EqParse Documentation

Release

Haroon Arshad

December 16, 2014

CONTENTS

Contents:

CHAPTER

ONE

EQPARSE PACKAGE

1.1 eqparse Package

The equation parse package is a universal parser from a single-input file describing a larger ODE system to different programming languages. Multiple file output of the same language can easily be created and other parser modules can easily be implemented for other language outputs with some python knowledge eqparse import module initialise

1.2 baseparse Class

eqparse.__init__.**baseparse**
alias of eqparse.baseparse

1.3 baseparse module

class eqparse.baseparse.**BaseParse**
Bases: object

The main objective of BaseParse is to organise universal functions and member variables that are inherited in other class parser modules. Current supplied parsers include the MatlabParser, CppParser and XPPautParser as child modules.

Initialise universal variables usually important in most or all language syntax of the following. if a module inherits this class, the following variables usually need to be redefined according to its language syntax. The following parameters are automatically defined as __init__ is called

Parameters

- **l_enclose** – The left character to access a piece of memory is usually a left square bracket (C++) or a normal left bracket (MATLAB
- **r_enclose** – Similar to :param l_enclose:
- **comment_prefix** – character used for single-line comment
- **vec_counter_start** – starting number to access first element of a container/array (default: 0)
- **container_type** – a list of container types used
- **PERM_I_LIST** – the input file contain preconditioned statements in csv input file such as if statements or power function.

- **operations** – the required result during parse of the :param PERM_I_LIST: list to the respective language syntax

close_file(*file_n_str*)

end()

get_directory()

get_index(*index_to_get*)

get_list(*mKey*, *mLib=None*, *mPrefix=''*, *mPostfix=''*, *mIndex=None*)
get a list of a particular property of the variables. The list of properties depends on a key which associates a particular property of a variables library (e.g. comment, rhs).

Parameters

- **mKey** – the property type of a variable
- **mLib** – library to use e.g self.data (normal format) self.data_vec (vectorised format) (default - self.data)
- **mPrefix** – pre-fix the returned property with a string.
- **mPostfix** – post-fix the returned property with a string.

get_names()

get_only_names(*oNames*, *mIndex*)

initialise_library(*lib*)
In order to initialise library

Parameters **lib** – The created library from the `createlibrary` class

Returns void

new_dependancy_index(*index_to_order*, *key_to_order*, *is_return=True*)

new_index(**argv*)

new_names(*oNames, mIndex, specifier=['', ''], return_it=True*)

new_names_restricted(*oNames, mIndex, specifier=['', ''], return_it=True*)

new_order(**argv*)

open_file(*file_n_str_key*)

order_index(*index_to_order*, *order_by=None*)

pattern_change(*gIndex*, *orig*, **argv*)

pattern_write(*w_file*, *w_data_tuple*, *w_str_tuple*, *w_lib=None*, *mIndex=None*, *end_wrr_='n'*, *beg_wrr_=''*)
Write to file output each of the required variables with a similar coded pattern

Parameters

- **w_file** (*str*) – file to write to specified by string used in open_file function
- **w_data_tuple** (*list*) – combination of strings (if same for each variable) and lists (each with same size) that will be printed in consecutive order
- **w_str_tuple** (*str*) –
- **w_lib** – post-fix the returned property with a string.
- **mIndex** (*list*) – specify index of variables to go through.

rasterise_dict_modifiers(*refi*)

replace_operations(*ops=None*)

search_and_replace(*origContainer*, *repNames*, *origNames=None*, *return_it=True*)
Search and replace the variables of a container of equations (normally the RHS or initial condition) to specified new names. All containers should have same length and coincide to variable name properties

Parameters

- **origContainer** – data of formular/equations containing variables one wants to change
- **repNames** – the names one wants to replace to parallel to :param *origNames*:
- **origNames** – the original names to replace (default - original variable names)

vectorise_name(*mIndex*, *specifier*, *names_list=None*, *return_it=True*)

write(*fileAttr*, *strAttr*, *end_wrr_='n'*, *beg_wrr_=''*)

write_comment(*fileAttr*, *strAttr*, *end_wrr_='n'*, *beg_wrr_=''*)

1.4 cppparser module

class eqparse.cppparser.**CppParser**(*Lib*)
Bases: eqparse.baseparse.BaseParse

cpp_original()

1.5 createlibrary module

create Library of variables : Read saved model file (current formats: csv,)

class eqparse.createlibrary.**CreateLibrary**(*g_abspath_title*, *name_of_file_list*)
Bases: object

Before calling a parser module, one must call this class with the csv files to be parsed, organised and managed into a dictionary for easy look-up and readability for the different parsers. Currently this module only accepts CSV files. In the future anticipate other possible formats such as XML and possibly a GUI.

Examples are included that parse example ode problems into several formats

OK

add_index(*define_dict_keys*, *cond_on={}*, *cond_not_on={}*)

add_variable(*mDict*)

change_variable(*mname*, *mval*)

complete()

copy_lib()

copy_lib_vec()

create_data()

dup2(*n*)

function_order(**argv*)

get_index_dependency(*m_key_to_order='init-value'*, *resp_order_val='name'*, *cond_remove={}*, *cond_include={}*)

set_directory()
Set directory where file specific parser files are saved to, the directory can be a relative path and not necessarily an absolute path

Parameters **dir_str** – string of path to directory

table = []

eqparse.createlibrary.**change_a_type**(*Lib*, *vName*, *vKey*, *vToo*)

1.6 matlabparser module

class eqparse.matlabparser.**MatlabParser**(*Lib*)
Bases: eqparse.baseparse.BaseParse

To create matlab parsed file

FSA_cvodes()

FSA_ode()
Create FSA matlab ODE file (ode.m) to be used with multi_runfile and multi_runfile_slider modules

inset_runfile_slider(*newSubDirectory=''*, *file_ext=''*, *no_guess=None*, *yes_guess=None*)

multi_odefile(*newSubDirectory=''*, *file_ext=''*, *no_guess=None*, *yes_guess=None*, *wr_modifiers=''*)
Create matlab ODE file (ode.m) to be used with multi_runfile and multi_runfile_slider modules

multi_paramfile(*newSubDirectory=''*, *file_ext=''*, *no_guess=None*, *yes_guess=None*)

multi_runfile(*newSubDirectory=''*, *file_ext=''*, *no_guess=None*, *yes_guess=None*, *wr_modifiers=''*)
Create run file (run.m) that uses the ode file created from running alongside with multi_odefile module. This format is currently set for MATLAB defined implicit ode function - ode15s.

Parameters

- **newSubDirectory** –
- **file_ext** –
- **no_guess** –
- **yes_guess** –
- **wr_modifiers** –

multi_runfile_slider(*newSubDirectory=''*, *file_ext=''*, *no_guess=None*, *yes_guess=None*)

p_est_main(*param_to_est*)

solve_single()
generate single MATLAB (single_generic.m) file to solve equations. Runs the Euler finite differece scheme to model with dt=5 (ms)

supp_initialise_data()

supp_input_ic()

vectorised_to_readable()

1.7 smc_helper_functions module

eqparse.smc_helper_functions.**error**(*err_str*)

1.8 timer module

class eqparse.timer.**Timer**

finish(*m_str=None*)

start()

1.9 xppautparser module

class eqparse.xppautparser.**XppautParser**(*Lib*)

Bases: eqparse.baseparse.BaseParse

parse_one(*concateq_unordered=[]*)

printarray(*c*, *specific=None*)

set_temp_ic_name()

truncate_ics()

xpp_search_and_define()

1.10 Module contents

eqparse import module initialise

e