QGRAF 3.4.2 — an update[1,2]

P. Nogueira

CeFEMA, Instituto Superior Técnico,
Universidade de Lisboa (ULisboa), Lisbon, Portugal.

Abstract

This document is a sort of addendum to the guide for version 3.0, and describes what has changed since that version was released. The manual for version 3.0 is still the main documentation source, to be complemented by this document.

To find out what is new in version 3.4.2 see the Changelog (last page).

[1] April 2019 (due to a lapse, this updated file was absent from the corresponding distribution for the first few days)

[2] an integrated, revised guide is in preparation

Contents

1. The 'vsum' and 'psum' statements

The ability to define functions (ie parameters associated to fields, propagators, and vertices) in the model-file leads implicitly to new diagram selection methods. It is now possible to impose some numerical constraints that depend on those functions that are integer. The existence of a mechanism to (eg) restrict the powers of coupling constants seems particularly relevant, specially in models with two or more independent coupling constants. In fact, partial radiative corrections based on subsets of diagrams defined by such conditions have been routinely considered in the particle physics literature.

In practice the new filters consist of optional statements that may appear in the file `qgraf.dat` (see also the guide for version 3.0, subsection 4.3). Let `g_power` be a v-function mapping every vertex to an integer equal to the power of a certain coupling constant g in the Feynman rule for that vertex. To restrict the sum of such values for the vertices of a diagram, and thus the power of g in the diagram amplitude, one may write eg

```
true = vsum[ g_power, 4, 4] ;
```

The operator `vsum` has three arguments: the first is a v-function and the other two are numerical and similar to the corresponding arguments of operators `iprop`, `brigde`, ... , with the exception that they can be negative. The function values have to be integer numbers, which means that definitions like

```
g_power = '1/2'
```

```
g_power = ' 2'
```

are not accepted. In principle it is possible to convert a constraint involving rationals into another one depending on integers only; however, such integers should not be too large (there should be no problem if their absolute value does not exceed 10^6, say).

Obviously, this filter may be used for purposes other than selecting the powers of coupling constants. For example, let `one` be a v-function that maps every vertex to `'1'`; if `one` is used as the first argument of a `vsum` statement then the number of vertices in the diagrams will be restricted.

Here is another simple example: let V_1 be a subset of the interaction vertices, and `binary` a v-function that maps vertices in V_1 to `'1'` and vertices not in V_1 to `'0'`; the statement

```
false = vsum[ binary, 0, 0 ];
```

selects diagrams containing at least one vertex in V_1.

There is an analogous statement for propagators. For example,

```
true = psum[ pweight, -1, 1] ;
```

restricts the sum of the values of the p-function `pweight` taken over the diagram propagators.

2. The 'elink' statement

The `elink` statement provides a way to restrict the configuration of external lines — ie diagrams can be selected or rejected depending on whether some external fields are attached (or not) to the same vertex or set of vertices. Various constraints of this type, to be dubbed (external) linking conditions, may be simulated by carefully extending the model — defining new ('`external`') fields and new vertices — but that approach can be rather complicated.

Let us take a look at some examples, assuming the generic process

$$\psi_1\ \psi_2\ \rightarrow\ \phi_1\ \phi_2.$$

The field-indices of those external fields are, from left to right, equal to -1, -3, -2, -4. The statement

```
true = elink[ -1, -3, incl, 1, 1 ] ;
```

selects those diagrams in which ψ_1 and ψ_2 connect to the same vertex, *inclusively* — meaning that the other external fields may connect to any vertex, as far as this statement is concerned. For example, diagrams like D_1 and D_3 (Fig. 1) are validated, and D_2 rejected. The next statement

```
true = elink[ -1, -3, excl, 1, 2 ] ;
```

selects those diagrams in which ψ_1 and ψ_2 connect (in total) to either one or two vertices, this time *exclusively* — ie ψ_1 and ψ_2 may or may not connect to the same vertex, but in either case the other external fields must be attached to *other* vertices. This condition rejects D_2 and D_3, for instance.

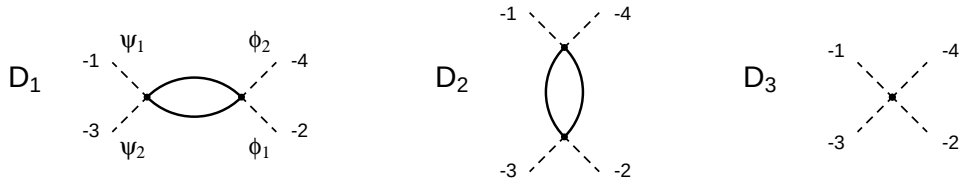

Fig. 1. Illustrating the `elink` statement.

A generic `elink` statement includes three types of arguments. Each argument of the first type should be the field-index of an external field, and therefore a negative integer; no repetitions are allowed. Then comes a non-numerical argument, either `excl` or `incl`, to specify whether the linking condition is exclusive or inclusive. The third set of arguments consists of two positive integers (a and b, say) which specify the range for the number of vertices involved in the linking condition. If k denotes the number of arguments of the first type then the inequalities

$$1 \leq a \leq b \leq k$$

should hold, otherwise an error will occur. Here are some more examples: the statement

```
false = elink[ -1, excl, 1, 1 ];
```

requires ψ_1 to link-up with some other external field(s), whereas

```
false = elink[ -1, -2, incl, 1, 1 ];
```

requires ψ_1 not to link-up with ϕ_1 (a condition which diagram D_3 fails to satisfy). Some `elink` statements are trivial, eg (still in the case of the above mentioned process)

```
true = elink[ -1, incl, 1, 1 ];
true = elink[ -1, -3, incl, 1, 2 ];
true = elink[ -1, -2, -3, -4, excl, 1, 4 ];
```

Their negation rejects every diagram, of course.

3. The 'plink' statement

This section addresses the case where the selection criterion consists in either the absence or existence of a bridge-type propagator with a specific nonzero momentum — more precisely, a momentum $\mathbf{P}$ that may be expressed as a sum of the momenta (with plus or minus signs) of certain external fields. The ability to either disallow or require the existence of those propagators may help select eg diagrams whose amplitude has some type of resonance, or tree diagrams contributing to the s-, t- and u-channels. In the case of non-tree diagrams it also provides a way to have some external fields 'on-shell' and others 'off-shell' (like a selective `onshell` option, see below).

If one picks a connected diagram that has such a propagator then cutting the corresponding edge into two pieces will split the set of external fields into two non-empty subsets — X_1 and X_2, say — each containing the external fields belonging to one of the two subdiagrams obtained. If $\mathbf{p}_i$ and $\mathbf{q}_j$ denote the incoming and outgoing momenta, respectively, then $\mathbf{P}$ can be written as

$$\mathbf{P} = \sum_i a_i \, \mathbf{p}_i - \sum_j b_j \, \mathbf{q}_j \, ,$$

with $a_i, b_j \in \{0, 1\}$ (here, the global sign will be ignored); in any case, the relative signs of the momenta in the above relation are fixed. As the external momenta satisfy the momentum conservation relation $\sum \mathbf{p}_i = \sum \mathbf{q}_j$, it should be clear that $\mathbf{P}$ can be expressed as a linear combination of the external momenta in two different ways.

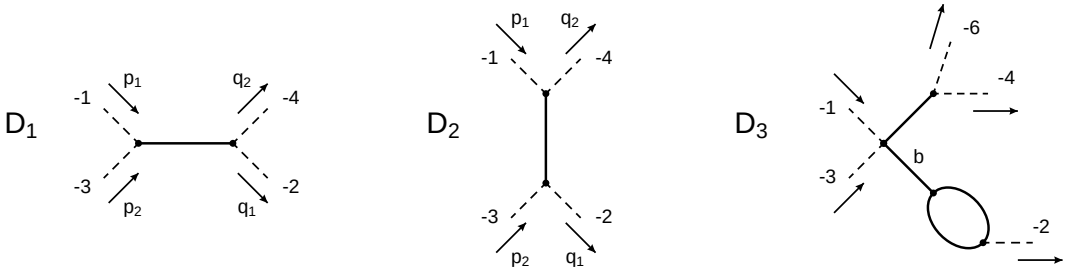

Fig. 2. Illustrating the `plink` statement.

The arguments of the `plink` statement should be the field-indices of the external fields in either X_1 or X_2; the number of arguments cannot be equal to zero, nor equal to the number of legs, as $\mathbf{P}$ would then be null. Let us take a look at some examples: the statement

```
true = plink[ -1, -3 ];
```

selects those diagrams that have at least one propagator with momentum $\mathbf{p}_1+\mathbf{p}_2$, like diagram D_1 (Fig. 2); diagram D_2, which has a propagator with momentum $\mathbf{p}_1-\mathbf{q}_2$ (or $\mathbf{p}_2-\mathbf{q}_1$), would be selected by either of the statements

```
true = plink[ -1, -4 ];
true = plink[ -3, -2 ];
```

The `plink` statement also provides a way to choose which external fields should be 'on-shell'; for instance,

```
false = plink[ -2 ];
```

rejects any diagram with a bridge separating external field -2 from all other external fields, such as diagram D_3 (which has one such bridge, labelled with the letter b).

4. Option 'bipart'

The option `bipart` selects those diagrams whose topology is a bipartite graph — ie a graph G whose node-set can be partitioned into two subsets A and B in such a way that every edge joins a node $u \in A$ to a node $v \in B$. That is the same as requiring that G has no circuit of odd length; in particular, G has no self-loops (to eliminate circuits of length 2 as well, one may add option `simple`). Clearly, every tree is bipartite. The dual option is `nonbipart`.

5. Option 'cycli'

Given the usual types of Feynman rules in momentum space, the evaluation of some diagrams involves a 'factorizable' integration, ie decomposable into a product of two or more independent integrations (for 2-loop and higher order diagrams, obviously). Even if the complete integrand does not have such a property it might be possible to decompose it into a sum of factorizable expressions, if the diagram topology is right.

Given a graph G, a circuit (or simple cycle, which physicists may call a loop) is a non-empty set of edges that, together with their endnodes, define a connected subgraph of G in which every node has degree 2. It follows from the definition that a circuit cannot be decomposed into two or more circuits. Circuits of length 1 and 2 are allowed: the former case corresponds to self-loops, and the latter to pairs of parallel edges (self-loops excluded) as in diagrams D_2 and D_3, Fig. 3. Let E_b and E_c denote the set of bridges and the set of non-bridges, respectively (a bridge is an edge that is not part of any circuit).

Definition: a cycle-block (or cycle-component) of a graph G is a non-empty subset $S \subseteq E_c$ such that

- if any circuit C contains at least one edge $e \in S$ then every edge of C is in S;
- for any two edges $e_1, e_2 \in S$, whether distinct or not, there is a circuit containing e_1 and e_2.

Observations: the first condition ensures that S contains the circuit(s) involved in the second condition; any self-loop is a cycle-block.

The diagrams validated by option `cycli`[4] are those that have a non-factorizable cycle space, ie they have at most one 'cycle-block'; the dual option is `cyclr`. Since `cycli` ignores bridge-type propagators, it is the following combination,

```
cycli, onepi
```

which selects sets of diagrams even more 'primitive' than those selected by option `onepi` only.

There is a simple interpretation of cycle-blocks in terms of sub-diagrams. Let D be an l-loop, non-tree diagram, and P_c the set of its non-bridge propagators; assume that momentum conservation has been taken into account and that $k_1, k_2 \ldots k_l$ denote the independent integration momenta; additionally, ignore the external momenta (eg set them to zero). Then each cycle-block is a minimal[5], non-empty subset S of P_c such that the two sets of momenta flowing through the propagators in S and $\bar{S} = P_c \setminus S$, respectively, are independent of each other (ie can be parametrized independently); the case where $S = P_c$ and $\bar{S}$ is empty is implicitly allowed, as there may exist a single cycle-component. Fig. 3. shows a few examples.

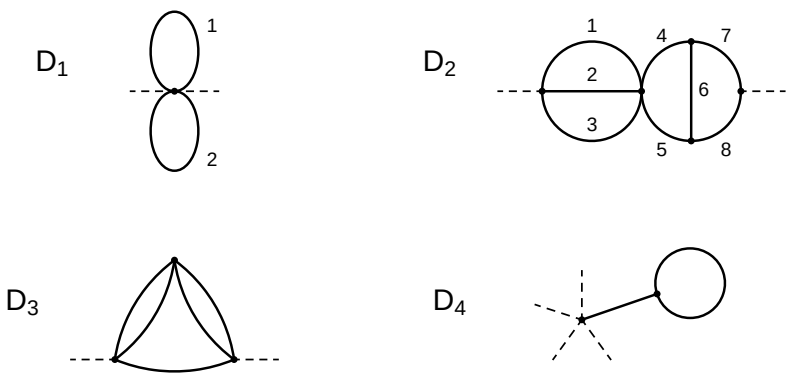

Fig. 3. Illustrating option `cycli`.

Diagram D_1 will be rejected by `cycli`: there are two self-loops and therefore two cycle-blocks. The momenta on propagators 1 and 2 (as marked), which can be defined as k_1 and k_2 (respectively), are independent and there is no propagator whose momentum has to be expressed as a linear combination in which the coefficients of k_1 and k_2 are both nonzero.

Diagram D_2 also has two cycle-components, which are defined by the subsets of propagators $S_1 = \{1, 2, 3\}$ and $S_2 = \{4, 5, 6, 7, 8\}$. Here it is possible to express eg k_3 in terms of $\{k_1, k_2\}$, and $\{k_5, k_6, k_8\}$ in terms of $\{k_4, k_7\}$ (excluding the contributions of the external momentum, obviously). Since it is possible to do so within each of the above subsets, D_2 will also be rejected.

That type of partitioning is not possible for diagram D_3, which will be validated; D_4 will be validated too, just like any tree or 1-loop diagram.

NB: in the case of 1-particle irreducible diagrams there is some overlap between options `cycli` and (see next section) `onevi`, but they do not coincide; however, if both bridges and self-loops are excluded, then they become identical.

---

[4] to be interpreted as 'cycle-irreducible'

[5] ie non-decomposable into two or more sets of the same kind

6. Option 'onevi'

Option `onevi` selects those diagrams whose topology has no articulation point, or cut vertex (in the standard terminology of graph theory). This means those diagrams that remain connected upon the removal of any single vertex (interaction vertex, that is, as external nodes are completely ignored); by definition, the empty graph is connected. Consequently, diagrams discarded by option `onevi` must have at least three vertices. The dual option is `onevr` — `onevi` means '1-vertex irreducible', and `onevr` '1-vertex reducible', of course.

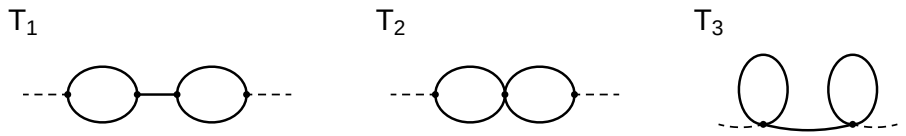

Fig. 4. Illustrating option `onevi`.

In Fig. 4, D_1 and D_2 are 1-vertex reducible: deleting any of the vertices that do not connect to an external line will generate two disjoint components. Diagram D_3 is irreducible as it has too few vertices to be reducible. There is a class of diagrams that are both 1-particle reducible and 1-vertex irreducible, for which D_3 may be seen as a kind of prototype: they have exactly two vertices (u and v, say), joined by a single propagator; u and v may have self-loops; the external fields link-up with u and/or v, obviously.

The previous figures also provide useful examples: the 1-vertex reducible diagrams are D_3 in Fig. 2, and D_2 in Fig. 3.

7. Option 'onshellx'

The option `onshell` discards diagrams that have a propagator whose splitting generates two separate diagrams, provided one of those is a 2-point diagram. In that case there will be a bridge-type propagator whose momentum equals the momentum of one of the external lines (as in diagrams D_2 and D_6, Fig. 5, where the bridge is labelled with the letter b and the corresponding external line with the letter e).

Occasionally, a more extensive elimination may be useful. The option `onshellx` also rejects diagrams for which the above mentioned bridge is absent, as if it had contracted to a single point and its two end-vertices had fused (compare eg D_1 and D_2). Diagrams rejected by `onshellx`, but not by `onshell`, will have at least one vertex of degree $\delta \geq 4$. Like most of the other options, `onshellx` has a 'topological' character: only the diagram topology matters, not the actual fields.

Diagram D_1 will be rejected by option `onshellx`, but not by `onshell`; it will be rejected by `onshellx` whether or not there exists (in the model given as input) a similar diagram D_2 with the same fields except for the bridge b. Diagram D_2 will be discarded by either option, obviously.

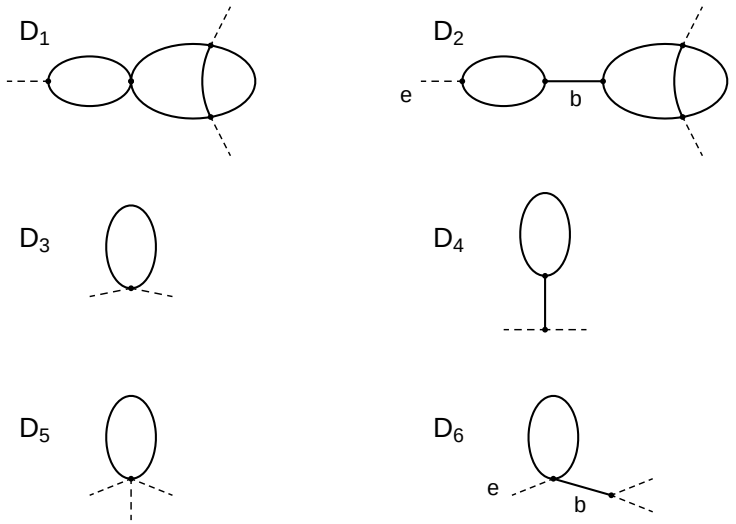

Fig. 5. Illustrating option `onshellx`.

Diagram D_3 will not be rejected by `onshellx`, unlike D_5: although D_3 may be obtained by contracting the bridge in D_4, that bridge does not isolate a single leg (and no suitable diagram exists), whereas D_5 can be obtained from D_6, which has a suitable form. If options `onshellx` and `nosnail` are used simultaneously then all six diagrams will be rejected.

The combination

```
offshellx, onshell
```

yields the diagrams rejected by `onshellx` and validated by `onshell`, and this allows a basic cross-check to be done (in the case where those diagrams are supposed to evaluate to zero). Regarding cross-checks that have been performed so far, the numbers obtained for the combination

```
onshellx, nosnail
```

agree with several numbers from L. Dixon and S. Badger (private communication[6]) for 1-loop and 2-loop diagrams, respectively. Additionally, a second algorithm has been developed for `onshellx`, and an agreement with the original algorithm was found.

8. Duplicate vertices

A model has 'duplicate vertices' if its description contains at least two vertex declarations with the exact same fields and multiplicities (ie for every field Φ_k, the multiplicity of Φ_k is the same in either declaration). Each vertex may in fact have several duplicates, and although the corresponding declarations may be identical, that needs not be the case: the values of the v-functions may differ, as well as the field ordering in each vertex.

Duplicate vertices have been tolerated for some time but typically QGRAF generated more diagrams than necessary — with the appropriate 'symmetry factors', though. Things have improved with version 3.3 : further symmetries are taken into account in this case, and the number of diagrams may now be smaller (and some factors larger).

[6] the introduction of `onshellx` is based on a suggestion made by Lance Dixon

9. The 'config' statement

The `config` statement allows the specification of a number of options that are not related to diagrams. It may appear at most once, necessarily as the first statement in file `qgraf.dat` .

9.1 The screen-modes

Given the current number of possible warning messages, it is convenient to have some control over the information sent to the 'standard output' (by default, the screen). There are three possible options, namely

`noinfo` `info` `verbose`

which may appear (not concurrently) in the `config` statement, eg

```
config = info ;
```

The 'screen-modes' defined by those options are as follows:

- `noinfo`: nothing will be displayed unless an error occurs;
- `info`: the program's name and version number, the input commands from the control-file, and the number of generated diagrams (per vertex partition, and the total number) are displayed in that order; if the diagram generation finishes but there were suppressed warning messages (which would have been shown in `verbose` mode) a warning sign is added to the line showing the total number of diagrams, eg

  ```
  total =  12343 diagrams  (w!)
  ```

- `verbose`: in addition to the information displayed in `info` mode, the program displays every 'alert' it can, as well as a summary of the model consisting of various numbers of propagators and vertices; this screen-mode may help with the debugging of input files, as it should be clear.

If an error condition is detected the corresponding error message is shown and then the program stops, obviously, irrespective of the screen-mode. If the `config` statement is either absent or contains none of those three options, QGRAF will behave as described next:

- if an output-file is to be created, the screen-mode is set to `info`;
- otherwise, the screen-mode is set to `verbose`.

9.2 Improving the write performance

If the operating system is one of those for which 'newline' is ASCII's `line feed` (`LF`) control character, like Linux/Gnu based systems, there is an experimental feature that should speed up the write operation. That feature is enabled by option `lf`, eg

```
config = lf ;
```

This should lead to a small to moderate (overall) performance increase, provided the output-file is large enough.

Compatible systems are listed in Wikipedia's *Newline*[7] page, section *Representation*. One may always run the program with and without that option (a single test case may be sufficient), and check if the output-files are identical.

9.3 Other configuration options

Option `nolist` disables creating an output-file even if a filename is declared, eg

```
config = nolist ;
output = 'dlist' ;
```

9.4 A missing option

When the number of diagrams is 'big' (larger than 10^5, say), the output-file will also be quite large, obviously, and it might be desirable to split it into several files of a more manageable size if (eg) the diagram processing software can handle them more efficiently than it would handle a single large file. Although QGRAF does not provide a way to perform this type of operation, that may not be a real problem as there exist tools which, with little effort, can be used for the job. The remaining of this sub-section describes a possible method (for Linux/Gnu operating systems).

The two lines below (in typewriter font) show an excerpt of a conceptual style-file: they represent the last line of diagram section (an extra, artificial line introduced for the present purpose) and the line declaring the epilogue section.

```
#␣<diagram_index>xyx
<epilogue>
```

We will assume that the string `xyx` does not normally appear in the output-file, ie that it is generated only when that extra line is added; if that is not the case then some suitable string should be used instead. Now we will rely on the operating system and execute the shell-command

```
csplit --prefix='xx' --digits=3 dlist '/0000xyx/' '{*}'
```

This splits output-file `dlist` into smaller files (the 'pieces'), each containing the description of 10^4 diagrams (except for the last file, obviously, which could have a smaller number), since the pattern `0000xyx` appears every time the diagram index is a multiple of 10000. In this example the pieces will be named `xx000`, `xx001`, `xx002`, and so on; the prefix `xx` and the length of the suffix (ie the number of digits) may be specified as arguments of the `csplit` command. The prefix should be chosen with care, as `csplit` overwrites existing files; also, to keep the number of pieces within reasonable bounds, the number of diagrams per file should typically be larger than 1% of the total number of diagrams (hence having 10^4 diagrams per file might be acceptable if the total number of diagrams does not exceed one million, say).

One may have a number of diagrams per file that is not a power of ten, eg 5 times a power of ten, but that is a bit more complex. It is also possible to separate the diagrams from the prologue and epilogue sections by including the string to be matched in these other sections of the style-file, more precisely in the last line of the prologue and in the first line of

[7] `https://en.wikipedia.org/w/index.php?title=Newline&oldid=863813417`

the epilogue. In any case, if necessary, the additional lines can be eliminated with the help of a simple `bash` script, eg

```
rm -f xx0tmp
for xf in `ls xx*` ; do
  grep -v xyx $xf > xx0tmp
  mv -f xx0tmp $xf
done;
```

It is safer to do the whole procedure in a directory containing only the output-file and its pieces, of course.

10. The 'index_offset' statement

When combining the output-files of two or more runs into a single file, it may be useful not to have the first diagram of every output-file being assigned the diagram index `1`. An optional statement which may appear in file `qgraf.dat`, immediately below the `options` statement, eg

```
index_offset = 1071 ;
```

instructs the program to add a non-negative constant to the default diagram index. This offset is disabled in the epilogue section, where the keyword `<diagram_index>` is still replaced by the number of diagrams listed in the output-file.

The following example shows a possible application of the `index_offset` statement. Let us suppose that the diagram selection criteria involve not a conjunction like

```
true = A ;
true = B ;
```

(where A and B represent valid expressions), but some other logical connective of two or more conditions, eg

```
( true = A ) ∨ ( true = B ).
```

Although this type of statement is not accepted, there is a way out: the inclusive disjunction can be split into three non-overlapping cases, namely (case 1)

```
true = A ;
false = B ;
```

then (case 2)

```
false = A ;
true = B ;
```

and finally (case 3)

```
true = A ;
true = B ;
```

which may be run separately; similarly, the exclusive disjunction can be divided into two pairwise-disjoint cases, the equivalence $A \Leftrightarrow B$ into two cases also, and so on.

11. Additional changes

There are other differences between versions 3.4 and 3.0, the most relevant of which are presented below.

Recent versions accept input files containing a so called `UTF-8` byte order mark, which is included by some applications when saving text. This provides the ability to create input files with a broader range of software (the actual text characters must still be the printable ASCII characters, though).

From version 3.2 onwards, and only for 1-point diagrams, the behaviour of options `nosnail` and `snail` is not the same as before: instead of rejecting all such diagrams, `nosnail` is in that case equivalent to `onshellx`.

The implicit definition of option `floop` has not remained completely fixed since its original implementation. In version 3.2 and later versions the use of `floop` requires that

- there is at least one anti-commuting field;
- every vertex has either 0 or 2 anti-commuting fields.

The diagrams rejected by `floop` are those that have at least one loop (ie circuit) of odd length for which every propagator is that of an anti-commuting field. The idea behind `floop` is Furry's theorem for QED, but it is up to the user to decide on its use. The dual option `notfloop` is also available.

There is a different presentation of the model partitions, which now includes some additional types of propagators (this refers to the screen output). Here is an example

```
propagators:  (8)    2N+  3C+  1N-  2C-
```

which is to be interpreted according to the following rules:

- The signs '`+`' and '`-`' denote the propagator sign (ie, whether the propagators are defined in terms of commuting or anti-commuting fields).
- The letter '`N`' refers to 'neutral' propagators, ie those for which the particle is equal to the anti-particle.
- The letter '`C`' refers to 'charged' propagators, ie those for which the particle and the anti-particle differ.

For instance, the propagators of Dirac fermions (and the propagators of some ghost fields as well) contribute to the coefficient of `C-`, while the propagators of Majorana fermions contribute to the coefficient of `N-`. Therefore, in the above example the model-file defines eight propagators, five 'bosonic' and three 'fermionic'; two of those bosonic propagators are neutral (string `2N+`) and the other three are charged (string `3C+`); in the case of the fermionic propagators, one is neutral and two are charged.

Recent versions differentiate between 'true-propagators' and 'non-propagators': if a declaration contains the keyword `external` then it counts as a 'non-propagator', otherwise as a 'true-propagator'. An extra line may then be displayed on the screen, eg

```
non-propagators:  (1)    1N+
propagators:  (7)    1N+  3C+  1N-  2C-
```

The corresponding numbers are mutually exclusive (in this example the model-file contains eight propagator declarations too).

In some cases the program is able to detect that there are no diagrams, either for certain vertex partitions or even for all partitions, without trying to generate any such diagram (this is work in progress). Those situations can be created by some combinations of options and/or statements. In such cases (those detected, that is) the screen output will have two additional asterisks near the corresponding number of diagrams (which will be zero, of course). In the case of a vertex partition it will look like this

```
3^2  4^1     ....      0  **
```

and if the constraints are independent of the vertex partitions (for instance, if the diagrams are required to be trees and also have tadpoles) then the program will simply display

```
total =  0 diagrams  **
```

without listing any partial result. In either case, the presence of those asterisks should mean that the usual diagram generation was not attempted.

One other possibility of that kind consists in detecting the violation (by the process given as input) of conserved particle numbers or, equivalently, of conserved quantum numbers — where 'conserved' means conserved in the model described in the model-file, obviously. For now, however, there is a separate program (QGRAF-R) which may be used for that purpose.

It is not advisable to change the program parameters, since there is a real risk of ending up with a defective program (specially for version 3.1.5 and later versions). The only changes that should be 'safe' consist in increasing one or more of the parameters

```
scbuff   sibuff   sxbuff
```

should the program report that the actual values are too small (they control the size of some arrays). That should be by no means a frequent occurrence: there would have to be eg an unusually large input file or a model with a large number of vertices (or at least some vertices of high degree). One may then try to double the value of each reported parameter, more than once if necessary, until there is no error (alas, there is also an upper limit for those parameters). Note that each of those parameters is declared in multiple subroutines, and that every such declaration should be exactly the same in every part of the Fortran program.

Changelog for release 3

3.1 (May 2005)

Introduces the `vsum` and `psum` statements. Screen output is more detailed.

3.1.1 (April 2008)

Fixes an error with `psum`, as well as a minor bug related to the screen output.

3.1.2 (September 2010)

Fixes an error with `vsum`.

3.1.3 (November 2011)

Removes non-standard options of the `OPEN` statements from the source code (following a suggestion made by the `GoSam` collaboration) to make it automatically compatible with `gfortran`.

3.1.4 (October 2012)

Introduces ability to read files containing ASCII text prepended by an `UTF-8` byte order mark.

3.1.5 (February 2018)

Includes a minimal, partial 'fix' to one of the performance bottlenecks, which nevertheless provides, in some cases, the ability to generate diagrams at roughly one additional order of perturbation theory.

3.2 (June 2018)

Introduces option `onshellx` (and its dual). Introduces option `notfloop`, the dual of `floop`. Modifies behaviour of option `nosnail` (and its dual) for 1-point diagrams. Fixes an error with the style-file processing. Fixes a problem with the diagram index (there is an error if the number of diagrams is very large, ie $> 8 \cdot 10^8$), and allows that index to exceed 2^{31}. Minor optimizations.

3.3 (July 2018)

Introduces the `elink` and `plink` statements. Improves treatment of duplicate vertices. Introduces the diagram index offset. Introduces option `bipart` and its dual. Fixes an error with the model-file processing (involving the model-constants).

3.4 (January 2019)

Introduces options `cycli`, `onevi`, and their duals. Introduces the `config` statement. Fixes a problem with option `nosigma`, and one other problem with the screen output (involving the propagator types).

3.4.1 (March 2019)

Fixes an error with with the `plink` statement, and another error involving the simultaneous use of the two options `nosnail` and `notadpole`. Adds adjective `connected` to screen output, to stress the fact that disconnected diagrams are not generated.

3.4.2 (April 2019)

Fixes a bug which generates problematic error messages about model-functions.