ADG - Automated Diagram Generator Documentation

Release 1.0.1

ADG Dev Team

Jul 03, 2020

Contents

CHAPTER 1

The ADG Project

1.1 Description

ADG is a tool generating diagrams and producing their expressions for given many-body formalisms. Diagrammatic rules from the formalism are combined with graph theory objects to produce diagrams and expressions in a fast, simple and error-safe way.

The only input consists in the theory and order of interest, and the N-body character of the operators of interest. The main output is a LaTeX file containing the diagrams, their associated expressions and additional informations that can be compiled by ADG id needed. Other computer-readable files may be produced as well.

1.2 Status

As for now, the code is capable of handling two different formalisms, i.e. Many-Body Perturbation Theory (MBPT) and Bogoliubov Many-Body Perturbation Theory (BMBPT).

- For MBPT, the code generates all Hartree-Fock energy diagrams at any given order along with their expression and additional information (conjugate diagram, excitation level...).
- For BMBPT, the code generates all diagrams for a generic observable commuting with the Hamiltonian, along with their time-dependent and time-integrated expressions.

1.3 Future developments

Extensions under discussions are diagrams and expressions for Particle-Number Projected BMBPT as well as diagrams and expressions generation for Gorkov Self-Consistent Green's Functions (GSCGF).

CHAPTER 2

Install ADG on your computer

2.1 Install

To install ADG, download the source files and run

```
pip2 install <project_folder>
```

or alternatively

```
python2 setup.py install
```

If you want to install ADG in `develop` mode, then run

```
pip2 install -e <project_folder>
```

2.2 Dependencies

In order to run the code, you will need a Python2 install >= 2.7.1 and the following Python libraries:

- networkx >= 2.0 and <2.3
- numpy < 1.17.0
- scipy < 1.3.0

If you want ADG to compile the LaTeX output file, you will need a Latex install with the PDFLaTeX compiler and the feynmp and feynmp-auto packages installed, which are standard packages in most recent distributions.

CHAPTER 3

Generate diagrams with ADG

3.1 Run ADG

To run the program and generate BMBPT diagrams at order 4 for example, use

```
adg -o 4 -t BMBPT -d -c
```

where the `-o` flag is for the order, `-t` for the type of theory, `-d` indicates you want the diagrams to be drawn and `-c` that you want ADG to compile the LaTeX output.

You can alternatively run the program in interactive mode by typing

```
adg -i
```

Finally, to obtain more information on all the available flags, use

```
adg -h
```

3.2 CLI options

3.2.1 Generic options:

-o, --order order of the diagrams [1-9]

-t, --theory theory of interest: MBPT or BMBPT

-i, --interactive execute ADG in interactive mode

3.2.2 BMBPT options:

-can, --canonical consider only canonical diagrams

-nobs, --nbody_observable maximal n-body character of the observable [1-3], default = 2

-3NF, --with_3NF use two and three-body forces for BMBPT diagrams

-dt, --draw_tsds draw Time-Structure Diagrams

3.2.3 MBPT option:

-cd, --cd_output produce computer-readable output for automated frameworks

3.2.4 Run management options:

-d, --draw_diags draw the diagrams using FeynMF

-c, --compile compile the LaTeX output file with PDFLaTeX

3.3 Output files

The output of the program is stored in a folder named after the theory, and a subfolder named after the order, e.g. `/MBPT/Order-4`. In the case of BMBPT, suffixes are added depending on the n-body forces of the observable, and if three-body forces were used or only canonical diagrams computed, i.e. for our previous example, results would be stored under `BMBPT/Order-4_2body_observable`.

The main output file of the program, called `result.tex`, is a LaTeX file containing the expressions of the diagrams along other basic infos on their structure, and, if flag `-d` has been used, drawing instructions. The file is automatically compiled and produces a PDF file `result.pdf` when using the `-c` file.

A list of the adjacency matrices associated with the diagrams is printed separately in the `adj_matrices.list` file to allow for an easy use with another many-body diagrams code.

In the case of a MBPT calculations, it is possible to produce output specifically tailored for automated calculations framework by using the `-cd` flag. The associated output files use `CD_` as a prefix.

CHAPTER 4

ADG Reference for Developers

4.1 Main script

4.2 Run & CLI management

4.3 Generic Diagram

Routines and class for all types of diagrams, inherited by others.

class adg.diag.**Diagram**(*nx_graph*)

Bases: object

Describes a diagram with its related properties.

graph

The actual graph.

Type NetworkX MultiDiGraph

unsorted_degrees

The degrees of the graph vertices

Type tuple

degrees

The ascendingly sorted degrees of the graph vertices.

Type tuple

unsort_io_degrees

The list of in- and out-degrees for each vertex of the graph, stored in a (in, out) tuple.

Type tuple

io_degrees

The sorted version of unsort_io_degrees.

Type tuple

max_degree

The maximal degree of a vertex in the graph.

Type int

tags
The tag numbers associated to a diagram.

Type list

adjacency_mat
The adjacency matrix of the graph.

Type NumPy array

write_graph(*latex_file*, *directory*, *write_time*)
Write the graph of the diagram to the LaTeX file.

Parameters

- **latex_file** (*file*) – The LaTeX ouput file of the program.
- **directory** (*str*) – Path to the result folder.
- **write_time** (*bool*) – (Here to emulate polymorphism).

adg.diag.**check_vertex_degree**(*matrices*, *three_body_use*, *nbody_max_observable*, *canonical_only*, *vertex_id*)
Check the degree of a specific vertex in a set of matrices.

Parameters

- **matrices** (*list*) – Adjacency matrices.
- **three_body_use** (*bool*) – `True` if one uses three-body forces.
- **nbody_max_observable** (*int*) – Maximum body number for the observable.
- **canonical_only** (*bool*) – `True` if one draws only canonical diagrams.
- **vertex_id** (*int*) – The position of the studied vertex.

```
>>> test_matrices = [[[0, 1, 2], [1, 0, 1], [0, 2, 0]],          [[2, 0, 2], [1,
↪ 2, 3], [1, 0, 0]],          [[0, 1, 3], [2, 0, 8], [2, 1, 0]]]
>>> check_vertex_degree(test_matrices, True, 3, False, 0)
>>> test_matrices
[[[0, 1, 2], [1, 0, 1], [0, 2, 0]], [[2, 0, 2], [1, 2, 3], [1, 0, 0]]]
>>> check_vertex_degree(test_matrices, False, 2, False, 0)
>>> test_matrices
[[[0, 1, 2], [1, 0, 1], [0, 2, 0]]]
```

adg.diag.**draw_diagram**(*directory*, *result_file*, *diagram_index*, *diag_type*)
Copy the diagram feynmanmp instructions in the result file.

Parameters

- **directory** (*str*) – The path to the output folder.
- **result_file** (*file*) – The LaTeX ouput file of the program.
- **diagram_index** (*int*) – The number associated to the diagram.
- **diag_type** (*str*) – The type of diagram used here.

adg.diag.**extract_denom**(*start_graph*, *subgraph*)
Extract the appropriate denominator using the subgraph rule.

Parameters

- **start_graph** (*NetworkX MultiDiGraph*) – The studied graph.
- **subgraph** (*NetworkX MultiDiGraph*) – The subgraph used for this particular denominator factor.

Returns The denominator factor for this subgraph.

Return type (str)

adg.diag.**feynmf_generator**(*graph*, *theory_type*, *diagram_name*)
Generate the feynmanmp instructions corresponding to the diagram.

Parameters

- **graph** (*NetworkX MultiDiGraph*) – The graph of interest.
- **theory_type** (*str*) – The name of the theory of interest.
- **diagram_name** (*str*) – The name of the studied diagram.

adg.diag.**label_vertices**(*graphs_list*, *theory_type*)
Account for different status of vertices in operator diagrams.

Parameters

- **graphs_list** (*list*) – The Diagrams of interest.
- **theory_type** (*str*) – The name of the theory of interest.

adg.diag.**no_trace**(*matrices*)
Select matrices with full 0 diagonal.

Parameters **matrices** (*list*) – A list of adjacency matrices.

Returns The adjacency matrices without non-zero diagonal elements.

Return type (list)

```
>>> test_matrices = [[[0, 1, 2], [2, 0, 1], [5, 2, 0]],     [[2, 2, 2], [1, 2,␣
↪3], [0, 0, 0]],     [[0, 1, 3], [2, 0, 8], [2, 1, 0]]]
>>> no_trace(test_matrices)
[[[0, 1, 2], [2, 0, 1], [5, 2, 0]], [[0, 1, 3], [2, 0, 8], [2, 1, 0]]]
>>> no_trace()
Traceback (most recent call last):
  File "/usr/lib/python2.7/doctest.py", line 1315, in __run
    compileflags, 1) in test.globs
  File "<doctest __main__.no_trace[4]>", line 1, in <module>
    no_trace()
TypeError: no_trace() takes exactly 1 argument (0 given)
```

adg.diag.**print_adj_matrices**(*directory*, *diagrams*)
Print a computer-readable file with the diagrams' adjacency matrices.

Parameters

- **directory** (*str*) – The path to the output directory.
- **diagrams** (*list*) – All the diagrams.

adg.diag.**propagator_style**(*prop_type*)
Return the FeynMF definition for the appropriate propagator type.

Parameters **prop_type** (*str*) – The type of propagators used in the diagram.

Returns The FeynMF definition for the propagator style used.

Return type (str)

adg.diag.**to_skeleton**(*graph*)
Return the bare skeleton of a graph, i.e. only non-redundant links.

Parameters **graph** (*NetworkX MultiDiGraph*) – The graph to be turned into a skeleton.

Returns The skeleton of the initial graph.

Return type (NetworkX MultiDiGraph)

adg.diag.**topologically_distinct_diagrams**(*diagrams*)
Return a list of diagrams all topologically distinct.

Parameters **diagrams** (*list*) – The Diagrams of interest.

Returns Topologically unique diagrams.

Return type (list)

4.4 MBPT diagram

4.5 BMBPT Diagram

Routines and class for Bogoliubov MBPT diagrams.

class adg.bmbpt.**BmbptFeynmanDiagram**(*nx_graph*, *tag_num*)

Bases: *adg.diag.Diagram*

Describes a BMBPT Feynman diagram with its related properties.

two_or_three_body
The 2 or 3-body characted of the vertices.

Type int

time_tag
The tag number associated to the diagram's associated TSD.

Type int

tsd_is_tree
The tree or non-tree character of the associated TSD.

Type bool

feynman_exp
The Feynman expression associated to the diagram.

Type str

diag_exp
The Goldstone expression associated to the diagram.

Type str

vert_exp
The expression associated to the vertices.

Type list

hf_type
The Hartree-Fock, non-Hartree-Fock or Hartree-Fock for the energy operator only character of the graph.

Type str

attribute_expressions(*time_diag*)
Attribute the correct Feynman and Goldstone expressions.

Parameters **time_diag** (TimeStructureDiagram) – The associated TSD.

attribute_qp_labels()
Attribute the appropriate qp labels to the graph's propagators.

extract_integral()
Return the integral part of the Feynman expression of the diag.

Returns The integral part of its Feynman expression.

Return type (str)

extract_numerator()
Return the numerator associated to a BMBPT graph.

Returns The numerator of the graph.

Return type (str)

has_crossing_sign()
Return True for a minus sign associated with crossing propagators.

Use the fact that all lines propagate upwards and the canonical representation of the diagrams and vertices.

Returns

Encode for the sign factor associated with crossing propagators.

Return type (bool)

multiplicity_symmetry_factor()
Return the symmetry factor associated with propagators multiplicity.

Returns The symmetry factor associated with equivalent lines.

Return type (str)

time_tree_denominator(*time_graph*)
Return the denominator for a time-tree graph.

Parameters **time_graph** (*NetworkX MultiDiGraph*) – Its associated time-structure graph.

Returns The denominator of the graph.

Return type (str)

vertex_exchange_sym_factor()
Return the symmetry factor associated with vertex exchange.

Returns The symmetry factor for vertex exchange.

Return type (str)

vertex_expression(*vertex*)
Return the expression associated to a given vertex.

Parameters **vertex** (*int*) – The vertex of interest in the graph.

write_diag_exps(*latex_file*, *norder*)
Write the expressions associated to a diagram in the LaTeX file.

Parameters

- **latex_file** (*file*) – The LaTeX outputfile of the program.
- **norder** (*int*) – The order in BMBPT formalism.

write_graph(*latex_file*, *directory*, *write_time*)
Write the BMBPT graph and its associated TSD to the LaTeX file.

Parameters

- **latex_file** (*file*) – The LaTeX output file of the program.
- **directory** (*str*) – The path to the result folder.
- **write_time** (*bool*) – True if we want informations on the associated TSDs.

write_section(*result*, *commands*, *diags_nbs*)
Write section and subsections for BMBPT result file.

Parameters

- **result** (*file*) – The LaTeX output file of the program.

- **commands** (*dict*) – The flags associated with run management.
- **diags_nbs** (*dict*) – The number of diagrams per type.

write_tsd_info (*diagrams_time*, *latex_file*)
Write info related to the BMBPT associated TSD to the LaTeX file.

Parameters

- **diagrams_time** (*list*) – The associated TSDs.
- **latex_file** (*file*) – The LaTeX output file of the program.

write_vertices_values (*latex_file*, *mapping*)
Write the qp energies associated to each vertex of the diag.

Parameters

- **latex_file** (*file*) – The LaTeX output file of the program.
- **mapping** (*dict*) – A mapping between the vertices in the diagram and the vertices in its euivalent TSD, since permutations between vertices are possible.

adg.bmbpt.**check_unconnected_spawn** (*matrices*, *max_filled_vertex*, *length_mat*)
Exclude some matrices that would spawn unconnected diagrams.

Parameters

- **matrices** (*list*) – The adjacency matrices to be checked.
- **max_filled_vertex** (*int*) – The furthest vertex until which the matrices have been filled.
- **length_mat** (*int*) – The size of the square matrices.

```
>>> mats = [[[0, 2, 0], [2, 0, 0], [0, 0, 0]],                    [[0, 2, 1], [2,␣
↪0, 1], [0, 0, 0]]]
>>>
>>> check_unconnected_spawn(mats, 1, 3)
>>> mats
[[[0, 2, 1], [2, 0, 1], [0, 0, 0]]]
```

adg.bmbpt.**diagrams_generation** (*p_order*, *three_body_use*, *nbody_obs*, *canonical*)
Generate diagrams for BMBPT from bottom up.

Parameters

- **p_order** (*int*) – The BMBPT perturbative order of the studied diagrams.
- **three_body_use** (*bool*) – Flag for the use of three-body forces.
- **nbody_obs** (*int*) – N-body character of the obervable of interest.
- **canonical** (*bool*) – `True` if one draws only canonical diagrams.

Returns NumPy arrays encoding the adjacency matrices of the graphs.

Return type (list)

```
>>> diagrams_generation(1, False, 2, False) #doctest: +NORMALIZE_WHITESPACE
[array([[0, 4], [0, 0]]), array([[0, 2], [0, 0]])]
>>> diagrams_generation(1, True, 3, False)  #doctest: +NORMALIZE_WHITESPACE
[array([[0, 6], [0, 0]]), array([[0, 4], [0, 0]]), array([[0, 2], [0, 0]])]
>>> diagrams_generation(2, False, 2, True)  #doctest: +NORMALIZE_WHITESPACE
[array([[0, 2, 2], [0, 0, 2], [0, 0, 0]]),
 array([[0, 1, 1], [0, 0, 3], [0, 0, 0]])]
```

adg.bmbpt.**order_diagrams** (*diagrams*)
Order the BMBPT diagrams and return number of diags for each type.

Parameters **diagrams** (*list*) – Possibly redundant BmbptFeynmanDiagrams.

Returns

First element is the list of topologically unique, ordered diagrams. Second element is a dict with the number of diagrams for each major type.

Return type (tuple)

adg.bmbpt.**produce_expressions**(*diagrams*, *diagrams_time*)
Produce and store the expressions associated to the BMBPT diagrams.

Parameters

- **diagrams** (*list*) – The list of all BmbptFeynmanDiagrams.
- **diagrams_time** (*list*) – Their associates TSDs.

adg.bmbpt.**write_header**(*tex_file*, *commands*, *diags_nbs*)
Write overall header for BMBPT result file.

Parameters

- **tex_file** (*file*) – The ouput LaTeX file of the program.
- **commands** (*Namespace*) – Flags for the program run.
- **diags_nbs** (*dict*) – The number of diagrams per type.

4.6 Time-Structure Diagram

Module with functions relative to time-stucture diagrams, called by ADG.

class adg.tsd.**TimeStructureDiagram**(*bmbpt_diag*, *tag_num*)
Bases: *adg.diag.Diagram*

Describes a time-structure diagram with its related properties.

perms
The permutations on the vertices for all the BMBPT diagrams associated to this TSD.

Type dict

equivalent_trees
The tag numbers of the equivalent tree TSDs associated to a non-tree TSD.

Type list

is_tree
The tree or non-tree character of a TSD.

Type bool

expr
The Goldstone denominator associated to the TSD.

Type str

draw_equivalent_tree_tsds(*latex_file*)
Draw the equivalent tree TSDs for a given non-tree TSD.

Parameters **latex_file** (*file*) – The output LaTeX file of the priogram.

resummation_power()
Calculate the resummation power of the tree TSD.

Returns The resummation power associated to the TSD.abs

Return type (int)

treat_cycles()
Find and treat cycles in a TSD diagram.

Returns The unique tree TSDs associated to a non-tree TSD.

Return type (list)

write_graph(*latex_file*, *directory*, *write_time*)
Write the graph of the diagram to the LaTeX file.

Parameters

- **latex_file** (*file*) – The LaTeX ouput file of the program.
- **directory** (*str*) – Path to the result folder.
- **write_time** (*bool*) – (Here to emulate polymorphism).

adg.tsd.**disentangle_cycle**(*time_graph*, *cycle_nodes*)
Separate a cycle in a sum of tree diagrams.

Parameters

- **time_graph** (*NetworkXn MultiDiGraph*) – A time-structure diagram.
- **cycle_nodes** (*tuple*) – Integers encoding the positions of the end nodes of the cycle.

Returns New graphs produced from treating the cycles in the TSD.

Return type (list)

adg.tsd.**equivalent_labelled_tsds**(*equivalent_trees*, *labelled_tsds*)
Return the list of labelled TSDs corresponding to equivalent TSDs.

Parameters

- **equivalent_trees** (*list*) – The equivalent tree TSDs of a non-tree TSD.
- **labelled_tsds** (*list*) – The labelled TSDs obtained from BMBPT diagrams.

Returns The list of tag numbers of the equivalent TSDs.

Return type (str)

adg.tsd.**find_cycle**(*graph*)
Return start and end nodes for an elementary cycle.

Parameters **graph** (*NetworkX MultiDiGraph*) – A TSD with cycle(s) to be treated.

Returns Positions of the two end nodes of a cycle in the graph.

Return type (tuple)

adg.tsd.**time_structure_graph**(*graph*)
Return the time-structure graph associated to the graph.

Parameters **graph** (*NetwrokX MultiDiGraph*) – The BMBPT graph of interest.

Returns The time-structure diagram.

Return type (NetworkX MultiDiGraph)

adg.tsd.**treat_tsds**(*diagrams_time*)
Order TSDs, produce their expressions, return also number of trees.

Parameters **diagrams_time** (*list*) – All the associated TSDs.

Returns List of TSDs, number of tree TSDs

Return type (tuple)

adg.tsd.**tree_time_structure_den**(*time_graph*)
Return the denominator associated to a tree time-structure graph.

Parameters **time_graph** (*NetworkX MultiDiGraph*) – The TSD of interest.

Returns The denominator associated to the TSD.

Return type (str)

adg.tsd.**write_section**(*latex_file*, *directory*, *pdiag*, *time_diagrams*, *nb_tree_tsds*)
Write the appropriate section for tsd diagrams in the LaTeX file.

Parameters

- **latex_file** (*file*) – The LaTeX output file of the program.
- **directory** (*str*) – Path to the output folder.
- **pdiag** (*bool*) – `True` if diagrams are to be drawn.
- **time_diagrams** (*list*) – The ensemble of TSDs.
- **nb_tree_tsds** (*int*) – Number of tree TSDs.

CHAPTER 5

Developers Team

They have been involved in the making of ADG over the past years:

- Pierre Arthuis - University of Surrey (previously Irfu, CEA, Université Paris-Saclay & CEA, DAM, DIF)
- Thomas Duguet - Irfu, CEA, Université Paris-Saclay & KU Leuven, IKS
- Jean-Paul Ebran - CEA, DAM, DIF
- Raphaël-David Lasseri - ESNT, Irfu, CEA, Université Paris-Saclay (previously IPN, CNRS/IN2P3, Université Paris-Sud, Université Paris-Saclay)
- Alexander Tichai - ESNT, Irfu, CEA, Université Paris-Saclay

CHAPTER 6

Citing

If you use ADG in your research work, we kindly ask you to cite the following paper: P. Arthuis, T. Duguet, A. Tichai, R.-D. Lasseri and J.-P. Ebran, Comput. Phys. Commun. **240**, 202-227 (2019). It is available under the following DOI.

CHAPTER 7

License

ADG is licensed under under GNU General Public License version 3 (see LICENSE.txt for the full GPLv3 License).

```
Copyright (C) 2018-2019 ADG Dev Team
Pierre Arthuis
Thomas Duguet
Jean-Paul Ebran
Raphaël-David Lasseri
Alexander Tichai
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

T

U

V

W