UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

INTEGRACIÓ NUMÈRICA D'EDOS

Autor: Antoni Pech Alberich

Director: Dr. Miquel Bosch Gual
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 24 de gener de 2022

Abstract

The goal of this work is to study several methods for solving initial value problems in ordinary differential equations (ODEs). Particularly, we focus on the most common one-step methods, Euler, Heun and Runge-Kutta. In addition, we will prove some classic theorems about the convergence of general one-step methods and their order of error. After that, we apply numerical integration to analyse a case study which proposes a socio-economic model of the interaction among three social classes to predict the raises and falls of civilisations.

Resum

L'objectiu d'aquest treball és l'estudi d'alguns mètodes per a la resolució de problemes de valor inicial en equacions diferencials ordinàries (EDOs). En particular ens centrem en els tres mètodes d'un pas més comuns, Euler, Heun i Runge-Kutta. Demostrem teoremes clàssics sobre la convergència de mètodes d'un pas generals i sobre l'ordre del seu error. Posteriorment, estudiem una aplicació de la integració numèrica d'EDOs en un cas pràctic d'un model socio-econòmic que modelitza la interacció entre tres classes socials per a predir l'apogeu i caiguda de civilitzacions.

2020 Mathematics Subject Classification: 65L05, 91-10.

Agraïments

Vull agrair al meu tutor Miquel Bosch Gual pel seu guiatge durant l'elaboració d'aquest treball i especialment per proposar-me el cas pràctic que s'ha estudiat a la segona part del treball.

També aprofito per agrair a la meva família el seu suport que m'han donat durant aquests anys.

Índex

Introducció

Els problemes de valor inicial en equacions diferencials ordinàries (EDOs) apareixen en moltes àrees diverses, com per exemple la física, l'enginyeria, o l'economia, quan s'utilitzen tècniques de modelització matemàtica per a descriure gran varietat de fenòmens on les variables evolucionen contínuament en el temps. Només les equacions diferencials senzilles admeten solucions donades per expressions analítiques amb fórmules explícites, és a dir, que en la majoria dels casos s'ha d'usar integració numèrica per obtenir una funció aproximada a la solució.

El cas més senzill d'EDO és l'equació diferencial de la forma

$$\frac{dy}{dx} = y' = f(x, y).$$

En general hi ha infinites funcions que resolen aquest problema. Tot i així, fixant una condició inicial (x_0, y_0), és a dir, $y(x_0) = y_0$, només hi haurà una funció que resolgui l'EDO i satisfaci la condició de valor inicial:

$$y(x) = y_0 + \int_{x_0}^{x} f(t, y)dt.$$

Aquests problemes d'equacions diferencials ordinàries amb condició inicial s'anomenen problemes de Cauchy.

L'ordre d'una equació diferencial és l'ordre de la seva màxima derivada. Així, una EDO d'ordre n és una equació de la forma

$$y^{(n)} = f(x, y, y', \dots y^{(n-1)}),$$

on $y = y(x)$, $y : \mathbb{R} \to \mathbb{R}$ i, $y^{(n)}$ és la derivada n-èssima de y. A més, si f es pot escriure com una combinació lineal de les derivades de y,

$$y^{(n)} = \sum_{i=0}^{n-1} a_i(x)y^{(i)} + r(x)$$

amb $a_i(x), r(x)$ funcions contínues de x, direm que és una equació diferencial lineal. En el cas d'una EDO d'ordre n, la solució al problema de Cauchy és una funció n vegades diferenciable tal que compleix l'EDO i la condició inicial $y^{(i)}(x_0) = y_{i_0}$, $i = 0, 1, \dots, n-1$.

Un teorema central en aquest camp és [2, Theorem (7.1.1) pàg. 467], que ens assegura l'existència i unicitat en una banda $S = \{(x, y) : a \leq x \leq b, y \in \mathbb{R}\}$ de la funció solució de $y'(x) = f(x, y(x))$, sempre i quan es compleixi que la funció $f(x, y(x))$ sigui Lipschitz pels punts de S.

Més generals que els problemes de valor inicial són els problemes de valor a la frontera. No els tractarem en aquest treball peró són interessants de comentar, ja que utilitzen altres mètodes diferents dels que s'usen en els problemes de condició inicial. Un exemple n'és el mètode del Tir (*Shotting method*). Els problemes de

valor inicial normalment tenen una solució única, però els problemes de valor a la frontera poden no tenir-ne cap o no ser única.

L'objectiu d'aquest treball és estudiar mètodes d'integració numèrica d'un pas per a equacions diferencials ordinàries amb condició inicial, més concretament, els quatre mètodes més comuns, d'Euler, de Heun, d'Euler modificat i de Runge-Kutta, dels quals s'ha fet una implementació en llenguatge C. Demostrarem teoremes clàssics sobre la convergència de mètodes d'un pas generals i sobre l'ordre del seu error. Posteriorment, estudiarem una aplicació de la integració numèrica d'EDOs en un cas pràctic (*case study*), extret del llibre [3, pàg. 125]. Es tracta d'un model socio-econòmic de la interacció entre tres classes socials. En aquest context serà especialment interessant calcular les solucions corresponents als punts d'equilibri i a les òrbites periòdiques per la seva interpretació en el problema, ja que es poden utilitzar per a predir l'apogeu i la caiguda de civilitzacions. Amb aquesta finalitat, i usant integració numèrica, s'ha programat l'aplicació de Poincaré i un algorisme per a trobar-ne els punts fixos, en cas d'existir. Finalment, s'estudia l'efecte de la variació dels paràmetres de les EDOs del model en el comportament de les solucions.

Com a punt de partida d'aquest treball prenem els continguts de les assignatures d'Elements de programació, Programació científica, Mètodes numèrics I i II, Models matemàtics i sistemes dinàmics, Modelització, i Equacions diferencials.

Estructura de la memòria

Al primer capítol introduïm els problemes de valor inicial en equacions diferencials ordinàries i expliquem diferents mètodes d'integració d'un pas per resoldre numèricament el problema (Seccions 1.1 i 1.2). A la Secció 1.3 presentem un petit experiment numèric amb una EDO de la qual coneixem la seva solució analítica, que permet comparar els quatre mètodes d'un pas més comuns, d'Euler, d'Euler modificat, de Heun i de Runge-Kutta d'ordre 4. A la Secció 1.4 estudiem la convergència de mètodes d'integració d'un pas generals i a la Secció 1.5 analitzem l'expansió asimptòtica del seu error. Per tal de mantenir aquest error controlat introduïm a la Secció 1.6 dues maneres d'implementar un control de pas d'integració. A la Secció 1.7 posem de relleu la utilitat del control del pas a través d'un altre experiment numèric en el cas del mètode de Runge-Kutta d'ordre 4, contraposant la implementació amb punts equidistants amb la que incorpora un control de pas. Seguidament per tal de contextualitzar tot el que s'ha presentat al treball fins al moment, fem a la Secció 1.8 un breu comentari sobre els mètodes de múltiples passos, i a la Secció 1.9 comentem un tipus d'equacions diferencials que tenen una propietat que fan que els mètodes vistos fins al moment no siguin adequats. Aquestes equacions són anomenades *Stiff* o equacions rígides. Per acabar el primer capítol, a la Secció 1.10 fem una breu anàlisi sobre com afectarien petits canvis en la condició inicial als mètodes introduïts.

El segon capítol presenta el paradigma d'integració numèrica que s'usarà en l'estudi del cas pràctic, focalitzant en el càlcul d'òrbites periòdiques. A la Secció 2.1 recordem conceptes apresos a l'assignatura d'Equacions Diferencials i que seran

usats més endavant en el treball. A la Secció 2.2 s'explica l'ús del mètode de Runge-Kutta per a calcular l'aplicació de Poincaré.

El tercer capítol està completament dedicat a l'estudi d'un cas pràctic. A la Secció 3.1 s'analitzen les equacions del model demogràfic proposat i es fa una interpretació teòrica dels paràmetres. La Secció 3.2 presenta la resolució numèrica del problema de valor inicial usant el mètode de Runge-Kutta d'ordre 4 amb un control de pas, donant detalls particulars de la seva implementació per al sistema d'EDOs del model. A continuació, usant les nocions introduïdes al segon captítol, s'explica la implementació del programari per trobar òrbites periòdiques a la Secció 3.3. També es fa una anàlisi qualitativa en els punts d'equilibri a la Secció 3.4. Finalment, es fa un estudi per alguns valors dels paràmetres del model a la Secció 3.5.

El quart i darrer capítol presenta les conclusions de tot el treball.

La memòria conté diversos annexos on s'han adjuntat els programes usats en tots els càlculs del treball. L'Annex A presenta el programa de comparació dels quatre mètodes d'un pas. A l'Annex B es troba el programa que compara la implementació de Runge-Kutta d'ordre 4 amb punts equidistants en contraposició a la implementació del mateix mètode amb control de pas. A l'Annex C s'ha inclòs el programa que s'ha usat per calcular la solució numèrica del model del cas pràctic. Es tracta d'un mètode de Runge-Kutta d'ordre 4 amb un control de pas. L'Annex D conté el programa que troba les òrbites periòdiques del model donats uns valors per als paràmetres. L'Annex E presenta el programa que calcula el punt d'equilibri del model i la part real dels VAPs del sistema per a cada valor del paràmetre quan varia en un rang donat. Finalment, a l'Annex F es troben les instruccions de llenguatge R que s'han usat per dibuixar les gràfiques que es troben en el treball.

1 Problema de valor inicial d'EDOs

A partir d'ara treballarem amb equacions de la forma $y'(x) = f(x, y)$ per facilitar l'explicació. En tots els casos es pot generalitzar a un sistema amb n equacions de la següent manera: considerant un sistema de n EDOs de primer ordre,

$$\begin{aligned} y_1'(x) &= f_1(x, y_1(x), \ldots, y_n(x)), \\ y_2'(x) &= f_2(x, y_1(x), \ldots, y_n(x)), \\ &\vdots \\ y_n'(x) &= f_n(x, y_1(x), \ldots, y_n(x)), \end{aligned}$$

per a n funcions reals desconegudes $y_i(x),\ i = 1, \ldots, n$. Amb la notació anterior entenem que $y = (y_i)$ i $f(x, y) = (f_i(x, y_i))$ són vectors i, per tant, el valor absolut $|\cdot|$ es tradueix a la norma $\|\cdot\|$.

En aquest capítol seguirem principalment la introducció als mètodes d'integració numèrica que es presenta en [2], i ho complementarem amb [1] i [5].

1.1 Mètodes d'un pas

Considerem una EDO de problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0.$$

Primerament veurem *mètodes d'un pas* (*one-step methods*) per a trobar una aproximació a la solució. Aquests mètodes, en general estan generats per una certa funció $\Phi(x, y; h; f)$, que molts cops per simplicitat escriurem $\Phi(x, y; h)$. Començant des del valor inicial (x_0, y_0) i iterant, anem obtenint les aproximacions $\eta_i \approx y(x_i)$ de la solució exacta $y(x)$:

$$\begin{aligned} \eta_0 &= y_0, \\ \eta_{i+1} &= \eta_i + h\Phi(x_i, \eta_i; h; f), \\ x_{i+1} &= x_i + h. \end{aligned}$$

La funció solució que aproxima $y(x)$ serà denotada per $\eta(x; h)$.

El *mètode d'Euler* es basa en observar que $f(x, y)$ és el pendent de la corba solució que volem trobar $y(x)$. Aleshores per la definició de derivada, per a $h \neq 0$,

$$\frac{y(x+h) - y(x)}{h} \approx f(x, y(x)),$$

i per tant,

$$y(x+h) \approx y(x) + hf(x, y(x)).$$

Un cop escollit el pas $h \neq 0$ podem usar el mètode iteratiu anterior:

$$\begin{aligned} \eta_0 &= y_0, \\ \eta_{i+1} &= \eta_i + hf(x_i, \eta_i), \\ x_{i+1} &= x_i + h. \end{aligned}$$

Així tenim el mètode d'Euler amb $\Phi(x,y;h) = f(x,y(x))$ que, en aquest cas, és independent de h.

Siguin ara (x,y) valors arbitraris però fixats, i sigui $z(t)$ la funció solució exacta a l'EDO $z'(t) = f(t,z(t))$, $z(x) = y$. Aleshores definim

$$\Delta(x,y;h) := \begin{cases} \frac{z(x+h)-y}{h} & \text{si } h \neq 0, \\ f(x,y) & \text{si } h = 0. \end{cases}$$

Ara podem definir la funció $\tau(x,y;h)$ com la diferència entre el quocient diferència de la solució exacta per un pas h, $\Delta(x,y;h)$, i el quocient diferència de la funció aproximada usant $\Phi(x,y;h)$:

$$\tau(x,y;h) := \Delta(x,y;h) - \Phi(x,y;h).$$

Aquesta magnitut ens indica si el valor de la solució exacta en $x+h$ segueix l'equació aproximada. A la funció $\tau(x,y;h)$ se l'anomena *error local discretitzat* en el punt (x,y). Per a un mètode d'un pas és necessari que $\lim_{h\to 0} \tau(x,y;h) = 0$. Normalment parlem de *mètodes d'ordre* p si

$$\tau(x,y;h) = O(h^p)$$

per a tot $x \in [a,b]$, $y \in \mathbb{R}$ i $f \in F_p(a,b)$, on $F_p(a,b)$ és el conjunt de funcions f tal que totes les derivades parcials, fins l'ordre p inclòs, existeixen a la banda $S = \{(x,y) : a \leq x \leq b, y \in \mathbb{R}^n\}$.

Ara demostrarem que el mètode d'Euler compleix $\tau(x,y;h) = O(h)$ i, per tant, és d'ordre 1. Per fer-ho calculem l'expansió de Taylor de la solució $z(t)$ en el punt $t = x$,

$$z(x+h) = z(x) + hz'(x) + \frac{h^2}{2}z''(x) + \cdots + \frac{h^p}{p!}z^{(p)}(x+\theta h), \quad 0 < \theta < 1.$$

Com que $z(x) = y$ i $z'(t) = f(t,z(t))$ es compleix

$$z''(x) = \left.\frac{d}{dt}f(t,z(t))\right|_{t=x} = f_x(x,y) + f_y(x,y)f(x,y),$$

$$z'''(x) = f_{xx}(x,y) + 2f_{xy}(x,y)f(x,y) + f_{yy}(x,y)f(x,y)^2 + f_y(x,y)z''(x),$$

etc, i per tant,

$$\begin{aligned} \Delta(x,y;h) &= z'(x) + \frac{h}{2}z''(x) + \cdots + \frac{h^{p-1}}{p!}z^{(p)}(x+\theta h) = \\ &= f(x,y) + \frac{h}{2}(f_x(x,y) + f_y(x,y)f(x,y)) + \cdots . \end{aligned} \tag{1.1}$$

En el cas del mètode d'Euler tenim $\Phi(x,y;h) = f(x,y)$, i per tant

$$\tau(x,y;h) = \Delta(x,y;h) - \Phi(x,y;h) = \frac{h}{2}(f_x(x,y) + f_y(x,y)f(x,y)) + \cdots = O(h).$$

Per obtenir mètodes d'ordre superior podem agafar per a $\Phi(x, y; h)$ seccions de Taylor de $\Delta(x, y; h)$. Per exemple, prenent

$$\Phi(x, y; h) = f(x, y) + \frac{h}{2}\left[f_x(x, y) + f_y(x, y)f(x, y)\right]$$

donaria un mètode d'ordre 2. Tot i així, aquests mètodes no són òptims, ja que per a cada aproximació $(x_i, \eta_i) \to (x_{i+1}, \eta_{i+1})$ no només hem de calcular f, sinó també les derivades f_x, f_y, i d'altres si uséssim mètodes d'ordre més elevat. Uns mètodes d'ordre superior més senzills poden trobar-se usant

$$\Phi(x, y; h) = a_1 f(x, y) + a_2 f(x + p_1 h, y + p_2 h f(x, y)), \tag{1.2}$$

on les constants a_1, a_2, p_1, p_2 són escollides de manera que l'expansió de Taylor de $\Delta(x, y; h) - \Phi(x, y; h)$ comenci amb l'exponent més elevat de h. L'expansió de Taylor de (1.2) és

$$\Phi(x, y; h) = (a_1 + a_2)f(x, y) + a_2 h(p_1 f_x(x, y) + p_2 f_y(x, y)f(x, y)) + O(h^2).$$

Imposant $\tau(x, t; h) = \Delta(x, y; h) - \Phi(x, y; h) = O(h^2)$, s'obtenen mètodes de segon ordre. Com que sabem l'expansió de Taylor (1.1) de $\Delta(x, y; h)$ i la de $\Phi(x, y; h)$, obtenim les següents equacions:

$$a_1 + a_2 = 1, \quad a_2 p_1 = \frac{1}{2}, \quad a_2 p_2 = \frac{1}{2}.$$

Una possible solució d'aquest sistema és:

$$a_1 = \frac{1}{2}, \quad a_2 = \frac{1}{2}, \quad p_1 = 1, \quad p_2 = 1,$$

que correspon al mètode de Heun (1900):

$$\Phi(x, y; h) = \frac{1}{2}\left[f(x, y) + f(x + h, y + hf(x, y))\right].$$

Per millorar l'aproximació d'Euler, el mètode de Heun fa ús del mètode d'Euler avaluant f en dos punts per cada pas. Una altra solució que té nom és:

$$a_1 = 0, \quad a_2 = 1, \quad p_1 = \frac{1}{2}, \quad p_2 = \frac{1}{2},$$

i s'anomena mètode d'Euler modificat, proposada per Collatz (1960). Amb aquesta solució es torna a obtenir un mètode de segon ordre i necessita fer dues avaluacions d'f per cada pas:

$$\Phi(x, y; h) = f(x + \frac{1}{2}h, y + \frac{1}{2}hf(x, y)).$$

1.2 Mètode de Runge-Kutta

El mètode més conegut de Runge-Kutta, ideat per Runge (1895) i Kutta (1901), és el d'ordre 4, però n'hi ha d'ordres més grans estudiats en profunditat per Butcher

(1964). Exemples concrets d'aquests mètodes van ser proposats pel mateix Butcher, per Fehlberg (1964, 1966, 1969), Shanks (1966) i d'altres. Definim una s-fase del mètode de Runge-Kutta com el mètode d'un pas donat per la funció $\Phi(x, y; h; f)$ definida pels nombres reals $c_1, c_2, \ldots, c_s, \alpha_2, \alpha_3, \ldots, \alpha_s$, i $\beta_{i,j}$ amb $2 \leq i \leq s$ i $1 \leq j \leq i-1$ de la següent manera:

$$\Phi(x, y; h; f) = c_1 k_1 + \cdots + c_s k_s,$$

on

$$\begin{aligned} k_1 &= f(x, y), \\ k_2 &= f(x + \alpha_2 h, y + h\beta_{21} k_1), \\ k_3 &= f(x + \alpha_3 h, y + h(\beta_{31} k_1 + \beta_{32} k_2)), \\ &\vdots \\ k_s &= f(x + \alpha_s h, y + h(\beta_{s1} k_1 + \cdots + \beta_{s,s-1} k_{s-1})). \end{aligned}$$

Si considerem les equacions relatives al mètode de Runge-Kutta d'ordre 4, la seva resolució més usada és la que mostrem a continuació, extreta de [1, pàg. 356]:

$$\Phi(x, y; h) = \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right),$$

on

$$\begin{aligned} k_1 &= f(x, y), \\ k_2 &= f(x + \frac{1}{2}h, y + \frac{1}{2}hk_1), \\ k_3 &= f(x + \frac{1}{2}h, y + \frac{1}{2}hk_2), \\ k_4 &= f(x + h, y + hk_3). \end{aligned}$$

És interessant remarcar que si la funció $f(x, y)$ no depèn de y, aleshores la solució al problema de valor inicial

$$y' = f(x), \quad y(x_0) = y_0$$

és la integral $y(x) = y_0 + \int_{x_0}^{x} f(t)dt$. En aquest cas, el mètode de Heun correspon a l'aproximació de $y(x)$ mitjançant la suma de trapezoides, el mètode modificat d'Euler correspon a la regla del punt mig, i el mètode de Runge-Kutta és equivalent a la regla de Simpson.

1.3 Experiment numèric 1

En aquesta secció volem comparar el mètode d'Euler, el d'Euler modificat, el de Heun i el de Runge-Kutta d'ordre 4. Per fer-ho hem escollit la següent EDO de problema de valor inicial $y(0) = 4$:

$$\frac{dy}{dx} = (y + 1)(x + 1)\cos\left(x^2 + 2x\right). \tag{1.3}$$

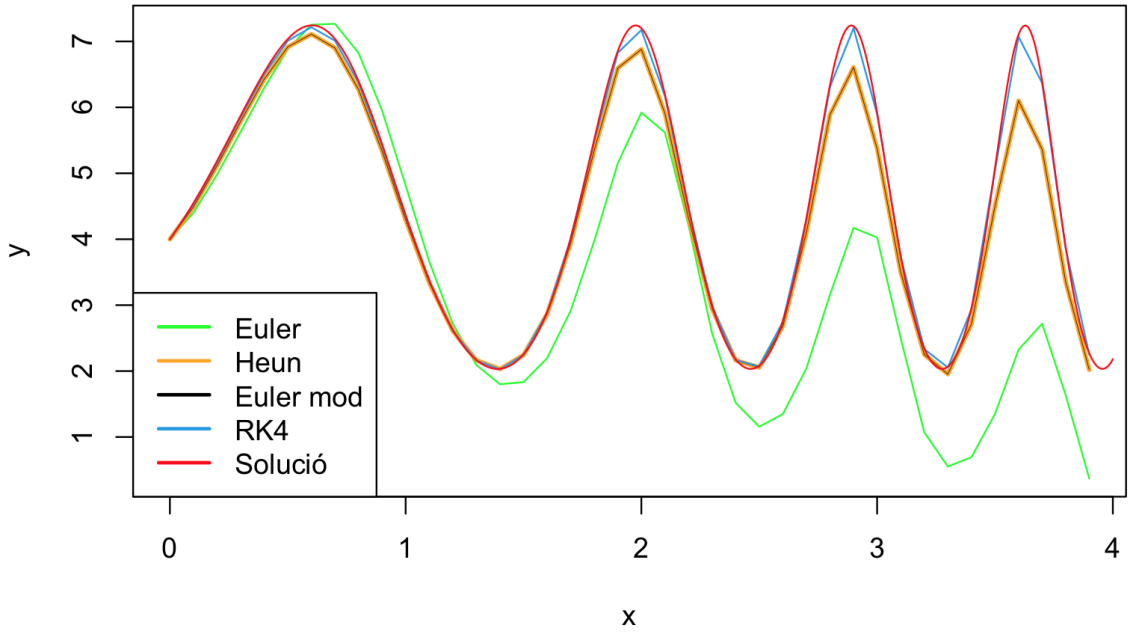

Figura 1: Comparació de diferents mètodes per integrar l'equació (1.3) i condició inicial $y(0) = 4$. La solució analítica d'aquest problema és (1.4). El programa per aquesta implementació es troba a l'Annex A.

La solució analítica d'aquesta equació diferencial és

$$y(x) = 5e^{\frac{1}{2}\sin(x^2+2x)} - 1. \tag{1.4}$$

Al programa de l'Annex A hem implementat els mètodes d'Euler, d'Euler modificat, de Heun, i de Runge-Kutta amb un pas $h = 0.1$ des de $x = 0$ fins a $x = 4$. També s'ha dibuixat la solució analítica amb $h = 0.01$. A la Figura 1 s'observa que efectivament el mètode de Runge-Kutta és el que millor s'aproxima a la corba solució. En aquest cas el mètode d'Euler, que és l'únic de primer ordre, és el que pitjor s'aproxima. També s'observa que els petits errors al principi de les aproximacions cada cop van fent-se més grans quan la x augmenta. En conclusió el millor mètode per solucionar el problema és el de Runge-Kutta, que és el que té l'ordre més alt.

1.4 Convergència dels mètodes d'un pas

En aquesta secció volem demostrar la convergència, quan h tendeix a zero, d'una aproximació $\eta(x;h)$ aconseguida per un mètode d'un pas general. Assumim que $f \in F_1(a,b)$ i denotem $y(x)$ com la funció solució exacta per al problema de valor inicial,

$$y' = f(x,y), \quad y(x_0) = y_0.$$

Sigui $\Phi(x,y;h)$ la funció del mètode d'un pas amb el qual calculem la solució aproximada $\eta(x;h)$. La funció $\eta(x;h)$ està definida per a x pertanyent al conjunt $R_h = \{x_0 + ih : i = 0,1,2,\dots\}$. El que ens interessarà tractar serà *l'error*

global de la discretització,

$$e(x;h) := \eta(x;h) - y(x)$$

per un x fixat i $h \to 0$, $h \in H_x = \{(x - x_0)/n : n = 1, 2, \dots\}$. Com que $e(x;h)$, de la mateixa manera que $\eta(x;h)$, està només definit per a valors de $h \in H_x$, aleshores hem d'estudiar

$$e(x;h_n), \quad h_n = \frac{x - x_0}{n}, \quad n \to \infty.$$

Direm que un mètode d'un pas és convergent si per a tot $x \in [a,b]$ i per a tota $f \in F_1(a,b)$:

$$\lim_{n\to\infty} e(x;h_n) = 0.$$

Veurem que per a $f \in F_p(a,b)$ els mètodes d'ordre $p > 0$ són convergents i satisfan que $e(x;h_n) = O(h_n^p)$.

Teorema 1.1. *[2, Theorem (7.2.2.3), pàg. 478] Considerem, per a $x_0 \in [a,b]$, $y_0 \in \mathbb{R}$, el problema de condició inicial*

$$y' = f(x,y), \quad y(x_0) = y_0,$$

que té la solució exacta $y(x)$. Sigui Φ una funció contínua a

$$G := \{(x,y,h) : a \le x \le b, |y - y(x)| \le \gamma, 0 \le |h| \le h_0\},\ h_0 > 0,\ \gamma > 0,$$

i siguin M i N dues constants positives tals que

$$|\Phi(x,y_1;h) - \Phi(x,y_2;h)| \le M|y_1 - y_2|,$$

per a tot $(x,y_i,h) \in G$, $i = 1,2$, i

$$|\tau(x,y(x);h)| = |\Delta(x,y(x);h) - \Phi(x,y(x);h)| \le N|h|^p, \quad p > 0,$$

per a tot $x \in [a,b]$, $|h| \le h_0$. Aleshores existeix $\overline{h}$, $0 < \overline{h} \le h_0$, tal que l'error global $e(x;h) = \eta(x;h) - y(x)$,

$$|e(x;h)| \le |h_n|^p N \frac{e^{M|x-x_0|} - 1}{M}$$

per a tot $x \in [a,b]$ i per a tot $h_n = (x - x_0)/n$, $n = 1, 2, \dots$ amb $|h_n| \le \overline{h}$. Si $\gamma = \infty$, aleshores $\overline{h} = h_0$.

Per fer la demostració d'aquest teorema necessitem el següent

Lema 1.2. *Si els nombres ζ_i satisfan*

$$|\zeta_{i+1}| \le (a + \delta)|\zeta_i| + B, \quad \delta > 0, \quad B \ge 0, \quad i = 0, 1, 2, \dots,$$

aleshores,

$$|\zeta_n| \le e^{n\delta}|\zeta_0| + \frac{e^{n\delta} - 1}{\delta} B.$$

Demostració. De la condició dels ζ_i tenim

$$\begin{aligned}
|\zeta_1| &\leq (1+\delta)|\zeta_0| + B, \\
|\zeta_2| &\leq (1+\delta)^2|\zeta_0| + B(1+\delta) + B, \\
&\vdots \\
|\zeta_n| &\leq (1+\delta)^n|\zeta_0| + B(1 + (1+\delta) + (1+\delta)^2 + \cdots + (1+\delta)^{n-1}) \\
&= (1+\delta)^n|\zeta_0| + B\frac{(1+\delta)^n - 1}{\delta} \\
&\leq e^{n\delta}|\zeta_0| + B\frac{e^{n\delta} - 1}{\delta},
\end{aligned}$$

ja que $0 < 1 + \delta \leq e^{\delta}$ per a $\delta > -1$. □

Demostració del Teorema 1.1. La funció

$$\hat{\Phi}(x, y; h) := \begin{cases} \Phi(x, y; h) & \text{si } (x, y; h) \in G, \\ \Phi(x, y(x) + \gamma; h) & \text{si } x \in [a, b], |h| \leq h_0, y \geq y(x) + \gamma, \\ \Phi(x, y(x) - \gamma; h) & \text{si } x \in [a, b], |h| \leq h_0, y \leq y(x) + \gamma, \end{cases}$$

és contínua a $\hat{G} := \{(x, y, h) : x \in [a, b], y \in \mathbb{R}, |h| \geq h_0\}$ i satisfà la condició

$$|\hat{\Phi}(x, y_1; h) - \hat{\Phi}(x, y_2; h)| \leq M|y_1 - y_2|$$

per a tot $(x, y_i, h) \in \hat{G}$, $i = 1, 2$ i, com que $\hat{\Phi}(x, y(x); h) = \Phi(x, y(x); h)$, també satisfà la condició

$$|\Delta(x, y(x); h) - \hat{\Phi}(x, y(x); h)| \leq N|h|^p, \quad x \in [a, b], |h| \leq h_0.$$

Sigui un mètode d'un pas generat per $\hat{\Phi}$ que aproxima els valors $\hat{\eta}_i := \hat{\eta}(x_i; h)$ per $y_i := y(x_i)$, $x_i := x_0 + ih$, llavors

$$\hat{\eta}_{i+1} = \hat{\eta}_i + h\hat{\Phi}(x_i, \hat{\eta}_i; h).$$

En vista de

$$y_{i+1} = y_i + h\Delta(x_i, y_i; h),$$

definim l'error $\hat{e}_i := \hat{\eta}_i - y_i$. Calculem $\hat{e}_{i+1} = \hat{\eta}_{i+1} - y_{i+1}$. Per fer-ho sumem i restem $h\hat{\Phi}(x_i, y_i; h)$, i obtenim la fórmula

$$\hat{e}_{i+1} = \hat{e}_i + h(\hat{\Phi}(x_i, \hat{\eta}_i; h) - \hat{\Phi}(x_i, y_i; h)) + h(\hat{\Phi}(x_i, y_i; h) - \Delta(x_i, y_i; h)).$$

Ara usant les desigualtats anteriors tenim

$$|\hat{\Phi}(x_i, \hat{\eta}_i; h) - \hat{\Phi}(x_i, y_i; h)| \leq M|\hat{\eta}_i - y_i| = M|\hat{e}_i|,$$

$$|\Delta(x_i, y_i; h) - \hat{\Phi}(x_i, y_i; h)| \leq N|h|^p,$$

i per la fórmula de $\hat{e}_{i+1}$ tenim l'estimació

$$|\hat{e}_{i+1}| \leq (1 + |h|M)|\hat{e}_i| + N|h|^{p+1}.$$

Ara usem el Lemma 1.2, i $\hat{e}_0 = \hat{\eta}_0 - y_0 = 0$ ens dona

$$|\hat{e}_k| \leq N|h|^p \frac{e^{k|h|M} - 1}{M}.$$

Ara sigui $x \in [a, b]$, $x \neq x_0$ fixada, $h := h_n = (x - x_0)/n, n > 0$ un enter. Aleshores amb $x_n = x_0 + nh = x$ i la desigualtat de $|\hat{e}_k|$ amb $k = n$, com que $\hat{e}(x; h_n) = \hat{e}_n$ tenim

$$|\hat{e}(x; h_n)| \leq N|h_n|^p \frac{e^{M|x-x_0|} - 1}{M}$$

per a tot $x \in [a, b]$ i amb $|h_n| \leq h_0$. Com que $|x - x_0| \leq |b - a|$ i $\gamma > 0$ aleshores existeix un $\bar{h}$, $0 < \bar{h} \leq h_0$ tal que $|\hat{e}(x; h_n)| \leq \gamma$ per a tot $x \in [a, b]$, $|h_n| \leq \bar{h}$, això és, per mètodes d'un pas generats per Φ, tenim per $|h| \leq \bar{h}$, per definició de $\hat{\Phi}$,

$$\hat{\eta}_i = \eta_i, \quad \hat{e}_i = e_i, \quad \hat{\Phi}(x_i, \hat{\eta}_i; h) = \Phi(x_i, \eta_i; h).$$

Així doncs, l'afirmació del teorema,

$$|e(x; h_n)| \leq N|h_n|^p \frac{e^{M|x-x_0|} - 1}{M},$$

es compleix per a tot $x \in [a, b]$ i tota $h_n = (x - x_0)/n, x = 1, 2, \ldots$, amb $|h_n| \leq \bar{h}$.□

En particular, a partir del teorema es demostra que els mètodes d'ordre $p > 0$ la solució exacta dels quals satisfà les condicions de Lipschitz en un entorn de la forma $|\Phi(x, y_1; h) - \Phi(x, y_2; h)| \leq M|y_1 - y_2|$ són convergents. Observem que aquesta condició es compleix, si existeix $\partial\Phi/\partial y = (\partial/\partial y)\Phi(x, y; h)$ i és contínua en el domini G de la forma descrita al teorema.

El teorema també proporciona una fita superior per a l'error. Si sabéssim M i N podríem usar-les per calcular el pas h que és requerit per calcular $y(x)$ amb un error més petit que ϵ. Malauradament, aquesta pràctica no és factible ja que les constants M i N no són fàcils de trobar perquè requereixen de càlculs de derivades superiors. En el cas del mètode d'Euler

$$N \approx \frac{1}{2}|f_x(x, y(x)) + f_y(x, y(x))f(x, y(x))|,$$

$$M \approx |\partial\Phi/\partial y| = |f_y(x, y)|.$$

En el cas del mètode de Runge-Kutta s'haurien de calcular derivades de quart ordre.

1.5 Expansió asimptòtica de l'error

En aquesta secció veurem, amb l'ajuda del teorema [2, Theorem (7.2.3.2) pàg. 481] que l'aproximació de la solució té una expansió en termes de h:

$$\eta(x; h) = y(x) + e_p(x)h^p + e_{p+1}(x)h^{p+1} + \cdots$$

per a tot $h \in \{h_n = (x - x_0)/n,\ n = 1, 2, \ldots\}$ per a certs coeficients $e_k(x)$, $k = p, p + 1, \ldots$.

Teorema 1.3. *Sigui $f(x,y) \in F_{N+1}(a,b)$ i sigui $\eta(x;h)$ l'aproximació obtinguda amb un mètode d'un pas d'ordre p, $p \leq N$ per al problema de condició inicial*

$$y' = f(x,y), \quad y(x_0) = y_0, \quad x_0 \in [a,b]$$

el qual té com a solució exacta $y(x)$. Aleshores, $\eta(x;h)$ té una expansió asimptòtica de la forma

$$\eta(x;h) = y(x) + e_p(x)h^p + e_{p+1}h^{p+1} + \cdots + e_N(x)h^N + E_{N+1}(x;h)h^{N+1}$$

amb $e_k(x_0) = 0$ per $k \geq p$, la qual és vàlida per a tota $x \in [a,b]$ i per a tota $h \in \{h_n = (x-x_0)/n,\ n = 1,2\dots\}$. Les funcions $e_i(x)$ són diferenciables i independents respecte h i el residu $E_{N+1}(x;h)$ està acotat per x i per a tota $h = h_n = (x-x_0)/n$, $n = 1,2,\dots,\sup_n |E_{N+1}(x;h_n)| < \infty$.

Demostració. Suposem que un mètode d'un pas està donat per la funció $\Phi(x,y;h)$. Com que el mètode té ordre p i $f \in F_{N+1}$,

$$y(x+h) - y(x) - h\Phi(x,y;h) = d_{p+1}(x)h^{p+1} + \cdots + d_{N+1}(x)h^{N+1} + O(h^{N+2}).$$

Primer, només usem

$$y(x+h) - y(x) - h\Phi(x,y;h) = d_{p+1}(x)h^{p+1} + O(h^{p+2})$$

i demostrem que hi ha una funció diferenciable $e_p(x)$ tal que

$$\eta(x;h) - y(x) = e_p(x)h^p + O(h^{p+1}),\ e_p(x_0) = 0.$$

Amb aquest objectiu, considerem la funció

$$\hat{\eta}(x;h) := \eta(x;h) - e_p(x)h^p,$$

on l'eleció de e_p encara està per veure. És fàcil veure que $\hat{\eta}$ pot ser considerada com a resultat d'un altre mètode d'un pas,

$$\hat{\eta}(x+h;h) = \hat{\eta}(x;h) + h\hat{\Phi}(x, \hat{\Phi}(x,\hat{\eta}(x;h);h)),$$

si $\hat{\Phi}$ està definida per

$$\hat{\Phi}(x,y;h) := \Phi(x, y + e_p(x)h^p; h) - (e_p(x+h) - e_p(x))h^{p-1}.$$

Per l'expansió de Taylor respecte h, trobem

$$y(x+h) - y(x) - h\hat{\Phi}(x,y;h) = (d_{p+1}(x) - f_y(x,y(x))e_p(x) - e_p'(x))h^{p+1} + O(h^{p+2}).$$

Per tant, mètodes d'un pas pertanyents a $\hat{\Phi}$ tenen ordre $p+1$ si e_p es pren com a solució del problema de valor inicial

$$e_p'(x) = d_{p+1}(x) - f_y(x,y(x))e_p(x), \quad e_p(x_0) = 0.$$

Amb aquesta tria d'e_p i usant el Teorema 1.1 aplicat a $\hat{\Phi}$ demostra que

$$\hat{\eta}(x;h) - y(x) = \eta(x;h) - y(x) - e_p(x)h^p = O(h^{p+1}).$$

Podem argumentar anàlogament per $\hat{\Phi}$, i s'acaba la prova del teorema. □

El comportament asimptòtic és important perquè ens permet calcular una estimació de l'error $e(x;h)$. Suposem que el mètode d'ordre p té una expansió asimptòtica de la forma

$$e(x;h) = \eta(x;h) - y(x) = e_p(x)h^p + O(h^{p+1}).$$

Un cop hem trobat l'aproximació $\eta(x;h)$ amb un pas de h, calculem altre cop amb la mateixa x, però ara fent el pas més petit (per exemple $h/2$), obtenint l'aproximació $\eta(x;h/2)$. Per un valor de h prou petit i $e_p(x) \neq 0$ tenim

$$\eta(x;h) - y(x) = e_p(x)h^p,$$

$$\eta(x;\frac{h}{2}) - y(x) = e_p(x)\left(\frac{h}{2}\right)^p.$$

Ara restem la segona equació a la primera i obtenim

$$\eta(x;h) - \eta(x;\frac{h}{2}) = e_p(x)\left(\frac{h}{2}\right)^p(2^p - 1).$$

Aïllem $e_p(x)\,(h/2)^p$ i ho substituïm a la segona equació:

$$\eta(x;\frac{h}{2}) - y(x) = \frac{\eta(x;h) - \eta(x;\frac{h}{2})}{2^p - 1}. \tag{1.5}$$

En el cas del mètode de Runge-Kutta podem substituir la $p = 4$ i obtenim

$$\eta(x;\frac{h}{2}) - y(x) = \frac{\eta(x;h) - \eta(x;\frac{h}{2})}{15}.$$

1.6 Control de pas en l'implementació dels mètodes

A la pràctica, en els problemes de valor inicial interessa conèixer la funció $y(x)$ per algun valor $x \neq x_0$. Per aconseguir calcular $y(x)$ és temptador usar un pas $\overline{h} = x - x_0$, però com més gran sigui el pas $\overline{h}$ ens crearà un error més elevat, i per tant la tria de $\overline{h}$ seria inadequada. Per tant, normalment separem l'interval $[x_0, x]$ en punts intermitjos x_i, $i = 1, \ldots, k-1$, $x_0 \leq x_1 \leq \cdots \leq x_k = x$. Començant per $x_0, y_0 = y(x_0)$ es calculen les aproximacions usant el mètode triat amb el pas $h_i = x_{i+1} - x_i$:

$$\overline{y}(x_{i+1}) = \overline{y}(x_i) + h_i\Phi(x_i, \overline{y}(x_i); h_i), \quad x_{i+1} = x_i + h,$$

on $\overline{y}(x_i)$ és l'aproximació de $y(x_i)$. Aleshores és normal preguntar-nos de quina manera hem de triar el pas h_i. Volem aconseguir fer el pas h_i el més gran possible,

però a la vegada aconseguir que l'error es mantingui petit. Per tant, donats (x_0, y_0) determinem el pas h el més gran possible tal que l'error $e(x_0 + h; h)$ després d'una iteració estigui per sota d'una tolerància ϵ. Aquesta tolerància no hauria de ser escollida més petita que K eps, on eps és la precisió relativa de l'ordinador i K és una cota de la solució $y(x)$,

$$K \approx \max\{|y(x)| : x \in [x_0, x_0 + h]\}.$$

Per un mètode d'ordre p tenim l'aproximació de l'error:

$$e(x; h) = e_p(x)h^p.$$

A més, en el primer cas $x = x_0$ es compleix $e_p(x_0) = 0$, per tant, $e_p(x) = (x - x_0)e'_p(x_0)$. Així doncs, l'error compleix la tolerància $|e(x_0 + h; h)| = \epsilon$ si

$$\epsilon = |e_p(x_0 + h)h^p| = |h^{p+1}e'_p(x_0)|.$$

Si coneguéssim el valor de $e'_p(x_0)$, seríem capaços de saber quin és el valor de h. De fet, podem calcular una aproximació usant el pas H per calcular $\eta(x_0 + H; H)$ i $\eta(x_0 + H; H/2)$ i amb l'equació (1.5) tenim

$$e(x_0 + H, \frac{H}{2}) = \frac{\eta(x_0 + H; H) - \eta(x_0 + H; H/2)}{2^p - 1}.$$

Per l'altra banda tenim,

$$e(x_0 + H, \frac{H}{2}) = e_p(x_0 + H)\left(\frac{H}{2}\right)^p = e'_p(x_0)H\left(\frac{H}{2}\right)^p.$$

Finalment usant les dues equacions anteriors i aïllant $e'_p(x_0)$ obtenim la fórmula

$$e'_p(x_0) = \frac{1}{H^{p+1}}\frac{2^p}{2^p - 1}\left[\eta(x_0 + H; H) - \eta(x_0 + H; \frac{H}{2})\right].$$

Usant la igualtat anterior i $\epsilon = |h^{p+1}e'_p(x_0)|$ obtenim

$$\frac{H}{h} = \left(\frac{2^p}{2^p - 1}\frac{|\eta(x_0 + H; H) - \eta(x_0 + H; H/2)|}{\epsilon}\right)^{\frac{1}{p+1}}. \tag{1.6}$$

El procediment per usar-la en un algoritme seria: escollim un pas H; calculem $\eta(x_0+H; H)$, $\eta(x_0+H; H/2)$ i de la fórmula (1.6) obtenim h. Si $H/h \gg 2$, aleshores l'error de $e(x_0 + H; H/2)$ és molt més gran que la tolerància ϵ escollida al principi. Per tant, el que farem és reduir el pas H agafant com a nou $H = 2h$. Tornem a calcular $\eta(x_0 + H; H)$, $\eta(x_0 + H; H/2)$ i de (1.6) obtenim un nou h. Repetim fins que $|H/h| \leq 2$. Quan sigui el cas, aceptarem $\eta(x_0 + H; H/2)$ com una aproximació de $y(x_0 + H)$ i procedirem al següent pas d'integració canviant x_0, y_0 i H pels nous valors inicials $x_0 + H, \eta(x_0 + H; H/2)$ i $2h$.

Amb aquest mètode de control de pas necessitem calcular dues aproximacions $\eta(x_0+H; H)$ i $\eta(x_0+H; H/2)$ de la solució exacta $y(x_0+H)$. Hi ha d'altres maneres

d'usar mètodes de control de pas. La idea és calcular dues aproximacions amb el mateix pas H, però amb dos mètodes Φ_I, Φ_{II} de Runge-Kutta, un d'ordre p i l'altre d'ordre $p+1$. Aquests mètodes s'anomenen de Runge-Kutta-Fehlberg i tenen la següent forma:

$$\Phi_I(x,y;h) = \sum_{k=0}^{p} c_k f_k(x,y;h),$$

$$\Phi_{II}(x,y;h) = \sum_{k=0}^{p+1} \widehat{c}_k f_k(x,y;h),$$

$$f_k := f_k(x,y;h) = f\left(x+\alpha_k h, y+h\sum_{l=0}^{k-1}\beta_{kl} f_l\right), \quad k=0,1,\ldots,p+1.$$

Fixem-nos que els f_k s'usen als dos mètodes, per tant els mètodes Φ_I i Φ_{II} és diuen *embedded methods*. Les constants $\alpha_k, \beta_{kl}, c_k, \widehat{c}_k$ són determinades de tal manera que els mètodes tinguin ordre p i $p+1$,

$$\begin{aligned} \Delta(x,y(x);h) - \Phi_I(x,y(x);h) &= O(h^p), \\ \Delta(x,y(x);h) - \Phi_{II}(x,y(x);h) &= O(h^{p+1}), \\ \alpha_k = \sum_{j=0}^{k-1}\beta_{kj}, & \qquad k=0,1,\ldots,p+1. \end{aligned}$$

Resoldre aquestes equacions ens porta a complicats sistemes no lineals. A més, a l'hora de resoldre-les, els podem imposar que f_{p+1} de l'iteració i correspongui a la nova f_0 de l'iteració $i+1$, d'aquesta manera convertint-ho més econòmic. Un dels mètodes més usats és per $p=2,3$, estudiats per Fehlberg (1964,1966,1969) obtenint els resultats de Figura 2.

k	α_k	β_{k0}	β_{k1}	β_{k2}	c_k	$\hat{c}_k$
0	0	—	—	—	$\frac{214}{891}$	$\frac{533}{2106}$
1	$\frac{1}{4}$	$\frac{1}{4}$	—	—	$\frac{1}{33}$	0
2	$\frac{27}{40}$	$-\frac{189}{800}$	$\frac{729}{800}$	—	$\frac{650}{891}$	$\frac{800}{1053}$
3	1	$\frac{214}{891}$	$\frac{1}{33}$	$\frac{650}{891}$	—	$-\frac{1}{78}$

Figura 2: Valors dels coeficients per un control de pas amb mètodes de Runge-Kutta d'ordre 2 i 3 extreta de [2, pàg. 490].

Tot i així, s'usen més el mètodes trobats per Dormand i Prince (1980), que són dos mètodes *embedded* de Runge-Kutta d'ordres 4 i 5. Els valors de les constants les veiem a la figura 3. A més han estat escollits amb l'objectiu que l'error de Ψ_{II} sigui mínim.

Siguin $\widehat{y}_{i+1} = \overline{y}_i + h\Phi_I(x_i,\overline{y}_i;h)$ i $\overline{y}_{i+1} = \overline{y}_i + h\Phi_{II}(x_i,\overline{y}_i;h)$ les aproximacions de cada mètode respecticament. Podem considerar la resta entre les dues aproximacions $\overline{y}_{i+1} - \widehat{y}_{i+1} = h(\Phi_I(x_i,\overline{y}_i;h) - \Phi_{II}(x_i,\overline{y}_i;h))$. També sabem

k	α_k	β_{k0}	β_{k1}	β_{k2}	β_{k3}	β_{k4}	β_{k5}	c_k	$\hat{c}_k$
0	0							$\frac{35}{384}$	$\frac{5179}{57600}$
1	$\frac{1}{5}$	$\frac{1}{5}$						0	0
2	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
3	$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
4	$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
5	1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
6	1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$

Figura 3: Valors dels coeficients per un control de pas amb mètodes de Runge-Kutta d'ordre 4 i 5 extreta de [2, pàg. 490].

$\Phi_I(x_i, \overline{y}_i; h) - \Delta(x_i, \overline{y}_i; h) = h^p C_I(x)$ i $\Phi_{II}(x_i, \overline{y}_i; h) - \Delta(x_i, \overline{y}_i; h) = h^{p+1} C_{II}(x)$. Per una $|h|$ prou petita tenim:

$$\overline{y}_{i+1} - \widehat{y}_{i+1} = h^{p+1} C_I(x_i).$$

Suposem ara que la integració des de x_i a x_{i+1} ha estat satisfactòria amb una tolerància $\epsilon > 0$. Aleshores negligint els termes d'ordre superior tenim:

$$|h^{p+1} C_I(x_i)| \leq \epsilon.$$

Perquè la següent iteració sigui també satisfactòria volem que el nou valor del pas $h_{nou} = x_{i+2} - x_{i+1}$ compleixi

$$|C_I(x_{i+1}) h_{nou}^{p+1}| \leq \epsilon.$$

Ara mirant fins errors de primer ordre tenim $C_I(x_i) = C_I(x_{i+1})$. Però per $C_I(x_i)$ tenim una aproximació $|C_I(x_{i+1})| = \frac{|\overline{y}_{i+1} - \widehat{y}_{i+1}|}{|h|^{p+1}}$. Així obtenim la següent desigualtat

$$|\overline{y}_{i+1} - \widehat{y}_{i+1}| \left| \frac{h_{nou}}{h} \right|^{p+1} \leq \epsilon,$$

que podem usar per obtenir h_{nou}:

$$h_{nou} = h \left| \frac{\epsilon}{\overline{y}_{i+1} - \widehat{y}_{i+1}} \right|^{1/(p+1)}.$$

El control de pas seguint aquesta fórmula és molt barat en cost de computació, ja que calcular $\overline{y}_{i+1}$ i $\widehat{y}_{i+1}$ requereix de $p+1$ avaluacions de $f(x, y)$. Aquesta manera és més òptima que l'anterior. Cal remarcar que alguns autors, basats en experiments numèrics, recomanen fer la variació

$$h_{nou} = \alpha h \left| \frac{\epsilon h}{\overline{y}_{i+1} - \widehat{y}_{i+1}} \right|^{1/p},$$

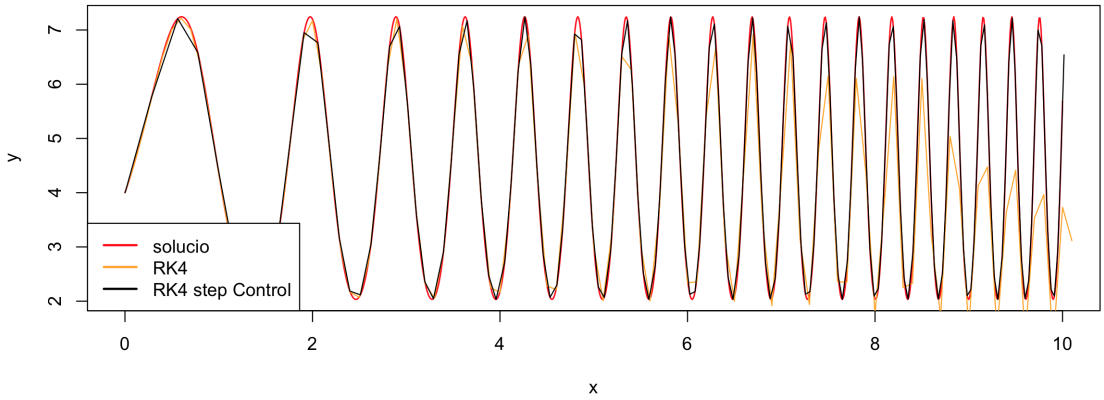

Figura 4: Comparació de la implementació de Runge-Kutta d'ordre 4 amb punts equidistants contra un control de pas usant la fórmula (1.6). L'equació diferencial és (1.3) amb condició inicial $y(0) = 4$. El programa es troba a l'Annex B.

on α és un factor d'ajustament $\alpha \approx 0.9$. Una cosa a tenir en compte és que, si es vol dibuixar una gràfica de la corba solució aproximada, es poden tenir problemes, perquè el mètode de Runge-Kutta haurà trobat punts massa distants en alguns llocs, si hem usat el control de pas. Una solució per aquest problema és construir des dels punts $(x_i, y_i), i \geq 0$ una aproximació per a la solució a tots els punts intermedis $\widehat{x}(\xi) := x_i + h\xi, 0 \leq \xi, \leq 1$ i $h = x_{i+1} - x_i$. Això s'anomena *mètodes continus de Runge-Kutta* (*continuous Runge-Kutta methods*).

1.7 Experiment numèric 2

En aquesta secció volem comparar el control de pas en contraposició amb punts equidistants en el cas del mètode de Runge-Kutta d'ordre 4. Per fer-ho hem usat l'EDO de l'Experiment numèric 1.3. Al programa B hem implementat el mètode de Runge-Kutta d'ordre 4 des de $x = 0$ fins a $x = 10$ amb punts equidistants usant $h = 0.1$ i també un control de pas usant la fórmula (1.6) amb $\epsilon = 10^{-5}$. A la Figura 4 s'observa que efectivament el control de pas es manté aprop de la solució mentre que el mateix Runge-Kutta amb punts equidistants deixa d'aproximar la solució a partir d'un lloc. En conclusió una implementació de control de pas assegura mantenir un error petit.

1.8 Mètodes de múltiples passos

En els mètodes de múltiples passos per trobar la solució d'un problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0,$$

es calcula l'aproximació η_{j+r} de $y(x_{j+r})$ amb $r \geq 2$ a partir de les aproximacions calculades anteriorment η_k de $y(x_k)$, on $k = j, j+1, \ldots, j+r-1$ en punts equidistants $x_k = x_0 + kh$. L'algorisme és que per a cada $j = 0, 1, 2, \ldots$ s'usen $\eta_j, \eta_{j+1}, \ldots, \eta_{j+r-1}$ per obtenir η_{j+r}. Per començar a fer iteracions amb aquest mètode és necessari tenir

prèviament calculats els r primers valors $\eta_0, \eta_1, \ldots \eta_{r-1}$. Per calcular-los podem usar els mètodes vists abans d'un pas.

1.9 Inestabilitat dels mètodes

Tots els mètodes tractats ens els capítols anteriors són mètodes explícits. Aquests mètodes no són adequats per resoldre els sistemes d'equacions *Stiff* o *equacions rígides*. Aquests sistemes tenen la propietat que les solucions són funcions abruptes amb canvis sobtats de velocitats. Encara que aquest canvi brusc sigui en un període de temps curt, ja pot generar una gran diferència al temps de la solució. Aquests comportaments es troben en sistemes d'equacions $y'(x) = f(x, y), y \in \mathbb{R}^n$ tals que la matriu $n \times n$ de $f_y(x, y)$ té els valors propis amb part real negativa, molt gran en valor absolut.

Un exemple d'aquests tipus d'equacions [2, pàg. 525] són les reaccions cinètiques químiques, ja que suposen una transformació d'un tipus de concentració en un altre en un període de temps molt curt. Suposem que tenim el sistema d'equacions

$$\begin{aligned} y_1'(x) &= \tfrac{\lambda_1+\lambda_2}{2} y_1 + \tfrac{\lambda_1-\lambda_2}{2} y_2\,, \\ y_2'(x) &= \tfrac{\lambda_1-\lambda_2}{2} y_1 + \tfrac{\lambda_1+\lambda_2}{2} y_2\,, \end{aligned}$$

amb λ_1, λ_2 constants negatives. La solució general és

$$\begin{aligned} y_1(x) &= C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x}, \\ y_2(x) &= C_1 e^{\lambda_1 x} - C_2 e^{\lambda_2 x}, \end{aligned}$$

on $x \geq 0$ i C_1, C_2 són constants d'integració. Si s'integra pel mètode d'Euler s'obté:

$$\begin{aligned} \eta_{1i}(x) &= C_1(1 + h\lambda_1)^i + C_2(1 + h\lambda_2)^i, \\ \eta_{2i}(x) &= C_1(1 + h\lambda_1)^i - C_2(1 + h\lambda_2)^i. \end{aligned}$$

Aquesta solució convergeix a 0 amb $i \to \infty$ només si la h és prou petita per tenir $|1 + \lambda_1 h| < 1$ i $|1 + \lambda_2 h| < 1$.

Ara suposem que $|\lambda_2|$ és molt més gran que $|\lambda_1|$. Com que $\lambda_2 < 0$, el terme $e^{\lambda_2 x}$ és negligible en comparació amb $e^{\lambda_1 x}$. Malauradament, això no és cert per a la integració numèrica del mètode d'Euler, perquè necessitem h tant petita que compleixi $h < 2/|\lambda_2|$. Per exemple, suposem $\lambda_1 = -1$ i $\lambda_2 = -1000$, aleshores $h = 0.002$. Per tant, encara que e^{-1000x} és pràcticament negligible, el factor 1000 limita la possibilitat del nombre h.

Aquest problema passa amb tots els mètodes explicats abans. Tot i així, se'ls poden fer modificacions per convertir-los en mètodes implícits. En la bibliografia [2, pàg. 527-530] s'explica els canvis als mètodes d'un pas proposats per Kaps i Rentrop (1979) per resoldre equacions *Stiff*.

1.10 Anàlisi de la sensibilitat a la condició inicial

Les EDOs de problema de valor inicial normalment depenen de paràmetres reals $p = (p_1, \ldots, p_{n_p})$,

$$y'(x; p) = f(x, y(x; p), p), \quad y(x_0; p) = y_0(p), \quad x_0 < x < x_1.$$

Segurament petits canvis en aquests paràmetres produeixen un gran canvi a la solució $y(x;p)$. Per això és important estudiar detingudament la seva dependència. Per fer-ho es necessita calcular la solució de la trajectòria $y(x;p)$ i la seva sensitivitat a través de la matriu

$$\left.\frac{\partial y(x;p)}{\partial p}\right|_p,$$

que es pot aproximar per

$$\left.\frac{\partial y(x;p)}{\partial p_i}\right|_p \approx \frac{y(x;p+\Delta p_i e_i)-y(x;p)}{\Delta p_i}, \quad i=1,\ldots,n_p.$$

Utilitzem e_i per referir-nos a la i-èssima coordenada del vector de $\mathbb{R}^{n_p}$. El cost computacional és acceptable i fàcil. Si volem aconseguir una major precisió és millor canviar el mètode de calcular la sensibilitat.

Un altre enfocament és trobar una solució a les equacions de sensibilitat:

$$\frac{\partial y'(x;p)}{\partial p} = \frac{\partial f(x,y;p)}{\partial y}\cdot\frac{\partial y(x;p)}{\partial p} + \frac{\partial f(x,y;p)}{\partial p}, \quad x_0 < x < x_1.$$

$$\frac{\partial y(x_0;p)}{\partial p} = \frac{\partial y_0(p)}{\partial p}.$$

Aquestes equacions formen la funció

$$z(x) := \partial y(x;p)/\partial p,$$

que és una matriu $n \times n_p$ si $y \in \mathbb{R}^n$, que soluciona les equacions de sensibilitat.

2 Aplicació al càlcul d'òrbites periòdiques

Aquest capítol fa un breu recordatori d'alguns conceptes de sistemes dinàmics que s'usaran posteriorment en l'estudi qualitatiu del sistema determinat per l'EDO del cas pràctic. També explicarem la implementació del mètode de Runge-Kutta per calcular l'aplicació de Poincaré, i les òrbites periòdiques, en cas d'existir.

2.1 Recordatori de conceptes clau de sistemes dinàmics

Començarem aquesta secció recordant alguns conceptes de l'assignatura d'Equacions Diferencials.

Definició 2.1. *Un sistema dinàmic és una tripleta (G, U, Ψ) on G és un grup commutatiu ($G = \{\mathbb{R}, \mathbb{Z}\}$) i Ψ és una aplicació*

$$\Psi : G \times U \to U$$

complint les condicions següents ($x \in U, s, t \in \mathbb{R}$):

1. *Ψ és contínua (respecte la variable x),*
2. *$\Psi(0, x) = x$,*
3. *$\Psi(t + s, x) = \Psi(t, \Psi(s, x))$.*

Definició 2.2. *Considerem el sistema dinàmic $(G, U \subset \mathbb{R}^n, \Psi)$. Sigui $x \in U$. Anomenem òrbita de x, i denotem per $\gamma(x)$ el conjunt*

$$\gamma(x) := \{\Psi(t, x), t \in G\}.$$

Teorema 2.3. *Sigui $\gamma(x)$ una òrbita del sistema dinàmic $(\mathbb{R}, U \subset \mathbb{R}^n, \Psi)$. Aleshores $\gamma(x)$ és o bé un punt, o bé homeomorfa a un cercle, o bé és la imatge injectiva de $\mathbb{R}$.*

Definició 2.4. *Si $\gamma(x) \equiv x$ diem que x és un punt singular, crític o d'equilibri. Altrament direm que és regular. Si $\gamma(x) \cong \mathbb{S}^1$ direm que l'òrbita de x és periòdica de període $T > 0$.*

Definició 2.5. *Sigui $U \subset \mathbb{R}^n$ obert i sigui $f : U \to \mathbb{R}^n$ de classe C^1. Llavors f defineix un camp vectorial a U.*

Definició 2.6. *Sigui $U \subset \mathbb{R}^n$ i sigui $f : U \to \mathbb{R}^n$ un camp vectorial de classe $C^r(U)$. Sigui $p \in U$ tal que $f(p) = 0$. Diem que p és un punt hiperbòlic si $Re(\lambda) \neq 0$ per a tot λ valor propi de la matriu $A := (Df)_p$.*

Definició 2.7. *Sigui $p \in U$ punt hiperbòlic d'un camp vectorial $f : U \to \mathbb{R}^n$. Sigui $A = (Df)_p$ i siguin $\lambda_1, \lambda_2, \ldots, \lambda_n, \lambda_j \in \mathbb{C}$ els valors propis de A. Si $Re(\lambda_j) < 0$ $\forall j \in \{1, \ldots, n\}$ direm que p és un atractor (local). Si $Re(\lambda_j) > 0$, $\forall j \in \{1, \ldots, n\}$ direm que p és un repulsor (local). Finalment, si existeix $0 < m < n$ tal que*

$$Re(\lambda_1) \leq \cdots \leq Re(\lambda_m) < 0 < Re(\lambda_{m+1}) \leq \cdots \leq Re(\lambda_n)$$

direm que p és un punt de sella (local).

Definició 2.8. *Sigui* $\gamma := \{\phi(t,p), 0 \leq t \leq T\}$ *una òrbita periòdica de període* T *d'un camp vectorial* $f : U \to \mathbb{R}^n$ *de classe* C^1. *Sigui* Σ *una secció transversal al camp* f *en el punt* p. *Definim l'**aplicació de Poincaré*** $\tau : \Sigma_0 \subset \Sigma \to \Sigma$ *com*

$$\tau(y) := \phi(s_0, y),\ y \in \Sigma_0,$$

on $t = s_0 > 0$ *determina el temps del primer tall de la solució* $\phi(t,y)$ *amb la secció* Σ. *Si* Σ_0 *és prou petit,* τ *és contínua respecte les condicions inicials. A més, compleix* $\tau(p) = \phi(s_0, p) = p$, *és a dir, és un punt fix.*

Definició 2.9. *Sigui* $U \subset \mathbb{R}^n$ *i sigui* $f : U \to \mathbb{R}^n$ *camp vectorial de classe* $C^1(U)$. *Sigui* $\phi(t) = \phi(t,p)$ *la corba integral de* f *passant pel punt* $p \in U$. *Definim el conjunt* ω*-límit de* p, *i el denotem* $\omega(p)$ *com*

$$\omega(p) = \{q \in U : \exists \{t_n\} \to \infty \ i\ \phi(t_n) \to q, n \to \infty\}.$$

2.2 Aplicació de Runge-Kutta al càlcul de l'aplicació de Poincaré

Una vegada introduïts els fonaments teòrics sobre òrbites periòdiques, explicarem la metodologia que seguirem per al càlcul d'aquestes òrbites. Acabem de veure que trobar una òrbita periòdica és equivalent a trobar un punt fix per l'aplicació de Poincaré. Per tant, hi haurà dos aspectes a tenir en compte en resoldre numèricament aquest problema:

- S'hauran d'integrar les equacions diferencials per poder calcular les òrbites o trajectòries, que seran la base per a determinar l'aplicació de Poincaré com un problema de valor inicial d'EDO.
- S'haurà de trobar un punt fix de l'aplicació de Poincaré, en cas d'existir.

Per abordar el primer punt, com a integrador numèric, hem implementat el mètode Runge-Kutta d'ordre 4, seguint [5, pàg. 215]. Suposem que tenim l'aproximació η_i, per calcular la següent aproximació η_{i+1} farem el següent càlcul (per una interpretació geomètrica veure la Figura 5):

$$\begin{aligned}
k_1 &= hf(x_i, \eta_i), \\
k_2 &= hf(x_i + \frac{h}{2}, \eta_i + \frac{1}{2}k_1), \\
k_3 &= hf(x_i + \frac{h}{2}, \eta_i + \frac{1}{2}k_2), \\
k_4 &= hf(x_i + h, \eta_i + k_3), \\
\eta_{i+1} &= \eta_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).
\end{aligned}$$

Aquesta elecció està justificada per la seva combinació de simplicitat i eficiència, i perquè és el més comunment usat. De fet està inclòs en molts manipuladors

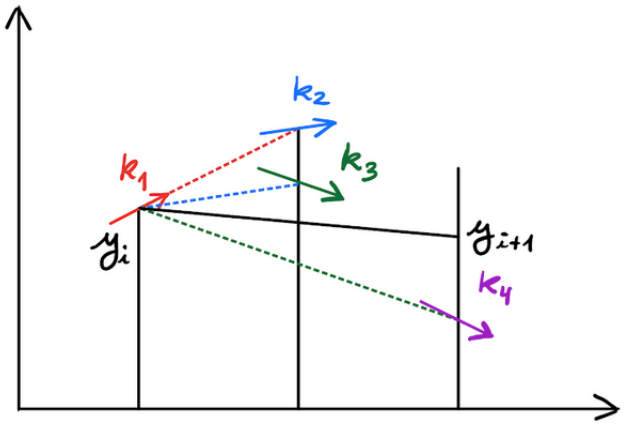

Figura 5: Interpretació geomètrica de les fórmules de Runge-Kutta d'ordre 4. Cada una de les k_i representa un pendent. El resultat és una mitjana ponderada d'aquests pendents.

Algorithm 1 Algorisme de control de pas

Input: punt inicial $(F, B, R) = (0.7, 0.1, 0.2)$
Output: fitxer amb els valors trobats

$\epsilon = 10^{-12}$
while $t \leq 300$ **do**
  *aproH $\leftarrow (F, B, R)$
  *aproH2 $\leftarrow (F, B, R)$
  runge_kutta(step,aproH)
  runge_kutta(step/2,aproH2)
  runge_kutta(step/2,aproH2)
  coef $= \left(\frac{2^4}{\epsilon(2^4 - 1)} \parallel \text{*aproH} - \text{*aproH2} \parallel \right)^{1/5}$
  if coef > 2 **then**
    step $= 2 \cdot$step/coef
  else
    $t+ =$ step
    $F, B, R \leftarrow$ *aproH2
    fitxer $\leftarrow (t, F, B, R)$
    step $= 2 \cdot$step/coef

algebraics com el Maple, MATLAB o Mathematica. Hem programat també un control de pas segons una tolerància de l'error ϵ basat en l'algorisme 1.

Per tal de resoldre la segona qüestió de trobar un punt fix de l'aplicació de Poincaré (en cas d'existir), s'ha implementat el mètode de Newton.

S'explicaran més detalls dels programes en el cas pràctic que s'estudiarà a les seccions següents.

3 Cas pràctic: apogeu i caiguda de civilitzacions

Aquesta segona part del treball consisteix en aplicar els programes que s'han implementat del Mètode de Runge-Kutta (veure Annex C) per tal d'estudiar les EDOs que modelitzen el fenomen sociodemogràfic dels apogeus i caigudes de civilitzacions.

L'aplicació triada constitueix un *case study* del llibre [3, pàg. 125], que a la vegada està documentat per l'article [4] de Feichtinger.

Els models socioeconòmics de les civilitzacions acostumen a estar inspirats en els cicles dinàstics de l'antiga Xina, ja que es tracta d'una de les civilitzacions més antigues de l'història que pot demostrar una certa continuïtat. La Xina ha tingut moltes dinasties en l'antiguitat, com per exemple Xia, Shang, Zhou, Qin, Han i Ming, i entre aquestes dinasties va haver-hi períodes on el nombre d'habitants va sofrir decaigudes degut al canvi de dinasties. Durant aquests períodes d'anarquisme les classes socials governants eren massa febles per controlar el nombre de bandits. Tot i així, l'estat d'anarquia no se sostenia en el temps i les classes socials governants tornaven a aconseguir el poder, contenint el nombre de bandits i fent que la població de servents i pagesos creixés. Estudiarem a continuació un model concret d'aquesta dinàmica social.

3.1 Model demogràfic

Per intentar explicar l'apogeu i la caiguda de les dinasties xineses es planteja un sistema de tres classes socials: $F(t)$ representa la població de pagesos i servents, $B(t)$ la població de bandits i $R(t)$ la població de governants i els soldats del seu exèrcit. Les equacions diferencials ordinàries que regulen aquestes poblacions són:

$$\frac{dF}{dt} = rF\left(1-\frac{F}{K}\right) - \frac{aFB}{b+F} - hFR, \tag{3.1}$$

$$\frac{dB}{dt} = \frac{eaFB}{b+F} - mB - \frac{cBR}{d+B}, \tag{3.2}$$

$$\frac{dR}{dt} = \frac{faFB}{b+F} - gR, \tag{3.3}$$

on r, K, a, b, h, e, d, m, c, f, g són constants positives del model. Aquest és un model depredador-presa. Les interaccions entre classes es poden descriure com: els bandits roben els pagesos, els governants capturen els bandits i els governants imposen taxes d'impostos als pagesos per poder continuar capturant bandits. Analitzem les EDOs del sistema més detingudament.

De l'equació (3.1), el model dels pagesos en absència de bandits i governants té un creixement logístic:

$$\frac{dF}{dt} = rF\left(1-\frac{F}{K}\right).$$

La constant r pot interpretar-se com la velocitat segons la qual la població de pagesos es reprodueix, i la constant K és el punt d'homeòstasi o punt d'equilibri del

creixement logístic. Així, la població de pagesos està controlada en cas que creixi excessivament, ja que per falta de recursos tendirà al punt d'equilibri.

Tant els bandits com els governants tenen la seva constant de mortalitat o jubilació del servei, m en el cas dels bandits i g en el cas dels governants. No es considera que els pagesos tinguin una constant de mortalitat perquè podem suposar que ja està implícita en la formulació del creixement logístic.

En la primera l'equació (3.1) de creixement dels pagesos, el terme $\dfrac{aFB}{(b+F)}$ és una taxa de saturació de bandits sobre els agricultors. Aquesta taxa de saturació es pot entendre com el cost del temps que tenen els bandits per trobar la seva presa òptima, i es coneix com una resposta funcional del *depredador* al canvi de densitat de la *presa*. Aquesta taxa de saturació actua de la següent manera: si el nombre de pagesos assassinats per un bandit és $T(F) = \dfrac{aF}{(b+F)}$, aleshores quan el nombre de pagesos és petit el comportament serà semblant a un model lineal $T(F) = \frac{a}{b}F$, mentre que si hi ha sobrepoblació de pagesos, quan $F \to \infty$, aleshores el terme tendeix a la constant $T(F) = a$, que representa la limitada capacitat dels depredadors o la perseverància quan la *presa* és abundant. Tot això està il·lustrat a la Figura 6.

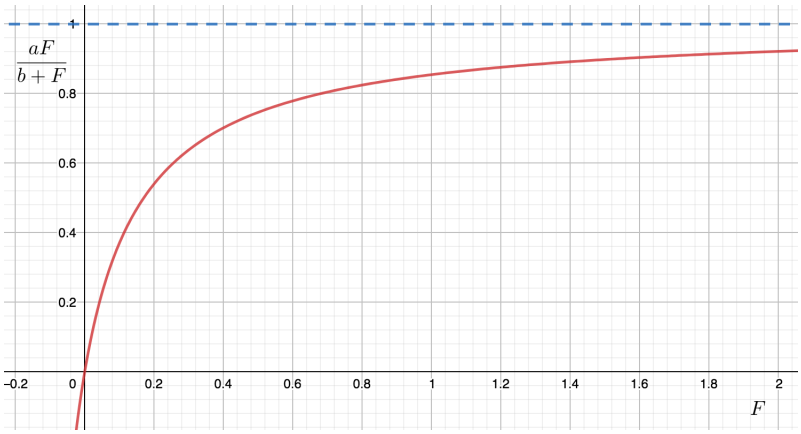

Figura 6: Gràfica de la taxa de saturació $T(F) = \dfrac{aF}{(b+F)}$ en funció de F, segons els valors de les costants a i b de la Taula 1. A mesura que F creix $T(F)$ tendeix a l'asímptota horitzontal d'ordenada a. Prop de l'origen s'observa un comportament lineal del tipus $T(F) \approx \frac{a}{b}F$.

A l'equació (3.2), els bandits tenen un terme de mortalitat addicional $\dfrac{cBR}{(d+B)}$, que són els bandits que moren quan es troben amb l'exèrcit dels governants. Altre cop, és una taxa de saturació ara envers els governants. Fa una aproximació del temps que els soldats han d'invertir per trobar els bandits. El nombre de bandits que un soldat pot matar és $\dfrac{cB}{(d+B)}$ que tendeix a la constant c en cas que el nombre de bandits sigui molt gran. Considerarem que els soldats estan entrenats professionalment i per això en una lluita entre bandits i soldats (governants) només

hi hauria baixes en el grup dels bandits.

El terme hFR en l'equació (3.1) representa una resposta lineal dels pagesos als impostos excessius dels governants. Fixem-nos que aquest terme és proporcional tant als pagesos com als governants, per tant quan més gran sigui el nombre de soldats, taxes més elevades seran necessàries per mantenir-los, i per tant una taxa més alta per cada pagès.

A les equacions (3.2) i (3.3), el creixement dels bandits i dels governants depenen del terme $\dfrac{aFB}{(b+F)}$, en cada cas ponderat per una constant diferent, e en el cas dels bandits i f en el cas dels governants. Això ens indica que els depredadors creixen proporcionalment segons la contribució de la presa. Les poblacions estan mesurades sobre la màxima capacitat del creixement logístic dels pagesos K, és a dir, que el nombre de pagesos, bandits i governants estaran tots a l'interval $[0, K]$.

$r = 1$	$a = 1$	$m = 0.4$	$e = 1.2$	$c = 0.4$	$h = 0.1$
$K = 1$	$b = 0.17$	$g = 0.009$	$f = 0.1$	$d = 0.42$	

Taula 1: Valors dels paràmetres que usarem extrets de [3, Table 5.1 pàg. 126]. Aquests paràmetres han estat escollits per tal d'observar cicles en les classes socials i no tant per tenir una acurada aproximació històrica. A més com a valors inicials del problema usem $F(0) = 0.7, B(0) = 0.2, R(0) = 0.1$.

3.2 Solució numèrica

Per resoldre les equacions diferencials de problema de valor inicial usarem els valors dels paràmetres en la Taula 1 i la condició inicial $(F(0), B(0), R(0)) = (0.7, 0.2, 0.1)$ a les EDOs (3.1), (3.1) i (3.1) que recordem aquí:

$$\begin{aligned}
\frac{dF}{dt} &= rF\left(1 - \frac{F}{K}\right) - \frac{aFB}{b+F} - hFR, \\
\frac{dB}{dt} &= \frac{eaFB}{b+F} - mB - \frac{cBR}{d+B}, \\
\frac{dR}{dt} &= \frac{faFB}{b+F} - gR.
\end{aligned}$$

El mètode d'integració numèrica que usarem serà d'un pas, més en concret serà Runge-Kutta d'ordre $p = 4$ amb un control de pas usant la fórmula (1.6). Ara explicarem més detalls de la programació del mètode de Runge-Kutta, el programa es troba a l'Annex C. S'ha programat en una funció de tipus *void* amb entrada el pas *step* i un vector **valors* on hi ha els valors de F, B, R. Per calcular els valors de les constants k_{ij} s'ha adaptat la manera com es fa a [5, pàg. 215] per a 3 equacions. Si normalment es necessiten 4 constants per fer una iteració de Runge-Kutta, al

tenir 3 equacions necessitem 12 constants. Les hem calculat de la següent forma.

$$\begin{aligned} F, B, R &\leftarrow *valors \\ k1i &\leftarrow step * fi(F, B, R) \\ k2i &\leftarrow step * fi(F + 0.5 * k11, B + 0.5 * k12, R + 0.5 * k13) \\ k3i &\leftarrow step * fi(F + 0.5 * k21, B + 0.5 * k22, R + 0.5 * k23) \\ k4i &\leftarrow step * fi(F + k31, B + k32, R + k33), \end{aligned}$$

on per cada $i = 1, 2, 3$ fi representa una equació, és a dir, $fi = 1$ (3.1), $fi = 2$ (3.2), $fi = 3$ (3.3). Finalment actualitzem els valors que tenim guardats a $*valors$:

$$\begin{aligned} valors[0] &\leftarrow F + (k11 + 2 * k21 + 2 * k31 + k41)/6 \\ valors[1] &\leftarrow B + (k12 + 2 * k22 + 2 * k32 + k42)/6 \\ valors[2] &\leftarrow R + (k13 + 2 * k23 + 2 * k33 + k43)/6. \end{aligned}$$

Durant la integració comprovarem que els passos h no siguin molt grans per després poder dibuixar la funció en gràfiques. El control de pas s'ha implementat amb la fórmula (1.6). S'ha assignat el valor a una variable *coef* i, si aquest era més gran que 2, aleshores es desestimava l'aproximació i es repetia el procés amb un pas $step \leftarrow 2 * step/coef$.

S'ha programat en llenguatge C i s'han anat guardant les dades en un fitxer. Després s'ha usat el llenguatge R per dibuixar les gràfiques. El resultat es pot veure a la Figura 7. És un resultat qualitativament i quantitativament molt similar al que es mostra a [3, Figure 5.14 pàg. 127].

Amb la tria d'aquests paràmetres la població de pagesos i bandits oscil·la. La població de pagesos quasi tot el temps està en el màxim de la seva capacitat que permet el medi, mentre que la població de bandits cau dramàticament a cada oscil·lació. Aquesta caiguda dels bandits coincideix amb les pujades dels pagesos, i les pujades dels bandits coincideixen amb la caiguda dels pagesos. La classe governant té el seu pic en el mateix moment que els bandits tenen la seva pitjor marca.

Per tal d'entendre millor aquestes oscil·lacions, s'ha representat la solució trobada en 3D a la Figura 8. Es pot observar el mateix cicle entre les tres classes socials, però va creixent en l'eix dels governants. Tal com es veu el gràfic podríem pensar que existeix una òrbita periòdica que seria l'ω límit de la solució del valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$. Ho investiguem i efectivament usant el programa D (explicat en detall a la Secció 3.3) es troba l'òrbita periòdica representada a la Figura 9. Així doncs, la solució al nostre problema d'EDOs amb valor inicial tendeix a una òrbita periòdica, i per tant a una interacció cíclica entre les poblacions de governants, pagesos i bandits.

3.3 Càlcul d'òrbites periòdiques

Ara s'explicarà com s'ha implementat el programa que es troba a l'Annex D. El programa calcula, en cas d'existir, l'òrbita periòdica del sistema d'equacions 3.1, 3.2, 3.3 amb condició inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$.

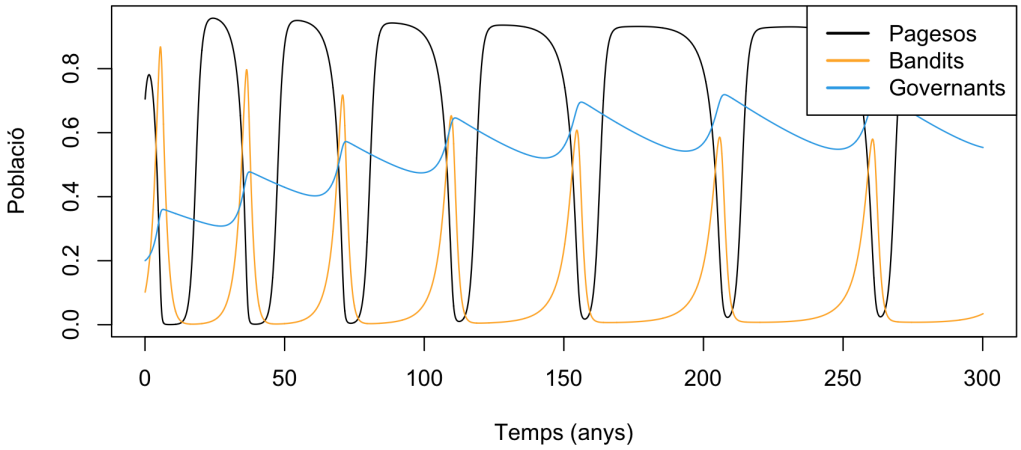

Figura 7: Funcions solució de les EDOs (3.1), (3.1) i (3.1) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.2, 0.1)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, pels valors de les constants segons la Taula 1.

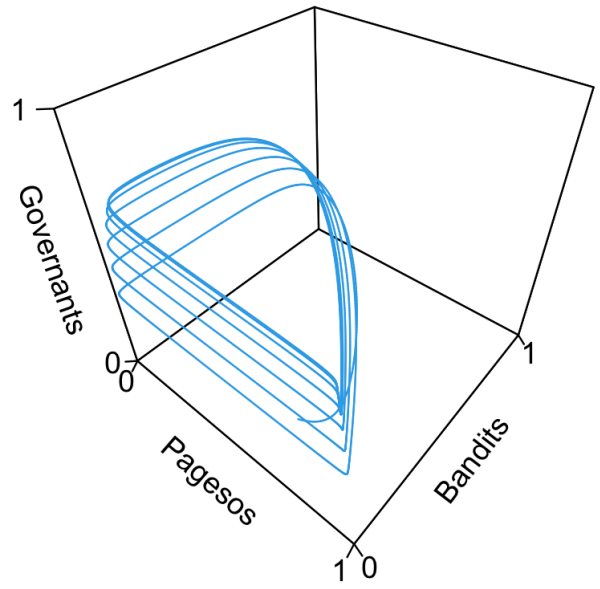

Figura 8: Representació 3D de la solució del sistema d'EDOs (3.1), (3.1) i (3.1) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.2, 0.1)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta C, pels valors de les constants segons la Taula 1.

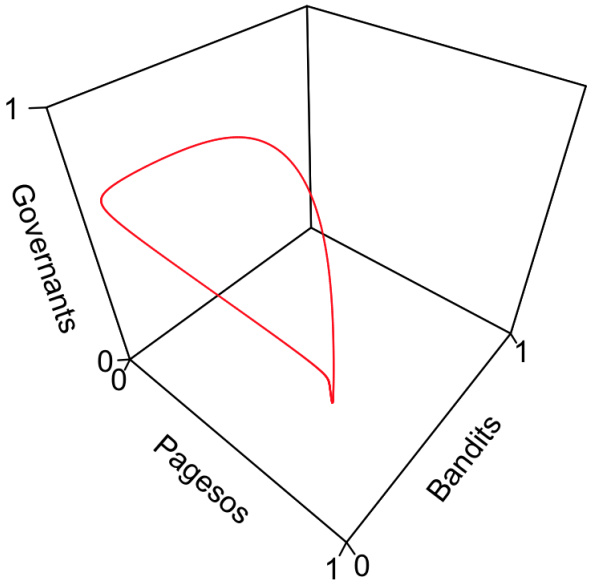

Figura 9: Representació 3D de l'òrbita periòdica del sistema d'EDOs (3.1), (3.1) i (3.1) obtinguda amb la nostra implementació del programa D, pels valors de les constants segons la Taula 1

Es considera la secció transversal $\Sigma = \{(F, B, R) \in \mathbb{R}^3 : B = 0.4\}$. S'anomena fix_B el valor fix de $B = 0.4$. Definim l'aplicació de Poincaré τ:

$$\tau : \Sigma \longrightarrow \Sigma$$

$$p = (F_0, fix_B, R_0) \to \tau(p) = (\tau_1(p), fix_B, \tau_2(p)) = (F, fix_B, R).$$

El primer problema que solucionem al programa és la sortida, ja que el punt inicial és $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2) \notin \Sigma$, així que ens trobem en el cas de l'esquema de la part esquerra de la Figura 10. Per trobar (F, R) primer buscarem el punt de tall amb Σ i aleshores podrem aplicar τ en aquest punt i trobarem (F, R). Per trobar la corba solució integrem usant el programa de l'Annex C, el qual té implementat un Runge-Kutta d'ordre 4 amb control de pas (explicat a la Secció 3.2).

El procés de trobar (F, R) es fa a la funció *int Poincare*, que té per entrada les tres components F_0, B_0, R_0 d'un punt de $\mathbb{R}^3$ i dos punters F, R, que són on es guarden els valors. El retorn d'aquesta funció simplement és 1 si s'ha arribat a la solució, o bé -1 en cas contrari. Per solucionar aquest problema d'inici, al programa mirem el nombre de talls que anem fent a Σ. A cada pas i de Runge-Kutta tenim valors aproximats F_i, B_i, R_i, aleshores es guarden els signes de $B_i - fix_B$. Quan ens trobem amb un canvi de signe significa que hem travessat Σ. En el primer cas $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ fixem-nos que serà al tercer tall que haurem trobat els valors de F i R. Es pot veure una representació d'aquesta situació a la Figura 10.

Després d'aquest primer cas sempre estarem en punts de Σ, és a dir en la situació de la dreta de la Figura 10. En aquests casos trobarem els valors de F, R al segon tall amb Σ.

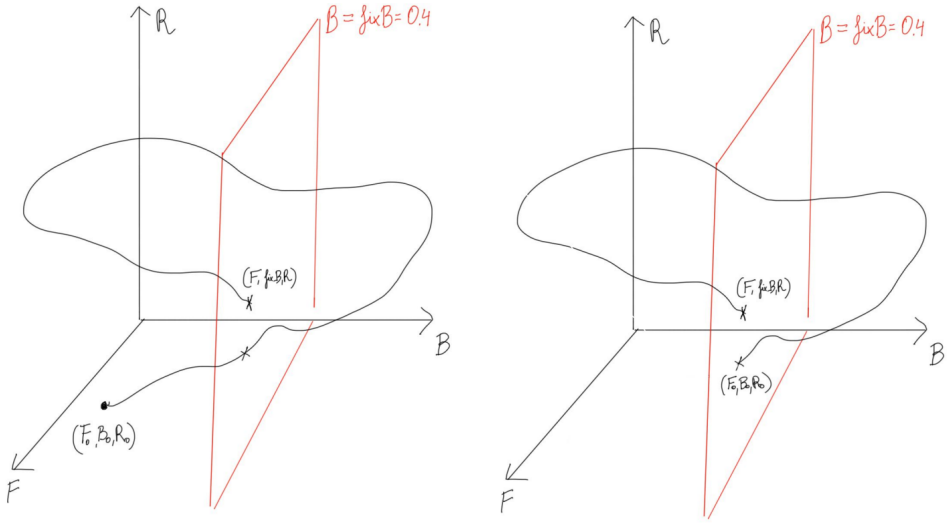

Figura 10: A la imatge de l'esquerra es mostra una situació en la qual el punt inicial (F_0, B_0, R_0) no es troba sobre la secció transversal $\Sigma = \{(F, B, R) \in \mathbb{R}^3 : B = 0.4\}$. Es mostra el que faria l'algoritme per calcular l'aplicació de Poincaré per trobar (F, R). A la figura de la dreta es mostra la situació en que el punt $(F_0, B_0, R_0) \in \Sigma$ i s'interpreta el que faria l'algoritme de l'aplicació de Poincaré per trobar els valors (F, R).

Suposem que s'està usant la funció *int Poincare* i s'ha partit del punt donat $(F_0, B_0 = fix_B, R_0)$ i s'acaba de fer el segon tall a Σ amb $k+1$ iteracions de Runge-Kutta. Així doncs, es té l'aproximació $F_{k+1}, B_{k+1}, R_{k+1}$ amb $B_{k+1} \approx fix_B$. El programa el que farà és agafar F_k, B_k, R_k i iterar (usant altre cop Runge-Kutta d'ordre 4) amb un pas $step = |B_k - fix_B|/2$ fins obtenir un error més petit que $|B_{k_s} - fix_B| \leq 10^{-8}$, on s és el nombre d'iteracions que s'han realitzat per arribar a aquest error. Això s'ha hagut d'implementar d'aquesta manera, perquè, si no es demanés més precisió per estar a Σ, no funcionaria correctament el següent pas de trobar l'òrbita periòdica.

Un cop tenim el punt de partida, suposant que estem a Σ, i que tenim (F_0, R_0) i la imatge per l'aplicació de Poincaré τ (F, R), definim la funció g com

$$g : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$$

$$q = (F_0, R_0) \to g(q) = (g_1(q), g_2(q)) =$$

$$= (\tau_1((F_0, fix_B, R_0)), \tau_2((F_0, fix_B, R_0))) - q = (F - F_0, R - R_0).$$

Per trobar l'òrbita periòdica hem de trobar dos valors F_0, R_0 tals que $g(F_0, R_0) = 0$. Per això aplicarem el mètode de Newton (implementat a la funció *int newton*) a la funció g:

$$(F_0, R_0)_{nou} = (F_0, R_0) - (Dg(F_0, R_0))^{-1} \cdot g(F_0, R_0)$$

on Dg és la matriu

$$Dg = \begin{pmatrix} \frac{\partial g_1(F_0,R_0)}{\partial F_0} & \frac{\partial g_1(F_0,R_0)}{\partial R_0} \\ \frac{\partial g_2(F_0,R_0)}{\partial F_0} & \frac{\partial g_2(F_0,R_0)}{\partial R_0} \end{pmatrix} = \begin{pmatrix} \frac{\partial \tau_1(F_0,fix_B,R_0)}{\partial F_0} - 1 & \frac{\partial \tau_1(F_0,fix_B,R_0)}{\partial R_0} \\ \frac{\partial \tau_2(F_0,fix_B,R_0)}{\partial F_0} & \frac{\partial \tau_2(F_0,fix_B,R_0)}{\partial R_0} - 1 \end{pmatrix}.$$

El problema ara es troba en calcular les derivades parcials de la funció de Poincaré $\tau_1(F_0, fix_B, R_0), \tau_2(F_0, fix_B, R_0)$. Ho farem calculant una aproximació usant la fórmula de definició de derivada i una σ prou petita:

$$\frac{\partial \tau_1(F_0, fix_B, R_0)}{\partial F_0} = \frac{\tau_1(F_0 + \sigma, fix_B, R_0) - \tau_1(F_0, fix_B, R_0)}{\sigma},$$

$$\frac{\partial \tau_2(F_0, fix_B, R_0)}{\partial F_0} = \frac{\tau_2(F_0 + \sigma, fix_B, R_0) - \tau_2(F_0, fix_B, R_0)}{\sigma}.$$

Fixem-nos que per obtenir els valors de $\tau_1(F_0+\sigma, fix_B, R_0)$ i $\tau_2(F_0+\sigma, fix_B, R_0)$ podem aconseguir-los cridant la funció *int Poincare* $(F_0+\sigma, fix_B, R_0, F, R)$ una vegada. Anàlogament ho podem fer per obtenir $\tau_1(F_0, fix_B, R_0+\sigma)$ i $\tau_2(F_0, fix_B, R_0+\sigma)$.

Al programa tenim la funció *void fillMatrx* que té per entrada una matriu, dos valors i σ. El que fa és plenar la matriu segons acabem d'explicar (cridant la funció *int Poincare*) i usant l'aproximació de la derivada. També tenim una funció que calcula la inversa d'una matriu 2×2 per poder invertir la matriu i aconseguir així calcular Dg.

En conclusió, quan cridem *newton* (F_0, R_0), primer calcula usant la funció *Poincare* els valors F, R. Després plena la matriu Dg i calcula la seva inversa, per finalment calcular $nextF, nextR$ usant el mètode iteratiu de Newton,

$$\begin{pmatrix} nextF \\ nextR \end{pmatrix} = \begin{pmatrix} F_0 \\ R_0 \end{pmatrix} - \left(Dg \begin{pmatrix} F_0 \\ R_0 \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} F - F_0 \\ R - R_0 \end{pmatrix}.$$

Un cop obtinguts els valors de $nextF, nextR$ actualitzem $F_0, R_0 \leftarrow nextF, nextR$, i tornem a començar cridant $newton(F_0, R_0)$ amb els valors actualitzats. L'algorisme s'atura quan l'error relatiu $ER < 10^{-12}$,

$$ER = \frac{\| \ (nextF - F_0, nextR - R_0) \ \|}{\| \ (nextF, nextR) \ \|}.$$

D'aquesta manera aconseguim trobar un punt $\hat{p} = (\hat{F}, fix_B, \hat{R})$ tal que $\tau(\hat{p}) = \hat{p}$. Arribat en aquest moment, integrem una volta usant Runge-Kutta guardant els valors a un fitxer, ja que seran els valors dels punts que definiran l'òrbita periòdica.

3.4 Anàlisi qualitatiu

A la Secció 3.2 hem vist que la solució al problema d'EDOs amb valor inicial del nostre cas pràctic tendeix a una òrbita periòdica, i per tant a una interacció cíclica entre les poblacions de governants, pagesos i bandits. És natural preguntar-se per la natura d'aquest comportament cíclic i si es manté en variar els paràmetres del sistema d'EDOs. Amb aquesta motivació, iniciem en aquesta secció un estudi qualitatiu del sistema d'EDOs.

En aquesta secció imposarem $\frac{dF}{dt} = \frac{dB}{dt} = \frac{dR}{dt} = 0$ per buscar els punts d'equilibri

del sistema d'equacions, i veure si es poden classificar segons la definició (2.7):

$$\begin{aligned}
\frac{dF}{dt} &= rF\left(1-\frac{F}{K}\right)-\frac{aFB}{b+F}-hFR=0, &(3.4)\\
\frac{dB}{dt} &= \frac{eaFB}{b+F}-mB-\frac{cBR}{d+B}=0, &(3.5)\\
\frac{dR}{dt} &= \frac{faFB}{b+F}-gR=0. &(3.6)
\end{aligned}$$

De (3.6) podem aïllar $R(F,B)=\frac{faFB}{g(b+F)}$. Ara substituïm aquesta expressió en (3.4),

$$rF\left(1-\frac{F}{K}\right)-\frac{aFB}{b+F}-hF\frac{faFB}{g(b+F)}=0,$$

i aïllem B

$$B=\frac{rF\left(1-\frac{F}{K}\right)}{\left(\frac{aF}{b+F}+hF\frac{faF}{g(b+F)}\right)}=\frac{r(b+F)(1-\frac{F}{K})}{a+\frac{hfa}{g}F},$$

per obtenir $B=B(F)$

$$B(F)=\frac{r(b+F)(1-\frac{F}{K})}{a+\frac{hfa}{g}F}. \tag{3.7}$$

Fixem-nos que usant (3.7) podem expressar $R(B,F)=R(F)$:

$$R(F)=\frac{rfF(1-\frac{F}{K})}{g+hfF}. \tag{3.8}$$

Ara substituïm l'expressió de $R(F,B)$ l'equació (3.5):

$$\frac{eaF}{b+F}B-mB-\frac{c}{d+B}\frac{faF}{g(b+F)}(B)^2=0.$$

Per tant tenim, o bé $B(F)=0$, o bé

$$\frac{eaF}{b+F}-m-\frac{c}{d+B}\frac{faF}{g(b+F)}B=0.$$

Substituint l'expressió de $B(F)$:

$$\frac{eaF}{b+F}-m-\frac{c}{d+\frac{r(b+F)(1-\frac{F}{K})}{a+\frac{hfa}{g}F}}\cdot\frac{faF}{g(b+F)}\frac{r(b+F)(1-\frac{F}{K})}{a+\frac{hfa}{g}F}=0,$$

$$\frac{eaF}{b+F}-m-\frac{crfaF(1-\frac{F}{K})}{dg(a+\frac{hfa}{g}F)+rg(b+F)(1-\frac{F}{K})}=0.$$

Finalment tenim una funció $\Psi(F)=0$, on

$$\Psi(F)=\frac{eaF}{b+F}-m-\frac{crfaF(1-\frac{F}{K})}{d(ag+hfaF)+rg(b+F)(1-\frac{F}{K})} \tag{3.9}$$

Ens falta observar què succeeix en el cas $B(F) = 0$. Com que les constants són positives, $a + hfaF/g > 0$, es dedueix que o bé $F = -b$, o bé $F = K$. Però sabem que el rang de F és l'interval $(0, K)$, ja que K és la constant que determina la saturació màxima del medi, i $F(t)$ té un creixement logístic depenent d'aquesta K. Per tant, mai passarà que $B(F) = 0$.

Un cop fixades les constants tenim la funció $\Psi(F)$, els zeros de la qual volem trobar, ja que ens donaran els punts d'equilibri $\{(F_{pe}, B_{pe}, R_{pe}) : \Psi(F_{pe}) = 0, B_{pe} = B(F_{pe}), R_{pe} = R(F_{pe})\}$ del sistema. Usarem el mètode de Newton per trobar aquests zeros. Per tant, necessitem conèixer $\frac{d\Psi}{dF}$. Calculem-la:

$$\frac{d\Psi}{dF} = \frac{ea(b+F) - eaF}{(b+F)^2} - \left(\frac{crfa(1-\frac{2F}{K})(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} + \right.$$

$$\left. + \frac{-crfaF(1-\frac{F}{K})(dhfa + rg(-\frac{b}{K} + 1 - \frac{2F}{K}))}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right) =$$

$$= \frac{eab}{(b+F)^2} - crfa \left(\frac{(1-\frac{F}{K}-\frac{F}{K})(dag + dhfaF + rgb - \frac{rgbF}{K} + rgF - \frac{rgF^2}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} + \right.$$

$$\left. + \frac{(1-\frac{F}{K})(-dhfaF + \frac{rgbF}{K} - rgF + \frac{2rgF^2}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right) =$$

$$= \frac{eab}{(b+F)^2} - \left(\frac{crfa(-\frac{F}{K})(dag + dhfaF + rgb - \frac{rgbF}{K} + rgF - \frac{rgF^2}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} + \right.$$

$$\left. + \frac{crfa(1-\frac{F}{K})(dag + rgb + \frac{rgF^2}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right) =$$

$$= \frac{eab}{(b+F)^2} - \left(\frac{crfa(-\frac{F}{K})(2dag + dhfaF + 2rgb - \frac{rgbF}{K} + rgF)}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} + \right.$$

$$\left. + \frac{crfa(dag + rgb + \frac{rgF^2}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right) =$$

$$= \frac{eab}{(b+F)^2} - crfa \left(\frac{dag + rgb - \frac{F}{K}(2dag + dhfaF + 2rgb - \frac{rgbF}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right).$$

Finalment,

$$\frac{d\Psi}{dF} = \frac{eab}{(b+F)^2} - crfa \left(\frac{dag + rgb - \frac{F}{K}(2dag + dhfaF + 2rgb - \frac{rgbF}{K})}{(d(ag+hfaF) + rg(b+F)(1-\frac{F}{K}))^2} \right). \quad (3.10)$$

Ara, ja podem aplicar el mètode de Newton

$$F_{i+1} = F_i - \frac{\Psi(F_i)}{\frac{\partial \Psi}{\partial F}(F_i)},$$

que convergirà a la component F_{pe} del punt d'equilibri. Per calcular B_{pe} i R_{pe} farem ús de les equacions (3.7) substituint F per F_{pe} i obtenint el valor B_{pe}. Anàlogament, usant l'equació (3.8) per trobar R_{pe}.

Un cop trobats els punts d'equilibri, calculem la matriu diferencial del sistema per veure si poden ser classificats. Considerem $U(F,B,R)$ que sigui l'equació (3.1), $V(F,B,R)$ sigui l'equació (3.2) i $W(F,B,R)$ sigui l'equació (3.3):

$$\begin{pmatrix} \frac{\partial U(F,B,R)}{\partial F} & \frac{\partial U(F,B,R)}{\partial B} & \frac{\partial U(F,B,R)}{\partial R} \\ \frac{\partial V(F,B,R)}{\partial F} & \frac{\partial V(F,B,R)}{\partial B} & \frac{\partial V(F,B,R)}{\partial R} \\ \frac{\partial W(F,B,R)}{\partial F} & \frac{\partial W(F,B,R)}{\partial B} & \frac{\partial W(F,B,R)}{\partial R} \end{pmatrix} =$$

$$= \begin{pmatrix} r(1-\frac{2F}{K}) - \frac{abB}{(b+F)^2} - hR & -\frac{aF}{b+F} & -hF \\ \frac{eabB}{(b+F)^2} & \frac{eaF}{b+F} - m - \frac{cdR}{(d+B)^2} & -\frac{cB}{d+B} \\ \frac{fabB}{(b+F)^2} & \frac{faF}{b+F} & -g \end{pmatrix}.$$

Per poder classificar els punts hem de substituir $F = F_{pe}$, $B = B_{pe}$, $R = R_{pe}$ i calcular el polinomi característic de la matriu. Després de calcular el polinomi característic hem de trobar les seves arrels, que seran els VAPs. Per resoldre el polinomi usarem les fórmules de Cardano. Donada una equació de tercer grau $ax^3 + bx^2 + cx + d = 0$, el mètode de resolució de Cardano diu el següent. Primer convertim el polinomi en mònic dividint per a,

$$x^3 + \frac{b}{a}x^2 + \frac{c}{a}x + \frac{d}{a} = 0.$$

Fem un canvi de variable $x = z - \frac{b}{3a}$, i així aconseguim que desaparegui el terme quadràtic:

$$z^3 + pz + q = 0,$$
$$p = \frac{3ac - b^2}{3a^2}, \quad q = \frac{2b^3 - 9abc + 27a^2d}{27a^3}.$$

Ara hem de calcular el discriminant i separar per casos

$$\Delta = q^2 + \frac{4p^3}{27}.$$

En el cas que $\Delta > 0$, l'equació té una solució real i dues de complexes. Definim u, v de la següent manera

$$u = \sqrt[3]{\frac{-q + \sqrt{\Delta}}{2}}, \; v = \sqrt[3]{\frac{-q - \sqrt{\Delta}}{2}}.$$

Les solucions són

$$\begin{cases} x_1 &= u+v-\frac{b}{3a}, \\ x_2 &= -\frac{1}{2}(u+v)+\frac{b}{3a}+i\left(\frac{\sqrt{3}}{2}(u-v)\right), \\ x_3 &= -\frac{1}{2}(u+v)-\frac{b}{3a}+i\left(\frac{\sqrt{3}}{2}(u-v)\right). \end{cases}$$

En el cas que $\Delta = 0$, i p, q són zero. Aleshores només té una solució real triple $x_{1,2,3} = -\dfrac{b}{3a}$. En cas contrari, té dues solucions reals (una simple i una doble),

$$\begin{cases} x_1 &= 2\sqrt[3]{-\frac{q}{2}}-\frac{b}{3a}, \\ x_2 = x_3 &= -\sqrt[3]{-\frac{q}{2}}-\frac{b}{3a}. \end{cases}$$

Finalment, si $\Delta < 0$, aleshores l'equació té tres solucions reals,

$$\begin{cases} x_1 &= 2\sqrt{\frac{-p}{3}}\cos\left(\frac{\theta}{3}\right)-\frac{b}{3a}, \\ x_2 &= 2\sqrt{\frac{-p}{3}}\cos\left(\frac{\theta+2\pi}{3}\right)-\frac{b}{3a}, \\ x_3 &= 2\sqrt{\frac{-p}{3}}\cos\left(\frac{\theta+4\pi}{3}\right)-\frac{b}{3a}, \end{cases}$$

amb $\theta = \arccos\left(\frac{3q}{2p}\sqrt{\frac{-3}{p}}\right)$.

Un cop tenim calculats els VAPs amb l'ajuda de les fórmules de Cardano, guardem a un fitxer les dades del valor del paràmetre que estem modificant, i la part real dels tres VAPs. Així després podrem fer una gràfica i estudiar els canvis de signe.

S'ha programat en C el programa de l'Annex E, que efectua tots els càlculs explicats en aquesta secció. Degut a la gran quantitat de paràmetres, el programa comença preguntant quin serà el paràmetre que voldràs modificar i en quin interval el mourem. Un cop triat, el que fa és cridar la funció $newton_{Psi}$, que busca el punt d'equilibri usant Newton sobre la funció $\Psi(F)$ (3.9), a partir de la seva derivada (3.10). L'algoritme de Newton itera de la següent manera,

$$F_{i+1} = F_i - \frac{\Psi(F_i)}{\Psi'(F_i)}.$$

Suposem que convergeix a un F_{pe} tal que $\Psi(F_{pe}) = 0$. Aleshores usant les equacions (3.7) i (3.8) es troben B_{pe} i R_{pe}. Un cop tenim el punt d'equilibri F_{pe}, B_{pe}, R_{pe} substituïm els valors a la matriu

$$M = \begin{pmatrix} r(1-\frac{2F_{pe}}{K})-\frac{abB_{pe}}{(b+F_{pe})^2}-hR & -\frac{aF_{pe}}{b+F_{pe}} & -hF_{pe} \\ \frac{eabB_{pe}}{(b+F_{pe})^2} & \frac{eaF_{pe}}{b+F_{pe}}-m-\frac{cdR_{pe}}{(d+B_{pe})^2} & -\frac{cB_{pe}}{d+B_{pe}} \\ \frac{fabB_{pe}}{(b+F_{pe})^2} & \frac{faF_{pe}}{b+F_{pe}} & -g \end{pmatrix}.$$

La funció implementada amb el nom *inicialitzar*, que té per entrada la matriu M i el punt d'equilibri, plena amb els valors la matriu M. Després calculem el polinomi característic de la matriu M,

$$P_M(\lambda) = \det(M - \lambda I).$$

Un cop tenim el polinomi característic, calculem les seves arrels usant les fórmules implementades a la funció *cardano*, que té per entrada un polinomi i un vector on guardarem les parts reals de les 3 arrels. Ara incrementem el valor del paràmetre que estem estudiant i repetim el procés fins que arribem a l'altre extrem de l'interval introduït.

3.5 Estudi per a diversos valors dels paràmetres

El model proposat inclou diversos paràmetres i és natural preguntar-se quin paper té la seva variació en les trajectòries de la solució. En aquesta secció estudiem el canvi de valor d'alguns del paràmetres. Especialment estem interessats en saber si el comportament cíclic de les poblacions aproximant-se a una òrbita periòdica és un comportament aïllat o prou general, i si hi ha d'altres possibilitats, com per exemple que les tres poblacions de governants, pagesos i bandits estabilitzin tendint a un punt d'equilibri.

Considerem primer variacions del paràmetre h, que representa la taxa d'impostos que imposen els governants sobre els pagesos, tal i com hem discutit a la descripció del model de la Secció 3.1. Anteriorment hem considerat el valor de $h = 0.1$. Ara l'augmentem a $h = 2$ i deixem la resta de paràmetres iguals, tal com s'indica a la Taula 2, i estudiem com afecta aquest canvi a la corba solució.

$r = 1$	$a = 1$	$m = 0.4$	$e = 1.2$	$c = 0.4$	$h = 2.0$
$K = 1$	$b = 0.17$	$g = 0.009$	$f = 0.1$	$d = 0.42$	

Taula 2: Valors dels paràmetres on hem variat h respecte la Taula inicial 1

Com es pot veure a la Figura 11 després d'algunes oscil·lacions entre les tres classes socials s'arriba a un estat d'equilibri. Pot semblar sorprenent que amb una taxa més elevada d'impostos dels governants sobre els pagesos s'arribi a un estat d'equilibri després d'un temps. Una possible explicació seria que amb una taxa més elevada la classe governant tindria més recursos per controlar la població de bandits.

Per tal d'entendre millor l'estat d'estabilitat al qual s'arriba, s'ha representat la solució trobada en 3D a la Figura 12. Tal i com es veu al gràfic, sembla que amb $h = 2$ el punt d'equilibri del sistema d'equacions (3.1),(3.2),(3.3) seria un punt hiperbòlic atractor. Ho confirmem usant el programa E explicat a la Secció 3.4 d'anàlisi qualitatiu.

Estudiem la variació del paràmetre h en tot el rang que va des de 0.1 fins a 2 amb un *step* de 0.01. Per a cada h es calculen els VAPs en el punt d'equilibri del sistema d'EDOs. Es guarden les parts reals dels VAPs en un fitxer, i s'han dibuixat gràficament a la Figura 13. S'observa que la part real del primer VAP sempre és negativa. Les parts reals del segon i tercer VAPs són iguals entre elles i presenten un únic canvi de signe en el rang estudiat, entre $h = 0.1$ i $h = 2$. De fet, quan $h = 2$ totes les parts reals dels VAPs són negatives. Així doncs, usant la Definició 2.7 podem dir que per a $h = 2$ el punt hiperbòlic del sistema és un atractor, i la

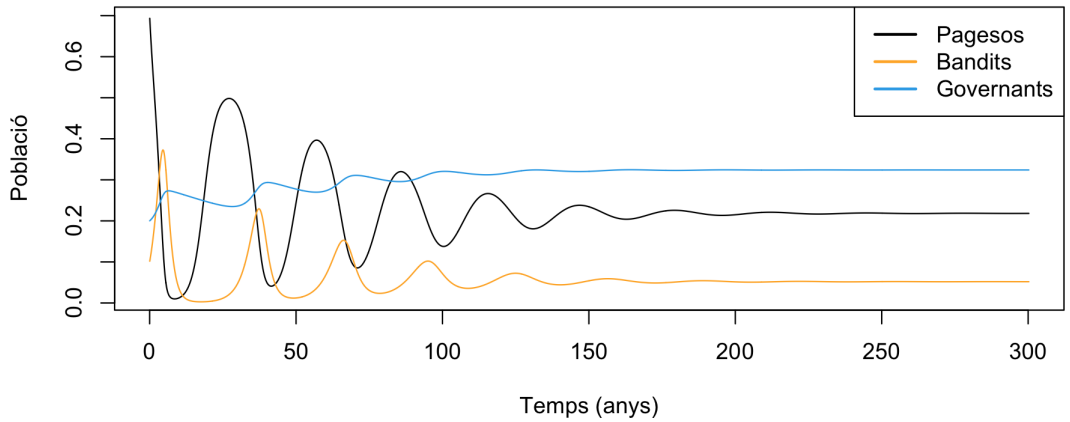

Figura 11: Funcions solucions de les EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2.

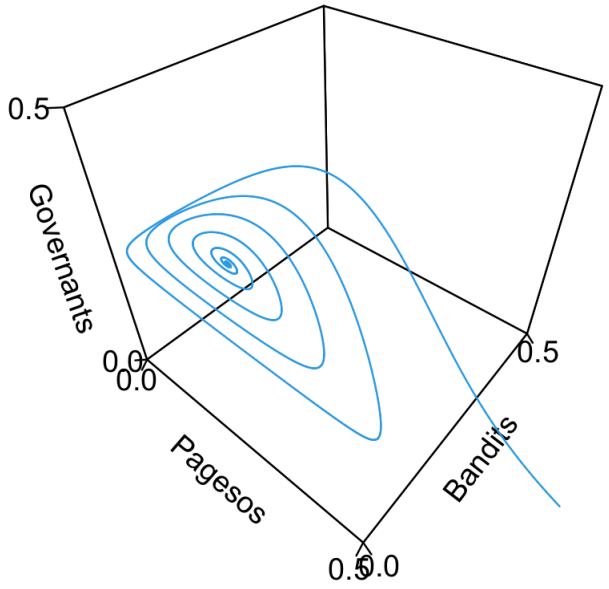

Figura 12: Representació 3D de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2.

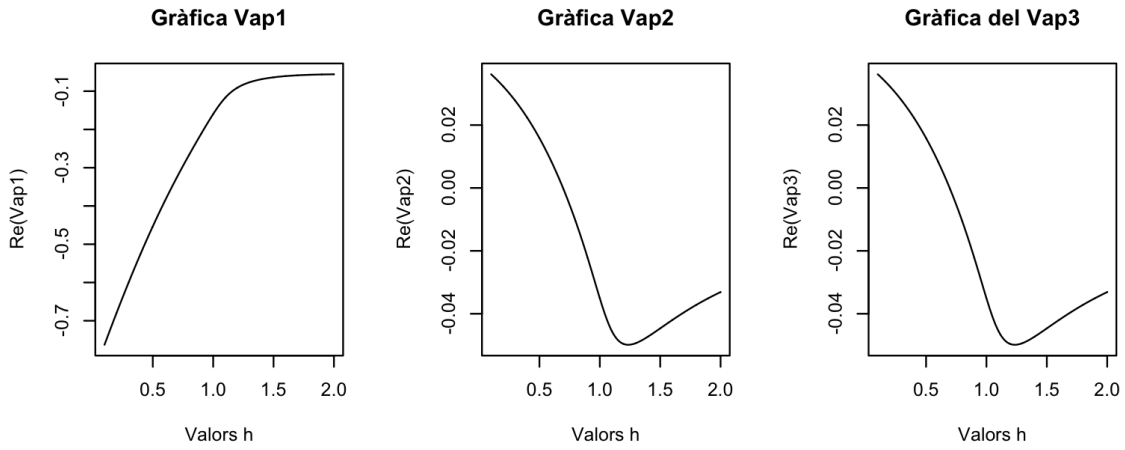

Figura 13: Gràfica que expressa en l'eix y la part real dels tres VAPs (del punt d'equilibri del sistema d'EDOs) en funció del paràmetre h en l'eix x. Els valors han estat obtinguts usant el programa de l'Annex E.

corba solució tendeix a aproximar-se a aquest punt.

De l'estudi anterior dels VAPs, podem argumentar que quan $h \approx 0.695195$ tenim dos VAPs que tenen la part real 0. En aquest cas no podríem classificar el punt d'equilibri, ja que per poder-ho fer les parts reals dels VAPs han de ser diferents de 0. Tot i així, quan $h \in [0.1, 0.69]$, com que tenim una part real dels VAPs negativa i dues de positives, el punt d'equilibri serà un punt de sella (local), segons la definició 2.7. Mentre que si $h \in [0.7, 2]$, com que totes les parts reals dels VAPs són negatives, el punt d'equilibri serà un atractor (local).

Estudiem ara què passa prop del valor crític de $h \approx 0.695195$ en ambdós costats. Considerem primer el cas $h = 0.5$. A les Figures 14 i 15 es veu com l'òrbita periòdica s'ha fet més petita. Quan hem executat el programa de l'Annex D per calcular l'òrbita periòdica ens hem trobat amb el problema que la secció transversal que teníem d'abans $\Sigma = \{(F, B, R) \in \mathbb{R}^3 : B = 0.4\}$ no era realment transversal a l'òrbita i el programa no la trobava. Hem modificat la secció perquè sigui transversal, prenent $\Sigma_1 = \{(F, B, R) \in \mathbb{R}^3 : B = 0.1\}$. En canvi, si considerem ara el cas $h = 0.8$ i mirem la representació gràfica dels càlculs a la Figura 16, es veu com l'òrbita periòdica ha desaparegut i tenim un punt d'equilibri atractor.

Ens resta la incògnita de què passa al valor crític $h = 0.695195$. En principi, el punt d'equilibri no el podríem classificar perquè tindria un VAP (de fet dos) amb part real igual a 0. Si executem el nostre programa de Runge-Kutta per $h = 0.695195$ obtenim la Figura 17. Per la gràfica podríem interpretar que encara hi ha una òrbita periòdica molt a prop del punt d'equilibri $p = (0.43, 0.07, 0.59)$. Tot i així, el nostre programa no ha estat capaç de trobar-la, ni afinant la secció transversal a $\Sigma_2 = \{(F, B, R) \in \mathbb{R}^3 : B = 0.07\}$.

Estudiem ara la variació del paràmetre c, que representa la taxa de saturació dels gobernants envers els bandits, com s'ha vist a la Secció 3.1. Considerem el canvi de valor de c, i l'augmentem de $c = 0.4$ a $c = 0.8$, és a dir, prenem els valors

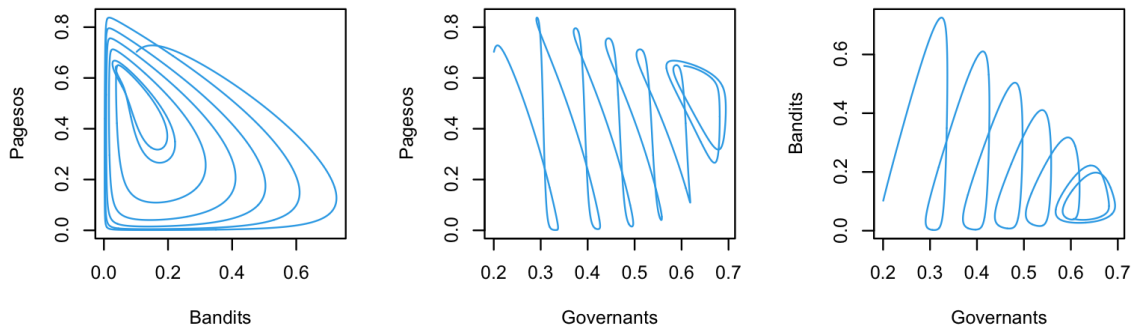

Figura 14: Funcions solucions de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2 canviant $h = 0.5$.

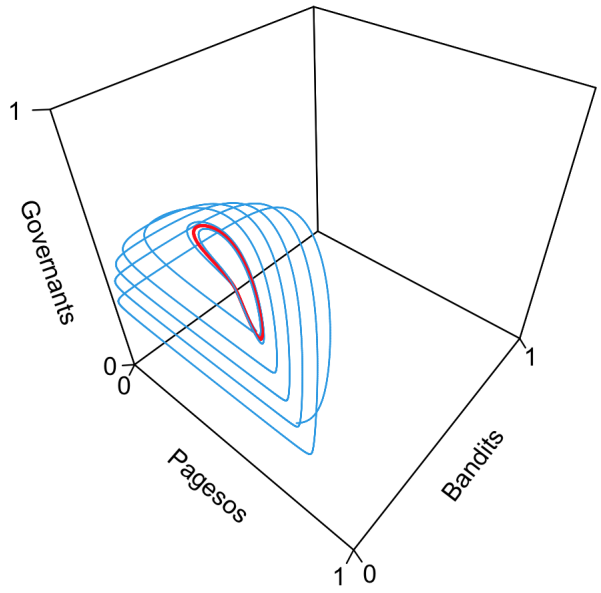

Figura 15: Representació 3D de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2 canviant $h = 0.5$. També hi ha dibuixada en vermell l'òrbita periòdica usant la secció transversal $\Sigma_1 = \{(F, B, R) \in \mathbb{R}^3 : B = 0.1\}$.

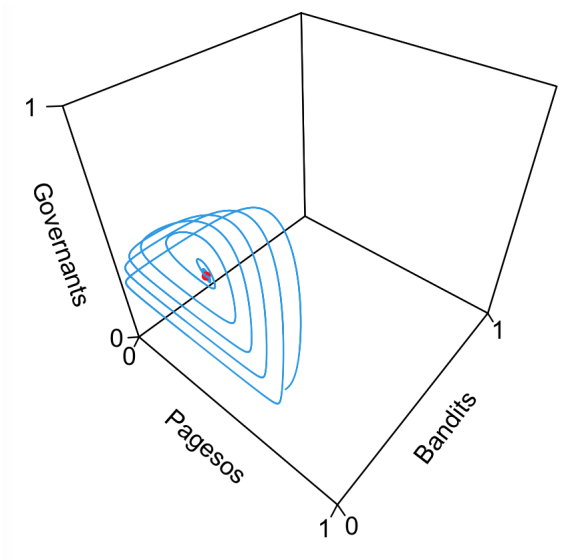

Figura 16: Representació 3D de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2 canviant $h = 0.8$. També està dibuixat en vermell el punt d'equilibri $(0.43, 0.07, 0.56)$.

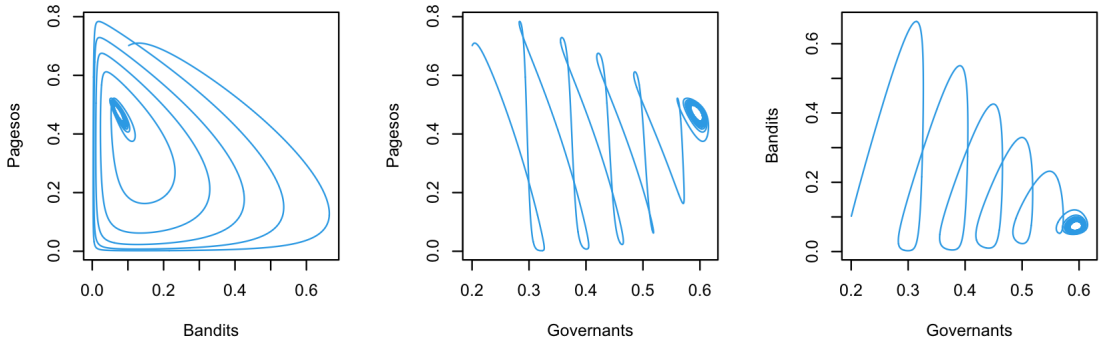

Figura 17: Funcions solucions de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 2 canviant $h = 0.695195$.

$r = 1$	$a = 1$	$m = 0.4$	$e = 1.2$	$c = 0.8$	$h = 0.1$
$K = 1$	$b = 0.17$	$g = 0.009$	$f = 0.1$	$d = 0.42$	

Taula 3: Valors dels paràmetres on hem variat c respecte la Taula inicial 1.

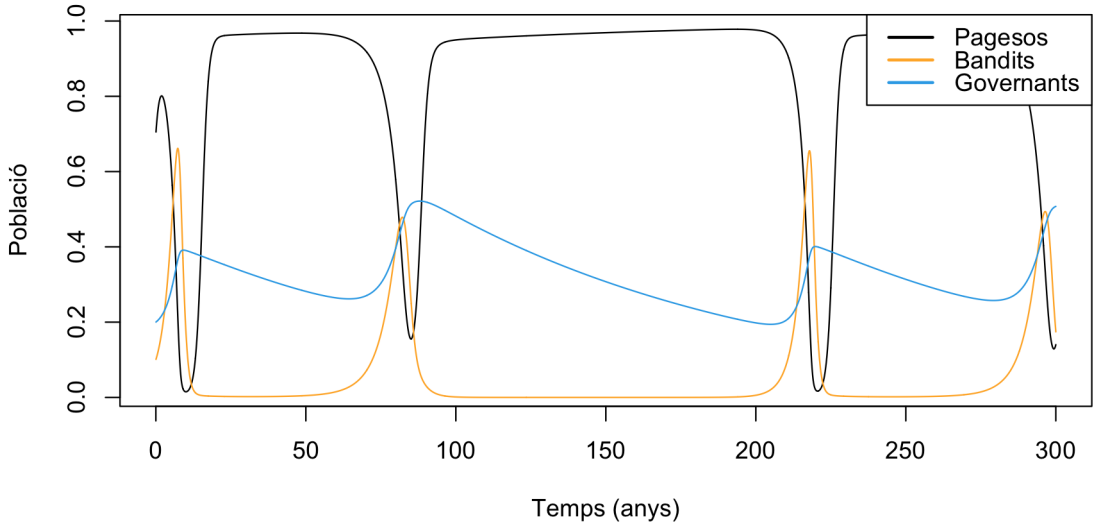

Figura 18: Funcions solucions de les EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 3.

de la Taula 3. Com podem veure a la Figura 18, amb aquest canvi, els períodes abans de la caiguda de la dinastia tarden més a succeir que no pas amb els valors de la primera Taula 1. Era d'esperar que pujant el valor de la $c = 0.8$ obtinguéssim un control dels bandits més llarg. Perquè la c és una constant que influeix en les batalles entre bandits i soldats, i si la pugem estem afavorint als soldats. Fem una representació de la solució en 3D a la Figura 19 on també està representada l'òrbita periòdica. Per tal de comprovar que l'òrbita trobada és atractora, en lloc de calcular Runge-Kutta per 300 anys ho augmentem a 3000 anys i ho representem a la Figura 20 juntament amb l'òrbita periòdica. Gràcies a haver augmentat el temps a 3000 anys podem veure que la solució tendeix a l'òrbita periòdica alternant-se entre un costat i altre de l'òrbita.

En conclusió, per al paràmetre $c = 0.8$ hi ha a l'inici una alternança entre dos valors del període entre dues crisis socials consecutives. Però al cap del temps aquests dos períodes s'igualen fins convergir a l'òrbita periòdica.

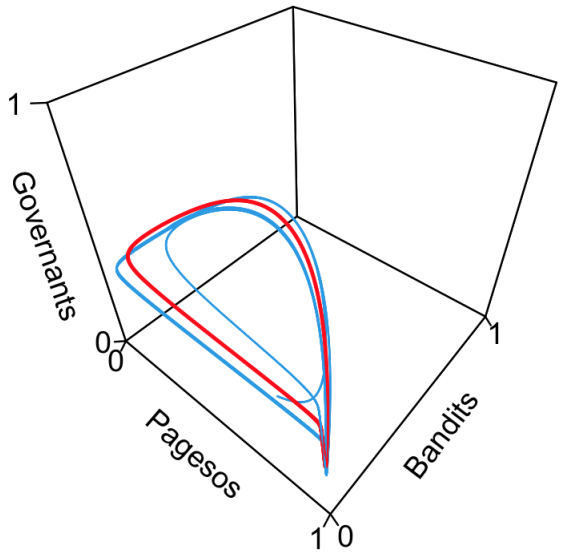

Figura 19: Representació 3D de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 3. També està dibuixada en vermell l'òrbita periòdica.

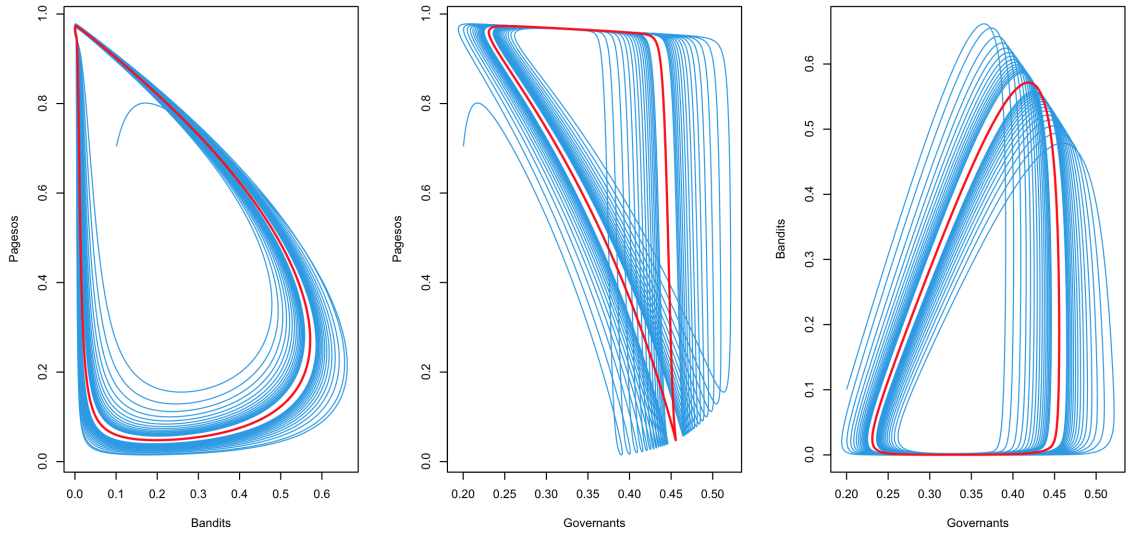

Figura 20: Funcions solucions de la solució del sistema d'EDOs (3.1), (3.2), (3.3) amb valor inicial $(F(0), B(0), R(0)) = (0.7, 0.1, 0.2)$ obtingudes amb la nostra implementació del mètode de Runge-Kutta, per als valors de la Taula 3 i fins a 3000 anys. En vermell està dibuixada l'òrbita periòdica.

4 Conclusions

En aquest treball hem fet una introducció a alguns mètodes d'integració numèrica per a la resolució de problemes de valor inicial en equacions diferencials ordinàries (EDOs). En primer lloc, després d'una breu motivació sobre la necessitat d'usar aquests mètodes, hem presentat els mètodes d'un pas més coneguts, d'Euler, d'Euler modificat, de Heun i de Runge-Kutta d'ordre 4, i els hem comparat en un petit experiment numèric, mostrant, a través d'un exemple d'una EDO de la qual coneixem la seva solució analítica, que el d'ordre més alt aproxima molt millor. D'aquesta manera hem fet evident que l'ordre és una de les propietats fonamentals que caracteritza un mètode d'integració. Després hem demostrat la convergència de mètodes d'integració generals d'ordre p d'un pas h i hem provat que l'error global de la discretització té ordre h^p.

Per tal de mantenir aquest error controlat hem introduït dues maneres d'implementar un control de pas d'integració en un algoritme. A més, hem vist a través d'un exemple la diferència de l'estabilitat numèrica en el cas del mètode de Runge-Kutta d'ordre 4, contraposant la implementació amb punts equidistants amb la que incorpora un control de pas. Això permet concloure que l'aproximació amb el control de pas es manté amb un error petit aprop de la solució, ja que el pas es reescala a cada pas, mentre que la implementació amb punts equidistants per a temps més llargs a cada iteració acumula més error.

Seguidament, de forma breu, hem comentat els mètodes de múltiples passos, que aconsegueixen una precisió alta sense haver d'avaluar moltes vegades la funció de l'equació a cada pas, ja que usen les aproximacions calculades en iteracions anteriors. També hem introduït un tipus d'equacions anomenades *Stiff* o equacions rígides que presenten canvis bruscos de creixement a la solució, generen problemes als mètodes estudiats i es fa necessari per a la seva integració l'ús d'altres mètodes anomenats implícits. Acabem el primer capítol explicant la sensibilitat de les solucions respecte petites variacions de la condició inicial.

D'altra banda, hem fet un concís recordatori d'alguns conceptes de sistemes dinàmics que s'han usat posteriorment en l'estudi qualitatiu del sistema determinat per l'EDO del cas pràctic. També hem explicat l'ús del mètode de Runge-Kutta per a calcular l'aplicació de Poincaré, i els detalls de la implementació del nostre algorisme per al càlcul d'òrbites periòdiques, en cas d'existir.

Per acabar, hem presentat un model demogràfic d'onze paràmetres que pretén modelitzar la interacció entre tres classes socials, pagesos, bandits i governants, i que es va plantejar amb l'objectiu de predir l'apogeu i la caiguda de les dinasties Xineses. Per trobar la solució numèrica del model hem implementat un algoritme de Runge-Kutta d'ordre 4 amb un control de pas. Després d'analitzar la solució i veure la possible existència d'òrbites periòdiques, hem programat un algoritme usant Runge-Kutta que calcula l'aplicació de Poincaré per tal de trobar-les. Així, a més de donar la solució numèrica al problema de valor inicial, hem trobat l'òrbita periòdica a la qual tendeix la trajectòria solució, per a diferents valors del paràmetre h. Hem demostrat que aquest paràmetre h té gran influència en la naturalesa de

l'ω-límit de la solució: o bé és una òrbita periòdica, o bé és un punt atractor. A més, hem analitzat qualitativament el model en el punt d'equilibri per a diferents valors del paràmetre h i hem classificat aquest punt segons punt de sella o punt atractor. Després, hem analitzat un altre paràmetre c i hem vist que aquest no afecta a l'existència de l'òrbita periòdica, encara que sí que influeix en la manera com s'hi aproxima la solució.

Referències

[1] Gerald, C.F., Wheatley, P.O. (1990). *Applied Numerical Analysis* (4a edició). Addison-Wesley publishing company.

[2] Stoer, J., Bulirsh, R. (2002). *Introduction to Numerical Analysis* (3a edició). Springer.

[3] Barnes, B., Fulford, G.R. (2009). *Mathematical Modelling with Case Studies: A Differential Equations Approach Using Maple and MATLAB* (2a edició). CRC Press.

[4] Feichtinger, G., Forst C., Piccardi, C. (1996). A Nonlinear Dynamical Model for the Dynastic Cycle. *Chaos, Solitons and Fractals, 7*, (2), 257-271.

[5] Burden, R.L., Faires, D.J., Burden, A.M. (2017). *Análisis Numérico* (10a edició). CENGAGE Learning.

A Programa de comparació de mètodes d'un pas

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>




/*y = t*e^{-t^2}*/

double runge_kutta(double, double, double);
double heun (double h, double t, double y);
double euler(double, double, double);
double f (double, double);
void f_sol (double);
double euler_mod (double h, double t, double y);

int main (void){
        double t,y, h;
        double e, he, rk, emod;
        FILE *dad;
        dad = fopen("comparacio.txt","w");
        t=0.; y=4.;
        h=0.1;
        e=y; he=y;rk=y;emod=y;

        do{
                fprintf(dad,"%lf,",t);

                fprintf(dad,"%lf,",e);
                e = euler(h,t,e);

                fprintf(dad,"%lf,",he);
                he = heun(h,t,he);

                fprintf(dad,"%lf,",emod);
                emod = heun(h,t,emod);

                fprintf(dad,"%lf\n",rk);
                rk = runge_kutta(h,t,rk);

                t += h;
        }while(t <=4.);

```

```
44          fclose(dad);
45          f_sol(4.);
46          return 0;
47  }
48
49  double euler_mod (double h, double t, double y){
50          double v;
51          v= y+h*f(t,y)/2.;
52          y= y + h*f(t+h/2.,v);
53          return y;
54  }
55
56  double heun (double h, double t, double y){
57          double v;
58          v= y +h*f(t,y);
59          y= y + h*(f(t,y)+ f(t+h, v))/2.;
60          return y;
61  }
62
63  double euler (double h, double t, double y){
64          y= y+ h*f(t,y);
65          return y;
66  }
67
68  double runge_kutta(double h, double t, double y){
69          double k1, k2, k3, k4;
70
71          k1 = h*f(t,y);
72          k2 = h*f(t+h/2.,y+0.5*k1);
73          k3 = h*f(t+h/2.,y+0.5*k2);
74          k4 = h*f(t+h,y+k3);
75
76          y = y + (k1+2*k2+2*k3+k4)/6.;
77
78          return y;
79  }
80
81  double f (double t, double y){
82          double v;
83          if (t==0){
84                  return 4;
85          }
86          v= (y+1)*(t+1)*cos(t*t+2.*t);
87          return v;
88  }
```

```
89
90  void f_sol (double tmax){
91          double v,h,t;
92          FILE *solu;
93          solu=fopen("solucio.txt","w");
94          h=0.01;
95          t=0;
96          do{
97                  v = 5.*exp(sin(t*t+2*t)/2.)-1.;
98                  fprintf(solu,"%lf,%lf\n", t,v);
99                  t += h;
100         }while(t <=tmax);
101         fclose(solu);
102         return;
103 }
```

B Programa de comparació de Runge-Kutta amb punts equidistants o amb control de pas

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>




/*y = t*e^{-t^2}*/

double runge_kutta(double, double, double);
void stepControl (double tmax, double y0);
double f (double, double);
void f_sol (double);

int main (void){
        double t,y, h, tmax;
        double rk;
        FILE *dad;

        tmax=10.;
        dad = fopen("RK.txt","w");
        t=0.; y=4.; h=0.1;
        rk=y;
        fprintf(dad,"%lf,%lf\n",t, rk);
        do{
                rk = runge_kutta(h,t,rk);
                t += h;
                fprintf(dad,"%lf,%lf\n",t, rk);

        }while(t <=tmax);

        fclose(dad);
        f_sol(tmax);
        stepControl(tmax,y);
        return 0;
}

void stepControl (double tmax, double y0){
        double t, epsilon = 1e-5, coef, aproH2, aproH, h;
        FILE *spC;
        spC = fopen("RK_sCon.txt", "w");
```

```
42
43          h =0.1; t=0;
44          fprintf(spC,"%lf, %lf \n",t, y0);
45          do{
46                  aproH =y0;
47                  aproH2 =y0;
48
49                  aproH = runge_kutta(h,t, aproH);
50
51                  aproH2 = runge_kutta(h/2., t, aproH2);
52                  aproH2 = runge_kutta(h/2., t+h/2., aproH2);
53
54                  /*Com usem rungekutta 4 , p=4*/
55                  coef = 2*2*2*2/(2*2*2*2-1);
56                  coef = coef*fabs(aproH-aproH2)/epsilon;
57                  coef = pow( coef, 1/5.);
58
59                  if (coef > 2.) {
60                          h = 2. * h / coef;
61                  }
62                  else {
63                          t += h;
64                          y0 = aproH2;
65                          h = 2. * h / coef;
66
67                          fprintf(spC,"%lf, %lf \n",t, y0);
68                  }
69
70          }while(t <= tmax);
71          fclose(spC);
72          return;
73  }
74
75
76  double runge_kutta(double h, double t, double y){
77          double k1, k2, k3, k4;
78
79          k1 = h*f(t,y);
80          k2 = h*f(t+h/2.,y+0.5*k1);
81          k3 = h*f(t+h/2.,y+0.5*k2);
82          k4 = h*f(t+h,y+k3);
83
84          y = y + (k1+2*k2+2*k3+k4)/6.;
85
86          return y;
```

```
87  }
88
89  double f (double t, double y){
90          double v;
91          if (t==0){
92                  return 4;
93          }
94          v= (y+1)*(t+1)*cos(t*t+2.*t);
95          return v;
96  }
97
98  void f_sol (double tmax){
99          double v,h,t;
100         FILE *solu;
101         solu=fopen("solucio.txt","w");
102         h=0.01;
103         t=0;
104         do{
105                 v = 5.*exp(sin(t*t+2*t)/2.)-1.;
106                 fprintf(solu,"%lf,%lf\n", t,v);
107                 t += h;
108         }while(t <=tmax);
109         fclose(solu);
110         return;
111 }
```

C Programa de Runge-Kutta d'ordre 4

Aquí està el programa de Runge-Kutta d'ordre 4 usant el control de pas.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double r = 1.;
double K = 1.;
double a = 1.;
double b = 0.17;
double h = 0.1;
double e = 1.2;
double d = 0.42;
double m = 0.4;
double c = 0.4;
double f = 0.1;
double g = 0.009;

double f1(double F, double B, double R);
double f2(double F, double B, double R);
double f3(double F, double B, double R);
void runge_kutta(double h, double* aprox);
double norma_2 (double *, int);

int main (void) {
        double F=0.7, B=0.1,R=0.2;
        double aproH[3], aproH2[3], restaH[3];
        double t=0., epsilon = 1e-12, coef, step;
        int iter, rows;
        FILE *outfile;

        outfile = fopen("stepControl.txt", "w");

        step =1.;
        iter = 0;
        rows=0;
        do{
                aproH[0]=F;
                aproH[1]=B;
                aproH[2]=R;

                aproH2[0]=F;
                aproH2[1]=B;
```

```
42                  aproH2[2]=R;
43
44                  runge_kutta(step, aproH);
45
46                  runge_kutta(step/2., aproH2);
47                  runge_kutta(step/2., aproH2);
48
49                  restaH[0] = aproH[0] - aproH2[0];
50                  restaH[1] = aproH[1] - aproH2[1];
51                  restaH[2] = aproH[2] - aproH2[2];
52                  coef = norma_2(restaH,3);
53                  coef = 2*2*2*2*coef/(epsilon*(2*2*2*2-1));
54                  coef = pow( coef, 1/5.);
55
56                  if (coef > 2.) {
57                          step = 2. * step / coef;
58                  }
59                  else {
60                          t += step;
61
62                          F = aproH2[0];
63                          B = aproH2[1];
64                          R = aproH2[2];
65                          fprintf(outfile,"%.4e, %.10e, %.10e,
                            ↪  %.10e \n",t, F, B,R);
66
67                          step = 2. * step / coef;
68                          rows ++;
69                  }
70                  ++iter;
71          }while(t <= 300 && rows < 1e6);
72          printf("Nombre d'iteracions: %d\n", iter);
73          printf("Nombre de files en el fitxer: %d\n", rows);
74          printf("hem arribat fins al temps t = : %.2e\n", t);
75
76          fclose(outfile);
77          return 0;
78  }
79
80  void runge_kutta(double step, double *valors){
81          double F, B, R;
82          double k11, k12, k13;
83          double k21, k22, k23;
84          double k31, k32, k33;
85          double k41, k42, k43;
```

```
86
87          F = valors[0];
88          B = valors[1];
89          R = valors[2];
90
91          k11 = step*f1(F,B,R);
92          k12 = step*f2(F,B,R);
93          k13 = step*f3(F,B,R);
94
95          k21 = step*f1(F+0.5*k11,B+0.5*k12,R+0.5*k13);
96          k22 = step*f2(F+0.5*k11,B+0.5*k12,R+0.5*k13);
97          k23 = step*f3(F+0.5*k11,B+0.5*k12,R+0.5*k13);
98
99          k31 = step*f1(F+0.5*k21,B+0.5*k22,R+0.5*k23);
100         k32 = step*f2(F+0.5*k21,B+0.5*k22,R+0.5*k23);
101         k33 = step*f3(F+0.5*k21,B+0.5*k22,R+0.5*k23);
102
103         k41 = step*f1(F+k31,B+k32,R+k33);
104         k42 = step*f2(F+k31,B+k32,R+k33);
105         k43 = step*f3(F+k31,B+k32,R+k33);
106
107         valors[0] = F + (k11+2*k21+2*k31+k41)/6.;
108         valors[1] = B + (k12+2*k22+2*k32+k42)/6.;
109         valors[2] = R + (k13+2*k23+2*k33+k43)/6.;
110
111         return;
112 }
113
114 double f1(double F, double B, double R){
115         double result;
116         result = r*F*(1.-F/K) - a*F*B/(b+F) - h*F*R;
117         return result;
118 }
119
120 double f2(double F, double B, double R){
121         double result;
122         result = e*a*F*B/(b+F) -m*B -c*B*R/(d+B);
123         return result;
124 }
125
126 double f3(double F, double B, double R){
127         double result;
128         result = f*a*F*B/(b+F) - g*R;
129         return result;
130 }
```

```
131
132 double norma_2 (double *v, int n){
133         double result=0., pot;
134         int i;
135
136         for (i=0; i<n; ++i){
137                 pot = v[i]*v[i];
138                 result += pot;
139         }
140         result = sqrt(result);
141         return result;
142 }
```

D Programa de cerca d'òrbita periòdica

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdbool.h>

double r = 1.;
double K = 1.;
double a = 1.;
double b = 0.17;
double h = 0.1;
double e = 1.2;
double d = 0.42;
double m = 0.4;
double c = 0.4;
double f = 0.1;
double g = 0.009;

double f1(double F, double B, double R);
double f2(double F, double B, double R);
double f3(double F, double B, double R);
void runge_kutta(double h, double* aprox);
double norma_2 (double *, int);
void inverse_M2x2 (double M[2][2]);
void fillMatrx (double M[2][2], double x, double y, double
↪  sigma);
int signe(double v);
int Poincare (double F0, double B0, double R0, double *F, double
↪  *R);
int newton(double F0, double B0);
void OrbitaPeriodica (double F0, double B0, double R0);

double fix_B=0.4;

int main(void){
        double F0, R0, F,R;
        F0=0.7;
        R0=0.2;

        printf("1a crida de Poincare per trobar el punt de
        ↪  tall\n");
        /*Amb les condicions inicials del llibre de referencia
        ↪  busquem el primer tall amb fix_B=0.4 per poder
        ↪  comenÃ§ar amb newton*/
```

```
39          if (Poincare (F0, 0.1, R0, &F, &R)==1){
40                  printf("Ara comenÃ§a el Newton\n");
41                  if (newton(F,R)==-1){
42                          printf("Problema\n");
43                  }
44          }
45          else{
46                  printf("No hem trobat res\n");
47          }
48          return 0;
49  }
50
51  void fillMatrx (double M[2][2], double x, double y, double
    ↪  sigma){
52          double fx,fy;
53          /*Poincare(x+sigma,fix_B,y, &fx,&fy) ens donara
            ↪  P1(x+sigma,y)=fx i P2(x+sigma,y)=fy*/
54          if (Poincare(x+sigma,fix_B,y, &fx,&fy)==-1){
55                  printf("No hem trobat punt a la crida 1 de
                    ↪  Poincare en fillMatrx\n");
56                  return;
57          }
58          M[0][0] = fx;
59          M[1][0] = fy;
60
61          if (Poincare(x,fix_B,y+sigma, &fx,&fy)==-1){
62                  printf("No hem trobat punt a la crida 2 de
                    ↪  Poincare en fillMatrx\n");
63                  return;
64          }
65          M[0][1] = fx;
66          M[1][1] = fy;
67
68          if (Poincare(x,fix_B,y, &fx,&fy)==-1){
69                  printf("No hem trobat punt a la crida 3 de
                    ↪  Poincare en fillMatrx\n");
70                  return;
71          }
72          M[0][0] = (M[0][0] - fx)/sigma -1.;
73          M[0][1] = (M[0][1] - fx)/sigma;
74          M[1][0] = (M[1][0] - fy)/sigma;
75          M[1][1] = (M[1][1] - fy)/sigma -1.;
76          return;
77  }
78
```

```
79  void inverse_M2x2 (double M[2][2]){
80          double det, aux;
81
82          det = M[0][0]*M[1][1] -M[1][0]*M[0][1];
83
84          if (fabs(det) <1e-8){
85                  printf("La matriu Dg no te inversa: det =
                    ↪  %+.4e\n", det);
86          }
87
88          aux = M[0][0];
89          M[0][0] = M[1][1]/det;
90          M[1][1] = aux/det;
91
92          M[1][0] = -M[1][0]/det;
93          M[0][1] = -M[0][1]/det;
94
95          return;
96  }
97
98  int Poincare (double F0, double B0, double R0, double *F, double
    ↪  *R){
99          /* Retornarem 1 si es troba solucio, -1 si no es troba
            ↪  solucio*/
100         int iter, cont, sign[2], num_talls;
101         double aproH[3], aproH2[3], restaH[3], prev_sol[3];
102         double step=1., epsilon = 1e-12, coef;
103         /*Busquem la seccio de poincare al pla B = fix_B*/
104         iter = 0;
105         cont=0;
106         /*Quan el numero de talls al pla fix_B sigui 3, vol dir
            ↪  que ja hem arribat a on voliem.
107         * Necessitem que siguin 3 perque no sempre comencem en
    ↪  B0=fix_B, per aixo sera, en general, la 3a vegada que
    ↪  tallem quan haurem completat una volta.
108         * Si comencem sobre el pla, aleshores hem d'aturar-nos
    ↪  la segona vegada que tallem el pla.*/
109         num_talls=3;
110         if (B0== fix_B){
111                 num_talls=2;
112         }
113
114         do{
115                 if (iter > 1e4){
```

```
116                     printf("Hem passat el limit d'iteracions
                        ↪  a la funcio de Poincare: %d\n",
                        ↪  iter);
117                     return -1;
118             }
119
120             aproH[0]=F0;
121             aproH[1]=B0;
122             aproH[2]=R0;
123
124             aproH2[0]=F0;
125             aproH2[1]=B0;
126             aproH2[2]=R0;
127
128             runge_kutta(step, aproH);
129
130             runge_kutta(step/2., aproH2);
131             runge_kutta(step/2., aproH2);
132
133             /*Com usem rungekutta 4 , p=4*/
134             restaH[0] = aproH[0] - aproH2[0];
135             restaH[1] = aproH[1] - aproH2[1];
136             restaH[2] = aproH[2] - aproH2[2];
137             coef = norma_2(restaH,3);
138             coef = 2*2*2*2*coef/(epsilon*(2*2*2*2-1));
139             coef = pow( coef, 1/5.);
140
141             if (coef > 2.) {
142                     step = 2. * step / coef;
143             }
144             else {
145                     prev_sol[0]=F0;
146                     prev_sol[1]=B0;
147                     prev_sol[2]=R0;
148
149                     F0 = aproH2[0];
150                     B0 = aproH2[1];
151                     R0 = aproH2[2];
152
153                     /*anem guardant els signes en un vector
                        ↪  de 2 components per poder-ho anar
                        ↪  comprovant si hi ha hagut algun
                        ↪  canvi de signe*/
154                     sign[cont%2] = signe(fix_B-B0);
155
```

```
156                     if (cont > 1 && sign[0] != sign[1]){
157                             num_talls --;
158                             printf("CANVI SIGNE F = %+.2le  B
                                ↪  = %+.2le  R = %+.2le\n",
                                ↪  F0,B0,R0);
159                             if (num_talls ==0){
160                                     /*Hem fet una volta.
                                        ↪  Agafem la solucio
                                        ↪  anterior i demanem
                                        ↪  mes precisio per
                                        ↪  estar sobre el
                                        ↪  fix_B*/
161                                     aproH[0]=prev_sol[0];
162                                     aproH[1]=prev_sol[1];
163                                     aproH[2]=prev_sol[2];
164
165                                     int iterator=0;
166                                     do{
167                                             step = fabs(fix_B
                                                ↪  -
                                                ↪  aproH[1])/2.;
168                                             runge_kutta(step,
                                                ↪  aproH);
169                                             ++iterator;
170                                     }while(fabs(fix_B
                                        ↪  -aproH[1])>1e-8);
171
172                                     *F=aproH[0];
173                                     *R=aproH[2];
174                                     printf("Poincare: Punt
                                        ↪  despres d'una volta
                                        ↪  en el pla B=%.2lf amb
                                        ↪  un error de %.4le\n",
                                        ↪  fix_B,
                                        ↪  fabs(aproH[1]-fix_B));
175                                     printf("F = %+.2le  B =
                                        ↪  %+.2le  R = %+.2le\t
                                        ↪  %d\n",
                                        ↪  aproH[0],aproH[1],
                                        ↪  aproH[2], iterator);
176                                     return 1;
177                             }
178                     }
179                     step = 2. * step / coef;
180                     cont++;
```

```
181                 }
182                 ++iter;
183         }while(true);
184         return 1;
185 }
186
187 int newton(double F0, double R0){
188         int iter;
189         double Dg[2][2], nextF, nextR, v[2], F, R, ER;
190         FILE *err;
191         err = fopen("errNewton.txt","w");
192
193         iter=0;
194         do{
195                 if (iter>10){
196                         printf("Hem passat el limit d'iteracions
                            ↪  a la funcio Newton: %d\n", iter);
197                         return -1;
198                 }
199
200                 /*Busquem la F i R cridant a la funcio Poincare,
                    ↪  ja sabem que comenÃ§em des del pla B=
                    ↪  fix_B*/
201                 if (Poincare (F0, fix_B, R0, &F, &R)==-1){
202                         printf("No hem trobat la volta de
                            ↪  Poincare en la funcio de newton\n");
203                         return -1;
204                 }
205                 fillMatrx(Dg,F0,R0,1e-3);
206                 Dg[0][0]*Dg[1][1]-Dg[0][1]*Dg[1][0]);
207                 inverse_M2x2(Dg);
208
209                 nextF = F0 - (Dg[0][0]*(F-F0) + Dg[0][1]*(R-R0));
210                 nextR = R0 - (Dg[1][0]*(F-F0) + Dg[1][1]*(R-R0));
211
212                 /*Calcul del error relatiu*/
213                 v[0] = nextF-F0;
214                 v[1] = nextR-R0;
215                 ER= norma_2(v,2);
216                 v[0] = nextF;
217                 v[1] = nextR;
218                 ER=ER/norma_2(v,2);
219
220                 fprintf(err, "%d %+.3le\n",iter, ER);
221
```

```
222                 if (ER <1e-12){
223                         printf("Hem arribat a un zero amb %d
                            ↪  iteracions per Newton: F=%.4e, R=
                            ↪  %.4e\n", iter, nextF, nextR);
224                         printf("Guardem la orbita en un
                            ↪  fitxer\n");
225                         /*Cridem OrbitaPeriodica, que es la
                            ↪  mateixa funcio que poincare pero
                            ↪  guarda els valors en un fitxer.*/
226                         OrbitaPeriodica (nextF, fix_B, nextR);
227                         return 1;
228                 }
229                 F0 = nextF;
230                 R0 = nextR;
231                 ++iter;
232         }while(true);
233         return 1;
234 }
235
236 int signe(double v) {
237         if (v<0){
238                 return -1;
239         }
240         return 1;
241 }
242
243 void OrbitaPeriodica (double F0, double B0, double R0){
244         int iter, cont, sign[2], num_talls=2;
245         double aproH[3], aproH2[3], restaH[3];
246         double step=1., epsilon = 1e-12, coef;
247         FILE *orbita;
248
249         orbita = fopen("newtonOrbit.txt", "w");
250         iter = 0;
251         cont=0;
252         do{
253                 aproH[0]=F0;
254                 aproH[1]=B0;
255                 aproH[2]=R0;
256
257                 aproH2[0]=F0;
258                 aproH2[1]=B0;
259                 aproH2[2]=R0;
260
261                 runge_kutta(step, aproH);
```

```
262
263                 runge_kutta(step/2., aproH2);
264                 runge_kutta(step/2., aproH2);
265
266                 /*Com usem rungekutta 4 , p=4*/
267                 restaH[0] = aproH[0] - aproH2[0];
268                 restaH[1] = aproH[1] - aproH2[1];
269                 restaH[2] = aproH[2] - aproH2[2];
270                 coef = norma_2(restaH,3);
271                 coef = 2*2*2*2*coef/(epsilon*(2*2*2*2-1));
272                 coef = pow( coef, 1/5.);
273
274                 if (coef > 2.) {
275                         step = 2. * step / coef;
276                 }
277                 else {
278                         F0 = aproH2[0];
279                         B0 = aproH2[1];
280                         R0 = aproH2[2];
281                         fprintf(orbita, "%+.8le, %+.8le,
                            ↪  %+.8le\n", F0,B0,R0);
282
283                         sign[cont%2] = signe(fix_B-B0);
284
285                         if (cont > 1 && sign[0] != sign[1]){
286                                 num_talls --;
287                                 if (num_talls ==0){
288                                         /*Hem donat una volta*/
289                                         fclose(orbita);
290                                         return;
291                                 }
292                         }
293                         step = 2. * step / coef;
294                         cont++;
295                 }
296                 ++iter;
297         }while(iter < 1e4);
298         printf("Ens hem passat d'iteracions en la funcio
            ↪  OrbitaPeriodica\n");
299         fclose(orbita);
300         return;
301 }
302
303 void runge_kutta(double step, double *valors){
304         double F, B, R;
```

```
305         double k11, k12, k13;
306         double k21, k22, k23;
307         double k31, k32, k33;
308         double k41, k42, k43;
309 
310         F = valors[0];
311         B = valors[1];
312         R = valors[2];
313 
314         k11 = step*f1(F,B,R);
315         k12 = step*f2(F,B,R);
316         k13 = step*f3(F,B,R);
317 
318         k21 = step*f1(F+0.5*k11,B+0.5*k12,R+0.5*k13);
319         k22 = step*f2(F+0.5*k11,B+0.5*k12,R+0.5*k13);
320         k23 = step*f3(F+0.5*k11,B+0.5*k12,R+0.5*k13);
321 
322         k31 = step*f1(F+0.5*k21,B+0.5*k22,R+0.5*k23);
323         k32 = step*f2(F+0.5*k21,B+0.5*k22,R+0.5*k23);
324         k33 = step*f3(F+0.5*k21,B+0.5*k22,R+0.5*k23);
325 
326         k41 = step*f1(F+k31,B+k32,R+k33);
327         k42 = step*f2(F+k31,B+k32,R+k33);
328         k43 = step*f3(F+k31,B+k32,R+k33);
329 
330         valors[0] = F + (k11+2*k21+2*k31+k41)/6.;
331         valors[1] = B + (k12+2*k22+2*k32+k42)/6.;
332         valors[2] = R + (k13+2*k23+2*k33+k43)/6.;
333 
334         return;
335 }
336 
337 double f1(double F, double B, double R){
338         double result;
339         result = r*F*(1.-F/K) - a*F*B/(b+F) - h*F*R;
340         return result;
341 }
342 
343 double f2(double F, double B, double R){
344         double result;
345         result = e*a*F*B/(b+F) -m*B -c*B*R/(d+B);
346         return result;
347 }
348 
349 double f3(double F, double B, double R){
```

```
350         double result;
351         result = f*a*F*B/(b+F) - g*R;
352         return result;
353 }
354
355 double norma_2 (double *v, int n){
356         double result=0., pot;
357         int i;
358
359         for (i=0; i<n; ++i){
360                 pot = v[i]*v[i];
361                 result += pot;
362         }
363         result = sqrt(result);
364         return result;
365 }
```

E Programa de càlcul de punt d'equilibri i de la part real dels seus VAPs

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define pi acos(-1)

double r = 1.;
double K = 1.;
double a = 1.;
double b = 0.17;
double h = 0.1;
double e = 1.2;
double d = 0.42;
double m = 0.4;
double c = 0.4;
double f = 0.1;
double g = 0.009;

/*En aquest programa el que farem es analitzar el comportament
 ↪  de la part real dels vaps en els punts d'equilibri. Perque
 ↪  sabem que si totes les parts reals son negatives aleshores
 ↪  el punt d'equilibri es un atractor, i si totes son
 ↪  positives es un repulsor.
Per trobar els punts d'equilibri hem de buscar zeros de la
 ↪  funcio Psi que hem obtingut  imposant que les derivades de
 ↪  F, B i R (respecte el temps) siguin 0 al mateix temps.
La funcio aqui representada sera Psi(F), es a dir que dependra
 ↪  de F, farem les altres funcions (funB, funR) per tal que un
 ↪  cop obtingut el valor del de F, obtenir el de B i el de
 ↪  R.*/

double funPsi (double );
double der_funPsi(double );
double funB (double );
double funR (double );
void inicialitzar(double M[3][3], double*);
void poly_car (double M[3][3], double *);
void cardano (double *, double *);
int signe(double);
```

```c
void newton_Psi(double *);
void main_exe (double*);
double ava_pol (double, double *, int);
void printScreen(double *pe, double M[3][3], double *polcar,
 ↪  double *vaps);


int main(void){
        char var;

        printf("Quina variable (r, a, b, h, e, d, m, c, f, g)
        ↪  voldras fer variar?\n");
        scanf("%c", &var);

        if (var == 'r'){main_exe (&r);}
        else if (var == 'a'){main_exe (&a);}
        else if (var == 'b'){main_exe (&b);}
        else if (var == 'h'){main_exe (&h);}
        else if (var == 'e'){main_exe (&e);}
        else if (var == 'd'){main_exe (&d);}
        else if (var == 'm'){main_exe (&m);}
        else if (var == 'c'){main_exe (&c);}
        else if (var == 'f'){main_exe (&f);}
        else if (var == 'g'){main_exe (&g);}
        else {
                printf("No s'ha reconegut bÃ© la variable
                ↪  donada.\n");
        }
        return 0;
}

void main_exe (double *cnst){
        double M[3][3], pol[4], punt_equilibri[3], vaps[3], step,
        ↪  i1, i2;
        FILE *out;
        char nom[30];

        /*Demanem el step i el interval del parametre*/
        printf("Quin serÃ  l'agument a cada pas?\n I l'interval
        ↪  on es mourÃ ? (recorda que han de ser positives)\n");
        scanf("%lf %lf %lf", &step, &i1, &i2);

        /*Obrim el fitxer on guardarem les dades*/
        printf("Com vols que es digui el fitxer on guardarem les
        ↪  dades dels vaps?\n");
```

```
72          scanf("%s", &nom);
73          out = fopen(nom, "w");
74          if (out == NULL){printf("Error en obrir el
            ↪  fitxer\n");return;}
75
76          *cnst = i1;
77          do{
78          /*Busquem el punt d'equilibri amb Newton*/
79                  newton_Psi(punt_equilibri);
80          /*Inicialitzem la matriu, el que fem es plenar-la*/
81                  inicialitzar(M, punt_equilibri);
82          /*Calculem el polinomi caracteristic*/
83                  poly_car(M,pol);
84          /*Resolem el polinomi caracteritic per trobar els
            ↪  vaps*/
85                  cardano(pol,vaps);
86          /*En el fitxer printem el parametre que modifiquem,
            ↪  part_real(lamba1), part_real(lamba2),
            ↪  part_real(lambda3)*/
87                  fprintf(out, "%lf, %e, %e, %e\n", *cnst, vaps[0],
                    ↪  vaps[1], vaps[2]);
88                  /*Si volem printar per pantalla tota la
                    ↪  informacio trobada descomenta el seguent
                    ↪  codi*/
89                  /*printScreen(punt_equilibri, M, pol, vaps);*/
90                  *cnst = *cnst+ step;
91          }while(*cnst <= i2);
92
93          fclose(out);
94          return;
95  }
96
97  void newton_Psi(double *solucio){
98          int iter;
99          double nextF, F0, ER;
100
101         /*Per comencar amb un valor prenem F0=0.5, ja que F pot
            ↪  valdre (0,K) i K=1*/
102         iter =0;
103         F0=0.5;
104         do{
105                 nextF = F0 - funPsi(F0)/der_funPsi(F0);
106                 ER = fabs(nextF -F0)/fabs(nextF);
107
108                 if (ER < 1e-6){
```

```
109                         /*printf("Hem acabat Newton perque hem
                            ↪  trobat una solucio amb un error
                            ↪  relatiu de %.2le\n", ER);*/
110                         solucio[0]= nextF;
111                         solucio[1]= funB(nextF);
112                         solucio[2]= funR(nextF);
113                         /*printf("Punt d'equilibri:\n
                            ↪  F,B,R=(%lf,  %lf,  %lf)\n",
                            ↪  solucio[0], solucio[1],
                            ↪  solucio[2]);*/
114                         return;
115                 }
116                 F0 = nextF;
117                 ++iter;
118         }while(iter < 100);
119         printf("Hem passat el lÃmit d'iteracions de newton\n");
120         return;
121 }
122
123 void inicialitzar(double M[3][3], double *v){
124         double F, B, R, den1, den2;
125         F=v[0];
126         B=v[1];
127         R=v[2];
128         den1 = b+F;
129         den2= d+B;
130
131         M[0][0]= r-2*r*F/K -a*b*B/(den1*den1) -h*R;
132         M[0][1]= -a*F/den1;
133         M[0][2]= -h*F;
134         M[1][0]= a*b*e*B/(den1*den1);
135         M[1][1]= e*a*F/den1 -m -c*d*R/(den2*den2);
136         M[1][2]= -c*B/den2;
137         M[2][0]= a*b*f*B/(den1*den1);
138         M[2][1]= f*a*F/den1;
139         M[2][2]= -g;
140         return;
141 }
142
143 double funR (double F){
144         double valor;
145         valor = r*f*F*(1-F/K)/(g+h*f*F);
146         return valor;
147 }
148
```

```
149 double funB (double F){
150         double valor;
151         valor = r*(b+F)*(1-F/K)/(a+a*h*f*F/g);
152         return valor;
153 }
154
155 double funPsi (double F){
156         double avalu;
157         avalu = e*a*F/(b+F) -m -c*r*f*a*F*(1-F/K)/(d*a*g +
          ↪  d*h*f*a*F + r*g*(b+F)*(1-F/K));
158         return avalu;
159 }
160
161 double der_funPsi(double F){
162         double avalu, den;
163         den = d*a*g + d*h*f*a*F + r*g*(b+F)*(1-F/K);
164         avalu = e*a*b/((b+F)*(b+F));
165         avalu = avalu - c*r*f*a*(d*a*g+r*g*b-
          ↪  F*(2*d*a*g+d*h*f*a*F+2*r*g*b-r*g*b*F/K)/K)/
          ↪  (den*den);
166         if (fabs(avalu) <1e-8){
167                 printf("La derivada de Psi Ã©s 0!\n");
168         }
169         return avalu;
170 }
171
172 void poly_car (double M[3][3], double *pol){
173         /*Entenem que M es una matriu 3x3, i per tant el pol
          ↪  sera d'ordre 3, es a dir de dimensio 4. La posicio 0
          ↪  sera el terme independent*/
174         pol[0] = M[0][0]*M[1][1]*M[2][2] +
          ↪  M[1][0]*M[2][1]*M[0][2] +M[0][1]*M[1][2]*M[2][0]
          ↪  -M[0][2]*M[2][0]*M[1][1] - M[2][1]*M[1][2]*M[0][0]
          ↪  -M[1][0]*M[0][2]*M[2][2];
175         pol[1] = -M[1][1]*M[2][2] - M[0][0]*(M[1][1]+M[2][2]) +
          ↪  M[0][2]*M[2][0] + M[2][1]*M[1][2] + M[1][0]*M[0][1];
176         pol[2] = M[0][0]+M[1][1]+M[2][2];
177         pol[3] = -1.;
178         return;
179 }
180
181 void cardano (double *pol, double *arrel){
182         double p, q, discr, u, v, delta, canv, aux;
183         int s;
184
```

```
185         p= pol[1]/pol[3] - pol[2]*pol[2]/(3*pol[3]*pol[3]);
186         q= 2*pol[2]*pol[2]*pol[2]/(27*pol[3]*pol[3]*pol[3]) -
            ↪  pol[2]*pol[1]/(3*pol[3]*pol[3]) + pol[0]/pol[3];
187
188         discr = q*q +4*p*p*p/27.;
189         canv = pol[2]/(pol[3]*3.);
190
191         /*Si el discriminant Ã@s zero*/
192         if (fabs(discr) < 1e-10){
193                 /*Potser es una arrel triple 0*/
194                 if (fabs(p) < 1e-8 && fabs(q) < 1e-10){
195                         arrel[0]=0.;
196                         arrel[1]=0.;
197                         arrel[2]=0.;
198                 }
199                 aux = -q/2.;
200                 s = signe(aux);
201                 arrel[0] = 2.*s*pow(s*aux,1/3.) -canv;
202                 arrel[1] = -1.*s*pow(s*aux,1/3.) - canv;
203                 arrel[2] = arrel[1];
204         }
205         else if (discr <0) {
206                 delta = acos((3*q/(2*p))*sqrt(-3./p));
207                 arrel[0] =2*sqrt(-p/3.)*cos(delta/3.) -canv;
208                 arrel[1]
                    ↪  =2*sqrt(-p/3.)*cos((delta+2*pi)/3.)-canv;
209                 arrel[2]
                    ↪  =2*sqrt(-p/3.)*cos((delta+4*pi)/3.)-canv;
210         }
211         else if (discr > 0) {
212                 aux = (-q+sqrt(discr))/2.;
213                 s= signe(aux);
214                 u= s*pow(s*aux, 1/3.);
215                 aux= (-q-sqrt(discr))/2.;
216                 s= signe(aux);
217                 v= s*pow(s*aux, 1/3.);
218                 /*Com que el que volem veure es si es atractor o
                    ↪  repulsor nomes ens interesa la part real*/
219                 arrel[0] = u+v -canv;
220                 arrel[1] = -(u+v)/2.-canv;
221                 arrel[2] = -(u+v)/2.-canv;
222         }
223         return;
224 }
225
```

```
226 int signe(double v) {
227         if (v<0){
228                 return -1;
229         }
230         return 1;
231 }
232
233 double ava_pol (double x, double *pol, int grau){
234         int i;
235         double im =0., expo=1.;
236
237         for (i=0;i <= grau; ++i){
238                 im += pol[i] * expo;
239                 expo *= x;
240         }
241         return im;
242 }
243
244 void printScreen(double *pe, double M[3][3], double *polcar,
    ↪  double *vaps){
245         int i, j;
246         printf("La matriu pel punt d'equilibri (%lf,%lf,%lf)
         ↪  es:\n", pe[0],pe[1],pe[2]);
247         for (i=0; i < 3; ++i){for (j=0; j< 3; ++j){printf("%+lf
         ↪  %3c", M[i][j], ' ');}printf("\n");}
248
249         printf("El polinomi caracteristic es:\n");
250         for(i=0; i <4; ++i){printf("%+.4lf x^%d ", polcar[3-i],
         ↪  3-i);}printf("\n");
251
252         printf("VAPS: ");
253         for (i=0; i< 3; ++i){printf("%.4lf ",
         ↪  vaps[i]);}printf("\n");
254         return;
255 }
```

F Comandes d'R

```
############## Antoni Pech Alberich ##############

#Obrim la llibreria per poder usar-la, la primera per llegir fitxers, la segona per fer grafiques en 3D
library(readr)
library(plot3D)

############## Obrim els fitxers que volem ##############
#Fitxer on hem guardat els errors relatius del metode de Newton quan s'ha calculat la orbita periodica
ERNewton <- read_csv("/Users/MariaAlberich/Dropbox/LINUX/TFG programes/errNewton.txt", skip =0,col_names=FALSE)


funPsi <- read_csv("/Users/MariaAlberich/Dropbox/LINUX/TFG programes/funPsi.txt", skip =0,col_names=FALSE)

#Fitxer on hem guardat l'orbita trobada amb el metode de Newton (Poincare)
NewOrb <- read_csv("/Users/MariaAlberich/Dropbox/LINUX/TFG programes/newtonOrbit.txt", skip =0,col_names=FALSE)

#Fitxer on hem guardat les corbes solucions t, F, B, R, usant l'step control
spC <- read_csv("/Users/MariaAlberich/Dropbox/LINUX/TFG programes/stepControl.txt", skip =0,col_names=FALSE)

#Fitxer on guardem els valors reals dels VAPS per al parametre canviat
vap <- read_csv("/Users/MariaAlberich/Dropbox/LINUX/TFG programes/vaps.txt", skip =0,col_names=FALSE)


########Comandes per dibuixar les grafiques ########
#Conjunt d'instruccions per dibuixar temps-Farmers, temps-Bandits, temps-Rulers en grafiques diferents
dades <- spC
#Pots canviar el nom del fitxer per executar-ho amb un altre.
op <- par(mfrow=c(1,3))
par(mfrow=c(1,3))
matplot(dades$X1, dades$X2,type="l", xlab="Temps (anys)",ylab="Pagesos", main=paste("GrÃ fica dels Pagesos"))
```

```r
plot(dades$X1, dades$X3, type="l",xlab="Temps (anys)",ylab="Bandits", main=paste("GrÃ fica dels Bandits"))
plot(dades$X1, dades$X4, type="l", xlab="Temps (anys)",ylab="Governants", main=paste("GrÃ fica dels Governants"))
par(op)

#Conjunt d'instruccions per dibuixar temps-Farmers, temps-Bandits, temps-Rulers en un mateix eix
dades <- spC

lines(dades$X1, dades$X3, type="l", col="orange")
lines(dades$X1, dades$X4, type="l",col=4)
legend ("topright", legend=c("Pagesos","Bandits","Governants"), col = c(1,"orange",4), lwd=2, xpd =TRUE, bg="white")

#Conjunt de comandes per dibuixar totes les combinacions possibles de (Farmers, Bandits, Rulers)
# 1 es el temps, 2 son els Farmers, 3 son els Bandits, 4 son els Rulers, enlloc de posar dades£X2 == dades[,2]#
dades <- spC
par(mfrow=c(3,3))
matplot(dades$X2, dades$X2,type="l", xlab="Pagesos",ylab="Pagesos")
matplot(dades$X3, dades$X2,type="l", xlab="Bandits",ylab="Pagesos")
matplot(dades$X4, dades$X2, type="l",xlab="Governants",ylab="Pagesos")

matplot(dades$X2, dades$X3, type="l", xlab="Pagesos",ylab="Bandits")
matplot(dades$X3, dades$X3,type="l", xlab="Bandits",ylab="Bandits")
matplot(dades$X4, dades$X3, type="l", xlab="Governants",ylab="Bandits")

matplot(dades$X2, dades$X4,type="l", xlab="Pagesos",ylab="Governants")
matplot(dades$X3, dades$X4, type="l",xlab="Bandits",ylab="Governants")
matplot(dades$X4, dades$X4, type="l", xlab="Governants",ylab="Governants")
par(op)

########## Dibuixar en 3D ##########
```

```
# Usem xlim=c(0,1), ylim=c(0,1), zlim=c(0,1) per fixar els eixos de 0, 1 podriem canviar els valors perque les dades s'ajustessin i sortis una grafica que s'entengues mes.
dades <- spC
lines3D(dades$X2, dades$X3, dades$X4, ticktype="detailed", nticks=2, xlab= "Pagesos", ylab="Bandits", zlab="Governants", col=4, xlim=c(0,1), ylim=c(0,1), zlim=c(0,1))

#Canviem els eixos
lines3D(dades$X2, dades$X4, dades$X3, ticktype="detailed", nticks=2, xlab= "Pagesos", ylab="Governants", zlab="Bandits", col =4, xlim=c(0,1), ylim=c(0,1), zlim=c(0,1))

#Per dibuixar la Orbita periodica per Newton-Poincare
dades1 <- NewOrb
#Per dibuixar-la sobre la solucio de spC
lines3D(dades1$X1, dades1$X2, dades1$X3,add=TRUE,lwd=2, col="red", xlim=c(0,1), ylim=c(0,1), zlim=c(0,1))

#Per dibuixar la orbita sola
lines3D(dades1$X1, dades1$X2, dades1$X3, ticktype="detailed", nticks=2, xlab= "Pagesos", ylab="Bandits", zlab="Governants", col="red", xlim=c(0,1), ylim=c(0,1), zlim=c(0,1))


### Comandes per analitzar els vaps del parametre
fitx <- vap

#Dibuixem la grafica del valor real dels tres vaps, cada un en una grafica diferent
op <- par(mfrow=c(1,3))
par(mfrow=c(1,3))
matplot(fitx$X1, fitx$X2,type="l", xlab="Valors h",ylab="Vap real", main=paste("GrÃ fica Vap1"))
matplot(fitx$X1, fitx$X3, type="l",xlab="Valors h",ylab="Vap imag1", main=paste("GrÃ fica Vap2"))
matplot(fitx$X1, fitx$X4, type="l", xlab="Valors h",ylab="Vap imag2", main=paste("GrÃ fica del Vap3"))
par(op)
```