Theory and Applications of Numerical Algebraic Geometry

Silviana Amethyst

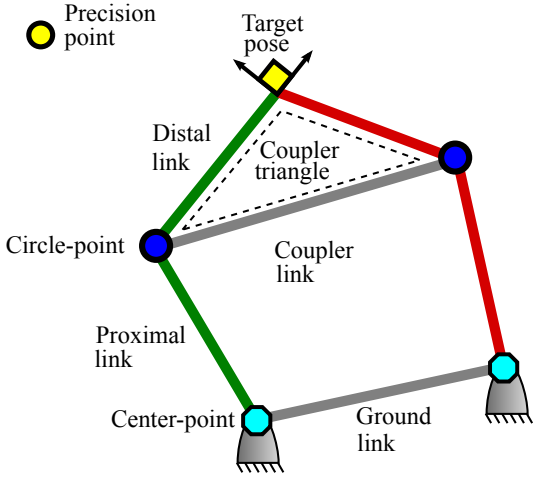
13 October, 2020

University of Wisconsin
Eau Claire

The Power of
AND

Outline

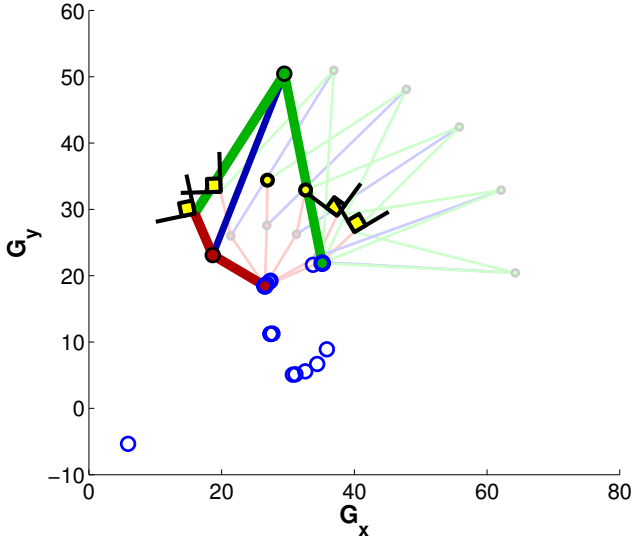
Four-bar mechanisms



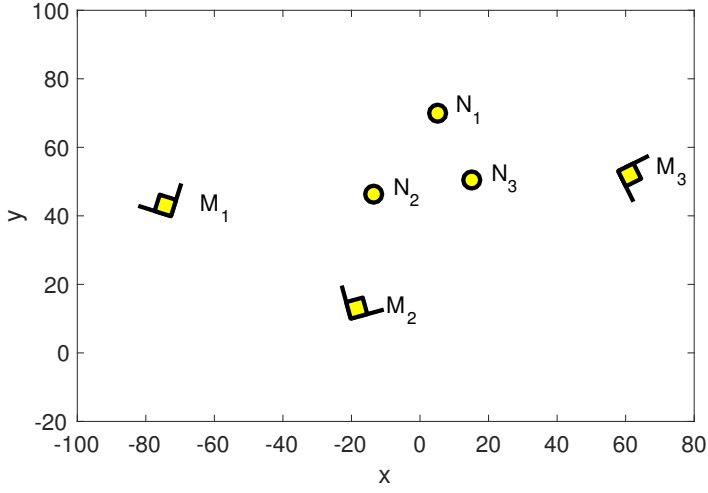
see this video, 1:37

A design problem

Design a mechanism passing through the **points** and achieving the **poses**



Challenge: design the four-bar



Can **you** design a four-bar satisfying these constraints?

Challenge: 3-3 fourbar system

$$
\begin{bmatrix}
-(a_i i + a_r - d_1 - t_1(x_r + x_i i))(D_1 + a_i i - a_r - T_1(x_i i - x_r)) + (a_i i + a_r - x_i i - x_r)(a_i i - a_r - x_i i + x_r) \\
-(a_i i + a_r - d_2 - t2(x_r + x_i i))(D_2 + a_i i - a_r - T_2(x_i i - x_r)) + (a_i i + a_r - x_i i - x_r)(a_i i - a_r - x_i i + x_r) \\
(a_i i + a_r - x_i i - x_r)(a_i i - a_r - x_i i + x_r) - (D_3 + a_i i - a_r + (t_{3,i} i - t_{3,r})(x_i i - x_r))(a_i i + a_r - d_3 - (t_{3,r} + t_{3,i} i)(x_r + x_i i)) \\
(a_i i + a_r - x_i i - x_r)(a_i i - a_r - x_i i + x_r) - (D_4 + a_i i - a_r + (t_{4,i} i - t_{4,r})(x_i i - x_r))(a_i i + a_r - d_4 - (t_{4,r} + t_{4,i} i)(x_r + x_i i)) \\
(a_i i + a_r - x_i i - x_r)(a_i i - a_r - x_i i + x_r) - (D_5 + a_i i - a_r + (t_{5,i} i - t_{5,r})(x_i i - x_r))(a_i i + a_r - d_5 - (t_{5,r} + t_{5,i} i)(x_r + x_i i)) \\
-(b_i i + b_r - d_1 - t_1(y_r + y_i i))(D_1 + b_i i - b_r - T_1(y_i i - y_r)) + (b_i i + b_r - y_i i - y_r)(b_i i - b_r - y_i i + y_r) \\
-(b_i i + b_r - d_2 - t_2(y_r + y_i i))(D_2 + b_i i - b_r - T_2(y_i i - y_r)) + (b_i i + b_r - y_i i - y_r)(b_i i - b_r - y_i i + y_r) \\
(b_i i + b_r - y_i i - y_r)(b_i i - b_r - y_i i + y_r) - (D_3 + b_i i - b_r + (t_{3,i} i - t_{3,r})(y_i i - y_r))(b_i i + b_r - d_3 - (t_{3,r} + t_{3,i} i)(y_r + y_i i)) \\
(b_i i + b_r - y_i i - y_r)(b_i i - b_r - y_i i + y_r) - (D_4 + b_i i - b_r + (t_{4,i} i - t_{4,r})(y_i i - y_r))(b_i i + b_r - d_4 - (t_{4,r} + t_{4,i} i)(y_r + y_i i)) \\
(b_i i + b_r - y_i i - y_r)(b_i i - b_r - y_i i + y_r) - (D_5 + b_i i - b_r + (t_{5,i} i - t_{5,r})(y_i i - y_r))(b_i i + b_r - d_5 - (t_{5,r} + t_{5,i} i)(y_r + y_i i)) \\
-(t_{3,i} i - t_{3,r})(t_{3,r} + t_{3,i} i) - 1 \\
-(t_{4,i} i - t_{4,r})(t_{4,r} + t_{4,i} i) - 1 \\
-(t_{5,i} i - t_{5,r})(t_{5,r} + t_{5,i} i) - 1
\end{bmatrix}
$$

- 14 variables (a,b,x,y,t), 13 functions,
  14 numerical parameters (angles and positions)
- underdetermined, defines some positive dimensional objects

Q: **how are we to solve this???**
A: **numerical algebraic geometry**

Questions NAG answers

Given a polynomial system f, understand

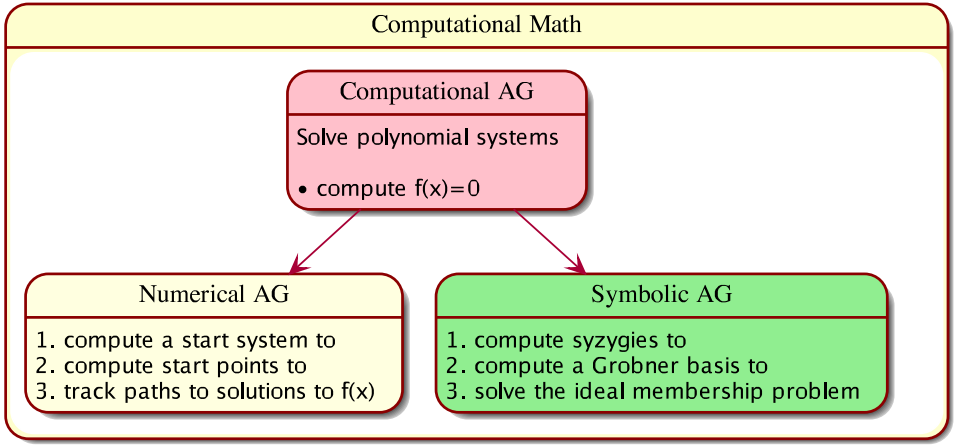
$$\mathcal{V}(f) = \{z \in \mathbb{C}^N \,:\, f(z) = 0\}$$

the *algebraic variety* of f.

- What are the *dimensions* of the components f defines?
- What are the *degrees* of the components of $\mathcal{V}(f)$?
- Describe the *real part* of $\mathcal{V}(f)$?
- Do the components of $\mathcal{V}(f)$ *intersect*?
- Given a point p on $\mathcal{V}(f)$, what is the *local geometry* around p?
- What can we do to **reduce the computation time**? How can we **make it more convenient**?

Outline

Computational algebraic geometry is two parts



What is NAG?

numerical algebraic geometry (NAG), or **numerical nonlinear algebra** is a field of *computational algebraic geometry* which parallels symbolic algebraic geometry

- Algebraic geometry studies
  zero sets of systems of polynomials
- Computational AG computes the zero sets
- *Symbolic* AG computes them via *manipulation of polynomials*
  $\Rightarrow$ think *gaussian elimination*
- *Numerical* AG computes them via *points*

Fundamental tools of NAG

- **Homotopy continuation** – numerical path tracking
- **Form a start system** – choice impacts solve time
- **Bertini's theorem** – theoretical foundation
- **Endgames** – tools to compute singular points

Homotopy continuation – an example

Let's solve this two-variable *target system*:

$$f(x, y) = \begin{bmatrix} \sqrt{3}x^2 + 2xy - 4 \\ 0.5xy - x^3 \end{bmatrix}$$

To do so, let's instead solve this easier *start system*:

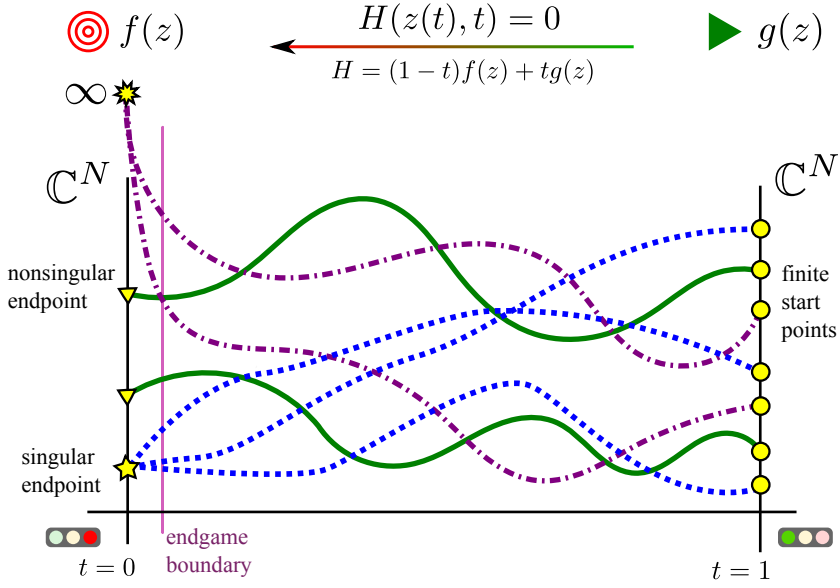
$$g(x, y) = \begin{bmatrix} x^2 - 1 \\ y^3 - 1 \end{bmatrix}$$

Then we deform g into f using this *homotopy*:

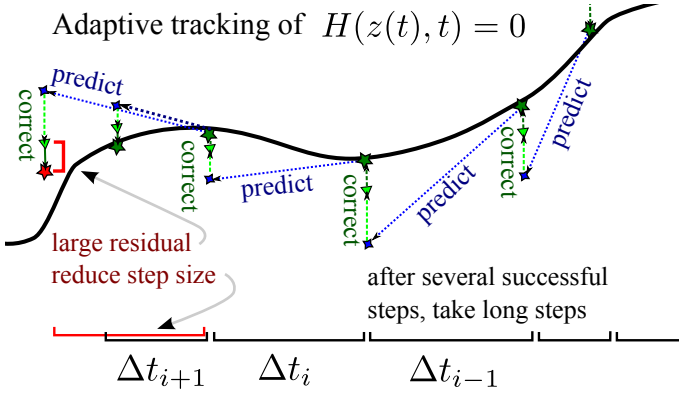
$$H(x, y, t) = (1 - t)f + tg = 0$$

starting from $t_{\text{start}} = 1$ to $t_{\text{end}} = 0$

Homotopy continuation – in general



Predictor-corrector tracking



Solve the *Davidenko Differential Equation*

$$\frac{\partial H}{\partial z}\frac{\mathrm{d}z}{\mathrm{d}t} + \frac{\partial H}{\partial t} = 0 \qquad \Rightarrow \qquad \frac{\mathrm{d}z}{\mathrm{d}t} = -\left[JH(z,t)^{-1}\right]\frac{\partial H(z,t)}{\partial t}$$

Start system

To solve f, we first solve g, the **start system**. Some are:

- **Roots of unity** – high root count, closely spaced solutions, trivial to write / create, bad conditioning
- **Total degree** with randomization – high root count, easy to form, spatially separated points

- And other more complicated ones $\star$
  - **Linear product** – Build up by taking products of linears. Lower root count by discarding singularities along the way
  - **M-homogeneous** – Root count may be lower, may be higher. Combinatorially many to choose from
  - **Polyhedral** – Very sharp root count, hard to form

$\star$ Better start systems reduce total track time.

Why does NAG work?

- **Bertini's theorem** – a generalization of the *fundamental theorem of algebra*.

  The FToA tells us that

  $f(z)$ is a univariate polynomial of degree d,

  $\Rightarrow f$ has d roots over $\mathbb{C}$, counting multiplicity

  Bertini's theorem tells us a system of polynomials splits over $\mathbb{C}$.

- Bertini's theorem provides *upper bounds* on the $\#$ of solutions a polynomial system can have.
- The first important one is the *Bezout bound* $\Longleftrightarrow$ total degree $g(x)$

Implementation

Some highlights of implementation details:

- The use of **arbitrary precision arithmetic** allows computation near singularities during tracking.
- **Adaptive step length** allows one to invent schemes for staying near target path
- Combining both adaptive step length and adaptive precision gives the best of both worlds
- **Endgames** allow one to compute singular endpoints

Bertini – software for NAG

bertini.nd.edu

Outline

Algorithms of NAG

- **Witness sets**
  represent a positive dimensional object as
  (linear slice) $\cap$ system
- **Regeneration**
  build up solution sets from nothing via
  $\prod$ (linear function)
- **Deflation**
  magically make singular $\rightarrow$ non-singular
- **Endgames**
  special methods for computing singular roots
- **Certification**
  Proving that candidates are approximate roots

Witness sets come from slicing

To **slice** is to intersect a variety and an affine space.
Slicing produces a **witness (super)set**: a triple $W = \{\{w_i\}, \mathcal{L}, f\}$:

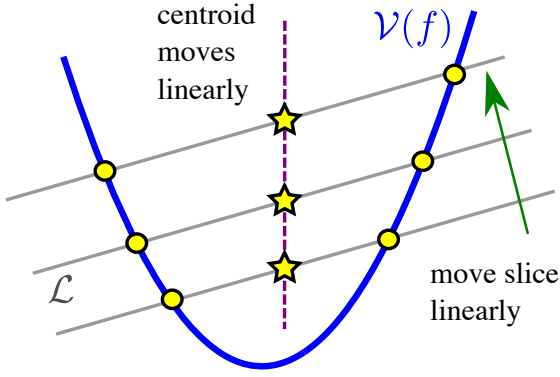
1. Linear Slice $\mathcal{L} : \mathbb{C}^N \to \mathbb{C}^{N-n}$
2. System $f : \mathbb{C}^N \to \mathbb{C}^n$
3. Points $\{w_i\} = \mathcal{V}(f) \cap \mathcal{V}(\mathcal{L})$

Two imporant properties, when slicing done correctly:

- $\#(\{w_i\}) = \sum \deg(C_j) + \text{junk}$,
- $\dim(C_j) = N - n = k$

Trace Test

The *trace test* numerically verifies completeness of a set

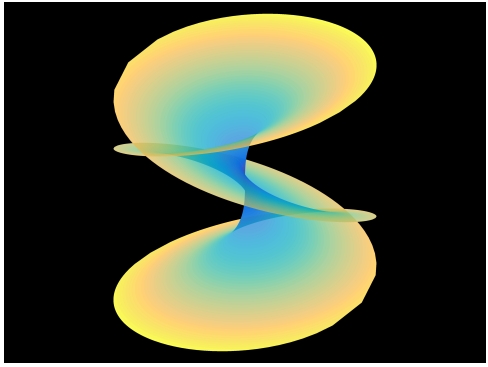


The trace is 0 $\iff$ set of points is complete

Monodromy

To do *monodromy* is to walk in a circle and collect data.

- Compute new connections to known points
- Separate sets of points by connections
- Compute series expansion coefficients
- Estimate singular solutions



Regeneration

To *regenerate* is to use products of linears to build up a start system, and solve by homotoping as

$$H(x,t) = (1-t)\begin{bmatrix} f(x) \\ \begin{bmatrix} \tilde{f}(x) \\ \end{bmatrix} \end{bmatrix} + t\begin{bmatrix} f(x) \\ \prod_{i=1}^{d_1} L_{1,i}(x) \\ \vdots \\ \prod_{i=1}^{d_k} L_{k,i}(x) \end{bmatrix} = 0$$

- d_j is the degree of j^{th} polynomial in $\tilde{f}$.

This permits *intersections* and other fun computations

Example – regeneration to find curve critical points (left nullspace method)

Regeneration uses products of linears to build up a start system for a homotopy. We're solving by homotoping as

$$H(x, v; t) = (1-t)\begin{bmatrix} f(x) \\ v^{\mathsf{T}} \cdot \begin{bmatrix} Jf(x) \\ J\pi_1 \end{bmatrix} \\ \text{patch}_v \end{bmatrix} + t\begin{bmatrix} f(x) \\ M_1(v)\prod_{i=1}^{\delta} L_{1,i}(x) \\ \vdots \\ M_N(v)\prod_{i=1}^{\delta} L_{N,i}(x) \\ \text{patch}_v \end{bmatrix} = 0$$

- δ is the maximum degree any polynomial in f.
- π is some linear projection

Deflation

Deflation

- Making the untrackable, trackable.
- Regularizing witness and other points.
- Reduces one solution component at a time.
- Produces a new system for which Newton's method hypothetically converges quadratically.
- We are deflating the multiplicity of the singularity.
- Isosingular sets are closures of sets with the same deflation sequence.

Deflation – nullspace

given $f : \mathbb{C}^N \to \mathbb{C}^n$ with dnull(Jf) at a generic point x, choose a general linear system $\mathcal{L} : \mathbb{C}^N \to \mathbb{C}^d$, and set

$$g_e(x, \eta) = \begin{bmatrix} f(x) \\ Jf(x) \cdot \eta \\ \mathcal{L}(\eta) \end{bmatrix}$$

yields

$$g_e : \mathbb{C}^{2N} \to \mathbb{C}^{2n+d}$$

[Leykin, Verschelde, Zhao '06]

Deflation – determinental

This method of deflations adds no variables, but instead **lots** of functions

$$g_{\text{det}} = \begin{bmatrix} f(x) \\ \det J_{\sigma_1} f(x) \\ \vdots \\ \det J_{\sigma_m} f(x) \end{bmatrix}$$

where $\{\sigma_i\}$ indexes the set of $(n-d+1)$x$(n-d+1)$ submatrices of the Jabobian. d is still dimnull of Jf.

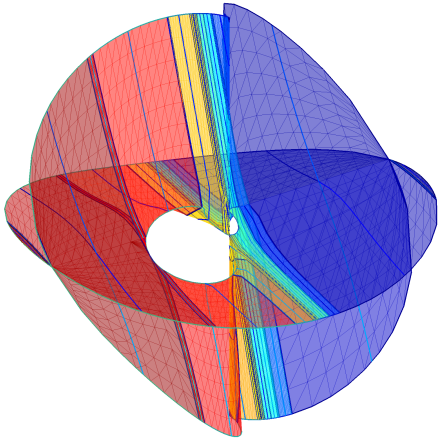
$$g_{\text{det}} : \mathbb{C}^N \to \mathbb{C}^{\text{scary}}$$

[Hauenstein, Wampler '13]

Application of deflation – finding singular curves

1. Compute witness set for critical curve.
2. Separate singular witness points from nonsingular.
   - Nonsingular $\rightarrow$ critical curve.
   - Singular $\rightarrow$ singular curve(s).
3. Separate singular witness points by deflation sequence.
4. Decompose each singular curve.

Example - Solitude



Example - Solitude

Solitude:

$$f(x, y, z) = x^2yz + xy^2 + y^3 + y^3z - x^2z^2$$

$$\mathcal{D}^1 = \begin{bmatrix} x^2yz + xy^2 + y^3 + y^3z - x^2z^2 \\ \\ y^2/4 - (xz^2)/2 + (xyz)/2 \\ (xy)/2 + (x^2z)/4 + (3y^2z)/4 + (3y^2)/4 \\ (x^2y)/4 - (x^2z)/2 + y^3/4 \end{bmatrix}$$

Example curve deflated - Solitude

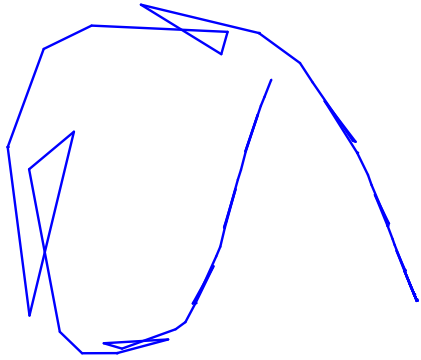
$$\mathcal{D}^2 = \begin{bmatrix} x^2yz + xy^2 + y^3 + y^3z - x^2z^2 \\ \\ y^2/4 - (xz^2)/2 + (xyz)/2 \\ (xy)/2 + (x^2z)/4 + (3y^2z)/4 + (3y^2)/4 \\ (x^2y)/4 - (x^2z)/2 + y^3/4 \\ \\ (y^2z^2)/8 - (x^2z^3)/24 + (y^2z^3)/8 - (y^3z^2)/8 - (y^3z)/8 + y^3/24 + (xy^2z)/24 + (x^2yz^2)/24 \\ (x^2z^3)/12 + (y^3z^2)/24 + (xy^3)/24 - (y^4z)/24 - (xy^2z)/12 - (x^2yz^2)/8 + (x^2y^2z)/24 \\ (x^2y^2)/24 + (xy^3)/8 - y^4/24 - (xy^2z)/4 - (x^2yz)/12 + (xy^3z)/12 - (xy^2z^2)/4 \\ -(x(x^2z^2 - 3y^2z^2 + 2xz^2 + 6yz^2 - 3y^2z + 6yz^3 + y^2 + xyz))/24 \\ (x^3z^2)/24 - (x^2y^2)/48 + y^4/48 + (x^2yz)/12 + (xy^3z)/12 - (xy^2z^2)/8 \\ (xy^3)/12 - (x^2y^2)/16 + (x^3z)/12 + (x^4z)/48 + (y^4z)/16 + y^4/16 + (x^2yz)/4 + (x^2yz^2)/4 \\ (x^3z^2)/24 - (x^2y^2)/16 - (xy^3)/8 + y^4/16 + (xy^2z)/4 + (x^2yz)/6 + (xy^2z^2)/8 \\ -(x(x^2y^2 + 2x^2z^2 + xy^2 - 2y^3z + y^4 - 2x^2yz))/24 \\ -(y(4x^2y^2 + 6x^2y + 4x^3 + x^4 + 3y^4))/48 \\ (y^2z^2)/12 - (x^2z^2)/36 - (xz^2)/36 - (yz^2)/12 + (y^2z)/12 - (yz^3)/12 - y^2/36 - (xyz)/36 \\ (x^2z^2)/24 - (y^2z^2)/24 - (xy^2)/36 + (y^3z)/24 - (x^2yz)/72 + (xyz)/18 \\ (x^2z)/18 - (x^2y)/72 - (xy^2)/12 + (x^3z)/72 + y^3/24 + (xyz^2)/6 - (xy^2z)/24 + (xyz)/6 \\ (x^2z^2)/24 - (y^2z^2)/24 - (xy^2)/36 + (y^3z)/24 - (x^2yz)/72 + (xyz)/18 \\ -(x^2(y^2 - 3yz + 3z^2))/36 \\ -(xy(2x - 6yz + x^2 + 3y^2))/72 \\ (x^2z)/18 - (x^2y)/72 - (xy^2)/12 + (x^3z)/72 + y^3/24 + (xyz^2)/6 - (xy^2z)/24 + (xyz)/6 \\ -(xy(2x - 6yz + x^2 + 3y^2))/72 \\ -(x^2y^2)/24 - (x^2y)/12 - x^3/36 - x^4/144 - y^4/16 - (x^2yz)/12 \end{bmatrix}$$

Endgames

Singular endpoints require special techniques to compute; using regular ODE methods will probably not work.
Choices:

1. Power series – approximate curve with a powerseries, extrapolate
2. Cauchy – walk circles around $t = 0$ in complex space, integrate

a numerical homotopy path



default bertini settings, a path projected onto its real coordinates

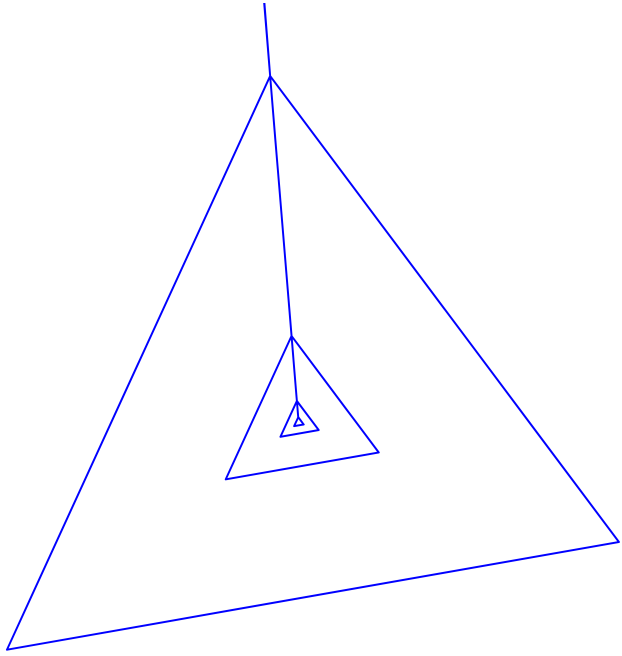
Cauchy endgame – procedure

- Fact: the path is a Puiseaux series in t.
- Method: Extrapolate to $t = t_0$ via a Cauchy integral (just compute the mean!)
- Tool: Walk circles around t_0 to gather numerical data to do Trapezoid Rule quadrature.

$$f(s) = \frac{1}{2\pi i} \int_{\circ} \frac{f(s)}{s - t_0} \, dz$$

1. Guess the cycle number
   - 1.1 Approximate the first terms of the power series
   - 1.2 Extrapolate forward to t'
   - 1.3 Track to next time
   - 1.4 Choose c which minimizes error at t'
2. Extrapolate to t_0

Cauchy endgame – a picture



Certification – general

Let

$$\beta(f,x) = ||x - \text{Newtonstep}(f,x)||$$
$$\gamma(f,x) = sup_{k\leq 2} \left|\left| \frac{Jf(x)^{-1} J^k f(x)}{k!} \right|\right|^{\frac{1}{k-1}}$$
$$\alpha(f,x) = \beta(f,x)\,\gamma(f,x)$$

if

$$\alpha(f,x) < \frac{13 - 3\sqrt{17}}{4} \approx 0.157671$$

then Newton's method will converge to some ξ with

$$f(\xi) = 0 \quad \text{exactly}$$

Certification – real

To check that $x = \bar{x}$ – that x is an approximate solution to a *real* ξ:

1. Compute β, γ, α
2. If $||x - \text{real}(x)|| > 2\beta$, not real
3. If $\alpha < 0.03$ && $||x - \text{real}(x)|| < \frac{1}{20\gamma}$, real
4. take a newton step, goto 1

Alpha certified

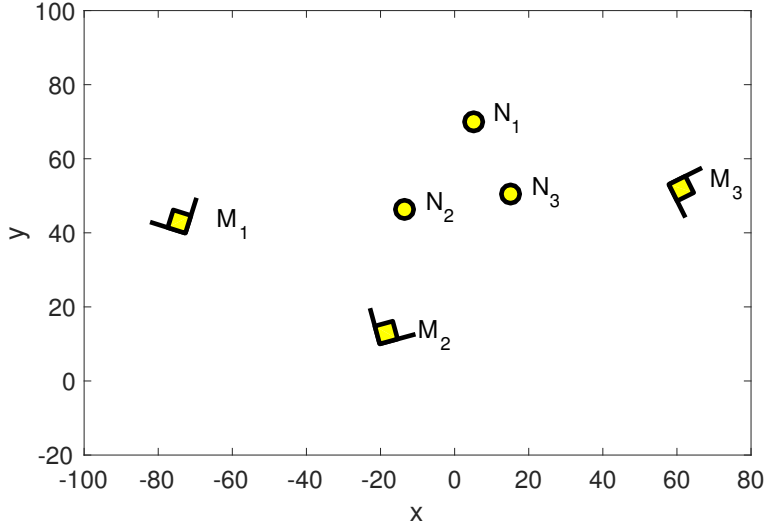
Alpha certified – by Jon Hauenstein and Frank Sottile

- Command-line program.
- See manual for input format.
- Certifies polynomial and poly-exponential systems.
- Soft or Hard certification.

```
alphaCertified v1.3.0 (October 16, 2013)
Jonathan D. Hauenstein and Frank Sottile
       GMP v6.1.2 & MPFR v4.0.1
```

Outline

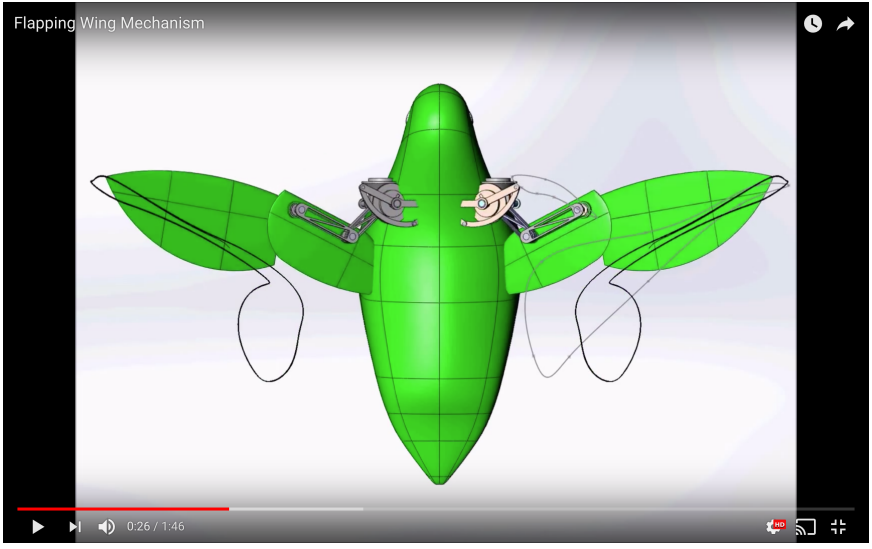
Challenge 1: design the four-bar



Challenge 1: Response

- The preceding set of 9 constraints yields a *complex curve of degree 362*, each point of which is a robotic design.
- We'll come to computing the real curve soon, but for now, add one more constraint to get a discrete set of solutions.

You can compute a bird wing



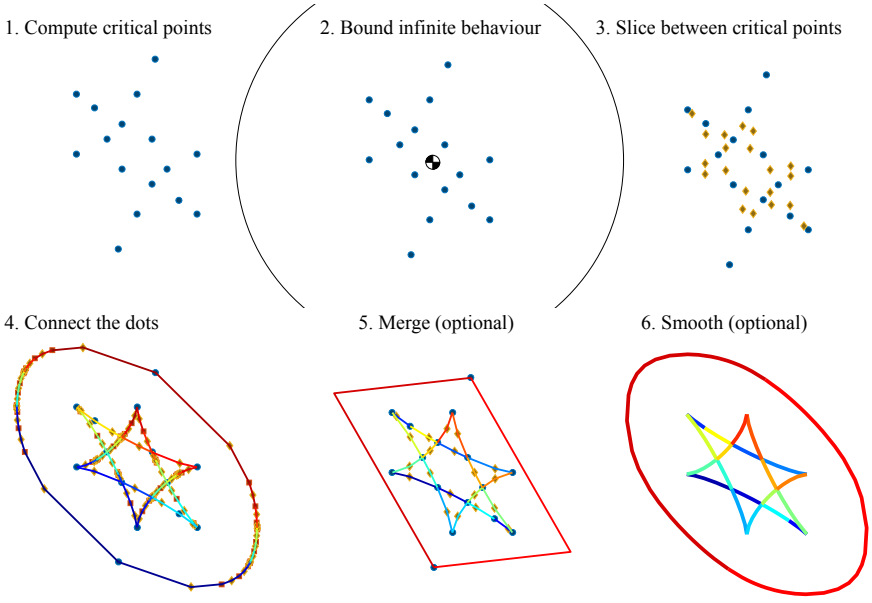
https://www.youtube.com/watch?v=7aXmze9Ynis&feature=youtu.be

Challenge 2 – Compute the real part

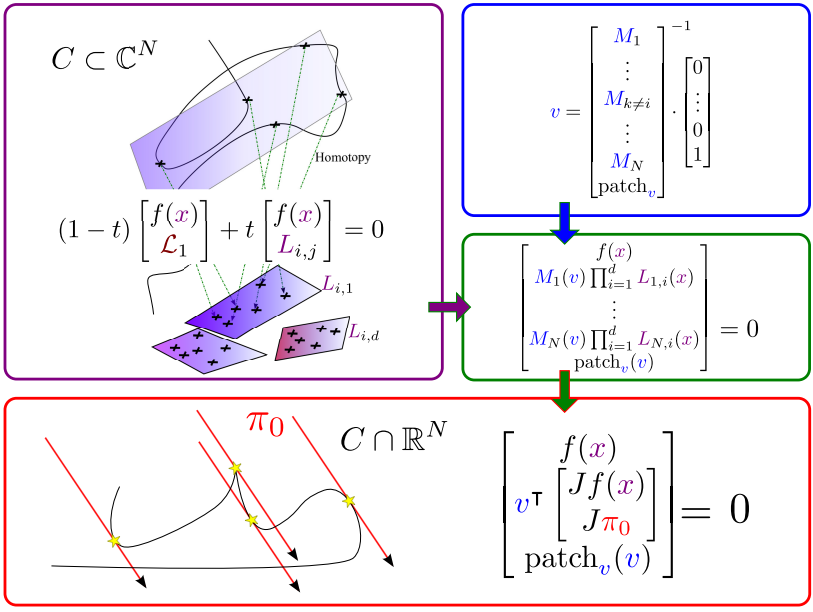
Extracting the real part of an arbitrary complex component is *very* hard.
So far, we can:

- Fully decompose real curves
- Fully decompose real surfaces
- Compute at least one point on every part the real object
- And maybe play tricks to get other information

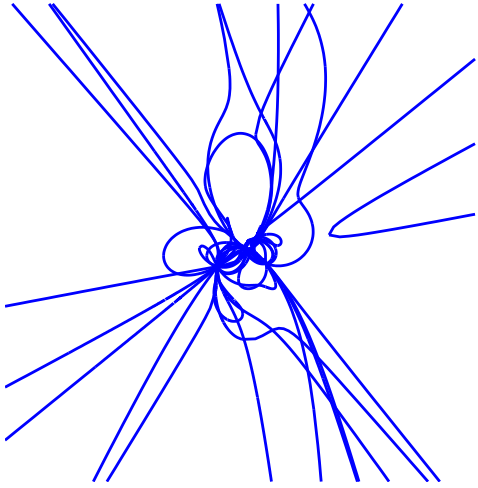
Curve decomposition



Regeneration to compute critical points



Example – Real curve decomposition



A degree 362 curve in 14-dimensional space

A point on every connected component

If

- you just care about knowing whether there are any real solutions
- your variety or component is too complicated – dimension too high, etc

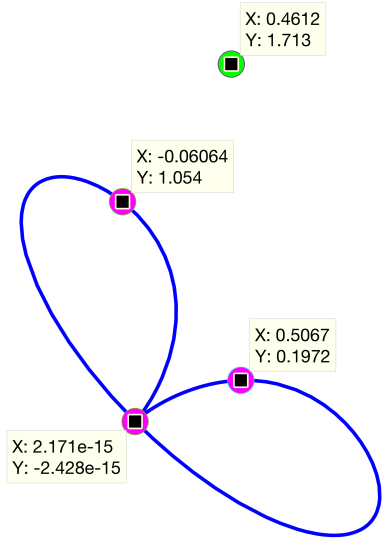
then you should probably settle for computing a point on every component.

A point on every connected component

Solve this homotopy:

$$H(x, \lambda, t) = \begin{bmatrix} f(x) - t\epsilon \\ \lambda_0(x - y) + \sum \lambda_j \nabla f_j(x)^\intercal \\ 1 - \sum \alpha_j \lambda_j \end{bmatrix}$$

- ϵ – a random complex perturbation
- λ_j – synthetic variables, representing nullvector of Jacobian
- α_j – a random complex patch on the λ_j
- y – a real point not on variety
- t – path variable, goes to 0


X: 0.4612
Y: 1.713
X: -0.06064
Y: 1.054
X: 0.5067
Y: 0.1972
X: 2.171e-15
Y: -2.428e-15

Bertini_real

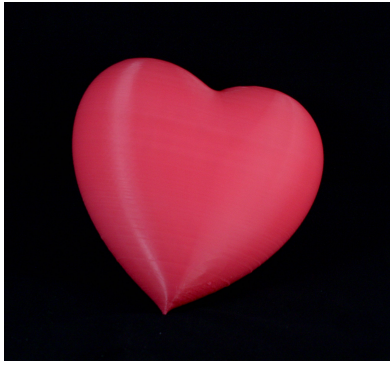
- Command-line MPI-parallel program.
- Uses Bertini 1 as its path tracker.
- C++ code, with options for Matlab or Python for symbolic operations.
- Matlab and Python visualization suites.
- Must self-compile. Must also self-compile $\mathbb{B}^1$. Sorry.

```
BertiniReal(TM) v1.6.0

D.A. Brake with
D.J. Bates, W. Hao, J.D. Hauenstein,
A.J. Sommese, C.W. Wampler

(using GMP v6.1.2, MPFR v3.1.5)


 Library-linked Bertini(TM) v1.6
  (February 27, 2017)

D.J. Bates, J.D. Hauenstein,
A.J. Sommese, C.W. Wampler

(using GMP v6.1.2, MPFR v3.1.5)
```

bertinireal.com

Outline

Thank you for your kind attention!



silviana.org
silviana.amethyst@gmail.com