UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

Mètodes de Runge-Kutta

Estudi teòric i aplicació pràtica

Autor: Martí Rubio Estrada

Director: Dr. Àngel Jorba
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 19 de gener de 2020

Resum

Els mètodes de Runge-Kutta ens ofereixen una manera de resoldre aproximadament equacions diferencials ordinàries amb un algoritme iteratiu. En aquest treball estudiarem aquests mètodes de manera teòrica, i implementarem un integrador numèric en C++ basat en els valors trobats per Jim Verner [1] l'any 1978. També programarem una aritmètica de sèries per a poder usar el mètode de la parametrització i així trobar les equacions de les varietats estables i inestables al problema del pèndol pertorbat.

Resumen

Los métodos de Runge-Kutta nos ofrecen una manera de resolver aproximadamente ecuaciones diferenciales ordinarias con un algoritmo iterativo. En este trabajo estudiaremos estos métodos de manera teórica, y a su vez implementaremos un integrador numérico en C++ basado en los valores encontrados por Jim Verner [1] en 1978. También programaremos una aritmética de séries para poder usar el método de la parametrización con tal de encontrar las ecuaciones de las variedades estables e inestables en el problema del péndulo perturbado.

Abstract

The Runge-Kutta methods provide a way to approximately solve Ordinary Differential Equations in an iterative algorithm. In this document we will study the Runge-Kutta methods from a theoretical point of view, and we will implement a numerical integrator in C++ using the values found by Jim Verner [1] in 1978. We will also program a jet series arithmetic to be able to find the equations of the stable and unstable varieties in the disturbed pendulum problem.

2010 Mathematics Subject Classification. 65L06, 70K42

Agraïments

Voldria començar aquesta secció d'agraïments donant les gràcies al tutor d'aquest treball, el doctor Àngel Jorba, el qual va donar la idea inicial que va acabar essent aquest projecte ja acabat. Li vull agrair la capacitat didàctica i la bona manera com ha sabut gestionar els meus moments de pèrdua dins del mar de codi i fórmules que és aquest treball. Una altra gran part dels agraïments van cap a la doctora Anna Puig, la qual va donar un nivell de suport altíssim a la part més informàtica del treball. Va ser imprescindible la seva ajuda a l'hora de reorganitzar i simplificar el codi del treball per a donar-li una estructura coherent.

Fora ja dels agraïments més tècnics, vull començar agraint als meus pares el seu suport emocional i les seves cares d'aprovació mentre explicava coses que els eren completament alienes. L'esforç de comprensió que han fet aquests mesos ha estat impagable. També vull agrair a la meva germana tota la part estètica del treball, la qual no va deixar de donar-me consells i d'aportar la seva visió més gràfica al treball. Si alguna imatge en aquest treball és estèticament maca, és gràcies a ella.

Per últim, també vull agrair als meus amics —el Xavi, l'Aleix i la Marina— les seves aportacions constants, les seves recerques de la paraula més adequada per a cada frase i els seus cafès de descans després d'hores fent feina sense aturador a les granges. A tots tres, gràcies.

Índex

Índex de figures

Part I
Introducció

1 Sobre el projecte

1.1 El treball

Els mètodes de Runge-Kutta ens ofereixen un algoritme d'integració numèrica d'equacions diferencials ordinàries de manera iterativa. En aquest treball volem estudiar el funcionament i els errors que ens poden donar de manera teòrica. Un cop tinguem aquest estudi teòric fet sobre els mètodes, en triarem un —el mètode explícit de Runge-Kutta-Verner 7(8)— per a programar-lo en C++ i així poder usar-lo per a calcular òrbites de punts.

Una altra tasca d'aquest projecte serà aprendre a programar de manera efectiva una aritmètica de sèries (o aritmètica de jets) per a ser capaços de trobar parametritzacions de varietats lineals. Això ho farem gràcies al mètode de la parametrització, que també és un algoritme que haurem d'estudiar de manera teòrica i programar posteriorment.

1.2 Estructura de la Memòria

La memòria està estructurada en els següents blocs: en primer lloc tenim els Objectius, en els quals exposarem amb detall què volem aconseguir en aquest treball. Després tenim l'apartat sobre l'Equació del pèndol pertorbat, en el qual estudiem de manera teòrica aquesta equació diferencial basant-nos en el que sabem del pèndol clàssic. En aquesta secció també demostrem l'existència d'òrbites periòdiques, fet que ens cal per a les properes seccions.

Seguint endavant tenim la secció més extensa, on estudiem amb detall els mètodes d'integració numèrica i, en particular, els mètodes de Runge-Kutta, els quals són la base d'aquest treball. Després d'una introducció explicant aquests mètodes, ens dediquem a calcular els mètodes més senzills, els de dues i tres etapes de manera explícita. Per últim comentem la variant del control de pas, la qual ens permet integrar cada pas amb un error desitjat.

La següent secció és la del mètode de la parametrització, en la qual comentem aquest algoritme que ens permetrà calcular les varietats lineals, juntament amb l'ajuda de la següent secció: l'aritmètica de sèries. En aquesta darrera mostrem com farem els càlculs entre polinomis en C++.

Com a última secció del cos del treball tenim la implementació en C++, en la qual comentem en detall els arxius i el diagrama de classes del codi, així com els patrons de programació usats i la complexitat teòrica d'aquest. Finalment tenim la secció de simulacions i resultats en la que validem i apliquem tota la teoria estudiada al problema del pèndol pertorbat.

Per últim tenim la secció de conclusions, on ens plantegem si tot el que ens havíem proposat als objectius ho hem acomplert. Si mirem als annexos, tenim la taula que conté els valors del mètode que hem triat, el Runge-Kutta-Verner 7(8). També trobem una petita guia de l'ús del codi, així com la guia d'on es troba i com es genera la documentació automàtica amb l'eina *Doxygen*.

2 Objectius

En aquest treball tenim dos tipus d'objectius sobre el contingut. En primer lloc, tenim uns objectius teòrics dividits en dues parts: estudi dels mètodes de Runge-Kutta i estudi del problema del pèndol pertorbat. Volem demostrar que existeixen punts fixos per a valors d'irregularitat propers a zero.

Aquest problema és especialment interessant d'estudiar, primer de tot, pel fet que l'equació respon a un procés físic que, més o menys, ens podem imaginar. Per altra banda, és interessant pel fet que té un cas molt simple (el cas del pèndol clàssic) i ens és especialment fàcil de treballar, ja que en el problema general la solució es troba fora de l'abast d'aquest treball.

Quant a objectius pràctics, el primer de tots i el més important és programar un integrador numèric que usi el mètode de Runge-Kutta-Verner 7(8). L'altre propòsit pràctic serà aplicar aquest integrador numèric al nostre problema, que ens hauria de permetre trobar el punt fix de l'equació diferencial si existeix. Una meta important és la programació d'una aritmètica de sèries. Aquesta ens servirà per a trobar una aproximació polinòmica d'unes corbes en particular: les varietats lineals del punt fix.

Per acabar, i en l'àmbit informàtic ens proposem saber resoldre dubtes de manera autònoma, així com poder organitzar el codi d'una manera correcta.

Part II

Cos del treball

1 Equació del pèndol pertorbat

Considerem l'equació diferencial del pèndol pertorbat:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -\sin x + \varepsilon \sin \omega t \end{cases}$$

Aquesta explica el moviment d'un pèndol sota l'acció d'un camp magnètic de magnitud ε (més endavant ens referirem a l'equació com $f(x, y) = (y, -\sin(x) + \varepsilon \sin(\omega t)))$). Sigui $\phi(t, t_0, x_0, y_0, \varepsilon)$ l'equació del flux usant com a condició inicial (x_0, y_0) en t_0 (suposarem ω fixat per a aquesta equació).

Definició 1.1 (Aplicació periòdica)**.** *Sigui $f : \mathbb{R} \times \mathbb{R}^n \longrightarrow \mathbb{R}^n$ una equació diferencial, diem que aquesta és periòdica de període T si aquest és el mínim valor (major a zero) que compleix:*

$$f(t, x) = f(t + T, x) \quad \forall t, x$$

Usant aquesta definició, veiem que la nostra equació diferencial és una funció periòdica de període $T = 2\pi/\omega$.

Definició 1.2 (Aplicació de Poincaré)**.** *Sigui $\varphi(t, t_0, v_0)$ el flux d'una equació diferencial periòdica, podem definir l'aplicació de Poincaré com*

$$P(v_0) := \varphi(T, 0, v_0) = v_1$$

Definició 1.3 (Òrbita periòdica)**.** *A partir de les definicions anteriors, diem òrbita periòdica al flux que té com a condició inicial un punt fix per l'aplicació de Poincaré. Dit d'altra manera, sigui φ el flux d'una equació diferencial periòdica i sigui P la respectiva aplicació de Poincaré, aleshores si existeix $v \in \mathbb{R}^n$ tal que $P(v) = v$, diem que $\varphi(t, 0, v)$ és una òrbita periòdica.*

Nosaltres tenim com a objectiu demostrar que existeixen òrbites periòdiques en l'equació del pèndol pertorbat per tal de poder calcular el punt fix i les seves varietats de manera numèrica més endavant. Així que abans de veure això podem trobar-ho per a un cas simplificat.

1.1 Pèndol clàssic

Si en la nostra equació diferencial del pèndol pertorbat traiem l'acció del camp magnètic (posem la seva magnitud $\varepsilon = 0$), obtenim l'equació diferencial del pèndol clàssic:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -\sin x \end{cases}$$

Si a aquesta equació li busquem els punts d'equilibri (punts on $\dot{x} = 0, \dot{y} = 0$), obtenim que:

$$\begin{cases} \dot{x} = y = 0 \\ \dot{y} = -\sin(x) = 0 \end{cases} \Rightarrow \begin{cases} x = k\pi, k \in \mathbb{Z} \\ y = 0 \end{cases}$$

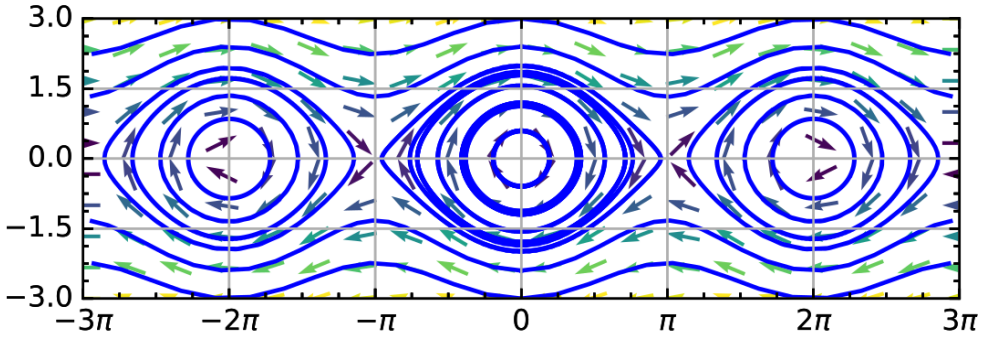


Figura 1: Retrat de fase del pèndol clàssic

Si dibuixem el seu retrat de fase, obtenim la imatge (1).[2] En aquesta imatge podem veure que els punts d'equilibri de la forma $(2k\pi, 0)$ són hiperbòlics, mentre que els de la forma $((2k+1)\pi, 0)$ són punts de sella, els quals tenen una varietat estable i una inestable. És per això que ens centrarem en aquests darrers punts, perquè al tenir varietats les podem buscar de manera numèrica. I ja com a última simplificació, d'ara endavant ens centrarem en el punt $(-\pi, 0)$, ja que tots es comporten igual per ser una funció periòdica.

Si analitzem la imatge, podem imaginar l'eix vertical com la velocitat que porta el nostre pèndol, així com l'eix horitzontal com l'angle que forma el pèndol amb la vertical. Així el punt que ens interessa és el pèndol que es troba en una posició d'equilibri inestable (és a dir, el pèndol cap per avall).

1.2 Existència d'òrbites periòdiques

Tornant al nostre problema, podem definir l'equació de Poincaré per al pèndol pertorbat com

$$P_\varepsilon(x, y) = \phi(T, 0, x, y, \varepsilon)$$

la qual té per entrada un valor de $\mathbb{R}^n$ i per sortida el punt d'arribada després d'un període. Aleshores ens pot ser útil definir una aplicació Z_ε com

$$Z_\varepsilon := P_\varepsilon(x, y) - \begin{pmatrix} x \\ y \end{pmatrix}$$

que és zero si el punt (x, y) és fix per l'aplicació de Poincaré. Aleshores és clar que per al punt trobat anteriorment $(-\pi, 0)$ i amb $\varepsilon = 0$, $Z_0(-\pi, 0)$ és un zero, ja que era un punt fix.

Ara, usarem el teorema de la funció implícita per tal de veure que existeix $\zeta(\varepsilon) = (x(\varepsilon), y(\varepsilon))$ tal que $Z_\varepsilon(\zeta(\varepsilon)) = (0, 0)$ per a un entorn prou petit de ε al voltant de zero. És a dir, que existeixen òrbites periòdiques per a valors propers a zero.

Teorema 1.4. *Existeix una òrbita periòdica per a valors propers a $\varepsilon = 0$.*

Demostració. Ja hem vist abans d'enunciar el teorema que $(x, y, \varepsilon) = (-\pi, 0, 0)$ és un zero de la funció Z_0. És també cert que $Z_\varepsilon \in \mathscr{C}^1(\mathbb{R}^2 \times (-\varepsilon_0, \varepsilon_0))$. El nostre objectiu en aquesta demostració és veure que $\det(D_{(x,y)} Z_0(-\pi, 0)) \neq 0$ per a poder aplicar el Teorema de la Funció Implícita.

[2] Imatge extreta de [2]

Per la definició de Z_ε:

$$D_{(x,y)}Z_0(-\pi,0) = D_{(x,y)}P_0(-\pi,0) - \mathrm{Id} \tag{1.1}$$

Té sentit ara mateix escriure la aplicació de Poincaré com el flux, és a dir:

$$P_\varepsilon(x,y) = \phi(T,x,y;\varepsilon) \Rightarrow D_{(x,y)}P_0(-\pi,0) = D_{(x,y)}\phi(T,-\pi,0;0)$$

Si reescrivim (1.1) segons la substitució que acabem de fer, obtenim la equació diferencial

$$\begin{cases} \frac{\partial}{\partial t}D_{(x,y)}\phi(T,-\pi,0;0) = D_{(x,y)}f(t,\phi(T,-\pi,0;0))D_{(x,y)}\phi(T,-\pi,0;0) \\ D_{(x,y)}\phi(0,-\pi,0;0) = \mathrm{Id} \end{cases}$$

La primera condició la podem reescriure calculant la matriu $D_{(x,y)}\phi(T,-\pi,0;0)$ (que simplificarem escrivint $D\phi$):

$$\frac{\partial}{\partial t}D\phi = \left(\begin{pmatrix} 0 & 1 \\ -\cos x & 0 \end{pmatrix} \Bigg|_{(x,y)=(-\pi,0)} \right) D\phi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} D\phi \Rightarrow$$

$$D\phi = e^{\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)t} = \begin{pmatrix} \cosh t & \sinh t \\ \sinh t & \cosh t \end{pmatrix} \quad \text{i ara, com} \quad DZ_0 = D\phi - \mathrm{Id}$$

$$\det(DZ_0) = \begin{vmatrix} \cosh t - 1 & \sinh t \\ \sinh t & \cosh t - 1 \end{vmatrix} = 0 \quad \text{si } t = 0.$$

Però $t = 2\pi/\omega$. És per això que $\det(DZ_0) \neq 0$ i, per tant, podem afirmar finalment que existeix la òrbita periòdica al voltant d'$\varepsilon = 0$. □

Per a obtenir també la matriu $D_{(x,y)}Z = D_{(x,y)}P - I$ ens cal obtenir la matriu $D_{(x,y)}P$. Ens cal definir el següent problema de Cauchy:

$$\begin{cases} \dot{x} = f(t,x(t)) \\ \dot{M}(t) = D_x f(t,x(t))v \\ x(0) = x_0 = \begin{pmatrix} -\pi \\ 0 \end{pmatrix} \\ M(0) = \mathrm{Id} \end{cases}$$

On $M(t) = D_{x_0}x(t)$. Recordem la definció del wronskià d'una matriu $M(t)$ com

$$w(t) = w(0) * e^{\int \mathrm{tr}\left(D_x f\left(s,x(s)\right)\right)ds}$$

Si apliquem això al nostre problema, obtenim $w(t) = w(0) * e^0 = w(0)$ ja que la matriu $D_x f$ té traça zero. Això ens indica que, usant el teorema de Liouville, el determinant de $M(t)$ serà constant, i en particular és igual a 1.

Un cop hem vist que, efectivament, existeix un punt fix, aleshores l'hem de trobar. Nosaltres busquem el punt ζ tal que $P(\zeta) = \zeta \iff Z(\zeta) = P(\zeta) - \zeta = 0$. Numèricament ho podem resoldre usant el mètode de Newton. Nosaltres sabem que si fem el desenvolupament de Taylor de $Z(\zeta_0 + h)$ en $h = 0$ ens queda $Z(\zeta_0 + h) = Z(\zeta_0) + Z_\zeta(\zeta_0)h + O_2$. Per a poder seguir endavant ens cal obviar el terme O_2, així arribarem a un mètode que tindrà error quadràtic.

Tal com hem dit, i obviant el terme d'ordre 2 per a obtenir una aproximació lineal on busquem h tal que

$$\begin{aligned} D_z Z(\zeta_0)h &= -Z(\zeta_0) \\ &\Downarrow \\ (A - \mathrm{Id})h &= \zeta_0 - P(\zeta_0) \end{aligned}$$

on $A = D_z P(\zeta_0)$. Notem que aquest sistema només té solució si $M - \mathrm{Id}$ és invertible, això no té perquè passar, tot i que veurem més endavant que en la nostra equació no planteja cap problema.

Finalment, i suposant que $M - \mathrm{Id}$ és invertible, resolem per a h i aplicant la iteració $z_1 = z_0 + h$ obtenim una successió de termes que tenen cada cop error quadràtic respecte de l'anterior.

2 Mètodes numèrics per a equacions diferencials

Una situació que sorgeix sovint treballant amb equacions diferencials és el fet que no sempre sabem —o podem— trobar una solució explícita, és a dir, una expressió que ens digui com es comporten els punts del domini al llarg del temps. En canvi, si no la podem trobar, podem intentar aproximar el moviment dels punts de manera numèrica a partir de l'equació diferencial. I aquesta última és la nostra situació.

La primera idea que podem pensar —i segurament la més senzilla— per a calcular el moviment d'un punt es basaria en substituir el valor a l'equació diferencial i fent avançar el punt en aquella direcció una petita distància. Si intentem formalitzar aquesta primera idea obtenim l'anomenat mètode d'Euler, el qual expressa el que narra aquest paràgraf de la següent manera.

Definició 2.1 (Mètode d'Euler)**.** *Sigui $\dot{x}(t) = f(t, x(t))$ una equació diferencial, on es compleix que $f : \mathbb{R} \times \mathbb{R}^m \longrightarrow \mathbb{R}^m$. Si escollim una condició inicial $(t_0, x(t_0)) = (t_0, x_0)$, definim el mètode d'Euler com els passos per a aconseguir el conjunt de punts:*

$$\begin{cases} t_1 = t_0 + h \\ x_1 = x_0 + hf(t_0, x(t_0)) \end{cases} \quad \textit{i generalitzem a} \quad \begin{cases} t_{i+1} = t_i + h \\ x_{i+1} = x_i + hf(t_i, x(t_i)) \end{cases}$$

Després d'aquesta primera introducció en la qual hem comentat què son els mètodes numèrics per a resoldre equacions diferencials, nosaltres hem optat per usar els anomenats mètodes de Runge-Kutta. Aquests es diferencien del mètode d'Euler ja que avaluen el camp en multiples punts, mentre que el d'Euler només té en compte el punt que estem seguint com podem veure a la imatge (2) (extreta de [3], pàgina 90). Hem usat aquest

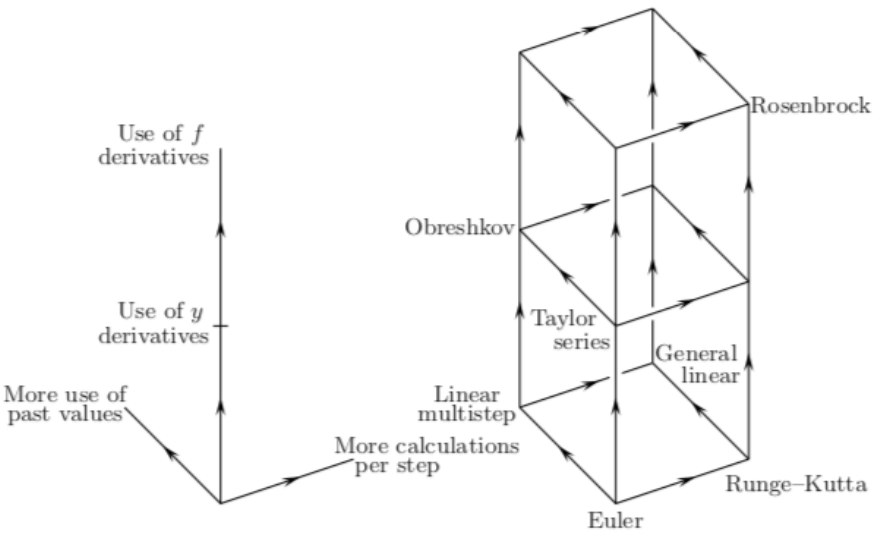


Figura 2: Diverses opcions de mètodes numèrics segons idea fonamental

mètode ja que el mètode d'Euler té moltes mancances a nivell de precisió (degut a un comportament d'error lineal). Un altre motiu de tria ha estat que suposen una base a l'hora d'implementar integradors numèrics. Alhora, també són els més coneguts i els que primer s'expliquen, per la seva fàcil implementació. Passem a veure com estan definits.

2.1 Mètodes de Runge-Kutta

La definició dels mètodes de Runge-Kutta d'ordre s és la següent:

$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i \quad \text{on}$$

$$k_i = f(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij}k_j) \quad \text{i} \quad a_{ij}, b_j, c_i \in \mathbb{R}\ \forall\ i, j.$$

Si mirem la fórmula amb deteniment podem observar que, si no imposem cap condició sobre s —anomenat el nombre d'etapes—, obtenim un sistema d'equacions que ens cal resoldre, ja que per a avaluar cada k_i ens caldria haver avaluat les altres k_j diferents. Aquests mètodes es diuen mètodes implícits. Tot i això nosaltres imposarem que $s = i-1$, així podem anar trobant cada terme k_i a partir dels valors de $k_{i-1}, \ldots, k_1$. Aquests mètodes es diuen mètodes explícits.

La idea dels mètodes explícits (i dels de Runge-Kutta en general) és la següent: primer avaluem el camp a $k_1 = f(t_n, x_n)$, la posició i el temps del punt que volem seguir. Després definim uns punts a cada $t + hc_i$ i un punt d'aquest temps $x + hp_i$ on p_i és una ponderació de les avaluacions anteriors del camp. Per últim, totes aquestes avaluacions es ponderen per les b_i i es calcula el nou punt x_{n+1}.

Una manera de simplificar la definició dels mètodes de Runge-Kutta es basa en escriure els coeficients a_{ij}, b_i, c_i en una taula de la següent forma

$$\begin{array}{c|ccc} c_1 & a_{11} & \ldots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \ldots & a_{ss} \\ \hline & b_1 & \ldots & b_s \end{array}$$

Com ja hem dit, nosaltres ens centrarem en els mètodes explícits, és a dir, aquells que $a_{ij} = 0$ si $j \geq i$. Veurem després que té sentit imposar la restricció $c_i = \sum_{j=1}^{i-1} a_{ij}$, per això per ara assumirem que $c_1 = 0$. Així la taula ens queda de la següent manera:

$$\begin{array}{c|cccccc} 0 & 0 & 0 & \ldots & 0 & 0 \\ c_2 & a_{21} & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{s-1} & a_{s-1,1} & a_{s-1,2} & \ldots & 0 & 0 \\ c_s & a_{s,1} & a_{s,2} & \ldots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \ldots & b_{s-1} & b_s \end{array}$$

Fem notar que en escriure els subíndexos dels coeficients a ho farem indistintament amb coma o sense, sempre que no generi risc de confusió i simplifiqui la seva lectura. També, i d'ara endavant, quan s'escrigui la funció f, f_t o f_x sense indicar els paràmetres d'entrada ens referim a t_n, x_n.

Per a determinar els coeficients el mètode sempre és el mateix, sigui quin sigui el valor de s. La idea es basa en derivar els termes k_i i igualar-ho al polinomi de Taylor de $x(t+h)$

per a $h = 0$, és a dir:

$$x(t+h) = x(t) + hx'(t) + \cdots + \frac{h^n}{n!}x^{(n)}(t) + O(h^{n+1})$$

Definició 2.2 (Error local i global)**.** *Per a valorar com es comporta l'error d'un mètode, ens cal definir la funció d'error com:*

$$E_{n+1} := x(t_{n+1}) - x_{n+1}$$

on el primer terme recordem que era el valor real i el segon la seva aproximació.

Definició 2.3 (Ordre d'un mètode de Runge-Kutta)**.** *Si un mètode de Runge-Kutta compleix que*

$$E_{n+1} = O(h^p)$$

aleshores aquest mètode té un error global d'ordre $p-1$. I a partir de la demostració feta per [4] (pàgines 156-157), també obtenim que l'error local és d'ordre p (és a dir, $E_1 = O(h^p)$).

Per últim, diem que l'error d'un mètode és el seu error global.

Definició 2.4 (Estabilitat)**.** *Diem que un mètode numèric de resolució d'equacions diferencials és estable quan l'òrbita d'una condició inicial per a l'equació diferencial $\dot{x} = \lambda x$ (per a $\lambda < 0$) està acotada per a tot valor de pas $h \in \mathbb{R}$ que usem.*

Notem que aquesta equació te solució $x(t) = e^{\lambda t}$ (multiplicat per la condició inicial). Aquest fet justifica aquesta definició, ja que tots els punts tendeixen a zero per aquesta equació.

2.2 Runge-Kutta de dues etapes

A partir d'aquesta definició podem buscar un primer mètode de dues etapes a mode d'exemple, és a dir, usant $s = 2$. Comencem buscant un error local d'ordre 2: $E_1 = O(h^2)$. Sabem que $x(t_{n+1}) = x(t_n) + hx'(t_n) + O(h^2)$. També $x(t)$: $x(t_n) = x_n$, $x'(t_n) = f(t_n, x_n)$ i per tant

$$x(t_{n+1}) = x_n + hf + O(h^2)$$

El nostre mètode ara té la forma:

$$x_{n+1} = x_n + h(b_1k_1 + b_2k_2) = x_n + hb_1f + hb_2f(t_n + c_2h, x_n + ha_{21}f)$$

que ens genera la taula

$$\begin{array}{c|cc} 0 & \cdot & \cdot \\ c_2 & a_{21} & \cdot \\ \hline & b_1 & b_2 \end{array}$$

Aleshores, si intentem minimitzar

$$E_1 = \cancel{x_n} + hf + O(h^2) - \Big(\cancel{x_n} + h\big(b_1f + b_2f(t_n + c_2h, x_n + a_{21}f)\big)\Big)$$

ens cal cancel·lar el terme hf. Tenim dues maneres de fer-ho, o bé posant $a_{21} = c_2 = 0$ i imposant $b_1 + b_2 = 1$ o bé $b_2 = 0, b_1 = 1$. En qualsevol d'aquests dos casos:

$$E_1 = O(h^2)$$

i per tant té ordre local 2 i ordre global 1. Ara podem intentar demanar condicions sobre les variables a fi d'aconseguir error local d'ordre 3, $E_1 = O(h^3)$. Sabem que $x(t_{n+1}) = x(t_n) + hx'(t_n) + \frac{h^2}{2}x''(t_n) + O(h^3)$. Per la definició de $x(t)$: $x(t_n) = x_n$, $x'(t_n) = f(t_n, x_n)$ i

$$x''(t_n) = \frac{\partial}{\partial t}x'(t_n) = \frac{\partial}{\partial t}f(t_n, x_n) = f_t(t_n, x_n) + f(t_n, x_n)f_x(t_n, x_n) = f_t + ff_x$$

Ara, reescrivint:

$$x(t_{n+1}) = x_n + hf + \frac{h^2}{2}(f_t + ff_x) + O(h^3)$$

Notem que si desenvolupem el polinomi de Taylor per a $h = 0$ de k_2 obtindrem una cosa prou similar al terme $f_t + ff_x$ que volem eliminar:

$$f(t_n + c_2h, x_n + ha_{21}f) = f + h(c_2f_t + a_{21}ff_x) + O(h^2) = (*)$$

i com ja hem comentat anteriorment que $c_i = \sum_{j=1}^{i-1} a_{ij}$, això ens implica que $c_2 = a_{21}$ i per tant podem agrupar en:

$$(*) = f + hc_2(f_t + ff_x) + O(h^2)$$

Per tant, el mètode el podem reescriure com:

$$x_{n+1} = x_n + hb_1f + hb_2(f + hc_2(f_t + ff_x)) = x_n + hf(b_1 + b_2) + h^2c_2b_2(f_t + ff_x) + O(h^3)$$

Si ens fixem, notem que ja tenim un terme x_n, un f i un $f_t + ff_x$, així que ja podem intentar minimitzar E_1:

$$E_1 = \cancel{x_n} + hf + \frac{h^2}{2}(f_t + ff_x) + O(h^3 - (\cancel{x_n} + hf(b_1 + b_2) + h^2c_2b_2(f_t + ff_x) + O(h^3)) =$$

$$= hf(1 - (b_1 + b_2)) + h^2(f_t + ff_x)(\frac{1}{2} - c_2b_2) + O(h^3)$$

Que ens genera el sistema:

$$\begin{cases} b_1 + b_2 = 1 \\ c_2b_2 = \frac{1}{2} \end{cases}$$

Això, escrit en forma de taula ens queda (si $c_2 = \nu$):

0	·	·
ν	ν	·
	1 - $1/2\nu$	$1/2\nu$

Notem que això està ben definit ja que té sentit imposar $\nu \neq 0$, ja que, si no, el mètode acaba essent de $s = 1$:

$$x_{n+1} = x_n + hb_1f + hb_2f(t_n + c_2h, x_n + ha_{21}f) = x_n + hb_1f + hb_2f = x_n + h(b_1 + b_2)f$$

usant que $c_2 = a_{21} = 0$. Aleshores, per a qualsevol valor de $\nu \in \mathbb{R} \setminus \{0\}$ obtenim un mètode d'ordre local 3 i ordre global 2.

Exemple 2.5. Per a il·lustrar això que acabem de trobar, farem un exemple amb una equació diferencial que sapiguem resoldre:

$$\dot{x} = 2x \Rightarrow x(t) = e^{2t}x_0$$

Usarem $x_0 = 1$ i agafarem, també, un pas de mida $h = 0.25$. Els valors reals d'aquest punt són els següents:

t	$x = e^{2t}$
$t = 0$	1
$t = h = 0.25$	$\sqrt{e} = 1.648$
$t = 0.5$	$e = 2.718$
$t = 0.75$	$\sqrt{e^3} = 4.481$
$t = 1$	$e^2 = 7.389$

Ara, calcularem els punts que ens sortirien amb un Runge-Kutta d'ordre 2 (per exemple, agafant $\nu = 1$). Cada pas d'integració es defineix de la següent manera per a aquest problema:

$$x_{n+1} = x_n + h(b_1k_1 + b_2k_2) \quad \text{on} \quad \begin{cases} h = 0.25 \\ b_1 = 0.5 \\ k_1 = f(t_n, x_n) \\ b_2 = 0.5 \\ k_2 = f(t_n + 0.25, x_n + 0.25f(t_n, x_n)) \end{cases}$$

Com comencem per $x_0 = 1$, el següent valor és

$$x_1 = 1 + 0.25\Big(0.5f(0,1) + 0.5f\big(0.25, 1 + 0.25f(0,1)\big)\Big) = \cdots = 1 + 0.625 = 1.625$$

Si anem calculant fins a $t = 1$, obtenim:

t	$x_{n+1} = x_n + h(\sum b_ik_i)$
$t = 0$	1
$t = 0.25$	1.625
$t = 0.5$	2.6406
$t = 0.75$	4.2901
$t = 1$	6.9729

que, si ho comparem amb els valors reals obtenim uns errors absoluts de:

t	$\lvert x_n - e^{2t}\rvert$
$t = 0$	0
$t = 0.25$	0.0237
$t = 0.5$	0.0776
$t = 0.75$	0.1906
$t = 1$	0.4161

Podem comprovar que, com podíem esperar, entren dins del marge d'error $O(h^2)$, perquè $h^2 = 0.0625$. Ara podem provar de fer el mateix mètode usant $2h$. Això ens servirà ja que com l'error es comporta com h^2, si posem $2h$, l'error es multiplica per 4 aproximadament (ja que $(2h)^2 = 4h^2$).

t	x_n
$t = 0$	1
$t = 0.5$	2.5

Notem aleshores que les diferencies són:

t	$h = 0.25$	$h = 0.5$
$t = 0$	0	0
$t = 0.5$	0.0776	0.2182

2.3 Runge-Kutta d'ordre 3

Si seguim pujant l'ordre, podem intentar definir un mètode d'ordre 3, és a dir, busquem condicions sobre els paràmetres.

$$\begin{array}{c|ccc} 0 & \cdot & \cdot & \cdot \\ c_2 & a_{21} & \cdot & \cdot \\ c_3 & a_{31} & a_{32} & \cdot \\ \hline & b_1 & b_2 & b_3 \end{array}$$

tals que ens asseguren $E_{n+1} = O(h^4)$. Fent paral·lelismes amb l'apartat anterior, sabem que $x(t_{n+1}) = x(t_n) + hx'(t_n) + \frac{h^2}{2}x''(t_n) + \frac{h^3}{6}x'''(t_n) + O(h^4)$, on:

$$\begin{cases} x'(t) = f \\ x''(t) = f' = f_t + ff_x \\ x'''(t) = f'' = f_{tt} + f_x f_t + 2ff_{tx} + f^2 f_{xx} + ff_x^2 \end{cases}.$$

Per altra banda, i seguint el raonament de l'apartat anterior, ens cal usar un mètode de Runge-Kutta amb $s = 3$. Aquest l'escriurem de la següent manera:

$$x_{n+1} = x_n + h(b_1k_1 + b_2k_2 + b_3k_3) \quad \text{on}$$

$$\begin{cases} k_1 = f(x_n) = f_n \\ k_2 = f(x_n + hc_2, x_n + ha_{21}k_1) \\ k_3 = f(x_n + hc_3, x_n + h(a_{31}k_1 + a_{32}k_2)) \end{cases}$$

Igual que abans, podem, a priori, definir unes restriccions sobre aquestes variables:

$$\begin{cases} c_2 = a_{21} \neq 0 \text{ ja que, si no, } k_1 = k_2 \\ c_3 = a_{31} + a_{32} \\ b_1 + b_2 + b_3 = 1 \end{cases}$$

Abans de començar, hem de definir una manera de simplificar la notació, ja que per a $s = 3$ les derivades (i en particular $\dfrac{\partial^2}{\partial h^2}k_3$) augmenten de mida considerablement. És per

això que escriurem k_1, k_2 i k_3 com k, K i $\tilde{\mathrm{K}}$ respectivament. Notem que, avaluades en $h = 0$, totes valen f_n. La nostra estratègia per a reduir E_{n+1} a ordre h^4 serà trobar el polinomi de Taylor respecte de h de k_2 i k_3 fins a $O(h^3)$.

La sèrie de Taylor de k_2 és

$$K = f(x_n + hc_2, x_n + ha_{21}k) = f(t_n, x_n) + \boxed{h\frac{\partial}{\partial h}K|_{h=0}} + \boxed{\frac{h^2}{2}\frac{\partial^2}{\partial h^2}K|_{h=0}} + O(h^3) \quad \text{on}$$

$$\frac{\partial}{\partial h}K = \frac{\partial}{\partial h}f(t + c_2h, x_n + hc_2k) = c_2K_t + c_2kK_x \overset{h=0}{=} c_2f_t + c_2ff_x = \boxed{c_2(f_t + ff_x)}$$

$$\frac{\partial^2}{\partial h^2}K = \frac{\partial^2}{\partial h^2}f(t + c_2h, x_n + hc_2k) = \frac{\partial}{\partial h}c_2(K_t + kK_x) =$$

$$= c_2(c_2K_{tt} + c_2kK_{tx} + k(c_2K_{xt} + c_2kK_{xx})) \overset{h=0}{=} \boxed{c_2^2f_{tt} + 2c_2^2ff_{tx} + c_2^2f^2f_{xx}}$$

De la mateixa manera, la sèrie de Taylor de k_3 és

$$\tilde{\mathrm{K}} = f(x_n + hc_3, x_n + h(a_{31}k + a_{32}K)) =$$

$$f(t_n, x_n) + \boxed{h\frac{\partial}{\partial h}\tilde{\mathrm{K}}\,|_{h=0}} + \boxed{\frac{h^2}{2}\frac{\partial^2}{\partial h^2}\tilde{\mathrm{K}}\,|_{h=0}} + O(h^3) \quad \text{on}$$

$$\frac{\partial}{\partial h}\tilde{\mathrm{K}} = c_3\,\tilde{\mathrm{K}}_t + \tilde{\mathrm{K}}_x(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)) \overset{h=0}{=} c_3f_t + ff_x(a_{31} + a_{32}) =$$

$$= \boxed{c_3(f_t + ff_x)}$$

ja que $a_{31} + a_{32} = c_3$ com hem comentat a priori. Per últim la segona derivada de $\tilde{\mathrm{K}}$:

$$\frac{\partial^2}{\partial h^2}\tilde{\mathrm{K}} = c_3\Big[c_3\,\tilde{\mathrm{K}}_{tt} + \tilde{\mathrm{K}}_{tx}\left(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)\right)\Big] +$$

$$+a_{31}k\Big[c_3\,\tilde{\mathrm{K}}_{tx} + \tilde{\mathrm{K}}_{xx}\left(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)\right)\Big] +$$

$$+a_{32}\Big[\tilde{\mathrm{K}}_x\left(c_2K_t + c_2kK_x\right) + K\Big(c_3\,\tilde{\mathrm{K}}_{tx} + \tilde{\mathrm{K}}_{xx}\left(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)\right)\Big)\Big] +$$

$$+ c_2a_{32}\Big[\tilde{\mathrm{K}}_x\,K_t + hK_t\Big(c_3\,\tilde{\mathrm{K}}_{tx} + \tilde{\mathrm{K}}_{xx}\left(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)\right)\Big) +$$

$$+ h\,\tilde{\mathrm{K}}_x\left(c_2K_{tt} + c_2kK_{tx}\right)\Big] +$$

$$+ c_2a_{32}k\Big[\tilde{\mathrm{K}}_x\,K_x + hK_x\Big(c_3\,\tilde{\mathrm{K}}_{tx} + \tilde{\mathrm{K}}_{xx}\left(a_{31}k + a_{32}K + a_{32}h(c_2K_t + c_2kK_x)\right)\Big) +$$

$$+ h\,\tilde{\mathrm{K}}_x\left(c_2K_{xt} + c_2kK_{xx}\right)\Big] =$$

Aleshores, posant $h = 0$,

$$= c_3^2f_{tt} + c_3ff_{xt}(a_{31} + a_{32}) + a_{31}f\left(c_3f_{tx} + ff_{xx}(a_{31} + a_{32})\right) + c_2a_{32}f_x(f_t + ff_x) +$$

$$+ a_{32}f(c_3f_{tx} + ff_{xx}(a_{31} + a_{32})) + c_2a_{32}f_xf_t + a_{32}a_{21}ff_x^2 =$$

I arreglant i agrupant el resultat obtenim:

$$= \boxed{c_3^2 f_{tt} + 2c_3^2 f f_{xt} + c_3^2 f f_{xt} + 2c_2 a_{32} f_t f_x + 2c_2 a_{32} f f_x^2}$$

Per últim, escrivint la forma explícita del mètode de Runge-Kutta trobat:

$$x_{n+1} = x_n + hb_1 f + hb_2\left(f + \boxed{h\frac{\partial}{\partial h}K|_{h=0}} + \boxed{\frac{h^2}{2}\frac{\partial^2}{\partial h^2}K|_{h=0}} + O(h^3)\right) +$$
$$+ hb_3\left(f + \boxed{h\frac{\partial}{\partial h}\tilde{\mathrm{K}}\,|_{h=0}} + \boxed{\frac{h^2}{2}\frac{\partial^2}{\partial h^2}\tilde{\mathrm{K}}\,|_{h=0}} + O(h^3)\right) =$$

$$= x_n + hf(b_1 + b_2 + b_3) + h^2(f_t + f f_x)(b_2 c_2 + b_3 c_3) +$$
$$+ h^3\left(\left(\frac{b_2 c_2^2 + b_3 c_3^2}{2}\right)(f_{tt} + f^2 f_{xx}) + (b_2 c_2^2 + b_3 c_3^2) f f_{tx} + (b_3 c_2 a_{32})(f_t f_x + f f_x^2)\right) + O(h^4)$$

Per tal de simplificar l'enteniment de tot això, ho escriurem a la següent taula. A l'esquerra tenim la columna del polinomi de Taylor i a la dreta el mètode de Runge-Kutta

$x(t_{n+1})$	Coeficient	x_{n+1}
1	f	$b_1 + b_2 + b_3$
$1/2$	f_t	$b_2 c_2 + b_3 c_3$
$1/2$	$f f_x$	$b_2 c_2 + b_3 c_3$
$1/6$	f_{tt}	$(c_2^2 b_2 + c_3^2 b_3)/2$
$1/3$	$f f_{tx}$	$c_2^2 b_2 + c_3^2 b_3$
$1/6$	$f^2 f_{xx}$	$c_2^2 b_2 + c_3^2 b_3/2$
$1/6$	$f_t f_x$	$b_3 c_2 a_{32}$
$1/6$	$f f_x^2$	$b_3 c_2 a_{32}$

Així podem plantejar totes les restriccions de les variables:

$$\begin{cases} c_2 = a_{21} \\ c_3 = a_{31} + a_{32} \\ b_1 + b_2 + b_3 = 1 \\ b_2 c_2 + b_3 c_3 = \frac{1}{2} \\ b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \\ b_3 c_2 a_{32} = \frac{1}{6} \end{cases}$$

Amb aquestes condicions, la nostra taula es simplifica de la següent manera:

$$\begin{array}{c|ccc} 0 & \cdot & \cdot & \cdot \\ c_2 & a_{21} & \cdot & \cdot \\ c_3 & a_{31} & a_{32} & \cdot \\ \hline & b_1 & b_2 & b_3 \end{array} \longrightarrow \begin{array}{c|ccc} 0 & \cdot & \cdot & \cdot \\ \gamma & \gamma & \cdot & \cdot \\ \frac{1/2-\alpha\gamma}{\beta} & \frac{3\gamma-6\alpha\gamma^2-1}{6\beta\gamma} & \frac{1}{6\gamma\beta} & \cdot \\ \hline & 1-(\alpha-\beta) & \alpha & \beta \end{array}$$

$$\text{on hem usat que } b_2 = \alpha, b_3 = \beta, c_2 = a_{21} = \gamma$$

Aleshores, escollint valors per a α, β i γ obtenim un mètode de Runge-Kutta d'ordre 3. Notem que no està definit per a combinacions de valors on $\beta = 0$ ó $\gamma = 0$, però té sentit imposar-ho, ja que si β és zero, aleshores k_3 està multiplicat per zero i tornem a tenir un mètode amb només k_1 i k_2. De la mateixa manera, si permetem $\gamma = 0$, tenim que $k_2 = k_1$ i ja hem comentat abans que té sentit imposar que no sigui així.

Ara potser ens podríem plantejar fer mètodes amb ordres més grans, tot i això la tasca es complica. Com diuen Griffiths and Higham [5], amb un mètode de 4 etapes podem aconseguir ordre global 4, tot i això, si busquem mètodes amb ordres globals majors ens cal un nombre major d'etapes. Per exemple, per a aconseguir el mètode d'ordre 8 que usem més endavant cal usar 13 etapes.

2.4 Control de pas

Per a parlar del control de pas primer hem de fer unes consideracions prèvies. Fins ara hem estat pensant en els mètodes de Runge-Kutta com una successió donada per $x_{n+1} = x_n + h(\sum_i b_i k_i)$ on es calcula el següent punt amb un pas h fix. Un altre plantejament que podem fer és: podem adaptar la variable h a un error màxim —diguem-li tolerància— per a cada pas d'integració? La resposta a això és que sí, pensem el següent exemple. Suposem que tenim dos mètodes, un que ens ofereix un error de $O(h^p)$ (li direm σ_p) i un altre amb error $O(h^{p+1})$ (anomenat σ_{p+1}).

Nosaltres sabem que aquests dos mètodes els podem veure com:

$$\sigma_p \simeq e + ch^p$$
$$\sigma_{p+1} \simeq e + dh^{p+1}$$

on e és el valor aproximat trobat i $c, d \in \mathbb{R}$. Podem trobar una aproximació de l'error mitjançant el càlcul de $|\sigma_{p+1} - \sigma_p| \simeq ch^p$. Aquí poden passar dos casos diferents, tot i que els podem tractar de la mateixa manera. En cas que $|\sigma_{p+1} - \sigma_p| > \text{tol}$, podem buscar $\alpha \in \mathbb{R}$ tal que αh ens doni un error menor a la tolerància triada:

$$c(\alpha h)^p = \alpha^p c h^p = \alpha^p |\sigma_{p+1} - \sigma_p| \leq \text{tol} \Rightarrow \alpha \leq \sqrt[p]{\frac{\text{tol}}{|\sigma_{p+1} - \sigma_p|}}$$

Aleshores, si tornéssim a repetir l'últim pas d'integració amb αh, obtindríem exactament la tolerància desitjada. Una cosa a tenir en compte és que, tot i que haguem tractat l'error com un terme d'ordre h^p, no hem d'oblidar que també tenim termes d'error h^{p+1} en endavant, tot i que els haguem obviat. Aquests errors es poden anar acumulant i fent que el nostre nou valor de pas excedeixi l'error màxim demanat. Per això definim un valor ε, al qual anomenarem factor de control, que serà un valor a $(0, 1)$ i servirà per a contrarestar aquest error que hem obviat, així com un altre detall que comentarem més endavant. Aquest càlcul d'α també es pot fer quan $|\sigma_{p+1} - \sigma_p| < \text{tol}$.

Abans de seguir ens cal recordar per què buscàvem α. La nostra idea era arribar a un valor d'error demanat; per això, si amb el primer pas provat ja ens hem quedat per sota de la tolerància, no el repetiríem. El que seria molt interessant en aquest moment és intentar preveure abans de fer el primer càlcul quin seria el valor α ideal. I aquí torna a entrar en joc el factor de control; ja que podem assumir que una equació diferencial es comporta de manera similar entre punts propers. Així que posant un factor —tipus

$\alpha = 0.9$— podem esperar que entre un pas d'integració i el següent no s'hagi de recalcular amb una nova α.

Per a recapitular, i a mode d'esquema, escriurem el pseudocodi del possible algoritme de control de pas:

1. Pas inicial amb $h = 0.01$
2. Calculem x_n mitjançant σ_{p+1} amb pas h
3. Comprovem si $|\sigma_{p+1} - \sigma_p| < \text{tol}$:
   (a) Si $|\sigma_{p+1} - \sigma_p| > \text{tol}$, calculem α, assignem $h \longleftarrow h\alpha$ i anem al pas 2
   (b) Si $|\sigma_{p+1} - \sigma_p| \leq \text{tol}$, calculem α, assignem $h \longleftarrow h\alpha$ i anem al pas 2 per a calcular x_{n+1}

Notem que, en algun cas particular, podria ser que anéssim calculant α sense aconseguir mai que l'error fos menor a la tolerància desitjada. Una possible correció d'aquest error seria introduir un comptador que aturi el programa si ens passem del nombre d'iteracions definides.

Com ja hem dit a l'inici d'aquest apartat, per a tenir un control de pas ens calen dos mètodes d'ordre diferent. En aquest treball usarem el mètode de Runge-Kutta-Verner, el qual usa dos mètodes, un d'onze etapes i l'altre de tretze etapes, que donen un terme d'error global d'ordres h^7 i h^8 respectivament. La gràcia d'aquests mètodes és que tenen les variables a_{ij} i c_i de mateix valor, així el camp s'avalua de la mateixa manera per als dos mètodes i simplement canvia la ponderació de les avaluacions amb les variables b_i (és a dir, les variables b del primer mètode no són, o no tenen perquè ser, les mateixes que les del segon).

Un últim detall que podríem comentar és el fet que aquests dos mètodes ens serveixen per a tenir una certa idea de l'error. Per això és important decidir quin dels dos punts calculats usem com a sortida per a la propera iteració. El nostre programa per defecte usa el mètode de 13 etapes ja que podem esperar un error menor, tot i que el programa ofereix la opció de canviar aquesta tria.

A l'apartat de Resultats i Validacions en podem trobar un exemple.

3 Mètode de la parametrització

Una altra eina que hem usat en aquest treball és l'anomenat mètode de la parametrització. Donat un punt fix per una aplicació de Poincaré, podem intentar parametritzar la corba que formen les seves varietats estables i inestables. Sigui $z(s) = \sum_{k\geq 0} a_k s^k$ la parametrització de la corba, podem afirmar que la imatge $P\big(z(s)\big)$ torna a ser ella mateixa per la definició del que és una varietat, tot i que els punts no van a ells mateixos.

Si l'aplicació P fos lineal,

$$P\begin{pmatrix}x\\y\end{pmatrix} = A\begin{pmatrix}x\\y\end{pmatrix} \quad \text{i} \quad \begin{cases}\lambda \in \operatorname{spec}(A)\\ v \text{ vep}\end{cases} \Rightarrow Av = \lambda v$$

i aleshores $z(s) = vs \Rightarrow P\big(z(s)\big) = Avs = \lambda vs = z(\lambda s)$. Veiem que si fos lineal, ens seria molt fàcil trobar els valors a_k que parametritzen la corba. I en el cas que no sigui lineal ara veurem que també ho podem trobar terme a terme.

Proposició 3.1. *Si $\lambda^i \notin \operatorname{spec}(DP(a_0))\ \forall i > 1$, aleshores sempre podem trobar valors d'a_k que compleixen aquesta condició.*

Demostració. Comencem per a_0:

$$P(a_0 + a_1 s) = a_0 + \lambda a_1 s \overset{s=0}{\Rightarrow} P(a_0) = a_0$$

i per tant a_0 és el punt fix que hem trobat abans. Per a trobar a_1 ho farem per Taylor:

$$\cancel{P(a_0)} + DP(a_0)a_1 s + O(s^2) = \cancel{a_0} + \lambda a_1 s + O(s^2) \Rightarrow DP(a_0)a_1 s = \lambda a_1 s$$

i per tant a_1 és el vector propi de la varietat que estiguem calculant ($a_1 = v$). Per últim, suposem que tenim calculats els valors fins a a_m:

$$P(a_0 + \cdots + a_m s^m + a_{m+1}s^{m+1} + O(s^{m+2})) = a_0 + \cdots + \lambda^{m+1} a_{m+1} s^{m+1} + O(s^{m+2})$$

Si fem Taylor de $a_0 + \cdots + a_m s^m$:

$$P(a_0 + \cdots + a_m s^m) + DP(a_0 + \cdots + a_m s^m)(a_{m+1}s^{m+1} + O(s^{m+2})) + O(s^{m+2}) =$$

$$= a_0 + \cdots + \lambda^m a_m s^m + O(s^{m+1}) + DP(a_0)a_{m+1}s^{m+1} + O(s^{m+2}) = *$$

i això ho igualem a

$$* = a_0 + \cdots + \lambda^m a_m s^m + \lambda^{m+1} a_{m+1} s^{m+1} + O(s^{m+2}) \Rightarrow$$

$$O(s^{m+1}) + DP(a_0)a_{m+1}s^{m+1} = \lambda^{m+1} a_{m+1} s^{m+1}$$

El terme $O(s^{m+1})$ ens genera problemes, així que el canviem per $O(s^{m+1}) = b_{m+1}s^{m+1} + O(s^{m+2})$ i ara podem resoldre a_{m+1}:

$$DP(a_0)a_{m+1} - \lambda^{m+1}a_{m+1} = -b_{m+1} \Rightarrow \big[DP(a_0) - \lambda^{m+1}\operatorname{Id}\big]a_{m+1} = -b_{m+1}$$

Per a poder resoldre aquest sistema final ens cal que la matriu $DP(a_0) - \lambda^{m+1}\operatorname{Id}$ sigui invertible, és a dir, que $\lambda^{m+1} \notin \operatorname{spec}(DP(a_0))$. I efectivament λ^{m+1} no hi pertany per a $m > 1$ per la condició de l'enunciat. Així que podem resoldre finalment:

$$a_{m+1} = -\big[DP(a_0) - \lambda^{m+1}\operatorname{Id}\big]^{-1} b_{m+1}$$

Per tant, si trobem el valor de b_{m+1}, trobem el valor de a_{m+1}. I aquest el podem trobar de manera experimental mitjançant el mètode de Runge-Kutta que volguem, usant com a punt inicial la sèrie:

$$\begin{pmatrix} a_{0,0} \\ a_{0,1} \end{pmatrix} + s \begin{pmatrix} a_{1,0} \\ a_{1,1} \end{pmatrix} + \cdots + s^m \begin{pmatrix} a_{m,0} \\ a_{m,1} \end{pmatrix} + s^{m+1} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

i arribant a b_{m+1} el terme de s^{m+1}. □

Un cop tinguem aquest polinomi calculat fins a un grau arbitrari, aleshores podem assegurar que $P(z(s)) \simeq z(\lambda s)$ per a valors de s prou petits. Una cosa que podem usar al nostre favor és que un mètode per a dibuixar la varietat sense dependre dels errors que et dóna el polinomi per a valors de s grans és agafar un seguit de punts equidistants entre $z(s)$ i $z(\lambda s)$ i integrar-los per a un nombre arbitrari d'iteracions. Més endavant veurem en una imatge per què és una bona idea.

Si triem un total de q punts entre $z(s)$ i $z(\lambda s)$, que podem generar equidistants com:

$$\begin{cases} \zeta_0 = z(s) \\ \zeta_i = z\Big(s + \dfrac{(\lambda - 1)s}{q} i\Big) \\ \zeta_q = z(\lambda s) \end{cases}$$

Aquests punts al cap d'un període d'integració es trobaran entre $\zeta_q = P(\zeta_0) = z(\lambda s)$ i $P(\zeta_q) = z(\lambda^2 s)$ i així successivament.

Tots aquests mètodes i idees ens plantegen el problema: com treballem amb sèries amb els mètodes de Runge-Kutta? La resposta la veurem a la següent secció.

4 Aritmètica de sèries

Abans de començar volem remarcar que al llarg d'aquesta secció anirem marcant amb marcs els càlculs per a aconseguir una operació, tot el que no tingui un marc serà una demostració del càlcul. La referència d'aquesta secció així com la notació usada ha estat extreta de [6].

Com hem comentat, en algun moment hem volgut calcular varietats invariants mitjançant el mètode de la parametrització. Això ens demana una nova eina per a treballar amb sèries. Usarem el que s'anomenen derivades normalitzades, que es defineixen de la següent manera:

$$x^{[n]}(t) := \frac{1}{n!}x^{(n)}(t)$$

Aquesta definició, que a priori sembla simple, ens permet calcular sèries de Taylor de funcions complexes a partir de funcions senzilles o conegudes. Per exemple, si tenim una suma de dues sèries, aleshores les derivades normalitzades del resultat es poden calcular com:

$$z(t) = x(t) + y(t) \Rightarrow \boxed{z^{[n]}(t) = x^{[n]}(t) + y^{[n]}(t)} \ \forall n$$

De la mateixa manera es defineix la resta de sèries. Si avancem en la llista d'operacions que podem fer, podríem tenir z com el producte de dues sèries, i els seus coeficients es calcularien com:

$$z(t) = x(t)y(t) \Rightarrow \boxed{z^{[n]}(t) = \sum_{i=0}^{n} x^{[i]}(t)y^{[n-i]}(t)}$$

ja que

$$z^{[n]} = \frac{1}{n!}z^{(n)}(t) = \frac{1}{n!}\sum_{i=0}^{n}\binom{n}{i}x^{(i)}(t)y^{(n-i)}(t) = \frac{1}{\cancel{n!}}\sum_{i=0}^{n}\cancel{n!}\frac{x^{(i)}(t)}{i!}\frac{y^{(n-i)}(t)}{(n-i)!} = \sum_{i=0}^{n}x^{[i]}(t)y^{[n-i]}(t)$$

com volíem veure. Un altre cas que podem valorar és el del quocient de dues funcions. En aquest cas es pot calcular com

$$z(t) = \frac{x(t)}{y(t)} \Rightarrow \boxed{z^{[n]}(t) = \frac{1}{y_0(t)}\left(x^{[n]}(t) - \sum_{i=1}^{n} y^{[i]}(t)z^{[n-i]}(t)\right)}$$

que surt a partir de pensar-ho com $z(t)y(t) = x(t)$. Després podem voler calcular el logaritme d'una sèrie, que el podem trobar com:

$$z(t) = \log(x(t)) \Rightarrow \boxed{z^{[n]}(t) = \frac{1}{x_0(t)}\left(x^{[n]}(t) - \frac{1}{n}\sum_{j=1}^{n-1}(n-j)x^{[j]}(t)z^{[n-j]}(t)\right)}$$

Això ho trobem a través d'uns quants fets. Primer de tot, derivant obtenim una relació sobre $z'(t) = \dfrac{x'(t)}{x(t)}$. Ara, si sabem com relacionar z' amb z ja ho tindríem, i ho sabem ja que:

$$\frac{\partial}{\partial t}z^{[i]}(t) = \frac{\partial}{\partial t}\left(\frac{1}{i!}z^{(i)}(t)\right) = \frac{1}{i!}z^{(i+1)}(t) = (i+1)z^{[i+1]}(t)$$

Usant aquests dos fets podem demostrar el que hem dit:

$$z^{[n]}(t) = \frac{1}{n}\frac{\partial}{\partial t}z^{[n-1]} = \frac{1}{nx_0}\left(\frac{\partial}{\partial t}x^{[n-1]} - \sum_{j=1}^{n-1}x^{[j]}(t)\frac{\partial}{\partial t}z^{[n-1-j]}\right) =$$

$$= \frac{1}{nx_0}\left(nx^{[n]} - \sum_{j=1}^{n-1} x^{[j]}(t)(n-j)z^{[n-j]}\right) \Rightarrow$$

$$z^{[n]}(t) = \frac{1}{x_0(t)}\left(x^{[n]}(t) - \frac{1}{n}\sum_{j=1}^{n-1}(n-j)x^{[j]}(t)z^{[n-j]}(t)\right)$$

També podem fer l'exponencial d'una funció, que resulta:

$$z(t) = e^{x(t)} \Rightarrow \boxed{z^{[n]}(t) = \frac{1}{n}\sum_{j=0}^{n-1}(n-j)z^{[j]}(t)x^{[n-j]}(t)}$$

Aquesta es pot demostrar de manera similar al logaritme. Primer, si derivem ens trobem amb $z'(t) = e^{x(t)}x'(t) = z(t)x'(t)$; i ara:

$$z^{[n]} = \frac{1}{n}\frac{\partial}{\partial t}z^{[n-1]} = \frac{1}{n}\sum_{j=0}^{n-1} z^{[j]}(t)\frac{\partial}{\partial t}x^{[n-1-j]}(t) = \frac{1}{n}\sum_{j=0}^{n-1}(n-j)z^{[j]}(t)x^{[n-j]}(t)$$

A partir d'aquestes dues fórmules per a la exponencial i el logaritme podem obtenir les potències $z(t) = x(t)^\alpha$ com

$$\log(z(t)) = \log(x(t)^\alpha) = \alpha\log(x(t)) \Rightarrow z(t) = e^{\alpha\log(x(t))}$$

Per últim, podem necessitar les funcions trigonomètriques $z(t) = \sin(x(t))$ i $y(t) = \cos(x(t))$. Aquestes dues s'han de tractar alhora per la seva condició cíclica. Si les derivem obtenim les següents relacions: $y' = -\sin x x' = -zx$ i $z' = \cos(x)x' = zx'$. Ara, si apliquem la definició de la multiplicació, obtenim:

$$z^{[n]} = \frac{1}{n}\frac{\partial}{\partial t}z^{[n-1]} = \frac{1}{n}\sum_{j=0}^{n-1} y^{[n-1-j]}\frac{\partial}{\partial t}x^{[j]} = \frac{1}{n}\sum_{j=0}^{n-1}(j+1)x^{[j+1]}y^{[n-(1+j)]} \Rightarrow$$

$$z(t) = \sin(x(t)) \Rightarrow \boxed{z^{[n]} = \frac{1}{n}\sum_{j=1}^{n} jx^{[j]}y^{[n-j]}}$$

$$y^{[n]} = \frac{-1}{n}\frac{\partial}{\partial t}y^{[n-1]} = \frac{-1}{n}\sum_{j=0}^{n-1} z^{[n-1-j]}\frac{\partial}{\partial t}x^{[j]} = \frac{-1}{n}\sum_{j=0}^{n-1}(j+1)x^{[j+1]}z^{[n-(1+j)]} \Rightarrow$$

$$y(t) = \cos(x(t)) \Rightarrow \boxed{y^{[n]} = \frac{-1}{n}\sum_{j=1}^{n} jx^{[j]}z^{[n-j]}}$$

Notem que, com no usem $j = 0$ al sumatori, no usem ni y_n ni z_n als càlculs de z_n i y_n respectivament. És per això que podem trobar els valors de manera iterativa, calculant a cada pas tant la y com la z que toqui.

Un cop tenim ja tots els càlculs que ens cal fer per a poder operar amb sèries, passarem a comentar la implementació que hem fet d'això, juntament amb l'algoritme de Runge-Kutta-Verner en C++.

5 Implementació en C++

Com ja hem comentat, l'objectiu d'aquest treball era programar un mètode en C++ per tal de poder-lo usar com a integrador numèric. A part de l'integrador, també hem parlat del mètode de la parametrització, el qual ens podia ser útil per tal de calcular les varietats invariants d'un punt fix per l'aplicació de Poincaré. Si volem programar-ho també en C++ ens caldrà implementar una aritmètica de sèries, la qual ens permeti treballar amb sèries de Taylor. I és per això que hem escollit C++, ja que la seva funcionalitat de sobrecàrrega d'operadors ens permet programar amb una notació menys farragosa.

Quan parlem de sobrecàrrega d'operadors ens referim a la possibilitat de reescriure les funcions bàsiques d'un llenguatge $(+, -, ...)$ d'una manera natural. C++ ens permet, sempre que li defineixis prèviament, escriure una suma entre dos objectes que no siguin dels tipus predeterminats del llenguatge. És a dir, així com si volem fer una suma de dos objectes `p,q` definits per nosaltres en altres llenguatges s'hauria de fer d'una manera similar a `sum(p,q)`, en C++ ho podem fer com `p+q`. I aquí rau la decisió; per una banda, ens simplifica molt la lectura del codi per a solucionar errors i, per l'altra, ens permet usar els mateixos arxius tant si estem treballant amb un objecte `Serie` com amb un tipus `double`.

I d'entre tots els mètodes que podríem haver escollit hem triat els mètodes de Runge-Kutta. Com ja hem vist anteriorment a la figura (2), els mètodes de Runge-Kutta difereixen del mètode d'Euler pel nombre d'avaluacions del camp: així com aquest darrer en fa només una, els primers en fan tantes com l'ordre del mètode. I això és fàcil de programar, no cal una estructura molt complexa per a avaluar una equació diferencial més cops. I a més a més, quan parlem de mètodes numèrics per a resoldre integrals, aquests mètodes —els de Runge-Kutta— sempre són presents.

Una cosa que s'ha dit unes seccions enrere i que no estaria de més recordar és que el mètode de Runge-Kutta triat és el trobat per Verner a [1], el qual usa dos mètodes, un mètode d'onze etapes i un de tretze per a obtenir un terme d'error com h^8.

Així doncs, i un cop comentada i justificada la tria de llenguatge i mètode, anem a veure com s'ha fet la implementació.

5.1 Diagrama de classes

L'estructura del codi és la que podem veure a la imatge (3). El paquet central `UtilFunctions` és el que podem importar per a usar funcions ja predefinides. Aquest importa, segons la funció, els dos paquets que es veuen just a sobre, els quals contenen l'algortime de Runge-Kutta per a valors *double* (`RKV78`) i l'algortime per a sèries (`RKV78_Serie`). Aquest últim usa les funcions que es troben a la classe `Serie`, les quals defineixen els constructors i els *getters* i *setters*, així com les operacions per a treballar amb sèries. Com ja hem comentat, això últim ho fem gràcies a la sobrecàrrega d'operadors de C++.

Per últim, aquests dos paquets usen les equacions diferencials definides a la part inferior de la imatge. Aquestes —segons si treballen amb sèries o amb *doubles*— hereten d'una classe abstracta, la qual no defineix cap equació diferencial però indica el format que han de seguir les classes que hereten d'ella. Això segueix un patró *Strategy*.

Si entrem a valorar el contingut de cada element i no la seva connexió, podem observar en primer lloc que la classe `Serie` ens ofereix tres constructors: un buit, el qual crea un

polinomi de zeros de grau predeterminat a l'arxiu main; un on es passa un enter, el qual crea un polinomi zero d'aquella mida; i per últim un que rep tant el grau com els coeficients a usar. Aquests dos parametres són els que veiem com a *private* a l'apartat de coeficients.

Quant a les equacions diferencials, aquestes només consten d'una funció anomenada `differential_equation`, la qual rep el punt inicial entre altres paràmetres i efectua els càlculs necessaris per a avaluar el camp en aquella posició.

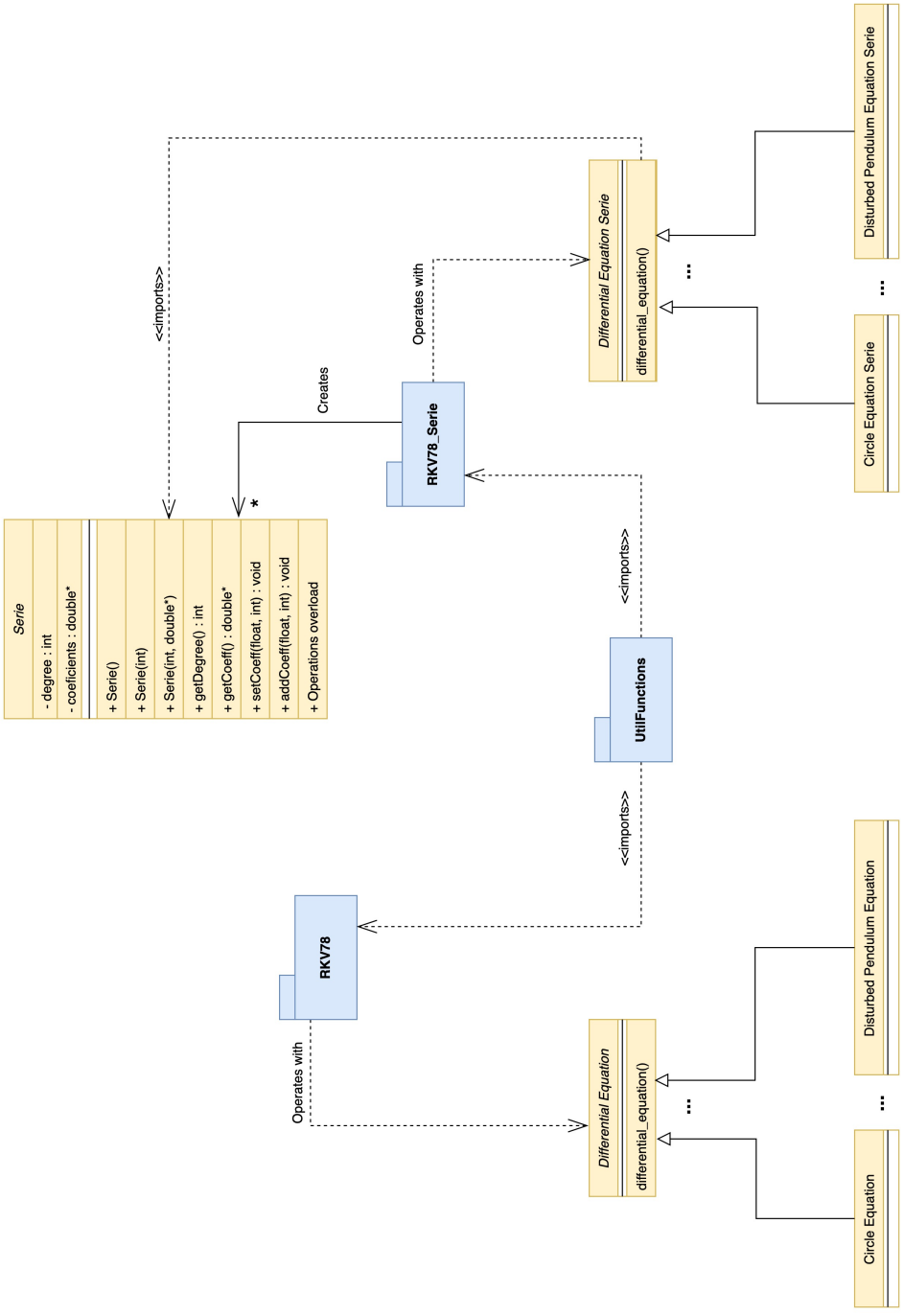


Figura 3: Diagrama de classes de l'aplicació

Per últim, cal comentar els arxius `utils.cpp` i `utils.h`, que contenen funcions:

1. `buscar_punt_fix`
2. `buscar_vap_vep`
3. `buscar_varietat`

les quals donades unes condicions inicials i una equació diferencial et troben, respectivament, el punt fix, els valors i vectors propis, i la varietat del valor propi triat.

5.2 Complexitat teòrica

Si volem parlar de la complexitat del nostre codi cal que parlem primer de la complexitat dels mètodes de Runge-Kutta. Ja hem comentat que els mètodes Runge-Kutta es basen en l'avaluació del camp en diversos punts a fi d'aconseguir una millor aproximació. Aleshores, és obvi que un paràmetre de la nostra complexitat serà el nombre d'avaluacions del camp que fem.

Per altra banda, cal tenir en compte que els mètodes de Runge-Kutta executen aquestes avaluacions del camp des d'un temps inicial fins a un temps final. Per això el nombre de passos també afecta de manera lineal a la complexitat del mètode. Un fet que estem obviant en aquesta simplificació és que estem suposant que usem el mètode amb un pas fix, sense variar la mida de la variable h. Si ho tinguéssim en compte la complexitat aleshores depèn de molts paràmetres externs:

1. Tolerància demanada: és obvi que si demanem un error menor ens cal fer passos més petits, augmentant així el nombre d'avaluacions del camp.
2. Equació diferencial: aquesta també influeix, ja que no és el mateix una equació que sigui molt "suau"en tot el camp que una que exhibeixi un comportament més aviat "caòtic". Afectarà a la mida del pas, i per tant al nombre d'avaluacions
3. Punt inicial: de la mateixa manera que l'equació diferencial, segons el punt que triem, la seva òrbita pot ser més o menys suau, fent així una integració més o menys ràpida.

És per això que només valorarem el cas de pas fix, ja que l'altre es troba fora de l'abast d'aquest treball. Aleshores, un mètode de Runge-Kutta amb s etapes, començant a x_0, acabant exactament a x_f i amb mida de pas h, té una complexitat de $O(sp)$ on $p = \left\lceil \frac{|x_f - x_0|}{h} \right\rceil$ (i on la notació $O(\cdot)$ indica la notació asimptòtica).

Si ara centrem aquesta explicació en el nostre algoritme, obtenim que la nostra complexitat es comportarà com $O(p)$, on el valor p variarà entre cada condició sobre el pas o el valor d'aturada que triem. Notem que ja no hi apareix el valor s; això és degut al fet que és constant per a totes les proves que fem, així que no té cap sentit tenir-lo en compte. Veurem una comprovació d'aquest fet a l'apartat de validacions.

5.3 Millores i optimitzacions

Una de les millores que es pot fer és respecte als arxius `rkv78` i `rkv78_serie`. Al codi actual aquests dos arxius defineixen dues llibreries amb funcions estàtiques, les quals es poden cridar com si fossin mòduls de C++. Però idealment aquests dos paquets haurien de ser dues classes. Això és degut a que les funcions de l'arxiu comparteixen variables que usen, i aquest fet és una mala pràctica si aquestes funcions no formen part de la mateixa classe, o com a mínim té més sentit si hi formen part.

El desenvolupament dels objectes d'aquests arxius es va intentar fer però per problemes de dependències cícliques no es va poder realitzar de manera efectiva, així que es va optar per una solució híbrida. El problema va ser que no es tenia planificada l'estructura del codi abans de començar a programar, i va ser un cop amb el codi gairebé enllestit que es va intentar canviar. Això va generar molts reptes a nivell d'estructura dels arxius que no es van poder resoldre garantint un bon funcionament del codi.

Una altra millora que es podria realitzar després d'aquest canvi a objectes seria valorar usar el patró *builder*, el qual permetria crear els dos objectes d'una manera similar, ja que al cap i a la fi tenen continguts molt similars. Per altra banda també podríem decidir si usar un patró *singleton* pot ser útil, ja que a priori sembla que no tingui gaire sentit tenir dues instàncies d'un objecte `rkv78`, per exemple.

Una altra millora que es podria fer és fer la introducció de dades al programa de manera externa. Ara mateix per a canviar els paràmetres d'execució cal obrir l'arxiu `main.cpp` i decidir tant els valors de les variables com les funcions a usar. Això genera que s'hagi de compilar l'arxiu cada cop que es vol canviar un paràmetre. Tot i això, i a mode de justificar la situació actual volem comentar que quan es vol programar una nova equació diferencial ens cal obrir i modificar els arxius `differential_equation.cpp /.h`. És per això que si uses el programa com a desenvolupador no hauria d'haver-hi cap problema.

Tot i això, si es fa servir com a usuari sí que és interessant oferir una abstracció que permeti treballar sense haver de compilar el codi. Podríem aleshores pensar en un arxiu `txt` el qual guarda tots els paràmetres a usar. O pensar també d'introduir-los per línia de comandes, indicant quins valors estem usant.

Tenint en compte tot això, és possible que la millor opció —tot i que la més costosa a nivell de temps de desenvolupament— sigui generar una interfície gràfica què permetés indicar l'equació diferencial a usar, els paràmetres i oferís una sortida gràfica de l'òrbita seguida per la condició inicial demanada.

6 Simulacions i resultats

6.1 Comprovació de la correcta implementació de l'algoritme

Com s'ha comentat abans, el mètode de Runge-Kutta-Verner que hem programat té un total de 127 valors que s'han de passar a net des d'una taula de manera correcta (els valors es poden consultar a la taula de l'annex). És per això que és realment important comprovar que funciona de la manera esperada. És per això que hem fet una prova usant l'equació diferencial:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -x \end{cases}$$

La solució general d'aquesta equació sabem que és $x(t) = \sin(t), y = \cos(t)$ (suposant condicions inicials $x(0) = 0, y(0) = 1$). És a dir, que si comencem en la condició inicial, hauríem d'esperar que, amb poc error, el mòdul dels punts trobats hauria de mantenir-se igual a 1.

Es va programar l'algoritme per tal que comencés al punt $(1, 0)$ i funcionés des de $t_0 = 0$ fins a $t_f = 40\pi$, és a dir, 20 voltes al cercle. Un cop fets els càlculs, el punt final on arribem és el $(1+4*10^{-15}, -2.73*10^{-11})$. Podem veure que, si després de 20 voltes l'error és de l'ordre de 10^{-11}, podem assegurar que no ens hem equivocat copiant els valors i programant l'algoritme.

Com a últim apunt volem comentar que en el total de 12568 punts calculats fins a donar les 20 voltes, els errors dels radis es troben en l'interval $(-64 * 10^{-16}, 82 * 10^{-16})$. Així si grafiquem els punts veurem un cercle perfecte, com ens mostra la figura (4).

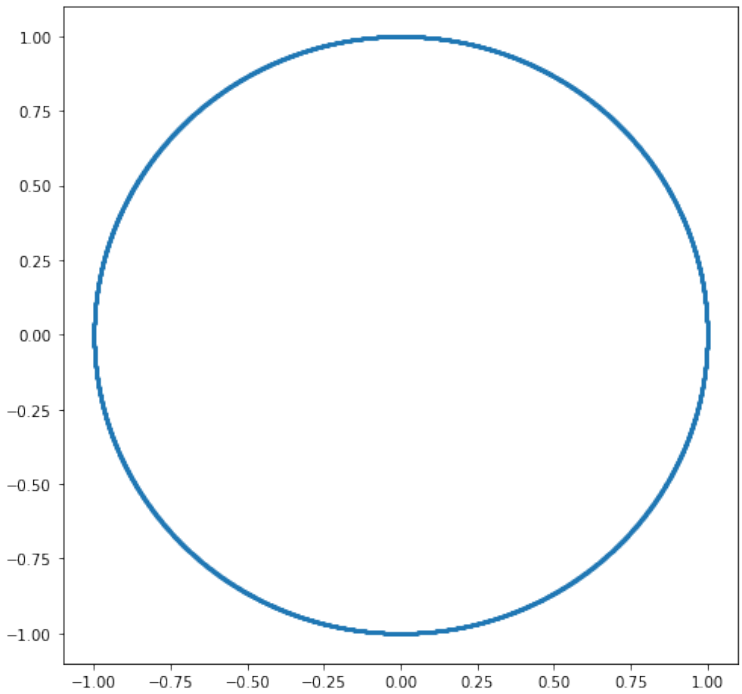


Figura 4: Comprovació del radi durant 20 voltes

6.2 Control de pas

Pensem en la equació del pèndol pertorbat que hem comentat abans:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -\sin x + \varepsilon \sin \omega t \end{cases}$$

amb condició inicial $(0, 0)$. Fent càlculs des de $t_0 = 0$ fins a $t_f = 10T$ —recordem que anteriorment hem definit T com el període de la funció, igual a $2\pi/\omega$— podem calcular que ens caldràn 4443 passos de l'integrador usant un pas constant de $h = 0.01$. El recorregut del punt és el que es troba a la Figura (5) (usant $\omega = \sqrt{2}$ i $\varepsilon = 0.1$). Notem que aquests passos surten del càlcul $\left\lceil \frac{10T}{h} \right\rceil = \lceil 4442.88 \rceil = 4443$.

En canvi, si demanem una tolerància igual a 10^{-14}, de manera experimental ens calen només 465 iteracions, unes 9.5 vegades menys passos de l'integrador que usant el pas fix. El moviment del punt és el que podem veure a la Figura (6).
Si insertem en un gràfic els passos que succeeixen obtenim les figures (8a) i (8b), que mostra els 10 primers valors del pas.

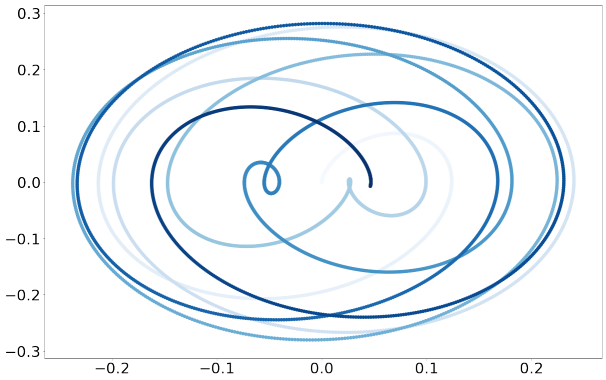


Figura 5: Moviment del punt (0,0) amb $h = 0.01$

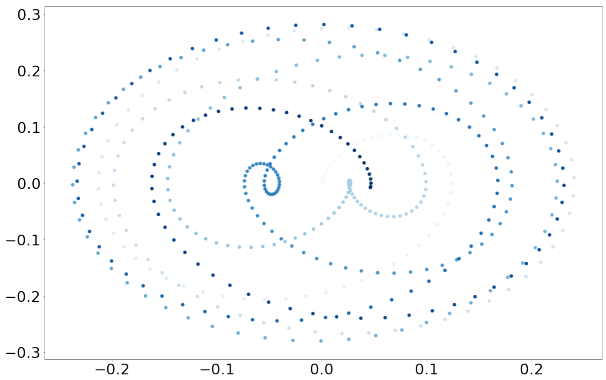


Figura 6: Moviment del punt (0,0) amb control de pas

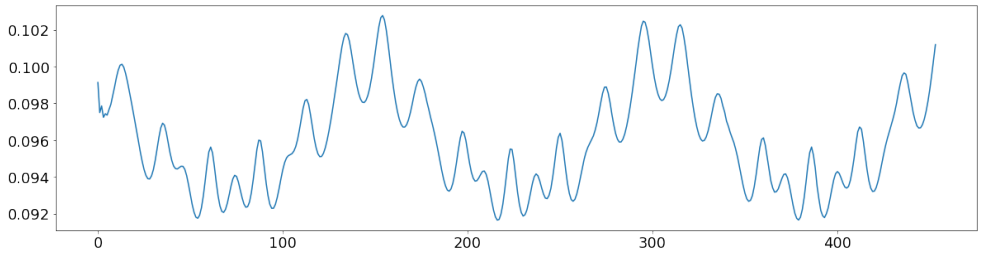
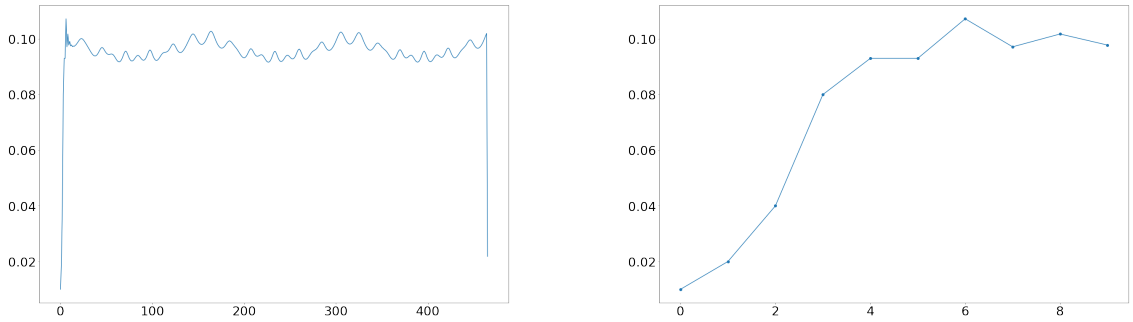


Figura 7: Variació del pas després dels 5 passos inicials



(a) Valors de pas

(b) Primers 10 valors de pas

Figura 8: Gràfics relacionats amb el control de pas

6.3 Punts fixos segons el paràmetre ε

Com ja hem demostrat abans de manera teòrica, aquesta equació té punts fixos per a valors d'ε propers a zero. El nostre objectiu serà comprovar que, efectivament, aquests punts existeixen de manera pràctica tot trobant els seus valors. A la següent taula veiem els resultats de buscar el punt fix usant el mètode de Newton per a matrius 2×2.

La complexitat de buscar els punts fixos amb el mètode de Newton depèn de dos factors: la precisió demanada i la complexitat de càlcul de la funció. Podem suposar que el temps de càlcul del camp a cada punt no variarà tot i tenir un ε diferent. En canvi, segons la precisió que demanem sí que hauria de canviar.

Sabem que el mètode de Newton té una complexitat quadràtica, és a dir, redueix l'error a la meitat en un entorn de convergència. És per això que, donada una bona aproximació inicial, hauríem d'esperar una complexitat de l'ordre $O(\log_2(N))$, on N és el nombre de xifres decimals que volem correctament respecte del paràmetre d'iteracions.

El valor per a $\varepsilon = 0$ és trobat de manera teòrica. Un apunt a fer és que el càlcul del temps s'ha fet amb una mitjana de 10 execucions.

Valor d'ε	Xifres decimals	Punt fix trobat	Temps	Iteracions	log_2(Xifres decimals)
0	valor exacte	$(-\pi, 0)$	-	-	-
0.02	5		$0.071 \times 10^{-3} s$	3	2.321
0.02	10	$(-3.14159265358979\mathbf{4}, -0.00942810511855\mathbf{2})$	$0.433 \times 10^{-3} s$	4	3.321
0.02	15		$1.910 \times 10^{-3} s$	5	3.906
0.04	5		$0.091 \times 10^{-3} s$	3	2.321
0.04	10	$(-3.14159265358979\mathbf{3}, -0.01885629845\mathbf{48})$	$0.449 \times 10^{-3} s$	4	3.321
0.04	15		$2.093 \times 10^{-3} s$	5	3.906
0.05	5		$0.189 \times 10^{-3} s$	4	2.321
0.05	10	$(-3.14159265358979\mathbf{4}, -0.02357045774\mathbf{38})$	$0.423 \times 10^{-3} s$	5	3.321
0.05	15		$2.272 \times 10^{-3} s$	5	3.906
0.06	5		$0.107 \times 10^{-3} s$	4	2.321
0.06	10	$(-3.14159265358979\mathbf{3}, -0.02828466823\mathbf{345})$	$0.357 \times 10^{-3} s$	5	3.321
0.06	15		$1.430 \times 10^{-3} s$	5	3.906
0.08	5		$0.180 \times 10^{-3} s$	4	2.321
0.08	10	$(-3.14159265358979\mathbf{3}, -0.03771330269\mathbf{249})$	$0.487 \times 10^{-3} s$	5	3.321
0.08	15		$2.049 \times 10^{-3} s$	6	3.906
0.1	5		$0.210 \times 10^{-3} s$	5	2.321
0.1	10	$(-3.14159265358979\mathbf{4}, -0.04714229009\mathbf{0529})$	$0.600 \times 10^{-3} s$	6	3.321
0.1	15		$1.943 \times 10^{-3} s$	6	3.906

Notem que, tot i que en el cas d'$\varepsilon = 0.02$ sí que veiem una tendència similar, en els altres valors no; això pot ser degut a dos factors. El primer, i possiblement el més important, és

el fet que cada valor d'ε té un punt fix diferent, així que usar la mateixa aproximació pot ser contraproduent. La segona és el fet que amb tan poques iteracions es fa difícil veure una millora substancial en el càlcul de l'algoritme.

6.4 Comprovació de punt fix

També podem comprovar si el punt trobat per la funció `buscar_punt_fix` efectivament és el punt per l'aplicació de retorn que hem comentat anteriorment. Si dibuixem la òrbita de la condició inicial $(0, 0)$ no és periòdica, com podem veure a la imatge (9). En canvi, si calculem l'òrbita del punt trobat per la funció, $(-3.141592653589794, -0.02129588804090621)$ en aquest cas amb $\varepsilon = 1, \omega = \sqrt{2}$) podem comprovar que efectivament el punt torna a la condició inicial després de $\dfrac{2\pi}{\omega}$ (Imatge (10)).

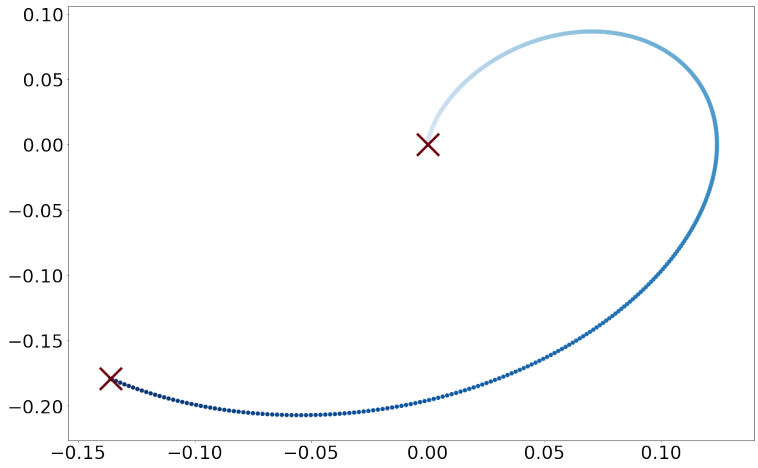


Figura 9: Moviment del punt $(0, 0)$ per l'aplicació de retorn

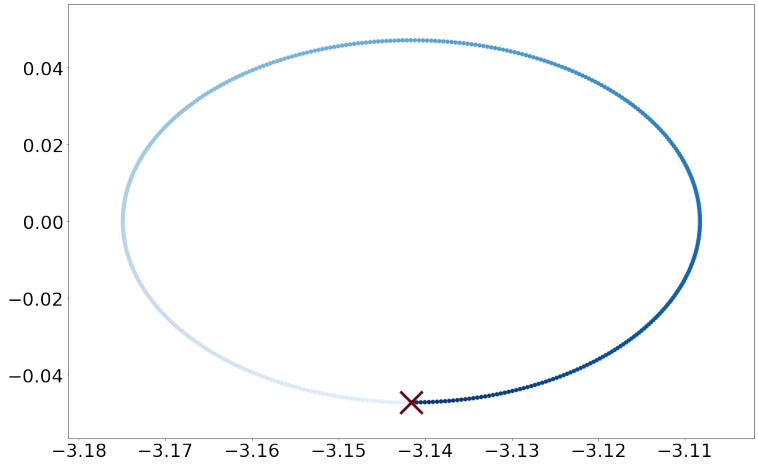


Figura 10: Moviment del punt fix per l'aplicació de retorn

6.5 Validació dels temps d'execució

Ja hem comentat a la secció anterior que a nivell teòric esperem una complexitat com $O(p)$ (on p són el nombre de passos) si usem un pas d'integració fix. Ho podem comprovar a la següent taula, la qual mostra diverses execucions amb diferents passos a realitzar

t_0	(x_0, y_0)	h	t_f	Passos a realitzar	Temps	Proporció
0	$(0,0)$	0.05	π/ω	45	$95.7 * 10^{-6} s$	$0.47 * 10^6$
			$2\pi/\omega$	89	$190.6 * 10^{-6} s$	$0.46 * 10^6$
			$4\pi/\omega$	178	$417 * 10^{-6} s$	$0.42 * 10^6$
		0.01	π/ω	223	$0.60 * 10^{-3} s$	$0.37 * 10^6$
			$2\pi/\omega$	445	$1.12 * 10^{-3} s$	$0.39 * 10^6$
			$4\pi/\omega$	889	$2.29 * 10^{-3} s$	$0.38 * 10^6$
	$(-\pi, 0)$	0.05	π/ω	45	$97.7 * 10^{-6} s$	$0.46 * 10^6$
			$2\pi/\omega$	89	$193.1 * 10^{-6} s$	$0.46 * 10^6$
			$4\pi/\omega$	178	$393.2 * 10^{-6} s$	$0.45 * 10^6$
		0.01	π/ω	223	$0.70 * 10^{-3}$	$0.31 * 10^6$
			$2\pi/\omega$	445	$1.24 * 10^{-3} s$	$0.35 * 10^6$
			$4\pi/\omega$	889	$2.25 * 10^{-3} s$	$0.39 * 10^6$
usant $\omega = \sqrt{2}$						

Com podem veure, i com ja havíem predit anteriorment, segueix una relació lineal en funció del nombre d'avaluacions com indica la columna Proporció. Podem notar que variar el punt inicial no canvia el temps d'execució del programa.

6.6 Càlcul de les varietats lineals

En un apartat anterior hem comentat que ens interessaria trobar la parametrització de les varietats lineals tal com s'ha exposat abans. Un cop tenim l'aritmètica de sèries ben programada i l'algoritme amb sèries funcionant, ens podem ocupar de trobar-les. Començarem per la varietat inestable.

El primer que ens cal fer, com ja s'ha vist, és crear un polinomi de grau 2 per a cada component: en aquest cas els anomenarem `x_pol` i `y_pol`. Sabem que els valors de grau zero i u són, respectivament, el punt fix i el vector propi del valor propi a usar. Per ara:

```
x_pol[0] = -3.141592653589794;
x_pol[1] = 0.7071077356024121;
x_pol[2] = 0;
y_pol[0] = -0.02129588804089878;
y_pol[1] = 0.7071058267693946;
y_pol[2] = 0;
```

Un cop haguem calculat els passos d'integració necessaris per a un període amb aquests valors arribarem als mateixos polinomis amb els coeficients de segon grau canviats. Recordem aleshores que aquests valors d'arribada (`x_pol[2]` i `y_pol[2]`) no són els valors de grau 2, ens cal fer el càlcul:

$$\texttt{x\_pol[2]} \texttt{ <- } -\left[DP(a_0) - \lambda^{m+1}\,\mathrm{Id}\right]^{-1}\texttt{x\_pol[2]}$$

$$\texttt{y\_pol[2]} \texttt{ <- } -\left[DP(a_0) - \lambda^{m+1}\,\mathrm{Id}\right]^{-1}\texttt{y\_pol[2]}$$

Un cop tenim al coeficient de segon grau el valor correcte del polinomi, podem fer el mateix amb grau tres i successivament.

Nosaltres en aquest exemple hem calculat fins a grau 10, i els valors han estat els següents:

```
x_pol[0] = -3.141592653589786;        y_pol[0] = -0.02129588804089878;
x_pol[1] = 0.7071077356024121;        y_pol[1] = 0.7071058267693946;
x_pol[2] = -3.496859360156828e-05;    y_pol[2] = -0.0002185541662619199;
x_pol[3] = -0.007365730788913106;     y_pol[3] = -0.02209714314709052;
x_pol[4] = 1.778586529594716e-06;     y_pol[4] = 9.31873943827724e-06;
x_pol[5] = 0.0001381079512068035;     y_pol[5] = 0.0006905382677832601;
x_pol[6] = -5.396436265220213e-08;    y_pol[6] = -3.389169248162155e-07;
x_pol[7] = -3.082775955615545e-06;    y_pol[7] = -2.15793840525362e-05;
x_pol[8] = 1.567185816050616e-09;     y_pol[8] = 1.250652489083904e-08;
x_pol[9] = 7.492875413907507e-08;     y_pol[9] = 6.743573175566952e-07;
x_pol[10]= -4.669277035244308e-11;    y_pol[10]= -4.62733641179625e-10;
```

Aleshores amb el polinomi trobat comprovem que, efectivament, aquest parametritza la varietat inestable. Hem vist abans que si substituïm un valor `s`, calculem el resultat del polinomi i aquest l'integrem durant un període, arriba finalment al punt que surt de substituir al polinomi el valor `lambda*s`. Això es va comprovar triant un valor qualsevol com `s = 0.005`, calculant tot un període i comparant-ho amb el punt que sortia de calcular amb `lambda*0.05` (en aquest cas `lambda = 4.075359500917643`). Els resultats d'aquests dos càlculs són els següents:

```
punt_iterat   = [-3.127184139285571,-0.006887613495113405]
punt_calculat = [-3.127184139285594,-0.006887613495136298]
```

Així, podem assegurar que, efectivament, la varietat ha estat correctament calculada. Aleshores, i com ja hem avançat a l'apartat sobre el mètode de la parametrització, podem agafar un conjunt de punts entre el valor de `p(s)` i `p(lambda*s)` (on `p` és el nostre polinomi d'interpolació) i els podem anar iterant per a dibuixar assegurant menys error en els nostres càlculs. Així, aquesta primera imatge (11) ha estat generada agafant 100 punts entre `p(0.005)` i `p(lambda*0.005)` i fent 4 períodes d'integració. La imatge (12) ha estat generada fent 7 iteracions i calculant també la varietat amb `-0.005`.

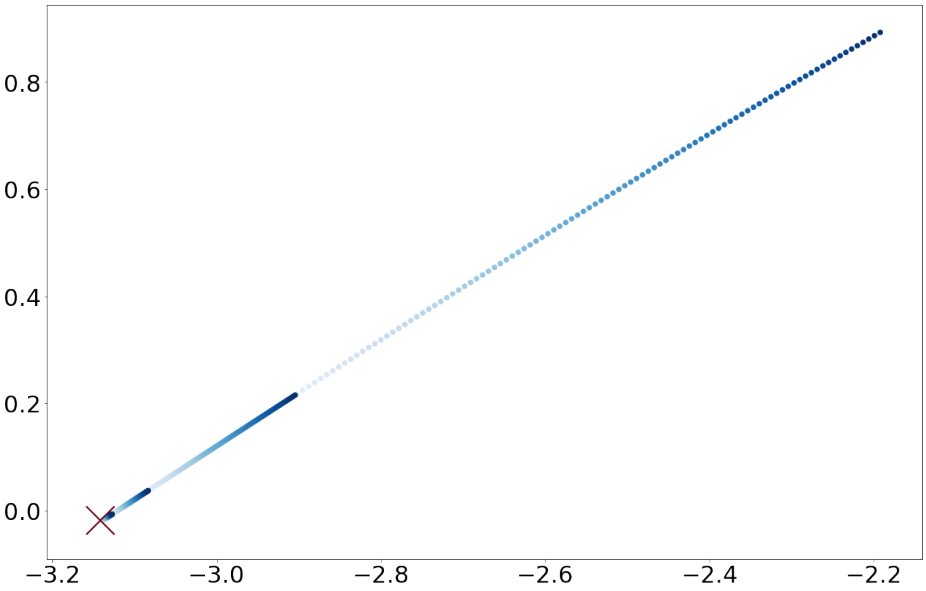


Figura 11: Varietat inestable fent 4 iteracions

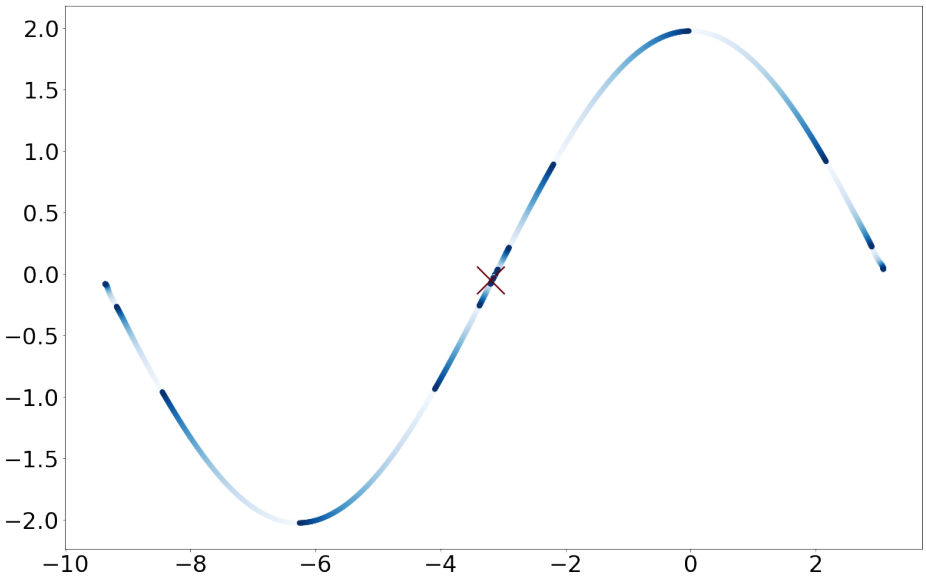


Figura 12: Varietat inestable fent 7 iteracions

Abans també hem comentat que, per a dibuixar, aquest mètode era millor que calcular-ho. I això ho podem comprovar amb la figura (13), la qual ens mostra els punts com a la imatge (12), però en vermell veiem graficada una part del polinomi.

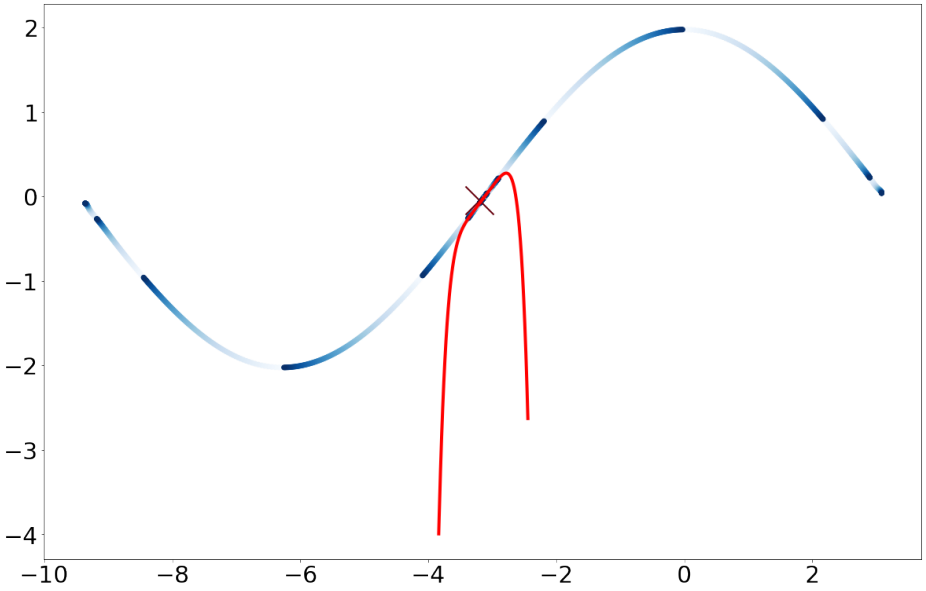


Figura 13: Comparació del polinomi amb els punts integrats

Treballant de la mateixa manera, podem buscar la varietat estable del punt fix, és a dir, usant el valor propi menor a 1. En aquest cas tenim `lambda = 0.2453771255701074`. Els valors trobats del polinomi són els següents:

```
x_pol[0] = -3.141592653589765;      y_pol[0] = -0.02129588804087706;
x_pol[1] = 0.7071077356024146;      y_pol[1] = -0.707105826769392;
x_pol[2] = 3.496859359900253e-05;   y_pol[2] = -0.0002185541662566978;
x_pol[3] = -0.007365730788911947;   y_pol[3] = 0.02209714314709049;
x_pol[4] = -1.778586529518625e-06;  y_pol[4] = 9.318739437973664e-06;
x_pol[5] = 0.0001381079512050267;   y_pol[5] = -0.0006905382677849704;
x_pol[6] = 5.396436264714549e-08;   y_pol[6] = -3.389169248053731e-07;
x_pol[7] = -3.082775955007396e-06;  y_pol[7] = 2.157938405282546e-05;
x_pol[8] = -1.567185813825308e-09;  y_pol[8] = 1.250652489207647e-08;
x_pol[9] = 7.492875399850557e-08;   y_pol[9] = -6.743573176381693e-07;
x_pol[10]= 0.0009923675597056651;   y_pol[10]= 0.001150334393104366;
```

Igual que abans i agafant 100 punts en l'interval entre `p(s)` i `p(lambda*s)`, obtenim la imatge (14) fent 7 iteracions. De la mateixa manera, hem agafat `s = 0.05` per a fer una varietat i `s=-0.05` per a fer l'altra. Per a generar les dades d'aquesta imatge hem hagut d'usar l'algoritme usant un temps negatiu, ja que els punts en aquest cas s'apropen al punt fix. Tot i això, aquesta situació no és gaire problemàtica, ja que l'algoritme programat té en compte aquesta situació: si es crida amb un pas `ah` menor a zero, l'algoritme funciona com hauria i retorna el següent pas menor a zero. L'únic canvi que s'ha hagut de fer és cridar a la funció de la llibreria *útils* que permet treballar amb temps negatius. L'única diferència entre aquesta funció i l'anterior és que la comparació entre el temps inicial i final s'ha de fer de manera inversa (volem seguir integrant si `at > atf`).

A la imatge (15) podem veure les dues varietats alhora, amb la varietat estable en vermell i la inestable en blau.

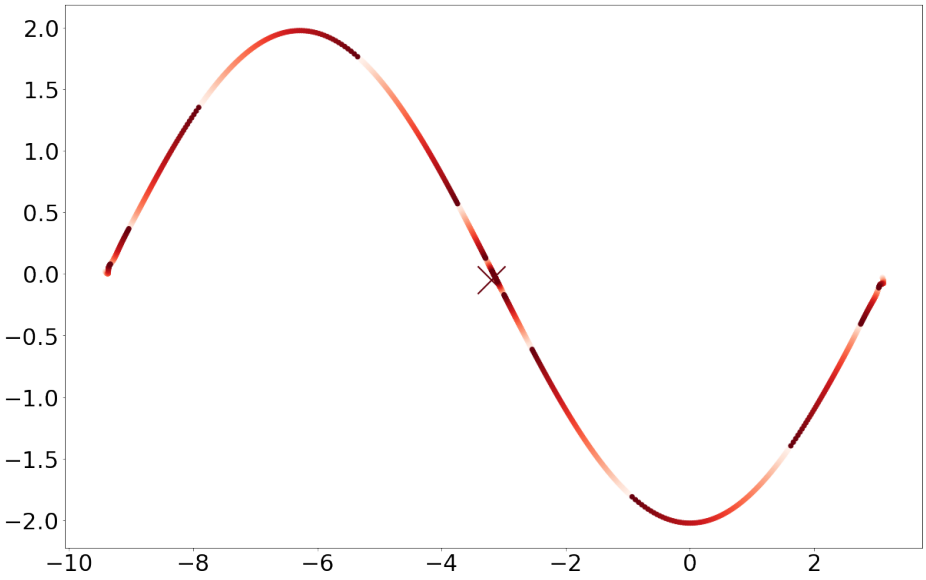


Figura 14: Varietat estable fent 7 iteracions

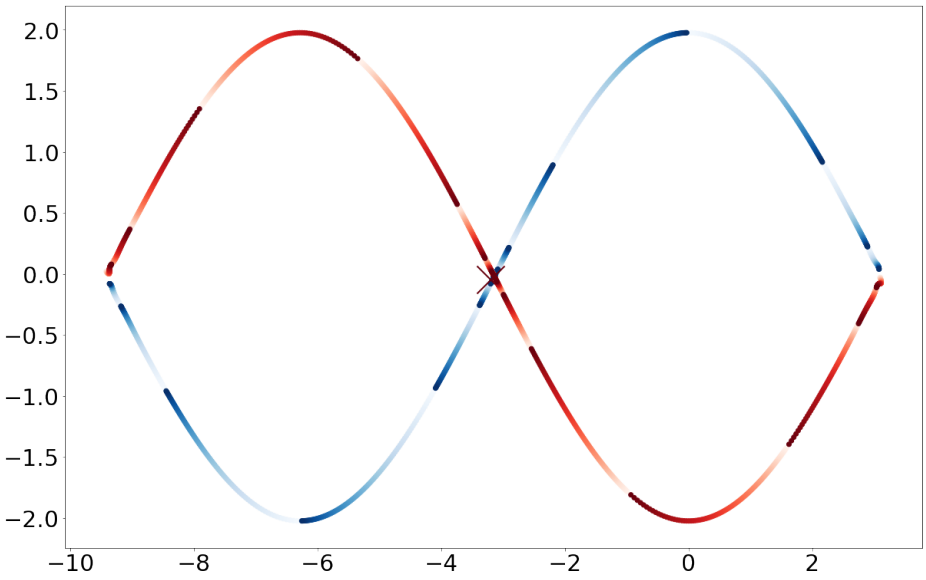


Figura 15: Varietat estable i inestable fent 7 iteracions

Un últim comentari a fer sobre aquests valors és que són els coeficients que parametritzen la varietat inestable usant $\omega = \sqrt{20}$. S'ha fet amb aquest número i no amb el valor de tot el treball de $\sqrt{2}$ ja que en aquest cas tenim el valor propi major a 1 amb resultat $\lambda \simeq 4.07$, mentre que si treballem amb el valor $\sqrt{2}$ obtenim un valor propi proper a 85. Això ens feia el dibuix posterior de la varietat molt menys esclaridor, ja que la longitud de cada nou sector augmentava 85 vegades.

7 Conclusions

Un cop acabat el treball és moment de fer balanç de què s'ha aconseguit en aquest treball i si s'han acomplert els objectius. Quant a l'objectiu d'estudiar l'existència i el comportament dels punts fixos en l'equació diferencial triada podem dir que s'han acomplert: hem pogut demostrar mitjançant el teorema de la funció implícita que existeixen.

Quant a l'estudi dels mètodes de Runge-Kutta també creiem que s'han acomplert amb escreix, ja que hem arribat a entendre i a generar mètodes a partir d'ordres donats. Per últim hem assimilat les variants d'aquests mètodes que usen un sistema de control de pas.

Si valorem els objectius informàtics d'aquest treball, ens atrevim a dir que també s'han acomplert de manera satisfactòria. En primer lloc, l'integrador numèric `rkv78` hem vist que funciona de la manera esperada i, per tant, podem usar-lo per a futurs projectes. Per últim, l'aritmètica de sèries també s'ha pogut implementar de manera satisfactòria com hem pogut comprovar a l'hora de dibuixar-les.

Per últim, i respecte als objectius generals del treball, considerem que hem sigut capaços d'entendre i assimilar el contingut dels recursos que ens ha proposat el tutor. També hem sabut trobar de manera autònoma recursos per a solucionar problemes que hem anat tenint durant el desenvolupament d'aquest treball.

7.1 Propers passos

Amb les conclusions acabades, és interessant plantejar-nos quins podrien ser els propers passos d'aquest treball. Una primera idea seria usar aquest programa amb altres equacions diferencials interessants d'estudiar. Això no hauria de ser gens difícil: el programa és prou modular per a simplement introduir una nova classe que sigui l'equació diferencial desitjada. Una cosa que sí que s'hauria de programar de manera més general —per a permetre un ús més exhaustiu— és la funció de càlcul dels valors i vectors propis. En aquest treball només ho hem necessitat per a dimensió 2, però es podria fer extensible a un nombre arbitrari de dimensions, programant, segurament, un mòdul per a nombres complexos.

Una altra extensió que se'ns acut que es podria fer és un anàlisi exhaustiu d'ús de recursos d'aquest mètode respecte d'altres més senzills com els mètodes d'Euler o altres mètodes més similars en quant a potència com els de Taylor.

Referències

[1] Jim Verner, "Explicit Runge-Kutta methods with estimates of the local truncation error. (English)," *SIAM Journal on Numerical Analysis (SINUM)*, vol. 15, no. 4, pp. 772–790, 1978.

[2] Wikimedia Commons, "Pendulum phase portrait," 2017. Accessed: 2019-12-31.

[3] John C. Butcher, *Ordinary Differential Equations*. Hoboken, New Jersey: Wiley, 2008.

[4] Hairer et al., *Solving Ordinary Differential Equations I.* Berlin, Germany: Springer, 1993.

[5] David Griffiths and Desmond Higham, *Numerical Methods for Ordinary Differential Equations*. Berlin, Germany: Springer, 2010.

[6] Àngel Jorba and Maorong Zou, "A software package for the numerical integration of odes by means of high-order taylor methods," *Experimental Mathematics*, vol. 14, no. 1, pp. 99–117, 2005.

Part III

Annexos

A Valors del mètode de Runge-Kutta-Verner 7(8)

i \ j	c_i	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0													
2	$\frac{1}{4}$	$\frac{1}{4}$												
3	$\frac{1}{12}$	$\frac{5}{72}$	$\frac{1}{72}$											
4	$\frac{1}{8}$	$\frac{1}{32}$	0	$\frac{3}{32}$										
5	$\frac{2}{5}$	$\frac{106}{125}$	0	$\frac{-408}{125}$	$\frac{352}{125}$									
6	$\frac{1}{2}$	$\frac{1}{48}$	0	0	$\frac{8}{33}$	$\frac{125}{528}$								
7	$\frac{6}{7}$	$\frac{-1263}{2401}$	0	0	$\frac{39936}{26411}$	$\frac{-64125}{26411}$	$\frac{5520}{2401}$							
8	$\frac{1}{7}$	$\frac{37}{392}$	0	0	0	$\frac{1625}{9408}$	$\frac{-2}{15}$	$\frac{61}{6720}$						
9	$\frac{2}{3}$	$\frac{17176}{25515}$	0	0	$\frac{-47104}{25515}$	$\frac{1325}{504}$	$\frac{-41792}{25515}$	$\frac{20237}{145800}$	$\frac{4312}{6075}$					
10	$\frac{2}{7}$	$\frac{-23834}{180075}$	0	0	$\frac{-77824}{1980825}$	$\frac{-636635}{633864}$	$\frac{254048}{300125}$	$\frac{-183}{7000}$	$\frac{8}{11}$	$\frac{-324}{3773}$				
11	1	$\frac{12733}{7600}$	0	0	$\frac{-20032}{5225}$	$\frac{456485}{80256}$	$\frac{-42599}{7125}$	$\frac{339227}{912000}$	$\frac{-1029}{4180}$	$\frac{1701}{1408}$	$\frac{5145}{2432}$			
12	$\frac{1}{3}$	$\frac{-27061}{204120}$	0	0	$\frac{40448}{280665}$	$\frac{-1353775}{1197509}$	$\frac{17662}{25515}$	$\frac{-71687}{1166400}$	$\frac{98}{225}$	$\frac{1}{16}$	$\frac{3773}{11664}$	0		
13	1	$\frac{11203}{8680}$	0	0	$\frac{-38144}{11935}$	$\frac{2354425}{458304}$	$\frac{-84046}{16275}$	$\frac{673309}{1636800}$	$\frac{4704}{8525}$	$\frac{9477}{10912}$	$\frac{-1029}{992}$	0	$\frac{729}{341}$	
$\bar{b}_j$		$\frac{13}{288}$	0	0	0	0	$\frac{32}{125}$	$\frac{31213}{144000}$	$\frac{2401}{12375}$	$\frac{1701}{14080}$	$\frac{2401}{19200}$	$\frac{19}{450}$		
b_j		$\frac{31}{720}$	0	0	0	0	$\frac{16}{75}$	$\frac{16807}{79200}$	$\frac{16807}{79200}$	$\frac{243}{1760}$	0	0	$\frac{243}{1760}$	$\frac{31}{720}$

$$EE = \frac{-1}{480}g_1 - \frac{16}{375}g_6 - \frac{2401}{528000}g_7 + \frac{2401}{132000}g_8 + \frac{243}{14080}g_9 - \frac{2401}{19200}g_{10} - \frac{19}{450}g_{11} + \frac{243}{1760}g_{12} + \frac{31}{720}g_{13}$$

B Guia d'ús del programa

En aquest apartat de l'annex farem una petita guia de com funciona el codi per a usuaris. El desenvolupament d'aquesta guia seguirà el del treball: comença buscant el punt fix fins a calcular la varietat inestable. Per a obtenir més informació sobre els arxius que no s'han mencionat en aquest apartat es pot trobar a l'apartat de guia de desenvolupadors, més endavant en l'annex.

De tots els arxius que consta el treball, els dos únics que cal canviar són els arxius `main.cpp` i `differential_equation.cpp`. Narrarem aquest tutorial seguint un exemple, imaginant que volem acabar calculant la varietat inestable del punt fix del pèndol pertorbat.

$$\begin{cases} \dot{x} = y \\ \dot{y} = -\sin(x) + \varepsilon \sin(\omega t) \end{cases} \qquad \varepsilon = 0.1, \omega = \sqrt{2} \tag{B.1}$$

Primer de tot, hem de programar l'equació diferencial a l'arxiu que toca. Totes les equacions diferencials segueixen el patró *strategy*, així que les equacions diferencials hereten de dues classes abstractes segons si usen valors `double` o objectes sèrie. Per ara volem programar només la versió amb `double`, així que, tenint en compte que la nostra equació diferencial és (B.1), necessitem expressar això com veiem al codi.

```
class Disturbed_Pendulum: public differential_equation{

public:
    static void edo(double t, double x[], int n, double result[]){
        double epsilon = 0.1;
        double omega = sqrt(2);
        result[0] = x[1];
        result[1] = -sin(x[0]) + epsilon*sin(omega*t);
    }
};
```

Notem que l'array `x` són els punts d'entrada i l'array `result` són els diferencials. La variable `n` ens indica la dimensió dels punts, la longitud de la llista. Aquesta equació ens permet agafar una condició inicial i fer passos d'integració fins on volguem. Si seguim endavant en la llista de tasques que hem de fer, volem trobar el punt fix, per això ens cal programar una equació molt similar la qual tingui en compte la matriu diferencial per a poder aplicar el mètode de Newton, com hem vist ja a l'apartat sobre l'existència d'òrbites periòdiques. Recordem que la equació diferencial completa quedava com

$$\begin{cases} \dot{x} = y \\ \dot{y} = -\sin(x) + \varepsilon \sin(\omega t) \\ \dot{M}(t) = \begin{pmatrix} 0 & 1 \\ -\cos(x) & 0 \end{pmatrix} M(t) \end{cases} \qquad \varepsilon = 0.1, \omega = \sqrt{2} \tag{B.2}$$

Per això hem de definir una segona classe que tingui en compte també els valors de la matriu diferencial.

```
class Dist_Pendulum_Mat: public differential_equation{

public:
    static void edo(double t, double x[], int n, double result[]){
        double epsilon = 0.1;
        double omega = sqrt(2);
        result[0] = x[1];
        result[1] = -sin(x[0]) + epsilon*sin(omega*t);
        result[2] = x[4];
        result[3] = x[5];
        result[4] = -cos(x[0])*x[2];
        result[5] = -cos(x[0])*x[3];
    }
};
```

Un cop fet això, aleshores ja podem buscar el punt fix de l'equació diferencial. A l'arxiu `utils.cpp` hi ha una funció anomenada `buscar_punt_fix`, a la qual li podem passar el punter a una equació diferencial i ella busca el punt fix mitjançant el mètode de Newton. Tot i això per a cridar a la funció ens calen unes variables imprescindibles:

1. temps inicial: `double*`
2. condició inicial: `double[]`
3. dimensió punts: `int`
4. pas inicial: `double*`
5. flag sobre l'ús de pas variable: `int`
6. tolerància: `double*`
7. temps final d'integració: `double*`
8. flag sobre sortir si l'error és major a la tolerància: `double*`
9. equació diferencial: `void`

En el nostre cas, el contingut de les variables serà:

Posició	Variable	Valor
1	at	0
2	x	$\{-\pi, 0, 1, 0, 0, 1\}$
3	n	6
4	ah	0.01
5	sc	1
6	tol	10^{-16}
7	atf	$2\pi/\sqrt{2}$
8	aer	1
9	edo	`&Dist_Pendulum_Mat::edo`

ja que comencem en $t_0 = 0$, començarem pel punt $(-\pi, 0)$ ja que serà prou proper al nostre punt fix, la matriu comença a la identitat $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, la mida de l'array és 2+4, ja que tenim un punt de dues coordenades i una matriu de mida 2×2. El pas inicial que triem i la tolerància es poden canviar, ja que no afecten al resultat, simplement a la quantitat de punts calculats. El temps final l'agafem com la periodicitat de la funció. Per últim, el flag el triem 1 per tal que no pari si l'error es fa major del demanat i triem l'equació diferencial que volem usar.

Un punt a comentar sobre l'equació diferencial és que nosaltres li passem el punter de la funció, i ja el mètode de Runge-Kutta crida a aquella funció. Així que la nostra definició de la equació diferencial la farem com

```
// Si usem la equació amb doubles
void (*dif)(double, double *, int, double *) = &Dist_Pendulum_Mat::edo;

// Si usem la equació amb Series
void (*dif_s)(double, Serie *, int, Serie *) = &Dist_Pendulum_Serie::edo;
```

Quan aquesta funció hagi acabat, tindrem el punt fix que ha trobat a la nostra variable `x` com el punt `(x[0], x[1])`. Ara, i un cop trobat el punt fix, seguim endavant i ens cal trobar el valors valors propis i vectors propis del punt. Per això hi ha una altra funció a l'arxiu `utils.cpp` anomenada `buscar_vap_vep`, la qual troba els dos valors i els dos vectors propis (notem que, per tant, aquesta funció només s'ha programat per a casos on $n = 2$). Per a cridar-la ens cal passar-li les mateixes variables que a la funció explicada anteriorment però amb quatre variables afegides: `vap1, vap2, vep1, vep2`, les quals són, respectivament, `double, double, double[], double[]`. Aquestes a l'inici les creem buides i quan acaba la crida de la funció contenen els valors demanats, sempre que siguin els valors propis reals.

Un fet a comentar sobre aquesta funció és que només troba els valors si són valors reals. En aquest programa no hem afegit una funcionalitat de nombres complexos. És per això que en cas que els valors propis siguin imaginaris el programa imprimeix per pantalla un avís. En cas que els valors siguin reals, el programa troba els vectors propis amb el mètode de Cramer de resolució de sistemes. Aquest és un sistema que no és factible si tenim dimensions altes, però com que estem treballant amb punts de dues dimensions, aleshores hi ha una fórmula explícita ràpida de calcular.

Per últim, cal cridar la funció anomenada `buscar_varietat`, la qual es crida de la mateixa manera que abans, tot i que ens calen uns paràmetres extres:

1. Grau del polinomi: `int`
2. Flag indicant si volem trobar varietat estable o inestable: `int`
3. Equació diferencial amb sèries: `void`

En el nostre cas, el contingut de les variables serà:

Posició	Variable	Valor
1	degree	6
2	vap	0
3	edo_serie	`&Dist_Pendulum_Serie::edo`

ja que volem, per exemple, un polinomi de grau 6. Quant al vap podem triar si volem calcular la varietat estable o inestable, triant el valor zero o 1 respectivament.

En cas de voler usar simplement l'integrador numèric sense fer ús de les funcions esmentades a l'arxiu `utils.cpp`, la funció `rkv_78` es pot cridar directament des de l'arxiu `main.cpp`. Aquesta s'utilitza amb els mateixos paràmetres que la funció `buscar_punt_fix` i cada cop que es crida avalua un pas d'integració. De la mateixa manera, però usant la equació diferencial amb sèries, es pot cridar a la funció `rkv_78` (que en aquest cas anirem a la funció de l'arxiu `rkv78_serie.cpp`), que també avalua un pas d'integració, però treballant amb sèries.

B.1 Instal·lació

El programa funciona exclusivament amb C++ desenvolupat amb l'editor *CLion*, propietat de JetBrains. És per això que per a compilar-lo cal seguir les instruccions per defecte dels projectes d'aquest editor.

Primer de tot, ens situem a la carpeta on figura l'arxiu `main.cpp` de la nostra aplicació. Si no existeix la carpeta `cmake-build-debug`, la creem usant la comanda

```
$ mkdir cmake-build-debug
```

Un cop creada la carpeta, executem la comanda

```
$ [Ruta a l'aplicacio cmake] --build [ruta a la carpeta
↪  cmake-build-debug] --target clean -- -j [num. processadors]
```

Per últim, un cop feta aquesta comanda ens cal l'última instrucció, la qual compila els arxius:

```
$ [Ruta a l'aplicacio cmake] --build [ruta a la carpeta
↪  cmake-build-debug] --target [nom de la carpeta que conte el
↪  projecte] -- -j [num. processadors]
```

En el nostre cas, el nom de la carpeta és `clion_project`.

C Manual de desenvolupador

C.1 Documentació

Tota la documentació del codi s'ha fet amb el format *Doxygen*. Aquesta es troba en la mateixa carpeta del codi i no s'ha posat al treball ja que ocupa 42 pàgines i, al cap i a la fi, està només dirigida a qui vulgui treballar i desenvolupar a partir del codi ja proveït. Per a generar la documentació de manera automàtica cal seguir els següents passos: primer de tot, cal tenir instal·lat `doxygen` i un compilador de LaTeX. Si aquests requisits estan complerts, ens cal generar la documentació executant la següent comanda a la carpeta on es troba l'arxiu `Doxyfile`.

```
$ doxygen Doxyfile
```

Un cop fet això, ens generarà dues carpetes anomenades `html` i `latex`. La primera genera la documentació en format *HTML*, que es pot veure obrint l'arxiu `index.html` amb un navegador. Si entrem a la carpeta `latex` i executem la comanda

```
$ make
```

Generarem la documentació en un arxiu en format *PDF* anomenat `refman.pdf`.