# **BlueSky Prototype Model**

Release v1.1

**U.S. Energy Information Administration** 

## **CONTENTS**

1	Sphi	nx Documentation Structure	-
	1.1	Package Overview	1
		Submodules and Subpackages	
		Example Structure	
	1.4	Model Structure	2
Pv	thon ]	Module Index	141

## SPHINX DOCUMENTATION STRUCTURE

Sphinx organizes the documentation into the following sections:

## 1.1 Package Overview

The documentation starts with a general overview of the main package structure. In this project, the top-level package is *src*. Inside the *src* package, you will find the following main packages:

- electricity: Contains modules related to electricity modeling and calculations.
- hydrogen: Contains modules for hydrogen energy production, storage, and consumption.
- residential: Contains models and utilities related to residential energy use.
- **integrator**: Integrates components from various energy sources (electricity, hydrogen, etc.) into a cohesive system.

## 1.2 Submodules and Subpackages

Each package may contain additional submodules and subpackages, which are organized in the documentation as follows:

- **Package**: Each package (e.g., electricity, hydrogen) is documented with a high-level description of its purpose and contents.
- **Submodules**: The individual Python modules within each package are listed and documented. For example, the *integrator* package may contain submodules like *runner.py*, *utilites.py*, and *progress\_plot.py*. Each of these modules will have its own section.
- **Subpackages**: If a package contains nested subpackages, these are also documented. For instance, if *electricity* has a subpackage *scripts*, it will have its own subsection with corresponding submodules.

Each module's docstrings are captured to provide detailed information about functions, classes, methods, and attributes.

## 1.3 Example Structure

For the src package, Sphinx might organize the contents as follows:

```
src (package)/
    integrator (subpackage)/
    input
        runner.py (module)
        models (subpackage)/
```

(continues on next page)

(continued from previous page)

In the HTML and Markdown outputs, each package and subpackage is represented with links **below** to the respective modules, making it easy to navigate between different sections of the documentation.

Use the **Search bar** on the top left to search for a specific function or module.

## 1.4 Model Structure

src.models	
STC	This directory contains the subdirectory for the integrator module and the subdirectories for the sectoral modules.

## 1.4.1 src.models

#### **Modules**

```
electricity
hydrogen
residential
```

## src.models.electricity

## **Modules**



## src.models.electricity.scripts

## **Modules**

electricity_model	Electricity Model, a pyomo optimization model of the electric power sector.
postprocessor	This file is the main postprocessor for the electricity model.
preprocessor	This file is the main preprocessor for the electricity model.
runner	This file is a collection of functions that are used to build, run, and solve the electricity model.
utilities	This file is a collection of functions that are used in support of the electricity model.

## $src.models.electricity.scripts.electricity\_model$

Electricity Model, a pyomo optimization model of the electric power sector.

The class is organized by sections: settings, sets, parameters, variables, objective function, constraints, plus additional misc support functions.

## **Functions**

<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if
	necessary.

## **Classes**

HI(region, year)	(region, year)
Model(*args, **kwds)	This is the base model class for the models.
PowerModel(*args, **kwds)	A PowerModel instance.
defaultdict	defaultdict(default_factory=None, /, [])> dict with
	default factory
em	alias of ElectricityMethods

class src.models.electricity.scripts.electricity\_model.PowerModel(\*args, \*\*kwds)

A PowerModel instance. Builds electricity pyomo model.

## **Parameters**

- all\_frames (dictionary of pd.DataFrames) Contains all dataframes of inputs
- setA (Sets) Contains all other non-dataframe inputs

\_active

## src.models.electricity.scripts.postprocessor

This file is the main postprocessor for the electricity model.

## It writes out all relevant model outputs (e.g., variables, parameters, constraints). It contains:

- A function that converts pyomo component objects to dataframes
- A function that writes the dataframes to output directories
- A function to make the electricity output sub-directories
- · The postprocessor function, which loops through the model component objects and applies the

functions to convert and write out the data to dfs to the electricity output sub-directories

#### **Functions**

<pre>create_obj_df(mod_object)</pre>	takes pyomo component objects (e.g., variables, parameters, constraints) and processes the pyomo data and converts it to a dataframe and then writes the dataframe out to an output dir.
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if necessary.
<pre>make_elec_output_dir(output_dir)</pre>	generates an output subdirectory to write electricity model results.
postprocessor(instance)	master postprocessor function that writes out the final dataframes from to the electricity model.
report_obj_df(mod_object, instance, dir_out,)	Creates a df of the component object within the pyomo model, separates the key data into different columns and then names the columns if the names are included in the cols_dict.

## **Classes**

Path(*args, **kwargs)	PurePath subclass that can make system calls.
-----------------------	---

## $\verb|src.models.electricity.scripts.postprocessor.make\_elec\_output\_dir(|\mathit{output\_dir})|$

generates an output subdirectory to write electricity model results. It includes subdirs for vars, params, constraints.

## Returns

the name of the output directory

## Return type

string

## src.models.electricity.scripts.postprocessor.postprocessor(instance)

master postprocessor function that writes out the final dataframes from to the electricity model. Creates the output directories and writes out dataframes for variables, parameters, and constraints. Gets the correct columns names for each dataframe using the cols\_dict.

#### Parameters

instance (pyomo model) - electricity concrete model

#### Returns

output directory name

## Return type

string

Creates a df of the component object within the pyomo model, separates the key data into different columns and then names the columns if the names are included in the cols\_dict. Writes the df out to the output directory.

## **Parameters**

- **obj** (*pyomo component object*) e.g., pyo. Var, pyo. Set, pyo. Param, pyo. Constraint
- instance (pyomo model) electricity concrete model
- dir\_out (str) output electricity directory
- **sub\_dir** (*str*) output electricity sub-directory

## src.models.electricity.scripts.preprocessor

This file is the main preprocessor for the electricity model.

## It established the parameters and sets that will be used in the model. It contains:

- · A class that contains all sets used in the model
- A collection of support functions to read in and setup parameter data
- The preprocessor function, which produces an instance of the Set class and a dict of params
- A collection of support functions to write out the inputs to the output directory

## **Functions**

<pre>add_season_index(cw_temporal, df, pos)</pre>	adds a season index to the input dataframe
<pre>avg_by_group(df, set_name, map_frame)</pre>	takes in a dataframe and groups it by the set specified and then averages the data.
<pre>capacitycredit_df(all_frames, setin)</pre>	builds the capacity credit dataframe
<pre>create_hourly_params(all_frames, key, cols)</pre>	Expands params that are indexed by season to be indexed by hour
<pre>create_hourly_sets(all_frames, df)</pre>	expands sets that are indexed by season to be indexed by hour
<pre>create_other_sets(all_frames, setin)</pre>	creates other (non-supply curve) sets
<pre>create_sc_sets(all_frames, setin)</pre>	creates supply curve sets
<pre>create_subsets(df, col, subset)</pre>	Create subsets off of full sets
fill_values(row, subset_list)	Function to fill in the subset values, is used to assign all years within the year solve range to each year the model will solve for.
<pre>hourly_sc_subset(all_frames, subset)</pre>	Creates sets/subsets that are related to the supply curve
<pre>hr_sub_sc_subset(all_frames, T_subset, hr_subset)</pre>	creates supply curve subsets by hour
load_data(tablename, metadata, engine)	loads the data from the SQL database; used in readin_sql function.
<pre>makedir(dir_out)</pre>	creates a folder directory based on the path provided
output_inputs(OUTPUT_ROOT)	function developed initial for QA purposes, writes out to csv all of the dfs and sets passed to the electricity model to an output directory.
preprocessor(setin)	main preprocessor function that generates the final dataframes and sets sent over to the electricity model.
	continues on next page

Table 9 – continued from previous page

<pre>print_sets(setin)</pre>	function developed initially for QA purposes, prints out all of the sets passed to the electricity model.
readin_csvs(all_frames)	Reads in all of the CSV files from the input dir and returns a dictionary of dataframes, where the key is the file name and the value is the table data.
readin_sql(all_frames)	Reads in all of the tables from a SQL databased and returns a dictionary of dataframes, where the key is the table name and the value is the table data.
scale_load(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year) and LoadScalar.csv (a multiplier for all model years).
scale_load_with_enduses(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year), EnduseBaseShares.csv (the shares of demand for each enduse in the base year) and EnduseScalar.csv (a multiplier for all model years by enduse category).
step_sub_sc_subset(all_frames, T_subset,)	creates supply curve subsets by step
<pre>subset_dfs(all_frames, setin, i)</pre>	filters dataframes based on the values within the set
<pre>time_map(cw_temporal, rename_cols)</pre>	create temporal mapping parameters

#### **Classes**

Path(*args, **kwargs)	PurePath subclass that can make system calls.
Sets(settings)	Generates an initial batch of sets that are used to solve
	electricity model.

## class src.models.electricity.scripts.preprocessor.Sets(settings)

Generates an initial batch of sets that are used to solve electricity model. Sets include:

- Scenario descriptor and model switches
- · Regional sets
- · Temporal sets
- Technology type sets
- Supply curve step sets
- Other

src.models.electricity.scripts.preprocessor.add\_season\_index(cw\_temporal, df, pos)
 adds a season index to the input dataframe

#### **Parameters**

- **cw\_temporal** (*dataframe*) dataframe that includes the season index
- **df** (*dataframe*) parameter data to be modified
- **pos** (*int*) column position for the seasonal set

#### Returns

modified parameter data now indexed by season

## Return type

dataframe

src.models.electricity.scripts.preprocessor.avg\_by\_group(df, set\_name, map\_frame) takes in a dataframe and groups it by the set specified and then averages the data.

#### **Parameters**

- **df** (dataframe) parameter data to be modified
- **set\_name** (*str*) name of the column/set to average the data by
- map\_frame (dataframe) data that maps the set name to the new grouping for that set

#### Returns

parameter data that is averaged by specified set mapping

## **Return type**

dataframe

src.models.electricity.scripts.preprocessor.capacitycredit\_df(all\_frames, setin)
builds the capacity credit dataframe

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- setin (Sets) an initial batch of sets that are used to solve electricity model

#### Returns

formatted capacity credit data frame

## Return type

pd.DataFrame

src.models.electricity.scripts.preprocessor.create\_hourly\_params(all\_frames, key, cols)

Expands params that are indexed by season to be indexed by hour

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- **key** (*str*) name of data frame to access
- cols (list[str]) column names to keep in data frame

#### Returns

data frame with name key with new hourly index

#### **Return type**

pd.DataFrame

src.models.electricity.scripts.preprocessor.create\_hourly\_sets(all\_frames, df) expands sets that are indexed by season to be indexed by hour

## **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- df (pd.DataFrame) data frame containing seasonal data

#### Returns

data frame containing updated hourly set

## **Return type**

pd.DataFrame

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- setin (Sets) an initial batch of sets that are used to solve electricity model

#### Returns

updated Sets which has non-supply curve-related sets updated

#### Return type

Sets

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- **setin** (Sets) an initial batch of sets that are used to solve electricity model

#### Returns

updated Set containing all sets related to supply curve

#### Return type

Sets

src.models.electricity.scripts.preprocessor.create\_subsets(df, col, subset)

Create subsets off of full sets

#### **Parameters**

- **df** (pd.DataFrame) data frame of full data
- col (str) column name
- **subset** (list[str]) names of values to subset

#### Returns

data frame containing subset of full data

#### **Return type**

pd.DataFrame

src.models.electricity.scripts.preprocessor.fill\_values(row, subset\_list)

Function to fill in the subset values, is used to assign all years within the year solve range to each year the model will solve for.

#### **Parameters**

- row (int) row number in df
- **subset\_list** (*list*) list of values to map

## Returns

value from subset\_list

## Return type

int

src.models.electricity.scripts.preprocessor.hourly\_sc\_subset(all\_frames, subset)

Creates sets/subsets that are related to the supply curve

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- **subset** (*list*) list of technologies to subset

#### Returns

data frame containing sets/subsets related to supply curve

#### Return type

pd.DataFrame

src.models.electricity.scripts.preprocessor.hr\_sub\_sc\_subset(all\_frames, T\_subset, hr\_subset)
 creates supply curve subsets by hour

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- **T\_subset** (*list*) list of technologies to subset
- **hr\_subset** (*list*) list of hours to subset

#### Returns

data frame containing supply curve related hourly subset

## **Return type**

pd.DataFrame

src.models.electricity.scripts.preprocessor.load\_data(tablename, metadata, engine)

loads the data from the SQL database; used in readin\_sql function.

#### **Parameters**

- **tablename** (*string*) table name
- metadata (SQL metadata) SQL metadata
- engine (SQL engine) SQL engine

#### Returns

table from SQL db as a dataframe

#### **Return type**

dataframe

src.models.electricity.scripts.preprocessor.makedir(dir\_out)

creates a folder directory based on the path provided

#### **Parameters**

**dir\_out** (*str*) – path of directory

src.models.electricity.scripts.preprocessor.output\_inputs(OUTPUT\_ROOT)

function developed initial for QA purposes, writes out to csv all of the dfs and sets passed to the electricity model to an output directory.

## **Parameters**

**OUTPUT\_ROOT** (str) – path of output directory

#### Returns

- all\_frames (dictionary) dictionary of dataframes where the key is the file name and the value is the table data
- setin (Sets) an initial batch of sets that are used to solve electricity model

## src.models.electricity.scripts.preprocessor.preprocessor(setin)

main preprocessor function that generates the final dataframes and sets sent over to the electricity model. This function reads in the input data, modifies it based on the temporal and regional mapping specified in the inputs, and gets it into the final formatting needed. Also adds some additional regional sets to the set class based on parameter inputs.

#### **Parameters**

**setin** (Sets) – an initial batch of sets that are used to solve electricity model

#### Returns

- all\_frames (dictionary) dictionary of dataframes where the key is the file name and the value is the table data
- setin (Sets) an initial batch of sets that are used to solve electricity model

## src.models.electricity.scripts.preprocessor.print\_sets(setin)

function developed initially for QA purposes, prints out all of the sets passed to the electricity model.

#### **Parameters**

setin (Sets) – an initial batch of sets that are used to solve electricity model

## src.models.electricity.scripts.preprocessor.readin\_csvs(all\_frames)

Reads in all of the CSV files from the input dir and returns a dictionary of dataframes, where the key is the file name and the value is the table data.

#### **Parameters**

**all\_frames** (*dictionary*) – empty dictionary to be filled with dataframes

#### Returns

completed dictionary filled with dataframes from the input directory

#### Return type

dictionary

## src.models.electricity.scripts.preprocessor.readin\_sql(all\_frames)

Reads in all of the tables from a SQL databased and returns a dictionary of dataframes, where the key is the table name and the value is the table data.

#### **Parameters**

**all\_frames** (*dictionary*) – empty dictionary to be filled with dataframes

#### Returns

completed dictionary filled with dataframes from the input directory

## Return type

dictionary

creates supply curve subsets by step

#### **Parameters**

- all\_frames (dict of pd.DataFrame) dictionary of dataframes where the key is the file name and the value is the table data
- **T\_subset** (*list*) technologies to subset

• **step\_subset** (*list*) – step numbers to subset

#### Returns

data frame containing supply curve subsets by step

## **Return type**

pd.DataFrame

src.models.electricity.scripts.preprocessor.subset\_dfs(all\_frames, setin, i)

filters dataframes based on the values within the set

#### **Parameters**

- all\_frames (dictionary) dictionary of dataframes where the key is the file name and the value is the table data
- setin (Sets) contains an initial batch of sets that are used to solve electricity model
- i (string) name of the set contained within the sets class that the df will be filtered based on.

#### Returns

completed dictionary filled with dataframes filtered based on set inputs specified

## **Return type**

dictionary

#### **Parameters**

- cw\_temporal (pd.DataFrame) temporal crosswalks
- rename\_cols (dict) columns to rename from/to

#### Returns

data frame with temporal mapping parameters

## **Return type**

pd.DataFrame

## src.models.electricity.scripts.runner

This file is a collection of functions that are used to build, run, and solve the electricity model.

#### **Functions**

build_elec_model(all_frames, setin)	building pyomo electricity model
<pre>check_results(results, SolutionStatus,)</pre>	Check results for termination condition and solution sta-
	tus
<pre>cost_learning_func(instance, tech, y)</pre>	function for updating learning costs by technology and
	year
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if
	necessary.
<pre>init_old_cap(instance)</pre>	initialize capacity for 0th iteration
<pre>log_infeasible_constraints(m[, tol, logger,])</pre>	Logs the infeasible constraints in the model.
<pre>run_elec_model(settings[, solve])</pre>	build electricity model (and solve if solve=True) after
	passing in settings

continues on next page

Table 11 – continued from previous page

select_solver(instance)	Select solver based on learning method
<pre>set_new_cap(instance)</pre>	calculate new capacity after solve iteration
<pre>solve_elec_model(instance)</pre>	solve electicity model
<pre>update_cost(instance)</pre>	update capital cost based on new capacity learning

## **Classes**

Config_settings(config_path[, args, test])	Generates the model settings that are used to solve.
Path(*args, **kwargs)	PurePath subclass that can make system calls.
PowerModel(*args, **kwds)	A PowerModel instance.
SolutionStatus(*values)	
SolverStatus(*values)	
TerminationCondition(*values)	
TicTocTimer([ostream, logger])	A class to calculate and report elapsed time.
<pre>datetime(year, month, day[, hour[, minute[,)</pre>	The year, month and day arguments are required.

 $\verb|src.models.electricity.scripts.runner.build_elec_model| (all\_frames, setin) \rightarrow PowerModel| \\building pyomo electricity model|$ 

## **Parameters**

- all\_frames (dict of pd.DataFrame) input data frames
- **setin** (Sets) input settings Sets

#### Returns

built (but unsolved) electricity model

## Return type

**PowerModel** 

src.models.electricity.scripts.runner.cost\_learning\_func(instance, tech, y)

function for updating learning costs by technology and year

#### **Parameters**

- instance (PowerModel) electricity pyomo model
- **tech** (*int*) technology type
- **y** (int) year

#### Returns

updated capital cost based on learning calculation

## Return type

int

src.models.electricity.scripts.runner.init\_old\_cap(instance)

initialize capacity for 0th iteration

#### **Parameters**

instance (PowerModel) - unsolved electricity model

 $\verb|src.models.electricity.scripts.runner.run_elec_model| (settings: Config_settings, solve=True) \rightarrow PowerModel|$ 

build electricity model (and solve if solve=True) after passing in settings

#### **Parameters**

- settings (Config\_settings) Configuration settings
- solve (bool, optional) solve electricity model?, by default True

#### Returns

electricity model

#### Return type

**PowerModel** 

src.models.electricity.scripts.runner.set\_new\_cap(instance)
 calculate new capacity after solve iteration

#### **Parameters**

instance (PowerModel) - solved electricity pyomo model

src.models.electricity.scripts.runner.solve\_elec\_model(instance)
solve electicity model

#### **Parameters**

instance (PowerModel) - built (but not solved) electricity pyomo model

src.models.electricity.scripts.runner.update\_cost(instance)

update capital cost based on new capacity learning

#### **Parameters**

instance (PowerModel) - electricity pyomo model

## src.models.electricity.scripts.utilities

This file is a collection of functions that are used in support of the electricity model.

#### **Functions**

annual_count(hour, m)	return the aggregate weight of this hour in the represen- tative year we know the hour weight, and the hours are unique to days, so we can get the day weight
<pre>check_results(results, SolutionStatus,)</pre>	Check results for termination condition and solution status
<pre>create_obj_df(mod_object)</pre>	takes pyomo component objects (e.g., variables, parameters, constraints) and processes the pyomo data and converts it to a dataframe and then writes the dataframe out to an output dir.

## **Classes**

<pre>ElectricityMethods(*args, **kwds)</pre>	a collection of functions used within the electricity model that aid in building the model.
Model(*args, **kwds)	This is the base model class for the models.
	continues on next page

Table 14 – continued from previous page

Path(*args, **kwargs)	PurePath subclass that can make system calls.
defaultdict	defaultdict(default_factory=None, /, [])> dict with
	default factory

class src.models.electricity.scripts.utilities.ElectricityMethods(\*args, \*\*kwds)

a collection of functions used within the electricity model that aid in building the model.

#### **Parameters**

Model (Class) – generic model class

\_active

## populate\_RM\_sets\_rule()

Creates new reindexed sets for reserve margin constraint

#### **Parameters**

**m** (PowerModel) – pyomo electricity model instance

## populate\_by\_hour\_sets\_rule()

Creates new reindexed sets for dispatch\_cost calculations

#### **Parameters**

**m** (PowerModel) – pyomo electricity model instance

#### populate\_demand\_balance\_sets\_rule()

Creates new reindexed sets for demand balance constraint

#### **Parameters**

**m** (PowerModel) – pyomo electricity model instance

#### populate\_hydro\_sets\_rule()

Creates new reindexed sets for hydroelectric generation seasonal upper bound constraint

## **Parameters**

m (PowerModel) – pyomo electricity model instance

## populate\_reserves\_sets\_rule()

Creates new reindexed sets for operating reserves constraints

#### **Parameters**

m (PowerModel) – pyomo electricity model instance

## populate\_trade\_sets\_rule()

Creates new reindexed sets for trade constraints

#### **Parameters**

m (PowerModel) - pyomo electricity model instance

## $src.models.electricity.scripts.utilities.annual\_count(hour, m) \rightarrow int$

return the aggregate weight of this hour in the representative year we know the hour weight, and the hours are unique to days, so we can get the day weight

#### **Parameters**

**hour** (int) – the rep\_hour

#### Returns

the aggregate weight (count) of this hour in the rep\_year. NOT the hour weight!

## Return type

int

Check results for termination condition and solution status

#### **Parameters**

- results (str) Results from pyomo
- **SolutionStatus** (*str*) Solution Status from pyomo
- **TerminationCondition** (*str*) Termination Condition from pyomo

## Return type

results

src.models.electricity.scripts.utilities.create\_obj\_df(mod\_object)

takes pyomo component objects (e.g., variables, parameters, constraints) and processes the pyomo data and converts it to a dataframe and then writes the dataframe out to an output dir. The dataframe contains a key column which is the original way the pyomo data is structured, as well as columns broken out for each set and the final values.

#### **Parameters**

mod\_object (pyomo component object) - pyomo component object

#### Returns

contains the pyomo model results for the component object

## Return type

pd.DataFrame

## src.models.hydrogen

#### **Modules**

model
network
utilities

## src.models.hydrogen.model

## **Modules**

actions	A sequencer for actions in the model.
h2_model	The Hydrogen Model takes in a Grid object and uses it to populate a Pyomo model that solves for the least cost to produce and distribute Hydrogen by electrolysis across the grid to satisfy a given demand, returning the duals as shadow prices.
validators	set of validator functions for use in model

## src.models.hydrogen.model.actions

A sequencer for actions in the model. This may change up a bit, but it is a place to assert control of the execution sequence for now

## **Functions**

build_grid(grid_data)	build a grid from grid_data
<pre>build_model(grid, **kwds)</pre>	build model from grd
<pre>check_optimal_termination(results)</pre>	This function returns True if the termination condition for the solver is 'optimal', 'locallyOptimal', or 'globally-Optimal', and the status is 'ok'
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if necessary.
<pre>load_data(path_to_input, **kwds)</pre>	load data for model
<pre>make_h2_outputs(output_path, model)</pre>	save model outputs
<pre>quick_summary(solved_hm)</pre>	print and return summary of solve
<pre>run_hydrogen_model(settings)</pre>	run hydrogen model in standalone
solve(hm)	_summary_
<pre>solve_it(hm)</pre>	solve hm
value(obj[, exception])	A utility function that returns the value of a Pyomo object or expression.

## **Classes**

```
Grid([data])

GridData(data_folder[, regions_of_interest])

H2Model(*args, **kwds)

Path(*args, **kwargs)

PurePath subclass that can make system calls.

SolverResults(*args, **kwargs)
```

```
src.models.hydrogen.model.actions.build_grid(grid_data: GridData) 	o Grid build a grid from grid_data  \begin{array}{c} \mathbf{Parameters} \\ \mathbf{grid\_data} \ (obj) - \mathbf{GridData} \ object \ to \ build \ grid \ from \\ \mathbf{Returns} \\ \mathbf{Grid} - \mathbf{Grid} \ object \end{array}
```

Return type obj

 $\verb|src.models.hydrogen.model.actions.build_model(|grid:|Grid,|**kwds)| \rightarrow H2Model| \\ build model from grd|$ 

## **Parameters**

grid(obj) – Grid object to build model from

```
Returns
              H2Model - H2Model object
          Return type
              obj
src.models.hydrogen.model.actions.load\_data(path\ to\ input:\ Path,\ **kwds) \rightarrow GridData
     load data for model
          Parameters
              path_to_input (Path) - Data folder path
          Returns
              GridData - Grid Data object from path
          Return type
              obj
src.models.hydrogen.model.actions.make_h2_outputs(output_path, model)
     save model outputs
          Parameters
              model (obj) – Solved H2Model
src.models.hydrogen.model.actions.quick\_summary(solved\_hm: H2Model) \rightarrow None
     print and return summary of solve
          Parameters
              solved_hm(obj) - Solved H2Model
          Returns
              res – Printed summary
          Return type
              str
src.models.hydrogen.model.actions.run_hydrogen_model(settings)
     run hydrogen model in standalone
          Parameters
              settings (obj) – Config setup instance
src.models.hydrogen.model.actions.solve_it(hm: H2Model) → SolverResults
     solve hm
          Parameters
              hm (objH2Mode1) – H2Model to solve
          Returns
              SolverResults – results of solve
          Return type
              obj
```

## src.models.hydrogen.model.h2\_model

The Hydrogen Model takes in a Grid object and uses it to populate a Pyomo model that solves for the least cost to produce and distribute Hydrogen by electrolysis across the grid to satisfy a given demand, returning the duals as shadow prices. It can be run in stand-alone or integrated runs. If stand-alone, a function for generated temporally varying data must be supplied. By default it simply projects geometric growth for electricity price and demand.

## **Functions**

<pre>check_optimal_termination(results)</pre>	This function returns True if the termination condition for the solver is 'optimal', 'locallyOptimal', or 'globally-Optimal', and the status is 'ok'
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if necessary.
resolve(hm[, new_demand,])	For convenience: After building and solving the model initially:
solve(hm)	_summary_
value(obj[, exception])	A utility function that returns the value of a Pyomo object or expression.

## **Classes**

Block(*args, **kwds)	Blocks are indexed components that contain other components (including blocks).
ConcreteModel(*args, **kwds)	A concrete optimization model that does not defer construction of components.
Constraint(*args, **kwds)	This modeling component defines a constraint expression using a rule function.
Grid([data])	
H2Model(*args, **kwds)	
HI(region, year)	(region, year)
LinearExpression([args, constant,])	An expression object for linear polynomials.
Objective(*args, **kwds)	This modeling component defines an objective expression.
Param(*args, **kwds)	A parameter value, which may be defined over an index.
RangeSet(*args, **kwds)	A set object that represents a set of numeric values
Set(*args, **kwds)	A component used to index other Pyomo components.
SolverResults(*args, **kwargs)	
Suffix(*args, **kwargs)	A model suffix, representing extraneous model data
Var(*args, **kwargs)	A numeric variable, which may be defined over an index.
defaultdict	defaultdict(default_factory=None, /, [])> dict with default factory

class src.models.hydrogen.model.h2\_model.H2Model(\*args, \*\*kwds)

## \_active

```
_filter_update_info(data: dict[HI, float]) \rightarrow dict[HI, float] quick filter to remove regions that don't exist in the model
```

It is possible (right now) that the H2 network is unaware of particular regions

because no baseline data for them was ever provided.... so it is possible to recieve and "unkown" region here, even though it was selected, due to lack of data

## **Parameters**

```
• hm (H2Model) - self
                   • data (dict[HI, float]) - hydrogen index : value
               Returns
                  regions index: value with missing data removed
               Return type
                  dict[HI, float]
     _update_demand(new_demand)
          update the demand parameter with new demand data
               insert new demand as a dict in the format: new_demand[region, year]
               Parameters
                   • hm (H2Model) - self
                   • new_demand (dict) - new demand values
     _update_electricity_price(new_electricity_price)
          update electricity price parameter
               Parameters
                   • hm (H2Model) - self
                   • new_electricity_price (dict) – region, year : electricity price
     poll_electric_demand() \rightarrow dict[HI, float]
          compute the electrical demand by region-year after solve
          Note: we will use production * 1/eff to compute electrical demand
               Parameters
                  hm (H2Model) - self
               Returns
                  electricity demand by region, year. (region, year):demand
               Return type
                  dict[HI, float]
     update_exchange_params(new demand=None, new electricity price=None)
          update exchange parameters in integrated mode
               Parameters
                   • hm (H2Model) – model
                   • new_demand (dict, optional) - new demand (region, year):value. Defaults to None.
                   • new_electricity_price (dict, optional) - new electricity prices (re-
                     gion, year): value. Defaults to None.
src.models.hydrogen.model.h2_model.resolve(hm: H2Model, new_demand=None,
                                                   new_electricity_price=None, test=False)
```

1.4. Model Structure 19

if you want to solve without annual data by applying a geometric growth rate to exhcange params

For convenience: After building and solving the model initially:

**Parameters** 

```
• hm (H2Model) - model
```

- **new\_demand** (*dict*, *optional*) new\_demand[region,year] for H2demand in (region,year). Defaults to None.
- **new\_electricity\_price** (*dict*, *optional*) new\_electricity\_price[region,year]. Defaults to None.
- **test** (*bool*, *optional*) is this just a test? Defaults to False.

```
src.models.hydrogen.model.h2_model.solve(hm: H2Model)
    _summary_
```

*j* \_

**Parameters** 

hm (H2Model) – self

Raises

**RuntimeError** – no optimal solution to problem

## src.models.hydrogen.model.validators

set of validator functions for use in model

#### **Functions**

region_validator(hm, region)	checks if region name is string or numeric

```
src.models.hydrogen.model.validators.region_validator(hm: H2Model, region)
```

checks if region name is string or numeric

hm

[H2Model] model

region

[any] region name

Raises

**ValueError** – region wrong type

Returns

is correct type

Return type

bool

## src.models.hydrogen.network

## **Modules**

grid	GRID CLASS
grid_data	GRIDDATA CLASS
hub	HUB CLASS
region	REGION CLASS
registry	REGISTRY CLASS
transportation_arc	TRANSPORTATION ARC CLASS

## src.models.hydrogen.network.grid

## **GRID CLASS**

This is the central class that binds all the other classes together. No class instance exists in a reference that isn't fundamentally contained in a grid. The grid is used to instantiate a model, read data, create the regionality and hub / arc network within that regionality, assign data to objects and more.

notably, the grid is used to coordinate internal methods in various classes to make sure that their combined actions keep the model consistent and accomplish the desired task.

#### **Classes**

```
Grid([data])

GridData(data_folder[, regions_of_interest])

Hub(name, region[, data])

Region(name[, grid, kind, data, parent])

Registry()

TransportationArc(origin, destination, capacity)
```

**class** src.models.hydrogen.network.grid.**Grid**(data: GridData | None = None)

#### aggregate\_hubs(hublist, region)

combine all hubs in hublist into a single hub, and place them in region. Arcs that connect to any of these hubs also get aggegated into arcs that connect to the new hub and their original origin / destination that's not in hublist.

#### **Parameters**

- hublist (list) list of hubs to aggregate
- region (Region) region to place them in

## arc\_generation(df)

generate arcs from the arc data

#### **Parameters**

df (DataFrame) - arc data

#### build\_grid(vis=True)

builds a grid fom the GridData by recursively adding regions starting at top-level region 'world'.

## **Parameters**

**vis** (bool, optional) – if True, will generate an image of the hub-network with regional color-coding. Defaults to True.

## collapse(region\_name)

make a region absorb all it's sub-regions and combine all its and its childrens hubs into one

#### **Parameters**

**region\_name** (str) – region to collapse

#### collapse\_level(level)

collapse all regions at a specific level of depth in the regional hierarchy, with world = 0

#### **Parameters**

**level** (*int*) – level to collapse

## combine\_arcs(arclist, origin, destination)

combine a set of arcs into a single arc with given origin and destination

#### **Parameters**

- arclist (list) list of arcs to aggregate
- **origin** (str) new origin hub
- **destination** (*str*) new destination hub

## connect\_subregions()

create an arc for all hubs in bottom-level regions to whatever hub is located in their parent region

## create\_arc(origin, destination, capacity, cost=0.0)

Creates and arc from origin to destination with given capacity and cost

#### **Parameters**

- **origin** (*str*) origin hub name
- **destination** (str) destination hub name
- capacity (float) capacity of arc
- **cost** (*float*, *optional*) cost of transporting 1kg H2 along arc. Defaults to 0.

## create\_hub(name, region, data=None)

creates a hub in a given region

#### **Parameters**

- name (str) hub name
- region (Region) Region hub is placed in
- data (DataFrame, optional) dataframe of hub data to append. Defaults to None.

## create\_region(name, parent=None, data=None)

creates a region with a given name, parent region, and data

#### **Parameters**

- name (str) name of region
- parent (Region, optional) parent region. Defaults to None.
- data (DataFrame, optional) region data. Defaults to None.

## delete(thing)

deletes a hub, arc, or region

## **Parameters**

thing (Hub, Arc, or Region) - thing to delete

## load\_hubs()

load hubs from data

## recursive\_region\_generation(df, parent)

cycle through a region dataframe, left column to right until it hits data column, adding new regions and subregions according to how it is hierarchically structured. Future versions should implement this with a graph structure for the data instead of a dataframe, which would be more natural.

#### **Parameters**

- **df** (DataFrame) hierarchically structured dataframe of regions and their data.
- parent (Region) Parent region

```
test()
test run
visualize()
visualize the grid network using graphx
write_data()
_write data to file
```

## src.models.hydrogen.network.grid\_data

#### **GRIDDATA CLASS**

grid\_data is the the data object that grids are generated from. It reads in raw data with a region grid\_data is the the data object that grids are generated from. It reads in raw data with a region filter, and holds it in one structure for easy access

#### Classes

```
GridData(data_folder[, regions_of_interest])

Path(*args, **kwargs)

PurePath subclass that can make system calls.
```

## src.models.hydrogen.network.hub

## **HUB CLASS**

class objects are individual hubs, which are fundamental units of production in the model. Hubs belong to regions, and connect to each other with transportation arcs.

#### **Classes**

```
Hub(name, region[, data])
```

class src.models.hydrogen.network.hub.Hub(name, region, data=None)

```
add_inbound(arc)
     add an inbound arc to hub
         Parameters
             arc (Arc) – add an inbound arc to hub
add_outbound(arc)
     add an outbound arc to hub
         Parameters
             arc (Arc) - arc to add
change_region(new_region)
     move hub to new region
         Parameters
             new_region (Region) – region hub should be moved to
cost(technology, year)
     return a cost value in terms of data fields
         Parameters
             • technology (str) – technology type
             • year (int) - year
         Returns
             a cost value
         Return type
             float
display_outbound()
     print all outbound arcs from hub
get_data(quantity)
     fetch quantity from hub data
         Parameters
             quantity (str) – name of data field to fetch
         Returns
             quantity to be fetched
         Return type
             float or str
remove_inbound(arc)
     remove an inbound arc from hub
         Parameters
             arc (Arc) – arc to remove
remove_outbound(arc)
     remove an outbound arc from hub
         Parameters
```

arc (Arc) - arc to remove

## src.models.hydrogen.network.region

#### **REGION CLASS**

Class objects are regions, which have a natural tree-structure. Each region can have a parent region and child regions (subregions), a data object, and a set of hubs.

## **Functions**

getLogger([name])	Return a logger with the specified name, creating it if
	necessary.

#### **Classes**

```
Region(name[, grid, kind, data, parent])
```

```
absorb_subregions()
```

delete subregions, acquire their hubs and subregions

```
absorb_subregions_deep()
```

absorb subregions recursively so that region becomes to the deepest level in the hierarchy

## add\_hub(hub)

add a hub to region

## **Parameters**

hub (Hub) - hub to add

## add\_subregion(subregion)

make a region a subregion of self

#### **Parameters**

subregion (Region) – new subregion

## aggregate\_subregion\_data(subregions)

combine the data from subregions and assign it to self

#### Parameters

**subregions** (list) – list of subregions

assigned\_names = {}

create\_subregion(name, data=None)

create a subregion

## **Parameters**

- name(str) subregion name
- data (DataFrame, optional) subregion data. Defaults to None.

```
delete()
     delete self, reassign hubs to parent, reassign children to parent
display_children()
     display child regions
display_hubs()
     display hubs
get_data(quantity)
     pull data from region data
         Parameters
             quantity (str) - name of data field in region data
         Returns
             value of data
         Return type
             str, float
remove_hub(hub)
     remove hub from region
         Parameters
             hub (Hub) – hub to remove
remove_subregion(subregion)
     remove a subregion from self
         Parameters
             subregion (Region) – subregion to remove
update_data(df)
     change region data
         Parameters
             df (DataFrame) - new data
update_parent(new_parent)
     change parent region
         Parameters
             new_parent (Region) - new parent region
```

## src.models.hydrogen.network.registry

## **REGISTRY CLASS**

This class is the central registry of all objects in a grid. It preserves them in dicts of object-name:object so that they can be looked up by name. it also should serve as a place to save data in different configurations for faster parsing - for example, depth is a dict that organizes regions according to their depth in the region nesting tree.

## Classes

Hub(name, region[, data])

continues on next page

Table 28 – continued from previous page

```
Region(name[, grid, kind, data, parent])
 Registry()
 TransportationArc(origin, destination, capacity)
class src.models.hydrogen.network.registry.Registry
     add(thing)
          add a thing to the registry. Thing can be Hub, Arc, or Region
              Parameters
                  thing (Arc, Region, or Hub) – thing to add to registry
              Returns
                  thing being added gets returned
              Return type
                  Arc, Region, or Hub
     remove(thing)
          remove thing from registry
              Parameters
                  thing (Arc, Hub, or Region) - thing to remove
     update_levels()
          update dictionary of regions by level
src.models.hydrogen.network.transportation_arc
TRANSPORTATION ARC CLASS
objects in this class represent individual transportation arcs. An arc can exist with zero capacity, so they only represent
possible arcs.
Classes
 TransportationArc(origin, destination, capacity)
class src.models.hydrogen.network.transportation_arc.TransportationArc(origin, destination,
                                                                                    capacity, cost=0)
```

1.4. Model Structure 27

**new\_destination** (Hub) – new destination hub

**change\_destination**(new\_destination) change the destination hub of arc

**Parameters** 

change\_origin(new\_origin)

change the origin hub of arc

#### **Parameters**

new\_origin (Hub) - new origin hub

#### disconnect()

disconnect arc from it's origin and destination

## src.models.hydrogen.utilities

#### **Modules**

h2_functions	This file is a collection of functions that are used in sup-
	port of the hydrogen model.

## src.models.hydrogen.utilities.h2\_functions

This file is a collection of functions that are used in support of the hydrogen model.

## **Functions**

<pre>get_demand(hm, region, time)</pre>	get demand for region at time.
<pre>get_elec_price(hm, region, year)</pre>	get electricity price in region, year
<pre>get_electricity_consumption_rate(hm, tech)</pre>	the electricity consumption rate for technology type tech
<pre>get_electricty_consumption(hm, region, year)</pre>	get electricity consumption for region, year
<pre>get_gas_price(hm, region, year)</pre>	get gas price for region, year
<pre>get_production_cost(hm, hub, tech, year)</pre>	return production cost for tech at hub in year

src.models.hydrogen.utilities.h2\_functions.get\_demand(hm: H2Model, region, time) get demand for region at time. If mode not standard, just increase demand by 5% per year

#### **Parameters**

- hm (H2Model) model
- **region** (str) region
- time (int) year

#### Returns

demand

## **Return type**

float

src.models.hydrogen.utilities.h2\_functions.get\_elec\_price(hm: H2Model, region, year)
get electricity price in region, year

#### **Parameters**

- hm (H2Model) \_model
- region (str) region
- year (int) year

## Returns

electricity price in region and year

## Return type

float

the electricity consumption rate for technology type tech

#### **Parameters**

- hm (H2Model) model
- **tech** (*str*) technology type

#### **Returns**

GWh per kg H2

## Return type

float

get electricity consumption for region, year

#### **Parameters**

- hm (H2Model) model
- region(str) region
- year (int) year

#### Returns

the elecctricity consumption for a region and year in the model

## Return type

float

src.models.hydrogen.utilities.h2\_functions.get\_gas\_price(hm: H2Model, region, year)
get gas price for region, year

#### **Parameters**

- hm (H2Model) model
- region (str) region
- year (int) year

## Returns

gas price in region and year

## Return type

float

src.models.hydrogen.utilities.h2\_functions.**get\_production\_cost**(*hm:* H2Model, *hub*, *tech*, *year*) return production cost for tech at hub in year

#### **Parameters**

- hm (H2Model) model
- **hub** (str) hub
- **tech** (*str*) technology type
- year (int) year

#### Returns

production cost of H2 for tech at hub in year

## **Return type**

float

#### src.models.residential

#### **Modules**

preprocessor		
scripts		

## src.models.residential.preprocessor

#### **Modules**

generate_inputs	This file contains the options to re-create the input files.
	It creates:

## src.models.residential.preprocessor.generate inputs

## This file contains the options to re-create the input files. It creates:

- Load.csv: electricity demand for all model years (used in residential and electricity)
- BaseElecPrice.csv: electricity prices for initial model year (used in residential only)

Uncomment out the functions at the end of this file in the "if \_\_name\_\_ == '\_\_main\_\_'" statement in order to generate new load or base electricity prices.

## **Functions**

base_price()	Runs the electricity model with base price configuration settings and then merges the electricity prices and tem- poral crosswalk data produced from the run to generate
	base year electricity prices.
main([settings])	Runs model as defined in settings

## **Classes**

Path ("args, ""kwargs) Purerain subclass that can make system cans.	Path(*args, **kwargs)	PurePath subclass that can make system calls.
---	-----------------------	---

## src.models.residential.preprocessor.generate\_inputs.base\_price()

Runs the electricity model with base price configuration settings and then merges the electricity prices and temporal crosswalk data produced from the run to generate base year electricity prices.

## **Returns**

dataframe that contains base year electricity prices for all regions/hours

#### Return type

pandas.core.frame.DataFrame

## src.models.residential.scripts

#### **Modules**

residential	Residential Model.

## src.models.residential.scripts.residential

Residential Model. This file contains the residential Module class which contains a representation of residential electricity prices and demands.

#### **Functions**

getLogger([name])	Return a logger with the specified name, creating it if necessary.
<pre>run_residential(settings)</pre>	This runs the residential model in stand-alone mode.
scale_load(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year) and LoadScalar.csv (a multiplier for all model years).
scale_load_with_enduses(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year), EnduseBaseShares.csv (the shares of demand for each enduse in the base year) and EnduseScalar.csv (a multiplier for all model years by enduse category).

## **Classes**

Config_settings(config_path[, args, test])	Generates the model settings that are used to solve.
Path(*args, **kwargs)	PurePath subclass that can make system calls.
residentialModule([settings, loadFile,])	This contains the Residential model and its associated functions.

class src.models.residential.scripts.residential.residentialModule(settings: Config\_settings |

None = None, loadFile: str | None = None, load\_df: DataFrame | None = None, calibrate: bool | None = False)

This contains the Residential model and its associated functions. Once an object is instantiated, it can calculate new Load values for updated prices. It can also calculate estimated changes to the Load if one of the input variables is changed by a specified percent. The model will be created in a symbolic form to be easily manipulated, and then values can be filled in for calculations.

## baseYear = 0

## complex\_step\_sensitivity(prices, change\_var, percent)

This estimates how much the output Load will change due to a change in one of the input variables. It can calculate these values for changes in price, price elasticity, income, income elasticity, or long term trend. The Load calculation requires input prices, so this function requires that as well for the base output Load.

Then, an estimate for Load is calculated for the case where the named 'change\_var' is changed by 'percent' %.

#### **Parameters**

- prices (dataframe or Pyomo Indexed Parameter) Price values used to calculate the Load value
- change\_var (string) -

#### Name of variable of interest for sensitivity. This can be:

```
'income', 'i_elas', 'price', 'p_elas', 'trendGR'
```

• **percent** (*float*) – A value 0 - 100 for the percent that the variable of interest can change.

#### Returns

Indexed values for the calculated Load at the given prices, the Load if the variable of interest is increased by 'percent'%, and the Load if the variable of interest is decreased by 'percent'%

## Return type

dataframe

```
demandF(price, load, year, basePrice=1, p_elas=-0.1, baseYear=None, baseIncome=1, income=1, i_elas=1, trend=0, priceIndex=1, incomeIndex=1, p_lag=1, i_lag=1)
```

The demand function. Wraps the sympy demand function with some defaults

#### **Parameters**

```
• price (_type_) - _description_
```

```
• load (_type_) - _description_
```

- basePrice (int, optional) \_description\_, by default 1
- **p\_elas** (*float*, *optional*) \_description\_, by default -0.10
- baseYear (\_type\_, optional) \_description\_, by default None
- baseIncome (int, optional) \_description\_, by default 1
- income (int, optional) \_description\_, by default 1
- i\_elas (int, optional) \_description\_, by default 1
- **trend** (*int*, *optional*) \_description\_, by default 0
- **priceIndex** (int, optional) \_description\_, by default 1
- incomeIndex (int, optional) \_description\_, by default 1
- **p\_lag** (int, optional) \_description\_, by default 1
- i\_lag (int, optional) \_description\_, by default 1

## Returns

```
_description_
```

## Return type

\_type\_

```
hr_map = Empty DataFrame Columns: [] Index: []
```

 $loads = {}$ 

### make\_block(prices, pricesindex)

Updates the value of 'Load' based on the new prices given. The new prices are fed into the equations from the residential model. The new calculated Loads are used to constrain 'Load' in pyomo blocks.

#### **Parameters**

- prices (pyo.Param) Pyomo Parameter of newly updated prices
- pricesindex (pyo. Set) Pyomo Set of indexes that matches the prices given

### Returns

Block containing constraints that set 'Load' variable equal to the updated load values

# Return type

pyo.Block

```
prices = {}
```

### sensitivity(prices, change\_var, percent)

This estimates how much the output Load will change due to a change in one of the input variables. It can calculate these values for changes in price, price elasticity, income, income elasticity, or long term trend. The Load calculation requires input prices, so this function requires that as well for the base output Load. Then, an estimate for Load is calculated for the case where the named 'change\_var' is changed by 'percent'  $\frac{1}{100}$ .

#### **Parameters**

- prices (dataframe or Pyomo Indexed Parameter) Price values used to calculate the Load value
- change\_var (string) -

# Name of variable of interest for sensitivity. This can be:

```
'income', 'i_elas', 'price', 'p_elas', 'trendGR'
```

• **percent** (*float*) – A value 0 - 100 for the percent that the variable of interest can change.

### Returns

Indexed values for the calculated Load at the given prices, the Load if the variable of interest is increased by 'percent'%, and the Load if the variable of interest is decreased by 'percent'%

# Return type

dataframe

### update\_load(p)

Takes in Dual pyomo Parameters or dataframes to update Load values

### **Parameters**

p (pyo. Param) – Pyomo Parameter or dataframe of newly updated prices from Duals

### Returns

Load values indexed by region, year, and hour

# Return type

pandas DataFrame

```
view_output_load(values: DataFrame, regions: list[int] = [1], years: list[int] = [2023])
```

This is used to display the updated Load values after calculation. It will create a graph for each region and year combination.

# **Parameters**

• values (pd.DataFrame) - The Load values calculated in update\_load

- regions (list[int], optional) The regions to be displayed
- years (list[int], optional) The years to be displayed

view\_sensitivity(values: DataFrame, regions: list[int] = [1], years: list[int] = [2023])

This is used by the sensitivity method to display graphs of the calculated values

### **Parameters**

- values (pd.DataFrame) indexed values for the Load, upper change, and lower change
- regions (list[int], optional) regions to be graphed
- years (list[int], optional) years to be graphed

src.models.residential.scripts.residential.run\_residential(settings: Config\_settings)

This runs the residential model in stand-alone mode. It can run update\_load to calculate new Load values based on prices, or it can calculate the new Load value along with estimates for the Load if one of the input variables changes.

### **Parameters**

settings (Config\_settings) – information given from run\_config to set several values

# 1.4.2 src

This directory contains the subdirectory for the integrator module and the subdirectories for the sectoral modules. The integrator module contains code for the different solve methods (standalone, iterative, or unified). The sectoral modules currently represented in this prototype include the electricity model, hydrogen model, and residential model. For more details on each of these, please see the READMEs located within each subdirectory.

## **Modules**

common	
integrator	
models	
sensitivity	

#### src.common

# **Modules**

config_setup	This file contains Config_settings class.
model	Establish a base model class for the sectoral modules to
	inherit.
utilities	A gathering of utility functions for dealing with model interconnectivity

# src.common.config\_setup

This file contains Config\_settings class. It establishes the main settings used when running the model. It takes these settings from the run\_config.toml file. It has universal configurations (e.g., configs that cut across modules and/or solve options) and module specific configs.

# **Functions**

create_temporal_mapping(sw_temporal)	Combines the input mapping files within the electricity model to create a master temporal mapping dataframe.
make_dir(dir_name)	generates an output directory to write model results, output directory is the date/time at the time this function executes.

### **Classes**

Config_settings(config_path[, args, test])	Generates the model settings that are used to solve.
Path(*args, **kwargs)	PurePath subclass that can make system calls.
<pre>datetime(year, month, day[, hour[, minute[,)</pre>	The year, month and day arguments are required.

Generates the model settings that are used to solve. Settings include:

- Iterative Solve Config Settings
- Spatial Config Settings
- Temporal Config Settings
- · Electricity Config Settings
- Other

# \_additional\_year\_settings(name, value)

Checks year related settings to see if values are within expected ranges and updates other settings linked to years if years is changed.

### **Parameters**

- name (str) attribute name
- **value** (\_type\_) attribute value

### Raises

**TypeError** – Error

# \_check\_elec\_expansion\_settings(name, value)

Checks that switches for reserve margin and learning are on only if expansion is on.

## **Parameters**

- **name** (*str*) attribute name
- **value** (\_type\_) attribute value

### Raises

**TypeError** – Error

\_check\_int(name, value)

Checks if attribute is an integer

### **Parameters**

- name (str) attribute name
- value (\_type\_) attribute value

### **Raises**

TypeError - Error

\_check\_regions(name, value)

Checks to see if region is between the current default values of 1 and 25.

### **Parameters**

- **name** (*str*) attribute name
- **value** (\_type\_) attribute value

#### Raises

TypeError - Error

\_check\_res\_settings(name, value)

Checks if view year or region settings are subsets of year or region

# **Parameters**

- name (str) attribute name
- value (\_type\_) attribute value

# Raises

TypeError - Error

\_check\_true\_false(name, value)

Checks if attribute is either true or false

# **Parameters**

- **name** (*str*) attribute name
- **value** (\_type\_) attribute value

# **Raises**

**TypeError** – Error

\_check\_zero\_one(name, value)

Checks if attribute is either zero or one

#### **Parameters**

- name (str) attribute name
- **value** (\_type\_) attribute value

# Raises

TypeError-Error

## \_has\_all\_attributes(attrs: set)

Determines if all attributes within the set exist or not

## **Parameters**

**attrs** (*set*) – set of setting attributes

#### Returns

True or False

### Return type

bool

### src.common.model

Establish a base model class for the sectoral modules to inherit.

#### **Functions**

getLogger([name])	Return a logger with the specified name, creating it if
	necessary.

#### Classes

Model(*args, **kwds)	This is the base model class for the models.
defaultdict	defaultdict(default_factory=None, /, [])> dict with default factory

# class src.common.model.Model(\*args, \*\*kwds)

This is the base model class for the models.

This class contains methods for declaring pyomo components, extracting duals, and decorating expressions. The model class methods and attributes provide functionality for keeping track of index labels and ordering for all pyomo components; this is essential for integration tasks without the use of hard-coded indices and allows for easy post-processing tasks.

# class ConstraintExpression(\*args, \*\*kwargs)

Constraint Expression decorator that works the same as pyomo decorators, while keeping column dictionary updated for any indexed parameters given.

# class DefaultDecorator(model, \*args, \*\*kwargs)

Default decorator class that handles assignment of model scope/pointer in order to use pyomo-style parameter and constraint decorators.

Upon initialization, the decorator handles model assignment at class level to ensure inheriting classes have access to the models within local scope.

# classmethod assign\_model(model)

Class-method that assigns a model instance to DefaultDecorator

#### Parameters

model (pyo.ConcreteModel) – A pyo model instance

### class ParameterExpression(\*args, \*\*kwargs)

Parameter Expression decorator that works the same as pyomo decorators, while keeping column dictionary updated for any indexed parameters given.

\_active

```
_declare_set_with_dict(sname: str, sdata: Dict | DefaultDict, scols: MutableSequence, return_set: bool | None = False, switch: bool | None = True, create_indexed_set: bool | None = True, use values: bool | None = False) \rightarrow Set
```

Declares a pyomo Set object named 'sname' using input index values and labels.

Function takes a dictionary argument and creates pyomo set object from keys, values, or both.

If an indexed set is desired, set create\_indexed\_set to True; the function will create an indexed set with its own indices set as keys. Otherwise, an Ordered Scalar Set will be created, either from the keys or the values of 'sdata' depending on the value for 'use\_values' (False for keys, True for values).

Names for the indices handled by scols; user must provide.

### **Parameters**

- sname (str) Name of set
- sdata (Dict) Data object that contains set values
- **scols** (*Sequence | None, optional*) List of column names corresponding to index labels and position, by default None
- **return\_set** (bool | None, optional) Return the set rather than assign within function, by default False
- **switch** (bool | None, optional) Return None if False, by default True
- **create\_indexed\_set** (*bool | None*, *optional*) Indicator for whether output set should include values as well as new index (IndexedSets), by default True
- **use\_values** (*bool | None*, *optional*) If create\_indexed\_set is False, use the values of sdata rather than keys for pyo Set members, by default False

### Returns

Pyomo Set Object

# Return type

pyo.Set

```
_declare_set_with_iterable(sname: str, sdata: Sequence | Set | array, scols: Sequence[str] | None = None, return_set: bool | None = False, switch: bool | None = True) \rightarrow Set
```

Declares a pyomo Set object named 'sname' using input index values and labels.

Function can take iterable objects such as tuples, lists, etc as data inputs. Note that if the dimension of the index is larger than 1, user needs to provide a list of names for each set dimension.

# **Parameters**

- **sname** (*str*) Name of set
- sdata (Sequence) Data object that contains set values
- **scols** (*Sequence | None*, *optional*) List of column names corresponding to index labels and position, by default None
- **return\_set** (*bool | None*, *optional*) Return the set rather than assign within function, by default False
- switch (bool | None, optional) Return None if False, by default True

#### **Returns**

Pyomo Set Object

# Return type

pyo.Set

```
_declare_set_with_pandas(sname: str, sdata: DataFrame | Series, return_set: bool | None = False, switch: bool | None = True, use_columns: bool | None = False)
```

Declares a pyomo Set object named 'sname' using input index values and labels from a Pandas object.

Function assumes that the index values are the desired data to construct set object. User can specify whether to create set with column values instead

#### **Parameters**

- sname (str) Name of set
- sdata (MutableSequence | dict) Data object that contains set indices
- **return\_set** (*bool | None*, *optional*) Return the set rather than assign within function, by default False
- switch (bool | None, optional) Return None if False, by default True
- use\_columns (bool | None, optional) Use columns as indices for pyo set rather than row index, by default False

#### Returns

Pyomo Set Object

# Return type

pyo.Set

# classmethod build()

Default build command; class-level build to create and return an instance of Model.

This will work for any class inheriting the method, but it is recommended to replace this with model-specific build instructions if this functionality is desired.

# Returns

Instance of Model object

# Return type

Object

```
declare_ordered_time_set(sname: str, *sets: Set, return_set: bool | None = False)
```

Unnest the time sets into a single, unnested ordered, synchronous time set, an IndexedSet object keyed by the values in the time set, and an IndexedSet object keyed by the combined, original input sets.

These three set outputs are directly set as attributes of the model instance:

```
sname: (1,), (2,), ..., (N) sname_time_to_index: (1,):[set1, set2, set3], (2,):[set1, set2, set3] sname_index_to_time: (set1, set2, set3): [1], (set1, set2, set3): [2]
```

In summary, this function creates three sets, creating a unique, ordered set from input sets with the assumption that they are given to the function in hierarchical order. For example, for a desired time set that orders Year, Month, Day values, the args for the function should be provided as:

m. Year, m. Month, m. Day

Pyomo set products are used to unpack and create new set values that are ordered by the hierarchy provided:

```
(year1, month1, day1), (year1, month1, day2), ..., (year2, month1, day1), ... (yearY, monthM, dayD)
```

# **Parameters**

• **sname** (*str*) – Desired root name for the new sets

- **sets** (*pyo.Set*) A series of unnamed arguments assumed to contain pyo.Set in order of temporal hierarchy
- return\_set(bool | None, optional) Return the set rather than assign within function, by default False

### **Returns**

No return object; all sets assigned to model internally

### Return type

None

#### Raises

**ValueError** – "No sets provided in args; provide pyo. Set objects to use this function"

**declare\_param**( $pname: str, p\_set: Set \mid None, data: dict \mid DataFrame \mid Series \mid int \mid float, return\_param: bool \mid None = False, default: int \mid None = 0, mutable: bool \mid None = False) <math>\rightarrow$  Param

Declares a pyo Parameter component named 'pname' with the input data and index set.

Unpacks column dictionary of index set for param instance and creates pyo.Param; either assigns the value internally or returns the object based on return\_param.

### **Parameters**

- pname (str) Desired name of new pyo.Param instance
- p\_set (pyo.Set) Pyomo Set instance to index new Param
- data (dict | pd.DataFrame | pd.Series) Data to initialize Param instance
- return\_param (bool | None, optional) Return the param after function call rather than assign to self, by default False
- **default** (int | None, optional) pyo.Param keyword argument, by default 0
- mutable (bool | None, optional) pyo.Param keyword argument, by default False

## Returns

A pyomo Parameter instance

## Return type

pyo.Param

# Raises

**ValueError** – Raises error if input data not in format supported by function

declare\_set(sname: str, sdata: MutableSequence | DataFrame | Series | Dict, scols: MutableSequence |
 None = None, return\_set: bool | None = False, switch: bool | None = True, create\_indexed\_set:
 bool | None = True, use\_values: bool | None = False, use\_columns: bool | None = False)

Declares a pyomo Set object named 'sname' using input index values and labels.

Function handles input values and calls appropriate declare\_set methods based on data type of sdata

#### **Parameters**

- sname (str) Name of set
- sdata (Dict) Data object that contains set values
- **scols** (*Sequence | None, optional*) List of column names corresponding to index labels and position, by default None
- return\_set (bool | None, optional) Return the set rather than assign within function, by default False

- switch (bool | None, optional) Return None if False, by default True
- **create\_indexed\_set** (*bool | None*, *optional*) If dict, indicator for whether output set should include values as well as new index (IndexedSets), by default True
- use\_values (bool | None, optional) If dict and create\_indexed\_set is False, use the values of sdata rather than keys for pyo Set members, by default False
- use\_columns (bool | None, optional) If Pandas, use columns as indices for pyo set rather than row index, by default False

### **Returns**

Pyomo Set Object

# Return type

pyo.Set

 $declare\_set\_with\_sets(sname: str, *sets: Set, return\_set: bool | None = False, switch: bool | None = True) \rightarrow Set$ 

Declares a new set object using input sets as arguments.

Function creates a set product with set arguments to create a new set. This is how pyomo handles set creation with multiple existing sets as arguments.

However, this function finds each pyomo set in column dictionary and unpacks the names, so that the new set can be logged in the column dictionary too.

#### **Parameters**

- sname (str) Desired name of new set
- \*sets (tuple of pyo.Set) Unnamed arguments assumed to be pyomo sets
- **return\_set** (*bool | None*, *optional*) Return the set rather than assign within function, by default False
- switch (bool | None, optional) Return None if False, by default True

#### Returns

Pyomo Set Object

# Return type

pyo.Set

A generalize shifting function that creates sets compatible with leads or lags in pyomo components.

For example, with a storage constraint where the current value is contrained to be equal to the value of storage in the previous period:

```
model.storage[t] == model.storage[t-1] + ...
```

The indexing set must be consistent with the storage variable, but not include elements that are undefined for this constraint. In this example, the set containing values for t must not include t = 1 (e.g. the lagged value must be defined). This function creates a shifted time set by removing values from the input sets to comply with the lags or leads.

Function inputs require a shift size (in the example above, this would be 1), a shift type (lead or lag), and the sets used to construct the new, shifted set (model.timestep). If a lag or lead is required on a single dimension of the new set, the 'shift\_sets' argument can include a list of pyo.set names (included in the arguments) to shift by the other args.

For example...

```
model.storage[hub, season] == model.storage[hub, season - 1]
```

In this case, season = 1 is always invalid due to the lag; so index (1, 2) or the value for hub = 1 and season = 2 is valid, but (2, 1) remains an invalid argument as there is no season = 0. A new set composed of hub and season, with shift\_sets = ["season"] and sets = model.hub, model.season, is created to lag on one index value while leaving others unchanged.

Default is to create set product of all input sets and lag/lead w/ resulting elements.

#### **Parameters**

- sname (str) Desired name for new set
- **shift\_size** (int) Size of shift in set
- **shift\_type** (str in ["lag", "lead"]) Type of shift (e.g. t-1 or t+1)
- \*sets (*Unnamed arguments*) A series of unnamed arguments assumed to contain pyo.Set in order of temporal hierarchy
- return\_set(bool | None, optional) Return the set rather than assign within function, by default False
- **shift\_sets** (*List | None, optional*) List of pyo.Set (by name) in \*sets to shift, by default None

#### Returns

A pyomo Set

### Return type

pyo.Set

### Raises

- ValueError Shift sets don't align with \*sets names
- **ValueError** Type argument is neither lead nor lag

```
declare_var(vname: str, v\_set: Set, return\_var: bool | None = False, within: Literal['NonNegativeReals', 'Binary', 'Reals', 'NonNegativeIntegers'] | None = 'NonNegativeReals', bound: tuple | None = (0, 1000000000), switch: bool | None = True) <math>\rightarrow Var
```

Declares a pyo Variable component named 'vname' with index set 'v\_set'.

Creates variable indexed by previously defined pyo Set instance 'v\_set' and assigns to self; function will return the component if return\_var is set to True. Other keywords passed to pyo. Var are within and bound.

### **Parameters**

- **vname** (str) Desired name of new pyo Variable
- **v\_set** (*pyo.Set*) Index set for new pyo Variable
- return\_var (bool | None, optional) Return component rather than assign internally, by default False
- within (str in ["NonNegativeReals", "Binary", "Reals", "NonNegativeIntegers"] | None, optional) pyo.Var keyword argument, by default "NonNegativeReals"
- bound (tuple | None, optional) pyo.Var keyword argument, by default (0, 1000000000)

# Returns

A pyomo Variable instance

### Return type

pyo.Var

 $get\_duals(component\_name: str) \rightarrow defaultdict$ 

Extract duals from a solved model instance

#### **Parameters**

**component\_name** (str) – Name of constraint

#### Returns

Dual values w/ index values

### **Return type**

defaultdict

 $populate\_sets\_rule(sname, set\_base\_name=None, set\_base2=None) \rightarrow Set$ 

Generic function to create a new re-indexed set for a pyomo ConcreteModel instance which should speed up build time. Must pass non-empty (either) set\_base\_name or set\_base2

#### **Parameters**

- m1 (pyo.ConcreteModel) pyomo model instance
- **sname** (*str*) name of input pyomo set to base reindexing
- **set\_base\_name** (*str*, *optional*) the name of the set to be the base of the reindexing, if left blank, uses set\_base2, by default "
- **set\_base2** (*list*, *optional*) the list of names of set columns to be the base of the reindexing, if left blank, should use set\_base\_name, by default [] these will form the index set of the indexed set structure

# Returns

reindexed set to be added to model

# Return type

pyomo set

```
reorganize_index_set(sname: str, new_sname: str, return_set: bool | None = False, create_indexed_set:
bool | None = False, reorg_set_cols: List[str] | None = None, reorg_set_sname: str
| None = None)
```

Creates new pyomo sets based on an input set and a desired set of indices for an output set. User should provide either names of columns desired for reorganized output set OR the name of a set that mirrors the desired indexing.

For instance, an input set indexed by (yr, region, month, day) can be reorganized into an output set:

```
(yr, region):[(month,day), (month,day), (month,day)]
```

```
when ["yr", "region"] is provided for reorg set cols.
```

If only the set keys are desired, without creating an indexed set object as illustrated above, the user can set 'create\_indexed\_set' to false. If true, the output is a pyo.IndexedSet, with each element of the IndexedSet containing the values of other indices

## **Parameters**

- **sname** (str) Name of input set
- **new\_sname** (*str*) Name of output set or IndexedSet
- **create\_indexed\_set** (*bool | None*, *optional*) Indicator for whether output set should include values as well as new index (IndexedSets), by default False

- return\_set (bool | None, optional) Indicator for whether to return the constructed set
- reorg\_set\_cols (List[str] | None, optional) List of columns to index output set contained in 'sname', by default None
- reorg\_set\_sname(str | None, optional)—Name of set to use for identifying output set indices, by default None

#### Returns

Pyomo Set or IndexedSet object reorganized based on input set

# Return type

Pyo.Set

### **Raises**

- **ValueError** Populate function is either-or for reorg\_set\_cols and reorg\_set\_sname, received both
- ValueError Populate function is either-or for reorg\_set\_cols and reorg\_set\_sname, received neither
- **ValueError** Elements missing from input set desired in new set

**unpack\_set\_arguments**( $sname: str, sets: Tuple[Set], return_set_product: bool | None = True) <math>\rightarrow$  Set Handles unnamed pyo.Set arguments for multiple declaration functions.

For an arbitrarily large number of set inputs, this function unpacks the names for each set stored in the column dictionary, creates a new list of the index labels and ordering, and then provides the pyo. Set product result as an output.

# **Parameters**

- **sname** (*str*) Name of new set
- sets (tuple of pyo.Set) Tuple of pyo.Set arguments to be used to generate new set
- return\_set\_product (bool) If True, return the unpacked set product

# Returns

**new\_set** – Set product result from input sets, by order of sets arguments

# Return type

pyo.Set

# src.common.utilities

A gathering of utility functions for dealing with model interconnectivity

#### **Functions**

getLogger([name])	Return a logger with the specified name, creating it if
	necessary.
<pre>get_args()</pre>	Parses args
<pre>make_dir(dir_name)</pre>	generates an output directory to write model results, output directory is the date/time at the time this function
	executes.

Table 45 – continued from previous page

scale_load(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year) and LoadScalar.csv (a multiplier for all model years).
scale_load_with_enduses(data_root)	Reads in BaseLoad.csv (load for all regions/hours for first year), EnduseBaseShares.csv (the shares of demand for each enduse in the base year) and EnduseScalar.csv (a multiplier for all model years by enduse category).
setup_logger(settings)	initiates logging, sets up logger in the output directory specified

### **Classes**

Path(*args, **kwargs)	PurePath subclass that can make system calls.

# src.common.utilities.get\_args()

Parses args

#### Returns

**args** – Contains arguments pass to main.py executable

# Return type

Namespace

# src.common.utilities.make\_dir(dir\_name)

generates an output directory to write model results, output directory is the date/time at the time this function executes. It includes subdirs for vars, params, constraints.

#### Returns

the name of the output directory

### Return type

string

# src.common.utilities.scale\_load(data\_root)

Reads in BaseLoad.csv (load for all regions/hours for first year) and LoadScalar.csv (a multiplier for all model years). Merges the data and multiplies the load by the scalar to generate new load estimates for all model years.

### Returns

dataframe that contains load for all regions/years/hours

# Return type

pandas.core.frame.DataFrame

# src.common.utilities.scale\_load\_with\_enduses(data\_root)

Reads in BaseLoad.csv (load for all regions/hours for first year), EnduseBaseShares.csv (the shares of demand for each enduse in the base year) and EnduseScalar.csv (a multiplier for all model years by enduse category). Merges the data and multiplies the load by the adjusted enduse scalar and then sums up to new load estimates for all model years.

### Returns

dataframe that contains load for all regions/years/hours

# Return type

pandas.core.frame.DataFrame

src.common.utilities.setup\_logger(settings)

initiates logging, sets up logger in the output directory specified

# **Parameters**

output\_dir (path) - output directory path

# src.integrator

# **Modules**

gaussseidel	Iteratively solve 2 models with GS methodology
progress_plot	A plotter that can be used for combined solves
runner	A gathering of functions for running models solo
unified	Unifying the solve of both H2 and Elec and Res
utilities	A gathering of utility functions for dealing with model
	interconnectivity

# src.integrator.gaussseidel

Iteratively solve 2 models with GS methodology see README for process explanation

# **Functions**

<pre>convert_elec_price_to_lut(prices)</pre>	convert electricity prices to dictionary, look up table
<pre>convert_h2_price_records(records)</pre>	simple coversion from list of records to a dictionary LUT
	repeat entries should not occur and will generate an error
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if
	necessary.
<pre>get_elec_price(instance[, block])</pre>	pulls hourly electricity prices from completed Power-
	Model and de-weights them.
<pre>init_old_cap(instance)</pre>	initialize capacity for 0th iteration
<pre>namedtuple(typename, field_names, *[,])</pre>	Returns a new subclass of tuple with named fields.
<pre>plot_it(OUTPUT_ROOT[, h2_price_records,])</pre>	cheap plotter of iterative progress
poll_h2_demand(model)	Get the hydrogen demand by rep_year and region
<pre>poll_h2_prices_from_elec(model, tech, regions)</pre>	poll the step-1 H2 price currently in the model for re-
	gion/year, averaged over any steps
<pre>poll_hydrogen_price(model[, block])</pre>	Retrieve the price of H2 from the H2 model
<pre>regional_annual_prices(m[, block])</pre>	pulls all regional annual weighted electricity prices
<pre>run_elec_model(settings[, solve])</pre>	build electricity model (and solve if solve=True) after
	passing in settings
run_gs(settings)	Start the iterative GS process
select_solver(instance)	Select solver based on learning method
set_new_cap(instance)	calculate new capacity after solve iteration
simple_solve(m)	a simple solve routine
<pre>simple_solve_no_opt(m, opt)</pre>	Solve concrete model using solver factory object
update_cost(instance)	update capital cost based on new capacity learning
<pre>update_h2_prices(model, h2_prices)</pre>	Update the H2 prices held in the model
·	

# **Classes**

EI(region, year, hour)	(region, year, hour)
Path(*args, **kwargs)	PurePath subclass that can make system calls.
residentialModule([settings, loadFile,])	This contains the Residential model and its associated
	functions.

src.integrator.gaussseidel.run\_gs(settings)

Start the iterative GS process

### **Parameters**

**settings** (*obj*) – Config\_settings object that holds module choices and settings

# src.integrator.progress plot

A plotter that can be used for combined solves

# **Functions**

<pre>plot_it(OUTPUT_ROOT[, h2_price_records,])</pre>	cheap plotter of iterative progress
<pre>plot_price_distro(OUTPUT_ROOT,</pre>	cheap/quick analysis and plot of the price records
price_records)	

# **Classes**

Path(*args, **kwargs)	PurePath subclass that can make system calls.
<pre>datetime(year, month, day[, hour[, minute[,)</pre>	The year, month and day arguments are required.

```
src.integrator.progress\_plot. \textbf{plot\_it}(OUTPUT\_ROOT, h2\_price\_records=[], elec\_price\_records=[], h2\_obj\_records=[], elec\_obj\_records=[], h2\_demand\_records=[], elec\_demand\_records=[], load\_records=[], elec\_price\_to\_res\_records=[])
```

cheap plotter of iterative progress

# src.integrator.runner

A gathering of functions for running models solo

# **Functions**

getLogger([name])	Return a logger with the specified name, creating it if
	necessary.
	continues on next page

Table 52 – continued from previous page

<pre>plot_price_distro(OUTPUT_ROOT, price_records)</pre>	cheap/quick analysis and plot of the price records
<pre>run_elec_model(settings[, solve])</pre>	build electricity model (and solve if solve=True) after passing in settings
<pre>run_elec_solo([settings])</pre>	Runs electricity model by itself as defined in settings
run_h2_solo([settings])	Runs hydrogen model by itself as defined in settings
<pre>run_hydrogen_model(settings)</pre>	run hydrogen model in standalone
<pre>run_residential(settings)</pre>	This runs the residential model in stand-alone mode.
<pre>run_residential_solo([settings])</pre>	Runs residential model by itself as defined in settings
run_standalone(settings)	Runs standalone methods based on settings selections; running 1 or more modules
<pre>value(obj[, exception])</pre>	A utility function that returns the value of a Pyomo object or expression.

### **Classes**

Config_settings(config_path[, args, test])	Generates the model settings that are used to solve.
Path(*args, **kwargs)	PurePath subclass that can make system calls.

# src.integrator.runner.run\_elec\_solo(settings: Config\_settings | None = None)

Runs electricity model by itself as defined in settings

#### **Parameters**

**settings** (Config\_settings) – Contains configuration settings for which regions, years, and switches to run

# src.integrator.runner.run\_h2\_solo(settings: Config\_settings | None = None)

Runs hydrogen model by itself as defined in settings

### **Parameters**

**settings** (Config\_settings) — Contains configuration settings for which regions and years to run

# src.integrator.runner.run\_residential\_solo(settings: Config\_settings | None = None)

Runs residential model by itself as defined in settings

#### **Parameters**

 $\textbf{settings} \; (\texttt{Config\_settings}) - \texttt{Contains} \; \texttt{configuration} \; \texttt{settings} \; \texttt{for} \; \texttt{which} \; \texttt{regions} \; \texttt{and} \; \texttt{years} \; \texttt{to} \; \texttt{run}$ 

# src.integrator.runner.run\_standalone(settings: Config\_settings)

Runs standalone methods based on settings selections; running 1 or more modules

#### **Parameters**

**settings** (Config\_settings) – Instance of config\_settings containing run options, mode and settings

# src.integrator.unified

Unifying the solve of both H2 and Elec and Res

#### Dev Notes:

1. The "annual demand" constraint that is present and INACTIVE is omitted here for clarity. It may likely be needed in some form - at a later time. Recall, the key linkages to share the electrical demand primary variable are:

- an annual level demand constraint
- an accurate price-pulling function that can consider weighted duals from both constraints
- 2. This model has a 2-solve update cycle as commented on near the termination check
- elec\_prices gleaned from cycle[n] results -> solve cycle[n+1]
- new\_load gleaned from cycle[n+1] results -> solve cycle[n+2]
- elec\_pices gleaned from cycle[n+2]

# **Functions**

<pre>convert_elec_price_to_lut(prices)</pre>	convert electricity prices to dictionary, look up table
<pre>convert_h2_price_records(records)</pre>	simple coversion from list of records to a dictionary LUT repeat entries should not occur and will generate an error
getLogger([name])	Return a logger with the specified name, creating it if necessary.
<pre>get_elec_price(instance[, block])</pre>	pulls hourly electricity prices from completed Power-Model and de-weights them.
<pre>init_old_cap(instance)</pre>	initialize capacity for 0th iteration
poll_h2_demand(model)	Get the hydrogen demand by rep_year and region
<pre>poll_hydrogen_price(model[, block])</pre>	Retrieve the price of H2 from the H2 model
<pre>regional_annual_prices(m[, block])</pre>	pulls all regional annual weighted electricity prices
<pre>run_elec_model(settings[, solve])</pre>	build electricity model (and solve if solve=True) after passing in settings
run_unified(settings)	Runs unified solve method based on
select_solver(instance)	Select solver based on learning method
set_new_cap(instance)	calculate new capacity after solve iteration
simple_solve(m)	a simple solve routine
<pre>simple_solve_no_opt(m, opt)</pre>	Solve concrete model using solver factory object
<pre>update_cost(instance)</pre>	update capital cost based on new capacity learning
update_h2_prices(model, h2_prices)	Update the H2 prices held in the model

# **Classes**

<pre>Config_settings(config_path[, args, test])</pre>	Generates the model settings that are used to solve.
EI(region, year, hour)	(region, year, hour)
HI(region, year)	(region, year)
defaultdict	defaultdict(default_factory=None, /, [])> dict with default factory
deque	deque([iterable[, maxlen]])> deque object
residentialModule([settings, loadFile,])	This contains the Residential model and its associated functions.

src.integrator.unified.run\_unified(settings: Config\_settings)

Runs unified solve method based on

# **Parameters**

 $\textbf{settings} \ (\texttt{Config\_settings}) - Instance \ of \ config\_settings \ containing \ run \ options, \ mode \ and \ settings$ 

# src.integrator.utilities

A gathering of utility functions for dealing with model interconnectivity

Dev Note: At some review point, some decisions may move these back & forth with parent models after it is decided if it is a utility job to do .... or a class method.

Additionally, there is probably some renaming due here for consistency

# **Module Attributes**

EI(region, year, hour)	(region, year, hour)
HI(region, year)	(region, year)

# **Functions**

<pre>convert_elec_price_to_lut(prices) convert_h2_price_records(records)</pre>	convert electricity prices to dictionary, look up table simple coversion from list of records to a dictionary LUT
	repeat entries should not occur and will generate an error
<pre>create_temporal_mapping(sw_temporal)</pre>	Combines the input mapping files within the electricity model to create a master temporal mapping dataframe.
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if necessary.
<pre>get_annual_wt_avg(elec_price)</pre>	takes annual weighted average of hourly electricity prices
<pre>get_elec_price(instance[, block])</pre>	pulls hourly electricity prices from completed Power-Model and de-weights them.
<pre>namedtuple(typename, field_names, *[,])</pre>	Returns a new subclass of tuple with named fields.
<pre>poll_h2_demand(model)</pre>	Get the hydrogen demand by rep_year and region
<pre>poll_h2_prices_from_elec(model, tech, regions)</pre>	poll the step-1 H2 price currently in the model for region/year, averaged over any steps
<pre>poll_hydrogen_price(model[, block])</pre>	Retrieve the price of H2 from the H2 model
<pre>poll_year_avg_elec_price(price_list)</pre>	retrieve a REPRESENTATIVE price at the annual level from a listing of prices
<pre>regional_annual_prices(m[, block])</pre>	pulls all regional annual weighted electricity prices
select_solver(instance)	Select solver based on learning method
simple_solve(m)	a simple solve routine
<pre>simple_solve_no_opt(m, opt)</pre>	Solve concrete model using solver factory object
<pre>update_elec_demand(self, elec_demand)</pre>	Update the external electical demand parameter with demands from the H2 model
<pre>update_h2_prices(model, h2_prices)</pre>	Update the H2 prices held in the model
<pre>value(obj[, exception])</pre>	A utility function that returns the value of a Pyomo object or expression.
	*

# Classes

ConcreteModel(*args, **kwds)	A concrete optimization model that does not defer construction of components.
EI(region, year, hour)	(region, year, hour)
HI(region, year)	(region, year)
Path(*args, **kwargs)	PurePath subclass that can make system calls.

defaultdict(default\_factory=None, /, [...]) --> dict with

Table 58 – continued from previous page

default factory

defaultdict

```
class src.integrator.utilities.EI(region, year, hour)
     (region, year, hour)
     _asdict()
           Return a new dict which maps field names to their values.
     _field_defaults = {}
     _fields = ('region', 'year', 'hour')
     classmethod _make(iterable)
           Make a new EI object from a sequence or iterable
     _replace(**kwds)
           Return a new EI object replacing specified fields with new values
     hour
           Alias for field number 2
     region
           Alias for field number 0
     vear
           Alias for field number 1
class src.integrator.utilities.HI(region, year)
     (region, year)
     _asdict()
           Return a new dict which maps field names to their values.
     _field_defaults = {}
     _fields = ('region', 'year')
     classmethod _make(iterable)
           Make a new HI object from a sequence or iterable
     _replace(**kwds)
           Return a new HI object replacing specified fields with new values
     region
           Alias for field number 0
     year
           Alias for field number 1
src.integrator.utilities.convert\_elec\_price\_to\_lut(prices: list[tuple[EI, float]]) \rightarrow dict[EI, float]
     convert electricity prices to dictionary, look up table
           Parameters
               prices (list[tuple[EI, float]]) - list of prices
           Returns
               dict of prices
```

### Return type

dict[EI, float]

 $src.integrator.utilities.convert_h2\_price\_records(records: list[tuple[HI, float]]) \rightarrow dict[HI, float]$  simple coversion from list of records to a dictionary LUT repeat entries should not occur and will generate an error

# src.integrator.utilities.create\_temporal\_mapping(sw\_temporal)

Combines the input mapping files within the electricity model to create a master temporal mapping dataframe. The df is used to build multiple temporal parameters used within the model. It creates a single dataframe that has 8760 rows for each hour in the year. Each hour in the year is assigned a season type, day type, and hour type used in the model. This defines the number of time periods the model will use based on cw\_s\_day and cw\_hr inputs.

#### Returns

a dataframe with 8760 rows that include each hour, hour type, day, day type, and season. It also includes the weights for each day type and hour type.

# Return type

dataframe

 $src.integrator.utilities.get\_annual\_wt\_avg(elec\_price: DataFrame) \rightarrow dict[HI, float]$  takes annual weighted average of hourly electricity prices

# **Parameters**

elec\_price (pd.DataFrame) - hourly electricity prices

#### Returns

annual weighted average electricity prices

## Return type

dict[HI, float]

```
\label{eq:src.integrator.utilities.get_elec_price} src.integrator.utilities.get_elec_price(instance: PowerModel | \textit{ConcreteModel}, block=None) \rightarrow DataFrame
```

pulls hourly electricity prices from completed PowerModel and de-weights them.

Prices from the duals are weighted by the day and year weights applied in the OBJ function This function retrieves the prices for all hours and removes the day and annual weights to return raw prices (and the day weights to use as needed)

# **Parameters**

- instance (PowerModel) solved electricity model
- **block** (*ConcreteModel*) reference to the block if the electricity model is a block within a larger model

### **Returns**

df of raw prices and the day weights to re-apply (if needed) columns: [r, y, hour, day\_weight, raw\_price]

# **Return type**

pd.DataFrame

```
src.integrator.utilities.poll_h2_demand(model: PowerModel) \rightarrow dict[HI, float]
```

Get the hydrogen demand by rep\_year and region

Use the Generation variable for h2 techs

NOTE: Not sure about day weighting calculation here!!

```
Returns
               dictionary of prices by H2 Index: price
           Return type
               dict[HI, float]
src.integrator.utilities.poll_h2\_prices\_from\_elec(model: PowerModel, tech, regions: Iterable) <math>\rightarrow
                                                               dict[Any, float]
      poll the step-1 H2 price currently in the model for region/year, averaged over any steps
           Parameters
                 • model (PowerModel) – solved PowerModel
                  • tech(str) - h2 tech
                  • regions (Iterable)
           Returns
               a dictionary of (region, seasons, year): price
           Return type
               dict[Any, float]
src.integrator.utilities.poll\_hydrogen\_price(model: H2Model | ConcreteModel, block=None) \rightarrow
                                                         list[tuple[HI, float]]
      Retrieve the price of H2 from the H2 model
           Parameters
                  • model (H2Model) - the model to poll
                 • block (optional) – block model to poll
           Returns
               list of H2 Index, price tuples
           Return type
               list[tuple[HI, float]]
src.integrator.utilities.poll\_year\_avg\_elec\_price(price\_list: list[tuple[EI, float]]) \rightarrow dict[HI, float]
      retrieve a REPRESENTATIVE price at the annual level from a listing of prices
      This function computes the AVERAGE elec price for each region-year combo
           Parameters
               price_list (list[tuple[EI, float]]) - input price list
           Returns
               a dictionary of (region, year): price
           Return type
               dict[HI, float]
src.integrator.utilities.regional\_annual\_prices(\textit{m: }PowerModel \mid ConcreteModel, block=None) \rightarrow
                                                             dict[HI, float]
      pulls all regional annual weighted electricity prices
           Parameters
                  • m (Union['PowerModel', ConcreteModel]) - solved PowerModel
```

1.4. Model Structure 53

• block (optional) – solved block model if applicable, by default None

### Returns

dict with regional annual electricity prices

# Return type

dict[HI, float]

src.integrator.utilities.select\_solver(instance: ConcreteModel)

Select solver based on learning method

#### **Parameters**

instance (PowerModel) - electricity pyomo model

### Returns

The pyomo solver

# Return type

solver type (?)

src.integrator.utilities.simple\_solve(m: ConcreteModel)

a simple solve routine

Solve concrete model using solver factory object

### **Parameters**

- m (ConcreteModel) Pyomo model
- **opt** (*SolverFactory*) Solver object initiated prior to solve

 $\verb|src.integrator.utilities.update_elec_demand| (\textit{self}, \textit{elec\_demand}: \textit{dict[HI}, \textit{float]}) \rightarrow \verb|None| |$ 

Update the external electical demand parameter with demands from the H2 model

#### Parameters

elec\_demand (dict[HI, float]) - the new demands broken out by hyd index (region, year)

 $src.integrator.utilities.update_h2\_prices(model: PowerModel, h2\_prices: dict[HI, float]) \rightarrow None$  Update the H2 prices held in the model

# **Parameters**

h2\_prices (list[tuple[HI, float]]) - new prices

# src.sensitivity

### **Modules**

babymode1	Baby Model This file contains the TestBabyModel class, which is a subclass of ConcreteModel, along with scripts that help generate the model parameters and structure.
faster_sensitivity	faster_sensitivity
sensitivity_tools	Sensitivity Tools This file contains the AutoSympy, SensitivityMatrix, CoordMap, and DifferentialMapping classes.
speed_test	Speed Test This is a script with some functions to run speed and accuracy tests on test models constructed with babymodel.

# src.sensitivity.babymodel

Baby Model This file contains the TestBabyModel class, which is a subclass of ConcreteModel, along with scripts that help generate the model parameters and structure.

The model structure is randomly generated through the functions generate and connect\_subregions, with model parameters as input.

# **Functions**

E1(z)	Classical case of the generalized exponential integral.
Eijk(*args, **kwargs)	Represent the Levi-Civita symbol.
GramSchmidt(vlist[, orthonormal])	Apply the Gram-Schmidt process to a set of vectors.
LC(f, *gens, **args)	Return the leading coefficient of f.
LM(f, *gens, **args)	Return the leading monomial of f.
LT(f, *gens, **args)	Return the leading term of f.
N(x[, n])	Calls x.evalf(n, **options).
POSform(variables, minterms[, dontcares])	The POSform function uses simplified_pairs and a redundant-group eliminating algorithm to convert the list of all input combinations that generate '1' (the minterms) into the smallest product-of-sums form.
SOPform(variables, minterms[, dontcares])	The SOPform function uses simplified_pairs and a redundant group- eliminating algorithm to convert the list of all input combos that generate '1' (the minterms) into the smallest sum-of-products form.
Ynm_c(n, m, theta, phi)	Conjugate spherical harmonics defined as
abundance(n)	Returns the difference between the sum of the positive
anam+(f[ v full])	proper divisors of a number and the number.
<pre>apart(f[, x, full])</pre>	Compute partial fraction decomposition of a rational function.
<pre>apart_list(f[, x, dummies])</pre>	Compute partial fraction decomposition of a rational function and return the result in structured form.
<pre>apply_finite_diff(order, x_list, y_list[, x0])</pre>	Calculates the finite difference approximation of the derivative of requested order at x0 from points provided in x_list and y_list.
approximants(l[, X, simplify])	Return a generator for consecutive Pade approximants for a series.
are_similar(e1, e2)	Are two geometrical entities similar.
arity(cls)	Return the arity of the function if it is known, else None.
<pre>ask(proposition[, assumptions, context])</pre>	Function to evaluate the proposition with assumptions.
<pre>assemble_partfrac_list(partial_list)</pre>	Reassemble a full partial fraction decomposition from a structured result obtained by the function apart_list.
assuming(*assumptions)	Context manager for assumptions.
banded(*args, **kwargs)	Returns a SparseMatrix from the given dictionary describing the diagonals of the matrix.
besselsimp(expr)	Simplify bessel-type functions.
binomial_coefficients(n)	Return a dictionary containing pairs $(k1, k2) : C_k n$ where $C_k n$ are binomial coefficients and $n = k1 + k2$ .
binomial_coefficients_list(n)	Return a list of binomial coefficients as rows of the Pascal's triangle.
block_collapse(expr)	Evaluates a block matrix expression
blockcut(expr, rowsizes, colsizes)	Cut a matrix expression into Blocks

continues on next page

Table 60 – continued from previous page

	d from previous page
bool_map(bool1, bool2)	Return the simplified version of <i>bool1</i> , and the mapping of variables that makes the two expressions <i>bool1</i> and <i>bool2</i> represent the same logical behaviour for some correspondence between the variables of each.
<pre>bottom_up(rv, F[, atoms, nonbasic])</pre>	Apply F to all expressions in an expression tree from the bottom up.
<pre>bspline_basis(d, knots, n, x)</pre>	The \$n\$-th B-spline at \$x\$ of degree \$d\$ with knots.
<pre>bspline_basis_set(d, knots, x)</pre>	Return the len(knots)-d-1 B-splines at $x$ of degree $d$ with $knots$ .
<pre>cacheit(func)</pre>	
<pre>cancel(f, *gens[, _signsimp])</pre>	Cancel common factors in a rational function f.
capture(func)	Return the printed output of func().
casoratian(seqs, n[, zero])	Given linear difference operator L of order 'k' and homogeneous equation Ly = 0 we want to compute kernel of L, which is a set of 'k' sequences: $a(n)$ , $b(n)$ ,.
cbrt(arg[, evaluate])	Returns the principal cube root.
<pre>ccode(expr[, assign_to, standard])</pre>	Converts an expr to a string of c code
centroid(*args)	Find the centroid (center of mass) of the collection containing only Points, Segments or Polygons.
<pre>chebyshevt_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the first kind $T_n(x)$ .
<pre>chebyshevu_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the second kind $U_n(x)$ .
<pre>check_assumptions(expr[, against])</pre>	Checks whether assumptions of expr match the T/F assumptions given (or possessed by against).
<pre>checkodesol(ode, sol[, func, order,])</pre>	Substitutes sol into ode and checks that the result is 0.
<pre>checkpdesol(pde, sol[, func, solve_for_func])</pre>	Checks if the given solution satisfies the partial differential equation.
<pre>checksol(f, symbol[, sol])</pre>	Checks whether sol is a solution of equation $f == 0$ .
<pre>classify_ode(eq[, func, dict, ics, prep,])</pre>	Returns a tuple of possible dsolve() classifications for an ODE.
<pre>classify_pde(eq[, func, dict, prep])</pre>	Returns a tuple of possible pdsolve() classifications for a PDE.
<pre>closest_points(*args)</pre>	Return the subset of points from a set of points that were the closest to each other in the 2D plane.
cofactors(f, g, *gens, **args)	Compute GCD and cofactors of f and g.
<pre>collect(expr, syms[, func, evaluate, exact,])</pre>	Collect additive terms of an expression.
<pre>collect_const(expr, *vars[, Numbers])</pre>	A non-greedy collection of terms with similar number coefficients in an Add expr.
combsimp(expr)	Simplify combinatorial expressions.
comp(z1, z2[, tol])	Return a bool indicating whether the error between $z1$ and $z2$ is $le$ to1.
compose(f, g, *gens, **args)	Compute functional composition f(g).
composite(nth)	Return the nth composite number, with the composite numbers indexed as $composite(1) = 4$ , $composite(2) = 6$ , etc
<pre>compositepi(n)</pre>	Return the number of positive composite numbers less than or equal to n.
connect_regions(region_map, hubmap,)	given a mapping of regions to lists of hubs, and hubs to regions, creates a set of arcs between hubs such that:
construct_domain(obj, **args)	Construct a minimal domain for a list of expressions.

Table 60 – continued from previous page

Table 00 - Continue	ed from previous page
content(f, *gens, **args)	Compute GCD of coefficients of f.
continued_fraction(a)	Return the continued fraction representation of a Ratio-
	nal or quadratic irrational.
<pre>continued_fraction_convergents(cf)</pre>	Return an iterator over the convergents of a continued
<u> </u>	fraction (cf).
<pre>continued_fraction_iterator(x)</pre>	Return continued fraction expansion of x as iterator.
<pre>continued_fraction_periodic(p, q[, d, s])</pre>	Find the periodic continued fraction expansion of a
1 12 / 2/	quadratic irrational.
<pre>continued_fraction_reduce(cf)</pre>	Reduce a continued fraction to a rational or quadratic
` '	irrational.
<pre>convex_hull(*args[, polygon])</pre>	The convex hull surrounding the Points contained in the
	list of entities.
<pre>convolution(a, b[, cycle, dps, prime,])</pre>	Performs convolution by determining the type of desired
	convolution using hints.
<pre>cosine_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency cosine trans-
	form of f, defined as
count_ops(expr[, visual])	Return a representation (integer or expression) of the op-
- , 1	erations in expr.
<pre>count_roots(f[, inf, sup])</pre>	Return the number of roots of f in [inf, sup] interval.
covering_product(a, b)	Returns the covering product of given sequences.
cse(exprs[, symbols, optimizations,])	Perform common subexpression elimination on an ex-
	pression.
<pre>cxxcode(expr[, assign_to, standard])</pre>	C++ equivalent of ccode().
cycle_length(f, x0[, nmax, values])	For a given iterated sequence, return a generator that
, , , , , , , , , , , , , , , , , , , ,	gives the length of the iterated cycle (lambda) and the
	length of terms before the cycle begins (mu); if values
	is True then the terms of the sequence will be returned
	instead.
<pre>cyclotomic_poly(n[, x, polys])</pre>	Generates cyclotomic polynomial of order $n$ in $x$ .
decompogen(f, symbol)	Computes General functional decomposition of f.
	Given an expression f, returns a list $[f_1, f_2,,$
	$f_n$ , where:: $f = f_1 \circ f_2 \circ f_n = f_1(f_2( f_n))$ .
decompose(f, *gens, **args)	Compute functional decomposition of f.
<pre>default_sort_key(item[, order])</pre>	Return a key that can be used for sorting.
deg(r)	Return the degree value for the given radians ( $pi = 180$
	degrees).
degree(f[, gen])	Return the degree of f in the given variable.
degree_list(f, *gens, **args)	Return a list of degrees of f in all variables.
denom(expr)	
derive_by_array(expr, dx)	Derivative by arrays.
det(matexpr)	Matrix Determinant
<pre>det_quick(M[, method])</pre>	Return det(M) assuming that either there are lots of ze-
	ros or the size of the matrix is small.
diag(*values[, strict, unpack])	Returns a matrix with the provided values placed on the
	diagonal.
diagonalize_vector(vector)	
<pre>dict_merge(*dicts)</pre>	Merge dictionaries into a single dictionary.
<pre>diff(f, *symbols, **kwargs)</pre>	Differentiate f with respect to symbols.
<pre>difference_delta(expr[, n, step])</pre>	Difference Operator.
	continues on next page

Table 60 – continued from previous page

Table 60 - Continue	a nom previous page
<pre>differentiate_finite(expr, *symbols[,])</pre>	Differentiate expr and replace Derivatives with finite differences.
diophantine(eq[, param, syms, permute])	Simplify the solution procedure of diophantine equation eq by converting it into a product of terms which should equal zero.
<pre>discrete_log(n, a, b[, order, prime_order])</pre>	Compute the discrete logarithm of a to the base b modulo n.
discriminant(f, *gens, **args)	Compute discriminant of f.
div(f, g, *gens, **args)	Compute polynomial division of f and g.
<pre>divisor_count(n[, modulus, proper])</pre>	Return the number of divisors of n.
divisors(n[, generator, proper])	Return all divisors of n sorted from 1n by default.
<pre>dotprint(expr[, styles, atom, maxdepth,])</pre>	DOT description of a SymPy expression tree
<pre>dsolve(eq[, func, hint, simplify, ics, xi,])</pre>	Solves any (supported) kind of ordinary differential equation and system of ordinary differential equations.
egyptian_fraction(r[, algorithm])	Return the list of denominators of an Egyptian fraction expansion $[1]$ of the said rational $r$ .
<pre>epath(path[, expr, func, args, kwargs])</pre>	Manipulate parts of an expression selected by a path.
<pre>euler_equations(L[, funcs, vars])</pre>	Find the Euler-Lagrange equations [1] for a given Lagrangian.
evaluate(x)	Control automatic evaluation
<pre>expand(e[, deep, modulus, power_base,])</pre>	Expand an expression using methods given as hints.
<pre>expand_complex(expr[, deep])</pre>	Wrapper around expand that only uses the complex hint.
<pre>expand_func(expr[, deep])</pre>	Wrapper around expand that only uses the func hint.
<pre>expand_log(expr[, deep, force, factor])</pre>	Wrapper around expand that only uses the log hint.
<pre>expand_mul(expr[, deep])</pre>	Wrapper around expand that only uses the mul hint.
<pre>expand_multinomial(expr[, deep])</pre>	Wrapper around expand that only uses the multinomial hint.
<pre>expand_power_base(expr[, deep, force])</pre>	Wrapper around expand that only uses the power_base hint.
<pre>expand_power_exp(expr[, deep])</pre>	Wrapper around expand that only uses the power_exp hint.
<pre>expand_trig(expr[, deep])</pre>	Wrapper around expand that only uses the trig hint.
<pre>exptrigsimp(expr)</pre>	Simplifies exponential / trigonometric / hyperbolic functions.
exquo(f, g, *gens, **args)	Compute polynomial exact quotient of f and g.
eye(*args, **kwargs)	Create square identity matrix n x n
<pre>factor(f, *gens[, deep])</pre>	Compute the factorization of expression, f, into irreducibles.
<pre>factor_list(f, *gens, **args)</pre>	Compute a list of irreducible factors of f.
factor_nc(expr)	Return the factored form of expr while handling non-commutative expressions.
<pre>factor_terms(expr[, radical, clear,])</pre>	Remove common factors from terms in all arguments without changing the underlying structure of the expr.
<pre>factorint(n[, limit, use_trial, use_rho,])</pre>	Given a positive integer n, factorint(n) returns a dict containing the prime factors of n as keys and their respective multiplicities as values.
<pre>factorrat(rat[, limit, use_trial, use_rho,])</pre>	Given a Rational r, factorrat(r) returns a dict containing the prime factors of r as keys and their respective multiplicities as values.
failing_assumptions(expr, **assumptions)	Return a dictionary containing assumptions with values not matching those of the passed assumptions.
	continues on next page

Table 60 – continued from previous page

Table 60 – continue	d from previous page
<pre>farthest_points(*args)</pre>	Return the subset of points from a set of points that were
	the furthest apart from each other in the 2D plane.
<pre>fcode(expr[, assign_to])</pre>	Converts an expr to a string of fortran code
fft(seq[, dps])	Performs the Discrete Fourier Transform (DFT) in the
	complex domain.
<pre>field(symbols, domain[, order])</pre>	Construct new rational function field returning (field, x1,
	, xn).
<pre>field_isomorphism(a, b, *[, fast])</pre>	Find an embedding of one number field into another.
filldedent(s[, w])	Strips leading and trailing empty lines from a copy of s,
	then dedents, fills and returns it.
<pre>finite_diff_weights(order, x_list[, x0])</pre>	Calculates the finite difference weights for an arbitrarily
	spaced one-dimensional grid (x_list) for derivatives at
	x0 of order 0, 1,, up to order using a recursive for-
flattom/itamable[layele ele]	mula.
<pre>flatten(iterable[, levels, cls]) fourier_series(f[, limits, finite])</pre>	Recursively denest iterable containers.  Computes the Fourier trigonometric series expansion.
fourier_transform(f, x, k, **hints)	Compute the unitary, ordinary-frequency Fourier trans-
Tourier_transform(r, x, k, ** inints)	form of f, defined as
fps(f[, x, x0, dir, hyper, order, rational,])	Generates Formal Power Series of f.
fraction(expr[, exact])	Returns a pair with expression's numerator and denomi-
Truction(expri, exact)	nator.
fu(rv[, measure])	Attempt to simplify expression by using transformation
. ( . [)	rules given in the algorithm by Fu et al.
fwht(seq)	Performs the Walsh Hadamard Transform (WHT), and
\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \	uses Hadamard ordering for the sequence.
<pre>galois_group(f, *gens[, by_name, max_tries,])</pre>	Compute the Galois group for polynomials $f$ up to de-
	gree 6.
gammasimp(expr)	Simplify expressions with gamma functions.
gcd(f[,g])	Compute GCD of f and g.
<pre>gcd_list(seq, *gens, **args)</pre>	Compute GCD of a list of polynomials.
<pre>gcd_terms(terms[, isprimitive, clear, fraction])</pre>	Compute the GCD of terms and put them together.
gcdex(f, g, *gens, **args)	Extended Euclidean algorithm of f and g.
<pre>generate([num_regions, hubs_per_region,])</pre>	generates a random network with all parameters required
	to initialize a ToyBabyModel
getLogger([name])	Return a logger with the specified name, creating it if
	necessary.
<pre>get_contraction_structure(expr)</pre>	Determine dummy indices of expr and describe its
<pre>get_indices(expr)</pre>	Structure  Determine the outer indices of expression expr
gff(f, *gens, **args)	Compute greatest factorial factorization of f.
gff_list(f, *gens, **args)	Compute a list of greatest factorial factors of f.
glsl_code(expr[, assign_to])	Converts an expr to a string of GLSL code
groebner(F, *gens, **args)	Computes the reduced Groebner basis for a set of poly-
ground (r, gono, urgo)	nomials.
<pre>ground_roots(f, *gens, **args)</pre>	Compute roots of f by factorization in the ground do-
<u>3</u>	main.
<pre>group(seq[, multiple])</pre>	Splits a sequence into a list of lists of equal, adjacent
1 1/	elements.
gruntz(e, z, z0[, dir])	Compute the limit of $e(z)$ at the point $z0$ using the Gruntz
	algorithm.
hadamard_product(*matrices)	Return the elementwise (aka Hadamard) product of ma-
	trices.
	continues on next nade

Table 60 – continued from previous page

Table 60 – continue	ed from previous page
half_gcdex(f, g, *gens, **args)	Half extended Euclidean algorithm of f and g.
<pre>hankel_transform(f, r, k, nu, **hints)</pre>	Compute the Hankel transform of $f$ , defined as
has_dups(seq)	Return True if there are any duplicate elements in seq.
has_variety(seq)	Return True if there are any different elements in seq.
<pre>hermite_poly(n[, x, polys])</pre>	Generates the Hermite polynomial $H_n(x)$ .
<pre>hermite_prob_poly(n[, x, polys])</pre>	Generates the probabilist's Hermite polynomial $He_n(x)$ .
<pre>hessian(f, varlist[, constraints])</pre>	Compute Hessian matrix for a function f wrt parame-
	ters in varlist which may be given as a sequence or a
	row/column vector.
<pre>homogeneous_order(eq, *symbols)</pre>	Returns the order $n$ if $g$ is homogeneous and None if it is
	not homogeneous.
horner(f, *gens, **args)	Rewrite a polynomial in Horner form.
<pre>hyperexpand(f[, allow_hyper, rewrite, place])</pre>	Expand hypergeometric functions.
hypersimilar(f, g, k)	Returns True if f and g are hyper-similar.
hypersimp(f, k)	Given combinatorial term f(k) simplify its consecutive
	term ratio i.e. $f(k+1)/f(k)$ .
idiff(eq, y, x[, n])	Return dy/dx assuming that eq $== 0$ .
ifft(seq[, dps])	Performs the Discrete Fourier Transform (DFT) in the
	complex domain.
ifwht(seq)	Performs the Walsh Hadamard Transform (WHT), and
	uses Hadamard ordering for the sequence.
igcd(*args)	Computes nonnegative integer greatest common divisor.
ilcm(*args)	Computes integer least common multiple.
<pre>imageset(*args)</pre>	Return an image of the set under transformation f.
<pre>init_printing([pretty_print, order,])</pre>	Initializes pretty-printer depending on the environment.
<pre>init_session([ipython, pretty_print, order,])</pre>	Initialize an embedded IPython or Python session.
<pre>integer_log(y, x)</pre>	Returns (e, bool) where e is the largest nonnegative integer such that $ y  \ge  x^e $ and bool is True if $y = x^e$ .
<pre>integer_nthroot(y, n)</pre>	Return a tuple containing $x = floor(y^{**}(1/n))$ and a boolean indicating whether the result is exact (that is, whether $x^{**}n == y$ ).
<pre>integrate(f, var,)</pre>	•
<pre>interactive_traversal(expr)</pre>	Traverse a tree asking a user which branch to choose.
interpolate(data, x)	Construct an interpolating polynomial for the data points
•	evaluated at point x (which can be symbolic or numeric).
<pre>interpolating_poly(n, x[, X, Y])</pre>	Construct Lagrange interpolating polynomial for n data
	points.
$\verb interpolating_spline (d, x, X, Y) $	Return spline of degree $d$ , passing through the given $X$ and $Y$ values.
<pre>intersecting_product(a, b)</pre>	Returns the intersecting product of given sequences.
<pre>intersection(*entities[, pairwise])</pre>	The intersection of a collection of GeometryEntity in-
	stances.
<pre>intervals(F[, all, eps, inf, sup, strict,])</pre>	Compute isolating intervals for roots of f.
<pre>intt(seq, prime)</pre>	Performs the Number Theoretic Transform (NTT),
	which specializes the Discrete Fourier Transform ( <b>DFT</b> ) over quotient ring $Z/pZ$ for prime $p$ instead of complex
	numbers $C$ .
inv_quick(M)	Return the inverse of M, assuming that either there are
	lots of zeros or the size of the matrix is small.
	continues on next page

Table 60 – continued from previous page

Table 00 Continue	tu nom previous page
<pre>inverse_cosine_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse cosine transform of <i>F</i> , defined as
<pre>inverse_fourier_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse Fourier transform of <i>F</i> , defined as
<pre>inverse_hankel_transform(F, k, r, nu, **hints)</pre>	Compute the inverse Hankel transform of $F$ defined as
<pre>inverse_laplace_transform(F, s, t[, plane])</pre>	Compute the inverse Laplace transform of $F(s)$ , defined
	as
<pre>inverse_mellin_transform(F, s, x, strip, **hints)</pre>	Compute the inverse Mellin transform of $F(s)$ over the fundamental strip given by strip=(a, b).
<pre>inverse_mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with indices of sequence as bitmasks.
<pre>inverse_sine_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse sine transform of $F$ , defined as
<pre>invert(f, g, *gens, **args)</pre>	Invert f modulo g when possible.
is_abundant(n)	Returns True if n is an abundant number, else False.
is_amicable(m, n)	Returns True if the numbers $m$ and $n$ are "amicable", else False.
<pre>is_convex(f, *syms[, domain])</pre>	Determines the convexity of the function passed in the argument.
<pre>is_decreasing(expression[, interval, symbol])</pre>	Return whether the function is decreasing in the given interval.
is_deficient(n)	Returns True if n is a deficient number, else False.
<pre>is_increasing(expression[, interval, symbol])</pre>	Return whether the function is increasing in the given interval.
<pre>is_mersenne_prime(n)</pre>	Returns True if n is a Mersenne prime, else False.
<pre>is_monotonic(expression[, interval, symbol])</pre>	Return whether the function is monotonic in the given interval.
is_nthpow_residue(a, n, m)	Returns True if $x^*n == a \pmod{m}$ has solutions.
is_perfect(n)	Returns True if n is a perfect number, else False.
<pre>is_primitive_root(a, p)</pre>	Returns True if a is a primitive root of p.
<pre>is_quad_residue(a, p)</pre>	Returns True if a (mod p) is in the set of squares mod p, i.e a % p in set( $[i^{**}2 \% p \text{ for i in range}(p)]$ ).
<pre>is_strictly_decreasing(expression[,])</pre>	Return whether the function is strictly decreasing in the given interval.
<pre>is_strictly_increasing(expression[,])</pre>	Return whether the function is strictly increasing in the given interval.
<pre>is_zero_dimensional(F, *gens, **args)</pre>	Checks if the ideal generated by a Groebner basis is zero- dimensional.
<pre>isolate(alg[, eps, fast])</pre>	Find a rational isolating interval for a real algebraic number.
isprime(n)	Test if n is a prime number (True) or not (False).
<pre>itermonomials(variables, max_degrees[,])</pre>	max_degrees and min_degrees are either both integers or both lists.
<pre>jacobi_normalized(n, a, b, x)</pre>	Jacobi polynomial $P_n^{\left( left(alpha, betaright) \right)}(x)$ .
<pre>jacobi_poly(n, a, b[, x, polys])</pre>	Generates the Jacobi polynomial $P_n^{(a,b)}(x)$ .
<pre>jacobi_symbol(m, n)</pre>	Returns the Jacobi symbol $(m/n)$ .
<pre>jn_zeros(n, k[, method, dps])</pre>	Zeros of the spherical Bessel function of the first kind.
<pre>jordan_cell(eigenval, n)</pre>	Create a Jordan block:
<pre>jscode(expr[, assign_to])</pre>	Converts an expr to a string of javascript code
<pre>julia_code(expr[, assign_to])</pre>	Converts <i>expr</i> to a string of Julia code.
kronecker_product(*matrices)	The Kronecker product of two or more arguments.
kroneckersimp(expr)	Simplify expressions with KroneckerDelta.
	continues on next page

Table 60 – continued from previous page

Table 60 – continue	d from previous page
<pre>laguerre_poly(n[, x, alpha, polys])</pre>	Generates the Laguerre polynomial $L_n^{(alpha)}(x)$ .
lambdify(args, expr[, modules, printer,])	Convert a SymPy expression into a function that allows
	for fast numeric evaluation.
<pre>laplace_transform(f, t, s[, legacy_matrix])</pre>	Compute the Laplace Transform $F(s)$ of $f(t)$ ,
lcm(f[,g])	Compute LCM of f and g.
<pre>lcm_list(seq, *gens, **args)</pre>	Compute LCM of a list of polynomials.
<pre>legendre_poly(n[, x, polys])</pre>	Generates the Legendre polynomial $P_n(x)$ .
legendre_symbol(a, p)	Returns the Legendre symbol $(a/p)$ .
limit(e, z, z0[, dir])	Computes the limit of $e(z)$ at the point $z0$ .
<pre>limit_seq(expr[, n, trials])</pre>	Finds the limit of a sequence as index n tends to infinity.
<pre>line_integrate(field, Curve, variables)</pre>	Compute the line integral.
<pre>linear_eq_to_matrix(equations, *symbols)</pre>	Converts a given System of Equations into Matrix form.
linsolve(system, *symbols)	Solve system of \$N\$ linear equations with \$M\$ vari-
	ables; both underdetermined and overdetermined sys-
	tems are supported.
list2numpy(l[, dtype])	Converts Python list of SymPy expressions to a NumPy
	array.
<pre>logcombine(expr[, force])</pre>	Takes logarithms and combines them using the following
	rules:
<pre>maple_code(expr[, assign_to])</pre>	Converts expr to a string of Maple code.
<pre>mathematica_code(expr, **settings)</pre>	Converts an expr to a string of the Wolfram Mathematica
	code
<pre>matrix2numpy(m[, dtype])</pre>	Converts SymPy's matrix to a NumPy array.
${\tt matrix\_multiply\_elementwise}(A,B)$	Return the Hadamard product (elementwise product) of
	A and B
<pre>matrix_symbols(expr)</pre>	
<pre>maximum(f, symbol[, domain])</pre>	Returns the maximum value of a function in the given
	domain.
<pre>mellin_transform(f, x, s, **hints)</pre>	Compute the Mellin transform $F(s)$ of $f(x)$ ,
<pre>memoize_property(propfunc)</pre>	Property decorator that caches the value of potentially
	expensive <i>propfunc</i> after the first evaluation.
mersenne_prime_exponent(nth)	Returns the exponent i for the nth Mersenne prime
	(which has the form $2^i - 1$ ).
<pre>minimal_polynomial(ex[, x, compose, polys,])</pre>	Computes the minimal polynomial of an algebraic ele-
	ment.
minimum(f, symbol[, domain])	Returns the minimum value of a function in the given
	domain.
minpoly(ex[, x, compose, polys, domain])	This is a synonym for minimal_polynomial().
<pre>mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with
	indices of sequence as bitmasks.
<pre>mod_inverse(a, m)</pre>	Return the number $c$ such that, $a$ times $c = 1$
	pmod{m}\$ where \$c\$ has the same sign as \$m\$.
monic(f, *gens, **args)	Divide all coefficients of f by LC(f).
<pre>multiline_latex(lhs, rhs[, terms_per_line,])</pre>	This function generates a LaTeX equation with a mul-
	tiline right-hand side in an align*, eqnarray or
	IEEEeqnarray environment.
<pre>multinomial_coefficients(m, n)</pre>	Return a dictionary containing pairs {(k1,k2,,km)
	: C_kn} where C_kn are multinomial coefficients such
	that $n=k1+k2++km$ .
<pre>multiplicity(p, n)</pre>	Find the greatest integer m such that $p^{**}m$ divides n.
n_order(a, n)	Returns the order of a modulo n.
	continues on next page

Table 60 – continued from previous page

	ed from previous page
nextprime(n[, ith])	Return the ith prime greater than n.
nfloat(expr[, n, exponent, dkeys])	Make all Rationals in expr Floats except those in exponents (unless the exponents flag is set to True) and those in undefined functions.
nonlinsolve(system, *symbols)	Solve system of \$N\$ nonlinear equations with \$M\$ variables, which means both under and overdetermined systems are supported.
<pre>not_empty_in(finset_intersection, *syms)</pre>	Finds the domain of the functions in finset_intersection in which the finite_set is not-empty.
<pre>npartitions(n[, verbose])</pre>	Calculate the partition function $P(n)$ , i.e. the number of ways that n can be written as a sum of positive integers.
<pre>nroots(f[, n, maxsteps, cleanup])</pre>	Compute numerical approximations of roots of f.
<pre>nsimplify(expr[, constants, tolerance,])</pre>	Find a simple representation for a number or, if there are free symbols or if rational=True, then replace Floats with their Rational equivalents.
nsolve(*args[, dict])	Solve a nonlinear equation system numerically: nsolve(f, [args,] x0, modules=['mpmath'], **kwargs).
nth_power_roots_poly(f, n, *gens, **args)	Construct a polynomial with n-th powers of roots of f.
nthroot_mod(a, n, p[, all_roots])	Find the solutions to $x^* = a \mod p$ .
ntt(seq, prime)	Performs the Number Theoretic Transform (NTT), which specializes the Discrete Fourier Transform (DFT) over quotient ring $\mathbb{Z}/p\mathbb{Z}$ for prime $p$ instead of complex numbers $C$ .
<pre>numbered_symbols([prefix, cls, start, exclude])</pre>	Generate an infinite stream of Symbols consisting of a prefix and increasing subscripts provided that they do not occur in exclude.
numer(expr)	
octave_code(expr[, assign_to])	Converts <i>expr</i> to a string of Octave (or Matlab) code.
ode_order(expr, func)	Returns the order of a given differential equation with respect to func.
ones(*args, **kwargs)	Returns a matrix of ones with rows rows and cols columns; if cols is omitted a square matrix will be returned.
ordered(seq[, keys, default, warn])	Return an iterator of the seq where keys are used to break ties in a conservative fashion: if, after applying a key, there are no ties then no other keys will be computed.
<pre>pager_print(expr, **settings)</pre>	Prints expr using the pager, in pretty form.
parallel_poly_from_expr(exprs, *gens, **args)	Construct polynomials from expressions.
<pre>parse_expr(s[, local_dict, transformations,])</pre>	Converts the string s to a SymPy expression, in local_dict.
<pre>pde_separate(eq, fun, sep[, strategy])</pre>	Separate variables in partial differential equation either by additive or multiplicative separation approach.
pde_separate_add(eq, fun, sep)	Helper function for searching additive separable solutions.
<pre>pde_separate_mul(eq, fun, sep)</pre>	Helper function for searching multiplicative separable solutions.
pdiv(f, g, *gens, **args)	Compute polynomial pseudo-division of f and g.
<pre>pdsolve(eq[, func, hint, dict, solvefun])</pre>	Solves any (supported) kind of partial differential equation.
	continues on next name

Table 60 – continued from previous page

Table 60 – continued from previous page	
per(matexpr)	Matrix Permanent
<pre>perfect_power(n[, candidates, big, factor])</pre>	Return (b, e) such that $n == b^*e$ if n is a unique perfect power with $e > 1$ , else False (e.g. 1 is not a perfect power).
<pre>periodicity(f, symbol[, check])</pre>	Tests the given function for periodicity in the given symbol.
<pre>permutedims(expr[, perm, index_order_old,])</pre>	Permutes the indices of an array.
pexquo(f, g, *gens, **args)	Compute polynomial exact pseudo-quotient of f and g.
<pre>piecewise_exclusive(expr, *[, skip_nan, deep])</pre>	Rewrite Piecewise with mutually exclusive conditions.
<pre>piecewise_fold(expr[, evaluate])</pre>	Takes an expression containing a piecewise function and returns the expression in piecewise form.
plot(*args[, show])	Plots a function of a single variable as a curve.
plot_implicit(expr[, x_var, y_var,])	A plot function to plot implicit equations / inequalities.
plot_parametric(*args[, show])	Plots a 2D parametric curve.
polarify(eq[, subs, lift])	Turn all numbers in eq into their polar equivalents (under
potatity(eqt, subs, int)	the standard choice of argument).
pollard_pm1(n[, B, a, retries, seed])	Use Pollard's p-1 method to try to extract a nontrivial
F	factor of n.
pollard_rho(n[, s, a, retries, seed,])	Use Pollard's rho method to try to extract a nontrivial factor of n.
poly(expr, *gens, **args)	Efficiently transform an expression into a polynomial.
<pre>poly_from_expr(expr, *gens, **args)</pre>	Construct a polynomial from an expression.
posify(eq)	Return eq (with generic symbols made positive) and a dictionary containing the mapping between the old and new symbols.
postfixes(seq)	Generate all postfixes of a sequence.
<pre>postorder_traversal(node[, keys])</pre>	Do a postorder traversal of a tree.
<pre>powdenest(eq[, force, polar]) powsimp(expr[, deep, combine, force, measure])</pre>	Collect exponents on powers as assumptions allow.  Reduce expression by combining powers with similar
	bases and exponents.
<pre>pprint(expr, **kwargs)</pre>	Prints expr in pretty form.
<pre>pprint_try_use_unicode()</pre>	See if unicode output is available and leverage it if possible
<pre>pprint_use_unicode([flag])</pre>	Set whether pretty-printer should use unicode by default
pquo(f, g, *gens, **args)	Compute polynomial pseudo-quotient of f and g.
prefixes(seq)	Generate all prefixes of a sequence.
prem(f, g, *gens, **args)	Compute polynomial pseudo-remainder of f and g.
<pre>pretty_print(expr, **kwargs)</pre>	Prints expr in pretty form.
<pre>preview(expr[, output, viewer, euler,])</pre>	View expression or LaTeX markup in PNG, DVI, PostScript or PDF form.
<pre>prevprime(n)</pre>	Return the largest prime smaller than n.
prime(nth)	Return the nth prime, with the primes indexed as $prime(1) = 2$ , $prime(2) = 3$ , etc.
<pre>prime_decomp(p[, T, ZK, dK, radical])</pre>	Compute the decomposition of rational prime $p$ in a number field.
<pre>prime_valuation(I, P)</pre>	Compute the <i>P</i> -adic valuation for an integral ideal <i>I</i> .
<pre>primefactors(n[, limit, verbose])</pre>	Return a sorted list of n's prime factors, ignoring multi- plicity and any composite factor that remains if the limit was set too low for complete factorization.
<pre>primerange(a[, b])</pre>	Generate a list of all prime numbers in the range [2, a), or [a, b).
<pre>primitive(f, *gens, **args)</pre>	Compute content and the primitive form of £.
	continues on next nade

Table 60 – continued from previous page

	lueu IIoIII previous page
<pre>primitive_element(extension[, x, ex, polys])</pre>	Find a single generator for a number field given by several generators.
<pre>primitive_root(p)</pre>	Returns the smallest primitive root or None.
primorial(n[, nth])	Returns the product of the first n primes (default) or the primes less than or equal to n (when nth=False).
<pre>print_ccode(expr, **settings)</pre>	Prints C representation of the given expression.
print_fcode(expr, **settings)	Prints the Fortran representation of the given expression.
print_gls1(expr, **settings)	Prints the GLSL representation of the given expression.
print_gtk(x[, start_viewer])	Print to Gtkmathview, a gtk widget capable of rendering
	MathML.
<pre>print_jscode(expr, **settings)</pre>	Prints the Javascript representation of the given expression.
<pre>print_latex(expr, **settings)</pre>	Prints LaTeX representation of the given expression.
<pre>print_maple_code(expr, **settings)</pre>	Prints the Maple representation of the given expression.
<pre>print_mathml(expr[, printer])</pre>	Prints a pretty representation of the MathML code for expr.
<pre>print_python(expr, **settings)</pre>	Print output of python() function
print_rcode(expr, **settings)	Prints R representation of the given expression.
<pre>print_tree(node[, assumptions])</pre>	Prints a tree representation of "node".
prod(a[, start])	Return product of elements of a. Start with int 1 so if only
<pre>product(*args, **kwargs)</pre>	Compute the product.
<pre>proper_divisor_count(n[, modulus])</pre>	Return the number of proper divisors of n.
proper_divisors(n[, generator])	Return all divisors of n except n, sorted by default.
public(obj)	Append obj's name to globalall variable (call
- · · · ·	site).
pycode(expr, **settings)	Converts an expr to a string of Python code
<pre>python(expr, **settings)</pre>	Return Python interpretation of passed expression (can be passed to the exec() function without any modifica- tions)
<pre>quadratic_congruence(a, b, c, p)</pre>	Find the solutions to $a x^* + b x + c = 0 \mod p$ .
quadratic_residues(p)	Returns the list of quadratic residues.
quo(f, g, *gens, **args)	Compute polynomial quotient of f and g.
rad(d)	Return the radian value for the given degrees (pi = 180 degrees).
<pre>radsimp(expr[, symbolic, max_terms])</pre>	Rationalize the denominator by removing square roots.
randMatrix(r[, c, min, max, seed,])	Create random matrix with dimensions r x c.
<pre>random_poly(x, n, inf, sup[, domain, polys])</pre>	Generates a polynomial of degree n with coefficients in [inf, sup].
randprime(a, b)	Return a random prime number in the range [a, b).
rational_interpolate(data, degnum[, X])	Returns a rational interpolation, where the data points are element of any integral domain.
ratsimp(expr)	Put an expression over a common denominator, cancel
-	and reduce.
ratsimpmodprime(expr, G, *gens[, quick,])	Simplifies a rational expression expr modulo the prime ideal generated by G.
<pre>rcode(expr[, assign_to])</pre>	Converts an expr to a string of r code
<pre>rcollect(expr, *vars)</pre>	Recursively collect sums in an expression.
real_root(arg[, n, evaluate])	Return the real <i>n</i> 'th-root of <i>arg</i> if possible.
real_roots(f[, multiple])	Return a list of real roots with multiplicities of f.
reduce_abs_inequalities(exprs, gen)	Reduce a system of inequalities with nested absolute values.
	continues on next page

Table 60 – continued from previous page

Table 60 – continued from previous page	
reduce_abs_inequality(expr, rel, gen)	Reduce an inequality with nested absolute values.
<pre>reduce_inequalities(inequalities[, symbols])</pre>	Reduce a system of inequalities with rational coefficients.
reduced(f, G, *gens, **args)	Reduces a polynomial f modulo a set of polynomials G.
refine(expr[, assumptions])	Simplify an expression using assumptions.
refine_root(f, s, t[, eps, steps, fast,])	Refine an isolating interval of a root to the given preci-
· · · · · · · · · · · · · · · · · · ·	sion.
register_handler(key, handler)	Register a handler in the ask system.
rem(f, g, *gens, **args)	Compute polynomial remainder of f and g.
remove_handler(key, handler)	Removes a handler from the ask system.
reshape(seq, how)	Reshape the sequence according to the template in how.
residue(expr, x, x0)	Finds the residue of expr at the point $x=x0$ .
<pre>resultant(f, g, *gens[, includePRS])</pre>	Compute resultant of f and g.
ring(symbols, domain[, order])	Construct a polynomial ring returning (ring, x_1, , x_n).
<pre>root(arg, n[, k, evaluate])</pre>	Returns the <i>k</i> -th <i>n</i> -th root of arg.
<pre>rootof(f, x[, index, radicals, expand])</pre>	An indexed root of a univariate polynomial.
roots(f, *gens[, auto, cubics, trig,])	Computes symbolic roots of a univariate polynomial.
rot_axis1(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 1-axis.
rot_axis2(theta)	Returns a rotation matrix for a rotation of theta (in radi-
	ans) about the 2-axis.
rot_axis3(theta)	Returns a rotation matrix for a rotation of theta (in radi-
	ans) about the 3-axis.
<pre>rot_ccw_axis1(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 1-axis.
<pre>rot_ccw_axis2(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 2-axis.
<pre>rot_ccw_axis3(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 3-axis.
<pre>rot_givens(i, j, theta[, dim])</pre>	Returns a a Givens rotation matrix, a a rotation in the plane spanned by two coordinates axes.
rotations(s[, dir])	Return a generator giving the items in s as list where each
	subsequent list has the items rotated to the left (default)
	or right (dir=-1) relative to the previous list.
<pre>round_two(T[, radicals])</pre>	Zassenhaus's "Round 2" algorithm.
rsolve(f, y[, init])	Solve univariate recurrence with rational coefficients.
rsolve_hyper(coeffs, f, n, **hints)	Given linear recurrence operator $operatorname\{L\}$ of order $k$ with polynomial coefficients and inhomogeneous equation $operatorname\{L\}$ $y = f$ we seek for all hypergeometric solutions over field $K$ of characteristic zero.
rsolve_poly(coeffs, f, n[, shift])	Given linear recurrence operator <i>operatorname{L}</i> of or-
- <u>-</u>	der $k$ with polynomial coefficients and inhomogeneous
	equation operatorname $\{L\}$ $y = f$ , where $f$ is a polyno-
	mial, we seek for all polynomial solutions over field $K$
	of characteristic zero.
rsolve_ratio(coeffs, f, n, **hints)	Given linear recurrence operator operatorname{L} of or-
	der k with polynomial coefficients and inhomogeneous
	equation operatorname{L} $y = f$ , where $f$ is a polyno-
	mial, we seek for all rational solutions over field $K$ of
	characteristic zero.
<pre>rust_code(expr[, assign_to])</pre>	Converts an expr to a string of Rust code
	continues on next page

Table 60 – continued from previous page

Table 00 - Continue	
<pre>satisfiable(expr[, algorithm, all_models,])</pre>	Check satisfiability of a propositional sentence.
<pre>separatevars(expr[, symbols, dict, force])</pre>	Separates variables in an expression, if possible.
sequence(seq[, limits])	Returns appropriate sequence object.
<pre>series(expr[, x, x0, n, dir])</pre>	Series expansion of expr around point $x = x\theta$ .
seterr([divide])	Should SymPy raise an exception on 0/0 or return a nan?
sfield(exprs, *symbols, **options)	Construct a field deriving generators and domain from
\ 1 / J / 1 /	options and input expressions.
shape()	Return the shape of the <i>expr</i> as a tuple.
sift(seq, keyfunc[, binary])	Sift the sequence, seq according to keyfunc.
signsimp(expr[, evaluate])	Make all Add sub-expressions canonical wrt sign.
simplify(expr[, ratio, measure, rational,])	Simplifies the given expression.
simplify_logic(expr[, form, deep, force,])	This function simplifies a boolean function to its simpli-
	fied version in SOP or POS form.
<pre>sine_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency sine transform of $f$ , defined as
<pre>singularities(expression, symbol[, domain])</pre>	Find singularities of a given function.
singularityintegrate(f, x)	This function handles the indefinite integrations of Sin-
	gularity functions.
<pre>smtlib_code(expr[, auto_assert,])</pre>	Converts expr to a string of smtlib code.
solve(f, *symbols, **flags)	Algebraically solves equations and systems of equations.
<pre>solve_linear(lhs[, rhs, symbols, exclude])</pre>	Return a tuple derived from $f = 1hs - rhs$ that is
	one of the following: (0, 1), (0, 0), (symbol,
	solution), (n, d).
<pre>solve_linear_system(system, *symbols, **flags)</pre>	Solve system of \$N\$ linear equations with \$M\$ vari-
	ables, which means both under- and overdetermined sys-
	tems are supported.
<pre>solve_linear_system_LU(matrix, syms)</pre>	Solves the augmented matrix system using LUsolve and
SOLVE_LINEAL_SYSCEM_ES(MATIX, SYMS)	returns a dictionary in which solutions are keyed to the
	symbols of <i>syms</i> as ordered.
<pre>solve_poly_inequality(poly, rel)</pre>	Solve a polynomial inequality with rational coefficients.
solve_poly_system(seq, *gens[, strict])	Return a list of solutions for the system of polynomial
solve_poly_system(seq, gens[, strict])	equations or else None.
solve_rational_inequalities(eqs)	Solve a system of rational inequalities with rational co-
sorve_racronar_mequarrcres(eqs)	efficients.
colve triangulated(nolve *cone **cree)	Solve a polynomial system using Gianni-Kalkbrenner al-
<pre>solve_triangulated(polys, *gens, **args)</pre>	· · · · · · · · · · · · · · · · · · ·
calus undetermined assissing	gorithm.
<pre>solve_undetermined_coeffs(equ, coeffs,)</pre>	Solve a system of equations in \$k\$ parameters that is
	formed by matching coefficients in variables coeffs
	that are on factors dependent on the remaining variables
	(or those given explicitly by syms.
solve_univariate_inequality(expr, gen[,])	Solves a real univariate inequality.
solveset(f[, symbol, domain])	Solves a given inequality or equation with set as output
sqf(f, *gens, **args)	Compute square-free factorization of f.
sqf_list(f, *gens, **args)	Compute a list of square-free factors of f.
<pre>sqf_norm(f, *gens, **args)</pre>	Compute square-free norm of f.
<pre>sqf_part(f, *gens, **args)</pre>	Compute square-free part of f.
<pre>sqrt(arg[, evaluate])</pre>	Returns the principal square root.
<pre>sqrt_mod(a, p[, all_roots])</pre>	Find a root of $x^**2 = a \mod p$ .
<pre>sqrt_mod_iter(a, p[, domain])</pre>	Iterate over solutions to $x^**2 = a \mod p$ .
<pre>sqrtdenest(expr[, max_iter])</pre>	Denests sqrts in an expression that contain other square
	roots if possible, otherwise returns the expr unchanged.
	continues on next page

Table 60 – continued from previous page

Table 60 – continued from previous page	
sring(exprs, *symbols, **options)	Construct a ring deriving generators and domain from
	options and input expressions.
<pre>stationary_points(f, symbol[, domain])</pre>	Returns the stationary points of a function (where deriva-
	tive of the function is 0) in the given domain.
sturm(f, *gens, **args)	Compute Sturm sequence of f.
<pre>subresultants(f, g, *gens, **args)</pre>	Compute subresultant PRS of f and g.
<pre>subsets(seq[, k, repetition])</pre>	Generates all k-subsets (combinations) from an n-
	element set, seq.
<pre>substitution(system, symbols[, result,])</pre>	Solves the <i>system</i> using substitution method.
<pre>summation(f, *symbols, **kwargs)</pre>	Compute the summation of f with respect to symbols.
<pre>swinnerton_dyer_poly(n[, x, polys])</pre>	Generates n-th Swinnerton-Dyer polynomial in $x$ .
symarray(prefix, shape, **kwargs)	Create a numpy ndarray of symbols (as an object array).
symbols(names, *[, cls])	Transform strings into instances of Symbol class.
<pre>symmetric_poly(n, *gens[, polys])</pre>	Generates symmetric polynomial of order <i>n</i> .
<pre>symmetrize(F, *gens, **args)</pre>	Rewrite a polynomial in terms of elementary symmetric
	polynomials.
<pre>sympify(a[, locals, convert_xor, strict,])</pre>	Converts an arbitrary expression to a type that can be
	used inside SymPy.
take(iter, n)	Return n items from iter iterator.
tensorcontraction(array, *contraction_axes)	Contraction of an array-like object on the specified axes.
tensordiagonal(array, *diagonal_axes)	Diagonalization of an array-like object on the specified
1 (4)	axes.
tensorproduct(*args)	Tensor product among scalars or array-like objects.
terms_gcd(f, *gens, **args)	Remove GCD of terms from f.
textplot(expr, a, b[, W, H])	Print a crude ASCII art plot of the SymPy expression
	'expr' (which should contain a single symbol, e.g. x or
three ded/free	something else) over the interval [a, b].
threaded(func)	Apply func to subelements of an object, including Add.
+imad(funa[ satur limit])	
<pre>timed(func[, setup, limit]) to_cnf(expr[, simplify, force])</pre>	Adaptively measure execution time of a function.  Convert a propositional logical sentence expr to con-
to_cnr(expr[, sniipiny, force])	junctive normal form: ((A   ~B  ) & (B   C
	) &).
to_dnf(expr[, simplify, force])	Convert a propositional logical sentence expr to dis-
to_unit(expit, simpiny, force))	junctive normal form: ((A & ~B &)   (B & C
	&)  ).
to_nnf(expr[, simplify])	Converts expr to Negation Normal Form (NNF).
to_number_field(extension[, theta, gen, alias])	Express one algebraic number in the field generated by
co_namber_frequencient, theat, gen, anasj)	another.
together(expr[, deep, fraction])	Denest and combine rational expressions using symbolic
together (expression)	methods.
topological_sort(graph[, key])	Topological sort of graph's vertices.
total_degree(f, *gens)	Return the total_degree of f in the given variables.
trace(expr)	Trace of a Matrix.
trailing(n)	Count the number of trailing zero digits in the binary
(	representation of n, i.e. determine the largest power of 2
	that divides n.
trigsimp(expr[, inverse])	Returns a reduced expression by using known trig iden-
J P C P C	tities.
trunc(f, p, *gens, **args)	Reduce f modulo a constant p.
unbranched_argument(arg)	Returns periodic argument of arg with period as infinity.
unflatten(iter[, n])	Group iter into tuples of length n.
	continues on next page

Table 60 – continued from previous page

<pre>unpolarify(eq[, subs, exponents_only])</pre>	If $p$ denotes the projection from the Riemann surface of the logarithm to the complex line, return a simplified version $eq'$ of $eq$ such that $p(eq') = p(eq)$ .
<pre>use(expr, func[, level, args, kwargs])</pre>	Use func to transform expr at the given level.
var(names, **args)	Create symbols and inject them into the global namespace.
<pre>variations(seq, n[, repetition])</pre>	Returns an iterator over the n-sized variations of seq (size N).
vfield(symbols, domain[, order])	Construct new rational function field and inject generators into global namespace.
<pre>viete(f[, roots])</pre>	Generate Viete's formulas for f.
<pre>vring(symbols, domain[, order])</pre>	Construct a polynomial ring and inject $x_1, \ldots, x_n$ into the global namespace.
<pre>wronskian(functions, var[, method])</pre>	Compute Wronskian for [] of functions
xfield(symbols, domain[, order])	Construct new rational function field returning (field, $(x1,, xn)$ ).
<pre>xring(symbols, domain[, order])</pre>	Construct a polynomial ring returning (ring, $(x_1,, x_n)$ ).
xthreaded(func)	Apply func to subelements of an object, excluding Add.
zeros(*args, **kwargs)	Returns a matrix of zeros with rows rows and cols columns; if cols is omitted a square matrix will be returned.

## Classes

Abs(arg)	Return the absolute value of the argument.
AccumBounds	alias of AccumulationBounds
Add(*args[, evaluate, _sympify])	Expression representing addition operation for algebraic group.
Adjoint(*args, **kwargs)	The Hermitian adjoint of a matrix expression.
AlgebraicField(dom, *ext[, alias])	Algebraic number field QQ(a)
<pre>AlgebraicNumber(expr[, coeffs, alias])</pre>	Class for representing algebraic numbers in SymPy.
And(*args)	Logical AND function.
AppliedPredicate(predicate, *args)	The class of expressions resulting from applying Predicate to the arguments.
Array	alias of ImmutableDenseNDimArray
AssumptionsContext	Set containing default assumptions which are applied to the ask() function.
Atom(*args)	A parent class for atomic things.
AtomicExpr(*args)	A parent class for object which are both atoms and Exprs.
AutoSympy(model)	
Basic(*args)	Base class for all SymPy objects.
BlockDiagMatrix(*mats)	A sparse matrix with block matrices along its diagonals
BlockMatrix(*args, **kwargs)	A BlockMatrix is a Matrix comprised of other matrices.
CRootOf	alias of ComplexRootOf
Chi(z)	Cosh integral.
Ci(z)	Cosine integral.
Circle(*args, **kwargs)	A circle in space.

continues on next page

Table 61 – continued from previous page

Table 61 – continue	ed from previous page
Complement(a, b[, evaluate])	Represents the set difference or relative complement of a set with another set.
<pre>ComplexField([prec, dps, tol])</pre>	Complex numbers up to the given precision.
ComplexRegion(sets[, polar])	Represents the Set of all Complex Numbers.
ComplexRootOf(f, x[, index, radicals, expand])	Represents an indexed complex root of a polynomial.
<pre>ConditionSet(sym, condition[, base_set])</pre>	Set of elements which satisfies a given condition.
Contains(x, s)	Asserts that x is an element of the set S.
CoordMap(var_vector, eq_duals, ineq_duals,)	
CosineTransform(*args)	Class representing unevaluated cosine transforms.
Curve(function, limits)	A curve in space.
<pre>DeferredVector(name, **assumptions)</pre>	A vector whose components are deferred (e.g. for use with lambdify).
DenseNDimArray(*args, **kwargs)	
Derivative(expr, *variables, **kwargs)	Carries out differentiation of the given expression with respect to symbols.
Determinant(mat)	Matrix Determinant
DiagMatrix(vector)	Turn a vector into a diagonal matrix.
DiagonalMatrix(*args, **kwargs)	DiagonalMatrix(M) will create a matrix expression that behaves as though all off-diagonal elements, $M[i, j]$ where $i!=j$ , are zero.
DiagonalOf(*args, **kwargs)	DiagonalOf(M) will create a matrix expression that is equivalent to the diagonal of <i>M</i> , represented as a single column matrix.
Dict(*args)	Wrapper around the builtin dict object.
DifferentialMapping(US, coord2item,)	TT J
<pre>DiracDelta(arg[, k])</pre>	The DiracDelta function and its derivatives.
DisjointUnion(*sets)	Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.
Domain()	Superclass for all domains in the polys domains system.
DotProduct(arg1, arg2)	Dot product of vector matrices
Dummy([name, dummy_index])	Dummy symbols are each unique, even if they have the same name:
EPath(path)	Manipulate expressions using paths.
Ei(z)	The classical exponential integral.
Ellipse([center, hradius, vradius, eccentricity])	An elliptical GeometryEntity.
Eq	alias of Equality
Equality(lhs, rhs, **options)	An equal relation between two objects.
Equivalent(*args)	Equivalence relation.
Expr(*args)	Base class for algebraic expressions.
ExpressionDomain()	A class for arbitrary expressions.
FF	alias of FiniteField
FF_gmpy	alias of GMPYFiniteField
FF_python	alias of PythonFiniteField
FallingFactorial(x, k)	Falling factorial (related to rising factorial) is a double
rallingractorial(x, k)	valued function arising in concrete mathematics, hypergeometric functions and series expansions.
<pre>FiniteField(mod[, symmetric])</pre>	Finite field of prime order GF(p)
FiniteSet(*args, **kwargs)	Represents a finite set of Sympy expressions.
Float(num[, dps, precision])	Represent a floating-point number of arbitrary precision.
	continues on next page

Table 61 – continued from previous page

Class representing unevaluated Fourier transforms. A class for representing multivariate rational function fields. Function("args)	Table 61 – continue	ed from previous page
FractionField(domain_or_field[, symbols, order]) Function(*args) Function(*args, **ekwargs) Function(args, **ekwargs) FiniteField(mod[, symmetric]) Ge	FourierTransform(*args)	Class representing unevaluated Fourier transforms.
Function(*ags) Function(*ags) Function(*ags) Function(*ags) Function(*ags, **#kwargs) Function(*	· · · · · · · · · · · · · · · · · · ·	A class for representing multivariate rational function
FunctionClass(*args, **tkwargs) FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF GMPYIniteField(mod[, symmetric])  GMPYRationalField()  GMPYIntegerRing()  GMPYRationalField()  GreaterThan(lhs, rhs, **options)  GreaterThan(lhs, rhs, **options)  GroeaterThan(lhs, rhs, rhs, rhs, rhs, rhs, rhs, rhs, r	Function(*args)	
FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  GMPYRationalField()  GreaterThan(lbs, rhs, **options)  GroebneBasis(F, *gens, **args)  Gt adias of GreaterThan  GroebneBasis(F, *gens, **args)  Gt adiasof GreaterThan  HadamardPower(base, exp)  Hadama		
gives outputs according to the coordinates of each matrix entries.  GF GHPYIniteField(mod[, symmetric]) GMPYIniteGerRing() GMPYIniteGerRing() GMPYRationalField() GRPYRationalField() Ge GreatetThan(lhs, rhs, **options) GreatetThan(lhs, rhs, **options) GroebnerBasis(F, *gens, **args) GreatetThan(lhs, rhs, estephical) Ge AdamardPower(base, exp) HadamardPower(base, exp) HadamardProduct(*args[, evaluate, check]) HankelTransform(*args) HadamardProduct(*args[, evaluate, check]) HankelTransform(*args) Heaviside(arg[, H0]) Heaviside step function. Heaviside step function at step step step step step step step ste		
trix entries.  GF  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYIntegerRing()  GMPYIntegerRing()  GMPYAtionalField()  Rational field based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpz type.  GreaterThan(lhs, rhs, **options)  GreaterThan(Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  alias of StrictGreaterThan  HadamardPower(base, exp)  HadamardProduct(*argsl, evaluate, check])  HarkelTransform(*args)  HadamardProduct(*argsl, evaluate, check])  HarkelTransform(*args)  Haviside(argl, HOl)  Heaviside step function.  ITE(*args)  If-then-else clause.  If-then-else clause.  If-then-else clause.  If-then-else clause.  ImmutableDenseMatrix(*args, **kwargs)  ImmutableDenseMatrix(*args, **kwargs)  ImmutableDenseMatrix(*args, **kwargs)  ImmutableDenseMatrix(*args, **skwargs)  ImmutableDenseMatrix(*args, **skwargs)  ImmutableDenseMatrix(*args, **skwargs)  ImmutableSparseMatrix(*args, **skwargs)  ImmutableSparseMatrix(*args, **skwargs)  ImmutableSparseMatrix(*args, **skwargs)  Indexed(base, *args, **skw_args)  Indexed(base, *args, **skw_args)  Indexed(base, *args, **skw_args)  Indexed(base, *args, **skw_args)  IntegerRing()  IntegerRing()  IntegerRing()  IntegerRing()  IntegerRing()  IntegerRing()  IntegerRing()  InterperAl(Inction, *symbols, **assumptions)  Intersection(*args, **swargs)  Represents a mathematical object with indices.  Represents integer numbers of any size.  The domain ZZ representing the integers mathbb[Z].  Represents a mitersection of sets as a Set.  Represents an intersection of sets as a Set.  Interval(start, end, left_open, right_open))  InverseCosineTransform(*args)  Class representing unevaluated inverse Fourier transforms.  InverseHallanTransform(*args)  Class representing unevaluated inverse Fourier transforms.  Class representing unevaluated inverse Hankel transforms.  InverseMellinTransform(*args)  Class representing unevalua	2 4110 (2011) (2011) (2011) (411)	
GMPYIniteField(mod[, symmetric])   Finite field based on GMPY integers.		
GMPYIniteField(mod[, symmetric])   Finite field based on GMPY integers.	GF	alias of FiniteField
MPYIntegerRing()	<pre>GMPYFiniteField(mod[, symmetric])</pre>	
GMPYRationalField() Ge alias of GreaterThan GreaterThan(lhs, rhs, **options) GroebnerBasis(F, *gens, **args) GroebnerBasis(F, *gens, **args) Gt alias of StrictGreaterThan HadamardPower(base, exp) HadamardPower(base, exp) HadamardProduct(*args], evaluate, check]) HankelTransform(*args) HakelTransform(*args)  If-then-else clause.  ImageSet(flambda, *sets) ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) IndexedBase(label], shape, offset, strides])  Implies(*args) IndexedBase(label], shape, offset, strides]) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Intersection(*args, **sumptions) Intersection(*args) Class representing unevaluated inverse Fourier transforms. InverseBellinTransform(*args) Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated		
GreaterThan(lhs, rhs, **options) GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. Gt HadamardPower(base, exp) HadamardProduct(*args[, evaluate, check]) HadamardProduct(*args[, evaluate, check]) HadamardProduct(*args[, evaluate, check]) Hakaltransform(*args) Heaviside(arg[, H0]) Heaviside (arg[, H0]) Heaviside (arg[, H0]) Heaviside (arg[, H0]) Heaviside (arger) Hatrix (lentity I - multiplicative identity Heaviside (langer) Represents an integer index as an Integer or integer expression. ImageSet(flambda, *sets) ImmutablePenseMatrix(*args, **kwargs) ImmutableDenseNDimArray(iterable[, shape])  ImmutableMatrix ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Integer(i) Represents a mathematical object with indices. Represents integer numbers of an indexed object Integer(i) Integer(i) Represents integer numbers of an indexed object Interval(start, end], left_open, right_open) InverseCosineTransform(*args) InverseHalnkelTransform(*args) Class representing unevaluated inverse Casine transforms. InverseHalnaceTransform(*args) Class representing unevaluated inverse Hallour transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Hallour transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Sinetransforms. Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Sinetransforms.		
GroebnerBasis(F, *gens, **args) Gt alias of StrictGreaterThan HadamardPower(base, exp) HadamardProduct(*args[, evaluate, check]) HankelTransform(*args) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside step function.  ITE(*args) Identity(n) ItE(*args) Identity(n) ItAx(label[, range]) Represents an integer index as an Integer or integer expression.  ImageSet(flambda, *sets) ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) Indexed(base, *args, **kwargs) Indexed(base, *args, **kwargs) Integer(i) Represents an anthematical object with indices. IntegerRing() IntegerRing() IntegerRing() Integral(function, *symbols, **assumptions) Interval(start, end], left_open, right_open]) InverseCosineTransform(*args) InverseFourieTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseHalinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse sine transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.		· · ·
GroebnerBasis(F, *gens, **args) Gt alias of StrictGreaterThan HadamardPower(base, exp) HadamardProduct(*args[, evaluate, check]) HankelTransform(*args) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside step function.  ITE(*args) Identity(n) ItE(*args) Identity(n) ItAx(label[, range]) Represents an integer index as an Integer or integer expression.  ImageSet(flambda, *sets) ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) Indexed(base, *args, **kwargs) Indexed(base, *args, **kwargs) Integer(i) Represents an anthematical object with indices. IntegerRing() IntegerRing() IntegerRing() Integral(function, *symbols, **assumptions) Interval(start, end], left_open, right_open]) InverseCosineTransform(*args) InverseFourieTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseHalinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse sine transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.	GreaterThan(lhs, rhs, **options)	Class representations of inequalities.
HadamardPower(base, exp)   Elementwise power of matrix expressions   HadamardProduct(*args], evaluate, check])   Elementwise product of matrix expressions   HankelTransform(*args)   Class representing unevaluated Hankel transforms.   Heaviside(arg[, HO])   Heaviside step function.   Heaviside step function.   ITE(*args)   If-then-else clause.   Heaviside step function.   The Matrix Identity I - multiplicative identity   Idx(label], range])   Represents an integer index as an Integer or integer expression.   Image of a set under a mathematical function.   Create an immutableDenseMatrix(*args, **kwargs)   Create an immutable version of a matrix.   ImmutableDenseMatrix(*args, **kwargs)   Create an immutable version of a sparse matrix.   ImmutableSparseMatrix(*args, **kwargs)   Logical implication.   Indexed(base, *args, **kw_args)   Represents a mathematical object with indices.   Indexed(base, *args, **kw_args)   Represent the base or stem of an indexed object   Integer(i)   Represent the base or stem of an indexed object   Represents integer numbers of any size.   Integer(i)   Integer(i)   Represents an intersection of sets as a Set.   Interval(start, end], left_open, right_open )   Intersect(ont*args, **kwargs)   Class representing unevaluated inverse Fourier transforms.   InverseFourierTransform(*args)   Class representing unevaluated inverse Hankel transforms.   InverseLaplaceTransform(*args)   Class representing unevaluated inverse Mellin transforms.   InverseMellinTransform(*args)   Class representing unevaluated inverse sine transforms.   InverseSineTransform(*args)   Class representing unevaluated inverse	GroebnerBasis(F, *gens, **args)	Represents a reduced Groebner basis.
HadamardProduct(*args], evaluate, check])  HankelTransform(*args)  Heaviside(argf, H0f)  ITE(*args)  Identity(n)  Idx(label[, range])  ImageSet(flambda, *sets)  ImmutableDenseMatrix(*args, **kwargs)  ImmutableDenseNDimArray(iterable[, shape])  ImmutableSparseMatrix(*args, **kwargs)  ImmutableSparseMortin(*args)  Indexed(base, *args, **kw_args)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Intersection(*args, **kwargs)  Intersection(*args, **kwargs)  Intersection(*args, **kwargs)  Intersection(*args, **kwargs)  Intersection(*args, **kwargs)  InverseCosineTransform(*args)  InverseHankelTransform(*args)  Class representing unevaluated inverse delin transforms.  InverseHankelTransform(*args)  Class representing unevaluated inverse delin transforms.  Create an immutable version of a sparse matrix.  Create an immutable version of a sparse matrix.  Create an immutable version of a sparse matrix.  Logical implication.  Represents a mathematical object with indices.  Represents the base or stem of an indexed object Represents integer numbers of any size.  Represents unevaluated integral.  Represents unevaluated integral.  Represents a real interval as a Set.  Represents a real interval as a Set.  Class representing unevaluated inverse Cosine transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated inverse Laplace transforms.	Gt	alias of StrictGreaterThan
HankelTransform(*args) Heaviside(arg[, H0]) Heaviside(arg[, H0]) Heaviside step function.  ITE(*args) If-then-else clause.  Identity(n) The Matrix Identity I - multiplicative identity Idx(label[, range]) Represents an integer index as an Integer or integer expression.  Image Set(flambda, *sets) Image of a set under a mathematical function.  ImmutableDenseMatrix(*args, **kwargs) ImmutableMatrix ImmutableMatrix ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseNDimArray([iterable, shape])  Implies(*args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Integer(i) Integer(i) IntegerRing() Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Intersection(*args, **kwargs) Represents integer numbers of any size. Interval(start, end , left_open, right_open ) Represents a real interval as a Set. Inverse(mat[, exp]) InverseCosineTransform(*args) Class representing unevaluated inverse of an matrix expression InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.	HadamardPower(base, exp)	Elementwise power of matrix expressions
Heaviside(arg[, H0])	<pre>HadamardProduct(*args[, evaluate, check])</pre>	<u>.</u>
If-then-else clause.   If-then-else clause.   The Matrix Identity I - multiplicative identity	<pre>HankelTransform(*args)</pre>	Class representing unevaluated Hankel transforms.
Identity(n) Idx(label[, range]) Represents an integer index as an Integer or integer expression.  ImageSet(flambda, *sets) Image of a set under a mathematical function.  ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseMotinArray(iterable[, shape])  ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMotinArray([iterable, shape])  ImmutableSparseMotinArray([iterable, shape])  Implies(*args) Indexed(base, *args, **kwargs) Indexed(base, *args, **kwargs) Indexed(base, *args, **kwargs) Indexed(base, *args, **kwargs) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(integer	Heaviside(arg[, H0])	Heaviside step function.
Represents an integer index as an Integer or integer expression.   ImageSet(flambda, *sets)   Image of a set under a mathematical function.   Create an immutableDenseMatrix(*args, **kwargs)   Create an immutable version of a matrix.	ITE(*args)	If-then-else clause.
Expression.   Image Set(flambda, *sets)   Image of a set under a mathematical function.	Identity(n)	
ImageSet(flambda, *sets) Image of a set under a mathematical function.  ImmutableDenseMatrix(*args, **kwargs)  ImmutableDenseNDimArray(iterable[, shape])  ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseNDimArray([iterable, shape])  Implies(*args) Implies(*args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) IndexedBase(label[, shape, offset, strides]) Integer(i) Integer(i) Integer(i) Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Interval(start, end[, left_open, right_open]) Inverse(osineTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseLaplaceTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Sine transforms. Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Mellin transforms.	Idx(label[, range])	
ImmutableDenseMatrix(*args, **kwargs) ImmutableDenseNDimArray(iterable[, shape])  ImmutableMatrix ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseNDimArray([iterable, shape])  Implies(*args) Implies(*args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Integer(i) Integer(i) IntegerRing() Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Inverse(mat[, exp]) Inverse(mat[, exp]) InverseCosineTransform(*args) InverseFourierTransform(*args) Class representing unevaluated inverse cosine transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.	ImageSet(flambda, *sets)	•
ImmutableDenseNDimArray(iterable[, shape])  ImmutableMatrix ImmutableSparseMatrix(*args, **kwargs) ImmutableSparseNDimArray([iterable, shape])  Implies(*args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Integer(i) Integer(i) IntegerRing() IntegerRing() IntegerAl(function, *symbols, **assumptions) Intersection(*args, **kwargs) Interval(start, end[, left_open, right_open]) InverseCosineTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseLaplaceTransform(*args) InverseLaplaceTransform(*args) InverseMellinTransform(*args) InverseSineTransform(*args) Class representing unevaluated inverse Laplace transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.		
ImmutableSparseMatrix(*args, **kwargs)       Create an immutable version of a sparse matrix.         ImmutableSparseNDimArray([iterable, shape])       Create an immutable version of a sparse matrix.         Implies(*args)       Logical implication.         Indexed(base, *args, **kw_args)       Represents a mathematical object with indices.         IndexedBase(label[, shape, offset, strides])       Represents a mathematical object with indices.         Integer(i)       Represents a mathematical object with indices.         Integer(i)       Represents a mathematical object with indices.         Integer(i)       Represents a mathematical object with indices.         Integer(ii)       Represents a mathematical object with indices.         Integer(iii)       Represents a mathematical object with indices.         Integer(iii)       Represents a mathematical object with indices.         Integer(iiii)       Represents a mathematical object with indices.         Integer(iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii		
ImmutableSparseNDimArray([iterable, shape])  Implies(*args)	ImmutableMatrix	alias of ImmutableDenseMatrix
ImmutableSparseNDimArray([iterable, shape])  Implies(*args)	<pre>ImmutableSparseMatrix(*args, **kwargs)</pre>	Create an immutable version of a sparse matrix.
Indexed(base, *args, **kw_args)  IndexedBase(label[, shape, offset, strides])  Integer(i)  IntegerRing()  IntegerRing()  Integral(function, *symbols, **assumptions)  Intersection(*args, **kwargs)  Interval(start, end[, left_open, right_open])  Inverse(mat[, exp])  InverseCosineTransform(*args)  InverseFourierTransform(*args)  InverseHankelTransform(*args)  InverseLaplaceTransform(*args)  InverseMellinTransform(*args)  InverseMellinTransform(*args)  InverseSineTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.		
Indexed(base, *args, **kw_args)  IndexedBase(label[, shape, offset, strides])  Integer(i)  IntegerRing()  IntegerRing()  Integral(function, *symbols, **assumptions)  Intersection(*args, **kwargs)  Interval(start, end[, left_open, right_open])  Inverse(mat[, exp])  InverseCosineTransform(*args)  InverseFourierTransform(*args)  InverseHankelTransform(*args)  InverseLaplaceTransform(*args)  InverseMellinTransform(*args)  InverseMellinTransform(*args)  InverseSineTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.	<pre>Implies(*args)</pre>	Logical implication.
IndexedBase(label[, shape, offset, strides])  Integer(i)  IntegerRing()  IntegerRing()  Integral(function, *symbols, **assumptions)  Intersection(*args, **kwargs)  Interval(start, end[, left_open, right_open])  InverseCosineTransform(*args)  InverseFourierTransform(*args)  InverseHankelTransform(*args)  InverseLaplaceTransform(*args)  InverseMellinTransform(*args)  InverseMellinTransform(*args)  InverseSineTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Mellin transforms.		
IntegerRing() Integral(function, *symbols, **assumptions) Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Represents unevaluated integral. Interval(start, end[, left_open, right_open]) Represents a real interval as a Set. Inverse(mat[, exp]) Inverse(mat[, exp]) InverseCosineTransform(*args) Class representing unevaluated inverse cosine transforms. InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms. InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.	<pre>IndexedBase(label[, shape, offset, strides])</pre>	Represent the base or stem of an indexed object
Integral(function, *symbols, **assumptions)  Intersection(*args, **kwargs)  Interval(start, end[, left_open, right_open])  Inverse(mat[, exp])  Inverse(osineTransform(*args)  InverseFourierTransform(*args)  InverseHankelTransform(*args)  InverseLaplaceTransform(*args)  InverseMellinTransform(*args)  InverseMellinTransform(*args)  Class representing unevaluated inverse Fourier transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated inverse Mellin transforms.	Integer(i)	Represents integer numbers of any size.
Intersection(*args, **kwargs) Represents an intersection of sets as a Set.  Interval(start, end[, left_open, right_open]) Represents a real interval as a Set.  Inverse(mat[, exp]) The multiplicative inverse of a matrix expression Class representing unevaluated inverse cosine transforms.  InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse sine transforms.	<pre>IntegerRing()</pre>	The domain ZZ representing the integers $mathbb\{Z\}$ .
Interval(start, end[, left_open, right_open])  Inverse(mat[, exp])  InverseCosineTransform(*args)  InverseFourierTransform(*args)  Class representing unevaluated inverse cosine transforms.  InverseHankelTransform(*args)  Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args)  Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.	<pre>Integral(function, *symbols, **assumptions)</pre>	Represents unevaluated integral.
Inverse(mat[, exp]) InverseCosineTransform(*args) Class representing unevaluated inverse cosine transforms.  InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.	<pre>Intersection(*args, **kwargs)</pre>	Represents an intersection of sets as a Set.
InverseCosineTransform(*args)  Class representing unevaluated inverse cosine transforms.  InverseFourierTransform(*args)  Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args)  Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.	<pre>Interval(start, end[, left_open, right_open])</pre>	<u> </u>
forms.  InverseFourierTransform(*args)  Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args)  Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.		•
forms.  InverseHankelTransform(*args)  Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.	<pre>InverseCosineTransform(*args)</pre>	· •
forms.  InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.	InverseFourierTransform(*args)	· · · · · · · · · · · · · · · · · · ·
forms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.	<pre>InverseHankelTransform(*args)</pre>	
InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.	<pre>InverseLaplaceTransform(*args)</pre>	Class representing unevaluated inverse Laplace trans-
InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.	<pre>InverseMellinTransform(*args)</pre>	Class representing unevaluated inverse Mellin trans-
	InverseSineTransform(*args)	Class representing unevaluated inverse sine transforms.

Table 61 – continue	d from previous page
<pre>KroneckerDelta(i, j[, delta_range])</pre>	The discrete, or Kronecker, delta function.
<pre>KroneckerProduct(*args[, check])</pre>	The Kronecker product of two or more arguments.
Lambda(signature, expr)	Lambda(x, expr) represents a lambda function similar to
	Python's 'lambda x: expr'.
LambertW(x[,k])	The Lambert W function \$W(z)\$ is defined as the in-
	verse function of \$w exp(w)\$ [1]
LaplaceTransform(*args)	Class representing unevaluated Laplace transforms.
Le	alias of LessThan
LessThan(lhs, rhs, **options)	Class representations of inequalities.
LeviCivita(*args)	Represent the Levi-Civita symbol.
Li(z)	The offset logarithmic integral.
Limit(e, z, z0[, dir])	Represents an unevaluated limit.
Line(*args, **kwargs)	An infinite line in space.
Line2D(p1[, pt, slope])	An infinite line in space 2D.
Line3D(p1[, pt, direction_ratio])	An infinite 3D line in space.
Lt	alias of StrictLessThan
<pre>MatAdd(*args[, evaluate, check, _sympify])</pre>	A Sum of Matrix Expressions
<pre>MatMul(*args[, evaluate, check, _sympify])</pre>	A product of matrix expressions
<pre>MatPow(base, exp[, evaluate])</pre>	
Matrix	alias of MutableDenseMatrix
<pre>MatrixBase()</pre>	Base class for matrix objects.
<pre>MatrixExpr(*args, **kwargs)</pre>	Superclass for Matrix Expressions
<pre>MatrixPermute(mat, perm[, axis])</pre>	Symbolic representation for permuting matrix rows or
	columns.
MatrixSlice(parent, rowslice, colslice)	A MatrixSlice of a Matrix Expression
MatrixSymbol(name, n, m)	Symbolic representation of a Matrix object
Max(*args)	Return, if possible, the maximum value of the list.
MellinTransform(*args)	Class representing unevaluated Mellin transforms.
Min(*args)	Return, if possible, the minimum value of the list.
Mod(p, q)	Represents a modulo operation on symbolic expressions.
Monomial(monom[, gens])	Class representing a monomial, i.e. a product of powers.
Mul(*args[, evaluate, _sympify])	Expression representing multiplication operation for algebraic field.
<pre>MutableDenseMatrix(*args, **kwargs)</pre>	
<pre>MutableDenseNDimArray([iterable, shape])</pre>	
MutableMatrix	alias of MutableDenseMatrix
<pre>MutableSparseMatrix(*args, **kwargs)</pre>	
<pre>MutableSparseNDimArray([iterable, shape])</pre>	
NDimArray(iterable[, shape])	N-dimensional array.
Nand(*args)	Logical NAND function.
Ne	alias of Unequality
Nor(*args)	Logical NOR function.
Not(arg)	Logical Not function (negation)
Number(*obj)	Represents atomic numbers in SymPy.
NumberSymbol()	
0	alias of Order
	continues on next page

Table 61 – continued from previous page

	Parrecents ordinal exponential and multiplication terms
OmegaPower(a, b)	Represents ordinal exponential and multiplication terms
0 W 1 1 ( 1 ( 1)	one of the building blocks of the Ordinal class.
OneMatrix(m, n[, evaluate])	Matrix whose all entries are ones.
Options(gens, args[, flags, strict])	Options manager for polynomial manipulation module.
Or(*args)	Logical OR function
Order(expr, *args, **kwargs)	Represents the limiting behavior of some function.
Ordinal(*terms)	Represents ordinals in Cantor normal form.
Parabola([focus, directrix])	A parabolic GeometryEntity.
Permanent(mat)	Matrix Permanent
PermutationMatrix(perm)	A Permutation Matrix
Piecewise(*_args)	Represents a piecewise function.
Plane(p1[, a, b])	A plane is a flat, two-dimensional surface.
Point(*args, **kwargs)	A point in a n-dimensional Euclidean space.
Point2D(*args[, _nocheck])	A point in a 2-dimensional Euclidean space.
Point3D(*args[, _nocheck])	A point in a 3-dimensional Euclidean space.
Poly(rep, *gens, **args)	Generic class for representing and operating on polynomial expressions.
Polygon(*args[, n])	A two-dimensional polygon.
PolynomialRing(domain_or_ring[, symbols, order])	A class for representing multivariate polynomial rings.
Pow(b, e[, evaluate])	Defines the expression x**y as "x raised to a power y"
PowerSet(arg[, evaluate])	A symbolic object representing a power set.
Predicate(*args, **kwargs)	Base class for mathematical predicates.
Product(function, *symbols, **assumptions)	Represents unevaluated products.
ProductSet(*sets, **assumptions)	Represents a Cartesian Product of Sets.
	Class for representing pure polynomials.
PurePoly(rep, *gens, **args)	
PythonFiniteField(mod[, symmetric])	Finite field based on Python's integers.
PythonIntegerRing()	Integer ring based on Python's int type.
PythonRational	alias of PythonMPQ
QQ_gmpy	alias of GMPYRationalField
QQ_python	alias of PythonRationalField
Quaternion([a, b, c, d, real_field, norm])	Provides basic quaternion operations.
Range(*args)	Represents a range of integers.
Rational(p[, q, gcd])	Represents rational numbers (p/q) of any size.
RationalField()	Abstract base class for the domain QQ.
Ray(p1[, p2])	A Ray is a semi-line in the space with a source point and
	a direction.
Ray2D(p1[, pt, angle])	A Ray is a semi-line in the space with a source point and a direction.
Ray3D(p1[, pt, direction_ratio])	A Ray is a semi-line in the space with a source point and
kay 5D(p1[, pt, uncerton_tatto])	a direction.
RealField([prec, dps, tol])	Real numbers up to the given precision.
RealNumber	alias of Float
RegularPolygon(c, r, n[, rot])	
	A regular polygon.
Rel	alias of Relational
Rem(p, q)	Returns the remainder when p is divided by q where p is finite and q is not equal to zero.
RisingFactorial(x, k)	Rising factorial (also called Pochhammer symbol [1]_)
	is a double valued function arising in concrete mathe-
D .05(6 F. 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	matics, hypergeometric functions and series expansions.
RootOf(f, x[, index, radicals, expand])	Represents a root of a univariate polynomial.
RootSum(expr[, func, x, auto, quadratic]) Segment(p1, p2, **kwargs)	Represents a sum of all roots of a univariate polynomial.
	A line segment in space.

Table 61 – continued from previous page

Segment2D(p1, p2, **kwargs) Segment3D(p1, p2, **kwargs) SensitivityMatrix(sympification, duals,)  SeqAdd(*args, **kwargs) SeqFormula(formula[, limits]) SeqMul(*args, **kwargs) SeqPer(periodical[, limits]) Set(*args) Shi(z) Si(z) Sine integral. Sieve() An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes. SingUlarityFunction(variable, offset, exponent) SparseMatrix SparseNDimArray(*args, **kwargs) StrPrinter([settings])  StrictLessThan(lhs, rhs, **options) Sum(function, *symbols, **assumptions) Sum(function, *symbols, **assumptions) Represents in a 3D space. A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  A line segment in a 3D space.  Segment3Dtspace.  A line segment in a 3D space.  A line segment in a 3D space.  Segment3Dtspace.  Segment3Dtspace.  Sepresents term-wise addition of sequences.  Represents term-wise addition of sequences.  Represents in evaluated side in a formula.  SeqMul*(args, **kwargs)  Class representations of inequalities.  Class represents unevaluated substitutions of an expresson sum(function, *symbols, **assumptions)  Represents unevaluated summation.	a dy-
SeqAdd(*args, **kwargs)  SeqFormula(formula[, limits])  SeqMul(*args, **kwargs)  SeqFormula(formula[, limits])  SeqMul(*args, **kwargs)  SeqPer(periodical[, limits])  Set(*args)  Set(*args)  Shi(z)  Sine integral.  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args)  SingularityFunction(variable, offset, exponent)  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Subs(expr, variables, point, **assumptions)  Represents term-wise addition of sequences.  Represents unevaluated substitutions of sequences.  Represents unevaluated substitutions of an expresents unevaluated substitutions of an expresents.	a dy-
SeqAdd(*args, **kwargs)  SeqFormula(formula[, limits])  Represents sequence based on a formula.  SeqMul(*args, **kwargs)  Represents term-wise multiplication of sequences  SeqPer(periodical[, limits])  Represents a periodic sequence.  Set(*args)  The base class for any kind of set.  Shi(z)  Sinh integral.  Si(z)  Sine integral.  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SingUlarityFunction(variable, offset, exponent)  Singularity functions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrictGreaterThan(lhs, rhs, **options)  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents term-wise addition of sequences.  Represents term-wise multiplication of sequences.  Represents defined as option of sequences.  Represents term-wise multiplication of sequences.  Represents defined as option of sequences.  Represents term-wise multiplication of sequences.  Represents defined as option of sequences.  Represents term-wise multiplication of sequence.  Represents term-wise multiplication of sequence.  Represents term-wise multiplication of sequence.  Represents defined as option of sequence.  Represents term-wise multiplication of sequence.  Represents defined as option of sequence.  Represents a periodic sequence.  Represents de	a dy-
SeqFormula(formula[, limits])  SeqMul(*args, **kwargs)  SeqPer(periodical[, limits])  Set(*args)  Shi(z)  Sink integral.  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SingularityFunction(variable, offset, exponent)  SingularityFunction(variable, offset, exponent)  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Subs(expr, variables, point, **assumptions)  Represents sequence based on a formula.  Represents sequence based on a formula.  Represents sequence based on a formula.  Represents emultiplication of sequences  Represents a periodic sequence.  Sing hintegral.  Sine integral.  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  Class representing unevaluated sine transforms.  Singularity functions are a class of discontinuous tions.  Singularity functions are a class of discontinuous tions.  Class representations of inequalities.  Class representations of inequalities.  Represents unevaluated substitutions of an expresents unevaluated substitutions of an expresentations.	a dy-
SeqFormula(formula[, limits])  SeqMul(*args, **kwargs)  SeqPer(periodical[, limits])  Set(*args)  Shi(z)  Sink integral.  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SingularityFunction(variable, offset, exponent)  Singularity functions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Subs(expr, variables, point, **assumptions)  Represents sequence based on a formula.  Represents sequence based on a formula.  Represents sequence based on a formula.  Represents term-wise multiplication of sequences  Represents a periodic sequence.  Sing hard of set.  Sinh integral.  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  Class representing unevaluated sine transforms.  Singularity functions are a class of discontinuous tions.  Class representations of inequalities.  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an expresents unevaluated substitutions of an expresentations of an expresentatio	a dy-
SeqMul(*args, **kwargs)Represents term-wise multiplication of sequencesSeqPer(periodical[, limits])Represents a periodic sequence.Set(*args)The base class for any kind of set.Shi(z)Sinh integral.Si(z)Sine integral.Sieve()An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.SineTransform(*args)Class representing unevaluated sine transforms.SingularityFunction(variable, offset, exponent)Singularity functions are a class of discontinuous tions.SparseMatrixalias of MutableSparseMatrixSparseNDimArray(*args, **kwargs)StrPrinter([settings])StrictGreaterThan(lhs, rhs, **options)Class representations of inequalities.StrictLessThan(lhs, rhs, **options)Class representations of inequalities.Subs(expr, variables, point, **assumptions)Represents unevaluated substitutions of an expresentations.	a dy-
SeqPer(periodical[, limits])  Set(*args)  The base class for any kind of set.  Shi(z)  Sinh integral.  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args)  SingularityFunction(variable, offset, exponent)  SingularityFunctions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Subs(expr, variables, point, **assumptions)  Represents a periodic sequence.  Represents a periodic sequence.  Sind sequence.  Represents of inequalities.  Class representations of inequalities.  Represents unevaluated substitutions of an expresents unevaluated substitutions of an expresents unevaluated substitutions of an expresents	a dy-
Set(*args)  Shi(z)  Sink integral.  Si(z)  Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args)  SingularityFunction(variable, offset, exponent)  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Subs(expr, variables, point, **assumptions)  The base class for any kind of set.  Sinh integral.  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  Class representing unevaluated sine transforms.  Singularity functions are a class of discontinuous tions.  Schief (Settings)  Class representations of inequalities.  Class representations of inequalities.  Represents unevaluated substitutions of an express	
Shi(z) Sine integral. Sieve() An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args) Class representing unevaluated sine transforms.  SingularityFunction(variable, offset, exponent) Singularity functions are a class of discontinuous tions.  SparseMatrix SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options) Class representations of inequalities. StrictLessThan(lhs, rhs, **options) Class representations of inequalities. Subs(expr, variables, point, **assumptions) Represents unevaluated substitutions of an expresentations.	
Si(z) Sine integral. Sieve() An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args) Class representing unevaluated sine transforms. SingularityFunction(variable, offset, exponent) Singularity functions are a class of discontinuous tions.  SparseMatrix SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options) Class representations of inequalities. StrictLessThan(lhs, rhs, **options) Class representations of inequalities. StrictLessThan(lhs, rhs, **assumptions) Represents unevaluated substitutions of an express	
Sieve()  An infinite list of prime numbers, implemented as namically growing sieve of Eratosthenes.  SineTransform(*args)  Class representing unevaluated sine transforms.  SingularityFunction(variable, offset, exponent)  Singularity functions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Class representations of inequalities.  StrictLessThan(lhs, point, **assumptions)  Represents unevaluated substitutions of an expresentations.	
namically growing sieve of Eratosthenes.  SineTransform(*args)  SingularityFunction(variable, offset, exponent)  Singularity functions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **assumptions)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an express	
SingularityFunction(variable, offset, exponent)  Singularity functions are a class of discontinuous tions.  SparseMatrix  SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **assumptions)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an express	func-
tions.  SparseMatrix alias of MutableSparseMatrix  SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options) Class representations of inequalities.  StrictLessThan(lhs, rhs, **options) Class representations of inequalities.  Subs(expr, variables, point, **assumptions) Represents unevaluated substitutions of an expresentations.	func-
SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an expression	
SparseNDimArray(*args, **kwargs)  StrPrinter([settings])  StrictGreaterThan(lhs, rhs, **options)  StrictLessThan(lhs, rhs, **options)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an expression of the content of	
StrictGreaterThan(lhs, rhs, **options) StrictLessThan(lhs, rhs, **options) Class representations of inequalities. Class representations of inequalities. Subs(expr, variables, point, **assumptions) Represents unevaluated substitutions of an expres	
StrictGreaterThan(lhs, rhs, **options) StrictLessThan(lhs, rhs, **options) Class representations of inequalities. Class representations of inequalities. Subs(expr, variables, point, **assumptions) Represents unevaluated substitutions of an expres	
StrictLessThan(lhs, rhs, **options)  Class representations of inequalities.  Subs(expr, variables, point, **assumptions)  Represents unevaluated substitutions of an expres	
Subs(expr, variables, point, **assumptions) Represents unevaluated substitutions of an expres	
Sum(function, *symbols, **assumptions) Represents unevaluated summation.	sion.
Symbol(name, **assumptions) Assumptions:	
SymmetricDifference(a, b[, evaluate]) Represents the set of elements which are in either sets and not in their intersection.	of the
TableForm(data, **kwarg) Create a nice table representation of data.	
TestBabyModel(*args, **kwds)	
Trace(mat) Matrix Trace	
Transpose(*args, **kwargs)  The transpose of a matrix expression.	
Triangle(*args, **kwargs)  A polygon with three vertices and three sides.	
Tuple(*args, **kwargs) Wrapper around the builtin tuple object.	
Unequality(lhs, rhs, **options)  An unequal relation between two objects.	
UnevaluatedExpr(arg, **kwargs) Expression that is not evaluated unless released.	
Union(*args, **kwargs) Represents a union of sets as a Set.	
Wild(name[, exclude, properties])  A Wild symbol matches anything w whatever is explicitly excluded.	ithout
WildFunction(*args)  A WildFunction function matches any function (warguments).	ith its
Xor(*args) Logical XOR (exclusive OR) function.	
Ynm(n, m, theta, phi)  Spherical harmonics defined as	
ZZ_gmpy alias of GMPYIntegerRing	
ZZ_python alias of PythonIntegerRing	
ZeroMatrix(m, n) The Matrix Zero 0 - additive identity	
Znm(n, m, theta, phi)  Real spherical harmonics defined as	
acos(arg) The inverse cosine function.	
$a\cosh(x)$ is the inverse hyperbolic cosine of $x$ .	
acot(arg) The inverse cotangent function.	
acoth(arg) acoth(x) is the inverse hyperbolic cotangent of	

Table 61 – continued from previous page

	d from previous page
acsc(arg)	The inverse cosecant function.
acsch(arg)	acsch(x) is the inverse hyperbolic cosecant of $x$ .
adjoint(arg)	Conjugate transpose or Hermite conjugation.
airyai(arg)	The Airy function \$operatorname{Ai}\$ of the first kind.
airyaiprime(arg)	The derivative \$operatorname{Ai}^prime\$ of the Airy
	function of the first kind.
airybi(arg)	The Airy function \$operatorname{Bi}\$ of the second kind.
airybiprime(arg)	The derivative \$operatorname{Bi}^prime\$ of the Airy function of the first kind.
andre(n)	Andre numbers / Andre function
appellf1(a, b1, b2, c, x, y)	This is the Appell hypergeometric function of two vari-
*	ables as:
arg(arg)	Returns the argument (in radians) of a complex number.
asec(arg)	The inverse secant function.
asech(arg)	asech(x) is the inverse hyperbolic secant of x.
asin(arg)	The inverse sine function.
asinh(arg)	asinh(x) is the inverse hyperbolic sine of x.
assoc_laguerre(n, alpha, x)	Returns the $n\$ th generalized Laguerre polynomial in $x\$ , $L_n(x)\$ .
$assoc\_legendre(n, m, x)$	assoc_legendre(n, m, x) gives $P_n^m(x)$ ,
	where \$n\$ and \$m\$ are the degree and order or an
	expression which is related to the nth order Legendre
	polynomial, $P_n(x)$ in the following manner:
atan(arg)	The inverse tangent function.
atan2(y, x)	The function $atan2(y, x)$ computes operator-
	$name\{atan\}(y/x)$ taking two arguments $y$ and $x$ .
atanh(arg)	atanh(x) is the inverse hyperbolic tangent of x.
bell(n[, k_sym, symbols])	Bell numbers / Bell polynomials
bernoulli(n[, x])	Bernoulli numbers / Bernoulli polynomials / Bernoulli function
besseli(nu, z)	Modified Bessel function of the first kind.
besselj(nu, z)	Bessel function of the first kind.
besselk(nu, z)	Modified Bessel function of the second kind.
bessely(nu, z)	Bessel function of the second kind.
beta(x[,y])	The beta integral is called the Eulerian integral of the first kind by Legendre:
betainc(*args)	The Generalized Incomplete Beta function is defined as
betainc_regularized(*args)	The Generalized Regularized Incomplete Beta function
	is given by
binomial(n, k)	Implementation of the binomial coefficient.
<pre>carmichael(*args)</pre>	Carmichael Numbers:
cartes	alias of product
catalan(n)	Catalan numbers
ceiling(arg)	Ceiling is a univariate function which returns the small-
2. 5,	est integer value not less than its argument.
chebyshevt(n, x)	Chebyshev polynomial of the first kind, $T_n(x)$ .
chebyshevt_root(n, k)	chebyshev_root(n, k) returns the \$k\$th root (in-
•	dexed from zero) of the \$n\$th Chebyshev polynomial of
	the first kind; that is, if $0 le k < n$ , chebyshevt(n,
	$chebyshevt\_root(n, k)) == 0.$
chebyshevu(n, x)	Chebyshev polynomial of the second kind, $U_n(x)$ .
- (, ,	continues on next nage

Table 61 – continued from previous page

chebyshevu_root(n, k)  chebyshevu_root(n, k)  chebyshevu_root(n, k)  chebyshevu_root(n, k)  conjugate(arg)  Returns the complex conjugate[1] of an argument.  cos(arg)  The cosine function.  cosh(arg)  coth(arg)  coth(arg	Table 61 – continue	ed from previous page
conjugate(arg) Cos(arg) Cos(arg) Cos(arg) Cos(arg) Cot(arg) Cot(ar	chebyshevu_root(n, k)	dexed from zero) of the $n\$ th Chebyshev polynomial of the second kind; that is, if $0 le k < n$ , chebyshevu(n,
cos(arg) cosh(arg) cosh(arg) cosh(arg) coth(arg) coth(ar	conjugate(arg)	
cosh(arg)         cosh(x) is the hyperbolic cosine of x.           cot(larg)         The cotangent function.           cot(larg)         The cotangent function.           csc(arg)         The cose and function.           csc(larg)         csch(x) is the hyperbolic cosecant of x.           defaultdict         defaultdict(default_factory=None, /, []) -> dict with default factory           digamma(z)         The digamma function is the first derivative of the loggamma function.           divisor_sigma(a, k)         Calculate the divisor function sigma_k(n) for positive integer n           elliptic_e(mf, z)         Called with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined by           elliptic_k(m)         The Legendre incomplete elliptic integral of the first kind, defined by           elliptic_pi(n, mf, z])         Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the first kind, defined by           elliptic_pi(n, mf, z])         Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by           erf(arg)         The Gauss error function.           erf(arg)         The Gauss error function.           erf(x, y)         Two-argument error function.           erf(x, y)         Two-argument hreese error function.           erf(x)         Investe Complementary Error Functi		
cot(arg)		
coth(arg) csch(arg) csch(x) is the hyperbolic cosecant of x. defaultdict defaultdict defaultdict default factory default factory digamma(z) The digamma function is the first derivative of the loggamma function Dirichlet eta function.  dirichlet_eta(s[, a]) Dirichlet eta function.  divisor_sigma(n[, k]) Calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the divisor function sigma_k(n) for positive integer n calculate the div	The state of the s	
csc(arg) csch(arg) csch(ar	ν ε,	
csch(arg)         csch(x) is the hyperbolic cosecant of x defaultdict (default_factory=None, /, []) -> dict with default factory           digamma(z)         The digamma function is the first derivative of the loggamma function           dirichlet_eta(s[, a])         Dirichlet eta function.           divisor_sigma(n[, k])         Calculate the divisor function sigma_k(n) for positive integer n logger n           elliptic_e(m[, z])         Called with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined by           elliptic_f(z, m)         The Legendre incomplete elliptic integral of the first kind, defined by           elliptic_pi(n, m[, z])         Called with three arguments \$s\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by           elliptic_pi(n, m[, z])         Called with three arguments \$s\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by           erf(arg)         The Gauss error function.           erf(arg)         Two-argument error function.           erf(arg)         Two-argument error function.           erf(arg)         Two-argument perror function.           erf(arg)         Inverse cror function.           erf(arg)         Inverse cror function.           erf(arg)         Inverse Cror function.           erf(arg)         Inverse Error Function.           erf(inv(z) <t< td=""><td>· ·</td><td>• • • • • • • • • • • • • • • • • • • •</td></t<>	· ·	• • • • • • • • • • • • • • • • • • • •
defaultdict         defaultdict(default_factory=None, /, []) -> dict with default factory           digamma(z)         The digamma function is the first derivative of the logamma function.           dirichlet_eta(s[a])         Dirichlet teat function.           divisor_sigma(n[, k])         Calculate the divisor function sigma_k(n) for positive integer n           elliptic_e(m[, z])         Called with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined by           elliptic_k(m)         The Legendre incomplete elliptic integral of the first kind, defined by           elliptic_pi(n, m[, z])         Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by           erf(arg)         The Gauss error function.           erf2(x)         Two-argument ror function.           erf2(x)         Two-argument nerror function.           erf2(arg)         Complementary Error Function.           erf2(arg)         Complementary Error Function.           erf2(nv(z)         Inverse Complementary Error Function.           erf1(x)         Inverse Error Function.           erf1(x)         Euler numbers / Euler polynomials / Euler function exp(arg)           exp_polar(*args)         Represent a polar number (see g-function Sphinx documentation).           exp_polar(*args)         Generalized exponential integral. <tr< td=""><td>, e,</td><td></td></tr<>	, e,	
loggamma function		defaultdict(default_factory=None, /, [])> dict with
divisor_sigma(n[, k])  clause the divisor function sigma_k(n) for positive integer n  clause with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined by  elliptic_f(z, m)  The Legendre incomplete elliptic integral of the first kind, defined by  elliptic_k(m)  The complete elliptic integral of the first kind, defined by  elliptic_pi(n, m[, z])  Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the first kind, defined by  erf(arg)  The Gauss error function.  erf2(x, y)  Two-argument error function.  erf2(arg)  The Gauss error function.  erf2(arg)  Complementary Error Function.  erf(arg)  Inverse Complementary Error Function.  erfi(n)  Inverse Complementary Error Function.  erfi(n)  Inverse Error Function.  erfi(n)  Euler numbers / Euler polynomials / Euler function exp(arg)  Exp_polar(*args)  Represent a polar number (see g-function Sphinx documentation).  expint(nu, z)  Generalized exponential integral.  factorial(n)  Implementation of factorial integral.  factorial(n)  Fibonacci (n[, sym])  Fibonacci numbers / Fibonacci polynomials  floor(arg)  Fibonacci numbers / Fibonacci polynomials  floor(arg)  Fica(arg)  Represents the fractional part of x  fresnel s(z)  fresnel integral C.  fresnel integral C.  fresnel integral C.  fresnels(z)  fresmel integral S.  gamma(arg)  Gegenbauer polynomial \$C_n^{\{left(alpharight)\}(x)}\$.	$\operatorname{digamma}(z)$	The digamma function is the first derivative of the
elliptic_e(m[, z])  elliptic_e(m[, z])  elliptic_e(m[, z])  elliptic_f(z, m)  elliptic_f(z, m)  elliptic_k(m)  elliptic_k(m)  elliptic_pi(n, m[, z])  called with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the first kind, defined by  elliptic_pi(n, m[, z])  called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the first kind, defined by  erf(arg)  erf(arg)  erf(arg)  erf(arg)  frow-argument error function.  erf2inv(x, y)  Two-argument havese error function.  erf2inv(x, y)  frow-argument havese error function.  erfc(arg)  erf(arg)  frow ergument arror function.  erf(arg)  erfinv(z)  linverse Complementary Error Function.  erfinv(z)  erfinv(z)  ender(n[, x])  euler(n[, x])  euler(n[, x])  exp(arg)  frow expensity  frow double factorial function over nonnegative integers.  frow integers.  frow integers.  frow integers.  frow integers integer floor is a univariate function which returns the largest integer value not greater than its argument.  frac(arg)  fresnel (z)  fresnel (z)  fresnel (z)  fresnel integral C.  fresnel integral C.  fresnel integral C.  fresnel integral C.  fresnel integral S.  gamma(arg)  The gamma function  Gegenbauer polynomial \$C_n^{left(alpharight)}(x)\$.	<pre>dirichlet_eta(s[, a])</pre>	Dirichlet eta function.
elliptic_e(m[, z])  class and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined by second kind, defined by second kind, defined by second kind, defined by the complete elliptic integral of the first kind, defined by the complete elliptic integral of the first kind, defined by the complete elliptic integral of the first kind, defined by the complete elliptic integral of the first kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the third kind, defined by the complete elliptic integral of the complete elliptic integral error function.  erfical error function.	$divisor\_sigma(n[,k])$	
kind, defined by elliptic_k(m) The complete elliptic integral of the first kind, defined by elliptic_pi(n, m[, z]) Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by erf(arg) The Gauss error function. erf2(x, y) Two-argument error function. erf2inv(x, y) Two-argument Inverse error function. erf2(arg) Complementary Error Function. erfc(arg) Complementary Error Function. erfc(arg) Inverse Complementary Error Function. erfinv(z) Inverse Error Function. erfinv(z) Inverse Error Function. erfinv(z) Inverse Error Function. euler(n[, x]) Euler numbers / Euler polynomials / Euler function exp(arg) The exponential function, ex exp_polar(*args) Represent a polar number (see g-function Sphinx documentation).  expint(nu, z) Generalized exponential integral. factorial(n) Implementation of factorial function over nonnegative integers. factorial2(arg) The double factorial n!!, not to be confused with (n!)! ff alias of FallingFactorial fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials floor(arg) Represents the fractional part of x fresnelc(z) Fresnel integral C. fresnels(z) Fresnel integral C. fresnels(z) Fresnel integral S. gamma(arg) Gegenbauer polynomial \$C_n^{[left(alpharight)]}(x)\$.	elliptic_e(m[, z])	Called with two arguments \$z\$ and \$m\$, evaluates the incomplete elliptic integral of the second kind, defined
elliptic_pi(n, m[, z])  Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by  erf(arg)  The Gauss error function. erf2(x, y)  Two-argument error function. erf2(arg)  Complementary Error Function. erfc(arg)  Complementary Error Function. erfi(z)  Inverse Complementary Error Function. erfinv(z)  Inverse Complementary Error Function. erfinv(z)  Inverse Error Function. erfinv(z)  Inverse Error Function. euler(n[, x])  Euler numbers / Euler polynomials / Euler function exp(arg)  The exponential function, e <sup>x</sup> . exp_polar(*args)  Represent a polar number (see g-function Sphinx documentation). expint(nu, z)  Generalized exponential integral. factorial(n)  Implementation of factorial function over nonnegative integers.  factorial2(arg)  The double factorial n!!, not to be confused with (n!)! ff  alias of FallingFactorial fibonacci(n[, sym])  fibonacci (numbers / Fibonacci polynomials floor(arg)  Fibonacci unumbers / Fibonacci polynomials floor(arg)  Fibonacci unumbers / Fibonacci polynomials frac(arg)  Represents the fractional part of x fresnelc(z) fresnels(z) Fresnel integral S. gamma(arg)  Gegenbauer polynomial \$C_n^{left(alpharight)}(x)\$.	$elliptic_f(z, m)$	
elliptic_pi(n, m[, z])       Called with three arguments \$n\$, \$z\$ and \$m\$, evaluates the Legendre incomplete elliptic integral of the third kind, defined by         erf(arg)       The Gauss error function.         erf2inv(x, y)       Two-argument error function.         erf2inv(x, y)       Two-argument Inverse error function.         erfc(arg)       Complementary Error Function.         erfcinv(z)       Inverse Complementary Error Function.         erfi(z)       Imaginary error function.         erfinv(z)       Inverse Error Function.         euler(n[, x])       Euler numbers / Euler polynomials / Euler function         exp(arg)       The exponential function, e <sup>x</sup> .         exp_polar(*args)       Represent a polar number (see g-function Sphinx documentation).         expint(nu, z)       Generalized exponential integral.         factorial(n)       Implementation of factorial function over nonnegative integers.         factorial2(arg)       The double factorial n!!, not to be confused with (n!)!         ff       alias of FallingFactorial         fibonacci(n[, sym])       Fibonacci numbers / Fibonacci polynomials         floor(arg)       Floor is a univariate function which returns the largest integer value not greater than its argument.         fresnel(c2)       Fresnel integral C.         fresnel integral S.       gamma(arg)       The gamma funct	elliptic_k(m)	
erf2(x, y) erf2inv(x, y) Erf2inv(x, y) Two-argument Inverse error function.  erfc(arg) Complementary Error Function.  erfcinv(z) Inverse Complementary Error Function.  erfi(z) Inverse Complementary Error Function.  erfi(z) Inverse Error Function.  erfinv(z) Inverse Error Function.  euler(n[, x]) Euler numbers / Euler polynomials / Euler function  exp(arg) The exponential function, ex.  exp_polar(*args) Represent a polar number (see g-function Sphinx documentation).  expint(nu, z) Generalized exponential integral.  factorial(n) Implementation of factorial function over nonnegative integers.  factorial2(arg) The double factorial n!!, not to be confused with (n!)!  ff alias of FallingFactorial fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials floor(arg) Fibonacci numbers / Fibonacci polynomials floor(arg) Represents the fractional part of x fresnel(z) fresnel(z) Fresnel integral C. fresnel integral S. gamma(arg) The gamma function Gegenbauer polynomial \$C_n^{[eft(alpharight)](x)\$.	<pre>elliptic_pi(n, m[, z])</pre>	ates the Legendre incomplete elliptic integral of the third
erf2inv(x, y)  erfc(arg)  complementary Error Function.  erfcinv(z)  erfi(z)  linverse Complementary Error Function.  erfiv(z)  erfinv(z)  linverse Error Function.  erfinv(z)  erfinv(z)  erfinv(z)  linverse Error Function.  euler(n[, x])  euler (n], x])  exp(arg)  factorial(n)  limplementation of factorial function over nonnegative integers.  factorial2(arg)  factorial2(arg)  factorial2(arg)  fibonacci(n[, sym])  fibonacci(n[, sym])  fibonacci(n(arg)  floor(arg)  Fibonacci numbers / Fibonacci polynomials  floor(arg)  Fibori is a univariate function which returns the largest integer value not greater than its argument.  frac(arg)  fresnel c(z)  fresnel integral C.  fresnels(z)  gamma(arg)  Fresnel integral S.  gamma function  Gegenbauer polynomial \$C_n^{[eft(alpharight)](x)\$.	erf(arg)	The Gauss error function.
erfciarg) Complementary Error Function. erfcinv(z) Inverse Complementary Error Function. erfi(z) Imaginary error function. erfinv(z) Inverse Error Function. euler(n[, x]) Euler numbers / Euler polynomials / Euler function exp(arg) The exponential function, ex. exp_polar(*args) Represent a polar number (see g-function Sphinx documentation). expint(nu, z) Generalized exponential integral. factorial(n) Implementation of factorial function over nonnegative integers. factorial2(arg) The double factorial n!!, not to be confused with (n!)! ff alias of FallingFactorial fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials floor(arg) Floor is a univariate function which returns the largest integer value not greater than its argument. frac(arg) fresnelc(z) fresnelc(z) fresnels(z) Fresnel integral C. fresnels(z) gamma(arg) The gamma function Gegenbauer polynomial \$C_n^{[eft(alpharight)](x)\$.	erf2(x, y)	Two-argument error function.
erfcinv(z) Inverse Complementary Error Function.  erfit(z) Imaginary error function.  erfinv(z) Inverse Error Function.  euler(n[, x]) Euler numbers / Euler polynomials / Euler function  exp(arg) The exponential function, e <sup>x</sup> .  exp_polar(*args) Represent a polar number (see g-function Sphinx documentation).  expint(nu, z) Generalized exponential integral.  factorial(n) Implementation of factorial function over nonnegative integers.  factorial2(arg) The double factorial n!!, not to be confused with (n!)!  ff alias of FallingFactorial  fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials  floor(arg) Floor is a univariate function which returns the largest integer value not greater than its argument.  frac(arg) Represents the fractional part of x  fresnelc(z) Fresnel integral C.  fresnels(z) Fresnel integral S.  gamma(arg) The gamma function  Gegenbauer polynomial \$C_n^{{left(alpharight)}(x)}\$.	erf2inv(x, y)	Two-argument Inverse error function.
erfinv(z) Inverse Error Function.  euler(n[, x]) Euler numbers / Euler polynomials / Euler function  exp(arg) The exponential function, ex.  exp_polar(*args) Represent a polar number (see g-function Sphinx documentation).  expint(nu, z) Generalized exponential integral.  factorial(n) Implementation of factorial function over nonnegative integers.  factorial2(arg) The double factorial n!!, not to be confused with (n!)!  ff alias of FallingFactorial  fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials  floor(arg) Floor is a univariate function which returns the largest integer value not greater than its argument.  frac(arg) Represents the fractional part of x  fresnelc(z) Fresnel integral C.  fresnels(z) Fresnel integral S.  gamma(arg) The gamma function  gegenbauer(n, a, x) Gegenbauer polynomial \$C_n^{{left(alpharight)}(x)}\$.	erfc(arg)	Complementary Error Function.
erfinv(z)Inverse Error Function.euler(n[, x])Euler numbers / Euler polynomials / Euler functionexp(arg)The exponential function, $e^x$ .exp_polar(*args)Represent a polar number (see g-function Sphinx documentation).expint(nu, z)Generalized exponential integral.factorial(n)Implementation of factorial function over nonnegative integers.factorial2(arg)The double factorial $n!!$ , not to be confused with $(n!)!$ ffalias of FallingFactorialfibonacci(n[, sym])Fibonacci numbers / Fibonacci polynomialsfloor(arg)Floor is a univariate function which returns the largest integer value not greater than its argument.frac(arg)Represents the fractional part of xfresnelc(z)Fresnel integral C.fresnels(z)Fresnel integral S.gamma(arg)The gamma functiongegenbauer (n, a, x)Gegenbauer polynomial $C_n^{\{left(alpharight)\}(x)\}$ .	erfcinv(z)	Inverse Complementary Error Function.
euler(n[, x])Euler numbers / Euler polynomials / Euler functionexp(arg)The exponential function, $e^x$ .exp_polar(*args)Represent a polar number (see g-function Sphinx documentation).expint(nu, z)Generalized exponential integral.factorial(n)Implementation of factorial function over nonnegative integers.factorial2(arg)The double factorial $n!!$ , not to be confused with $(n!)!$ ffalias of FallingFactorialfibonacci(n[, sym])Fibonacci numbers / Fibonacci polynomialsfloor(arg)Floor is a univariate function which returns the largest integer value not greater than its argument.frac(arg)Represents the fractional part of xfresnelc(z)Fresnel integral C.fresnels(z)Fresnel integral S.gamma(arg)The gamma functiongegenbauer(n, a, x)Gegenbauer polynomial $C_n^{\{eft(alpharight)\}(x)\}$ .	erfi(z)	
$\begin{array}{c} \exp(\arg) & \text{The exponential function, } e^x. \\ \exp\text{polar}(*\args) & \text{Represent a } polar  number  (\text{see g-function Sphinx documentation}). \\ \exp\text{polar}(*\args) & \text{Generalized exponential integral.} \\ \operatorname{factorial}(n) & \operatorname{Implementation of factorial function over nonnegative integers.} \\ \operatorname{factorial2(arg)} & \text{The double factorial } n!!,  \text{not to be confused with } (n!)! \\ \operatorname{ff} & \text{alias of FallingFactorial} \\ \operatorname{fibonacci}(n[, \operatorname{sym}]) & \operatorname{Fibonacci numbers } / \operatorname{Fibonacci polynomials} \\ \operatorname{floor}(\arg) & \operatorname{Floor is a univariate function which returns the largest integer value not greater than its argument.} \\ \operatorname{frac}(\arg) & \operatorname{Represents the fractional part of } x \\ \operatorname{fresnelc(z)} & \operatorname{Fresnel integral C.} \\ \operatorname{fresnels(z)} & \operatorname{Fresnel integral S.} \\ \operatorname{gamma}(\arg) & \operatorname{The gamma function} \\ \operatorname{gegenbauer}(n, a, x) & \operatorname{Gegenbauer polynomial} \$C_n^{\{\text{left}(alpharight)\}(x)\$.} \\ \end{array}$	erfinv(z)	Inverse Error Function.
exp_polar(*args)  expint(nu, z)  factorial(n)  factorial2(arg)  fif  fibonacci(n[, sym])  floor(arg)  fresnelc(z)  fresnelc(z)  fresnels(z)  gamma(arg)  Represent a polar number (see g-function Sphinx documentation).  Represent a polar number (see g-function Sphinx documentation).  Generalized exponential integral.  Implementation of factorial function over nonnegative integers.  The double factorial n!!, not to be confused with (n!)!  ff alias of FallingFactorial  Fibonacci numbers / Fibonacci polynomials  Floor is a univariate function which returns the largest integer value not greater than its argument.  Fresnel integral C.  Fresnel integral C.  Fresnel integral S.  gamma(arg)  The gamma function  Gegenbauer polynomial \$C_n^{left(alpharight)}(x)\$.	euler(n[,x])	Euler numbers / Euler polynomials / Euler function
mentation).  expint(nu, z) Generalized exponential integral.  factorial(n) Implementation of factorial function over nonnegative integers.  factorial2(arg) The double factorial n!!, not to be confused with (n!)!  ff alias of FallingFactorial  fibonacci(n[, sym]) Fibonacci numbers / Fibonacci polynomials  floor(arg) Floor is a univariate function which returns the largest integer value not greater than its argument.  frac(arg) Represents the fractional part of x  fresnelc(z) Fresnel integral C.  fresnels(z) Fresnel integral S.  gamma(arg) The gamma function  gegenbauer(n, a, x) Gegenbauer polynomial \$C_n^{left(alpharight)}(x)\$.		
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	<pre>exp_polar(*args)</pre>	
$\begin{array}{ccc} & & & & & & \\ \text{factorial2(arg)} & & & & & \\ \text{The double factorial } \textit{n!!}, \text{ not to be confused with } \textit{(n!)!} \\ \text{ff} & & & & \\ \text{alias of FallingFactorial} \\ \text{fibonacci(n[, sym])} & & & \\ \text{Fibonacci numbers / Fibonacci polynomials} \\ \text{floor(arg)} & & & \\ \text{Floor is a univariate function which returns the largest integer value not greater than its argument.} \\ \text{frac(arg)} & & & \\ \text{Represents the fractional part of } x \\ \text{fresnelc(z)} & & \\ \text{Fresnel integral C.} \\ \text{fresnels(z)} & & \\ \text{Fresnel integral S.} \\ \text{gamma(arg)} & & \\ \text{The gamma function} \\ \text{gegenbauer(n, a, x)} & & \\ \text{Gegenbauer polynomial $C_n^{\{\text{left(alpharight)}\}(x)$\$.} \\ \end{array}$		
$\begin{array}{ll} \mbox{factorial 2(arg)} & \mbox{The double factorial $n!!$, not to be confused with $(n!)!$} \\ \mbox{ff} & \mbox{alias of FallingFactorial} \\ \mbox{fibonacci(n[, sym])} & \mbox{Fibonacci numbers / Fibonacci polynomials} \\ \mbox{floor(arg)} & \mbox{Fibonacci numbers / Indicate function which returns the largest integer value not greater than its argument.} \\ \mbox{frac(arg)} & \mbox{Represents the fractional part of $x$} \\ \mbox{fresnelc(z)} & \mbox{Fresnel integral $C$.} \\ \mbox{fresnels(z)} & \mbox{Fresnel integral $S$.} \\ \mbox{gamma(arg)} & \mbox{The gamma function} \\ \mbox{gegenbauer(n, a, x)} & \mbox{Gegenbauer polynomial $C_n^{\{left(alpharight)\}(x)$}.} \\  \end{array}$	factorial(n)	<u> </u>
$\begin{array}{lll} \mbox{fibonacci} (n[, sym]) & \mbox{Fibonacci} numbers / \mbox{Fibonacci} polynomials \\ \mbox{floor} (arg) & \mbox{Floor} is a univariate function which returns the largest integer value not greater than its argument. \\ \mbox{frac} (arg) & \mbox{Represents the fractional part of } x \\ \mbox{fresnel} c(z) & \mbox{Fresnel integral } C. \\ \mbox{fresnel} s(z) & \mbox{Fresnel integral } S. \\ \mbox{gamma} (arg) & \mbox{The gamma function} \\ \mbox{gegenbauer} (n, a, x) & \mbox{Gegenbauer polynomial } C_n^{\{left(alpharight)\}(x)\}}. \end{array}$	factorial2(arg)	=
$\begin{tabular}{lll} Floor is a univariate function which returns the largest integer value not greater than its argument. \\ frac(arg) & Represents the fractional part of x \\ fresnelc(z) & Fresnel integral C. \\ fresnels(z) & Fresnel integral S. \\ gamma(arg) & The gamma function \\ gegenbauer(n, a, x) & Gegenbauer polynomial $C_n^{\{left(alpharight)\}(x)$}. \\ \end{tabular}$	ff	alias of FallingFactorial
$\begin{array}{c} \text{integer value not greater than its argument.} \\ \text{frac(arg)} & \text{Represents the fractional part of } x \\ \text{fresnelc(z)} & \text{Fresnel integral C.} \\ \text{fresnels(z)} & \text{Fresnel integral S.} \\ \text{gamma(arg)} & \text{The gamma function} \\ \text{gegenbauer(n, a, x)} & \text{Gegenbauer polynomial $C_n^{\left( \text{left(alpharight)}\right)}(x)$.} \end{array}$	<pre>fibonacci(n[, sym])</pre>	
$\begin{array}{ll} & \text{Represents the fractional part of x} \\ & \text{fresnelc(z)} & \text{Fresnel integral C.} \\ & \text{fresnels(z)} & \text{Fresnel integral S.} \\ & \text{gamma(arg)} & \text{The gamma function} \\ & \text{gegenbauer(n, a, x)} & \text{Gegenbauer polynomial $C_n^{\left( \text{left(alpharight)}\right)(x)$}.} \end{array}$	floor(arg)	
$\begin{tabular}{ll} $\text{gamma function} \\ $\text{gegenbauer}(n,a,x)$ & $\text{Gegenbauer polynomial $C_n^{\left( \text{left}(alpharight) }\right)(x)$.} \end{tabular}$		
$\label{eq:gegenbauer} gegenbauer(n, a, x) \qquad \qquad Gegenbauer \ polynomial \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \$		
		•
	gegenpauer(n, a, x)	

Table 61 – continued from previous page

1able 61 – co	ntinued from previous page
genocchi(n[, x])	Genocchi numbers / Genocchi polynomials / Genocchi
	function
hankel1(nu, z)	Hankel function of the first kind.
hankel2(nu, z)	Hankel function of the second kind.
<pre>harmonic(n[, m])</pre>	Harmonic numbers
hermite(n, x)	hermite(n, x) gives the \$n\$th Hermite polynomial in
	$x$ , $H_n(x)$ .
hermite_prob(n, x)	hermite_prob(n, x) gives the \$n\$th probabilist's
	Hermite polynomial in $x$ , $He_n(x)$ .
hn1(nu, z)	Spherical Hankel function of the first kind.
hn2(nu, z)	Spherical Hankel function of the second kind.
hyper(ap, bq, z)	The generalized hypergeometric function is defined by a
•	series where the ratios of successive terms are a rational
	function of the summation index.
im(arg)	Returns imaginary part of expression.
jacobi(n, a, b, x)	Jacobi polynomial $P_n^{\left( \text{left}(alpha, betaright)}(x)$ .
jn(nu, z)	Spherical Bessel function of the first kind.
laguerre(n, x)	Returns the \$n\$th Laguerre polynomial in \$x\$,
,	$L_n(x)$ .
legendre(n, x)	legendre(n, x) gives the \$n\$th Legendre polynomial
	of $x$ , $P_n(x)$
lerchphi(*args)	Lerch transcendent (Lerch phi function).
li(z)	The classical logarithmic integral.
ln	alias of log
log(arg[, base])	The natural logarithm function $ln(x)$ or $log(x)$ .
loggamma(z)	The loggamma function implements the logarithm of the
(-)	gamma function (i.e., \$logGamma(x)\$).
lowergamma(a, x)	The lower incomplete gamma function.
lucas(n)	Lucas numbers
marcumq(m, a, b)	The Marcum Q-function.
mathieuc(a, q, z)	The Mathieu Cosine function $C(a,q,z)$ .
mathieucprime(a, q, z)	The derivative $C^{\text{prime}}(a,q,z)$ of the Mathieu Co-
1 (m) 1)	sine function.
mathieus(a, q, z)	The Mathieu Sine function $S(a,q,z)$ .
mathieusprime(a, q, z)	The derivative $S^{\text{prime}}(a,q,z)$ of the Mathieu Sine
	function.
meijerg(*args)	The Meijer G-function is defined by a Mellin-Barnes
e_Je_g(gs)	type integral that resembles an inverse Mellin transform.
mobius(n)	Mobius function maps natural number to {-1, 0, 1}
motzkin(n)	The nth Motzkin number is the number
multigamma(x, p)	The multivariate gamma function is a generalization of
mar erganima(x, p)	the gamma function
partition(n)	Partition numbers
periodic_argument(ar, period)	Represent the argument on a quotient of the Riemann
	surface of the logarithm.
<pre>polar_lift(arg)</pre>	Lift argument to the Riemann surface of the logarithm,
	using the standard branch.
polygamma(n, z)	The function polygamma(n, z) returns
	log(gamma(z)).diff(n + 1).
polylog(s, z)	Polylogarithm function.
<pre>preorder_traversal(node[, keys])</pre>	Do a pre-order traversal of a tree.
	continues on next page

Table 61 – continued from previous page

	p p p
<pre>primenu(n)</pre>	Calculate the number of distinct prime factors for a pos-
	itive integer n.
<pre>primeomega(n)</pre>	Calculate the number of prime factors counting multi-
	plicities for a positive integer n.
<pre>primepi(n)</pre>	Represents the prime counting function $pi(n)$ = the num-
	ber of prime numbers less than or equal to n.
<pre>principal_branch(x, period)</pre>	Represent a polar number reduced to its principal branch
	on a quotient of the Riemann surface of the logarithm.
re(arg)	Returns real part of expression.
reduced_totient(n)	Calculate the Carmichael reduced totient function
	lambda(n)
rf	alias of RisingFactorial
riemann_xi(s)	Riemann Xi function.
sec(arg)	The secant function.
sech(arg)	sech(x) is the hyperbolic secant of x.
sign(arg)	Returns the complex sign of an expression:
sin(arg)	The sine function.
sinc(arg)	Represents an unnormalized sinc function:
sinh(arg)	sinh(x) is the hyperbolic sine of x.
<pre>stieltjes(n[, a])</pre>	Represents Stieltjes constants, \$gamma_{k}\$ that occur
	in Laurent Series expansion of the Riemann zeta func-
	tion.
subfactorial(arg)	The subfactorial counts the derangements of \$n\$ items
	and is defined for non-negative integers as:
tan(arg)	The tangent function.
tanh(arg)	tanh(x) is the hyperbolic tangent of x.
totient(n)	Calculate the Euler totient function phi(n)
transpose(arg)	Linear map transposition.
<pre>tribonacci(n[, sym])</pre>	Tribonacci numbers / Tribonacci polynomials
trigamma(z)	The trigamma function is the second derivative of the
	loggamma function
uppergamma(a, z)	The upper incomplete gamma function.
vectorize(*mdargs)	Generalizes a function taking scalars to accept multidi-
	mensional arguments.
yn(nu, z)	Spherical Bessel function of the second kind.
zeta(s[, a])	Hurwitz zeta function (or Riemann zeta function).

## **Exceptions**

BasePolynomialError	Base class for polynomial related exceptions.
CoercionFailed	
ComputationFailed(func, nargs, exc)	
DomainError	
EvaluationFailed	
${\tt ExactQuotientFailed}(f,g[,dom])$	

Table 62 – continued from previous page

	continued from provided page
ExtraneousFactors	
FlagError	
GeneratorsError	
GeneratorsNeeded	
GeometryError	An exception raised by classes in the geometry module.
HeuristicGCDFailed	All exception faised by classes in the geometry module.
HomomorphismFailed	
IsomorphismFailed	
MultivariatePolynomialError	
NonSquareMatrixError	
NotAlgebraic	
NotInvertible	
NotReversible	
OperationNotSupported(poly, func)	
OptionError	
PoleError	
PolificationFailed(opt, origs, exprs[, see	q])
PolynomialDivisionFailed(f, g, domain)	
PolynomialError	
PrecisionExhausted	
RefinementFailed	
ShapeError	Wrong matrix shape
SympifyError(expr[, base_exc])	Trong man onape
UnificationFailed	
UnivariatePolynomialError	

class src.sensitivity.babymodel.TestBabyModel(\*args, \*\*kwds)

\_active

#### solve()

src.sensitivity.babymodel.connect\_regions(region\_map, hubmap, base\_trans\_cap)

given a mapping of regions to lists of hubs, and hubs to regions, creates a set of arcs between hubs such that:

- 1. the grid is connected
- 2. each region has a main hub that all other hubs in the region are connected to

#### **Parameters**

- **region\_map** (*dict*) dictionary of region names to lists of hubs
- **hubmap** (dict) dictionary of hub names to their parent region
- base\_trans\_cap (float) base transportation capacity for arcs

#### Returns

- arcs (list) list of tuples of hubs, representing start and endpoints
- **outbound** (*dict*) dictionary of hub:list of arcs originating from hub
- **inbound** (*dict*) dictionary of hub:list of arcs terminating at hub
- trans\_capacity (dict) dictionary of arcs:transportation capacity of arc

generates a random network with all parameters required to initialize a ToyBabyModel

#### **Parameters**

- num\_regions (int, optional) number of regions. Defaults to 3.
- hubs\_per\_region (int, optional) number of hubs per region. Defaults to 2.
- base\_elec\_price (float, optional) the average electricity price in all regions. Defaults to 5.0.
- base\_prod\_capacity (int, optional) the average production capacity for all hubs. Defaults to 5000.
- **demand\_fraction** (*float*, *optional*) the average fraction of capacity initial demand is set to. Defaults to 0.7.
- base\_transport\_cost (float, optional) the base transportation cost for all arcs. Defaults to 22.3.
- base\_elec\_consumption (float, optional) the electricity consumption rate for production. Defaults to 9.8.

#### Returns

- hublist (list)
- region\_list (list)
- hub\_map (dict)
- region\_map (dict)
- elec\_price (dict)
- prod\_capacity (dict)

- demand (dict)
- base\_elec\_consumption (float)
- base\_transport\_cost (float)

## src.sensitivity.faster sensitivity

faster\_sensitivity

This file contains the class SensitivityMatrix which takes in sympy objects that have been converted from pyomo. It builds the matrix of partials to be used in sensitivity analysis.

It also contains class AutoSympy which takes in pyomo models and converts the objects into sympy.

Finally, it contains class toy\_model, the sensitivity method in action and then runs toy\_model with input n=5.

# The file babymodel.py can be also use this method by importing this file instead of sensitivity\_tools.py by: from faster\_sensitivity import \*

#### **Classes**

AutoSympy(model)	This class take in pyomo models and converts the objects into sympy.
SensitivityMatrix(sympification, duals,)	This class takes in sympy objects that have been converted from pyomo.
date	date(year, month, day)> date object
<pre>datetime(year, month, day[, hour[, minute[,)</pre>	The year, month and day arguments are required.
time	<pre>time([hour[, minute[, second[, microsecond[, tz- info]]]]])&gt; a time object</pre>
timedelta	Difference between two datetime values.
timezone	Fixed offset from UTC implementation of tzinfo.
toy_model(n)	An example of the method in action that scales by the given 'n' value
tzinfo	Abstract base class for time zone info objects.

## class src.sensitivity.faster\_sensitivity.AutoSympy(model)

This class take in pyomo models and converts the objects into sympy. This is useful for problems that needs methods such as derivatives to be calculated on the equations. We use these derivatives to calculate a sensitivity matrix that estimates the changes in variables due to changes in parameters

## check\_complimentarity\_all()

## generate\_duals(constraints, duals)

Uses dual values and slack values to classify each constraint. It also stores the dual values for substitution later.

#### **Parameters**

- **constraints** (*model.component\_objects* (*pyo.Constraint*)) All of the constraint objects from the pyomo model
- duals (model.dual) All of the dual (or Suffix) objects from the pyomo model

#### Returns

\_description\_

## **Return type**

dict of lists and dicts

#### get\_constraints()

This function converts all of the constraints in the pyomo object and converts the pyomo expressions into sympy expressions.

#### Returns

Returns 2 dictionaries: equality\_constraints: keys are tuples (constraint\_name, index) and values are sympy expressions inequality\_constraints: keys are tuples (constraint\_name, index) and values are sympy expressions

#### Return type

dict, dict

## get\_objective()

This converts the pyomo objective function into a sympy function.

#### Returns

The pyomo objective function converted into sympy

## **Return type**

sympy equation

#### get\_parameters()

Convert pyomo parameters into sympy objects. This procedure creates sympy IndexedBase objects and sympy Symbol objects of similar names. The IndexedBase datatype is necessary to parse the equations, but it does not work well with derivatives. We will substitute in Symbols when the equations are all created, so they need to map to each other. To keep the columns in order through all procedures, all parameters are given a unique column number by the variable "position" This position is stored in the class dict param\_position\_map

#### Returns

Returns 4 dictionaries: parameters: keys are pyomo parameter names and values are sympy IndexedBase objects with the same name parameter\_values: keys are sympy symbols and values are the numerical values of the pyomo objects parameter\_index\_sets: keys are pyomo parameter names and values are lists of that parameters indices symbol\_map: keys are pyomo parameters with an index and values are sympy symbols with a similarly styled name and index

#### Return type

dict, dict, dict, dict

## get\_sensitivity\_matrix(parameters\_of\_interest=None)

This function gathers all of the new sympy objects and creates a SensitivityMatrix object.

#### **Parameters**

**parameters\_of\_interest** (*dict*, *optional*) – Specified subset of the parameters if more information is known about needless parameters, by default None

#### Returns

a SensitivityMatrix object that contains the sensitivity matrix and commands to use it.

## **Return type**

SensitivityMatrix

#### get\_sets()

Convert pyomo sets into sympy indexes

#### Returns

The first dictionary has the pyomo objects' names as keys and newly created sympy indexes as values. The second dictionary has the new sympy indexes as keys and the pyomo sets' values as the dict values.

#### Return type

dict, dict

## get\_variables()

Convert pyomo variables into sympy objects. This procedure creates sympy IndexedBase objects and sympy Symbol objects of similar names. The IndexedBase datatype is necessary to parse the equations, but it does not work well with derivatives. We will substitute in Symbols when the equations are all created, so they need to map to each other. To keep the columns in order through all procedures, all parameters are given a unique column number by the variable "position" This position is stored in the class dict param\_position\_map

#### Returns

Returns 2 dictionaries: variables: keys are pyomo variable names and values are sympy IndexedBase objects with the same name variable values: keys are sympy symbols and values are the numerical values of the pyomo objects It is also worth mentioning that this adds entries to the self.symbol\_map in the same way parameters do. Symbol map entries have keys of IndexedBase objects and the values are their associated sympy Symbol

## Return type

dict, dict

## 

This class takes in sympy objects that have been converted from pyomo. It builds the matrix of partials to be used in sensitivity analysis.

#### generate\_matrix()

This creates all of the matrices that will be combined into the U and S matrices.

#### Returns

Returns 2 dictionaries. The first is the dictionary of matrix components with their names as keys. The second dictionary is a map from the symbols to their values.

#### Return type

dict, dict

#### get\_partial(x, a)

Retrieve the value of a particular partial derivative. The value retrieved will be dx/da.

#### **Parameters**

- **x** (sp. Symbol) The symbol for the variable that you wish to know the change effect
- a (sp. Symbol) The symbol for the parameter that you wish to change to cause an effect on a variable

#### Returns

The value of the partial derivative dx/da

## Return type

float

## get\_partials\_matrix()

Calculate the matrix of all partials as  $U^{(-1)} * S$  Thus far, this is found to run the fastest when  $U^{(-1)}$  and S are numpy arrays

#### Returns

Full partials matrix

## **Return type**

np.ndarray

#### get\_sensitivity\_range(x, a, percent)

The estimated values for "x" if the parameter "a" changes by percent% (as number 0% to 100%). It will return values for an increase and decrease of the percent given.

#### **Parameters**

- **x** (sp. Symbol) The symbol for the variable that you wish to know the change effect
- a (sp. Symbol) The symbol for the parameter that you wish to change to cause an effect on a variable
- **percent** (*float*) A number 0-100 for the percent change in "a"

#### Returns

Returns the estimated value for "x" if "a" is increased by percent% and decreased by percent%

## Return type

float, float

#### invert\_U()

Calculates the inverse of the U matrix. The fastest method found for this so far has been to convert to numpy and use its inverse function

#### Returns

Calculated matrix for the inverse of U

## Return type

np.ndarray

#### matrix\_assembly(components, subs dict)

Combines matrix components to create U and S matrices from the literature.

#### **Parameters**

- components (dict) Dictionary of all precalculated matrix components
- **subs\_dict** (*dict*) Dictionary that maps symbols to their values

#### Returns

Returns the U and S matrices respectively with all symbols replaced by corresponding values

## Return type

sp.Matrix, sp.Matrix

#### matrix\_sub(M, subs)

A function that substitutes values into a matrix. This is the same result as sp.Matrix().subs(subs). This speeds up runtime by only attempting to substitute values into symbols that actually exist in each cell.

#### **Parameters**

- M (sp.Matrix) The matrix to have symbols substituted for values
- **subs** (*dict*) Dictionary of values for sympy symbols

#### Returns

The original matrix with all given values substituted into their symbols

#### Return type

sp.Matrix

## new\_jacobian(f, values, map)

A function that returns the same result as Matrix.jacobian(values). This speeds up runtime by only taking derivatives of symbols that exist. The original function takes the derivative wrt everything in values.

## **Parameters**

- **f** (*sp.Matrix*) Matrix of equations
- values (list) List of symbols that the function will take derivatives with respect to
- map (dict) Dictionary of column locations for each symbol

#### Returns

Returns jacobian of given f matrix

## Return type

sp.Matrix

class src.sensitivity.faster\_sensitivity.toy\_model(n)

An example of the method in action that scales by the given 'n' value

create\_model()

## src.sensitivity.sensitivity\_tools

Sensitivity Tools This file contains the AutoSympy, SensitivityMatrix, CoordMap, and DifferentialMapping classes. These classes serve as the data structures and containers of methods to get from a Pyomo ConcreteModel to easily accessible sensitivities.

The flow goes:

Pyomo model -> AutoSympy -> SensitivityMatrix -> DifferentialMapping

sensitivities and sensitivity-based calculations can be done through the Differential Mapping object.

## **Functions**

E1(z)	Classical case of the generalized exponential integral.
Eijk(*args, **kwargs)	Represent the Levi-Civita symbol.
GramSchmidt(vlist[, orthonormal])	Apply the Gram-Schmidt process to a set of vectors.
LC(f, *gens, **args)	Return the leading coefficient of f.
LM(f, *gens, **args)	Return the leading monomial of f.
LT(f, *gens, **args)	Return the leading term of f.
N(x[, n])	Calls x.evalf(n, **options).
POSform(variables, minterms[, dontcares])	The POSform function uses simplified_pairs and a redundant-group eliminating algorithm to convert the list of all input combinations that generate '1' (the minterms) into the smallest product-of-sums form.
SOPform(variables, minterms[, dontcares])	The SOPform function uses simplified_pairs and a redundant group- eliminating algorithm to convert the list of all input combos that generate '1' (the minterms) into the smallest sum-of-products form.
Ynm_c(n, m, theta, phi)	Conjugate spherical harmonics defined as
abundance(n)	Returns the difference between the sum of the positive proper divisors of a number and the number.
apart(f[, x, full])	Compute partial fraction decomposition of a rational function.
<pre>apart_list(f[, x, dummies])</pre>	Compute partial fraction decomposition of a rational function and return the result in structured form.

continues on next page

Table 64 – continued from previous page

lable 64 – contin	ued from previous page
<pre>apply_finite_diff(order, x_list, y_list[, x0])</pre>	Calculates the finite difference approximation of the derivative of requested order at <b>x0</b> from points provided in <b>x_list</b> and <b>y_list</b> .
approximants(l[, X, simplify])	Return a generator for consecutive Pade approximants for a series.
are_similar(e1, e2)	Are two geometrical entities similar.
arity(cls)	Return the arity of the function if it is known, else None.
ask(proposition[, assumptions, context])	Function to evaluate the proposition with assumptions.
<pre>assemble_partfrac_list(partial_list)</pre>	Reassemble a full partial fraction decomposition from a structured result obtained by the function apart_list.
assuming(*assumptions)	Context manager for assumptions.
banded(*args, **kwargs)	Returns a SparseMatrix from the given dictionary describing the diagonals of the matrix.
besselsimp(expr)	Simplify bessel-type functions.
binomial_coefficients(n)	Return a dictionary containing pairs $(k1, k2)$ : $C_k n$
	where $C_k n$ are binomial coefficients and $n = k1 + k2$ .
<pre>binomial_coefficients_list(n)</pre>	Return a list of binomial coefficients as rows of the Pascal's triangle.
block_collapse(expr)	Evaluates a block matrix expression
<pre>blockcut(expr, rowsizes, colsizes)</pre>	Cut a matrix expression into Blocks
bool_map(bool1, bool2)	Return the simplified version of <i>bool1</i> , and the mapping of variables that makes the two expressions <i>bool1</i> and <i>bool2</i> represent the same logical behaviour for some correspondence between the variables of each.
bottom_up(rv, F[, atoms, nonbasic])	Apply F to all expressions in an expression tree from the bottom up.
bspline_basis(d, knots, n, x)	The \$n\$-th B-spline at \$x\$ of degree \$d\$ with knots.
<pre>bspline_basis_set(d, knots, x)</pre>	Return the len(knots)-d-1 B-splines at $x$ of degree $d$ with $knots$ .
<pre>cacheit(func)</pre>	
<pre>cancel(f, *gens[, _signsimp])</pre>	Cancel common factors in a rational function f.
capture(func)	Return the printed output of func().
<pre>casoratian(seqs, n[, zero])</pre>	Given linear difference operator L of order 'k' and homogeneous equation $Ly = 0$ we want to compute kernel of L, which is a set of 'k' sequences: $a(n)$ , $b(n)$ ,.
cbrt(arg[, evaluate])	Returns the principal cube root.
<pre>ccode(expr[, assign_to, standard])</pre>	Converts an expr to a string of c code
centroid(*args)	Find the centroid (center of mass) of the collection containing only Points, Segments or Polygons.
<pre>chebyshevt_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the first kind $T_n(x)$ .
<pre>chebyshevu_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the second kind $U_n(x)$ .
<pre>check_assumptions(expr[, against])</pre>	Checks whether assumptions of expr match the T/F assumptions given (or possessed by against).
<pre>checkodesol(ode, sol[, func, order,])</pre>	Substitutes sol into ode and checks that the result is 0.
<pre>checkpdesol(pde, sol[, func, solve_for_func])</pre>	Checks if the given solution satisfies the partial differential equation.
checksol(f, symbol[, sol])	Checks whether sol is a solution of equation $f == 0$ .
<pre>classify_ode(eq[, func, dict, ics, prep,])</pre>	Returns a tuple of possible dsolve() classifications for an ODE.
	continues on next nage

Table 64 – continued from previous page

classify_pde(eq[, func, dict, prep])  PDE.  closest_points(*args)  Return the subset of points from a set of points that were the closest to each other in the 2D plane.  cofactors(f, g, *gens, **args)  collect(expr, syms, func, evaluate, exact,))  collect(expr, syms, func, evaluate, exact,))  collect const(expr, *vars[, Numbers])  Compute GCD and cofactors of f and g.  Collect additive terms of an expression.  composite(pr)  combisimp(expr)  composite(pr)  composite(pr)  composite(pr)  composite(f, g, *gens, **args)  composite(pr)	lable 64 – continue	ed from previous page
the closest to each other in the 2D plane.  cofactors(f, g, *gens, **args)  collect(expr, syms , func, evaluate, exact, )  collect_const(expr, *vars[, Numbers])  collect_const(expr, *vars[, Numbers])  combsimp(expr)  comp(z , z2 , tol )  comp(z , z2 , tol )  composite(nth)  composite(nth)  composite(nth)  composite(nth)  composite(nth)  construct_domain(obj, **args)  construct_domain(obj, **args)  construct_domain(obj, **args)  continued_fraction(a)  continued_fraction(a)  continued_fraction_convergents(cf)  continued_fraction_ieraction(a)  continued_fraction_periodic(p, q , d, s])  continued_fraction_periodic(p, q , d, s])  continued_fraction_reduce(cf)  continued_fraction_convergents  continued_fraction_periodic(p, q , d, s])  continued_fraction_convergents  convex_hull(*args , polygon )  continued_fraction_convergents  convex_hull(*args , polygon )  continued_fraction_fraction_fraction  convex_hull(*args , polygon )  convex_hull(*args , polyg	<pre>classify_pde(eq[, func, dict, prep])</pre>	
collect(expr, syms[, func, evaluate, exact,])  collect_const(expr, *vars[, Numbers])  A non-greedy collection of terms with similar number coefficients in an Add expr.  Simplify combinatorial expressions.  Composite(nt)  Return a bool indicating whether the error between z1 and z2 is SleS to1.  Composite(nth)  Return the number of positive composite numbers, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  Compositepi(n)  Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args)  Construct_atomain(obj, **args)  Construct_atomain(obj, **args)  Construct_atomain(obj, **args)  Continued_fraction(a)  Return the number of positive composite numbers less than or equal to n.  Construct_atomain(obj, **args)  Construct_an minimal domain for a list of expressions.  Continued_fraction(a)  Return the continued fraction representation of a Rational or quadratic irrational.  continued_fraction_convergents(cf)  Return an iterator over the convergents of a continued fraction expansion of a siterator.  continued_fraction_periodic(p, q[, d, s])  Find the periodic continued fraction expansion of a quadratic irrational.  continued_fraction_reduce(cf)  Reduce a continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convex_hull(*args[, inf, sup])  Return a representation (integer or expression) of the operations in exp.  count_roots(f[, inf, sup])  Return the number of positive composite equal to n.  cxxcode(expr[, assign_to, standard])  cv+equivalent of ccode().  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the gives the length of the iterated explee (lambda) and the gives the length of the iterated explee (lambda) and the gives the length of the iterated explee (lambda) and th	<pre>closest_points(*args)</pre>	the closest to each other in the 2D plane.
collect_const(expr, *vars[, Numbers])  combsimp(expr)  compose(f, g, *gens, **args)  composite(nth)  composite	cofactors(f, g, *gens, **args)	Compute GCD and cofactors of f and g.
combisimp(expr) Simplify combinatorial expressions.  compose(f, g, *gens, **args) Composite(nth) Return a bool indicating whether the error between zl and zl is Sles tol.  composite(nth) Return the nth composite number, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  compositepi(n) Return the nth composite number, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  compositepi(n) Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args) Construct_a minimal domain for a list of expressions.  content(f, *gens, **args) Continued_fraction(a) Return the continued fraction representation of a Rational or quadratic irrational.  continued_fraction_convergents(cf) Return an iterator over the convergents of a continued fraction in equal fraction expansion of a quadratic irrational.  continued_fraction_periodic(p, q[, d, s]) Find the periodic continued fraction expansion of a quadratic irrational.  continued_fraction_reduce(cf) Reduce a continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon]) The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,]) Performs convolution by determining the type of desired convolution using hints.  continued_fraction_fraction_fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon]) Return a representation (integer or expression) of the operations in exp.  count_roots(f[, inf, sup]) Return a representation (integer or expression) of the operations in exp.  count_roots(f[, inf, sup]) Return the number of roots of f in [inf, sup] interval.  covering_product(a, b) Returns the covering product of given sequences.  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard]) C++ equivalent of ccode().  cycle_length(f, x0[, nmax, values]) Generates eyclotomic polynomial of order n in x.  Compute the the terms of the sequence will be returned	<pre>collect(expr, syms[, func, evaluate, exact,])</pre>	Collect additive terms of an expression.
compose(f, g, *gens, **args)  composite(nth)  Return a bool indicating whether the error between zl and z2 is \$le\$ tol.  Composite(nth)  Return the nth composite number, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  compositepi(n)  Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args)  content(f, *gens, **args)  continued_fraction(a)  Return the continued faction representation of a Rational or quadratic irrational.  Continued_fraction_convergents(cf)  continued_fraction_iterator(x)  continued_fraction_periodic(p, q[, d, s])  continued_fraction_reduce(cf)  continued_fraction_reduce(cf)  continued_fraction_reduce(cf)  continued_fraction_reduce(cf)  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  convolution(a, b[, cycle, dps, prime,])  convolution(a, b[, cycle, dps, prime,])  Return a representation (integer or expression) of the operations in expr.  count_pos(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots([i, inf, sup])  count_roots([i, inf, sup])  count_roots([i, inf, sup])  covering_product(a, b)  cse(exprs[, symbols, optimizations,])  Return the number of roots of f in [inf, sup] interval.  cvering_product(a, b)  Cse(exprs[, assign_to, standard])  Cycle_length(f, x0[, nmax, values))  Generates cyclotomic polynomial of order n in x.  decompose(f, *gens, **args)  decompose(f, *gens, **args)  Compute the unitary had be used for sorting.		coefficients in an Add expr.
compose(f, g, *gens, **args) Compute functional composition f(g).  Return the nth composite number, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  Compositepi(n) Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args) Content(f, *gens, **args) Continued_fraction(a) Return the continued fraction representation of a Rational or qual and the continued fraction representation of a Rational or quadratic irrational.  Continued_fraction_iterator(x) Continued_fraction_iterator(x) Continued_fraction_periodic(p, q[, d, s]) Continued_fraction_reduce(cf) Return ontinued fraction expansion of x as iterator.  Continued_fraction_reduce(cf) Reduce a continued fraction to a rational or quadratic irrational.  Convox_hull(*args[, polygon]) The convex hull surrounding the Points contained in the list of entities.  Convolution(a, b[, cycle, dps, prime,]) Performs convolution by determining the type of desired convolution using hints.  Cosine_transform(f, x, k, **hints) Compute the unitary, ordinary-frequency cosine transform of , defined as  Count_pos(cxpr[, visual]) Return a representation (integer or expression) of the operations in expr.  count_roots([], inf, sup]) Covering_product(a, b) Returns the covering product of given sequences.  Perform common subexpression climination on an expression.  Cxxcode(expr[, assign_to, standard]) C++ equivalent of ccode().  Cycle_length(f, x0[, nmax, values])  Generates cyclotomic polynomial of order n in x.  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = [ 1 o [ 2 0		
Return the nth composite number, with the composite numbers indexed as composite(1) = 4, composite(2) = 6, etc  compositepi(n)  Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args)  Construct a minimal domain for a list of expressions.  Content(f, *gens, **args)  Compute GCD of coefficients of f.  Continued_fraction(a)  Return the continued fraction representation of a Rational or quadratic irrational.  continued_fraction_convergents(cf)  Return an iterator over the convergents of a continued fraction expansion of x as iterator.  continued_fraction_periodic(p, ql, d, sl)  continued_fraction_periodic(p, ql, d, sl)  Continued_fraction_periodic(p, ql, d, sl)  Find the periodic continued fraction expansion of x as iterator.  continued_fraction_periodic(p, ql, d, sl)  Reduce a continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  count_roots(f[, inf, sup])  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Returns the covering product of given sequences.  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  Cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  decompose(f, symbol), order n in x.  Compute General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1[f_2(f_n)).  Compute functional decomposit		and z2 is \$le\$ to1.
numbers indexed as composite(1) = 4, composite(2) = 6, etc  compositepi(n)  Return the number of positive composite numbers less than or equal to n.  Construct_domain(obj, **args)  content(f, *gens, **args)  continued_fraction(a)  Return the continued fraction representation of a Rational or quadratic irrational.  continued_fraction_convergents(cf)  Return an iterator over the convergents of a continued fraction (ef).  continued_fraction_iterator(x)  continued_fraction_periodic(p, q[, d, s])  continued_fraction_periodic(p, q[, d, s])  continued_fraction_reduce(cf)  Return continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  covering_product(a, b)  Return the number of roots of f in [inf, sup] interval.  cxcode(expr[, assign_to, standard])  cxcode(expr[, assign_to, standard])  cycle_length(f, x0[, nmax, values])  Generates cyclotomic polyn(n[, x, polys])  decompogen(f, symbol)  Generates cyclotomic polynomial of order n in x.  Compute functional decomposition f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 lo f_2 o f_n = f_1[f_2(f_n]).  Copute functional decomposition f.  Return a key that can be used for sorting.		
than or equal to n.  Construct_domain(obj, **args) Construct a minimal domain for a list of expressions.  Content(f, *gens, **args) Compute GCD of coefficients of f.  Return the continued fraction representation of a Rational or quadratic irrational.  Continued_fraction_convergents(cf) Return an iterator over the convergents of a continued fraction (cf).  Return continued fraction expansion of x as iterator.  Continued_fraction_terator(x) Return continued fraction expansion of x as iterator.  Continued_fraction_periodic(p, q[, d, s]) Return continued fraction to a rational or quadratic irrational.  Continued_fraction_reduce(cf) Reduce a continued fraction to a rational or quadratic irrational.  Convex_hull(*args[, polygon]) The convex hull surrounding the Points contained in the list of entities.  Convolution(a, b[, cycle, dps, prime,]) Performs convolution by determining the type of desired convolution using hints.  Cosine_transform(f, x, k, **hints) Compute the unitary, ordinary-frequency cosine transform of f, defined as  Count_ops(expr[, visual]) Return a representation (integer or expression) of the operations in expr.  Count_roots(f[, inf, sup]) Return the number of roots of f in [inf, sup] interval.  Covering_product(a, b) Returns the covering product of given sequences.  Perform common subexpression elimination on an expression.  Cxxcode(expr[, assign_to, standard]) Cycle_length(f, x0[, nmax, values]) For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  Cyclotomic_poly(n[, x, polys]) Generates cyclotomic polynomial of order n in x.  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = [1 o [2 o f_n = [1](2( f_n))).  decompose(f, *gens, **args) Compute functional decomposition of f.	composite(nth)	numbers indexed as $composite(1) = 4$ , $composite(2) = 6$ ,
construct_domain(obj, **args)	<pre>compositepi(n)</pre>	• •
content(f, *gens, **args)  continued_fraction(a)  Return the continued fraction representation of a Rational or quadratic irrational.  continued_fraction_convergents(cf)  Return an iterator over the convergents of a continued fraction (cf).  Return continued fraction expansion of x as iterator.  continued_fraction_periodic(p, q[, d, s])  continued_fraction_periodic(p, q[, d, s])  continued_fraction_reduce(cf)  Reduce a continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,])  C++ equivalent of ccode().  cycle_length(f, x0[, nmax, values])  C++ equivalent of ccode().  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  decompogen(f, symbol)  Computes General functional decomposition of f. Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 of_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f. Given an expression becomes and the content of the	<pre>construct_domain(obj, **args)</pre>	•
continued_fraction(a)  Return the continued fraction representation of a Rational or quadratic irrational.  Return an iterator over the convergents of a continued fraction (cf).  Return continued fraction expansion of x as iterator.  Continued_fraction_periodic(p, q[, d, s])  Continued_fraction_reduce(cf)  Reduce a continued fraction to a rational or quadratic irrational.  Convex_hull(*args[, polygon])  Convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  Convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  Cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  Count_pos(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  Count_roots(f[, inf, sup])  Covering_product(a, b)  Returns the covering product of given sequences.  Cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  Cxxcode(expr[, assign_to, standard])  C++ equivalent of ccode().  Cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  Cyclotomic_poly(n[, x, polys])  decompogen(f, symbol)  Generates cyclotomic polynomial of order n in x.  Compute functional decomposition of f. Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 of_2 o f = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 of_2 o f = f_1(f_2( f_n)).		
continued_fraction_iterator(x)  continued_fraction_periodic(p, q[, d, s])  continued_fraction_periodic(p, q[, d, s])  continued_fraction_reduce(cf)  continued_fraction_reduce(cf)  convex_hull(*args[, polygon])  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_pos(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  covering_product(a, b)  Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompogen(f, symbol)  Computes General functional decomposition of f. Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 of_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.		Return the continued fraction representation of a Ratio-
continued_fraction_periodic(p, q[, d, s])  continued_fraction_reduce(cf)  continued_fraction_reduce(cf)  convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  Cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  covering_product(a, b)  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Return the number of roots of given sequences.  cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  C++ equivalent of ccode().  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  decompogen(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	<pre>continued_fraction_convergents(cf)</pre>	
quadratic irrational.  continued_fraction_reduce(cf) Reduce a continued fraction to a rational or quadratic irrational.  convex_hull(*args[, polygon]) The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,]) Performs convolution by determining the type of desired convolution using hints.  cosine_transform(f, x, k, **hints) Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual]) Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup]) Return the number of roots of f in [inf, sup] interval.  covering_product(a, b) Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,]) Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard]) C++ equivalent of ccode().  cycle_length(f, x0[, nmax, values]) For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys]) Generates cyclotomic polynomial of order n in x.  decompose(f, symbol) Computes General functional decomposition of f. Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1[(f_2( f_n)).  decompose(f, *gens, **args) Compute functional decomposition of f. Return a key that can be used for sorting.	<pre>continued_fraction_iterator(x)</pre>	Return continued fraction expansion of x as iterator.
convex_hull(*args[, polygon])  The convex hull surrounding the Points contained in the list of entities.  convolution(a, b[, cycle, dps, prime,])  Performs convolution by determining the type of desired convolution using hints.  cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  C++ equivalent of ccode().  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompose(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 of_2 o f_n = f_1(f_2( f_n)), decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	<pre>continued_fraction_periodic(p, q[, d, s])</pre>	
list of entities.	continued_fraction_reduce(cf)	•
convolution using hints.  cosine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency cosine transform of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompogen(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	<pre>convex_hull(*args[, polygon])</pre>	
form of f, defined as  count_ops(expr[, visual])  Return a representation (integer or expression) of the operations in expr.  count_roots(f[, inf, sup])  Return the number of roots of f in [inf, sup] interval.  covering_product(a, b)  Returns the covering product of given sequences.  cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompogen(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	convolution(a, b[, cycle, dps, prime,])	
count_roots(f[, inf, sup])  covering_product(a, b)  Return the number of roots of f in [inf, sup] interval.  Returns the covering product of given sequences.  Cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  Cxxcode(expr[, assign_to, standard])  Cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompogen(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	<pre>cosine_transform(f, x, k, **hints)</pre>	
covering_product(a, b)  Returns the covering product of given sequences.  Perform common subexpression elimination on an expression.  Cxxcode(expr[, assign_to, standard])  C++ equivalent of ccode().  Cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.	<pre>count_ops(expr[, visual])</pre>	
cse(exprs[, symbols, optimizations,])  Perform common subexpression elimination on an expression.  cxxcode(expr[, assign_to, standard])  cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  decompogen(f, symbol)  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  default_sort_key(item[, order])  Return a key that can be used for sorting.	_	
pression.  Cxxcode(expr[, assign_to, standard])  Cycle_length(f, x0[, nmax, values])  For a given iterated sequence, return a generator that gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.  Cyclotomic_poly(n[, x, polys])  Generates cyclotomic polynomial of order n in x.  Computes General functional decomposition of f.  Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)).  decompose(f, *gens, **args)  Compute functional decomposition of f.  Return a key that can be used for sorting.		
		pression.
$\begin{array}{c} \text{gives the length of the iterated cycle (lambda) and the} \\ \text{length of terms before the cycle begins (mu); if values} \\ \text{is True then the terms of the sequence will be returned} \\ \text{instead.} \\ \text{cyclotomic_poly(n[, x, polys])} \\ \text{decompogen(f, symbol)} \\ \text{Computes General functional decomposition of } f. \\ \text{Given an expression } f, \text{ returns a list } [f_1, f_2, \ldots, f_n], \text{ where:: } f = f_1 \text{ o } f_2 \text{ o } \ldots  f_n = f_1  (f_2  \ldots  f_n)). \\ \text{decompose(f, *gens, **args)} \\ \text{default\_sort\_key(item[, order])} \\ \text{Return a key that can be used for sorting.} \\ \end{array}$		
$\begin{tabular}{lll} decompogen(f, symbol) & Computes General functional decomposition of f. \\ Given an expression f, returns a list [f_1, f_2,, f_n], where:: f = f_1 o f_2 o f_n = f_1(f_2( f_n)). \\ decompose(f, *gens, **args) & Compute functional decomposition of f. \\ default\_sort\_key(item[, order]) & Return a key that can be used for sorting. \\ \end{tabular}$		gives the length of the iterated cycle (lambda) and the length of terms before the cycle begins (mu); if values is True then the terms of the sequence will be returned instead.
default_sort_key(item[, order]) Return a key that can be used for sorting.		Given an expression f, returns a list $[f_1, f_2,, f_n]$ , where:: $f = f_1 \circ f_2 \circ \circ f_n = f_1(f_2( \circ f_n))$ .
•		
	derault_sort_key(item[, order])	Return a key that can be used for sorting.

Table 64 – continued from previous page

Table 64 – continue	ed from previous page
deg(r)	Return the degree value for the given radians (pi = 180 degrees).
degree(f[, gen])	Return the degree of f in the given variable.
<pre>degree_list(f, *gens, **args)</pre>	Return a list of degrees of f in all variables.
denom(expr)	
<pre>derive_by_array(expr, dx)</pre>	Derivative by arrays.
det(matexpr)	Matrix Determinant
<pre>det_quick(M[, method])</pre>	Return det(M) assuming that either there are lots of zeros or the size of the matrix is small.
diag(*values[, strict, unpack])	Returns a matrix with the provided values placed on the diagonal.
diagonalize_vector(vector)	
<pre>dict_merge(*dicts)</pre>	Merge dictionaries into a single dictionary.
<pre>diff(f, *symbols, **kwargs)</pre>	Differentiate f with respect to symbols.
<pre>difference_delta(expr[, n, step])</pre>	Difference Operator.
<pre>differentiate_finite(expr, *symbols[,])</pre>	Differentiate expr and replace Derivatives with finite differences.
<pre>diophantine(eq[, param, syms, permute])</pre>	Simplify the solution procedure of diophantine equation eq by converting it into a product of terms which should equal zero.
<pre>discrete_log(n, a, b[, order, prime_order])</pre>	Compute the discrete logarithm of a to the base b modulo n.
discriminant(f, *gens, **args)	Compute discriminant of f.
div(f, g, *gens, **args)	Compute polynomial division of f and g.
<pre>divisor_count(n[, modulus, proper])</pre>	Return the number of divisors of n.
divisors(n[, generator, proper])	Return all divisors of n sorted from 1n by default.
<pre>dotprint(expr[, styles, atom, maxdepth,]) dsolve(eq[, func, hint, simplify, ics, xi,])</pre>	DOT description of a SymPy expression tree Solves any (supported) kind of ordinary differential equation and system of ordinary differential equations.
<pre>egyptian_fraction(r[, algorithm])</pre>	Return the list of denominators of an Egyptian fraction expansion $[1]$ of the said rational $r$ .
<pre>epath(path[, expr, func, args, kwargs])</pre>	Manipulate parts of an expression selected by a path.
<pre>euler_equations(L[, funcs, vars])</pre>	Find the Euler-Lagrange equations [1] for a given Lagrangian.
evaluate(x)	Control automatic evaluation
expand(e[, deep, modulus, power_base,])	Expand an expression using methods given as hints.
<pre>expand_complex(expr[, deep])</pre>	Wrapper around expand that only uses the complex hint.
<pre>expand_func(expr[, deep])</pre>	Wrapper around expand that only uses the func hint.
expand_log(expr[, deep, force, factor])	Wrapper around expand that only uses the log hint.
<pre>expand_mul(expr[, deep])</pre>	Wrapper around expand that only uses the mul hint.
<pre>expand_multinomial(expr[, deep])</pre>	Wrapper around expand that only uses the multinomial hint.
<pre>expand_power_base(expr[, deep, force])</pre>	Wrapper around expand that only uses the power_base hint.
<pre>expand_power_exp(expr[, deep])</pre>	Wrapper around expand that only uses the power_exp hint.
<pre>expand_trig(expr[, deep])</pre>	Wrapper around expand that only uses the trig hint.
exptrigsimp(expr)	Simplifies exponential / trigonometric / hyperbolic functions.
exquo(f, g, *gens, **args)	Compute polynomial exact quotient of f and g.
	continues on next page

Table 64 – continued from previous page

Table 64 – continue	ed from previous page
eye(*args, **kwargs)	Create square identity matrix n x n
<pre>factor(f, *gens[, deep])</pre>	Compute the factorization of expression, f, into irreducibles.
factor list(f *gaps **args)	
factor_list(f, *gens, **args)	Compute a list of irreducible factors of f.
factor_nc(expr)	Return the factored form of expr while handling non- commutative expressions.
<pre>factor_terms(expr[, radical, clear,])</pre>	Remove common factors from terms in all arguments without changing the underlying structure of the expr.
<pre>factorint(n[, limit, use_trial, use_rho,])</pre>	Given a positive integer n, factorint(n) returns a dict containing the prime factors of n as keys and their respective multiplicities as values.
<pre>factorrat(rat[, limit, use_trial, use_rho,])</pre>	Given a Rational r, factorrat(r) returns a dict containing the prime factors of r as keys and their respective multiplicities as values.
<pre>failing_assumptions(expr, **assumptions)</pre>	Return a dictionary containing assumptions with values not matching those of the passed assumptions.
<pre>farthest_points(*args)</pre>	Return the subset of points from a set of points that were the furthest apart from each other in the 2D plane.
fcode(expr[, assign_to])	Converts an expr to a string of fortran code
fft(seq[, dps])	Performs the Discrete Fourier Transform ( <b>DFT</b> ) in the complex domain.
<pre>field(symbols, domain[, order])</pre>	Construct new rational function field returning (field, x1,, xn).
<pre>field_isomorphism(a, b, *[, fast])</pre>	Find an embedding of one number field into another.
filldedent(s[, w])	Strips leading and trailing empty lines from a copy of s, then dedents, fills and returns it.
<pre>finite_diff_weights(order, x_list[, x0])</pre>	Calculates the finite difference weights for an arbitrarily spaced one-dimensional grid (x_list) for derivatives at x0 of order 0, 1,, up to order using a recursive formula.
<pre>flatten(iterable[, levels, cls])</pre>	Recursively denest iterable containers.
<pre>fourier_series(f[, limits, finite])</pre>	Computes the Fourier trigonometric series expansion.
<pre>fourier_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency Fourier transform of f, defined as
fps(f[, x, x0, dir, hyper, order, rational,])	Generates Formal Power Series of f.
fraction(expr[, exact])	Returns a pair with expression's numerator and denominator.
fu(rv[, measure])	Attempt to simplify expression by using transformation rules given in the algorithm by Fu et al.
fwht(seq)	Performs the Walsh Hadamard Transform (WHT), and uses Hadamard ordering for the sequence.
<pre>galois_group(f, *gens[, by_name, max_tries,])</pre>	Compute the Galois group for polynomials $f$ up to degree 6.
gammasimp(expr)	Simplify expressions with gamma functions.
gcd(f[, g])	Compute GCD of f and g.
gcd_list(seq, *gens, **args)	Compute GCD of a list of polynomials.
gcd_terms(terms[, isprimitive, clear, fraction])	Compute the GCD of terms and put them together.
gcdex(f, g, *gens, **args)	Extended Euclidean algorithm of f and g.
getLogger([name])	Return a logger with the specified name, creating it if
	necessary.
<pre>get_contraction_structure(expr)</pre>	Determine dummy indices of expr and describe its structure
	continues on next nage

Table 64 – continued from previous page

Table 64 – continue	d from previous page
<pre>get_indices(expr)</pre>	Determine the outer indices of expression expr
gff(f, *gens, **args)	Compute greatest factorial factorization of f.
gff_list(f, *gens, **args)	Compute a list of greatest factorial factors of f.
glsl_code(expr[, assign_to])	Converts an expr to a string of GLSL code
groebner(F, *gens, **args)	Computes the reduced Groebner basis for a set of poly-
<pre>ground_roots(f, *gens, **args)</pre>	nomials.  Compute roots of f by factorization in the ground do-
	main.
group(seq[, multiple])	Splits a sequence into a list of lists of equal, adjacent elements.
gruntz(e, z, z0[, dir])	Compute the limit of $e(z)$ at the point $z0$ using the Gruntz algorithm.
hadamard_product(*matrices)	Return the elementwise (aka Hadamard) product of matrices.
half_gcdex(f, g, *gens, **args)	Half extended Euclidean algorithm of f and g.
<pre>hankel_transform(f, r, k, nu, **hints)</pre>	Compute the Hankel transform of <i>f</i> , defined as
has_dups(seq)	Return True if there are any duplicate elements in seq.
has_variety(seq)	Return True if there are any different elements in seq.
hermite_poly(n[, x, polys])	Generates the Hermite polynomial $H_n(x)$ .
<pre>hermite_prob_poly(n[, x, polys])</pre>	Generates the probabilist's Hermite polynomial $He_n(x)$ .
hessian(f, varlist[, constraints])	Compute Hessian matrix for a function f wrt parame-
	ters in varlist which may be given as a sequence or a row/column vector.
homogeneous_order(eq, *symbols)	Returns the order $n$ if $g$ is homogeneous and None if it is not homogeneous.
horner(f, *gens, **args)	Rewrite a polynomial in Horner form.
<pre>hyperexpand(f[, allow_hyper, rewrite, place])</pre>	Expand hypergeometric functions.
hypersimilar(f, g, k)	Returns True if f and g are hyper-similar.
$ ext{hypersimp}(f, k)$	Given combinatorial term $f(k)$ simplify its consecutive term ratio i.e. $f(k+1)/f(k)$ .
idiff(eq, y, x[, n])	Return $dy/dx$ assuming that $eq == 0$ .
ifft(seq[, dps])	Performs the Discrete Fourier Transform ( <b>DFT</b> ) in the complex domain.
ifwht(seq)	Performs the Walsh Hadamard Transform (WHT), and uses Hadamard ordering for the sequence.
igcd(*args)	Computes nonnegative integer greatest common divisor.
ilcm(*args)	Computes integer least common multiple.
imageset(*args)	Return an image of the set under transformation f.
<pre>init_printing([pretty_print, order,])</pre>	Initializes pretty-printer depending on the environment.
<pre>init_session([ipython, pretty_print, order,])</pre>	Initialize an embedded IPython or Python session.
<pre>integer_log(y, x)</pre>	Returns (e, bool) where e is the largest nonnegative integer such that $ y  \ge  x^e $ and bool is True if $y = x^e$ .
<pre>integer_nthroot(y, n)</pre>	Return a tuple containing $x = floor(y^{**}(1/n))$ and a boolean indicating whether the result is exact (that is, whether $x^{**}n == y$ ).
<pre>integrate(f, var,)</pre>	
<pre>interactive_traversal(expr)</pre>	Traverse a tree asking a user which branch to choose.
interpolate(data, x)	Construct an interpolating polynomial for the data points evaluated at point x (which can be symbolic or numeric).
	continues on next page

Table 64 – continued from previous page

lable 64 – continue	ed from previous page
$\verb interpolating_poly (n,x[,X,Y]) $	Construct Lagrange interpolating polynomial for n data points.
$\verb interpolating_spline (d,x,X,Y)$	Return spline of degree <i>d</i> , passing through the given <i>X</i> and <i>Y</i> values.
<pre>intersecting_product(a, b)</pre>	Returns the intersecting product of given sequences.
<pre>intersection(*entities[, pairwise])</pre>	The intersection of a collection of GeometryEntity instances.
<pre>intervals(F[, all, eps, inf, sup, strict,])</pre>	Compute isolating intervals for roots of f.
<pre>intt(seq, prime)</pre>	Performs the Number Theoretic Transform ( <b>NTT</b> ), which specializes the Discrete Fourier Transform ( <b>DFT</b> ) over quotient ring $\mathbb{Z}/p\mathbb{Z}$ for prime $p$ instead of complex numbers $C$ .
inv_quick(M)	Return the inverse of M, assuming that either there are lots of zeros or the size of the matrix is small.
<pre>inverse_cosine_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse cosine transform of $F$ , defined as
<pre>inverse_fourier_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse Fourier transform of $F$ , defined as
<pre>inverse_hankel_transform(F, k, r, nu, **hints)</pre>	Compute the inverse Hankel transform of $F$ defined as
<pre>inverse_laplace_transform(F, s, t[, plane])</pre>	Compute the inverse Laplace transform of $F(s)$ , defined as
<pre>inverse_mellin_transform(F, s, x, strip, **hints)</pre>	Compute the inverse Mellin transform of $F(s)$ over the fundamental strip given by $strip=(a, b)$ .
<pre>inverse_mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with indices of sequence as bitmasks.
<pre>inverse_sine_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse sine transform of $F$ , defined as
<pre>invert(f, g, *gens, **args)</pre>	Invert f modulo g when possible.
is_abundant(n)	Returns True if n is an abundant number, else False.
is_amicable(m, n)	Returns True if the numbers $m$ and $n$ are "amicable", else False.
<pre>is_convex(f, *syms[, domain])</pre>	Determines the convexity of the function passed in the argument.
is_decreasing(expression[, interval, symbol])	Return whether the function is decreasing in the given interval.
<pre>is_deficient(n) is_increasing(expression[, interval, symbol])</pre>	Returns True if n is a deficient number, else False.  Return whether the function is increasing in the given interval.
<pre>is_mersenne_prime(n)</pre>	Returns True if n is a Mersenne prime, else False.
<pre>is_monotonic(expression[, interval, symbol])</pre>	Return whether the function is monotonic in the given interval.
$\verb is_nthpow_residue (a,n,m)$	Returns True if $x^*n == a \pmod{m}$ has solutions.
<pre>is_perfect(n)</pre>	Returns True if n is a perfect number, else False.
<pre>is_primitive_root(a, p)</pre>	Returns True if a is a primitive root of p.
is_quad_residue(a, p)	Returns True if a (mod p) is in the set of squares mod p, i.e a % p in set( $[i**2 \% p \text{ for i in range}(p)]$ ).
<pre>is_strictly_decreasing(expression[,])</pre>	Return whether the function is strictly decreasing in the given interval.
<pre>is_strictly_increasing(expression[,])</pre>	Return whether the function is strictly increasing in the given interval.
<pre>is_zero_dimensional(F, *gens, **args)</pre>	Checks if the ideal generated by a Groebner basis is zero- dimensional.
	continues on next nage

Table 64 – continued from previous page

Table 64 – continue	Table 64 – continued from previous page	
<pre>isolate(alg[, eps, fast])</pre>	Find a rational isolating interval for a real algebraic number.	
isprime(n)	Test if n is a prime number (True) or not (False).	
<pre>itermonomials(variables, max_degrees[,])</pre>	max_degrees and min_degrees are either both integers or both lists.	
<pre>jacobi_normalized(n, a, b, x)</pre>	Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.	
<pre>jacobi_poly(n, a, b[, x, polys])</pre>	Generates the Jacobi polynomial $P_n^{(a,b)}(x)$ .	
jacobi_symbol(m, n)	Returns the Jacobi symbol $(m/n)$ .	
jn_zeros(n, k[, method, dps])	Zeros of the spherical Bessel function of the first kind.	
jordan_cell(eigenval, n)	Create a Jordan block:	
jscode(expr[, assign_to])	Converts an expr to a string of javascript code	
julia_code(expr[, assign_to])	Converts <i>expr</i> to a string of Julia code.	
kronecker_product(*matrices)	The Kronecker product of two or more arguments.	
kroneckersimp(expr)	Simplify expressions with KroneckerDelta.	
laguerre_poly(n[, x, alpha, polys])	Generates the Laguerre polynomial $L_n^{(alpha)}(x)$ .	
lambdify(args, expr[, modules, printer,])	Convert a SymPy expression into a function that allows	
	for fast numeric evaluation.	
<pre>laplace_transform(f, t, s[, legacy_matrix])</pre>	Compute the Laplace Transform $F(s)$ of $f(t)$ ,	
lcm(f[, g])	Compute LCM of f and g.	
<pre>lcm_list(seq, *gens, **args)</pre>	Compute LCM of a list of polynomials.	
<pre>legendre_poly(n[, x, polys])</pre>	Generates the Legendre polynomial $P_n(x)$ .	
legendre_symbol(a, p)	Returns the Legendre symbol $(a/p)$ .	
limit(e, z, z0[, dir])	Computes the limit of e(z) at the point z0.	
<pre>limit_seq(expr[, n, trials])</pre>	Finds the limit of a sequence as index n tends to infinity.	
<pre>line_integrate(field, Curve, variables)</pre>	Compute the line integral.	
<pre>linear_eq_to_matrix(equations, *symbols)</pre>	Converts a given System of Equations into Matrix form.	
linsolve(system, *symbols)	Solve system of \$N\$ linear equations with \$M\$ variables; both underdetermined and overdetermined systems are supported.	
<pre>list2numpy(l[, dtype])</pre>	Converts Python list of SymPy expressions to a NumPy array.	
<pre>logcombine(expr[, force])</pre>	Takes logarithms and combines them using the following rules:	
<pre>maple_code(expr[, assign_to])</pre>	Converts expr to a string of Maple code.	
<pre>mathematica_code(expr, **settings)</pre>	Converts an expr to a string of the Wolfram Mathematica code	
<pre>matrix2numpy(m[, dtype])</pre>	Converts SymPy's matrix to a NumPy array.	
$\verb matrix_multiply_elementwise (A,B) $	Return the Hadamard product (elementwise product) of A and B	
matrix_symbols(expr)		
<pre>maximum(f, symbol[, domain])</pre>	Returns the maximum value of a function in the given domain.	
<pre>mellin_transform(f, x, s, **hints)</pre>	Compute the Mellin transform $F(s)$ of $f(x)$ ,	
memoize_property(propfunc)	Property decorator that caches the value of potentially expensive <i>propfunc</i> after the first evaluation.	
mersenne_prime_exponent(nth)	Returns the exponent i for the nth Mersenne prime (which has the form $2^{i} - 1$ ).	
<pre>minimal_polynomial(ex[, x, compose, polys,])</pre>	Computes the minimal polynomial of an algebraic element.	
minimum(f, symbol[, domain])	Returns the minimum value of a function in the given domain.	
	continues on next page	

Table 64 – continued from previous page

<pre>minpoly(ex[, x, compose, polys, domain])</pre>	This is a synonym for minimal_polynomial().
<pre>mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with
	indices of sequence as bitmasks.
mod_inverse(a, m)	Return the number $c$ such that, $a$ times $c = 1$
	$pmod\{m\}$ \$ where \$c\$ has the same sign as \$m\$.
monic(f, *gens, **args)	Divide all coefficients of f by LC(f).
<pre>multiline_latex(lhs, rhs[, terms_per_line,])</pre>	This function generates a LaTeX equation with a mul-
	tiline right-hand side in an align*, eqnarray or
	IEEEeqnarray environment.
<pre>multinomial_coefficients(m, n)</pre>	Return a dictionary containing pairs { (k1, k2,,km)
	: C_kn} where C_kn are multinomial coefficients such
	that n=k1+k2++km.
multiplicity(p, n)	Find the greatest integer m such that p**m divides n.
n_order(a, n)	Returns the order of a modulo n.
nextprime(n[, ith])	Return the ith prime greater than n.
nfloat(expr[, n, exponent, dkeys])	Make all Rationals in expr Floats except those in expo-
niiouc(expi[, ii, exponent, ukeys])	nents (unless the exponents flag is set to True) and those
	in undefined functions.
nonlinsolve(system, *symbols)	Solve system of \$N\$ nonlinear equations with \$M\$ vari-
HollTitsofve(system, 'symbols)	ables, which means both under and overdetermined sys-
	· ·
not ampty in(finest intersection *array)	tems are supported.  Finds the domain of the functions in
<pre>not_empty_in(finset_intersection, *syms)</pre>	Finds the domain of the functions in finset_intersection in which the finite_set is
	not-empty.
<pre>npartitions(n[, verbose])</pre>	Calculate the partition function P(n), i.e. the number of
. (0)	ways that n can be written as a sum of positive integers.
nroots(f[, n, maxsteps, cleanup])	Compute numerical approximations of roots of f.
<pre>nsimplify(expr[, constants, tolerance,])</pre>	Find a simple representation for a number or, if there are
	free symbols or if rational=True, then replace Floats
	with their Rational equivalents.
nsolve(*args[, dict])	Solve a nonlinear equation system numerically:
	<pre>nsolve(f, [args,] x0, modules=['mpmath'],</pre>
	**kwargs).
nth_power_roots_poly(f, n, *gens, **args)	Construct a polynomial with n-th powers of roots of f.
nthroot_mod(a, n, p[, all_roots])	Find the solutions to $x^*n = a \mod p$ .
ntt(seq, prime)	Performs the Number Theoretic Transform (NTT),
	which specializes the Discrete Fourier Transform ( <b>DFT</b> )
	over quotient ring $\mathbb{Z}/p\mathbb{Z}$ for prime $p$ instead of complex
	numbers C.
<pre>numbered_symbols([prefix, cls, start, exclude])</pre>	Generate an infinite stream of Symbols consisting of a
	prefix and increasing subscripts provided that they do not
	occur in exclude.
numer(expr)	
octave_code(expr[, assign_to])	Converts <i>expr</i> to a string of Octave (or Matlab) code.
ode_order(expr, func)	Returns the order of a given differential equation with
	respect to func.
ones(*args, **kwargs)	Returns a matrix of ones with rows rows and cols
	columns; if cols is omitted a square matrix will be re-
	turned.
	continues on next nage

Table 64 – continued from previous page

Table 64 – continued from previous page	
ordered(seq[, keys, default, warn])	Return an iterator of the seq where keys are used to break ties in a conservative fashion: if, after applying a key, there are no ties then no other keys will be computed.
<pre>pager_print(expr, **settings)</pre>	Prints expr using the pager, in pretty form.
<pre>parallel_poly_from_expr(exprs, *gens, **args)</pre>	Construct polynomials from expressions.
<pre>parse_expr(s[, local_dict, transformations,])</pre>	Converts the string s to a SymPy expression, in local_dict.
<pre>pde_separate(eq, fun, sep[, strategy])</pre>	Separate variables in partial differential equation either by additive or multiplicative separation approach.
pde_separate_add(eq, fun, sep)	Helper function for searching additive separable solutions.
<pre>pde_separate_mul(eq, fun, sep)</pre>	Helper function for searching multiplicative separable solutions.
pdiv(f, g, *gens, **args)	Compute polynomial pseudo-division of f and g.
<pre>pdsolve(eq[, func, hint, dict, solvefun])</pre>	Solves any (supported) kind of partial differential equation.
per(matexpr)	Matrix Permanent
<pre>perfect_power(n[, candidates, big, factor])</pre>	Return (b, e) such that $n == b^*e$ if n is a unique perfect power with $e > 1$ , else False (e.g. 1 is not a perfect power).
<pre>periodicity(f, symbol[, check])</pre>	Tests the given function for periodicity in the given symbol.
<pre>permutedims(expr[, perm, index_order_old,])</pre>	Permutes the indices of an array.
pexquo(f, g, *gens, **args)	Compute polynomial exact pseudo-quotient of f and g.
<pre>piecewise_exclusive(expr, *[, skip_nan, deep])</pre>	Rewrite Piecewise with mutually exclusive conditions.
<pre>piecewise_fold(expr[, evaluate])</pre>	Takes an expression containing a piecewise function and returns the expression in piecewise form.
plot(*args[, show])	Plots a function of a single variable as a curve.
<pre>plot_implicit(expr[, x_var, y_var,])</pre>	A plot function to plot implicit equations / inequalities.
<pre>plot_parametric(*args[, show])</pre>	Plots a 2D parametric curve.
<pre>polarify(eq[, subs, lift])</pre>	Turn all numbers in eq into their polar equivalents (under the standard choice of argument).
<pre>pollard_pm1(n[, B, a, retries, seed])</pre>	Use Pollard's p-1 method to try to extract a nontrivial factor of n.
pollard_rho(n[, s, a, retries, seed,])	Use Pollard's rho method to try to extract a nontrivial factor of n.
poly(expr, *gens, **args)	Efficiently transform an expression into a polynomial.
poly_from_expr(expr, *gens, **args)	Construct a polynomial from an expression.
posify(eq)	Return eq (with generic symbols made positive) and a dictionary containing the mapping between the old and new symbols.
postfixes(seq)	Generate all postfixes of a sequence.
<pre>postorder_traversal(node[, keys])</pre>	Do a postorder traversal of a tree.
<pre>powdenest(eq[, force, polar])</pre>	Collect exponents on powers as assumptions allow.
<pre>powsimp(expr[, deep, combine, force, measure])</pre>	Reduce expression by combining powers with similar bases and exponents.
pprint(expr, **kwargs)	Prints expr in pretty form.
<pre>pprint_try_use_unicode()</pre>	See if unicode output is available and leverage it if possible
pprint_use_unicode([flag])	Set whether pretty-printer should use unicode by default
pquo(f, g, *gens, **args)	Compute polynomial pseudo-quotient of f and g.
<pre>prefixes(seq)</pre>	Generate all prefixes of a sequence.
	continues on next page

Table 64 – continued from previous page

Table 64 – continue	ed from previous page
<pre>prem(f, g, *gens, **args)</pre>	Compute polynomial pseudo-remainder of $f$ and $g$ .
<pre>pretty_print(expr, **kwargs)</pre>	Prints expr in pretty form.
<pre>preview(expr[, output, viewer, euler,])</pre>	View expression or LaTeX markup in PNG, DVI, PostScript or PDF form.
prevprime(n)	Return the largest prime smaller than n.
<pre>prime(nth)</pre>	Return the nth prime, with the primes indexed as
	prime(1) = 2, $prime(2) = 3$ , etc.
<pre>prime_decomp(p[, T, ZK, dK, radical])</pre>	Compute the decomposition of rational prime $p$ in a number field.
<pre>prime_valuation(I, P)</pre>	Compute the $P$ -adic valuation for an integral ideal $I$ .
<pre>primefactors(n[, limit, verbose])</pre>	Return a sorted list of n's prime factors, ignoring multi- plicity and any composite factor that remains if the limit was set too low for complete factorization.
<pre>primerange(a[, b])</pre>	Generate a list of all prime numbers in the range [2, a), or [a, b).
<pre>primitive(f, *gens, **args)</pre>	Compute content and the primitive form of f.
<pre>primitive_element(extension[, x, ex, polys])</pre>	Find a single generator for a number field given by several generators.
<pre>primitive_root(p)</pre>	Returns the smallest primitive root or None.
<pre>primorial(n[, nth])</pre>	Returns the product of the first n primes (default) or the primes less than or equal to n (when nth=False).
<pre>print_ccode(expr, **settings)</pre>	Prints C representation of the given expression.
<pre>print_fcode(expr, **settings)</pre>	Prints the Fortran representation of the given expression.
<pre>print_glsl(expr, **settings)</pre>	Prints the GLSL representation of the given expression.
<pre>print_gtk(x[, start_viewer])</pre>	Print to Gtkmathview, a gtk widget capable of rendering MathML.
<pre>print_jscode(expr, **settings)</pre>	Prints the Javascript representation of the given expression.
<pre>print_latex(expr, **settings)</pre>	Prints LaTeX representation of the given expression.
<pre>print_maple_code(expr, **settings)</pre>	Prints the Maple representation of the given expression.
<pre>print_mathml(expr[, printer])</pre>	Prints a pretty representation of the MathML code for expr.
<pre>print_python(expr, **settings)</pre>	Print output of python() function
<pre>print_rcode(expr, **settings)</pre>	Prints R representation of the given expression.
<pre>print_tree(node[, assumptions])</pre>	Prints a tree representation of "node".
prod(a[, start])	Return product of elements of a. Start with int 1 so if only
product(*args, **kwargs)	Compute the product.
<pre>proper_divisor_count(n[, modulus])</pre>	Return the number of proper divisors of n.
<pre>proper_divisors(n[, generator])</pre>	Return all divisors of n except n, sorted by default.
public(obj)	Append obj's name to globalall variable (call site).
pycode(expr, **settings)	Converts an expr to a string of Python code
<pre>python(expr, **settings)</pre>	Return Python interpretation of passed expression (can be passed to the exec() function without any modifica- tions)
<pre>quadratic_congruence(a, b, c, p)</pre>	Find the solutions to $a x**2 + b x + c = 0 \mod p$ .
quadratic_residues(p)	Returns the list of quadratic residues.
quo(f, g, *gens, **args)	Compute polynomial quotient of f and g.
rad(d)	Return the radian value for the given degrees (pi = 180 degrees).
<pre>radsimp(expr[, symbolic, max_terms])</pre>	Rationalize the denominator by removing square roots.
	continues on next page

Table 64 – continued from previous page

lable 64 – continue	d from previous page
randMatrix(r[, c, min, max, seed,])	Create random matrix with dimensions r x c.
<pre>random_poly(x, n, inf, sup[, domain, polys])</pre>	Generates a polynomial of degree n with coefficients in [inf, sup].
randprime(a, b)	Return a random prime number in the range [a, b).
rational_interpolate(data, degnum[, X])	Returns a rational interpolation, where the data points
_	are element of any integral domain.
ratsimp(expr)	Put an expression over a common denominator, cancel
- · · • ·	and reduce.
<pre>ratsimpmodprime(expr, G, *gens[, quick,])</pre>	Simplifies a rational expression expr modulo the prime ideal generated by G.
<pre>rcode(expr[, assign_to])</pre>	Converts an expr to a string of r code
rcollect(expr, *vars)	Recursively collect sums in an expression.
<pre>real_root(arg[, n, evaluate])</pre>	Return the real <i>n</i> 'th-root of <i>arg</i> if possible.
real_roots(f[, multiple])	Return a list of real roots with multiplicities of f.
<pre>reduce_abs_inequalities(exprs, gen)</pre>	Reduce a system of inequalities with nested absolute val-
	ues.
reduce_abs_inequality(expr, rel, gen)	Reduce an inequality with nested absolute values.
<pre>reduce_inequalities(inequalities[, symbols])</pre>	Reduce a system of inequalities with rational coefficients.
reduced(f, G, *gens, **args)	Reduces a polynomial f modulo a set of polynomials G.
<pre>refine(expr[, assumptions])</pre>	Simplify an expression using assumptions.
refine_root(f, s, t[, eps, steps, fast,])	Refine an isolating interval of a root to the given precision.
register_handler(key, handler)	Register a handler in the ask system.
rem(f, g, *gens, **args)	Compute polynomial remainder of f and g.
remove_handler(key, handler)	Removes a handler from the ask system.
reshape(seq, how)	Reshape the sequence according to the template in how.
residue(expr, x, x0)	Finds the residue of expr at the point $x=x0$ .
<pre>resultant(f, g, *gens[, includePRS])</pre>	Compute resultant of f and g.
ring(symbols, domain[, order])	Construct a polynomial ring returning (ring, x_1, , x_n).
<pre>root(arg, n[, k, evaluate])</pre>	Returns the <i>k</i> -th <i>n</i> -th root of arg.
<pre>rootof(f, x[, index, radicals, expand])</pre>	An indexed root of a univariate polynomial.
<pre>roots(f, *gens[, auto, cubics, trig,])</pre>	Computes symbolic roots of a univariate polynomial.
rot_axis1(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 1-axis.
rot_axis2(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 2-axis.
rot_axis3(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 3-axis.
<pre>rot_ccw_axis1(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 1-axis.
<pre>rot_ccw_axis2(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 2-axis.
rot_ccw_axis3(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 3-axis.
<pre>rot_givens(i, j, theta[, dim])</pre>	Returns a a Givens rotation matrix, a a rotation in the plane spanned by two coordinates axes.
<pre>rotations(s[, dir])</pre>	Return a generator giving the items in s as list where each subsequent list has the items rotated to the left (default)
round two(T[ rodicale])	or right (dir=-1) relative to the previous list.
round_two(T[, radicals])	Zassenhaus's "Round 2" algorithm.

Table 64 – continued from previous page

Solve_hyper(coeffs, f, n, **hints)	Table of Continue	a nom previous page
der k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f we seek for all hypergeometric solutions over field K of characteristic zero.  Given linear recurrence operator operatorname[L] of order k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f, where f is a polynomial we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname[L] of order k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f, where f is a polynomial operatorname[L] y = f, where f is a polynomial operatorname[L] y = f, where f is a polynomial operatorname[L] y = f, where f is a polynomial operatorname[L] y = f, where f is a polynomial operatorname[L] y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  Converts an expr to a string of Rust code characteristic zero.  Converts an expr to a string of Rust code characteristic zero.  Converts an expr to a string of Rust code characteristic zero.  Sequence(seq.[, limits)]  Sequence(seq.[, limits)]  Sequence(seq.[, limits)]  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Separates variables in an expression, if possible.  Sequence(seq.[, limits))  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Separates variables in an expression in expression in an expression in expression in the control of the expression of expraround point x = x0.  Shape()  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Seterr([divide])  Signal in expression in expression in the		Solve univariate recurrence with rational coefficients.
cquation operatorname(LL) y = f we seek for all hyper-geometric solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(LJ) of order k with polynomial coefficients and inhomogeneous equation operatorname(LJ) y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(LJ) of order k with polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(LJ) of order k with polynomial coefficients and inhomogeneous equation operatorname(LJ) y = f, where f is a polynomial, we seek for all polynomial coefficients and inhomogeneous equation operatorname(LJ) y = f, where f is a polynomial, we seek for all polynomial coefficients and inhomogeneous equation operatorname(LJ) y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(LJ) of order k with polynomial coefficients and inhomogeneous equation operatorname(LJ) y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(LJ) of characteristic zero.  Given linear expertsion operatorname(LJ) of characteristic zero.	rsolve_hyper(coeffs, f, n, **hints)	
rsolve_poly(coeffs, f, n[, shift])  rsolve_poly(coeffs, f, n[, shift])  rsolve_ratio(coeffs, f, n, **hints)  rust_code(expr[, assign_to))  satisfiable(expr[, algorithm, all_models,])  separatevars(expr[, symbols, dict, force])  sequence(seq, limits))  series(expr[, x, x0, n, dir])  series(expr[, x, x0, n, dir])  seterr([divide])  sfield(expr, *symbols, **options)  shape()  sift(seq, keyfuncl, binary)  signsimp(expr[, evaluate])  signsimp(expr[, ratio, measure, rational,])  signipify(expr[, ratio, measure, rational,])  simplify(expr[, ratio, measure, rational,])  singlarities(expression, symbol], domain])  singularities(expression, symbols, exclude])  solve_linear_system_(system, *symbols, **flags)  solve_linear_system_(system, *symbols, **flags)  solve_linear_system_(system, *symbols, **flags)  solve_linear_system_(system, *symbols, **flags)  solve_linear_system_(squ, *gens, *rargs)  solve_rational_inequalities(eqs)  solve_triangulated(polys, *gens, *rargs)		
rsolve_poly(coeffs, f, n , shift )  Given linear recurrence operator operatorname(L) for order k with polynomial coefficients and inhomogeneous equation operatorname(L) y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(L) of order k with polynomial coefficients and inhomogeneous equation operatorname(L) y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  Fust_code(expr , assign_to)  Fust_code(expr , assign_to)  Fust_code(expr , assign_to)  Separatevars(expr , symbols, dict, force)  Separatevars(expr , symbols, dict, force)  Sequence(seqf , imits)  Series(expr , x, x0, n, dir)  Seterr([divide)  Series(expr , x, x0, n, dir)  Series expansion of expr around point x = x0.  Should SymPy raise an exception on 00 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Shape()  Sift(seq, keyfunc , binary )  Signsimp(expr , evaluate )  Simplify(expr , ratio, measure, rational, )  Simplify(expr , ratio, measure, rational, )  Simplify(spr , ratio, measure, rational, )  Singularities(expression, symbol , domain )  Singularities of a given function.  This function simplifies aboolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularities of a given function.  This function simplifies a boolean function to its simplified version in SOP or POS form.  Converts expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Return a uple derived from f = 1 hs - rhs that is one of the following: (0, 1), (0, 0), (symbol), solve and preturns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve		
der k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Frolve_ratio(coeffs, f, n, **hints)  Frolve_ratio(coeffs, f, n, f, coeffs, f, n, r, f, coeffs, f, coeffs, f, n, r,		geometric solutions over field $K$ of characteristic zero.
cquation operatorname(L) y = f, where f is a polynomial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatorname(L) of order k with polynomial coefficients and inhomogeneous equation operatorname(L) y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  rust_code(expr[, assign_to])	<pre>rsolve_poly(coeffs, f, n[, shift])</pre>	
mial, we seek for all polynomial solutions over field K of characteristic zero.  Given linear recurrence operator operatormame[L] of order k with polynomial coefficients and inhomogeneous equation operatormate[L] of exhibit polynomial coefficients and inhomogeneous equation operatormate[L] of exhibit polynomial coefficients and inhomogeneous equation operatormate[L] of exhibit polynomial coefficients and inhomogeneous equation operatormate[L] of characteristic zero.  Converts an expr to a string of Rust code  Converts an expr to a string of Rust code  Separatevars(expr[, algorithm, all_models,])  sequence(seqt, limits])  series(expr[, ax, x0, n, dir])  seterr([divide])  Seties(expr[, x, x0, n, dir])  seterr([divide])  Should SymPy raise an exception on 0/0 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Shape()  Sift(seq, keyfunc[, binary])  Sift the sequence, seq according to keyfunc.  Simplifies the shape of the expr as a tuple.  Sift(seq, keyfunc[, binary])  Simplifies the spee expr as a tuple.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the given expression.  Simplifies the given expression.  Simplifies the given expression.  Compute the unitary, ordinary-frequency sine transform of f, defined as  singularities(expression, symbol[, domain])  singularityintegrate(f, x)  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  singularity functions.  Compute the unitary ordinary-frequency sine transform of f, defined as  solve_linear(lns], rhs, symbols, exclude])  solve(f, *symbols, **flags)  Algebraically solves equations and systems of equations.  Return a tuple derived from f = 1hs - rhs that is one of the following: (0, 1), (0, 0), (symbol), solution), (n, d).  Solve apolynomial inequality with rational coefficients.  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_G,		· · · · · · · · · · · · · · · · · · ·
of characteristic zero.  Given linear recurrence operator operatorname[L] of order k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  Converts an expr to a string of Rust code  Separatevars(expr[, algorithm, all_models,])  separatevars(expr[, symbols, dict, force])  sequence(seq[, limits])  series(expr[, x, x0, n, dir])  seterr([divide])  sfield(exprs, *symbols, **options)  sfield(exprs, *symbols, **options)  shape()  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  simplify(expr[, ratio, measure, rational,])  simplify(expr[, ratio, measure, rational,])  simplify(expr[, ratio, measure, rational,])  simplify(expr[, ratio, measure, rational,])  simplify(expr[, auto, measure, rational,])  singularrities(expression, symbol[, domain])  singularrities(expression, symbols, domain])  singularrities(expression, symbols, exclude])  solve_linear_system(system, *symbols, exclude])  solve_linear_system(system, *symbols, **flags)  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  solve_poly_system(seq, *gens[, strict])  solve_poly_system(seq, *gens[, strict])  solve_poly_system(seq, *gens[		
rsolve_ratio(coeffs, f, n, **hints)  Given linear recurrence operator operatormame[L] of order k with polynomial coefficients and inhopeaneous equation operatormame[L] y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  Converts an expr to a string of Rust code Check satisfiability of a propositional sentence. Separatevars(expf, algorithm, all_models,) Separatevars(expf, symbols, dict, force]) Sequence(seqf, limits]) Series(expf, x, x0 n, dir]) Series(expf, x, x0 n, dir]) Seterr((divide)) Seterr((divide)) Sfield(exprs, *symbols, **options) Shape() Sift(seq, keyfuncl, binary]) Signsimp(expf, evaluate) Signsimp(expf, evaluate) Simplify(expf, ratio, measure, rational,]) Simplify(expf, ratio, measure, rational,]) Simplify_logic(expf, form, deep, force,]) Singularities(expression, symbol, domain]) Singularities(expression, symbol, domain)) Singularities(expression, symbol, domain)) Singularityintegrate(f, x) Solve(f, *symbols, **flags) Solve_linear_system(system, *symbols, **flags) Solve_linear_system(system, *symbols, **flags) Solve_linear_system(system, *symbols, **flags) Solve_linear_system_LU(matrix, syms)  Give hone Solve_rational_inequalities(exp) Solve apolynomial system using Gianni-Kalkbrenner algorithm.		
der k with polynomial coefficients and inhomogeneous equation operatorname[L] y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  Converts an expr to a string of Rust code Satisfiable(expr[, algorithm, all_models,]) separatevars(expr[, symbols, dict, force]) sequence(seq[, limits]) sequence(seq[, limits]) Series(expr[, x, x0, n, dir]) Series(expr[, x, x0, n, dir]) Seterr([divide]) Should SymPy raise an exception on 0/0 or return a nan? Sfield(exprs, *symbols, **options)  Shape() Sift(seq, keyfunc[, binary]) Signsimp(expr[, evaluate]) Simplify(expr[, ratio, measure, rational,]) Simplify(expr[, ratio, measure, rational,]) Simplify_logic(expr[, form, deep, force,]) Singularities(expression, symbol], domain]) Singularityintegrate(f, x)  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as Singularityintegrate(f, x)  This function simplifies a boolean function to Singularity functions.  Smtlib_code(expr[, auto_assert,])  Solve_linear_system(system, *symbols, exclude])  Solve_linear_system(system, *symbols, *sflags)  Solve_linear_system(system, *symbols, *sflags)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel) Solve_poly_inequality(poly, rel) Solve_poly_inequality(poly, rel) Solve_poly_inequality(poly, rel) Solve_rational_inequalities(eqs)  Solve_a polynomial system using Gianni-Kalkbrenner algorithm.		
equation operatorname[L] y = f, where f is a polynomial, we seek for all rational solutions over field K of characteristic zero.  rust_code(expr[, assign_to])	<pre>rsolve_ratio(coeffs, f, n, **hints)</pre>	
mial, we seek for all rational solutions over field K of characteristic zero.  rust_code(expr[, assign_to])		
characteristic zero.  Converts an expr to a string of Rust code  satisfiable(expr[, assign_to])  separatevars(expr[, symbols, dict, force])  sequence(seq[, limits])  series(expr[, x, x0, n, dir])  seterr([divide])  sfield(exprs, *symbols, **options)  shape()  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  simplify(expr[, fratio, measure, rational,])  simplify(expr[, fratio, measure, rational,])  sine_transform(f, x, k, **hints)  singularities(expression, symbol[, domain])  singularities(expression, symbol], domain])  singularities(expression, symbol], domain])  solve_linear_system(system, *symbols, exclude])  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_poly_inequality(poly, rel)  solve_rational_inequalities(eqs)  solve_triangulated(polys, *gens, **args)  solve_triangulated(polys, *gens, **args)  solve_triangulated(polys, *gens, **args)  characteristic exprex on a string of Rust code  Converts an expr to a string of Rust code  Separatev aria expr to a string of Rust code  Converts an expr to a string of Rust code  Converts an expr to a string of Rust code  Construct a field expression, if possible.  Return a tuple derived from f = 1hs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  solve_poly_inequality(poly, rel)  solve_poly_inequality(poly, rel)  solve_poly_inequality(poly, rel)  solve_rational_inequalities(eqs)  solve_triangulated(polys, *gens, **args)		
rust_code(expr[, assign_to])		
satisfiable(expr[, algorithm, all_models,])  separatevars(expr[, symbols, dict, force])  sequence(seq[, limits])  series(expr[, x, x0, n, dir])  seterr([divide])  sfield(exprs, *symbols, **options)  shape()  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  simplify(expr[, ratio, measure, rational,])  simplify_logic(expr[, form, deep, force,])  singlarities(expression, symbol[, domain])  singularities(expression, symbol[, domain])  singularityintegrate(f, x)  smtlib_code(expr[, auto_assert,])  solve_linear_system(system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  characterists and spream and system using Culonus are supported.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  solve_triangulated(polys, *gens, **args)  Check satisfiability of a propositional expression; for a cxpression, if possible.  Returns appropriate sequence object.  Separates variables in an expression, if possible.  Returns appropriate sequence object.  Series expansion of expr around point x = x0.  Should SymPy raise an exception on 000 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Should SymPy raise an exception on 000 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Simplify the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the sequence, seq according to keyfunc.  Make all Add sub-expressions.  This function simplifies at boolean function to its simplified version in SOP or POS form.  Converte expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Solve apolynomial inequalit		
separatevars(expr[, symbols, dict, force]) sequence(seq[, limits]) series(expr[, x, x0, n, dir]) seterr([divide]) Series(expr[, x, x0, n, dir]) seterr([divide]) Should SymPy raise an exception on 0/0 or return a nan? Sfield(exprs, *symbols, **options)  Shape() Sift(seq, keyfunc[, binary]) Signsimp(expr[, evaluate]) Simplify(expr[, raito, measure, rational,]) Simplify(expr[, raito, measure, rational,]) Simplify(expr[, raito, measure, rational,]) Singliarities(expression, symbol[, domain]) Singularities(expression, symbol[, domain]) Singularities(expression, symbol[, domain]) Singularityintegrate(f, x) Solves[, *symbols, **flags) Solve_linear_system_(system, *symbols, exclude]) Solve_linear_system_(system, *symbols, **flags) Solve_linear_system_LU(matrix, syms) Solve_poly_inequality(poly, rel) Solve_poly_system(seq, *gens[, strict]) Setions (seq and solve applynomial system using Gianni-Kalkbrenner algorithm.		
sequence(seq[, limits]) series(exprf, x, x0, n, dir]) series(exprf, x, x0, n, dir]) seterr([divide]) sfield(exprs. *symbols, **options)  Should SymPy raise an exception on 0/0 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Shape()  Sift(seq, keyfunc[, binary]) Sift es sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplify(expr[, ratio, measure, rational,]) Simplify_logic(expr[, form, deep, force,]) Simplify_logic(expr[, form, deep, force,])  Sine_transform(f, x, k, **hints)  Sine_transform(f, x, k, **hints)  Compute the unitary, ordinary-frequency sine transform of f, defined as  singularities(expression, symbol[, domain])  singularityintegrate(f, x)  This function handles the indefinite integrations of Singularity functions.  Solve_linear(lhs[, rhs, symbols, exclude])  solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  solve_poly_inequality(poly, rel) solve_poly_system(seq, *gens[, strict])  solve_poly_system(seq, *gens[, strict])  solve_triangulated(polys, *gens, **args)  Return a tuple derived matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve a polynomial inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve_a polynomial system using Gianni-Kalkbrenner algorithm.	· · · · · · · · · · · · · · · · · · ·	* 1 1
series(expr[, x, x0, n, dir]) seterr([divide]) Series expansion of expr around point x = x0.  Should SymPy raise an exception on 0/0 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Shape() Sift(seq, keyfunc[, binary]) Signsimp(expr[, evaluate]) Signsimp(expr[, evaluate]) Simplify(expr[, ratio, measure, rational,]) Simplify_logic(expr[, form, deep, force,]) Simplify_logic(expr[, form, deep, force,]) Simplifies the given expression.  Sine_transform(f, x, k, **hints) Compute the unitary, ordinary-frequency sine transform of f, defined as  Singularities(expression, symbol[, domain]) Singularityintegrate(f, x) This function handles the indefinite integrations of Singularity functions.  Smtlib_code(expr[, auto_assert,]) Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel) Solve apolynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs) Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args) Solve a spolynomial system using Gianni-Kalkbrenner algorithm.	· · · · · · · · · · · · · · · · · ·	1 1
seterr([divide])  sfield(exprs, *symbols, **options)  Should SymPy raise an exception on 0/0 or return a nan?  Construct a field deriving generators and domain from options and input expressions.  Return the shape of the expr as a tuple.  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  simplify(expr[, ratio, measure, rational,])  simplify_logic(expr[, form, deep, force,])  simplify_logic(expr[, form, deep, force,])  sine_transform(f, x, k, **hints)  Simplifies the given expression canonical wrt sign.  Singularities(expression, symbol[, domain])  singularityintegrate(f, x)  singularityintegrate(f, x)  solve_fi, *symbols, **flags)  solve_linear(lhs[, rhs, symbols, exclude])  solve_linear_system_system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_poly_inequality(poly, rel)  solve_poly_inequality(poly, rel)  solve_rational_inequalities(eqs)  solve_triangulated(polys, *gens, **args)  Should SymPy raise an exception on 0/0 or return a nan?  Construct a field deriving generators and domain from options and inel fled from options and the expr as a tuple.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifies the given expressions canonical wrt sign.  Simplifies the given expression canonical wrt sign.  Simplifies the given expression expression son.  Th		
sfield(exprs, *symbols, **options)  Shape()  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  simplify(expr[, ratio, measure, rational,])  simplify_logic(expr[, form, deep, force,])  sine_transform(f, x, k, **hints)  Singularities(expression, symbol[, domain])  singularityintegrate(f, x)  solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel)  solve_triangulated(polys, *gens, **args)  Construct a field deriving generators and domain from options and input expressions.  Return the shape of the expr as a tuple.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplify(expr[, ratio, measure, rational,])  Simplifies the given expression.  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularities of a given function.  This function handles the indefinite integrations of Singularity functions.  Solve_fined as  Find singularities of a given function.  Converts expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Return a tuple derived from f = 1hs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  Solve_linear_system(system, *symbols, **flags)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solve a polynomial inequality with rational coefficients.  Return a list of solutions for the system of polynomial equations or else None.  Solve a polynomial inequalities with rational coefficients.  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		• • •
shape() sift(seq, keyfunc[, binary]) sift(seq, keyfunc[, binary]) signsimp(expr[, evaluate]) simplify(expr[, ratio, measure, rational,]) simplify_logic(expr[, form, deep, force,]) simplify_logic(expr[, form, deep, force,]) sine_transform(f, x, k, **hints) singularities(expression, symbol[, domain]) singularities(expression, symbol[, domain]) singularityintegrate(f, x) solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  coptions and input expressions. Return the shape of the expr as a tuple. Sift the sequence, seq according to key func. Make all Add sub-expressions canonical wrt sign. Simplifies the given expression. This function simplifies a boolean function to its simplified version in SOP or POS form. Compute the unitary, ordinary-frequency sine transform of f, defined as singularities of a given function. This function handles the indefinite integrations of Singularity functions. Converts expr to a string of smtlib code. Algebraically solves equations and systems of equations. Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d). Solve_linear_system_system, *symbols, **flags) Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported. Solve a polynomial inequality with rational coefficients. Solve_poly_inequality(poly, rel) Solve a polynomial inequality with rational coefficients. Solve_rational_inequalities(eqs) Solve a system of rational inequalities with rational coefficients. Solve_traingulated(polys, *gens, **args) Solve a polynomial system using Gianni-Kalkbrenner algorithm.	(5)	
shape()  sift(seq, keyfunc[, binary])  signsimp(expr[, evaluate])  signsimp(expr[, evaluate])  simplify(expr[, ratio, measure, rational,])  simplify_logic(expr[, form, deep, force,])  simplify_logic(expr[, form, deep, force,])  sine_transform(f, x, k, **hints)  singularities(expression, symbol[, domain])  singularityintegrate(f, x)  smtlib_code(expr[, auto_assert,])  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  solve_triangulated(polys, *gens, **args)  solve_triangulated(polys, *gens, **args)  Return the shape of the expr as a tuple.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Sift the sequence, seq according to keyfunc.  Make all Add sub-expressions canonical wrt sign.  Simplifes the given expressions canonical wrt sign.  Simplifes the given expression.  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f. defined as  Find singularities of a given function.  This function simplifies aboolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f. defined as  Find singularities of a given function.  This function simplifies abolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-freq	stretu(exprs, symbols, options)	
sift(seq, keyfunc[, binary]) signsimp(expr[, evaluate]) simplify(expr[, ratio, measure, rational,]) simplify_logic(expr[, form, deep, force,]) simplify_logic(expr[, form, deep, force,]) simplify_logic(expr[, form, deep, force,]) sine_transform(f, x, k, **hints)  sine_transform(f, x, k, **hints)  singularities(expression, symbol[, domain]) singularityintegrate(f, x)  singularityintegrate(f, x)  solve_linear(lhs[, rhs, symbols, exclude])  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_poly_inequality(poly, rel) solve_poly_system(seq, *gens[, strict])  solve_rational_inequalities(eqs)  solve_triangulated(polys, *gens, **args)  Sift the sequence, seq according to keyfunc. Make all Add sub-expressions canonical wrt sign. Make all Add sub-expressions canonical wrt sign. Make all Add sub-expressions canonical wrt sign.  Make all Add sub-expressions canonical wrt sign.  Simplifies the given expression. Simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find super form of given function.  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularity ordinary-frequency sine transform of f, defined as  Find singularity interious.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularity functions.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularity functions.  Converts expr t	shane()	· · ·
signsimp(expr[, evaluate]) simplify(expr[, ratio, measure, rational,]) simplify(expr[, ratio, measure, rational,]) simplify_logic(expr[, form, deep, force,]) simplify_logic(expr[, form, deep, force,]) sine_transform(f, x, k, **hints)  sine_transform(f, x, k, **hints)  compute the unitary, ordinary-frequency sine transform of f, defined as singularities(expression, symbol[, domain]) singularityintegrate(f, x)  sine_transform(f, x, k, **hints)  compute the unitary, ordinary-frequency sine transform of f, defined as singularityintegrate(f, x)  Find singularities of a given function.  This function handles the indefinite integrations of Singularity functions.  Converts expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  Solve system of \$NS\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  Solve a polynomial inequality with rational coefficients.  Return a list of solutions for the system of polynomial equations or else None.  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	_ "	
simplify(expr[, ratio, measure, rational,]) simplify_logic(expr[, form, deep, force,]) simplify_logic(expr[, form, deep, force,]) sine_transform(f, x, k, **hints)  Sine_transform(f, x, k, **hints)  Singularities(expression, symbol[, domain]) singularityintegrate(f, x)  Singularityintegrate(f, x)  Solve(f, *symbols, **flags)  Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solve_poly_inequality(poly, rel) Solve a polynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs)  Solve a system using Gianni-Kalkbrenner algorithm.		
simplify_logic(expr[, form, deep, force,])  sine_transform(f, x, k, **hints)  Sine_transform(f, x, k, **hints)  Singularities(expression, symbol[, domain])  singularities(expression, symbol[, domain])  singularityintegrate(f, x)  Singularityintegrate(f, x)  Singularityintegrate(f, x)  Solve_f, *symbols, **flags)  Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel)  Solve_poly_system(seq, *gens[, strict])  Solve_rational_inequalities(eqs)  Solve_triangulated(polys, *gens, **args)  This function simplifies a boolean function to its simplified version in SOP or POS form.  Compute the unitary, ordinary-frequency sine transform of f, defined as  Find singularity; of a given function.  This function handles the indefinite integrations of Singularity functions.  Converts expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solve the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
fied version in SOP or POS form.  Sine_transform(f, x, k, **hints)  Singularities(expression, symbol[, domain])  Singularityintegrate(f, x)  Singularityintegrate(f, x)  Singularityintegrate(f, x)  Solve(f, *symbols, **flags)  Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel)  Solve_poly_system(seq, *gens[, strict])  Solve_rational_inequalities(eqs)  Solve_triangulated(polys, *gens, **args)  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity, ordinary-frequency sine transform of f, defined as  Find singularity functions.  Find singularities of a given function.  This function handles the indefinite integrations of Singularity functions.  Solves expr to a string of smtlib code.  Algebraically solves equations and systems of equations.  Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  Solve system of \$N\$\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
singularities(expression, symbol[, domain]) singularityintegrate(f, x)  Smtlib_code(expr[, auto_assert,]) Solve(f, *symbols, **flags) Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel) Solve_poly_system(seq, *gens[, strict])  Solve_rational_inequalities(eqs) Solve_triangulated(polys, *gens, **args)  Solve a system using Gianni-Kalkbrenner algorithm.		
singularities(expression, symbol[, domain]) singularityintegrate(f, x)  Smtlib_code(expr[, auto_assert,]) Solve(f, *symbols, **flags) Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_poly_inequality(poly, rel) Solve_poly_system(seq, *gens[, strict])  Solve_rational_inequalities(eqs) Solve_triangulated(polys, *gens, **args)  Solve a system using Gianni-Kalkbrenner algorithm.	<pre>sine_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency sine transform
singularityintegrate(f, x)  Smtlib_code(expr[, auto_assert,])  Solve(f, *symbols, **flags)  Solve_linear(lhs[, rhs, symbols, exclude])  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system(system, *symbols, **flags)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_LU(matrix, syms)  Solve_linear_system_system_symbols, **flags)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solve she augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
gularity functions.  smtlib_code(expr[, auto_assert,])  solve(f, *symbols, **flags)  solve_linear(lhs[, rhs, symbols, exclude])  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_linear_system_linear_system_linear_system is solve_linear_system is solve_linear_system is solve_linear_system.  Solve a polynomial inequality with rational coefficients.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  Solve a polynomial inequality with rational coefficients.  solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	<pre>singularities(expression, symbol[, domain])</pre>	Find singularities of a given function.
smtlib_code(expr[, auto_assert,])  solve(f, *symbols, **flags)  solve_linear(lhs[, rhs, symbols, exclude])  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_system_symbols, **flags)  solve_linear_system_system_symbols, **flags)  solve_linear_system_system_symbols, **flags)  solve_linear_system_system_symbols, **flags)  solve_linear_system_symbols, **flags)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  solve_poly_inequality(poly, rel)  solve a polynomial inequality with rational coefficients.  solve_rational_inequalities(eqs)  solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	singularityintegrate(f, x)	This function handles the indefinite integrations of Sin-
solve(f, *symbols, **flags)Algebraically solves equations and systems of equations.solve_linear(lhs[, rhs, symbols, exclude])Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).solve_linear_system(system, *symbols, **flags)Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.solve_linear_system_LU(matrix, syms)Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.solve_poly_inequality(poly, rel)Solve a polynomial inequality with rational coefficients.solve_poly_system(seq, *gens[, strict])Return a list of solutions for the system of polynomial equations or else None.solve_rational_inequalities(eqs)Solve a system of rational inequalities with rational coefficients.solve_triangulated(polys, *gens, **args)Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
Return a tuple derived from f = lhs - rhs that is one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  solve_linear_system(system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Return a list of solutions for the system of polynomial equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		- •
one of the following: (0, 1), (0, 0), (symbol, solution), (n, d).  solve_linear_system(system, *symbols, **flags)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
solve_linear_system(system, *symbols, **flags)  solve_linear_system(system, *symbols, **flags)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  solve_linear_system_LU(matrix, syms)  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  solve_rational_inequalities(eqs)  solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	<pre>solve_linear(lhs[, rhs, symbols, exclude])</pre>	
Solve_linear_system(system, *symbols, **flags)  Solve system of \$N\$ linear equations with \$M\$ variables, which means both under- and overdetermined systems are supported.  Solve_linear_system_LU(matrix, syms)  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
ables, which means both under- and overdetermined systems are supported.  solve_linear_system_LU(matrix, syms)  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
tems are supported.  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	<pre>solve_linear_system(system, *symbols, **flags)</pre>	
Solve_linear_system_LU(matrix, syms)  Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  Solve_poly_inequality(poly, rel)  Solve a polynomial inequality with rational coefficients.  Solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  Solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		•
returns a dictionary in which solutions are keyed to the symbols of syms as ordered.  solve_poly_inequality(poly, rel) Solve a polynomial inequality with rational coefficients.  solve_poly_system(seq, *gens[, strict]) Return a list of solutions for the system of polynomial equations or else None.  solve_rational_inequalities(eqs) Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args) Solve a polynomial system using Gianni-Kalkbrenner algorithm.	1 11	
symbols of syms as ordered.  solve_poly_inequality(poly, rel)  solve_poly_system(seq, *gens[, strict])  Solve a polynomial inequality with rational coefficients.  Return a list of solutions for the system of polynomial equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	solve_linear_system_LU(matrix, syms)	
solve_poly_inequality(poly, rel)Solve a polynomial inequality with rational coefficients.solve_poly_system(seq, *gens[, strict])Return a list of solutions for the system of polynomial equations or else None.solve_rational_inequalities(eqs)Solve a system of rational inequalities with rational coefficients.solve_triangulated(polys, *gens, **args)Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
solve_poly_system(seq, *gens[, strict])  Return a list of solutions for the system of polynomial equations or else None.  solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	colve poly inequality/poly (1)	
equations or else None.  Solve_rational_inequalities(eqs)  Solve a system of rational inequalities with rational coefficients.  solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.		
solve_rational_inequalities(eqs)Solve a system of rational inequalities with rational coefficients.solve_triangulated(polys, *gens, **args)Solve a polynomial system using Gianni-Kalkbrenner algorithm.	Solve_poly_System(seq, "gens[, strict])	
efficients. solve_triangulated(polys, *gens, **args) Solve a polynomial system using Gianni-Kalkbrenner algorithm.	solve rational inequalities(eqs)	=
solve_triangulated(polys, *gens, **args)  Solve a polynomial system using Gianni-Kalkbrenner algorithm.	301ve_ractonar_inequatictes(eqs)	
gorithm.	solve triangulated(nolve *gene **arge)	
	sorve_crranguraceu(porys, gens, args)	
Antinuas an navi nada		continues on next page

Table 64 – continued from previous page

Table 64 – continued from previous page	
<pre>solve_undetermined_coeffs(equ, coeffs,)</pre>	Solve a system of equations in \$k\$ parameters that is formed by matching coefficients in variables coeffs that are on factors dependent on the remaining variables (or those given explicitly by syms.
<pre>solve_univariate_inequality(expr, gen[,])</pre>	Solves a real univariate inequality.
solveset(f[, symbol, domain])	Solves a given inequality or equation with set as output
sqf(f, *gens, **args)	Compute square-free factorization of f.
sqf_list(f, *gens, **args)	Compute a list of square-free factors of f.
sqf_norm(f, *gens, **args)	Compute square-free norm of f.
sqf_norm(i, *gens, **args) sqf_part(f, *gens, **args)	Compute square-free norm of 1.  Compute square-free part of f.
	1 1 1
sqrt(arg[, evaluate])	Returns the principal square root. Find a root of $x^**2 = a \mod p$ .
sqrt_mod(a, p[, all_roots])	
<pre>sqrt_mod_iter(a, p[, domain])</pre>	Iterate over solutions to $x^**2 = a \mod p$ .
<pre>sqrtdenest(expr[, max_iter])</pre>	Denests sqrts in an expression that contain other square
· / • 1 1 • • · ·	roots if possible, otherwise returns the expr unchanged.
sring(exprs, *symbols, **options)	Construct a ring deriving generators and domain from options and input expressions.
<pre>stationary_points(f, symbol[, domain])</pre>	Returns the stationary points of a function (where deriva-
	tive of the function is 0) in the given domain.
sturm(f, *gens, **args)	Compute Sturm sequence of f.
<pre>subresultants(f, g, *gens, **args)</pre>	Compute subresultant PRS of f and g.
<pre>subsets(seq[, k, repetition])</pre>	Generates all $k$ -subsets (combinations) from an $n$ -element set, seq.
<pre>substitution(system, symbols[, result,])</pre>	Solves the <i>system</i> using substitution method.
summation(f, *symbols, **kwargs)	Compute the summation of f with respect to symbols.
swinnerton_dyer_poly(n[, x, polys])	Generates n-th Swinnerton-Dyer polynomial in <i>x</i> .
	• • •
symarray(prefix, shape, **kwargs)	Create a numpy ndarray of symbols (as an object array).
symbols(names, *[, cls])	Transform strings into instances of Symbol class.
symmetric_poly(n, *gens[, polys])	Generates symmetric polynomial of order <i>n</i> .
<pre>symmetrize(F, *gens, **args)</pre>	Rewrite a polynomial in terms of elementary symmetric polynomials.
<pre>sympify(a[, locals, convert_xor, strict,])</pre>	Converts an arbitrary expression to a type that can be used inside SymPy.
take(iter, n)	Return n items from iter iterator.
tensorcontraction(array, *contraction_axes)	Contraction of an array-like object on the specified axes.
tensordiagonal(array, *diagonal_axes)	Diagonalization of an array-like object on the specified axes.
tensorproduct(*args)	Tensor product among scalars or array-like objects.
terms_gcd(f, *gens, **args)	Remove GCD of terms from f.
textplot(expr, a, b[, W, H])	Print a crude ASCII art plot of the SymPy expression
23 cp = 2 c (0pr.; u, 0[,, 11])	'expr' (which should contain a single symbol, e.g. x or
	something else) over the interval [a, b].
threaded(func)	Apply func to subelements of an object, including
timed(fine)[ cotum limit])	Add.
timed(func[, setup, limit])	Adaptively measure execution time of a function.
to_cnf(expr[, simplify, force])	Convert a propositional logical sentence expr to con-
	junctive normal form: ((A   ~B  ) & (B   C
	) &).
to_dnf(expr[, simplify, force])	Convert a propositional logical sentence expr to dis-
	junctive normal form: ((A & ~B &)   (B & C
	&)  ).
to_nnf(expr[, simplify])	Converts expr to Negation Normal Form (NNF).
	continues on next page

Table 64 – continued from previous page

to_number_field(extension[, theta, gen, alias])	Express one algebraic number in the field generated by another.
together(expr[, deep, fraction])	Denest and combine rational expressions using symbolic methods.
<pre>topological_sort(graph[, key])</pre>	Topological sort of graph's vertices.
total_degree(f, *gens)	Return the total_degree of f in the given variables.
trace(expr)	Trace of a Matrix.
trailing(n)	Count the number of trailing zero digits in the binary representation of n, i.e. determine the largest power of 2 that divides n.
<pre>trigsimp(expr[, inverse])</pre>	Returns a reduced expression by using known trig identities.
trunc(f, p, *gens, **args)	Reduce f modulo a constant p.
unbranched_argument(arg)	Returns periodic argument of arg with period as infinity.
<pre>unflatten(iter[, n])</pre>	Group iter into tuples of length n.
<pre>unpolarify(eq[, subs, exponents_only])</pre>	If $p$ denotes the projection from the Riemann surface of the logarithm to the complex line, return a simplified version $eq'$ of $eq$ such that $p(eq') = p(eq)$ .
use(expr, func[, level, args, kwargs])	Use func to transform expr at the given level.
var(names, **args)	Create symbols and inject them into the global namespace.
<pre>variations(seq, n[, repetition])</pre>	Returns an iterator over the n-sized variations of seq (size N).
vfield(symbols, domain[, order])	Construct new rational function field and inject generators into global namespace.
<pre>viete(f[, roots])</pre>	Generate Viete's formulas for f.
<pre>vring(symbols, domain[, order])</pre>	Construct a polynomial ring and inject $x_1, \ldots, x_n$ into the global namespace.
<pre>wronskian(functions, var[, method])</pre>	Compute Wronskian for [] of functions
xfield(symbols, domain[, order])	Construct new rational function field returning (field, $(x1,, xn)$ ).
<pre>xring(symbols, domain[, order])</pre>	Construct a polynomial ring returning (ring, $(x_1,, x_n)$ ).
xthreaded(func)	Apply func to subelements of an object, excluding Add.
zeros(*args, **kwargs)	Returns a matrix of zeros with rows rows and cols columns; if cols is omitted a square matrix will be returned.

## Classes

Abs(arg)	Return the absolute value of the argument.
AccumBounds	alias of AccumulationBounds
Add(*args[, evaluate, _sympify])	Expression representing addition operation for algebraic group.
Adjoint(*args, **kwargs)	The Hermitian adjoint of a matrix expression.
<pre>AlgebraicField(dom, *ext[, alias])</pre>	Algebraic number field QQ(a)
<pre>AlgebraicNumber(expr[, coeffs, alias])</pre>	Class for representing algebraic numbers in SymPy.
And(*args)	Logical AND function.
AppliedPredicate(predicate, *args)	The class of expressions resulting from applying Predicate to the arguments.

continues on next page

Table 65 – continued from previous page

Iable 65 – continue	d from previous page
Array	alias of ImmutableDenseNDimArray
AssumptionsContext	Set containing default assumptions which are applied to
-	the ask() function.
Atom(*args)	A parent class for atomic things.
AtomicExpr(*args)	A parent class for object which are both atoms and Exprs.
AutoSympy(model)	
Basic(*args)	Base class for all SymPy objects.
BlockDiagMatrix(*mats)	A sparse matrix with block matrices along its diagonals
<pre>BlockMatrix(*args, **kwargs)</pre>	A BlockMatrix is a Matrix comprised of other matrices.
CRootOf	alias of ComplexRootOf
Chi(z)	Cosh integral.
Ci(z)	Cosine integral.
Circle(*args, **kwargs)	A circle in space.
Complement(a, b[, evaluate])	Represents the set difference or relative complement of
	a set with another set.
<pre>ComplexField([prec, dps, tol])</pre>	Complex numbers up to the given precision.
ComplexRegion(sets[, polar])	Represents the Set of all Complex Numbers.
ComplexRootOf(f, x[, index, radicals, expand])	Represents an indexed complex root of a polynomial.
ConditionSet(sym, condition[, base_set])	Set of elements which satisfies a given condition.
Contains(x, s)	Asserts that x is an element of the set S.
CoordMap(var_vector, eq_duals, ineq_duals,)	
CocineTransform(*ercs)	Class representing unavaluated assine transforms
CosineTransform(*args) Curve(function, limits)	Class representing unevaluated cosine transforms.  A curve in space.
DeferredVector(name, **assumptions)	A curve in space.  A vector whose components are deferred (e.g. for use
belefied vector (name, assumptions)	with lambdify).
DenseNDimArray(*args, **kwargs)	with famounty).
20130132111224)(4180, 11/4180)	
Derivative(expr, *variables, **kwargs)	Carries out differentiation of the given expression with
	respect to symbols.
Determinant(mat)	Matrix Determinant
DiagMatrix(vector)	Turn a vector into a diagonal matrix.
DiagonalMatrix(*args, **kwargs)	DiagonalMatrix(M) will create a matrix expression that
	behaves as though all off-diagonal elements, M[i, j]
	where $i != j$ , are zero.
<pre>DiagonalOf(*args, **kwargs)</pre>	DiagonalOf(M) will create a matrix expression that is
	equivalent to the diagonal of $M$ , represented as a single
	column matrix.
Dict(*args)	Wrapper around the builtin dict object.
<pre>DifferentialMapping(US, coord2item,)</pre>	
Diva chalta(anal 1-1)	The Dime Delte function and its decire
DiracDelta(arg[, k]) Discipit Union(*cata)	The DiracDelta function and its derivatives.
DisjointUnion(*sets)	Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.
Domain()	Superclass for all domains in the polys domains system.
DotProduct(arg1, arg2)	Dot product of vector matrices
Dummy([name, dummy_index])	Dummy symbols are each unique, even if they have the
Zammy ([name, dammy_midex])	same name:
EPath(path)	Manipulate expressions using paths.
Ei(z)	The classical exponential integral.
Ellipse([center, hradius, vradius, eccentricity])	An elliptical GeometryEntity.
	continues on next page

Table 65 – continued from previous page

Table 65 – continue	d from previous page
Eq	alias of Equality
Equality(lhs, rhs, **options)	An equal relation between two objects.
Equivalent(*args)	Equivalence relation.
Expr(*args)	Base class for algebraic expressions.
<pre>ExpressionDomain()</pre>	A class for arbitrary expressions.
FF	alias of FiniteField
FF_gmpy	alias of GMPYFiniteField
FF_python	alias of PythonFiniteField
FallingFactorial(x, k)	Falling factorial (related to rising factorial) is a double
	valued function arising in concrete mathematics, hyper- geometric functions and series expansions.
<pre>FiniteField(mod[, symmetric])</pre>	Finite field of prime order GF(p)
	Represents a finite set of Sympy expressions.
FiniteSet(*args, **kwargs) Float(num[, dps, precision])	Represents a floating-point number of arbitrary precision.
FourierTransform(*args)	Class representing unevaluated Fourier transforms.
FractionField(domain_or_field[, symbols, order])	A class for representing multivariate rational function
rractionrietu(domani_or_neid[, symbols, order])	fields.
Function(*args)	Base class for applied mathematical functions.
<pre>FunctionClass(*args, **kwargs)</pre>	Base class for function classes.
FunctionMatrix(rows, cols, lamda)	Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.
GF	alias of FiniteField
<pre>GMPYFiniteField(mod[, symmetric])</pre>	Finite field based on GMPY integers.
GMPYIntegerRing()	Integer ring based on GMPY's mpz type.
GMPYRationalField()	Rational field based on GMPY's mpq type.
Ge	alias of GreaterThan
GreaterThan(lhs, rhs, **options)	Class representations of inequalities.
GroebnerBasis(F, *gens, **args)	Represents a reduced Groebner basis.
Gt	alias of StrictGreaterThan
HadamardPower(base, exp)	Elementwise power of matrix expressions
<pre>HadamardProduct(*args[, evaluate, check])</pre>	Elementwise product of matrix expressions
<pre>HankelTransform(*args)</pre>	Class representing unevaluated Hankel transforms.
Heaviside(arg[, H0])	Heaviside step function.
ITE(*args)	If-then-else clause.
Identity(n)	The Matrix Identity I - multiplicative identity
Idx(label[, range])	Represents an integer index as an Integer or integer expression.
<pre>ImageSet(flambda, *sets)</pre>	Image of a set under a mathematical function.
ImmutableDenseMatrix(*args, **kwargs)	Create an immutable version of a matrix.
ImmutableDenseNDimArray(iterable[, shape])	
ImmutableMatrix	alias of ImmutableDenseMatrix
ImmutableSparseMatrix(*args, **kwargs)	Create an immutable version of a sparse matrix.
ImmutableSparseNDimArray([iterable, shape])	
Implies(*args)	Logical implication.
<pre>Indexed(base, *args, **kw_args)</pre>	Represents a mathematical object with indices.
<pre>IndexedBase(label[, shape, offset, strides])</pre>	Represent the base or stem of an indexed object
Integer(i)	Represents integer numbers of any size.
<pre>IntegerRing()</pre>	The domain ZZ representing the integers $mathbb\{Z\}$ .
<pre>Integral(function, *symbols, **assumptions)</pre>	Represents unevaluated integral.
	continues on next nage

Table 65 – continued from previous page

Table 65 – continued	d from previous page
Intersection(*args, **kwargs)	Represents an intersection of sets as a Set.
<pre>Interval(start, end[, left_open, right_open])</pre>	Represents a real interval as a Set.
Inverse(mat[, exp])	The multiplicative inverse of a matrix expression
InverseCosineTransform(*args)	Class representing unevaluated inverse cosine trans-
, , , , , , , , , , , , , , , , , , ,	forms.
<pre>InverseFourierTransform(*args)</pre>	Class representing unevaluated inverse Fourier trans-
( <i>b</i> /	forms.
InverseHankelTransform(*args)	Class representing unevaluated inverse Hankel trans-
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	forms.
<pre>InverseLaplaceTransform(*args)</pre>	Class representing unevaluated inverse Laplace trans-
•	forms.
<pre>InverseMellinTransform(*args)</pre>	Class representing unevaluated inverse Mellin trans-
· · · · · ·	forms.
<pre>InverseSineTransform(*args)</pre>	Class representing unevaluated inverse sine transforms.
<pre>KroneckerDelta(i, j[, delta_range])</pre>	The discrete, or Kronecker, delta function.
<pre>KroneckerProduct(*args[, check])</pre>	The Kronecker product of two or more arguments.
Lambda(signature, expr)	Lambda(x, expr) represents a lambda function similar to
	Python's 'lambda x: expr'.
LambertW(x[, k])	The Lambert W function \$W(z)\$ is defined as the in-
	verse function of \$w exp(w)\$ [1]
LaplaceTransform(*args)	Class representing unevaluated Laplace transforms.
Le	alias of LessThan
LessThan(lhs, rhs, **options)	Class representations of inequalities.
LeviCivita(*args)	Represent the Levi-Civita symbol.
Li(z)	The offset logarithmic integral.
Limit(e, z, z0[, dir])	Represents an unevaluated limit.
Line(*args, **kwargs)	An infinite line in space.
Line2D(p1[, pt, slope])	An infinite line in space 2D.
Line3D(p1[, pt, direction_ratio])	An infinite 3D line in space.
Lt	alias of StrictLessThan
<pre>MatAdd(*args[, evaluate, check, _sympify])</pre>	A Sum of Matrix Expressions
<pre>MatMul(*args[, evaluate, check, _sympify])</pre>	A product of matrix expressions
<pre>MatPow(base, exp[, evaluate])</pre>	
Matrix	alias of MutableDenseMatrix
MatrixBase()	Base class for matrix objects.
<pre>MatrixExpr(*args, **kwargs)</pre>	Superclass for Matrix Expressions
<pre>MatrixPermute(mat, perm[, axis])</pre>	Symbolic representation for permuting matrix rows or
	columns.
<pre>MatrixSlice(parent, rowslice, colslice)</pre>	A MatrixSlice of a Matrix Expression
<pre>MatrixSymbol(name, n, m)</pre>	Symbolic representation of a Matrix object
Max(*args)	Return, if possible, the maximum value of the list.
MellinTransform(*args)	Class representing unevaluated Mellin transforms.
Min(*args)	Return, if possible, the minimum value of the list.
Mod(p, q)	Represents a modulo operation on symbolic expressions.
Monomial(monom[, gens])	Class representing a monomial, i.e. a product of powers.
<pre>Mul(*args[, evaluate, _sympify])</pre>	Expression representing multiplication operation for al-
	gebraic field.
<pre>MutableDenseMatrix(*args, **kwargs)</pre>	
<pre>MutableDenseNDimArray([iterable, shape])</pre>	

Table 65 – continued from previous page

MutableMatrix	alias of MutableDenseMatrix
MutableSparseMatrix(*args, **kwargs)	
inconstruction ( ungs, in ungs)	
<pre>MutableSparseNDimArray([iterable, shape])</pre>	
2 2 2 2 7 1 37	
NDimArray(iterable[, shape])	N-dimensional array.
Nand(*args)	Logical NAND function.
Ne	alias of Unequality
Nor(*args)	Logical NOR function.
Not(arg)	Logical Not function (negation)
Number(*obj)	Represents atomic numbers in SymPy.
NumberSymbol()	
0	alias of Order
OmegaPower(a, b)	Represents ordinal exponential and multiplication terms
	one of the building blocks of the Ordinal class.
OneMatrix(m, n[, evaluate])	Matrix whose all entries are ones.
Options(gens, args[, flags, strict])	Options manager for polynomial manipulation module.
Or(*args)	Logical OR function
Order(expr, *args, **kwargs)	Represents the limiting behavior of some function.
Ordinal(*terms)	Represents ordinals in Cantor normal form.
Parabola([focus, directrix])	A parabolic GeometryEntity.
Permanent(mat)	Matrix Permanent
PermutationMatrix(perm)	A Permutation Matrix
Piecewise(*_args)	Represents a piecewise function.
Plane(p1[, a, b])	A plane is a flat, two-dimensional surface.
Point(*args, **kwargs)	A point in a n-dimensional Euclidean space.
Point2D(*args[, _nocheck])	A point in a 2-dimensional Euclidean space.
Point3D(*args[, _nocheck])	A point in a 3-dimensional Euclidean space.
Poly(rep, *gens, **args)	Generic class for representing and operating on polyno-
	mial expressions.
Polygon(*args[, n])	A two-dimensional polygon.
PolynomialRing(domain_or_ring[, symbols, order])	A class for representing multivariate polynomial rings.
Pow(b, e[, evaluate])	Defines the expression x**y as "x raised to a power y"
PowerSet(arg[, evaluate])	A symbolic object representing a power set.
Predicate(*args, **kwargs)	Base class for mathematical predicates.
Product(function, *symbols, **assumptions)	Represents unevaluated products.
ProductSet(*sets, **assumptions)	Represents a Cartesian Product of Sets.
PurePoly(rep, *gens, **args)	Class for representing pure polynomials.
<pre>PythonFiniteField(mod[, symmetric])</pre>	Finite field based on Python's integers.
PythonIntegerRing()	Integer ring based on Python's int type.
PythonRational	alias of PythonMPQ
QQ_gmpy	alias of GMPYRationalField
QQ_python	alias of PythonRationalField
Quaternion([a, b, c, d, real_field, norm])	Provides basic quaternion operations.
Range(*args)	Represents a range of integers.
	Represents rational numbers (p/q) of any size.
Rational(p[, q, gcd])	
Rational(p[, q, gcd]) RationalField()	Abstract base class for the domain QQ.
* - * * -:	Abstract base class for the domain QQ.  A Ray is a semi-line in the space with a source point and
RationalField()	
RationalField()	A Ray is a semi-line in the space with a source point and

Table 65 – continued from previous page

Table 65 – continue	ed from previous page
Ray3D(p1[, pt, direction_ratio])	A Ray is a semi-line in the space with a source point and
	a direction.
RealField([prec, dps, tol])	Real numbers up to the given precision.
RealNumber	alias of Float
RegularPolygon(c, r, n[, rot])	A regular polygon.
Rel	alias of Relational
Rem(p, q)	Returns the remainder when p is divided by q where p is
	finite and q is not equal to zero.
RisingFactorial(x, k)	Rising factorial (also called Pochhammer symbol [1]_)
	is a double valued function arising in concrete mathe-
	matics, hypergeometric functions and series expansions.
RootOf(f, x[, index, radicals, expand])	Represents a root of a univariate polynomial.
RootSum(expr[, func, x, auto, quadratic])	Represents a sum of all roots of a univariate polynomial.
Segment(p1, p2, **kwargs)	A line segment in space.
Segment2D(p1, p2, **kwargs)	A line segment in 2D space.
Segment3D(p1, p2, **kwargs)	A line segment in a 3D space.
SensitivityMatrix(sympification, duals,)	
SeqAdd(*args, **kwargs)	Represents term-wise addition of sequences.
SeqFormula(formula[, limits])	Represents sequence based on a formula.
SeqMul(*args, **kwargs)	Represents term-wise multiplication of sequences.
SeqPer(periodical[, limits])	Represents a periodic sequence.
Set(*args)	The base class for any kind of set.
Shi(z)	Sinh integral.
Si(z)	Sine integral.
Sieve()	An infinite list of prime numbers, implemented as a dy-
	namically growing sieve of Eratosthenes.
SineTransform(*args)	Class representing unevaluated sine transforms.
SingularityFunction(variable, offset, exponent)	Singularity functions are a class of discontinuous func-
ChangeMatrix	tions.
SparseMatrix SparseNDimArray(*args, **kwargs)	alias of MutableSparseMatrix
Spar Sendimarray( 'aigs, ''kwaigs)	
StrPrinter([settings])	
StrictGreaterThan(lhs, rhs, **options)	Class representations of inequalities.
StrictLessThan(lhs, rhs, **options)	Class representations of inequalities.
Subs(expr, variables, point, **assumptions)	Represents unevaluated substitutions of an expression.
Sum(function, *symbols, **assumptions)	Represents unevaluated summation.
Symbol(name, **assumptions)	Assumptions:
SymmetricDifference(a, b[, evaluate])	Represents the set of elements which are in either of the
,	sets and not in their intersection.
TableForm(data, **kwarg)	Create a nice table representation of data.
Trace(mat)	Matrix Trace
Transpose(*args, **kwargs)	The transpose of a matrix expression.
Triangle(*args, **kwargs)	A polygon with three vertices and three sides.
Tuple(*args, **kwargs)	Wrapper around the builtin tuple object.
Unequality(lhs, rhs, **options)	An unequal relation between two objects.
<pre>UnevaluatedExpr(arg, **kwargs)</pre>	Expression that is not evaluated unless released.
Union(*args, **kwargs)	Represents a union of sets as a Set.
<pre>Wild(name[, exclude, properties])</pre>	A Wild symbol matches anything, or anything without
	whatever is explicitly excluded.
	continues on next page

Table 65 – continued from previous page

WildFunction(*args)  A WildFunction function matches any function (with its arguments).  Xor(*args)  Logical XOR (exclusive OR) function.  Ymm(n, m, theta, phi)  Spherical harmonics defined as  ZZ_pmyp  alias of GMPYITHGE-RKing  ZZ_pmython  ZE_pmython  ZE_pmython  The Matrix Zero 0 - additive identity  Zmm(n, m, theta, phi)  Real spherical harmonics defined as  acos(arg)  acos(arg)  acos(arg)  acos(arg)  acos(arg)  acot(arg)  aliyaira(arg)  The Airy function Soperatorname[Ai] Sof the first kind.  airyabirime(arg)  The derivative Soperatorname[Bi] of the second kind.  airybirime(arg)  The derivative Soperatorname[Bi] of the second kind.  adre(n)  appellfi(a, b1, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  asec(arg)  The inverse secant function.  asec(arg)  The inverse she prebolic secant of x.  Returns the argument (in radians) of a complex number.  Accturns the argument (in radians) of a complex number.  Accturns the safety generalized Laguerre polynomial in secandary  assoc_laguerre(n, alpha, x)  assoc_laguerre(n, m, x) gives SP_n^m(x)S, where SS, and Sm, are the degree and order or an expression which is related to the nth order Legendre polynomial, SP_n(x)S in the following manner:  atan(arg)  The function atan2(y, x) computes operatorname(arg) yallowing manner:  atan(arg)  The function of the f	Table 6	65 – continued from previous page
Logical XOR (exclusive OR) function.	WildFunction(*args)	A WildFunction function matches any function (with its
Ymm(n, m, theta, phi)  Z_gmpy alias of GMPYIntegerRing alias of FythonIntegerRing  Z_python alias of FythonIntegerRing  Z_python alias of FythonIntegerRing  Z_pompton  Real spherical harmonics defined as acos(arg) The Matrix Zero 0 - additive identity  Zeroffatrix(m, n) Real spherical harmonics defined as acos(arg) The inverse cosine function. acosh(xg) acosh(xg) The inverse cosine function. acot(arg) The inverse costangent function. acot(arg) The Adri		arguments).
2Z_pmpy   alias of CPYIntegerRing	Xor(*args)	Logical XOR (exclusive OR) function.
ZZ_python	Ynm(n, m, theta, phi)	Spherical harmonics defined as
ZeroMartix(m, n)  Znm(n, m, theta, phi)  Znm(n, m, theta, phi)  Real spherical harmonics defined as acos(arg)  acos(arg)  acos(arg)  acot(arg)  adjoint(arg)  The inverse cosecant function.  alryaiprime(arg)  The derivative Soperatorname [Ai] So fithe first kind.  airyaiprime(arg)  The Airy function Soperatorname [Bi] So fithe second kind.  airybiarg  The Airy function Soperatorname [Bi] So fithe second kind.  airybiprime(arg)  The derivative Soperatorname [Bi] So fithe second kind.  adre(n)  appellf1(a, b1, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  Returns the argument (in radians) of a complex number.  asec(arg)  asec(arg)  asech(arg)  asech(x) is the inverse hyperbolic secant of x.  The inverse secant function.  asech(arg)  asinh(arg)  asinh(arg)  asinh(arg)  asinh(arg)  asinh(arg)  asinh(x) is the inverse hyperbolic sine of x.  Returns the SnSth generalized Laguerre polynomial in S.S.S.L.(ax).  assoc_legendre(n, m, x) gives SP_n^nm(x)S, where SnS and SmS are the degree and order or an expression which is related to the nth order Legendre polynomial, SP_n(x)S in the following manner:  atana(arg)  The inverse tangent function.  atana(arg)  The inverse tangent function.  atanh(x) is the inverse hyperbolic tangent of x.  Bell numbers / Bell polynomials / Bernoulli function  bessel (nu, z)  Benoulli (ni, x))  Benoulli numbers / Bernoulli polynomials / Bernoulli function  bessel (nu, z)  Bessel function of the first k	ZZ_gmpy	alias of GMPYIntegerRing
Zmm(n, m, theta, phi)       Real spherical barmonics defined as acos(arg)         acos(arg)       The inverse cosine function.         acos(arg)       acos(x)         acot(arg)       acot(x)         acot(arg)       acot(x)         acot(arg)       acot(x)         acot(arg)       acot(x)         acsch(arg)       acot(x)         airyai(arg)       The Airy function Soperatorname [Ai]S of the first kind.         airyai(arg)       The Airy function Soperatorname [Ai]PorineS of the Airy function of the first kind.         airybiprime(arg)       The derivative Soperatorname [Bi]S of the second kind.         airybiprime(arg)       The derivative Soperatorname [Bi]S of the second kind.         airybiprime(arg)       The inverse scant function         airybiprime(arg)       The inverse inverse function         airybiprime(arg)       The inverse the function         airybiprime(arg)       The inverse the function         airybiprime(arg)       The inverse the function	ZZ_python	alias of PythonIntegerRing
acos(arg) acos(arg) acos(arg) acos(arg) acos(arg) acot(arg) acot(a	ZeroMatrix(m, n)	The Matrix Zero 0 - additive identity
acosh(arg) acosh(arg) acot(arg) argualtranspose or Hermite conjugation. alryai(arg) The Airy function Soperatorname [Ai]'s of the Airy function of the first kind. alrybi(arg) The Airy function Soperatorname [Bi]'s of the Second kind. airybi(arg) acot(arg)	Znm(n, m, theta, phi)	Real spherical harmonics defined as
acosh(arg) acosh(arg) acot(arg) argualtranspose or Hermite conjugation. alryai(arg) The Airy function Soperatorname [Ai]'s of the Airy function of the first kind. alrybi(arg) The Airy function Soperatorname [Bi]'s of the Second kind. airybi(arg) acot(arg)	acos(arg)	The inverse cosine function.
acot(arg) acot(arg) acot(x) ac	· •	acosh(x) is the inverse hyperbolic cosine of x.
acoth(xg) acsch(xg) acsch(	· •	• • • • • • • • • • • • • • • • • • • •
acsc(arg) acsch(arg) acsch(x) is the inverse hyperbolic cosecant of x. Conjugate transpose or Hermite conjugation. airyai(arg) The Airy function Soperatorname [Ai] \$ of the first kind. airyaiprime(arg) The Airy function Soperatorname [Ai] \$ of the first kind. airyaiprime(arg) The derivative Soperatorname [Bi] \$ of the second kind. airybi(arg) The Airy function \$ operatorname [Bi] \$ of the second kind. airybiprime(arg) The Airy function \$ operatorname [Bi] \$ of the second kind. airybiprime(arg) The derivative \$ operatorname [Bi] \$ of the Airy function of the first kind. andre(n) Andre numbers / Andre function appellf1(a, b1, b2, c, x, y) This is the Appell hypergeometric function of two variables as: arg(arg) Returns the argument (in radians) of a complex number. The inverse secant function. assin(arg) asech(arg) The inverse secant function. asinh(arg) assinh(arg) assinh(arg) assinh(arg) assoc_laguerre(n, alpha, x) Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$ s n(x) \$ is the inverse hyperbolic sine of x.  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$ s n(x) \$ is the inverse hyperbolic sine of x.  assoc_legendre(n, m, x) assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner: atan(arg) The inverse tangent function. atan2(y, x) The function atan2(y, x) computes operatorname(aln1(x)x) taking two arguments y and x. atanh(arg) atanh(x) is the inverse hyperbolic tangent of x. bel1(n1, k_sym, symbols) Bel1 numbers / Bel1 polynomials / Bernoulli function bessel1(nu, z) Bessel1(nu, z) Bessel2(nu, z) Bessel2(nu, z) Bessel3 function of the first kind. bessel4(x, y) The beta integral is called the Eulerian integral of the first kind by Legendre: betainc_*args) The Generalized Regularized Incomplete Beta function is defined as betainc_*args)	. •	
acsch(arg) adjoint(arg) Conjugate transpose or Hermite conjugation. airyai(arg) The Airy function Soperatorname {Ai}^prime\$ of the Airy function soperatorname {Ai}^prime\$ of the Airy function of the first kind. airyaiprime(arg) The Airy function Soperatorname{Bi}^s of the second kind. airybi(arg) The Airy function Soperatorname{Bi}^s of the second kind. airybi(arg) The derivative Soperatorname{Bi}^o fithe Airy function of the first kind. andre(n) Andre numbers / Andre function appellf1(a, b1, b2, c, x, y) This is the Appell hypergeometric function of two variables as: arg(arg) Returns the argument (in radians) of a complex number. asec(arg) Asech(arg) Asech(arg) Asech(arg) Asech(arg) Assoc_laguerre(n, alpha, x) The inverse scant function. assoc_laguerre(n, alpha, x) Assoc_laguerre(n, alpha, x) Assoc_laguerre(n, m, x) Assoc_legendre(n,		**
adjoint(arg) Conjugate transpose or Hermite conjugation. airyai(arg) The Airy function Soperatorname{Ai}^orprime\$ of the Airy function of the first kind. airybi(arg) The Airy function Soperatorname{Bi}} of the Second kind. airybi(arg) The Airy function Soperatorname{Bi}} of the Second kind. airybi(arg) The Airy function Soperatorname{Bi}} of the Second kind. airybi(arg) The derivative Soperatorname{Bi}^orprime\$ of the Airy function of the first kind. airybiprime(arg) The derivative Soperatorname{Bi}^orprime\$ of the Airy function of the first kind. Andre numbers / Andre function appellf1(a, b1, b2, c, x, y) This is the Appell hypergeometric function of two variables as: arg(arg) Returns the argument (in radians) of a complex number. The inverse secant function. assec(arg) The inverse sine function. assen(arg) The inverse sine function. asinh(arg) The inverse sine function. asinh(arg) The inverse sine function. assoc_laguerre(n, alpha, x) SxS, SL_n(x)S. assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)S, where \$S_h\$ and \$S_h\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)S in the following manner: atan(arg) The inverse tangent function. atan(arg) The inverse tangent function. atan(arg) The inverse tangent function.  atan(arg) The function atan2(y, x) computes operatorname{tanh(x)} taking two arguments y and x. atanh(arg) Bernoulli(n[, x]) Bernoulli(n[, x]) Bernoulli numbers / Bernoulli polynomials / Bernoulli function  bessel(nn, z) Bessel function of the first kind. bessel(nn, z) Bessel function of the first kind. bessel(nn, z) Bessel function of the second kind. bessel(nn, z) The derivative \$S_orproprime* of the first kind by Legendre: betain(-*args) The Generalized Regularized Incomplete Beta function is defined as betain(-*args)	· · ·	
airyai(arg) The Airy function Soperatorname [Ai] sof the first kind. airyaiprime(arg) The derivative Soperatorname [Ai] primes of the Airy function of the first kind. airybi(arg) The Airy function Soperatorname [Bi] of the second kind. airybiprime(arg) The derivative Soperatorname [Bi] primes of the Airy function of the first kind. andre(n) Andre numbers / Andre function appellf1(a, b1, b2, c, x, y) This is the Appell hypergeometric function of two variables as: arg(arg) Returns the argument (in radians) of a complex number. asec(arg) The inverse secant function. assech(arg) asech(x) is the inverse hyperbolic secant of x. The inverse sine function. asin(arg) asinh(x) is the inverse hyperbolic sine of x. assoc_laguerre(n, alpha, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg) atan(x) The function atan2(y, x) computes operatorname[atan](y/x) taking two arguments y and x. atanh(arg) atanh(x) is the inverse hyperbolic tangent of x. bel1(n[, k_sym, symbols]) Bell numbers / Bell polynomials   Bernoulli numbers / Bernoulli polynomials / Bernoulli function bessel (nu, z) Bessel function of the first kind. bessel y(nu, z) Bessel function of the second kind. bessel y(nu, z) Bessel function of the second kind. bessel y(nu, z) Bessel function of the second kind. bessel y(nu, z) The beta integral is called the Eulerian integral of the first kind by Legendre: betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args)		
airyaiprime(arg)  The derivative Soperatorname{Ai}^primeS of the Airy function of the first kind.  airybi(arg)  The Airy function Soperatorname{Bi}\S of the second kind.  airybiprime(arg)  The derivative Soperatorname{Bi}\prime\ of the Airy function of the first kind.  andre(n)  Andre numbers / Andre function  appellf1(a, b1, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  Asec(arg)  The inverse secant function.  asec(arg)  The inverse secant function.  asin(arg)  asin(arg)  The inverse sine function.  asin(arg)  asin(arg)  asinh(arg)  asinh(x) is the inverse hyperbolic sine of x.  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$x\$, \$x\$, \$x\$, \$x\$, \$x\$, \$x\$, \$x\$		
function of the first kind.  airybi(arg)  The Airy function Soperatorname{Bi}\$ of the second kind.  airybiprime(arg)  The derivative Soperatorname{Bi}^prime\$ of the Airy function of the first kind.  Andre numbers / Andre function  appellf1(a, b1, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  Returns the argument (in radians) of a complex number.  asec(arg)  The inverse secant function.  asech(arg)  asech(x) is the inverse hyperbolic secant of x.  asin(arg)  asin(arg)  asin(x) is the inverse hyperbolic sine of x.  assoc_laguerre(n, alpha, x)  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x)  assoc_legen		• • • • • • • • • • • • • • • • • • • •
airybiprime(arg)  airybiprime(arg)  The derivative <code>Soperatorname{Bi}S</code> of the second kind.  The derivative <code>Soperatorname{Bi}prime</code> of the Airy function of the first kind.  andre(n)  Andre numbers / Andre function  appellf1(a, bl, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  Returns the argument (in radians) of a complex number.  asec(arg)  The inverse secant function.  asenh(arg)  asinh(arg)  The inverse sine function.  asinh(arg)  asinh(x) is the inverse hyperbolic secant of x.  Returns the <code>SnSth</code> generalized Laguerre polynomial in <code>\$x\$, SL_n(x)S</code> .  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives <code>SP_n^m(x)S, where SnS</code> and SmS are the degree and order or an expression which is related to the nth order Legendre polynomial, SP_n(x)S in the following manner:  atan(arg)  atan(arg)  atan(x)  atan(x)  atanh(x)  atanh(x)  bell(n[, k, sym, symbols))  bell(n[, k, sym, symbols))  bernoulli(n[, x])  besselj(nu, z)  besselj(nu, z)  bessell(nu, z)  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	,	
kind.  The derivative Soperatorname [Bi]^primeS of the Airy function of the first kind.  andre(n)	airvbi(arg)	
airybiprime(arg)  The derivative \$operatorname{Bi}^prime\$ of the Airy function of the first kind.  Andre numbers / Andre function  appellf1(a, bl, b2, c, x, y)  This is the Appell hypergeometric function of two variables as:  arg(arg)  Returns the argument (in radians) of a complex number.  asec(arg)  The inverse secant function.  asech(arg)  asinh(arg)  asinh(arg)  asinh(x) is the inverse hyperbolic secant of x.  asin(arg)  assoc_laguerre(n, alpha, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$n\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n , k_sym, symbols )  Bell numbers / Bell polynomials  bernoulli(n , x )  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besselj(nu, z)  Modified Bessel function of the first kind.  besselk(nu, z)  Bessel function of the second kind.  besselk(nu, z)  The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	, 5 <b>-</b> (418)	* * *
function of the first kind.  andre(n)  Andre numbers / Andre function  appellf1(a, b1, b2, c, x, y)  This is the Appel hypergeometric function of two variables as:  arg(arg)  Returns the argument (in radians) of a complex number.  asec(arg)  The inverse secant function.  asech(arg)  asech(x) is the inverse hyperbolic secant of x.  asinh(arg)  asinh(arg)  asinh(x) is the inverse hyperbolic sine of x.  Beturns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x)  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  The inverse tangent function.  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(x)  bell(n[, k_sym, symbols])  bell(n[, k_sym, symbols])  bell(n[, k_sym, symbols])  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  bessel(nu, z)  bessel(nu, z)  Bessel function of the first kind.  bessel(nu, z)  Bessel function of the second kind.  bessel(nu, z)  Bessel function of the second kind.  bessel(nu, z)  Bessel function of the second kind.  bessel(nu, z)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	airvbiprime(arg)	
andre(n) appellf1(a, b1, b2, c, x, y) This is the Appell hypergeometric function of two variables as:  arg(arg) Returns the argument (in radians) of a complex number.  Asec(arg) The inverse secant function.  asech(arg) asech(arg) asech(arg) asech(arg)  asech(arg)  asech(arg)  asech(arg)  asech(arg)  asech(x) is the inverse hyperbolic secant of x.  The inverse sine function.  asinh(arg) asinh(arg) asinh(x) is the inverse hyperbolic sine of x.  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_laguerre(n, alpha, x)  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg) atan(arg) atan(y, x) The function atan2(y, x) computes operatorname[atan](y/x) taking two arguments y and x.  atan(arg) atanh(arg) atanh(x) is the inverse hyperbolic tangent of x.  bell(n , k_sym, symbols ) Bell numbers / Bell polynomials  bernoulli(n , x ) Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besselj(nu, z) Bessel function of the first kind.  bessell(nu, z) Bessel function of the first kind.  besselk(nu, z) Bessel function of the second kind.  besselk(nu, z) Bessel function of the second kind.  beta(x , y ) The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Incomplete Beta function is given by	411)	
appellf1(a, b1, b2, c, x, y)  arg(arg)  Returns the argument (in radians) of a complex number. asec(arg)  The inverse secant function.  asech(arg)  asech(arg)  asech(arg)  asech(arg)  asin(arg)  asin(arg)  asin(arg)  asin(arg)  asin(arg)  asin(arg)  assoc_laguerre(n, alpha, x)  assoc_laguerre(n, alpha, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan(arg)  atan(x)  atanh(arg)  atanh(x)  bell(n(, k_sym, symbols))  Bell numbers / Bell polynomials  bernoulli(n(, x))  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  bessell(nu, z)  Modified Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  betainc(*args)  The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is gliven by	andre(n)	
arg(arg) Returns the argument (in radians) of a complex number. asec(arg) The inverse secant function. asech(arg) asech(x) is the inverse hyperbolic secant of x. asin(arg) The inverse sine function. asinh(arg) asinh(x) is the inverse hyperbolic sine of x. assoc_laguerre(n, alpha, x) Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner: atan(arg) The inverse tangent function. atan2(y, x) The function atan2(y, x) computes operator name[atan](y/x) taking two arguments y and x. atanh(arg) atanh(x) is the inverse hyperbolic tangent of x. bell(n[, k_sym, symbols]) Bell numbers / Bell polynomials bernoulli(n[, x]) Bernoulli numbers / Bernoulli polynomials / Bernoulli function besseli(nu, z) Bessel function of the first kind. bessely(nu, z) Bessel function of the second kind. bessely(nu, z) Bessel function of the second kind. beta(x[, y]) The beta integral is called the Eulerian integral of the first kind by Legendre: betainc_*args) The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by		
arg(arg) asec(arg) The inverse secant function. asech(arg) asech(arg) asech(arg) asech(arg) asech(arg) The inverse secant function. asinh(arg) asinh(arg) asinh(arg) asinh(x) is the inverse hyperbolic secant of x.  The inverse sine function. asinh(arg) asinh(x) is the inverse hyperbolic sine of x.  Beturns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_laguerre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg) The inverse tangent function.  atan2(y, x) The function atan2(y, x) computes operator-name[atan](y/x) taking two arguments y and x.  atanh(arg) atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols]) Bell numbers / Bell polynomials bernoulli(n[, x]) Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z) Bessel function of the first kind.  bessely(nu, z) Bessel function of the second kind.  bessely(nu, z) Bessel function of the second kind.  betainc(*args) The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by	appell11(a, 01, 02, c, n, y)	
asec(arg) asech(arg) asech(arg) asech(x) is the inverse hyperbolic secant of x.  asin(arg) asin(arg) asin(x) is the inverse hyperbolic sine of x.  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(arg)  atan(x)  bell(n[, k_sym, symbols])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bessel function of the first kind.  bessel(nu, z)  Modified Bessel function of the first kind.  bessely(nu, z)  Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	arg(arg)	
asech(arg) asin(arg) The inverse sine function. asinh(arg) assoc_laguerre(n, alpha, x) assoc_legendre(n, m, x) assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  The inverse tangent function.  atan2(y, x) The function atan2(y, x) computes operatorname(atan)(y/x) taking two arguments y and x.  atanh(arg) atanh(arg) atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols]) Bell numbers / Bell polynomials bernoulli(n[, x]) Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z) Modified Bessel function of the first kind.  besselj(nu, z) Bessel function of the first kind.  bessely(nu, z) Bessel function of the second kind.  bessely(nu, z) Bessel function of the second kind.  betain([, y]) The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args) The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by	- · · · ·	
asin(arg) asinh(arg) asinh(x) is the inverse hyperbolic sine of x.  assoc_laguerre(n, alpha, x)  assoc_legendre(n, m, x)  where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan(arg)  atan(x)  atanh(arg)  atanh(x)  bell(n[, k_sym, symbols])  bell(n[, k_sym, symbols])  bernoulli(n[, x])  Bernoulli numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  bessel(nu, z)  Modified Bessel function of the first kind.  bessel(nu, z)  Modified Bessel function of the second kind.  bessel(nu, z)  Bessel function of the second kind.  bessel(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Regularized Incomplete Beta function is given by	· •	
asinh(arg) assoc_laguerre(n, alpha, x)  Returns the \$n\$th generalized Laguerre polynomial in \$x\$, \$L_n(x)\$.  assoc_legendre(n, m, x) assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan2(y, x)  The function atan2(y, x) computes operator-name{atan}(y/x) taking two arguments y and x.  atanh(arg)  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  bell(n[, k_sym, symbols])  besseli(nu, z)  besseli(nu, z)  besselj(nu, z)  besselj(nu, z)  besselk(nu, z)  bessely(nu, z)  bessely(nu, z)  bessely(nu, z)  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		
assoc_laguerre(n, alpha, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan2(y, x)  The inverse tangent function.  atan2(y, x) computes operator-name{atan}(y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  besselj(nu, z)  Bessel function of the first kind.  besselk(nu, z)  Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		
assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  atan(2(y, x)  The inverse tangent function.  atanh(arg)  atanh(arg)  atanh(arg)  bell(n[, k_sym, symbols])  bell(n[, k_sym, symbols])  bernoulli(n[, x])  Bell numbers / Bell polynomials / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  besselj(nu, z)  Bessel function of the first kind.  bessely(nu, z)  Bessel function of the first kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Regularized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		* •
assoc_legendre(n, m, x)  assoc_legendre(n, m, x) gives \$P_n^m(x)\$, where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  The inverse tangent function.  atan2(y, x)  The function atan2(y, x) computes operator- name{atan}(y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  bessely(nu, z)  Bessel function of the first kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as  betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	assoc_ragaerre(ii, aipiia, x)	
where \$n\$ and \$m\$ are the degree and order or an expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg) The inverse tangent function.  atan2(y, x) The function atan2(y, x) computes operatorname[atan](y/x) taking two arguments y and x.  atanh(arg) atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols]) Bell numbers / Bell polynomials  bernoulli(n[, x]) Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z) Modified Bessel function of the first kind.  besselj(nu, z) Bessel function of the first kind.  besselk(nu, z) Modified Bessel function of the second kind.  bessely(nu, z) Bessel function of the second kind.  beta(x[, y]) The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	assoc legendre(n m x)	
expression which is related to the nth order Legendre polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  The inverse tangent function.  The function atan2(y, x) computes operatorname{atan}(y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  besselj(nu, z)  Bessel function of the first kind.  besselk(nu, z)  Modified Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	abboc_regenare(n, m, x)	
polynomial, \$P_n(x)\$ in the following manner:  atan(arg)  The inverse tangent function.  The function atan2(y, x) computes operator- name[atan](y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  bessely(nu, z)  Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is given by		
atan(arg)  atan2(y, x)  The function atan2(y, x) computes operator- name[atan](y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bell(n[, k_sym, symbols])  bernoulli(n[, x])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  besselj(nu, z)  Bessel function of the first kind.  besselk(nu, z)  Modified Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		1
atan2(y, x)  The function atan2(y, x) computes operator- name[atan](y/x) taking two arguments y and x.  atanh(arg)  atanh(x) is the inverse hyperbolic tangent of x.  bel1(n[, k_sym, symbols])  Bell numbers / Bell polynomials  bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  bessely(nu, z)  Modified Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is given by	atan(arg)	
$\begin{array}{c} \textit{name\{atan\}(y/x)$ taking two arguments $y$ and $x$.} \\ \textit{atanh(arg)} & \textit{atanh(x)} \text{ is the inverse hyperbolic tangent of } \textbf{x}. \\ \textit{bell(n[,k_sym, symbols])} & \textit{Bell numbers / Bell polynomials} \\ \textit{bernoulli(n[,x])} & \textit{Bernoulli numbers / Bernoulli polynomials / Bernoulli function} \\ \textit{besseli(nu, z)} & \textit{Modified Bessel function of the first kind.} \\ \textit{besselj(nu, z)} & \textit{Bessel function of the first kind.} \\ \textit{besselk(nu, z)} & \textit{Modified Bessel function of the second kind.} \\ \textit{bessely(nu, z)} & \textit{Bessel function of the second kind.} \\ \textit{beta(x[,y])} & \textit{The beta integral is called the Eulerian integral of the first kind by Legendre:} \\ \textit{betainc(*args)} & \textit{The Generalized Incomplete Beta function is defined as betainc_regularized(*args)} & \textit{The Generalized Regularized Incomplete Beta function is given by} \\ \end{array}$		
atanh(arg)atanh(x) is the inverse hyperbolic tangent of x.bell(n[, k_sym, symbols])Bell numbers / Bell polynomialsbernoulli(n[, x])Bernoulli numbers / Bernoulli polynomials / Bernoulli functionbesseli(nu, z)Modified Bessel function of the first kind.bessely(nu, z)Bessel function of the second kind.bessely(nu, z)Bessel function of the second kind.beta(x[, y])The beta integral is called the Eulerian integral of the first kind by Legendre:betainc(*args)The Generalized Incomplete Beta function is defined asbetainc_regularized(*args)The Generalized Regularized Incomplete Beta function is given by	$\mathcal{L}(\mathcal{G}, \mathcal{K})$	
bell(n[, k_sym, symbols])Bell numbers / Bell polynomialsbernoulli(n[, x])Bernoulli numbers / Bernoulli polynomials / Bernoulli functionbesseli(nu, z)Modified Bessel function of the first kind.besselj(nu, z)Bessel function of the first kind.besselk(nu, z)Modified Bessel function of the second kind.bessely(nu, z)Bessel function of the second kind.beta(x[, y])The beta integral is called the Eulerian integral of the first kind by Legendre:betainc(*args)The Generalized Incomplete Beta function is defined asbetainc_regularized(*args)The Generalized Regularized Incomplete Beta function is given by	atanh(arg)	
bernoulli(n[, x])  Bernoulli numbers / Bernoulli polynomials / Bernoulli function  besseli(nu, z)  Modified Bessel function of the first kind.  besselj(nu, z)  Bessel function of the first kind.  besselk(nu, z)  Modified Bessel function of the second kind.  bessely(nu, z)  Bessel function of the second kind.  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		•••
function besseli(nu, z) Modified Bessel function of the first kind. besselj(nu, z) Bessel function of the first kind. besselk(nu, z) Modified Bessel function of the second kind. bessely(nu, z) Bessel function of the second kind. beta(x[, y]) The beta integral is called the Eulerian integral of the first kind by Legendre: betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by		
besseli(nu, z)  besselj(nu, z)  besselk(nu, z)  besselk(nu, z)  bessely(nu, z)  bessely(nu, z)  beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	Definoutif(II[, A])	
besselj(nu, z)  Bessel function of the first kind.  Modified Bessel function of the second kind.  Bessely(nu, z)  Bessel function of the second kind.  Bessel function of the second kind.  Bessel function of the second kind.  The beta integral is called the Eulerian integral of the first kind by Legendre:  Betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	hesseli(nu z)	
$\begin{array}{lll} besselk(nu,z) & Modified Bessel function of the second kind. \\ bessely(nu,z) & Bessel function of the second kind. \\ beta(x[,y]) & The beta integral is called the Eulerian integral of the first kind by Legendre: \\ betainc(*args) & The Generalized Incomplete Beta function is defined as \\ betainc\_regularized(*args) & The Generalized Regularized Incomplete Beta function is given by \\ \end{array}$		
bessely(nu, z)  Bessel function of the second kind.  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		
beta(x[, y])  The beta integral is called the Eulerian integral of the first kind by Legendre:  betainc(*args)  The Generalized Incomplete Beta function is defined as betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by		
first kind by Legendre: betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by		
betainc(*args) The Generalized Incomplete Beta function is defined as betainc_regularized(*args) The Generalized Regularized Incomplete Beta function is given by	υς εα(λ[, γ])	
betainc_regularized(*args)  The Generalized Regularized Incomplete Beta function is given by	hatainc(*args)	· · · · · · · · · · · · · · · · · · ·
is given by		
· · · · · · · · · · · · · · · · · · ·	becariic_regurarizeu( args)	· · · · · · · · · · · · · · · · · · ·
		continues on next page

Table 65 – continued from previous page

Table 65 – continue	d from previous page
binomial(n, k)	Implementation of the binomial coefficient.
carmichael(*args)	Carmichael Numbers:
cartes	alias of product
catalan(n)	Catalan numbers
ceiling(arg)	Ceiling is a univariate function which returns the small-
	est integer value not less than its argument.
chebyshevt(n, x)	Chebyshev polynomial of the first kind, $T_n(x)$ .
$chebyshevt\_root(n, k)$	<pre>chebyshev_root(n, k) returns the \$k\$th root (in-</pre>
	dexed from zero) of the \$n\$th Chebyshev polynomial of
	the first kind; that is, if $0 le k < n$ , chebyshevt(n,
	chebyshevt_root(n, k)) == $0$ .
chebyshevu(n, x)	Chebyshev polynomial of the second kind, $U_n(x)$ .
<pre>chebyshevu_root(n, k)</pre>	<pre>chebyshevu_root(n, k) returns the \$k\$th root (in-</pre>
	dexed from zero) of the \$n\$th Chebyshev polynomial of
	the second kind; that is, if $0 \le k < n$ , chebyshevu(n,
	chebyshevu_root(n, k)) == $0$ .
conjugate(arg)	Returns the <i>complex conjugate</i> [1]_ of an argument.
cos(arg)	The cosine function.
cosh(arg)	cosh(x) is the hyperbolic cosine of x.
cot(arg)	The cotangent function.
coth(arg)	coth(x) is the hyperbolic cotangent of x.
csc(arg)	The cosecant function.
csch(arg)	csch(x) is the hyperbolic cosecant of x.
defaultdict	defaultdict(default_factory=None, /, [])> dict with
	default factory
digamma(z)	The digamma function is the first derivative of the
	loggamma function
dirichlet_eta(s[, a])	Dirichlet eta function.
<pre>divisor_sigma(n[, k])</pre>	Calculate the divisor function $sigma\_k(n)$ for positive integer $n$
elliptic_e(m[, z])	teger n Called with two arguments \$z\$ and \$m\$, evaluates the
elliptic_e(m[, z])	incomplete elliptic integral of the second kind, defined
	by
$elliptic_f(z, m)$	The Legendre incomplete elliptic integral of the first
elliptic_i(z, iii)	kind, defined by
elliptic_k(m)	The complete elliptic integral of the first kind, defined
CIIIPCIC_k(III)	by
elliptic_pi(n, m[, z])	Called with three arguments \$n\$, \$z\$ and \$m\$, evalu-
	ates the Legendre incomplete elliptic integral of the third
	kind, defined by
erf(arg)	The Gauss error function.
erf2(x, y)	Two-argument error function.
erf2inv(x, y)	Two-argument Inverse error function.
erfc(arg)	Complementary Error Function.
erfcinv(z)	Inverse Complementary Error Function.
erfi(z)	Imaginary error function.
erfinv(z)	Inverse Error Function.
euler(n[, x])	Euler numbers / Euler polynomials / Euler function
exp(arg)	The exponential function, $e^x$ .
exp_polar(*args)	Represent a polar number (see g-function Sphinx docu-
	mentation).
expint(nu, z)	Generalized exponential integral.
	continues on next page

Table 65 – continued from previous page

factorial(n)	Table 65 – continue	d from previous page
factorial2(arg)  ff fibonacci (n[, sym])  fibonacci (n[, sym])  fibonacci (n[, sym])  fibonacci numbers / Fibonacci polynomials  floor(arg)  floor(arg)  floor is a univariate function which returns the largest integer value not greater than its argument.  frac(arg)  fresnelc(x)  fresnelc(x)  fresnel integral C.  fresnel integral S.  gamma(arg)  gegenbauer(n, a, x)  gegenbauer(n, a, x)  gegenbauer(n, a, x)  genocchi(n[, x])  function  hankel1(nu, z)  hankel2(nu, z)  hankel2(nu, z)  hankel2(nu, z)  harmonic(n, m])  harmonic(n, m])  harmonic(n, m)  harmonic(n, m)  harmonic(n, m)  harmonic (n)  hankel prob(n, x) gives the Sn5th Hermite polynomial in Sx5, SH_n(x)S.  hermite_prob(n, x)  sys, SH_n(x)S.  hermite_prob(n, x) gives the Sn5th probabilist's Hermite polynomial in Sx5, SHe_n(x)S.  hal(nu, z)  hyper(ap, bq, z)  The generalized phyergeometric function is defined by a series where the ratios of successive terms are a rational function of the second kind.  happer(ap, bq, z)  Jacobi (n, a, b, x)  Jacobi (n, a, b, x)  Jacobi (n, a, b, x)  legendre(n, x)  leg	factorial(n)	Implementation of factorial function over nonnegative
ff fibonacci (n[, sym]) firesnel(c() frac(arg) fresnel(c() fresnel(c() fresnel(c() fresnel(c() fresnel(c() fresnel(c() fresnel(n], x) gembauer (n, a, x) gembauer (n, a, x) gembauer (n, a, x) gemocchi (n[, x]) Genocchi numbers / Genocchi polynomials / Genocchi numbers / Genocchi polynomials / Genocchi function hankell(nu, z) hankell(nu, z) hankell(nu, z) harmonic(n[, m]) harmonic(n, m]) harmonic(n, m]) hermite(n, x) Spherical Hankel function of the first kind. hn2(nu, z) hn2(nu, z) hn2(nu, z) hn3(nu, z) hn4(nu, z) hn4(nu, z) hn5(nu, z) hn5(nu, z) hn6(nu, z) hn6(nu, z) hn7(nu, z) hn8(nu, z) hn9(nu,		integers.
Fibonacci (nl, syml) Fibonacci numbers / Fibonacci polynomials floor(arg) Fibonacci numbers / Fibonacci polynomials floor(arg) From is a univariate function which returns the largest integer value not greater than its argument. frac(arg) Fresnels(2) Fresnels(2) Fresnels(3) Fresnel integral C. Fresnel integral C. Fresnel integral S. gamma(arg) Fresnel integral S. gamma(arg) Fresnel integral S. genocchi (nl, xl) Genocchi numbers / Genocchi polynomials / Genocchi numbers / Genocchi polynomials / Genocchi numbers / Genocchi polynomials / Genocchi numbers / Hankel function of the first kind. hankel2(nu, z) Hankel function of the first kind. harmonic(nl, ml) Harmoric numbers hermite(n, x) Sx, SH_n(x)S. hermite_prob(n, x) Hermite_prob(n, x) Hermite_prob(n, x) Hermite_prob(n, x) Hermite_prob(n, x) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the second kind. hn2(nu, z) Free generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.  In(arg) Returns imaginary part of expression. Jacobi polynomial SP_n^{(left(alpha, betaright))}(x)S. Spherical Bessel function of the first kind.  Returns the SnSth Laguerre polynomial in SxS, SP_n(x)S Legendre(n, x) Legendre(n, x) Jacobi polynomial SP_n^{(left(alpha, betaright))}(x)S. The classical logarithmic integral.  In alias of log The classical logarithmic integral.  In alias of log The natural logarithmic integral.  In alias of log The natural logarithmic integral.  In Genorative SC (prime) (a, 2,2)S of the Mathieu Cosine function (e.e., SlogGamma(x)S).  The Mathieuc(a, q. z) The Mathieuc Sc (prime) (a, 2,2)S of the Mathieu Co	factorial2(arg)	The double factorial $n!!$ , not to be confused with $(n!)!$
Floor(arg)  Floor(arg)  Floor is a univariate function which returns the largest integer value not greater than its argument.  Represents the fractional part of x  Fresnel.c(x)  Fresnel integral C.  Fresnel integral S.  gamma(arg)  geopenbauer(n, a, x) geopenbauer(n, a, x) Geonocchi numbers / Genocchi polynomials / Genocchi function  hankel.l(nu, z)  hankel.l(nu, z)  hankel.l(nu, z)  hankel.l(nu, z)  hankel.prob(n, x)  hermite(n, x)  hermite(n, x)  hermite(n, x)  hermite(n, x)  hermite(n, x)  spherical Hankel function of the first kind.  hanz(nu, z)  halmal function of the second kind.  harmonic numbers  hermite(n, x)  hermite(n, x)  spherical Hankel function of the first kind.  hanz(nu, z)  halmal function of the second kind.  hermite(n, x)  hermite(n, x)  spherical Hankel function of the first kind.  hanz(nu, z)  Spherical Hankel function of the first kind.  hanz(nu, z)  Spherical Hankel function of the first kind.  hanz(nu, z)  Spherical Hankel function of the first kind.  hyper(ap, bq, z)  The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.  im(arg)  Returns imaginary part of expression.  jacobi (n, a, b, x)  jurity  Spherical Bessel function of the first kind.  Returns the SnSth Laguerre polynomial in SxS, St_n(x)S.  legendre(n, x)  elgendre(n, x)  elgen	ff	alias of FallingFactorial
integer value not greater than its argument.  frac(arg) fresnelc(z) fresnels(z) fresnels(z) fresnels(z) fresnels(z) fresnels(z) fresnels(z) fresnels(z) fresnelintegral C. fresnel integral C. fresnel integral S. gamma(arg) gementa function gegenbauer(n, a, x) genocchi(n , x ) Genocchi numbers / Genocchi polynomials / Genocchi hankel1(nu, z) hankel1(nu, z) Hankel function of the first kind. hankel2(nu, z) Hankel function of the second kind. harmonic(n , m ) Harmonic numbers hermite(n, x) sxs, SH_n(x)S. hermite_prob(n, x) hermite_prob(n, x) hermite_prob(n, x) hermite_prob(n, x) hermite_prob(n, x) sysherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the first kind. hnyper(ap, bq, z) The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index. im(arg) jacobi(n, a, b, x) jacobi (n, a, b, x) spherical Bessel function of the first kind. laguerre(n, x) Returns imaginary part of expression. Jacobi polynomial SP_n^{left(alpha, betaright)  (x)S. Spherical Bessel function of the first kind. laguerre(n, x) gender(n, x) gender(n, x) gives the SnSth Laguerre polynomial in SxS, SL_n(x)S. legendre(n, x) gendre(n,	<pre>fibonacci(n[, sym])</pre>	Fibonacci numbers / Fibonacci polynomials
frac(arg) fresnelc(z) fresnelc(z) fresnels(z) fresnels	floor(arg)	Floor is a univariate function which returns the largest
fresnels(z) fresnelitegral S. fresnel integral S. frespending S. fresnel integral S. frespending S. fresnel integral S. frespending S. fresnel integral S. freegenbauer integral S. frespending S. fresnel integral S. frespending S. fresnel integral S. frespending S. fresnel integral S. freegenbauer integral S. fresnel integral S. freschiften internet integral S. french internet intention of the first kind. harkel function of the second kind. harmonic numbers harmite probonnial in \$x\$, S. Sherrical Hankel function of the second kind. harlon, a, b, x frespending in \$x\$, S. Hermite polynomial in \$x\$, Sherrical Hankel function of the second kind. harlon, x frespending in \$x\$, S. Sherrical Hankel function of the second kind. harlon, x frespending in \$x\$, S. Sherrical Hankel function of the first kind. harlon, x frespending in \$x\$, S. Sherrical Hankel function of the first kind. harlon, x frespending in \$x\$, S. Sherrical Hankel function of the first kind. harlon, x frespending in \$x\$, S. Sherrical Essel function of the first kind. harlon, x frespending in \$x\$, S. Sherrical Essel function of the first kind. harlon, x frespending in \$x\$, Sherrical		integer value not greater than its argument.
fresnels(z) gamma(arg) gamma(arg) gegenbauer(n, a, x) gegenbauer(n, a, x) genocchi(n[, x]) Genocchi numbers / Genocchi polynomial \$C_n^{left(alpharight)}(x)\$. genocchi(n[, x]) Genocchi numbers / Genocchi polynomials / Genocchi function hankel1(nu, z) Hankel function of the first kind. harwonic(n[, m]) Harmonic numbers hermite(n, x) hermite(n, x) hermite(n, x) hermite(n, x) hermite_prob(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$. hermite_prob(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$. hermite_prob(n, x) hermite_prob(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$. hermite_prob(n, x) gives the \$n\$th probabilist's Hermite polynomial in \$x\$, \$H_n(x)\$. hermite_prob(n, x)  The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index. im(arg) Returns imaginary part of expression. Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.  Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.  Jin(nu, z) Spherical Bessel function of the first kind. Returns the \$n\$th Laguerre polynomial in \$x\$, \$\$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$L_n(x)\$.  The classical logarithm function $ln(x)$ or $log(x)$ .  The classical logarithm function $ln(x)$ or $log(x)$ .  The classical logarithm function $ln(x)$ or $log(x)$ .  The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).  The Loggamma function implements the logarithm of the gamma function $ln(x)$ or $ln(x)$ or $ln(x)$ or $ln(x)$ or $ln(x)$ or $ln(x)$ or $ln$	frac(arg)	Represents the fractional part of x
gamma(arg) gegenbauer(n, a, x) gegenbauer polynomial \$C_n^{left(alpharight)](x)\$. genoccht(n[, x]) Gegenbauer polynomial \$C_n^{left(alpharight)](x)\$. genoccht(n[, x]) Genocchi numbers / Genocchi polynomials / Genocchi function hankel1(nu, z) Hankel function of the first kind. harmonic(n[, m]) Harmonic numbers hermite(n, x) hermite(n, x) hermite_prob(n, x) hermite(n, x) hermite(	fresnelc(z)	
gegenbauer(n, a, x)         Gegenbauer polynomial SC_n^{left(alpharight)}(x)S.           genocchi(n[, x])         Genocchi numbers / Genocchi polynomials / Genocchi numbers / Genocchi punction           hankel 1(nu, z)         Hankel function of the first kind.           hankel 2(nu, z)         Hankel function of the second kind.           harmonic(n[, ml])         Harmonic numbers           hermite(n, x)         hermite(n, x) gives the SnSth Hermite polynomial in SxS, SH_n(x)S.           hermite_prob(n, x)         hermite_prob(n, x) gives the SnSth probabilist's Hermite polynomial in SxS, SHE_n(x)S.           hn1(nu, z)         Spherical Hankel function of the first kind.           hn2(nu, z)         Spherical Hankel function of the first kind.           hn2(nu, z)         Spherical Hankel function of the second kind.           hyper(ap, bq, z)         The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.           im(arg)         Returns imaginary part of expression.           jacobi (n, a, b, x)         Jacobi oplynomial Sp_n^{left}[talpha, betaright)}(x)S.           jn(nu, z)         Spherical Bessel function of the first kind.           laguerre(n, x)         Returns the SnSth Laguerre polynomial in SxS, SL_n(x)S.           legendre(n, x)         legendre(n, x)         gives the SnSth Legendre polynomial in SxS, SL_n(x)S.           l	fresnels(z)	Fresnel integral S.
genocchi(n[, x])  Genocchi numbers / Genocchi polynomials / Genocchi function  hankel1(nu, z)  hankel2(nu, z)  hankel2(nu, z)  harmonic(n[, m])  hermite(n, x)  hermite(n, x)  hermite(n, x)  hermite_prob(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$.  hn1(nu, z)  hn2(nu, z)  Spherical Hankel function of the first kind.  hn2(nu, z)  Spherical Hankel function of the first kind.  hn2(nu, z)  Spherical Hankel function of the second kind.  hyper(ap, bq, z)  The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.  Returns imaginary part of expression.  Jacobi polynomial \$P_n^{n}[left(alpha, betaright)](x)\$.  Spherical Bessel function of the first kind.  Returns the \$n\$th Laguerre polynomial in \$x\$,  \$L_n(x)\$.  legendre(n, x)  spherical Bessel function of the first kind.  Returns the \$n\$th Laguerre polynomial in \$x\$,  \$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$  legendre(n, x)  spherical Bessel function integral.  alias of 1og  log(argl, basel)  The classical logarithmic integral.  alias of 1og  log(argl, basel)  The natural logarithm function ln(x) or log(x).  The loggamma(z)  The loggamma function implements the logarithm of the gamma function.  Lucas numbers  The Mareum Q-function.  The Ma	gamma(arg)	
function	gegenbauer(n, a, x)	
hankel1(nu, z) Hankel function of the first kind. hankel2(nu, z) Hankel function of the second kind. harmonic(n[, m]) hermite(n, x) hermite(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$. hermite_prob(n, x) hermite_prob(n, x) gives the \$n\$th probabilist's Hermite polynomial in \$x\$, \$H_n(x)\$. hn1(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the second kind. hyper(ap, bq, z) The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index. im(arg) Returns imaginary part of expression. jacobi(n, a, b, x) Jacobi polynomial \$P_n^{1}[eff(alpha, betaright)](x)\$. spherical Bessel function of the first kind. laguerre(n, x) Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$. legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$ lerchphi(*args) Lerch transcendent (Lerch phi function). li(z) The classical logarithm function ln(x) or log(x). loganma(z) The natural logarithm function ln(x) or log(x). loganma(z) The natural logarithm function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$). lowergamma(a, x) The lower incomplete gamma function. lucas(n) Lucas numbers marcum(m, a, b) The Marcum Q-function. mathieuc(a, q, z) The Marcum Q-function \$C(a,q,z)\$. mathieucprime(a, q, z) The Marcum Q-function \$C(a,q,z)\$. mathieusprime(a, q, z) The Marcum Q-function \$C(a,q,z)\$. mathieusprime(a, q, z) The Marcum Sine function \$C(a,q,z)\$. mathieusprime(a, q, z) The Marcum Q-function solone solone function. meijerg(*args) The Marcum O-function solone function solone function mathieus (a, q, z) The Marcum O-function mathieus Sine function solone function solone function mathieus sine function maps natural number to {-1, 0, 1} motzkin(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number	<pre>genocchi(n[, x])</pre>	Genocchi numbers / Genocchi polynomials / Genocchi
harmoni c(nt, m))       Harmonic numbers         hermite(n, x)       Harmonic numbers         hermite(n, x)       hermite(n, x) gives the \$n\$th Hermite polynomial in \$x\$, \$H_n(x)\$.         hermite_prob(n, x)       hermite_prob(n, x) gives the \$n\$th probabilist's Hermite polynomial in \$x\$, \$He_n(x)\$.         hn1(nu, z)       Spherical Hankel function of the first kind.         hn2(nu, z)       Spherical Hankel function of the second kind.         hyper(ap, bq, z)       The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.         im(arg)       Returns imaginary part of expression.         jacobi(n, a, b, x)       Jacobi polynomial \$P_n*{left(alpha, betaright)}(x)\$.         jn(nu, z)       Spherical Bessel function of the first kind.         laguerre(n, x)       Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$.         legendre(n, x)       legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$         lerchphi(*args)       Lerch transcendent (Lerch phi function).         li(2)       The classical logarithmic integral.         ln       alias of log         log(argl, base)       The natural logarithm function $ln(x)$ or $log(x)$ .         loggamma(z)       The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).         lower gamma(a, x)       The		
harmonic (n[, m]) hermite(n, x) hermite(n, x) sys, sh Ln(x)s. hermite_prob(n, x) gives the SnSth probabilist's Hermite polynomial in SxS, SHE_n(x)S. hn1(nu, z) Spherical Hankel function of the first kind. hn2(nu, z) Spherical Hankel function of the second kind. hyper(ap, bq, z) The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index. im(arg) Returns imaginary part of expression. jacobi(n, a, b, x) Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.  spherical Bessel function of the first kind. laguerre(n, x) Returns the \$nSth Laguerre polynomial in \$x\$, \$L_n(x)\$. legendre(n, x) legendre(n, x) legendre(n, x) gives the SnSth Legendre polynomial of \$x\$, \$P_n(x)\$ lerchphi(*args) Lerch transcendent (Lerch phi function). li(z) The classical logarithmic integral. ln alias of log log(argl, base]) The natural logarithm function ln(x) or log(x). The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$). lowergamma(a, x) The lower incomplete gamma function. lucas(n) Lucas numbers marcumq(m, a, b) The Marcum Q-function. mathieuc(a, q, z) The Marcum Q-function. mathieuc(a, q, z) The Marcum Q-function SC(a,q,z)\$. mathieusprime(a, q, z) The Marcum Q-function SC(a,q,z)\$ of the Mathieu Cosine function.  meijerg(*args) The Marcum Q-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform. mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number	hankel1(nu, z)	
hermite(n, x)		Hankel function of the second kind.
Sx\$, \$H_n(x)\$. hermite_prob(n, x) gives the \$n\$th probabilist's Hermite_prob(n, x) gives the \$n\$th probabilist's Hermite_prob(n, x) gives the \$n\$th probabilist's Hermite polynomial in \$x\$, \$He_n(x)\$. hn1(nu, z)	<pre>harmonic(n[, m])</pre>	
hermite_prob(n, x) hermite_prob(n, x) gives the \$n\$th probabilist's Hermite polynomial in \$x\$, \$He_n(x)\$. hn1(nu, z)	hermite(n, x)	
Hermite polynomial in \$x\$, \$He_n(x)\$.		
hn1(nu, z)	hermite_prob(n, x)	
hn2(nu, z)		
hyper(ap, bq, z)  The generalized hypergeometric function is defined by a series where the ratios of successive terms are a rational function of the summation index.  im(arg)  Returns imaginary part of expression.  Jacobi (n, a, b, x)  Jacobi polynomial \$P_n^{[left(alpha, betaright)]}(x)\$.  jn(nu, z)  Spherical Bessel function of the first kind.  laguerre(n, x)  Returns the \$n\$th Laguerre polynomial in \$x\$, \$\$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$P_n(x)\$  lerchphi(*args)  Lerch transcendent (Lerch phi function).  li(z)  The classical logarithmic integral.  ln  alias of log  log(arg[, base])  The natural logarithm function ln(x) or log(x).  loggamma(z)  The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).  lowergamma(a, x)  The lower incomplete gamma function.  lucas(n)  Lucas numbers  marcumq(m, a, b)  The Marcum Q-function.  mathieuc(a, q, z)  The Mathieu Cosine function \$C(a,q,z)\$,  mathieucprime(a, q, z)  The Mathieu Cosine function \$C(a,q,z)\$,  mathieusprime(a, q, z)  The Mathieu Sine function \$S(a,q,z)\$ of the Mathieu Cosine function.  meijerg(*args)  The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  Mobius function maps natural number to [-1, 0, 1]  motzkin(n)  The nth Motzkin number is the number		
$ \begin{array}{c} series \ where the ratios of successive terms are a rational function of the summation index. \\ im(arg) & Returns imaginary part of expression. \\ jacobi(n, a, b, x) & Jacobi polynomial SP_n^*\{left(alpha, betaright)\}(x)$. Spherical Bessel function of the first kind. \\ laguerre(n, x) & Spherical Bessel function of the first kind. \\ laguerre(n, x) & Returns the SnSth Laguerre polynomial in SxS, SL_n(x)S$. \\ legendre(n, x) & legendre(n, x) gives the SnSth Legendre polynomial of SxS, SP_n(x)S$ lerchphi(*args) & Lerch transcendent (Lerch phi function). \\ 1i(z) & The classical logarithmic integral. \\ 1n & alias of 10g $		•
function of the summation index.  im(arg) Returns imaginary part of expression. jacobi(n, a, b, x) Jacobi polynomial \$P_n^{[left(alpha, betaright)](x)\$.  jn(nu, z) Spherical Bessel function of the first kind. laguerre(n, x) Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$.  legendre(n, x) legendre(n, x) legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$  lerchphi(*args) Lerch transcendent (Lerch phi function).  li(z) The classical logarithmic integral.  alias of log log(arg[, base]) The natural logarithm function ln(x) or log(x).  loggamma(z) The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).  lowergamma(a, x) The lower incomplete gamma function.  lucas(n) Lucas numbers marcumq(m, a, b) The Marcum Q-function.  mathieuc(a, q, z) The Mathieu Cosine function \$C(a,q,z)\$.  mathieucprime(a, q, z) The derivative \$C^{prime}(a,q,z)\$ of the Mathieu Cosine function.  mathieus(a, q, z) The Mathieu Sine function \$S(a,q,z)\$.  mathieusprime(a, q, z) The Mathieu Sine function \$S(a,q,z)\$ of the Mathieu Sine function.  meijerg(*args) The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number	hyper(ap, bq, z)	
im(arg) jacobi(n, a, b, x) jacobi(n, a, b, x) jacobi(n, a, b, x) jn(nu, z) Spherical Bessel function of the first kind.  laguerre(n, x) Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$.  legendre(n, x) joint the shift Laguerre polynomial in \$x\$, \$L_n(x)\$.  legendre(n, x) legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$ lerchphi(*args) Lerch transcendent (Lerch phi function).  li(z) The classical logarithmic integral.  ln alias of log log(arg[, base]) The natural logarithm function ln(x) or log(x).  logarma(z) The loggarma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).  lowergamma(a, x) The lower incomplete gamma function.  lucas(n) Lucas numbers marcumq(m, a, b) The Marcum Q-function.  mathieuc(a, q, z) The Mathieu Cosine function \$C(a,q,z)\$.  mathieucprime(a, q, z) The derivative \$C^{prime}{(a,q,z)}\$ of the Mathieu Cosine function.  mathieus(a, q, z) The derivative \$S^{prime}{(a,q,z)}\$ of the Mathieu Sine function.  meijerg(*args) The Mathieu Sine function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number		
jacobi(n, a, b, x) jn(nu, z) Spherical Bessel function of the first kind.  laguerre(n, x) Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$.  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$P_n(x)\$  legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$\$P_n(x)\$  lerchphi(*args) Lerch transcendent (Lerch phi function).  li(z) The classical logarithmic integral.  ln alias of log log(arg[, base]) The natural logarithm function $ln(x)$ or $log(x)$ .  loggamma(z) The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).  lowergamma(a, x) The lower incomplete gamma function.  lucas(n) Lucas numbers marcumq(m, a, b) The Marcum Q-function.  mathieuc(a, q, z) The Mathieu Cosine function \$C(a,q,z)\$.  mathieucprime(a, q, z) The derivative \$C^{prime}(a,q,z)\$ of the Mathieu Cosine function.  mathieus(a, q, z) The Mathieu Sine function \$S(a,q,z)\$.  mathieus(a, q, z) The derivative \$S^{prime}(a,q,z)\$ of the Mathieu Sine function.  meijerg(*args) The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number		
jn(nu, z)Spherical Bessel function of the first kind.laguerre(n, x)Returns the \$n\$th Laguerre polynomial in \$x\$, \$L_n(x)\$.legendre(n, x)legendre(n, x) gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$lerchphi(*args)Lerch transcendent (Lerch phi function).li(z)The classical logarithmic integral.lnalias of loglog(arg[, base])The natural logarithm function $ln(x)$ or $log(x)$ .loggamma(z)The loggamma function implements the logarithm of the gamma function (i.e., \$logGamma(x)\$).lowergamma(a, x)The lower incomplete gamma function.lucas(n)Lucas numbersmarcumq(m, a, b)The Marcum Q-function.mathieuc(a, q, z)The Mathieu Cosine function \$C(a,q,z)\$.mathieucprime(a, q, z)The derivative \$C^{prime}(a,q,z)\$ of the Mathieu Cosine function.mathieus(a, q, z)The derivative \$S^{prime}(a,q,z)\$ of the Mathieu Sine function.meijerg(*args)The derivative \$S^{prime}(a,q,z)\$ of the Mathieu Sine function.meijerg(*args)The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.mobius(n)Mobius function maps natural number to {-1, 0, 1}motzkin(n)The nth Motzkin number is the number	. •	
$ \begin{array}{c} laguerre(n,x) & Returns \ the \ sn\$th \ Laguerre \ polynomial \ in \ \$x\$, \ \$L_n(x)\$. \\ legendre(n,x) & legendre(n,x) \ gives the \ sn\$th \ Legendre \ polynomial \ of \ \$x\$, \ \$P_n(x)\$ \\ lerchphi(*args) & Lerch \ transcendent \ (Lerch \ phi \ function). \\ li(z) & The \ classical \ logarithmic \ integral. \\ ln & alias \ of \ log \\ log(arg[, base]) & The \ natural \ logarithm \ function \ ln(x) \ or \ log(x). \\ loggamma(z) & The \ loggamma \ function \ implements \ the \ logarithm \ of \ the \ gamma \ function \ (i.e., \$logGamma(x)\$). \\ lowergamma(a,x) & The \ lower \ incomplete \ gamma \ function. \\ lucas(n) & Lucas \ numbers \\ marcumq(m,a,b) & The \ Marcum \ Q-function. \\ mathieuc(a,q,z) & The \ Mathieu \ Cosine \ function \ \$C(a,q,z)\$. \\ mathieucprime(a,q,z) & The \ Mathieu \ Sine \ function. \\ mathieus(a,q,z) & The \ Mathieu \ Sine \ function. \\ mathieus(a,q,z) & The \ Mathieu \ Sine \ function. \\ meijerg(*args) & The \ Meijer \ G-function \ is \ defined \ by \ a \ Mellin-Barnes \ type \ integral \ that \ resembles \ an \ inverse \ Mellin \ transform. \\ mobius(n) & Mobius \ function \ maps \ natural \ number \ to \ \{-1,0,1\} \\ motzkin(n) & The \ nth \ Motzkin \ number \ is \ the \ number \end{aligned}$		
SL_n(x)\$.   legendre(n, x)   gives the \$n\$th Legendre polynomial of \$x\$, \$P_n(x)\$   Lerchphi(*args)   Lerch transcendent (Lerch phi function).		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	laguerre(n, x)	
$\begin{tabular}{ll} of $x\$, $P_n(x)\$ \\ lerchphi(*args) & Lerch transcendent (Lerch phi function). \\ li(z) & The classical logarithmic integral. \\ ln & alias of log \\ log(arg[, base]) & The natural logarithm function ln(x) or log(x). \\ loggamma(z) & The loggamma function implements the logarithm of the gamma function (i.e., $logGamma(x)$). \\ lowergamma(a, x) & The lower incomplete gamma function. \\ lucas(n) & Lucas numbers \\ marcumq(m, a, b) & The Marcum Q-function. \\ mathieuc(a, q, z) & The Mathieu Cosine function $C(a,q,z)$. \\ mathieucprime(a, q, z) & The derivative $C^{\prime}(a,q,z)$ of the Mathieu Cosine function. \\ mathieus(a, q, z) & The Mathieu Sine function $S(a,q,z)$. \\ mathieusprime(a, q, z) & The derivative $S^{\prime}(a,q,z)$ of the Mathieu Sine function. \\ meijerg(*args) & The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform. \\ mobius(n) & Mobius function maps natural number to $-1, 0, 1$ motzkin(n) & The nth Motzkin number is the number \\ \hline \end{tabular}$		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	legendre(n, x)	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
$\begin{array}{llllllllllllllllllllllllllllllllllll$		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	-	•
$\begin{array}{llllllllllllllllllllllllllllllllllll$	loggamma(z)	
$\begin{array}{llll} lucas(n) & Lucas numbers \\ marcumq(m, a, b) & The Marcum Q-function. \\ mathieuc(a, q, z) & The Mathieu Cosine function C(a,q,z). \\ mathieucprime(a, q, z) & The derivative C^{prime}(a,q,z) of the Mathieu Cosine function. \\ mathieus(a, q, z) & The Mathieu Sine function S(a,q,z). \\ mathieusprime(a, q, z) & The derivative C^{prime}(a,q,z) of the Mathieu Sine function. \\ meijerg(*args) & The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform. \\ mobius(n) & Mobius function maps natural number to C^{prime}(a,q,z) motzkin(n) & The nth Motzkin number is the number \\ \end{array}$		
$\begin{array}{lll} \text{marcumq}(m,a,b) & \text{The Marcum Q-function.} \\ \text{mathieuc}(a,q,z) & \text{The Mathieu Cosine function $C(a,q,z)$.} \\ \text{mathieucprime}(a,q,z) & \text{The derivative $C^{\text{prime}}(a,q,z)$ of the Mathieu Cosine function.} \\ \text{mathieus}(a,q,z) & \text{The Mathieu Sine function $S(a,q,z)$.} \\ \text{mathieusprime}(a,q,z) & \text{The derivative $S^{\text{prime}}(a,q,z)$ of the Mathieu Sine function.} \\ \text{meijerg}(*args) & \text{The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.} \\ \text{mobius}(n) & \text{Mobius function maps natural number to $\{-1,0,1\}$} \\ \text{motzkin}(n) & \text{The nth Motzkin number is the number} \\ \end{array}$		
$\begin{array}{lll} \text{mathieuc(a, q, z)} & \text{The Mathieu Cosine function $C(a,q,z)$.} \\ \text{mathieucprime(a, q, z)} & \text{The derivative $C^{\text{prime}}(a,q,z)$ of the Mathieu Cosine function.} \\ \text{mathieus(a, q, z)} & \text{The Mathieu Sine function $S(a,q,z)$.} \\ \text{mathieusprime(a, q, z)} & \text{The derivative $S^{\text{prime}}(a,q,z)$ of the Mathieu Sine function.} \\ \text{meijerg(*args)} & \text{The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.} \\ \text{mobius(n)} & \text{Mobius function maps natural number to $\{-1,0,1\}$} \\ \text{motzkin(n)} & \text{The nth Motzkin number is the number} \\ \end{array}$		
$\begin{array}{ccc} & sine \ function. \\ \\ \text{mathieus}(a,q,z) & The \ Mathieu \ Sine \ function \ \$S(a,q,z)\$. \\ \\ \text{mathieusprime}(a,q,z) & The \ derivative \ \$S^{prime}(a,q,z)\$ \ of \ the \ Mathieu \ Sine \ function. \\ \\ \text{meijerg}(*args) & The \ Meijer \ G-function \ is \ defined \ by \ a \ Mellin-Barnes \ type \ integral \ that \ resembles \ an \ inverse \ Mellin \ transform. \\ \\ \text{mobius}(n) & Mobius \ function \ maps \ natural \ number \ to \ \{-1,0,1\} \\ \\ \text{motzkin}(n) & The \ nth \ Motzkin \ number \ is \ the \ number \\ \end{array}$		
$\begin{tabular}{lll} mathieus(a,q,z) & The Mathieu Sine function $S(a,q,z)$. \\ mathieusprime(a,q,z) & The derivative $S^{prime}(a,q,z)$ of the Mathieu Sine function. \\ meijerg(*args) & The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform. \\ mobius(n) & Mobius function maps natural number to \{-1,0,1\} motzkin(n) & The nth Motzkin number is the number$	$\mathtt{mathieucprime}(a,q,z)$	
$\begin{tabular}{lll} mathieusprime(a,q,z) & The derivative $S^{prime}(a,q,z)$ of the Mathieu Sine function. \\ meijerg(*args) & The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform. \\ mobius(n) & Mobius function maps natural number to \{-1,0,1\} motzkin(n) & The nth Motzkin number is the number$		
function.  meijerg(*args) The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number	· · · ·	=
meijerg(*args) The Meijer G-function is defined by a Mellin-Barnes type integral that resembles an inverse Mellin transform.  mobius(n) Mobius function maps natural number to {-1, 0, 1} motzkin(n) The nth Motzkin number is the number	$\mathtt{mathieusprime}(a,q,z)$	
	meijerg(*args)	· · · · · · · · · · · · · · · · · · ·
motzkin(n) The nth Motzkin number is the number		
· ·	` '	
continues on next page	motzkin(n)	

Table 65 – continued from previous page

Table 65 – Continue	a ironi previous page
$\operatorname{multigamma}(x, p)$	The multivariate gamma function is a generalization of the gamma function
partition(n)	Partition numbers
periodic_argument(ar, period)	Represent the argument on a quotient of the Riemann surface of the logarithm.
<pre>polar_lift(arg)</pre>	Lift argument to the Riemann surface of the logarithm, using the standard branch.
$\operatorname{polygamma}(n,z)$	The function polygamma(n, z) returns log(gamma(z)).diff(n + 1).
polylog(s, z)	Polylogarithm function.
<pre>preorder_traversal(node[, keys])</pre>	Do a pre-order traversal of a tree.
primenu(n)	Calculate the number of distinct prime factors for a positive integer n.
<pre>primeomega(n)</pre>	Calculate the number of prime factors counting multiplicities for a positive integer n.
<pre>primepi(n)</pre>	Represents the prime counting function $pi(n)$ = the number of prime numbers less than or equal to n.
<pre>principal_branch(x, period)</pre>	Represent a polar number reduced to its principal branch on a quotient of the Riemann surface of the logarithm.
re(arg)	Returns real part of expression.
reduced_totient(n)	Calculate the Carmichael reduced totient function lambda(n)
rf	alias of RisingFactorial
riemann_xi(s)	Riemann Xi function.
sec(arg)	The secant function.
sech(arg)	sech(x) is the hyperbolic secant of x.
sign(arg)	Returns the complex sign of an expression:
sin(arg)	The sine function.
sinc(arg)	Represents an unnormalized sinc function:
sinh(arg)	sinh(x) is the hyperbolic sine of x.
<pre>stieltjes(n[, a])</pre>	Represents Stieltjes constants, \$gamma_{k}\$ that occur in Laurent Series expansion of the Riemann zeta function.
subfactorial(arg)	The subfactorial counts the derangements of \$n\$ items and is defined for non-negative integers as:
tan(arg)	The tangent function.
tanh(arg)	tanh(x) is the hyperbolic tangent of $x$ .
totient(n)	Calculate the Euler totient function phi(n)
transpose(arg)	Linear map transposition.
tribonacci(n[, sym])	Tribonacci numbers / Tribonacci polynomials
trigamma(z)	The trigamma function is the second derivative of the loggamma function
uppergamma(a, z)	The upper incomplete gamma function.
vectorize(*mdargs)	Generalizes a function taking scalars to accept multidimensional arguments.
yn(nu, z)	Spherical Bessel function of the second kind.
zeta(s[,a])	Hurwitz zeta function (or Riemann zeta function).

## **Exceptions**

BasePolynomialError	Base class for polynomial related exceptions.
CoercionFailed	1 2
ComputationFailed(func, nargs, exc)	
DomainError	
EvaluationFailed	
<pre>ExactQuotientFailed(f, g[, dom])</pre>	
ExtraneousFactors	
FlagError	
GeneratorsError	
GeneratorsNeeded	
GeometryError HeuristicGCDFailed	An exception raised by classes in the geometry module.
HomomorphismFailed	
IsomorphismFailed	
MultivariatePolynomialError	
NonSquareMatrixError	
NotAlgebraic	
NotInvertible	
NotReversible	
OperationNotSupported(poly, func)	
OptionError	
PoleError	
<pre>PolificationFailed(opt, origs, exprs[, seq])</pre>	
${\tt PolynomialDivisionFailed}(f,g,domain)$	
PolynomialError	
PrecisionExhausted	
	continues on next page

Table 66 – continued from previous page

	rable of command nom provided page	
RefinementFailed		
ShapeError	Wrong matrix shape	
SympifyError(expr[, base_exc])		
UnificationFailed		
UnivariatePolynomialError		

class src.sensitivity.sensitivity\_tools.AutoSympy(model)

## check\_complimentarity\_all()

quick check of complementarity conditions

## generate\_duals(constraints, duals)

cycles through constraints, extracts duals, categorizes by complementary slackness conditions

#### **Parameters**

- **constraints** (*dict*) iterator of pyomo constraints
- duals (obj) duals object from pyomo model

## Returns

dictionary of duals sorts by type

#### Return type

dict

## get\_constraints()

extract constraint expressions from self.model and convert to Sympy expressions in terms of the extracted Sympy Symbols for variables and parameters

## Returns

- equality\_constraints (*dict*) dictionary of names and indices of equality constraints: Sympy expression for lhs of constraint
- **inequality\_constraints** (*dict*) dictionary of names and indices of inequality constraints:Sympy expression for lhs of constraint

## get\_objective()

extract objective expression from self.model

## Returns

objective expression

#### **Return type**

expr

## get\_parameters()

extracts parameters from self.model and converts to Sympy Symbol or IndexedBase + index objects depending on whether they are indexed components or not.

#### Returns

- parameters (dict) dictionary of parameter names: IndexedBase objects with that name
- parameters\_values (dict) dictionary of Sympy objects:their numeric value

• parameter\_index\_sets (dict) – dictionary of parameter name:list of indices for the name

## get\_sensitivity\_matrix(parameters\_of\_interest=None)

generates a SensitivityMatrix object based on the sympy representation, keeping parameters in parameters\_of\_interest and substituting numeric values for the rest in all expressions.

#### **Parameters**

**parameters\_of\_interest** (*list*, *optional*) – list of parameters to keep symbolic. These are the parameters to evaluate sensitivity for. Defaults to None.

#### Returns

SensitivityMatrix object for the model, with parameters\_of\_interest considered.

## Return type

obj

#### get\_sets()

extracts sets from self.model and converts them to Sympy Idx objects

#### Returns

- sets (dict) dictionary of set names:Idx objects
- set\_values (dict) dictionary of Idx objects:corresponding element of the Pyomo set

## get\_variables()

extracts variables from self.model and converts them to Sympy Symbol or IndexedBase + index objects

#### Returns

- **variables** (*dict*) dictionary of variable names:corresponding Sympy object (Symbol or IndexedBase)
- **variable\_values** (*dict*) dictionary of Sympy objects (Symbol or IndexedBase):numeric value

## substitute\_values(substitution\_dict)

substitutes numeric values for sympy symbols according to substitution\_dict

## **Parameters**

**substitution\_dict** (*dict*) – dictionary of symbols:values to substitute

**class** src.sensitivity.sensitivity\_tools.**CoordMap**(*var\_vector*, *eq\_duals*, *ineq\_duals*, *params*)

extrapolate(current\_values, param\_delta)

## given the current value of all varying quantities (variables, duals, objective) as a vector,

and a delta of parameter values as a vector, calculates the effect of perturbing the parameters by the parameter delta.

## **Parameters**

- current\_values (Matrix) vector of [variables values, dual values, objective value]
- param\_delta (Matrix) vector of [parameter values]

#### Returns

- item\_delta (obj) column Matrix of items
- item\_delta\_map (dict) map between items and their changes

- item\_new\_value (*Matrix*) column Matrix of new values after change
- item\_new\_value\_map (dict) map between items and their new values

## sensitivity(item, parameter)

picks out the sensitivity of item with respect to parameters

#### **Parameters**

- item (symbol, str) however the particular item was stored, usually symbol
- parameter (symbol, str) however the particular parameter was stored, usually symbol

## Returns

the sensitivity of item with respect to parameter.

#### **Return type**

float

## create\_substitution\_dictionary(values\_of\_interest=None)

creates a single dictionary with all values to be substituted into the sensitivity matrix.

#### **Parameters**

**values\_of\_interest** (*dict*, *optional*) – Subset of all possible substitutions you'd like to perform. Defaults to None.

#### Returns

dictionary of substitution values

## Return type

dict

## generate\_matrix()

generates the submatrix components of the sensitivity matrix that will be used to express sensitivities and extrapolate from a given solution point

#### Returns

dictionary of submatrix name: Matrix object.

## Return type

dict

## get\_sensitivities()

creates a set of dictionaries to keep track of the relationships between symbol names, symbol objects, and their position in the respective vectors of parameters and variable quantities. Calculates U^-1\*S and Creates a DifferentialMapping object. Sensitivities of quantities with respect to parameters will be corresponding entries in the matrix generated, which can be queried directly in the DifferentialMapping object

## Returns

an DifferentialMapping object that stores the sensitivity matrix and dictionaries matching coordinates to symbols

#### Return type

obj

## matrix\_assembly(components)

assemble the submatrix components into U,S

#### **Parameters**

**components** (*dict*) – dictionary of submatrix names:Matrix objects

#### **Returns**

- obj Matrix object U
- obj Matrix object S

## resolve\_kronecker(expr)

evaluates KroneckerDelta terms

#### **Parameters**

**obj** – sympy expression

#### **Returns**

sympy expr with KroneckerDelta terms evaluated to 0 or 1

## Return type

obj

## substitute\_values(values\_dict=None)

substitute values into self.U and self.S according to a given substitution dictionary, or a substition dictionary for all values

#### **Parameters**

values\_dict (dict, optional) - dictionary of symbols:values

#### **Returns**

- *obj* the stored Matrix object U after substituting numeric values
- obj the stored matrix S after substituting numeric values

## src.sensitivity.speed\_test

Speed Test This is a script with some functions to run speed and accuracy tests on test models constructed with baby-model.

It can be run directly. Parameter values are at the top of the file and any desired value can be entered into the declarations. The script will then run a sequence of functions to time the build, sympification, and sensitivity calculation for the TestBabyModel constructed with those input values.

## **Functions**

E1(z)	Classical case of the generalized exponential integral.
Eijk(*args, **kwargs)	Represent the Levi-Civita symbol.
<pre>GramSchmidt(vlist[, orthonormal])</pre>	Apply the Gram-Schmidt process to a set of vectors.
LC(f, *gens, **args)	Return the leading coefficient of f.
LM(f, *gens, **args)	Return the leading monomial of f.
LT(f, *gens, **args)	Return the leading term of f.
N(x[, n])	Calls x.evalf(n, **options).
POSform(variables, minterms[, dontcares])	The POSform function uses simplified_pairs and a redundant-group eliminating algorithm to convert the list of all input combinations that generate '1' (the minterms) into the smallest product-of-sums form.

continues on next page

Table 67 – continued from previous page

Table 07 continue	ed from previous page
SOPform(variables, minterms[, dontcares])	The SOPform function uses simplified_pairs and a redundant group- eliminating algorithm to convert the list of all input combos that generate '1' (the minterms) into the smallest sum-of-products form.
Ynm_c(n, m, theta, phi)	Conjugate spherical harmonics defined as
abundance(n)	Returns the difference between the sum of the positive
	proper divisors of a number and the number.
apart(f[, x, full])	Compute partial fraction decomposition of a rational
-F(-[,,])	function.
<pre>apart_list(f[, x, dummies])</pre>	Compute partial fraction decomposition of a rational
<b>apar</b> ===== ((x <sub>1</sub> ,, dammos <sub>1</sub> )	function and return the result in structured form.
<pre>apply_finite_diff(order, x_list, y_list[, x0])</pre>	Calculates the finite difference approximation of the
	derivative of requested order at x0 from points provided
	in x_list and y_list.
approximants([[, X, simplify])	Return a generator for consecutive Pade approximants
approximates(i, 11, simpiny)	for a series.
are_similar(e1, e2)	Are two geometrical entities similar.
arity(cls)	Return the arity of the function if it is known, else None.
ask(proposition[, assumptions, context])	Function to evaluate the proposition with assumptions.
assemble_partfrac_list(partial_list)	Reassemble a full partial fraction decomposition from a
assemble_parerrae_rrse(param_nse)	structured result obtained by the function apart_list.
assuming(*assumptions)	Context manager for assumptions.
banded(*args, **kwargs)	Returns a SparseMatrix from the given dictionary de-
banded aigs, kwaigs)	scribing the diagonals of the matrix.
besselsimp(expr)	Simplify bessel-type functions.
binomial_coefficients(n)	Return a dictionary containing pairs $(k1, k2) : C_k n$
DIROMIAI_COEIIICIERCS(II)	where $C_k n$ are binomial coefficients and $n = k1 + k2$ .
binomial_coefficients_list(n)	Return a list of binomial coefficients as rows of the Pas-
binomial_coefficients_fist(n)	cal's triangle.
block_collapse(expr)	Evaluates a block matrix expression
blockcut(expr, rowsizes, colsizes)	Cut a matrix expression into Blocks
bool_map(bool1, bool2)	Return the simplified version of <i>bool1</i> , and the mapping
boo1_map(boo11, boo12)	of variables that makes the two expressions <i>bool1</i> and
	bool2 represent the same logical behaviour for some cor-
	respondence between the variables of each.
bottom_up(rv, F[, atoms, nonbasic])	Apply F to all expressions in an expression tree from the
boccom_up(1 v, 1 [, atoms, nonoasic])	bottom up.
bspline_basis(d, knots, n, x)	The \$n\$-th B-spline at \$x\$ of degree \$d\$ with knots.
bspline_basis_set(d, knots, x)	Return the len(knots)-d-1 B-splines at $x$ of degree $d$
bspline_basis_set(u, knots, x)	with <i>knots</i> .
cacheit(func)	with Millis.
cacher c(tune)	
<pre>cancel(f, *gens[, _signsimp])</pre>	Cancel common factors in a rational function £.
capture(func)	Return the printed output of func().
casoratian(seqs, n[, zero])	Given linear difference operator L of order 'k' and homo-
20001 actum(3043, 11, 2010])	geneous equation $Ly = 0$ we want to compute kernel of
	L, which is a set of 'k' sequences: a(n), b(n), .
cbrt(arg[, evaluate])	Returns the principal cube root.
ccode(expr[, assign_to, standard])	Converts an expr to a string of c code
centroid(*args)	Find the centroid (center of mass) of the collection con-
centroru( mgs)	taining only Points, Segments or Polygons.
	continues on next page

Table 67 – continued from previous page

Table 07 - Continue	a nom previous page
<pre>chebyshevt_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the first kind $T_n(x)$ .
<pre>chebyshevu_poly(n[, x, polys])</pre>	Generates the Chebyshev polynomial of the second kind $U_n(x)$ .
<pre>check_assumptions(expr[, against])</pre>	Checks whether assumptions of expr match the T/F assumptions given (or possessed by against).
<pre>checkodesol(ode, sol[, func, order,])</pre>	Substitutes sol into ode and checks that the result is <b>0</b> .
checkpdesol(pde, sol[, func, solve_for_func])	Checks if the given solution satisfies the partial differential equation.
checksol(f, symbol[, sol])	Checks whether sol is a solution of equation $f == 0$ .
<pre>classify_ode(eq[, func, dict, ics, prep,])</pre>	Returns a tuple of possible dsolve() classifications for an ODE.
<pre>classify_pde(eq[, func, dict, prep])</pre>	Returns a tuple of possible pdsolve() classifications for a PDE.
<pre>closest_points(*args)</pre>	Return the subset of points from a set of points that were the closest to each other in the 2D plane.
cofactors(f, g, *gens, **args)	Compute GCD and cofactors of f and g.
collect(expr, syms[, func, evaluate, exact,])	Collect additive terms of an expression.
<pre>collect_const(expr, *vars[, Numbers])</pre>	A non-greedy collection of terms with similar number coefficients in an Add expr.
combsimp(expr)	Simplify combinatorial expressions.
comp(z1, z2[, tol])	Return a bool indicating whether the error between z1 and z2 is \$le\$ to1.
compose(f, g, *gens, **args)	Compute functional composition f(g).
composite(nth)	Return the nth composite number, with the composite
	numbers indexed as composite(1) = 4, composite(2) = 6, etc
<pre>compositepi(n)</pre>	Return the number of positive composite numbers less than or equal to n.
connect_regions(region_map, hubmap,)	given a mapping of regions to lists of hubs, and hubs to regions, creates a set of arcs between hubs such that:
<pre>construct_domain(obj, **args)</pre>	Construct a minimal domain for a list of expressions.
content(f, *gens, **args)	Compute GCD of coefficients of f.
continued_fraction(a)	Return the continued fraction representation of a Rational or quadratic irrational.
<pre>continued_fraction_convergents(cf)</pre>	Return an iterator over the convergents of a continued fraction (cf).
<pre>continued_fraction_iterator(x)</pre>	Return continued fraction expansion of x as iterator.
$\label{eq:continued_fraction_periodic} \textbf{continued\_fraction\_periodic}(p,q[,d,s])$	Find the periodic continued fraction expansion of a quadratic irrational.
<pre>continued_fraction_reduce(cf)</pre>	Reduce a continued fraction to a rational or quadratic irrational.
<pre>convex_hull(*args[, polygon])</pre>	The convex hull surrounding the Points contained in the list of entities.
<pre>convolution(a, b[, cycle, dps, prime,])</pre>	Performs convolution by determining the type of desired convolution using hints.
<pre>cosine_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency cosine transform of <i>f</i> , defined as
<pre>count_ops(expr[, visual])</pre>	Return a representation (integer or expression) of the operations in expr.
<pre>count_roots(f[, inf, sup])</pre>	Return the number of roots of f in [inf, sup] interval.
covering_product(a, b)	Returns the covering product of given sequences.
	continues on next page

Table 67 – continued from previous page

Table 67 – continued from previous page		
cse(exprs[, symbols, optimizations,])	Perform common subexpression elimination on an ex-	
	pression.	
<pre>cxxcode(expr[, assign_to, standard])</pre>	C++ equivalent of ccode().	
cycle_length(f, x0[, nmax, values])	For a given iterated sequence, return a generator that	
	gives the length of the iterated cycle (lambda) and the	
	length of terms before the cycle begins (mu); if values	
	is True then the terms of the sequence will be returned	
	instead.	
<pre>cyclotomic_poly(n[, x, polys])</pre>	Generates cyclotomic polynomial of order $n$ in $x$ .	
decompogen(f, symbol)	Computes General functional decomposition of f.	
	Given an expression $f$ , returns a list $[f_1, f_2,,$	
	$f_n$ , where:: $f = f_1 \circ f_2 \circ f_n = f_1(f_2( f_n))$ .	
decompose(f, *gens, **args)	Compute functional decomposition of f.	
<pre>default_sort_key(item[, order])</pre>	Return a key that can be used for sorting.	
deg(r)	Return the degree value for the given radians ( $pi = 180$	
	degrees).	
degree(f[, gen])	Return the degree of f in the given variable.	
<pre>degree_list(f, *gens, **args)</pre>	Return a list of degrees of f in all variables.	
denom(expr)		
derive_by_array(expr, dx)	Derivative by arrays.	
det(matexpr)	Matrix Determinant	
<pre>det_quick(M[, method])</pre>	Return det(M) assuming that either there are lots of ze-	
11 (4 1 5 11)	ros or the size of the matrix is small.	
diag(*values[, strict, unpack])	Returns a matrix with the provided values placed on the diagonal.	
diagonalize_vector(vector)	·	
dist manage(*dists)	Marga diationarias into a single diationary	
<pre>dict_merge(*dicts) diff(f, *symbols, **kwargs)</pre>	Merge dictionaries into a single dictionary.  Differentiate f with respect to symbols.	
difference_delta(expr[, n, step])	Difference Operator.	
differentiate_finite(expr, *symbols[,])	Differentiate expr and replace Derivatives with finite dif-	
	ferences.	
<pre>diophantine(eq[, param, syms, permute])</pre>	Simplify the solution procedure of diophantine equation	
	eq by converting it into a product of terms which should	
	equal zero.	
<pre>discrete_log(n, a, b[, order, prime_order])</pre>	Compute the discrete logarithm of a to the base b mod-	
1::	ulo n.	
discriminant(f, *gens, **args)	Compute discriminant of f.	
div(f, g, *gens, **args)	Compute polynomial division of f and g.	
divisor_count(n[, modulus, proper])	Return the number of divisors of n.	
divisors(n[, generator, proper])	Return all divisors of n sorted from 1n by default.	
dotprint(expr[, styles, atom, maxdepth,])	DOT description of a SymPy expression tree	
dsolve(eq[, func, hint, simplify, ics, xi,])	Solves any (supported) kind of ordinary differential equation and system of ordinary differential equations.	
egyptian_fraction(r[, algorithm])	Return the list of denominators of an Egyptian fraction	
cyperal_raceron(real, argorithms)	expansion [1] of the said rational $r$ .	
epath(path[, expr, func, args, kwargs])	Manipulate parts of an expression selected by a path.	
euler_equations(L[, funcs, vars])	Find the Euler-Lagrange equations [1] for a given La-	
(2[, 12000, 1200])	grangian.	
evaluate(x)	Control automatic evaluation	
expand(e[, deep, modulus, power_base,])	Expand an expression using methods given as hints.	
1	continues on next page	

Table 67 – continued from previous page

Table 07 - continue	ed from previous page
<pre>expand_complex(expr[, deep])</pre>	Wrapper around expand that only uses the complex hint.
<pre>expand_func(expr[, deep])</pre>	Wrapper around expand that only uses the func hint.
<pre>expand_log(expr[, deep, force, factor])</pre>	Wrapper around expand that only uses the log hint.
<pre>expand_mul(expr[, deep])</pre>	Wrapper around expand that only uses the mul hint.
<pre>expand_multinomial(expr[, deep])</pre>	Wrapper around expand that only uses the multinomial hint.
<pre>expand_power_base(expr[, deep, force])</pre>	Wrapper around expand that only uses the power_base hint.
<pre>expand_power_exp(expr[, deep])</pre>	Wrapper around expand that only uses the power_exp hint.
<pre>expand_trig(expr[, deep])</pre>	Wrapper around expand that only uses the trig hint.
exptrigsimp(expr)	Simplifies exponential / trigonometric / hyperbolic functions.
exquo(f, g, *gens, **args)	Compute polynomial exact quotient of f and g.
eye(*args, **kwargs)	Create square identity matrix n x n
<pre>factor(f, *gens[, deep])</pre>	Compute the factorization of expression, f, into irreducibles.
<pre>factor_list(f, *gens, **args)</pre>	Compute a list of irreducible factors of f.
factor_nc(expr)	Return the factored form of <b>expr</b> while handling non-commutative expressions.
<pre>factor_terms(expr[, radical, clear,])</pre>	Remove common factors from terms in all arguments without changing the underlying structure of the expr.
<pre>factorint(n[, limit, use_trial, use_rho,])</pre>	Given a positive integer n, factorint(n) returns a dict containing the prime factors of n as keys and their respective multiplicities as values.
<pre>factorrat(rat[, limit, use_trial, use_rho,])</pre>	Given a Rational r, factorrat(r) returns a dict containing the prime factors of r as keys and their respective multiplicities as values.
<pre>failing_assumptions(expr, **assumptions)</pre>	Return a dictionary containing assumptions with values not matching those of the passed assumptions.
<pre>farthest_points(*args)</pre>	Return the subset of points from a set of points that were the furthest apart from each other in the 2D plane.
fcode(expr[, assign_to])	Converts an expr to a string of fortran code
fft(seq[, dps])	Performs the Discrete Fourier Transform ( <b>DFT</b> ) in the complex domain.
<pre>field(symbols, domain[, order])</pre>	Construct new rational function field returning (field, x1,, xn).
<pre>field_isomorphism(a, b, *[, fast])</pre>	Find an embedding of one number field into another.
<pre>filldedent(s[, w])</pre>	Strips leading and trailing empty lines from a copy of s, then dedents, fills and returns it.
<pre>finite_diff_weights(order, x_list[, x0])</pre>	Calculates the finite difference weights for an arbitrarily spaced one-dimensional grid (x_list) for derivatives at x0 of order 0, 1,, up to order using a recursive formula.
<pre>flatten(iterable[, levels, cls])</pre>	Recursively denest iterable containers.
<pre>fourier_series(f[, limits, finite])</pre>	Computes the Fourier trigonometric series expansion.
<pre>fourier_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency Fourier transform of f, defined as
fps(f[, x, x0, dir, hyper, order, rational,])	Generates Formal Power Series of £.
<pre>fraction(expr[, exact])</pre>	Returns a pair with expression's numerator and denominator.
	continues on next page

Table 67 – continued from previous page

Table 67 – continued from previous page		
fu(rv[, measure])	Attempt to simplify expression by using transformation	
	rules given in the algorithm by Fu et al.	
fwht(seq)	Performs the Walsh Hadamard Transform (WHT), and	
` *	uses Hadamard ordering for the sequence.	
<pre>galois_group(f, *gens[, by_name, max_tries,])</pre>	Compute the Galois group for polynomials $f$ up to de-	
	gree 6.	
gammasimp(expr)	Simplify expressions with gamma functions.	
gcd(f[,g])	Compute GCD of f and g.	
gcd_list(seq, *gens, **args)	Compute GCD of a list of polynomials.	
gcd_terms(terms[, isprimitive, clear, fraction])	Compute the GCD of terms and put them together.	
gcdex(f, g, *gens, **args)	Extended Euclidean algorithm of f and g.	
<pre>generate([num_regions, hubs_per_region,])</pre>	generates a random network with all parameters required	
	to initialize a ToyBabyModel	
<pre>getLogger([name])</pre>	Return a logger with the specified name, creating it if	
3 33 (1)	necessary.	
<pre>get_contraction_structure(expr)</pre>	Determine dummy indices of expr and describe its	
· · · · · · ·	structure	
<pre>get_indices(expr)</pre>	Determine the outer indices of expression expr	
gff(f, *gens, **args)	Compute greatest factorial factorization of <b>f</b> .	
gff_list(f, *gens, **args)	Compute a list of greatest factorial factors of f.	
glsl_code(expr[, assign_to])	Converts an expr to a string of GLSL code	
groebner(F, *gens, **args)	Computes the reduced Groebner basis for a set of poly-	
	nomials.	
<pre>ground_roots(f, *gens, **args)</pre>	Compute roots of f by factorization in the ground do-	
	main.	
<pre>group(seq[, multiple])</pre>	Splits a sequence into a list of lists of equal, adjacent	
	elements.	
gruntz(e, z, z0[, dir])	Compute the limit of $e(z)$ at the point $z0$ using the Gruntz	
	algorithm.	
hadamard_product(*matrices)	Return the elementwise (aka Hadamard) product of ma-	
	trices.	
half_gcdex(f, g, *gens, **args)	Half extended Euclidean algorithm of f and g.	
<pre>hankel_transform(f, r, k, nu, **hints)</pre>	Compute the Hankel transform of f, defined as	
has_dups(seq)	Return True if there are any duplicate elements in seq.	
has_variety(seq)	Return True if there are any different elements in seq.	
hermite_poly(n[, x, polys])	Generates the Hermite polynomial $H_n(x)$ .	
hermite_prob_poly(n[, x, polys])	Generates the probabilist's Hermite polynomial $He_n(x)$ .	
hessian(f, varlist[, constraints])	Compute Hessian matrix for a function f wrt parame-	
	ters in varlist which may be given as a sequence or a	
	row/column vector.	
homogeneous_order(eq, *symbols)	Returns the order $n$ if $g$ is homogeneous and None if it is	
	not homogeneous.	
horner(f, *gens, **args)	Rewrite a polynomial in Horner form.	
<pre>hyperexpand(f[, allow_hyper, rewrite, place])</pre>	Expand hypergeometric functions.	
hypersimilar(f, g, k)	Returns True if f and g are hyper-similar.	
hypersimp(f, k)	Given combinatorial term f(k) simplify its consecutive	
	term ratio i.e. $f(k+1)/f(k)$ .	
idiff(eq, y, x[, n])	Return $dy/dx$ assuming that $eq == 0$ .	
ifft(seq[, dps])	Performs the Discrete Fourier Transform ( <b>DFT</b> ) in the	
	complex domain.	
ifwht(seq)	Performs the Walsh Hadamard Transform (WHT), and	
	uses Hadamard ordering for the sequence.	
	continues on next page	

Table 67 – continued from previous page

Table 07 - Continue	a nom providuo pago
igcd(*args)	Computes nonnegative integer greatest common divisor.
ilcm(*args)	Computes integer least common multiple.
<pre>imageset(*args)</pre>	Return an image of the set under transformation f.
<pre>init_printing([pretty_print, order,])</pre>	Initializes pretty-printer depending on the environment.
init_session([ipython, pretty_print, order,])	Initialize an embedded IPython or Python session.
integer_log(y, x)	Returns (e, bool) where e is the largest nonnegative
integer_rog(y, x)	integer such that $ y  \ge  x^e $ and bool is True if \$y =
	$ x  =  x $ and $ y  \ge  x $ and $ y  =  x $
integer nthroat(v n)	Return a tuple containing $x = floor(y^{**}(1/n))$ and a
<pre>integer_nthroot(y, n)</pre>	
	boolean indicating whether the result is exact (that is,
into mate (f	whether $x^* = y$ .
<pre>integrate(f, var,)</pre>	
internaling to a second 2(0,00)	T
interactive_traversal(expr)	Traverse a tree asking a user which branch to choose.
<pre>interpolate(data, x)</pre>	Construct an interpolating polynomial for the data points
	evaluated at point x (which can be symbolic or numeric).
$interpolating_poly(n, x[, X, Y])$	Construct Lagrange interpolating polynomial for n data
	points.
$interpolating\_spline(d, x, X, Y)$	Return spline of degree $d$ , passing through the given $X$
	and Y values.
<pre>intersecting_product(a, b)</pre>	Returns the intersecting product of given sequences.
<pre>intersection(*entities[, pairwise])</pre>	The intersection of a collection of GeometryEntity in-
	stances.
<pre>intervals(F[, all, eps, inf, sup, strict,])</pre>	Compute isolating intervals for roots of f.
<pre>intt(seq, prime)</pre>	Performs the Number Theoretic Transform (NTT),
	which specializes the Discrete Fourier Transform ( <b>DFT</b> )
	over quotient ring $\mathbb{Z}/p\mathbb{Z}$ for prime $p$ instead of complex
	numbers $C$ .
inv_quick(M)	Return the inverse of M, assuming that either there are
	lots of zeros or the size of the matrix is small.
<pre>inverse_cosine_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse cosine
	transform of $F$ , defined as
<pre>inverse_fourier_transform(F, k, x, **hints)</pre>	Compute the unitary, ordinary-frequency inverse Fourier
(, , , , , , , , , , , , , , , , , , ,	transform of $F$ , defined as
<pre>inverse_hankel_transform(F, k, r, nu, **hints)</pre>	Compute the inverse Hankel transform of $F$ defined as
inverse_laplace_transform(F, s, t[, plane])	Compute the inverse Laplace transform of $F(s)$ , defined
inverse_raprace_transform(r, s, t[, plane])	as
<pre>inverse_mellin_transform(F, s, x, strip, **hints)</pre>	Compute the inverse Mellin transform of $F(s)$ over the
	fundamental strip given by strip=(a, b).
<pre>inverse_mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with
THVCT SC_MODIUS_CLANSIOLM(SCYL, SUUSCIJ)	indices of sequence as bitmasks.
<pre>inverse_sine_transform(F, k, x, **hints)</pre>	•
Inverse_sine_transform(r, k, x, ***mins)	Compute the unitary, ordinary-frequency inverse sine
in	transform of $F$ , defined as
invert(f, g, *gens, **args)	Invert f modulo g when possible.
is_abundant(n)	Returns True if n is an abundant number, else False.
<pre>is_amicable(m, n)</pre>	Returns True if the numbers $m$ and $n$ are "amicable", else
(6.4)	False.
<pre>is_convex(f, *syms[, domain])</pre>	Determines the convexity of the function passed in the
	argument.
<pre>is_decreasing(expression[, interval, symbol])</pre>	Return whether the function is decreasing in the given
	interval.
is_deficient(n)	Returns True if n is a deficient number, else False.
	continues on next page

Table 67 – continued from previous page

Table 67 - contin	lued from previous page
<pre>is_increasing(expression[, interval, symbol])</pre>	Return whether the function is increasing in the given interval.
<pre>is_mersenne_prime(n)</pre>	Returns True if n is a Mersenne prime, else False.
<pre>is_monotonic(expression[, interval, symbol])</pre>	Return whether the function is monotonic in the given
	interval.
<pre>is_nthpow_residue(a, n, m)</pre>	Returns True if $x^*n == a \pmod{m}$ has solutions.
is_perfect(n)	Returns True if n is a perfect number, else False.
<pre>is_primitive_root(a, p)</pre>	Returns True if a is a primitive root of p.
is_quad_residue(a, p)	Returns True if a (mod p) is in the set of squares mod p,
	i.e a % p in set( $[i**2 \% p \text{ for i in range}(p)]$ ).
<pre>is_strictly_decreasing(expression[,])</pre>	Return whether the function is strictly decreasing in the given interval.
<pre>is_strictly_increasing(expression[,])</pre>	Return whether the function is strictly increasing in the
5(v r v v v v v v v v v v v v v v v v v v	given interval.
is_zero_dimensional(F, *gens, **args)	Checks if the ideal generated by a Groebner basis is zero-
	dimensional.
<pre>isolate(alg[, eps, fast])</pre>	Find a rational isolating interval for a real algebraic num-
	ber.
isprime(n)	Test if n is a prime number (True) or not (False).
itermonomials(variables, max_degrees[,])	max_degrees and min_degrees are either both integers or both lists.
<pre>jacobi_normalized(n, a, b, x)</pre>	Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.
<pre>jacobi_poly(n, a, b[, x, polys])</pre>	Generates the Jacobi polynomial $P_n^{(a,b)}(x)$ .
jacobi_symbol(m, n)	Returns the Jacobi symbol $(m/n)$ .
jn_zeros(n, k[, method, dps])	Zeros of the spherical Bessel function of the first kind.
jordan_cell(eigenval, n)	Create a Jordan block:
jscode(expr[, assign_to])	Converts an expr to a string of javascript code
julia_code(expr[, assign_to])	Converts <i>expr</i> to a string of Julia code.
kronecker_product(*matrices)	The Kronecker product of two or more arguments.
kroneckersimp(expr)	Simplify expressions with KroneckerDelta.
laguerre_poly(n[, x, alpha, polys])	Generates the Laguerre polynomial $L_n^{(alpha)}(x)$ .
lambdify(args, expr[, modules, printer,])	Convert a SymPy expression into a function that allows
	for fast numeric evaluation.
<pre>laplace_transform(f, t, s[, legacy_matrix])</pre>	Compute the Laplace Transform $F(s)$ of $f(t)$ ,
lcm(f[,g])	Compute LCM of f and g.
<pre>lcm_list(seq, *gens, **args)</pre>	Compute LCM of a list of polynomials.
<pre>legendre_poly(n[, x, polys])</pre>	Generates the Legendre polynomial $P_n(x)$ .
legendre_symbol(a, p)	Returns the Legendre symbol $(a/p)$ .
limit(e, z, z0[, dir])	Computes the limit of $e(z)$ at the point $z0$ .
<pre>limit_seq(expr[, n, trials])</pre>	Finds the limit of a sequence as index n tends to infinity.
<pre>line_integrate(field, Curve, variables)</pre>	Compute the line integral.
<pre>linear_eq_to_matrix(equations, *symbols)</pre>	Converts a given System of Equations into Matrix form.
linsolve(system, *symbols)	Solve system of \$N\$ linear equations with \$M\$ vari-
	ables; both underdetermined and overdetermined sys-
	tems are supported.
list2numpy(l[, dtype])	Converts Python list of SymPy expressions to a NumPy array.
<pre>logcombine(expr[, force])</pre>	Takes logarithms and combines them using the following
	rules:
<pre>maple_code(expr[, assign_to])</pre>	Converts expr to a string of Maple code.
mathematica_code(expr, **settings)	Converts an expr to a string of the Wolfram Mathematica
	code
	continues on next page

Table 67 – continued from previous page

	ed from previous page
<pre>matrix2numpy(m[, dtype])</pre>	Converts SymPy's matrix to a NumPy array.
${\tt matrix\_multiply\_elementwise}(A,B)$	Return the Hadamard product (elementwise product) of A and B
<pre>matrix_symbols(expr)</pre>	
<pre>maximum(f, symbol[, domain])</pre>	Returns the maximum value of a function in the given domain.
<pre>mellin_transform(f, x, s, **hints)</pre>	Compute the Mellin transform $F(s)$ of $f(x)$ ,
<pre>memoize_property(propfunc) mersenne_prime_exponent(nth)</pre>	Property decorator that caches the value of potentially expensive <i>propfunc</i> after the first evaluation.  Returns the exponent i for the nth Mersenne prime
	(which has the form $2^i - 1$ ).
<pre>minimal_polynomial(ex[, x, compose, polys,])</pre>	Computes the minimal polynomial of an algebraic element.
minimum(f, symbol[, domain])	Returns the minimum value of a function in the given domain.
<pre>minpoly(ex[, x, compose, polys, domain])</pre>	This is a synonym for minimal_polynomial().
<pre>mobius_transform(seq[, subset])</pre>	Performs the Mobius Transform for subset lattice with indices of sequence as bitmasks.
mod_inverse(a, m)	Return the number $c$ such that, $a$ times $c = 1$ pmod $m$ where $c$ has the same sign as $m$ .
monic(f, *gens, **args)	Divide all coefficients of f by LC(f).
<pre>multiline_latex(lhs, rhs[, terms_per_line,])</pre>	This function generates a LaTeX equation with a multiline right-hand side in an align*, eqnarray or IEEEeqnarray environment.
<pre>multinomial_coefficients(m, n)</pre>	Return a dictionary containing pairs {(k1,k2,,km) : C_kn} where C_kn are multinomial coefficients such that n=k1+k2++km.
<pre>multiplicity(p, n)</pre>	Find the greatest integer m such that p**m divides n.
n_order(a, n)	Returns the order of a modulo n.
<pre>nextprime(n[, ith])</pre>	Return the ith prime greater than n.
<pre>nfloat(expr[, n, exponent, dkeys])</pre>	Make all Rationals in expr Floats except those in exponents (unless the exponents flag is set to True) and those in undefined functions.
nonlinsolve(system, *symbols)	Solve system of \$N\$ nonlinear equations with \$M\$ variables, which means both under and overdetermined systems are supported.
<pre>not_empty_in(finset_intersection, *syms)</pre>	Finds the domain of the functions in finset_intersection in which the finite_set is not-empty.
<pre>npartitions(n[, verbose])</pre>	Calculate the partition function $P(n)$ , i.e. the number of ways that n can be written as a sum of positive integers.
<pre>nroots(f[, n, maxsteps, cleanup])</pre>	Compute numerical approximations of roots of f.
nsimplify(expr[, constants, tolerance,])	Find a simple representation for a number or, if there are free symbols or if rational=True, then replace Floats with their Rational equivalents.
nsolve(*args[, dict])	Solve a nonlinear equation system numerically: nsolve(f, [args,] x0, modules=['mpmath'], **kwargs).
<pre>nth_power_roots_poly(f, n, *gens, **args)</pre>	Construct a polynomial with n-th powers of roots of f.
nthroot_mod(a, n, p[, all_roots])	Find the solutions to $x^*n = a \mod p$ .
	continues on next page

Table 67 – continued from previous page

Table 67 – continued from previous page	
ntt(seq, prime)	Performs the Number Theoretic Transform ( <b>NTT</b> ), which specializes the Discrete Fourier Transform ( <b>DFT</b> ) over quotient ring $\mathbb{Z}/p\mathbb{Z}$ for prime $p$ instead of complex numbers $C$ .
<pre>numbered_symbols([prefix, cls, start, exclude])</pre>	Generate an infinite stream of Symbols consisting of a prefix and increasing subscripts provided that they do not occur in exclude.
numer(expr)	
octave_code(expr[, assign_to])	Converts <i>expr</i> to a string of Octave (or Matlab) code.
ode_order(expr, func)	Returns the order of a given differential equation with respect to func.
ones(*args, **kwargs)	Returns a matrix of ones with rows rows and cols columns; if cols is omitted a square matrix will be returned.
ordered(seq[, keys, default, warn])	Return an iterator of the seq where keys are used to break ties in a conservative fashion: if, after applying a key, there are no ties then no other keys will be computed.
<pre>pager_print(expr, **settings)</pre>	Prints expr using the pager, in pretty form.
<pre>parallel_poly_from_expr(exprs, *gens, **args)</pre>	Construct polynomials from expressions.
<pre>parse_expr(s[, local_dict, transformations,])</pre>	Converts the string s to a SymPy expression, in local_dict.
<pre>pde_separate(eq, fun, sep[, strategy])</pre>	Separate variables in partial differential equation either by additive or multiplicative separation approach.
pde_separate_add(eq, fun, sep)	Helper function for searching additive separable solutions.
<pre>pde_separate_mul(eq, fun, sep)</pre>	Helper function for searching multiplicative separable solutions.
pdiv(f, g, *gens, **args)	Compute polynomial pseudo-division of f and g.
<pre>pdsolve(eq[, func, hint, dict, solvefun])</pre>	Solves any (supported) kind of partial differential equation.
per(matexpr)	Matrix Permanent
<pre>perfect_power(n[, candidates, big, factor])</pre>	Return (b, e) such that $n == b^*e$ if n is a unique perfect power with $e > 1$ , else False (e.g. 1 is not a perfect power).
<pre>periodicity(f, symbol[, check])</pre>	Tests the given function for periodicity in the given symbol.
<pre>permutedims(expr[, perm, index_order_old,])</pre>	Permutes the indices of an array.
pexquo(f, g, *gens, **args)	Compute polynomial exact pseudo-quotient of f and g.
<pre>piecewise_exclusive(expr, *[, skip_nan, deep])</pre>	Rewrite Piecewise with mutually exclusive conditions.
<pre>piecewise_fold(expr[, evaluate])</pre>	Takes an expression containing a piecewise function and returns the expression in piecewise form.
<pre>plot(*args[, show])</pre>	Plots a function of a single variable as a curve.
<pre>plot_implicit(expr[, x_var, y_var,])</pre>	A plot function to plot implicit equations / inequalities.
<pre>plot_parametric(*args[, show])</pre>	Plots a 2D parametric curve.
<pre>polarify(eq[, subs, lift])</pre>	Turn all numbers in eq into their polar equivalents (under the standard choice of argument).
pollard_pm1(n[, B, a, retries, seed])	Use Pollard's p-1 method to try to extract a nontrivial factor of n.
pollard_rho(n[, s, a, retries, seed,])	Use Pollard's rho method to try to extract a nontrivial factor of n.
poly(expr, *gens, **args)	Efficiently transform an expression into a polynomial.
	continues on next page

Table 67 – continued from previous page

iable 6/ – continue	ed from previous page
<pre>poly_from_expr(expr, *gens, **args)</pre>	Construct a polynomial from an expression.
posify(eq)	Return eq (with generic symbols made positive) and a dictionary containing the mapping between the old and new symbols.
postfixes(seq)	Generate all postfixes of a sequence.
postorder_traversal(node[, keys])	Do a postorder traversal of a tree.
powdenest(eq[, force, polar])	Collect exponents on powers as assumptions allow.
powsimp(expr[, deep, combine, force, measure])	Reduce expression by combining powers with similar bases and exponents.
<pre>pprint(expr, **kwargs)</pre>	Prints expr in pretty form.
<pre>pprint_try_use_unicode()</pre>	See if unicode output is available and leverage it if possible
<pre>pprint_use_unicode([flag])</pre>	Set whether pretty-printer should use unicode by default
pquo(f, g, *gens, **args)	Compute polynomial pseudo-quotient of f and g.
<pre>prefixes(seq)</pre>	Generate all prefixes of a sequence.
<pre>prem(f, g, *gens, **args)</pre>	Compute polynomial pseudo-remainder of f and g.
<pre>pretty_print(expr, **kwargs)</pre>	Prints expr in pretty form.
<pre>preview(expr[, output, viewer, euler,])</pre>	View expression or LaTeX markup in PNG, DVI, PostScript or PDF form.
prevprime(n)	Return the largest prime smaller than n.
prime(nth)	Return the nth prime, with the primes indexed as $prime(1) = 2$ , $prime(2) = 3$ , etc.
<pre>prime_decomp(p[, T, ZK, dK, radical])</pre>	Compute the decomposition of rational prime $p$ in a number field.
<pre>prime_valuation(I, P)</pre>	Compute the $P$ -adic valuation for an integral ideal $I$ .
<pre>primefactors(n[, limit, verbose])</pre>	Return a sorted list of n's prime factors, ignoring multi- plicity and any composite factor that remains if the limit was set too low for complete factorization.
<pre>primerange(a[, b])</pre>	Generate a list of all prime numbers in the range [2, a), or [a, b).
<pre>primitive(f, *gens, **args)</pre>	Compute content and the primitive form of f.
<pre>primitive_element(extension[, x, ex, polys])</pre>	Find a single generator for a number field given by several generators.
<pre>primitive_root(p)</pre>	Returns the smallest primitive root or None.
<pre>primorial(n[, nth])</pre>	Returns the product of the first n primes (default) or the primes less than or equal to n (when nth=False).
<pre>print_ccode(expr, **settings)</pre>	Prints C representation of the given expression.
<pre>print_fcode(expr, **settings)</pre>	Prints the Fortran representation of the given expression.
<pre>print_glsl(expr, **settings)</pre>	Prints the GLSL representation of the given expression.
<pre>print_gtk(x[, start_viewer])</pre>	Print to Gtkmathview, a gtk widget capable of rendering MathML.
<pre>print_jscode(expr, **settings)</pre>	Prints the Javascript representation of the given expression.
<pre>print_latex(expr, **settings)</pre>	Prints LaTeX representation of the given expression.
<pre>print_maple_code(expr, **settings)</pre>	Prints the Maple representation of the given expression.
<pre>print_mathml(expr[, printer])</pre>	Prints a pretty representation of the MathML code for expr.
<pre>print_python(expr, **settings)</pre>	Print output of python() function
<pre>print_rcode(expr, **settings)</pre>	Prints R representation of the given expression.
<pre>print_tree(node[, assumptions])</pre>	Prints a tree representation of "node".
prod(a[, start])	Return product of elements of a. Start with int 1 so if only
	continues on next nage

Table 67 – continued from previous page

Table 67 – Continue	ed from previous page
<pre>product(*args, **kwargs)</pre>	Compute the product.
<pre>proper_divisor_count(n[, modulus])</pre>	Return the number of proper divisors of n.
<pre>proper_divisors(n[, generator])</pre>	Return all divisors of n except n, sorted by default.
public(obj)	Append obj's name to globalall variable (call
	site).
pycode(expr, **settings)	Converts an expr to a string of Python code
<pre>python(expr, **settings)</pre>	Return Python interpretation of passed expression (can
•	be passed to the exec() function without any modifica-
	tions)
<pre>quadratic_congruence(a, b, c, p)</pre>	Find the solutions to $a x**2 + b x + c = 0 \mod p$ .
quadratic_residues(p)	Returns the list of quadratic residues.
quo(f, g, *gens, **args)	Compute polynomial quotient of f and g.
rad(d)	Return the radian value for the given degrees ( $pi = 180$
	degrees).
<pre>radsimp(expr[, symbolic, max_terms])</pre>	Rationalize the denominator by removing square roots.
<pre>randMatrix(r[, c, min, max, seed,])</pre>	Create random matrix with dimensions $\mathbf{r} \times \mathbf{c}$ .
<pre>random_poly(x, n, inf, sup[, domain, polys])</pre>	Generates a polynomial of degree n with coefficients in
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	[inf, sup].
randprime(a, b)	Return a random prime number in the range [a, b).
rational_interpolate(data, degnum[, X])	Returns a rational interpolation, where the data points
	are element of any integral domain.
ratsimp(expr)	Put an expression over a common denominator, cancel
• ( 1 /	and reduce.
<pre>ratsimpmodprime(expr, G, *gens[, quick,])</pre>	Simplifies a rational expression expr modulo the prime
	ideal generated by G.
<pre>rcode(expr[, assign_to])</pre>	Converts an expr to a string of r code
rcollect(expr, *vars)	Recursively collect sums in an expression.
real_root(arg[, n, evaluate])	Return the real <i>n</i> 'th-root of <i>arg</i> if possible.
real_roots(f[, multiple])	Return a list of real roots with multiplicities of f.
reduce_abs_inequalities(exprs, gen)	Reduce a system of inequalities with nested absolute val-
	ues.
<pre>reduce_abs_inequality(expr, rel, gen)</pre>	Reduce an inequality with nested absolute values.
<pre>reduce_inequalities(inequalities[, symbols])</pre>	Reduce a system of inequalities with rational coeffi-
. 1	cients.
reduced(f, G, *gens, **args)	Reduces a polynomial f modulo a set of polynomials G.
refine(expr[, assumptions])	Simplify an expression using assumptions.
refine_root(f, s, t[, eps, steps, fast,])	Refine an isolating interval of a root to the given preci-
	sion.
register_handler(key, handler)	Register a handler in the ask system.
rem(f, g, *gens, **args)	Compute polynomial remainder of f and g.
remove_handler(key, handler)	Removes a handler from the ask system.
reshape(seq, how)	Reshape the sequence according to the template in how.
residue(expr, x, x0)	Finds the residue of expr at the point x=x0.
resultant(f, g, *gens[, includePRS])	Compute resultant of f and g.
ring(symbols, domain[, order])	Construct a polynomial ring returning (ring, x_1, .
2.7	, x_n).
root(arg, n[, k, evaluate])	Returns the <i>k</i> -th <i>n</i> -th root of arg.
rootof(f, x[, index, radicals, expand])	An indexed root of a univariate polynomial.
roots(f, *gens[, auto, cubics, trig,])	Computes symbolic roots of a univariate polynomial.
rot_axis1(theta)	Returns a rotation matrix for a rotation of theta (in radi-
	ans) about the 1-axis.
	continues on next nage

Table 67 – continued from previous page

Table 67 – continu	ed from previous page
rot_axis2(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 2-axis.
rot_axis3(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 3-axis.
<pre>rot_ccw_axis1(theta)</pre>	Returns a rotation matrix for a rotation of theta (in radians) about the 1-axis.
rot_ccw_axis2(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 2-axis.
rot_ccw_axis3(theta)	Returns a rotation matrix for a rotation of theta (in radians) about the 3-axis.
<pre>rot_givens(i, j, theta[, dim])</pre>	Returns a a Givens rotation matrix, a a rotation in the plane spanned by two coordinates axes.
rotations(s[, dir])	Return a generator giving the items in s as list where each subsequent list has the items rotated to the left (default) or right (dir=-1) relative to the previous list.
<pre>round_two(T[, radicals])</pre>	Zassenhaus's "Round 2" algorithm.
rsolve(f, y[, init])	Solve univariate recurrence with rational coefficients.
rsolve_hyper(coeffs, f, n, **hints)	Given linear recurrence operator $operatorname\{L\}$ of order $k$ with polynomial coefficients and inhomogeneous equation $operatorname\{L\}$ $y = f$ we seek for all hypergeometric solutions over field $K$ of characteristic zero.
rsolve_poly(coeffs, f, n[, shift])	Given linear recurrence operator <i>operatorname{L}</i> of order $k$ with polynomial coefficients and inhomogeneous equation <i>operatorname{L}</i> $y = f$ , where $f$ is a polynomial, we seek for all polynomial solutions over field $K$ of characteristic zero.
rsolve_ratio(coeffs, f, n, **hints)	Given linear recurrence operator <i>operatorname{L}</i> of order $k$ with polynomial coefficients and inhomogeneous equation <i>operatorname{L}</i> $y = f$ , where $f$ is a polynomial, we seek for all rational solutions over field $K$ of characteristic zero.
<pre>run_timed(params, parameter_name)</pre>	generates an instance of TestBabyModel with given parameters, sympifies it, and follows the sequence of steps
<pre>rust_code(expr[, assign_to])</pre>	Converts an expr to a string of Rust code
<pre>satisfiable(expr[, algorithm, all_models,])</pre>	Check satisfiability of a propositional sentence.
<pre>separatevars(expr[, symbols, dict, force])</pre>	Separates variables in an expression, if possible.
sequence(seq[, limits])	Returns appropriate sequence object.
series(expr[, x, x0, n, dir])	Series expansion of expr around point $x = x0$ .
seterr([divide])	Should SymPy raise an exception on 0/0 or return a nan?
sfield(exprs, *symbols, **options)	Construct a field deriving generators and domain from options and input expressions.
shape()	Return the shape of the <i>expr</i> as a tuple.
sift(seq, keyfunc[, binary])	Sift the sequence, seq according to keyfunc.
signsimp(expr[, evaluate])	Make all Add sub-expressions canonical wrt sign.
simplify(expr[, ratio, measure, rational,])	Simplifies the given expression.
<pre>simplify_logic(expr[, form, deep, force,])</pre>	This function simplifies a boolean function to its simplified version in SOP or POS form.
<pre>sine_transform(f, x, k, **hints)</pre>	Compute the unitary, ordinary-frequency sine transform of $f$ , defined as
<pre>singularities(expression, symbol[, domain])</pre>	Find singularities of a given function.
singularityintegrate(f, x)	This function handles the indefinite integrations of Singularity functions.
	continues on next nage

Table 67 – continued from previous page

Table 67 – Continue	ed from previous page
<pre>smtlib_code(expr[, auto_assert,])</pre>	Converts expr to a string of smtlib code.
solve(f, *symbols, **flags)	Algebraically solves equations and systems of equations.
<pre>solve_linear(lhs[, rhs, symbols, exclude])</pre>	Return a tuple derived from $f = lhs - rhs$ that is one of the following: $(0, 1)$ , $(0, 0)$ , (symbol, solution), $(n, d)$ .
<pre>solve_linear_system(system, *symbols, **flags)</pre>	Solve system of \$N\$ linear equations with \$M\$ vari-
	ables, which means both under- and overdetermined systems are supported.
<pre>solve_linear_system_LU(matrix, syms)</pre>	Solves the augmented matrix system using LUsolve and returns a dictionary in which solutions are keyed to the symbols of <i>syms</i> as ordered.
<pre>solve_poly_inequality(poly, rel)</pre>	Solve a polynomial inequality with rational coefficients.
<pre>solve_poly_system(seq, *gens[, strict])</pre>	Return a list of solutions for the system of polynomial equations or else None.
<pre>solve_rational_inequalities(eqs)</pre>	Solve a system of rational inequalities with rational coefficients.
<pre>solve_triangulated(polys, *gens, **args)</pre>	Solve a polynomial system using Gianni-Kalkbrenner algorithm.
<pre>solve_undetermined_coeffs(equ, coeffs,)</pre>	Solve a system of equations in \$k\$ parameters that is formed by matching coefficients in variables coeffs that are on factors dependent on the remaining variables (or those given explicitly by syms.
<pre>solve_univariate_inequality(expr, gen[,])</pre>	Solves a real univariate inequality.
solveset(f[, symbol, domain])	Solves a given inequality or equation with set as output
speed_accuracy_test(params, parameter_name, p)	performs a run_timed execution with given parameters, takes the returned values, and measures the effect of increasing and
sqf(f, *gens, **args)	Compute square-free factorization of f.
sqf_list(f, *gens, **args)	Compute a list of square-free factors of f.
sqf_norm(f, *gens, **args)	Compute square-free norm of f.
<pre>sqf_part(f, *gens, **args)</pre>	Compute square-free part of f.
sqrt(arg[, evaluate])	Returns the principal square root.
<pre>sqrt_mod(a, p[, all_roots])</pre>	Find a root of $x^**2 = a \mod p$ .
<pre>sqrt_mod_iter(a, p[, domain])</pre>	Iterate over solutions to $x^{**}2 = a \mod p$ .
<pre>sqrtdenest(expr[, max_iter])</pre>	Denests sqrts in an expression that contain other square roots if possible, otherwise returns the expr unchanged.
<pre>sring(exprs, *symbols, **options)</pre>	Construct a ring deriving generators and domain from options and input expressions.
<pre>stationary_points(f, symbol[, domain])</pre>	Returns the stationary points of a function (where derivative of the function is 0) in the given domain.
sturm(f, *gens, **args)	Compute Sturm sequence of f.
subresultants(f, g, *gens, **args)	Compute subresultant PRS of f and g.
subsets(seq[, k, repetition])	Generates all $k$ -subsets (combinations) from an $n$ - element set, seq.
<pre>substitution(system, symbols[, result,])</pre>	Solves the <i>system</i> using substitution method.
<pre>summation(f, *symbols, **kwargs)</pre>	Compute the summation of f with respect to symbols.
<pre>swinnerton_dyer_poly(n[, x, polys])</pre>	Generates n-th Swinnerton-Dyer polynomial in $x$ .
symarray(prefix, shape, **kwargs)	Create a numpy ndarray of symbols (as an object array).
symbols(names, *[, cls])	Transform strings into instances of Symbol class.
<pre>symmetric_poly(n, *gens[, polys])</pre>	Generates symmetric polynomial of order <i>n</i> .
<pre>symmetrize(F, *gens, **args)</pre>	Rewrite a polynomial in terms of elementary symmetric
	polynomials.
	continues on next nage

Table 67 – continued from previous page

	ed from previous page
<pre>sympify(a[, locals, convert_xor, strict,])</pre>	Converts an arbitrary expression to a type that can be
	used inside SymPy.
take(iter, n)	Return n items from iter iterator.
tensorcontraction(array, *contraction_axes)	Contraction of an array-like object on the specified axes.
tensordiagonal(array, *diagonal_axes)	Diagonalization of an array-like object on the specified axes.
tensorproduct(*args)	Tensor product among scalars or array-like objects.
terms_gcd(f, *gens, **args)	Remove GCD of terms from f.
textplot(expr, a, b[, W, H])	Print a crude ASCII art plot of the SymPy expression 'expr' (which should contain a single symbol, e.g. x or something else) over the interval [a, b].
threaded(func)	Apply func to subelements of an object, including Add.
<pre>timed(func[, setup, limit])</pre>	Adaptively measure execution time of a function.
to_cnf(expr[, simplify, force])	Convert a propositional logical sentence expr to conjunctive normal form: ((A   ~B  ) & (B   C  ) &).
to_dnf(expr[, simplify, force])	Convert a propositional logical sentence expr to disjunctive normal form: ((A & $\sim$ B &)   (B & C &)  ).
to_nnf(expr[, simplify])	Converts expr to Negation Normal Form (NNF).
to_number_field(extension[, theta, gen, alias])	Express one algebraic number in the field generated by another.
together(expr[, deep, fraction])	Denest and combine rational expressions using symbolic methods.
<pre>topological_sort(graph[, key])</pre>	Topological sort of graph's vertices.
total_degree(f, *gens)	Return the total_degree of f in the given variables.
trace(expr)	Trace of a Matrix.
trailing(n)	Count the number of trailing zero digits in the binary
	representation of n, i.e. determine the largest power of 2 that divides n.
<pre>trigsimp(expr[, inverse])</pre>	Returns a reduced expression by using known trig identities.
trunc(f, p, *gens, **args)	Reduce f modulo a constant p.
unbranched_argument(arg)	Returns periodic argument of arg with period as infinity.
unflatten(iter[, n])	Group iter into tuples of length n.
<pre>unpolarify(eq[, subs, exponents_only])</pre>	If $p$ denotes the projection from the Riemann surface of the logarithm to the complex line, return a simplified version $eq'$ of $eq$ such that $p(eq') = p(eq)$ .
<pre>use(expr, func[, level, args, kwargs])</pre>	Use func to transform expr at the given level.
var(names, **args)	Create symbols and inject them into the global namespace.
<pre>variations(seq, n[, repetition])</pre>	Returns an iterator over the n-sized variations of seq (size N).
vfield(symbols, domain[, order])	Construct new rational function field and inject generators into global namespace.
<pre>viete(f[, roots])</pre>	Generate Viete's formulas for f.
<pre>vring(symbols, domain[, order])</pre>	Construct a polynomial ring and inject x_1,, x_n into the global namespace.
<pre>wronskian(functions, var[, method])</pre>	Compute Wronskian for [] of functions
xfield(symbols, domain[, order])	Construct new rational function field returning (field,
	(x1,, xn)).
	continues on next page

Table 67 – continued from previous page

<pre>xring(symbols, domain[, order])</pre>	Construct a polynomial ring returning (ring, $(x_1, \dots, x_n)$ ).
xthreaded(func)	Apply func to subelements of an object, excluding Add.
zeros(*args, **kwargs)	Returns a matrix of zeros with rows rows and cols columns; if cols is omitted a square matrix will be returned.

## **Classes**

Abs(arg)	Return the absolute value of the argument.
AccumBounds	alias of AccumulationBounds
Add(*args[, evaluate, _sympify])	Expression representing addition operation for algebraic
	group.
Adjoint(*args, **kwargs)	The Hermitian adjoint of a matrix expression.
<pre>AlgebraicField(dom, *ext[, alias])</pre>	Algebraic number field QQ(a)
AlgebraicNumber(expr[, coeffs, alias])	Class for representing algebraic numbers in SymPy.
And(*args)	Logical AND function.
AppliedPredicate(predicate, *args)	The class of expressions resulting from applying Predicate to the arguments.
Array	alias of ImmutableDenseNDimArray
AssumptionsContext	Set containing default assumptions which are applied to
	the ask() function.
Atom(*args)	A parent class for atomic things.
AtomicExpr(*args)	A parent class for object which are both atoms and Exprs.
AutoSympy(model)	
Basic(*args)	Base class for all SymPy objects.
BlockDiagMatrix(*mats)	A sparse matrix with block matrices along its diagonals
BlockMatrix(*args, **kwargs)	A BlockMatrix is a Matrix comprised of other matrices.
CRootOf	alias of ComplexRootOf
Chi(z)	Cosh integral.
Ci(z)	Cosine integral.
Circle(*args, **kwargs)	A circle in space.
Complement(a, b[, evaluate])	Represents the set difference or relative complement of a set with another set.
<pre>ComplexField([prec, dps, tol])</pre>	Complex numbers up to the given precision.
ComplexRegion(sets[, polar])	Represents the Set of all Complex Numbers.
ComplexRootOf(f, x[, index, radicals, expand])	Represents an indexed complex root of a polynomial.
<pre>ConditionSet(sym, condition[, base_set])</pre>	Set of elements which satisfies a given condition.
Contains(x, s)	Asserts that x is an element of the set S.
CoordMap(var_vector, eq_duals, ineq_duals,)	
CosineTransform(*args)	Class representing unevaluated cosine transforms.
Curve(function, limits)	A curve in space.
DeferredVector(name, **assumptions)	A vector whose components are deferred (e.g. for use with lambdify).
DenseNDimArray(*args, **kwargs)	•
Derivative(expr, *variables, **kwargs)	Carries out differentiation of the given expression with respect to symbols.
	continues on next page

Table 68 – continued from previous page

Determinant (mat   DiagNatrix(vector)   Turn a vector into a diagonal matrix.	Table 68 – continue	d from previous page
DiagonalMatrix(#args, **kwargs)  DiagonalOf(*args, **kwargs)  DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k])  DiracDelta function and its derivatives.  Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain()  Dummy(mobil are acid unique, even if they have the same name:  Path(path)  DiracDelta function and its derivatives.  Domain()  Dupmay(mobil union (also known as the external disjoint union) of a finite nunique, even if they have the same name:  Path(path)  DiracDelta function and its derivatives.  Represents the disjoint union (also known as the external disjoint union) of a finite nunique, even if they have the same name:  Path(path)  DiracDelta function and its derivatives.  Represents the disjoint union (also known as the external disjoint union) of a finite nunique, even if they have the same name:  Path(path)  DiracDelta function and its derivatives.  Represents and indemains in the polys domains system.  Dot product of vector matrices  Dummy(mobil are acid unique, even if they have the same name:  Path(path)  Manipulate expressions using paths.  The classical exponential integral.  The classical exponential integral.  Path(path)  An elliptical GeometryEntity.  alias of Equality  An equal relation between two objects.  Equivalence relation.  ExpressionDomain()  An elliptical GeometryEntity.  An elliptical GeometryEntit	Determinant(mat)	Matrix Determinant
behaves as though all off-diagonal elements, M[i, j] where i = j, are zero.  DiagonalOf(*args, **kwargs)  DiagonalOf(*m) will create a matrix expression that is equivalent to the diagonal of M, represented as a single column matrix.  Dict(*args)  Wrapper around the builtin dict object.  DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  Domain()  Superclass for all domains in the polys domains system Dot product of vector matrices  Domain()  Domy(Iname, dummy_index))  Dummy(Iname, dummy_index))  EPath(path)  Epath(path)  Epath(path)  Equivalent(*args)  Equivalent(*args)  Equivalent(*args)  Expr(*args)  Equivalent(*args)  Equivalent(*args)  Equivalent(*args)  Equivalent(*args)  Expr(*args)  FF_gmpy  alias of FiniteField  FallingFactorial(x, k)  FallingFactorial(x, k)  Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric])  FiniteSet(*args, **kwargs)  FrourierTransform(*args)  FrourierTransform(*args)  FactionField(domain_or_field[, symbols, order])  Function(*args)  Base class for applied mathematical functions  Function(*args)  Base class for function classes.  Cass representing unevaluated Fourier transforms.  A class for prime order GF(p)  FiniteSet(*args, **kwargs)  FractionField(domain_or_field[, symbols, order])  Finite field or prime order GF(p)  FiniteField(mod[, symmetric])  Finite field based on GMPY's mpz type.  GMPYEntionFiniteField  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField  GRPYFiniteField  GRPYF	DiagMatrix(vector)	Turn a vector into a diagonal matrix.
where i = j, arc zero.  DiagonalOf(*args, **kwargs)  DiagonalOf(*m) will create a matrix expression that is equivalent to the diagonal of M, represented as a single column matrix.  Dict(*args)  DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DisjointUnion(*sets)  Expresents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain()  Domain()  Domain()  Domain()  Dommy(plame, dummy_index))  EPath(path)  EPath(path)  Ei(z)  Ellipse((enter, hradius, vradius, eccentricity))  Eq alias of Equality  Equality(lhs, rhs, **options)  ExpressionDomain()  A n equal relation between two objects.  Equivalent(*args)  Equivalent(*args)  ExpressionDomain()  FF gmpy  FlalingFactorial(x, k)  FallingFactorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric])  FiniteSet(*args, **kwargs)  FractionField(domain, on_field[, symbols, order])  Function(*args)  Base class for faloaing-point interfield  Filoat(mun_dbs, precision)  Represents a finite set of Sympy expressions.  Finite field of prime order GF(p)  FiniteSet(*args, **kwargs)  FractionField(modin, symmetric))  Finite field of prime order GF(p)  FiniteSet(*args, **kwargs)  Fase class for applied mathematical functions.  Function(*args)  Base class for faloaing-point number of abitary precision.  Class representing multivariate rational function fields.  Function(*args, **hwargs)  Base class for function classes.  Function(*args, **hwargs)  Base class for function classes.  Function(*args, **hwargs)  FiniteField of prime order GF(p)  FiniteField of prime order GF	<pre>DiagonalMatrix(*args, **kwargs)</pre>	DiagonalMatrix(M) will create a matrix expression that
DiagonalOf(*my will create a matrix expression that is equivalent to the diagonal of M, represented as a single column matrix.  Dict(*args)  DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  DiracDelta(arg[, k])  Superclass for all domains in the polys domains system. DotProduct(arg1, arg2)  DotProduct(arg1, arg2)  Dummy([name, dummy_index])  EPath(path)  EPath(path)  Eli(2)  EPath(path)  Equality(lhs, rhs, **options)  Equivalent(*args)  ExpressionDomain()  A classical exponential integral.  Elilipse([center, hradius, vradius, eccentricity])  Equality(lhs, rhs, **options)  ExpressionDomain()  FF  ExpressionDomain()  FF  ExpressionDomain()  FF  ExpressionDomain()  For alias of FiniteField  FallingFactorial(x, k)  Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric])  FiniteSet(*args, **kwargs)  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(domain_or_field[, symbols, order)]  Function(*args)  Function(*args)  Function(*args)  Function(arss, **kwargs)  FiniteField mathematical functions.  Expresents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  GMPYEniteField(mod[, symmetric])  GMPYEniteField(mod[, symmetric])  GMPYEniteField(mod[, sympotype.  Alias of FiniteField based on GMPY in gegers.  GMPYEniteFi		behaves as though all off-diagonal elements, $M[i, j]$
equivalent to the diagonal of M, represented as a single column matrix.  Dict(*args) Wrapper around the builtin dict object.  DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k]) The DiracDelta function and its derivatives.  DisjointUnion(*sets) Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain() Superclass for all domains in the polys domains system.  DotProduct(arg1, arg2) Dot product of vector matrices  Dummy(Iname, dummy_index]) Dummy symbols are each unique, even if they have the same name:  EPath(path) Manipulate expressions using paths.  Ei(a) The classical exponential integral.  Ellipse([center, hradius, vradius, eccentricity]) An elliptical GeometryEntity.  Eq alias of Equallity  Equality(Ihs, rhs, **options) An equal relation between two objects.  Equivalence relation.  Expr(*args) Base class for ablitary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for BrinteField  FF_gmpy alias of GMPYFiniteField  FF_gmpy alias of GMPYFiniteField  FF_gmpy Bails of GMPYFiniteField  Falling Factorial (related to rising factorial) is a double valued function arising in concrete m		where $i != j$ , are zero.
Dict(*args) DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k]) DiracDelta(arg[, k]) DisjointUnion(*sets) Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain() Superclass for all domains in the polys domains system.  DotProduct(arg1, arg2) Dot product of vector matrices Dummy(Iname, dummy_index]) Dummy symbols are each unique, even if they have the same name: EPath(path) Ellipse([center, hradius, vradius, eccentricity]) Eq Ellipse([center, hradius, vradius, eccentricity]) Eq Equality(lhs, rhs, **options) Equivalent(*args) Equivalent(*args) ExpressionDomain() FF gmpy FF gmpy Ac lass for algebraic expressions. ExpressionDomain() FF_gmpy Ac lass of Farbitrary expressions. FFF gllingFactorial(x, k) FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hyper-geometric functions and series expansions. FiniteField(mod[, symmetric]) FiniteSet(*args, **skwargs) FiniteField(mod[, symmetric]) FiniteField(mod[, symmetric]) FiniteField(mod[, symmetric]) FiniteField(domain_or_field[, symbols, order]) Function(*args) Function(*args) Function(*args) Function(*args) Function(*args) Function(*args) Function(*args) Function(*args) Base class for prepresenting multivariate rational function fields.  Function(*args) Base class for prepresenting multivariate rational function fields.  Function(*args) Base class for prepresenting multivariate rational function fields.  Function(*args) Base class for prepresenting multivariate rational function fields.  Function(*args) FiniteField(mod[, symmetric]) FiniteFie	DiagonalOf(*args, **kwargs)	DiagonalOf(M) will create a matrix expression that is
Dict(*args) DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k]) DisjointUnion(*sets) Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain() Superclass for all domains in the polys domains system.  DotProduct(arg1, arg2) Dot product of vector matrices  Dummy(lname, dummy_index]) Dot product of vector matrices Dummy(lname, dummy_index] Dummy symbols are each unique, even if they have the same name:  Partices (lname, dummy_index] Dummy symbols are each unique, vece if they have the sa		equivalent to the diagonal of $M$ , represented as a single
DifferentialMapping(US, coord2item,)  DiracDelta(arg[, k]) DisjointUnion(*sets) Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain() Superclass for all domains in the polys domains system. DotProduct(arg1, arg2) Dummy([name, dummy_index]) Dummy([name, dummy_index]) Dummy([name, dummy_index]) Dummy([name, dummy_index]) EPath(path) Epath(path) Elijpse([center, hradius, vradius, eccentricity]) Eq alias of Equality Equality(lhs, rhs, **options) Equality(lhs, rhs, **options) Expr(*args) Equivalence relation. Expr(*args) Expr(*args		column matrix.
DiracDelta(arg[, k]) DiracDelta(arg[, k]) DisjointUnion(*sets) Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Domain() DotProduct(arg1, arg2) Dotproduct of vector matrices Dummy([name, dummy_index]) Dummy(mame, dummy_index]) Dummy symbols are each unique, even if they have the same name:  EPath(path) Bi(2) An elliptical GeometryEntity. Eq Equality(lhs, rhs, **options) Expr(*args) An equal relation between two objects. Equivalent(*args) Expr(*args) Base class for algebraic expressions. ExpressionDomain() A class for argebraic expressions. ExpressionDomain() A class for argebraic expressions. FF Fgmpy Alias of FfiniteField FF_gmpy Alias of FyThonFiniteField Alias of GMPYFiniteField FallingFactorial(x, k) Finite field of prime order GF(p) FiniteSet(*args, **kwargs) Float(numl, dps, precision]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) Base class for applied mathematical functions. Function(*args) Base class for public mathematical function fields. Function(*args) Base class for representing multivariate rational function fields. Function(*args) Function(*args) Function(*args) Finite field based on GMPY's mpz type. GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY's mpz type. GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY's mpz type. Action(Field(mod, symmetric)) Finite field based on GMPY's mpz type. Action(Field(mod, symmetric)) Finite field based on GMPY's mpz type. Action(Field(mod, symmetric)) Class representations of inequalities. GF GMPYFiniteField(mod[, symmetric]) GRPYRationalField() Rational field based on GMPY's mpz type. Action(Fields) GroebnerBasis(F, *gens, **args) Class representations of inequalities. GroebnerBasis(F, *gens, **args) Action(Fields) Action(Fields) Action(Fields) Action(Fields) Action(Fields) Action(Fields) Action(Fields) Action(Field) Action(Fields) Action(Fields) Action(Field) Action(Field) Action(Fields) Action(Fields) Action(Fields) Action(Fields) Action(F	Dict(*args)	Wrapper around the builtin dict object.
DisjointUnion(*sets)  Represents the disjoint union (also known as the external disjoint union) of a finite number of sets.  Superclass for all domains in the polys domains system.  DotProduct(arg1, arg2)  Dot product of vector matrices  Dummy([name, dummy_index])  Dummy symbols are each unique, even if they have the same name:  EPath(path)  Ei(z)  The classical exponential integral.  An ellipiteal Geometry Entity.  Equality(lhs, rhs, **options)  Equivalent(*args)  Equivalent(*args)  ExpressionDomain()  An elqual relation between two objects.  Equivalent(*args)  ExpressionDomain()  A class for arbitrary expressions.  ExpressionDomain()  A class for arbitrary expressions.  ExpressionDomain()  FF_gmy  Flailing Factorial (x, k)  Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(mod[, sympercision])  FourierTransform(*args)  Float(num[, dps, recision])  FourierTransform(*args)  FractionField(domain_or_field[, symbols, order])  Function(*args)  Function(*args)  Base class for applied mathematical functions.  Function(*args)  Base class for representing multivariate rational function fields.  Function(*args)  FactionField(mod[, symmetric])  Finite field of prime order GF(p)  FiniteField mathematical functions.  A class for representing multivariate rational function fields.  Function(*args)  Base class for function classes.  Function(*args)  Base class for function classes.  Function(*args)  FactionField(mod[, symmetric])  Finite field based on GMPY integers.  Affine field based on GMPY integers.  MPYIntegerRing()  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField  GMPYFiniteField(mod[, symmetric])  Finite field based on GMPY integers.  Actional field based on GMPY's mpc type.  GroebnerBasis(F, *gens, **args)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  All as of FrinterField  GroebnerBasis(F, *gens, **args)	DifferentialMapping(US, coord2item,)	
Domain() Superclass for all domains in the polys domains system.  DotProduct(arg1, arg2) Dot product of vector matrices  Dummy([name, dummy_index]) Dummy symbols are each unique, even if they have the same name:  EPath(path) Manipulate expressions using paths.  Ei(z) The classical exponential integral. An elliptical GeometryEntity.  Eq alias of Equality Equality(lhs, rhs, **options) An equal relation between two objects.  Equivalent(*args) Equivalent(*args) Expr(*args) Base class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  FF_gmpy Alias of FiniteField FF_python Alias of FythonFiniteField FallingFactorial(x, k) FiniteField(mod[, symmetric]) FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) Fioartioum[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) Base class for applied mathematical function fields.  Function(*args, **kwargs) Base class for interior aliase.  GF Alias of FiniteField GMPYFiniteField GMPYFiniteField(mod[, symmetric]) FiniteField(mod[, symmetric]) FiniteField(mod[, symbols, order]) Function(*args) Base class for interior classes.  Function(*args, **kwargs) Base class for interior classes.  Function(*args, **kwargs) Base class for interior classes.  GF Alias of FiniteField GMPYFiniteField GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY's mpz type. GMPYFiniteField(mod[, symmetric]) GMPYRationalField() Rational field based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpz type. GMPYRationalField() Represents a matrix using a function (classes. Represents a rational field based on GMPY's mpz type. GMPYRationalField() Represents a matrix using a function fienqualities. Represents a rational field based on GMPY's mpz type. GMPYRationalField() Represents a rational field based on GMPY's mpz ty	<pre>DiracDelta(arg[, k])</pre>	The DiracDelta function and its derivatives.
Domain()   Superclass for all domains in the polys domains system.	DisjointUnion(*sets)	Represents the disjoint union (also known as the external
Dot Product (arg 1, arg 2)   Dot product of vector matrices		disjoint union) of a finite number of sets.
Dummy([name, dummy_index])  Dummy symbols are each unique, even if they have the same name:  EPath(path)  Ei(z)  The classical exponential integral.  An elliptical GeometryEntity.  Equality(lhs, rhs, **options)  An equal relation between two objects.  Equivalent(*args)  ExpressionDomain()  Frequivalent(*args)  ExpressionDomain()  Frequivalent(*args)  ExpressionDomain()  A class for adjustrary expressions.  ExpressionDomain()  A class for affirative expressions.  Frequivalence relation.  ExpressionDomain()  A class for affirative expressions.  I alias of FiniteField  Frequivalence relation.  ExpressionDomain()  A class for affirative expressions.  I alias of FiniteField  Frequivalence relation.  ExpressionDomain()  A class for affirative expressions.  I alias of FiniteField  Frequivalence relation.  ExpressionDomain()  A class for affirative expressions.  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FiniteField(mod[, symmetric])  FourierTransform(*args)  FractionField(domain_or_field[, symbols, order])  Function(*args)  Function(*args)  Function(*args)  Function(*args)  Function(*args)  Function(*args)  Function(*args)  Function(*args, **kwargs)  Base class for applied mathematical function fields.  Base class for function classes.  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  Alias of FiniteField  GMPYFiniteField(mod[, symmetric])  Finite field based on GMPY integers.  GMPYIntegerRing()  GMPYRationalField()  Rational field based on GMPY's mpq type.  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a founced Groebner basis.  alias of StrictGreaterThan		Superclass for all domains in the polys domains system.
EPath(path)	DotProduct(arg1, arg2)	
EPath(path) Ei(z) The classical exponential integral. Ellipse([center, hradius, vradius, eccentricity]) Eq alias of Equality Equality(lhs, rhs, **options) An equal relation between two objects. Equivalent(*args) Equivalent(*args) Expr(*args) Expr(*args) Expr(*args) ExpressionDomain() A class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  EFF_gmpy A class for arbitrary expressions.  FF_gmpy A class of GMPYFiniteField FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) Finite field of prime order GF(p) FiniteSet(*args, **kwargs) Float(num[, dps, precision]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) Function(*args) Function(*args) Function(*args, **kwargs) Base class for applied mathematical functions fields.  Function(*args, **kwargs) Base class for applied mathematical functions.  Expresents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF GMPYFiniteField(mod[, symmetric])  GMPYTiniteGerRing()  GMPYFiniteField(mod[, symmetric])  Finite field based on GMPY's mpz type.  GMPYIntegerRing()  GMPYRiniteField()  Rational field based on GMPY's mpg type.  GMPYRationalField()  Rational field based on GMPY's mpg type.  GMPYRationalField()  Rational field based on GMPY's mpg type.  GRPYRationalField()  Rational field based on GMPY's mpg type.  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Groebner Basis(F, *gens, **args)  Represents a reduced Groebner basis.	<pre>Dummy([name, dummy_index])</pre>	Dummy symbols are each unique, even if they have the
Ei(z) The classical exponential integral.  Ellipse([center, hradius, vradius, eccentricity]) An elliptical GeometryEntity.  Eq alias of Equality  Equality(lhs, rhs, **options) An equal relation between two objects.  Equivalent(*args) Equivalence relation.  Expr(*args) Base class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  FF alias of FiniteField  FF_gmpy alias of GMPYFiniteField  FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) FiniteField of prime order GF(p)  FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions.  Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields.  Function(*args) Base class for applied mathematical functions.  Function(*args) Base class for pupied mathematical functions.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpz type.  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GreaterThan(spense) Represents a reduced Groebner basis.  Gt		
Ellipse([center, hradius, vradius, eccentricity]) Eq alias of Equality Equality(lhs, rhs, **options) An equal relation between two objects. Equivalent(*args) Equivalence relation. Expr(*args) Base class for algebraic expressions. ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for reprise the defended of printer order Greet mathematics, hypergeometric function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) FiniteSet(*args, **kwargs) FiniteField(mod[, sympolis, order]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) FractionField(domain_or_field[, symbols, order]) Function(*args) Base class for representing unevaluated Fourier transforms.  Function(*args) Base class for applied mathematical function fields.  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF GMPYFiniteField(mod[, symmetric]) GMPYInitegerRing() Integer ring based on GMPY integers.  GMPYIntegerRing() Rational field based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpz type.  GREATERTHAN(lhs, rhs, **options) Class representations of inequalities.  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GreaterThan(lhs, rhs, **args) Represents a reduced Groebner basis.  GreeteerThan(leger ring) Rational relation between two objects.  Expressents a reduced Groebner basis.  GreaterThan(legerTing) Rational relation between two objects.  Finite field deserved for expression for expressions.  Finite field for prime order GF(p) Rational relation for expressions.  Finite field for p		
Eq alias of Equality  Equality(lhs, rhs, **options) An equal relation between two objects.  Equivalent(*args) Equivalence relation.  Expr(*args) Base class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  ExpressionDomain() A class for arbitrary expressions.  FF alias of FiniteField  FF-gmpy alias of GMPYFiniteField  FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hyper-geometric functions and series expansions.  FiniteField(mod[, symmetric]) Finite field of prime order GF(p)  FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions.  Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields.  Function(*args) Base class for applied mathematical functions.  Function(*args) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  Integer ring based on GMPY's mpz type.  GMPYIntegerRing() Rational field based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpz type.  GRETTHAN (lhs, rhs, **options) Class represents a reduced Groebner basis.  GroebnerBasis(F, *gens, **args) alias of StrictGreaterThan		
Equality(lhs, rhs, **options) Equivalent(*args) Equivalent(*args) ExpressionDomain() FXPTEXSION A class for algebraic expressions. EXPRESSIONDOMAIN() FR alias of FiniteField FF_gmpy FI_opthon FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions. FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) Float(num[, dps, precision]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) Function(*args) FunctionMatrix(rows, cols, lamda)  GF GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() GMPYRationalField() GMPYRationalField() GRESSION Class representations of inequalities. GreeaterThan(lhs, rhs, **options) GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. GroebnerBasis(F, *gens, **args) A class of Symptons of GMPY integers. Class representations of inequalities. GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. A class of Symither of Arbitrary precision. Class representations of inequalities. GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. A class of StrictGreaterThan	Ellipse([center, hradius, vradius, eccentricity])	• •
Equivalent(*args) Expr(*args) Base class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  FF gmpy alias of FiniteField FF_gmpy alias of GMPYFiniteField FF_python FallingFactorial(x,k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) FiniteSet(*args, **skwargs) Float(num[, dps, precision]) FourierTransform(*args) FourierTransform(*args) Class represents a finite set of Sympy expressions. FractionField(domain_or_field[, symbols, order]) Function(*args) FunctionClass(*args, **skwargs) Base class for representing multivariate rational function fields.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF GMPYFiniteField(mod[, symmetric]) GMPYIntegerRing() Integer ring based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpq type. alias of GreaterThan GreaterThan(lhs, rhs, **options) GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. Gt	-	
ExpressionDomain() A class for algebraic expressions.  ExpressionDomain() A class for arbitrary expressions.  FF alias of FiniteField FF_gmpy FF_python Alias of GMPYFiniteField FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) FiniteField(mod[, sympercision]) FourierTransform(*args) Float(num[, dps, precision]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) Function(*args) Function(*args) Function(*args) FunctionClass(*args, **kwargs) Base class for applied mathematical functions.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY's mpz type.  GMPYIntegerRing() Integer ring based on GMPY's mpt type.  GMPYRationalField() Rational field based on GMPY's mpt type.  GREVENTHAM (Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  GIass of StrictGreaterThan  Class representations of inequalities.		
ExpressionDomain()  A class for arbitrary expressions.  alias of FiniteField  FF_gmpy  alias of GMPYFiniteField  FF_python  FallingFactorial(x, k)  Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric])  FiniteSet(*args, **kwargs)  Finite field of prime order GF(p)  FiniteSet(*args, **kwargs)  Float(num[, dps, precision])  FourierTransform(*args)  FractionField(domain_or_field[, symbols, order])  FractionField(domain_or_field[, symbols, order])  Function(*args)  Function(*args)  FunctionClass(*args, **kwargs)  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  alias of FiniteField  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  Rational field based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpq type.  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  alias of StrictGreaterThan		•
FF alias of FiniteField  FF_gmpy alias of GMPYFiniteField  FF_python alias of PythonFiniteField  FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hyper-geometric functions and series expansions.  FiniteField(mod[, symmetric]) Finite field of prime order GF(p)  FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions.  Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields.  Function(*args) Base class for applied mathematical functions.  FunctionClass(*args, **kwargs) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYTRationalField() Rational field based on GMPY's mpz type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
FF_gmpy	-	* *
FF_python		
FallingFactorial(x, k) Falling factorial (related to rising factorial) is a double valued function arising in concrete mathematics, hypergeometric functions and series expansions.  FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) FiniteSet(*args, **kwargs) Float(num[, dps, precision]) FourierTransform(*args) FractionField(domain_or_field[, symbols, order]) FractionField(domain_or_field[, symbols, order]) Function(*args) Function(*args) FunctionClass(*args, **kwargs) FunctionMatrix(rows, cols, lamda) FunctionMatrix(rows, cols, lamda)  GMPYFiniteField(mod[, symmetric])  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  GMPYRationalField()  GRED Rational field based on GMPY's mpz type.  GMPYRationalField()  GRED Rational field based on GMPY's mpq type.  GeaterThan(lhs, rhs, **options)  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  Gliss representations of inequalities.  GreobenerBasis(F, *gens, **args)  Represents a reduced Groebner basis.		
valued function arising in concrete mathematics, hyper- geometric functions and series expansions.  FiniteField(mod[, symmetric]) Finite field of prime order GF(p)  FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions.  Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields.  Function(*args) Base class for applied mathematical functions.  FunctionClass(*args, **kwargs) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
geometric functions and series expansions.  FiniteField(mod[, symmetric]) Finite field of prime order GF(p)  FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions.  Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision.  FourierTransform(*args) Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields.  Function(*args) Base class for applied mathematical functions.  FunctionClass(*args, **kwargs) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan	FallingFactorial(x, k)	
FiniteField(mod[, symmetric]) FiniteSet(*args, **kwargs) Represents a finite set of Sympy expressions. Float(num[, dps, precision]) Represent a floating-point number of arbitrary precision. FourierTransform(*args) Class representing unevaluated Fourier transforms. FractionField(domain_or_field[, symbols, order]) A class for representing multivariate rational function fields. Function(*args) Base class for applied mathematical functions. FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField GMPYFiniteField(mod[, symmetric]) GMPYIntegerRing() Integer ring based on GMPY integers. GMPYIntegerRing() Rational field based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpq type. Ge alias of GreaterThan GreaterThan(lhs, rhs, **options) Class representations of inequalities. GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. Gt alias of StrictGreaterThan		
FiniteSet(*args, **kwargs)  Float(num[, dps, precision])  Represent a floating-point number of arbitrary precision.  FourierTransform(*args)  Class representing unevaluated Fourier transforms.  FractionField(domain_or_field[, symbols, order])  Function(*args)  Function(*args)  FunctionClass(*args, **kwargs)  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  GMPYFiniteField(mod[, symmetric])  GMPYFiniteGerRing()  GMPYRationalField()  GMPYRationalField()  GRepterBasis(F, *gens, **args)  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Represents a finite set of Sympy expressions.  Represent a floating-point number of arbitrary precision.  Representing unevaluated Fourier transforms.  A class for representing unevaluated Fourier transforms.  Function(*args)  A class for applied mathematical function fields.  Function(*args)  Finite field based on GMPY integers.  GMPYINTERION (*args)  Finite field based on		•
Float(num[, dps, precision])  Represent a floating-point number of arbitrary precision.  FourierTransform(*args)  Class representing unevaluated Fourier transforms.  A class for representing multivariate rational function fields.  Function(*args)  Base class for applied mathematical functions.  FunctionClass(*args, **kwargs)  Base class for function classes.  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  Rational field based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpq type.  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan		
FourierTransform(*args)  FractionField(domain_or_field[, symbols, order])  FractionField(domain_or_field[, symbols, order])  Function(*args)  FunctionClass(*args, **kwargs)  FunctionMatrix(rows, cols, lamda)  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  GMPYRationalField()  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Gt  Class representing unevaluated Fourier transforms.  A class for representing multivariate rational function fields.  A class for representing multivariate rational function fields.  Base class for applied mathematical functions.  Base class for function classes.  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  Gives outputs according to the coordinates of each matrix entries.  Integer ring based on GMPY integers.  Integer ring based on GMPY's mpz type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan		
FractionField(domain_or_field[, symbols, order])  A class for representing multivariate rational function fields.  Function(*args)  Base class for applied mathematical functions.  Base class for function classes.  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  alias of FiniteField  GMPYFiniteField(mod[, symmetric])  Finite field based on GMPY integers.  GMPYIntegerRing()  Integer ring based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpq type.  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan		
Function(*args) Base class for applied mathematical functions.  FunctionClass(*args, **kwargs) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
Function(*args) Base class for applied mathematical functions.  FunctionClass(*args, **kwargs) Base class for function classes.  FunctionMatrix(rows, cols, lamda) Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan	FractionField(domain_or_field[, symbols, order])	
FunctionClass(*args, **kwargs)  FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  alias of FiniteField  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  Integer ring based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpq type.  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan	T (*)	
FunctionMatrix(rows, cols, lamda)  Represents a matrix using a function (Lambda) which gives outputs according to the coordinates of each matrix entries.  GF  alias of FiniteField  GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  Integer ring based on GMPY's mpz type.  GMPYRationalField()  Rational field based on GMPY's mpq type.  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  Class representations of inequalities.  GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan		**
gives outputs according to the coordinates of each matrix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
trix entries.  GF alias of FiniteField  GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers.  GMPYIntegerRing() Integer ring based on GMPY's mpz type.  GMPYRationalField() Rational field based on GMPY's mpq type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan	FunctionMatrix(rows, cols, lamda)	
GF alias of FiniteField GMPYFiniteField(mod[, symmetric]) Finite field based on GMPY integers. GMPYIntegerRing() Integer ring based on GMPY's mpz type. GMPYRationalField() Rational field based on GMPY's mpq type. Ge alias of GreaterThan GreaterThan(lhs, rhs, **options) Class representations of inequalities. GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis. Gt alias of StrictGreaterThan		
GMPYFiniteField(mod[, symmetric])  GMPYIntegerRing()  GMPYRationalField()  Ge  alias of GreaterThan  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Gt  Finite field based on GMPY integers.  Integer ring based on GMPY's mpz type.  Rational field based on GMPY's mpq type.  Class representations of inequalities.  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan	CP.	
GMPYIntegerRing()  GMPYRationalField()  Rational field based on GMPY's mpz type.  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Gt Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
GMPYRationalField()  Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Gt Against Agriculture Rational field based on GMPY's mpq type.  Alias of GreaterThan  Class representations of inequalities.  Represents a reduced Groebner basis.  Alias of StrictGreaterThan		
Ge alias of GreaterThan  GreaterThan(lhs, rhs, **options) Class representations of inequalities.  GroebnerBasis(F, *gens, **args) Represents a reduced Groebner basis.  Gt alias of StrictGreaterThan		
GreaterThan(lhs, rhs, **options)  GroebnerBasis(F, *gens, **args)  Gt  Class representations of inequalities.  Represents a reduced Groebner basis.  alias of StrictGreaterThan		• •
GroebnerBasis(F, *gens, **args)  Represents a reduced Groebner basis.  Gt  alias of StrictGreaterThan		
Gt alias of StrictGreaterThan	=	•
continues on next page	nauamarurower (base, exp)	Elementwise power of matrix expressions

Table 68 – continued from previous page

HadamardProduct("args], evaluate, check])  HankelTransform("args)  Haviside(arg[, H0])  Haviside(arg[, H0])  Haviside set pfunction.  If-then-else clause.  Heaviside set pfunction.  If-then-else clause.  Heaviside set pfunction.  If-then-else clause.  Represents an integer index as an Integer or integer expression.  ImmutableDenseMplimArray(iterable], shape])  ImmutableDenseMplimArray(iterable], shape])  ImmutableDenseMplimArray(iterable], shape])  ImmutableSparseMatrix("args, "*kwargs)  ImmutableSparseMplimArray(iterable], shape])  Implies("args)  Indexed(base, "args, "*kw args)  Indexed(base, "args, "*kw args)  Indexed(base, "args, "*kw args)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Integer(i)  Intersection("args, **asumptions)  Intersection("args, *asumptions)  Intersection("args, *asumptions)  Intersection("args, *asumptions)  Intersection("args, *asumptions)  Intersection("args, *asumptions)  Intersection("args, *asumptions)  Intersec	Table 68 – continue	d from previous page
Heaviside(arg , H0 )	<pre>HadamardProduct(*args[, evaluate, check])</pre>	Elementwise product of matrix expressions
If-then-else clause:   Identity(n)   The Matrix Identity I multiplicative identity	HankelTransform(*args)	Class representing unevaluated Hankel transforms.
Identity(n)	Heaviside(arg[, H0])	Heaviside step function.
Idx(labell, range])  Represents an integer index as an Integer or integer expression.  ImageSet(flambda, *sets) ImmutableDenseNatrix(*args, **skwargs) ImmutableDenseNDimArray(iterable[, shape])  ImmutableSparseMatrix(*args, **skwargs) ImmutableSparseMatrix(*args, **skwargs) ImmutableSparseMatrix(flierable, shape])  Implies(*args) Indexed(base, *args, **skw_args) Indexed(base, *args, **skw_args) Indexed(base, *args, **skw_args) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Intersection(*args, **skwargs) Intersection(*args, **skwargs) Interval(start, end[, left_open, right_open]) Inverse(mail, expl) InverseCosineTransform(*args) InverseGosineTransform(*args) InverseBallaceTransform(*args) Class representing unevaluated inverse Cosine transforms.  InverseBallaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Sine transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Sine transforms.  The discrete, or Kronecker, delta function.  The KroneckerPoduct(*args[, check]) LambertW(x[, k]) The Lambert W function \$W(x)\$ is defined as the inverse function of \$w exp(w)\$ [1].  Class representing unevaluated limit.  Lamba(x, expr) represents a lambda function similar to Python's lambda x: expr.'  LambertW(x[, k]) Class representing on several conducted inverse sine transforms.  Alias of LeesThan (LeesThan unevaluated limit.  Class representing in the problem of \$w ex	ITE(*args)	If-then-else clause.
Idx(labell, range]) Represents an integer index as an Integer or integer expression.  ImageSet(flambda, *sets) ImmutableDenseNatrix(*args, **ikwargs) ImmutableDenseNDimArray(iterable[, shape])  ImmutableSparseNatrix(*args, **kwargs) ImmutableSparseNDimArray([iterable, shape])  ImmutableSparseNDimArray([iterable, shape])  Implies(*args) Indexed(base, *args, **kw_args) Indexed(base, *args, **kw_args) Indexed(base, *args, **ikw_args) Indexed(base, *args, **kw_args) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Integer(i) Intersection(*args, **sumptions) Intersection(*args, **sumptions) Intersection(*args, **sumptions) Intersection(*args, *args, **sumptions) Intersection(*args, *args, *args	Identity(n)	The Matrix Identity I - multiplicative identity
Image of a set under a mathematical function.	<pre>Idx(label[, range])</pre>	Represents an integer index as an Integer or integer
ImmutableDenseMatrix(*args, **kwargs)   ImmutableDenseMatrix   alias of ImmutableDenseMatrix   alias of ImmutableDenseMatrix   ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseMatrix(*args, **kwargs)   Indexed(base, *args, **kw_args)   Logical implication.   Represents a mathematical object with indices.   IndexedBase(label[, shape, offset, strides])   Represents integer numbers of any size.   IntegerRing()   IntegerRing()   The domain ZZ representing the integers mathbb[Z].   Integral(function, *symbols, **assumptions)   Intersection(*args, **kwargs)   Represents a real interval as a Set.   Represents an intersection of sets as a Set.   Inverse(mat[, exp])   The multiplicative inverse of a matrix expression   InverseCosineTransform(*args)   Class representing unevaluated inverse cosine transforms.   InverseHankelTransform(*args)   Class representing unevaluated inverse Fourier transforms.   InverseLaplaceTransform(*args)   Class representing unevaluated inverse Hankel transforms.   InverseSineTransform(*args)   Class representing unevaluated inverse Mellin transforms.   InverseSineTransform(*args)   Class representing unevaluated inverse sine transforms.   Class representing unevaluated inverse sine transforms.   Class repre		
ImmutableDenseNDimArray(iterable[, shape])   ImmutableSparseMatrix   alias of ImmutableDenseMatrix   ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseNDimArray([iterable, shape])   Implies(*args)   Logical implication.   Indexed(base, *args, **kwargs)   Represents a mathematical object with indices.   Represents integer numbers of any size.   IndexedBase(label[, shape, offset, strides])   Represents integer numbers of any size.   IntegerAl(function, *symbols, **assumptions)   Represents integer numbers of any size.   Integral(function, *symbols, **assumptions)   Represents unevaluated integral.   Represents a mathematical object with indices.   Represents integer numbers of any size.   The domain ZZ representing the integers mathbb[Z].   Represents unevaluated integral.   Represents a real interval as a Set.   Represents a mathematical object with indices.   Represents a mathematical object with indices.   Represents in ambers of any size.   Represents a mathematical object with indices.   Represents a mathematical object with indices.   Represents a mathematical object with indices.   Represents in mindexed object represents a mathematical object with indices.   Represents in mindexed object represents a mathematical object with indices.   Represents a method indexed object represents a lambable as a Set.   Represent the Levi-Civita symbol.   The offset logarithmic integral.   Represents a	<pre>ImageSet(flambda, *sets)</pre>	Image of a set under a mathematical function.
ImmutableMatrix   ImmutableSparseMatrix(*args, **kwargs)   Create an immutable version of a sparse matrix.	<pre>ImmutableDenseMatrix(*args, **kwargs)</pre>	Create an immutable version of a matrix.
ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseMDimArray([iterable, shape])	<pre>ImmutableDenseNDimArray(iterable[, shape])</pre>	
ImmutableSparseMatrix(*args, **kwargs)   ImmutableSparseMDimArray([iterable, shape])		
ImmutableSparseNDimArray([iterable, shape])  Implies(*args)	ImmutableMatrix	alias of ImmutableDenseMatrix
Implies(*args) Indexed(base, *args, **kw_args) Indexed(base(label[, shape, offset, strides]) Integer(i) Integer(i) Integer(i) Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Inverse(maf[, exp]) Inverse(maf[, exp]) Inverse(maf[, exp]) InverseFourierTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseMellinTransform(*args) InverseMellinT	<pre>ImmutableSparseMatrix(*args, **kwargs)</pre>	Create an immutable version of a sparse matrix.
Indexed(base, *args, **ekw_args) Represents a mathematical object with indices. IndexedBase(label[, shape, offset, strides]) Represents the base or stem of an indexed object Integer(i) Represents integer numbers of any size. Integeral(function, *symbols, **assumptions) Represents unevaluated integral. Intersection(*args, **ekwargs) Represents unevaluated integral. Interval(start, end[, left_open, right_open]) Represents an intersection of sets as a Set. Interval(start, end[, left_open, right_open]) Inverse(mat[, exp]) Inverse(mat[, exp]) InverseFourierTransform(*args) Class representing unevaluated inverse cosine transforms. InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse sine transforms. KroneckerPelta(i, j[, delta_range]) KroneckerPoduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'. Lambert W function \$W(z)\$ is defined as the inverse function of \$\mathb{SW}(z)\$ is defined as the inverse function of \$SW	<pre>ImmutableSparseNDimArray([iterable, shape])</pre>	
Indexed(base, *args, **ekw_args) Represents a mathematical object with indices. IndexedBase(label[, shape, offset, strides]) Represents the base or stem of an indexed object Integer(i) Represents integer numbers of any size. Integeral(function, *symbols, **assumptions) Represents unevaluated integral. Intersection(*args, **ekwargs) Represents unevaluated integral. Interval(start, end[, left_open, right_open]) Represents an intersection of sets as a Set. Interval(start, end[, left_open, right_open]) Inverse(mat[, exp]) Inverse(mat[, exp]) InverseFourierTransform(*args) Class representing unevaluated inverse cosine transforms. InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse sine transforms. KroneckerPelta(i, j[, delta_range]) KroneckerPoduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'. Lambert W function \$W(z)\$ is defined as the inverse function of \$\mathb{SW}(z)\$ is defined as the inverse function of \$SW		
IndexedBase(label[, shape, offset, strides])  Integer(i)  Intersection(*args, **stawargs)  Interval(start, end[, left_open, right_open])  Inverse(mat[, exp])  Inverse(mat[, exp])  Inverse(mat[, exp])  Inverse(mat[, exp])  Inverse(i)		Logical implication.
Integer(i) Represents integer numbers of any size. IntegerRing() IntegerAl(function, *symbols, **assumptions) Intersection(*args, **tewargs) Represents unevaluated integers mathbb(Z). Interval(start, end[, left_open, right_open]) Represents an intersection of sets as a Set. Interval(start, end[, left_open, right_open]) Represents a real interval as a Set. Inverse(mat[, exp]) InverseCosineTransform(*args) Class representing unevaluated inverse cosine transforms. InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms. InverseHankelTransform(*args) Class representing unevaluated inverse Laplace transforms. InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms. InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms. KroneckerPelta(i, ji, delta_range]) KroneckerPoduct(*args[, check]) The discrete, or Kronecker, delta function. KroneckerProduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1].  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  LessThan(lhs, rhs, **options) LessThan LessThan(ls, rhs, **options) Class represents an unevaluated limit.  The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  An infinite line in space.  Line (*args, **kwargs) Line 2D(p1[, pt, slope]) An infinite line in space.	<pre>Indexed(base, *args, **kw_args)</pre>	
IntegerRing() Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Intersection(*args, **kwargs) Intersection(*args, **kwargs) Interval(start, end[, left_open, right_open]) Inverse(mat[, exp]) Inverse(mat[, exp]) InverseCosineTransform(*args) InverseFourierTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHalinTransform(*args) InverseHalinTransform(*args) InverseHellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseSineTransform(*args) InverseSin	<pre>IndexedBase(label[, shape, offset, strides])</pre>	•
Integral(function, *symbols, **assumptions) Intersection(*args, **kwargs) Interval(start, end[, left_open, right_open]) Inverse(mat[, exp[)) Inverse(mat[, exp]) InverseCosineTransform(*args) InverseCosineTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseLaplaceTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseLaplaceTransform(*args) InverseLaplaceTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseSineTransform(*args) InverseSineTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*a	Integer(i)	Represents integer numbers of any size.
Intersection(*args, **kwargs) Represents an intersection of sets as a Set. Interval(start, end[, left_open, right_open]) Represents a real interval as a Set. Inverse(mat[, exp]) The multiplicative inverse of a matrix expression Class representing unevaluated inverse cosine transforms.  InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) Lambda(signature, expr) LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1].  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Lee alias of LessThan LessThan(lhs, rhs, **options) Class representing unevaluated Laplace transforms.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p[1, pt, slope]) An infinite 3D line in space.		
Interval(start, end[, left_open, right_open]) Inverse(mat[, exp]) Inverse(cosineTransform(*args) InverseCosineTransform(*args) InverseFourierTransform(*args) InverseFourierTransform(*args) InverseHankelTransform(*args) InverseHankelTransform(*args) InverseLaplaceTransform(*args) InverseLaplaceTransform(*args) InverseLaplaceTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseMellinTransform(*args) InverseSineTransform(*args) InverseSineTrans	<pre>Integral(function, *symbols, **assumptions)</pre>	Represents unevaluated integral.
Inverse(mat[, exp]) InverseCosineTransform(*args) Class representing unevaluated inverse cosine transforms.  InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  Lambda(xignature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1].  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le LesSThan(lhs, rhs, **options) Class representations of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) Line2D(p[1], pt, slope]) An infinite line in space.	<pre>Intersection(*args, **kwargs)</pre>	Represents an intersection of sets as a Set.
InverseCosineTransform(*args)  InverseFourierTransform(*args)  Class representing unevaluated inverse Fourier transforms.  Class representing unevaluated inverse Fourier transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Hankel transforms.  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, jf, delta_range)  KroneckerProduct(*args)  Class representing unevaluated inverse sine transforms.  KroneckerProduct(*args)  The discrete, or Kronecker, delta function.  KroneckerProduct(*argsf, checkf)  The Kronecker product of two or more arguments.  Lambda(signature, expr)  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$\mathbb{W}(z)\$ is defined as the inverse function of \$	<pre>Interval(start, end[, left_open, right_open])</pre>	Represents a real interval as a Set.
InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  KroneckerPelta(i, j[, delta_range]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) The Kronecker product of two or more arguments.  Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options) Class representations of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Limit(e, z, z0[, dir]) Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite line in space.	<pre>Inverse(mat[, exp])</pre>	
InverseFourierTransform(*args) Class representing unevaluated inverse Fourier transforms.  InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) The Kronecker product of two or more arguments.  Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(\mathbf{s}')\$ is defined as the inverse function of \$\$\mathbf{s}\$ \text{ expr}(	<pre>InverseCosineTransform(*args)</pre>	Class representing unevaluated inverse cosine trans-
InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse Mellin transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) The Kronecker product of two or more arguments.  Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$\\$ exp(w)\\$ [1].  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le alias of LessThan LessThan(lhs, rhs, **options) Class representations of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite line in space.		forms.
InverseHankelTransform(*args) Class representing unevaluated inverse Hankel transforms.  InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) The Kronecker product of two or more arguments.  Lambda(signature, expr) Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options) Class representions of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite 3D line in space.	<pre>InverseFourierTransform(*args)</pre>	
InverseLaplaceTransform(*args) Class representing unevaluated inverse Laplace transforms.  InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) The Kronecker product of two or more arguments.  Lambda(signature, expr) LambertW(x[, k]) The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options) Class represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite line in space.		
InverseLaplaceTransform(*args)  Class representing unevaluated inverse Laplace transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse Mellin transforms.  Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range])  KroneckerProduct(*args[, check])  Lambda(signature, expr)  Lambda(signature, expr)  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Class representing unevaluated Laplace transforms.  Le  alias of LessThan  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.	InverseHankelTransform(*args)	
forms.  InverseMellinTransform(*args)  Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args)  Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range])  KroneckerProduct(*args[, check])  Lambda(signature, expr)  Lambda(signature, expr)  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Class representing unevaluated Laplace transforms.  Le  alias of LessThan  LessThan(lhs, rhs, **options)  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		
InverseMellinTransform(*args) Class representing unevaluated inverse Mellin transforms.  InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) KroneckerProduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) LambertW(x[, k]) The Kronecker product of two or more arguments.  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW function \$W(z)\$ is defined as the inverse function of \$\sqrt{w} \exp(w)\\$ [1]  LaplaceTransform(*args) Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options) Class representations of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) Line2D(p1[, pt, slope]) Line3D(p1[, pt, direction_ratio]) An infinite line in space.	<pre>InverseLaplaceTransform(*args)</pre>	
InverseSineTransform(*args) Class representing unevaluated inverse sine transforms.  KroneckerDelta(i, j[, delta_range]) The discrete, or Kronecker, delta function.  KroneckerProduct(*args[, check]) Lambda(signature, expr) Lambda(signature, expr) LambertW(x[, k]) LambertW(x[, k]) LambertW(x[, k]) LaplaceTransform(*args) Le alias of LessThan LessThan(lhs, rhs, **options) LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Line(*args, **kwargs) Line2D(p1[, pt, slope]) Line3D(p1[, pt, direction_ratio]) An infinite line in space.  Line in space.  Class representation so finequalities.  Represents an unevaluated limit.  An infinite line in space.  An infinite line in space.		
InverseSineTransform(*args)  KroneckerDelta(i, j[, delta_range])  KroneckerProduct(*args[, check])  Lambda(signature, expr)  Lambda(signature, expr)  LambertW(x[, k])  LambertW(x[, k])  LaplaceTransform(*args)  LessThan(lhs, rhs, **options)  LeviCivita(*args)  Liit(e, z, z0[, dir])  Line(*args, **kwargs)  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  The discrete, or Kronecker, delta function.  The Kronecker product of two or more arguments.  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  Class represents a lambda function similar to Python's 'lambda x: expr'.  Class represents a lambda function similar to Python's 'lambda x: expr'.  Class represents unevaluated Laplace transforms.  Alias of LessThan  Class representations of inequalities.  Represent the Levi-Civita symbol.  The offset logarithmic integral.  An infinite line in space.  Line2D(p1[, pt, slope])  An infinite line in space.  An infinite 3D line in space.	<pre>InverseMellinTransform(*args)</pre>	
KroneckerDelta(i, j[, delta_range])  KroneckerProduct(*args[, check])  Lambda(signature, expr)  Lambda(signature, expr)  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Le alias of LessThan  LessThan(lhs, rhs, **options)  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  Line2D(p1[, pt, slope])  An infinite line in space.  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		
KroneckerProduct(*args[, check])The Kronecker product of two or more arguments.Lambda(signature, expr)Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.LambertW(x[, k])The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]LaplaceTransform(*args)Class representing unevaluated Laplace transforms.Lealias of LessThanLessThan(lhs, rhs, **options)Class representations of inequalities.LeviCivita(*args)Represent the Levi-Civita symbol.Li(z)The offset logarithmic integral.Limit(e, z, z0[, dir])Represents an unevaluated limit.Line(*args, **kwargs)An infinite line in space.Line2D(p1[, pt, slope])An infinite line in space 2D.Line3D(p1[, pt, direction_ratio])An infinite 3D line in space.		
Lambda(signature, expr)  Lambda(x, expr) represents a lambda function similar to Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Le alias of LessThan  LessThan(lhs, rhs, **options)  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		
Python's 'lambda x: expr'.  LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options)  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		•
LambertW(x[, k])  The Lambert W function \$W(z)\$ is defined as the inverse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options)  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Lime(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.	Lambda(signature, expr)	
verse function of \$w exp(w)\$ [1]  LaplaceTransform(*args)  Class representing unevaluated Laplace transforms.  Le alias of LessThan  LessThan(lhs, rhs, **options)  Class representations of inequalities.  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		•
LaplaceTransform(*args)  Le alias of LessThan  LessThan(lhs, rhs, **options)  LeviCivita(*args)  Li(z)  Class representations of inequalities.  Limit(e, z, z0[, dir])  Line(*args, **kwargs)  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  Class representing unevaluated Laplace transforms.  Alias of LessThan  Class representing unevaluates.  Represents an inequalities.  Represent the Levi-Civita symbol.  The offset logarithmic integral.  Represents an unevaluated limit.  An infinite line in space.  An infinite line in space.  An infinite 3D line in space.	LambertW(x[,k])	
Le alias of LessThan  LessThan(lhs, rhs, **options) Class representations of inequalities.  LeviCivita(*args) Represent the Levi-Civita symbol.  Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite line in space 2D.  Line3D(p1[, pt, direction_ratio]) An infinite 3D line in space.		• ' '
LessThan(lhs, rhs, **options)  LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  Limit(e, z, z0[, dir])  Line(*args, **kwargs)  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  Class representations of inequalities.  Represent the Levi-Civita symbol.  Represents an unevaluated limit.  An infinite line in space.  An infinite line in space 2D.  An infinite 3D line in space.	- ` ` ` ` ` `	• •
LeviCivita(*args)  Represent the Levi-Civita symbol.  Li(z)  The offset logarithmic integral.  Limit(e, z, z0[, dir])  Represents an unevaluated limit.  Line(*args, **kwargs)  An infinite line in space.  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite 3D line in space.		
Li(z) The offset logarithmic integral.  Limit(e, z, z0[, dir]) Represents an unevaluated limit.  Line(*args, **kwargs) An infinite line in space.  Line2D(p1[, pt, slope]) An infinite line in space 2D.  Line3D(p1[, pt, direction_ratio]) An infinite 3D line in space.		
Limit(e, z, z0[, dir])Represents an unevaluated limit.Line(*args, **kwargs)An infinite line in space.Line2D(p1[, pt, slope])An infinite line in space 2D.Line3D(p1[, pt, direction_ratio])An infinite 3D line in space.	_	
Line(*args, **kwargs)  Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite line in space 2D.  An infinite 3D line in space.		
Line2D(p1[, pt, slope])  Line3D(p1[, pt, direction_ratio])  An infinite line in space 2D.  An infinite 3D line in space.		<u>*</u>
Line3D(p1[, pt, direction_ratio]) An infinite 3D line in space.		<u>*</u>
		<u>*</u>
Lt alias of StrictLessThan		-
	Lt	alias of StrictLessThan
MatAdd(*args[, evaluate, check, _sympify])  A Sum of Matrix Expressions	MatAdd(*args[, evaluate, check, _sympify])	A Sum of Matrix Expressions

Table 68 – continued from previous page

14510 00 0011411	ided from previous page
<pre>MatMul(*args[, evaluate, check, _sympify])</pre>	A product of matrix expressions
MatPow(base, exp[, evaluate])	
Matrix	alias of MutableDenseMatrix
MatrixBase()	Base class for matrix objects.
<pre>MatrixExpr(*args, **kwargs)</pre>	Superclass for Matrix Expressions
<pre>MatrixPermute(mat, perm[, axis])</pre>	Symbolic representation for permuting matrix rows or columns.
MatrixSlice(parent, rowslice, colslice)	A MatrixSlice of a Matrix Expression
MatrixSymbol(name, n, m)	Symbolic representation of a Matrix object
Max(*args)	Return, if possible, the maximum value of the list.
<pre>MellinTransform(*args)</pre>	Class representing unevaluated Mellin transforms.
Min(*args)	Return, if possible, the minimum value of the list.
Mod(p, q)	Represents a modulo operation on symbolic expressions.
Monomial(monom[, gens])	Class representing a monomial, i.e. a product of powers.
Mul(*args[, evaluate, _sympify])	Expression representing multiplication operation for algebraic field.
MutableDenseMatrix(*args, **kwargs)	
<pre>MutableDenseNDimArray([iterable, shape])</pre>	
MutableMatrix	alias of MutableDenseMatrix
MutableSparseMatrix(*args, **kwargs)	
<pre>MutableSparseNDimArray([iterable, shape])</pre>	
NDimArray(iterable[, shape])	N-dimensional array.
Nand(*args)	Logical NAND function.
Ne	alias of Unequality
Nor(*args)	Logical NOR function.
Not(arg)	Logical Not function (negation)
Number(*obj)	Represents atomic numbers in SymPy.
NumberSymbol()	
0	alias of Order
OmegaPower(a, b)	Represents ordinal exponential and multiplication terms one of the building blocks of the Ordinal class.
<pre>OneMatrix(m, n[, evaluate])</pre>	Matrix whose all entries are ones.
Options(gens, args[, flags, strict])	Options manager for polynomial manipulation module.
Or(*args)	Logical OR function
Order(expr, *args, **kwargs)	Represents the limiting behavior of some function.
Ordinal(*terms)	Represents ordinals in Cantor normal form.
Parabola([focus, directrix])	A parabolic GeometryEntity.
Permanent(mat)	Matrix Permanent
PermutationMatrix(perm)	A Permutation Matrix
Piecewise(*_args)	Represents a piecewise function.
Plane(p1[, a, b])	A plane is a flat, two-dimensional surface.
Point(*args, **kwargs)	A point in a n-dimensional Euclidean space.
Point2D(*args[, _nocheck])	A point in a 2-dimensional Euclidean space.
Point3D(*args[, _nocheck])	A point in a 3-dimensional Euclidean space.
Poly(rep, *gens, **args)	Generic class for representing and operating on polyno-
	mial expressions.
	continues on next page

Table 68 – continued from previous page

Table 68 – continued from previous page			
Polygon(*args[, n])	A two-dimensional polygon.		
<pre>PolynomialRing(domain_or_ring[, symbols, order])</pre>	A class for representing multivariate polynomial rings.		
Pow(b, e[, evaluate])	Defines the expression $x^**y$ as "x raised to a power y"		
PowerSet(arg[, evaluate])	A symbolic object representing a power set.		
Predicate(*args, **kwargs)	Base class for mathematical predicates.		
<pre>Product(function, *symbols, **assumptions)</pre>	Represents unevaluated products.		
ProductSet(*sets, **assumptions)	Represents a Cartesian Product of Sets.		
PurePoly(rep, *gens, **args)	Class for representing pure polynomials.		
<pre>PythonFiniteField(mod[, symmetric])</pre>	Finite field based on Python's integers.		
PythonIntegerRing()	Integer ring based on Python's int type.		
PythonRational	alias of PythonMPQ		
QQ_gmpy	alias of GMPYRationalField		
QQ_python	alias of PythonRationalField		
Quaternion([a, b, c, d, real_field, norm])	Provides basic quaternion operations.		
Range(*args)	Represents a range of integers.		
Rational(p[, q, gcd])	Represents rational numbers (p/q) of any size.		
RationalField()	Abstract base class for the domain QQ.		
Ray(p1[, p2])	A Ray is a semi-line in the space with a source point and		
	a direction.		
Ray2D(p1[, pt, angle])	A Ray is a semi-line in the space with a source point and		
	a direction.		
Ray3D(p1[, pt, direction_ratio])	A Ray is a semi-line in the space with a source point and		
	a direction.		
RealField([prec, dps, tol])	Real numbers up to the given precision.		
RealNumber	alias of Float		
RegularPolygon(c, r, n[, rot])	A regular polygon.		
Rel	alias of Relational		
Rem(p, q)	Returns the remainder when p is divided by q where p is		
(p, 4)	finite and q is not equal to zero.		
RisingFactorial(x, k)	Rising factorial (also called Pochhammer symbol [1]_)		
<b>5</b>	is a double valued function arising in concrete mathe-		
	matics, hypergeometric functions and series expansions.		
RootOf(f, x[, index, radicals, expand])	Represents a root of a univariate polynomial.		
RootSum(expr[, func, x, auto, quadratic])	Represents a sum of all roots of a univariate polynomial.		
Segment(p1, p2, **kwargs)	A line segment in space.		
Segment2D(p1, p2, **kwargs)	A line segment in 2D space.		
Segment3D(p1, p2, **kwargs)	A line segment in a 3D space.		
SensitivityMatrix(sympification, duals,)			
( ) I			
SeqAdd(*args, **kwargs)	Represents term-wise addition of sequences.		
SeqFormula(formula[, limits])	Represents sequence based on a formula.		
SeqMul(*args, **kwargs)	Represents term-wise multiplication of sequences.		
SeqPer(periodical[, limits])	Represents a periodic sequence.		
Set(*args)	The base class for any kind of set.		
Shi(z)	Sinh integral.		
Si(z)	Sine integral.		
Sieve()	An infinite list of prime numbers, implemented as a dy-		
·	namically growing sieve of Eratosthenes.		
SineTransform(*args)	Class representing unevaluated sine transforms.		
SingularityFunction(variable, offset, exponent)	Singularity functions are a class of discontinuous func-		
, , , , , , , , , , , , , , , , , , ,	tions.		
SparseMatrix	alias of MutableSparseMatrix		
	continues on next page		

Table 68 – continued from previous page

lable 68 – continue	ed from previous page
SparseNDimArray(*args, **kwargs)	
StrPrinter([settings])	
StrictGreaterThan(lhs, rhs, **options)	Class representations of inequalities.
StrictLessThan(lhs, rhs, **options)	Class representations of inequalities.
Subs(expr, variables, point, **assumptions)	Represents unevaluated substitutions of an expression.
Sum(function, *symbols, **assumptions)	Represents unevaluated summation.
Symbol(name, **assumptions)	Assumptions:
SymmetricDifference(a, b[, evaluate])	Represents the set of elements which are in either of the
	sets and not in their intersection.
TableForm(data, **kwarg)	Create a nice table representation of data.
<pre>TestBabyModel(*args, **kwds)</pre>	
Trace(mat)	Matrix Trace
Transpose(*args, **kwargs)	The transpose of a matrix expression.
Triangle(*args, **kwargs)	A polygon with three vertices and three sides.
Tuple(*args, **kwargs)	Wrapper around the builtin tuple object.
<pre>Unequality(lhs, rhs, **options)</pre>	An unequal relation between two objects.
<pre>UnevaluatedExpr(arg, **kwargs)</pre>	Expression that is not evaluated unless released.
Union(*args, **kwargs)	Represents a union of sets as a Set.
<pre>Wild(name[, exclude, properties])</pre>	A Wild symbol matches anything, or anything without
	whatever is explicitly excluded.
WildFunction(*args)	A WildFunction function matches any function (with its
	arguments).
Xor(*args)	Logical XOR (exclusive OR) function.
Ynm(n, m, theta, phi)	Spherical harmonics defined as
ZZ_gmpy	alias of GMPYIntegerRing
ZZ_python	alias of PythonIntegerRing
ZeroMatrix(m, n)	The Matrix Zero 0 - additive identity
Znm(n, m, theta, phi)	Real spherical harmonics defined as
acos(arg)	The inverse cosine function.
acosh(arg)	acosh(x) is the inverse hyperbolic cosine of x.
acot(arg)	The inverse cotangent function.
acoth(arg)	acoth(x) is the inverse hyperbolic cotangent of x.
acsc(arg)	The inverse cosecant function.
acsch(arg)	acsch(x) is the inverse hyperbolic cosecant of x.  Conjugate transpose or Hermite conjugation.
adjoint(arg) airyai(arg)	The Airy function \$operatorname{Ai}\$ of the first kind.
airyairaig) airyaiprime(arg)	The derivative \$operatorname{Ai}^prime\$ of the Airy
allyalplime(dig)	function of the first kind.
airybi(arg)	The Airy function \$operatorname{Bi}\$ of the second
allybi(aig)	kind.
airybiprime(arg)	The derivative \$operatorname{Bi}^prime\$ of the Airy
allybipi ime(arg)	function of the first kind.
andre(n)	Andre numbers / Andre function
appellf1(a, b1, b2, c, x, y)	This is the Appell hypergeometric function of two vari-
(w, o., o., v, n, j)	ables as:
arg(arg)	Returns the argument (in radians) of a complex number.
asec(arg)	The inverse secant function.
asech(arg)	asech(x) is the inverse hyperbolic secant of x.
asin(arg)	The inverse sine function.
<i>( 6)</i>	continues on next page

Table 68 – continued from previous page

1able 68 – continue	d from previous page
asinh(arg)	asinh(x) is the inverse hyperbolic sine of x.
assoc_laguerre(n, alpha, x)	Returns the $n\$ th generalized Laguerre polynomial in $x\$ , $L_n(x)\$ .
assoc_legendre(n, m, x)	assoc_legendre(n, m, x) gives $P_n^m(x)$ , where $n$ and $m$ are the degree and order or an expression which is related to the nth order Legendre
atan(arg)	polynomial, $P_n(x)$ in the following manner: The inverse tangent function.
atan2(y, x)	The function atan2(y, x) computes operator-
	$name\{atan\}(y/x)$ taking two arguments y and x.
<pre>atanh(arg) bell(n[, k_sym, symbols])</pre>	atanh(x) is the inverse hyperbolic tangent of x. Bell numbers / Bell polynomials
	Bernoulli numbers / Bernoulli polynomials / Bernoulli
bernoulli(n[, x])	function
besseli(nu, z)	Modified Bessel function of the first kind.
besselj(nu, z)	Bessel function of the first kind.
besselk(nu, z)	Modified Bessel function of the second kind.
bessely(nu, z)	Bessel function of the second kind.
beta(x[,y])	The beta integral is called the Eulerian integral of the
1	first kind by Legendre:
betainc(*args)	The Generalized Incomplete Beta function is defined as
betainc_regularized(*args)	The Generalized Regularized Incomplete Beta function is given by
binomial(n, k)	Implementation of the binomial coefficient.
carmichael(*args)	Carmichael Numbers:
cartes	alias of product
catalan(n)	Catalan numbers
ceiling(arg)	Ceiling is a univariate function which returns the smallest integer value not less than its argument.
chebyshevt(n, x)	Chebyshev polynomial of the first kind, $T_n(x)$ .
chebyshevt_root(n, k)	chebyshev_root(n, k) returns the $k$ th root (indexed from zero) of the $n$ th Chebyshev polynomial of the first kind; that is, if $0 = k < n$ , chebyshevt(n, chebyshevt_root(n, k)) == 0.
chebyshevu(n, x)	Chebyshev polynomial of the second kind, $U_n(x)$ .
chebyshevu_root(n, k)	chebyshevu_root(n, k) returns the $k$ th root (indexed from zero) of the $n$ th Chebyshev polynomial of the second kind; that is, if $0 le k < n$ , chebyshevu(n, chebyshevu_root(n, k)) == $0$ .
conjugate(arg)	Returns the <i>complex conjugate</i> [1]_ of an argument.
cos(arg)	The cosine function.
cosh(arg)	cosh(x) is the hyperbolic cosine of x.
cot(arg)	The cotangent function.
coth(arg)	coth(x) is the hyperbolic cotangent of x.
csc(arg)	The cosecant function.
csch(arg)	csch(x) is the hyperbolic cosecant of x.
defaultdict	defaultdict(default_factory=None, /, [])> dict with default factory
$\operatorname{digamma}(z)$	The digamma function is the first derivative of the loggamma function
<pre>dirichlet_eta(s[, a])</pre>	Dirichlet eta function.
	continues on next page

Table 68 – continued from previous page

Table 68 – continue	d from previous page
<pre>divisor_sigma(n[, k])</pre>	Calculate the divisor function $sigma_k(n)$ for positive in-
	teger n
$elliptic_e(m[,z])$	Called with two arguments \$z\$ and \$m\$, evaluates the
	incomplete elliptic integral of the second kind, defined
	by
$elliptic_f(z, m)$	The Legendre incomplete elliptic integral of the first
	kind, defined by
elliptic_k(m)	The complete elliptic integral of the first kind, defined
	by
$elliptic_pi(n, m[, z])$	Called with three arguments \$n\$, \$z\$ and \$m\$, evalu-
	ates the Legendre incomplete elliptic integral of the third
	kind, defined by
erf(arg)	The Gauss error function.
erf2(x, y)	Two-argument error function.
erf2inv(x, y)	Two-argument Inverse error function.
erfc(arg)	Complementary Error Function.
erfcinv(z)	Inverse Complementary Error Function.  Imaginary error function.
<pre>erfi(z) erfinv(z)</pre>	Inverse Error Function.
extraction extraction for each of the contraction	Euler numbers / Euler polynomials / Euler function
exp(arg)	The exponential function, $e^x$ .
exp_polar(*args)	Represent a <i>polar number</i> (see g-function Sphinx docu-
exp_potat( args)	mentation).
expint(nu, z)	Generalized exponential integral.
factorial(n)	Implementation of factorial function over nonnegative
	integers.
factorial2(arg)	The double factorial $n!!$ , not to be confused with $(n!)!$
ff	alias of FallingFactorial
<pre>fibonacci(n[, sym])</pre>	Fibonacci numbers / Fibonacci polynomials
floor(arg)	Floor is a univariate function which returns the largest
	integer value not greater than its argument.
frac(arg)	Represents the fractional part of x
fresnelc(z)	Fresnel integral C.
fresnels(z)	Fresnel integral S.
gamma(arg)	The gamma function
gegenbauer(n, a, x)	Gegenbauer polynomial $C_n^{\ell}(st)$
genocchi(n[, x])	Genocchi numbers / Genocchi polynomials / Genocchi
1 1 14/	function
hankel1(nu, z)	Hankel function of the ground hind
hankel2(nu, z)	Hankel function of the second kind.
harmonic(n[, m])	Harmonic numbers
hermite(n, x)	hermite(n, x) gives the \$n\$th Hermite polynomial in
hermite_prob(n, x)	<pre>\$x\$, \$H_n(x)\$. hermite_prob(n, x) gives the \$n\$th probabilist's</pre>
HETHITCE_PIOD(II, A)	Hermite polynomial in $x$ ,
hn1(nu, z)	Spherical Hankel function of the first kind.
hn2(nu, z)	Spherical Hankel function of the second kind.
hyper(ap, bq, z)	The generalized hypergeometric function is defined by a
)1 (-r, - 1, -/	series where the ratios of successive terms are a rational
	function of the summation index.
im(arg)	Returns imaginary part of expression.
jacobi(n,a,b,x)	Jacobi polynomial \$P_n^{left(alpha, betaright)}(x)\$.
	continues on next page

Table 68 – continued from previous page

Table 68 – continue	d from previous page
jn(nu, z)	Spherical Bessel function of the first kind.
laguerre(n, x)	Returns the \$n\$th Laguerre polynomial in \$x\$,
	$L_n(x)$ .
legendre(n, x)	legendre(n, x) gives the \$n\$th Legendre polynomial
	of $x$ , $P_n(x)$
lerchphi(*args)	Lerch transcendent (Lerch phi function).
li(z)	The classical logarithmic integral.
ln	alias of log
log(arg[, base])	The natural logarithm function $ln(x)$ or $log(x)$ .
loggamma(z)	The loggamma function implements the logarithm of the
	gamma function (i.e., \$logGamma(x)\$).
lowergamma(a, x)	The lower incomplete gamma function.
lucas(n)	Lucas numbers
marcumq(m, a, b)	The Marcum Q-function.
mathieuc(a, q, z)	The Mathieu Cosine function $C(a,q,z)$ .
$\operatorname{mathieucprime}(a, q, z)$	The derivative $C^{(a,q,z)}$ of the Mathieu Co-
	sine function.
mathieus(a, q, z)	The Mathieu Sine function $S(a,q,z)$ .
$\operatorname{mathieusprime}(a, q, z)$	The derivative $S^{\{prime\}}(a,q,z)$ of the Mathieu Sine
	function.
meijerg(*args)	The Meijer G-function is defined by a Mellin-Barnes
	type integral that resembles an inverse Mellin transform.
mobius(n)	Mobius function maps natural number to {-1, 0, 1}
motzkin(n)	The nth Motzkin number is the number
multigamma(x, p)	The multivariate gamma function is a generalization of
	the gamma function
partition(n)	Partition numbers
periodic_argument(ar, period)	Represent the argument on a quotient of the Riemann
1 1:6./	surface of the logarithm.
polar_lift(arg)	Lift argument to the Riemann surface of the logarithm,
	using the standard branch.
polygamma(n, z)	The function polygamma(n, z) returns
malulan(s, z)	log(gamma(z)).diff(n + 1).
<pre>polylog(s, z) preorder_traversal(node[, keys])</pre>	Polylogarithm function.  Do a pre-order traversal of a tree.
primenu(n)	Calculate the number of distinct prime factors for a pos-
primenu(n)	itive integer n.
primeomega(n)	Calculate the number of prime factors counting multi-
primeomega(n)	plicities for a positive integer n.
primepi(n)	Represents the prime counting function $pi(n)$ = the num-
FF-(11)	ber of prime numbers less than or equal to n.
<pre>principal_branch(x, period)</pre>	Represent a polar number reduced to its principal branch
ppur_stunch(n, poriou)	on a quotient of the Riemann surface of the logarithm.
re(arg)	Returns real part of expression.
reduced_totient(n)	Calculate the Carmichael reduced totient function
<u> </u>	lambda(n)
rf	alias of RisingFactorial
riemann_xi(s)	Riemann Xi function.
sec(arg)	The secant function.
sech(arg)	sech(x) is the hyperbolic secant of x.
sign(arg)	Returns the complex sign of an expression:
sin(arg)	The sine function.
	continues on next page

Table 68 – continued from previous page

sinc(arg)	Represents an unnormalized sinc function:
sinh(arg)	sinh(x) is the hyperbolic sine of x.
<pre>stieltjes(n[, a])</pre>	Represents Stieltjes constants, \$gamma_{k}\$ that occur in Laurent Series expansion of the Riemann zeta function.
subfactorial(arg)	The subfactorial counts the derangements of \$n\$ items
	and is defined for non-negative integers as:
tan(arg)	The tangent function.
tanh(arg)	tanh(x) is the hyperbolic tangent of x.
totient(n)	Calculate the Euler totient function phi(n)
transpose(arg)	Linear map transposition.
<pre>tribonacci(n[, sym])</pre>	Tribonacci numbers / Tribonacci polynomials
trigamma(z)	The trigamma function is the second derivative of the
	loggamma function
uppergamma(a, z)	The upper incomplete gamma function.
vectorize(*mdargs)	Generalizes a function taking scalars to accept multidi-
	mensional arguments.
yn(nu, z)	Spherical Bessel function of the second kind.
zeta(s[, a])	Hurwitz zeta function (or Riemann zeta function).

# **Exceptions**

BasePolynomialError	Base class for polynomial related exceptions.
CoercionFailed	
ComputationFailed(func, nargs, exc)	
DomainError	
EvaluationFailed	
${\tt ExactQuotientFailed}(f,g[,dom])$	
ExtraneousFactors	
FlagError	
GeneratorsError	
GeneratorsNeeded	
GeometryError	An exception raised by classes in the geometry module.
HeuristicGCDFailed	· · · · · · · · · · · · · · · · · · ·
HomomorphismFailed	
IsomorphismFailed	
MultivariatePolynomialError	
	continues on next page

## Table 69 – continued from previous page

NonSquareMatrixError NotAlgebraic NotInvertible NotReversible OperationNotSupported(poly, func) OptionError PoleError PolificationFailed(opt, origs, exprs[, seq]) PolynomialDivisionFailed(f, g, domain) PolynomialError PrecisionExhausted RefinementFailed ShapeError Wrong matrix shape SympifyError(expr[, base\_exc]) UnificationFailed UnivariatePolynomialError

src.sensitivity.speed\_test.run\_timed(params, parameter\_name)

generates an instance of TestBabyModel with given parameters, sympifies it, and follows the sequence of steps

to get the sensitivity matrix. Prints times for each step.

#### **Parameters**

- params (dict) dictionary of parameter values for the TestBabyModel instance generated. These are set in the beginning of the script. Values can be changed in the declaration statements.
- **parameter\_name** (*str*) the name of the scalar parameter to evaluate sensitivities with respect to.

## Returns

tuple of model to be solved, the sympification of the model, the SensitivityMatrix, DifferentialMapping associated with parameter and model solve values

## Return type

tuple

src.sensitivity.speed\_test.speed\_accuracy\_test(params, parameter\_name, p)

# performs a run\_timed execution with given parameters, takes the returned values, and measures the effect of increasing and

decreasing the named parameter by p, and compares the result to resolving the model with that same perturbation.

## Returns

- **params** (*dict*) dictionary of parameter values for the TestBabyModel instance generated. These are set in the beginning of the script. Values can be changed in the declaration statements.
- **parameter\_name** (*str*) the name of the scalar parameter to evaluate sensitivities with respect to.
- **p** (*float*) percentage change up and down to be measured, as a decimal.

## **PYTHON MODULE INDEX**

```
S
                                                       30
                                               src.models.residential.scripts, 31
src, 34
                                               src.models.residential.scripts.residential,
src.common, 34
                                                       31
src.common.config_setup, 35
                                               src.sensitivity, 54
src.common.model, 37
                                               src.sensitivity.babymodel, 55
src.common.utilities, 44
                                               src.sensitivity.faster_sensitivity, 81
src.integrator, 46
                                               src.sensitivity.sensitivity_tools,85
src.integrator.gaussseidel, 46
                                               src.sensitivity.speed_test, 113
src.integrator.progress_plot, 47
src.integrator.runner, 47
src.integrator.unified, 48
src.integrator.utilities, 50
src.models.2
src.models.electricity, 2
src.models.electricity.scripts, 3
src.models.electricity.scripts.electricity_model,
src.models.electricity.scripts.postprocessor,
src.models.electricity.scripts.preprocessor,
src.models.electricity.scripts.runner, 11
src.models.electricity.scripts.utilities, 13
src.models.hydrogen, 15
src.models.hydrogen.model, 15
src.models.hydrogen.model.actions, 16
src.models.hydrogen.model.h2_model, 17
src.models.hydrogen.model.validators, 20
src.models.hydrogen.network, 20
src.models.hydrogen.network.grid, 21
src.models.hydrogen.network.grid_data, 23
src.models.hydrogen.network.hub, 23
src.models.hydrogen.network.region, 25
src.models.hydrogen.network.registry, 26
src.models.hydrogen.network.transportation_arc,
src.models.hydrogen.utilities, 28
src.models.hydrogen.utilities.h2_functions,
src.models.residential, 30
src.models.residential.preprocessor, 30
src.models.residential.preprocessor.generate_inputs,
```

# **INDEX**

\spxentry_active\spxextrasrc.common.model.Model \spxentry_filter_update_info()\spxextrasrc.models.hydrogen.model.h2_model attribute, 37 method, 18
\spxentry_active\spxextrasrc.models.electricity.scripts.electricity.el
\spxentry_active\spxextrasrc.models.electricity.scripts.utilitiesp\(\mathbb{E}\) beattyic_ityal\(\mathbb{E}\) the tryic_ityal\(\mathbb{E}\) the tryic_ityal\
\spxentry_active\spxextrasrc.models.hydrogen.model.h2_modpk&h2?Modpake()\spxextrasrc.integrator.utilities.HI class attribute, 18 method, 51
\spxentry_active\spxextrasrc.sensitivity.babymodel.TestBaby\\delta deltry_replace()\spxextrasrc.integrator.utilities.EI attribute, 79 method, 51
\spxentry_additional_year_settings()\spxextrasrc.common.cosping_ntetyupreplanteg)\spxtixxgsasrc.integrator.utilities.HI method, 35 method, 51
\spxentry_asdict()\spxextrasrc.integrator.utilities.EI \spxentry_update_demand()\spxextrasrc.models.hydrogen.model.h2_model method, 51 method, 19
\spxentry_asdict()\spxextrasrc.integrator.utilities.HI \spxentry_update_electricity_price()\spxextrasrc.models.hydrogen.model.h/method, 51 method, 19
\spxentry_check_elec_expansion_settings()\spxextrasrc.common.config_settup.Config_settings method, 35 \spxentryabsorb_subregions()\spxextrasrc.models.hydrogen.network.region
\spxentry_check_int()\spxextrasrc.common.config_setup.Config_settingsthod, 25 method, 36 \spxentryabsorb_subregions_deep()\spxextrasrc.models.hydrogen.network.ii
\spxentry_check_regions()\spxextrasrc.common.config_setup.Config_setupdd, 25 method, 36 \spxentryadd()\spxextrasrc.models.hydrogen.network.registry.Registry
\spxentry_check_res_settings()\spxextrasrc.common.config_setup.Config_lsettings method, 36 \spxentryadd_hub()\spxextrasrc.models.hydrogen.network.region.Region
\spxentry_check_true_false()\spxextrasrc.common.config_setup.Config_setup.Config_setup.gs5 method, 36 \spxentryadd_inbound()\spxextrasrc.models.hydrogen.network.hub.Hub
\spxentry_check_zero_one()\spxextrasrc.common.config_setup.Config_setup.Qonfig_setu
\spxentry_declare_set_with_dict()\spxextrasrc.common.model.Model method, 24 method, 38 \spxentryadd_season_index()\spxextrain module
\spxentry_declare_set_with_iterable()\spxextrasrc.common.model.Moster.models.electricity.scripts.preprocessor, method, 38
\spxentry_declare_set_with_pandas()\spxextrasrc.common.n\spxern\nyard_subregion()\spxextrasrc.models.hydrogen.network.region.Reg method, 39 method, 25
\spxentry_field_defaults\spxextrasrc.integrator.utilities.EI \spxentryaggregate_hubs()\spxextrasrc.models.hydrogen.network.grid.Grid method, 21
\spxentry_field_defaults\spxextrasrc.integrator.utilities.HI \spxentryaggregate_subregion_data()\spxextrasrc.models.hydrogen.network attribute, 51 method, 25
\spxentry_fields\spxextrasrc.integrator.utilities.EI at-\spxentryannual_count()\spxextrain module src.models.electricity.scripts.utilities, 14
\spxentry_fields\spxextrasrc.integrator.utilities.HI at-\spxentryarc_generation()\spxextrasrc.models.hydrogen.network.grid.Grid method, 21

```
class method, 37
                                                                                               method, 22
\spxentryassigned names\spxextrasrc.models.hydrogen.netw\spkeretgivonRecgitorelec price to lut()\spxextrain module
              attribute, 25
                                                                                               src.integrator.utilities, 51
\spxentryAutoSympy\spxextraclass
                                                                                 \spxentryconvert_h2_price_records()\spxextrain module
             src.sensitivity.faster sensitivity, 81
                                                                                               src.integrator.utilities, 52
\spxentryAutoSympy\spxextraclass
                                                                                 \spxentryCoordMap\spxextraclass
                                                                                                                                                            in
              src.sensitivity.sensitivity_tools, 110
                                                                                               src.sensitivity.sensitivity_tools, 111
\spxentryavg_by_group()\spxextrain
                                                                    module
                                                                                 \spxentrycost()\spxextrasrc.models.hydrogen.network.hub.Hub
             src.models.electricity.scripts.preprocessor,
                                                                                               method, 24
                                                                                 \spxentrycost_learning_func()\spxextrain
                                                                                                                                                     module
                                                                                               src.models.electricity.scripts.runner, 12
\spxentrybase_price()\spxextrain
                                                                    module
                                                                                 \verb|\spxentrycreate_arc()\spxextrasrc.models.hydrogen.network.grid.Grid|
              src.models.residential.preprocessor.generate_inputs,
                                                                                               method, 22
                                                                                 \spxentrycreate_hourly_params()\spxextrain
                                                                                                                                                     module
\spxentrybaseYear\spxextrasrc.models.residential.scripts.residential.residential.yodule.scripts.preprocessor, 7
              attribute, 31
                                                                                 \spxentrycreate_hourly_sets()\spxextrain
                                                                                                                                                     module
\spxentrybuild()\spxextrasrc.common.model.Model class
                                                                                               src.models.electricity.scripts.preprocessor,
              method, 39
\spxentrybuild_elec_model()\spxextrain
                                                                    module
                                                                                 \spxentrycreate hub()\spxextrasrc.models.hydrogen.network.grid.Grid
             src.models.electricity.scripts.runner, 12
                                                                                               method, 22
\spxentrybuild grid()\spxextrain
                                                                    module
                                                                                 \spxentrycreate model()\spxextrasrc.sensitivity.faster sensitivity.toy model
              src.models.hydrogen.model.actions, 16
                                                                                               method, 85
module
              method, 21
                                                                                               src.models.electricity.scripts.utilities, 15
\spxentrybuild_model()\spxextrain
                                                                    module
                                                                                 \spxentrycreate other sets()\spxextrain
                                                                                                                                                     module
              src.models.hydrogen.model.actions, 16
                                                                                               src.models.electricity.scripts.preprocessor,
\spxentrycapacitycredit_df()\spxextrain
                                                                    module
                                                                                 \spxentrycreate_region()\spxextrasrc.models.hydrogen.network.grid.Grid
              src.models.electricity.scripts.preprocessor,
                                                                                               method, 22
\spxentrychange_destination()\spxextrasrc.models.hydrogen.network.transportation_are_fransportationArc src.models.electricity.scripts.preprocessor,
                                                                                                                                                     module
              method, 27
\spxentrychange_origin()\spxextrasrc.models.hydrogen.network.transportation_arc.TransportationArc \spxentrycreate_subregion()\spxextrasrc.models.hydrogen.network.region.R
              method, 27
\verb|\spxentrychange_region()| \verb|\spxextrasrc.models.hydrogen.network.hub.Hubthod|, 25
                                                                                 \spxentrycreate_subsets()\spxextrain
                                                                                                                                                     module
              method, 24
\verb|\spxentrycheck_complimentarity_all()| \verb|\spxextrasrc.sensitivity.faster_sensitivity. AutoSympy | Sensitivity |
\spxentrycheck_complimentarity_all()\spxextrasrc.sensitivity.sensitivity_toois.Altrosympy method, 110
             method, 110
                                                                                 \spxentrycreate temporal mapping()\spxextrain module
                                                                    module
\spxentrycheck_results()\spxextrain
                                                                                               src.integrator.utilities, 52
              src.models.electricity.scripts.utilities, 15
method, 21
                                                                                             method, 39
method, 21
method, 22
method, 40 \spxentrycomplex_step_sensitivity()\spxextrasrc.models.residential.scripts.residential.residentialModule \spxentrydeclare_set_with_sets()\spxextrasrc.common.model.Model
              method, 31
                                                                                               method, 41
\spxentryConfig_settings\spxextraclass
                                                                                 \spxentrydeclare_shifted_time_set()\spxextrasrc.common.model.Model
             src.common.config_setup, 35
                                                                                               method, 41
\spxentryconnect_regions()\spxextrain
                                                                    module
                                                                                 \spxentrydeclare_var()\spxextrasrc.common.model.Model
              src.sensitivity.babymodel, 80
```

method, 42 \spxentrydelete()\spxextrasrc.models.hydrogen.networmethod, 22	rk.gric	d.Grid	<pre>get_elec_price()\spxextrain src.integrator.utilities, 52 get_elec_price()\spxextrain</pre>		module	
\spxentrydelete()\spxextrasrc.models.hydrogen.networ method, 25			e – 1		module ons,	
\spxentrydemandF()\spxextrasrc.models.residential.sc method, 32	cripts.r	esipokemtia)	getsidhattirihMydcdensumptio module src.models.hydrog			
\spxentryDifferentialMapping\spxextraclass	in		29	cm.uumues.n2	_ranctions,	
src.sensitivity.sensitivity_tools, 111			get_electricty_consumptior			
\spxentrydisconnect()\spxextrasrc.models.hydrogen.no method, 28	etwork	c.transpor	atlens <u>r</u> eren Ed <del>el</del> sphydatigen Au 29	r <b>ti</b> lities.h2_fui	nctions,	
\spxentrydisplay_children()\spxextrasrc.models.hydro	gen.ne	etspoudn.neg			module	
method, 26 \spxentrydisplay_hubs()\spxextrasrc.models.hydrogen	netw	ork region	src.models.hydrogen.utiliti	es.h2_functio	ons,	
method, 26			get_objective()\spxextrasrc.	sensitivity.fas	ster sensitivity.AutoSyn	nr
\spxentrydisplay_outbound()\spxextrasrc.models.hydr						r
method, 24		\spxentry	get_objective()\spxextrasrc.method, 110	sensitivity.sei	nsitivity_tools.AutoSym	ıp
\spxentryEI\spxextraclass in src.integrator.utilities, 51 \spxentryElectricityMethods\spxextraclass	in	\spxentry	get_parameters()\spxextrasi method, 82	rc.sensitivity.f	faster_sensitivity.AutoS	yn
src.models.electricity.scripts.utilities, 14 \spxentryextrapolate()\spxextrasrc.sensitivity.sensitivi	ty_toc	\spxentry ols.Differe	get_parameters()\spxextrasi ntialMapping	rc.sensitivity.s	sensitivity_tools.AutoSy	/n
method, 111		\spxentry	get_partial()\spxextrasrc.ser method, 83	nsitivity.faster	_sensitivity.Sensitivityl	M
src.models.electricity.scripts.preprocessor,	dule	\spxentry	get_partials_matrix()\spxex method, 83	trasrc.sensitiv	vity.faster_sensitivity.Se	ns
8		\spxentry	get_production_cost()\spxe		module	
\spxentrygenerate()\spxextrain mod	dule		src.models.hydrogen.utiliti			
src.sensitivity.babymodel, 80			get_sensitivities()\spxextras	src.sensitivity.	sensitivity_tools.Sensit	ĺV
\spxentrygenerate_duals()\spxextrasrc.sensitivity.faste method, 81		Spacifu y	gct_schsinvity_manix() sp.	xextrasrc.sens	itivity.faster_sensitivity	.A
\spxentrygenerate_duals()\spxextrasrc.sensitivity.sens method, 110	itivity <sub>.</sub>	_tools.Au \spxentry	desympty 82 get_sensitivity_matrix()\sp:	xextrasrc.sens	itivity.sensitivity_tools.	A
\spxentrygenerate_matrix()\spxextrasrc.sensitivity.fast method, 83		spacing y	gct_scrisitivity_rangc()\spx			Se
\spxentrygenerate_matrix()\spxextrasrc.sensitivity.sen method, 112	sitivit			tivity.faster_s	ensitivity.AutoSympy	
\spxentryget_annual_wt_avg()\spxextrain mod src.integrator.utilities, 52	dule		method, 82 get_sets()\spxextrasrc.sensi	tivity.sensitiv	ity_tools.AutoSympy	
src.common.utilities, 45			method, 111 get_variables()\spxextrasrc.	sensitivity.fas	ster_sensitivity.AutoSyn	np
\spxentryget_constraints()\spxextrasrc.sensitivity.faste method, 81		spacing y	gct_variables()\spacatiasic.	sensitivity.sei	nsitivity_tools.AutoSym	ıp
\spxentryget_constraints()\spxextrasrc.sensitivity.sens method, 110	itivity	tools.Au \spxentry	Offuspacauaciass		in	
\spxentryget_data()\spxextrasrc.models.hydrogen.netv method, 24		\spxentry	src.models.hydrogen.netwo GridData\spxextraclass	-	in	
\spxentryget_data()\spxextrasrc.models.hydrogen.netv	work.re	egion.Reg	src.models.hydrogen.netwo	ork.grid_data,	, 23	
method, 26			H2Model\spxextraclass		in	
\spxentryget_demand()\spxextrain mod src.models.hydrogen.utilities.h2_functions,	dule		src.models.hydrogen.mode		18	
src.models.nydrogen.utilities.nz_functions,			HI\spxextraclass in src.integ	-		
\spxentryget_duals()\spxextrasrc.common.model.Mod method, 43	del	\spxentry	hour\spxextrasrc.integrator. 51	utilities.EI at	tribute,	

\spxentryhourly_sc_subset()\spxextrain mod	lule \spxentrysrc.common.utilities, 44
src.models.electricity.scripts.preprocessor,	\spxentrysrc.integrator, 46
8	\spxentrysrc.integrator.gaussseidel, 46
\spxentryhr_map\spxextrasrc.models.residential.scripts	s.residentia\specidentiad\\htdghetor.progress_plot, 47
attribute, 32	\spxentrysrc.integrator.runner, 47
\spxentryhr_sub_sc_subset()\spxextrain mod	lule \spxentrysrc.integrator.unified, 48
src.models.electricity.scripts.preprocessor,	\spxentrysrc.integrator.utilities, 50
9	\spxentrysrc.models, 2
\spxentryHub\spxextraclass	in \spxentrysrc.models.electricity, 2
src.models.hydrogen.network.hub, 23	\spxentrysrc.models.electricity.scripts, 3
	\spxentrysrc.models.electricity.scripts.electricity_model,
\spxentryinit_old_cap()\spxextrain mod	iule 3
src.models.electricity.scripts.runner, 12	\spxentrysrc.models.electricity.scripts.postprocessor,
\spxentryinvert_U()\spxextrasrc.sensitivity.faster_sensi	itivity.SensitivityMatrix
method, 84	\spxentrysrc.models.electricity.scripts.preprocessor,
	5
\spxentryload_data()\spxextrain mod	\spxentrysrc.models.electricity.scripts.runner, 11
src.models.electricity.scripts.preprocessor,	\spxentrysrc.models.electricity.scripts.utilities, 13
9	\spxentrysrc.models.hydrogen, 15
\spxentryload_data()\spxextrain mod	\spxentrysrc.models.hydrogen.model, 15
src.models.hydrogen.model.actions, 17	\spxentrysrc.models.hydrogen.model.actions, 16
\spxentryload_hubs()\spxextrasrc.models.hydrogen.net	\spxentrysrc.models.hydrogen.model.actions, 16 twork.grid.\Grid \spxentrysrc.models.hydrogen.model.h2_model, 17
method, 22	
\spxentryloads\spxextrasrc.models.residential.scripts.re	\spxentrysrc.models.hydrogen.model.validators, 20 esidential.residentialModule \spxentrysrc.models.hydrogen.network, 20
attribute, 32	\spxentrysrc.models.hydrogen.network.grid, 21
\cnyantrymalza blaak/\\cnyaytracra madala rasidantial	.scripts.residential.residential.Module .scripts.residential.residential.Module .sprentrysrc models bydrogen network hub. 23
mathod 32	\spxentrysrc.models.hydrogen.network.hub, 23
method, 32	\spyantryere models hydrogen network region 25
\spxentrymake_dir()\spxextrain mod	\spxentrysrc.models.hydrogen.network.registry, 26
src.common.utilities, 45	Sprantygra models bydragan naturals transportation are
\spxentrymake_elec_output_dir()\spxextrain mod	27
src.models.electricity.scripts.postprocessor,	\coventryere models bydrogen utilities 28
\spxentrymake_h2_outputs()\spxextrain mod	\spxentrysrc.models.hydrogen.utilities.h2_functions,
src.models.hydrogen.model.actions, 17	20
\spxentrymakedir()\spxextrain mod	\spxentrysrc.models.residential, 30
src.models.electricity.scripts.preprocessor,	\spxentrysrc.models.residential.preprocessor, 30
9	ter_sensitivity.Sensitivity.Matrix
method 84	30
\spxentrymatrix_assembly()\spxextrasrc.sensitivity.sen method, 112	nsitivity_tools.selfsitytydarfx.residential.scripts, 31 \spxentrysrc.models.residential.scripts.residential,
\spxentrymatrix_sub()\spxextrasrc.sensitivity.faster_se method, 84	ensitivity.SensitivityMatrix \spxentrysrc.sensitivity, 54
\spxentryModel\spxextraclass in src.common.model, 3	\spxentrysrc.sensitivity.babymodel, 55
	\spycentrysrc sensitivity faster, sensitivity, 81
\spxentryModel.ConstraintExpression\spxextraclass	\spxentrysrc.sensitivity.sensitivity_tools, 85
src.common.model, 37	\covantryere cancitivity enand test 113
\spxentryModel.DefaultDecorator\spxextraclass	in spacinitysic.sensitivity.specu_test, 113
src.common.model, 37	\spxentrynew_jacobian()\spxextrasrc.sensitivity.faster_sensitivity.Sensitivity
\spxentryModel.ParameterExpression\spxextraclass	in method, 84
src.common.model, 37	
\spxentrymodule	\spxentryoutput_inputs()\spxextrain module
\spxentrysrc, 34	src.models.electricity.scripts.preprocessor,
\spxentrysrc.common, 34	9
\snxentrysrc common config setup 35	

\spxentrysrc.common.model, 37

\spxentryplot_it()\spxextrain src.integrator.progress_plot, 47	module	\spxentryRegion\spxextraclass src.models.hydrogen.network.region, 25	in
\spxentryplot_price_distro()\spxextrain src.integrator.progress_plot, 47	module	\spxentryregion\spxextrasrc.integrator.utilities.EI tribute, 51	at-
	dels.hydro	gesprocondes/rhe@ionordes/elekt@Mondin/tegrator.utilities.HI	at-
method, 19	,	tribute, 51	
\spxentrypoll_h2_demand()\spxextrain src.integrator.utilities, 52	module	\spxentryregion_validator()\spxextrain m src.models.hydrogen.model.validators, 20	nodule
\spxentrypoll_h2_prices_from_elec()\spxextrain src.integrator.utilities, 53	module	• •	nodule
\spxentrypoll_hydrogen_price()\spxextrain src.integrator.utilities, 53	module	\spxentryRegistry\spxextraclass src.models.hydrogen.network.registry, 27	in
•	module	\spxentryremove()\spxextrasrc.models.hydrogen.net method, 27	
	rc.models	. Aspecteriatityre on ipts_utrillio(i) Aspecteria i try Modellob dhydroge	n.network.region.Regio
method, 14		method, 26	
	spxextrasro	c.\spodett:;yteatrivetyinbriptst(i)t\streex Edeart:iritojlMethroi	hogen.network.hub.Hub
method, 14	•	method, 24	C
\spxentrypopulate_hydro_sets_rule()\spxextrasrc.	models.el	ekstpiceitytsycneiptoxut <u>i</u> löttielschilod())is jityelxlitetlsocksmodels.hy	drogen.network.hub.Hu
method, 14		method, 24	
lem:lem:lem:lem:lem:lem:lem:lem:lem:lem:	rc.models	.Akpoteintityreenipte <u>.u</u> stiibtiessi.biheidspixityiVastleadhodels.hy	ydrogen.network.region.
method, 14		method, 26	
\spxentrypopulate_RM_sets_rule()\spxextrasrc.m method, 14	odels.elec	ethispityntryiptongtilities. Fildetrisety/Mpthexdsasrc.commonths method, 43	on.model.Model
\spxentrypopulate_sets_rule()\spxextrasrc.commo	on.model.l	Msphedntryreport_obj_df()\spxextrain m	nodule
method, 43		src.models.electricity.scripts.postprocesso	or,
lem:lem:lem:lem:lem:lem:lem:lem:lem:lem:	models.ele		
method, 14		\spxentryresidentialModule\spxextraclass	in
\spxentrypostprocessor()\spxextrain src.models.electricity.scripts.postproces	module ssor,	src.models.residential.scripts.residential, 31	
4		\spxentryresolve()\spxextrain m	nodule
\spxentryPowerModel\spxextraclass	in	src.models.hydrogen.model.h2_model, 19	)
src.models.electricity.scripts.electricity	_model,	\spxentryresolve_kronecker()\spxextrasrc.sensitivity method, 113	sensitivity_tools.Sensit
\spxentrypreprocessor()\spxextrain	module	\spxentryrun_elec_model()\spxextrain m	nodule
src.models.electricity.scripts.preproces	sor,	src.models.electricity.scripts.runner, 12	
10		· · · · · · · · · · · · · · · · · · ·	nodule
\spxentryprices\spxextrasrc.models.residential.sc	ripts.resid		
attribute, 33		1 , 1	nodule
\spxentryprint_sets()\spxextrain	module	src.integrator.gaussseidel, 47	
src.models.electricity.scripts.preproces	sor,	1	nodule
10		src.integrator.runner, 48	
\\( \)			nodule
\spxentryquick_summary()\spxextrain	module	src.models.hydrogen.model.actions, 17	
src.models.hydrogen.model.actions, 17		1 7 - 1	nodule
\spxentryreadin_csvs()\spxextrain	module	src.models.residential.scripts.residential, 34	
src.models.electricity.scripts.preproces	sor,	\spxentryrun_residential_solo()\spxextrain m	nodule
10		src.integrator.runner, 48	
\spxentryreadin_sql()\spxextrain	module	•	nodule
src.models.electricity.scripts.preproces		src.integrator.runner, 48	
10 \spxentryrecursive_region_generation()\spxextras	src.models	\spxentryrun_timed()\spxextrain m .hydrogen.network.grid Grid test, 138	nodule
method, 22			nodule

src.integrator.unified, 49		\spxentrymodule, 47
		\spxentrysrc.integrator.unified
\spxentryscale_load()\spxextrain	module	\spxentrymodule, 48
src.common.utilities, 45		\spxentrysrc.integrator.utilities
lem:lem:lem:lem:lem:lem:lem:lem:lem:lem:	module	\spxentrymodule, 50
src.common.utilities, 45		\spxentrysrc.models
\spxentryselect_solver()\spxextrain	module	\spxentrymodule, 2
src.integrator.utilities, 54		\spxentrysrc.models.electricity
$\verb \spxentrysensitivity() \verb \spxextrasrc.models.residen  \\$	tial.script	s.residential-nesidential-Module
method, 33		\spxentrysrc.models.electricity.scripts
$\verb \spxentrysensitivity  () \verb \spxextrasrc.sensitivity.sens  \\$	itivity_too	ols.Differential Manning, 3
method, 112		\spxentrysrc.models.electricity.scripts.electricity_model
\spxentrySensitivityMatrix\spxextraclass	in	\spxentrymodule, 3
src.sensitivity.faster_sensitivity, 83		\spxentrysrc.models.electricity.scripts.postprocessor
$\verb \spxentrySensitivityMatrix\spxextraclass  \\$	in	\spxentrymodule, 4
src.sensitivity.sensitivity_tools, 112		\spxentrysrc.models.electricity.scripts.preprocessor
\spxentryset_new_cap()\spxextrain	module	\spxentrymodule, 5
src.models.electricity.scripts.runner, 13	3	\spxentrysrc.models.electricity.scripts.runner
\spxentrySets\spxextraclass	in	\spxentrymodule, 11
src.models.electricity.scripts.preproces	sor,	\spxentrysrc.models.electricity.scripts.utilities
6		\spxentrymodule, 13
\spxentrysetup_logger()\spxextrain	module	\spxentrysrc.models.hydrogen
src.common.utilities, 45		\spxentrymodule, 15
\spxentrysimple_solve()\spxextrain	module	\spxentrysrc.models.hydrogen.model
src.integrator.utilities, 54		\spxentrymodule, 15
\spxentrysimple_solve_no_opt()\spxextrain	module	\spxentrysrc.models.hydrogen.model.actions
src.integrator.utilities, 54		\spxentrymodule, 16
\spxentrysolve()\spxextrain	module	\spxentrysrc.models.hydrogen.model.h2_model
src.models.hydrogen.model.h2_model,		\spxentrymodule, 17
$\verb \spxentrysolve()\spxextrasrc.sensitivity.babymodeledge                                      $	el.TestBab	Whathrysrc.models.hydrogen.model.validators
method, 79		\spxentrymodule, 20
\spxentrysolve_elec_model()\spxextrain	module	\spxentrysrc.models.hydrogen.network
src.models.electricity.scripts.runner, 13	3	\spxentrymodule, 20
\spxentrysolve_it()\spxextrain	module	\spxentrysrc.models.hydrogen.network.grid
src.models.hydrogen.model.actions, 17	7	\spxentrymodule, 21
\spxentryspeed_accuracy_test()\spxextrain	module	\spxentrysrc.models.hydrogen.network.grid_data
src.sensitivity.speed_test, 138		\spxentrymodule, 23
\spxentrysrc		\spxentrysrc.models.hydrogen.network.hub
\spxentrymodule, 34		\spxentrymodule, 23
\spxentrysrc.common		\spxentrysrc.models.hydrogen.network.region
\spxentrymodule, 34		\spxentrymodule, 25
\spxentrysrc.common.config_setup		\spxentrysrc.models.hydrogen.network.registry
\spxentrymodule, 35		\spxentrymodule, 26
\spxentrysrc.common.model		\spxentrysrc.models.hydrogen.network.transportation_arc
\spxentrymodule, 37		\spxentrymodule, 27
\spxentrysrc.common.utilities		\spxentrysrc.models.hydrogen.utilities
\spxentrymodule, 44		\spxentrymodule, 28
\spxentrysrc.integrator		\spxentrysrc.models.hydrogen.utilities.h2_functions
\spxentrymodule, 46		\spxentrymodule, 28
\spxentrysrc.integrator.gaussseidel		\spxentrysrc.models.residential
\spxentrymodule, 46		\spxentrymodule, 30
\spxentrysrc.integrator.progress_plot		\spxentrysrc.models.residential.preprocessor
\spxentrymodule, 47		\spxentrymodule, 30
\spxentrysrc.integrator.runner		\spxentrysrc.models.residential.preprocessor.generate_input

```
\spxentrymodule, 30
                                                          \spxentryupdate_parent()\spxextrasrc.models.hydrogen.network.region.Reg
\spxentrysrc.models.residential.scripts
                                                                    method, 26
     \spxentrymodule, 31
                                                          \spxentryview_output_load()\spxextrasrc.models.residential.scripts.resident
\spxentrysrc.models.residential.scripts.residential
                                                                    method, 33
     \spxentrymodule, 31
                                                           \spxentryview_sensitivity()\spxextrasrc.models.residential.scripts.residentia
\spxentrysrc.sensitivity
                                                                    method, 34
     \spxentrymodule, 54
                                                           \spxentryvisualize()\spxextrasrc.models.hydrogen.network.grid.Grid
\spxentrysrc.sensitivity.babymodel
                                                                     method, 23
     \spxentrymodule, 55
\spxentrysrc.sensitivity.faster_sensitivity
                                                          \spxentrywrite_data()\spxextrasrc.models.hydrogen.network.grid.Grid
     \spxentrymodule, 81
                                                                    method, 23
\spxentrysrc.sensitivity_tools
     \spxentrymodule, 85
                                                          \spxentryyear\spxextrasrc.integrator.utilities.EI attribute,
\spxentrysrc.sensitivity.speed_test
     \spxentrymodule, 113
                                                           \spxentryyear\spxextrasrc.integrator.utilities.HI attribute,
\spxentrystep_sub_sc_subset()\spxextrain
                                                 module
         src.models.electricity.scripts.preprocessor,
\spxentrysubset_dfs()\spxextrain
                                                 module
         src.models.electricity.scripts.preprocessor,
\spxentrysubstitute_values()\spxextrasrc.sensitivity.sensitivity_tools.AutoSympy
          method, 111
\spxentrysubstitute values()\spxextrasrc.sensitivity.sensitivity tools.SensitivityMatrix
         method, 113
\spxentrytest()\spxextrasrc.models.hydrogen.network.grid.Grid
          method, 23
\spxentryTestBabyModel\spxextraclass
                                                      in
          src.sensitivity.babymodel, 79
\spxentrytime_map()\spxextrain
                                                 module
         src.models.electricity.scripts.preprocessor,
\spxentrytoy_model\spxextraclass
                                                      in
         src.sensitivity.faster_sensitivity, 85
\spxentryTransportationArc\spxextraclass
                                                      in
          src.models.hydrogen.network.transportation_arc,
\spxentryunpack_set_arguments()\spxextrasrc.common.model.Model
          method, 44
\spxentryupdate_cost()\spxextrain
                                                 module
         src.models.electricity.scripts.runner, 13
\spxentryupdate_data()\spxextrasrc.models.hydrogen.network.region.Region
         method, 26
\spxentryupdate_elec_demand()\spxextrain
                                                 module
         src.integrator.utilities, 54
\spxentryupdate_exchange_params()\spxextrasrc.models.hydrogen.model.h2_model.H2Model
         method, 19
\spxentryupdate_h2_prices()\spxextrain
                                                 module
         src.integrator.utilities, 54
\spxentryupdate_levels()\spxextrasrc.models.hydrogen.network.registry.Registry
         method, 27
\spxentryupdate_load()\spxextrasrc.models.residential.scripts.residential.residentialModule
         method, 33
```