The CRNcode library: outline of some key features

Murad Banaji

January 1, 2025

Contents

1 Introduction

These evolving notes accompany the CRNcode library, a set of algorithms for the analysis of chemical reaction networks (CRNs), mostly in C and to a lesser extent in C++. The algorithms have been written between around 2010 and the present. They call on a range or research, both my own and of others.

The goal of the library in its current form is primarily the parameter-free analysis of CRNs: i.e., analysis based on the network structure and which can be done without numerical simulation. Some of the algorithms are well tested; others are complex and may contain errors. Please contact me if you find errors. So far, only a few features of the library are documented here. Neither the code itself, nor these notes should be considered stable.

This manual assumes familiarity with the basics of CRN theory, including the terminology. But at some point I will put up detailed notes on the theory to be used alongside this manual.

A note on notation. We refer to a CRN with n species and m irreversible reactions as an "(n, m) CRN". If we also know that the CRN has rank r, we refer to it as an "$(\mathrm{n}, \mathrm{m}, \mathrm{r})$ CRN".

2 Notes and prerequisites

The code is in C and C++ and has only ever been compiled and run on Linux. It relies heavily on a number of C/C++/Linux packages and utilities. Most important of these are the following powerful, freely available or open source, packages:

1. nauty for graph isomorphism (https://users.cecs.anu.edu.au/~bdm/nauty/) which I installed on Ubuntu using:
   `$ sudo apt install nauty`
2. GiNaC for symbolic algebra (https://ginac.de/) which I installed on Ubuntu using:
   `$ sudo apt install libginac-dev*`
3. GLPK for linear programming (https://www.gnu.org/software/glpk/) which I installed on Ubuntu using:
   `$ sudo apt install glpk-utils libglpk-dev glpk-doc python-glpk`
4. csdp for semidefinite programming (https://github.com/coin-or/Csdp) which I installed on Ubuntu using:
   `$ sudo apt install coinor-csdp`

It is very natural that this library relies heavily on these tools. This reflects the following observations:

1. **Isomorphism testing** is at the heart of recognising and batch processing CRNs. If we wish to do any "statistics" on small networks, for example deciding how many networks up to some given size have some given property, then we must start with a complete list of nonisomorphic networks. We may also wish to check a hypothesis, for example, that all networks with property

A also have property B. For this purpose it is key to be able to compare lists of CRNs to see if they are identical even if the representation of a CRN differs in the lists. For all such task the graph isomorphism software `nauty` plays a pivotal role.

2. **Computational algebra** is at the heart of chemical reaction network theory (CRNT). A large number of problems in CRNT, for example analysing matrices or polynomials associated with CRNs, are naturally tackled with the help of simple algebra (or, indeed, more complex algebraic tools, but this library limits itself to solving simpler problems). `GiNaC` has proved to be a powerful and flexible environment allowing symbolic algebra to be integrated into a variety of tasks.

3. **Convex geometry** is also key. Right from the beginning, convex geometry has played a key role in the development of CRNT (consider, for example, the deficiency theorems of Feinberg, Horn and Jackson). Diverse applications in CRNT, from the study of persistence to the study of bifurcations, naturally give rise to problems such as checking whether the cone spanned by some vectors contains some other vectors; or whether all vertices of some polytope have some desired property. Linear programming software `GLPK` has been central to rapidly carrying out such computations.

4. **Polynomial positivity.** Many problems can be formulated in terms of the feasibility or otherwise of systems of polynomial equations. Often, if we can prove that a certain polynomial associated with a CRN is positive or nonnegative on some set, then this provides a certificate that some behaviour must occur, or cannot occur, in this CRN. And in finding such certificates, semidefinite programming plays a key role. Semidefinite programming software `csdp` allows the rapid solution of many such polynomial positivity problems.

Remark 2.1. Use of `GiNaC` and `GLPK` is integrated into the code, whereas `csdp` and `nauty` are called using system calls from within `C/C++`. (Obviously this is not great, but it's how it developed.)

2.1 Compiling and running

The assumption here is that the user's main interest is in running "utilities" rather than using functions from the library in their own programs. If all goes well, the library itself and the main utilities can be constructed with:

```
$ make
```

The resulting utilities live in a folder called "`bin`". It is assumed that they will be run locally within the base directory of the project, and that in this base directory there is also a folder called "`tempfiles`" where programs can put temporary files.

In Sections 4 onwards, we go through the functionality of some these binaries.

3 CRN formats

The algorithms require, or recognise, a number of different CRN formats. Consider, for example, the CRN

$$X \to 0, \quad 0 \to Y, \quad 0 \to Z, \quad Y + Z \to X, \quad Y \to X + Z\,.$$

This could be represented in the following "human-readable" way:

```
X-->
-->Y
-->Z
Y+Z-->X
Y-->X+Z
```

We refer to this format as the "`reacstr`" format. Notice that each line must contain exactly one reaction; and the zero complex is left empty. It is also possible to use `<-->` to represent reversible reactions, or to put them in as individual irreversible reactions. Multiple CRNs in a single file are permitted, and must be separated by a line containing only `******`.

The reacstr format is inconvenient if we wish to store tens of thousands of small networks for batch processing and analysis. For such analysis, it is most natural to represent a CRN via its Petri-Net graph, and store a representation of this graph.

3.1 digraph6 format

Petri-Net graphs of CRNs are stored in the compact "digraph6" format. See the `nauty` User's Guide at `https://users.cecs.anu.edu.au/~bdm/nauty/nug26.pdf` for a description of this format. The Petri-Net graph of a CRN is an edge-labelled, bipartite, digraph, and can be encoded in digraph6 format as a multilayer digraph, with edge-labels being reconstructed by examining the different layers (see the paragraphs entitled *Isomorphism of edge-colored graphs* in the `nauty` documentation at the URL above, to understand how we pass between layers and a finite set of edge-labels).

To make basic use the algorithms in the `CRNcode` library you do not need to have knowledge of the digraph6 format, or how CRNs are encoded in it: there are routines to convert between this format and other formats (see Section 6). But it is important to note some limitations.

1. We need to know the number of species and the number of irreversible reactions (i.e., the size of the two members of the bipartition on the vertices), in order to reconstruct a CRN from its digraph6 representation. A single file in digraph6 format should contain only CRNs with a fixed number of species and reactions.

2. Species labels are not stored; so when a network is converted to human-readable format from a digraph6 string, species are given generic names.

3. Size limitations: the digraph6 format is appropriate for CRNs where the number of species, number of reactions, and stoichiometries are not too large. For example, if the total stoichiometry of any chemical complex never exceeds three, then the total number of irreversible reactions plus the number of species can range up to, but must not exceed, 31.

Example 3.1 (Converting a CRN from human-readable to digraph6 format)**.** We can convert the CRN above to digraph6 format using the `convertformat` command, documented in Section 6. For example, if the human-readable CRN is stored in a file "`examples/BasicCRN`", then we can carry out the conversion by running

```
$ ./bin/convertformat examples/BasicCRN examples/BasicCRN.d6
```

If we examine the file "`examples/BasicCRN.d6`", we find that it contains the incomprehensible digraph6 string:

```
&OCG?BO?G_?@@?AA?C_?I_?W??O??_?@??A??C??G??O?
```

In order to reconstruct the network from this string, we need to know that it has 3 species and 5 reactions. We can then run the command

```
$ ./bin/convertformat examples/BasicCRN.d6 tempfiles/tmp.reacs 3 5
```

The file "`tempfiles/tmp.reacs`" now contains

```
A0-->
-->A1
-->A2
A1+A2-->A0
A1-->A0+A2
```

Notice that the species have been given canonical labels $\mathrm{A0}$, $\mathrm{A1}$ and $\mathrm{A2}$. It is also possible that the reaction ordering can change in this process.

3.2 Other formats

Some other formats are also understood by some programs in `CRNcode`, for example the sparse format documented at `https://arxiv.org/pdf/0901.3067.pdf`. But the key formats are human-readable ("reacstr") and digraph6.

4 Generate bimolecular CRNs: "genbimol"

Generate bimolecular CRNs with a prescribed number of species and reactions, and with other prescribed properties. A single representative of each isomorphism class of CRNs is created and stored in digraph6 format. The ultimate purpose is to be able to do some "statistics" on small, bimolecular CRNs, by generating complete lists of CRNs of a certain size, and then running various filters.

The documentation obtained by running `./bin/genbimol -h` is as follows:

`./bin/genbimol`

```
Example command:
        ./bin/genbimol -fWR s3r4WR.d6 3 4

Meaning:
        Generate all nonisomorphic bimolecular CRNs on 3 species and
        4 reactions which are also weakly reversible. Write these in
        digraph6 format to the file "s3r4WR.d6"

Options:
        "-v" means verbose output;
        "-f[filter]" means a filter. E.g., "DN" means dynamically
        nontrivial, "WR" means weakly reversible, "connect" means reactions
        with connected Petri Net graphs, "rev" means reversible,
        "ALL", which is the default. Some filters can be strung together
         - e.g. "DNgenuine" means dynamically nontrivial and genuine.

Arguments after the options, three mandatory, one optional:
        output file
        number of species
        number of reactions irreversible unless you used option -frev
        Finally an optional argument "open" means forbid reactions of the
        form 0 --> X and X --> 0. You can construct all fully open networks
        by adding in all the flows to these networks.

Note that you can always filter reactions later - this may be better than
filtering as they are generated. You can also later convert to other formats.
```

Example 4.1 (Generating bimolecular $(2,2)$ CRNs)**.** The following command:

```
$ ./bin/genbimol examples/s2r2.d6 2 2
```

produces a single representative of each two-species, tw-reaction, bimolecular CRN. The output (in digraph6 format) is stored in "examples/s2r2.d6". If we examine this file, we find that it contains 225 CRNs.

Example 4.2 (Generating bimolecular, dynamically nontrivial, genuine, $(3,4)$CRNs)**.** A CRN is "genuine" if every species figures in at least one reaction; and "dynamically nontrivial" ifthe kernel of the stoichiometric matrix includes any (strictly) positive vector. The following command:

```
$ ./bin/genbimol -fDNgenuine examples/s3r4DNG.d6 3 4
```

produces a single representative of each dynamically nontrivial, genuine three-species, four-reaction, bimolecular CRN. The output (in digraph6 format) is stored in "`examples/s3r4DNG.d6`". If we examine this file, we find that it contains $25,272$ CRNs.

Note that in Example 4.2 we generated and "filtered" a set of CRNs in one step (see Section 7). But it may often be a better to generate and filter separately. For example, the outcome of Example 4.2 could be achieved with the sequence of commands:

```
$ ./bin/genbimol tempfiles/s3r4.d6 3 4
$ ./bin/filterCRNs tempfiles/s3r4.d6 tempfiles/s3r4G 3 4 genuine
$ ./bin/filterCRNs tempfiles/s3r4G.d6 tempfiles/s3r4DNG 3 4 DN
```

The first command creates all bimolecular $(3,4)$ CRNs; the second extracts those which are genuine; and the final command extracts from these those which are dynamically nontrivial.

Filtering is discussed in more detail in Section 7.

5 Enlarge bimolecular CRNs: "supCRN"

We can enlarge a set of bimolecular CRNs by adding one species or one reaction, while preserving bimolecularity. It is possible to enlarge in all possible ways; or to preserve the rank; or to insist that the rank increases by 1. It is also possible to allow or forbid simple flow reactions of the form $0 \to X$ and $X \to 0$.

The documentation obtained by running `./bin/supCRN -h` is as follows:

`./bin/supCRN`

```
Example command:
        ./bin/supCRN infile.d6 outfile.d6 3 4 spec

Meaning:
        Take all the reactions in infile.d6: these have 3 species and
        4 irreversible reactions, and are assumed to be bimolecular.
        Enlarge in all possible ways by adding a new species, while
        preserving bimolecularity. Return the results - a list of
        nonisomorphic CRNs - in "outfile.d6".

Options:
        "-v" means full debugging output
        "-o" means forbid reactions of the form 0-->X and X-->0.
        "-r" followed by 0 means preserve rank; 1 means increase the rank by 1.
        The default is no checking of rank of enlarged network.

After the options, the remaining arguments are mandatory:
        input file
        output file
        number of species
        number of reactions
        "spec" to add species or "reac" to add reactions.
```

Example 5.1 (Adding a reaction to bimolecular $(2,2)$ CRNs)**.** Recally that in Exercise 4.1 we produced a list of bimolecular $(2,2)$ CRNs and stored them in "examples/s2r2.d6". Running the command

```
$ ./bin/supCRN examples/s2r2.d6 examples/s2r3.d6 2 2 reac
```

takes this list and adds a new reaction to these CRNs in all possible ways to create a list of bimolecular $(2,3)$ CRNs, stored in "examples/s2r3.d6". Note that we have not insisted that either set of CRNs is "genuine": there will be chemical species which do not figure in any reactions in some of the networks in these lists.

In the previous example, the list of reactions in "examples/s2r3.d6" could equally well have been created directly using the command

```
$ ./bin/genbimol examples/s2r3.d6 2 3
```

However the real purpose of the command supCRN is for "inheritance" results, where we have a list of CRNs known to have some property, and theory tells us that we can enlarge these CRNs in some prescribed way while preserving this property (see, for example, https://arxiv.org/abs/2201.13105). At the moment, only the simplest modifications are allowed, i.e., adding one new species, or one new reaction.

Example 5.2 (Inheritance of Hopf bifurcation)**.** The file "examples/s3r4Hopf.d6" contains a list of 136 $(3,4)$ bimolecular CRNs known to admit nondegenerate Andronov-Hopf bifurcation with mass action kinetics. The following commands give us a list of 264 bimolecular $(4,4)$ CRNs which must admit nondegenerate Andronov-Hopf bifurcation by inheritance.

```
./bin/supCRN -r0 examples/s3r4Hopf.d6 tempfiles/tmp.d6 3 4 spec
./bin/filterCRNs tempfiles/tmp.d6 examples/s4r4Hopfinherit.d6 4 4 genuine
```

In the first command we add a new species into the reations of networks in "examples/s3r4Hopf.d6" in all possible ways which preserve bimolecularity and preserve the rank of the network. In the second command we remove from the $(4,4)$ CRNs created in this way all those which are not genuine (i.e., which have some species which figures in no reactions). The resulting list of genuine bimolecular $(4,4,3)$ CRNs (i.e., bimolecular $(4,4)$ CRNs of rank 3) all admit nondegenerate Hopf bifurcation.

6 Convert file format: "convertformat"

We can convert lists of CRNs between digraph6 and other formats (digraph6 must be either the input or output format in any conversion). The conversion may be essential prior to running a number of tests on networks, as digraph6 is the only input format accepted by some of the utilities, including, most importantly, `filterCRNs` (Section 7).

The documentation obtained by running `./bin/convertformat -h` is as follows:

`./bin/convertformat`

```
Example command:
        ./bin/convertformat -idi6 -oreacstr infile.d6 outfile.reacs 3 4

Meaning:
        Take input file "infile.d6" containing 3-species, 4-reaction CRNs
        in digraph6 format, and write these to output file "outfile.reacs"
        in human readable "reacstr" format.

Options:
        "-i[inputformat]" tells us the input file format: "di6" or "reacstr"
        or "Sauro" or "simpstr". If not specified, the algorithm tries to
        determine the format from the file, but may fail.
        "-o[outformat]" tells us the output file format: "di6" or "reacstr"
        or "Sauro" or "simpstr" or "dot". If not specified, the algorithm
        tries to determine the format from the file ending, but may fail.

The arguments are:
        input file
        output file
        number of species needed only if input format is "di6" or "simpstr"
        number of reactions needed only if input format is "di6" or "simpstr"

One of either input or output file must be in digraph6 format.
```

7 Filter a list of CRNs: "filterCRNs"

The most important command. Take a list of CRNs (these *must* be in digraph6 format) and keep only those which satisfy some requirement. There are a large number of possible "filters" and "pseudofilters" (these output properties of a list of CRNs without specifically extracting a subset of the CRNs satisfying some criterion). There are also "isomorphism" tests which output only one representative of each equivalence class of CRNs from a list of CRNs.

A list of filters, pseudofilters, and isomorphism tests, which may not be up-to-date, is in Appendix A.

The documentation obtained by running `./bin/filterCRNs -h` is as follows:

`./bin/filterCRNs`

```
Example command:
        ./bin/filterCRNs -v infile.d6 outfile.d6 3 4 rank 3 2> logfiles/tmp.log

Meaning:
        Examine the file "infile.d6" containing 3-species, 4-reaction CRNs,
        in digraph6 format, and write to "outfile.d6" only those CRNs
        of rank 3. Write debugging output to "logfiles/tmp.log"

Options:
        "-v" means verbose output.

The following arguments are mandatory:
        input file in d6 format
        output file in d6 format
        number of species
        number of reactions
        filter
        [for some filters] an additional numerical argument.

The list of supported filters is long and can be found by examining
"allfilters_annotated".
```

7.1 Filtering examples

Example 7.1 (CRNs with connected Petri Net graphs)**.** We first generate all two-species, three-reaction bimolecular CRNs with the command

```
$ ./bin/genbimol examples/s2r3.d6 2 3
```

This writes 2044 CRNs to the file "examples/s2r3.d6" in digraph6 format. If we want to view the CRNs we have just created, we can write them to stdout using the pseudofilter "printreacs" using the command

```
$ ./bin/filterCRNs examples/s2r3.d6 NULL 2 3 printreacs
```

(Note that this does not write a new file, but we could choose to redirect stdout to a file.) We can then extract from the 2044 CRNs in "examples/s2r3.d6" those which have a connected Petri Net graph using the command

```
$ ./bin/filterCRNs examples/s2r3.d6 examples/s2r3con.d6 2 3 PNconnect
```

We find that 1934 of the 2044 networks have a connected Petri Net graph; i.e., no species is unused, and the CRN does not decompose into two subCRNs on disjoint sets of species.

Example 7.2 (Bounded stoichiometric classes)**.** A CRN has bounded stoichiometric classes if and only if the image of its stoichiometric matrix includes no nonnegative vector other than 0; or, equivalently, if and only if the kernel of the transpose of its stoichiometric matrix includes a (strictly) positive vector. Checking this is a problem in convex geometry. We can check how many of the $(2,3)$ CRNs with connected Petri Net graphs created in Example 7.1 have unbounded stoichiometric classes using the command

```
$ ./bin/filterCRNs examples/s2r3con.d6 examples/s2r3conbd.d6 2 3 notbdclass
```

We find that 1906 have unbounded stoichiometric classes (i.e., only 28 have bounded stoichiometric classes).

Example 7.3 (Dynamically nontrivial CRNs)**.** A basic property of a CRN is whether it is "dynamically nontrivial", a necessary condition for it to admit positive equilibria, or indeed any positive omega-limit sets. This is equivalent to asking whether the kernel of the stoichiometric matrix includes any (strictly) positive vector, essentially a problem in convex geometry which can be tackled with linear programming. We can check which of the $(2,3)$ CRNs with connected Petri Net graphs created in Example 7.1 are dynamically nontrivial using the command

```
$ ./bin/filterCRNs examples/s2r3con.d6 examples/s2r3conDN.d6 2 3 DN
```

We find that only 304 of the 1934 CRNs in "examples/s2r3con.d6" are in fact dynamically nontrivial.

Example 7.4 (Concordant CRNs)**.** A CRN is concordant if the Jacobian matrix, with positive general kinetics, is always an isomorphism on the stoichiometric subspace. We consider the dynamically nontrivial, genuine, $(3,4)$ CRNs created in Exercise 4.2, and stored in the file "examples/s3r4DNG.d6", and ask how many of these CRNs are concordant. Running the command:

```
$ ./bin/filterCRNs examples/s3r4DNG.d6 examples/s3r4DNGconc.d6 3 4 concord
```

we find that 8844 of these CRNs are concordant.

---

8 Compare lists of CRNs: "compareCRNlists"

Compare the CRNs in two files, assumed to all have the same dimensions. We can find out if one list of CRNs is a subset of the other, even if the CRNs are differently labelled. Each list is first canonically labelled (using nauty) and only one copy of each network is kept. After this, it is simply a matter of using the Linux utility comm to find common entries in the lists.

The documentation obtained by running ./bin/compareCRNlists -h is as follows:

./bin/compareCRNlists

```
Example command:
        ./bin/compareCRNlists file1.d6 file2.d6 3 4 a.d6 b.d6 c.d6

Meaning:
        Compare CRNs in "file1.d6" and "file2.d6" all assumed to have
        3 species and 4 irreversible reactions. Identify common reactions,
        up to isomorphism. Print one representative of each CRN in
        "file1.d6" but not "file2.d6" to "a.d6"; one representative
        of each CRN in "file2.d6" but not "file1.d6" to "b.d6";
        and one representative of each CRN common to both "file1.d6"
        and "file2.d6" to "c.d6". If some of these sets are empty or
        filenames are not given, then output files will not be created
        but a report will still be given.

Arguments: the first four are mandatory.
        input file in digraph6 format
        output file in digraph6 format
        number of species
        number of reactions
        optional: a file to hold CRNs in "file1.d6" but not "file2.d6"
        optional: a file to hold CRNs in "file2.d6" but not "file1.d6"
        optional: a file to hold CRNs in both "file1.d6" and "file2.d6".

Note: this program relies on calls to nauty and makes heavy use of Linux
tools such as "sort" and "comm".
```

Example 8.1 (Comparing two lists of CRNs)**.** The following command

```
$ ./bin/compareCRNlists examples/s3r4DNG.d6 examples/s3r4DNGrank3.d6 3 4
```

compares the networks in the files examples/s3r4DNG.d6 and examples/s3r4DNGrank3.d6. It produces the following output:

```
examples/s3r4DNG.d6: 25272
examples/s3r4DNGrank3.d6: 14670
```

```
networks in "examples/s3r4DNG.d6" but not in "examples/s3r4DNGrank3.d6": 10602
networks in "examples/s3r4DNGrank3.d6" but not in "examples/s3r4DNG.d6": 0
all networks in "examples/s3r4DNGrank3.d6" are also in "examples/s3r4DNG.d6".
```

This tells us that the first file contains $25,272$ CRNs, the second contains $14,670$ CRNs, and the CRNs in the second file are a subset of those in the first. In fact, this is no surprise. The first file (`examples/s3r4DNG.d6`) contains a list of all dynamically nontrivial, genuine, three-species, four-reaction, bimolecular CRNs. These were created using the command

```
$ ./bin/genbimol -fDNgenuine examples/s3r4DNG.d6 3 4
```

(See Section 4.) The second file (`examples/s3r4DNGrank3.d6`) contains the same set of reactions after filtering to keep only CRNs with rank 3. It was created by running the filter:

```
$ ./bin/filterCRNs examples/s3r4DNG.d6 examples/s3r4DNGrank3.d6 3 4 rank 3
```

(See Section 7). Thus the output from the `compareCRNlists` tells us, as expected, that the rank 3 networks are a subset of the full set.

Example 8.2 (Verifying a theorem for a particular class of networks)**.** As a more nontrivial example, let us consider the fact that CRNs which are weakly reversible and concordant are also structurally persistent (i.e., have no critical siphons). We can check this amongst bimolecular CRNs with three species and four irreversible reactions via the following sequence of commands:

```
$ ./bin/genbimol examples/s3r4.d6 3 4
$ ./bin/filterCRNs examples/s3r4.d6 examples/s3r4WR.d6 3 4 WR
$ ./bin/filterCRNs examples/s3r4WR.d6 examples/s3r4WRconc.d6 3 4 concord
$ ./bin/filterCRNs examples/s3r4.d6 examples/s3r4SP.d6 3 4 structpersist
$ ./bin/compareCRNlists examples/s3r4WRconc.d6 examples/s3r4SP.d6 3 4
```

The first command generates all three-species, four-reaction, bimolecular CRNs ($427,770$). The next filters these to keep only those which are weakly reversible (only 540). We then filter the weakly reversible ones to get those which are also concordant (219). At this point we could simply check that all of these 219 CRNs are structurally persistent with another filtering. Instead we filter all three-species, four-reaction, bimolecular CRNs to find those which are structurally persistent ($69,548$). Finally, we use `bin/compareCRNlists` to compare the lists of weakly reversible and concordant CRNs on the one hand, and those which are structurally persistent on the other. The final command produces the output:

```
examples/s3r4WRconc.d6: 219
examples/s3r4SP.d6: 69548
networks in "examples/s3r4WRconc.d6" but not in "examples/s3r4SP.d6": 0
networks in "examples/s3r4SP.d6" but not in "examples/s3r4WRconc.d6": 69329
all networks in "examples/s3r4WRconc.d6" are also in "examples/s3r4SP.d6".
```

We see that the weakly reversible and concordant CRNs are indeed a subset of those which are structurally persistent; but in fact they are a very small subset of the structurally persistent CRNs.

9 Reconstruct a list based on equivalence classes: "getallisomorphs"

Find all representatives of a given set of CRNs based on a lookup table (i.e., without isomorphism checking, and assuming that the labelling hasn't changed). The idea is to use this to speed up filtering as described in Section 7.

1. We start with a list of CRNs, say List 1.
2. We keep only one representative of each equivalence class in, say, List 2; we also keep the look-up table to show which CRNs in List 1 are equivalent to which CRNs in List 2. This is in the format generated by nauty, i.e., each index in the new list matched to a set of indices in the original list.
3. We now filter CRNs in List 2 (as many times as we like) to arrive at List 3.
4. We finally use the look-up table to find which CRNs in List 1 are equivalent to the CRNs we have found in List 3.

The documentation obtained by running ./bin/getallisomorphs -h is as follows:

./bin/getallisomorphs

```
Example command:
        ./bin/getallisomorphs isotable base.d6 reduced.d6 infile.d6 outfile.d6

Meaning:
        The idea is that we previously examined the CRNs in "base.d6",
        kept only one representative of each class in "reduced.d6", and
        stored a list of equivalence classes - in the format output by
        nauty - in "isotable". We now have a list of CRNs in "infile.d6",
        and wish to find all the CRNs from the original list in "base.d6"
        which are isomorphic to CRNs in "infile.d6"; but just using the
        lookup table, rather than doing isomorphism checking. The
        output is stored in "outfile.d6".

Four input files and one output file are needed, in this order:
        1. The isomorphism table in nauty format
        2. The file used to generate the isomorphism table in digraph6 format
        3. The output from isomorphism generation in digraph6 format
        4. The input file whose isomorphs we want to find in digraph6 format
        5. The output file to hold isomorphs in digraph6 format.

The lookup could be generated, for example, using "filterCRNs", e.g. via:

./bin/filterCRNs base.d6 reduced.d6 <numspec> <numreacs> dynisomorphMA 2> isotable

All files must be in digraph6 format.
```

10 Report on a network or a list of networks: "reacreport"

We can analyse a single CRN or a short list of CRNs. There are a variety of computations which may be carried out, depending on the options chosen, and some of these could be quite time-consuming unless the network is small. The tests that can be run are a subset of the filters and pseudofilters in Appendix A.

The documentation obtained by running `./bin/reacreport -h` is as follows:

`./bin/reacreport`

```
Example command:
        ./bin/reacreport -idi6 infile.d6 3 4

Meaning:
        Generate reports on basic properties and allowed behaviours of the
        CRNs in "infile.d6", which involve 3 species and 4 reactions,
        and are in digraph6 format. Write the report to standard output.

Options:
        "-i[inputformat]" tells us the input file format: "di6" or
        "reacstr" or "Sauro" or "simpstr". If not specified, the algorithm
        tries to determine the format from the file, but may fail.
        "-e[effort]": an integer from 0 to 5. The default is 0. Only
        affects some computations which involve polynomial positivity.
        "-f[filter]": to perform only some computations. E.g., "basic",
        "concord", "hopf".

The arguments are:
        input file
        number of species needed only if input format is "di6" or "simpstr"
        number of reactions needed only if input format is "di6" or "simpstr".

Note:  The CRNs are assumed to be small, otherwise some tests will
take too long, and the program will have to be killed manually.
```

Example 10.1 (Getting basic information about a CRN)**.** The file `networks/s3r4_example` contains the following CRN:

$$0 \to X, \quad X + Y \to 2Z, \quad Y + Z \to 2Y, \quad 2Z \to Z\,.$$

Abbreviated output from running

```
$ ./bin/reacreport networks/s3r4_example
```

is shown below

```
numspec: 3
numreac: 4
rank: 3
genuine: 1
DN: 1
structpersist: 0
concordant: 1
Hopf bifurcations forbidden?: 1
```

The portion of the output shown tells us that:

1. The network has 3 species, and 4 reactions, and rank 3.
2. It is genuine (all species figure in at least one reaction).
3. It is dynamically nontrivial (the kernel of the stoichiometric matrix has nonempty intersection with the positive orthant: nontrivial dynamics is not automatically ruled out).
4. It fails to be structurally persistent (i.e., it has critical siphons).
5. It is concordant (the Jacobian matrix with positive general kinetics, is an isomorphism on the stoichiometric subspace).
6. Hopf bifurcations are forbidden for mass action kinetics (in this case, by examining the more verbose output, we can find out that the mass action Jacobian matrix at equilibria forbids a pair of imaginary eigenvalues).

11 Search for potential Hopf bifurcation: "ruleouthopf"

Attempt to rule out Andronov-Hopf bifurcation in a list of (small) networks given in digraph6 format. Roughly speaking, the algorithm is equivalent to carrying out a number of filters on the CRNs, each of which is sufficient to rule out the passage of a pair of nonzero imaginary eigenvalues through the imaginary axis for the reduced Jacobian matrix. If ruleouthopf fails, this does not mean that a passage of imaginary pair is permitted; just that none of the tests were able to prove that it can't occur.

The documentation obtained by running ./bin/ruleouthopf -h is as follows:

./bin/ruleouthopf

```
You must supply at least five arguments.

Example command:
        ./bin/ruleouthopf -v -fMAonly -e0 infile.d6 hopf.d6 nohopf.d6 3 4 2> tempfiles/tmp.log

Meaning:
        Examine the 3-species, 4-reaction CRNs in "infile.d6" in
        digraph6 format for potential Andronov-Hopf bifurcation with
        mass action kinetics. Write CRNs where Hopf bifurcation is
        definitely ruled out to "nohopf.d6". Write the remaining CRNs
        to "hopf.d6". Effort level is minimal: "-e0".
        Debugging output to "tempfiles/tmp.log"

Options:
        "-v" means verbose output;
        "-f[filter]" means a filter: e.g., "MAonly" for only mass action,
        "GKonly" for only general kinetics, or "ALL"  for both.
        "-e[effort]" means effort, an integer from 0 to 3. The default is 0.

Arguments:
        input file in d6 format
        output file with potential Hopf in d6 format
        output file where Hopf ruled out in d6 format
        number of species
        number of reactions.
```

12 Test for polynomial positivity "polytest"

Many of the algorithms used for the analysis of CRNs rely on testing polynomial positivity. The function polytest allows examination and testing of these algorithms on user-supplied polynomials.

The documentation obtained by running ./bin/polytest -h is as follows:

./bin/polytest

```
Example command:
        ./bin/polytest -v -fposorth -e2 polyfile 2> tempfiles/tmp.log

Meaning:
        Examine the file "polyfile" assumed to contain a single multivariate
        polynomial, and attempt to determine if this polynomial is positive
        or nonnegative on the positive orthant. Put in a medium level of
        effort: "-e2". Send debugging output to "tempfiles/tmp.log"

Options:
        "-v" means verbose output;
        "-f[filter]" means a filter - see below for a list.
        "-e[effort]" means effort: an integer from 0 to 5, the default is 0.

Filters:
        The most important filters are "posorth": attempt to determine
        positivity on the positive orthant; and "posall": attempt to
        determine positivity on all of R^n, except possibly at 0. Other
        filters are "newton": examine the Newton polytope of the
        polynomial; "heuristic": do only heuristic tests to determine
        positivity on the positve orthant, i.e., avoid semidefinite
        programming; and "factor": attempt to factor using GiNaC.


Notes: The polynomial is homogenised, and may be simplified in many ways,
before any tests. If it has many terms or is of high degree, then
some tests will likely fail, and the programme may need to be killed.
The effort parameter is crucial if semidefinite programming is used
in any of the tests. If you really want to track what is being done,
then increase the debug level, by swapping the option "-v" for,
say, "-d5": but be warned much of this debugging output will be
hard to understand!
```

Example 12.1 (A trivial polynomial)**.** The file data/poly001 contains the following polynomial which is strictly positive on the positive orthant, but can take all signs on $\mathbb{R}^2$:

```
x^2-2*x+1+y
```

The command

```
$ ./bin/polytest -e0 data/poly001
```

produces the following output which confirms that this polynomial is positive on the positive orthant, and can take all signs on $\mathbb{R}^2$. Note the polynomial actually examined is a homogenised version of the original.

```
Examining the polynomial:
v3^2-2*v1*v3+v1^2+v2*v3
Searched for a factorisation (GiNaC). None found.

**Examining the Newton polytope.
The result of NewtonPolyVertSgn1: 1
This means that all vertex terms are positive (but not all terms are positive).

**Trying a heuristic to find an SOS+(positive terms) decomposition
The result of heuristic_squares (homogenised polynomial): 2
This means that on the positive orthant this polynomial is (strictly) positive.

**Examining the sign of the polynomial on the positive orthant using "ispospoly".
The result of ispospoly (not using SDP): 2
This means that on the positive orthant this polynomial is (strictly) positive.

**Examining the sign of this polynomial on all of R^n\{0} using "isfullpospoly".
The Newton polytope of this polynomial has nonsquare vertices. The polynomial
can be both positive and negative on R^n\{0}.
The result of isfullpospoly (not using SDP): -3
This means that on R^n\{0} this polynomial is indefinite.
```

Example 12.2 (A strictly positive polynomial)**.** The file `data/SOSbutstrict` contains the following polynomial:

$$(y - x^2)^2 + (y - x - 1)^2 = 2y^2 - 2x^2y - 2xy + 2y + x^4 + x^2 - 2x + 1$$

(in expanded form). The command

```
$ ./bin/polytest -fposall -e1 data/SOSbutstrict
```

aims to find out whether this polynomial is positive on all of $\mathbb{R}^2$. It produces the following output:

```
Examining the polynomial:
v3^2*v1^2+v1^4-2*v3^3*v1+2*v3^3*v2-2*v3^2*v1*v2+2*v3^2*v2^2-2*v3*v1^2*v2+v3^4

**Examining the sign of this polynomial on all of R^n\{0} using "isfullpospoly".
The result of isfullpospoly (trying SDP up to degree 0): 2
This means that on R^n\{0} this polynomial is (strictly) positive.
```

Notice that again it is the homogenised version which is examined (so positivity on $\mathbb{R}^3\backslash\{0\}$ amounts to positivity of the original polynomial on $\mathbb{R}^2$). Semidefinite programming is needed to confirm this.

Example 12.3 (The Motzkin polynomial)**.** We examine the Motzkin polynomial

$$x^4y^2 + x^2y^4 - 3x^2y^2 + 1$$

which is well known to be nonnegative on $\mathbb{R}^2$, but is not a sum of squares. Running the command

```
$ ./bin/polytest -e3 data/Motzkin
```

produces the following output:

```
Examining the polynomial:
-3*v1^2*v2^2*v3^2+v1^4*v2^2+v1^2*v2^4+v3^6
Searched for a factorisation (GiNaC). None found.

**Examining the Newton polytope.
The result of NewtonPolyVertSgn1: 1
This means that all vertex terms are positive (but not all terms are positive).

**Trying a heuristic to find an SOS+(positive terms) decomposition
The result of heuristic_squares (homogenised polynomial): -4
This means that on the positive orthant the heuristic algorithm failed to find
a simple SOS+(positive terms) decomposition.

**Examining the sign of the polynomial on the positive orthant using "ispospoly".
The result of ispospoly (tried SDP up to degree 2): 1
This means that on the positive orthant this polynomial is nonnegative.

**Examining the sign of this polynomial on all of R^n\{0} using "isfullpospoly".
The result of isfullpospoly (trying SDP up to degree 2): 1
This means that on R^n\{0} this polynomial is nonnegative.
```

In the last example, semidefinite programming was needed to confirm that the polynomial was nonnegative (on the positive orthant, and all of $\mathbb{R}^2$). Some examples of how semidefinite programming is used to solve such problems are given in Appendix B.

A Available filters for use with filterCRNs

The list of filters here is only partial. A more complete and up-to-date list is available in the file "allfilters_annotated". All filters which are not pseudofilters, or isomorphism filters, can also take "not" in front of them. So, for example, "notDN" means "not dynamically nontrivial" (i.e., dynamically trivial). And "notrank 3" means networks which have rank other than 3.

Notation in the table:

- $\mathcal{S}$: the stoichiometric matrix of the CRN.
- J_{GK}: the Jacobian matrix for positive general kinetics.
- $\mathrm{M}^{[2]}$: the second additive compound matrix of any matrix M.
- J: the mass action Jacobian matrix.
- J_e: the mass action Jacobian matrix at equilibria in canonical form, i.e., in terms of the species concentrations, and the a parameterisation of the positive part of the kernel of the stoichiometric matrix.
- Q. The first factor in the natural factorisation: $\mathrm{J}_e = \mathrm{QD}_z$, where D_z is a diagonal matrix of positive indeterminates.

Note. A $*$ in front of a test name means that the test can fail even though the condition holds. All commands with a $*$ take an additional argument which is the "effort" to use: usually an integer from 0–3. (Using larger effort can increase the chances of success, but significantly slow down the computations.) Consider, for example, JadmitsIpair. If it succeeds, it tells us that J_{GK} definitely admits a pair of imaginary eigenvalues. But if it fails, it may still be the case that J_{GK} admits a pair of imaginary eigenvalues. Equivalently, success of notJadmitsIpair only means that the test was unable to confirm that J_{GK} admits a pair of imaginary eigenvalues, not that this definitely cannot happen.

Note. A "P:" in front of a test means that it is a pseudofilter (in this case, the output file must be set to NULL); an "I:" means that it is an isomorphism filter, returning a single member of each isomorphism class, and writing a look-up table to stderr.

Table 1: A selection of filters for use with filterCRNs.

filter [alternative name]	meaning [additional arguments]
genuine	every species participates in at least one reaction
repeatedreacs	the network has repeated reactions
redundantreacs	the network has "redundant" reactions with some reactant vectors in the nonnegative span of other reactions on the same reactant complex
DN [dynnontriv]	dynamically nontrivial (a strictly positive vector in the kernel of the irreversible stoichiometric matrix)

`rank`	rank of network [rank (nonnegative integer)]
`PNconnect`	connected Petri-Net graph (the network doesn't decompose into two subnetworks without common species)
`bdclass`	bounded stoichiometric classes
`WR`	weakly reversible
`structpersist`	no critical siphons
`deficiency`	CRNs with given deficiency [deficiency (nonegative integer)]

Tests involving the positive general kinetics Jacobian matrix J_{GK}

`PGKnormal` [`notstructdiscord`]	not structurally discordant: J_{GK} does not identically fail to be an isomorphism of $\mathcal{S}$.
`concord`	J_{GK} is everywhere an isomorphism of $\mathcal{S}$
`DSRCondStarBoth`	DSR graph satisfies Condition $*$ (either before or after trying to merge reactions)
`accord`	$-\mathrm{J}_{\mathrm{GK}}$ is a P_0-matrix
* `JsquaredisP0`	$\mathrm{J}_{\mathrm{GK}}^2$ is a P_0-matrix
`DSR2CondStarBoth`	$\mathrm{DSR}^{[2]}$ graph satisfies Condition $*$ (either before or after trying to merge reactions)
* `Jcomp2isP0minus`	$-\mathrm{J}_{\mathrm{GK}}^{[2]}$ is a P_0-matrix
* `Jcomp2isP0`	$\mathrm{J}_{\mathrm{GK}}^{[2]}$ is a P_0-matrix
* `Jcomp2nonsing`	$\mathrm{J}_{\mathrm{GK}}^{[2]}$ is nonsingular
* `Jcomp2detsigned`	$\mathrm{J}_{\mathrm{GK}}^{[2]}$ has signed determinant (positive, negative or zero)
* `Jcomp2detunsigned`	$\mathrm{J}_{\mathrm{GK}}^{[2]}$ definitely has determinant which can take all three signs
* `JadmitsIpair`	J_{GK} definitely admits a pair of imaginary eigenvalues
* `J2pIdetpos`	$\mathrm{J}_{\mathrm{GK}}^2 + \mathrm{I}$ is nonsingular

Tests involving the mass action Jacobian matrices J and J_e

`normal`	normal: J does not identically fail to be an isomorphism of $\mathcal{S}$
`semiconcord`	semiconcordant: J is an isomorphism on $\mathcal{S}$
`MAdegenerate`	mass action positive equilibria are always degenerate
* `MAeqconcord`	J_e is an isomorphism on $\mathcal{S}$
`semiaccord`	semiaccordant: $-\mathrm{J}$ is a P_0 matrix
* `MAeqaccord`	$-\mathrm{J}_e$ is a P_0-matrix
* `QMAnegsemidef`	Q is negative semidefinite (hence $-\mathrm{J}_e$ is in the closure of the Hurwitz matrices)
* `JMAadmitsIpair`	J_e admits a pair of nonzero imaginary eigenvalues
* `JMAcomp2isP0minus`	$-\mathrm{J}_e^{[2]}$ is a P_0-matrix

* `JMAcomp2isP0`	$J_e^{[2]}$ is a P_0-matrix
* `JMAcomp2nonsing`	$J_e^{[2]}$ is nonsingular
* `JMAcomp2detsigned`	$J_e^{[2]}$ has signed determinant
* `JMAcomp2detunsigned`	$J_e^{[2]}$ has determinant which takes all signs
* `JMAsquaredisP0`	J_e^2 is a P_0-matrix
* `J2pIMAdetpos`	$J_e^2 + I$ is nonsingular
* `MAeqdegen2`	The second lowest order nontrivial coefficient in the characteristic polynomial of J_e can have all signs
* `JMAcomp2detnonstationary`	partial derivatives of $\det(J_e^{[2]})$ w.r.t. the positive variables z are never simultaneously zero

Isomorphism testing

I: `isomorph`	returns a unique representative in each equivalence class
I: `dynisomorphMA`	returns a unique representative in each "simple equivalence" class

Pseudotests

P: `printreacs`	print the reactions in human readable form
P: `printJ`	print J_{GK}
P: `printMAeq`	print J_e (the mass action Jacobian matrix at equilibria in canonical form)
P: `printQ`	print Q (the first factor in the mass action Jacobian matrix at equilibria in canonical form)
P: `printsiphons`	pseudotest to print out the siphons of a network
P: `molecularity`	pseudotest to print out a report on the molecularities of the reactions in the network
P: `JMArealspec`	prints out information on the real spectrum of J_e.
P: `linkageclasses`	prints the number of linkage classes to stdout
P: `deficiency`	prints the deficiency to stdout

B Polynomial positivity using csdp

The process of translating polynomial positivity questions into semidefinite programming problems in csdp is somewhat complex. We illustrate with a couple of basic examples.

B.1 A very basic example

We consider, as a very simple example, the polynomial

$$2x^2 - 4xy + 3y^2 .$$

Our goal is first to confirm, using csdp that the polynomial is strictly positive on the positive orthant. Let a, b, c, d, e, f be unknowns. We want to solve:

$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + dx^2 + exy + fy^2 = 2x^2 - 4xy + 3y^2 , \tag{1}$$

subject to $d, e, f \geq 0$, $d + e + f > 0$, and

$\begin{pmatrix} a & b \\ b & c \end{pmatrix}$ being positive semidefinite.

In practice we set the constraint $d + e + f > 0$ by defining a new nonnegative variable g, and setting $d + e + f - g = \delta$ where $\delta > 0$ is some small tolerance, set by default to be 1% of the absolute value of the coefficient with largest absolute value in the polynomial (in this case, $\delta = 0.04$).

Comparing coefficients in (1) we find the four constraints

$$\begin{aligned} a + d &= 2 \\ b + e &= -4 \\ c + f &= 3 \\ d + e + f - g &= \delta \end{aligned}$$

In practice we can put our nonnegative unknowns into a single (positive semidefinite) matrix

$$X = \begin{pmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} & & & & \\ & [d] & & & \\ & & [e] & & \\ & & & [f] & \\ & & & & [g] \end{pmatrix} .$$

(The remaining entries are zero.) Each constraint then corresponds to an expression of the form $\mathrm{tr}(A_i X) = 0$ for a matrix A_i with the same structure as X. For example, the constraing $a + d = 2$

corresponds to $\mathrm{tr}(A_1 X) = 0$, where

$$A_1 = \begin{pmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & & & & \\ & [1] & & & \\ & & [0] & & \\ & & & [0] & \\ & & & & [0] \end{pmatrix}.$$

Running the appropriate command generates the following csdp file containing this problem in the format described in the `csdp` user manual at `https://github.com/coin-or/Csdp/blob/master/doc/csdpuser.pdf`.

```
4
5
2 1 1 1 1
2.0000 -4.0000 3.0000 0.0400
1 1 1 1 1.0
1 2 1 1 1.0
2 1 1 2 1.0
2 3 1 1 1.0
3 1 2 2 1.0
3 4 1 1 1.0
4 2 1 1 1.0
4 3 1 1 1.0
4 4 1 1 1.0
4 5 1 1 -1.0
```

This says (i) that there are 4 constraints; (ii) there are 5 blocks in X; (iii) the block sizes are $2, 1, 1, 1$ and 1 respectively; (iv) the constraint right hand sides are $2, -4, 3$ and 0.04 respectively. This is followed by the four constraint matrices.

How does the problem change if we want to check only nonnegativity? The only change is that now we set $\delta = 0$.

How does this change if we want to confirm that the polynomial is positive on all of $\mathbb{R}^2 \backslash \{0\}$? In this case we want to solve:

$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + dx^2 + fy^2 = 2x^2 - 4xy + 3y^2, \tag{2}$$

subject to $d + f > 0$, and

$\begin{pmatrix} a & b \\ b & c \end{pmatrix}$ being positive semidefinite.

This time there are only four blocks in X, and running the appropriate command generates the following csdp file:

```
4
4
2 1 1 1
3.0000 -4.0000 2.0000 0.0400
1 1 1 1 1.0
1 2 1 1 1.0
2 1 1 2 1.0
3 1 2 2 1.0
3 3 1 1 1.0
4 2 1 1 1.0
4 3 1 1 1.0
4 4 1 1 -1.0
```

B.2 The Motzkin polynomial: a less trivial example

We consider the Motzkin polynomial

$$x^4y^2 + x^2y^4 - 3x^2y^2 + 1, \quad \text{and the corresponding form} \quad x^4y^2 + x^2y^4 - 3x^2y^2z^2 + z^6\,.$$

Our goal is to confirm that the Motzkin form is nonnegative on the positive orthant. After some initial processing, we arrive at the equation

$$\begin{pmatrix} x^2y & xy^2 & xyz & z^3 \end{pmatrix} \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & j \\ d & g & l & k \end{pmatrix} \begin{pmatrix} x^2y \\ xy^2 \\ xyz \\ z^3 \end{pmatrix} + xy \begin{pmatrix} xy & z^2 \end{pmatrix} \begin{pmatrix} l & m \\ m & n \end{pmatrix} \begin{pmatrix} xy \\ z^2 \end{pmatrix} +$$

$$xz \begin{pmatrix} xy & yz \end{pmatrix} \begin{pmatrix} p & q \\ q & r \end{pmatrix} \begin{pmatrix} xy \\ yz \end{pmatrix} + yz \begin{pmatrix} xy & xz \end{pmatrix} \begin{pmatrix} s & t \\ t & u \end{pmatrix} \begin{pmatrix} xy \\ xz \end{pmatrix}$$

$$= x^4y^2 + x^2y^4 - 3x^2y^2z^2 + z^6\,, \tag{3}$$

subject to each of the square matrices in this expression being positive semidefinite. Note that this form is arrived at once the algorithm has confirmed that a large number of degree six monomials in (x, y, z) cannot figure in the solution, leaving only 10 monomials. This leads to generation of the following csdp file:

```
11
5
4 2 2 2 1
1.0000 0.0 0.0 1.0000 0.0 -3.0000 0.0 0.0 0.0 1.0000 0
1 1 1 1 1.0
2 1 1 2 1.0
2 2 1 1 1.0
```

```
3 1 1 3 1.0
3 3 1 1 1.0
4 1 2 2 1.0
5 1 2 3 1.0
5 4 1 1 1.0
6 1 3 3 1.0
6 2 1 2 1.0
6 3 1 2 1.0
6 4 1 2 1.0
7 1 1 4 1.0
7 4 2 2 1.0
8 1 2 4 1.0
8 3 2 2 1.0
9 1 3 4 1.0
9 2 2 2 1.0
10 1 4 4 1.0
11 5 1 1 -1.0
```

Ten of the constraints are for each surviving monomial, while the final constraint is a redundant constraint, generated by default.

Considerably more complicated problems arise when we allow an additional polynomial to multiply the polynomial whose positivity we wish to determine, before examining the product.