stubs

Release 0.1.4

Justin Laughlin

Nov 20, 2021

GETTING STARTED:

STUBS is a biophysical simulation library that provides a level of abstraction to models, making it easier for users to develop, share, and simulate their mathematical models. STUBS is highly suited for building systems biology models and simulating them as deterministic partial differential equations *[PDEs]* in realistic geometries using the Finite Element Method *[FEM]* - the integration of additional physics such as electro-diffusion or stochasticity may come in future updates. Systems biology models are converted by STUBS into the appropriate systems of reaction-diffusion PDEs with proper boundary conditions. FEniCS is a core dependency of STUBS which handles the assembly of finite element matrices as well as solving the resultant linear algebra systems.

CHAPTER

ONE

INSTALLATION

Simply run `pip install fenics-stubs` in an environment with FEniCS installed. We recommend using a FEniCS docker container to minimize installation issues.

CHAPTER

TWO

MATHEMATICS

Mathematics related to stubs.

2.1 Multi-Dimensional Reaction-Diffusion Equations

Volumetric partial differential equations

$$\partial_t u_i^{(a)} = \nabla \cdot (D_i^{(a)} \nabla u_i^{(a)}) + f_i^{(a)}(u^{(a)}) \text{ in } \Omega^{(a)}$$
$$D_i^{(a)}(\nabla u_i^{(a)} \cdot n) = r_i^{(abc)}(u^{(a)}, u^{(b)}, v^{(abc)}) \text{ on } \Gamma^{(abc)}$$

Surface partial differential equations

$$\partial_t v_i^{(abc)} = \nabla_S \cdot (D_i^{(abc)} \nabla_S v_i^{(abc)}) + g_i^{(abc)}(u^{(a)}, u^{(b)}, v^{(abc)}) \text{ on } \Gamma^{(abc)}$$
$$D_i^{(abc)}(\nabla_S v_i^{(abc)} \cdot n) = 0 \text{ on } \partial\Gamma^{(abc)}$$

CHAPTER

THREE

FREQUENTLY ASKED QUESTIONS

1. How do I use `pip` to install `stubs`?

   The name of the package is `fenics-stubs` which unfortunately does not match the module name. Run `pip install fenics-stubs`.

CHAPTER

FOUR

PYTHON API REFERENCE

Modules

stubs.common	General functions: array manipulation, data i/o, etc
stubs.config	Configuration settings for simulation: plotting, reaction types, solution output, etc.
stubs.data_manipulation	Functions to help with managing solutions / post-processing
stubs.mesh	Wrapper around dolfin mesh class (originally for sub-mesh implementation - possibly unneeded now)
stubs.model_assembly	Classes for parameters, species, compartments, reactions, fluxes, and forms Model class contains functions to efficiently solve a system
stubs.model_building	Classes/functions used to construct models
stubs.model	Model class.
stubs.solvers	Solver classes.

4.1 stubs.common

General functions: array manipulation, data i/o, etc

Functions

append_meshfunction_to_meshdomains(mesh, ...)	
bmesh_to_parent(bmesh_emap_0, index)	
color_print(full_text, color)	
insert_dataframe_col(df, columnName[, ...])	pandas requires some weird manipulation to insert some things, e.g.
interp_limit_dy(t, y, max_dy[, interp_type])	Interpolates t and y such that dy between time points is never greater than max_dy Maintains all original t and y
json_to_ObjectContainer(json_str[, data_type])	Converts a json_str (either a string of the json itself, or a filepath to the json)

continues on next page

Table 2 – continued from previous page

mesh_vertex_to_dof(V, species_index, index)	
nan_to_none(df)	
np_smart_hstack(x1, x2)	Quality of life function.
read_smodel(filepath)	
round_to_n(x, n)	Rounds to n sig figs
submesh_dof_to_mesh_dof(Vsubmesh, submesh, ...)	Takes dof indices (single index or a list) on a submesh of a boundarymesh of a mesh and returns the dof indices of the original mesh.
submesh_dof_to_vertex(Vsubmesh, species_index)	
submesh_to_bmesh(submesh, index)	
sum_discrete_signals(ty1, ty2[, max_dy])	ty1: Numpy array of size N1 x 2 where the first column is time and the second column is the first signal ty2: Numpy array of size N2 x 2 where the first column is time and the second column is the second signal
write_smodel(filepath, pdf, sdf, cdf, rdf)	Takes a ParameterDF, SpeciesDF, CompartmentDF, and ReactionDF, and generates a .smodel file (a convenient concatenation of .json files with syntax similar to .xml)

Classes

ref(obj)	Pandas dataframe doesn't play nice with dolfin indexed functions since it will try to take the length but dolfin will not return anything.

4.2 stubs.config

Configuration settings for simulation: plotting, reaction types, solution output, etc.

Classes

Config()	Refactored config

4.3 stubs.data_manipulation

Functions to help with managing solutions / post-processing

Classes

`Data`(model, config)	

4.4 stubs.mesh

Wrapper around dolfin mesh class (originally for submesh implementation - possibly unneeded now)

Classes

`ChildMesh`(parent_mesh, dimensionality, ...)	Sub mesh of a parent mesh
`ParentMesh`([name, mesh_filename])	Mesh loaded in from data.

4.5 stubs.model_assembly

Classes for parameters, species, compartments, reactions, fluxes, and forms Model class contains functions to efficiently solve a system

Classes

`Compartment`(name[, Dict])	
`CompartmentContainer`([df, Dict])	
`Flux`(flux_name, species_name, symEqn, ...[, ...])	
`FluxContainer`([df, Dict])	
`Form`(dolfin_form, species, form_type[, ...])	
`FormContainer`()	
`Parameter`(name[, Dict])	
`ParameterContainer`([df, Dict])	
`Reaction`(name[, Dict, eqn_f_str, eqn_r_str, ...])	

continues on next page

Table 7 – continued from previous page

ReactionContainer([df, Dict])	
Species(name[, Dict])	
SpeciesContainer([df, Dict])	

4.6 stubs.model_building

Classes/functions used to construct models

Classes

CompartmentDF()	Dimensionality refers to the topological dimension (e.g.
ParameterDF()	A standard (non time-dependent) parameter has an associated value, unit,
ReactionDF()	A reaction is specified by the reactants/products on the left hand side (LHS), right hand side (RHS), kinetic parameters, the reaction type (STUBS will always assume mass action unless specified otherwise).
SpeciesDF()	IC assumed to be in terms concentration units

4.7 stubs.model

Model class. Consists of parameters, species, etc. and is used for simulation

Classes

Model(PD, SD, CD, RD, config, solver_system)	

4.8 stubs.solvers

Solver classes. Linear/Nonlinear are wrappers around dolfin solvers. MultiphysicsSolver uses a picard iteration scheme to solve coupled problems.

Classes

DolfinKrylovSolver([method, preconditioner, ...])	
LinearSolver([method, preconditioner])	
MultiphysicsSolver([method, eps_Fabs, ...])	
NonlinearNewtonSolver([maximum_iterations, ...])	Settings for dolfin nonlinear Newton solver
NonlinearPicardSolver([picard_norm, ...])	
NonlinearSolver([method, min_nonlinear, ...])	
Solver([framework])	
SolverSystem([final_t, initial_dt, ...])	

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

M

S