Informe de la pràctica

Arnau Escapa Farrés

13 de febrer de 2017

1 Retrat-fase i flux d'un sistema d'EDO

L'objectiu d'aquesta secció és crear utilitats que implementin el mètode de Runge-Kuta d'ordre 7 i 8 amb control de pas de Fehlberg per tal de solucionar PVI. Per verificar resultats es resolen diversos camps, alguns amb solució explicita coneguda.

1.1 `rf_pendol.c`

El programa `rf_pendol` utilitza la utilitat `rk78()` que implementa el mètode de Runge-Kuta d'ordre 7 i 8 amb control de pas de Fehlberg per representar el retrat de fase del pèndol. Per cada punt inicial que se li passa per stadard input el programa volca `np` punts de la òrbita del punt inicial. S'ha utilitzat `gnuplot` per graficar la sortida del programa obtinguda al passar a la rutina el fitxer de punts inicials `rf_inic.txt` (Figura 1)

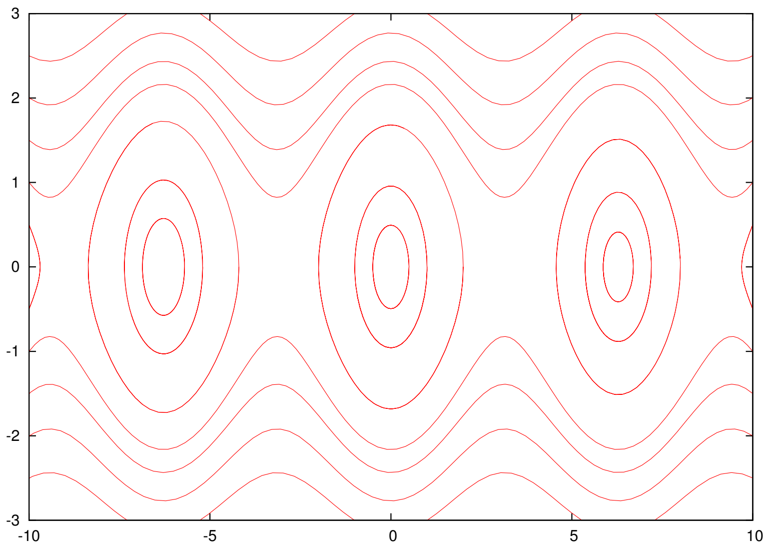

Figura 1: Retrat fase del pèndol

1.2 `flux.c`

El fitxer `flux.c` conté la funció C amb prototipus

```
int flux (double *t, double x[], double *h, double T, double pasmin,
          double pasmax, double tol, int npasmx,int n,
          int (*camp)(int n, double t, double x[], double f[], void *prm),
          void *prm);
```

que avalua $\phi(t_0+T; t_0, x)$ utilitzant el mètode RKF d'ordres 7 i 8 implementat al fitxer `rk78()`. La funció retorna 0 si acaba amb èxit i -1 si no. Pel funcionament correcte de la rutina és necessari que `h` (el pas inicial) i `T` tinguin el mateix signe. Per tal de verificar la rutina s'han integrat dos problemes de valors inicials amb solucions conegudes. Els camps s'implementen a `camp.c` i `oscilharm.c` i el seu prototipus es troben a la llibreria `camp.h`. Els fitxers `fluxTest.c` i `fluxTestbis.c` contenten les rutines que verifiquen la rutina. Es fan diverses proves per assegurar el funcionament del `flux()` per `T` positives i negatives i pels diferents camps amb solució explícita coneguda. Els resultats són satisfactoris ja que els errors són congruents amb la tolerància demanada en el flux. Per exemple, la utilitat `./fluxTestbis` soluciona el PVI del `camp.c` per diferents T, t_o i el resultat:

```
t0+T=5 x=24.99999999999578
Tol=1E-12 Error= 4.220623850414995E-12

t0+T=0.5 x=0.2500000000002143
Tol=1E-12 Error= -2.142730437526552E-13

t0+T=-0.5 x=0.2500000000002143
Tol=1E-12 Error= -2.142730437526552E-13

t0+T=-5 x=24.99999999999578
Tol=1E-12 Error= 4.220623850414995E-12
```

1.3 lorenz_int.c

El programa `lorenz_int` bolca per standard output

$$t_i \qquad \phi(t_i; 0, x_0) \qquad \text{per } i = 0, ..., n_t$$

on $\phi(t_i; 0, x_0)$ és el flux de l'atractor de Lorenz, que s'implementa a `lorenz.c` i el seu prototipus a `camp.h`.
La utilitat utilitza la funció `flux()` per resoldre el PVI. Observem que per trobar $\phi(t_i; 0, x_0)$ és suficient partir de l'anterior solució $\phi(t_{i-1}; 0, x_0)$ sense pèrdua de precisió.
S'ha utilitzat aquesta rutina per representar gràficament l'atractor de Lorenz volcant la sortida al fitxer de text `lorenz.txt` i utilitzant la representació de gràfics de Gnuplot (Figura 2) El document `lorenz_int.c` conté les comandes que s'han utilitzat per executar i graficar com a comentaris.

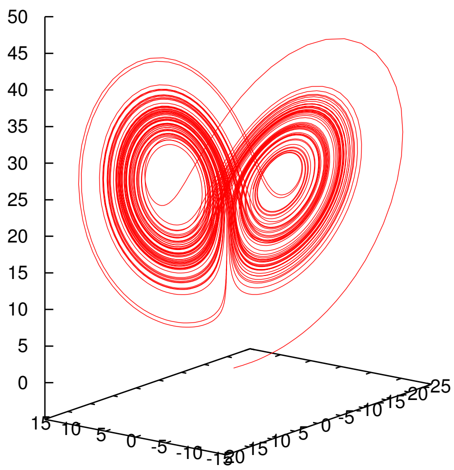

Figura 2: Atractor de Lorenz

1.4 rtbp_int.c

El programa `rtbp_int` és anàleg a `lorenz_int.c` però en aquest cas s'integra el camp del *problema restringit a 3 cossos* en coordenades sinòdiques implementat a `rtbps.c` i amb el prototipus definit a `rtbps.h`. També s'ha fet el gràfic dels resultats (Figures 3 i 4).

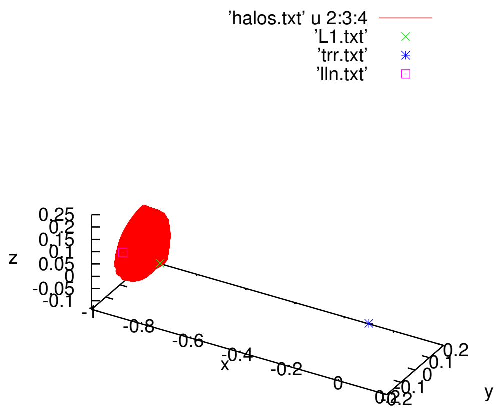

Figura 3: Problema restringit de 3 cossos

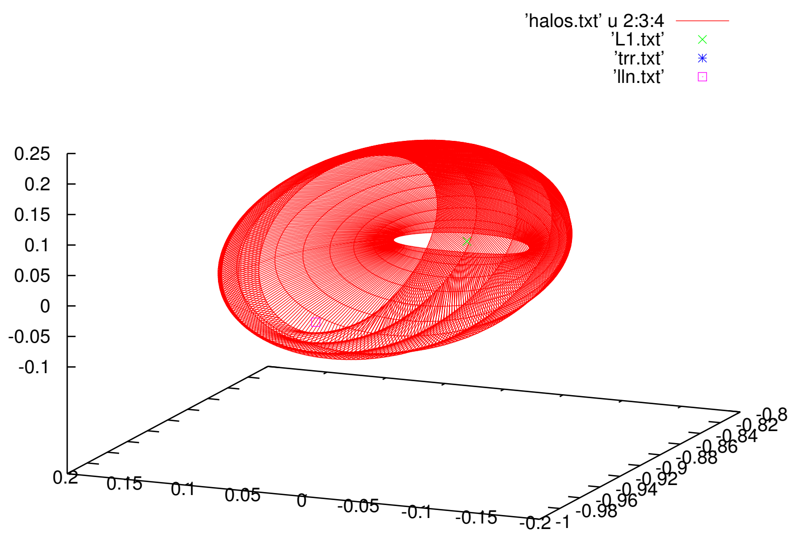

Figura 4: Problema restringit de 3 cossos

2 Les variacionals primeres i la diferencial del flux

En aquesta secció es dóna un mètode per avaluar la diferencial del flux utilitzant la utilitat `flux()`. Per avaluar numéricament la diferencial del flux respecte les condicions inicials n'hi ha prou en integrar el sistema PVI que contingui les integrals primeres al camp del qual se'n vol derivar el flux a més a més del propi camp.

2.1 variacionals.c

El programa `variacionals()` és un test per comprovar que podem avaluar la diferencial del flux satisfactoriament. Per fer-ho es comparen els resultats d'obtenir $D_x\phi(t_i; 0, x)$ per diferents mètodes. Per una banda integrel el PVI amb les variacionals i el camp amb la funcio flux de l'apartat 1. I per altre s'aproxima la diferencial a partir de la definició de derivada, concretament es fa servir la diferència finita centrada de primer ordre. El camp utilitzat per fer el test està implementat a `campvar.c` i el seu prototiupis a `camp.h`. Les diferències obtingudes són congruents amb les toleràncies utilitzades com mostren els següents resultats en que s'ha emprat `tol=1E-12` i δ`=1E-6`:

```
DxVar=0.9247686083837948 0.3495084932421975 -0.5893257067968297 0.7937558675114921
DFdif: 0.9247686084262741 0.3495084932580994 -0.5893257067346447 0.7937558675896739
Err: -4.24792423459E-11  -1.59018909151E-11 -6.21850348991E-11 -7.81817943718E-11
```

3 Mètode QR per sistemes sobredeterminats

3.1 qrres.c

El fitxer `flux.c` conté la funció C amb prototipus

```
void qrres (int m, int n, double *a, double *dr, double *b, double *x);
```

que implementa el mètode QR per resoldre sistemes d'equacions sobredeterminats. `a, x` i `b` implementen el sistema d'equacions $AX = b$. `m` i `n` són les dimensions de la matriu A que es guarda per columnes. `dr` guarda la diagonal de R. En finalitzar la rutina `x` conté la solució per mínims quadrats del sistema i `a` conté R a la part de sobre la digaonal (expcepte la seva diagonal) i les coordenades de les transformacions de Householder que s'han aplicat a part de sota la diagonal incluint aquesta. Si `b=NULL`, `x` i `b` no s'utilitzen.

3.2 qr_test.c

El programa `qr_test` és un test de la utilitat `qrres`. S'ha utilitzat la utilitat per resoldre un sistema amb solucions i descomposició QR conegudes.

3.3 qr_3.c

El programa `qr_3` utilitza la funció `qrres` per solucionar sistemes aleatoris de dimensions $n \times n$. S'han fet proves amb valors grans de n per comprovar l'eficiencia del programa.

3.4 qr_4.c

El programa `qr_4` és similar a la utilitat anterior amb la diferència que mostra per pantalla el temps que el programa ha tardat en executar-se. D'aquesta manera es pot comprovar que el temps d'execució per fer la descomposició QR és coherent amb el temps teòric. Com que el nombre d'operacions és $O(n^3)$ el temps d'execució s'ha de moultiplicar per 8 si doblem la n. S'han obtingut els següents resultats:

```
n:500 temps: 0.236 s
n:1000 temps: 1.839 s     t_1000/t_500 = 7.79    proporció teòrica: 8
n:2000 temps: 14.751 s    t_2000/t_1000 = 8.02   proporció teòrica: 8
n:3000 temps: 50.450 s    t_3000/t_500 = 27.43   proporció teòrica: 27
```

4 Càlcul de maniobres

El nostre objectiu és buscar zeros de la funció:

$$\boldsymbol{G}(\Delta \boldsymbol{v}) = \boldsymbol{\phi}_{\Delta t/2}\Bigg(\underbrace{\boldsymbol{\phi}_{\Delta t/2}\underbrace{\begin{pmatrix} \boldsymbol{r}_0 \\ \boldsymbol{v}_0 + \Delta \boldsymbol{v}_0 \end{pmatrix}}_{=:\,\boldsymbol{x}_{0.5}} + \begin{pmatrix} 0 \\ \Delta \boldsymbol{v}_1 \end{pmatrix}}_{=:\,\boldsymbol{x}_{1.5}}\Bigg) - \begin{pmatrix} \boldsymbol{r}_f \\ \boldsymbol{v}_f \end{pmatrix}.$$

S'utilitza el mètode de Newton per sistemes no lineals en diverses variables que s'implementa al fitxer `cmani.c`

4.1 `cmani_gdg`

Al fitxer `cmani.c` es troba la funció `cmani_gdg` amb prototipus

```
  int cmani_gdg (int m, double x0[], double xf[], double dt, double dv[], double g[],
double dg[],double pas0, double pasmin,  double pasmax, double tolfl,
int npasmx,int (*camp)(int n, double t, double x[], double f[], void *prm),void *prm);
```

Aquesta funció avalua G i DG, necessaris per aplicar el mètode de Newton, i en guarda els resultats a `g` i a `dg`.

4.2 `cmani`

Al fitxer `cmani.c` es troba la funció `cmani` amb prototipus

```
int cmani (int m, double x0[], double xf[], double dt, double dv[],double tol,
          int maxit,double pas0, double pasmin, double pasmax, double tolfl, int npasmx,
          int (*camp)(int n, double t, double x[], double f[], void *prm),void *prm);
```

Aquesta funció aplica el mètode de Newton per trobar els zeros de G. Cal doncs solucionar $DG(dv)X = G(dv)$ a cada iterat. Per fer-ho s'utilitzen les rutines `cmani_gdg` per avaluar G i DG i `qrres` per solucionar el sistema. Observem que per aplicar el mètode de Newton no es necessari calcular cap inversa. S'utilitza la norma 2 sobre $G(dv)$ i X com a criteri d'aturada (l'algoritme finalitza quan alguna d'elles és menor que `tol`). Les rutines `cmani_gdg` i `cmani` s'implementen al fitxer `cmani.c` i el seu prototipus es troben a la llibreria `cmani.h`.

4.3 `cmani_pendol`

Aquesta rutina és un test per a la funció `cmani` i `cmani_gdg`. S'ha resolt el problema de càlcul de maniobres per al pèndol amb resultats coneguts i s'han comparat. Els resultats han sigut satisfactoris.

4.4 `cmani_rtbp`

La utilitat `cmani_rtbp` soluciona el problema restringit de tres cossos utilitant les rutines del fitxer `cmani.c`. La utilitat respon a la crida

```
./cmani_rtbp mu tolnwt maxitnwt
```

i retorna les maniobres Δv_0 i Δv_1 per cada parell de de punts x_0 x_f que rebi per standard input. S'ha executat la rutina i comparat les solucions amb els del fitxer `cmani_rtbp.out` amb resultats satisfactoris.