Isolated Solutions (Lecture 1)

Ex 1: Get started with Bertini by solving $x^5 - x + 1 = 0$.

- Create input file:

```
variable_group x;
function f;
f = x^5 - x + 1;
```

- Run Bertini (>> bertini [input file]) — defaults to "input"

- What happened:
  (1) homogenized: $\mathbb{C} \hookrightarrow \mathbb{P}^1$
  (2) Setup homotopy on $\mathbb{P}^1$: $(x^5 - x \cdot h^4 + h^5) \cdot (1-t) + \gamma \cdot t \cdot (x^5 - h^5)$
  ($\gamma \in \mathbb{C}$ randomly selected)
  (3) Constructed start points for homotopy by solving $x^5 - h^5 = 0$ on $\mathbb{P}^1$
  (4) Tracked the 5 paths
  (5) Post-processed the data: dehomogenized and sorted

- Look at files: output (tracking information), main_data (endpoint data and quality information), finite_solutions (list of numerical approximations deemed to be "finite

Ex 2: Improve accuracy of solutions to $x^5 - x + 1 = 0$:

- FinalTol: used to improve accuracy via the endgame during the solve.

- SharpenDigits: used to improve accuracy via Newton's method after the endgame computed an approximation.

Create input file:

```
CONFIG
    FinalTol: 1e-12;
    SharpenDigits: 50;

END;
INPUT
    variable-group x;
    function f;
    f = x^5 - x + 1;

END;
```

Run Bertini and look at files output, main-data & finite-solutions.

- note: endgame error approximation based on internal coordinates (projective space)

Change tracking by ODEPredictor, TrackTol BeforeEG, TrackTol DuringEG, MaxNumberSteps

Some choices of ODEPredictor:
- -1: constant
- 0: Euler
- 1: Heun
- 2: Runge-Kutta
- ⋮
- 5: Runge-Kutta-Fehlberg
- ⋮
- 8: Runge-Kutta-Verner

```
input-tracker;
MaxNumberSteps: 1000000;
ODEPredictor: -1;
TrackTolBeforeEG: 1e-8;
TrackTolDuringEG: 1e-8;
```

Look at output.

Ex 3: Multihomogeneous system with solution at infinity

$$\begin{bmatrix} x^2 + 2x - 8 \\ xy + 2x + 4y - 3 \end{bmatrix}$$

- Create input file using a 2-hom setup:

```
variable-group x;
variable-group y;
function f1, f2;
f1 = x^2 + 2*x - 8;
f2 = x*y + 2*x + 4*y - 3;
```

This will take $\mathbb{C} \times \mathbb{C} \hookrightarrow \mathbb{P}^1 \times \mathbb{P}^1$ and solve on $\mathbb{P}^1 \times \mathbb{P}^1$.

By default, paths which appear to be going to "infinity" can be truncated.

Control how paths that appear to be going to "infinity" are handled using:

- Security Level: 0 - allow truncation
  1 - no truncation
- Security Max Norm: how "large" should one consider at "infinity" when truncating
- Endpoint Finite Threshold: how to classify "finite" vs. "infinite" sol

```
input-infinite:
    SecurityLevel: 1;
    EndpointFiniteThreshold: 1e10;
```

~ Look at main-data (accuracy estimates)

Ex 4: Multihomogeneous system on product of affine and projective spaces

- Solve on $\mathbb{C} \times \mathbb{P}^1$
  $x \quad \{y, z\}$

$$\begin{bmatrix} x^2 + 2x - 8 \\ xy + 2xz + 4y - 3z \end{bmatrix}$$

```
variable-group x;
hom-variable-group y, z;
function f1, f2;
f1 = x^2 + 2*x - 8;
f2 = x*y + 2*x*z + 4*y - 3*z;
```

Bertini checks for proper homogenization and returns solutions in a local chart where largest coordinate is 1.

Ex 5: Be wary of polynomials with large (or small) coefficients and high degree

. Wilkinson polynomial

```
variable_group x;
function f;
f: (x-1)*(x-2)*(x-3)*(x-4)*(x-5)*(x-6)*(x-7)*(x-8)*
   (x-9)*(x-10)*(x-11)*(x-12)*(x-13)*(x-14)*(x-15)*
   (x-16)*(x-17)*(x-18)*(x-19)*(x-20);
```

Look at CoeffBound & DegreeBound that are printed to screen - cause for concern!

---

input_scaled : rescale f by 1/20!

Look at classification of singular vs. nonsingular (based on CondNumThreshold)

```
input_scaled_sharpen:
    SharpenDigits: 50;
    CondNumThreshold: 1e100;
```

Look at output (adaptive precision & minimum sing. value)

Note: If problem converging with endgame, use more sample points which produced a higher order approximation. (NumSamplePoints) or switch endgame (EndgameNum).

Ex 6: Multiplicity.

- Bertini computes multiplicity based on number of paths with the same end-point (up to tolerance).

  - For square (# variables = # equations) systems, this is the multiplicity.

input-square: (Griewank-Osborne system)

```
variable-group x,y;
function f1,f2;
f1 = 29/16 * x^3 - 2*x*y;
f2 = y - x^2;
```

Multiplicity 3 at origin.

  - For overdetermined (# variables < # equations) systems, Bertini solves a randomized square system (which is num. stable) and returns the multiplicity = number of paths with respect to this square system.

input-overdetermined:

```
variable-group x,y;
function f1,f2,f3;
f1 = x^2;
f2 = x*y;
f3 = y^2;
```

Overdetermined system: mult 3 at origin

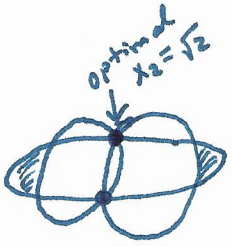
Randomized square system (e.g., $\begin{matrix} f_1 + \alpha \cdot f_3 \\ f_2 + \beta \cdot f_3 \end{matrix}$ $\alpha, \beta \in \mathbb{C}$) : mult 4 at origin = 4 paths ending at origin.

[If difficulty with clustering common endpoints, adjust Target To / Multiplier].

Ex 7: Solving a nonlinear optimization problem
(workflow: solve, sharpen, evaluate)

Use Bertini to solve:

$$\begin{aligned} \max \quad & x_2 \\ \text{s.t.} \quad & (x_1-1)^2 + x_2^2 - 3 \geq 0 \\ & (x_1+1)^2 + x_2^2 - 3 \geq 0 \\ & 2 - x_1^2/10 - x_2^2 \geq 0 \end{aligned}$$



(1) Setup system using Fritz John conditions and solve:

$(x_1, x_2) \in \mathbb{C}^2$

$[\lambda_0, \lambda_1, \lambda_2, \lambda_3] \in \mathbb{P}^3$

input

$$\begin{bmatrix} ((x_1-1)^2 + x_2^2 - 3)\cdot\lambda_1 \\ ((x_1+1)^2 + x_2^2 - 3)\cdot\lambda_2 \\ (2 - x_1^2/10 - x_2^2)\cdot\lambda_3 \\ \begin{bmatrix}0\\1\end{bmatrix}\cdot\lambda_0 + \begin{bmatrix}2(x_1-1)\\2x_2\end{bmatrix}\cdot\lambda_1 + \begin{bmatrix}2(x_1+1)\\2x_2\end{bmatrix}\cdot\lambda_2 + \begin{bmatrix}-2x_1/10\\-2x_2\end{bmatrix}\cdot\lambda_3 \end{bmatrix} =$$

Note that even though the optimizer has a 1-dim'l family of multipliers λ_1, we are guaranteed to compute at least 1 point on this connected component. $(0, \sqrt{2})$

(2) Use the post-processing sharpening module to sharpen the solutions.
Since singular solutions, we need to change to an endgame sharpener.

input-sharpen

```
CONFIG
  SharpenOnly: 1;
END;
```

This uses data from (1) to sharpen the solutions.

(3) Evaluate in high precision to test inequalities and report value of objective function.

- Rename finite_solutions as start_solutions.

> bertini [input file] [start points]

In this case: > bertini input_eval start_solutions

Note: TrackType: -4 : function evaluation
-3 : function & Jacobian evaluation
-2 : Newton iteration
-1 : Newton iteration with condition number estimation

MPType: 0 : double precision
1 : fixed precision (use "Precision" for number of bits)
2 : adaptive precision

Review "function" to determine maximum is $\sqrt{2}$.

Ex 8: User-defined homotopy

This is the most flexible setup in Bertini as one can define homotopy and start points.

2 types: { UserHomotopy:
1 - works on affine space using "variable" (system need not be polynomial!)
2 - works on product of affine and proj. space using "variable-group" and "hom-variable-group".

input-1

```
.....
CONFIG
  UserHomotopy: 1;
END;
INPUT
    variable x;
    function f;
    pathvariable t;  % always goes from t=1 to t=0 along real line
    parameter p;
    p = exp((1-t)* 2 * Pi * I);

    f = x^2 - p;
END;
```

input-2

```
- - - - -
CONFIG
  UserHomotopy: 2;
END;
INPUT
    variable-group x;
    function f;
    pathvariable t;
    parameter p;
    p = exp((1-t)* 2 * Pi * I);
    f = x^2 - p;
END;
```

Create list of start points

```
1
```

There is 1 start point whose coordinate is $1+0\cdot i$.

Start

```
1  0
```

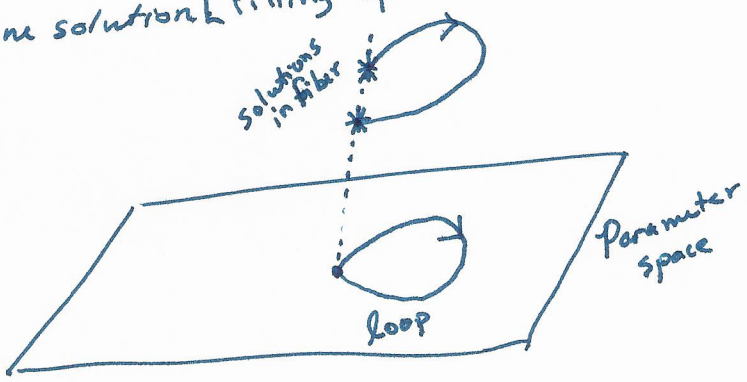
Run Bertini:

```
> bertini input_1 start

> bertini input_2 start
```

Note: Check "output" to verify start point satisfies the system.
Observe difference between the 2 types via internal coordinates.

This was an example of a monodromy loop.
Performing such loops (with various randomizations) is a great way to search for more solutions given one solution ["filling up the fiber"].



Ex 9: Parameter homotopies - perform a straight-line homotopy in a parameter space.

2 step process: { Ab initio phase
Parameter homotopy phase

(1) Ab initio phase (ParameterHomotopy: 1;)

- Picks random parameter values and solves (stored in start-parameters)

input_ab-initio

```
CONFIG
   ParameterHomotopy: 1;
END;
INPUT
   variable_group x;
   variable_group y;
   parameter a1, a2, a3, a4;
   function f1, f2;
   f1 = x^2 - (a1+a2) * x + a1*a2;
   f2 = (x-a1) * y + a3 * x + a4;
END;
```

Keep "start-parameters" and a file containing the start points of interest (must be nonsingular with respect to system).
↳ if singular, use deflation

e.g. rename "nonsingular_solutions" as "start_solutions".

(2) Parameter homotopy phase (ParameterHomotopy:2)

- Track from "start-parameters" to your selected "final-parameters" using the "start_solutions" as start points.

Create "final-parameters":

4

-4 0
2 0
2 0
-3 0

Corresponds with $\begin{bmatrix} x^2+2x-8 \\ xy+2x+4y-3 \end{bmatrix}$

Run Bertini:

> bertini input-parameter start-solutions.

Most problems arise from incorrect start points. Use, e.g., "output" to verify start points correctly satisfy system.

Ex 10: Regeneration

For problems where traditional upper bounds (e.g. Bézout, MultiHom Bézout, BKK = polyhedral) are vast over estimations of the number of isolated nonsingular solutions, an alternative approach is regeneration which computes the isolated nonsingular points in $V(f_1, \ldots, f_n)$ via the sequence $V(f_1) \to V(f_1, f_2) \to V(f_1, f_2, f_3) \to \ldots \to V(f_1, \ldots f_n)$.

Ex: Solve Burmester's problem: 4-bar linkages which are defined by 5 pose constraints (8 equations in 8 variables).

```
CONFIG
   UseRegeneration: 1;
END;
```

If you want to restart regeneration at a particular function, use RegenStartLevel (restart based on previously computed data).

Ex:

```
CONFIG
   UseRegeneration: 1;
   RegenStartLevel: 4;
END;
```

Ex 11 : Eigenvalues

Solve $Ax - \lambda x = 0$ where $A \in \mathbb{C}^{n \times n}$ is given

and $x \in \mathbb{P}^{n-1}$, $\lambda \in \mathbb{C}$.

- n bilinear equations on $\mathbb{C} \times \mathbb{P}^{n-1}$
  - generic # of solutions $= \binom{n}{1} = n$.

Solve using a 2-homogeneous homotopy

↳ guaranteed to find a point on each connected component.

Take a matrix that has Jordan canonical form

$$\begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & 1 & & \\ & & \lambda_2 & & \\ & & & \lambda_3 & \\ & & & & \lambda_3 \end{bmatrix}$$ where λ_i distinct.

So: λ_1 has algebraic = geometric = 1 multiplicities

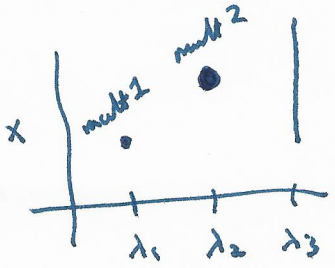
$\Rightarrow$ nonsingular solution to $Ax - \lambda x = 0$.

λ_2 has algebraic mult = 2 and geom. mult = 1

$\Rightarrow$ isolated singular solution to $Ax - \lambda x = 0$

λ_3 has alg. = geom = 2 multiplicities

$\Rightarrow$ pos. dim'l solution to $Ax - \lambda x = 0$.
(singular)



$\Rightarrow$ bertini input

Ex 12 : Rounding Errors

Effects of rounding errors can cause a change in the parameter subspace the coefficients lie in causing possibly unexpected behavior.

$$f(x,y;a,b) = \begin{bmatrix} xy^3 - ay^3 - 1 \\ x^2 - b \end{bmatrix}$$

- 6 solutions for generic $(a,b) \in \mathbb{C}^2$
  >> bertini input
- 3 solutions for $b = a^2$
  >> bertini input-subparameter.

If we take $\begin{cases} a = 1.4142 \\ b = 2 \end{cases}$
what should we expect?
>> bertini input-rounding

$\begin{cases} a = 1.4142 \\ b = a^2 \end{cases}$
expectation?
>> bertini input-rounding-subpa

If we take $\begin{cases} a = 2\wedge(Y_2); \\ b = 2; \end{cases}$
>> bertini input-sqrt2

Things to keep in mind:

- Formulation: problems can often be formulated in many different ways. Think about numerical properties of each formulation:
  - Low degree polynomials
  - Ease of evaluation (do not expand!)
  - Coefficients of ~ unit magnitude
  - Well-constrained system (e.g., for isolated: # equations = # variables)

- Parameter Space: is there a natural parameter space that can be exploited? does the system being solved lie on this parameter space?
  - Round-off errors: impact singular vs. nonsingular and finite vs. infinite.
  - Structure to exploit?
  - Local-to-global methods (e.g., monodromy)