Universitat autònoma de Barcelona

Grau en matemàtiques

Memòria de la pràctica 2

Mètodes numèrics

Aleix
Suárez Sayabera

Maig 2024

Índex

1 Introducció i objectius

Aquesta pràctica consisteix en proporcionar unes rutines de càlcul per tal de cercar zeros de funcions. Se'ns posa en el cas que el Departament de Dinàmica del Vol de l'Agència Espacial Catalana, precisa d'aquestes rutines per tal de fer servir el seu software de generació de trajectòries.
Per tal d'aconseguir això, se'ns demanen dos tipus de rutines de càlcul, una que utilitzi el mètode de la bisecció, i una altra que utilitzi el mètode de Newton.
L'objectiu és, doncs, programar aquestes dues rutines de la forma més eficient possible.

2 Implementació de les rutines per cercar zeros

Com he explicat, se'ns demana implementar dues rutines diferents per cercar zeros. La primera rutina utilitzarà el mètode de la bisecció, i la segona el mètode de Newton.
Aquestes rutines serveixen per, si tenim una funció f, continua, definida en l'interval $[a, b]$ amb valors en $\mathbb{R}$:

$$f : [a, b] \to \mathbb{R}$$

Ens retornaran un α tal que:

$$f(\alpha) = 0$$

En el cas que $f(a)f(b) \leq 0$, el teorema de Bolzano ens assegura que aquesta α existeix, i que està a l'interval $[a, b]$.

2.1 Mètode de la bisecció

En aquest últim cas, en el que $f(a)f(b) \leq 0$, podem utilitzar un algorisme recursiu per a cercar aquesta arrel de la funció. Això solament utilitzant el teorema de Bolzano de forma recursiva.
Primer, comprovarem que ni $f(a) = 0$ ni que $f(b) = 0$, ja que si això es compleix, ja hem trobat el zero.
Ara considerem el punt mitjà de l'interval $[a, b]$, diguem-li p, ara si avaluem la nostra f en aquest punt, tenim tres possibilitats, que sigui zero, i per tant ja hem acabat, i les altres dues possibilitats són, que doni un valor positiu o negatiu.
Ara el que hem de veure, en el cas que $f(p)$ no sigui zero, el signe del producte de $f(p)$ per $f(a)$ i el producte per $f(b)$. El que ens interessa és veure el signe, ja que si $f(a)f(p) \leq 0$, això vol dir que l'arrel de la funció està a l'interval $[a, p]$, altrament a l'interval $[p, b]$, això suposant que f te una única arrel a $[a, b]$.
Un cop fet això, el que hem de fer és iterar aquest procés, això es fa canviant p per a o b, depenent del cas en el qual estem.
Aquest algorisme és el mètode de la bisecció, i és una de les dues rutines que se'ns demana implementar.

2.1.1 Programació del mètode de la bisecció

La rutina que cal proporcionar a l'Agència Espacial Catalana, en el cas del mètode de la bisecció, se'ns demana que la funció tingui un prototipus específic:

Prototipus:

```
double biseccio(double *a, double *b, double tol, double (*f)(double, void*),
                void *prm, int ixrr);
```

La funció`biseccio` rep els següents paràmetres:

- `double *a`:
  És el punter al valor inicial del límit inferior de l'interval $[a, b]$, és útil que sigui un punter, ja que els valors de a s'aniran actualitzant.

- `double *b`:
  És semblant a a, amb la diferència que en aquest cas és el valor del límit superior de $[a, b]$.

- `double tol`:
  És la tolerància requerida per a la convergència del mètode. L'algorisme farà iteracions fins que la distància de a i b sigui menor o igual a aquesta tolerància.

- `double (*f)(double, void*)`:
  Punter a la funció de la qual es vol trobar l'arrel. La funció ha d'acceptar un argument del tipus `double`, que seria un valor real de l'interval $[a, b]$, i un punter genèric `void*`, que és el paràmetre `prm`, f ha de retornar un valor de tipus `double`, que serià la imatge de la funció.

- `void *prm`:
  Punter a paràmetres addicionals que es passen a la funció `f`. Pot ser `NULL` si no es requereixen paràmetres addicionals. Es suposa que `prm` apunta a uns paràmetres arbitraris que `f` necessita.

- `int ixrr`:
  És l'índex de xarrera. Si `ixrr` és 0, la funció `biseccio()` no escriurà res mentre calcula. Si `ixrr > 0`, la funció `biseccio()` informarà per stderr de com evoluciona el mètode de la bisecció, i per tant podem seguir el que fa `biseccio()` en cada iteració.

Per tal de programar això, la idea principal ha sigut utilitzar un `while`, i fer iteracions fins que la distància de l'interval sigui més petita que la tolerància.
Per tal de simplificar això, i tindre una expressió simple per saber quan s'ha d'acabar de fer iteracions, tenim aquesta desigualtat:

$$\frac{b-a}{2^n} \leq \texttt{tol}$$

Per tant, el nombre d'iteracions que cal fer és el valor de n, tal que es compleix la desigualtat anterior.

Al començament del codi, he declarat 6 variables:

- `double p`:
  Es el valor del punt mig de l'interval $[a, b]$, aquest valor en cada iteració s'anirà actualitzant.

- `double ai,bi`:
  Són els valors inicials dels punters de a i b, aquests dos `doubles` els utilitzo per a l'expressió de la desigualtat de la tolerància, ja que de no utilitzar-los, els valors de a i b anirien canviant.

- `int n`:
  És l'índex que utilitzo per a les iteracions, comença amb el valor zero, i a cada iteració augmenta el valor en 1. Això fins que es compleixi l'expressió de la tolerància.

- `double fa,fp`:
  Aquests dos `doubles` són els valors de `f` als punts corresponents a i b. Això l'utilitzo en el codi per no haver-hi de repetir càlculs.

El codi comença declarant aquestes variables, i seguidament començo el `while`, que pararà quan la desigualtat es compleixi.
Per implementar l´índex de xarrera, a dins del `while`, he introduït un `if`, tal que si aquest índex és diferent de 0, aleshores imprimeixi els càlculs que fa `biseccio()`.
Per fer els passos del mètode, simplement amb un `if`, i calculant el signe de $f(a)f(p)$, faig els passos del mètode.
Quan el mètode acaba, faig que la funció retorni l'últim valor de p, que es va actualitzant a cada iteració. Aquest valor seria l'aproximació de l'arrel α de la funció f.

2.2 Mètode de Newton

L'altra rutina que se'ns demana implementar és una rutina per cercar zeros amb el mètode de Newton. Que és un molt bon mètode si tenim una aproximació inicial d'aquesta arrel.
El mètode de Newton és un mètode que es basa a utilitzar la recta tangent en una aproximació de l'arrel, i trobant la seva intersecció amb l'eix x, trobar una nova aproximació millor a l'anterior.

La iteració del mètode de Newton ve donada per la següent fórmula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

On x_n és la n-èsima aproximació de l'arrel, $f(x_n)$ és el valor de la funció en x_n, i $f'(x_n)$ és la derivada de la funció en x_n.

El procés continua iterativament fins que es troba una aproximació prou propera de l'arrel.

2.2.1 Programació del mètode de Newton

La rutina que hem de proporcionar ha de tenir un prototipus específic:

```
int newton(double *x, double tolf, double tolx, int maxit,
           void (*fdf)(double,double*,double*,void*), void *prm, int ixrr)
```

La funció rep els següents paràmetres:

- `double *x`:
  Punter a la primera aproximació de l'arrel. Aquest valor s'actualitzarà amb cada iteració.
- `double tolf`:
  Tolerància per al valor de la funció $f(x)$.
- `double tolx`:
  Tolerància en el valor del canvi en x, es a dir, la distància entre x_{n+1} i x_n ha de ser més petita que `tolx`.
- `int maxit`:
  Nombre màxim d'iteracions permeses.
- `void (*fdf)(double,double*,double*,void*)`:
  Punter a la funció que calcula el valor de la funció i la seva derivada $f(x)$ i $f'(x)$. El primer paràmetre és l'aproximació x, el segon és el valor de $f(x)$, el següent $f'(x)$, i l'últim `void*` és el paràmetre `*prm` que necessita `f`.
- `void *prm`:
  Té el mateix ús que en bisecció.
- `int ixrr`:
  És l'índex de xarrera, si és diferent de zero, com a bisecció, `newton()` imprimirà els seus càlculs.

Per tal de programar aquesta rutina, en comptes d'utilitzar un `while`, he utilitzat un `for`, ja que al tindre el paràmetre `maxit`, és més senzill d'aquesta forma.
Per tal de tindre el codi una mica més organitzat, he declarat un `double` a, que és el valor del quocient entre la imatge de la funció i la derivada. Això també és útil per utilitzar-ho a les toleràncies. Ja que `newton` ha d'acabar quan $|f(x_{i+1})| \leq$ `tolf` i $|x_{i+1} - x_i| \leq$ `tolx`, i amb aquest nou paràmetre, la segona condició és |`a`| $\leq$ `tolx`.
En cada iteració faig els passos del mètode de Newton i comprovo si es compleixen les toleràncies amb un `if`, si es compleixen retorno l'índex `i+1`, que és el que utilitzo en el bucle. Retorno això ja que la rutina ha de retornar el nombre d'iteracions que s'han fet.
Similarment amb la rutina anterior, utilitzo el condicional `if`, per afegir l'índex de xarrera.
Si la funció `newton` no compleix les toleràncies en el nombre d'iteracions màxim, aleshores la funció retorna -1.

3 Validació de les rutines

El següent que cal fer, és validar les dues rutines que he programat. Per tal de fer això, se'ns ha proporcionat un codi que utilitza les rutines per cercar el zero de $f(x) = e^x - 2$.

3.1 Validació de `biseccio()`

Per tal de testejar la rutina, se'ns ha proporcionat aquesta taula de valors per provar el codi:

`a`	`b`	`tol`	`ixrr`
0.5	1	0.5	1
0.5	1	0.25	1
0.5	1	0.125	1
0.5	1	0.0625	1
0.5	1	0.03125	1
0.5	1	0.015625	1

Taula 1: Valors a passar a `biseccio()`

Passant aquests valors a la rutina he obtingut els següents resultats:

`p`	iteracions
0.75	1
0.625	2
0.6875	3
0.71875	4
0.703125	5
0.6953125	6

Taula 2: Valors de `biseccio()`

I efectivament això és un comportament correcte, ja que si utilitzem que $\frac{b-a}{2^n} \leq$ `tol`, podem calcular *a priori* quantes iteracions calen fer a la rutina, i si aquestes coincideixen amb les que retorna la rutina, aleshores el seu comportament és correcte. Calculant les iteracions *a priori* tenim que:

`a-b`	`tolx`	iteracions
0.5	0.5	1
0.5	0.25	2
0.5	0.125	3
0.5	0.0625	4
0.5	0.03125	5
0.5	0.015625	6

Taula 3: Càlcul de les iteracions *a priori* de `biseccio()`

Les iteracions les he calculat solucionant $\log_2\left(\frac{0.5}{\texttt{tol}}\right) < n$.

Aquest resultat té sentit, ja que per cada iteració de `biseccio()`, l'interval es divideix per la meitat, i la tolerància també es va dividint per la meitat, per tant, en cada valor que li passem, s'ha de fer solament una iteració més. En tots aquests càlculs, he comptat com a una iteració el cas inicial.

3.2 validació de `newton()`

Ara cal comprovar la rutina de càlcul del mètode de Newton, per fer això, igual que amb el mètode la bisecció, calcularé logaritme de 2, passant a `newton()` els següents valors:

x	tolf	tolx	maxit	ixrr
2	1e-8	1e-8	10	1
2	1e-8	1	10	1
2	1	1e-8	10	1
10	1e-12	1e-8	10	1
10	1e-12	1e-8	20	1

Taula 4: Valors a passar a `newton()`

La rutina dona com a resultat els valors següents:

iteracions	x
6	0.6931471805599453
5	0.6931471814512683
6	0.6931471805599453
10 (-1)	0.8850419134774479
15	0.6931471805599453

Taula 5: Valors retornats per `newton()`

Tenint en compte que logaritme de 2 és aproximadament 0.6931471805599453. La rutina aproxima bastant bé el resultat. Per als tres primers valors podem concloure que a `newton()` és més important tindre una tolerància en x més petita, ja que així s'aproxima millor el resultat. Aquesta conclusió es pot treure del fet que l'aproximació per als valors de la segona fila és pitjor que per als de la tercera, on a la tercera la tolerància en x és l'important.

Per a les dues últimes files, es prova la rutina per una aproximació bastant dolenta. Notem que per als valors de l'última fila, la rutina no aconsegueix superar les condicions de tolerància, i per tant la funció no compleix el seu objectiu i retorna un -1. Ara bé, al duplicar el `maxit`, la rutina sí que aconsegueix superar la tolerància en un total de 15 iteracions.

Amb això es comprova que per una bona aproximació inicial de l'arrel, la rutina del mètode de Newton funciona correctament. Ja que per una bona aproximació, es compleix la velocitat com a mínim quadràtica del mètode de Newton.

4 Propagació de trajectòries mitjançant `proptraj()`

El següent que es demana en la pràctica és calcular unes trajectòries amb la funció `proptraj()`. Aquest codi ha estat proporcionat per l'Agència Espacial Catalana, i utilitza integració numèrica per calcular les trajectòries. La funció rep una posició inicial, i amb certs paràmetres més, com la constant de Jacobi, calcula la posició i velocitats de la trajectòria. Se'ns demana calcular la trajectòria amb `proptraj()`, del punt inicial x_i, que és definit per l'expressió: $x_i = x_0 + i\left(\frac{x_f - x_0}{n}\right), \quad 0 \leq i \leq n$. On a `proptraj()` se li ha de passar una constant MU, que és un paràmetre del RTBP (problema restringit dels tres cossos), un pas màxim per a la integració numèrica, el signe de la velocitat inicial, la constant de Jacobi i la quantitat de talls amb l'eix que volem fer.

Per tal de programar això, he declarat com a `double` les variables `u` i `x_i`, i he fet un bucle `for` on he aplicat per a cada `i` la funció `proptraj()`. Per a cada crida, he fet que s'imprimís per pantalla i es guardés en un fitxer de text els valors de cada x_i i u_i.

Utilitzant el programa gnuplot, podem representar la trajectòria amb uns paràmetres específics. Que gràficament és:

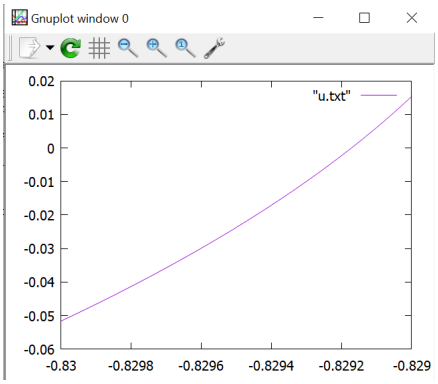

Figura 1: Trajectoria de u en funció de x

5 Implementació del mètode de la bisecció i Newton per calcular trajectòries

En aquesta secció se'ns demana utilitzar, juntament les rutines per cercar zeros de bisecció i Newton, i la rutina de càlcul de trajectòries `proptraj()`.

Se'ns demana fer una utilitat que rebi com a paràmetres `c`, `isgn`, `xa`, `xb`, `tolbis`, `tolnwt`, `maxitnwt` i `ixrr`. On `c` és la constant de Jacobi, `isgn` és el signe de la velocitat inicial, `xa`, `xb` són els límits inferior i superior de l'interval on aplicarem `bisecció()`, juntament amb `tolbis`, que és la seva tolerància. I on `tolnwt` i `maxitnwt` són respectivament, la tolerància en f de `newton()` i els màxims iterats. Finalment, `ixrr` és l'índex de xarrera, que se li passarà a les dues rutines.

Aquesta utilitat primer ha de fer `biseccio()` a un punt $x0$, i quan arribi a la tolerància que vulguem, aplicar-hi `newton()`. Per tal de fer la utilitat, abans d'aplicar les dues rutines, hem de declarar respectivament les funcions que passarem a `biseccio()` i a `newton()`. Per fer això, he hagut de declarar les respectives funcions `*f` i `*fdf` que s'utilitzen a `biseccio()` i `newton()`. Per tal de declarar-les abans, per tal de passar-los els valors de `c` i `isgn`, he hagut de declarar una `struct` que les contingues.

Seguidament, he declarat `f_bis` com a `double`, on s'implementen els paràmetres que he mencionat, i es retorna el valor de `u` amb una crida a `proptraj()`. Similarment, per al mètode de Newton, he declarat `f_new` com a `void`, on es reben els paràmetres `c` i `isgn` amb `struct`, i es crida a la funció `proptraj()`.

Amb això ja podem declarar el `double x0` i aplicar-li primer `bisecció` i després `newton()`.

El següent que se'ns demana, és representar la corresponent òrbita de Lyapunov plana a uns paràmetres concrets. Per fer això, he de propagar-la mitjançant la utilitat `rtbp_sim`, i seguidament passar-li el valor x que he calculat abans.

Gnuplot representa la seva trajectòria així:

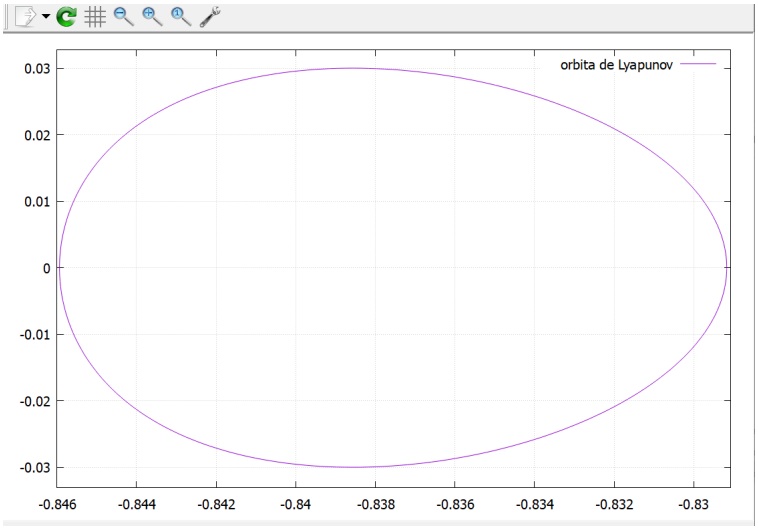

Figura 2: Trajectòria de Lyapunov