SKA SDP Parametric Model Documentation

Release CDR

Rosie Bolton, Francois Malan, Bojan Nikolic, Andreas Wicenec, P

Sep 06, 2021

Contents

The SDP parametric model is a collection of equations and software code that model the performance aspects of the SDP. We created an interactive software interface for performing computations and displaying their results so that people can easily interact and experiment with the model. This is accomplished using Python and Jupyter notebooks, which are documented below.

A central goal of the notebooks is to compute performance metrics and general sizing estimates for the SDP. Refer to the parametric model document for the fundamental choices of which algorithms are modelled. The basic assumption is that we employ a combination of w-snapshots and w-stacking combined with facetting for distribution.

CHAPTER 1

Documentation

1.1 Notebooks

Notebooks are the main user interface of the SKA SDP parametric model. To allow exploration of the design space, the parametric model provides a number of separate notebooks for different purposes.

1.1.1 Build Instructions

If you wish to use the notebooks interactively, clone the parametric model repository and run the notebooks yourself:

```
$ git clone https://github.com/ska-telescope/sdp-par-model
$ cd sdp-par-model
$ pipenv install
$ pipenv shell
$ jupyter notebook notebooks
```

This should allow you to interact with the notebooks using a web browser.

1.1.2 List of Notebooks

The following notebooks come with the parametric model. Follow the links for non-interactive version of the notebooks:

SKA1_Imaging_Performance_Model: Contains interactive cells for computing telescope parameters (data rates, FLOPs, image sizes etc).

SKA1_Sensitivity_Analysis: Investigates the sensitivity of telescope parameters on inputs such as baseline length and ionospheric time scale.

SKA1_SDP_Performance_Dashboard: Overview of FLOP rate predictions for different telescopes, pipelines and high-priority science objectives (HPSOs).

SKA1_Export: Allows exporting detailed parametric model output for usage with external programs. This notebook can also be used to compare existing parametric model exports to allow fine-grained change tracking of calculated parameters.

SKA1_System_Sizing: Uses the HPSOs to estimate the average computational requirements and output data rates of the SDP instances at the two telescopes. This informs the system sizing.

SKA1_SDP_Products, SKA1_Document_Formulas: Shows the formulas used by the parametric model for calculating parameters. Useful for checking what exactly the parametric model calculates.

Absolute_Baseline_length_distribution: Compute and visualize baseline distributions of SKA telescopes. This notebook is used for determining the baseline bins that go into the parametric model calculations.

1.2 Implementation

Most of the parametric model code is maintained as Python package, which is imported by the notebooks. Two important concepts are those of a *config.PipelineConfig*, which represents a pipeline configuration to query properties of, and *parameters.container.ParameterContainer*, which is used to accumulate information we have about such a telescope configuration.

Below follows the general structure of the model's code base:

1.2.1 config

class sdp_par_model.config.**PipelineConfig**(*telescope=None*, *pipeline=None*, *band=None*, *hpso=None*, *hpso_pipe=None*, *adjusts={}*, ***kwargs*)

A full SDP pipeline configuration. This collects all data required to parameterise a pipeline.

calc_tel_params(*verbose=False*, *adjusts={}*, *symbolify=''*, *optimize_expression='Rflop'*, *clear_symbolised=None*)

Calculates telescope parameters for this configuration. Some values may (optionally) be overwritten, e.g. the maximum baseline or number of frequency channels.

Parameters

- **cfg** – Valid pipeline configuration
- **verbose** – How chatty we are supposed to be
- **adjusts** – Dictionary of telescope parameters to adjust
- **symbolify** – Generate symbolified telescope parameters
- **optimize_expression** – Set free symbols in a way that minimises given telescope parameter (only if symbolify is not set)
- **clear_symbolised** – Whether to clear parameters with free symbols after optimisation. (only if symbolify is not set. Default on if optimize_expression is not None.)

describe()

Returns a name that identifies this configuration.

eval_expression_products(*expression='Rflop'*, *verbose=False*)

Evaluate a parameter sum for each product

Parameters

- **pipelineConfig** – Pipeline configuration to use

- **expression** – Procuct parameter to evaluate
- **verbose** – Verbosity to use for *calc_tel_params*

eval_param_sweep_1d(*expression_string='Rflop'*, *parameter_string='Rccf'*, *param_val_min=10*, *param_val_max=10*, *number_steps=1*, *verbose=False*)

Evaluates an expression for a range of different parameter values, by varying the parameter linearly in a specified range in a number of steps

Parameters

- **pipelineConfig** –
- **expression_string** – The expression that needs to be evaluated, as string (e.g. "Rflop")
- **parameter_string** – the parameter that will be swept - written as text (e.g. "Bmax")
- **param_val_min** – minimum value for the parameter's value sweep
- **param_val_max** – maximum value for the parameter's value sweep
- **number_steps** – the number of *intervals* that will be used to sweep the parameter from min to max
- **verbose** –

Returns Pair of parameter values and results

Raises **AssertionError** –

eval_param_sweep_2d(*expression_string='Rflop'*, *parameters=None*, *params_ranges=None*, *number_steps=2*, *verbose=False*)

Evaluates an expression for a 2D grid of different values for two parameters, by varying each parameter linearly in a specified range in a number of steps. Similar to *eval_param_sweep_1d()*, except that it sweeps a 2D parameter space, returning a matrix of values.

Parameters

- **pipelineConfig** –
- **expression_string** – The expression that needs to be evalued, as string (e.g. "Rflop")
- **parameters** – The two parameters to sweep
- **params_ranges** – Ranges to use for parameters
- **number_steps** – The number of *intervals* that will be used to sweep the parameters from min to max
- **verbose** –

Returns Triple of parameter values (both) and results

eval_products_symbolic(*expression='Rflop'*, *symbolify='product'*)

Returns formulas for the given product property.

Parameters

- **pipelineConfig** – Pipeline configuration to use.
- **expression** – Product property to query. FLOP rate by default.
- **symbolify** – How aggressively sub-formulas should be replaced by symbols.

eval_symbols(*symbols*, *recursive=False*, *symbolify=''*, *optimize_expression=None*)
Returns formulas for the given symbol names. This can be used to look up the definitions behind sympy Symbols returned by eval_products_symbolic or this function.

The returned dictionary will contain an entry for all symbols that we could look up sucessfully - this excludes symbols that are not defined or have only a tautological definition ("sym = sym").

Parameters

- **pipelineConfig** – Pipeline configuration to use.
- **symbols** – Symbols to query
- **recursive** – Look up free symbols in symbol definitions?
- **symbolify** – How aggressively sub-formulas should be replaced by symbols.

is_valid(*pure_pipelines=True*)
Checks integrity of the pipeline configuration.

Returns (okay?, list of errors/warnings)

telescope_and_band_are_compatible()
Checks whether the supplied telescope and band are compatible with each other.

1.2.2 evaluate

This file contains methods for programmatically evaluating the SKA SDP parametric model.

sdp_par_model.evaluate.**cheap_lambdify_curry**(*free_vars*, *expression*)
Translate sympy expression to an actual Python expression that can be evaluated quickly. This is roughly the same as sympy's lambdify, with a number of differences:

1. We only support a subset of functions. Note that sympy's list is incomplete as well, and actually has a wrong translation rule for "Max".
2. The return is curried, so for multiple "free_vars" (x, y) you will have to call the result as "f(x)(y)" instead of "f(x,y)". This means we can easily obtain a function that is specialised for a certain value of the outer variable.

sdp_par_model.evaluate.**collect_free_symbols**(*formulas*)
Returns the names of all free symbol in the given formulas. We always count all functions as free.

Parameters **formulas** – Formulas to search for free symbols.

sdp_par_model.evaluate.**evaluate_expression**(*expression*, *tp*)
Evaluate an expression by substituting the telescopec parameters into them. Depending on the type of expression, the result might be a value, a list of values (in case it is baseline-dependent) or a string (if evaluation failed).

Parameters

- **expression** – the expression, expressed as a function of the telescope parameters, Tsnap and Nfacet
- **tp** – the telescope parameters (ParameterContainer object containing all relevant parameters)

Returns

sdp_par_model.evaluate.**evaluate_expressions**(*expressions*, *tp*)
Evaluate a sequence of expressions by substituting the telescope_parameters into them.

Parameters

- **expressions** – An array of expressions to be evaluated
- **tp** – The set of telescope parameters that should be used to evaluate each expression

sdp_par_model.evaluate.**is_literal**(*expression*)
Returns true iff the expression is already a literal (e.g. float or integer) value that cannot be substituted or evaluated further. Used to halt attempts at further evaluating symbolic expressions

sdp_par_model.evaluate.**minimise_parameters**(*telescope_parameters, expression_string='Rflop', expression=None, lower_bound={}, upper_bound={}, only_one_minimum=['Nfacet'], verbose=False*)
Computes the optimal value for free variables in telescope parameters such that it minimizes the value of an expression (typically Rflop). Returns result as a dictionary.

Parameters

- **telescope_parameters** – Contains the definition of the expression that needs to be minimzed. This should be a symbolic expression that involves Deltaw_Stack and/or Nfacet.
- **expression_string** – The expression that should be minimized. This is typically assumed to be the computational load, but may also be, for example, buffer size.
- **expression** – TODO define
- **lower_bound** – Lower bound to use for symbols
- **upper_bound** – Upper bound to use for symbols
- **only_one_minimum** – Assume that the given (integer) symbol only has one minimum, so we can assume that any local optimum we find is the global optimum.
- **verbose** – Be more chatty about what's going on.

sdp_par_model.evaluate.**optimize_lambdified_expr**(*lam*, *bound_lower*, *bound_upper*)

1.2.3 reports

This file contains methods for generating reports for SKA SDP parametric model data using especially matplotlib and Jupyter. Having these functions separate allows us to keep notebooks free of clutter.

sdp_par_model.reports.**adjusts**()
Create widget for adjustments (with some suggestions)

sdp_par_model.reports.**check_pipeline_config**(*cfg*, *pure_pipelines*)
Check pipeline configuration, displaying a message in the Notebook for every problem found. Returns whether the configuration is usable at all.

sdp_par_model.reports.**compare_csv**(*result_file*, *ref_file*, *ignore_modifiers=True*, *ignore_units=True*, *row_threshold=0.01*, *export_html=''*, *return_diffs=False*)
Read and compare two CSV files with telescope parameters

Parameters

- **result_file** – CSV file with telescope parameters
- **ref_file** – CSV file with reference parameters
- **ignore_modifiers** – Ignore modifiers when matching columns (say, [blcoal])
- **ignore_units** – Ignore units when matching rows (say, [s])

- **export_html** – File to write table to. If empty, will be shown inline.
- **return_diffs** – Whether to also return differences as a dictionary

Returns Sum of differences found in specified rows, if requested

sdp_par_model.reports.**compare_telescopes_default**(*telescope_1*, *band_1*, *pipeline_1*, *adjusts_1*, *telescope_2*, *band_2*, *pipeline_2*, *adjusts_2*, *verbosity='Overview'*)

Evaluates two telescopes, both operating in a given band and pipeline, using their default parameters. A bit of an ugly bit of code, because it contains both computations and display code. But it does make for pretty interactive results. Plots the results side by side.

Parameters

- **telescope_1** –
- **telescope_2** –
- **band_1** –
- **band_2** –
- **pipeline_1** –
- **pipeline_2** –
- **adjusts_1** – Configuration adjustments for telescope 1. See PipelineConfig.
- **adjusts_2** – Configuration adjustments for telescope 2. See PipelineConfig.
- **verbosity** – amount of output to generate

sdp_par_model.reports.**default_rflop_plotting_colours**(*rows*)

Defines a default colour order used in plotting Rflop components

Returns List of HTML colour codes as string

sdp_par_model.reports.**evaluate_hpso_optimized**(*hpso*, *hpso_pipe*, *adjusts=''*, *verbosity='Overview'*)

Evaluates a High Priority Science Objective by optimizing NFacet and Tsnap to minimize the total FLOP rate

Parameters

- **hpso** – HPSO pipeline to evaluate
- **verbosity** –

sdp_par_model.reports.**evaluate_telescope_optimized**(*telescope*, *band*, *pipeline*, *max_baseline='default'*, *Nf_max='default'*, *blcoal=True*, *on_the_fly=False*, *scale_predict_by_facet=True*, *verbosity='Overview'*)

Evaluates a telescope with manually supplied parameters, but then automatically optimizes NFacet and Tsnap to minimize the total FLOP rate for the supplied parameters

Parameters

- **telescope** –
- **band** –
- **pipeline** –
- **max_baseline** –

- **Nf_max** –
- **blcoal** – Baseline dependent coalescing (before gridding)
- **on_the_fly** –
- **verbosity** –

sdp_par_model.reports.**find_csvs**(*csv_path='../data/csv'*)
Returns a map of all CSV files currently checked into the Git repository.

Returns A dictionary of (rev, hpsos/pipelines) pairs to file names

sdp_par_model.reports.**format_result**(*value*)
Format a result value for viewing. As we expect that most numbers should be in a "nice" range this means we truncate number accuracy by default.

sdp_par_model.reports.**format_result_cell**(*val*, *color='black'*, *colspan=1*, *typ='td'*)
Format a result value for inclusing in a table.

sdp_par_model.reports.**format_result_cells**(*val*, *color='black'*, *max_cols=1*)
Format a result value for inclusing in a table. If the value is a list, we genrate multiple cells up to "max_cells".

sdp_par_model.reports.**get_result_expressions**(*resultMap*, *tp*)
Returns the expression that needs to be evaluated

Parameters

- **resultMap** –
- **tp** –

Returns

sdp_par_model.reports.**get_result_sum**(*resultMap*)
Returns the corresponding entries of whether expressions should be summed or concatenated in a list.

Parameters **resultMap** –

Returns

sdp_par_model.reports.**lookup_csv**(*results*, *column_name*, *row_name*, *ignore_units=True*, *ignore_modifiers=True*)
Lookup a value in a CSV table

Parameters

- **results** – CSV table as returned by read_csv
- **column_name** – Column (pipeline/HPSO name) to look up
- **row_name** – Row (parameter name) to look up
- **ignore_units** – Ignore units when matching rows (say, [s])
- **ignore_modifiers** – Ignore modifiers when matching columns (say, [blcoal])

Returns Value if found, None otherwise

sdp_par_model.reports.**make_band_toggles**()
Create connected telescope/band toggle widgets that only allow selection of valid combinations

sdp_par_model.reports.**make_hpso_pipeline_toggles**()
Create connected HPSO/pipeline toggle widgets that only allow selection of valid combinations

sdp_par_model.reports.**mk_result_map_rows**(*verbosity='Overview'*)
Collects result map information for a given row set

Parameters **verbosity** – Row set to show. If None, we will use default rows.

Returns A tuple of the result map, the sorted list of the row names and a list of the row units.

sdp_par_model.reports.**newest_csv**(*csv_map*, *typ='hpsos'*, *rev='HEAD'*, *ignore_rev=False*)
Finds the CSV closest to the given revision according to the Git history

Parameters

- **csv_map** – CSV map, see find_csvs
- **typ** – Type of CSV to look for (hpsos/pipelines)
- **rev** – Git revision to start from
- **ignore_rev** – Don't return "rev" itself, begin search with parents

Returns CSV file name if found, None otherwise

sdp_par_model.reports.**plot_2D_surface**(*title*, *x_values*, *y_values*, *z_values*, *contours=None*, *xlabel=None*, *ylabel=None*, *zlabel=None*, *nlevels=15*)
Plots a series of (x,y) values using a line and data-point visualization.

Parameters

- **title** – The plot's title
- **x_values** – a 1D numpy array
- **y_values** – a 1D numpy array
- **z_values** – a 2D numpy array, indexed as (x,y)
- **contours** – optional array of values at which contours should be drawn
- **zlabel** –
- **ylabel** –
- **xlabel** –

Returns

sdp_par_model.reports.**plot_3D_surface**(*title*, *x_values*, *y_values*, *z_values*, *contours=None*, *xlabel=None*, *ylabel=None*, *zlabel=None*, *nlevels=15*)
Plots a series of (x,y) values using a line and data-point visualization.

Parameters

- **title** – The plot's title
- **x_values** – a 1D numpy array
- **y_values** – a 1D numpy array
- **z_values** – a 2D numpy array, indexed as (x,y)
- **contours** – optional array of values at which contours should be drawn
- **zlabel** –
- **ylabel** –
- **xlabel** –

Returns

sdp_par_model.reports.**plot_line_datapoints**(*title*, *x_values*, *y_values*, *xlabel=None*, *ylabel=None*)

Plots a series of (x,y) values using a line and data-point visualization.

Parameters

- **title** –
- **x_values** –
- **y_values** –

Returns

sdp_par_model.reports.**plot_pie**(*title*, *labels*, *values*, *colours=None*)

Plots a pie chart

Parameters

- **title** –
- **labels** –
- **values** – a numpy array
- **colours** –

sdp_par_model.reports.**plot_stacked_bars**(*title*, *labels*, *value_labels*, *dictionary_of_value_arrays*, *colours=None*, *width=0.35*, *save=None*, *xticks_rot='horizontal'*)

Plots a stacked bar chart, with any number of columns and components per stack (must be equal for all bars)

Parameters

- **title** –
- **labels** – The label belonging to each bar
- **dictionary_of_value_arrays** – A dictionary that maps each label to an array of values (to be stacked).
- **colours** –

Returns

sdp_par_model.reports.**read_csv**(*filename*)

Reads pipeline calculation results from a CSV file as written by _write_csv.

sdp_par_model.reports.**save_pie**(*title*, *labels*, *values*, *filename*, *colours=None*)

Works exactly same way as plot_pie(), but instead of plotting, saves a pie chart to SVG output file. Useful for exporting results to documents and such

Parameters

- **title** –
- **labels** –
- **values** – a numpy array
- **filename** –
- **colours** –

sdp_par_model.reports.**show_table**(*title*, *labels*, *values*, *units*, *docs=None*)

Plots a table of label-value pairs

Parameters

- **title** – string
- **labels** – string list / tuple
- **values** – string list / tuple
- **units** – string list / tuple
- **docs** – Optional documentation per row

Returns

sdp_par_model.reports.**show_table_compare**(*title*, *labels*, *values_1*, *values_2*, *units*)
Plots a table that for a set of labels, compares each' value with the other

Parameters

- **title** –
- **labels** –
- **values_1** –
- **values_2** –
- **units** –

Returns

sdp_par_model.reports.**stack_bars_hpsos**(*title*, *hpsos*, *adjusts={}*, *parallel=0*, *save=None*)
Evaluates all valid configurations of this telescope and dumps the result as a CSV file.

sdp_par_model.reports.**stack_bars_pipelines**(*title*, *telescopes*, *bands*, *pipelines*, *adjusts={}*, *parallel=0*, *save=None*)
Evaluates all valid configurations of this telescope and shows results as stacked bars.

sdp_par_model.reports.**strip_csv**(*csv*, *ignore_units=True*, *ignore_modifiers=True*)

sdp_par_model.reports.**toggles**(*opts*, **args*, ***kwargs*)
Helper for creating toggle buttons from given options

sdp_par_model.reports.**write_csv_hpsos**(*filename*, *hpsos*, *adjusts=''*, *verbose=False*, *parallel=0*)
Evaluates all valid configurations of this telescope and dumps the result as a CSV file.

sdp_par_model.reports.**write_csv_pipelines**(*filename*, *telescopes*, *bands*, *pipelines*, *adjusts=''*, *verbose=False*, *parallel=0*)
Evaluates all valid configurations of this telescope and dumps the result as a CSV file.

As the heart of the parametric model, these modules generate parametric equations from parameter definitions. The whole process is very open and flexible: Virtually all methods work by updating a common *container.ParameterContainer* object step by step until all required equations have been applied.

Note that this is designed to work no matter whether the parameters are given symbolically (as sympy expressions) or as plain values. Thus this code can be used to generate both values or formulae, with fine-grained control over terms at every calculation step.

The general workflow is that a fresh parameter container will get populated using data from *definitions* appropriate for the desired telescope configuration. Then *equations* will be used to derive parameter equations from them. The calculation can be "symbol-ified" at multiple stages, for example using `definitions.define_design_equations_variables()` or `equations.apply_imaging_equations()`.

1.2.4 parameters.container

Handles collections of telescope parameters. This module contains all the necessary plumbing to make the parameter definitions work.

ParameterContainer is centrally important and used throughout the model, but essentially is only a container class that is passed around between modules, and contains a set of parameters, values and variables that constitute the inputs and outputs of computations.

class sdp_par_model.parameters.container.**BLDep**(*pars*, *term*, *defaults={}*)

A baseline-dependent sympy expression.

Named baseline properties can be used as symbols in the sympy expression. Typical choices would be 'b' for the baseline length or 'bcount' for the baseline count.

Note that this mostly replicates functionality of numpy's own Lambda expression. The main difference here are that we assign semantics to the term and parameters (e.g. baseline properties). Furthermore, we also lift some arithmetic operations such that they can also work on baseline-dependent terms.

atoms(*typ*)

eval_sum(*bins*, *known_sums={}*)

Converts a possibly baseline-dependent terms (e.g. constructed using "BLDep" or "blsum") into a formula by summing over baselines.

Parameters

- **bins** – List of dictionaries with baseline properties. If it is a tuple with layout

  (symbol, lower limit, upper limit, terms)

  We are going to generate a symbolic sum where the symbol runs from the lower to the upper limit.
- **known_sums** – List of terms that we know the sum of

Returns Sum term

free_symbols

subs(**args*, ***kwargs*)

class sdp_par_model.parameters.container.**ParameterContainer**

Stores calculated telescope parameters.

All fields set on objects are either inputs or outputs of telescope parameter calculations. We expect all fields to have one of the following types:

- Simple value types such as integer, float, string or list. These are assumed to be constants or calculated values.
- Sympy expressions for not entirely evaluated values. Appear if parameters were left open, such as if the number of facets was not decided yet, or we are evaluating the model symbolically.
- Baseline-dependent expressions (see *BLDep*). Expressions that have a different value depending on the considered baseline.

clear_symbolised()

Remove any parameters with free variables.

This is useful if the telescope parameters were optimised for something, yet some parameters did not factor into the optimisation and were therefore left as symbols. Those parameter values are therefore undefined, so discarding them is the right thing to do.

get(*param_name*, *default=None*, *warn=True*)
Provides a method for reading a parameter by string.

Parameters

- **param_name** – The name of the parameter/field that needs to be read - provided as text. If the parameter contains a ".", it is interpreted as a product property.
- **default** – Default value to return if the parameter or product does not exist
- **warn** – Output a warning if parameter does not exist

Returns The parameter value.

get_products(*expression='Rflop'*, *scale=1*)
TODO:What does this method do exactly? Why does it default to Rflop?

make_symbol_name(*name*)
Make names used in our code into something more suitable to be used as a Latex symbol. This is a quick-n-dirty heuristic based on what the names used in equations.py tend to look like.

set_param(*param_name*, *value*, *prevent_overwrite=True*, *require_overwrite=False*)
Provides a method for setting a parameter. By default first checks that the value has not already been defined. Useful for preventing situations where values may inadvertently be overwritten.

Parameters

- **param_name** – The name of the parameter/field that needs to be assigned - provided as text
- **value** – the value to be written (as actual data type, i.e. not necessarily text)
- **prevent_overwrite** – Disallows this value to be overwritten once defined. Default = True.
- **require_overwrite** – Only allows value to be changed if it already exists. Default = False.

set_product(*product*, *T=None*, *N=1*, *bins=None*, ***args*)
Sets product properties using a task abstraction. Each property is expressed as a sum over baselines.

Parameters

- **product** – Product to set.
- **T** – Observation time covered by this task. Default is the entire observation (Tobs). Can be baseline-dependent.
- **N** – Task parallelism / rate multiplier. The number of tasks that work on the data in parallel. Can be baseline-dependent.
- **bmax_bins** – Maximum lengths of baseline bins to use
- **bcount_bins** – Size of baseline bins to use
- **args** – Task properties as rates. Will be multiplied by N. If it is baseline-dependent, it will be summed over all baselines to yield the final rate.

subs(*substs*)

symbolify()
Replace all parameters so far with symbols, so equations composed after this point are symbolic with respect to earlier results.

sdp_par_model.parameters.container.**blsum**(*b*, *expr*)
A baseline sum of an expression

Implemented as a weighted sum over baseline bins. Returns a BLDep object of the expression multiplied with the bin baseline count.

The baseline count parameter defaults to 1, so the following works as expected:

expr = blsum(b, . . .) expr2 = blsum(b, expr(b) * . . .)

sdp_par_model.parameters.container.**is_expr**(*e*)

1.2.5 parameters.definitions

Enumerates and defines the parameters of the telescopes, bands, pipelines, etc. Several methods are supplied by which values can be found by lookup as well (e.g. finding the telescope that is associated with a given mode)

Parameters defined here include telecope parameters as well as physical constants. Generally, the output of the functions are ParameterContainer objects (usually referred to by the variable o in the methods below) that has parameters as fields.

class sdp_par_model.parameters.definitions.**Bands**
Enumerate all possible bands (used in *apply_band_parameters()*)

Low = 'Low'

Mid1 = 'Mid1'

Mid2 = 'Mid2'

Mid5a = 'Mid5a'

Mid5b = 'Mid5b'

available_bands = ['Low', 'Mid1', 'Mid2', 'Mid5a', 'Mid5b']

telescope_bands = {'SKA1_Low': ['Low'], 'SKA1_Mid': ['Mid1', 'Mid2', 'Mid5a', 'Mid5b

class sdp_par_model.parameters.definitions.**Constants**
A new class that takes over the roles of sympy.physics.units and astropy.const, because it is simpler this way.

arcminute = 0.0002908882086657216

arcsecond = 4.84813681109536e-06

degree = 0.017453292519943295

giga = 1000000000

kilo = 1000

mega = 1000000

peta = 1000000000000000

tera = 1000000000000

class sdp_par_model.parameters.definitions.**HPSOs**
Enumerate the pipelines of each HPSO (used in *apply_hpso_parameters()*)

all_hpsos = {'hpso01', 'hpso02a', 'hpso02b', 'hpso04a', 'hpso04b', 'hpso04c', 'hpso05a

all_maxcases = {'max_low', 'max_mid_band1', 'max_mid_band2', 'max_mid_band5a_1', 'max_

available_hpsos = {'hpso01', 'hpso02a', 'hpso02b', 'hpso04a', 'hpso04b', 'hpso04c', 'hp

hpso01 = 'hpso01'

```
    hpso02a = 'hpso02a'
    hpso02b = 'hpso02b'
    hpso04a = 'hpso04a'
    hpso04b = 'hpso04b'
    hpso04c = 'hpso04c'
    hpso05a = 'hpso05a'
    hpso05b = 'hpso05b'
    hpso13 = 'hpso13'
    hpso14 = 'hpso14'
    hpso15 = 'hpso15'
    hpso18 = 'hpso18'
    hpso22 = 'hpso22'
    hpso27and33 = 'hpso27and33'
    hpso32 = 'hpso32'
    hpso37a = 'hpso37a'
    hpso37b = 'hpso37b'
    hpso37c = 'hpso37c'
    hpso38a = 'hpso38a'
    hpso38b = 'hpso38b'
    hpso_pipelines = {'hpso01':  ('Ingest', 'RCAL', 'FastImg', 'ICAL', 'DPrepA', 'DPrepB',
    hpso_telescopes = {'hpso01':  'SKA1_Low', 'hpso02a':  'SKA1_Low', 'hpso02b':  'SKA1_Lo
    max_low = 'max_low'
    max_mid_band1 = 'max_mid_band1'
    max_mid_band2 = 'max_mid_band2'
    max_mid_band5a_1 = 'max_mid_band5a_1'
    max_mid_band5a_2 = 'max_mid_band5a_2'
    max_mid_band5b_1 = 'max_mid_band5b_1'
    max_mid_band5b_2 = 'max_mid_band5b_2'
```

class sdp_par_model.parameters.definitions.**Pipelines**

Enumerate the SDP pipelines. These must map onto the Products. The HPSOs invoke these.

```
    DPrepA = 'DPrepA'
    DPrepA_Image = 'DPrepA_Image'
    DPrepB = 'DPrepB'
    DPrepC = 'DPrepC'
    DPrepD = 'DPrepD'
    FastImg = 'FastImg'
```

```
    ICAL = 'ICAL'
    Ingest = 'Ingest'
    PSS = 'PSS'
    PST = 'PST'
    RCAL = 'RCAL'
    SinglePulse = 'SinglePulse'
    all = ['Ingest', 'RCAL', 'FastImg', 'ICAL', 'DPrepA', 'DPrepA_Image', 'DPrepB', 'DPrep
    available_pipelines = ['Ingest', 'RCAL', 'FastImg', 'ICAL', 'DPrepA', 'DPrepA_Image',
    imaging = ['RCAL', 'FastImg', 'ICAL', 'DPrepA', 'DPrepA_Image', 'DPrepB', 'DPrepC']
    input = ['Ingest']
    nonimaging = ['PSS', 'PST', 'SinglePulse']
    output = ['FastImg', 'DPrepA', 'DPrepA_Image', 'DPrepB', 'DPrepC', 'PSS', 'PST', 'Sing
    pure_pipelines = ['Ingest', 'RCAL', 'FastImg', 'ICAL', 'DPrepA', 'DPrepA_Image', 'DPrep
    realtime = ['Ingest', 'RCAL', 'FastImg']
```

class sdp_par_model.parameters.definitions.**Products**

Enumerate the SDP Products used in pipelines

```
    Alert = 'Alert'
    Average = 'Average'
    Calibration_Source_Finding = 'Calibration Source Finding'
    Correct = 'Correct'
    DFT = 'DFT'
    Degrid = 'Degrid'
    Degridding_Kernel_Update = 'Degridding Kernel Update'
    Demix = 'Demix'
    Extract_LSM = 'Extract_LSM'
    FFT = 'FFT'
    Flag = 'Flag'
    Grid = 'Grid'
    Gridding_Kernel_Update = 'Gridding Kernel Update'
    IFFT = 'IFFT'
    Identify_Component = 'Identify Component'
    Image_Spectral_Averaging = 'Image Spectral Averaging'
    Image_Spectral_Fitting = 'Image Spectral Fitting'
    Notify_GSM = 'Update GSM'
    PhaseRotation = 'Phase Rotation'
    PhaseRotationPredict = 'Phase Rotation Predict'
```

QA = 'QA'

Receive = 'Receive'

Reprojection = 'Reprojection'

ReprojectionPredict = 'Reprojection Predict'

Select = 'Select'

Solve = 'Solve'

Source_Find = 'Source Find'

Subtract_Image_Component = 'Subtract Image Component'

Subtract_Visibility = 'Subtract Visibility'

Update_LSM = 'Update LSM'

Visibility_Weighting = 'Visibility Weighting'

class sdp_par_model.parameters.definitions.**Telescopes**
Enumerate the possible telescopes to choose from (used in *apply_telescope_parameters()*)

SKA1_Low = 'SKA1_Low'

SKA1_Mid = 'SKA1_Mid'

available_teles = ['SKA1_Low', 'SKA1_Mid']

sdp_par_model.parameters.definitions.**apply_band_parameters**(*o*, *band*)
Applies the parameters that apply to the band to the parameter container object o

Parameters

- **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)
- **band** –

Returns ParameterContainer

sdp_par_model.parameters.definitions.**apply_global_parameters**(*o*)
Applies the global parameters to the parameter container object o.

Parameters **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)

Returns ParameterContainer

sdp_par_model.parameters.definitions.**apply_hpso_parameters**(*o*, *hpso*, *hpso_pipe*)
Applies the parameters for the HPSO pipeline to the parameter container object o.

Parameters

- **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)
- **hpso** – The HPSO whose parameters we are applying
- **hpso_pipe** – The pipeline whose parameters we are applying

Returns ParameterContainer

sdp_par_model.parameters.definitions.**apply_pipeline_parameters**(*o*, *pipeline*)
Applies the parameters that apply to the pipeline to the parameter container object o

Parameters

- **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)
- **pipeline** – Type of pipeline

Raises **Exception** –

Returns ParameterContainer

sdp_par_model.parameters.definitions.**apply_telescope_parameters**(*o*, *telescope*)
Applies the parameters that apply to the supplied telescope to the parameter container object o

Parameters

- **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)
- **telescope** –

Returns ParameterContainer

sdp_par_model.parameters.definitions.**define_design_equations_variables**(*o*)
This method defines the *symbolic* variables that we will use during computations and that may need to be kept symbolic during evaluation. One reason to do this would be to allow the output formula to be optimized by varying these variables

Parameters **o** – A supplied ParameterContainer object, to which the symbolic variables are appended (in-place)

Returns ParameterContainer

sdp_par_model.parameters.definitions.**define_pipeline_products**(*o*, *pipeline*, *named_pipeline_products=[]*)

sdp_par_model.parameters.definitions.**define_symbolic_variables**(*o*)
This method defines the *symbolic* variables that we will use during computations and that need to be kept symbolic during evaluation of formulae. One reason to do this would be to allow the output formula to be optimized by varying this variable (such as with Tsnap and Nfacet)

Parameters **o** – The supplied ParameterContainer object, to which the symbolic variables are appended (in-place)

Returns ParameterContainer

1.2.6 parameters.equations

This module contains the actual equations that are used to compute the telescopes' performance values and computional requirements from the supplied basic parameters defined in ParameterDefinitions.

sdp_par_model.parameters.equations.**_apply_calibration_equations**(*o*)
Self-calibration using predicted visibilities

References: SKA-TEL-SDP-0000040 01D section 3.6.5 - Solve

sdp_par_model.parameters.equations.**_apply_channel_equations**(*o*, *symbolify*)
Determines the number of frequency channels to use in backward & predict steps.

References:

- SKA-TEL-SDP-0000040 01D section B - Covering the Frequency Axis
- SKA-TEL-SDP-0000040 01D section D - Visibility Averaging and Coalescing

sdp_par_model.parameters.equations.**_apply_coalesce_equations**(*o*, *symbolify*)
Determines amount of coalescing of visibilities in time.

References:

- SKA-TEL-SDP-0000040 01D section A - Covering the Time Axis
- SKA-TEL-SDP-0000040 01D section D - Visibility Averaging and Coalescing

sdp_par_model.parameters.equations.**_apply_common_equations**(*o*, *bins*, *binfracs*)
Calculate simple derived values that are going to get used fairly often.

sdp_par_model.parameters.equations.**_apply_correct_equations**(*o*)
Correction of gains

References: SKA-TEL-SDP-0000040 01D section 3.6.7 - Correct

sdp_par_model.parameters.equations.**_apply_dft_equations**(*o*)
Direct discrete fourier transform as predict alternative to Reproject+FFT+Degrid+Phase Rotation.

References: SKA-TEL-SDP-0000040 01D section 3.6.4 - Predict via Direct Fourier Transform

sdp_par_model.parameters.equations.**_apply_fft_equations**(*o*)
Discrete fourier transformation of grids to images (and back)

References: SKA-TEL-SDP-0000040 01D section 3.6.13 - FFT and iFFT

sdp_par_model.parameters.equations.**_apply_flag_equations**(*o*)
Flagging equations for non-ingest pipelines

sdp_par_model.parameters.equations.**_apply_flop_equations**(*o*)
Calculate overall flop rate

sdp_par_model.parameters.equations.**_apply_geometry_equations**(*o*, *symbolify*)
Telescope geometry in space and time, given curvature and rotation of the earth. This determines the maximum w-term that needs to be corrected for and hence the size of w-kernels.

References:

- SKA-TEL-SDP-0000040 01D section G - Convolution Kernel Sizes
- SKA-TEL-SDP-0000040 01D section H.1 - Imaging Pipeline Geometry Assumptions

sdp_par_model.parameters.equations.**_apply_grid_equations**(*o*)
Gridding and degridding of visibilities

References: SKA-TEL-SDP-0000040 01D section 3.6.11 - Grid and Degrid

sdp_par_model.parameters.equations.**_apply_image_equations**(*o*)
Calculate image parameters, such as resolution and size

References:

- SKA-TEL-SDP-0000040 01D section D - The Resolution and Extent of Images and uv Planes
- SKA-TEL-SDP-0000040 01D section H.2 - Faceting

sdp_par_model.parameters.equations.**_apply_ingest_equations**(*o*)
Ingest equations

References: SKA-TEL-SDP-0000040 01D section 3.3 - The Fast and Buffered pre-processing pipelines

sdp_par_model.parameters.equations.**_apply_io_equations**(*o*)
Compute the Buffer sizes

References: SKA-TEL-SDP-0000040 01D section H.3 - Convolution Kernel Cache Size

sdp_par_model.parameters.equations.**_apply_kernel_equations**(*o*)
Generate parameters for Convolution kernels

References:

- SKA-TEL-SDP-0000040 01D section 3.6.12 - Gridding Kernel Update
- SKA-TEL-SDP-0000040 01D section E - Re-use of Convolution Kernels

sdp_par_model.parameters.equations.**_apply_kernel_product_equations**(*o*)
Generate parameters for Convolution kernels

References:

- SKA-TEL-SDP-0000040 01D section E - Re-use of Convolution Kernels

sdp_par_model.parameters.equations.**_apply_major_cycle_equations**(*o*)
Subtract predicted visibilities from last major cycle

References: SKA-TEL-SDP-0000040 01D section 3.6.6 - Subtract

sdp_par_model.parameters.equations.**_apply_minor_cycle_equations**(*o*)
Minor Cycles implementing deconvolution / cleaning

References:

- SKA-TEL-SDP-0000040 01D section 3.6.16 - Subtract Image Component
- TCC-SDP-151123-2 - Recommendations

sdp_par_model.parameters.equations.**_apply_nonimaging_equations**(*o*)
Compute requirements for non-imaging pipelines.

sdp_par_model.parameters.equations.**_apply_phrot_equations**(*o*)
Phase rotation (for the faceting)

References: SKA-TEL-SDP-0000040 01D section 3.6.9 - Phase Rotation

sdp_par_model.parameters.equations.**_apply_reprojection_equations**(*o*)
Re-projection of skewed images as generated by w snapshots

References: SKA-TEL-SDP-0000040 01D section 3.6.14 - Reprojection

sdp_par_model.parameters.equations.**_apply_source_find_equations**(*o*)
Rough estimate of source finding flops.

References: SKA-TEL-SDP-0000040 01D section 3.6.17 - Source find

sdp_par_model.parameters.equations.**_apply_spectral_fitting_equations**(*o*)
Spectral fitting of the image for CASA-style MSMFS clean.

References: SKA-TEL-SDP-0000040 01D section 3.6.15 - Image Spectral Fitting

sdp_par_model.parameters.equations.**apply_imaging_equations**(*telescope_parameters*, *pipeline*, *bins*, *binfracs*, *verbose*, *symbolify=”*)
(Symbolically) computes a set of derived parameters using imaging equations described in PDR05 (version 1.85).

The derived parameters are added to the supplied *telescope_parameters* object (locally referred to as *o*). Where parameters are only described symbolically (using sympy) they can be numerically evaluated at a later stage, when unknown symbolic variables are suitably substituted.

Parameters

- **telescope_parameters** – *container.ParameterContainer* object containing the telescope parameters. Will be modified in-place by appending / overwriting the relevant fields
- **pipeline** – The pipeline
- **verbose** – displays verbose command-line output

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W