# QSMExplorer

## A package to explore one Quadrillion F-theory Standard Models

2021.11.17

17 November 2021

**Martin Bies**

**Muyang Liu**

**Martin Bies**
Email: martin.bies@alumni.uni-heidelberg.de
Homepage: https://martinbies.github.io/
Address: Department of Mathematics
University of Pennsylvania
David Rittenhouse Laboratory
209 S 33rd St
Philadelphia
PA 19104

**Muyang Liu**
Email: muyang@sas.upenn.edu
Homepage: https://github.com/lmyreg2017
Address: Department of Physics and Astronomy
University of Pennsylvania
209 South 33rd Street
Philadelphia, PA 19104-6396
United States

## Copyright

# Contents

# Chapter 1

# Introduction

## 1.1 Acknowledgements

This algorithm is the result of ongoing collaboration of Martin Bies, Mirjam Cvetič and Muyang Liu. The corresponding preprint will be available very soon.

## 1.2 What is the goal of the QSMExplorer package?

*QSMExplorer* provides functionality to easily and efficiently explore one Quadrillion F-theory Standard Models.

# Chapter 2

# Tools for investigation of one Quadrillion F-theory Standard Models

## 2.1 Find external files, scripts and binaries

### 2.1.1 FindDataBase

▷ `FindDataBase(`*none*`)` (operation)

**Returns:** the corresponding filename

This operation identifies the location of the data base in which we store essential information about Quadrillion F-theory Standard Models.

### 2.1.2 FindDualGraphScript

▷ `FindDualGraphScript(`*none*`)` (operation)

**Returns:** the corresponding filename

This operation identifies the location of a python script which plots the dual graph of the nodal quark-doublet curve.

### 2.1.3 FindRootCounterDirectory

▷ `FindRootCounterDirectory(`*none*`)` (operation)

**Returns:** the directory in which the binary is contained

This operation identifies the location of the C++-program which computes a lower bound to the number of limit roots with minimal number of global sections on a nodal curve.

### 2.1.4 FindSpecialityDirectory

▷ `FindSpecialityDirectory(`*none*`)` (operation)

**Returns:** the directory in which the binary is contained

This operation identifies the location of the C++-program which computes the speciality of a linear series.

## 2.2 Read information from database

### 2.2.1 ReadQSM (for IsInt)

▷ `ReadQSM(`*`Integer, i`*`)` (operation)

**Returns:** List of information

This function reads out the data of the i-th polytope listed in the QSM database. If the i-th polytope does not exist in the QSM database it returns fail.

### 2.2.2 ReadQSMByPolytope (for IsInt)

▷ `ReadQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** List of information of false.

This function reads out the data of the polytope with number i listed in the QSM database. If this polytope does not exist in the database, this function returns fail.

## 2.3 The 708 polytopes and their triangulations

### 2.3.1 PolytopeOfQSM (for IsInt)

▷ `PolytopeOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the polytopes whose FRSTs define the base spaces of the QSM in question.

### 2.3.2 PolytopeOfQSMByPolytope (for IsInt)

▷ `PolytopeOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the polytopes whose FRSTs define the base spaces of the QSM in question.

### 2.3.3 TriangulationEstimateInQSM (for IsInt)

▷ `TriangulationEstimateInQSM(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the triangulation estimate for a QSM.

### 2.3.4 TriangulationEstimateInQSMByPolytope (for IsInt)

▷ `TriangulationEstimateInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the triangulation estimate for a QSM.

### 2.3.5 MaxLatticePtsInFacetInQSM (for IsInt)

▷ `MaxLatticePtsInFacetInQSM(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the maximal number of lattice points in the facets of the polytope of a QSM.

### 2.3.6 MaxLatticePtsInFacetInQSMByPolytope (for IsInt)

▷ `MaxLatticePtsInFacetInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the maximal number of lattice points in the facets of the polytope of a QSM.

### 2.3.7 TriangulatonQuickForQSM (for IsInt)

▷ `TriangulatonQuickForQSM(`*`Integer, i`*`)` (operation)

**Returns:** true or false

This function returns if the triangulations can be computed in a reasonable time for a QSM.

### 2.3.8 TriangulationQuickForQSMByPolytope (for IsInt)

▷ `TriangulationQuickForQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** true or false

This function returns if the triangulations can be computed in a reasonable time for a QSM.

## 2.4 The toric 3-folds obtained from FRSTs of the 708 polytopes

### 2.4.1 BaseSpaceOfQSM (for IsInt)

▷ `BaseSpaceOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a toric variety

This function constructs a 3-fold base space for this QSM.

### 2.4.2 BaseSpaceOfQSMByPolytope (for IsInt)

▷ `BaseSpaceOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a toric variety

This function constructs a 3-fold base space for this QSM.

### 2.4.3 Kbar3OfQSM (for IsInt)

▷ `Kbar3OfQSM(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the triple intersection number Kbar^3 of the base space of this QSM.

### 2.4.4 Kbar3OfQSMByPolytope (for IsInt)

▷ `Kbar3OfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the triple intersection number Kbar^3 of the base space of this QSM.

## 2.5 The components of the nodal quark-doublet curve

### 2.5.1 GeneraOfCurvesInQSM (for IsInt)

▷ `GeneraOfCurvesInQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the genera of the non-trivial curves in a QSM.

### 2.5.2 GeneraOfCurvesInQSMByPolytope (for IsInt)

▷ `GeneraOfCurvesInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the genera of the non-trivial curves in a QSM.

### 2.5.3 DegreeOfKbarOnCurvesInQSM (for IsInt)

▷ `DegreeOfKbarOnCurvesInQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the degree of Kbar on the non-trivial curves in a QSM.

### 2.5.4 DegreeOfKbarOnCurvesInQSMByPolytope (for IsInt)

▷ `DegreeOfKbarOnCurvesInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the degree of Kbar on the non-trivial curves in a QSM.

### 2.5.5 IntersectionNumbersOfCurvesInQSM (for IsInt)

▷ `IntersectionNumbersOfCurvesInQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the non-trivial curves in a QSM.

### 2.5.6 IntersectionNumbersOfCurvesInQSMByPolytope (for IsInt)

▷ `IntersectionNumbersOfCurvesInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the non-trivial curves in a QSM.

### 2.5.7 IndicesOfTrivialCurvesInQSM (for IsInt)

▷ `IndicesOfTrivialCurvesInQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the indices of all trivial curves.

### 2.5.8 IndicesOfTrivialCurvesInQSMByPolytope (for IsInt)

▷ `IndicesOfTrivialCurvesInQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the indices of all trivial curves.

## 2.6 The dual graph of the nodal quark-doublet curve

### 2.6.1 ComponentsOfDualGraphOfQSM (for IsInt)

▷ `ComponentsOfDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of strings

This function returns the list of the names of the (non-trivial) components of the dual graph.

### 2.6.2 ComponentsOfDualGraphOfQSMByPolytope (for IsInt)

▷ `ComponentsOfDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of strings

This function returns the list of the genera of the (non-trivial) components of the dual graph.

### 2.6.3 GenusOfComponentsOfDualGraphOfQSM (for IsInt)

▷ `GenusOfComponentsOfDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the names of the (non-trivial) components of the dual graph.

### 2.6.4 GenusOfComponentsOfDualGraphOfQSMByPolytope (for IsInt)

▷ `GenusOfComponentsOfDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the genera of the (non-trivial) components of the dual graph.

### 2.6.5 DegreeOfKbarOnComponentsOfDualGraphOfQSM (for IsInt)

▷ `DegreeOfKbarOnComponentsOfDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the degrees of the anticanonical bundle on the (non-trivial) components of the dual graph.

### 2.6.6 DegreeOfKbarOnComponentsOfDualGraphOfQSMByPolytope (for IsInt)

▷ `DegreeOfKbarOnComponentsOfDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the degrees of the anticanonical bundle on the (non-trivial) components of the dual graph.

### 2.6.7 IntersectionNumberOfComponentsOfDualGraphOfQSM (for IsInt)

▷ `IntersectionNumberOfComponentsOfDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the (non-trivial) components of the dual graph.

### 2.6.8 IntersectionNumberOfComponentsOfDualGraphOfQSMByPolytope (for IsInt)

▷ `IntersectionNumberOfComponentsOfDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the (non-trivial) components of the dual graph.

### 2.6.9 DualGraphOfQSM (for IsInt)

▷ `DualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the dual graph of a QSM.

### 2.6.10 DualGraphOfQSMByPolytope (for IsInt)

▷ `DualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the dual graph of a QSM.

## 2.7 The simplified dual graph of the nodal quark-doublet curve

### 2.7.1 ComponentsOfSimplifiedDualGraphOfQSM (for IsInt)

▷ `ComponentsOfSimplifiedDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of strings

This function returns the list of the names of the (non-trivial) components of the simplified dual graph.

### 2.7.2 ComponentsOfSimplifiedDualGraphOfQSMByPolytope (for IsInt)

▷ `ComponentsOfSimplifiedDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of strings

This function returns the list of the genera of the (non-trivial) components of the simplified dual graph.

### 2.7.3 GenusOfComponentsOfSimplifiedDualGraphOfQSM (for IsInt)

▷ `GenusOfComponentsOfSimplifiedDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the names of the (non-trivial) components of the simplified dual graph.

### 2.7.4 GenusOfComponentsOfSimplifiedDualGraphOfQSMByPolytope (for IsInt)

▷ `GenusOfComponentsOfSimplifiedDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the genera of the (non-trivial) components of the simplified dual graph.

### 2.7.5 DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSM (for IsInt)

▷ `DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the degrees of the anticanonical bundle on the (non-trivial) components of the simplified dual graph.

### 2.7.6 DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSMByPolytope (for IsInt)

▷ `DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of integers

This function returns the list of the degrees of the anticanonical bundle on the (non-trivial) components of the simplified dual graph.

### 2.7.7 EdgeListOfSimplifiedDualGraphOfQSM (for IsInt)

▷ `EdgeListOfSimplifiedDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the (non-trivial) components of the simplified dual graph.

### 2.7.8 EdgeListOfSimplifiedDualGraphOfQSMByPolytope (for IsInt)

▷ `EdgeListOfSimplifiedDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function returns the intersection numbers among the (non-trivial) components of the simplified dual graph.

### 2.7.9 SimplifiedDualGraphOfQSM (for IsInt)

▷ `SimplifiedDualGraphOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the simplified dual graph of a QSM.

### 2.7.10 SimplifiedDualGraphOfQSMByPolytope (for IsInt)

▷ `SimplifiedDualGraphOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the simplified dual graph of a QSM.

### 2.7.11 SimplifiedDualGraphWithExternalLegsOfQSM (for IsInt)

▷ `SimplifiedDualGraphWithExternalLegsOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the simplified dual graph with external legs of a QSM.

### 2.7.12 SimplifiedDualGraphWithExternalLegsOfQSMByPolytope (for IsInt)

▷ `SimplifiedDualGraphWithExternalLegsOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints the simplified dual graph with external legs of a QSM.

## 2.8 The toric ambient space of the elliptic 4-fold

### 2.8.1 PF11

▷ `PF11()` (operation)

**Returns:** a toric variety

This function returns the toric variety PF11, i.e. the fibre ambient space.

### 2.8.2 ToricAmbientSpaceOfQSM (for IsInt)

▷ `ToricAmbientSpaceOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function return the ray generators of the toric ambient space 5-fold of a QSM.

### 2.8.3 ToricAmbientSpaceOfQSMByPolytope (for IsInt)

▷ `ToricAmbientSpaceOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** a list of lists of integers

This function return the ray generators of the toric ambient space 5-fold of a QSM.

## 2.9 The Picard lattice of the K3

### 2.9.1 IsK3OfQSMElliptic (for IsInt)

▷ `IsK3OfQSMElliptic(`*`Integer, i`*`)` (operation)

**Returns:** true or false

This function returns true if the K3 is elliptic and false otherwise.

### 2.9.2 IsK3OfQSMByPolytopeElliptic (for IsInt)

▷ `IsK3OfQSMByPolytopeElliptic(`*`Integer, i`*`)` (operation)

**Returns:** true or false

This function returns true if the K3 is elliptic and false otherwise.

### 2.9.3 RankOfPicardLatticeOfK3OfQSM (for IsInt)

▷ `RankOfPicardLatticeOfK3OfQSM(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the rank of the K3 in a QSM.

### 2.9.4 RankOfPicardLatticeOfK3OfQSMByPolytope (for IsInt)

▷ `RankOfPicardLatticeOfK3OfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** an integer

This function returns the rank of the K3 in a QSM.

## 2.10 Data summary of Quadrillion F-theory Standard Model build from given polytope

### 2.10.1 FullInformationOfQSM (for IsInt)

▷ `FullInformationOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints important information about the i-th QSM in our list.

### 2.10.2 FullInformationOfQSM (for IsInt, IsBool)

▷ `FullInformationOfQSM(`*`arg1, arg2`*`)` (operation)

### 2.10.3 FullInformationOfQSMByPolytope (for IsInt)

▷ `FullInformationOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** true or fail

This function prints important information about the QSM constructed from polytope i in our list.

### 2.10.4 FullInformationOfQSMByPolytope (for IsInt, IsBool)

▷ `FullInformationOfQSMByPolytope(`*`arg1, arg2`*`)` (operation)

## 2.11 Counting limit roots with minimal number of global sections

### 2.11.1 CountMinimalLimitRootsOfQSM (for IsInt)

▷ `CountMinimalLimitRootsOfQSM(`*`Integer, i`*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the i-th QSM.

### 2.11.2 CountMinimalLimitRootsOfQSM (for IsInt, IsBool)

▷ `CountMinimalLimitRootsOfQSM(`*`arg1, arg2`*`)` (operation)

### 2.11.3 CountMinimalLimitRootsOfQSMByPolytope (for IsInt)

▷ `CountMinimalLimitRootsOfQSMByPolytope(`*`Integer, i`*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the QSM defined by polytope i.

### 2.11.4 CountMinimalLimitRootsOfQSMByPolytope (for IsInt, IsBool)

▷ `CountMinimalLimitRootsOfQSMByPolytope(`*`arg1, arg2`*`)` (operation)

## 2.12 Counting distribution of limit roots

### 2.12.1 CountLimitRootDistributionOfQSM (for IsInt, IsInt, IsInt)

▷ `CountLimitRootDistributionOfQSM(`*`Integer, i, integer, L`*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the i-th QSM with at most L global sections.

### 2.12.2 CountLimitRootDistributionOfQSM (for IsInt, IsInt, IsInt, IsBool)

▷ `CountLimitRootDistributionOfQSM(`*`arg1, arg2, arg3, arg4`*`)` (operation)

### 2.12.3 CountLimitRootDistributionOfQSMByPolytope (for IsInt, IsInt, IsInt)

▷ `CountLimitRootDistributionOfQSMByPolytope(`*`Integer, i, integer, L`*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the QSM defined by polytope i with at most L global sections.

### 2.12.4 CountLimitRootDistributionOfQSMByPolytope (for IsInt, IsInt, IsInt, IsBool)

▷ `CountLimitRootDistributionOfQSMByPolytope(`*`arg1, arg2, arg3, arg4`*`)` (operation)

## 2.13 Counting distribution of limit roots with external legs

### 2.13.1 CountLimitRootDistributionWithExternalLegsOfQSM (for IsInt, IsInt, IsInt, IsList)

▷ `CountLimitRootDistributionWithExternalLegsOfQSM(`*`Integer, i, integer, L, list, w`*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the i-th QSM with at most L global sections for an assignment of weights w on the external legs.

### 2.13.2 CountLimitRootDistributionWithExternalLegsOfQSM (for IsInt, IsInt, IsInt, IsList, IsBool)

▷ `CountLimitRootDistributionWithExternalLegsOfQSM(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

### 2.13.3 CountLimitRootDistributionWithExternalLegsOfQSMByPolytope (for IsInt, IsInt, IsInt, IsList)

▷ `CountLimitRootDistributionWithExternalLegsOfQSMByPolytope(`*Integer, i, integer, L, list, w*`)` (operation)

**Returns:** integer or fail

This function computes the number of limit roots in the QSM defined by polytope i with at most L global sections for an assignment of weights w on the external legs.

### 2.13.4 CountLimitRootDistributionWithExternalLegsOfQSMByPolytope (for IsInt, IsInt, IsInt, IsList, IsBool)

▷ `CountLimitRootDistributionWithExternalLegsOfQSMByPolytope(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

## 2.14 Examples

Given the index i of polytope in the Kreuzer and Skarke list, DisplayQSMByPolytope( i ) shows important information of the QSMs built from this polytope. We mention that in particular Sage starts its iterations at 0. Thus, for example the 8-th polytope is in Sage obtained by asking for the polytope with index 7. In the QSM-Explorer, we obtain information about the QSM associated to the 8-th polytope in the Kreuzer-Skarke list as follows:

Example
```
gap> FullInformationOfQSMByPolytope( 8 );
The QSM defined by FRSTs of the 8th 3-dimensional polytope in the Kreuzer-Skar\
ke list
-------------------------------------------------------------------------------\
----------

Information on the 3-dimensional polytope:
(*) Vertices: [[-1, -1, -1], [1, -1, -1], [-1, 5, -1], [-1, -1, 5]]
(*) Maximal number of lattice points in facets: 28
(*) Estimated number of FRSTs: 25780000000000
(*) Can be computed in short time: False

Information of ONE particular 3-fold:
(*) Kbar^3: 6
(*) Number of homogeneous variables: 38
(*) Picard group: Z^35

Information about elliptic 4-fold:
```

```
(*) h11: 40
(*) h12: 16
(*) h13: 31
(*) h22: 296

Information about the K3-surface:
(*) IsElliptic: True
(*) Rank of Picard lattice: 19

Information on the nodal quark-doublet curve:
(*) Genus: 4
(*) Number of components: 22
(*) Components: ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8", "C9", "\
C14", "C15", "C19", "C20", "C23", "C24", "C26", "C27", "C28", "C29", "C32", "C\
37"]
(*) Genera: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(*) Degree of Kbar: [1, 3, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
0, 0, 0]

Information on simplified dual graph:
(*) Number of components: 4
(*) Components: ["C0", "C1", "C2", "C3"]
(*) Genera: [0, 1, 0, 0]
(*) Edge list of dual graph: [[3, 0], [2, 0], [2, 3], [0, 1], [1, 3], [1, 2]]

Root bundles:
(*) Looking for 12th root of line bundle M
(*) Degrees of line bundle M: [ 12, 36, 12, 12 ]
(*) Total number of root bundles: 429981696

true
```

Given the index i of polytope in our database.csv file, DisplayQSM( i ) shows the important information of the QSMs built from this polytope. For example, below is the important information of the QSMs from the 1st polytope in our list, which corresponds to the 3th polytope in the Kreuzer and Skarke list.

Example

```
gap> FullInformationOfQSM( 1 );

The QSM defined by FRSTs of the 4th 3-dimensional polytope in the Kreuzer-Skar\
ke list
-------------------------------------------------------------------------------\
----------

Information on the 3-dimensional polytope:
(*) Vertices: [[-1, -1, -1], [2, -1, -1], [-1, 2, -1], [-1, -1, 5]]
(*) Maximal number of lattice points in facets: 16
(*) Estimated number of FRSTs: 212533333333.333
(*) Can be computed in short time: True

Information of ONE particular 3-fold:
(*) Kbar^3: 6
(*) Number of homogeneous variables: 29
(*) Picard group: Z^26

Information about elliptic 4-fold:
(*) h11: 31
(*) h12: 10
(*) h13: 34
(*) h22: 284

Information about the K3-surface:
(*) IsElliptic: No
(*) Rank of Picard lattice: 18

Information on the nodal quark-doublet curve:
(*) Genus: 4
(*) Number of components: 21
(*) Components: ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8", "C9", "\
C13", "C14", "C16", "C17", "C21", "C24-0", "C24-1", "C25", "C27", "C28-0", "C2\
8-1"]
(*) Genera: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(*) Degree of Kbar: [1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
0, 0]

Information on simplified dual graph:
(*) Number of components: 4
(*) Components: ["C0", "C1", "C2", "C3"]
(*) Genera: [0, 0, 0, 0]
(*) Edge list of dual graph: [[3, 0], [2, 0], [2, 3], [1, 0], [1, 3], [1, 2], \
[1, 2]]

Root bundles:
(*) Looking for 12th root of line bundle M
(*) Degrees of line bundle M: [ 12, 24, 24, 12 ]
(*) Total number of root bundles: 429981696

true
```

We can display the dual graph of this polytope by issuing DualGraphOfQSM ( 702 ) and DualGraphOfQSMByPolytope( 7 ). We can compute the limit counting of selected polytope by issuing CountLimitRootsOfQSMByPolytope( i ) that corresponds to the i-th polytope in the Kreuzer and Skarke list. Below is the root counting for the 7-th polytope.

Example
```
gap> CountMinimalLimitRootOfQSMByPolytope( 8 );

Received diagram:
-----------------
Vertices: 0,1,2,3
Edge numbers: 3,3,3,3
Degrees: 12,36,12,12
Genera: 0,1,0,0
Edges: (3,0) (2,0) (2,3) (0,1) (1,3) (1,2)
Degree: 72
```

```
Root: 12
Genus: 4
h0_min: 3

Executing...
------------
Wait for 8 worker threads
Thread complete. Total roots found: 12960
Thread complete. Total roots found: 12960
Thread complete. Total roots found: 12960
Thread complete. Total roots found: 12960
Thread complete. Total roots found: 12960
Thread complete. Total roots found: 25920
Thread complete. Total roots found: 25920
Thread complete. Total roots found: 25920
Found 142560 minimal roots

Time for run: 0[s]

142560
```

Alternatively, we can compute the limit counting of selected polytope by issuing CountLimitRootsOfQSM( i ) that corresponds to the i-th polytope in our list. Below is the root counting for the 1st polytope.

Example

```
gap> CountMinimalLimitRootsOfQSM( 1 );

Received diagram:
-----------------
Vertices: 0,1,2,3
Edge numbers: 3,4,4,3
Degrees: 12,24,24,12
Genera: 0,0,0,0
Edges: (3,0) (2,0) (2,3) (1,0) (1,3) (1,2) (1,2)
Degree: 72
Root: 12
Genus: 4
h0_min: 3

Executing...
------------
Wait for 8 worker threads
Thread complete. Total roots found: 1010
Thread complete. Total roots found: 1010
Thread complete. Total roots found: 1010
Thread complete. Total roots found: 1010
Thread complete. Total roots found: 1010
Thread complete. Total roots found: 2020
Thread complete. Total roots found: 2020
Thread complete. Total roots found: 2020
Found 11110 minimal roots
```

```
Time for run: 0[s]

11110
```

Let us compute the number of minimal roots for the 8th polytope in the Kreuzer and Skarke list and the first polytope in our list. We pass as 2nd argument "false" to not display intermediate results of the C++ run.

Example
```
gap> n := CountMinimalLimitRootsOfQSMByPolytope( 8, false );;
gap> n;
[ 0, 0, 0, 142560 ]
gap> n := CountMinimalLimitRootsOfQSM( 1, false );;
gap> n;
[ 0, 0, 0, 11110 ]
```

We can also wonder how many limit roots exist with number of global sections at most L. For example, this is achieved by the following:

Example
```
gap> n1 := CountLimitRootDistributionOfQSMByPolytope( 8, 3, 4, false );;
gap> n1;
[ 0, 0, 0, 142560, 0 ]
gap> n2 := CountLimitRootDistributionOfQSM( 10, 3, 6, false );;
gap> n2;
[ 0, 0, 0, 781680888, 25196800, 106800, 0 ]
```

We can also compute the root distribution with weights on external legs of the quark doublet curve. Here is an example:

Example
```
gap> dist := CountLimitRootDistributionWithExternalLegsOfQSM( 1, 0, 4, [ 1,1,11,11,11,1 ], false
gap> dist;
[ 1716, 8325, 1260, 0, 0 ]
gap> dist2 := CountLimitRootDistributionWithExternalLegsOfQSMByPolytope( 8, 0, 4, [ 1, 11, 11, 11
[ 1573, 109395, 36036, 0, 0 ]
```

We can also perform a sufficient test to tell if the K3s are elliptic. To this end, it suffices to find an element of Pic( K3 ) with vanishing self-intersection number (i.e. is a g = 1 curve).

Example
```
gap> IsK3OfQSMByPolytopeElliptic( 8 );
true
gap> IsK3OfQSMByPolytopeElliptic( 4 );
false
gap> IsK3OfQSMElliptic( 2 );
true
gap> IsK3OfQSMElliptic( 1 );
false
```

Also, we can compute a lower bound to the rank of the Picard lattice of the K3.

Example
```
gap> RankOfPicardLatticeOfK3OfQSM( 2 );
19
gap> RankOfPicardLatticeOfK3OfQSMByPolytope( 8 );
```

```
19
gap> RankOfPicardLatticeOfK3OfQSM( 1 );
18
```

Similarly, we can easily access the triple intersection number of the anti-canonical class of the base space.

Example
```
gap> Kbar3OfQSM( 2 );
6
gap> Kbar3OfQSMByPolytope( 8 );
6
gap> Kbar3OfQSMByPolytope( 40 );
18
```

We can also plot the dual graph of the nodal quark-doublet curve. The corresponding script can be found as follows:

Example
```
gap> FindDualGraphScript();;
```

Each QSM is obtained from triangulations of certain polytopes. These polytopes we can read out:

Example
```
gap> PolytopeOfQSM( 1 );
<A polytope in |R^3>
gap> PolytopeOfQSMByPolytope( 8 );
<A polytope in |R^3>
```

An estimate (more precisely, an estimated lower bound) of the number of triangulations is distributed with the database in the QSMExplorer. We can read-out these estimates as follows:

Example
```
gap> TriangulationEstimateInQSM( 1 );
2.12533e+11
gap> TriangulationEstimateInQSMByPolytope( 8 );
2578000000000
```

These triangulations are obtained by triangulating facets of the 3d polytopes and then pieces these triangulations together to triangulations of the polytope. Whether or not this is possible, depends on how complicated the facets of the polytope in question are. The latter is measured by the lattice points in the facets. The number of lattice points for the largest facet of the polytope for a QSM can be read-off from our database:

Example
```
gap> MaxLatticePtsInFacetInQSM( 1 );
16
gap> MaxLatticePtsInFacetInQSMByPolytope( 8 );
28
```

We provide a pointer if or if not all these triangulations can be computed in a reasonable time (as used in the physics community):

Example
```
gap> TriangulatonQuickForQSM( 1 );
true
gap> TriangulationQuickForQSMByPolytope( 8 );
false
```

Each of these triangulations gives rise to a toric 3-fold base space. We provide a method that constructs one of these many bases.

Example
```
gap> BaseSpaceOfQSM( 1 );
<A toric variety of dimension 3>
gap> BaseSpaceOfQSMByPolytope( 8 );
<A toric variety of dimension 3>
```

For this very base space, we can also construct the toric 5-fold ambient space of the F-theory elliptic 4-fold:

Example
```
gap> ToricAmbientSpaceOfQSM( 1 );
<A toric variety of dimension 5>
gap> ToricAmbientSpaceOfQSMByPolytope( 8 );
<A toric variety of dimension 5>
```

Explicitly, we can read-out specific data from the QSMs with the following methods:

Example
```
gap> GeneraOfCurvesInQSM( 1 );;
gap> GeneraOfCurvesInQSMByPolytope( 8 );;
gap> DegreeOfKbarOnCurvesInQSM( 1 );;
gap> DegreeOfKbarOnCurvesInQSMByPolytope( 8 );;
gap> IntersectionNumbersOfCurvesInQSM( 1 );;
gap> IntersectionNumbersOfCurvesInQSMByPolytope( 8 );;
gap> IndicesOfTrivialCurvesInQSM( 1 );;
gap> IndicesOfTrivialCurvesInQSMByPolytope( 8 );;
gap> ComponentsOfDualGraphOfQSM( 1 );;
gap> ComponentsOfDualGraphOfQSMByPolytope( 8 );;
gap> GenusOfComponentsOfDualGraphOfQSM( 1 );;
gap> GenusOfComponentsOfDualGraphOfQSMByPolytope( 8 );;
gap> DegreeOfKbarOnComponentsOfDualGraphOfQSM( 1 );;
gap> DegreeOfKbarOnComponentsOfDualGraphOfQSMByPolytope( 8 );;
gap> IntersectionNumberOfComponentsOfDualGraphOfQSM( 1 );;
gap> IntersectionNumberOfComponentsOfDualGraphOfQSMByPolytope( 8 );;
gap> ComponentsOfSimplifiedDualGraphOfQSM( 1 );;
gap> ComponentsOfSimplifiedDualGraphOfQSMByPolytope( 8 );;
gap> GenusOfComponentsOfSimplifiedDualGraphOfQSM( 1 );;
gap> GenusOfComponentsOfSimplifiedDualGraphOfQSMByPolytope( 8 );;
gap> DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSM( 1 );;
gap> DegreeOfKbarOnComponentsOfSimplifiedDualGraphOfQSMByPolytope( 8 );;
gap> EdgeListOfSimplifiedDualGraphOfQSM( 1 );;
gap> EdgeListOfSimplifiedDualGraphOfQSMByPolytope( 8 );;
```

We can plot the (simplified) dual graphs as follows:

Example
```
gap> DualGraphOfQSM( 1 );;
gap> DualGraphOfQSMByPolytope( 8 );;
gap> SimplifiedDualGraphOfQSM( 1 );;
gap> SimplifiedDualGraphOfQSMByPolytope( 8 );;
```

With the nodal Higgs curve in mind, we can also consider the nodal quark-doublet curve with external legs:

Example
```
gap> SimplifiedDualGraphWithExternalLegsOfQSM( 1 );;
gap> SimplifiedDualGraphWithExternalLegsOfQSMByPolytope( 8 );;
```

We verify that the results computed in our PRD letter for the spaces with Kbar^3 = 6 are still reproduced by this software:

Example
```
gap> CountMinimalLimitRootsOfQSMByPolytope( 8, false );
[ 0, 0, 0, 142560 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 4, false );
[ 0, 0, 0, 11110 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 134, false );
[ 0, 0, 0, 10010 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 128, false );
[ 0, 0, 0, 8910 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 130, false );
[ 0, 0, 0, 8910 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 136, false );
[ 0, 0, 0, 8910 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 236, false );
[ 0, 0, 0, 8910 ]
```

Finally, we also verify that the results computed in our PRD letter for some spaces with Kbar^3 = 10 are still reproduced by this software:

Example
```
gap> CountMinimalLimitRootsOfQSMByPolytope( 88, false );
[ 0, 0, 0, 781680888 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 110, false );
[ 0, 0, 0, 738662983 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 272, false );
[ 0, 0, 0, 736011640 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 274, false );
[ 0, 0, 0, 736011640 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 387, false );
[ 0, 0, 0, 733798300 ]
```

The remaining results can be checked analogousyl as follows:

Example
```
gap> CountMinimalLimitRootsOfQSMByPolytope( 798, false );
[ 0, 0, 0, 690950608 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 808, false );
[ 0, 0, 0, 690950608 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 810, false );
[ 0, 0, 0, 690950608 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 812, false );
[ 0, 0, 0, 690950608 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 254, false );
[ 0, 0, 0, 35004914 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 52, false );
[ 0, 0, 0, 34980351 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 302, false );
[ 0, 0, 0, 34908682 ]
```

```
gap> CountMinimalLimitRootsOfQSMByPolytope( 786, false );
[ 0, 0, 0, 32860461 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 762, false );
[ 0, 0, 0, 32858151 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 417, false );
[ 0, 0, 0, 32857596 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 838, false );
[ 0, 0, 0, 32845047 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 782, false );
[ 0, 0, 0, 32844379 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 377, false );
[ 0, 0, 0, 30846440 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 499, false );
[ 0, 0, 0, 30846440 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 503, false );
[ 0, 0, 0, 30846440 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 1348, false );
[ 0, 0, 0, 30845702 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 882, false );
[ 0, 0, 0, 30840098 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 1340, false );
[ 0, 0, 0, 28954543 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 1879, false );
[ 0, 0, 0, 28950852 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 1384, false );
[ 0, 0, 0, 27178020 ]
gap> CountMinimalLimitRootsOfQSMByPolytope( 856, false );
[ 0, 0, 0, 30840098 ]
```

# Chapter 3

# Check speciality of line bundles

## 3.1 Determine if line bundle on nodal curve is special.

### 3.1.1 Speciality (for IsList, IsList)

▷ `Speciality(D, L)` (operation)

**Returns:** a bool

This operation constructs a simple nodal curve $C$ from a list of edges $E$ and a line bundle $L$ from its list of degrees $D$ on it. Provided that $C$ is tree-like, this method determines if $(C,L)$ is special. This algorithm has been formulated and proven by Prof. Dr. Ron Donagi (University of Pennsylvania). The current implementation performs no checks to tell if $C$ is indeed tree-like. It is the responsibility of the user to ensure that this is the case. An optional 3rd argument can be provided to supress intermediate output.

### 3.1.2 Speciality (for IsList, IsList, IsBool)

▷ `Speciality(arg1, arg2, arg3)` (operation)

## 3.2 Examples

A line bundle $L$ on a tree-like, rational, nodal curve $C$ is special iff its cohomologies jump under deformation of $C$. This algorithm, invented by Prof. Dr. Ron Donagi (University of Pennsylvania) decides if $(L,C)$ is special. Here are examples.

Example
```
gap> s1 := Speciality( [ 1, 1, 1, -1, -1, -1, -1 ], [ [0,1], [1,2], [0,3], [0,4], [1,5], [1,6] ]

This method works ONLY for tree-like curves.
NO CHECK FOR BEING TREE-LIKE IS CURRENTLY CONDUCTED. THE USER IS RESPONSIBLE
FOR PROVIDING VALID INPUT.

This algorithm was first formulated and its accuracy proven by Prof. Dr. Ron Donagi.


Result
```

```
(C,L) is SPECIAL.

true
```

We can decide to not display all this output by passing "false" as third argument:

Example
```
gap> s1 := Speciality( [ 1, 1, 1, -1, -1, -1, -1 ], [ [0,1], [1,2], [0,3], [0,4], [1,5], [1,6] ],
gap> s1;
true
gap> degrees := [ 1,1,1,1,2,-1,-1,-1,-1,-1,-1 ];;
gap> edges := [ [0,1],[1,2],[2,3],[3,4],[0,5],[1,6],[1,7],[2,8],[3,9],[3,10] ];;
gap> s2 := Speciality( degrees, edges, false );;
gap> s2;
true
gap> s3 := Speciality( [ 0, -5 ], [ [0,1] ], false );;
gap> s3;
false
gap> s4 := Speciality( [ 1, -5 ], [ [0,1] ], false );;
gap> s4;
true
gap> edges := [ [0,1], [1,2], [3,0], [4,0], [5,1], [6,1] ];;
gap> degrees := [ 1, 1, 1, -1, -1, -1, -1 ];;
gap> s5 := Speciality( degrees, edges, false );;
gap> s5;
true
gap> edges := [ [0,1], [1,2], [3,0], [4,0], [5,1], [6,1] ];;
gap> degrees := [2,1,1,-1,-1,-1,-1];;
gap> s6 := Speciality( degrees, edges, false );;
gap> s6;
false
gap> edges := [ [0,1], [1,2], [3,0], [4,0], [5,1], [6,1] ];;
gap> degrees := [ 1, 1, 0, -1, -1, -1, -1 ];;
gap> s7 := Speciality( degrees, edges, false );;
gap> s7;
false
gap> edges := [ [0,1], [1,2], [3,0], [4,0], [5,1], [6,1] ];;
gap> degrees := [ 1, 1, -1, -1, -1, -1, -1 ];;
gap> s8 := Speciality( degrees, edges, false );;
gap> s8;
false
gap> degrees := [-1, -5, 3, -2, 3];;
gap> edges := [[0, 1], [1, 2], [2, 3], [3, 4]];;
gap> s9 := Speciality( degrees, edges, false );;
gap> s9;
true
```

Chapter 4

# Tools for investigation of the Higgs and RDQ curve in one Quadrillion F-theory Standard Models

## 4.1 Find root distribution on Higgs curve

### 4.1.1 LimitRootDistributionForHiggsCurveInQSM (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSM()` (operation)

**Returns:** a list

This operation returns a list corresponding to the limit root distribution on the Higgs curve in the i-th QSM.

### 4.1.2 LimitRootDistributionForHiggsCurveInQSM (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSM(`*arg1, arg2, arg3*`)` (operation)

### 4.1.3 LimitRootDistributionForHiggsCurveInQSMDivideStep (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSMDivideStep()` (operation)

**Returns:** a list

This operation performs the divide step for the computation of limit root distribution on the Higgs curve in the i-th QSM.

### 4.1.4 LimitRootDistributionForHiggsCurveInQSMDivideStep (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSMDivideStep(`*arg1, arg2, arg3*`)` (operation)

### 4.1.5 LimitRootDistributionForHiggsCurveInQSMConquerStep (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSMConquerStep()` (operation)

**Returns:** a list

This operation performs the conquer step for the computation of limit root distribution on the Higgs curve in the i-th QSM.

### 4.1.6 LimitRootDistributionForHiggsCurveInQSMConquerStep (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSMConquerStep(`*arg1*`, `*arg2*`, `*arg3*`)` (operation)

### 4.1.7 LimitRootDistributionForHiggsCurveInQSMByPolytope (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytope()` (operation)

**Returns:** a list

This operation returns a list corresponding to the limit root distribution on the Higgs curve in the QSM of the i-th polytope.

### 4.1.8 LimitRootDistributionForHiggsCurveInQSMByPolytope (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytope(`*arg1*`, `*arg2*`, `*arg3*`)` (operation)

### 4.1.9 LimitRootDistributionForHiggsCurveInQSMByPolytopeDivideStep (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytopeDivideStep()` (operation)

**Returns:** a list

This operation performs the divide step for the computation of limit root distribution on the Higgs curve in the QSM of the i-th polytope.

### 4.1.10 LimitRootDistributionForHiggsCurveInQSMByPolytopeDivideStep (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytopeDivideStep(`*arg1*`, `*arg2*`, `*arg3*`)` (operation)

### 4.1.11 LimitRootDistributionForHiggsCurveInQSMByPolytopeConquerStep (for IsInt, IsInt)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytopeConquerStep()` (operation)

**Returns:** a list

This operation performs the conquer step for the computation of limit root distribution on the Higgs curve in the QSM of the i-th polytope.

#### 4.1.12 LimitRootDistributionForHiggsCurveInQSMByPolytopeConquerStep (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForHiggsCurveInQSMByPolytopeConquerStep(`*`arg1, arg2, arg3`*`)` (operation)

#### 4.1.13 LimitRootDistributionForRDQCurveInQSM (for IsInt, IsInt)

▷ `LimitRootDistributionForRDQCurveInQSM()` (operation)

**Returns:** a list

This operation returns a list corresponding to the limit root distribution on the Higgs curve in the i-th QSM.

#### 4.1.14 LimitRootDistributionForRDQCurveInQSM (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForRDQCurveInQSM(`*`arg1, arg2, arg3`*`)` (operation)

#### 4.1.15 LimitRootDistributionForRDQCurveInQSMByPolytope (for IsInt, IsInt)

▷ `LimitRootDistributionForRDQCurveInQSMByPolytope()` (operation)

**Returns:** a list

This operation returns a list corresponding to the limit root distribution on the Higgs curve in the QSM of the i-th polytope.

#### 4.1.16 LimitRootDistributionForRDQCurveInQSMByPolytope (for IsInt, IsInt, IsBool)

▷ `LimitRootDistributionForRDQCurveInQSMByPolytope(`*`arg1, arg2, arg3`*`)` (operation)

## 4.2 Save all outfluxes

### 4.2.1 AllOutfluxesForHiggsCurveInQSMToFile (for IsInt, IsInt)

▷ `AllOutfluxesForHiggsCurveInQSMToFile()` (operation)

**Returns:** true on success and errors otherwise.

This operation saves all outfluxes in a file.

### 4.2.2 AllOutfluxesForHiggsCurveInQSMByPolytopeToFile (for IsInt, IsInt)

▷ `AllOutfluxesForHiggsCurveInQSMByPolytopeToFile()` (operation)

**Returns:** true on success and errors otherwise.

This operation saves all outfluxes in a file.

### 4.2.3 WriteOutfuxes (for IsRecord, IsInt)

▷ WriteOutfuxes(*arg1*, *arg2*) (operation)

## 4.3 Examples

We can compute the root distribution also on the Higgs curve. These computations typically require an extended amount of computational power. It would proceed as follows:

Example
```
gap> LimitRootDistributionForHiggsCurveInQSM( 1, 4 );
```

Alternatively, we can also decide to not display intermediate output from the C++ runs:

Example
```
gap> LimitRootDistributionForHiggsCurveInQSM( 1, 4, false );
```

Similarly, we can compute the limit root distribution for the RDQ-curve, which is by construction identical to the distribution on the Higgs curve.

Example
```
gap> LimitRootDistributionForRDQCurveInQSM( 1, 4 );;
gap> LimitRootDistributionForRDQCurveInQSM( 1, 4, false );
```

The limit root distributions on the Higgs curve are typically very computationally intensive. However, we can also employ this philosophy for simpler setups. Here are two examples along these lines:

Example
```
gap> genera := [ 0,0 ];;
gap> degrees_H1 := [ 4, 4 ];;
gap> degrees_H2 := [ 0, 0 ];;
gap> edges := [ [ 0, 1 ], [ 0, 1 ] ];;
gap> total_genus := 1;;
gap> root := 2;;
gap> external_legs := [ 2, 2 ];;
gap> number_processes := 2;;
gap> h0Max := 10;;
gap> data := [ genera, degrees_H1, degrees_H2, edges, total_genus, root, external_legs, number_pr
```

[ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ]

Example
```
gap> genera := [ 0, 0, 0, 0, 0 ];;
gap> degrees_H1 := [ 4, 4, 4, 4, 4 ];;
gap> degrees_H2 := [ 0, 0, 0, 0, 0 ];;
gap> edges := [ [ 0, 1 ], [ 0, 1 ], [ 1, 2 ], [ 1, 2 ], [ 1, 3 ], [ 1, 3 ], [ 0, 3 ], [ 0, 3 ], [
gap> total_genus := 6;;
gap> root := 4;;
gap> external_legs := [ 4, 2, 2, 2, 4 ];;
gap> number_processes := 2;;
gap> h0Max := 10;;
gap> data := [ genera, degrees_H1, degrees_H2, edges, total_genus, root, external_legs, number_pr
```

[ 8374246311441809, 852982581711208, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] We can also first perform the divide step and then the conquer step.

Example
```
gap> genera := [ 0, 0, 0, 0, 0 ];;
gap> degrees_H1 := [ 4, 4, 4, 4, 4 ];;
gap> degrees_H2 := [ 0, 0, 0, 0, 0 ];;
gap> edges := [ [ 0, 1 ], [ 0, 1 ], [ 1, 2 ], [ 1, 2 ], [ 1, 3 ], [ 1, 3 ], [ 0, 3 ], [ 0, 3 ], [
gap> total_genus := 6;;
gap> root := 4;;
gap> external_legs := [ 4, 2, 2, 2, 4 ];;
gap> number_processes := 2;;
gap> h0Max := 10;;
gap> data := [ genera, degrees_H1, degrees_H2, edges, total_genus, root, external_legs, number_pr
```

[ 8374246311441809, 852982581711208, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]

# Chapter 5

# Computation of minimal roots and their distribution for arbitrary nodal curves.

## 5.1 Computation of minimal roots

### 5.1.1 CountMinimals (for IsList, IsList, IsList, IsInt, IsInt)

▷ `CountMinimals(`*none*`)` (operation)

**Returns:** A list

This operation constructs a nodal curve based on the user input, which consists of a list of the genera, a list of the line bundle degrees, the list of the edges, the total genus and the radicant (i.e. telling us which roots to consider - 2 for 2nd roots, 3 for 3rd roots etc.).

### 5.1.2 CountMinimals (for IsList, IsList, IsList, IsInt, IsInt, IsBool)

▷ `CountMinimals(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

## 5.2 Computation of distributions

### 5.2.1 CountDistribution (for IsList)

▷ `CountDistribution(`*none*`)` (operation)

**Returns:** A list

This operation constructs a nodal curve based on the user input, which consists of a list of the genera, the list of the line bundle degrees, a list of the edges, the total genus, the radicant (i.e. telling us which roots to consider - 2 for 2nd roots, 3 for 3rd roots etc.) and finally an integer *L*, telling us to focus on limit roots with number of global sections not larger than *L*.

### 5.2.2 CountDistribution (for IsList, IsBool)

▷ `CountDistribution(`*arg1, arg2*`)` (operation)

## 5.3 Examples

We can compute the number of minimal weights and their distributions efficiently. Say for example that we have two genus $g = 0$ curves joined by two edges. On both curves, the line bundle has degree 4 and we want 2 roots. Then we count the minimal roots as follows:

Example
```
gap> n := CountMinimals( [ 0, 0 ], [ 4, 4 ], [ [ 0, 1 ], [ 0, 1 ] ], 1, 2, false );;
gap> n;
[ 0, 0, 0, 0, 1 ]
```

Likewise, we can compute the distribution. For this, we need so set also a limit $L$, so that we only count limit roots with at most $L$ global sections. Hence, we modify the above code as follows to count up to $h^0 = 20$:

Example
```
gap> n := CountDistribution( [ [ 0, 0 ], [ 4, 4 ], [ [ 0, 1 ], [ 0, 1 ] ], 1, 2, 4, 6 ], false );
gap> n;
[ 0, 0, 0, 0, 1, 0, 0 ]
```

Here are more examples:

Example
```
gap> n1 := CountDistribution( [ [ 0,0,0 ], [ 16, 16, 16 ], [ [0,1], [0,1], [0,1], [0,1], [0,2], [
gap> n1;
[ 0, 0, 0, 2030, 70, 0, 0, 0, 0 ]
gap> n2 := CountDistribution( [ [ 0,0,0,0 ], [ 16, 16, 16, 16 ], [ [0,1], [0,1], [0,1], [0,1], [0
gap> n2;
[ 0, 0, 0, 0, 9331, 5334, 42, 0, 0 ]
gap> n3 := CountDistribution( [ [ 0,0,0 ], [ 36, 8, 36 ], [ [0,1], [0,1], [0,1], [0,1], [0,2], [1
gap> n3;
[ 0, 0, 8080, 28631, 7552, 34, 0, 0, 0 ]
gap> n4 := CountDistribution( [ [ 0,0,0 ], [ 34, 8, 38 ], [ [0,1], [0,1], [0,1], [0,1], [0,2] ],
gap> n4;
[ 0, 0, 0, 1547, 1582, 35, 0, 0, 0 ]
```

# Chapter 6

# Computation of minimal roots and their distribution for arbitrary nodal curves with external legs.

## 6.1 Computation of root distributions on nodal curves with external legs.

### 6.1.1 CountDistributionWithExternalLegs (for IsList)

▷ `CountDistributionWithExternalLegs(`*none*`)` (operation)

**Returns:** A list

This operation constructs a nodal curve based on the user input, which consists of a list of the genera, the list of the line bundle degrees, a list of the edges, the total genus, the radicant (i.e. telling us which roots to consider - 2 for 2nd roots, 3 for 3rd roots etc.) and an integer *L*, telling us to focus on limit roots with number of global sections not larger than *L*. In addition, we provide a list which specifies the external legs and a second list which provides the weights on these external legs. We collect all of this data in a single list. Then this function cmputes the distribution of these roots on this nodal curve with external legs.

### 6.1.2 CountDistributionWithExternalLegs (for IsList, IsBool)

▷ `CountDistributionWithExternalLegs(`*arg1*, *arg2*`)` (operation)

## 6.2 Examples

You can also specify external legs and counts the roots on a nodal curve subject to these external legs. The following code executes this computation for a simple example:

Example
```
gap> CountDistributionWithExternalLegs( [ [ 0,0,0 ], [ 16, 16, 16 ], [ [0,1], [1,2], [2,0] ], 1,
```

Example
```
gap> n := CountDistributionWithExternalLegs( [ [ 0,0,0,1,0 ], [ 16, 16, 16, 16, 0 ], [ [0,1], [1,
gap> n;
```

```
  [ 0, 0, 0, 0, 13652, 64, 0, 0, 0 ]
```

# Index