City Research Online

City St George's, University of London

Citation: Gray, J., He, Y., Ilderton, A. & Lukas, A. (2009). STRINGVACUA: A Mathematica Package for Studying Vacuum Configurations in String Phenomenology. Computer Physics Communications Package, 180(1), pp. 107-119. doi: 10.1016/j.cpc.2008.08.009

This is the unspecified version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/875/

Link to published version: https://doi.org/10.1016/j.cpc.2008.08.009

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

arXiv:0801.1508 [hep-th]

STRINGVACUA

A *Mathematica* Package for Studying Vacuum Configurations in String Phenomenology

James Gray[1,2*], Yang-Hui He[2,3♯], Anton Ilderton[4♭] and André Lukas[2†]

[1]Institut d'Astrophysique de Paris and APC, Université de Paris 7,
98 bis, Bd. Arago 75014, Paris, France

[2]Rudolf Peierls Centre for Theoretical Physics, University of Oxford,
1 Keble Road, Oxford OX1 3NP, UK

[3]Merton College, Oxford, OX1 4JD and Mathematical Institute, Oxford University,
24-29 St. Giles', Oxford OX1 3LB, UK

[4]School of Mathematics and Statistics, University of Plymouth,
Drake Circus, Plymouth PL4 8AA, UK

arXiv:0801.1508v1 [hep-th] 9 Jan 2008

Abstract

We give a simple tutorial introduction to the *Mathematica* package STRINGVACUA, which is designed to find vacua of string-derived or inspired four-dimensional $N = 1$ supergravities. The package uses powerful algebro-geometric methods, as implemented in the free computer algebra system *Singular*, but requires no knowledge of the mathematics upon which it is based. A series of easy-to-use *Mathematica* modules are provided which can be used both in string theory and in more general applications requiring fast polynomial computations. The use of these modules is illustrated throughout with simple examples.

*email: j.gray1@physics.ox.ac.uk
♯email: hey@maths.ox.ac.uk
♭email: abilderton@plymouth.ac.uk
†email: lukas@physics.ox.ac.uk

Contents

1 Introduction **1**

2 Getting started: a simple supergravity model **2**
2.1 A quick first look: SimpleMinimise 2
2.2 A little more detail: sugra models in STRINGVACUA 4

3 Filtering and finding properties of vacua **6**
3.1 Filtering 6
3.2 Finding properties of vacua 9

4 Constraints on parameters **11**

5 Non-perturbative effects **13**
5.1 Creating non-perturbative models 13
5.2 Finding vacua 14
5.3 Constraints in non-perturbative models 15

6 STRINGVACUA: other applications **17**

7 Summary **18**

A Glossary – supergravity model list keywords **19**
A.1 CreateModel 19
A.2 CalcModel 20

1 Introduction

The systematic investigation of the turning points of a given potential is one of the most fundamental problems in physics. In particular, much of the current effort in making string theory reproduce the real world is concentrated in the study of effective theories with potentials capable of stabilising all of the moduli. The theoretical framework for deriving the potential given the particular scenario of compactification – be it heterotic or type II, with or without fluxes – has been considerably developed. However, the analysis of the resulting effective theory, due to the typically complicated forms of the (super-) potential, is often computationally insurmountable.

In [1, 2] a novel, algorithmic approach to this problem was presented. The key realisation of these papers was that the typical potential is a ratio of, albeit complicated, polynomials in the field variables (in the non-perturbative cases, dummy variables may be used as substitutions for exponential terms, recovering polynomial expressions). The most efficient method of studying systems of polynomials is *computational algebraic geometry*.

Recent advances in computer algebra, in particular the advent of such packages as *Macaulay 2* [3] and *Singular* [4], have rendered the subject of computational algebraic geometry an increasingly significant tool in string theory [5, 6, 7]. As outlined in [1], the starting point for

our analysis is a (string-derived or inspired) $\mathcal{N} = 1$ supergravity theory with complex chiral superfields ϕ_i, given in terms of the associated Kähler potential $K \equiv K(\phi, \bar{\phi})$ and superpotential $W \equiv W(\phi)$. The scalar potential V of such a theory is given by the well-known formula,

$$V = \exp(K/M^2) \left(K^{ij} F_i \bar{F}_j - \frac{3}{M^2} |W|^2 \right) . \tag{1}$$

Here $F_i \equiv \partial_i W + M^{-2} \partial_i K W$ are the F-terms, $K_{ij} = \frac{\partial^2 K}{\partial \phi_i \partial \bar{\phi}_j}$ is the Kähler metric, K^{ij} its inverse and M is the Planck mass. Our primary interest is in analysing the vacuum properties of this potential.

One tool which computational algebraic geometry provides us with to help in this analysis is the so-called saturation expansion [1]. This method takes the complicated polynomial equations describing the extrema of the potential of a system, and breaks them up into a series of simpler systems, each describing one particular locus of turning points. Further algorithmic methods then make it possible to single out extrema of interest and to extract the physical properties of the associated vacua. Another tool, called elimination, provides constraints on the parameters of a model which must be satisfied for vacua of given types to exist.

The computer package *Mathematica* has become an indispensable tool in modern theoretical physics and most researchers are familiar with its syntax. The algebraic geometry system *Singular* is perhaps less known to the community interested in effective supergravity theories. With this in mind, we have written a package for *Mathematica*, called STRINGVACUA, which externally calls *Singular* where appropriate while sustaining the user in the comfortable environment of *Mathematica*. A range of *Mathematica* modules, from low to high level, provide both ease of use in string theory calculations as well as flexibility for a wider range of applications. In particular, the tools described in the previous paragraph are implemented in module form - allowing the user to access them with a few simple *Mathematica* commands.

The methods of STRINGVACUA are of course far from restricted in use to problems in string phenomenology. Many of the modules will be of use in any problem involving polynomial computations. Another *Mathematica* package which makes use of *Singular* is [8]. In the ensuing pages we shall walk the reader through a quick and illustrative tutorial of how to use the basic commands of the package. The appendix contains a glossary of terms.

Throughout this tutorial text in Serif font indicates a function or object defined in *Mathematica*, while commands to be entered by the user into a *Mathematica* notebook are `typeset as so`.

2 Getting started: a simple supergravity model

2.1 A quick first look: SimpleMinimise

Installation of the package is easy. Instructions, along with the download of the package itself, can be found at the following URL.

http://www-thphys.physics.ox.ac.uk/user/Stringvacua

In this tutorial, we describe use of the package in a Unix environment, focussing on the *Mathematica* notebook interface although the package also works with the text interface or

running in emacs. The only significant differences from a Windows environment are during installation, for further details see the URL given above. Once you have the package installed start up X11, if it is not already running, and then to load the package simply evaluate (note the single open quote!)

```
<<Stringvacua`
```

A package header and links to examples and the browser help pages will be displayed, and the package is ready for calculations. For illustration, let us start with a very simple example based on one modulus $T = t + i\tau$, with Kähler potential and superpotential given by

$$K = -3\log(T + \bar{T}), \quad W = a + bT + cT^2 , \tag{2}$$

respectively. Here, a, b and c are (real) parameters.

We will first study the vacuum structure of this theory using the SimpleMinimise command, which is designed to provide an elementary illustration of the package's use and to familiarise the user with its output. Evaluating the command

```
SimpleMinimise[]
```

will result in the appearance of a dialogue box saying List of complex fields? (example: {T}). In our case our complex field is indeed T, thus we simply type `{T}` and click OK. The next two dialogue boxes ask for the real and imaginary parts of the complex fields. If we want these to be given by $T = t + i\tau$ then we enter `{t}` followed by `{\[Tau]}` (using the *Mathematica* syntax). We are then prompted to provide the Kähler potential, `-3 Log[T+Conjugate[T]]`, followed by the superpotential, which for our example is `a + b T + c T^2` and then the parameters in the model, `{a,b,c}`. Finally (in the unix release) the package asks how much time it is allowed to spend on each calculation that it will perform in *Singular*. Let us set this to 20 (seconds) for this example, by entering `20`.

This data having been entered, the package then works out everything else automatically. An xterm will appear on your screen (by default in the upper right corner) to which *Singular* writes its protocol. The details of this protocol should not concern us for now: we will simply use its presence as an indication that something is "happening" in our calculation. The final output looks as pictured in figure 1.

Let us discuss the interpretation of this result. Clearly the first few rows of the first table, labelled "Model", simply repeat the data defining the model which we have input. The last entry in this table is the scalar potential of the system, which the package has calculated using the standard $\mathcal{N} = 1$ four dimensional supergravity formula (1).

The second of the two tables contains the results. Each row represents an irreducible component of the vacuum manifold. Starting with the first row, the entry {} in the first column indicates that this component satisfies $F_t \equiv \mathrm{Re}(F_T) = 0$ and $F_\tau \equiv \mathrm{Im}(F_T) = 0$ (in notation we will use throughout) and is therefore a supersymmetric vacuum. The field values for this vacuum are obtained by setting the polynomials in the column entitled "0-dim. ideals" to zero, so for the first row they are found by solving $ct^2 - bt - 3a = 0$ and $\tau = 0$.

In[2]:= **SimpleMinimise[]**

Model

complex fields	{T}
real fields	{t}
imaginary fields	{tau}
Kahler potential	$-3\,\mathrm{Log}[T + \mathrm{Conjugate}[T]]$
superpotential	$a + b\,T + c\,T^2$
scalar potential	$-\frac{2\,b^2\,t+3\,a\,(b+2\,c\,t)+2\,c^2\,t\,(t^2+\mathrm{tau}^2)+b\,c\,(5\,t^2+3\,\mathrm{tau}^2)}{6\,t^2}$

Result

non-zero F terms	0-dim. ideals
{ }	$\{\mathrm{tau},\ -3\,a - b\,t + c\,t^2\}$
{ }	$\{3\,b + 2\,c\,t,\ 5\,b^2 - 4\,a\,c + 4\,c^2\,\mathrm{tau}^2\}$
{t}	{tau, −b − c t}

Figure 1: An example of the output of SimpleMinimise.

Similarly, the second row corresponds to another supersymmetric vacuum. The first entry in the last row, $\{t\}$, indicates that this extremum of the potential has $F_t \neq 0$ and all other F-terms zero (here, simply $F_\tau = 0$) and, hence, breaks supersymmetry in the t direction. SimpleMinimise only provides isolated turning points of the potential in field space and so all three components correspond to isolated extrema (as opposed to vacua with flat directions). The absence of a row starting with $\{t, \tau\}$ shows that this model has no vacuum where supersymmetry is broken in the directions of both t and τ.

If no warning messages are given by the package, as is the case in this example, then SimpleMinimise returns *all* of the isolated turning points of the potential which are present for generic parameter values. Vacua which appear only for non-generic values of the parameters will be discussed in section 4.

2.2 A little more detail: sugra models in STRINGVACUA

For more serious calculations SimpleMinimise is not the most flexible or convenient way in which to perform such a decomposition of the vacuum space in STRINGVACUA. Let us repeat the above calculation, therefore, using some mainstream package modules.

In this approach, we first need to define our supergravity model within *Mathematica*. This is easily done using the module CreateModel. For the above case the appropriate command reads

```
SingleTmodel = CreateModel[{T},{t},{\[Tau]},-3 Log[T + Conjugate[T]],
  a + b T + c T^2, Par->{a,b,c}];
```

In general, this module has five obligatory arguments which are, in order, the list of complex fields, the list of the real parts of these fields, the list of imaginary parts, the Kähler potential and the superpotential. In addition, we have used the option Par to tell the package about the model parameters a, b, c. The output of CreateModel is a *model list* of *Mathematica* assignments,

here called SingleTmodel, of the form keyword → value. Each keyword in the model list contains a piece of information about the supergravity model. Examples of model keywords include Fields for the list of complex fields, KPot for the Kähler potential, SPot for the superpotential and Pot for the scalar potential. For example, the potential for the above model can be obtained by evaluating `Pot/.SingleTmodel` which returns

$$-\frac{2b^2t+3a(b+2ct)+2c^2t(t^2+\tau^2)+bc(5t^2+3\tau^2)}{6t^2}\,. \tag{3}$$

This is indeed the bosonic potential of the model (2), calculated using (1). After evaluating CreateModel the model list contains all of the information about the supergravity model which is relevant for an examination of its vacuum structure.

Hidden from the user, the package uses techniques from algorithmic algebraic geometry in order to decompose the vacuum space and describe its properties [1, 2]. This requires that we process the equations of the supergravity model into a particular form. The required operations are performed by the *Mathematica* module CalcModel.

```
SingleTmodel = CalcModel[SingleTmodel];
```

After executing CreateModel and CalcModel a model list contains over 20 pieces of data, each referred to by its keyword, and has essentially all of the information one would want for a further analysis of the model. A complete list of the physical content of a model list is given in Appendix A.

As mentioned earlier, one of the key applications of STRINGVACUA is to break up the equations describing the vacua of a model into smaller pieces – giving one system of equations for each type of vacuum. Let us see how such an analysis is performed given the model list we have just created. We simply evaluate

```
satex = SaturationExpand[SingleTmodel]
```

Once this command has been executed information will start to scroll past in the *Singular* window. While this happens a new *Mathematica* protocol notebook appears. Within this is printed the data shown in figure 2. The terms in the second line of this notebook appear

```
Calculate 4 saturation terms :

{1, {}, "ok"} {2, {t}, "ok"} {3, {τ}, "ok"} {4, {t, τ}, "ok"}

Finished !
```

Figure 2: An example of a STRINGVACUA protocol notebook.

successively until the calculation is finished. Typically after a few seconds, the final result is returned:

$$\begin{aligned}&\big\{\{\}\to\big\{\{\mathrm{Gen}\to\{\tau,-3a-bt+ct^2\},\mathrm{Dim}\to\{0\},\mathrm{Var}\to\{t,\tau\},\mathrm{Par}\to\{a,b,c\}\big\},\\&\quad\{\mathrm{Gen}\to\{3b+2ct,5b^2-4ac+4c^2\tau^2\},\mathrm{Dim}\to\{0\},\mathrm{Var}\to\{t,\tau\},\mathrm{Par}\to\{a,b,c\}\}\big\},\\&\quad\{t\}\to\{\{\mathrm{Gen}\to\{\tau,-b-ct\},\mathrm{Dim}\to\{0\},\mathrm{Var}\to\{t,\tau\},\mathrm{Par}\to\{a,b,c\}\}\}\,.\end{aligned} \tag{4}$$

Each entry in this list represents an irreducible part of the vacuum manifold of our potential, the package has split up the equations for an extremum of the scalar potential as desired. It is sometimes useful to represent results, such as the above, in table form and the package provides a specific command, TF, for this purpose. Evaluating

```
satex//TF
```

we obtain the table shown in figure 3. The results in figure 3 should be compared to those

	Dim(compl.dim.)	Gen(Generators)	Par(Parameters)	Var(Variables)
{}	{0}	$\{\tau, -3\,a - b\,t + c\,t^2\}$	{a, b, c}	{t, τ}
{}	{0}	$\{3\,b + 2\,c\,t, 5\,b^2 - 4\,a\,c + 4\,c^2\,\tau^2\}$	{a, b, c}	{t, τ}
{t}	{0}	{τ, -b - c t}	{a, b, c}	{t, τ}

Figure 3: An example of the output of SaturationExpand.

in figure 1 which were obtained by applying SimpleMinimise to the same supergravity theory. We see that they are the same and that the presentation is similar in the two cases. In figure 3 we may obtain the equations for each of the vacua by setting the polynomials in the column Gen(Generators) to zero. The first column tells us which F-terms are zero in each vacuum. Thus, as before, we see that this model has two sets of supersymmetric vacua and one non-supersymmetric vacuum with $F_t \neq 0$. As with SimpleMinimise, if no warning messages are given by the package, SaturationExpand returns *all* of the isolated turning points of the potential which appear for generic parameter values. In summary, we have found the complete isolated vacuum structure of the model (2) by performing three simple commands, namely CreateModel, CalcModel and SaturationExpand.

These are, in fact, the three commands being performed during a call to SimpleMinimise. However, as we introduce more of the package's abilities it will be useful to have all the information we need close to hand – this information is stored in the model lists generated by CalcModel and CreateModel.

3 Filtering and finding properties of vacua

In the cases discussed above, the equations returned by SimpleMinimise and SaturationExpand were so simple that they may be solved trivially to find the vacua. Any physical property of those vacua may subsequently be calculated using the field values obtained. It may not always be the case that these modules simplify the equations to such an extent. We would like, therefore, to have an automated method for extracting the physical properties of the vacua which we discover, which does not require solving the returned equations. STRINGVACUA provides two types of facilities towards this end, *filtering* and *Sturm queries*.

3.1 Filtering

As STRINGVACUA performs a computation it can discard vacua which do not meet criteria specified by the user. This process, which we call filtering, can greatly speed up computations.

It avoids costly calculations being carried out on parts of the problem which are of no physical interest. Many of the modules of STRINGVACUA have a range of optional arguments, some of which deal with the task of filtering[1]. When omitted in module calls, the filtering options are set to sensible default values.

Dimension filtering

Perhaps the simplest type of filtering offered by the package is dimension filtering. By specifying an optional argument of the form Dim→list of dimensions, the user can choose to filter out any vacua which do not have a particular dimension (the number of flat directions). In fact, we have already used this type of filtering in the previous section! In SimpleMinimise and SaturationExpand this option is set by default to Dim→0. Thus, in the previous section, STRINGVACUA was only keeping vacua which are dimension zero – that is, vacua which do not have flat directions. Had we wished to search for vacua with 1 or 2 flat directions instead we could have evaluated the following.

```
SaturationExpand[SingleTModel, Dim->{1,2}]
```

The result, {}, tells us that there are no vacua with 1 or 2 flat directions in the model defined in (2) for generic values of the parameters.

Field value filtering

Another example of physically uninteresting vacua which one may want to filter out are those which lie in an unacceptable region of field space. For example, in a four dimensional theory obtained by compactifying a higher dimensional supergravity, we would wish for the size of the hidden dimensions to be larger than the string scale. If a particular field were to represent the volume of the hidden dimensions, and if our conventions were such that the string scale was set to 1, we may therefore only be interested in vacua where this field takes a value greater than 1.

We can instruct modules, such as SaturationExpand, to throw away all vacua which do not satisfy *any* set of conditions, as long as those conditions can each be stated as a sign condition on a polynomial in the fields. Let us see how this would work, using the condition described above, in the following model.

$$\begin{aligned} &K = -3\log(T+\bar{T}) - \log(S+\bar{S})\,, \quad W = aS + bST + cT^2\,, \\ &T = t + i\tau, \quad S = s + i\sigma\,. \end{aligned} \tag{5}$$

We begin by creating a model list, containing all of the relevant data of the supergravity theory, and add to this the ancillary mathematical structures the package requires. This is achieved, as in section 2, by use of the CreateModel followed by the CalcModel modules.

```
modelnew=CreateModel[{T,S}, {t,s}, {\[Tau],\[Sigma]}, -3 Log[T + Conjugate[T]]
  -Log[S + Conjugate[S]], a S + b S T + c T^2, Par->{a,b,c}];
modelnew=CalcModel[modelnew];
```

[1]We will encounter some of the optional arguments found in STRINGVACUA modules in the following sections. A comprehensive list of options for any module can be found in its *Mathematica* help file.

We now have a complete model list for the case in hand: modelnew. We now wish to apply SaturationExpand to this model, filtering out all vacua which do not obey the condition $t > 1$. The relevant command is

```
satexmodelnew = SaturationExpand[modelnew, VarPar->{a->1, b->-1, c->1},
   RRoots->"Probab", NRoots->1, SturmPoly->{t-1}, SturmSigns->1];
```

This module call makes use of some new optional arguments[2]. The argument

$$\text{VarPar} \rightarrow \{\text{a} \rightarrow 1, \text{b} \rightarrow \text{-1}, \text{c} \rightarrow 1\} \tag{6}$$

sets the parameters of the model (5) to the indicated values for the duration of this module call. Filtering can only be performed in cases where the parameters have been fixed; whether or not the required filtering conditions are met would otherwise be parameter dependent and would not have a unique answer. Ranges of parameter values can be scanned simply by writing an appropriate loop over module calls in *Mathematica*. The remaining options are

$$\text{RRoots} \rightarrow \text{“Probab”}, \text{NRoots} \rightarrow 1, \text{SturmPoly} \rightarrow \{\text{t-1}\}, \text{SturmSigns} \rightarrow 1 \tag{7}$$

The first two of these, in order, turn on field value filtering and only keep equations that have at least one real solution. The second two specify $t - 1$ as the polynomial whose sign is to be checked and state that the sign of this polynomial should be positive in any vacua to be kept in the output. In short, these options tell the module to only keep vacua for which the polynomial $t - 1$ takes positive values, i.e. for which $t > 1$

It is worth mentioning here that the calls to *Singular* will, of course, take different amounts of time on different computers. On a modern computer the above call will finish in under 30 seconds, however on older machines it may take longer. In the Unix release there is a default maximum time for *Singular* calculations, set at 60 seconds. This is controlled by the option SingularMaxTime. For example, to set the maximum time to 120 seconds for a particular calculation, simply add to any call the option[3] `SingularMaxTime->120`.

The result of the above call to SaturationExpand, put in table form using `satexmodelnew//TF`, is as shown in figure 4.

" "	"Dim(compl.dim.)"	"Gen(Generators)"	"NRoots(# real roots)"	"Par(Parameters)"	"Var(Variables)"
{s, t}	{0}	{τ, σ, −2 + s − 2 t, −2 − 4 t + 3 t^2}	{2}	{}	{s, t, σ, τ}

Figure 4: A filtered output from SaturationExpand.

Thus only a single set of vacua exist for this model with $t > 1$ and they spontaneously break supersymmetry. The first entry in the results table, {s, t} , indicates that the real parts of both

[2]To find out about the options available for SaturationExpand evaluate `?SaturationExpand`. This returns a brief summary of the module ending with “More...”. Click on “More...” to open the *Mathematica* help browser at the relevant page.

[3]To change the SingularMaxTime setting globally, refer to `?StringvacuaOptions`

F-terms are non-zero (while the imaginary parts vanish). The second entry tells us that this vacuum has no flat directions (recall the dimension filtering is set to only keep isolated extrema by default). The third entry is the polynomials which we set to zero to obtain the equations describing the vacua. The fourth column tells us that these equations have 2 real solutions. As always, since no warning messages were given during this computation, the results in figure 4 constitute *all* of the vacua of the system which satisfy our filtering constraints.

3.2 Finding properties of vacua

Once we have found vacua we will wish to investigate their physical properties (other than those determined by the filtering - which we already know!). This can be achieved in an automated manner using the SturmQuery module.

As an example let us examine further the result we obtained at the end of the previous subsection. What else might we like to know about the vacua we have found? We know, from using the SaturationExpand options, that the extrema lie in a sensible part of field space. However, we do not yet know if they are stable – have we found saddle points, minima, or maxima?

To answer this question one would usually compute the eigenvalues of the Hesse matrix in the vacuum and check to see if they are positive, negative or zero – but this is just a sign test of the type described above! Below, we will show how to formulate the relevant test by hand, in the interest of introducing more of the package modules. However, as stability is such an important issue, the package admits an option 'TestStability' for SaturationExpand, which automates the relevant sign tests. To illustrate, we will repeat the above SaturationExpand call, but adding this new option,

```
SaturationExpand[modelnew, VarPar->{a->1, b->-1, c->1},
  RRoots->"Probab", NRoots->1, SturmPoly->{t-1}, SturmSigns->1,
  TestStability->"Hesse"]//TF
```

These options, as before, instruct the package to retain only equations which have at least one real solution in the allowed region of field space. The additional, final, option TestStability has been set to "Hesse" – this instructs the package to determine the stability of the extrema described by the retained equations. The output, which we have here passed straight to table form, is just as in figure 4 (so that we recover the solutions we found previously) except that an additional column has been added to the table, containing

Sturm(Sturm Query)
{{Hessian}, {{1,-1},{2,2}},{{1,-1},{2,2}} }

How are we to interpret this? Each entry in the column is a list (one for each set of equations retained). The first entry in a list, {Hesse}, simply labels the type of sign test performed – here it reminds us that we are testing the signs of the Hesse matrix eigenvalues. There are *two* further entries (each a list itself), because the equations have *two* real solutions – in general, then, the list would contain a label and one further list for each real solution. These entries tell

us about the signs of the eigenvalues at each of root of the system. In this case they are both identical,

$$\{\{1,-1\},\{2,2\}\}\ . \tag{8}$$

The interpretation of this is simple: at both roots, the extrema have *two positive* (the first 2 goes with the +1) and *two negative* eigenvalues (the second 2 goes with the -1). Thus the non-supersymmetric vacua we have been studying are actually saddle points.

Let us take a closer look at how the package arrived at this result. The output of the filtered SaturationExpand call which led to this result was called satexmodelnew. The non-supersymmetric vacuum we found may be extracted from this list by using its label, as given in the first column of figure 4, as follows, `{s,t}/.satexmodelnew` which returns

$$\begin{aligned}\{\{\text{Gen}\to\{\tau,\sigma,-2+s-2t,-2-4t+3t^2\},\text{Dim}\to\{0\},\text{NRoots}\to\{2\},\\ \text{Var}\to\{s,t,\sigma,\tau\},\text{Par}\to\{\}\}\}\end{aligned} \tag{9}$$

This list contains the information in the table of figure 4, in a format which the modules of STRINGVACUA can understand. We will now pass this information to the SturmQuery module to find out about the stability properties of these non-supersymmetric vacua using the sign tests introduced above.

In fact, using SturmQuery we can check the signs taken by any rational functions, the signs taken by the determinants of any matrices of rational functions, and the signs taken by the eigenvalues of any matrix with real eigenvalues in the vacua we have found. In our case we of course wish to check the eigenvalues of the Hesse matrix. The first piece of information we require is the Hesse matrix itself. Fortunately this has already been calculated by the package and stored in the model list! The relevant keyword to access this matrix, as may be discovered from the help files, is Hesse. Thus evaluating `Hesse/.modelnew` will return the Hesse matrix of the system. The result is rather lengthy and so we do not reproduce it here.

To identify the signs of the eigenvalues of this matrix in the vacua given by (9) we simply perform the following SturmQuery call,

```
SturmQuery[{s,t}/.satexmodelnew, TestMatEigen->Hesse/.modelnew,
  VarPar->{a->1, b->-1, c->1}, SturmLabel->"The Hessian", RRoots->"Probab"]
```

The first argument above is the component of the vacuum space which we wish to test, as detailed in (9). The second argument is the matrix whose eigenvalues are to be checked, as we just described. The third argument sets the parameters in the Hesse matrix to the correct values – just those that were selected in the SaturationExpand call of the previous section, from which (9) was obtained. The fourth argument allows a user-defined label for the results of SturmQuery. Finally the fifth argument tells the module which method to use in its computations. The result of this call is shown in figure 5 (as earlier we have put the result in table form by appending `//TF` to the end of the result of the module call). We see that there is one additional piece of data in this table now, as compared to figure 4: the result of the sign check we have requested. This is precisely the same output we found using SaturationExpand[... TestStability→ "Hesse"], thus we recover the fact that these non-supersymmetric vacua are saddle points. A very large range of properties of vacua can be studied using SturmQuery. For more examples we refer the reader to the module's *Mathematica* help file.

Var(Variables)	{s, t, σ, τ}
Par(Parameters)	{}
Gen(Generators)	$\{\tau,\ \sigma,\ -2+s-2t,\ -2-4t+3t^2\}$
Dim(compl.dim.)	{0}
NRoots(# real roots)	{2}
Sturm(Sturm query)	{The Hessian, {{1, -1}, {2, 2}}, {{1, -1}, {2, 2}}}

Figure 5: A example output from SturmQuery.

4 Constraints on parameters

As well as finding the vacua of a given $\mathcal{N}=1$ theory, our methods may be used to identify constraints on the parameters of a model which are necessary for vacua of a given type to exist. As before we shall illustrate this with a simple example. Consider the following theory, obtained in [9] in the context of non-geometric compactifications of type II string theory.

$$\begin{aligned} K &= -3\log(-i(T-\bar{T}) - \log(-i(S-\bar{S})) - 3\log(-i(U-\overline{U})), \\ W &= a_0 - 3a_1T + 3a_2T^2 - a_3T^3 \\ &\quad + S(-b_0 + 3b_1T - 3b_2T^2 + b_3T^3) \\ &\quad + 3U(c_0 + (\hat{c}_1 + \check{c}_1 - \tilde{c}_1)T - (\hat{c}_2 + \check{c}_2 - \tilde{c}_2)T^2 - c_3T^3)\ , \end{aligned} \tag{10}$$

Here S, T and U are superfields while all other quantities are parameters. The parameters obey the following set of conditions, which come from consideration of tadpoles and integrability of Bianchi identities. We have,

$$\begin{aligned} 0 &= a_0b_3 - 3a_1b_2 + 3a_2b_1 - a_3b_0 - 16\ , \\ 0 &= a_0c_3 + a_1(\check{c}_2 + \hat{c}_2 - \tilde{c}_2) - a_2(\check{c}_1 + \hat{c}_1 - \tilde{c}_1) - a_3c_0\ , \end{aligned} \tag{11}$$

together with the equations below and another set given by exchanging all hats and checks,

$$\begin{aligned} 0 &= c_0b_2 - \tilde{c}_1b_1 + \hat{c}_1b_1 - \check{c}_2b_0\ , \quad & 0 &= c_0\tilde{c}_2 - \check{c}_1^2 + \tilde{c}_1\hat{c}_1 - \hat{c}_2c_0\ , \\ 0 &= \check{c}_1b3 - \hat{c}_2b_2 + \tilde{c}_2b_2 - c_3b_1\ , \quad & 0 &= c_3\tilde{c}_1 - \check{c}_2^2 + \tilde{c}_2\hat{c}_2 - \hat{c}_1c_3\ , \\ 0 &= c_0b3 - \tilde{c}_1b_2 + \hat{c}_1b_2 - \check{c}_2b_1\ , \quad & 0 &= c_3c_0 - \check{c}_2\hat{c}_1 + \tilde{c}_2\check{c}_1 - \hat{c}_1\tilde{c}_2\ , \\ 0 &= \check{c}_1b_2 - \hat{c}_2b_1 + \tilde{c}_2b_1 - c_3b_0\ , \quad & 0 &= \hat{c}_2\tilde{c}_1 - \tilde{c}_1\check{c}_2 + \check{c}_1\hat{c}_2 - c_0c_3\ . \end{aligned}$$

As an example, we shall describe how to find further constraints on the parameters of this system which are necessary (and in this case sufficient) for the existence of supersymmetric Minkowski vacua. These vacua are obtained as solutions to the equations,

$$\partial_T W = 0\ ,\ \partial_S W = 0\ ,\ \partial_U W = 0\ ,\ W = 0\ , \tag{12}$$

together, of course with the vanishing of the conditions on the parameters given above. This large and complicated set of equations contains information about what values the flux parameters can take in order for solutions to exist, but this data is entangled with all of the other information about the extrema (such as field values). Our objective is to separate out

the information concerning the constraints on the parameters. This may be achieved using the package routine Elimination.

First, we must collect the various equations describing the vacua we wish to analyse. We create the model list for the above system using the commands CreateModel and CalcModel as in earlier sections. Let us call the resulting model list model. The equations in (12), being information about our supergravity model, are now stored in this model list and may be extracted, in a form STRINGVACUA modules will understand, by evaluating the following command.

```
{IDSPot/.model, ISPot/.model}
```

See the appendix for a description of these keywords. Next we must enter the conditions on the parameters given above into *Mathematica*. Using the usual *Mathematica* syntax, these are included as a list of the polynomials which the conditions dictate vanish, let us call it Constraints.

```
Constraints={a0 b3 - 3 a1 b2 + 3 a2 b1 - a3 b0 - 16,...}
```

Now we have entered all of the relevant equations we may simply plug them into the module Elimination to obtain our constraints. The first argument in this module call is simply the collection of equations we wish to vanish. The optional argument list VarPar instructs the module to eliminate some set of variables from the equations it is passed. Here, we set `VarPar->{T,S,U}` in order to eliminate the field variables T, S and U from the equations. A final argument Meth is optional. Here we use it to select one of the faster methods of elimination available. The full command to be entered is

```
NewConstraints = Elimination[{IDSPot/.model, ISPot/.model, Constraints},
  VarPar->{T,S,U}, Meth->"Prod"];
```

We are left with NewConstraints which contains a list of polynomials. As usual we may display this object in a nice form by evaluating `NewConstraints//TF`. The result is shown in figure 6. When set to zero, the polynomials in the box "Gen (Generators)" represent the constraints on the flux parameters which are necessary and sufficient for the existence of supersymmetric Minkowski vacua[4]. Note that in figure 6 we have relabelled the parameters in an obvious way for ease of input into *Mathematica* (ct1 for $\tilde{c}_1$, etc).

Use of constraints, such as those found in this section, is one of the ways in which vacua which only appear for non-generic values of the parameters can be found using STRINGVACUA. One can simply solve for a sufficiently large number of parameters until the constraints for the required type of vacuum are automatically satisfied. Alternatively if one is running loops over calls to SaturationExpand, one for each set of flux parameters, the constraints in this section can be used to greatly speed up the calculation. Any set of parameters which do not satisfy the constraints need not be investigated.

[4]Often, one can apply one of the other STRINGVACUA modules, PrimDec, to the output of Elimination in order to separate this list into smaller, simpler subsets, the vanishing of each subset representing a different way to solve the full set of constraints.

Var(Variables)	{a0, a1, a2, a3, b0, b1, b2, b3, c0, c3, cc1, cc2, ch1, ch2, ct1, ct2}
Par(Parameters)	{}
Gen(Generators)	{ct2, ct1, ch2 + ct2, ch1 + ct1, cc2 − ch2, cc1 − ch1, c3, c0, 16 + a3 b0 − 3 a2 b1 + 3 a1 b2 − a0 b3, 16 a3^2 b0^2 − 96 a2 a3 b0 b1 − 288 a2^2 b1^2 + 432 a1 a3 b1^2 + 54 a2^3 b1^3 − 81 a1 a2 a3 b1^3 + 27 a0 a3^2 b1^3 + 432 a1 a3 b0 b2 − 27 a2^2 a3 b0^2 b2 + 48 a1 a3^2 b0^2 b2 − 288 a0 a3 b1 b2 − 18 a1 a2 a3 b0 b1 b2 − 45 a0 a3^2 b0 b1 b2 − 54 a1 a2^2 b1^2 b2 + 81 a1^2 a3 b1^2 b2 − 27 a0 a2 a3 b1^2 b2 + 54 a0 a2 a3 b0 b2^2 + 27 a0 a1 a3 b1 b2^2 − 27 a0^2 a3 b2^3 − 288 a1 a2 b0 b3 − 32 a0 a3 b0 b3 + 27 a2^3 b0^2 b3 − 45 a1 a2 a3 b0^2 b3 + 432 a0 a2 b1 b3 − 27 a1 a2^2 b0 b1 b3 + 54 a1^2 a3 b0 b1 b3 + 48 a0 a2 a3 b0 b1 b3 + 18 a0 a2^2 b1^2 b3 − 81 a0 a1 a3 b1^2 b3 − 144 a0 a1 b2 b3 + 27 a1^2 a2 b0 b2 b3 − 54 a0 a2^2 b0 b2 b3 − 51 a0 a1 a3 b0 b2 b3 + 27 a0 a1 a2 b1 b2 b3 + 45 a0^2 a3 b1 b2 b3 − 27 a0 a1^2 b2^2 b3 + 27 a0^2 a2 b2^2 b3 + 16 a0^2 b3^2 − 27 a1^3 b0 b3^2 + 45 a0 a1 a2 b0 b3^2 + 27 a0 a1^2 b1 b3^2 − 48 a0^2 a2 b1 b3^2 + 3 a0^2 a1 b2 b3^2}

Figure 6: Constraints on parameters obtained using Elimination.

5 Non-perturbative effects

So far we have addressed models for which the scalar potential, the F-terms, and all other relevant quantities are all rational functions of the fields. In many cases, such as in models where non-perturbative contributions to the superpotential play an important role, we may be interested in more complicated situations. These can be addressed using the STRINGVACUA package by the simple expedient of replacing any non-polynomial terms in the equations with dummy variables. As in earlier sections we will illustrate this with some examples.

5.1 Creating non-perturbative models

Consider the following toy model as an example.

$$\begin{aligned} &W = T + \exp(-S)\ , \quad K = -\log(T+\bar{T}) - 3\log(S+\overline{S})\ , \\ &T = t + i\tau\ , \quad S = s + i\sigma \end{aligned} \tag{13}$$

The supergravity model is set up in the package using CreateModel, just as before.

```
model = CreateModel[{T,S}, {t,s}, {\[Tau],\[Sigma]},
  -Log[T + Conjugate[T]] - 3 Log[S + Conjugate[S]], T + Exp[-S]];
```

Before we can apply our methods we must find a way to transform the system into a set of equations STRINGVACUA can understand – essentially into polynomials. This may be achieved by using dummy variables to represent non-algebraic objects. A good choice of dummy variables, in this case, is the set,

$$X \equiv \exp(-S), \quad Y \equiv \exp(-\overline{S})\ , \quad x \equiv \exp(-s), \quad y \equiv \exp(-i\sigma)\ . \tag{14}$$

These variables are built into the model as a set of replacements included in CalcModel (which, recall, takes supergravity objects and converts them into expressions the package can use) through two options. These are CDummies→..., a list of replacements for the superfields, and Dummies→..., a list of replacements for the real fields. The dummy variables defined in (14) are entered as shown below.

```
fullmodel = CalcModel[model, CDummies->{Exp[-S]->X, Exp[-Conjugate[S]]->Y},
  Dummies->{Exp[-s]->x, Exp[-I\[Sigma]]->y}];
```

To see the effect of the dummy variables, compare the supergravity potential,

```
Pot/.fullmodel
```

$$\frac{\mathrm{e}^{-2s}}{48s^3t}\left(3+12s+4s^2+3\mathrm{e}^{2s}t^2+4\mathrm{e}^{2s}\tau^2+6\mathrm{e}^{s}(2s-1)t\cos(\sigma)-6\mathrm{e}^{s}(2a+1)\tau\sin(\sigma)\right), \tag{15}$$

with its numerator, converted into a polynomial,

```
IPot/.fullmodel
```

$$\{\mathrm{Var}\to\{t,s,\tau,\sigma,x,y\},\mathrm{Par}\to\{\},\mathrm{Gen}\to\{-3tx+6stx+3t^2y+3x^2y+12sx^2y+4s^2x^2y \\ -3txy^2+6stxy^2+3ix\tau+6isx\tau-3ixy^2\tau-6isxy^2\tau+3y\tau^2\}\}\,. \tag{16}$$

Here, the dummy variables $\{x,y\}$ have been appended to the list of true variables $\{t,s,\tau,\sigma\}$ and the potential has been reduced to a polynomial as desired.

5.2 Finding vacua

The routine SaturationExpand may be used to identify and classify all vacua of this model in a few seconds. Note that we are working in an enlarged field space with dummy variables, in which true isolated vacua may appear to have flat directions. To ensure that this is accounted for, we adjust the Dim option in SaturationExpand to retain vacua with any number of flat directions up to and including the number of (real field) dummy variables included. SaturationExpand is then called using

```
satexNP = SaturationExpand[fullmodel, Dim ->{0,1,2}]
```

which returns a list satexNP of vacua classified by their SUSY breaking in a manner which is now familiar (see, for example, figure 4). Consider the vacuum space in which the imaginary parts of both F-terms vanish but the real parts are non-zero. We can extract the vacua of this kind from the full list by using their identifier, thus `{s,t}/.satexNP`. This list contains two potential vacua, as seen by evaluating `Length[{s,t}/.satexNP]`. Extracting one of these pieces using `({s,t}/.satexNP)[[1]]` we find

$$\{\mathrm{Gen}\to\{\tau,1+y,-3t^2+7tx+2stx+2x^2+4sx^2,-7t-2st+x+8sx+4s^2x, \\ -11t^2+2st^2+11tx+6x^2+16sx^2,-15t-2st+2s^2t-3x-4sx, \\ 3t^3-24t^2x+23tx^2+10x^3+24sx^3,3t^4-18t^3x+11t^2x^2-28tx^3-4x^4, \\ -18-78s-41s^2+4s^3+4s^4\},\mathrm{Dim}\to\{2\},\mathrm{Var}\to\{s,t,x,y,\sigma,\tau\},\mathrm{Par}\to\{\}\} \tag{17}$$

Thus, even in these non-perturbative cases, SaturationExpand can be used to break up the equations describing the vacuum space into smaller, more manageable pieces. In the above we could easily solve the equation given by setting the last polynomial in Gen to zero to find s and hence x. The solutions for τ and y (and thus σ) are evident from the first two polynomials. These solutions could then be plugged into the remaining polynomials to find t. In other words, in many cases the decomposition of the vacuum space provided by SaturationExpand makes the equations to be solved trivial. In situations where this is not the case the algorithm developed in [2] should be used. This algorithm can be executed using the modules of the package. This is, however, more involved than the level of this simple tutorial and we refer the user to [2] and to the package help files for more information.

The essential point we wish to convey is that one of the core keys to the power of STRINGVACUA – breaking up the equations for the vacuum space into smaller pieces – is applicable even in these non-polynomial cases.

5.3 Constraints in non-perturbative models

The other application of STRINGVACUA which we have introduced in this tutorial, finding constraints on parameters, also works in the case where non-perturbative effects are present.

As an example consider the supergravity model,

$$\begin{aligned} K &= -3\log\left(T+\bar{T}\right) - 3\log\left(Z+\bar{Z}\right) - \log\left(S+\bar{S}\right) , \\ W &= i(\xi + ieT) + (\epsilon + ipT)Z + i(\mu + iqT)Z^2/2 + \lambda\,\exp(-cS) , \\ T &= t + i\tau,\ Z = z + i\zeta,\ S = s + i\sigma\ . \end{aligned} \tag{18}$$

which is obtained by compactifying heterotic string theory on a generalised half-flat manifold in the presence of gaugino condensation [10, 11]. We can input this model into STRINGVACUA using CreateModel and CalcModel as usual, using the CDummies and Dummies options to specify the replacements for the exponentials. To clarify the presentation we here first define the Kähler and superpotentials,

```
K = -3 Log[T+Conjugate[T]] -3 Log [Z+Conjugate[Z]]-Log[S+Conjugate[S]];
W = I(\[Xi]+I e T)+(\[Epsilon]+I p T) Z + I (\[Mu]+I q T) Z^2/2
  +\[Lambda] Exp[-c S];
```

These are then entered into CreateModel using the identifiers K and W to create the supergravity model,

```
NPmodel = CreateModel[{T,S,Z}, {t,s,z}, {\[Tau],\[Sigma],\[Zeta]}, K, W,
  Par->{\[Xi], e, \[Epsilon], p, \[Mu], q, \[Lambda], c}];
```

Now we use CalcModel to create the extended model list, including our choice of dummy variables for the exponential function,

```
NPmodel = CalcModel[NPmodel, CDummies->{Exp[-c S]->X, Exp[-c Conjugate[S]]->Y},
  Dummies->{Exp[-c s]-> x, Exp[-I c \[Sigma]]-> y}];
```

There is one condition on the parameters which follows from the integrability of the Bianchi identity, $q\epsilon - p\mu = 0$, which we enter simply as

```
BIcondition = q \[Epsilon] - p \[Mu];
```

We can find further constraints on the parameters of this model which are necessary for the existence of supersymmetric vacua satisfying certain physical conditions. For various physical reasons we might be interested in vacua with field values $t = 1$, $s = 24$, and $\sigma = 0$, for example. We require, then, that the F-terms, and the polynomials $t - 1$, $s - 24$ and σ all vanish in our vacua.

Note that setting the value of $\sigma = 0$ implies a fixed value for $y \equiv \exp(-ic\sigma) = 1$, so we should also set the polynomial $y - 1$ to vanish. At this stage, even though we have chosen a value for s, the value of the dummy variable x still depends on the parameter c. As such, we cannot fix its value. To find our constraints, we simply collect all of our equations and then eliminate all of the fields using the command Elimination, exactly as in section 4.

```
ConstraintsNP = Elimination[{IFTerms/.NPmodel, BIcondition, t-1, s-24,
  \[Sigma], y-1}, VarPar-> {s, t, y, z, \[Zeta], \[Sigma], \[Tau]}];
```

The output of this module call is shown in figure 7, put in table form by evaluating

```
ConstraintsNP//TF
```

Var(Variables)	{c, e, p, q, x, ϵ, λ, μ, ξ}
Par(Parameters)	{}
Gen(Generators)	$\{q\,\epsilon - p\,\mu,\ e\,q\,\mu - q^2\,\xi,\ e\,p\,\mu - p\,q\,\xi,\ e^2\,\mu - e\,q\,\xi,$ $-e\,\mu + 96\,c\,e\,\mu + q\,\xi - 96\,c\,q\,\xi,\ e\,p\,\epsilon - p^2\,\xi,\ -e\,\epsilon + 96\,c\,e\,\epsilon + p\,\xi - 96\,c\,p\,\xi,$ $-4\,e^2\,\mu + 3\,e\,x\,\lambda\,\mu + 4\,e\,q\,\xi - 3\,q\,x\,\lambda\,\xi,\ e\,\mu - x\,\lambda\,\mu + 24\,c\,x\,\lambda\,\mu - q\,\xi,$ $-4\,e^2\,\epsilon + 3\,e\,x\,\epsilon\,\lambda + 4\,e\,p\,\xi - 3\,p\,x\,\lambda\,\xi,\ e\,\epsilon - x\,\epsilon\,\lambda + 24\,c\,x\,\epsilon\,\lambda - p\,\xi,$ $-q\,x\,\lambda + 24\,c\,q\,x\,\lambda,\ -p\,x\,\lambda + 24\,c\,p\,x\,\lambda,\ x\,\lambda\,\xi - 120\,c\,x\,\lambda\,\xi + 2304\,c^2\,x\,\lambda\,\xi,$ $2\,e\,p\,\epsilon - 3\,p\,x\,\epsilon\,\lambda - 12\,e\,x\,\lambda\,\mu + 9\,x^2\,\lambda^2\,\mu - 2\,p^2\,\xi + 6\,q\,x\,\lambda\,\xi,$ $-p^2\,x\,\lambda - 2\,e\,q\,x\,\lambda + 3\,q\,x^2\,\lambda^2,\ 4\,e\,x\,\lambda - 96\,c\,e\,x\,\lambda - 3\,x^2\,\lambda^2 + 72\,c\,x^2\,\lambda^2,$ $2\,e\,x\,\lambda - 112\,c\,e\,x\,\lambda + 1536\,c^2\,e\,x\,\lambda - x^2\,\lambda^2 + 24\,c\,x^2\,\lambda^2\}$

Figure 7: Constraints in a non-perturbative example using Elimination.

The returned polynomials in the entry labelled "Gen (Generators)" should be set to zero to obtain the desired constraints. The vanishing of these polynomials describes necessary constraints on the flux parameters of this model (necessary and sufficient constraints only being obtained in the case of perturbative supersymmetric Minkowski vacua) for the existence of vacua of the desired type [1, 2]. As mentioned in section 4, constraints such as these can frequently be simplified using other STRINGVACUA modules - for example PrimDec. The user is directed to the *Mathematica* help files for further information.

6 STRINGVACUA: other applications

Since the package is tailored to the fast manipulations of systems of polynomials, its applications clearly extend far beyond finding the vacua of string-inspired or derived models as described above. Problems such as finding the dimension of the space defined by a system of polynomials, or how many real roots there are, etc., are ubiquitous in countless fields of modern research. The convenient *Mathematica* interface we have developed, which requires neither an understanding of computational algebraic geometry nor of the program *Singular*, should prove useful to researchers in diverse fields of interest. In this short section we will illustrate the solution of questions one may encounter in more abstract situations.

Suppose we are given a formidable-looking system of polynomials, given in figure 8 and called sys, and we wish to check whether its zero-locus has dimension zero (i.e., no flat directions) and if so find how many real roots this system has. A system of polynomials, in the language of algebraic geometry, is called an *ideal*. Ideals are stored within STRINGVACUA as a list of assignments, called an *ideal list*, as follows.

```
ideal = {Gen->{polynomials}, Var->{variables}, Par->{parameters}}
```

Additional assignments can be added to ideal lists by some of the STRINGVACUA modules – we shall see an example of this shortly.

```
sys = {-7 w^2 + 7 w^4 + 35 w^3 x - 2 x^2 + 2 w^2 x^2 + 10 w x^3 - 2 w y + 51 w^3 y - x y +
    67 w^2 x y + 19 w x^2 y + 16 x^3 y - 4 y^2 + 74 w^2 y^2 + 43 w x y^2 + 24 x^2 y^2 + 44 w y^3 + 40 x y^3 +
    32 y^4 - 8 w z + 36 w^3 z - 10 x z + 92 w^2 x z + 58 w x^2 z + 12 x^3 z - 4 y z + 75 w^2 y z +
    170 w x y z + 88 x^2 y z + 110 w y^2 z + 137 x y^2 z + 36 y^3 z - 7 z^2 + 95 w^2 z^2 + 123 w x z^2 +
    76 x^2 z^2 + 89 w y z^2 + 98 x y z^2 + 92 y^2 z^2 + 92 w z^3 + 122 x z^3 + 39 y z^3 + 56 z^4,
  30 w^4 + 95 w^3 x + 134 w^2 x^2 + 83 w x^3 + 18 x^4 + 41 w^3 y + 133 w^2 x y + 153 w x^2 y + 93 x^3 y +
    101 w^2 y^2 + 200 w x y^2 + 137 x^2 y^2 + 52 w y^3 + 132 x y^3 + 70 y^4 + 21 w^3 z + 96 w^2 x z + 127 w x^2 z +
    71 x^3 z + 98 w^2 y z + 186 w x y z + 141 x^2 y z + 76 w y^2 z + 209 x y^2 z + 113 y^3 z + 102 w^2 z^2 +
    172 w x z^2 + 127 x^2 z^2 + 95 w y z^2 + 218 x y z^2 + 198 y^2 z^2 + 36 w z^3 + 99 x z^3 + 126 y z^3 + 81 z^4,
  21 w^2 x + 21 w x^2 + 3 x^3 + 63 w^2 y + 93 w x y + 30 x^2 y + 90 w y^2 + 72 x y^2 + 27 y^3 + 35 w^2 z +
    59 w x z + 32 x^2 z + 122 w y z + 140 x y z + 87 y^2 z + 40 w z^2 + 57 x z^2 + 76 y z^2 + 20 z^3,
  9 w^4 + 34 w^3 x + 79 w^2 x^2 + 54 w x^3 + 24 x^4 + 91 w^3 y + 192 w^2 x y + 193 w x^2 y + 94 x^3 y +
    133 w^2 y^2 + 230 w x y^2 + 148 x^2 y^2 + 103 w y^3 + 106 x y^3 + 28 y^4 + 14 w^3 z + 67 w^2 x z + 89 w x^2 z +
    70 x^3 z + 127 w^2 y z + 236 w x y z + 158 x^2 y z + 107 w y^2 z + 136 x y^2 z + 56 y^3 z + 95 w^2 z^2 +
    119 w x z^2 + 126 x^2 z^2 + 211 w y z^2 + 243 x y z^2 + 99 y^2 z^2 + 54 w z^3 + 117 x z^3 + 72 y z^3 + 81 z^4}
```

Figure 8: A given system of polynomial equations.

Given this general structure, the dimension in our current example is easily calculated using the command

```
Dim/.DimIdeal[{Var-> {x, y, z, w}, Gen-> sys}]
```

As described above, the Var → option specifies that the variables are $\{x, y, z, w\}$, the Gen → sys dictates that the system to be studied is our list sys of 4 polynomials in 4 variables and there is no Par assignment in this case as there are no parameters in our system. The command

DimIdeal[ ] then tells us the dimension of the space thus defined; it adds the assignment Dim → {the dimension} to the ideal list. Here we have extracted the dimension from this list using Dim /. The answer is very quickly found to be 0.

We can now try to find the real roots. The *Mathematica* command Solve[ ] is readily seen to be ineffective in solving this system. However, methods of computational algebraic geometry can first reduce this system into much simpler pieces by a technique known as *primary decomposition*. What this means is that the package can split sys up into a finite number of simpler components, the union of whose zero-loci is the zero-locus of sys itself – in other words, solving any of the simpler sets of equations gives a solution to the full set sys:

```
pieces = PrimDec[{Var->{x,y,z,w}, Gen->sys}];
```

The output is rather complicated so we will not present it here; suffice it to say that pieces is a list of length 11, check using `Length[pieces]`, meaning that there are 11 simple components to sys, each of dimension 0 and each represented by its own ideal list[5]. Now, we can find the number of real roots of any 0-dimensional system of polynomials with the command NumRoots[ ]. This adds the assignment NRoots → {number of roots} to an ideal list. We can then extract the number with the keyword NRoots in the same way as we extracted the dimension above.

We can perform this for each of the 11 components by entering

```
Table[NRoots/.NumRoots[pieces[[i]]], {i, 1, 11}]
```

giving us

$$\{\{1\},\{0\},\{0\},\{1\},\{0\},\{0\},\{1\},\{0\},\{0\},\{0\},\{0\}\}$$

Therefore, we see that eight of the components do not have real roots while three of them have a single real root. In other words, sys has a total of 3 real roots.

We can, of course, do more. Since we have the explicit expressions for the actual pieces we can find what the roots are explicitly. We see from the above that the 1st, 4th and 7th components have real roots. Executing `pieces[[1]]` and similarly extracting pieces[[4]] and pieces[[7]] will give us the relevant equations. In fact, here they all turn out to be

$$\{\text{Gen} \to \{\text{z, y, x, w}\},\ \text{Dim} \to \{0\},\ \text{Var} \to \{\text{w, x, y, z}\},\ \text{Par} \to \{\}\}$$

We see that the Gen keyword is set to $\{z, y, x, w\}$. To find the real roots we must therefore solve the equations obtained by setting each of the polynomials in this list to zero - which is just the origin. In other words, all of the roots here are at $(0, 0, 0, 0)$ and therefore the original system, sys, has a triple root at the origin and no other real roots.

7 Summary

We have briefly described, illustrating our discussion with simple examples, how the *Mathematica* package STRINGVACUA may be used to aid in the investigation of vacuum configurations

[5]By default, PrimDec retains only dimension 0 pieces. As before this may be adjusted using the Dim→ option.

of models of string phenomenology. We have focused on two main applications. First, we described how to split the large equations describing the vacuum space of a model up into smaller systems, each of which can be dealt with more easily. Second, we discussed how to obtain constraints on the parameters in a theory necessary for the existence of given types of vacua.

In fact, in this short note, we have only used 7 of the package's 17 modules and we have far from exhausted the flexibility provided by optional arguments even in those. The message is that there is much more STRINGVACUA can do to aid the user than the simple automated routines described in this document. Many of the methods imported to *Mathematica* from *Singular* by this package are not restricted to use in string theory, but will benefit researchers in any area where a problem involves polynomial computations. The papers [1, 2], along with the *Mathematica* help files for the various modules, should go a long way towards helping the user in apply the techniques of STRINGVACUA in a more flexible manner. This, as always with a computer algebra system, can lead to huge improvements in what may be achieved using the package.

Acknowledgments

It is a pleasure to thank Beatriz de Carlos, Vishnu Jejjala, Sven-Ludwig Krippendorf, Eran Palti, Fernando Quevedo, Mario Serna, and George Weatherill for testing and making useful suggestions on the beta version of Stringvacua. J. G. is supported by STFC, A. L. is supported by the EC 6th Framework Programme MRTN-CT-2004-503369, and Y.-H. H is supported by the FitzJames Fellowship of Merton College, Oxford.

A Glossary – supergravity model list keywords

A.1 CreateModel

The command `model=CreateModel[...]` takes a list of fields, a Kähler potential and a superpotential, and creates a model list, here called model. A model list contains essentially all of the information about a supergravity theory which is useful in an analysis of its vacuum space. Objects, such as the F-terms and the scalar potential, can be extracted via 'keywords' by executing a command of the form

```
keyword/.model
```

where `keyword` can be any of the following:

Keyword	Description
Fields	A list of the superfields in the model, e.g. $\{T\}$.
ReFields	A list of the real parts of the superfields in the model, e.g. $\{t\}$.
ImFields	A list of the imaginary parts of the superfields in the model, e.g. $\{\tau\}$.
ComplexToReal	A list of the relations between the Fields and their real and imaginary parts, in the form $\{T \to t + i\tau\}$.
Par	A list of the flux parameters in the model, e.g. $\{a\}$.
KPot	The Kähler potential.
SPot	The superpotential.
DeltaPot	The non-supersymmetric extension to the scalar potential.
DSPot	A list of first derivatives of the superpotential with respect to the superfields, in the form "{field → derivative with respect to field}".
FTerms	A list of the F-terms, in the form "{field → F-term}".
Pot	The supergravity scalar potential, calculated from the superpotential, Kähler potential and non-SUSY extension.
DPot	A list of the first derivatives of the scalar potential, in the same format as DSPot, above.
Hesse	The Hessian matrix of the scalar potential, computed with respect to the real fields.

A.2 CalcModel

CalcModel creates an extended model list which contains the keywords above and added assignments where the supergravity objects have been processed into a form suitable for our methods – an "I" is prefixed to the above keywords to indicate this. For example, Pot/.model is now accompanied by IPot/.model. All data is accessed in the manner described above. The following optional inputs of CalcModel are also keywords:

Keyword	Description
CDummies	A list of replacement rules for non-algebraic functions of superfields, in the form "function→ dummy variable".
Dummies	A list of replacement rules for non-algebraic functions of real fields, in the form "function→ dummy variable".
RDummies	A list of rules for replacing non-algebraic functions with dummy variables, in the form "function→ dummy variable".

References

[1] J. Gray, Y. H. He and A. Lukas, "Algorithmic algebraic geometry and flux vacua," JHEP **0609**, 031 (2006) [arXiv:hep-th/0606122].

[2] J. Gray, Y. H. He, A. Ilderton and A. Lukas, "A new method for finding vacua in string phenomenology," JHEP **0707**, 023 (2007) [arXiv:hep-th/0703249].

[3] D. Grayson and M. Stillman, "Macaulay 2, a software system for research in algebraic geometry." Available at `http://www.math.uiuc.edu/Macaulay2/` .

[4] G.-M. Greuel, G. Pfister, and H. Schönemann, "Singular: a computer algebra system for polynomial computations," Centre for Computer Algebra, University of Kaiserslautern (2001). Available at `http://www.singular.uni-kl.de/` .

[5] L. B. Anderson, Y. H. He and A. Lukas, "Heterotic compactification, an algorithmic approach," JHEP **0707**, 049 (2007) [arXiv:hep-th/0702210].

[6] J. Gray, Y. H. He, V. Jejjala and B. D. Nelson, "Exploring the vacuum geometry of N = 1 gauge theories," Nucl. Phys. B **750**, 1 (2006) [arXiv:hep-th/0604208].

[7] J. Distler and U. Varadarajan, "Random polynomials and the friendly landscape," arXiv:hep-th/0507090.

[8] Singular.m by M. Kauers and V. Levandovskyy, available at `http://www.risc.uni-linz.ac.at/research/combinat/software/Singular/` .

[9] J. Shelton, W. Taylor and B. Wecht, "Nongeometric flux compactifications," JHEP **0510**, 085 (2005) [arXiv:hep-th/0508133].

[10] S. Gurrieri, A. Lukas and A. Micu, "Heterotic on half-flat," Phys. Rev. D **70**, 126009 (2004) [arXiv:hep-th/0408121].

[11] B. de Carlos, S. Gurrieri, A. Lukas and A. Micu, "Moduli stabilisation in heterotic string compactifications," JHEP **0603**, 005 (2006) [arXiv:hep-th/0507173].