Exactness in numerical algebraic computations

Dan Bates
Jon Hauenstein
Tim McCoy
Chris Peterson
Andrew Sommese

Wednesday, December 17, 2008
MSRI Workshop on Algebraic Statistics
Main goals for this talk
Main goals for this talk

1. Give a quick and dirty primer on the basics of *numerical algebraic geometry* (for newcomers).
Main goals for this talk

1. Give a quick and dirty primer on the basics of *numerical algebraic geometry* (for newcomers).

2. Report on the (early) progress on a new project in this direction, perhaps entirely unrelated to algebraic statistics.
The Gameplan

- Motivation
- Methods for recovering exactness
- Available numerical data
- The new algorithm
- An example
- Unresolved issues
Motivation (exactness)
Motivation (exactness)

We want to study polynomial systems in $\mathbb{Z}[x_1,\ldots,x_n]$ and their solution sets in $\mathbb{C}^n$. 
We want to study polynomial systems in $\mathbb{Z}[x_1,...,x_n]$ and their solution sets in $\mathbb{C}^n$.

There are symbolic methods, which have benefits and drawbacks.
Motivation (exactness)

We want to study polynomial systems in $\mathbb{Z}[x_1, \ldots, x_n]$ and their solution sets in $\mathbb{C}^n$.

There are symbolic methods, which have benefits and drawbacks.

There are also numerical methods, also with benefits and drawbacks.
Motivation (exactness)

We want to study polynomial systems in $\mathbb{Z}[x_1,\ldots,x_n]$ and their solution sets in $\mathbb{C}^n$.

There are symbolic methods, which have benefits and drawbacks.

There are also numerical methods, also with benefits and drawbacks.

Today, we’ll look at a bridge from numerical computation to symbolic information.
We want to study polynomial systems in $\mathbb{Z}[x_1,\ldots,x_n]$ and their solution sets in $\mathbb{C}^n$.

There are symbolic methods, which have benefits and drawbacks.

There are also numerical methods, also with benefits and drawbacks.

Today, we’ll look at a bridge from numerical computation to symbolic information.

**WARNING**: This project may not have anything to do with algebraic statistics (?).
Motivation (exactness)

More specifically: Given some approximations of points on each component, can we:

(a) find a polynomial (with exact coefficients) that vanishes on a given irreducible component?

(b) find an ideal for each irreducible component?

(c) recover the original ideal?
More specifically: Given some approximations of points on each component, can we:

(a) find a polynomial (with exact coefficients) that vanishes on a given irreducible component?

(b) find an ideal for each irreducible component?

(c) recover the original ideal?

Who cares anyway? Numerical methods may be more efficient, at least in certain cases.
Motivation (NAG)
Motivation (NAG)

Basic homotopy continuation:
Motivation (NAG)

**Basic homotopy continuation:**

Want the isolated (complex) solutions of the (complex) target system $F(z)=0$. 
Basic homotopy continuation:

Want the isolated (complex) solutions of the (complex) target system \( F(z) = 0 \).

Choose a start system \( G(z) \).
Basic homotopy continuation:

Want the isolated (complex) solutions of the (complex) target system $F(z)=0$.

Choose a start system $G(z)$.

Form $H(z,t) = F(z)(1-t) + G(z)t$ such that:
- $H(z,0) = F(z)$.
- $H(z,1) = G(z)$. 
Basic homotopy continuation:

Want the isolated (complex) solutions of the (complex) target system \( F(z) = 0 \).

Choose a \textit{start system} \( G(z) \).

Form \( H(z,t) = F(z) \cdot (1-t) + G(z) \cdot t \) such that:
  - \( H(z,0) = F(z) \).
  - \( H(z,1) = G(z) \).

\textbf{WARNING:} As stated, this is incorrect.
Motivation (NAG)

Basic homotopy continuation:

\[ t = 0 \quad \text{and} \quad t = 1 \]
Motivation (NAG)

Basic homotopy continuation:

\[ t = 0 \quad t = 1 \]
Motivation (NAG)

Basic homotopy continuation:
Motivation (NAG)

Basic homotopy continuation:
Motivation (NAG)

Basic homotopy continuation:

Number of paths depends on choice of $g(z)$:
- At most the Bézout number, and
- At least the BKK number.
Basic homotopy continuation:
Motivation (NAG)

Basic homotopy continuation:

\[ t = 0 \quad t = 1 \]
Motivation (NAG)

Basic homotopy continuation:

\[ t = 0 \quad t = 1 \]

\[ t \]
Motivation (NAG)

Basic homotopy continuation:
Basic homotopy continuation:

Motivation (NAG)
Motivation (NAG)

Basic homotopy continuation:

\[ t = 0 \quad t = 1 \]
Basic homotopy continuation:

Motivation (NAG)

(Also not entirely correct.)
Motivation (NAG)
Motivation (NAG)

(Just add linear equations to \( F(z) \).)
Motivation (NAG)
(Remove one linear equation from $F(z)$.)
Motivation (NAG)
Motivation (NAG)

(Membership or LDT)
Motivation (NAG)
Motivation (NAG)
Motivation (NAG)
Motivation (NAG)

**Bottom line:** If you need to find solutions of polynomial systems, there are numerical methods for doing this.
**Bottom line:** If you need to find solutions of polynomial systems, there are numerical methods for doing this.

In particular, these methods work for “large” systems.
Bottom line: If you need to find solutions of polynomial systems, there are numerical methods for doing this.

In particular, these methods work for “large” systems.

For more on numerical algebraic geometry, see:
- the 2005 book by Sommese-Wampler
- me
- Anton Leykin
- Philipp Rostalski
- (others)
Motivation (NAG)

A few more advanced methods that may interest you:
- Component sampling and membership
- Parameter continuation (for parameterized systems)
- Regularity and multiplicity structure of a zero-scheme (Dayton-Zeng ‘04 or B-Peterson-Sommese ‘05)
- Local dimension test (B-Peterson-Sommese ‘08)
- Numerical primary decomposition (Leykin ‘08)
Motivation (NAG)

Software:
- PHCPack (Jan Verschelde)
- Bertini (B, Hauenstein, Sommese, Charles Wampler)
  - 1.1 released last week
  - Parallel and Mac versions available now

- HOM4PS-2.0 + mixedVol (T.Y. Li, ....)

Others:
- HomLab (Wampler)
- PHoM (Kojima, ....)
- HOMPACK (Layne Watson, ....)
Motivation (NAG)

Software:
- PHCPack (Jan Verschelde)
- Bertini (B, Hauenstein, Sommese, Charles Wampler)
  - 1.1 released last week
  - Parallel and Mac versions available now
- HOM4PS-2.0 + mixedVol (T.Y. Li, ....)

Others:
- HomLab (Wampler)
  - PHoM (Kojima, ....)
- HOMPACK (Layne Watson, ....)
A word about words:
A word about words:
Numerical algebraic geometry: Numerical methods for algebraic geometry based on homotopy continuation.
A word about words:

Numerical algebraic geometry: Numerical methods for algebraic geometry based on homotopy continuation.

This is probably too restrictive:
- Stetter (Numerical polynomial algebra)
- ApCoCoA (APCOA)
- Kaltofen/Szanto (clusters of roots)
- Shub/Smale (clusters of roots)
- others
Motivation (exactness)

Back to recovery of exactness (finally):

We want functions. Why not interpolate?
Motivation (exactness)

Back to recovery of exactness (finally):

We want functions. Why not interpolate?

**Example**: Twisted cubic (degree 3 curve)

\[ I = \langle xy-z, y^2-xz, x^2-y \rangle. \]
Motivation (exactness)

Back to recovery of exactness (finally):

We want functions. Why not interpolate?

Example: Twisted cubic (degree 3 curve)

\[ I = \langle xy-z, y^2-xz, x^2-y \rangle. \]

20 (exact) sample points:

3 generators, 7 “simple” elements of \( I \).
Motivation (exactness)

Back to recovery of exactness (finally):

We want functions. Why not interpolate?

**Example**: Twisted cubic (degree 3 curve)

\[ I = \langle xy-z, y^2-xz, x^2-y \rangle. \]

20 (exact) sample points:

3 generators, 7 “simple” elements of \( I \).

Added perturbations to sample points:

Numerical results (no nonzero coefficients).
Motivation (exactness)

Problems with interpolation:

- Numerical instability.

- Yields inexact coefficients, so worthless for us anyway.
Problems with interpolation:

- Numerical instability.
- Yields inexact coefficients, so worthless for us anyway.

Q: So how do we move from real numbers to rational numbers (or integers)?
Problems with interpolation:

- Numerical instability.
- Yields inexact coefficients, so worthless for us anyway.

Q: So how do we move from real numbers to rational numbers (or integers)?

A: Lean on some algorithms related to multivariate factorization, the numerical approximation of numerical constants, etc.
The Gameplan

- Motivation
- **Methods for recovering exactness**
- Available numerical data
- The new algorithm
- An example
- Unresolved issues
Methods for Recovering Exactness

Given floating point numbers $x_1, \ldots, x_N$, how can we find integers $a_1, \ldots, a_N$ such that $a_1 x_1 + \ldots + a_N x_N \approx 0$. 
Methods for Recovering Exactness

Given floating point numbers $x_1, \ldots, x_N$, how can we find integers $a_1, \ldots, a_N$ such that $a_1 x_1 + \ldots + a_N x_N \approx 0$.

All methods boil down to some numerical linear algebra.
Methods for Recovering Exactness

Given floating point numbers $x_1, \ldots, x_N$, how can we find integers $a_1, \ldots, a_N$ such that $a_1x_1 + \ldots + a_Nx_N \approx 0$.

All methods boil down to some numerical linear algebra.

For this project, we treat these as blackboxes.
Methods for Recovering Exactness

Given floating point numbers $x_1, \ldots, x_N$, how can we find integers $a_1, \ldots, a_N$ such that $a_1x_1 + \ldots + a_Nx_N \approx 0$.

All methods boil down to some numerical linear algebra.

For this project, we treat these as blackboxes.

1. **LLL** [Lenstra-Lenstra-Lovasz, ‘82]

Canonical choice.
Methods for Recovering Exactness

Given floating point numbers $x_1, \ldots, x_N$, how can we find integers $a_1, \ldots, a_N$ such that $a_1x_1 + \ldots + a_Nx_N \approx 0$.

All methods boil down to some numerical linear algebra.

For this project, we treat these as blackboxes.

1. **LLL** [Lenstra-Lenstra-Lovasz, ‘82]

Canonical choice.

*Original motivation*: Find a nearly-orthogonal basis which contains a “short” vector (“shortest” is NP-hard).
**Main application:**

Factorization of polynomials:

- **Univariate**: Lenstra-Lenstra-Lovasz, 1982
- **Dense multivariate**: Kaltofen, 1985
- **Sparse multivariate**: von zur Gathen-Kaltofen, 1985
- **Using straight-line programs**: Kaltofen, 1989
- **Using blackbox representation**: Kaltofen-Trager, 1990
Some drawbacks:

- “Short” isn’t very short. \[ \|f_1\| \leq 2^{\frac{n-1}{2}} \|f\| \]
- It is a precision hog.
- There is no guarantee that failure to find a short vector means that there is no relation (could be precision).
- It is numerically unstable.
- The time complexity is higher than with other methods.
Methods for Recovering Exactness

Some drawbacks:

• “Short” isn’t very short. \[ \| f_1 \| \leq 2^{\frac{n-1}{2}} \| f \| \]

• It is a precision hog.

• There is no guarantee that failure to find a short vector means that there is no relation (could be precision).

• It is numerically unstable.

• The time complexity is higher than with other methods.

Q: Why is this the standard?
Methods for Recovering Exactness

Some drawbacks:

• “Short” isn’t very short. \( \| f_1 \| \leq 2^{\frac{n-1}{2}} \| f \| \)

• It is a precision hog.

• There is no guarantee that failure to find a short vector means that there is no relation (could be precision).

• It is numerically unstable.

• The time complexity is higher than with other methods.

Q: Why is this the standard?

A: It was the first polynomial-time algorithm.
2. **Ferguson-Forcade** (1979)

First algorithm for more than two real numbers.

Horrific complexity, numerically unstable, etc.

3. **PSOS** (Ferguson, 1988)

4. **HJLS** (Ferguson-Bailey, 1992)
Methods for Recovering Exactness

5. **PSLQ** [Ferguson-Bailey, ’92, ’99 [with Arno]]

**Original motivation:**

Given a set of floating point numbers $x_1, \ldots, x_N$, find integers $a_1, \ldots, a_N$ such that $a_1x_1 + \ldots + a_Nx_N \approx 0$. 
Methods for Recovering Exactness

5. **PSLQ** [Ferguson-Bailey, ’92, ’99 (with Arno)]

**Original motivation:**

Given a set of floating point numbers $x_1, \ldots, x_N$, find integers $a_1, \ldots, a_N$ such that $a_1x_1 + \ldots + a_Nx_N \approx 0$.

Used for the numerical approximation of constants such as $\pi$ (see Bailey, Borwein, & Plouffe, ‘97).
Methods for Recovering Exactness

Nice properties:

• Better run times than the others.
• Requires very little working precision.
• Clear signal when precision is the problem (so it yields proof that there are no relations).
• Experimentally, it seems to be clear when an integer relation is not good (based on the number of digits in the coefficients).
Implementations:

There are several standalone + Maple + SAGE, etc.

We use Maple for now (for better or for worse).

Homemade versions are easy – the recipe is short and clear.
The Gameplan

- Motivation
- Methods for recovering exactness
- **Available numerical data** (back to the project)
- The new algorithm
- An example
- Unresolved issues
In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note**: Witness points can be computed (easily) to extremely high precision – thousands of digits.
In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note:** Witness points can be computed (easily) to extremely high precision – thousands of digits.

**Example:** Twisted cubic, \(C\).
Available Numerical Data

In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note:** Witness points can be computed (easily) to extremely high precision – thousands of digits.

**Example:** Twisted cubic, C.

We find $\text{deg}(C) = 3$ generic points on $C$, to any level of precision: 100 digits
Available Numerical Data

In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note**: Witness points can be computed (easily) to extremely high precision – thousands of digits.

**Example**: Twisted cubic, C.

We find $\text{deg}(C) = 3$ *generic* points on C, to any level of precision: 100 digits

$$x = 5.6314107615877350795833922848242299211272438438669484384092992850671788826485246989796508159770987243094930 \times 10^{-1} - 1.12320489429173333835435539623419057484692554537452935133853396707416186304459939469372866768524811973737370 \times i$$
Available Numerical Data

In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note:** Witness points can be computed (easily) to extremely high precision – thousands of digits.

**Example:** Twisted cubic, C.

We find \( \text{deg}(C) = 3 \) *generic* points on \( C \), to any level of precision: 200 digits

\[
x = 3.9197465138191660465712811447587188128203328058318661346603501189698389649958346259012843266938619500423964375074888131457932524622207810884033227298487616071117431179255869301825665364084123052278236504559e-1 + 4.4441741110805382158797385083336333452278199985389583031952667832191509839507410010849749989145096065581143228358551455961263242760890123674580689345515648123455164049866168455557689411175364760011241137231e-2i
\]
Available Numerical Data

In numerical algebraic geometry, we use numerical (homotopy) methods to compute witness sets.

**Important note**: Witness points can be computed (easily) to extremely high precision – thousands of digits.

**Example**: Twisted cubic, C.

We find \( \text{deg}(C) = 3 \) generic points on C, to any level of precision: 1000 digits

X=0.268995305911330353427551035799105477150928100996860990185255160614498539395432977863383556631501068538525753810808087018401584613217087388632073778252063015466919672864489858897422078026261383919050471094586864516096370160882965876817129114266284543220874624363421775717441412700091047086431039482694324207939583513991451583131558826364874483432111129597
91934363796193005600429424577770131466811188111376542780415160851011156057257674140385249213732972032324107365080824069557356298516006991708404123846246651396649247940687557317684628396230008929260426239797690636435538689776275671669778332003855090680821727538044779228126796112930957297760577367161934663821527126185559665847053579979062006082952458485637691912777907257856668128685749602701548115298221130962362289918074246832498541525517263917279874947599694708773699661751903914990576451790185019363513956885087386414173552395161664166478450069961197507370200423816404830056377180303076151466976968826763358892712662139e-0.245344914356068939081630408692657503772121667885424601282558420185633769221936910795643048224706888135884368767667518513145420658289149567801155573561516855558479469935323205120823096119558534711913737438663766640526740414123374123221670203276409214808883053040719053642870191199595661326522665660365870914627796353475771949252355885766312369061041776531287025601038589297265674669624413606861879888030969459423040681601972147481601371940467066243605621475799681811908527345704221590000351818411188036849572596155647501633776723406297183171600409416591133443887079830827303085356741208320668673920649738652602750518439078431292497259596802350995332596510431951429567573729624853466766689914341425892484556914573760207386174677426638794320133888579686166862925775623641306303490892200711568645890461741340331995563064083801691543645699178377776770785259259199837852913985208374698008752336664434367861321067178772699340162907760901411852330292536475394848144299115243249050714282246841e0
In numerical algebraic geometry, we use numerical [homotopy] methods to compute witness sets.

Important note. Witness points can be computed [easily] to extremely high precision — thousands of digits.

Example. Twisted cubic $C$.

We find $\text{deg}(C) = 3$ generic points on $C$, to any level of precision, 10000 digits.
The Gameplan

• Motivation
• Methods for recovering exactness
• Available numerical data
• The new algorithm
• An example
• Unresolved issues
The new algorithm

**Input**: Point P on some reduced*, irreducible component C of some complex algebraic set, available to any level of precision.

**Output**: A set of generators for the ideal defining C.

*: We have an algorithm for the nonreduced case, too, though that is for another talk....
The new algorithm

For D (degree) from 1 to MAX_D:
The new algorithm

For D (degree) from 1 to $\text{MAX_D}$:

$\text{PREC} = \text{MIN_PREC}$. 

Form the degree D Veronese embedding $P_D$ of point $P$. 
The new algorithm

For D (degree) from 1 to \text{MAX\_D}:

\text{PREC} = \text{MIN\_PREC}.

Form the degree D Veronese embedding $P_D$ of point P.

While not done: (loop within this degree)
The new algorithm

For D (degree) from 1 to \( \text{MAX}_D \):

- PREC = MIN_PREC.
- Form the degree D Veronese embedding \( P_D \) of point P.
- While not done: (loop within this degree)
  - Run PSLQ to obtain polynomial \( f \) or detect lack of precision.
The new algorithm

For D (degree) from 1 to MAX_D:

PREC = MIN_PREC.

Form the degree D Veronese embedding $P_D$ of point P.

While not done: (loop within this degree)

Run PSLQ to obtain polynomial $f$ or detect lack of precision.

If more precision is needed, increase PREC and restart loop.
The new algorithm

For D (degree) from 1 to \text{MAX}_D:

\text{PREC} = \text{MIN}_\text{PREC}.

Form the degree D Veronese embedding \( P_D \) of point P.

While not done:  (loop within this degree)

Run PSLQ to obtain polynomial f or detect lack of precision.

If more precision is needed, increase \text{PREC} and restart loop.

Otherwise, decide if f is actually a relation and if f is new.
The new algorithm

For D (degree) from 1 to MAX_D:

- PREC = MIN_PREC.
- Form the degree D Veronese embedding $P_D$ of point P.
- While not done: (loop within this degree)
  - Run PSLQ to obtain polynomial f or detect lack of precision.
  - If more precision is needed, increase PREC and restart loop.
  - Otherwise, decide if f is actually a relation and if f is new.
  - If f is a new relation, store it, trim $P_D$, and restart loop.
The new algorithm

For D (degree) from 1 to \text{MAX\_D}:

- PREC = MIN\_PREC.

- Form the degree D Veronese embedding \( P_D \) of point P.

- While not done: (loop within this degree)
  
  - Run PSLQ to obtain polynomial f or detect lack of precision.
  
  - If more precision is needed, increase PREC and restart loop.
  
  - Otherwise, decide if \( f \) is \text{actually a relation} and if \( f \) is \text{new}.
  
  - If \( f \) is a new relation, store it, trim \( P_D \), and restart loop.
  
  - If \( f \) is not new, just trim \( P_D \) and restart loop.
The new algorithm

For D (degree) from 1 to \( \text{MAX}_D \):

\( \text{PREC} = \text{MIN}_\text{PREC} \).

Form the degree D Veronese embedding \( P_D \) of point \( P \).

While not done:  (loop within this degree)

Run PSLQ to obtain polynomial \( f \) or detect lack of precision.

If more precision is needed, increase \( \text{PREC} \) and restart loop.

Otherwise, decide if \( f \) is actually a relation and if \( f \) is new.

If \( f \) is a new relation, store it, trim \( P_D \), and restart loop.

If \( f \) is not new, just trim \( P_D \) and restart loop.

If \( f \) is not a relation (i.e., it is junk), break to outer loop.
The new algorithm

**MAX\_D:** Castelnuovo-Mumford regularity. (better??)

**actually a relation:** The number of digits in meaningless integer relations is around PREC/vector size. Can we make a mistake? Yes, but we can correct it.

“f is new”: Any \( p(x)f(x) \) where \( f(x) \) is a generator will also be in the ideal. So, we see if any potentially new generator is in the ideal formed by the previously discovered generators.
The new algorithm

Q: Will this algorithm terminate?
The new algorithm

Q: Will this algorithm terminate?

A: Yes. The degree of the generators (in a minimal set of generators) is bounded. We could even compute the bound.
The new algorithm

Q: Will this algorithm terminate?

A: Yes. The degree of the generators (in a minimal set of generators) is bounded. We could even compute the bound.

Q: Can we detect whether we are done?
The new algorithm

Q: Will this algorithm terminate?

A: Yes. The degree of the generators (in a minimal set of generators) is bounded. We could even compute the bound.

Q: Can we detect whether we are done?

A: Yes, numerically and/or symbolically. This is good since the bounds are awful.
The Gameplan

• Motivation
• Methods for recovering exactness
• Available numerical data
• The new algorithm
• An example
• Unresolved issues
An example

To make a long story short:

Example: Twisted cubic.

Using 100 digits (maybe less?), recovered all 3 generators:
An example

To make a long story short:

**Example**: Twisted cubic.

Using 100 digits (maybe less?), recovered all 3 generators:

\[
\text{GENS} := \{x z - y, y w - z, x w - y z\}
\]
An example

To make a long story short:

**Example**: Twisted cubic.

Using 100 digits (maybe less?), recovered all 3 generators:

\[
\text{GENS} := \{x z - y^2, y w - z^2, x w - y z\}
\]

\>
\text{runLLLNA}(\text{GENS}, \text{VARS}, \text{BET_DIGITS});
Cascase Summary

NOTE: nonsingular vs singular is based on rank deficiency and identical ends

<table>
<thead>
<tr>
<th>codim</th>
<th>paths</th>
<th>witness superset</th>
<th>nonsingular</th>
<th>singular</th>
<th>nonsolutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

| total | 15 |

Witness Set Summary

NOTE: nonsingular vs singular is based on rank deficiency and identical ends

<table>
<thead>
<tr>
<th>codim</th>
<th>witness points</th>
<th>nonsingular</th>
<th>singular</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Witness Set Decomposition

dimension: components: classified: unclassified

| 1     | 1 | 3 | 0 |

Decomposition by Degree

Dimension 1: 1 classified component

degree 3: 1 component
degree = 1

degree = 2

\[-x^2 w + y z\]

\[-y^2 w + z\]

\[-x^2 z + y\]

degree = 3

\[2\]

\[-x^2 w + y z w\]

\[-y^2 w + z w\]

\[-x^2 w + y z x\]

\[-x^2 z w + z y\]

\[-y^2 x w + y z\]

\[-y^2 w - y z x - z x + y + y z + z y\]

Relations:

\[-x^2 w + y z\]

\[-y^2 w + z\]

\[-x^2 z + y\]
The Gameplan

• Motivation
• Methods for recovering exactness
• Available numerical data
• The new algorithm
• An example
• Unresolved issues
Unresolved issues

• Detect how much precision we will need \textit{a priori} (either in the data or for computation)?

• Are there better bounds on the degrees of the generators?

• Complexity? Is this really faster than prime decomposition?

• Stability? I.e., suppose we know our points to 3 digits. Could we accidentally pick out some other algebraic set?

• Avoid Gröbner bases, control growth in Veronese embedding?
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.

• You can detect positive-dimensional components numerically by finding points on them.
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.

• You can detect positive-dimensional components numerically by finding points on them.

• There is a way to move from numerical data to exact information, in various settings.
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.

• You can detect positive-dimensional components numerically by finding points on them.

• There is a way to move from numerical data to exact information, in various settings.

• New approach for the salmon conjecture? Can you find a point on your algebraic set?
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.

• You can detect positive-dimensional components numerically by finding points on them.

• There is a way to move from numerical data to exact information, in various settings.

• New approach for the salmon conjecture? Can you find a point on your algebraic set?

• Max Buot: Maximum Likelihood Equations....
What I hope you get out of this:

• You can find isolated solutions of polynomial systems with numerical methods.

• You can detect positive-dimensional components numerically by finding points on them.

• There is a way to move from numerical data to exact information, in various settings.

• New approach for the salmon conjecture? Can you find a point on your algebraic set?

• Max Buot: Maximum Likelihood Equations....

• I will bet that Bernd has some ideas....
A few references


Thank you!
Methods for Recovering Exactness

Example [from von zur Gathen-Gerhard]:

Lattice generated by \((12,2), (13,4)\).  \textbf{LLL}: \((1,2), (9, -4)\).

How short is “short”?
This time, shortest.

Often nearly the shortest.

Proven bound:
\[
\|f_1\| \leq 2^{\frac{n-1}{2}} \|f\|
\]
**Methods for Recovering Exactness**

**Basic procedure:** Gram-Schmidt, with a twist.

Usual Gram Schmidt: \[ e_j = v_j - \sum_{i=1}^{j-1} \frac{\langle v_j, e_i \rangle}{\langle e_j, e_i \rangle} \cdot e_i \]

LLL’s “Gram Schmidt”: \[ e_j = v_j - \sum_{i=1}^{j-1} \frac{\langle v_j, e_i \rangle}{\langle e_j, e_i \rangle} \cdot e_i \]

LLL uses this idea, along with reordering the vectors between steps (to work towards short vectors).

**Result:** Basis vectors grow by a factor of 2 (or more).
Main idea: Replace Gram-Schmidt with Householder transformations for numerical stability (in addition to various other changes).

Nice properties:
- Better run times than the others.
- Requires very little working precision.
- Clear signal when precision is the problem (so it yields proof that there are no relations).
- Experimentally, it seems to be clear when an integer relation is not good (based on the number of digits in the coefficients).
Methods for Recovering Exactness

**Popular Application:** (Bailey-Borwein-Plouffe, ‘97)

Apply PSLQ to \( \{X_1, X_2, ..., X_8, \pi\} \) where

\[
X_j = \sum_{k=0}^{\infty} \frac{1}{16^k (8k + j)} = \frac{1}{j} + \frac{1}{16(8 + j)} + \frac{1}{16^2 (8 \cdot 2 + j)} + \ldots
\]

PSLQ yields \([4,0,0,-2,-1,-1,0,0,-1]\).

So \( \pi = \sum_{i=0}^{\infty} P_i = \sum_{i=0}^{\infty} \frac{1}{16^i} \left( \frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right) \)

\[
\frac{P_{i-1}}{P_i} \rightarrow 16, \text{ so we get 1 hexadecimal digit per step.} \]