Algunas aplicaciones del Álgebra Computacional. Geometría algebraica numérica

Sofia Ruà

Máster Interuniversitario en Matemáticas

Universidad de Granada

Granada, 2017

Trabajo Fin de Máster

Algunas aplicaciones del Álgebra Computacional. Geometría algebraica numérica

Sofia Ruà

Dirección: Prof. Dr. D. Pascual Jara Martínez

Máster Interuniversitario de Matemáticas
Universidad de Granada
Granada, 2017

Índice general

Introducción

A partir de la aparición del algoritmo de Buchberger que permite calcular bases de Gröbner de ideales de anillos de polinomios, el cálculo simbólico ha experimentado un gran desarrollo tanto teórico como en sus aplicaciones; fundamentalmente en lo que se refiere a la resolución de sistemas de ecuaciones polinomiales. Recientemente, la combinación de análisis numérico y geometría algebraica, la Geometría Algebraica Numérica, ha permitido extender aún más los límites de la resolución de estos sistemas de ecuaciones polinomiales.

La idea básica en Geometría Algebraica Numérica es el uso de los métodos de continuación homotópica para describir las componentes de la solución de sistemas de ecuaciones polinomiales. No sólo se encuentran las soluciones aisladas, sino que también se pueden calcular los conjuntos de soluciones de dimensión positiva para, posteriormente, descomponerlos en sus componentes irreducibles. Los métodos utilizados son *simbólico-numéricos*, ya que métodos numéricos se aplican para encontrar resultados enteros.

En este texto vamos a considerar el software `Bertini`, el cual se puede usar para calcular todas las raíces aisladas de un sistema de polinomios con coeficientes complejos. Es de destacar que `Bertini` usa métodos de continuación homotópica para calcular las soluciones aisladas de sistemas polinomiales. La idea es escribir el sistema polinomial, del que queremos calcular las soluciones, como una familia parametrizada de sistemas, uno de los cuales, el *start system*, tiene soluciones fáciles de calcular. Luego, el programa, escoge un camino desde el *start system* al sistema original, construye una homotopía entre los dos y traza los caminos soluciones usando métodos *predictor-corrector*, empleando un *step size* flexible.

Muchos de los resultados producidos por la Geometría Algebraica Numérica, por ejemplo los obtenidos con `Bertini`, no están certificados, ya que

vienen generados usando métodos eurísticos. Es pues importante certificar que las soluciones halladas lo son realmente. En este trabajo se estudiarán técnicas de certificación para las soluciones aisladas de sistemas polinomiales.

Para sistemas polinomiales cuadrados, usaremos la *α-theory* de Smale para certificar las soluciones aproximadas dadas. *α-theory* garantiza la convergencia cuadrática de la iteración de Newton, empezando en un punto $\mathbf{z} \in \mathbb{C}^n$, si se verifica que una constante $\alpha(f, \mathbf{z})$, donde f es un sistema polinomial cuadrado, es menor de una constante universal dada. Además, *α–theory* proporciona una cota de la distancia entre el punto inicial $\mathbf{z}$ y la posible raíz $\xi \in \mathbb{C}^n$, y garantiza la unicidad de esta raíz para esa distancia. El software `alphaCertified`, implementado por F. Sottile y J. Hauenstein, permite el rápido cálculo de las constantes y la certificación de las soluciones aisladas de un sistema cuadrado, por ejemplo las obtenidas con `Bertini`.

Para sistemas sobredeterminados sobre $\mathbb{Q}$, A. Szanto, J. Hauenstein y T.A. Akoglu proponen una certificación diferente, que se basa en métodos simbólico-numéricos, usando, bien las raíces aproximadas del sistema, bien la división con resto de polinomios en una variable sobre $\mathbb{Q}$, para calcular la *rational univariate representation* (RUR) de una componente del sistema. La idea es, para sistemas polinomiales con raíces simples, calcular una RUR inicial a partir de las raíces aproximadas y luego aumentar la precisión a través de iteraciones de Newton hasta encontrar la RUR exacta. Una vez encontrada ésta, y ya que es un sistema cuadrado, se le puede aplicar α-theory.

En el Capítulo 1 daremos unos conceptos básicos para entender cómo funcionan los métodos de continuación homotópica para encontrar soluciones aisladas y veremos algunos ejemplos usando el software `Bertini`.

En los Capítulos 2 y 3 veremos las técnicas de certificación para las soluciones aisladas de sistemas polinomiales obtenidas con métodos de la Geometría Algebraica Numérica. En el Capítulo 2 estudiaremos α-theory y el software `alphaCertified`, ilustrando el funcionamiento a través de unos ejemplos; y en el Capítulo 3 veremos teóricamente y con unos ejemplos el método de certificación propuesto por A. Szanto y sus colaboradores para la certificación de las raíces aproximadas de sistemas polinomiales sobredeterminados sobre $\mathbb{Q}$.

CAPÍTULO 1

Métodos de continuación homotópica para encontrar soluciones aisladas

En Geometría Algebraica Numérica se aplican métodos de continuación homotópica para describir las componentes de la solución de sistemas polinomiales. No sólo se encuentran las soluciones aisladas, sino también se pueden calcular los conjuntos de soluciones de dimensión positiva para luego descomponerlos en componentes irreducibles. Los métodos utilizados son simbólico-numéricos, ya que métodos numéricos se aplican para encontrar resultados enteros.

1.1. Métodos de continuación homotópica

Sea $f:\ \mathbb{C}^n \to \mathbb{C}^m,\ m \geq n$ un sistema polinomial

$$f(\mathbf{x}) = \begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \qquad \vdots \\ f_m(x_1, \ldots, x_n) = 0 \end{cases}$$

con $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{C}^n$. Si todos los polinomios f_i fuesen lineales, se podrían aplicar los métodos del álgebra lineal y del álgebra lineal númerica. Los polinomios no lineales llevan a situaciones más complicadas: un polinomio de

grado d en n variables tiene un total de $\binom{n+d}{n}$ monomios de grado menor o igual que d.

Los sistemas polinomiales pueden tener conjuntos de soluciones formados por puntos (cero-dimensionales) o de dimensión positiva (curvas, superficies, ...). Vamos a estudiar sólo los sistemas polinomiales que tienen soluciones cero-dimensionales o aisladas. Las soluciones aisladas pueden ser regulares o singulares.

En el caso de polinomios en una variable, el concepto de multiplicidad es fácil de definir: la multiplicidad de una solución x^* de un polinomio $p(x) \in \mathbb{C}[x]$ es k si $p^{(j)}(x^*) = 0$ para $0 \leq j < k$ y $p^{(k)}(x^*) \neq 0$. Las soluciones singulares son las que tienen multiplicidad $k > 1$.

En el caso de polinomios en n variables, la situación se hace un poco más complicada. Consideramos un sistema cuadrado $f : \mathbb{C}^n \to \mathbb{C}^n$.

Definición 1.1. Una solución $\mathbf{x} \in \mathbb{C}^n$ se dice *aislada* si existe $r > 0$ tal que la única solución contenida en

$$B_r(\mathbf{x}) := \{\mathbf{y} \in \mathbb{C}^n \mid \|\mathbf{y} - \mathbf{x}\| < r\}$$

es $\mathbf{x}$.
Una solución $\mathbf{x} \in \mathbb{C}^n$ se dice *regular* si la matriz jacobiana de f en $\mathbf{x}$, $Jf(\mathbf{x})$, tiene rango n. Si no, se dice que $\mathbf{x}$ es *singular*.

Se tiene que una solución regular es aislada pero puede que una solución aislada sea singular: por ejemplo, la solución doble $x = 0$ para $f(x) = x^2$. El estudio de las singularidades es importante en los métodos numéricos: el método de Newton puede divergir o no converger cuadráticamente alrededor de soluciones singulares.

Sistemas sobredeterminados

Las definiciones anteriores valen en el caso de sistemas cuadrados, es decir, cuando $n = m$. Si $m < n$, el sistema tiene necesariamente un conjunto de solución de dimensión positiva. Vamos a ver cómo tratar el caso de sistemas sobredeterminados, es decir, con $m > n$. La técnica estándar para reducir un sistema de m polinomios a uno de $n < m$ polinomios es reemplazar $f_1, \ldots, f_m$ con n combinaciones lineales de los polinomios. Dicho de otra manera, eligimos una matriz aleatoria $A \in \mathbb{C}^{n \times m}$ y reemplazamos el sistema $f : \mathbb{C}^n \to \mathbb{C}^m$ con el sistema $A \cdot f : \mathbb{C}^n \to \mathbb{C}^n$. Esta técnica se llama *randomization*.

El teorema de Bertini ([3], Theorem 1.15) asegura con *probability one* que todas las soluciones aisladas de f son también soluciones aisladas de $A \cdot f$.

Aplicando la *randomization* a un sistema sobredeterminado f, sólo se pueden añadir soluciones aisladas, nunca removerlas: esas soluciones extras, llamadas *nonsolutions*, se pueden detectar fácilmente y descartar ya que no satisfacen f.

Con *probability one* significa que la elección aleatoria de la matriz $A \in \mathbb{C}^{n\times m}$ hace que el teorema de Bertini valga siempre, salvo para algunas matrices A que pero pertenecen a un conjunto de medida cero.

Métodos de continuación homotópica

Los métodos de continuación homotópica constituyen la computación básica en geometría algebraica numérica. Para encontrar las soluciones aisladas de un sistema

$$f(\mathbf{x}) = \begin{cases} f_1(x_1,\ldots,x_n) = 0 \\ \vdots \\ f_n(x_1,\ldots,x_n) = 0 \end{cases}$$

con $\mathbf{x} = (x_1,\ldots,x_n) \in \mathbb{C}^n$, la idea es calcular todas las soluciones aisladas de un sistema fácil de resolver $g(\mathbf{x})$ y luego, usando la continuación homotópica, deformar el sistema $g(\mathbf{x})$ con sus soluciones en el sistema $f(\mathbf{x})$ con sus soluciones.

En los métodos de continuación homotópica hay tres pasos:

1. Construir y resolver un *start system* $g(\mathbf{x})$. Se utiliza la estructura del sistema $f(\mathbf{x}) = 0$ para encontrar un *root count* y construir un *start system* $g(\mathbf{x}) = 0$ que tiene exactamente el mismo número de soluciones regulares del *root count*. La elección más simple es la homotopía de grado total (1.2).

2. Construir una homotopía entre $f(\mathbf{x})$ y $g(\mathbf{x})$. La manera más simple viene dada por

$$h(\mathbf{x},t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = 0, \quad t \in [0,1], \tag{1.1}$$

   con $\gamma \in \mathbb{C}$. Obtenemos caminos que unen los *startpoints*, las soluciones de $g(\mathbf{x}) = 0$, a los *endpoints*. Cada camino es la imagen de una aplicación continua $p : [0,1] \to \mathbb{C}^n$: se tiene $h(p(t),t) = 0$, para $t \in [0,1]$.

3. *Path tracking*. El *path tracking* es un proceso numérico para aproximar los caminos, obteniendo así aproximaciones de las soluciones de $h(\mathbf{x},1) = 0$ a partir de las soluciones aproximadas, conocidas, de $h(\mathbf{x},0) = 0$.

Las propiedades que nos esperamos de una homotopía ([6],Theorem, pag.60) son:

1. Trivialidad: las soluciones para $t = 0$ son fáciles de encontrar;
2. Regularidad: no hay singularidades a lo largo de los caminos soluciones;
3. Se llega a una solución aislada de multiplicidad m con exactamente m caminos.

Los métodos de continuación son técnicas numéricas estándar para trazar caminos soluciones usando métodos *predictor-corrector*. La propiedad de regularidad de las homotopías polinomiales complejas asegura que el camino nunca vuelve atrás, así que durante la corrección se puede suponer que el parámetro t esté fijo.

La ventaja de estos métodos es que los algoritmos no requieren un conocimiento a priori de las soluciones del sistemas, es decir, no hacen falta *start points*.

Vamos a dar una idea de como funciona con un ejemplo sencillo en una variable.

Ejemplo 1.1. Consideramos la ecuación

$$f(x) = x^2 + 3x - 4 = 0,$$

y tomamos como *start system*

$$g(x) = x^2 - 4 = 0.$$

Las soluciones de $g(x) = 0$ son sencillas de calcular y son $x = \pm 2$. Queremos resolver $f(x) = 0$ y encontrar una manera de calcular las soluciones de $f(x) = 0$ a partir de las de $g(x) = 0$.
Definimos la ecuación

$$h(x, t) = x^2 + 3tx - 4 = 0.$$

Observamos que $h(x, 0) = g(x)$ y $h(x, 1) = f(x)$. Como t varía entre 0 y 1, $h(x, t) = 0$ cambia de $g(x) = 0$ a $f(x) = 0$. Vamos a eligir un incremento $\Delta t = 0.1$. Con $x = 2$ como punto inicial, vamos a aplicar el método de Newton a $h(x, \Delta t) = 0$, que es una ecuación en una variable, x:

$$h(x, \Delta t) = x^2 + 0.3x - 4 = 0.$$

Como Δt es pequeño, se puede suponer que las soluciones de $h(x,0) = 0$ estén cerca de las de $h(x, \Delta t) = 0$. Con ello, el método de Newton converge rápidamente, encontrando una solución x_1 de $h(x, \Delta t) = 0$. Luego, resolvemos $h(x, 2\Delta t) = 0$ usando x_1 como punto inicial, obteniendo una solución x_2. De esta manera se sigue hasta llegar a $\Delta t \approx 1$.
Análogamente, se encuentra la otra solución de $f(x) = 0$ a partir de $x = -2$. El código en `Mathematica` para resolver la ecuación:

```
>h[x_, t_] := x^2 + 3*t*x - 4;
>Cont[x_, DeltaT_] := Module[{X, k, F, sol, H}, k = 1; sol = x;
  While[k*DeltaT <= 1, H = h[X, k*DeltaT];
    F = FindRoot[H == 0, {X, sol}, Method -> "Newton"];
    sol = X /. F; k += 1]
   Print[sol];]
>Cont[2, 0.1]
 1.
>Cont[-2, 0.1]
 -4.
```

Homotopía de grado total

Consideramos un sistema cuadrado $f = (f_1, \ldots, f_n)$ con $d_i = \deg(f_i)$, se construye un *start system* $g(\mathbf{x}) = 0$ de la siguiente manera:

$$g(\mathbf{x}) = \begin{cases} \alpha_1 x_1^{d_1} - \beta_1 = 0 \\ \alpha_2 x_2^{d_2} - \beta_2 = 0 \\ \quad \vdots \\ \alpha_n x_n^{d_n} - \beta_n = 0 \end{cases} \tag{1.2}$$

donde $\alpha_i, \beta_i \in \mathbb{C}$, para $i = 1, \ldots, n$. Con ello, $g(\mathbf{x}) = 0$ tiene exactamente tantas soluciones regulares como el grado total $D = \prod_{i=1}^{n} d_i$. Por tanto, la homotopía

$$h(\mathbf{x}, t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = 0, \quad t \in [0,1], \quad \gamma \in \mathbb{C},$$

usando $g(\mathbf{x})$ como en (1.2), define D caminos soluciones. De hecho, el Teorema de Bézout predice D como el número de soluciones en el espacio proyectivo complejo.

Dada una solución aislada $\mathbf{x}^*$ de $f(\mathbf{x}) = 0$, la homotopía de grado total asegura que el número de caminos que acaban en $\mathbf{x}^*$ es igual que la multiplicidad de $\mathbf{x}^*$ como solución de $f(\mathbf{x}) = 0$ ([3], Theorem 3.1).

Si el sistema a resolver es sobredeterminado, le aplicamos la *randomization* obteniendo un sistema cuadrado F y una homotopía $h(\mathbf{x}, t)$ como en (1.2). Dada una solución aislada $\mathbf{x}^*$ de $F(\mathbf{x}) = 0$, el número de caminos que terminan en $\mathbf{x}^*$ es igual que la multiplicidad de $\mathbf{x}^*$ como solución de $F(\mathbf{x}^*) = 0$. Por el teorema de Bertini, puede ser mayor de la multiplicidad de $\mathbf{x}^*$ como solución del sistema original: por tanto el número de caminos que llevan a $\mathbf{x}^*$ es una cota superior para la multiplicidad de $\mathbf{x}^*$.

1.2. Regularidad y acotación de las soluciones

Para una resolución numérica, necesitamos que los caminos soluciones sean libres de singularidades. Una singularidad aparece cuando la matriz jacobiana Jh de la homotopía $h(\mathbf{x}, t) = 0$ tiene determinante nulo. Las singularidades a lo largo de los caminos soluciones son soluciones del sistema

$$\begin{cases} h(\mathbf{x}, t) = 0 \\ \det(Jh(\mathbf{x}, t)) = 0 \end{cases} \tag{1.3}$$

Si el sistema tiene alguna raíz con $t \in [0, 1)$, entonces hay al menos un camino solución con singularidades. Para resolver el problema, se elige una constante compleja aleatoria γ. Por razones numéricas, generalmente se escoge la constante γ con $\|\gamma\| \approx 1$, es decir, $\gamma = e^{i\theta}$, para $\theta \in \mathbb{R}$ aleatorio. De hecho, restringiendo γ a la circunferencia unitaria, el proceso sigue funcionando con *probability one* ya que el número finito de malas semirectas en $\mathbb{C}$ interseca la circunferencia unitaria en un número finito de puntos que hay que evitar.

La elección aleatoria de la constante γ hace que todas las raíces del sistema (1.3) yacen fuera del intervalo $[0, 1)$, salvo para algunos valores de γ que pero pertenecen a un conjunto de medida cero (por el Principio de Independencia en la elección de los coeficientes del sistema 1.3, [6]).

Ejemplo 1.2. Sea

$$f(x, y) = \begin{cases} x^2 + 4y^2 - 4 \\ 2y^2 - x \end{cases}$$

el sistema polinomial que queremos resolver y tomamos como *start system* el sistema

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

Consideramos la homotopía

$$h(x, y, t) = g(x, y)(1 - t) + f(x, y)t = 0. \tag{1.4}$$

Vamos ahora a ver si la homotopía tiene alguna singularidad. Resolvemos el sistema

$$\begin{cases} h(x,y,t) = 0 \\ \det(Jh(x,y,t)) = 8t^2y + 4xy + 4txy = 0 \end{cases}$$

```
>f = {x^2 + 4*y^2 - 4, 2*y^2 - x};
>g = {x^2 - 1, y^2 - 1};
>h = t*f + (1 - t)*g;
>eh = Expand[h];
>jh = {{D[eh[[1]], x], D[eh[[1]], y]}, {D[eh[[2]], x],
   D[eh[[2]], y]}} // MatrixForm;
>sys = {eh[[1]], eh[[2]], Det[jh]};
>gb = GroebnerBasis[sys, {x, y, t}, {x, y}]
 {-1 + t - 5 t^2 + 10 t^3 + 13 t^4 + 29 t^5 + 21 t^6 + 12 t^7}
>NSolve[gb]
 {{t -> -0.881854 - 0.9177 I}, {t -> -0.881854 + 0.9177 I},
  {t -> -0.20116 - 0.887729 I}, {t -> -0.20116 + 0.887729 I},
  {t -> 0.00685376 - 0.392797 I}, {t -> 0.00685376 + 0.392797 I},
  {t -> 0.40232}}
```

Como t varía entre 0 y 1, en $t = 0.40232$ vamos a encontrar una singularidad. Resolvemos el problema eligiendo como constante $\gamma = 1 + i$ y consideramos la homotopía

$$h(x,y,t) = \gamma\, g(x,y)(1-t) + f(x,y)t = 0.$$

```
>H = t*f + (1 + I)*(1 - t)*g;
>eH = Expand[H];
>jH = {{D[eH[[1]], x], D[eH[[1]], y]}, {D[eH[[2]], x],
   D[eH[[2]], y]}} // MatrixForm;
>sys2 = {eH[[1]], eH[[2]], Det[jH]};
>gb2 = GroebnerBasis[sys2, {x, y, t}, {x, y}]
 {(2 - 2 I) - (14 - 2 I) t + (22 + 10 I) t^2 - (5 + 10 I) t^3 +
  (4 - 24 I) t^4 - (34 - 8 I) t^5 + (3 + 15 I) t^6 + (2 + I) t^7}
>NSolve[gb2]
 {{t -> -4.90598 - 2.67493 I}, {t -> -0.628098 + 0.665272 I},
  {t -> -0.229099 - 2.01927 I}, {t -> 0.285571 - 0.278505 I},
  {t -> 0.371627 - 0.767499 I}, {t -> 0.420395 - 0.532027 I},
  {t -> 0.485584 + 0.20696 I}}
```

La elección de γ ha hecho que todas las raíces ahora son complejas y yacen fuera del intervalo $[0, 1)$.

La misma constante aleatoria γ asegura que todos los caminos estén acotados para cada $t \in [0, 1)$. Esto significa que ningún camino diverge al infinito para $t \in [0, 1)$ o, equivalentemente, el sistema $h(\mathbf{x}, t) = 0$ no tiene soluciones en el infinito para cada $t \in [0, 1)$. El marco más adecuado para este estudio lo proporciona el espacio proyectivo.

Para verlo, hacemos una transformación a coordenadas homogéneas introduciendo una nueva coordenada X_0 y consideramos el sistema en el espacio proyectivo. Pasar a coordenadas homogéneas permite estabilizar el cálculo númerico y transforma caminos divergentes de longitud infinita en el espacio afín en caminos de longitud finita en el espacio proyectivo. Esto permite que un camino de este tipo pueda ser trazado numéricamente hasta su punto final. Además, el cálculo del *endpoint* de un camino en el espacio proyectivo aporta una manera más fiable para determinar si un camino diverge o no.
Cada solución que no diverge al infinito luego se puede deshomogeneizar para encontrar su valor correspondiente en las coordenadas afines originales. La deshomogeneización es el simple proceso de dividir por la coordenada X_0 y eliminarla: por supuesto este proceso está definido solo para las soluciones finitas, es decir, las que tienen $X_0 \neq 0$.

Consideramos el sistema homogeneizado $H(X_0, \mathbf{X}, t) = 0$ y

$$\begin{cases} H(X_0, \mathbf{X}, t) = 0 \\ X_0 = 0 \end{cases} . \tag{1.5}$$

Como H es homogéneo en $X_0, \mathbf{X} = (X_1, \ldots, X_n)$, las soluciones se encuentran en el espacio proyectivo. Al aplicar eliminación a $H(0, \mathbf{X}, t) = 0$ se concluye que, salvo para valores especiales de γ, no hay soluciones al infinito para $t \in [0, 1)$. Los valores especiales de γ pertenecen a un conjunto de medida cero $A \subset \mathbb{C}$.
Observamos que, si los polinomios en el *start system* $g(\mathbf{x}) = 0$ tienen grado menor que sus equivalentes en $f(\mathbf{x}) = 0$, entonces $H(X_0, \mathbf{X}, t) = 0$ puede que tenga soluciones al infinito para $t = 0$.

En resumen, dadas las coordenadas afines $\mathbf{x} = (x_1, \ldots, x_n)$, un polinomio $f(\mathbf{x}) \in \mathbb{C}[x_1, \ldots, x_n]$ de grado d viene homogeneizado como

$$\overline{f}(X_0, X_1, \ldots, X_n) = X_0^d f(X_1/X_0, \ldots, X_n/X_0).$$

Cada solución $[X_0, \ldots, X_n]$ de $\overline{f} = 0$ con $X_0 \neq 0$ se dice finita y se deshomogeneiza como

$$(x_1, \ldots, x_n) = (X_1/X_0, \ldots, X_n/X_0).$$

Ejemplo 1.3. Consideramos la homotopía

$$h(x,y,t) = \left(\left\{ \begin{array}{l} x^2-1=0 \\ y^2-1=0 \end{array} \right) (1-t) + \left(\left\{ \begin{array}{l} y^2-1=0 \\ x^2-3=0 \end{array} \right) t.$$

Vamos a ver para cuales valores de t hay caminos soluciones que divergen en el infinito. Consideramos el sistema homogeneizado $H(X,Y,Z,t)=0$ obtenido quitando Z de los denominadores en $h(X/Z, Y/Z, Z, t) = 0$ y resolvemos el sistema (1.5).

```
>f = {y^2 - 1, x^2 - 3};
>g = {x^2 - 1, y^2 - 1};
>h = t*f + (1 - t)*g;
>eh = Expand[h];
>ehZ = eh /. {x -> x/z, y -> y/z};
>ehZ = Simplify[ehZ*z^2]
 {-(-1 + t) x^2 + t y^2 - z^2, y^2 - z^2 + t (x^2 - y^2 - 2 z^2)}
```

Como h es homogéneo en X, Y, Z, las soluciones se encuentran en el espacio proyectivo; por tanto, las soluciones de $H(X,Y,0,t)=0$ satisfacen también $H(X/Y, 1, 0, t) = 0$ o $H(1, Y/X, 0, t) = 0$.

```
>ehZ = ehZ /. {x -> x/y, y -> 1, z -> 0};
>gb = GroebnerBasis[ehZ, {x, y, t}, {x, y}]
 {-1 + 2 t}
>NSolve[gb == 0, t]
 {{t -> 0.5}}
```

Se tiene entonces que para $t = 0.5$ hay un camino que diverge a infinito. Para evitar eso, análogamente al ejemplo anterior, consideramos la homotopía:

$$h(x,y,t) = \gamma \left(\left\{ \begin{array}{l} x^2-1=0 \\ y^2-1=0 \end{array} \right) (1-t) + \left(\left\{ \begin{array}{l} y^2-1=0 \\ x^2-3=0 \end{array} \right) t,$$

con $\gamma = 1 + i$.

```
>H = t*f + (1 + I)*(1 - t)*g;
>eH = Expand[H];
>eHZ = eH /. {x -> x/z, y -> y/z};
>eHZ = Simplify[eHZ*z^2]
 {(-1 - I) (-1 + t) x^2 - (1 + I) z^2 + t (y^2 + I z^2),
 t (x^2 - (1 + I) y^2 - (2 - I) z^2) + (1 + I) (y^2 - z^2)}
```

Como antes,

```
>eHZ = eHZ /. {x -> x/y, y -> 1, z -> 0};
>gb2 = GroebnerBasis[eHZ, {x, y, t}, {x, y}]
 {2 - 4 t + (2 + I) t^2}
>Solve[gb2 == 0, t]
 {{t -> 0.6 + 0.2 I}, {t -> 1. - 1. I}}
```

Con la constante compleja γ ya no hay caminos divergentes con $t \in [0, 1)$.

1.3. Técnicas de continuación

Para trazar caminos soluciones definidos por la homotopía se usan métodos *predictor-corrector*. El *predictor* devuelve en cada paso del método un nuevo valor del parámetro de continuación y predice una solución aproximada del correspondiente nuevo sistema en la homotopía. Luego, la solución aproximada viene corregida aplicándole el *corrector*, por ejemplo el método de Newton. El parámetro de continuación puede quedarse fijo durante la corrección de la solución y eso lleva a los llamados métodos de continuación *increment-and-fix*.

Supongamos que tenemos una familia de funciones $h : \mathbb{C}^n \times [0, 1] \to \mathbb{C}^n$ y una curva diferenciable $\mathbf{x} : [0, 1] \to \mathbb{C}^n$ tal que

- $h(\mathbf{x}(t), t) = 0$, para $t \in [0, 1)$;
- la matriz jacobiana de h con respecto a $\mathbf{x} = (x_1, \ldots, x_n)$ tiene rango n en $(\mathbf{x}(t), t)$, con $t \in [0, 1)$.

Estamos interesados en ver como $\mathbf{x}$ cambia al variar de t: aplicamos el operador $\frac{\partial}{\partial t}$ a la homotopía y obtenemos la ecuación diferencial de Davidenko:

$$\frac{\partial h(\mathbf{x}(t), t)}{\partial t} + \sum_{i=1}^{n} \frac{\partial h(\mathbf{x}(t), t)}{\partial x_i} \frac{\partial x_i(t)}{\partial t} = 0, \text{ con } \mathbf{x}(0) = p_0. \tag{1.6}$$

Con ello, trazar un camino equivale a resolver este problema de valor inicial.

Sea $Jh(\mathbf{x}, t)$ la matriz jacobiana con respecto a las variables $\mathbf{x}$ evaluada en $(\mathbf{x}, t)$ y sea $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ una solución de (1.6). Podemos reescribir la ecuación diferencial de Davidenko como

$$\frac{\partial h(\mathbf{x}(t), t)}{\partial t} + Jh(\mathbf{x}(t), t) \cdot \frac{\partial \mathbf{x}(t)}{\partial t} = 0, \tag{1.7}$$

o, si $Jh(\mathbf{x}, t)$ es invertible a lo largo del camino,

$$\frac{\partial \mathbf{x}(t)}{\partial t} = -Jh^{-1}(\mathbf{x}(t), t)\frac{\partial h(\mathbf{x}(t), t)}{\partial t}. \tag{1.8}$$

Existen diferentes métodos númericos para resolver este problema de valor inicial, que tiene dos características principales:

- El camino construido satisface una función explícita $h(\mathbf{x}(t), t) = 0$ y con ello la función $h(\mathbf{x}, t)$ se puede usar para corregir el resultado del método.
- El camino se encuentra en una curva algebraica y por tanto se puede construir un modelo local de la curva cerca de un punto $\mathbf{p}^*$, modelo que resulta particularmente útil si $\mathbf{p}^*$ es una solución singular de $h(\mathbf{x}, 1) = 0$.

Si $\mathbf{p}^*$ es una solución singular de $h(\mathbf{x}, 1) = 0$, la matriz jacobiana de h con respecto a $\mathbf{x}$ se hace singular para $t \to 1$. Se usan métodos diferentes cuando $t \approx 1$ y cuando t está lejos del 1: la continuación con $t \approx 1$ se dice *endgame* ([3]: métodos de Cauchy, expansión en serie de potencias).

El método Euler-Newton

Un ejemplo de método *predictor-corrector* viene dado por el método Euler-Newton: primero se aplica el método de Euler para encontrar la siguiente aproximación y luego a esa se le aplica el método de Newton para corregirla.

Se resuelve (1.8) empezando en $t_0 = 0$ con p_0 como valor inicial y calculando en seguida las sucesivas aproximaciones $p_1, p_2, \ldots$ en $t_1 < t_2 < \cdots < 1$. El método de Euler calcula la aproximación

$$\mathbf{p}_{i+1} = \mathbf{p}_i - Jh^{-1}(\mathbf{p_i}, t_i)\frac{\partial h(\mathbf{p}_i, t_i)}{\partial t}\Delta t_i,$$

donde $\Delta t_i = t_{i+1} - t_i$. Geométricamente corresponde a predecir a lo largo de la recta tangente al camino solución en el punto actual del camino.

Una vez obtenida una aproximación de $\mathbf{p}_{i+1}$, le sigue una corrección por el método de Newton para $h(\mathbf{x}, t_{i+1})$ empezando en $\mathbf{x}_0 = \mathbf{p}_{i+1}$. El método de Newton viene dado por la fórmula iterativa

$$\mathbf{z}_{i+1} = \mathbf{z}_i - Jh^{-1}(\mathbf{z_i}, t_{i+1})h(\mathbf{z}_i, t_i).$$

Generalmente una o dos iteraciones son suficientes para obtener una predicción más precisa de $\mathbf{z}(t_{i+1})$. A $\mathbf{p}_{i+1}$ se le sustituye el valor corregido antes de empezar un nuevo ciclo *predictor-corrector*.

1.4. Homotopías poliédricas

Vamos a tratar primero el caso de polinomios en una variable. Consideramos el siguiente problema:

In: k monomios distintos en una variable x:
$x^{a_1}, \ldots, x^{a_k}$, con $a_i \neq a_j$ para $i \neq j$.

Out: coeficientes $c_{a_1}, \ldots, c_{a_k}$ tales que $f(x) = c_{a_1}x^{a_1} + \cdots + c_{a_k}x^{a_k}$ tiene $k-1$ raíces reales positivas.

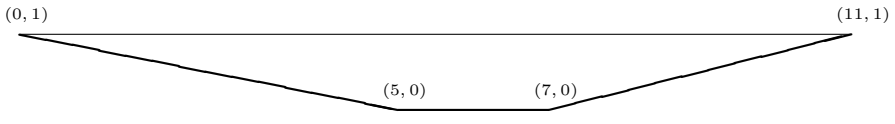
El *politopo de Newton* de una homotopía $h(x,t) = 0$ es el polígono convexo generado por los vectores exponentes de los monomios en $h(x,t) = 0$. A cada lado de la parte inferior del politopo se le hace corresponder una homotopía para encontrar una raíz real positiva. Para encontrar las homotopías hay que considerar los vectores ortogonales a los lados, las llamadas *normales interiores*.

Ejemplo 1.4. Consideramos los monomios $1, x^5, x^7, x^{11}$. Queremos encontrar coeficientes c_0, c_5, c_7, c_{11} tales que $f(x) = c_0 + c_5x^5 + c_7x^7 + c_{11}x^{11}$ tiene exactamente 3 soluciones reales positivas.
Se puede reducir el problema a uno unidimensional, considerando la homotopía

$$h(x,t) = t - x^5 + x^7 - x^{11}t = 0, \quad \text{para } t \geq 0. \tag{1.9}$$

La alternancia de los signos en los coeficientes permite maximizar el número de raíces positivas. El politopo de Newton de la homotopía $h(x,t)$ está generado por $\{(0,1),(5,0),(7,0),(11,1)\}$. La elección de las potencias de t es tal que la parte inferior del politopo de Newton de h contiene entre sus vértices todos los exponentes de los monomios dados. Con el software `Mathematica`

```
>h[X_, T_] := T - X^5 + X^7 - X^(11)*T
>hom = h[x, t]
 t - x^5 + x^7 - t x^11
```



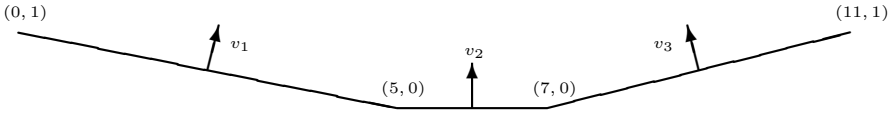
En $t = 0$, la homotopía $h(x,0) = -x^5(1-x^2) = 0$ tiene una raíz real positiva $x = 1$. Eligimos $t = \Delta t = 0.1$ tal que el método de Newton aplicado

a $h(x, \Delta t) = 0$ converge cuadráticamente a una raíz real positiva empezando con $x = 1$. En este caso se podría escoger Δt arbitrariamente grande ya que $h(1, t) \equiv 0$ para cada valor de t. Observamos además que los monomios en $h(x, 0)$ corresponden al lado central inferior del politopo de Newton.

```
>Solve[h[x, 0] == 0, x]
 {{x -> -1}, {x -> 0}, {x -> 0}, {x -> 0}, {x -> 0}, {x -> 0},
  {x -> 1}}
>FindRoot[h[x, 0.1], {x, 1}, Method -> "Newton"]
 {x -> 1.}
```

Para cada lado de la parte inferior del politopo vamos a usar una homotopía y para encontrarlas necesitamos las normales interiores, dadas por

$$\begin{aligned} v_1 &= (\tfrac{1}{5}, 1), \\ v_2 &= (0, 1), \\ v_3 &= (-\tfrac{1}{4}, 1). \end{aligned}$$



Consideramos v_1 y vemos que verifica el producto escalar mínimo con los vértices que generan el primer lado de la parte inferior del politopo:

$$\langle \left(\frac{1}{5}, 1\right), \{(0,1), (5,0), (7,0), (11,1)\} \rangle = \left\{1, 1, \frac{7}{5}, \frac{16}{5}\right\}.$$

Hacemos ahora el cambio de coordenadas $x = yt^{1/5}$ y obtenemos

```
>hy1 = hom /. {x -> y*t^(1/5)}
 t - t y^5 + t^(7/5) y^7 - t^(16/5) y^11
>hy11 = Collect[hy1/t, y] /. {t -> 0}
 1 - y^5
>Solve[hy11 == 0, y]
 {{y -> 1}, {y -> -(-1)^(1/5)}, {y -> (-1)^(2/5)},
  {y -> -(-1)^(3/5)}, {y -> (-1)^(4/5)}}
>FindRoot[h[y, 0.1], {y, 1}, Method -> "Newton"]
 {y -> 1.}
```

Observamos que $\frac{1}{t}h(y,0) = 1 - y^5 = 0$ tiene una raíz real positiva $y = 1$. Como antes, eligimos $t = \Delta t = 0.1$ tal que el método de Newton aplicado a $h(y, \Delta t) = 0$ converge cuadráticamente a una raíz real positiva y^* empezando con $y = 1$ y calculamos $x^* = y^*(\Delta t)^{1/5}$.

```
>sol1 = y*t^(1/5) /. {y -> 1, t -> 0.1}
 0.630957
```

Consideramos ahora v_3 y análogamente vemos que verifica el producto escalar mínimo con los vértices que generan el último lado de la parte inferior del politopo:

$$\langle\left(-\frac{1}{4},1\right),\{(0,1),(5,0),(7,0),(11,1)\}\rangle = \left\{1,-\frac{5}{4},-\frac{7}{4},-\frac{7}{4}\right\}.$$

Con el cambio de coordenadas $x = yt^{-1/4}$ obtenemos

```
>hy2 = hom /. {x -> y*t^(-1/4)}
 t - y^5/t^(5/4) + y^7/t^(7/4) - y^11/t^(7/4)
>hy22 = Collect[hy2/t^(-7/4), y] /. {t -> 0}
 y^7 - y^11
>Solve[hy22 == 0, y]
 {{y -> -1}, {y -> 0}, {y -> 0}, {y -> 0}, {y -> 0}, {y -> 0},
  {y -> 0}, {y -> 0}, {y -> -I}, {y -> I}, {y -> 1}}
>FindRoot[h[y, 0.1], {y, 1}, Method -> "Newton"]
 {y -> 1.}
```

Observamos que $t^{7/4}h(y,0) = y^7(1 - y^4) = 0$ tiene una raíz real positiva $y = 1$. Análogamente, eligimos $t = \Delta t = 0.1$ tal que el método de Newton aplicado a $h(y, \Delta t) = 0$ converge cuadráticamente a una raíz real positiva y^* empezando con $y = 1$ y calculamos $x^* = y^*(\Delta t)^{-1/4}$.

```
>sol2 = y*t^(-1/4) /. {y -> 1, t -> 0.1}
 1.77828
```

Para $\Delta t = 0.1$, $h(x, 0.1)$ tiene tres raíces reales positivas aproximadas:

```
{1, 0.630957, 1.77828}
```

1.4.1. El Cayley trick y el teorema de Minkowski

Vamos ahora a ver como construir homotopías poliédricas cuando el sistema que queremos resolver tiene más que una variable. Para eso necesitamos definir el volumen mixto de r regiones convexas de $\mathbb{R}^n$ y enunciar el Teorema

de Minkowski.

Sean $K_1, \ldots, K_r$ regiones convexas de $\mathbb{R}^n$.

Definición 1.2. El *volumen mixto* de $K_1, \ldots, K_r$ se define como

$$V(K_1, \ldots, K_r) = \frac{1}{n!} \frac{\partial^r}{\partial\lambda_1 \cdots \partial\lambda_r}\Big|_{\lambda_1=\cdots=\lambda_r=0} \mathrm{Vol}(\lambda_1 K_1 + \cdots + \lambda_r K_r),$$

donde $\lambda_1 K_1 + \cdots + \lambda_r K_r$ es la suma de Minkowski y $\lambda_i \geq 0,\ i = 1, \ldots, r$.

Teorema 1.1 (Teorema de Minkowski)**.** *Sean $K_1, \ldots, K_r$ regiones convexas de $\mathbb{R}^n$. $f(\lambda_1, \ldots, \lambda_r) := \mathrm{Vol}(\lambda_1 K_1 + \cdots + \lambda_r K_r)$ es un polinomio homogéneo en los coeficientes de la combinación.*

En particular, los coeficientes de la combinación son volúmenes mixtos.

El Cayley trick es un método que permite construir una homotopía calculando los volúmenes mixtos. La versión para homotopías poliédricas se debe a B. Sturmfels.
El *politopo de Cayley* de r politopos es la región convexa de los politopos situados en los vértices de un simplex unitario de dimensión $r - 1$.

Vamos a construir la homotopía poliédrica para un sistema en dos variables.

Ejemplo 1.5. Consideramos el sistema

$$f(x, y) = \begin{cases} x^2 - xy + 2 = 0 \\ y - xy^2 - 1 = 0 \end{cases}$$

cuyos vectores exponentes son

$$\mathcal{A} = (A_1, A_2), \quad \text{con} \quad \begin{array}{l} A_1 = \{(2,0), (1,1), (0,0)\} \\ A_2 = \{(0,1), (1,2), (0,0)\} \end{array}$$

Los politopos de Newton son las regiones convexas de los soportes A_1 y A_2. El politopo de Cayley lo vemos en la Figura 1.1.

Imaginamos hacer cortes paralelos a la base del politopo de Cayley: esos cortes producen regiones mixtas (con dos vértices pertenecientes al mismo polígono) y copias escaladas de los polígonos originales en símplices no mixtos (los tres vértices pertenecen al mismo polígono).

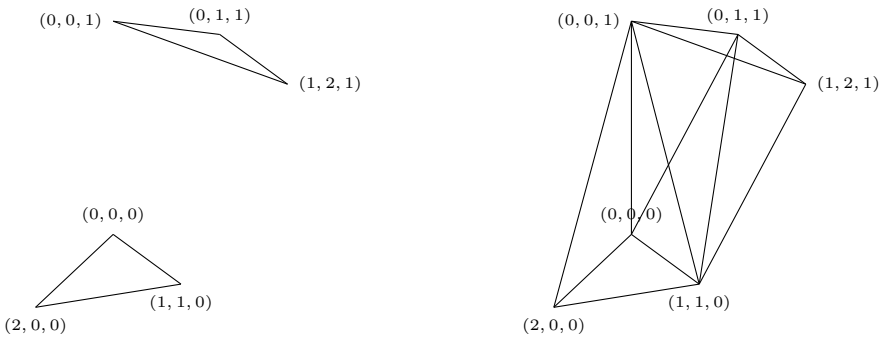


Figura 1.1: El politopo de Cayley de dos polígonos.

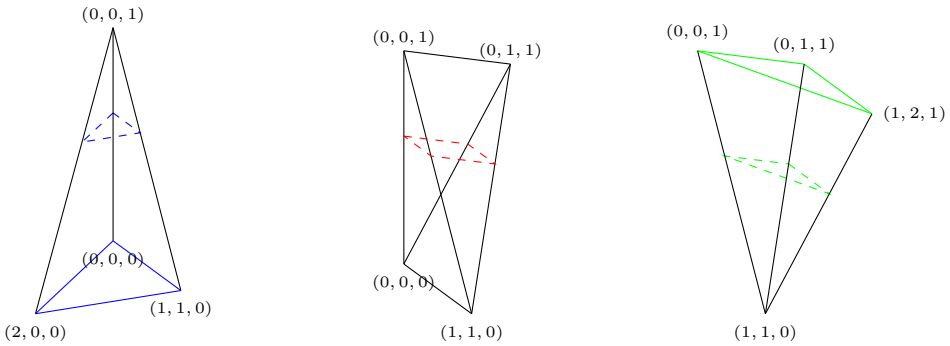


Figura 1.2: Triangulación del politopo.

Sean $\lambda_1, \lambda_2 \geq 0$ con $\lambda_1 + \lambda_2 = 1$ y consideramos la suma de Minkowski $\lambda_1 P_1 + \lambda_2 P_2$, donde P_1 es el polígono que define la base, P_2 es el polígono de la parte superior del politopo y $\lambda_1 P_1 + \lambda_2 P_2$ la obtenemos como suma convexa de todas las sumas de vértices de los polígonos. El área de los triángulos en el corte son $\lambda_1^2 \text{area}(P_1)$ y $\lambda_2^2 \text{area}(P_2)$, como cada lado del triángulo se escala por λ_1 y λ_2 respectivamente. El área de la región mixta viene escalada por $\lambda_1 \lambda_2$, ya que se genera a partir de un lado de P_1 y otro de P_2.

Vamos ahora a calcular la suma de Minkowski $\lambda_1 P_1 + \lambda_2 P_2$. En nuestro caso, el teorema de Minkowski dice

$$\text{area}(\lambda_1 P_1 + \lambda_2 P_2) = V(P_1, P_1)\lambda_1^2 + V(P_1, P_2)\lambda_1 \lambda_2 + V(P_2, P_2)\lambda_2^2,$$

donde $V(P_1, P_1) = \text{area}(P_1)$, $V(P_2, P_2) = \text{area}(P_2)$ y $V(P_1, P_2)$ es el área de la región mixta.

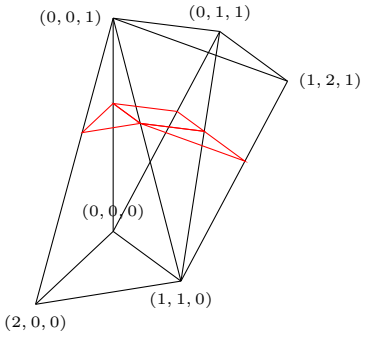


Figura 1.3: Subdivisión mixta inducida por la triangulación del politopo.

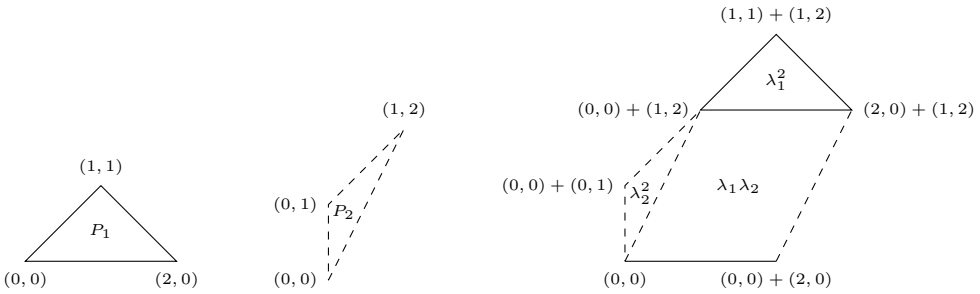


Figura 1.4: Subdivisión de la suma de los dos polígonos P_1 y P_2.

Por tanto,

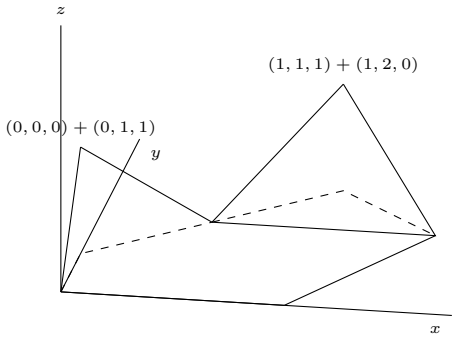
$$\text{area}(\lambda_1 P_1 + \lambda_2 P_2) = \lambda_1^2 + 4\lambda_1\lambda_2 + \frac{1}{2}\lambda_2^2.$$

Las subdivisiones que queremos en la suma de Minkowski son las que vienen inducidas por un *lifting*: se les llama subdivisiones regulares y definen homotopías poliédricas. El soporte *lifted* es $\hat{\mathcal{A}} = (\hat{A}_1, \hat{A}_2)$, con

$$\begin{cases} \hat{A}_1 = \{(2,0,0), (1,1,1), (0,0,0)\} \\ \hat{A}_2 = \{(0,1,1), (1,2,0), (0,0,0).\} \end{cases}$$

En general, para cada región mixta en la subdivisión de los politopos de Newton hay que considerar una homotopía $h(\mathbf{x}, t) = 0$, donde las potencias de t son los valores del *lifting* de los soportes que han inducido la subdivisión. En nuestro ejemplo, sólo hay que considerar una homotopía:

$$h(x, y, t) = \begin{cases} x^2 - xyt + 2 = 0 \\ yt - xy^2 - 1 = 0 \end{cases}.$$



1.4.2. Cálculo de los volúmenes mixtos y continuación poliédrica

Necesitamos conocer el número de raíces (o una cota superior, según el Teorema B de Bernstein) de un sistema polinomial para poder crear un *start system* para la homotopía. El Teorema A de Bernstein proporciona un *root count* para un sistema genérico, usano los volúmenes mixtos de sus politopos.

Teorema 1.2 (Teorema A de Bernstein)**.** *El número de raíces de un sistema polinomial genérico es igual que el volumen mixto de sus politopos de Newton.*

La manera en la que se calculan los volúmenes mixtos determina la forma con la que resolvemos el sistema genérico. Hay dos técnicas principales: la primera se basa en el Cayley trick y calcula todas las celdas en una subdivisión mixta; la segunda usa programación lineal y lleva a una enumeración eficiente de todas las celdas mixtas en una subdivisión mixta.

El Cayley trick

Con el Cayley trick se puede obtener una subdivisión mixta regular como una triangulación regular de cada politopo. Con este método se construye una triangulación gradualmente, añadiendo los puntos uno tras el otro.

La idea es descomponer cada punto con respecto a un dado simplex como combinación lineal de los vértices del simplex, imponiendo que los coeficientes de la combinación sumen 1. A esta descomposición se le llama *descomposición baricéntrica* de un punto con respecto a un simplex. Los signos negativos en la combinación van a indicar cuáles vértices del simplex se pueden cambiar por el nuevo punto para crear un nuevo simplex en la triangulación.

Ejemplo 1.6. Consideramos el simplex $[c_0, c_1, c_2]$ generado por $c_0 = (0, 0)$, $c_1 = (4, 2)$ y $c_2 = (2, 3)$. Si consideramos otro punto, hay tres posibilidades:

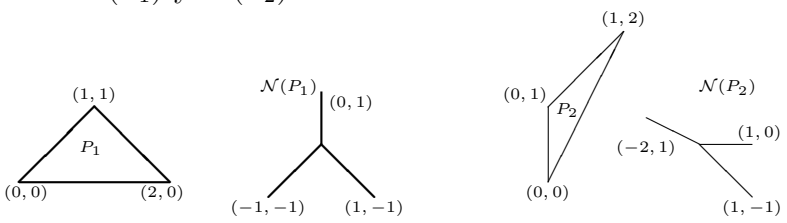
punto	decomposición baricéntrica	nuevo simplex
$x = (1, 1)$	$x = \frac{5}{8}c_0 + \frac{1}{8}c_1 + \frac{1}{4}c_2$	no
$y = (0, 2)$	$y = \frac{1}{2}c_0 - \frac{1}{2}c_1 + c_2$	$[c_0, y, c_2]$
$z = (1, 5)$	$z = -\frac{3}{8}c_0 - \frac{7}{8}c_1 + \frac{9}{4}c_2$	$[z, c_1, c_2], [c_0, z, c_2]$

El algoritmo para calcular triangulaciones regulares gradualmente lleva a un *solver* poliédrico incremental que resuelve sistemas polinomiales añadiendo un monomio tras el otro. Si la estructura del sistema es tal que la mayoría de los polinomios comparten el mismo soporte o generan el mismo politopo de Newton, ese método se revela bastante eficaz.

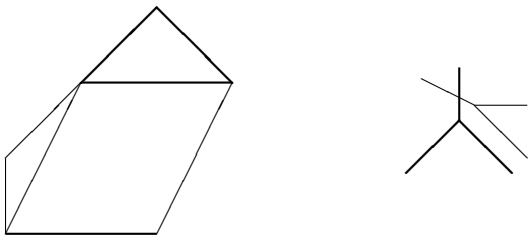
Programación lineal

Para los cambios de coordenadas en las homotopías poliédricas hay que conocer las normales interiores de las celdas mixtas. Por tanto, vamos a usar una representación dual de los politopos. El *normal fan* de un politopo es el conjunto de los conos normales a cada cara del politopo. El cono normal de una cara contiene a todas las normales interiores que definen la cara.

Ejemplo 1.7. Para la subdivisión mixta obtenida en el Ejemplo 1.5, los conos normales $\mathcal{N}(P_1)$ y $\mathcal{N}(P_2)$ son



y la representación dual de la subdivisión mixta es



Estamos interesados en las celdas mixtas de una subdivisión mixta y, en particular, en las normales interiores a las caras (o lados) de la suma de Minkowski que definen las celdas mixtas.
La búsqueda de todas las normales interiores de una celda mixta en una subdivisión mixta lleva a un sistema de igualdades y desigualdades. Para un

soporte $\mathcal{A} = (A_1, \ldots, A_n)$ de un sistema polinomial $f = (f_1, \ldots, f_n)$, consideramos un lado del k-ésimo politopo, generado por $\{a, b\} \subseteq A_k$. Entonces la normal interior v a ese lado satisface

$$\begin{cases} \langle a, v\rangle = \langle b, v\rangle \\ \langle a, v\rangle \leq \langle c, v\rangle, \quad \forall c \in A_k. \end{cases} \tag{1.10}$$

Enumerar todos los lados de un politopo es por tanto equivalente a enumerar todas las posibles soluciones del sistema (1.10). Hacer variar k desde 1 hasta n en el sistema (1.10) aplicado a los conjuntos de puntos *lifted* $\hat{A}_k$ proporciona el modelo dual de programación lineal para enumerar todas las normales interiores a las celdas mixtas en una subdivisión regular mixta.

1.4.3. El Teorema B de Bernstein

El volumen mixto es un *generically sharp root count* para el número de soluciones aisladas con todas las coordenadas distintas de cero (Teorema B de Bernstein). *Generically sharp* significa que, salvo para sistemas con coeficientes en un subconjunto específico, el *root count* es exacto. Los sistemas genéricos se suelen usar como *start system* para resolver sistemas polinomiales que tienen los mismos politopos de Newton.

Trazando caminos que divergen al infinito, nos preguntamos cuando terminar. Si un camino parece que diverge, tenemos que saber si hay verdadera divergencia o convergencia a una raíz con componentes muy grandes. El Teorema B de Bernstein proporciona un certificado para la divergencia.

Para un sistema $f(x) = 0$ con soporte $\mathcal{A} = (A_1, \ldots, A_n)$, podemos escribir sus ecuaciones $f = (f_1, \ldots, f_n)$ como

$$f_i(x) = \sum_{\mathbf{a} \in A_i} c_{i\mathbf{a}} x^{\mathbf{a}}, \quad i = 1, \ldots, n.$$

Los politopos de Newton de f los denotamos por $\mathcal{P} = (P_1, \ldots, P_n)$, donde $P_i = \text{conv}(A_i)$, para $i = 1, \ldots, n$.

Para $\omega \in \mathbb{C}^n$, $\omega \neq 0$ definimos la tupla de caras

$$\partial_\omega \mathcal{P} = (\partial_\omega P_1, \ldots, \partial_\omega P_n), \text{ con } \partial_\omega P_i := \text{conv}(\partial_\omega A_i),$$

donde

$$\partial_\omega A_i := \{\mathbf{a} \in A_i \mid \langle \mathbf{a}, \omega\rangle = \min_{\mathbf{a}' \in A_i} \langle \mathbf{a}', \omega\rangle\}.$$

El conjunto $\partial_\omega A_i$ es el soporte de la cara del i-ésimo polinomio f_i:

$$\partial_\omega f_i(x) = \sum_{\mathbf{a}\in\partial_\omega A_i} c_{i\mathbf{a}} x^{\mathbf{a}}.$$

Teorema 1.3 (Teorema B de Bernstein)**.** *Si denotamos el sistema polinomial determinado por $\omega \neq 0$ como $\partial_\omega f = (\partial_\omega f_1, \ldots, \partial_\omega f_n)$, y el volumen mixto de $\mathcal{P}$ como $V(\mathcal{P})$, tenemos:*

- *Si $\omega \neq 0$ y $\partial_\omega f(x) = 0$ no tiene soluciones en $(\mathbb{C}^*)^n$, entonces $V(\mathcal{P})$ es exacto y todas las soluciones son aisladas.*
- *Si no, para $V(\mathcal{P}) \neq 0$, se tiene $V(\mathcal{P}) > n^o$ soluciones aisladas.*

Observación 1.1. Aunque parezca que el volumen mixto lleva siempre a un *root count* exacto, hay que considerar que los vértices de los politopos no son aleatorios, sino que ocurren como exponentes de los polinomios. En el caso de politopos de Newton genéricos, puede que haya caras k-dimensionales generadas por más de $k+1$ vértices.

1.5. El software Bertini

El software Bertini se puede usar para calcular todas las raíces aisladas de un sistema de polinomios con coeficientes complejos. Además, puede encontrar *witness points* en cada componente irreducible del conjunto algebraico correspondiente al sistema polinomial. Bertini detecta la descomposición irreducible del conjunto algebraico especificando al menos un punto para cada componente. Una vez calculado un *witness set*, el usuario puede encontrar puntos en cada componente y ejecutar un test de pertenencia a una componente para determinar, dada una lista de puntos, qué puntos yacen en el conjunto algebraico correspondiente al *witness set*.

Como funciona

Bertini usa métodos de continuación homotópica para calcular las soluciones aisladas de sistemas polinomiales. La idea es escribir el sistema polinomial, del que queremos calcular las soluciones, como una familia parametrizada de sistemas, uno de los cuales, el *start system*, tiene soluciones fáciles de calcular. Tras eligir la familia, Bertini escoge un camino desde el *start system* al sistema original, construye una homotopía entre los dos y traza los caminos soluciones usando métodos *predictor-corrector* (por ejemplo, el método de Euler y el método de Newton), usando un *step size* flexible.

Si necesario, `Bertini` homogeniza el sistema; además, si el sistema es lineal y el número n de variables es mayor que el número m de ecuaciones, añade $n - m$ ecuaciones para hacerlo cuadrado. `Bertini` crea un *start system* homogéneo, lo resuelve para obtener los puntos iniciales y une los dos sistemas en una homotopía, usando el γ *trick* para asegurar que no hayan singularidades. Luego traza cada camino independientemente y devuelve los resultados al usuario.

Ejemplo 1.8. Consideramos el sistema cuadrado dado por

$$f(x, y, z) = \begin{cases} x^2 - 1 = 0 \\ y^2 - 2 = 0 \\ z^2 - 3 = 0 \end{cases} .$$

El fichero de input para `Bertini` es

```
CONFIG
TRACKTOLBEFOREEG: 1e-7;
TRACKTOLDURINGEG: 1e-7;
FINALTOL: 1e-12;
END;
INPUT
function f1,f2,f3;
variable_group x,y,z;
f1 = x^2-1;
f2 = y^2-2;
f3 = z^2-3;
END;
```

Las soluciones aproximadas son

```
1.000000000000000e+00 0.000000000000000e+00
1.414213562373095e+00 -1.110223024625157e-16
1.732050807568877e+00 0.000000000000000e+00

9.999999999999999e-01 0.000000000000000e+00
1.414213562373095e+00 -5.551115123125783e-17
-1.732050807568876e+00 3.330669073875470e-16

1.000000000000000e+00 0.000000000000000e+00
-1.414213562373095e+00 0.000000000000000e+00
```

```
1.732050807568877e+00 0.000000000000000e+00

1.000000000000000e+00 0.000000000000000e+00
-1.414213562373095e+00 -1.110223024625157e-16
-1.732050807568878e+00 -1.110223024625157e-16

-1.000000000000000e+00 5.551115123125783e-17
1.414213562373095e+00 -2.220446049250313e-16
1.732050807568877e+00 -1.110223024625157e-16

-9.999999999999998e-01 0.000000000000000e+00
1.414213562373094e+00 5.551115123125783e-17
-1.732050807568877e+00 -1.110223024625157e-16

-9.999999999999998e-01 5.551115123125783e-17
-1.414213562373095e+00 3.330669073875470e-16
1.732050807568877e+00 -4.440892098500626e-16

-1.000000000000000e+00 1.110223024625157e-16
-1.414213562373095e+00 -2.220446049250313e-16
-1.732050807568877e+00 1.110223024625157e-16
```

Vamos a ver más en detalle como funciona. Usando el grado total del sistema $f(x, y, z) = 0$, creamos un *start system* $g(x, y, z) = 0$ de la siguiente manera:

$$g(x, y, z) = \begin{cases} x^2 - 1 = 0 \\ y^2 - 1 = 0 \\ z^2 - 1 = 0 \end{cases},$$

cuyas soluciones son fáciles de calcular. Calculamos los *start points* con `Mathematica`:

```
>f = {X^2 - 1, Y^2 - 2, Z^2 - 3}; (* sistema a resolver *)
>g = {X^2 - 1, Y^2 - 1, Z^2 - 1}; (* start system *)
>h[X_, Y_, Z_, t_] := (1 + I)*(1 - t)*{X^2 - 1, Y^2 - 1, Z^2 - 1} +
  t*{X^2 - 1, Y^2 - 2, Z^2 - 3}

>Collect[h[x, y, z, t], t]
 {(1 + I) (-1 + x^2) - I t (-1 + x^2), (1 + I) (-1 + y^2) +
  t (-2 + y^2 - (1 + I) (-1 + y^2)), (1 + I) (-1 + z^2) +
  t (-3 + z^2 - (1 + I) (-1 + z^2))}
```

```
>S = Solve[h[x, y, z, 0] == 0, {x, y, z}] (* start points *)
 {{x -> -1, y -> -1, z -> -1}, {x -> 1, y -> -1, z -> -1},
 {x -> -1, y -> 1, z -> -1}, {x -> 1, y -> 1, z -> -1},
 {x -> -1, y -> -1, z -> 1}, {x -> 1, y -> -1, z -> 1},
 {x -> -1, y -> 1, z -> 1}, {x -> 1, y -> 1, z -> 1}}
```

La función `Cont` calcula las soluciones aproximadas usando la homotopía $h(x, y, z, t) = \gamma(1-t)g(x, y, z) + f(x, y, z)t = 0$, con $\gamma = 1 + i$. La función `SOL` devuelve todas las soluciones aproximadas a partir de los *start points* en S.

```
>Cont[x_, DeltaT_] :=
  Module[{X, Y, Z, k, F, sol, H, a, b, c}, k = 1; sol = x;
   While[k*DeltaT <= 1, H = h[X, Y, Z, k*DeltaT];
     F = FindRoot[
       H == 0, {{X, sol[[1]]}, {Y, sol[[2]]}, {Z, sol[[3]]}},
       Method -> "Newton"];
     sol[[1]] = X /. {F[[1]]}; sol[[2]] = Y /. {F[[2]]};
     sol[[3]] = Z /. {F[[3]]}; k += 1]
    Print[sol];]

>SOL[S_, DeltaT_] := Module[{sol, t, sp}, t = DeltaT; sol = S;
  For[i = 1, i <= Length[sol], i++, sp = {x, y, z} /. sol[[i]];
   Cont[sp, t]];
  ]
```

Ejecutando la función `SOL` obtenemos

```
>SOL[S, 0.1]
 {-1.+0. I,-1.41421+3.47099*10^-28 I,-1.73205+4.62714*10^-26 I}
 {1. +0. I,-1.41421+3.47099*10^-28 I,-1.73205+4.62714*10^-26 I}
 {-1.+0. I,1.41421 -3.47099*10^-28 I,-1.73205+4.62714*10^-26 I}
 {1. +0. I,1.41421 -3.47099*10^-28 I,-1.73205+4.62714*10^-26 I}
 {-1.+0. I,-1.41421+3.47099*10^-28 I,1.73205 -4.62714*10^-26 I}
 {1. +0. I,-1.41421+3.47099*10^-28 I,1.73205 -4.62714*10^-26 I}
 {-1.+0. I,1.41421 -3.47099*10^-28 I,1.73205 -4.62714*10^-26 I}
 {1. +0. I,1.41421 -3.47099*10^-28 I,1.73205 -4.62714*10^-26 I}
```

```
 Bertini(TM) v1.5.1
  (August 29, 2016)

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

(using GMP v4.2.3, MPFR v2.3.1)


NOTE: You have requested to use adaptive path tracking.  Please make sure that you have
setup the following tolerances appropriately:
CoeffBound: 6.618223000000e+00, DegreeBound: 2.000000000000e+00
AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 1024

Tracking path 0 of 8

Finite Solution Summary

NOTE: nonsingular vs singular is based on condition number and identical endpoints

                | Number of real solns  |  Number of non-real solns     |  Total
-------------------------------------------------------------------------------------------
Non-singular    |       8               |               0               |   8
Singular        |       0               |               0               |   0
-------------------------------------------------------------------------------------------
Total           |       8               |               0               |   8


Finite Multiplicity Summary

  Multiplicity  |  Number of real solns |  Number of non-real solns
-------------------------------------------------------------------------------------------
        1       |       8               |       0


-------------------------------------------------------------------------------------------
The following files may be of interest to you:

main_data:             A human-readable version of the solutions - main output file.
raw_solutions:         A list of the solutions with the corresponding path numbers.
raw_data:              Similar to the previous, but with the points in Bertini's homogeneous
                         coordinates along with more information about the solutions.
real_finite_solutions: A list of all real finite solutions.
finite_solutions:      A list of all finite solutions.
nonsingular_solutions: A list of all nonsingular solutions.
singular_solutions:    A list of all singular solutions.
-------------------------------------------------------------------------------------------

Paths Tracked: 8
```

Figura 1.5: Ejercicio 1.8 con `Bertini`

CAPÍTULO 2

Certificación de las soluciones de un sistema polinomial cuadrado: `alphaCertified`

`alphaCertified` es un programa, creado por F. Sottile y J. Hauenstein, que implementa elementos de α-theory para certificar las soluciones numéricas de sistemas de ecuaciones polinomiales, usando sea aritmética exacta sea aritmética *floating point* de precisión arbitraria.

La ventaja de este programa es que basta sólo con certificar el resultado de un cálculo numérico, evitando la complejidad de certificar los caminos usados para llegar a dicha solución (Beltrán y Leykin).

En `alphaCertified` vienen implementados algoritmos para sistemas polinomiales cuadrados y sobredeterminados, es decir, sistemas en los que el número de polinomios es mayor que el número de variables. En este capítulo nos quedaremos sólo con el caso de sistemas polinomiales cuadrados, ya que vamos a tratar la certificación de soluciones de sistemas sobredeterminados con otro método en el siguiente capítulo.

La idea es que, dados un sistema polinomial cuadrado $f : \mathbb{C}^n \to \mathbb{C}^n$ y un conjunto de puntos $X \subset \mathbb{C}^n$, `alphaCertified` usa α-theory para ver:

- a partir de qué puntos de X el método de Newton converge cuadráticamente a una solución de f;

- a partir de qué puntos de X el método de Newton converge cuadráticamente a soluciones distintas de f;
- si f es un sistema polinomial real, desde qué puntos de X el método de Newton converge cuadráticamente a soluciones reales de f.

2.1. α-theory de Smale

Sea $f : \mathbb{C}^n \to \mathbb{C}^n$ un sistema de n ecuaciones polinomiales en n variables con conjunto de ceros $V(f) = \{\xi \in \mathbb{C}^n \mid f(\xi) = 0\}$. Sea $Jf(\mathbf{x})$ la matriz jacobiana del sistema en el punto $\mathbf{x}$.
Consideramos la aplicación $N_f : \mathbb{C}^n \to \mathbb{C}^n$ definida por

$$N_f(\mathbf{x}) := \begin{cases} \mathbf{x} - Jf(\mathbf{x})^{-1} f(\mathbf{x}) & \text{si } J_f(\mathbf{x}) \text{ es invertible} \\ \mathbf{x} & \text{si } J_f(\mathbf{x}) \text{ no es invertible.} \end{cases}$$

Llamamos a $N_f(\mathbf{x})$ la *iteración de Newton de f que empieza en* $\mathbf{x}$. Para $k \in \mathbb{N}$, definimos la *k-ésima iteración de Newton de f que empieza en* $\mathbf{x}$ como

$$N_f^{(k)}(\mathbf{x}) := \underbrace{N_f \circ \cdots \circ N_f(\mathbf{x})}_{k \text{ veces}}.$$

Definición 2.1. Sea $f : \mathbb{C}^n \to \mathbb{C}^n$ un sistema polinomial. Un punto $\mathbf{x} \in \mathbb{C}^n$ es una *solución aproximada* de f con solución asociada $\xi \in V(f)$ si, para cada $k \in \mathbb{N}$,

$$\|N_f^{(k)}(\mathbf{x}) - \xi\| \leq \left(\frac{1}{2}\right)^{2^k - 1} \|\mathbf{x} - \xi\|, \tag{2.1}$$

donde $\|\cdot\|$ es la norma hermitiana usual en $\mathbb{C}^n$.

Con ello, la sucesión $\{N_f^{(k)} \mid k \in \mathbb{N}\}$ converge cuadráticamente a ξ.

α-theory de Smale describe condiciones, basadas en las constantes $\alpha(f, \mathbf{x})$, $\beta(f, \mathbf{x})$ y $\gamma(f, \mathbf{x})$, que implican si un dado punto $\mathbf{x}$ es una solución aproximada de f o no.
Si $Jf(\mathbf{x})$ es invertible, definimos

$$\begin{aligned} \alpha(f, \mathbf{x}) &:= \beta(f, \mathbf{x})\gamma(f, \mathbf{x}), \\ \beta(f, \mathbf{x}) &:= \|\mathbf{x} - N_f(\mathbf{x})\| = \|Jf(\mathbf{x})^{-1} f(\mathbf{x})\|, \\ \gamma(f, \mathbf{x}) &:= \sup_{k \geq 2} \left\| \frac{Jf(\mathbf{x})^{-1} J^k f(\mathbf{x})}{k!} \right\|^{\frac{1}{k-1}}, \end{aligned}$$

donde $J^k f(\mathbf{x})$ es la k-ésima derivada de f en $\mathbf{x}$ como aplicación k-lineal. La norma en la definición de γ es la norma del operador $Jf(\mathbf{x})^{-1}J^k f(\mathbf{x}) : S^k\mathbb{C}^n \to \mathbb{C}^n$, definida con respecto a la norma en $S^k\mathbb{C}^n$ (el cuerpo de las k-ésimas potencias simétricas de $\mathbb{C}^n$)[1] que es dual a la norma estándar invariante bajo grupos unitarios de los polinomios homogéneos ([5]):

$$\Big\|\sum_{|\nu|=d} a_\nu \mathbf{x}^\nu\Big\|^2 := \sum_{|\nu|=d} |a_\nu|^2 / \binom{d}{\nu},$$

donde $\nu = (\nu_1, \ldots, \nu_n)$ es un vector exponente de enteros no negativos con $\mathbf{x}^\nu = x_1^{\nu_1} \cdots x_n^{\nu_n}$, $|\nu| = \nu_1 + \cdots + \nu_n$ y $\binom{d}{\nu} = \frac{d!}{\nu_1! \cdots \nu_n!}$ es el coeficiente multinomial.

Si $\mathbf{x} \in V(f)$ y $Jf(\mathbf{x})$ no es invertible, definimos

$$\alpha(f, \mathbf{x}) = \beta(f, \mathbf{x}) := 0,$$
$$\gamma(f, \mathbf{x}) := \infty.$$

Si $\mathbf{x} \notin V(f)$ y $Jf(\mathbf{x})$ no es invertible, definimos

$$\alpha(f, \mathbf{x}) = \beta(f, \mathbf{x}) = \gamma(f, \mathbf{x}) := \infty.$$

El siguiente teorema nos da un certificado de que un punto $\mathbf{x}$ es una solución aproximada de f.

Teorema 2.1. *Si $f : \mathbb{C}^n \to \mathbb{C}^n$ es un sistema polinomial y $\mathbf{x} \in \mathbb{C}^n$ verificando*

$$\alpha(f, \mathbf{x}) < \frac{13 - 3\sqrt{7}}{4} \approx 0.157671\,, \tag{2.2}$$

entonces $\mathbf{x}$ es una solución aproximada de f. Además, $\|\mathbf{x} - \xi\| \leq 2\beta(f, \mathbf{x})$, donde $\xi \in V(f)$ es la solución asociada a $\mathbf{x}$.

Si se verifica (2.2), se dice que $\mathbf{x}$ es una *solución aproximada certificada* de f.

Para certificar que dos soluciones aproximadas tienen la misma solución asociada, vamos a usar el siguiente teorema.

[1] Sea V un espacio vectorial sobre K con $\dim_K(V) = m < \infty$. Denotamos

$$V^{\otimes k} := \begin{cases} V \otimes_K \cdots \otimes_K V & \text{si } k > 0 \\ K & \text{si } k = 0 \\ 0 & \text{si } k < 0 \end{cases}$$

Definimos $S^k(V)$ como el espacio cociente de $V^{\otimes k}$ por el subespacio generado por $v_1 \otimes \cdots \otimes v_k - v_{\sigma(1)} \otimes \cdots \otimes v_{\sigma(k)}$, para cada $v_i \in V$ y cada permutación $\sigma : \{1, \ldots, m\} \to \{1, \ldots, m\}$.

Teorema 2.2. *Sean* $f:\ \mathbb{C}^n \to \mathbb{C}^n$ *un sistema polinomial,* $\mathbf{x} \in \mathbb{C}^n$ *con* $\alpha(f,\mathbf{x}) < 0.03$ *y* $\xi \in V(f)$ *la solución asociada a* $\mathbf{x}$*. Si* $\mathbf{y} \in \mathbb{C}^n$ *verifica*

$$\|\mathbf{x}-\mathbf{y}\| < \frac{1}{20\gamma(f,\mathbf{x})},$$

entonces $\mathbf{y}$ *es una solución aproximada de* f *con solución asociada* ξ*.*

En general, la constante $\gamma(f,\mathbf{x})$ es difícil de calcular, debido al comportamiento de las derivadas de orden superior de f en el punto $\mathbf{x}$, pero se puede dar una cota superior.
Para un polinomio $g:\ \mathbb{C}^n \to \mathbb{C}$ de grado d, $g = \sum_{|\nu|\leq d} a_\nu \mathbf{x}^\nu$, definimos

$$\|g\|^2 := \sum_{|\nu|\leq d} |a_\nu|^2 \, \frac{\nu!(d-|\nu|)!}{d!},$$

luego $\|\cdot\|$ es la norma estándar invariante bajo grupos unitarios en la homogeneización de g.
Para un sistema polinomial $f:\ \mathbb{C}^n \to \mathbb{C}^n$, $f = (f_1,\ldots,f_n)$, definimos

$$\|f\|^2 := \sum_{i=1}^{n} \|f_i\|^2,$$

y para un punto $\mathbf{x} \in \mathbb{C}^n$, vamos a definir

$$\|\mathbf{x}\|_1^2 := 1 + \|\mathbf{x}\|^2 = 1 + \sum_{i=1}^{n} |x_i|^2.$$

Sea $\Delta_{(d)}(\mathbf{x})$ la matriz diagonal $n \times n$ con

$$\Delta_{(d)}(\mathbf{x})_{i,i} := d_i^{\frac{1}{2}} \|\mathbf{x}\|_1^{d_i-1},$$

donde $d_i = \deg(f_i)$.
Si $Jf(\mathbf{x})$ es invertible, definimos

$$\mu(f,\mathbf{x}) := \max\{1, \|f\| \cdot \|Jf(\mathbf{x})^{-1}\Delta_{(d)}(\mathbf{x})\|\}.$$

Proposición 2.3. *Sea* $f:\ \mathbb{C}^n \to \mathbb{C}^n$ *un sistema polinomial con* $d_i = \deg(f_i)$ *y* $D = \max d_i$*. Si* $\mathbf{x} \in \mathbb{C}^n$ *es tal que* $Jf(\mathbf{x})$ *es invertible, entonces*

$$\gamma(f,\mathbf{x}) \leq \frac{\mu(f,\mathbf{x}) D^{\frac{3}{2}}}{2\|\mathbf{x}\|_1}. \tag{2.3}$$

2.2. Algoritmos para sistemas polinomiales cuadrados

Sea $f: \mathbb{C}^n \to \mathbb{C}^n$ un sistema polinomial cuadrado y sea $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\} \subset \mathbb{C}^n$ un conjunto de puntos. Los algoritmos están pensados para sistemas polinomiales complejos pero vienen implementados para sistemas polinomiales con coeficientes en $\mathbb{Q}(i)$.

Para cada $i = 1, \ldots, k$, `alphaCertified` primero verifica si $f(\mathbf{x}_i) = 0$. Si se tiene que $f(\mathbf{x}_i) \neq 0$, entonces va a averiguar si la matriz jacobiana en $\mathbf{x}_i$, $Jf(\mathbf{x}_i)$, es invertible. Si lo es, calcula $\beta(f, \mathbf{x}_i)$ y cotas superiores para $\alpha(f, \mathbf{x}_i)$ y $\gamma(f, \mathbf{x}_i)$.

Algoritmo 1: `ComputeConstants`$(f, \mathbf{x})$

Data: $f: \mathbb{C}^n \to \mathbb{C}^n$, $\mathbf{x} \in \mathbb{C}^n$ tal que $Jf(\mathbf{x})$ es invertible.
Result: (α, β, γ).
$\beta = \|Jf(\mathbf{x})^{-1} f(\mathbf{x})\|$;
$\gamma = \dfrac{\mu(f, \mathbf{x}) D^{\frac{3}{2}}}{2\|\mathbf{x}\|_1}$;
$\alpha = \beta \cdot \gamma$;

El siguiente algoritmo calcula, usando el Teorema 2.1, un subconjunto $Y \subset X$ que contiene a los puntos que son soluciones aproximadas certificadas de f.

Algoritmo 2: `CertifySolns`(f, X)

Data: $f: \mathbb{C}^n \to \mathbb{C}^n$, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{C}^n$.
Result: Y.
begin
  $Y = \emptyset$;
  for $i = 1$ **to** k **do**
    if $f(\mathbf{x}_j) = 0$ **then**
      $Y = Y \cup \{\mathbf{x}_j\}$;
    else
      if $Jf(\mathbf{x}_j)$ *es invertible* **then**
        (α, β, γ)=`ComputeConstants` $(f, \mathbf{x}_j)$;
        if $\alpha < \frac{13 - 3\sqrt{17}}{4}$ **then**
          $Y = Y \cup \{\mathbf{x}_j\}$;
return Y

Sea ahora $\mathbf{x}$ una solución aproximada de f con solución asociada ξ tal que $Jf(\mathbf{x})$ es invertible. Como $\mathbf{x}$ es una solución aproximada, eso implica que $\beta(f, N_f^{(k)}(\mathbf{x}))$ converge a cero. Ya que $\gamma(f, \mathbf{x})$ es el supremo de un número finito de funciones continuas, se tiene que $\gamma(f, N_f^{(k)}(\mathbf{x}))$ está acotado. Con ello, por definición, $\alpha(f, N_f^{(k)}(\mathbf{x}))$ converge a cero también.

Dadas dos soluciones aproximadas $\mathbf{x}_1$ y $\mathbf{x}_2$ de f con soluciones asociadas, respectivamente, ξ_1 y ξ_2, vamos a usar los resultados de la sección anterior para determinar si ξ_1 y ξ_2 son iguales:

- si $\|\mathbf{x}_1 - \mathbf{x}_2\| > 2\,(\beta(f, \mathbf{x}_1) + \beta(f, \mathbf{x}_2))$, entonces $\xi_1 \neq \xi_2$;
- si $\alpha(f, \mathbf{x}_i) < 0.03$ y $\|\mathbf{x}_1 - \mathbf{x}_2\| < \frac{1}{20\gamma(f, \mathbf{x}_i)}$, para $i = 1, 2$, entonces $\xi_1 = \xi_2$.

El siguiente algoritmo devuelve, usando los criterios anteriores, un valor booleano que describe si $\xi_1 \neq \xi_2$.

Algoritmo 3: `CertifyDistinctSolns`(f, x_1, x_2)

Data: $f : \mathbb{C}^n \to \mathbb{C}^n$, $\mathbf{x}_1, \mathbf{x}_2$ soluciones aproximadas con soluciones asociadas ξ_1, ξ_2.
Result: **True** si $\xi_1 \neq \xi_2$, **False** si $\xi_1 = \xi_2$.

```
for i = 1 to 2 do
    (α_i, β_i, γ_i)=ComputeConstants (f, x_i);
if ‖x_1 − x_2‖ > 2(β_1 + β_2) then
    return True;
if (α_1 < 0.03 and ‖x_1 − x_2‖ < 1/(20γ_1)) or (α_2 < 0.03 and
  ‖x_1 − x_2‖ < 1/(20γ_2)) then
    return False;
for i = 1 to 2 do
    x_i = N_f(x_i);
return to 1;
```

Observamos que el algoritmo va a terminar, ya que $\beta(f, N_f^{(k)}(\mathbf{x}_i))$ converge cuadráticamente y $\gamma(f, N_f^{(k)}(\mathbf{x}_i))$ está acotado.

Vamos ahora a considerar el problema de certificar soluciones reales de un sistema polinomial real. Recordamos que un sistema polinomial $f : \mathbb{C}^n \to \mathbb{C}^n$ es real si $\{\overline{f_1}, \ldots, \overline{f_n}\} = \{f_1, \ldots, f_n\}$ y en tal caso se tiene que las soluciones de $f(\mathbf{x}) = 0$ son reales o complejas conjugadas. También se tiene que

$N_f(\overline{\mathbf{x}}) = \overline{N_f(\mathbf{x})}$, así que $N_f : \mathbb{R}^n \to \mathbb{R}^n$ es una aplicación real.

Sea $\mathbf{x}$ una solución aproximada de f con solución asociada ξ. No vamos a suponer que $\mathbf{x}$ es real, ya que los métodos numéricos suelen devolver soluciones aproximadas complejas. Por suposición, $\overline{\mathbf{x}}$ es también una solución aproximada de f con solución asociada $\overline{\xi}$.

Consideramos la proyección natural $\pi_{\mathbb{R}} : \mathbb{C}^n \to \mathbb{R}^n$ definida por

$$\pi_{\mathbb{R}}(\mathbf{x}) = \frac{\mathbf{x} + \overline{\mathbf{x}}}{2}.$$

Ya que $\|\mathbf{x} - \overline{\mathbf{x}}\| = 2\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\|$, se tiene que si $\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\| > 2\beta(f, \mathbf{x})$, entonces ξ no es real.
Para demostrar que ξ es real, hay dos enfoques: uno local y uno global.

- Criterio local:
  - si $\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\| > 2\beta(f, \mathbf{x})$, entonces ξ no es real;
  - si $\alpha(f, \mathbf{x}) < 0.03$ y $\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\| < \frac{1}{20\gamma(f,\mathbf{x})}$, entonces $\pi_{\mathbb{R}}(\mathbf{x})$ es también una solución aproximada de f con solución asociada ξ. Ya que N_f es una aplicación real y $\pi_{\mathbb{R}}(\mathbf{x}) \in \mathbb{R}^n$, eso implica que $\xi \in \mathbb{R}^n$.
- Criterio global:
  supongamos que conocemos a priori que f tiene exactamente k soluciones $\mathbf{x}_1, \ldots, \mathbf{x}_n$ con soluciones asociadas distintas. Si, para cada $j \neq i$, $\overline{\mathbf{x}_i}$ y $\mathbf{x}_j$ también corresponden a distintas soluciones, entonces $\mathbf{x}_i$ y $\overline{\mathbf{x}_i}$ tienen que corresponder a la misma solución, que es por tanto real.

El siguiente algoritmo usa el enfoque local para determinar si una solución aproximada corresponde a una solución asociada real o no.

Algoritmo 4: `CertifyRealSolns`(f, x)

Data: $f : \mathbb{C}^n \to \mathbb{C}^n$, $\mathbf{x} \in \mathbb{C}^n$ con solución asociada ξ tal que $Jf(\mathbf{x})$ es invertible.
Result: **True** si $\xi \in \mathbb{R}^n$, **False** si $\xi \notin \mathbb{R}^n$.

1 (α, β, γ)=`ComputeConstants` $(f, \mathbf{x})$;
2 **if** $\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\| > 2\beta$ **then**
3 **return False**;
4 **if** $\alpha < 0.03$ **and** $\|\mathbf{x} - \pi_{\mathbb{R}}(\mathbf{x})\| < \frac{1}{20\gamma}$ **then**
5 **return True**;
6 $\mathbf{x} = N_f(\mathbf{x})$;
7 return to 1;

2.3. Implementación de `alphaCertified`

El programa `alphaCertified` está escrito en C y depende de las librerias GMP y MPFR para las computaciones con aritmética racional exacta y aritmética *floating point* de precisión arbitraria. Por defecto, usa aritmética racional y todos los cálculos se pueden certificar. `alphaCertified` permite también al usuario de eligir la precisión y usar aritmética *floating point*, pero en tal caso lo que devuelve es un *soft certificate*.

Las operaciones lineales se hacen usando una decomposición LU[2] y la norma matricial espectral $\|A\|_2 = \sqrt{\lambda_{max}(A^*A)}$ (donde λ_{max} es el máximo autovalor de la matrix A^*A, con A^* la matrix transpuesta conjugada de A) está acotada superiormente usando la norma de Frobenius dada por $\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\text{traza}(A^*A)}$.
Eso hace peor aún la aproximación de γ y tiene dos consecuencias directas. Primero, implica que el valor de β tiene que ser más pequeño para poder certificar las soluciones aproximadas en `CertifySolns`. Además, los algoritmos `CertifyDistinctSolns` y `CertifyRealSolns` pueden necesitar más iteraciones de Newton. A parte del coste computacional añadido, el uso de GMP y MPFR permite ejecutar estos cálculos aunque se están usando aproximaciones de γ. En el caso de aritmética *floating point*, la precisión inicial aumenta automáticamente en cada iteración.

Para determinar si un sistema polinomial cuadrado f en n variables es real, `alphaCertified` usa dos tests:

1. Test 1: determina si los coeficientes de f son reales.

2. Test 2: elige un punto pseudo-aleatorio $\mathbf{y} \in \mathbb{Q}^n$ y determina si $\{f_1(\mathbf{y}), \ldots, f_n(\mathbf{y})\} = \{\overline{f_1(\mathbf{y})}, \ldots, \overline{f_n(\mathbf{y})}\}$.

Si fallan todos los tests, luego `alphaCertified` no realiza la certificación de soluciones reales. Si no, para cada solución aproximada $\mathbf{x}$ con solución asociada ξ, `alphaCertified` determina si existe una solución real aproximada que también le corresponde a ξ.

[2]La decomposición LU factoriza una matriz en el producto de una matriz triangular inferior y una matriz triangular superior. A veces, el producto incluye también una matriz de permutación.

2.4. Ejemplos

Ejemplo 2.1. Consideramos el sistema del Ejemplo 1.8

$$f(x,y,z) = \begin{cases} x^2 - 1 = 0 \\ y^2 - 2 = 0 \\ z^2 - 3 = 0 \end{cases} .$$

y vamos a certificar las soluciones aproximadas obtenidas con `Bertini`. Imponemos que use aritmética *floating point*, ya que las raíces aproximadas no vienen con componentes racionales:

```
ALGORITHM: 2;
ARITHMETICTYPE: 1;
```

Al ejecutar `alphaCertified`, resulta que todas las raíces vienen certificadas y son distintas y reales.

```
Floating point (96 bits) soft certification results:

Number of points tested:                8
Certified approximate solutions:        8
Certified distinct solutions:           8
Certified real distinct solutions:      8
```

Figura 2.1: Ejercicio 2.1 con `alphaCertified`

Los valores de las constantes α, β y γ para la primera raíz aproximada del Ejemplo 1.8 vienen dados por

```
3.967828787628881e-15
7.107350707467626e-16
5.582711408148075
```

Ejemplo 2.2. Consideramos la ecuación

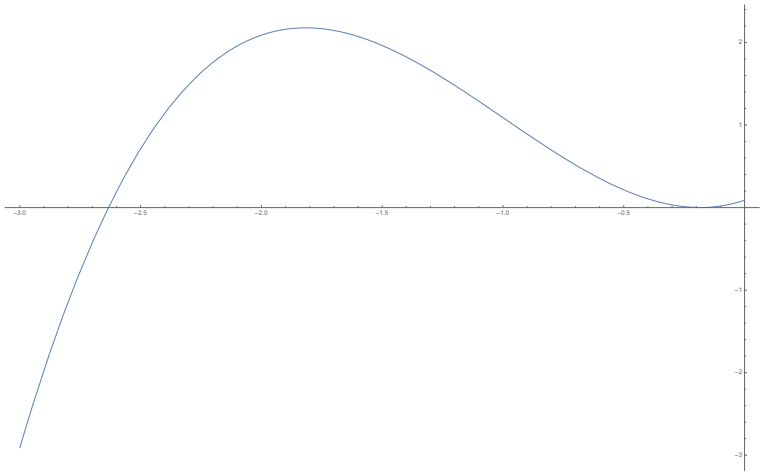
$$f(x) = x^3 + 3x^2 + x + 0.09 = 0.$$

Supongamos, erróneamente, que hemos encontrado tres raíces reales x_1, x_2, x_3.

$$x_1 = -2.63322$$
$$x_2 = x_3 = -0.183392$$

Ejecutamos ahora `alphaCertified` con estas raíces y resulta que sólo se certifica la primera, x_1.

Los valores de las constantes α, β y γ para $x_2 = x_3$ son



```
Floating point (96 bits) soft certification results:

Number of points tested:                3
Certified approximate solutions:        1
Certified distinct solutions:           1
Certified real distinct solutions:      1
```

```
4.276131367622519e4
2.449824154929502
1.745485021452722e4
```

y no cumplen las hipótesis del Teorema 2.1, por tanto no se pueden certificar como raíces aproximadas de $f(x) = 0$.

CAPÍTULO 3

Certificación de las soluciones de un sistema polinomial sobredeterminado sobre $\mathbb{Q}$

En este capítulo vamos a ver una forma efectiva de realizar los cálculos antes estudiados. Para ello restringimos a un cuerpo compatible como $\mathbb{Q}$.

La certificación propuesta se basa en métodos simbólico-numéricos, usando bien las raíces aproximadas del sistema o bien la división con resto de polinomios en una variable sobre $\mathbb{Q}$, para calcular la *rational univariate representation* (RUR) de una componente del sistema. La idea es, para sistemas polinomiales con raíces simples, calcular una RUR inicial a partir de las raíces aproximadas y luego aumentar la precisión a través de iteraciones de Newton hasta encontrar la RUR exacta. Una vez encontrada, ya que es un sistema cuadrado, se le puede aplicar α-theory para certificar que los ceros aproximados dados son ceros para la RUR exacta y por tanto para el sistema original.

3.1. Preliminares

Vamos a recordar primero la definición de RUR. Sea $f = (f_1, \ldots, f_m) \in \mathbb{Q}[x_1, \ldots, x_n]$ para $m \geq n$ y supongamos que el ideal $\mathcal{I} := \langle f_1, \ldots, f_m \rangle$ es radical y cero-dimensional. El anillo cociente $\mathbb{Q}[x_1, \ldots, x_n]/\mathcal{I}$ es un espacio vectorial de dimensión finita sobre $\mathbb{Q}$ y denotamos por $\delta := \dim_{\mathbb{Q}} \mathbb{Q}[x_1, \ldots, x_n]/\mathcal{I}$.

Además, casi para cada $(\lambda_1, \ldots, \lambda_n) \in \mathbb{Q}^n$, la combinación lineal dada por $u(x_1, \ldots, x_n) := \lambda_1 x_1 + \cdots + \lambda_n x_n$ es un elemento primitivo de $\mathcal{I}$.
Sea $q(T) \in \mathbb{Q}[T]$ el polinomio mínimo de u en $\mathbb{Q}[x_1, \ldots, x_n]/\mathcal{I}$ y sean $x_i = v_i(u)$ los polinomios que expresan las funciones coordenadas como combinación lineal de las potencias de u en $\mathbb{Q}[x_1, \ldots, x_n]/\mathcal{I}$. Se tiene por tanto

$$u = \lambda_1 v_1(u) + \cdots + \lambda_n v_n(u) \quad \text{y}$$

$$\langle q(T), x_1 - v_1(T), \ldots, x_n - v_n(T)\rangle = \langle \mathcal{I}, T - (\lambda_1 x_1 + \cdots + \lambda_n x_n)\rangle.$$

Definición 3.1. Sea $\mathcal{I} = \langle f_1, \ldots, f_m\rangle \subset \mathbb{Q}[x_1, \ldots, x_n]$ un ideal radical y cero-dimensional. La *Rational Univariate Representation* (RUR) de $\mathcal{I}$ viene dada por

- un elemento primitivo $u = \lambda_1 x_1 + \cdots + \lambda_n x_n$ de $\mathcal{I}$ para $\lambda_1, \ldots, \lambda_n \in \mathbb{Q}$;
- el polinomio mínimo $q(T) \in \mathbb{Q}[T]$ de u en $\mathbb{Q}[x_1, \ldots, x_n]/\mathcal{I}$, polinomio mónico libre de cuadrados de grado δ;
- la parametrización de las coordenadas de los ceros de $\mathcal{I}$ por los ceros de q, dada por
$$v_1(T), \ldots, v_n(T) \in \mathbb{Q}[T],$$
con $\deg(v_i) \leq \delta - 1$, $\forall i = 1, \ldots, n$, y cumpliendo
$$\lambda_1 v_1(T) + \ldots + \lambda_n v_n(T) \equiv T \text{ mód } q(T).$$

También se puede definir para un subconjunto $V \subset V(\mathcal{I})$ la RUR de la correspondiente componente de $\mathcal{I}$.

Definición 3.2. Sea $\mathcal{I} = \langle f_1, \ldots, f_m\rangle \subset \mathbb{Q}[x_1, \ldots, x_n]$ un ideal radical y cero-dimensional y sea $\lambda_1 x_1 + \cdots + \lambda_n x_n \in \mathbb{Q}[x_1, \ldots, x_n]$ un elemento primitivo de $\mathcal{I}$. Los polinomios

$$T - (\lambda_1 x_1 + \cdots + \lambda_n x_n),\ q(T),\ v_1(T), \ldots, v_n(T) \tag{3.1}$$

forman una *RUR de una componente de* $\mathcal{I}$ si se cumplen las siguientes propiedades:

- $q(T) \in \mathbb{C}[T]$ es un polinomio mónico de grado $d \leq \delta$ libre de cuadrados;
- $v_1(T), \ldots, v_n(T) \in \mathbb{C}[T]$ son todos de grado como máximo $d - 1$ y satisfacen
$$\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) \equiv T \text{ mód } q(T);$$

- para cada $i = 1, \ldots, m$ se tiene

$$f_i(v_1(T), \ldots, v_n(T)) \equiv 0 \text{ mód } q(T).$$

Además, si $q(T), v_1(T), \ldots, v_n(T) \in \mathbb{Q}[T]$, se le llama a (3.1) *RUR de una componente racional de* $\mathcal{I}$.

Observación 3.1. Observamos que el conjunto

$$\{q(T), x_1 - v_1(T), \ldots, x_n - v_n(T)\} \tag{3.2}$$

forma una base de Gröbner para el ideal que genera con respecto del orden lexicográfico con $T < x_1 < \cdots < x_n$. Si tenemos una RUR de $\mathcal{I}$, (3.2) es una base de Gröbner para

$$\langle \mathcal{I}, T - (\lambda_1 x_1 + \cdots + \lambda_n x_n) \rangle.$$

Vamos ahora a recordar la relación entre la RUR de una componente de $\mathcal{I}$ y las correspondientes raíces.
Sea $V := \{\xi_1, \ldots, \xi_d\} \subseteq V(\mathcal{I}) \subset \mathbb{C}^n$ el conjunto de las raíces exactas de una componente de $\mathcal{I}$ y denotamos $\xi_i = (\xi_{i,1}, \ldots, \xi_{i,n})$ para $i = 1, \ldots, d$. Para cada $(\lambda_1, \ldots, \lambda_n) \in \mathbb{Q}^n$ tal que

$$\lambda_1 \xi_{i,1} + \cdots + \lambda_n \xi_{i,n} \neq \lambda_1 \xi_{j,1} + \cdots + \lambda_n \xi_{j,n} \text{ si } i \neq j,$$

podemos definir un elemento primitivo $u = \lambda_1 x_1 + \cdots + \lambda_n x_n$ para V. Como las raíces son todas distintas, este elemento primitivo existe. Fijado $(\lambda_1, \ldots, \lambda_n) \in \mathbb{Q}^n$, definimos

$$\mu_i := \lambda_1 \xi_{i,1} + \cdots + \lambda_n \xi_{i,n}, \quad i = 1, \ldots, d. \tag{3.3}$$

Luego,

$$q(T) := \prod_{i=1}^{d} (T - \mu_i) \tag{3.4}$$

es el único polinomio mónico de grado d que se anula en los valores de u correspondientes a cada punto de V. Por último, $v_j(T)$ para cada j es el único interpolador de Lagrange cumpliendo

$$v_j(\mu_i) = \xi_{i,j} \text{ para } i = 1, \ldots, n. \tag{3.5}$$

Los coeficientes de la RUR de $\mathcal{I}$ son racionales, luego podemos calcularlos exactamente. Eso no siempre se cumple en el caso de la RUR de una componente de $\mathcal{I}$, sólo cuando se trata de una componente racional.

3.2. La RUR inicial

Para calcular una RUR inicial de una componente racional de $\mathcal{I}$:

1. **Método homotópico.** Sea $f = (f_1, \ldots, f_m) \in \mathbb{Q}[x_1, \ldots, x_n]$ para $m > n$ y supongamos que el ideal $\mathcal{I} := \langle f_1, \ldots, f_m \rangle$ sea radical y cero-dimensional. Siguiendo [4], para cada aplicación lineal $R : \mathbb{Q}^m \to \mathbb{Q}^n$, que podemos también considerar como una matriz $R \in \mathbb{Q}^{n \times m}$, definimos el sistema cuadrado $R(f) := R \cdot f$. Para casi cada elección de R, el ideal generado por $R(f)$ es radical y cero-dimensional. Fijada $R \in \mathbb{Q}^{n \times m}$, escribiremos

$$F = (F_1, \ldots, F_n) := R(f). \tag{3.6}$$

   Calculamos raíces aproximadas de todas las raíces en $V(F)$ usando métodos de continuación homotópica. Aplicando α-theory a F, se puede dar una cota superior ε (se vea 2.2) de la distancia de cada raíz aproximada de la exacta.

2. Para encontrar el subconjunto de las raíces aproximadas que corresponden a las raíces exactas de una componente de $V(\mathcal{I})$, hay diferentes métodos:

   - eligir las raíces que tienen los residuos de cada f_i menores que una tolerancia t, que podemos calcular en función de ε, de la altura y del grado de cada polinomio.
   - excluir los ceros que no son aproximaciones de $V(\mathcal{I})$, comparando las raíces aproximadas de $R(f)$ con las de otro sistema cuadrado aleatorio $R'(f)$, para $R' \in \mathbb{Q}^{n \times m}$.
   - si sabemos que $\mathcal{I}$ o una componente racional de $\mathcal{I}$ tiene un número muy pequeño de ceros, se pueden verificar todos los subconjuntos que tienen esa cardinalidad.

   Escribiremos d como el número de raíces escogidas.

3. **RUR inicial.** A partir de las d raíces encontradas en el paso anterior, se construye una RUR inicial con interpolación de Lagrange y reconstrucción de números racionales, usando (3.3), (3.4) y (3.5).

Observación 3.2. Observamos que el resultado depende de la elección de las raíces aproximadas en el paso 2. Para malas elecciones, la RUR iterada no converge a una RUR exacta de una componente racional de $\mathcal{I}$.

Una manera de evitar la elección de las raíces en el paso 2 es calcular una RUR exacta para todas las raíces calculadas en el paso 1. Este método, aunque menos eficiente, converge siempre a una RUR exacta, siempre que los ceros calculados en el paso 1 sean ceros certificados y que el error de redondear sea insignificante.

3.3. Incrementar la precisión de la RUR

3.3.1. Método de Newton local

La idea es repetir el cálculo de (3.3), (3.4) y (3.5) con raíces cada vez más precisas, calculadas aplicando el método de Newton a cada una de las d raíces.

Sea $F = (F_1, \ldots, F_n)$ como en (3.6). Sean $\{\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_d^{(0)}\} \subset \mathbb{C}^n$ las d raíces aproximadas y $\{\mathbf{z}_1^*, \ldots, \mathbf{z}_d^*\} \subset \mathbb{C}^n$ las correspondiente raíces exactas en $V(F)$. Para cada $i = 1, \ldots, d$ y $k \geq 0$, definimos la $(k+1)$-ésima iteración de Newton como

$$\mathbf{z}_i^{(k+1)} := \mathbf{z}_i^{(k)} - JF(\mathbf{z}_i^{(k)})^{-1} F(\mathbf{z}_i^{(k)}),$$

donde $JF(\mathbf{z}_i^{(k)})$ es la matriz jacobiana de F y suponemos que sea invertible. Tenemos entonces, para cada $i = 1, \ldots, d$,

$$\|\mathbf{z}_i^{(0)} - \mathbf{z}_i^*\|_2 \leq \varepsilon.$$

Usando que el método de Newton converge cuadráticamente de cada $\mathbf{z}_i^{(0)}$ a $\mathbf{z}_i^*$, obtenemos

$$\|\mathbf{z}_i^{(k)} - \mathbf{z}_i^*\|_2 \leq \varepsilon \left(\frac{1}{2}\right)^{2^k - 1}.$$

Observación 3.3. En el paso de interpolación (3.5) puede que se pierda precisión, debido al *condition number* del interpolador de Lagrange, que en la base de monomios es de orden exponencial con respecto al número de nodos d aunque los nodos μ_i estén en el intervalo $[-1, 1]$.

Para evitar esto, lo que se propone es cambiar a una base ortogonal de polinomios, por ejemplo los polinomios de Chebyshev, que tiene el *condition number* del interpolador de Lagrange lineal en d, suponiendo que los nodos μ_i estén en el intervalo $[-1, 1]$. Por último, se puede volver de la base de Chebyshev a la base monomial resolviendo un sistema lineal.

Sea $q^{(k)}(T)$, $v^{(k)}(T) = (v_1^{(k)}(T), \ldots, v_n^{(k)}(T))$ la RUR aproximada correspondiente a $\{\mathbf{z}_1^{(k)}, \ldots, \mathbf{z}_d^{(k)}\}$ y sea $q^*(T)$, $v^*(T) = (v_1^*(T), \ldots, v_n^*(T))$ la RUR exacta que corresponde a $\{\mathbf{z}_1^*, \ldots, \mathbf{z}_d^*\} \subset \mathbb{C}^n$. Luego, suponiendo que no hay errores debidos al redondeo, la cota para el error de los coeficientes de la interpolación polinomial viene dada por

$$\|v_j^{(k)}(T) - v_j^*(T)\|_2 \leq \varepsilon d^2 \left(\frac{1}{2}\right)^{2^k - 1 - d}, \tag{3.7}$$

que converge a cero si $k \to \infty$, ya que d y ε se quedan fijos durante la iteración.
Análogamente, la cota para la precisión del polinomio $q^{(k)}(T)$ viene dada por

$$\|q^{(k)}(T) - q^*(T)\|_2 \leq \varepsilon d^2 \left(\frac{1}{2}\right)^{2^k - 1 - d}.$$

3.3.2. Método de Newton global

El método de Newton global aumenta directamente la precisión de la RUR aproximada, usando aritmética polinomial y sin emplear aproximaciones de las raíces.

Dados $F = (F_1, \ldots, F_n)$ y $u = \sum_{i=1}^n \lambda_i x_i$ en $\mathbb{Q}[x_1, \ldots, x_n]$, definimos la aplicación

$$\Phi : \mathbb{Q}^{(n+1)d} \to \mathbb{Q}^{(n+1)d}$$

como la aplicación de los vectores coeficientes de los siguientes polinomios de grado $d-1$:

$$\Phi : \begin{bmatrix} v_1(T) \\ \vdots \\ v_n(T) \\ q(T) - T^d \end{bmatrix} \mapsto \begin{bmatrix} F_1(v(T)) \bmod q(T) \\ \vdots \\ F_n(v(T)) \bmod q(T) \\ \sum_{i=1}^n \lambda_i v_i(T) - T \end{bmatrix}. \tag{3.8}$$

Observación 3.4. Si u, $q(T)$, $v_1(T), \ldots, v_n(T)$ es una RUR exacta de una componente de $\langle F \rangle$, entonces

$$\Phi(v_1(T), \ldots, v_n(T), q(T) - T^d) = 0.$$

Observación 3.5. Ya que $\mathbb{Q}^{(n+1)d}$ y $(\mathbb{Q}[T]/\langle q(T)\rangle)^{n+1}$ son isomorfos como espacios vectoriales, podemos considerar Φ como una aplicación entre

$$\Phi : (\mathbb{Q}[T]/\langle q(T)\rangle)^{n+1} \to (\mathbb{Q}[T]/\langle q(T)\rangle)^{n+1}.$$

Aplicamos el método de Newton $(n+1)d$-dimensional para converger localmente al vector coeficiente de una RUR exacta que es un cero de Φ.

Lema 3.1. *Sean $F = (F_1, \ldots, F_n)$, u, $q(T)$, $v_1(T), \ldots, v_n(T)$ como antes. Para $i = 1, \ldots, n$ definimos $m_i(T)$ y $r_i(T)$ como el cociente y el resto de la siguiente división con resto:*

$$F_i(v(T)) = m_i(T)q(T) + r_i(T). \tag{3.9}$$

La matriz jacobiana de Φ respecta la estructura de álgebra de $(\mathbb{Q}[T]/\langle q(T)\rangle)^{n+1}$ y viene dada por

$$J\Phi(v(T), q(T) - T^d) := \left[\begin{array}{ccc|c} & JF(v(T)) & & \begin{array}{c} -m_1(T) \\ \vdots \\ -m_n(T) \end{array} \\ \hline \lambda_1 & \cdots & \lambda_n & 0 \end{array}\right] \text{ mód } q(T). \tag{3.10}$$

Para definir la iteración de Newton correspondiente a Φ necesitamos imponer unas hipótesis.

Hipótesis 3.1. Sean $F = (F_1, \ldots, F_n)$, $u = \sum_{i=1}^n \lambda_i x_i$, $q(T)$, $v_1(T), \ldots, v_n(T)$ polinomios sobre $\mathbb{Q}$ como antes. Suponemos que

1. $q(T)$ es mónico de grado d,
2. $v_i(T)$ es de grado $d-1$ como máximo,
3. $\frac{\partial q(T)}{\partial T}$ es invertible módulo $q(T)$,
4. $\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) = T$,
5. $JF(v(T))$ es invertible módulo $q(T)$,
6. $J\Phi := J\Phi(v(T), q(T) - T^d)$ es invertible módulo $q(T)$.

Definición 3.3. Si $F(x_1, \ldots, x_n)$, $u(x_1, \ldots, x_n)$, $q(T)$, $v(T)$ son polinomios sobre $\mathbb{Q}$ que verifican las Hipótesis 3.1, definimos

$$\begin{aligned}
&u = \textstyle\sum_{i=1}^n \lambda_i x_i = T, \\
&\omega(T) := v(T) - \left(JF(v(T))^{-1} F(v(T)) \text{ mód } q(T)\right), \\
&\Delta(T) := \textstyle\sum_{i=1}^n \lambda_i \omega_i(T) - T, \\
&r(T) := F(v(T)) \text{ mód } q(T), \\
&U(T) := \tfrac{\partial v(T)}{\partial T} - \left(JF(v(T))^{-1} \tfrac{\partial r(T)}{\partial T} \text{ mód } q(T)\right), \\
&\Lambda(T) := \textstyle\sum_{i=1}^n \lambda_i U_i(T), \\
&V(T) := \omega(T) - \left(\tfrac{\Delta(T)}{\Lambda(T)} U(T) \text{ mód } q(T)\right), \\
&Q(T) := q(T) - \left(\tfrac{\Delta(T)}{\Lambda(T)} \tfrac{\partial q(T)}{\partial T} \text{ mód } q(T)\right).
\end{aligned}$$

Vamos a ver ahora que $V(T)$ y $Q(T)$ son las iteraciones de Newton para la función Φ.

Proposición 3.2. *Sean $F(x_1,\ldots,x_n)$, $u(x_1,\ldots,x_n)$, $q(T)$, $v(T)$ polinomios sobre $\mathbb{Q}$ verificando las Hipótesis 3.1. Entonces, $\Lambda(T)$ es invertible módulo $q(T)$ y $V(T)$, $Q(T)$ están bien definidas.*
Además,

$$\begin{bmatrix} V(T) \\ Q(T)-T^d \end{bmatrix} = \begin{bmatrix} v(T) \\ q(T)-T^d \end{bmatrix} - J\Phi^{-1} \begin{bmatrix} F(v(T)) \\ \sum_{i=1}^n \lambda_i v_i(T) - T \end{bmatrix} \text{ mód } q(T). \quad (3.11)$$

También se tiene

$$\sum_{i=1}^{n} \lambda_i V_i(T) = T.$$

Corolario 3.3. *La iteración definida converge localmente y converge cuadráticamente a una RUR exacta de una componente de $\langle F\rangle$, siempre que las Hipótesis 3.1 se cumplan en cada iteración.*

Aplicando α-theory a la función Φ, se puede certificar que la RUR inicial tiene la propiedad de convergencia cuadrática y acotar la distancia a la RUR exacta.

Sea $Q^{(0)}(T)$, $V^{(0)}(T) = (V_1^{(0)}(T),\ldots,V_n^{(0)}(T))$ la RUR inicial aproximada y supongamos que converja cuadráticamente al cero exacto $Q^*(T)$ y $V^*(T) = (V_1^*(T),\ldots,V_n^*(T))$ de la función Φ. Supongamos que existe un número ν tal que

$$\|Q^{(0)}(T) - Q^*(T)\|_2 \le \nu, \quad \|V_i^{(0)}(T) - V_i^*(T)\|_2 \le \nu \quad i = 1,\ldots,n,$$

donde la norma es la norma euclídea usual de los vectores coeficientes de los polinomios sobre $\mathbb{C}$.
Usando la convergencia cuadrática de la iteración global de Newton, obtenemos la siguiente cota para el error de los coeficientes de los polinomios en la k-ésima iteración:

$$\|Q^{(k)}(T) - Q^*(T)\|_2 \le \nu \left(\frac{1}{2}\right)^{2^k-1}, \quad \|V_i^{(k)}(T) - V_i^*(T)\|_2 \le \nu \left(\frac{1}{2}\right)^{2^k-1}, \quad (3.12)$$

que converge a cero cuando $k \to \infty$.

3.4. Reconstrucción de números racionales

Una vez calculada una aproximación suficientemente buena de la RUR de una componente racional de $\mathcal{I}$, para calcular la RUR exacta la idea es

reconstruir los únicos números racionales con denominador acotado e indistinguibles de los coeficientes de los polinomios en la RUR aproximada dentro de las estimas de precisión. Luego, usando métodos simbólicos, se puede verificar si la RUR con los coeficientes racionales así encontrados es una RUR exacta para una componente del sistema original.

Ya que las coordenadas de las raíces aproximadas vienen dadas como números complejos *floating point*, se pueden considerar como racionales de Gauss en $\mathbb{Q}(i)$ y lo mismo se puede hacer con los coeficientes de la RUR aproximada. De todas formas, como la RUR exacta tiene coeficientes racionales, nos quedaremos con la parte real de los coeficientes de la RUR aproximada. Por tanto, supondremos que los coeficientes de la RUR aproximada están en $\mathbb{Q}$, dados como números *floating point*.

El siguiente resultado implica que, si un número está suficientemente cerca a un número racional con denominador pequeño, entonces se puede encontrar este racional en tiempo polinomial.

Teorema 3.4. *Existe un algoritmo en tiempo polinomial que, dados $c \in \mathbb{Q}$ y $B \in \mathbb{N}$, verifica si existe un par de enteros (z, d) con $1 \leq d \leq B$ y*

$$| \, c - z/d \, | < \frac{1}{2B^2},$$

y, si existe, encuentra el único par de enteros.

Observación 3.6. El Teorema 3.4 no garantiza la existencia del par $(z, d) \in \mathbb{Z}^2$, solo la unicidad.

En nuestro caso, para calcular el par (z, d) para cada coeficiente c de la RUR aproximada, vamos a usar la cota $B \in \mathbb{N}$ tal que $\frac{1}{2B^2} \cong E$, donde E representa la estima de la precisión de la RUR aproximada, como en (3.7) o en (3.12). Luego, podemos definir

$$B := \left\lceil \frac{1}{\sqrt{2E}} \right\rceil.$$

Para un cálculo eficiente de los números racionales, se pueden usar el algoritmo extendido de Euclides o, equivalentemente, las fracciones continuas. Los polinomios racionales calculados con denominador acotado los vamos a denotar por

$$\hat{q}(T) \quad \text{y} \quad \hat{v}(T) = (\hat{v}_1(T), \ldots, \hat{v}_n(T)). \tag{3.13}$$

Observación 3.7. En el caso que el algoritmo devuelva que no existe un número racional a distancia menor que E con denominador menor o igual que B, hay que aumentar la precisión de E, es decir, aumentar la cota B al denominador. Eso se hace aplicando ulteriores iteraciones de Newton a las raíces aproximadas.

3.5. Terminación

Una pregunta que nos ponemos es si se pueden terminar las iteraciones o si hay que aumentar la precisión de la RUR aproximada.

Sean $\hat{q}(T)$ y $\hat{v}(T) = (\hat{v}_1(T), \ldots, \hat{v}_n(T))$ los polinomios racionales con denominadores acotados. Volvemos a considerar el sistema original sobredeterminado $f = (f_1, \ldots, f_m) \in \mathbb{Q}[x_1, \ldots, x_n]$, para $m \geq n$, con $\mathcal{I} = \langle f_1, \ldots, f_m \rangle$.

Vamos a reducir cada polinomio f_i por la base de Gröbner

$$\{\hat{q}(T), x_1 - \hat{v}_1(T), \ldots, x_n - \hat{v}_n(T)\},$$

o, equivalentemente, calculamos $f_i(\hat{v}(T))$ mód $\hat{q}(T)$.
Si todos se reducen a cero, entonces (3.13) es la RUR exacta de $\mathcal{I}$.
Si no todos reducen a cero, entonces la precisión de la RUR aproximada es demasiado baja o la iteración no converge a una RUR de una componente racional de $\mathcal{I}$.

Para decidir en qué caso estamos, vamos usar cotas superiores *a priori* para la altura de los coeficientes de una RUR de una componente racional de $\mathcal{I}$. Recordamos ahora la definición de altura de un polinomio:

Definición 3.4. Sea $p(T) = T^d + a_{d-1}T^{d-1} + \cdots + a_0 \in \mathbb{Q}[T]$ donde cada $a_i = \frac{z_i}{d_i}$ con $z_i \in \mathbb{Z}$ y $d_i \in \mathbb{N}$. Sea $P(T) \in \mathbb{Z}[T]$ un polinomio entero que es un múltiplo entero de $p(T)$, por ejemplo

$$P(T) = b_d T^d + b_{d-1}T^{d-1} + \cdots + b_0 := \left(\prod_{i=0}^{d-1} d_i\right) p(T) \in \mathbb{Z}[T].$$

La *altura de* p se define como

$$H(p) = H(P) = \frac{\text{máx}\{|b_i| \,:\, i = 0, \ldots, d\}}{\gcd\{b_i \,:\, i = 0, \ldots, d\}},$$

y definimos la *altura logarítmica de* p como

$$h(p) := \log H(p).$$

Observación 3.8. Observamos que si $\gcd(z_i, d_i) = 1$ para cada i, entonces

$$H(p) \geq \max_i\{|z_i|, d_i\} \tag{3.14}$$

y podemos usar la altura para acotar la magnitud de los numeradores y denominadores que aparecen en los coeficientes de los polinomios.

Supongamos que los polinomios del sistema original tienen grado como máximo D, altura logarítmica como máximo h, y sea δ el número de raíces en $V(\mathcal{I})$. Siguiendo [1], una cota superior para las alturas logarítmicas de los polinomios en una RUR exacta $q^*(T)$, $v_1^*(T), \ldots, v_n^*(T)$ de una componente de $\mathcal{I}$ viene dada por

$$h(q^*), h(v_i^*) \leq 12\delta n^4 h D^{n+1} \log(n\delta) \quad i = 1, \ldots, n \tag{3.15}$$

de la que deducimos una cota superior para la altura

$$H(q^*), H(v_i^*) \leq Hn\delta e^{12\delta n^4 D^{n+1}} \quad i = 1, \ldots, n, \tag{3.16}$$

donde $H = e^h$.

Una vez tengamos una cota *a priori* para la altura de los polinomios, se puede usar (3.14) y verificar si la cota B para los denominadores sobrepasa la cota *a priori* dada en (3.16). Si la supera, se concluye que la iteración no converge a una RUR exacta de una componente racional de $\mathcal{I}$ y se termina. Si no, hay que seguir para aumentar la precisión de la aproximación.

Resumiendo (ver [1]):

Teorema 3.5. *Sea $\mathcal{I}$ como antes. Supongamos que $q^*(T)$, $v_1^*(T), \ldots, v_n^*(T)$ es una RUR exacta de una componente racional de $\mathcal{I}$. Definimos la* altura máxima *como*

$$H^* := \text{máx}\{H(q^*), H(v_1^*), \ldots, H(v_n^*)\}.$$

Supongamos que una RUR aproximada $q(T)$, $v_1(T), \ldots, v_n(T)$ cumple

$$\|q(T) - q^*(T)\|_2, \|v_i(T) - v_i^*(T)\|_2 \leq E < \frac{1}{2(H^*)^2} \quad i = 1, \ldots, n, \tag{3.17}$$

para $E > 0$, y sea $\hat{q}(T)$, $\hat{v}_1(T), \ldots, \hat{v}_n(T)$ obtenida por reconstrucción de números racionales sobre los coeficientes de $q(T)$, $v_1(T), \ldots, v_n(T)$ usando la cota $B := \lceil (2E)^{-1/2} \rceil > H^*$. Entonces

$$\hat{q}(T) = q^*(T),\ \hat{v}_1(T) = v_1^*(T), \ldots, \hat{v}_n(T) = v_n^*(T).$$

Teorema 3.6. *Sea $f \in \mathbb{Q}[x_1, \ldots, x_n]^m$ como antes y supongamos que H y D son la altura y el grado máximos de los polinomios en f, respectivamente. Sea δ la cardinalidad de las raíces de f en $\mathbb{C}^n$. Supongamos que tenemos una cota superior E para la precisión de la RUR aproximada $q(T)$, $v_1(T), \ldots, v_n(T)$, (3.7) o (3.12). Sea $B := \lceil (2E)^{-1/2} \rceil$ y supongamos que*

$$B \geq Hn\delta e^{12\delta n^4 D^{n+1}}.$$

Sea $\hat{q}(T)$, $\hat{v}_1(T), \ldots, \hat{v}_n(T)$ obtenida por reconstrucción de números racionales sobre los coeficientes de $q(T)$, $v_1(T), \ldots, v_n(T)$ usando la cota B para los denominadores. Si

$$f(\hat{v}(T)) \not\equiv 0 \text{ mód } \hat{q}(T),$$

entonces no existe una RUR exacta de una componente racional de $\mathcal{I}$ a distancia menor que E de $q(T)$, $v_1(T), \ldots, v_n(T)$.

En seguida damos el número de iteraciones necesarias asintóticamente en el *best case* y en el *worst case*.

Teorema 3.7. *Sea $f \in \mathbb{Q}[x_1, \ldots, x_n]^m$ como antes y $\mathcal{I} = \langle f \rangle$.*

1. *Supongamos que $q^*(T)$, $v_1^*(T), \ldots, v_n^*(T)$ es una RUR exacta de una componente racional de $\mathcal{I}$ y que $q^{(0)}(T)$, $v_1^{(0)}(T), \ldots, v_n^{(0)}(T)$ es una aproximación inicial de la RUR que converge cuadráticamente a $q^*(T)$, $v_1^*(T), \ldots, v_n^*(T)$. Entonces el número de iteraciones necesaria para encontrar $q^*(T)$, $v_1^*(T), \ldots, v_n^*(T)$ está asintóticamente acotado por*

   $$\mathcal{O}(\log(d) \log\log(H^* E_0)),$$

   donde $H^* = \max\{H(q^*), H(v_1^*), \ldots, H(v_n^*)\}$, $d = \deg_T(q)$ y E_0 es una cota superior de la distancia euclídea entre $q^{(0)}(T), v^{(0)}(T)$ y $q^*(T), v_1^*(T)$.

2. *Supongamos que $q^{(0)}(T)$, $v_1^{(0)}(T), \ldots, v_n^{(0)}(T)$ es una RUR inicial aproximada que converge cuadráticamente a los polinomios $v_1^*(T), \ldots, v_n^*(T)$, $q^*(T)$, que pero tienen coeficientes irracionales. En este caso se necesitan*

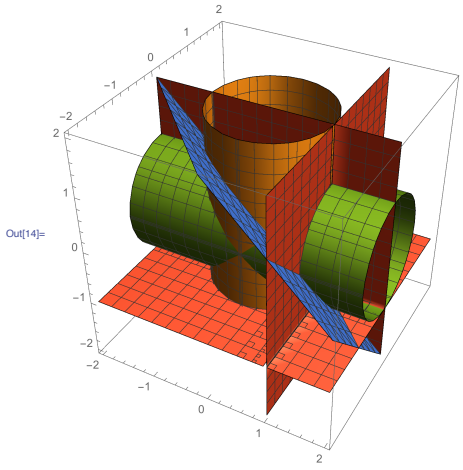
   $$\mathcal{O}(n \log(d\delta n D) \log\log(H E_0))$$

   iteraciones para concluir que nuestra iteración no converge a una RUR exacta de una componente racional de $\mathcal{I}$. H y D son la altura y el grado máximos de los polinomios en f, respectivamente, δ es el número de raíces en $V(\mathcal{I})$ y d, E_0 como antes en 1.

3.6. Ejemplos

Ejemplo 3.1. Consideramos el sistema polinomial sobredeterminado

$$f(x, y, z) = \begin{cases} x^2 + y^2 - 1 = 0 \\ x + y + z = 0 \\ y^2 + z^2 - 1 = 0 \\ xy + xyz - yz - y = 0 \end{cases}$$



El sistema tiene $d = 2$ soluciones

$$\xi_1 = (1, 0, -1)$$
$$\xi_2 = (-1, 0, 1)$$

Con $\lambda = (0, 0, 1) \in \mathbb{Q}^3$ se verifica que $\langle \lambda, \xi_1 \rangle \neq \langle \lambda, \xi_2 \rangle$ y con ello podemos elegir a $u = z$ como elemento primitivo para el ideal generado por los polinomios en f.

Vamos a calcular la RUR inicial.

$$\mu_1 = \langle \lambda, \xi_1 \rangle = -1,$$
$$\mu_2 = \langle \lambda, \xi_2 \rangle = 1.$$

Por tanto, el único polinomio mónico $q(T)$ de grado $d = 2$ que se anula en los valores de u correspondientes a cada punto de $V = \{\xi_1, \xi_2\}$ viene dado por

$$q(T) = (T + 1)(T - 1) = T^2 - 1.$$

Vamos ahora a calcular los interpoladores de Lagrange cumpliendo $v_j(\mu_i) = \xi_{i,j}$ para $i = 1, 2, 3$.

$$\begin{cases} v_1(\mu_1) = 1 \\ v_1(\mu_2) = -1 \end{cases} \Rightarrow v_1(T) = -T$$

$$\begin{cases} v_2(\mu_1) = 0 \\ v_2(\mu_2) = 0 \end{cases} \Rightarrow v_2(T) = 0$$

$$\begin{cases} v_3(\mu_1) = -1 \\ v_3(\mu_2) = 1 \end{cases} \Rightarrow v_3(T) = T$$

Por tanto, la RUR inicial viene dada por

$$q(T) = T^2 - 1,\ v_1(T) = -T,\ v_2(T) = 0,\ v_3(T) = T,$$

cuyos coeficientes son racionales.
Para ver si es exacta, verificamos si $f(v_1(T), v_2(T), v_3(T)) \equiv 0$ mód $q(T)$:

$$f(-T, 0, T) = \begin{pmatrix} -1 + T^2 \\ 0 \\ -1 + T^2 \\ 0 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ mód } q(T).$$

Como la RUR es exacta, consideramos ahora el sistema cuadrado

$$R(T, x, y, z) = \begin{cases} q(T) \\ x - v_1(T) \\ y - v_2(T) \\ z - v_3(T) \end{cases} = \begin{cases} T^2 - 1 = 0 \\ x + T = 0 \\ y = 0 \\ z - T = 0 \end{cases} . \tag{3.18}$$

Calculamos con `Bertini` las soluciones del sistema (3.18) y encontramos dos soluciones reales x_1 y x_2:

```
1.000000000000000e+00 0.000000000000000e+00
-1.000000000000000e+00 0.000000000000000e+00
-1.486505071804674e-18 1.391827266165299e-18
1.000000000000000e+00 0.000000000000000e+00

-1.000000000000000e+00 0.000000000000000e+00
1.000000000000000e+00 0.000000000000000e+00
2.012592539165592e-16 2.083320349820768e-16
-9.999999999999998e-01 4.996003610813204e-16
```

Como el sistema es cuadrado, podemos aplicarle `alphaCertified` y obtenemos:

```
Number of points tested:              2
Certified approximate solutions:      2
Certified distinct solutions:         2
Certified real distinct solutions:    2
```

Entonces, tenemos que las raíces aproximadas ξ_1 y ξ_2 son ceros certificados de $f(x, y, z) = 0$.

Bibliografía

[1] T.A. Akoglu, J.D. Hauenstein, and A. Szanto. Certifying solutions to overdetermined and singular polynomial systems over Q. arXiv:1408.2721, 2014.

[2] D.J. Bates, J.D. Hauenstein, C. Peterson, and A.J. Sommese. A numerical local dimension test for points on the solution set of a system of polynomial equations. *SIAM Journal on Numerical Analysis*, 47(5), 3608–3623, 2009.

[3] D.J. Bates, J.D. Hauenstein, A.J. Sommese and C.W. Wampler. Numerically Solving Polynomial Systems with Bertini. *Society for Industrial and Applied Mathematics*, 2013.

[4] J.D. Hauenstein and F. Sottile. Algorithm 921: alphaCertified: certifying solutions to polynomial systems. *ACM Trans. Math. Software*, 38(4):28, 2012.

[5] E. Kostlan. Random polynomials and the statistical fundamental theorem of algebra. 1987. Disponible en `http://www.developmentserver.com/randompolynomials/rpsfta`.

[6] A. Morgan. Solving polynomial systems using continuation for engineering and scientific problems. Prentice-Hall, 1987.

[7] A.J. Sommese, J. Verschelde, and C.W. Wampler. Introduction to Numerical Algebraic Geometry. Chapter 8, Solving Polynomial Equations: Foundations, Algorithms, and Applications, Algorithms and Computation in Mathematics 14, A. Dickenstein, I.Z. Emiris (Eds.), Springer, 2005, pp. 339–392.

[8] A. Szanto. Certification of approximate roots of exact polynomial systems. Notices AMS 63 (2016), 1160–1162.