introduce gotranx, a General ODE Translator for automatic code generation of ordinary differential equations (ODEs). The user writes the ODE in a markup language with the file extension `.ode` and the tool generates code with numerical schemes for solving the ODE in different programming languages.

gotranx implements a domain specific language (DSL) using [Lark](#) for representing ODEs. It can parse the variables and equations into a symbolic representation ([Meurer et al., 2017](#)), and generate numerical schemes based on codeprinters from Sympy, in particular the Rush-Larsen scheme ([Rush & Larsen, 1978](#)) which is very popular in the field of computational biology.



Figure 1: Logo of the software that illustrates that you take a dynamical system (i.e an ODE) and use gotranx (i.e a black box) to turn the ODE into computer code

The vision for gotranx is to implement the same features in gotranx as found in [gotran](#) along with additional features.

Statement of need

Systems of ordinary differential equations (ODEs) are equations of the form

$$y'(t) = f(t, y),$$

where t represents time and y is a vector of state variables. These equations are a fundamental concept in mathematics and science, and play a critical role in various fields such as physics, engineering, and economics.

Solving ODEs can be done analytically in simple cases but for most real world applications one needs to apply numerical methods ([Ascher & Petzold, 1998](#)). For this, a suite of well-established software exists, such as `scipy` ([Virtanen et al., 2020](#)) for Python, `DifferentialEquations.jl` ([Rackauckas & Nie, 2017](#)) for Julia, and `Sundials` ([Hindmarsh et al., 2005](#)) for C and C++ .

Computational biology is one field where ODEs play an essential role, for example in models of heart cells. The resulting ODEs can, in these cases, be quite involved with many parameters and state variables. For example, one of the more recent models of human heart cells (Tomek et al., 2019) has 112 parameters, 45 state variables, and 276 intermediate variables. Solutions of the state equations of these membrane models typically allows for specialized exponential integrators (Rush & Larsen, 1978) that are implemented as a numerical scheme in gotranx.

Furthermore, such models of a single heart cell are typically embedded into a spatial model, such as the Monodomain or Bidomain model (Sundnes et al., 2007), where each point in a 3D geometry represents a cell. This means that thousands or even millions of such ODEs needs to be coupled and solved in a larger systems. In these cases, traditional solvers are usually unsuitable and custom code are often developed to solve the ODEs. An alternative to this approach is to use an existing framework specialized for these types of problems (Cooper et al., 2020; Plank et al., 2021). However, introducing a big framework might not be ideal if the user wants to do a lot of customization. With gotranx you can easily generate framework independent code that can easily be integrated into most simulation pipelines. One example is `fenics-beat` (Finsberg, 2024), which uses gotranx to generate code solving ODEs in a Monodomain model.

While such models are typically developed in one programming language, different research groups use different programming languages to integrate and solve their models. To make these models programming language independent, it is common practice to write them in a markup language (Keating et al., 2020; Lloyd et al., 2004). However, when translating the models to a new programming language, user typically need to do this manually, which is likely to introduce errors in the code. Since gotranx also implements converters, e.g., for CellML through MyoKit (Clerx et al., 2016), it already supports most models that are used today.

gotranx is a full rewrite of `gotran`, which was first developed in 2012, primarily for generating code for solving problems in cardiac electrophysiology. `gotran` is very tightly coupled to `sympy` and at one point we were unable to upgrade the version of `sympy`, which drastically limited its usage. This was the main motivation for the upgrade. You can read more about this background in the [documentation](#).

Features

The core idea behind gotranx is

1. to define your ODE in a high level language (so called `.ode` file), and
2. to generate code for solving the ODE in different programming languages

The `.ode` format is text based and high level stripped for details. For example the following `.ode` file

```
states(x=1, y=0)
parameters(a=1.0)
```

```
dx_dt = a * y
dy_dt = -x
```

defines the ODE system

$$\frac{dx}{dt} = ay$$
$$\frac{dy}{dt} = -x$$

with the initial conditions $x(0) = 1$ and $y(0) = 0$ and the parameter a with a value of 1.0. We can now generate code for solving the ODE using the Generalized Rush-Larsen scheme

(Sundnes et al., 2009) with the following command (assuming you wrote the ODE in the file file.ode)

```
gotranx ode2py file.ode --scheme generalized_rush_larsen -o file.py
```

which will output the code a new python file named file.py. We can now use this code as follows to solve the ODE

```
import file as model
import numpy as np
import matplotlib.pyplot as plt

s = model.init_state_values()
p = model.init_parameter_values()
dt = 1e-4 # 0.1 ms
T = 2 * np.pi
t = np.arange(0, T, dt)

x_index = model.state_index("x")
x = [s[x_index]]
y_index = model.state_index("y")
y = [s[y_index]]

for ti in t[1:]:
    s = model.generalized_rush_larsen(s, ti, dt, p)
    x.append(s[x_index])
    y.append(s[y_index])

fig, ax = plt.subplots()
ax.plot(t, x, label="x")
ax.plot(t, y, label="y")
ax.set_xlabel("Time [s]")
ax.set_ylabel("State variables")
ax.legend()
plt.show()
```

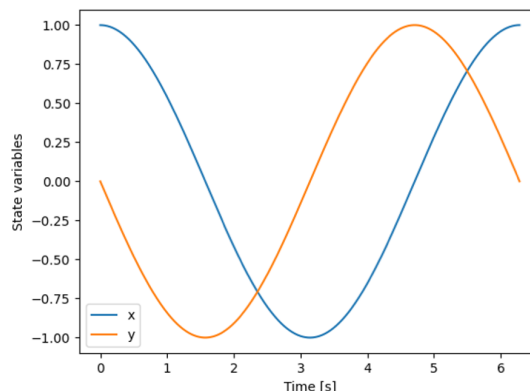


Figure 2: Example of code from loading and solving an ODE with the code generated from gotranx. On the right we see the solution.

At the time of writing, gotranx support code generation to C and Python, with backends for [numpy](#) and [jax](#). It also supports conversion to and from [CellML](#) via [MyoKit](#) (at the time of writing we still do not support unit conversion from gotranx to MyoKit). For a list of all features and the roadmap, please see the [roadmap](#).

Acknowledgements

Henrik Finsberg is the main developer of gotranx. The original gotran library was created by Johan Hake, and he is acknowledged with co-authorship for this. We would also like to acknowledge Kristian Hustad for valuable discussions and contributions to the original gotran library. This work has been financially supported by Simula Research Laboratory.

References

- Ascher, U. M., & Petzold, L. R. (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM. <https://doi.org/10.1137/1.9781611971224>
- Clerx, M., Collins, P., De Lange, E., & Volders, P. G. (2016). Myokit: A simple interface to cardiac cellular electrophysiology. *Progress in Biophysics and Molecular Biology*, 120(1-3), 100–114. <https://doi.org/10.1016/j.pbiomolbio.2015.12.008>
- Cooper, F. R., Baker, R. E., Bernabeu, M. O., Bordas, R., Bowler, L., Bueno-Orovio, A., Byrne, H. M., Carapella, V., Cardone-Noott, L., Jonatha, C., & others. (2020). Chaste: Cancer, heart and soft tissue environment. *Journal of Open Source Software*, 5(47). <https://doi.org/10.21105/joss.01848>

- Finsberg, H. (2024). *fenics-beat* (Version 0.1.1). <https://doi.org/10.5281/zenodo.13323643>
- Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., & Woodward, C. S. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, *31*(3), 363–396. <https://doi.org/10.1145/1089014.1089020>
- Keating, S. M., Waltemath, D., König, M., Zhang, F., Dräger, A., Chaouiya, C., Bergmann, F. T., Finney, A., Gillespie, C. S., Helikar, T., & others. (2020). SBML level 3: An extensible format for the exchange and reuse of biological models. *Molecular Systems Biology*, *16*(8), e9110. <https://doi.org/10.15252/msb.20199110>
- Lloyd, C. M., Halstead, M. D., & Nielsen, P. F. (2004). CellML: Its future, present and past. *Progress in Biophysics and Molecular Biology*, *85*(2-3), 433–450. <https://doi.org/10.1016/j.pbiomolbio.2004.01.004>
- Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., & others. (2017). SymPy: Symbolic computing in Python. *PeerJ Computer Science*, *3*, e103. <https://doi.org/10.7717/peerj-cs.103>
- Plank, G., Loewe, A., Neic, A., Augustin, C., Huang, Y.-L., Gsell, M. A., Karabelas, E., Nothstein, M., Prassl, A. J., Sánchez, J., & others. (2021). The openCARP simulation environment for cardiac electrophysiology. *Computer Methods and Programs in Biomedicine*, *208*, 106223. <https://doi.org/10.1016/j.cmpb.2021.106223>
- Rackauckas, C., & Nie, Q. (2017). Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, *5*(1), 15–15. <https://doi.org/10.5334/jors.151>
- Rush, S., & Larsen, H. (1978). A practical algorithm for solving dynamic membrane equations. *IEEE Transactions on Biomedical Engineering*, *4*, 389–392. <https://doi.org/10.1109/TBME.1978.326270>
- Sundnes, J., Artebrant, R., Skavhaug, O., & Tveito, A. (2009). A second-order algorithm for solving dynamic cell membrane equations. *IEEE Transactions on Biomedical Engineering*, *56*(10), 2546–2548. <https://doi.org/10.1109/TBME.2009.2014739>
- Sundnes, J., Lines, G. T., Cai, X., Nielsen, B. F., Mardal, K.-A., & Tveito, A. (2007). *Computing the electrical activity in the heart* (Vol. 1). Springer Science & Business Media. <https://doi.org/10.1007/3-540-33437-8>
- Tomek, J., Bueno-Orovio, A., Passini, E., Zhou, X., Mincholé, A., Britton, O., Bartolucci, C., Severi, S., Shrier, A., Virag, L., & others. (2019). Development, calibration, and validation of a novel human ventricular myocyte model in health, disease, and drug block. *Elife*, *8*, e48890. <https://doi.org/10.7554/eLife.48890>
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>