Capítulo 5

Búsqueda de funciones de Lyapunov con lil-gp

Se va a describir en este capítulo la solución desarrollada para alcanzar el objetivo del proyecto. Se recuerda en primera instancia el cometido que nos ocupa para situar al lector. El trabajo parte de un artículo publicado en la red, sobre búsqueda de funciones de Lyapunov haciendo uso de la programación genética [7].

1. EL PROBLEMA

Se recuerda en este punto, el problema que se está intentando resolver. Se trata de encontrar una función de Lyapunov para un sistema de la forma:

$$\dot{x} = f(x)$$

donde x pertenece a $\Re^n$. En particular, los sistemas que se van a estudiar son de dimensión 2, es decir, x pertenece a $\Re^2$ y los sistemas se pueden escribir como:

$$\dot{x}_1 = f_1(x_1, x_2)$$
$$\dot{x}_2 = f_2(x_1, x_2)$$

Las variables de estado pertenecerán a un cierto dominio D. El sistema de arriba será estable en el origen si en un cierto dominio D que contenga al origen, se encuentra una función $V(x_1, x_2)$ que cumpla las siguientes condiciones:

1. $V(0,0) = 0$
2. $V(x_1, x_2) > 0 \qquad \forall (x_1, x_2) \in D - \{(0,0)\}$
3. $\dot{V}(x_1, x_2) \leq 0 \qquad \forall (x_1, x_2) \in D$

Dicha función se dice que es una función de Lyapunov del sistema en cuestión, o bien, que posee un equilibrio estable en el origen . V debe estar definida en el origen pero puede que no sea igual a cero. Eso no representa un problema pues basta con definir una nueva función $\hat{V}(x_1, x_2) = V(x_1, x_2) - V(0,0)$ y considerar para la misma sólo las condiciones 2 y 3.

Por lo tanto, se pretende encontrar una función V que cumpla las condiciones de arriba. Enmarcando ésto en un problema de programación genética, se puede afirmar que los individuos de la población serán árboles que representarán funciones de dos variables independientes (x_1 y x_2).

2. PRIMEROS PASOS

Como se vió en la sección 2.2 del capítulo 3, antes de comenzar a desarrollar la solución al problema es necesario dar una serie de pasos previos como la elección de los parámetros que gobernarán la ejecución del programa o el conjunto de nodos terminales y no terminales que formarán los árboles de la población.

2.1. Parámetros

En el transcurso del proyecto, se han hecho pruebas con cuatro sistemas diferentes. Los parámetros elegidos para dichas pruebas[1] son los siguientes:

[1] Aunque uno de esos cuatro sistemas presentaba características especiales que hicieron variar de forma considerable algunos parámetros. Se verá más adelante.

- El tamaño de la población fué en la mayoría de los casos de 200 individuos aunque se probó también con poblaciones de 300, 400 e incluso de 500 individuos.

- El máximo número de generaciones permitidas en la ejecución del programa se estableció en 20. Numerosos experimentos han llevado a algunos expertos a la conclusión de que una buena proporción tamaño_de_población/máximo_número_generaciones, es la dada por 200/20.

- Cada individuo de la población fué testeado en 200 puntos diferentes (x_1, x_2) generados de forma aleatoria. En la aplicación desarrollada se definió el parámetro `app_no_casos`, en el que el usuario puede establecer el número de casos de prueba desde el fichero de entrada de parámetros.

- Los operadores utilizados fueron el cruce, la reproducción y la mutación. El cruce con una tasa de ocurrencia del 90%, la reproducción en una orquilla del 8-9% y la mutación con un procentaje que osciló entre un 1 y un 2%.

- Para los tres operadores mencionados, se utilizó el mismo método de selección de individuos: el *tournament*, estableciendo el valor de su parámetro asociado *size* = 4. El método de selección por torneo es el más adecuado para evitar el problema de la convergencia prematura.

- En cada ejecución, hay que darle un valor distinto a la semilla `random_seed` para que la población inicial generada sea diferente a las anteriores.

Todos estos parámetros van especificados en el fichero de entrada de parámetros denominado `input.file`.

2.2. Funciones y Terminales

Se debe tomar una decisión a cerca de los miembros que compondrán el conjunto de nodos terminales y no terminales. Decisión que en algunos casos puede resultar muy importante ya que, por ejemplo, incluir funciones innecesarias puede ralentizar la convergencia del proceso al mismo tiempo que dejar de considerar funciones realmente útiles puede poner barreras insalvables a la hora de llegar a la solución óptima.

En mi implementación, se han tomado como nodos terminales las dos variables de estado x_1 y x_2, así como constantes numéricas generadas de forma aleatoria (ERCs) en un intervalo de [-10, 10]. Las constantes numéricas pueden resultar bastante útiles a la hora de formar una función de Lyapunov. Además, se consideró que las ecuaciones de movimiento $f_1(x_1, x_2)$ y $f_2(x_1, x_2)$ podrían resultar beneficiosas como terminales. En consecuencia, el conjunto de terminales *T* se puede escribir como:

$$T = \{x_1, x_2, f_1, f_2, -10, ..., 10\}$$

El conjunto de nodos no terminales establece la lista de funciones de entrada (o funciones elementales) que se van a utilizar. En cualquier caso, las cuatro operaciones aritméticas deben estar presentes: suma, resta, multiplicación y división. Con la división hay que ser precavidos ya que se podría producir en un momento determinado de la evaluación de un individuo, una división entre cero. El programa debe estar preparado para este caso ya que en la evaluación de un árbol siempre se debe retornar un valor numérico válido. Por ello, la solución que se ha adoptado aquí es que la función división devuelva[2] un 1 cuando el divisor sea igual a 0.

Por otro lado, el resto de funciones incluidas depende del sistema que se esté analizando. La operación "elevado al cuadrado: x^2" puede resultar muy útil a la hora de cumplir con la condición que establece que una función de Lyapunov debe ser definida positiva. De hecho, siempre será considerada en nuestro caso. Por ejemplo, en un sistema que describa el comportamiento de un péndulo, las funciones seno y coseno pueden resultar vitales con lo que un conjunto válido podría ser:

$$F = \{ +, -, /, *, ( )^2, \text{seno}( ), \text{coseno}( ) \}$$

Todo el código fuente relacionado con la definición de estos conjuntos está en la función `app_build_function_set()`.

3. LA APTITUD

Quizás, la tarea más difícil y determinante en la implementación del problema sea la de calcular el grado de aptitud de un individuo. En el caso que nos ocupa, la aptitud deberá indicar cómo de cerca se encuentra un determinado árbol de ser una función de Lyapunov. Es decir, surge la necesidad de comprobar para cada individuo de la población, si éste cumple las condiciones:

1. $V(0,0) = 0$
2. $V(x_1, x_2) > 0 \qquad \forall (x_1, x_2) \in D - \{(0,0)\}$
3. $\dot{V}(x_1, x_2) \leq 0 \qquad \forall (x_1, x_2) \in D$

Y ésto para cada caso de prueba.

[2] Como se verá más adelante, esta solución puede acarrear otro tipo de problemas.

3.1. La Derivada de V(x) Respecto al Tiempo

A primera vista, aparece una primera complicación: se debe evaluar la derivada parcial de V con respecto al tiempo. Lo que se hace es una aproximación mediante la cuál, se calcula la parcial de V usando la siguiente expresión

$$\dot{V}(x) = \left(\frac{\partial V}{\partial x} \right)^T f(x)$$

Para ser más explícitos, el programa calculará esta derivada parcial usando la expresión:

$$\dot{V}(x_1, x_2) = \frac{\partial V}{\partial x_1} \cdot \dot{x}_1 + \frac{\partial V}{\partial x_2} \cdot \dot{x}_2$$

donde

$$\frac{\partial V}{\partial x_1} = \frac{V(x_1 + h, x_2) - V(x_1, x_2)}{h}$$

$$\frac{\partial V}{\partial x_2} = \frac{V(x_1, x_2 + h) - V(x_1, x_2)}{h}$$

Estas expresiones si se pueden calcular de forma exacta ya que conocemos las expresiones de f_1 y f_2, y cualquier función V se puede evaluar para cualquier par de valores de x_1 y x_2. Se toma un valor de h suficientemente pequeño como para que la aproximación sea válida, y suficientemente grande como para que la computadora sea capaz de distinguir x_1 de x_1+h. En el caso de que se tome un h tan pequeño como para que el sistema considere ambos valores iguales, el programa presenta un fragmento de código que detecta esta situación y la corrige. Se han obtenido resultados satisfactorios para un $h=10^{-5}$.

3.2. Los Casos de Prueba

El parámetro definido especialmente para esta aplicación `app_no_casos`, establece el número de puntos diferentes en los que será evaluado el individuo, con el fin de asignarle un valor de aptitud . Esto se traduce en que, por ejemplo si `app_no_casos = 200,` cada árbol de la población será evaluado en 200 pares (x_1, x_2) generados de forma alaetoria, antes de asignarle un valor de aptitud . Estos pares se almacenan en tablas definidas especialmente para ese cometido.

Por otro lado, surge la necesidad de evaluar un mismo individuo en distintos puntos, para un mismo caso de prueba. Es decir, para un determinado caso de prueba (x_1, x_2) = <lo que sea>, se tiene que calcular el valor del árbol en dicho punto; pero también es necesario evaluar el mismo árbol en los pares (x_1+h, x_2) y (x_1, x_2+h). La función de lil-gp `evaluate_tree()` devuelve el valor del árbol con los valores que en ese momento tengan las variables globales `g.x1` y `g.x2`. De ese modo, no hay más que poner las variables `g.x1` y `g.x2` al valor que nos interese justo antes de llamar a la función `evaluate_tree()`.

3.3. Cálculo de la Aptitud

El cálculo del nivel de aptitud de un individuo se ha basado en el porcentaje de puntos evaluados (casos de prueba) en los que aquél verifica las condiciones para ser una función de Lyapunov.

En primer lugar, se evalúa al individuo en el origen (0, 0) y su valor se almacena en la variable `vorig`. A continuación, para cada caso de prueba, se dan los siguientes pasos:

1) Se evalúa al individuo en el caso actual (x_1, x_2) y el resultado se guarda en la variable `vobj`. Además, se calcula la parcial respecto al tiempo de V siguiendo la expresión mostrada en la sección 3.1, y se guarda el resultado en la variable `vprima`.

2) Se comprueba si se cumple la primera condición para ser una función de Lyapunov, es decir, se comprueba si `vobj-vorig > 0`. En caso afirmativo, se incrementa en una unidad el *aptitud bruta* (`r_fitness`) y se activa una bandera (`cond1=1`) para señalar que la primera condición se cumple.

3) Se hace lo propio con la segunda condición; esto es, se mira si `vprima < = 0`. En caso afirmativo, se incrementa de nuevo `r_fitness` en una unidad y se activa una segunda bandera (`cond2=1`) para indicar que la segunda condición también se cumple.

4) Por último, si se cumplen ambas condiciones simultáneamente, se incrementa en una unidad el campo `hits` asociado al individuo que está siendo evaluado. Esto significa que dicho individuo cumple todos los requisitos para ser una función de Lyapunov, cuando la función es evaluada en (x_1, x_2).

Al finalizar el bucle para todos los casos de prueba, tendré en el campo `hits` del individuo, el número de puntos en los que ha sido testeado con éxito. En cuanto al *aptitud bruta*, éste se incrementaba en uno cada vez que verificaba una de las dos condiciones de Lyapunov. En consecuencia, si un individuo ha sido evaluado con éxito en todos los casos de prueba, la variable `r_fitness` valdrá al final del bucle 2 * `app_no_casos`.

El criterio que se ha adoptado en la solución al problema es considerar que un individuo representará una función de Lyapunov cuando sea evaluado con éxito en el 100% de los casos de prueba, o lo que es lo mismo, cuando se dé:

```
r_fitness = 2 * app_no_casos
```

Teniendo en cuenta las definiciones dadas en el capítulo 4 para los otros dos tipos de aptitud , *aptitud estándar* y *aptitud ajustada*, sus variables asociadas se escribirían como:

```
s_fitness = 2*app_no_casos – r_fitness

a_fitness = 1 / (1 + s_fitness)
```

Todo el código relacionado con el cálculo del grado de aptitud de un individuo, se encuentra en la función `app_eval_fitness ()`.

4. CRITERIO DE TERMINACIÓN Y PRODUCCIÓN DE RESULTADOS

El criterio de terminación que se ha adoptado obedece al siguiente enunciado: "La ejecución del programa se detendrá por dos motivos diferentes. Bien porque se haya alcanzado el máximo número de generaciones permitidas, o bien porque se haya encontrado un individuo que represente una solución óptima al problema planteado".

Para comprobar si un individuo representa una solución óptima al problema, simplemente se compara el valor de su campo `hits` con el número de puntos en que ha sido testeado. Si coinciden, entonces se considera que ese árbol es una función de Lyapunov. En otras palabras, si un determinado individuo cumple los requisitos para ser una función de Lyapunov en el 100% de los puntos en que ha sido testeado, entonces se considera que hemos encontrado una solución válida al problema.

Supongamos que el programa ha detenido su ejecución porque ha encontrado un individuo que representa una solución válida (es decir, una función de Lyapunov). Dicho individuo se imprime en el fichero de salida `lyapunov.bst`. A continuación hay que comprobar manualmente que, efectivamente, dicho individuo es realmente una función de Lyapunov.

Normalmente, dada una expresión matemática para una función de Lyapunov, resulta complicado analizar si la misma cumple las condiciones:

1. $V(0,0) = 0$
2. $V(x_1, x_2) > 0 \qquad \forall (x_1, x_2) \in D - \{(0,0)\}$
3. $\dot{V}(x_1, x_2) \leq 0 \qquad \forall (x_1, x_2) \in D$

Por ello, lo que se hace en primer lugar es calcular la derivada parcial con respecto al tiempo $\dot{V}$ y a continuación se representan gráficamente tanto V como $\dot{V}$ con la herramienta Matlab. Además, se ha escrito un pequeño programa en Matlab que detecta el mínimo de V y el máximo de $\dot{V}$, así como los puntos en donde ocurren (incluidos en el CD adjunto). Ésto facilita la tarea de comprobación de resultados a posteriori.

5. PROBLEMAS PLANTEADOS

Como ya se ha mencionado anteriormente, se han realizado pruebas con cuatro sistemas diferentes: un sistema polinomial simple, un sistema no lineal que contiene la función tangente y dos clases de péndulos. A lo largo de estas pruebas han surgido una serie de inconvenientes que se comentarán en esta sección.

5.1. Falsas Alarmas

Se dice que se produce una falsa alarma cuando el programa produce una función de Lyapunov y resulta que en realidad no lo es. Este es un fenómeno que apareció en numerosas ocasiones y que se ha intentado paliar con cierto éxito.

Una falsa alarma puede aparecer por varios motivos. Uno de ellos es que se dé el caso de que en todos los puntos de testeo, la función cumpla los requisitos necesarios aunque haya puntos del dominio del sistema en los que la función no verifique las condiciones de Lyapunov. Ante esta situación lo único que cabe hacer es aumentar la malla de puntos de testeo para minimizar la probabilidad de error. Pero se debe tener cuidado con esta medida ya que cuantos más casos de prueba se analicen, más tiempo se tarda en ejecutar el programa.

Otro tipo de falsa alarma se ve claramente con el siguiente ejemplo. En una de las ejecuciones que se realizaron, el programa propuso como función de Lyapunov la siguiente expresión:

$$V(x_1, x_2) = \sqrt{atan(-1.91 \times atan(x_1 \times \frac{\text{sen}(-1.47)^2}{x_1}))}$$

Esta función evaluada en el punto (0, 0) produce un valor de cero y en el resto de puntos, los valores de x_1 se cancelan y la función es una constante de valor 0.990058 aproximadamente. Es decir, una función que en (0, 0) vale cero y en el resto de puntos tiene un valor constante e igual a 0.990058, es una función que cumple los requisitos para ser una función de Lyapunov.

Sin embargo se ve claramente como no es así porque la variable de estado se cancela.

Otra causa habitual de falsa alarma es la aparición de una división por cero. Se recuerda que el programa está implementado de tal forma que cuando un árbol está siendo evaluado y se encuentra una división entre cero, el resultado de esta operación es igual a 1. Teniendo en cuenta ésto, considérese la siguiente función de Lyapunov según el programa genético:

$$V(x_1, x_2) = \frac{f_2 \cdot (x_1 + x_2)}{(f_2 + f_1 - 8.66) \cdot (f_2 - f_2)}$$

Cuando el programa evalúa esta función en (0, 0) toma como resultado 0, ya que el numerador vale cero. En el resto de puntos de testeo, sean cuales sean, el resultado de la evaluación es 1 porque tenemos una división entre cero. En consecuencia, por el mismo motivo del ejemplo anterior, el programa entiende que es una función de Lyapunov válida ya que:

1) $V(0,0) = 0$
2) $V(x_1, x_2) > 0 \qquad \forall (x_1, x_2) \in D - \{(0,0)\}$
3) $\dot{V}(x_1, x_2) \leq 0 \qquad \forall (x_1, x_2) \in D$

teniendo en cuenta las expresiones utilizadas para evaluar $\dot{V}$:

$$\dot{V}(x_1, x_2) = \frac{\partial V}{\partial x_1} \cdot \dot{x}_1 + \frac{\partial V}{\partial x_2} \cdot \dot{x}_2$$

$$\frac{\partial V}{\partial x_1} = \frac{V(x_1 + h, x_2) - V(x_1, x_2)}{h}$$

$$\frac{\partial V}{\partial x_2} = \frac{V(x_1, x_2 + h) - V(x_1, x_2)}{h}$$

Este problema ha sido combatido de la siguiente manera. En general, aquellos individuos que puedan presentar operaciones "problemáticas" como división entre cero o raíz cuadrada de un número negativo no nos van a interesar. Esto es, este tipo de individuos nunca representarán una solución real al problema planteado. Para descartarlos, se declara una variable global que actuará como bandera. En el cuerpo de la función C que implementan el operador división, se comprueba si el divisor es igual a cero. En caso afirmativo, se activa la bandera y a la hora de asignar un valor de aptitud al árbol que está siendo evaluado se mira aquella. Si está levantada la bandera, le damos aptitud 0 a dicho individuo. Esto es, se le está diciendo al programa genético que aquellos individuos con divisiones por cero, representan la peor solución posible al problema.

Se hizo lo mismo con la raíz cuadrada de números negativos y el resultado fué positivo. El programa dejó de producir falsas alarmas debido a esta causa.

5.2. El Tamaño de los Individuos

A lo largo de los numerosos ensayos que se han hecho con los distintos sistemas, se ha observado un fenómeno que se repite en todos ellos. A medida que las generaciones se van sucediendo, el tamaño de los individuos va creciendo cada vez más. Ésto puede representar un problema cuando ya han transcurrido bastantes generaciones ya que en términos generales, un árbol con un tamaño excesivo no representa habitualmente una función de Lyapunov válida.

Por ello, en la solución implementada (y ya que el sistema lil-gp lo permite) se ha limitado el tamaño de los individuos en profundidad sin establecer ningún límite en el número de nodos. El límite establecido ha oscilado entre 7 y 12. Estos valores han sido tomados sólo en función de la observación de los resultados producidos por el programa y pueden ser modificados por el usuario en cualquier momento con el fín de experimentar otras estrategias.

5.3. Expansión del Dominio

De los cuatro sistemas estudiados en este proyecto, en tres de ellos se encontraron funciones de Lyapunov que demostraban su estabilidad pero el cuarto presentó más problemas. Este sistema está descrito por las siguientes ecuaciones:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \operatorname{sen} x_1 - 12k_1 \operatorname{sen} x_1 \cos^2 x_1 + (3k_1 - 1)\frac{\operatorname{sen} x_1 \cos x_1}{1 + \cos x_1} - k_2 x_2 \cos x_1$$

Donde se tomaron valores para las constantes k_1 y k_2 de 1 en ambos casos, y las variables de estado pertenecen al dominio:

$$x_1 \in [-p, +p]$$
$$x_2 \in [-3, +3]$$

El intervalo para x_2 se puede variar sin que afecte mucho al resultado final obtenido.

Pues bien, se realizaron numerosas ejecuciones sin obtener ningún resultado. Existe la posibilidad de que el sistema no tenga función de Lyapunov pero se adoptó una estrategia para intentarlo de otro modo.

El objetivo es demostrar que el sistema posee un equilibrio estable en (0, 0) cuando las variables de estado pertenecen al dominio arriba indicado. Como las primeras pruebas no proporcinaron ningún resultado se hizo lo siguiente. Inicialmente se tiene en cuenta sólo un pequeño dominio en torno al origen (0, 0). En este pequeño dominio, es más probable que exista una función de Lyapunov para el sistema. El programa genético comienza pues, generando todos los casos de prueba pertenecientes a este dominio inicial. Si en una determinada generación aparece una función de Lyapunov, ahora no se detiene la ejecución del programa sino que hacemos el dominio un poco mayor y seguimos probando. El proceso continuaría hasta que apareciera una función de Lyapunov en el dominio original, momento en que se detendría el programa. La situación se muestra gráficamente en la figura 5.1.

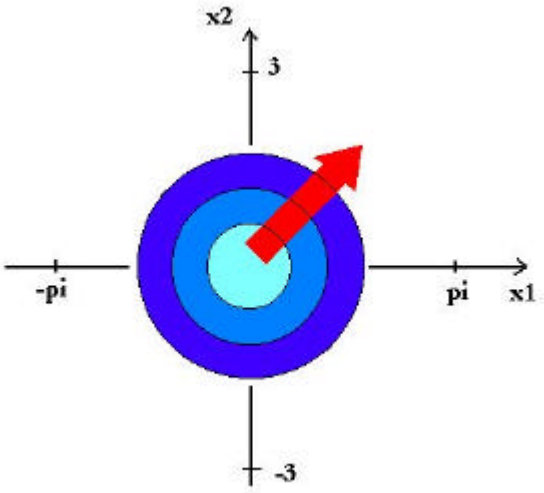

Figura 5.1 Dominio creciente en torno al origen

Ésto implicaba realizar algunos cambios en el código. En primer lugar, los casos de prueba se van a generar dentro de un dominio circular siguiendo un modelo de coordenadas polares. Por ejemplo, considérese un dominio inicial de radio 0.01p; entonces para cada caso de prueba se genera de forma aleatoria un radio *r* comprendido entre 0 y el máximo permitido, que actualmente es 0.01p, y un ángulo ? comprendido entre 0 y 2p. Dados estos dos parámetros, el punto de testeo se calcula como:

$$x_1 = r \cdot \operatorname{sen} q$$
$$x_2 = r \cdot \cos q$$

Por otro lado, el criterio de terminación de la ejecución del programa también debe ser modificado. Hasta aquí, si un individuo era evaluado con éxito en todos los casos de prueba, el programa se detenía. Sin embargo, si este hecho ocurre ahora, lo que se hace es aumentar gradualmente el radio del dominio y seguir con el proceso evolutivo. Esta forma de actuar conlleva el aumentar considerablemente el número máximo de generaciones permitidas ya que para alcanzar el objetivo final (una función de Lyapunov en el dominio completo) se debe culminar el proceso evolutivo en varias

ocasiones. Efectívamente, se pasó de un máximo de 20 generaciones a una orquilla de 70-100 generaciones.

En otras palabras, la idea es: "Se intuye que en un pequeño dominio en torno al origen puede existir una función de Lyapunov. Comenzamos pues probando en dicho dominio y si tenemos éxito, vamos aumentando el dominio poco a poco haciendo evolucionar esos mismos individuos en un entorno que cada vez se irá aproximando más al deseado".

Tras realizar varias pruebas se observa que si bien no se encuentran funciones de Lyapunov en el dominio original, el comportamiento del proceso evolutivo si es el esperado. Es decir, en un pequeño entorno al origen aparece una función de Lyapunov, luego se alcanza un pico en el nivel de aptitud ; en la siguiente generación el dominio ya es distinto y el nivel de aptitud comienza a bajar (no aparece ninguna función de Lyapunov). Varias generaciones después, aparece un nuevo individuo exitoso, y así.

Fué necesario crear un nuevo fichero de salida (`lyapunov.res`) para mostrar de forma adecuada estos resultados. En él se refleja para cada generación, el mejor individuo y el radio del dominio correspondiente.

Todo el código fuente desarrollado (fichero app.c) se puede ver en el apéndice A.

En el CD adjunto a la memoria se encuentra la distribución software de lil-gp y una carpeta llamada `Lyapunov` que contiene todos los ficheros necesarios para correr la aplicación. También se incluyen los dos programas Matlab que testean los resultados a posteriori, denominados `verv` y `vervprima` y que se encuentran en la carpeta `Matlab`.