Introduction to chaos: theoretical and numerical methods

Carlo F. Barenghi

September 2010

Contents

Preface

Chaos has been one of the most important recent discoveries of applied mathematics.

• **Philosophically**, the discovery of chaos meant the end of naive determininsm as originally envisaged by Newton and Laplace.

• **Practically**, chaos theory has applications ranging from weather forecasts to technology to physical and life sciences.

The discovery of chaos is also linked to the rise of **computational mathematics**, hence better ability to make **predictions**.

The aim of this module is to introduce chaos theory. We want:

• to determine **qualitative** but crucial aspects of the solutions of differential equations (for example whether these solutions are steady or oscillatory, decay to zero or grow to infinity), **without** actually solving the equations. We shall use **geometrical** tools instead.

• to develop **numerical methods** to solve these equations in a **quantitative** way. Most equations which have practical applications are **nonlinear**, thus very difficult or impossible to solve using pencil and paper. The ability to make predictions using computers has become an **essential skill** for applied mathematicians. It is therefore essential to learn some computer programming, in order to apply these numerical methods. The programming language chosen for this course is Fortran 90 - the superior language for scientific programming.

The following diagram outlines the structure of the module, and shows how analytical and numerical methods are integrated:

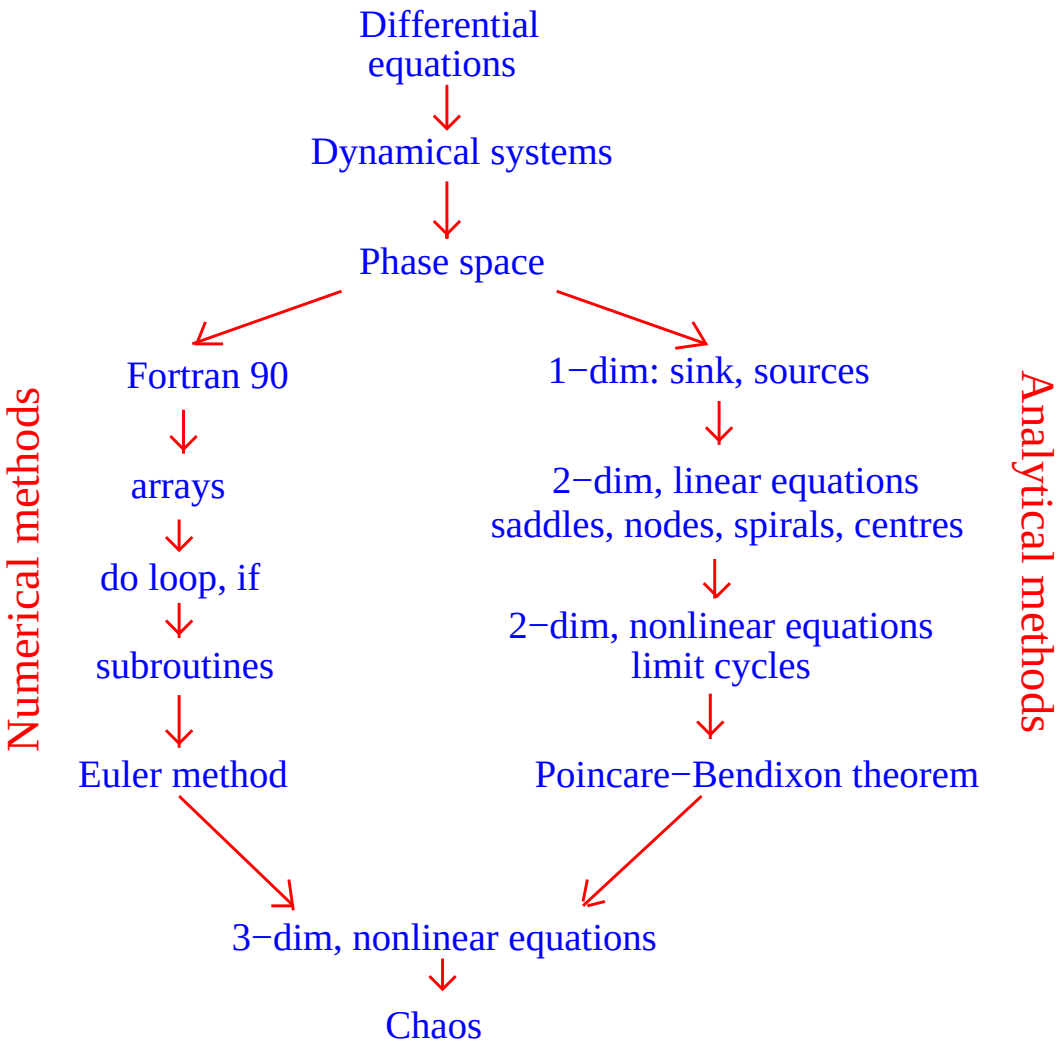


Figure 1:

The first part of this booklet covers the theoretical methods, and presents results of some applications in a qualitative way to broaden the student's horizons. The aim of the second part is to learn Fortran 90 programming and some computational mathematics following a simple hands-on approach. The second part of the booklet is thus meant to be read one-line, in front of a computer.

Students who want to know more about the topics discussed in this course

should consult the books of Drazin[2] or Strogatz[4], which describe in more detail the mathematics of dynamical systems and chaos. The remarkable characters and the events which led to the discovery of chaos are described in the easy-to-read book of Gleich[3]. The book of Brainard[1] is a good reference to programming with Fortran 90.

Finally, I would like to thank Anthony J. Mee for his help in developing the computational part of this course.

Part I

Analytical methods

Chapter 1

Phase space

1.1 Dynamical systems

Definition: We call a **dynamical system** of dimension N a system of N first–order differential equations for N variables $x_1(t), x_2(t), \cdots, x_N(t)$ which evolve with time t according to

$$\dot{x}_1 = f_1(x_1, x_2, ..., x_N, t), \tag{1.1}$$
$$\dot{x}_2 = f_2(x_1, x_2, ..., x_N, t), \tag{1.2}$$
$$\cdots, \tag{1.3}$$
$$\dot{x}_N = f_N(x_1, x_2, ..., x_N, t), \tag{1.4}$$

where $f_1, f_2, \cdots$ are assigned functions and a dot is a derivative with respect to time, eg $\dot{x}_1 = dx_1/dt$. In vector notation we have

$$\dot{\mathbf{x}} = \mathbf{f}, \tag{1.5}$$

where $\mathbf{x} = (x_1, x_2, ...x_N)$ and $\mathbf{f} = (f_1, f_2, ...f_N)$.

1.2 Critical points

Definition: A **critical point** (or **fixed point**, or **equilibrium point**) of the dynamical system $\dot{\mathbf{x}} = \mathbf{f}$ is a point $\mathbf{x}^*$ which satisfies

$$\mathbf{f}(\mathbf{x}^*) = \mathbf{0}. \tag{1.6}$$

A critical point is thus a place where $\mathbf{x}$ does not change with time.

1.3 Autonomous systems

Definition: Consider $\dot{\mathbf{x}} = \mathbf{f}$. If $\mathbf{f}$ depends only on $\mathbf{x}$, then the system is called **autonomous**. If $\mathbf{f}$ depends on both $\mathbf{x}$ and t, then the system is called **non-autonomous**.

Theorem: Any N-dimensional non-autonomous dynamical system can be transformed into an (N+1)-dimensional autonomous dynamical system.

The important property of autonomous systems is that trajectories do not intersect, hence solutions are unique. On the contrary, trajectories of non-autonomous systems can intersect, and solutions are not unique. Since any non-autonomous system of order N can be reduced to an autonomous system of order $N + 1$, then trajectories do not intersect in the $N + 1$ dimensional space, but may intersect in the N dimensional space.

Hereafter we shall be concerned with autonomous dynamical systems.

1.4 One–dimensional phase space

Consider the logistic equation

$$\frac{dx}{dt} = f(x) = x - x^2, \tag{1.7}$$

where $x = x(t) \geq 0$ is the population of a species; the equation describes a 1–dim dynamical system. **Phase space** consists of the semi–infinite plane $x \geq 0$. The fixed points are $x^* = 0$ and $x^* = 1$.

To each point of phase space we assign an arrow $\dot{x}$. The arrows form a 1-dim **vector field** $V = \dot{x}$, as shown in Fig. 1.1. To visualise the dynamical system, we imagine that a fluid flows in phase space with **phase velocity** given by $V = \dot{x}$. We think that a fluid particle, or **phase point**, is carried along by this flow.

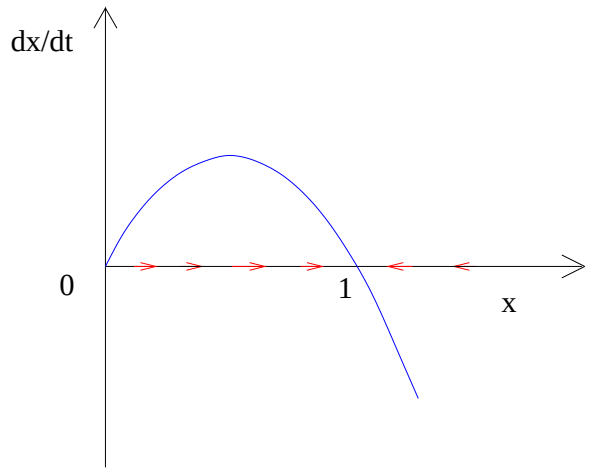


Figure 1.1: Vector field $\dot{x}$ (only few arrows are marked for clarity).

A phase point placed at initial location $x_0 < 1$ moves to the right (because $\dot{x} > 0$) until it stops at $x^* = 1$ (where $\dot{x} = 0$). A phase point which starts in the region $x_0 > 1$ moves to the left (because $\dot{x} < 0$) and also stops at $x^* = 1$. The motion of the phase point in phase space is a called a **trajectory**. Clearly the fixed point $x^* = 1$ **attracts** trajectories starting from both the right and the left. We say that $x^* = 1$ is **stable**: if we perturb slightly a phase point away from $x^* = 1$, the phase point will go back to $x^* = 1$. Vice versa the fixed point $x^* = 0$ is **unstable**: if we perturb slightly a phase point in the vicinity of $x^* = 0$, it will move away from x^*. We say that $x^* = 1$ is a **sink** and $x^* = 0$ is a **source**.

1.5 Two–dimensional phase space

Newton's equation $m\ddot{x} = -kx$ for a particle of mass m, position $x = x(t)$ and velocity $v = v(t)$ which is connected to the origin by a spring of stiffness k corresponds to the 2–dim dynamical system

$$\dot{x} = v, \tag{1.8}$$
$$\dot{v} = -\omega^2 x, \tag{1.9}$$

where $\omega^2 = k/m$. The (x, v) plane is the 2–dim phase space (not to be confused with the 1–dim physical space x). The phase velocity is the 2-dim vector field $\mathbf{V} = (\dot{x}, \dot{v})$ shown in Fig. 1.2 (not to be confused with the 1–dim physical velocity v). Fig. 1.3, which shows typical trajectories, is called the **phase portrait** of the dynamical system.

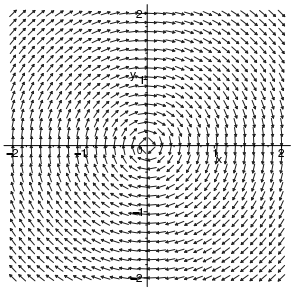


Figure 1.2: Vector field(only few arrows are marked for clarity).

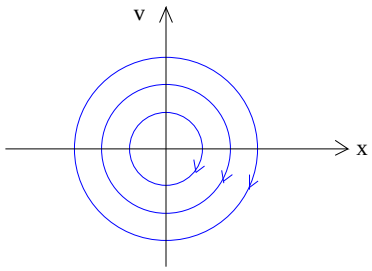


Figure 1.3: Phase portrait(only few trajectories are marked for clarity).

A phase point placed at any initial location goes around the origin with phase speed which increases with the distance from the origin; trajectories form **closed orbits**.

Fig.1.2. was made using the following *Maple* program:

```
ode1:=diff(x(t),t)=v(t):
ode2:=diff(v(t),t)=-x(t):
with(DEtools):
DEplot({ode1,ode2},[x(t),v(t)],t=0..1,x=-2..2,v=-2..2,
arrows=MEDIUM, dirgrid=[30,30],colour=blue);
```

1.6 Conservative systems

Consider a particle of mass m moving along x in the presence of a given force F. Newton's law $m\ddot{x} = F$ is equivalent to the 2–dim dynamical system

$$\dot{x} = v, \tag{1.10}$$

$$\dot{v} = \frac{F}{m}, \tag{1.11}$$

Theorem: Assume that $F = F(x)$ is independent of t and $\dot{x}$ (eg there is no damping, or friction, or time-dependent driving force), and that

$$F(x) = -\frac{d\phi}{dx}, \tag{1.12}$$

Then the quantity

$$E = E(x, v) = \frac{m}{2}v^2 + \phi(x), \tag{1.13}$$

called the **total energy**, is constant as a function of time along the trajectory of a phase point:

$$\frac{dE}{dt} = 0. \tag{1.14}$$

Chapter 2

Linear systems

2.1 Aim

Linear equations are the simplest to solve. The aim of this chapter is to introduce eigenvalues and eigenvectors of a linear system and classify the nature of fixed points.

2.2 Definition of linear dynamical system

Definition: An autonomous dynamical system $\dot{\mathbf{x}} = \mathbf{f}$ of dimension N is called **linear** if the function $\mathbf{f}$ is linear in $\mathbf{x}$, that is if $\mathbf{f}(\mathbf{x}) = A\mathbf{x}$, where A is an $N \times N$ constant matrix:

$$\dot{\mathbf{x}} = A\mathbf{x}. \tag{2.1}$$

Clearly the origin $\mathbf{x}^* = (0, 0, ..., 0) = \mathbf{0}$ is a fixed point.

2.3 The superposition theorem

Theorem: If both $\mathbf{x}$ and $\mathbf{y}$ are solutions of $\dot{\mathbf{x}} = A\mathbf{x}$, then any linear combination $\mathbf{z} = c_1\mathbf{x} + c_2\mathbf{y}$ with arbitrary coefficients c_1 and c_2 is also a solution.

2.4 Eigenvalues

Consider the general 2–dim linear system $\dot{\mathbf{x}} = A\mathbf{x}$:

$$\dot{x} = ax + by, \tag{2.2}$$
$$\dot{y} = cx + dy, \tag{2.3}$$

where a, b, c and d are constant, $\mathbf{x} = (x, y)$, $\dot{\mathbf{x}} = (\dot{x}, \dot{y})$ and

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}, \tag{2.4}$$

Assuming $\mathbf{x}(t) = \mathbf{X}e^{\lambda t}$ where $\mathbf{X} = (X_1, X_2)$ is a constant vector and λ is a parameter, we obtain the **eigenvalue equation** for the matrix A:

$$A\mathbf{X} = \lambda\mathbf{X}, \tag{2.5}$$

The trivial solution is $\mathbf{X} = (0, 0) = \mathbf{0}$. We rewrite the equation as

$$(A - \lambda I)\mathbf{X} = \mathbf{0}, \tag{2.6}$$

where I is the 2×2 identity matrix. The system

$$\begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} \begin{vmatrix} X_1 \\ X_2 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \end{vmatrix} \tag{2.7}$$

has non-trivial solution $\mathbf{X}$ if

$$det(A - \lambda I) = 0, \tag{2.8}$$

which is the **characteristic equation**

$$\lambda^2 - \tau\lambda + \delta = 0, \tag{2.9}$$

where

$$\tau = a + d = tr(A), \qquad \delta = ad - bc = det(A), \tag{2.10}$$

are respectively the **trace** and the **determinant** of the matrix A. The two roots

$$\lambda_1 = \frac{1}{2}[\tau + \sqrt{\tau^2 - 4\delta}\,], \qquad \lambda_2 = \frac{1}{2}[\tau - \sqrt{\tau^2 - 4\delta}\,], \tag{2.11}$$

of the characteristic equation are called the **eigenvalues** of the matrix A.

2.5 Eigenvectors

Suppose that $\tau^2 - 4\delta \neq 0$, so that $\lambda_1 \neq \lambda_2$. Since $det(A - \lambda I) = 0$, there exist non-zero vectors $\mathbf{U}$ and $\mathbf{V}$ such that

$$(A - \lambda_1 I)\mathbf{U} = \mathbf{0}, \tag{2.12}$$
$$(A - \lambda_2 I)\mathbf{V} = \mathbf{0}, \tag{2.13}$$

which is

$$A\mathbf{U} = \lambda_1 \mathbf{U}, \tag{2.14}$$
$$A\mathbf{V} = \lambda_2 \mathbf{V}, \tag{2.15}$$

The eigenvalue equation $A\mathbf{X} = \lambda\mathbf{X}$ has thus two solutions: $\mathbf{X} = \mathbf{U}$ and $\mathbf{X} = \mathbf{V}$. The vectors $\mathbf{U}$ and $\mathbf{V}$ are called the **eigenvectors** of the matrix A corresponding to the eigenvalues λ_1 and λ_2 respectively.

Scale factor:

The eigenvectors $\mathbf{U}$ and $\mathbf{V}$ are only determined up to a scale factor, because Eq. (2.5) is linear: if $\mathbf{U}$ is an eigenvector of A, that is, if $A\mathbf{U} = \lambda_1\mathbf{U}$, then, for any number $\alpha \neq 0$, the vector $\alpha\mathbf{U}$ is also an eigenvector of A, that is $A(\alpha\mathbf{U}) = \lambda_1(\alpha\mathbf{U})$.

Real and complex eigenvectors:

If $\tau^2 - 4\delta > 0$ then the eigenvalues λ_1 and λ_2 are both real, and so are the eigenvectors $\mathbf{U}$ and $\mathbf{V}$.

If $\tau^2 - 4\delta < 0$ then the eigenvalues λ_1 and λ_2 are complex conjugates of each other ($\overline{\lambda_1} = \lambda_2$, where the over-bar denotes the operation of complex conjugation); similarly $\mathbf{U}$ and $\mathbf{V}$ are complex conjugates of each other. However the combination

$$\mathbf{x}(t) = c_1\mathbf{U}e^{\lambda_1 t} + c_2\mathbf{V}e^{\lambda_2 t}, \tag{2.16}$$

must be real, because $\mathbf{x}(t)$ is real by definition. This implies that $\overline{c_1} = c_2$.

2.6 General solution

Suppose that $\tau^2 - 4\delta \neq 0$, so that $\lambda_1 \neq \lambda_2$. Then $\mathbf{x} = \mathbf{U}e^{\lambda_1 t}$ and $\mathbf{x} = \mathbf{V}e^{\lambda_2 t}$ are two linearly independent solutions. Application of the Superposition Theorem yields the general solution:

$$\mathbf{x}(t) = c_1\mathbf{U}e^{\lambda_1 t} + c_2\mathbf{V}e^{\lambda_2 t}, \tag{2.17}$$

where the constants c_1 and c_2 are determined by the initial condition

$$\mathbf{x}(0) = c_1\mathbf{U} + c_2\mathbf{V}, \tag{2.18}$$

The two components of this equation give two equations for the two unknowns c_1 and c_2.

2.7 Classification of solutions

The nature of the fixed point $\mathbf{x} = \mathbf{0}$ of $\dot{\mathbf{x}} = A\mathbf{x}$ depends on the eigenvalues λ_1 and λ_2; the geometry of the trajectories depends on the eigenvectors. Fig. 2.1 shows the possible solutions in the τ, δ plane.

The most common natures are **saddles**, stable and unstable **nodes**, stable and unstable **spirals** (clockwise and anticlocwise), and **centres** (clockwise and anticlockwise).

Less frequent natures are **lines of fixed points**, stable and unstable **stars**, and stable and unstable **degenerate nodes**.

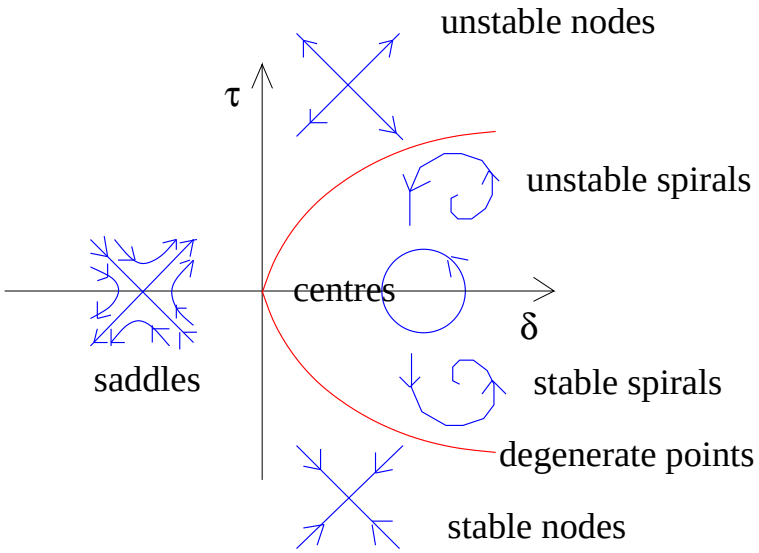


Figure 2.1: Nature of fixed points of $\dot{\mathbf{x}} = A\mathbf{x}$ in the τ, δ plane.

Consider the characteristic equatioon $\lambda = (1/2)[\tau \pm \sqrt{\tau^2 - 4\delta}]$. If $\delta < 0$ the argument of the square root is positive, the eigenvalues are real and have opposite signs, and the origin is a **saddle**. If $\delta > 0$ then either $\tau^2 - 4\delta > 0$, in which case the eigenvalues are real with the same sign (**nodes**), or $\tau^2 - 4\delta < 0$, in which case the eigenvalues are complex conjugate pairs (**spirals**). The parabola $\tau^2 - 4\delta = 0$ is the boundary between nodes and spirals: **degenerate nodes and spirals** live on the parabola. The stability of nodes and spirals depends on τ: if $\tau < 0$ both eigenvalues have negative real parts so the fixed points are stable; if $\tau > 0$ both eigenvalues have positive real parts and they are unstable. Neutrally stable **centre** exist on the line $\tau = 0$, where the eigenvalues are purely imaginary. If $\delta = 0$ then at least one of the eigenvalues is zero. Then the origin is not an isolated fixed point: there is either a whole line of fixed points or a plane of fixed points (if $\tau = 0$ too).

Saddles, nodes and spirals are the major types of fixed points, because they exist in large regions of the τ, δ plane. Centres, stars and degenerate nodes and non-isolated fixed points are borderline cases. Physically, the centres are the most important borderline cases, as they exist in systems in which the energy is conserved (eg mechanical systems without friction).

2.8 Saddle

If $\delta < 0$, then λ_1 and λ_2 are real and have opposite signs, say $\lambda_1 < 0$ and $\lambda_2 > 0$ and write $\lambda_1 = -|\lambda_1|$ and $\lambda_2 = |\lambda_2|$. The general solution is

$$\mathbf{x}(t) = c_1 \mathbf{U} e^{-|\lambda_1| t} + c_2 \mathbf{V} e^{|\lambda_2| t}, \tag{2.19}$$

For $t \to \infty$, $\mathbf{x}$ becomes very large and parallel (or anti-parallel) to $\mathbf{V}$ (depending on the sign of c_2):

$$\mathbf{x}(t) \to c_2 \mathbf{V} e^{|\lambda_2| t} \to \infty \qquad \text{for} \qquad t \to \infty, \tag{2.20}$$

For $t \to -\infty$, $\mathbf{x}$ becomes very large and parallel (or anti-parallel) to $\mathbf{U}$ (depending on the sign of c_1):

$$\mathbf{x}(t) \to c_1 \mathbf{U} e^{-|\lambda_1| t} \to \infty \qquad \text{for} \qquad t \to -\infty, \tag{2.21}$$

Trajectories start aligned along $\mathbf{U}$ for $t \to -\infty$, and finish aligned along $\mathbf{V}$ for $t \to \infty$. Each direction along $\mathbf{U}$ and $\mathbf{V}$ defines a **separatrix** (a boundary between two distinct types of behaviour). Trajectories which start exactly on a separatrix remain on it, because they correspond to either $c_1 = 0$ or $c_2 = 0$, so the phase point moves along an eigenvector. The fixed point at the origin is called a **saddle**.

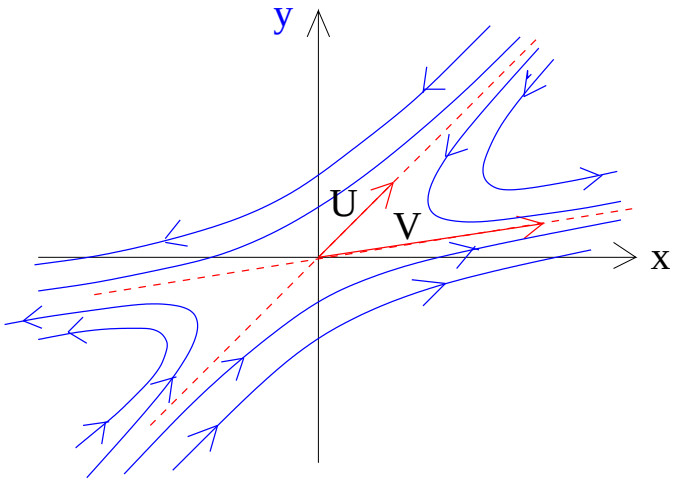


Figure 2.2: Saddle.

2.9 Node

Let $\delta > 0$ and $\tau^2 - 4\delta > 0$, Then λ_1 and λ_2 are both real and have the same sign, positive if $\tau > 0$ or negative if $\tau < 0$ (because $|\tau| > |\sqrt{\tau^2 - 4\delta}|$). The general solution is

$$\mathbf{x}(t) = c_1 \mathbf{U} e^{\lambda_1 t} + c_2 \mathbf{V} e^{\lambda_2 t}, \tag{2.22}$$

Suppose that $0 > \lambda_1 > \lambda_2$. Then

$$\mathbf{x}(t) \approx c_1 \mathbf{U} e^{\lambda_1 t} \to \mathbf{0} \qquad \text{for} \qquad t \to \infty, \tag{2.23}$$

and

$$\mathbf{x}(t) \approx c_2 \mathbf{V} e^{\lambda_2 t} \to \infty \qquad \text{for} \qquad t \to -\infty, \tag{2.24}$$

Trajectories come from infinity along $\mathbf{V}$, then move toward the origin becoming aligned along $\mathbf{U}$. The fixed point at the origin is called a **stable node**.

Suppose that $\lambda_1 > \lambda_2 > 0$. Then the behaviour is reversed and we have an **unstable node**.

Definition: We call **slow eigendirection** the direction spanned by the eigenvector with the smallest $|\lambda|$, and **fast eigendirection** the direction spanned by the eigenvector with the biggest $|\lambda|$. Typically trajectories approach the origin for $t \to \infty$ tangent to the slow eigendirection, whereas for $t \to -\infty$ the trajectories are parallel to the fast eigendirection.

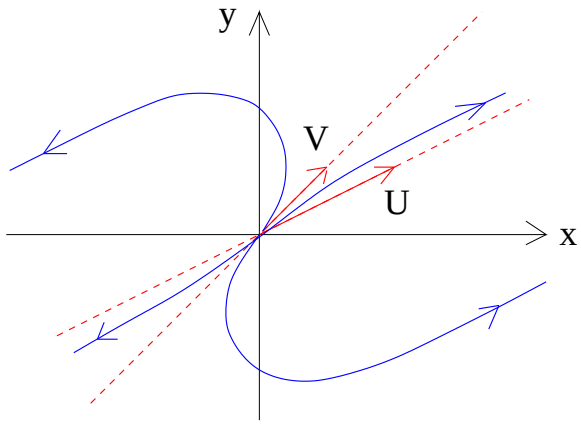


Figure 2.3: Unstable node.

2.10 Spiral

Let $\delta > 0$, $\tau^2 - 4\delta < 0$ with $\tau \neq 0$. Then the eigenvalues λ_1 and λ_2 are complex and distinct:

$$\lambda = \frac{1}{2}[\tau \pm \sqrt{4\delta - \tau^2}] = \alpha \pm i\theta, \tag{2.25}$$

where

$$\alpha = \frac{\tau}{2}, \qquad \theta = \frac{1}{2}\sqrt{4\delta - \tau^2}. \tag{2.26}$$

Theorem: Eigenvalues form a complex conjugate pair: $\overline{\lambda_1} = \lambda_2$.

Theorem: Let $\mathbf{U} = \mathbf{R} + i\mathbf{S}$ be the eigenvector of λ_1 where $\mathbf{R}$ and $\mathbf{S}$ are real vectors. Then the eigenvector of λ_2 is $\mathbf{V} = \mathbf{R} - i\mathbf{S}$, that is to say, the eigenvectors of A are complex conjugate of each other: $\overline{\mathbf{U}} = \mathbf{V}$.

Theorem: The general solution $\mathbf{x} = c_1 e^{\lambda_1 t}\mathbf{U} + c_2 e^{\lambda_2 t}\mathbf{V}$ can be written as

$$\mathbf{x}(t) = 2fe^{\tau t/2}[\mathbf{R}\cos(\eta + \theta t) - \mathbf{S}\sin(\eta + \theta t)]. \tag{2.27}$$

where f and η are arbitrary constants.

Trajectories are **stable spirals** which go toward the origin for $\tau < 0$ and **unstable spirals** which move away from the origin for $\tau > 0$.

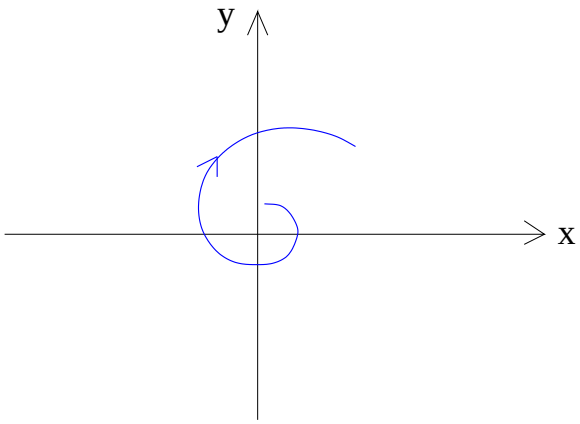


Figure 2.4: Unstable spiral.

2.11 Centre

Let $\delta > 0$, $\tau = 0$: in this case λ becomes

$$\lambda = \frac{1}{2}(\tau - i\sqrt{4\delta - \tau^2}) = \pm i\sqrt{\delta}, \tag{2.28}$$

Everything remains as in the previous section except that now $\alpha = \tau/2 = 0$, so the general solution is

$$\mathbf{x}(t) = f[\mathbf{R}\cos(\eta + \theta t) - \mathbf{S}\sin(\eta + \theta t)], \tag{2.29}$$

where $\mathbf{U} = \mathbf{R} + i\mathbf{S}$ and $\mathbf{V} = \mathbf{R} - i\mathbf{S}$ are the two complex conjugate eigenvectors, $\theta = \sqrt{\delta}$ and f and η are arbitrary constants.

The solution is periodic and the trajectories are circles around the origin, as in Fig. 2.5. The fixed point at the origin is called a **centre**. Since trajectories neither approach the centre nor move away from it, the centre is **neutrally stable**.

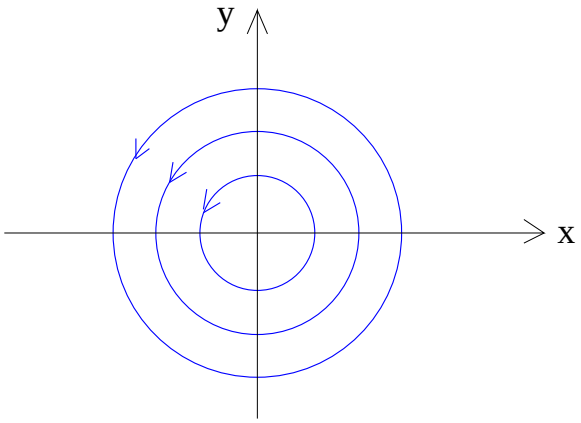


Figure 2.5: Centre.

2.12 Line of fixed points

The remaining natures correspond to **degenerate points**. If $\delta = 0$ and $\tau^2 - 4\delta = \tau^2 > 0$, which means that $\lambda_1 \neq 0$ but $\lambda_2 = 0$, then

$$\lambda = \frac{1}{2}(\tau \pm \sqrt{\tau^2}) = \frac{1}{2}(\tau \pm \tau), \tag{2.30}$$

thus $\lambda_1 = \tau$ and $\lambda_2 = 0$. The general solution is

$$\mathbf{x}(t) = c_1 \mathbf{U} e^{\tau t} + c_2 \mathbf{V}, \tag{2.31}$$

where the eigenvector $\mathbf{V}$ belongs to the eigenvalue $\lambda_2 = 0$. Any point on the $\mathbf{V}$ separatrix is an equilibrium point. See Fig. 2.6

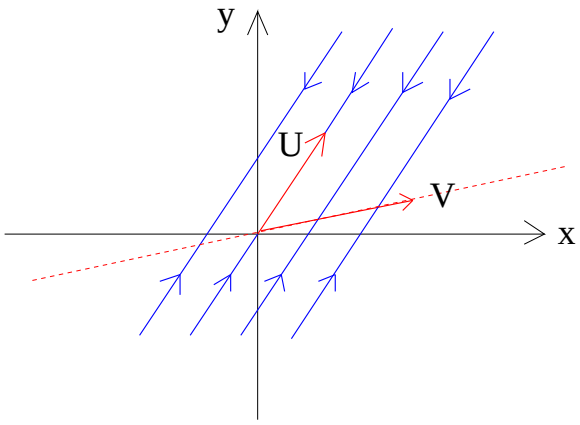


Figure 2.6: Line of fixed points.

2.13 Star and degenerate node

In this case $\delta > 0$, $\tau^2 - 4\delta = 0$ and the two roots are

$$\lambda_1 = \lambda_2 = \frac{1}{2}\tau, \tag{2.32}$$

There are two possibilities:

1. The first possibility is that the matrix A is diagonal:

$$A = \begin{vmatrix} \lambda & 0 \\ 0 & \lambda \end{vmatrix}; \tag{2.33}$$

Then **any** vector $\mathbf{C} = (C_1, C_2)$ is an eigenvector, and the solution is

$$\mathbf{x}(t) = \mathbf{C}e^{\lambda t}, \tag{2.34}$$

All trajectories are straight lines which pass through the origin. The fixed point at the origin is called a **star**; it is stable if $\lambda < 0$ and unstable if $\lambda > 0$.

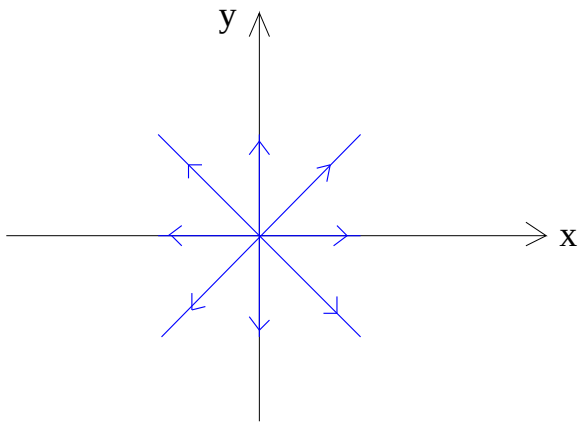


Figure 2.7: Unstable star.

2. The second possibility is that the matrix has the form

$$A = \begin{vmatrix} a & b \\ 0 & a \end{vmatrix}; \tag{2.35}$$

The matrix has only one eigenvector, $\mathbf{U} = (1, 0)$. The phase portrait is a node again, but there is only one distinguished direction, so it is called a **degenerate node** – see Fig. 2.8. The node is stable for $\lambda = \tau < 0$ and unstable for $\lambda = \tau > 0$. It can be verified directly that the general solution is

$$\mathbf{x}(t) = \begin{vmatrix} 1 \\ 0 \end{vmatrix} (c_1 + c_2 t)e^{\lambda t} + \begin{vmatrix} 0 \\ c_2/b \end{vmatrix} e^{\lambda t}. \tag{2.36}$$

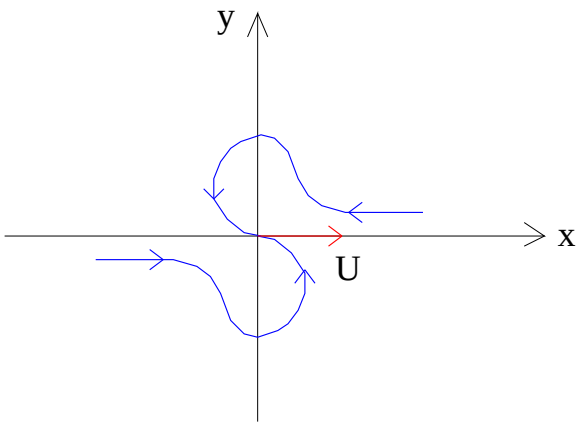


Figure 2.8: Degenerate node.

Chapter 3

Non-linear systems

3.1 Local behaviour

Consider the 2–dim autonomous system $\dot{\mathbf{x}} = \mathbf{f}$ where now $\mathbf{f}$ is a **nonlinear** function of $\mathbf{x}$. Suppose that $\mathbf{x}^*$ is a fixed point: $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$.

By expanding $\mathbf{f}(\mathbf{x})$ in Taylor series near $\mathbf{x} = \mathbf{x}^*$ and neglecting terms which are quadratic or of higher powers in $(\mathbf{x} - \mathbf{x}^*)$, we have

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^*) + D\mathbf{f}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + \cdots = D\mathbf{f}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + \cdots \tag{3.1}$$

where $D\mathbf{f}(\mathbf{x}^*)$ is the Jacobian matrix evaluated at $\mathbf{x} = \mathbf{x}^*$:

$$D\mathbf{f}(\mathbf{x}^*) = \begin{vmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{vmatrix} \tag{3.2}$$

The linear system

$$\dot{\mathbf{x}} = D\mathbf{f}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*), \tag{3.3}$$

is called the **linearised system** of $\dot{\mathbf{x}} = \mathbf{f}$ near the fixed point $\mathbf{x}^*$.

Let

$$\mathbf{X} = \mathbf{x} - \mathbf{x}^*, \qquad A = D\mathbf{f}(\mathbf{x}^*), \tag{3.4}$$

Since $\dot{\mathbf{X}} = \dot{\mathbf{x}}$ we have

$$\dot{\mathbf{X}} = A\mathbf{X}. \tag{3.5}$$

3.2 Hartman-Grobman theorem

Near a fixed point $\mathbf{x}^*$ the nonlinear system $\dot{\mathbf{x}} = \mathbf{f}$ can be approximated by the linearised system $\dot{\mathbf{X}} = A\mathbf{X}$, where $\mathbf{X} = (X, Y) = (x - x^*, y - y^*)$ and A is the Jacobian matrix evaluated ar $\mathbf{x}^*$:

$$\begin{vmatrix} \dot{X} \\ \dot{Y} \end{vmatrix} = \begin{vmatrix} \partial f_1/\partial x & \partial f_1/\partial y \\ \partial f_2/\partial x & \partial f_2/\partial y \end{vmatrix} \begin{vmatrix} X \\ Y \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} X \\ Y \end{vmatrix} \tag{3.6}$$

The eigenvalues of the Jacobian matrix determine the nature of the linearised system around $(X, Y) = (0, 0)$. The natural question to ask is whether these eigenvalues also determine the dynamics of the nonlinear system around (x^*, y^*). For that to happen, it is necessary that the quadratic and higher order terms in the Taylor expansion can be safely neglected. This is the case if the fixed point is a saddle, node or spiral. The other cases (centres, degenerate nodes and stars) are more delicate: their nature can be altered by the nonlinearity.

Definition: A fixed point $\mathbf{x}^*$ is called **hyperbolic** if all the eigenvalues of $D\mathbf{f}(\mathbf{x}^*)$ have a nonzero real part (thus a centre is not a hyperbolic fixed point).

Hartman-Grobman Theorem: The behaviour of the phase portrait of the nonlinear system $\dot{\mathbf{x}} = \mathbf{f}$ near its hyperbolic fixed point $\mathbf{x}^*$ is qualitatively the same as for the linearised system $\dot{\mathbf{x}} = D\mathbf{f}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)$.

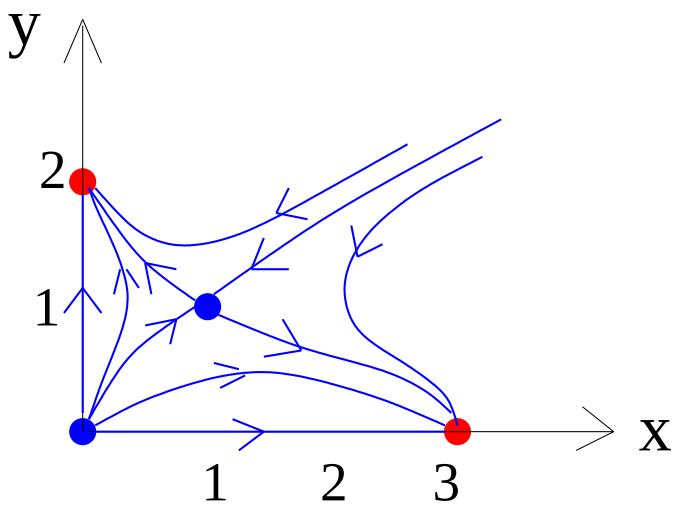


Figure 3.1: Application of the Hartman-Grobman Theorem to find the phase portrait of $\dot{x} = x(3 - x - y)$, $\dot{y} = y(2 - x - y)$ in $x \geq 0$, $y \geq 0$, where x represents a population of rabbits and y a population of sheep. There are four fixed points: two stable nodes, one unstable nodes and one saddle.

The Hartman-Grobman Theorem can be used to put together the phase portrait of a nonlinear dynamical system $\dot{\mathbf{x}} = \mathbf{f}$:

- Firstly, we find all the fixed points $\mathbf{x}^*$ and the Jacobian $D\mathbf{f}(\mathbf{x})$.
- Then, at each $\mathbf{x}^*$, we find eigenvalues and eigenvectors of $\dot{\mathbf{X}} = A\mathbf{X}$ where $\mathbf{X} = \mathbf{x} - \mathbf{x}^*$ and $A = D\mathbf{f}(\mathbf{x}^*)$, determine the nature of that $\mathbf{x}^*$ and draw trajectories near it
- Finally, we join trajectories near all $\mathbf{x}^*$ together, making sure that they do not intesect.

3.3 Basin of attraction

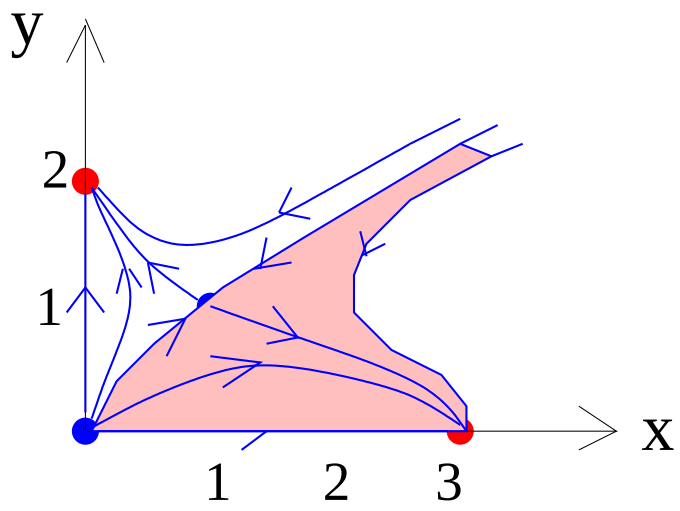


Figure 3.2: Basin of attraction of the fixed point $(3, 0)$.

Fig. 3.1 shows that trajectories which start near the origin go to the stable node on the x axis; other trajectories go to the stable node on the y axis. There is also a trajectory which is undecided between the two nodes and goes toward the saddle point. This trajectory is part of the **stable manifold** of the saddle; the other branch of the stable manifold is the trajectory which arrives at the saddle point from infinity.

Trajectories starting below the stable manifold end with the extinction of sheep ($y = 0$); trajectories starting above it end with the extinction of rabbits ($x = 0$).

Definition: Given an attracting fixed point $\mathbf{x}^*$, its **basin of attraction** is the set of initial conditions $\mathbf{x}(0)$ such that $\mathbf{x}(t) \to \mathbf{x}^*$ for $t \to \infty$.

Fig. 3.2 shows the basin of attraction of the fixed point $(3, 0)$.

The stable manifold separates the basins of attractions of the two nodes, so it is called a **basin boundary**. The trajectory along the stable manifold is a **separatrix**.

3.4 Conservative systems

An important property of conservative systems is the following

Theorem: A conservative system cannot have any attractive fixed point.

Definition: Trajectories that start and end at the same fixed point are called **homoclinic orbits**.

Homoclinic orbits are common in conservative systems. They are not periodic solutions - it takes an infinite time for a phase point to move along a homoclinic orbit and reach a fixed point.

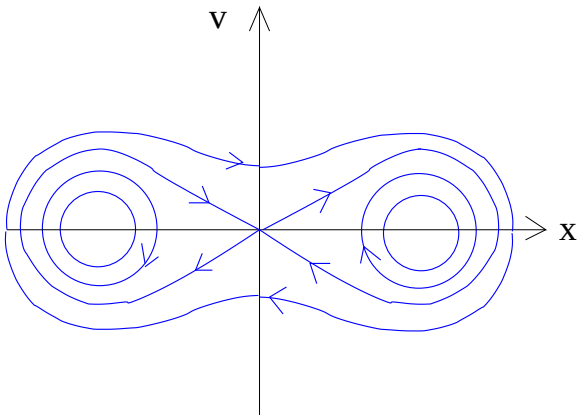


Figure 3.3: Phase portrait of $\dot{x} = v$, $\dot{v} = x - x^3$.

Fig. 3.3 shows the phase portrait of the conservative system

$$\ddot{x} = -\frac{d\phi}{dx} = x - x^3, \tag{3.7}$$

which we write as

$$\dot{x} = v, \tag{3.8}$$
$$\dot{v} = x - x^3. \tag{3.9}$$

There are two centres $(\pm 1, 0)$ and a saddle $(0, 0)$. Each centre is surrounded by closed orbits, and larger close orbits encircle all three fixed points. All solutions are **periodic** but the homoclinic orbits (the trajectories which start and end at the origin for $t \to \pm\infty$).

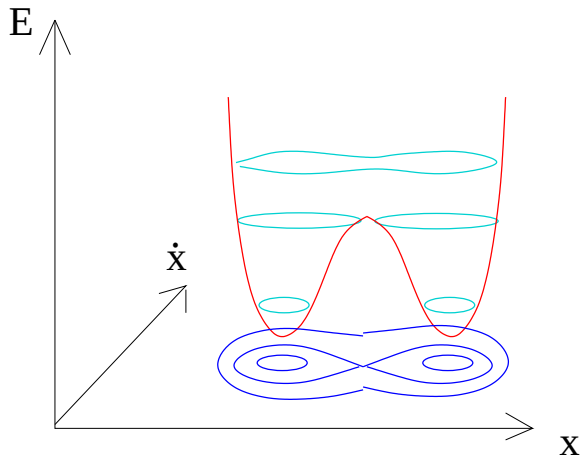


Figure 3.4: The energy of the dynamical system $\dot{x} = v$, $\dot{v} = x - x^3$.

The quantity shown in Fig. 3.4 is the energy

$$E(x, v) = \frac{1}{2}m\dot{x}^2 + \phi(x) = \frac{1}{2}v^2 - \frac{1}{2}x^2 + \frac{1}{4}x^4. \tag{3.10}$$

Trajectories are horizontal contours parallel to the $(x, \dot{x})$ plane along which $E(x, v)$ is constant. It is apparent that E has local minima corresponding to the two centres. Contours which are just above the local minima correspond to small orbits surrounding the centres. The saddle point and the homoclinic orbits are higher, and the large orbits which encircle all fixed points are even higher.

3.5 Reversible systems

Sometimes the dynamics of a system looks the same whether time runs forward or backward. If you watch a movie of a pendulum which swings back and forth, you cannot tell if the movie is projected from the beginning to the end or from the end to the beginning. But if you watch the movie of a glass window shattered by a brick which is thrown at it, you can tell if the movie is projected backward.

Definition: A second-order system that is invariant under $t \to -t$ and $v \to -v$ is called a **reversible system**..

Theorem: Any mechanical system which obeys Newton's law $m\ddot{x} = F(x)$ is symmetric under time reversal.

Geometrically, this means that every trajectory has a twin trajectory, which differs by the time-reversal and the reversal of the velocity (a reflection across the x axis in the x, v plane) as in Fig. 3.5.

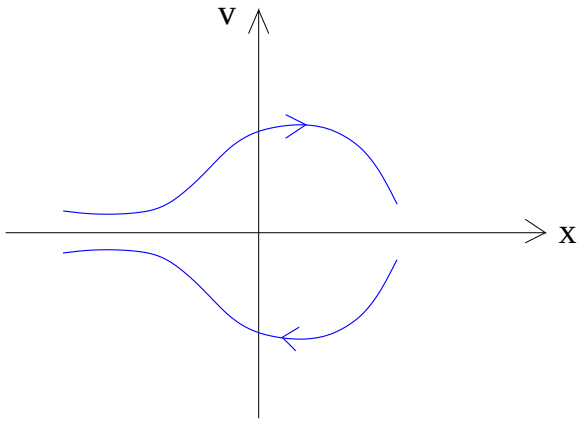


Figure 3.5: A trajectory and the corresponding trajectory with v replaced by $-v$ and t by $-t$.

Reversible systems are different from conservative systems, but they have similar properties, as shown by the two following theorems:

Theorem: Suppose that we have a conservative system $\dot{\mathbf{x}} = \mathbf{f}$, that is,, there exists a conserved quantity $E(\mathbf{x})$. Suppose that $\mathbf{x}^*$ is an isolated fixed point (isolated means that there are no other fixed points in the neighbourhood of $\mathbf{x}^*$). One can prove that, if $\mathbf{x}^*$ is a local minimum of $E(\mathbf{x})$, then all trajectories sufficiently close to $\mathbf{x}^*$ are closed.

Theorem: Suppose that $\mathbf{x}^* = 0$ is a linear centre for the 2-dim system $\dot{x} = f(x, y)$, $\dot{y} = g(x, y)$. Suppose that the system is reversible. Then, sufficiently close to the origin, all trajectories are closed curves.

Chapter 4

Limit cycles

4.1 Limit cycles

A very important feature of non-linear systems (which is absent in linear systems) is the possibility of **limit cycles**.

Definition: A **cycle**, or **periodic orbit**, of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, is any closed trajectory which is not a fixed point.

Definition: An **isolated** periodic orbit is called a **limit cycle**. Isolated means that that neighbouring trajectories are not closed: they either spiral toward the limit cycle or away from it - see Fig. 4.1.

Definition: If all neighbouring trajectories approach the limit cycle, then the limit cycle is said **stable**, or **attracting**. Otherwise it called **unstable**, or **repelling**.

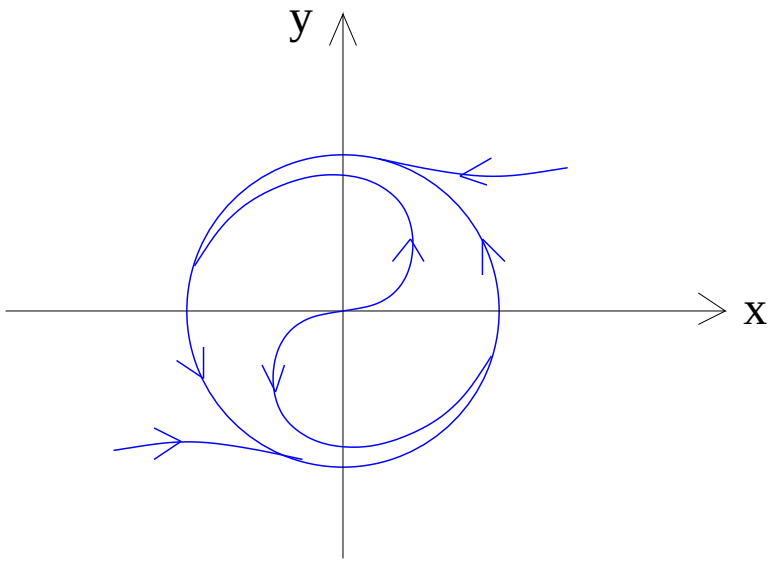


Figure 4.1: Limit cycle.

4.2 Systems without limit cycles

There are ways to exclude the possibility of closed orbits.

Definition: If the system $\dot{\mathbf{x}} = \mathbf{f}$ can be written as

$$\dot{\mathbf{x}} = (\dot{x}, \dot{y}) = -\nabla\phi = -(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}), \tag{4.1}$$

then the system is called a **gradient system** and the function $\phi(\mathbf{x})$ is called a **potential function**.

Theorem: Closed orbits are not possible in gradient systems.

The problem with this result is that most 2-dim systems are not gradient systems. But all 1-dim systems (vector fields on the line) are gradient system, so we have proved that

Theorem: There are no oscillations in a 1–dim dynamical system.

4.3 Poincare-Bendixson Theorem

Poincare-Bendixson Theorem: Consider the dynamical system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in an open set containing a closed, bounded subset R of the plane. Assume that R does not contain any fixed points, and that there exists a trajectory C which is confined within R, that is to say it starts in R and stays within R at all future times. Then either C is a closed orbit or it spirals towards a closed orbit as $t \to \infty$. In either case we conclude that R contains a closed orbit - see Fig. 4.2. Note that R is a ring-shaped because the close orbit must enclose a fixed point P but P is not allowed in R.

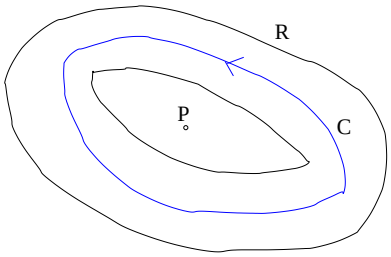


Figure 4.2: Poincare–Bendixson Theorem.

The difficulty with applying the theorem is that it is difficult to show that there exists a confined trajectory C. The trick consists in constructing a **trapping region**, a closed connected set R such that all along the boundaries of R the vector field points into R. Then all trajectories in R must be confined, and, if we can prove that there are no fixed points in R, the theorem applies and R must contain a closed orbit.

Chapter 5

Chaos

5.1 Motion in 3-dim phase space

In 2–dim phase space the Poincare-Bendixson Theorem constrains the motion: if a trajectory is confined to a closed, bounded region that contains no fixed points, it must eventually approach a closed orbit. Nothing more complicated is possible.

In 3-dim (and higher) dimensions the theorem does not apply, and something radically new can happen: trajectories may wander around forever in a bounded region without settling down to a fixed point or a closed orbit.

In some cases, 3–dim trajectories are attracted to complex geometrical object called **strange attractors**: fractal sets on which the motion is aperiodic and very sensitive to the exact value of the initial conditions.

This **sensitivity on the initial conditions**, which makes long-term predictions impossible, defines **chaos**. Two phase points which starts very close together can diverge rapidly from each other and have very different futures. Thus trajectories which are initially close can end up far from each other, anywhere on the attractor.

5.2 Liapunov exponent

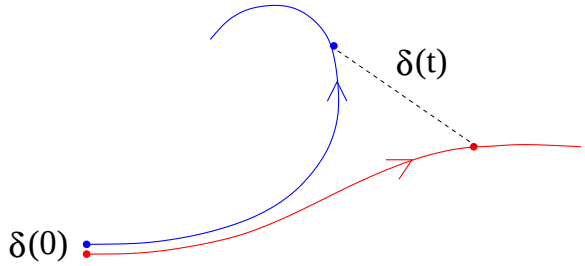


Figure 5.1: Exponential divergence of trajectories.

Let $\mathbf{x}(t)$ be a point on the attractor at time t, and $\mathbf{x}(t) + \mathbf{d}(t)$ be another point such that initially $\delta(0) = |\mathbf{d}(0)| \ll 1$.

Numerical experiments show that, as time proceeds, the separation $\delta(t)$ between the two trajectories increases as

$$\delta(t) \sim \delta(0)e^{\lambda t}, \tag{5.1}$$

where $\lambda > 0$ is called the **Liapunov exponent**. A **time-horizon** exists beyond which it is impossible to make long-term predictions because any initial uncertainty in the initial condition is amplified exponentially.

5.3 Time horizon

Suppose that the initial condition is measured with some initial error $\delta(0)$ (no measurement can be perfect). After time t, the discrepancy between the prediction and the actual evolution of the initial state is $\delta(0)e^{\lambda t}$. Let ϵ be the measure of our **tolerance**, that is, if our prediction differs from the true solution by less than ϵ then our prediction is considered acceptable. Then our prediction is unacceptable if

$$\delta(0)e^{\lambda t} > \epsilon, \tag{5.2}$$

which occurs if

$$t > t_{horizon} = \frac{1}{\lambda} \ln (\frac{\epsilon}{\delta(0)}), \tag{5.3}$$

.
Because the logarithmic dependence on $\delta(0)$, no matter how well we measure the initial condition and reduce $\delta(0)$, we cannot predict the future more than few times $1/\lambda$. We conclude that if the system is nonlinear and has a positive Liapunov exponent, there is an **intrinsic limitation** in our ability to predict the future.

5.4 Definition of chaos

At this point we are ready for the following:

Definition: Chaos is the aperiodic long–term behaviour in a deterministic system that exhibits sensitive dependence on the initial condition.

Aperiodic long-term behaviour means that trajectories do not settle down to fixed points, periodic orbits or quasi-periodic orbits as $t \to \infty$. **Deterministic** means that the irregular behaviour of the system arises from its being nonlinear, not from added external noise.

5.5 The Lorenz equations

The Lorenz equations played a key role in the discovery of chaos.

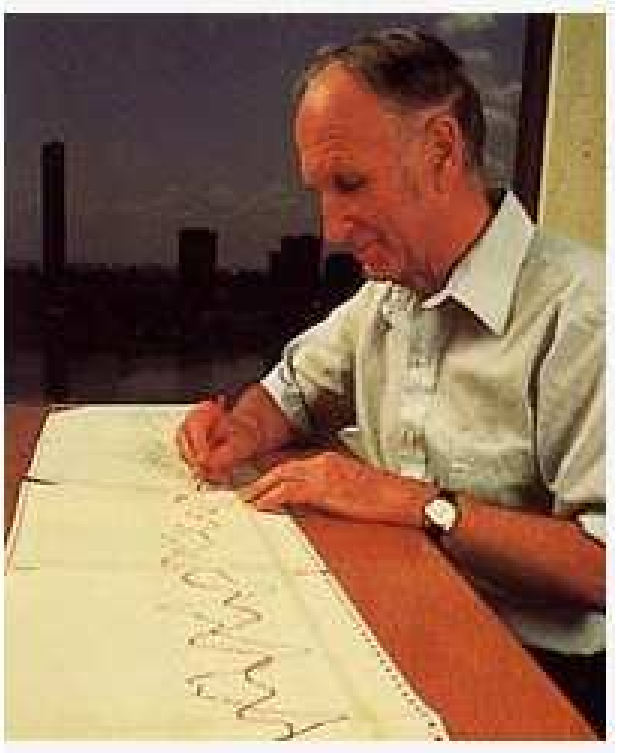


Figure 5.2: Edward Lorenz

In the early 1960's, Edward Lorenz (see Fig. 5.2), an American meteorologist at MIT, was working on weather prediction. The observed weather is often different from the predicted one. People blamed the unpredictability to extra effects which were not included in the model, and attempted to add stochastic terms (random forces) to the equations, with no success. At that time computers were a novelty. Lorenz pioneered their use in predicting the weather. His mathematical model of the weather consisted of differential equations to represent temperature, pressure, wind velocity, etc. The major ingredient of Lorenz's model was **convection**.

Air is essentially transparent to the solar radiation, so it is heated from the ground below. But a layer of fluid which is heated from below and cooled from above can be unstable. If the temperature gradient ΔT between the top and the bottom of the layer is large enough, the breaking effect of viscosity can be overcome: hot air rises and cool air sinks in the form of **convection rolls** – see Fig. 5.3.

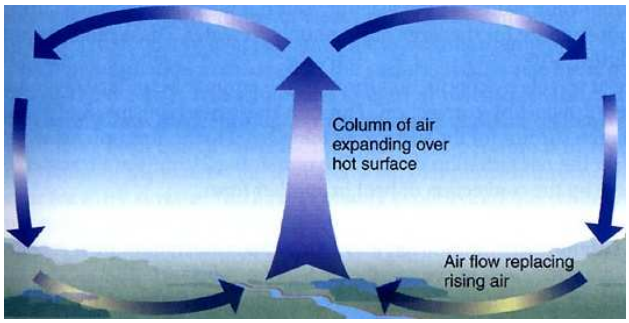


Figure 5.3: Convection cell

To cope with the limitation of the computers of the 1960's, Lorenz stripped down the equations governing convection and reduced them to the following **Lorenz equations** $\dot{\mathbf{x}} = (\dot{x}, \dot{y}, \dot{z}) = \mathbf{f} = (f_1, f_2, f_3)$ given by:

$$\dot{x} = f_1 = \sigma(y - x), \tag{5.4}$$
$$\dot{y} = f_2 = -xz + rx - y, \tag{5.5}$$
$$\dot{z} = f_3 = xy - bz, \tag{5.6}$$

where $r > 0$ corresponds to ΔT, $\sigma > 0$ is the ratio between energy losses due to viscosity and thermal conduction, $b > 0$ is related to the relative height of the fluid layer to the horizontal extent of the convective rolls within it, and the variables $x(t)$, $y(t)$ and $z(t)$ model respectively the convective overturning, the horizontal temperature variation and the vertical temperature variation.

Theorem: The Lorenz system $\dot{\mathbf{x}} = \mathbf{f}$ (Eq. 5.6) is **dissipative**, that is volume in phase space contracts under the flow:

$$\frac{dV}{dt} = \int_V \nabla \cdot \mathbf{f} dV < 0. \tag{5.7}$$

Theorem: The Lorenz system has no quasi-periodic solutions.

Theorem: The Lorenz system has no repelling fixed points or repelling closed orbits (by repelling we mean that all trajectories starting near a fixed point or a closed orbit are driven away from it).

Theorem: The origin is a fixed point of the Lorenz system: $(x^*, y^*, z^*) = (0, 0, 0)$. If $r > 1$ there are other two fixed points: $x^* = y^* = \pm\sqrt{b(r-1)}$, $z^* = r - 1$, called C^+ and C^-.

Theorem: Near the origin, the solution of the Lorenz system is such that $z(t) \to 0$ for $t \to \infty$. If $r < 1$, the motion along x and y points toward the origin too; it can be shown that all trajectories approach the origin for $t \to \infty$, thus the origin is **globally stable**. If $r > 1$, motion along x and y is a saddle; motion around C^+ and C^- is stable for

$$1 < r < r_c = \frac{\sigma(\sigma + b + 3)}{\sigma - b - 1}, \tag{5.8}$$

assuming also that $\sigma - b - 1 > 0$.

5.6 Chaos in the Lorenz system

What happens for $r > r_c$? We solve the Lorenz equations numerically and find that trajectories keep being repelled by C^+ and C^- like in a pinball machine. At the same time, trajectories are confined in a set whose volume shrinks to zero for $t \to \infty$.

Fig. 5.4 shows that the trajectory is an irregular oscillation which never repeats exactly. The motion is **aperiodic**.

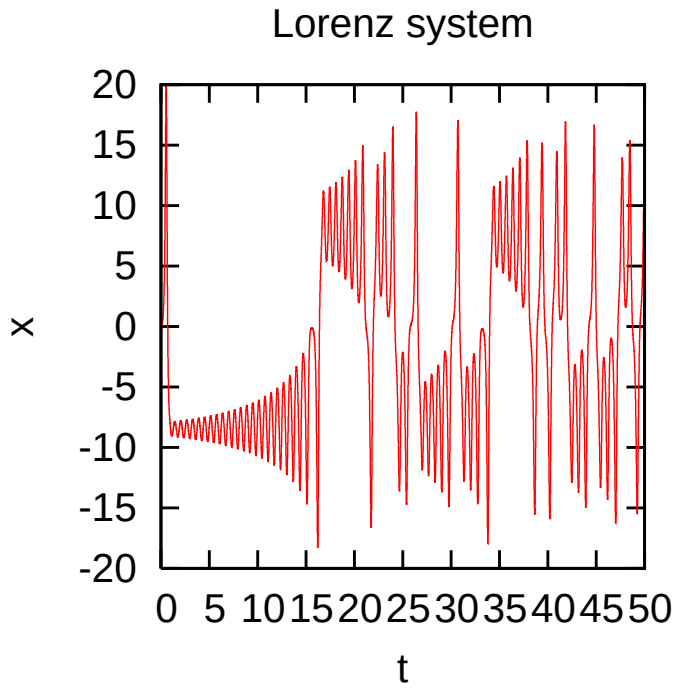


Figure 5.4: The solution $x = x(t)$ of the Lorenz equation for $\sigma = 10$, $r = 28$, $b = 8/3$ and initial condition $x(0 = y(0) = z(0) = 0.1$.

Fig. 5.5 plots the trajectory in the x, z plane. We see a **strange attractor**. The trajectory starts near the origin, then swings to C^+, then swings to the left spiralling around C^- many times, then shoots suddenly to C^+ again, and so on. The number of circuits around C^+ and C^- varies unpredictably from one cycle to the next. Physically, these swings correspond to reversing the rotation of the convection rolls.

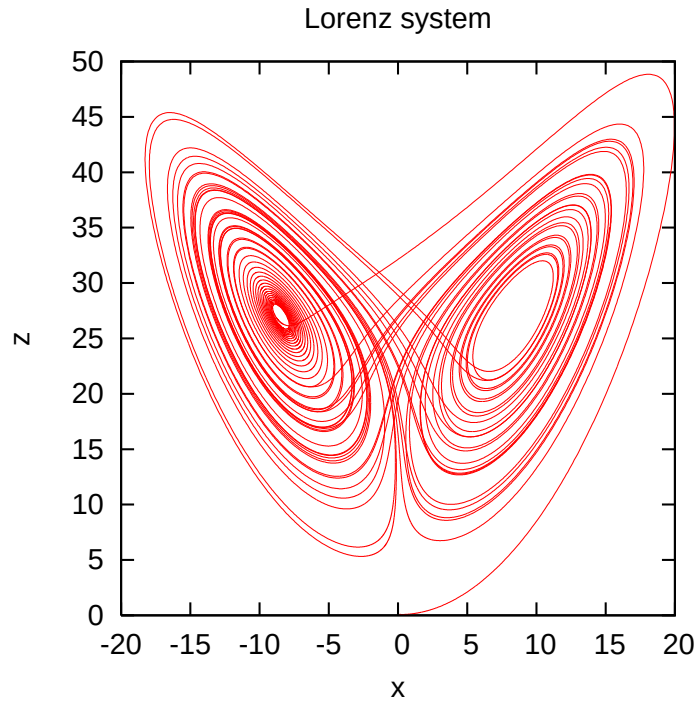


Figure 5.5: The solution of the Lorenz equation plotted in the x, z plane. The parameters are the same as in Fig. 5.4.

If we look at the trajectory in 3-dim phase space, the attractor looks like a pair of thin wings – see Fig. 5.6.

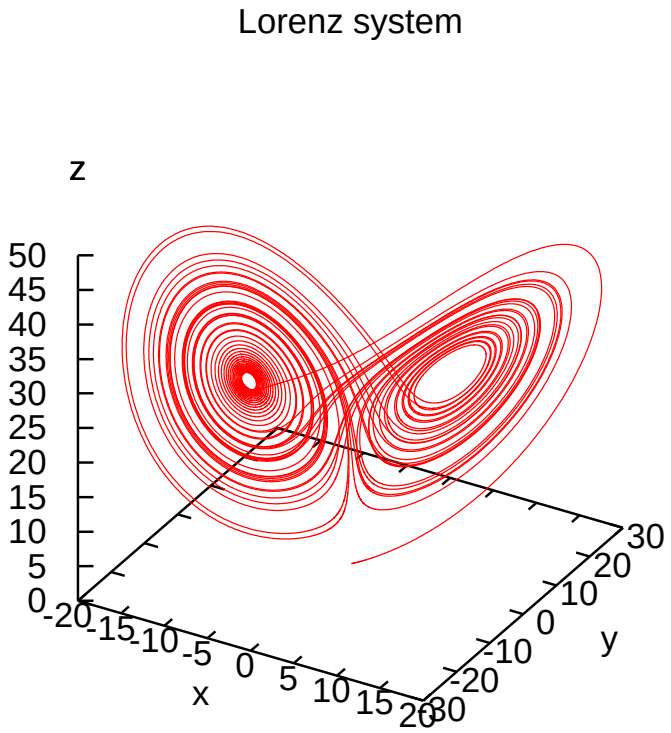


Figure 5.6: Solution (x, y, z) of the Lorenz equation plotted in three dimensions. The parameters are the same as in Fig. 5.4.

5.7 The butterfly effect

One day in the winter of 1961 Lorenz wanted to re-examine a sequence of numbers produced by his computer. Instead of restarting the entire calculation, he decided to save it and restart the run in the middle. He copied some numbers from a printout and entered them as initial conditions of a new calculation. He expected the data from the new run to match the data of the first run. Instead what he found was surprising. The data from the second run matched the data of the first run only at the beginning, then diverged dramatically - the results from the second run did not look like those of the first run at all.

After checking that his computer was not malfunctioning, Lorenz found the explanation. His printout only showed three digits, whereas the computer's memory used more digits when doing the calculation. Lorenz had entered the round-off data from the printout implicitly assuming that the small difference was negligible. By conventional wisdom (since Newton and Laplace), the second solution should have been very similar to the first solution. Surely the fourth decimal digit could not have a huge effect: in most experiments it is not even possible to measure a quantity that well (due to electronic, mechanical or thermal noise, or to inaccuracy of the equipment). Lorenz had found that a small change in the initial condition can have a huge effect. This **extreme sensitivity to initial conditions** is now recognised as the hallmark of chaos. It is not surprising that the weather is difficult to predict.

Extreme sensitivity to initial conditions is also called the **butterfly effect**, a term coined by Lorenz. What is meant is that the flapping of a butterfly's wing today in Brazil may trigger a tornado in Texas next month. Fig. 5.7 shows two trajectories which start with slightly different initial conditions, $x(0) = 0.1$ and 0.100001 respectively, and $y(0) = z(0) = 0$ for both. After a short time, the x coordinates of the two trajectories are very different from each other. The x component of the separation of the two trajectories in shown in Fig. 5.8. Note the logarithm vertical scale, hence the exponential divergence of δ (up to the size of the attractor).

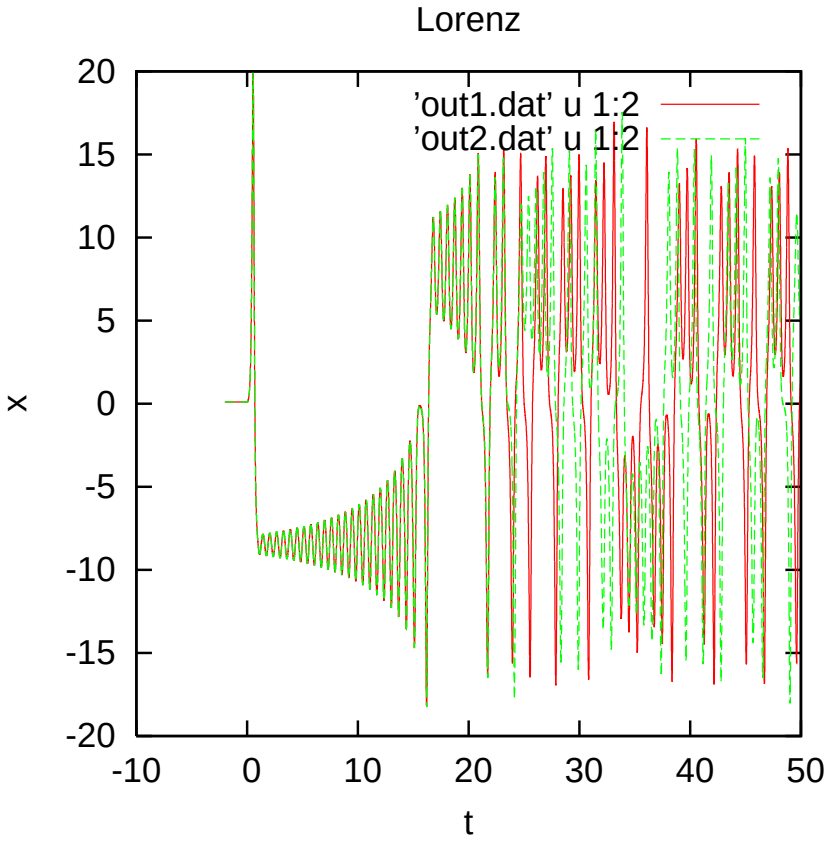


Figure 5.7: Solution of the Lorenz equations for for $\sigma = 10$, $b = 8/3$ and $r = 28$ corresponding to two slightly different initial conditions: $x(0) = 0.1$, $y(0) = 0.1$, $z(0) = 0.1$ (red line) and $x(0) = 0.10001$, $y(0) = 0.1$, $z(0) = 0.1$ (green line).

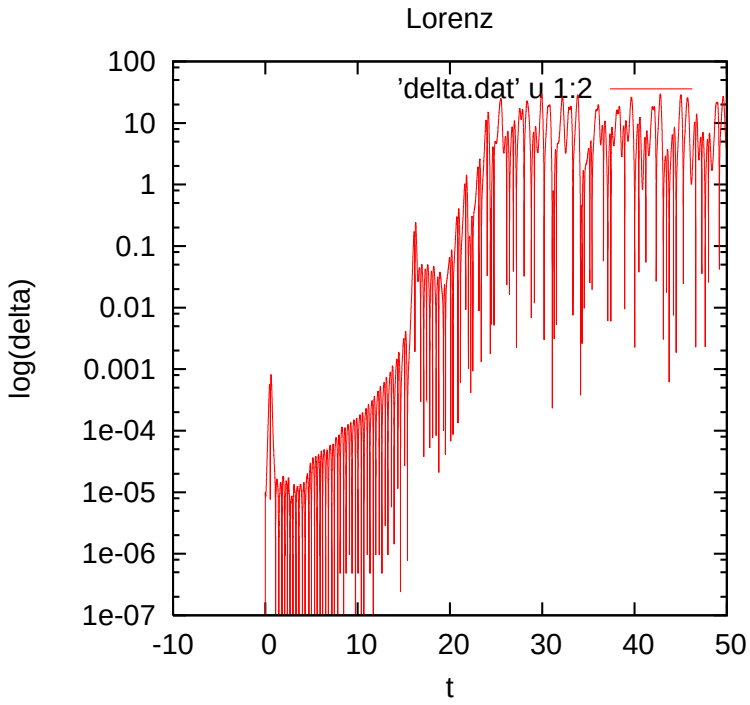


Figure 5.8: Plot of the separation Δx of the x component of two trajectories which start from slightly different initial conditions $(x, y, z) = (0.1, 0.1, 0.1)$ and $(0.100001, 0.1, 0.1)$. Parameters as in Fig. 5.7

.

5.8 Computer code

The Fortran 90 program listed below solves the Lorenz equations. The following Gnuplot programs are used to plot the solution.

```
!-----------------------------------------------------------
program lorenz1
! Aim: To find the solution of the Lorenz system
!            dx/dt=f1(x,y,z)=-sigma*(x-y)
!            dy/dt=f2(x,y,z)=r*x-y-x*z
!            dz/dt=f3(x,y,z)=-b*z+x*y
!      at given initial conditions x(0),y(0),z(0),
!      time step, and given b,sigma,r
!      Method: Euler's explicit method
implicit none
integer :: n,nn,skip
real :: t,h
real :: x,y,z,xold,yold,zold,f1old,f2old,f3old
real :: sigma,r,b

open(unit=7,file='out.dat')           ! file for output data

! Initial condition
x=0.1
y=0.1
z=0.1

! Inputs
sigma=10.0
r=28.0
b=8.0/3.0
h=0.001
nn=50000
skip=0
```

```
t=0.0
do n=1,nn                                  ! time loop
  xold=x                                   ! upgrade old value of x
  yold=y                                   ! upgrade old value of y
  zold=z                                   ! upgrade old value of z
  call get_f(sigma,b,r,xold,yold,zold,f1old,f2old,f3old)
  t=h*n                                    ! get new value of t
  x=xold+h*f1old                           ! get new value of x
  y=yold+h*f2old                           ! get new value of y
  z=zold+h*f3old                           ! get new value of z
  if(n.ge.skip) then                       ! print to file as desired
  write(unit=7,fmt="(4e12.4)")t,x,y,z      ! print time series
  endif
enddo
close(unit=7)                              ! close output file
end program lorenz1

!-------------------------------------------------------------

subroutine get_f(sigma,b,r,x,y,z,f1,f2,f3)
! Aim: function f1,f2,f3,f4
implicit none
real :: b,sigma,r,x,y,z,f1,f2,f3

f1=-sigma*(x-y)
f2=r*x-y-x*z
f3=-b*z+x*y

end subroutine

!-------------------------------------------------------------
```

```
#-------------------------------------------------------
#program gnu1
set terminal postscript color
set output "lorenz1a.ps"
set size square
set size 0.7,0.7
set nokey
set size square
set size 0.5,0.5
set title "Lorenz system"
set xlabel "t"
set ylabel "x"
plot'out.dat' u 1:2 w l
#-------------------------------------------------------
#program gnu2
set terminal postscript color
set output "lorenz1b.ps"
set size square
set size 0.7,0.7
set nokey
set title "Lorenz system"
set xlabel "x"
set ylabel "y"
set zlabel "z"
set ticslevel 0
splot'out.dat' u 2:3:4 w l
#-------------------------------------------------------
#program gnu3
set terminal postscript color
set output "lorenz1c.ps"
set size square
set size 0.7,0.7
set nokey
set title "Lorenz system"
set xlabel "x"
set ylabel "z"
plot'out.dat' u 2:4 w l
#-------------------------------------------------------
```

Chapter 6

The geometry of strange attractors

6.1 Fractals

The attractor shown in Fig. 5.6 is called a **strange attractor** because it is an attractor which exhibits sensitive dependence on the initial conditions. Strange attractors are called "strange" because they are **fractal sets**.

Fractals are complex geometrical shapes with fine structure at arbitrarily small scales. They are often self–similar, in the sense that if we zoom a part of a fractal we see the same features which appear at larger scales. i

Historically, it was Lewis Richardson (1881–1953) who first realized that the length of a coastline or of any other boundary depends on the scale of measurement, see Fig. 6.1. Starting from this observation, the French mathematician Benoit Mandelbrot created the theory of fractal geometry, coined the word **fractal** and introduced the concept of **fractal dimension**.

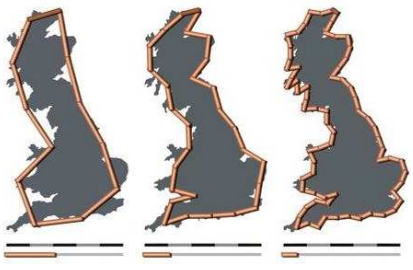


Figure 6.1: The British coastline is 2400, 2800 or 3400 Km long if it measured with unit of 200, 100 or 50 Km respectively.

Mandelbrot showed that many natural objects are fractals: coastlines, clouds, mountains, blood vessel networks, broccoli, trees, etc, as in Fig. 6.2, 6.3 and 6.4.

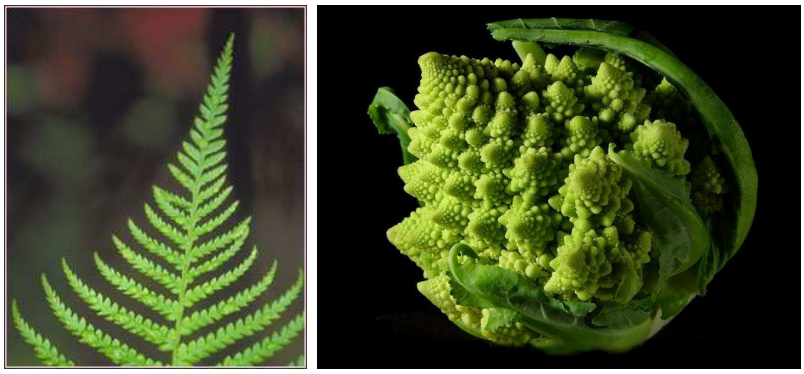


Figure 6.2: Fractals in Nature: ferns, broccoli,..

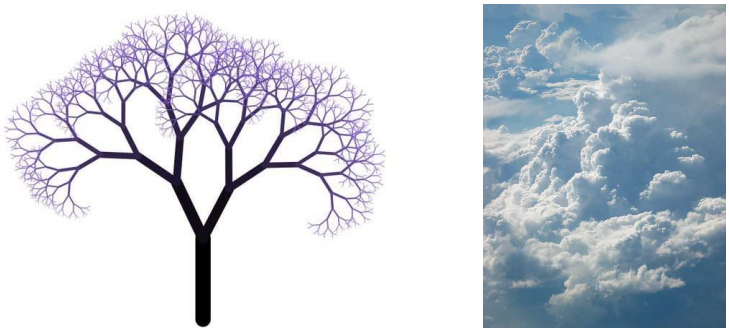


Figure 6.3: ...trees, clouds,..

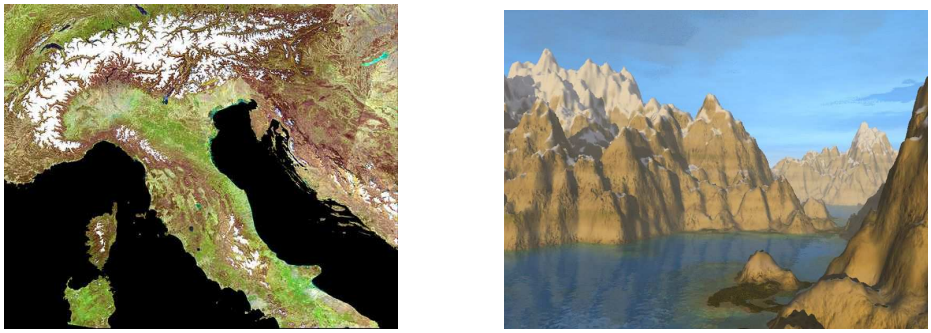


Figure 6.4: ...and mountains.

6.2 The Koch curve

The Koch curve, shown in Fig. 6.5, is an example of mathematical fractal. To build the Koch curve we start with a segment K_0; we replace the middle third of K_0 with two sides of an equilateral triangle and get K_1; we repeat the process and get K_2, then K_3, K_n, etc. The limit $n \to \infty$ is the **Koch curve** K.

One is tempted to say that the Koch curve, as any curve, has dimension 1. But note that K has infinite length. Let L be the the length of K_0. The length of K_1 is $(4/3)L$; the length of K_2 is $(4/3)^2 L$, the length of K_n is $(4/3)^n L$, etc. We conclude that, for $n \to \infty$, the length of K is infinite.

This result also means that the distance between any two points of the Koch curve is infinite, which suggests that K is more than 1–dimensional, but it would be wrong to call it 2–dimensional, as it does not have any "area". What is the dimension of the Koch curve, then ?

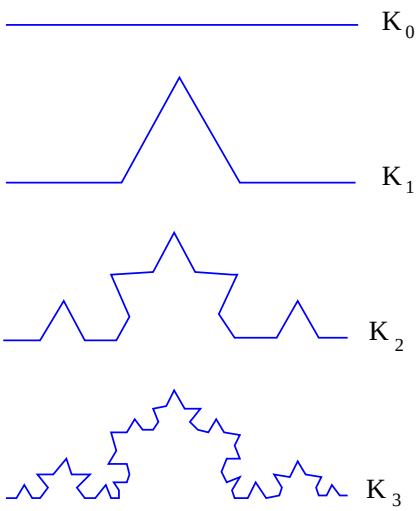


Figure 6.5: The first iterations of the Koch curve

.

6.3 Fractal dimension

Consider an ordinary geometrical object, such as a line, a square, or a cube, with linear size 1 in ordinary Euclidean space of dimension D. If we reduce the size of the object by the factor $1/\ell$ in each spatial direction, we need $N = \ell^D$ similar small objects to cover the original object. The dimension D is thus

$$D = \frac{\ln N}{\ln \ell}. \tag{6.1}$$

For example (see Fig. 6.6), take a segment of initial length $\ell = 1$ and reduce its size by the factor $1/\ell = 1/2$ with $\ell = 2$. To cover the original segment with the smaller segments we need $N = 2$ small segments, each of size $1/2$, hence $D = \ln N / \ln \ell = \ln 2 / \ln 2 = 1$, as expected (the initial segment has dimension 1).

Another example: take a cube of size $\ell = 1$ and reduce its size by the factor $1/\ell = 1/3$ with $\ell = 3$. To cover the original cube we need $N = 27$ small cubes, so $D = \ln N / \ln \ell = \ln 27 / \ln 3 = 3 \ln 3 / \ln 3 = 3$, as expected (the initial cube has dimension 2).

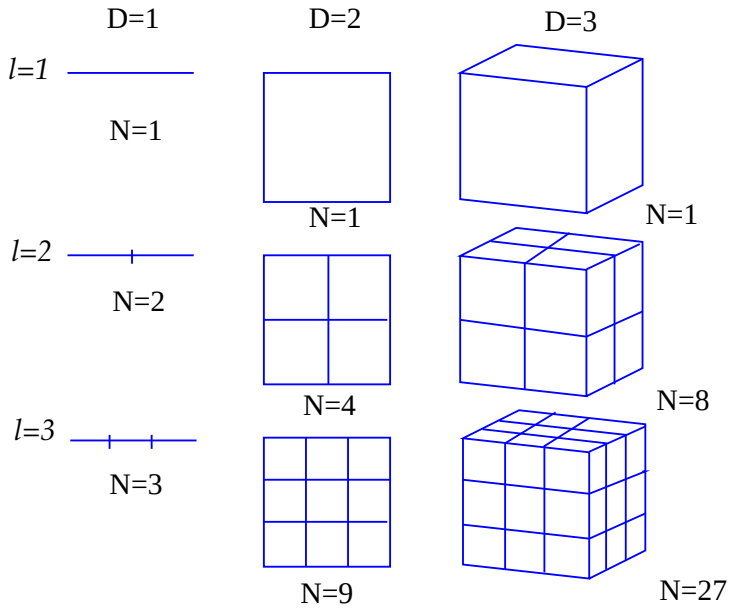


Figure 6.6: Computing the dimension of a line, a square and a cube

.

We generalize the above definition of dimension, and call

$$D = \lim_{\epsilon \to 0} \frac{\ln\left(N(\epsilon)\right)}{\ln\left(1/\epsilon\right)}, \tag{6.2}$$

the fractal dimension, where $N(\epsilon)$ is the number of smaller self–similar objects of linear size ϵ needed to cover the whole object.

In the case of the Koch curve the fractal dimension is

$$D = \frac{\ln 4}{\ln 3} = 1.26, \tag{6.3}$$

Going back to the Lorenz attractor, it can be shown that it is 2.05.

Chapter 7

Applications

7.1 Weather Forecast

Predicting the weather requires solving the governing hydrodynamical equations. The atmosphere does not always posses a low–dimensional chaotic attractor, but its dynamics is often subject to sensitivity to initial conditions.

Weather services perform *ensemble forecasts*, calculating the future of a number of slightly different initial conditions to assess the predictability of the weather.

Fig. 7.1 shows the actual outcome (top left) and 15 different 132-hour forecasts of mean sea level pressure over Europe obtained starting from slightly different initial conditions (from "Chaos and Weather Prediction" by Roberto Buizza, European Centre for Medium–Range Weather Forecasts, Jan 2001).

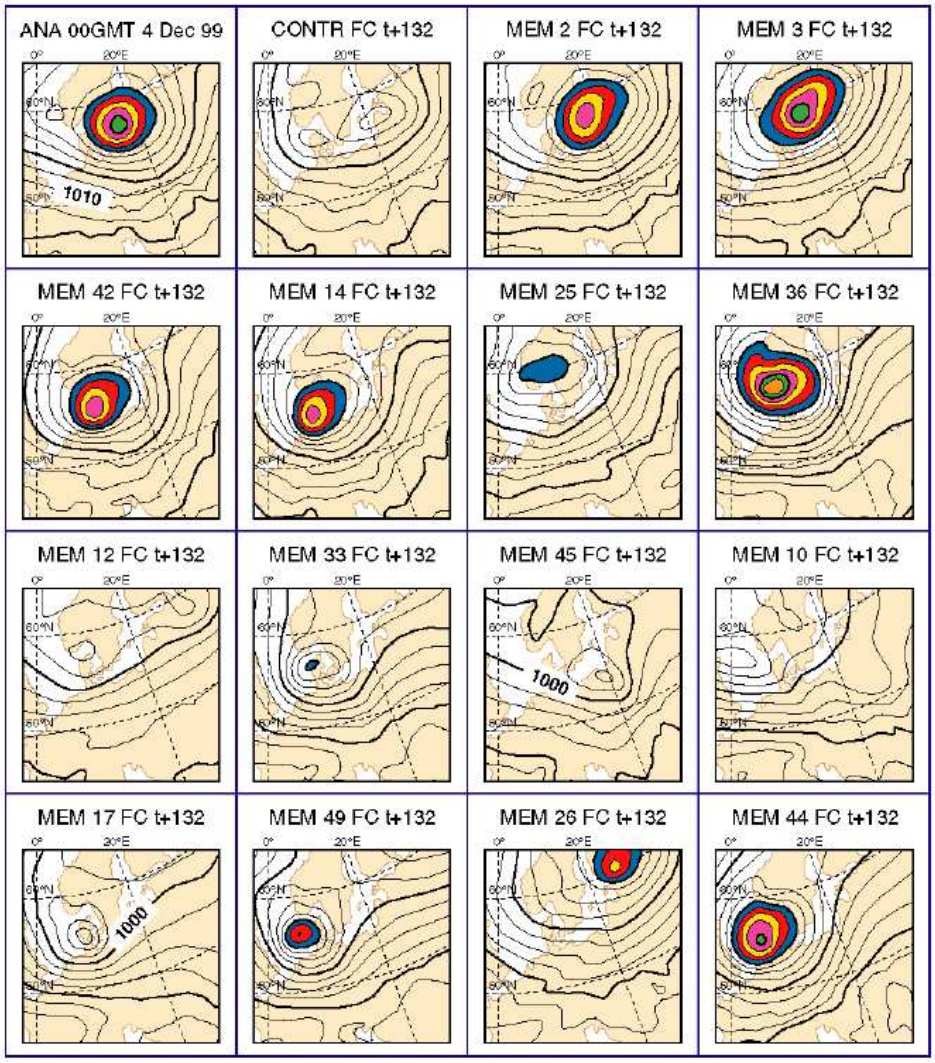


Figure 7.1: Chaos in weather forecast

7.2 Chaotic Advection

Small particles (molecules, dust, pollutants) in a fluid at rest spread by **diffusion**. Diffusion is a very slow process. Typical diffusion coefficients in water or air are $K \sim 10^{-9}\text{m}^2/\text{sec}$ and $10^{-5}\text{m}^2/\text{sec}$.

Suppose that you are at the restaurant and are waiting for the pizza which you have ordered. If you had to wait for diffusion to bring the smell of the pizza from the kitchen to you, it would take a time of the order of δ^2/K. If $\delta \sim 10\text{m}$ is the distance from your table to the kitchen, you would have to wait about $10^2/10^{-5} \sim 10^7\text{sec} = 115\text{days}$! The reason for which it takes a much shorter time to notice that the pizza is ready is that there are always turbulent currents in the air which **advect** the molecules.

Figure 7.2: Chaotic advection (from Biferale et al. Le Scienze, 443, 2005)

Advection is usually chaotic. Materials spread along filamentary fractal patterns and do not keep the same compact spatial distribution which they might have had at the initial time. In many cases the advected particles are not passive but undergo chemical (eg fire) or biological reactions (eg plankton) within the flow. Chaotic advection is thus important in the environment.

7.3 Advection in a shear flow

To model a 2-dim flow which extends over a region of space away from boundaries, we use a periodic square box of size L in both x and y directions. We assume that the flow has the form of a shear, with opposite velocities in opposite halves of each square. We also assume that it is periodic in time with period τ, such that it changes direction in each half–period. A simple realization of such flow is $\mathbf{v} = (u, v)$ where the components u and v are given by (see Fig. 7.3)

$$u(x,y,t) = \begin{cases} A\sin(2\pi y/L) & \text{if } 0 \le t < \tau/2 \\ 0 & \text{if } \tau/2 \le t < \tau \end{cases}$$

$$v(x,y,t) = \begin{cases} 0 & \text{if } 0 \le t < \tau/2 \\ A\sin(2\pi x/L) & \text{if } \tau/2 \le t < T \end{cases}$$

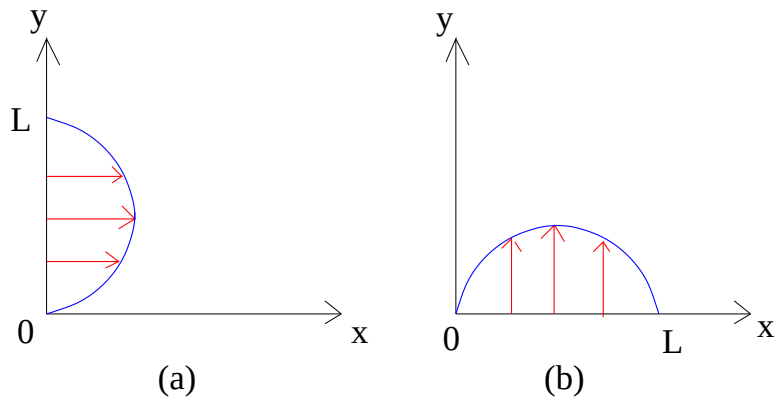


Figure 7.3: Flow for $0 \le t < \tau/2$ (a) and for $\tau/2 \le t < \tau$ (b).

A tracer particle of position $\mathbf{r}(t)$ obeys

$$\frac{d}{dt}\mathbf{r}(t) = \mathbf{v}(\mathbf{r}, t),$$

where the right hand side is the velocity of the flow at the position of the tracer. To model a blob of fluid, we follow the evolution of a large number ($N = 10,000$) of tracers, which initially are placed in a small region near the origin. Periodic boundary conditions are applied to the tracers (that it, when a tracer leaves the square on one side, it is advected back to the other side).

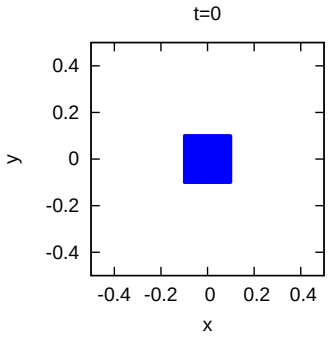


Figure 7.4: $t = 0$

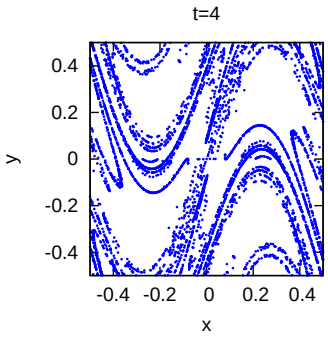


Figure 7.5: $t = 4$

The initial condition is shown in Fig. 7.4. Fig. 7.5, 7.6, and 7.7 show the evolution in time. Note that the blob is stretched and folded. This simple flow can create a great amount of **mixing**. Note also that there are islands where the tracers do not go.

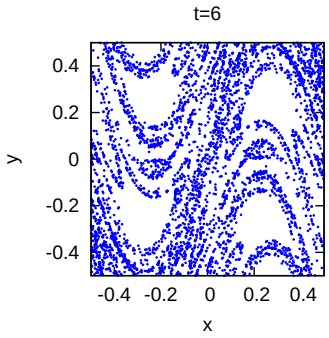


Figure 7.6: $t = 6$

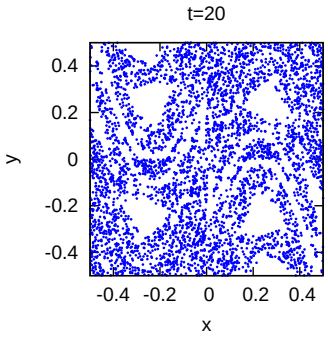


Figure 7.7: $t = 20$

7.4 Vortices

Leonardo da Vinci was the first to realize that turbulent flows consist of vortices.

Figure 7.8: Leonardo da Vinci

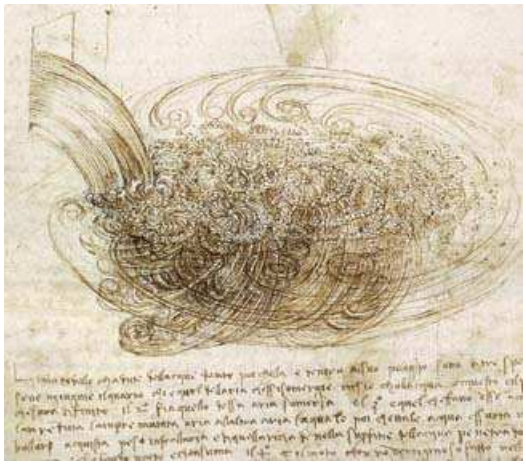


Figure 7.9: Leonardo's drawing of turbulence

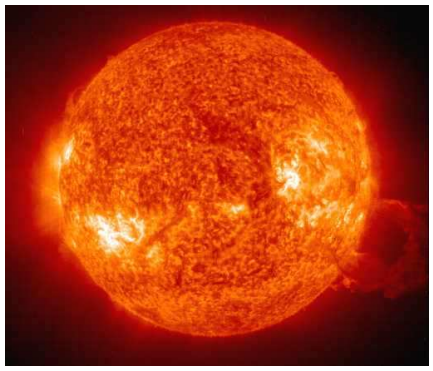


Figure 7.10: Turbulence on the Sun (image taken by SOHO telescope, European Space Agency)

Figure 7.11: Mount St Helens (image taken by USGS)

Figure 7.12: Waterspout at Singapore.

The simplest vortex is a **vortex point**, which models a thin–cored vortex, such as a tornado or a waterspout (see Fig. 7.12). We assume that the vortex is aligned aling z; the speed v of the flow $\mathbf{v}$ around the point (x_0, y_0) in the x, y plane decreases with the distance r from that point:

$$v = |\mathbf{v}| = \frac{\Gamma}{2\pi r}, \tag{7.1}$$

where the parameter Γ, called circulation, describes the strength of the vortex. The Cartesian components of $\mathbf{v}$ are thus

$$u = -\Gamma \frac{(y - y_0)}{2\pi r^2}, \tag{7.2}$$

$$v = \Gamma \frac{(x - x_0)}{2\pi r^2}, \tag{7.3}$$

where $r^2 = (x - x_0)^2 + (y - y_0)^2$. A single vortex point creates a long–range velocity field around it but remains stationary. A theorem of fluid dynamics states that a vortex moves with the local velocity field. This velocity field can arise from any other vortex in the flow. So, if there are two vortices, each vortex moves under the influence of the other vortex. The actual motion depends on the relative sign of the circulation.

If the two vortices have the same circulation then they rotate around each other at angular frequency which is inversally proportional to the mutual separation. This is shown in Fig 7.13:

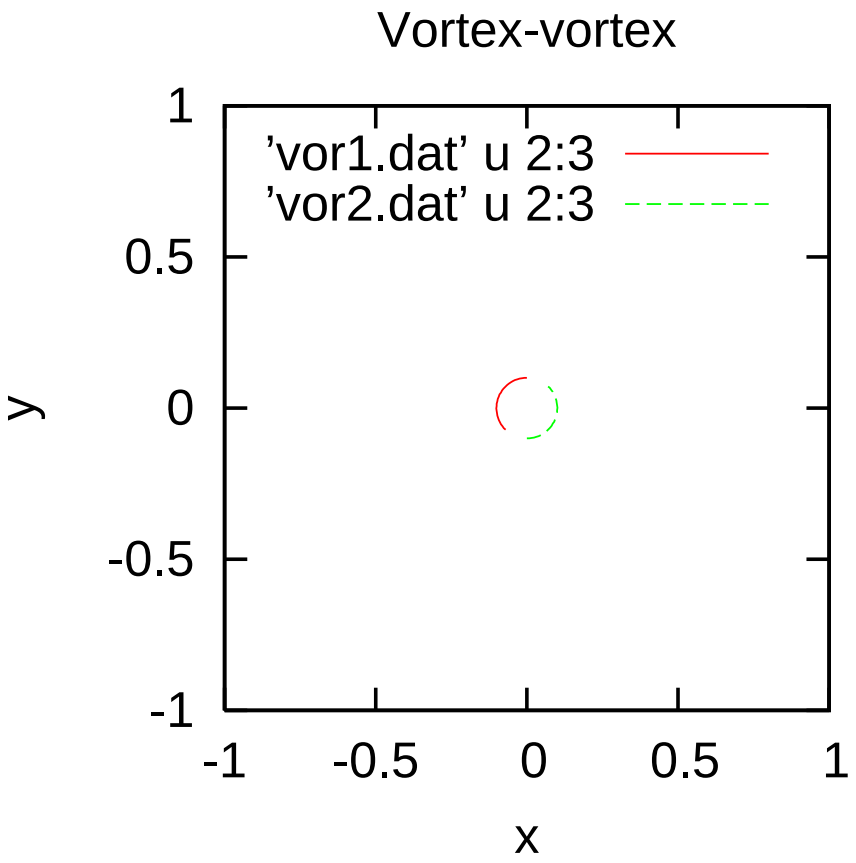


Figure 7.13: Trajectories for $0 \leq t \leq 0.3$ of two vortices with the same circulation $\Gamma = 1$ initially located at the points $(0, 0, 1)$ and $(0, -0.1)$.

If the two vortices have opposite circulation, the resulting motion is very different. in this case the vortex–antivortex pair has translational velocity. The speed is inversally proportional to the vortex separation, as shown in Fig. 7.14.

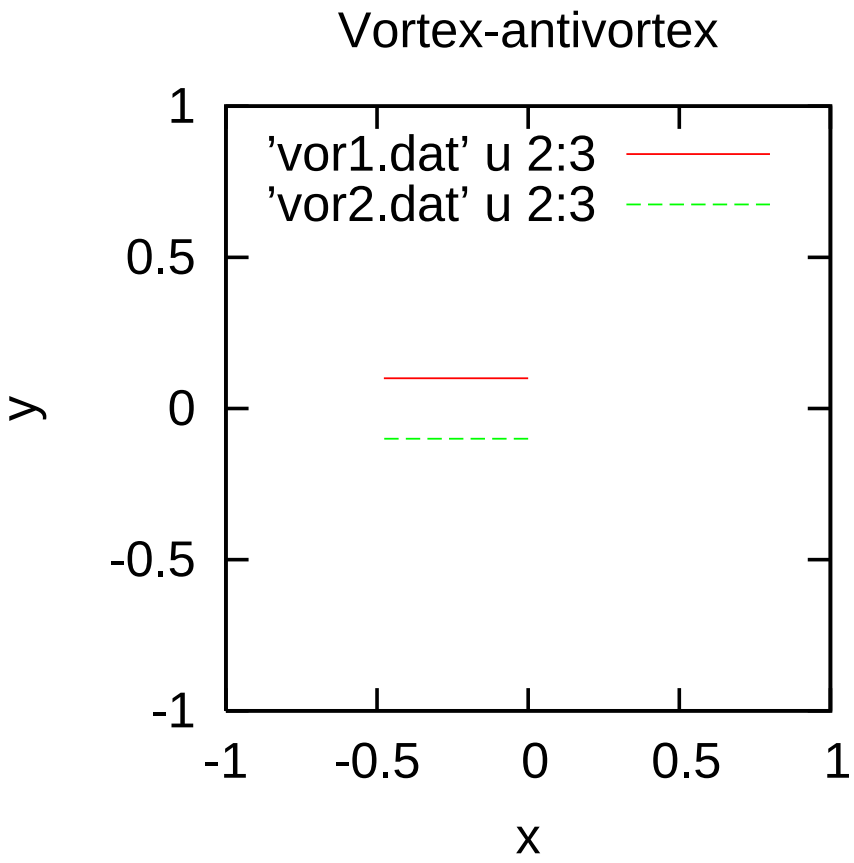


Figure 7.14: Trajectories for $0 \leq t \leq 0.6$ of two vortices with opposite circulation; The vortex at the point $(0, 0, 1)$ has circulation $\Gamma = -1$ and the vortex at $(0, -0.1)$ has circulation $\Gamma = 1$.

If the flow contains N vortices $j = 1., \cdot N$, then the velocity $(u_i, v_i) = (\dot{x}_i, \dot{y}_i)$ of the vortex i is given by

$$\dot{x}_i = -\sum_{j \neq i}^{N} \Gamma_j \frac{(y_J - y_i)}{2\pi r_{ij}^2}, \tag{7.4}$$

$$\dot{y}_i = \sum_{j \neq i}^{N} \Gamma_j \frac{(x_j - x_i)}{2\pi r_{ij}^2}, \tag{7.5}$$

where $r_{ij}^2 = (x_i - x_J)^2 + (y_i - y_j)^2$. Phase space has $2N - 5$ dimensions, because there are five conserved quantities (one is the energy for example). The motion of three vortices has dimension $2N - 5 = 1$ and is periodic. In the case of the four–vortex problem ($N = 4$), the dimension of phase space is $2N - 5 = 8 - 5 = 3$, thus **four or more vortices move chaotically** under the influence of each other.

Motion of four vortices:

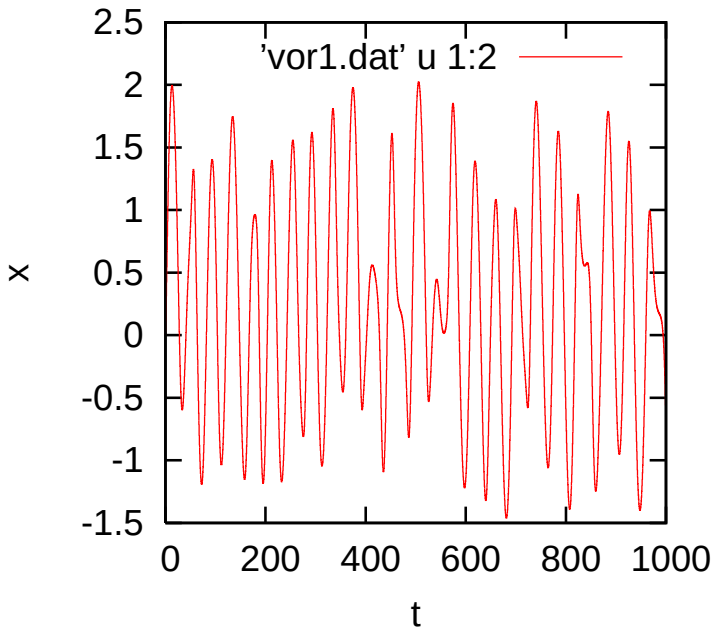


Figure 7.15: x coordinate of the first vortex plotted vs time.

Motion of four vortices:

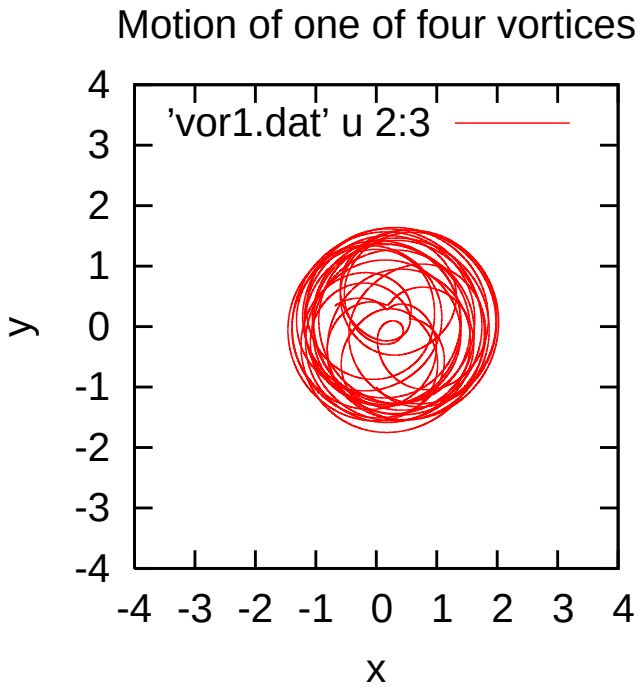


Figure 7.16: Trajectory of the first vortex in the xy plane.

7.5 Planets

According to Newton's theory of gravity, the equation of motion of a planet of mass m_i and position $\mathbf{r}_i$ in the presence of other planets of mass m_j and position $\mathbf{r}_j$ is

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = Gm_j \sum_{j \neq i} \frac{(\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3}, \tag{7.6}$$

where G is the universal constant of gravitation.

For a two–body system there is a simple solution: the trajectory is a conic section (circle, ellipse, parabola or hyperbola).

For a three–body system (eg the Earth, the Moon and the Sun) there is no analytic solution, despite centuries of attempts.

Figure 7.17: Mars

Figure 7.18: Asteroid Gaspra

Recent work has shown that the Solar System, previously the paradigm of determinism, contains chaotic phenomena, often associated with gravitational resonances (which occurs when the orbital periods of two planets are in the ratio of two small integers, e.g., 1:2, 3:5, etc).

- The asteroid belt is chaotic. Asteroid which cross a planet's orbit may collide with it, as it happened to the Earth many times.

- Saturn's satellite Hyperion has a positive inverse Liapunov exponent $1/\lambda \sim 10$ days.
- The inner planets, Mercury, Venus, Earth and Mars, have $1/\lambda \sim 5$ million years.
- The outer planets are more regular, but Pluto has $1/\lambda \sim 20$ million years.

These number may seem long on human scales, but they are still short on the lifetime of the Solar System, which is several billion years.

7.6 A bit of philosophy

The great successes of Newtonian mechanics in predicting the behaviour of physical objects, from small bodies (eg falling apples) to large bodies (eg planets), led to the belief that, if we know both the equations of motion of a system and its initial conditions, we can find its state at any later time.

Figure 7.19: Isaac Newton

Figure 7.20: Pierre-Simon de Laplace

Philosophers called this view **determinism**. They extrapolated that, since God must know both the equations of motion of the Universe and its current state with perfect accuracy, He must know the future too. Determinism was best expressed by the great French mathematician Laplace:

Une intelligence qui pour un instant donné connaîtrait toutes les forces dont la nature est animée et la situation respective des êtres qui la composent, si d'ailleurs elle était assez vaste pour soumettre ces données à l'analyse, embrasserait dans la même formule les mouvements des plus grands corps de l'Univers et ceux du plus léger atome : rien ne serait incertain pour elle, et l'avenir comme le passé seraient présents à ses yeux.

Translation: **An intelligence that at any given instant knew all of the forces that animate nature and the mutual positions of the beings that compose it, if this intellect were vast enough to submit the data to analysis, could condense into a single formula the movement of the greatest bodies of the Universe and that of the lightest atom; for such an intelligence nothing could be uncertain and the future just like the past would be present before its eyes**.

In practice we can physically measure the initial state of any system only with relative accuracy. If the governing equations of motion are *linear* the difference between prediction and actual outcome must be of the order of magnitude of the difference between the actual initial condition (which we cannot measure) and the approximate initial condition (which we measure). This means that, even if we cannot predict the future exactly, we expect to be able to reduce prediction errors by measuring the initial condition better.

Familiarity with linear equations led people to believe that this is true in general, hence the determinism as conceived by Newton and Laplace.

The discovery of chaos defined determinism. Now we know that if the governing equations of motion are nonlinear and phase space is big enough, it is possible that there are regions of parameter space where solutions are chaotic. If a system is chaotic, long-term predictions are impossible because we cannot measure the initial conditions with infinite precision. This restriction also affects numerical calculations, because computers have a finite memory and cannot keep the infinite digits required to represent a real number.

Chapter 8

Examples

1. **Example:** The following dynamical system describes the interaction of two populations x and y:

$$\dot{x} = 14x - 2x^2 - xy,$$
$$\dot{y} = 16y - 2y^2 - xy,$$

   (a) Find the fixed points of the system.

   (b) Interpret the fixed points in terms of extinction or co-existence of the populations.

2. **Example:** According to Newton's law, a body of mass m and position $x(t)$ which is under the action of the force $F(x,t)$ obeys the equation

$$m\ddot{x} = F,$$

   Show that Newton's equation is a 2–dimensional dynamical system.

3. **Example:** Consider the following model of the interaction between a population of rabbits, $R(t)$, and a population of foxes, $F(t)$, in a given area. In the absence of foxes the rabbit population grows without limit; in the absence of rabbits the foxes starve to death. Rabbits are eaten by foxes. No rabbits or foxes enter or exit the area. The resulting equations are:

$$\frac{dR}{dt} = \alpha R - \beta RF,$$
$$\frac{dF}{dt} = -\gamma F + \delta FR,$$

   where α, β, γ and δ are positive coefficients.

(a) Interpret the equations.

(b) Find the fixed points of the system.

(c) Write the equations as a dynamical system $\dot{\mathbf{x}} = \mathbf{f}$, hence define $\mathbf{x}$, $\mathbf{f}$ and the dimension N.

4. **Example:** Write the equation

$$m\ddot{x} + b\dot{x} + kx = F\cos\omega t,$$

(a) as a 2-dimensional non–autonomous dynamical system.

(b) as a 3-dimensional autonomous dynamical system.

5. **Example:** Write the equation

$$\ddot{x} + x\dot{x} + t\dot{x}^2 = 0,$$

(a) as a 2-dimensional non–autonomous dynamical system.

(b) as a 3-dimensional autonomous dynamical system.

6. **Example:** Consider the equation

$$\ddot{x} + 2\dot{x}^2 + 3x^3 = \sin t;$$

(a) Show that it is a 2-dim non-autonomous dynamical system of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$. Identify $\mathbf{x}$ and $\mathbf{f}$.

(b) Show that it can be transformed into a 3-dim autonomous dynamical system of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. Identify $\mathbf{x}$ and $\mathbf{f}$.

7. **Example:** A simple model of a population $x(t)$ is

$$\dot{x} = rx,$$

where $x \geq 0$ and r, the difference between the birth rate and the death rate, is called the growth rate. The growth rate can be either positive or negative.

(a) Find $x(t)$ in terms of the initial condition $x(0)$.

(b) Find the fixed points, sketch the 1–dimensional phase flow, and predict the long term behaviour of the solution $x(t)$ for $t \to \infty$.

(c) Discuss some evident limitations of the model.

8. **Example:** Consider the 1–dimensional dynamical system for $x \geq 0$:

$$\dot{x} = \sin x,$$

   (a) Find the fixed points.

   (b) Sketch the phase flow.

   (c) What is the long term behaviour $x(t)$ for $t \to \infty$ if the initial condition is $x(0) = 5\pi/2$?

9. **Example:** Consider the same equation as in the previous example,

$$\dot{x} = \sin x,$$

   (a) Separate the variables and find the general solution of the equation.

   (b) Find the solution which corresponds to the initial condition $x = \pi/2$ at $t = 0$. What is the long-term behaviour of this solution for $t \to \infty$?

10. **Example:** Consider the 1–dimensional dynamical system

$$\dot{x} = x^2 - x^4,$$

    in $-\infty < x < \infty$.

    (a) Find the fixed points.

    (b) Sketch the phase flow.

    (c) What is the long-term behaviour of the solution for $t \to \infty$ if the initial condition is $= 0.3$ at $t = 0$?

11. **Example:** A population $x(t)$ of deers in a forest obeys the logistic equation with carrying capacity K and growth rate $r > 0$. A constant number of deers is hunted every year. We have

$$\dot{x} = rx(1 - \frac{x}{K}) - h,$$

    where $x \geq 0$, $K > 0$ and $h > 0$ is the hunting rate.

    Let $F = rx(1 - x/K)$ and $H = h$. Plot the functions F and H and determine the fixed points. Sketch the 1–dimensional phase flow and predict the qualitative long term behaviour of the population depending on whether $F > H$ or $F < H$.

12. **Example:** To model diseases such as flu, measles, smallpox or AIDS we divide the population into three classes: susceptibles, $S(t)$, infectives, $I(t)$, and recovered, $R(t)$. Individuals are born in the S class which consists of people who have never been in contact with the disease. Members of the S class who catch the diseases move to the I class of infected people. Infective people can spread the disease to susceptible people. After a certain time typical of the disease, infective people move to the class R of people who have recovered and have become immune. If the disease evolves quickly we can neglect birth and death events, and the governing equations are

$$\begin{aligned} \dot{S} &= -\beta IS, \\ \dot{I} &= \beta IS - gI, \\ \dot{R} &= gI, \end{aligned}$$

The parameter β is the contact rate (susceptibles become infected at the rate βI). Infected individuals recover at the rate g, so $1/g$ measures the mean infectious period. The total population is constant, and for simplicity it is normalised to unity: $S + I + R = 1$.

We have assumed that the disease evolves quickly, so we have neglected the natural birth and rates (if the disease remains in the population for a long time, we need to include these effects, as newborn individuals replenish the class S and members of all classes have a certain probability of dying).

(a) Verify that
$$\frac{d}{dt}(S + I + R) = 0.$$

(b) Find the fixed points of the system.

(c) Make a graph of the fixed points in the (S, I, R) space. Divide the first equation by the third and show that
$$S(t) = S(0) \exp\left(-\beta(R(t) - R(0))/g\right).$$

13. **Example:** Now consider a disease which evolves slowly. Unlike the previous example, we must take into account births and deaths. For simplicity assume that birth rate and death rate are the same, m. As before, the total population is normalised to unity. The governing equations are

$$\begin{aligned}\dot{S} &= m - \beta IS - mS,\\ \dot{I} &= \beta IS - mI - gI,\\ \dot{R} &= gI - mR,\end{aligned}$$

Show that the above equations have two fixed points (S^*, I^*, R^*), one which corresponds to eradication of the disease ($I^* = R^* = 0$ and $S^* = 1$) and one which corresponds to epidemic equilibrium.

14. **Example:** The equation of motion of a body of mass m which is linked to the origin by a spring of stiffness k and which moves along the x axes is $m\ddot{x} = -kx$. Assume $k = m = 1$ hereafter.

    (a) Let $v = \dot{x}$ and write the equation as a 2–dimensional dynamical system.

    (b) Identify the position vector in phase space and the velocity vector in phase space.

    (c) Find the velocity vector in phase space at the four points $A = (0, 1)$, $B = (1, 0)$, $C = (0, -1)$ and $D = (-1, 0)$. Sketch these velocity vectors in phase space.

    (d) Find the magnitude of the velocity vector in phase space and show that it increases with the distance from the origin.

    (e) Solve the equation $\ddot{x} = -x$ and determine $x(t)$ and $v(t)$ such that $x(0) = 0$ and $v(0) = 1$.

    (f) Sketch x vs t and v vs t, identifying A, B, C and D.

    (g) Sketch v vs x (motion in phase space), identifying A, B, C and D.

15. **Example:** Consider the 1–dimensional dynamical system

$$\dot{x} = x - 2\sin x,$$

    for $x \geq 0$.

    (a) Find the fixed points (hint: think of a graphical method).

    (b) Determine whether the fixed points are stable (sinks) or unstable (sources), and sketch the phase portrait.

    (c) Determine the long–term behaviour $x(t)$ for $t \to \infty$ in terms of the initial condition $x(0)$.

16. **Example:** Suppose that $x_1(t)$ and $x_2(t)$ are two solutions of the equation
$$\dot{x} = x + x^2.$$
Is $x_3 = x_1 + x_2$ a third solution of the same equation ?

17. **Example:** The general solution $\mathbf{x}(t)$ of a linear 2-dimensional dynamical system $\dot{\mathbf{x}} = A\mathbf{x}$ is
$$\mathbf{x}(t) = c_1\mathbf{U}e^{\lambda_1 t} + c_2\mathbf{V}e^{\lambda_2 t},$$
where $\lambda_1 = 2$, $\lambda_2 = 1$, $\mathbf{U} = (1, 1)$, $\mathbf{V} = (1, -1)$ are respectively the eigenvalues and the eigenvectors of the matrix A; c_1 and c_2 are arbitrary constants. Write down the components $x(t)$ and $y(t)$ of the solution $\mathbf{x}(t)$ which corresponds to the initial condition $x(0) = 3$, $y(0) = 0$:

18. **Example:** Consider the dynamical system
$$\dot{x} = ax,$$
$$\dot{y} = -y,$$
   (a) Determine the nature of the fixed point at the origin and sketch trajectories in the four cases: $a < -1$, $a = -1$, $-1 < a < 0$, $a > 0$.
   (b) Assume the initial condition $x(0) = 1$, $y(0) = 1$ at $t = 0$. Predict the long term behaviour of the solution $x(t), y(t)$ for $t \to \infty$ in the two cases: $a = -3$ and $a = 3$.

19. **Example:** Consider the equations
$$\dot{x} = 3x - 6y,$$
$$\dot{y} = x - 4y,$$
   (a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.
   (b) Find the determinant and the trace of A.
   (c) Determine the eigenvalues λ_1 and λ_2.
   (d) Which kind of fixed point is the origin ?
   (e) Find the eigenvectors $\mathbf{U}$ and $\mathbf{V}$.

(f) Find the general solution $\mathbf{x}(t)$.

(g) Draw the phase portrait. Highlight any separatrix.

(h) What is the behaviour of $\mathbf{x}(t)$ for $t \to \pm\infty$?

(i) What happens to trajectories which start on $y = x$ and $y = x/6$?

(j) Determine the slope of the trajectories.

20. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= 4x - 2y, \\ \dot{y} &= x + y, \end{aligned}$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ_1 and λ_2.

(d) Which kind of fixed point is the origin ?

(e) Find the eigenvectors $\mathbf{U}$ and $\mathbf{V}$.

(f) Find the general solution $\mathbf{x}(t)$.

(g) Draw the phase portrait. Highlight any separatrix and consider the slope of trajectories.

(h) What is the behaviour of $\mathbf{x}(t)$ for $t \to \pm\infty$?

21. **Example:** Consider the dynamical system

$$\begin{aligned} \dot{x} &= -3x, \\ \dot{y} &= -2y, \end{aligned}$$

Sketch the phase portrait, paying attention to the slope of the trajectories for $t \to \pm\infty$ (hint: solve directly the decoupled equations).

22. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= 2x + y, \\ \dot{y} &= -x + 2y, \end{aligned}$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ_1 and λ_2 and show that the origin is an unstable spiral. Check that the eigenvalues are complex conjugates of each other.

(d) Determine the eigenvectors and show that they are complex conjugate of each other.

(e) Find the general solution $\mathbf{x}(t)$.

(f) Draw the phase portrait. Is the sense of rotation clockwise or anticlockwise ?

(g) Determine how the radius of the spiral changes with time.

23. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -x, \end{aligned}$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ_1 and λ_2 and show that the origin is a centre.

(d) Find the general solution $\mathbf{x}(t)$.

(e) Find the radius of the orbit and how it changes with time.

(f) Draw the phase portrait. Is the sense of rotation clockwise or anticlockwise ?

24. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= -y, \\ \dot{y} &= 4x, \end{aligned}$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ_1 and λ_2 and show that the origin is a centre.

(d) Draw the phase portrait. Is the rotation clockwise or anticlockwise ?

(e) Find the general solution.

(f) Find the solution which satisfies the condition $x(0) = 1$ and $y(0) = 0$. Sketch the trajectory, determine the period and find the minimum and maximum values of $x(t)$ and $y(t)$.

25. **Example:** Consider the equations

$$\dot{x} = 2x,$$
$$\dot{y} = 2x,$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ_1 and λ_2.

(d) Find the eigenvectors.

(e) Find the general solution $\mathbf{x}(t)$.

(f) Show that the y axis is a line of fixed points and sketch the phase portrait.

(g) Solve directly the two equations $\dot{x} = 2x$ and $\dot{y} = 2x$, and verify that the solution (written in terms of the initial condition $x(0)$ and $y(0)$) is the same.

26. **Example:** Consider the dynamical system

$$\dot{x} = y,$$
$$\dot{y} = y,$$

Show that the phase portrait is a line of fixed points.

27. **Example:** Consider the equations

$$\dot{x} = -5x,$$
$$\dot{y} = -5y,$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Show that there is only one eigenvalue.

(d) Find the corresponding eigenvector.

(e) Find the general solution $\mathbf{x}(t)$.

(f) Draw the phase portrait, show that the origin in a stable star node, and show that trajectories have constant slope.

(g) Show that the origin is a stable star node.

28. **Example:** Consider the equations

$$\dot{x} = 4x - y,$$
$$\dot{y} = 2x + y.$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Determine the eigenvalues λ and λ_2.

(d) Which kind of fixed point is the origin ?

(e) Find the eigenvectors $\mathbf{U}$ and $\mathbf{V}$.

(f) Find the general solution $\mathbf{x}(t)$.

(g) What is the behaviour of $\mathbf{x}(t)$ for $t \to \pm\infty$?

(h) Find the slope of the trajectories and sketch the phase portrait.

29. **Example:** Consider the dynamical system

$$\dot{x} = x + 13y,$$
$$\dot{y} = -2x - y.$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the matrix A and the vector $\mathbf{x}$.

(b) Find the determinant and the trace of A.

(c) Find the eigenvalues λ_1 and λ_2. Check that the eigenvalues are complex conjugate of each other.

(d) Which kind of fixed point is the origin ?

(e) Find the eigenvectors $\mathbf{U}$ and $\mathbf{V}$. Check that the eigenvectors are complex conjugate of each other.

(f) Consider the general solution

$$\mathbf{x}(t) = c_1\mathbf{U}e^{\lambda_1 t} + c_2\mathbf{V}e^{\lambda_2 t},$$

where c_1 and c_2 are arbitrary constants. Show that, for $\mathbf{x}(t)$ to be real, the complex conjugate of c_1 must be c_2.

(g) Let $\mathbf{U} = \mathbf{R} + i\mathbf{S}$, $c_1 = fe^{i\eta}$ and show that the general solution can be written as

$$\begin{aligned} x(t) &= 2f\cos(\eta + 5t), \\ y(t) &= -\frac{2}{13}f\cos(\eta + 5t) - \frac{10}{13}f\sin(\eta + 5t), \end{aligned}$$

where f and η are arbitrary real numbers.

30. **Example:** Consider the dynamical system

$$\begin{aligned} \dot{x} &= -x + y, \\ \dot{y} &= -y. \end{aligned}$$

(a) Put the equations in the form $d\mathbf{x}/dt = A\mathbf{x}$ and identify the vector $\mathbf{x}$ and the matrix A.

(b) Find the trace τ and the determinant of A.

(c) Show that $\lambda_1 = \lambda_2$.

(d) Find the general solution $\mathbf{x}(t)$. (hint: note that the equation for y does not contain x and can be solved directly. Once you know $y(t)$, integrate the equation for x to find $x(t)$.)

(e) Find the slope of the trajectories and state the nature of the fixed point at the origin; sketch the phase portrait.

31. **Example:** Consider the dynamical system

$$\begin{aligned} \dot{x} &= -2x + 2y, \\ \dot{y} &= 2x - 5y, \end{aligned}$$

(a) Show that the system can be written as $\dot{\mathbf{x}} = A\mathbf{x}$. Identify the vector $\mathbf{x}$ and the matrix A.

(b) Find the trace τ and the determinant δ of A.

(c) Find the eigenvalues λ_1 and λ_2.

(d) Determine the nature of the critical point at the origin.

(e) Determine the eigenvectors $\mathbf{U}$ and $\mathbf{V}$.

(f) Write the general solution.

(g) Sketch the phase portrait paying attention to the behaviour for $t \to \pm\infty$.

32. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= -3x + y, \\ \dot{y} &= x - 3y, \end{aligned}$$

(a) Determine the nature of the critical point at the origin.

(b) Find the eigenvectors.

(c) Sketch the phase portrait, paying attention to any vertical or horizontal slope of the trajectories.

(d) Find the solution which corresponds to the initial condition $x = 2$, $y = 0$ at $t = 0$. Sketch the resulting trajectories in the phase plane.

33. **Example:** Consider the equations

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -4x, \end{aligned}$$

(a) Show that the critical point at the origin is a centre.

(b) Are trajectories clockwise or anticlockwise around the origin ?

(c) Write the general solution.

(d) Do trajectories keep a constant distance from the origin ?

(e) Make a sketch of the orbits.

(f) what is the period ? Does it depend on the initial condition ?

34. **Example:** Two countries, X and Y, are locked in an arms race. Call $x(t)$ and $y(t)$ the military expenditures of country X and country Y at time t. By definition $x \geq 0$, $y \geq 0$. Each country's increase/decrease of military budget is proportional to the other's country's budget, so we model the situation by writing

$$\dot{x} = \alpha y,$$
$$\dot{y} = \beta x.$$

where the parameters α and β are positive. Show that the arms race will result in out-of-control military expenses, $x \to \infty$ and $y \to \infty$ for $t \to \infty$ (the model is clearly going to break down when x or y become too large, for example there will be a war, or an economic collapse).

35. **Example:** Steven Strogatz developed a mathematical model of the story of Romeo and Juliet[1]. Let $R(t)$ be Romeo's love/hate for Juliet at time t, and $J(t)$ be Juliet's love/hate for Romeo at time t. Positive values of R or J mean love, negative values mean hate, and zero means indifference. The model is

$$\dot{R} = bJ,$$
$$\dot{J} = cR.$$

The parameters b and c are the *responsiveness*. For example, $b > 0$ means that Romeo is excited by Juliet's love ($J > 0$): if both b and J are positive, the term bJ at the RHS is positive, which makes R to increase.

(a) Let us assume that $b < 0$, that is to say Romeo is a fickle lover: the more Juliet loves him (i.e. the greater J is), the more he dislikes her (the term bJ, large and negative, makes R to decrease); when Juliet loses interests in Romeo, his feelings for her warm up. We also assume that $c > 0$, which means that the love of Juliet for Romeo grows when he loves her, but she hates him when he hates her.

Show that the outcome of Romeo and Juliet's love affair is a never ending cycle of love and hate. Also show that at least Romeo and Juliet achieve simultaneous love 25 percent of the time.

[1] S.H. Strogatz, Mathematical Magazine **61**, 53 (1988).

(b) What is the outcome of the love story if $b > 0$, $c > 0$?

36. **Example:** We improve the previous model of Romeo and Juliet and write

$$\dot{R} = aR + bJ,$$
$$\dot{J} = cR + dJ,$$

where the parameters c, d are Romeo's and Juliet's *cautiousness*. We have seen that $b > 0$ means that Romeo gets excited by Juliet's love; $a > 0$ means that he is further excited by his own feelings; $a < 0$ means that the more he likes her (i.e. the larger $R > 0$ is), the more he wants to run away from her ($aR < 0$ makes R to decrease: perhaps he is afraid of a marriage decision). Strogatz suggested the following interpretations for positive and negative values of responsiveness and cautiousness: $a > 0$, $b > 0$: eager beaver; $a < 0$, $b < 0$: cautious and unresponsive (not a good chance for a romance); $a > 0$, $b < 0$: daring but unresponsive (more the narcisist type); $a < 0$, $b > 0$ cautious lover.

(a) What happens if Romeo and Juliet are equally cautious lovers ($a < 0$, $b > 0$ with $c = b$ and $d = a$) ?

(b) What happens if $a = b = 0$ (Romeo is a robot, nothing can change his feelings for Juliet) ?

37. **Example:** Consider the equations

$$\dot{x} = x - y^2 = f_1,$$
$$\dot{y} = xy - y = f_2,$$

(a) Find the fixed points $\mathbf{x}^*$.

(b) Find the Jacobian matrix $D\mathbf{f}(\mathbf{x})$.

(c) At each fixed point evaluate the Jacobian matrix; then write down the linearised system $\dot{\mathbf{X}} = A\mathbf{X}$ near that fixed point, and identify the vector $\mathbf{X}$ and the matrix A.

38. **Example:** Consider the equations

$$\dot{x} = x(3 - x - 2y),$$
$$\dot{y} = y(2 - x - y).$$

(a) Find the fixed points $\mathbf{x}^*$.

(b) Find the Jacobian matrix $D\mathbf{f}(\mathbf{x})$.

(c) At each fixed point evaluate the Jacobian matrix; write down the linearised system $\dot{\mathbf{X}} = A\mathbf{X}$ near that fixed point, and identify the vector $\mathbf{X}$ and the matrix A. Determine eigenvalues, eigenvectors, and the nature of the fixed point.

(d) Using the previous results, apply the Hartman-Grobman Theorem and sketch trajectories in phase space.

39. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= v, \\ \dot{v} &= x - x^3. \end{aligned}$$

(a) Show that the dynamical system is conservative and determine the potential ϕ.

(b) Plot $\phi(x)$.

40. **Example:** Consider the following 1–dimensional dynamical system:

$$\dot{\theta} = 1 - \cos\theta,$$

for $0 \leq \theta \leq 2\pi$. Show that $\theta = 0$ is a fixed point which is attracting but not Liapunov stable.

41. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= x - y - x(x^2 + y^2), \\ \dot{y} &= x + y - y(x^2 + y^2). \end{aligned}$$

(a) Introduce polar coordinates r, θ by letting $x = r\cos\theta$ and $y = r\sin\theta$. Show that the system becomes

$$\begin{aligned} \dot{\theta} &= 1, \\ \dot{r} &= r(1 - r^2). \end{aligned}$$

(b) Show that $r = 1$ is a limit cycle.

42. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= y + x^2 y, \\ \dot{y} &= -x + 2xy. \end{aligned}$$

Is this system a gradient system ?

43. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= y - yx^2, \\ \dot{y} &= 1 - y^2. \end{aligned}$$

Show that the system is reversible.

44. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= \sin y, \\ \dot{y} &= x \cos y. \end{aligned}$$

Show that the system is a gradient system, thus it has no closed orbits. Determine the potential ϕ.

45. **Example:** Consider the following system:

$$\begin{aligned} \dot{x} &= -2xe^{x^2+y^2}, \\ \dot{y} &= -2ye^{x^2+y^2}. \end{aligned}$$

Show that the system is a gradient system. Determine the potential ϕ and make a graph of the equipotential lines.

46. **Example:** Show that if a two-dimensional dynamical system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ is a gradient system then the trajectories cross equipotential lines at right angles.

47. **Example:** The motion of a body of mass m attached to a nonlinear spring is described by Newton's equation equation $m\ddot{x} = F$ where $F = -kx + \beta x^3$ and k and β are constant.

(a) Show that the force is the gradient of a potential ϕ:

$$F = -d\phi/dx,$$

hence determine ϕ.

(b) Show that the energy

$$E = \frac{mv^2}{2} + \frac{kx^2}{2} - \frac{\beta x^4}{4},$$

is conserved along trajectories.

(c) Let $m = 2$, $k = 2$ and $\beta = -4$ and $\dot{x} = v$. Show that Newton's equation can be transformed into the dynamical system

$$\dot{x} = v,$$
$$\dot{v} = -x - 2x^3,$$

Show that the only fixed point $\mathbf{x}^* = (0, 0)$ is a centre of the linearised equations. Sketch the trajectories in the phase space (x, v) corresponding to $E = 1$, 4, 16 and 36.

48. **Example:** Consider the dynamical system

$$\dot{x} = x - xy,$$
$$\dot{y} = 2y - 2xy.$$

(a) Show that there are two fixed points, $\mathbf{x}^* = (0, 0)$ and $(1, 1)$.

(b) Find the Jacobian matrix $D\mathbf{f}(\mathbf{x})$.

(c) Evaluate the Jacobian at $\mathbf{x}^* = (0, 0)$, determine the nature of this fixed point, and write the linearised equations around $\mathbf{x}^*$.

(d) Evaluate the Jacobian at $\mathbf{x}^* = (1, 1)$, determine the nature of this fixed point, and write the linearised equations around $\mathbf{x}^*$.

(e) Sketch the phase portrait.

49. **Example:** Consider the dynamical system

$$\dot{x} = \mu - x^2,$$
$$\dot{y} = -y.$$

where the parameter $\mu > 0$. Proceeding in the usual way, sketch trajectories in the phase plane, paying attention what happens at different values of μ.

50. **Example:** Today is 20 September 2020. The planet's climate has warmed and the Met Office is trying to predict the development of the first hurricane which threatens the British Isles. The tolerance required for the prediction is $\epsilon \sim 10^{-1}$; today's weather's is known to within relative accuracy $\delta \sim 10^{-3}$. Assume that the Liapunov exponent is $\lambda = 0.1$ hours^{-1}.

    (a) After how many days the prediction becomes unreliable ?

    (b) If we could measure today's weather 10 times more accurately, how many extra days of prediction would we gain ?

51. **Example:** You are the commander of a spaceship which is travelling to Pluto. Flying near Saturn, your ship suddenly develops engine problems and you need to make an emergency landing on Hyperion, the nearest satellite of Saturn. With the engines off, your ship computer tells you that it takes 12 hours to reach Hyperion at the current speed.

    Checking your flight manuals, you read that Hyperion has the shape of an irregular potato, its three main axes being 205, 130 and 110 km respectively. The only flat area which is suitable for landing is the bottom of a crater which is 10 km wide. You switch on your radar and determine the current coordinates of the crater with the accuracy of 1 km. Your ship computer can, at least in principle, solve the differential equation ns which govern the orbit of Hyperion and determine the future coordinates of yo ur landing site. Without engine power, you must aim your ship exactly at the location where the centre of this crater will be in 12 hours, before applying the braking rockets.

    In your flight manuals you also read that, because of the tug of Saturn on Hyperion's odd shape, Hyperion's orbit is chaotic: it tumbles irregularly around Saturn with Liapun ov exponent $\lambda \approx 10$ days^{-1}.

    Can you safely land your ship on Hyperion ? Explain your reasoning. (Hint: not all data presented are required for the solution).

52. **Example:** Write the finite–difference recursion formula to solve the differential equation

$$\frac{dx}{dt} = f(x) = x^2 - x + 1,$$

    using Euler's method with discretization $t_n = n\Delta t$, $x_n = x(t_n)$ ($n = 0, 1, 2, \cdots$) and time step Δt.

Given the initial condition $x(0) = 1$ at $t = 0$ and time step $\Delta t = 0.01$, implement your formula and find the approximate numerical solution $x(t)$ at $t = 0.01$, $t = 0.02$ and $t = 0.03$.

53. **Example:** Consider the following dynamical system for $x > 0$ and $y > 0$:

$$\dot{x} = x - \frac{xy}{2} = f_1,$$
$$\dot{y} = -\frac{3y}{4} + \frac{xy}{4} = f_2,$$

(a) Find the fixed points.

(b) Find the Jacobian.

(c) Evaluate the Jacobian at each fixed point, then determine the nature of each fixed point and the linearised solution near each fixed point. Sketch these approximate solutions.

(d) Use Maple to plot the phase diagram with arrows.

(e) Write a Fortran 90 program to solve the dynamical system using Euler's method from $t = 0$ to $t = 8$, using time step $\Delta t = 8 \times 10^{-4}$, Try three different initial conditions: $x(0) = 4$, $y(0) = 3.9$, $x(0) = 4$, $y(0) = 4.0$, and $x(0) = 4$, $y(0) = 4.1$. For each run, save the values of t, x and y to a file at all time steps; use Gnuplot to plot x vs y for the three initial conditions.

54. **Example:** Consider the dynamical system

$$\dot{x} = x + y - x(x^2 + y^2),$$
$$\dot{y} = -x + y - y(x^2 + y^2).$$

(a) Show that $\mathbf{x}^* = (0, 0)$ is a fixed point.

(b) Find the Jacobian matrix $D\mathbf{f}(\mathbf{x})$.

(c) Find the eigenvalues of $A = \mathbf{f}(0, 0)$ and show that the fixed point $(0, 0)$ of the linearised system is an unstable spiral. Sketch the trajectories around the fixed point of the linearised system.

(d) Let $x = r\cos\theta$ and $y = r\sin\theta$ where $r \geq 0$. Multiply the equation for $\dot{x}$ by x and the equation for $\dot{y}$ by y and then add the equations to show that

$$\dot{r} = r(1 - r^2).$$

Show that the acceptable fixed points of this equation are $r^* = 0$ and $r^* = 1$. Plot dr/dt versus r and show that, although the linearised equation predicts a spiral, the spiral does not extend to infinity, because trajectories for $r > 1$ move towards the origin, not away from it.

(e) Multiply the equation for $\dot{x}$ by y and the equation for $\dot{y}$ by x and then subtract the equations to show that

$$\dot{\theta} = -1,$$

then solve this equation.

(f) Put results together and draw a graph of the trajectories of the above system of equations.

(g) Write a Fortran 90 program called *euler9.f90* which solves the above system of equations using the Euler method. Run the program for $0 \leq t \leq 10$ using 10000 time steps and initial condition $x(0) = 0.5$, $y(0) = 0$. The program should output the solution onto a file. Run the program many times to produce data files corresponding to various initial conditions, say: $x(0) = 0.5$, $y(0) = 0$; $u(0) = -0.5$, $y(0) = 0$; $x(0) = 0$, $y(0) = 0.5$; $x(0) = 0$, $y(0) = -0.5$; $x(0) = 2$, $y(0) = 0$; $x(0) = -2$, $y(0) = 0$; $x(0) = 0$, $y(0) = 2$; $x(0) = 0$, $y(0) = -2$. Use Gnuplot to plot all these files to make a graph with many trajectories.

55. **Example:** Two species $x \geq 0$ and $y \geq 0$ interact according to the equations

$$\dot{x} = x(1 - x - y),$$
$$\dot{y} = 2y(x - 4),$$

Find the fixed points, and, proceeding in the usual way, sketch trajectories in the phase plane $x \geq 0$, $y \geq 0$. What is the outcome of the competition between the two species ?

56. **Example:** Start with a segment, divide it in three parts and remove the middle part. Apply the same procedure to the remaining parts, ad infinitum. The result is a fractal object called the Cantor set. What is its fractal dimension ?

Part II

Numerical methods

Chapter 9

Introduction to Programming

9.1 Introduction

These notes are a brief, step by step, introduction to Fortran 90, designed for self-study. They are not meant to be comprehensive. The aim is to quickly learn enough Fortran 90 to solve differential equations on a computer and start doing computational mathematics. Read these notes carefully in front of your computer, writing, compiling and running all the programs which are described. Do all exercises. The best way to learn is to try yourself - the "hands-on" approach is the most efficient. If you have any difficulty with the exercises, check the solutions in the last chapter.

9.2 What is programming ?

A program is a sequence of steps which are executed by the computer to carry out a task. There are hundreds of programming languages:

- **Machine codes** use strings of 0s and 1s to express instructions and they dependent on the underlying hardware.
- **Assembly languages** are also dependent on hardware and utilise a symbolic form to express instructions.
- **High level languages** were developed to ease the programming effort and to provide hardware independence. Different languages have different strengths. Fortran is popular with the scientific and engineering communities, Cobol is used for business applications and C for systems programming.

In general one desires a language with a notation that fits the problem and is simple to write and learn. Fortran is superior to other languages for mathematical and scientific computation, because it was created to match the mathematical notation. Moreover, many diverse and reliable libraries of routines are available and an official standard exists which helps towards portability. For these reasons, in this course you will learn a little Fortran 90.

9.3 Fortran

Fortran (mathematical FORmula TRANslation system) was originally developed in 1954 by IBM. Fortran was one the first high level languages, which saved the programmer from having to work with the details of the underlying computer architecture.

Since then Fortran has changed dramatically on several occasions; in 1958 the second version added subroutines, functions and common blocks. Version 4 in 1962 was an attempt to standardise the language after several flavours had appeared on different machines. 1966 and 1978 saw the first – Fortran 66 and second – Fortran 77 ANSI (American National Standards Institute) versions. The present version, Fortran 90, is the third ANSI standard from 1991 with some minor additions made in 1996 (Fortran 95).

Because of its long standing history Fortran is often dismissed by computing professionals as being old and dated. In fact, the continued development of Fortran 90/95 makes it one of the more modern languages; Fortran continues to be used for programming scientific and mathematical applications in a huge range of contexts, from finance to physics to parallel computer research and to the military.

9.4 The minimum program

Consider the following program, called `nothing.f90`:

```
program nothing

! does nothing

end program nothing
```

This is probably the simplest Fortran 90 program. It does nothing. The first statement simply tells the compiler that a program named `nothing` is to follow. The second statement is a comment (because of the exclamation mark)

and it is ignored by the compiler. The third and last statement informs the compiler that the program terminates at that point. Notice that statements between `program` and `end` are executed in the order that they are written (not strictly true but ok for the moment).

Fortran is case-insensitive: small case or a mixture of upper and lower case is acceptable. So PROGRAM, Program, PROgrAM are all acceptable and mean the same.

Comments are a very useful part of any programming language. Every program, no matter how simple, should contain plenty of comments that explain precisely what the program is doing in plain and simple English.

Chapter 10

Hands on Fortran 90

10.1 Development Tools

To proceed you must be in front of a computer. Go to any of the Windows clusters on campus, for example the cluster in the Herschel Building. Login to a PC entering your username and password (always remember to logout when you finish your session). You clearly need to have a Windows account for this. If you do not have an account, get one from the receptionist at the University's Information Systems and Services (ground floor of the Claremont Tower).

When you are ready, create a folder called `mas2106` to contain all the files related to this module. To create a new folder, you can use `My documents`, either clicking on the icon or accessing it from the *Start* menu. Select *File* from the menu bar on top, then *New*, then *Folder*. This will create a new folder which you can name `mas2106` by selecting *File and folder task* and *Rename this folder*. Make sure to put `mas2106` in your `My documents` folder, which is simply another name for the H: drive which holds your University Windows account. Choose sensible names for folders and files, and avoid gaps between words (which Windows allows you to do, but tends to cause problems to some applications); for example, do not call a file or a folder `october homework`, but rather `october-homework`.

The programs described in these notes are available on my website

`http://www.mas.ncl.ac.uk/ ncfb/`

To download a file, right click on it selecting *Save target as*. Make sure that you save the file in the correct folder, and that the file has the correct suffix, `.f90`, which characterises a Fortran 90 program; include this extension manually in the filename - do not save it as a *Text document* type, otherwise you will get a `.txt` suffix.

10.1.1 Editing, compiling and running programs

Since a Fortran 90 program is first of all a text file, you need an **editor** to write this file. You can use any editor which you may know already, e.g. **Notepad**. However, the implementation of Fortran 90 available on the University's clusters, called **Plato IDE** (for Interactive Development Environment), has a built-in editor already. The **Plato** environment allows you to write a program, compile it and run it, all within the same application, which is very convenient. The way to proceed is as follows.

From *Start*, select *All programs*, then *Programming languages*, then *Fortran 90*, then *Plato IDE*, following the arrows to the right. A window appears. From the menu bar on the top choose *File* and *New*. At this point you can type the text of the file directly onto the screen. For example type the following lines:

```
program hello
! Greet the world gracefully
  write(*,*) 'Hello'
end program hello
```

Select it File and *Save as* to save this file with the name `hello.f90`. The file `hello.f90` is called a **source file**. The extension `.f90` means that the file is recognised as a Fortran 90 program. All Fortran 90 programs must have the `.f90` suffix.

Before running the program you need to **compile** and **link** it; these operations produce the executable binary file which you will actually run. Choose *File* and select *File options*; a window appears to select the target compiler configuration. Choose *FTN90* and *OK*. On the top menu bar click *Project* and then *Compile file*. If there are no errors in the program `hello.f90`, then the compilation is successful and an **object file** called `hello.obj` is created. If there are errors, the compiler will list them, and you will have to edit the file `hello.f90` again to make changes and remove these errors. Many errors are simply typographical mistakes. Never write a program when tired: it takes a very short time to introduce errors (**bugs**) into a program, and a very long time to fix these bugs.

When the compilation is successful you can **link** the file by selecting *Project* and *Build file* from the menu bar. This creates the **executable file** called `hello.exe`. Now you are ready to actually run the program. Choose *Project*, *Execute* and *DOS application*. A new window is created on the computer's screen in which the output of the program appears. In this case the output is simply the word *Hello*.

If you have difficulties writing this or other programs, you can download the file `hello.f90` or other files from my website, as explained above, then open it using **Plato** and proceed with the compilation and execution as explained above.

10.1.2 How to list a program on paper

Sometimes it is easier to fix a program by looking at it on paper or perhaps showing it to a colleague. To print the source file `hello.f90` on one of the cluster's printer, select *File*, *Print* from **Plato**'s menu bar. Another way is to use any generic text editor such as **Notepad**, located under *Start*, *All Programs*, *Accessories and utilities*. **Notepad** can also be used to write the source file in the first place before compiling with **Plato**.

10.2 A simple Fortran 90 program

The general structure of a Fortran program is a list of statements in between the `program` statement and the `end program` statement, as in

```
program name
  ...
end program name
```

where the dots represent all the statements of the program.

When writing a new program, it is often easier to modify an existing program and save it under another name than to write the new program from scratch.

Exercise 1: Using **Plato**, modify the existing program `hello.f90` and write a program called `hello2.f90` which prints on the screen the words *Good Morning Anna* (replacing Anna with your own name).

(The solution to this and other exercises is in the last chapter).

10.3 The `print` and `write` statements

The statement

```
write(*,*) 'Hello'
```

is shorthand for the statement:

```
write (unit=*,fmt=*) 'hello'
```

Each device or file has an associated **unit number**, a numeric tag by which the computer refers to it, and the `unit` argument allows you to specify which one to write to. The asterix in `unit=*` says to write to the **default output unit**, which is the screen. The `fmt` statement (short for `format`) arises because you have the freedom to specify the style in which you write the output, for example how many decimals places to use for numbers. Again, the asterix in `fmt=*` means that you use the **default format**, in which the computer guess at what might look appropriate.

Because output to the screen is used so often (it can even be used to help debug your program), there is a special shorthand:

```
print*,'hello'
```

The `write` statement is more flexible than the `print` statement and can be used to output data to a wider range of devices: screen, files, disc drives, etc. The asterix in the `print` statement means that the default format is used.

What has been said for the `write` statement is also valid for the `read` statement. The **default input unit** however is the keyboard.

10.4 Text and arithmetic expressions

Write, compile and run the following program called `sum1.f90`:

```
program sum1
! Print some examples of text
! and arithmetic expressions
  write(*,*) '15+6'
  write(*,*) 15+6
  write(*,*) '15+6=',15+6
end program sum1
```

Firstly, notice the lines which beginning with exclamation marks. What follows an exclamation mark is a **comment**, something which is added to make the program more legible to the user, but which does not affect the operation of the program at all. It is good practice to put a comment line at the beginning of a program, subroutine, function or other code block to explain what the program is supposed to do in that block. Comments within the program are very useful to explain the purpose of the various statements

to the programmer and serve as a useful double check that you are asking the computer to do what you intended.

Now examine the three `write` statements. The first gives the output $'15 + 6'$ because $15 + 6$ is written within quotes in the program, and is thus considered as text. The second gives the output 21 because $15 + 6$ without quotes is an **arithmetic expression**. The third combines text and arithmetic expressions.

10.5 Data types

In Fortran 90 data can be of various types, including `character`, `integer`, `real` and `complex`. We have already seen that *'hello'* or *'15+6'* are text, that is `character` type. The `integer` type is used to represent whole numbers (positive or negative); e.g. -9 and $16 + 6$ are `integer` expressions. The `real` data type (sometimes called **floating point** type) is used to represent rational and real numbers (although with a finite precision); e.g. 15.0 and 3.141592/2.0 are real expressions. Real numbers are often represented as powers of 10. For example, 2300 can be written as $2.3e3$ which means 2.3×10^3. Similarly 0.0075 can be written as $7.5e-3$, and -0.01 can be written as $-0.1e-1$. Complex numbers are represented as ordered pairs of real numbers, e.g. i is $(0.0, 1.0)$ and $4 - 5i$ is $(4.0, -5.0)$.

The operators $+$, $-$, $*$ and / have the usual meaning of plus, minus, times and divide. Two asterisks indicate exponentiation, e.g. $3 * *2 = 9$. It is important to note that real division corresponds to the usual mathematical division, but integer division produces the integer result obtained by chopping off any fractional part of the mathematical result. For example, the mathematical result of 23/2 is 11.5. In Fortran 90, the arithmetic expression 23.0/2.0 yields 11.5 because it is the ratio of two real numbers, but 23/2, which is the ratio of two integers, yields the integer 11.

Since the integers are a subset of the real numbers and the real numbers are a subset of the complex numbers, conversions are made in evaluating expressions of mixed type. If one number is of type integer and the other is of type real, when combining the two numbers the integer is converted to real; if one number is real and the other is complex, when combining the two numbers the real number is converted to complex. So 23.0/2 is 11.5 because the integer 2 is first converted into the real 2.0 and then a division of two real numbers is performed.

When you write a Fortran 90 program you must always **declare** the type of the variable (e.g. integer, real, complex) which you use. Strictly speaking this is not necessary: by default, Fortran 90 automatically assumes that

variables which begin with letters from i to n are integers and other letters are real.

It is good practice to declare all variables explicitly and switch off the automatic initial-letter data type convention with the statement

```
implicit none
```

at the top of the program. This helps prevent bugs and leaves you more freedom to names you want for your variables.

For example, the following lines declare that x and $hours$ are real and kk and *steps* are integers:

```
real    :: x,hours
integer :: kk,steps
```

10.6 Input from the user

Write, compile and run the following program called `sum3.f90`. It prompts the user to input two real numbers from the keyboard, computes their sum and prints the result on the screen:

```
program sum3
! Compute the sum of two real numbers
  implicit none

  real :: x,y,sum

! Ask the user for the first number
  print *,'Enter first number'
  read *,x

! Ask the user for the second number
  print *,'Enter second number'
  read *,y

! Compute the sum
  sum=x+y
  print *,'The sum is ',sum
end program sum3
```

10.7 Intrinsic functions

Fortran 90 has many built-in functions, for example:

`sqrt`(x) square root
`sin`(x) sine
`cos`(x) cosine
`tan`(x) tangent
`exp`(x) exponential
`log`(x) natural logarithm
`log10`(x) logarithm base 10
`asin`(x) inverse sine
`acos`(x) inverse cosine
`atan`(x) inverse tangent
`abs`(x) absolute value
`int`(x) replace a real number with the corresponding integer
`real`(x) replace an integer number with the corresponding real

Another useful intrinsic function is `mod`(j,k) where j and k are integers, which is the reminder when j is divided by k. For example $mod(9, 5) = 4$.

Exercise 2: Modify the program `sum3.f90` and write, compile and run a new program called `trigo.f90`; the new program should ask for the input x, compute $sin(x) + cos(x)$ and print the result on the screen.

10.8 Explicit formats

If you go beyond the default format `fmt=*`, you can describe precisely how you would like your numbers displayed. Consider integers first. The format `fmt='(i6)'` means that the output is an integer and uses six columns. This format can thus write integers up to 999999 (but be careful that one column may be needed for the minus sign). Consider now real numbers. The format `fmt='(f10.4)'` (where f is the abbreviation of **floating point** number, i.e. real number) means that a total of ten columns is allowed, of which four columns are for decimal places. The output is automatically rounded. For example, if the expression 22.0/7 is written with format `fmt='(f10.4)'`, it appears as 3.1429 with four decimal places. Real numbers can also be represented using the exponential notation. For example,

the format `fmt='(e10.3)'` allows for a total of ten columns and three decimal places. Note that four of the other 10 columns will be used for the exponent, for example 3.429234×10^{21} would be printed as $3.429\mathrm{E}+21$.

The following program called `show.f90` calculates π (using the fact that $\tan(1.0) = \pi/4$), then outputs the value of π in different formats. Write, compile and run this program and notice the how the format is controlled:

```
program show
!
! Calculate an approximation to pi and
! display it in various ways
!
  implicit none

  real :: pi

! Use trig to get a value for pi
  pi=4.0*atan(1.0)

! Play with some different format statements
  write (unit=6,fmt="(f12.2)") pi
  write (unit=6,fmt="(f12.3)") pi
  write (unit=6,fmt="(f12.4)") pi
  write (unit=6,fmt="(f12.5)") pi
  write (unit=6,fmt="(f12.6)") pi
  write (unit=6,fmt="(e12.2)") pi
  write (unit=6,fmt="(e12.3)") pi
  write (unit=6,fmt="(e12.4)") pi
  write (unit=6,fmt="(e12.5)") pi
  write (unit=6,fmt="(e12.6)") pi
end program show
```

In some cases you want to have both numbers and text in the same formatted expression. For example, to write *1,234.56 pounds*, do the following:

```
write(unit=6,fmt='(f7.2,a)') x,' pounds'
```

where the symbol `a` in the statement means **alphanumeric**, that is to say text.

10.9 The do...loop

There are various ways to control the flow of the program; the most important are the **do...loop** and the **if** construct. Consider the **do...loop** first. Its structure is

```
do i=i1,i2,i3
  ...
end do
```

where $i1$, $i2$ and $i3$ are **integers** (positive or negative). The loop initially sets i to the value $i1$, then it increments it in steps of $i3$ until the value $i2$ is reached. If $i3$ is missing in the **do...loop** statement, then $i3$ is set equal to one. For example, consider the following **do...loop**:

```
do i=-4,6,2
   ...
end do
```

The statements inside the **do...loop** are executed six times, with the variable i taking the values $-4, -2, 0, 2, 4$ and 6 respectively. If the integer $i3$ of the **do...loop** is negative then counting is backward, that is to say i decreases (provided $i2$ and $i3$ make sense).

The following program called `loop.f90` uses a **do...loop** to sum the first ten integers. Write, compile and run this program:

```
program loop
! Calculate the sum of the first 10 integers
!
  implicit none
  integer::i,total

! Reset total
  total=0
! Add on each digit in turn
  do i=1,10
     total=total+i
  end do
! Print the result
  print *,total
end program loop
```

The program works in the following way. Initially the variable *total* is set equal to zero. At each iteration *total* is increased by the index i; after the tenth and final loop, the final value of *total* is printed on the screen.

It is important to appreciate that the statement $total = total + i$, which should be read as '**let** total equal total plus one ', is not a mathematical equality but rather and assignment. It means that *total* is overwritten by the value $total + i$. In general, the Fortran 90 statement $a = b$ means that what is in the 'box' (or memory location) labelled by a receives the number which is in the 'box' labelled by b. Many old languages explicitly used the word **let** to begin such statements but this was dropped as it was unnecessary.

10.10 The `if` statement

The `if` statement allows the selection of one of a number of statements during the execution. The simplest `if` statement has the form *if (logical expression) [statement]*. Logical expressions use symbols such as == (equal), / = (not equal), < (less than), <= (less than or equal), > (greater than), >= (greater than or equal), which can be used together with logical operators (`.and.` and `.or.`). An example is:

```
if (x>0.and.y<0) z=0
```

The following program called `if1.f90` (which you should write, compile and run) asks for the input x and computes the function y which is defined as $y = 3x$ if $x < 0$, $y = 0$ for $x = 0$, and $y = 4x$ for $x > 0$:

```
program if1
! Example of an 'if' conditional statement
  implicit none
  real :: x,y
! Go get an x value
  print *, 'Enter x'
  read *,x
! Set y based on some conditions on x
  if (x<0.0) y=3*x
  if (x==0.0) y=0.0
  if (x>0.0) y=4*x
! Display the outcome
  print *,y
end program if1
```

The `if` construct has an alternative block syntax to allow the inclusion of more than one statement for a given condition:

```
if (logical expression) then
  ...
end if
```

One can also specify what to do if a condition fails:

```
if (logical expression) then
  ...
else
  ...
end if
```

Exercise 3: Write a program called `irs.f90` which asks for your income and computes the tax which you must pay; the tax is 15 percent for income up to 17850 pounds, and 28 percent above 17850 pounds.

Exercise 4: Write a program called `divide.f90` which asks for two integers and determines if one number can be divided by the other. (Hint: use the intrinsic function `mod`).

10.11 Arrays

It is possible that data consist of ordered sets of scalars (`integer`, `real` or `complex`) which can be referred to collectively by a single name. This new object is called an **array**. For example, vectors and matrices can be represented as arrays. Arrays must be **declared** at the beginning of the program (though the specification of their exact size can be postponed till later in the program if needed!). For example, suppose that the integer array x consists of three components $x(1)$, $x(2)$ and $x(3)$; then, at the beginning of the program (after the `implicit none` statement), you must put the statement

```
integer, dimension(3) :: x
```

The actual numerical values of $x(1)$, $x(2)$ and $x(3)$ can be assigned one by one in the actual program, as in

```
x(1)= 9
x(2)=-1
x(3)= 3
```

or by a single statement, as in

```
x=(/9,-1,3/)
```

If the array x is real, it is declared by

```
real, dimension(3) :: x
```

The index i which denotes the components of the array x need not necessarily start from $i = 1$. For example, consider the following declarations:

```
real, dimension(0:2) :: x
real, dimension(-4:3) :: y
```

for the arrays with elements $x(0)$, $x(1)$, $x(2)$, and $y(-4)$, $y(-3)$, $y(-2)$, $y(-1)$, $y(0)$, $y(1)$, $y(2)$, $y(3)$.

In the following example, x could represent a matrix of 6 rows and 3 columns and y could represent a tensor of dimension $3 \times 4 \times 8$:

```
real, dimension(6,3) :: x
real, dimension(3,4,8) :: y
```

Write, compile and run the following program `array1.f90` which defines two arrays of dimension 3 and sums them, component by component, using a **do...loop**:

```
program array1
! Sum two given arrays
  implicit none
  integer :: i
  real, dimension (3):: x,y,z
! Initialize the array elements
  x(1)= 1.1
  x(2)=-0.1
  x(3)= 4.5
  y(1)=-0.1
  y(2)=-0.9
  y(3)= 3.5
! Calculate the sum, element by element
  do i=1,3
     z(i)=x(i)+y(i)
  enddo
  print *,z
end program array1
```

Using explicit do...loops is the most flexible way of operating with arrays but is not always the simplest. It is often possible to handle arrays via a single statement. For example, in the above program `array1.f90`, in place of

```
do i=1,3
  z(i)=x(i)+y(i)
enddo
```

you can write simply

```
z=x+y
```

10.12 Output to file

Suppose that you want to write the output of the calculation performed by you program into a file rather than to the screen. Pick a number as **output unit number**, associate this number with a file using the `open` statement, and `write` the program's output to this unit. The unit numbers should be in the range from 1 to 99, and should not be 5 (reserved for keyboard input) or 6 (reserved for screen output).

Write, compile and run the following program `out1.f90` which writes the text "Goal !" in a file called `out.dat`.

```
program out1
! Write some text to a file
  implicit none

! Open a file (by default for 'formatted' output)
  open(unit=7,file='out.dat')
  write(unit=7,fmt=*) 'Goal !'
  close(unit=7)
end program out1
```

After running the program, look at the file `out.dat` using **Notepad** to check that it indeed contains the word 'Goal !'.

Exercise 5: Write a program called `out2.f90` which outputs greek pi in exponential notation (with two decimals) to a file called `out.dat`.

10.13 Input from file

In the same way in which you use the `write` statement to write to a file, you can use the `read` statement to read data from a given file, rather than entering data from the keyboard.

10.14 Subroutines

We have already met intrinsic functions, like $sin(x)$. The idea of a `subroutine` generalises that of intrinsic function. A subroutine is a block of statements which is totally independent of the main program and is linked to it only via some declared input and output parameters. A subroutine is therefore a kind of 'black box'. Usually a complete program consists of a main program and some subroutines, as in

```
program name1
  ...
end program

  subroutine name2
    ...
  end subroutine

  subroutine name3
    ...
  end subroutine
```

The subroutines are **called** by the main program, to which they return their output as a function of the input which they have received. A subroutine can call another subroutine. The use of subroutines helps keeping the main program as slim and simple as possible.

To understand how subroutines work in practice, write, compile and run the following program called `bessel.f90`. The program computes the values of the spherical Bessel function of order 1, defined as $y(x) = sin(x)/x^2 - cos(x)/x$, on a set of equally spaced points 0.0, 0.1, 0.2, $\cdots$, 10.0. The values are printed on the screen and saved to a file called `out.dat`.

```
!*********************************************
program bessel
! Compute the Bessel function j1(x) at the points
!      x=0, 0.1, 0.2 ... 1.0
  implicit none
  integer :: i
  real :: x,y,dx
! Open the output file and attache it to unit number 7
  open(unit=7,file="out.dat")
  dx=0.1                            ! x increment
! For each of the 101 points
!   NB:  0 to 100 inclusive = 101 points
  do i=0,100
! Calculate the x-value of the ith point
    x=i*dx
! Calculate y from x using the j1 routine
    call j1(x,y)
! Display the x,y values for information
    print *,x,y
! Write the x and y values in exponential format,
!  together on one line
    write(unit=7,fmt="(2e12.4)") x,y
  end do
! Finished writing to the output file
  close(unit=7)
end program bessel
!*********************************************
  subroutine j1(x,y)
! Calculate Spherical Bessel function of order 1
!  for parameter x and return the result in the
!  second parameter y
    implicit none
    real :: x,y
    if (x==0.0) then
      y=0.0
    else
      y=sin(x)/x**2 -cos(x)/x
    end if
  end subroutine j1
!*********************************************
```

In the above program the subroutine has one input, (x), and one output, (y). Note that what matters is **the order of the inputs and outputs**, not the actual names of the variables used. Thus in the main program we call the subroutine *j1* with two real arguments x and y:

```
  call j1(x,y)
```

but in the subroutine at the end of the file we need not use x and y for the names of the two arguments; we can use a and b, provided the they are real numbers. What matters is the order: when we call the subroutine we establish a correspondence between x and a and between y and b:

```
  subroutine j1(a,b)
! Calculate Spherical Bessel function of order 1
!  for parameter a and return the result in the
!  second parameter b
    implicit none
    real :: a,b

    if (a==0.0) then
      b=0.0
    else
      b=sin(a)/a**2 -cos(a)/a
    end if
  end subroutine j1
!*********************************************
```

10.15 Gnuplot

To plot the results of your numerical calculations you need a computer graphics package. You can use either **Maple** if you are already familiar with it, or **Gnuplot**. You will find **Maple** and **Gnuplot** on the University Windows system. The advantage of **Gnuplot** is that it is free, so you can install it on your home computer at no extra cost.

To use **Gnuplot** to plot the data saved in the file `out.dat` by the previous program, do the following. From *Start*, select *Programs*, *Graphics software* and *Gnuplot for Windows*. A window appears. Click *ChDir* on the menu bar on the top and enter `mas2106` to go into the directory `mas2106` where our data file `out.dat` is. At Gnuplot's prompt, type:

```
plot 'out.dat'
```

The plot will appear on the screen. If you type

```
plot 'out.dat' w l
```

where *w l* is the abbreviation of *with lines*, then the data points are hidden and joined by a continuous line; similarly,

```
plot 'out.dat' w p
```

means to plot *with points* and

```
plot 'out.dat' w lp
```

means to plot *with lines and points*. You can also control the region which is plotted; for example, by writing

```
plot [0:12][-0.5:0.5} 'out.dat' w l
```

your data are plotted in the region $0 \leq x \leq 12$, $-0.5 \leq y \leq 0.5$. Finally, to exit Gnuplot type

```
quit
```

Chapter 11

Numerical methods

Nonlinear equations are usually very difficult to solve analytically, and we have to use numerical methods. The aim of this chapter is to introduce the simplest method: Euler's method.

11.1 Discretization of the equation

Consider the differential equation for the function $x(t)$:

$$\frac{dx}{dt} = f(t, x), \tag{11.1}$$

where t is time and the right-hand-side function f is assigned. We want to find the solution $x(T)$ at the time $t = T$, given the initial condition $x(0) = x_0$ at time $t = 0$.

Our numerical approach is based on discretizing the time interval $0 \leq t \leq T$ into a large but finite number of sub-intervals defined by the points $t_0 = 0$, $t_1 = h$, $t_2 = 2h$, $t_3 = 3h$, $\cdots$, $t_{N-1} = (N-1)h$, $t_N = Nh = T$, where $h = T/N \ll 1$ is the **time step**. We call $x_n = x(t_n)$ and $f_n = f(t_n, x_n)$.

The idea is to derive a **recursion formula** so that, starting from the given initial condition x_0, we can calculate x_1 from x_0, then x_2 from x_1, then x_3 from x_2, and so on, until we have the final value x_N which is the **numerical approximate solution** to the equation at the final time $t = T$. Clearly, the smaller the time step h is, the more accurate our numerical solution will be.

11.2 Euler's recursion formula

A simple recursion formula to move from x_n to x_{n+1} can be constructed in the following way. Integrate $dx/dt = f$ between time t_n and time t_{n+1}:

$$\int_{t_n}^{t_{n+1}} \frac{dx}{dt} dt = \int_{t_n}^{t_{n+1}} f(t, x) dt, \tag{11.2}$$

The definite integral at the left-hand-side can be evaluated exactly and we have

$$x_{n+1} - x_n = \int_{t_n}^{t_{n+1}} f(t, x) dt, \tag{11.3}$$

The right-hand-side is the area under the curve $f(t, x(t))$ between t_n and t_{n+1}. Since $t_{n+1} - t_n = h \ll 1$, this area is approximately equal to the area of the rectangle of base h and height $f_n = f(t_n, x_n)$ (see Fig. 11.1), hence

$$x_{n+1} - x_n \approx h f_n. \tag{11.4}$$

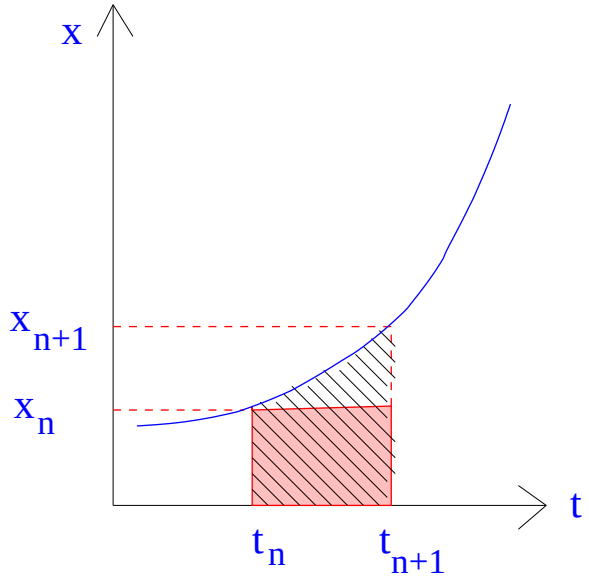


Figure 11.1: The pink area is the approximation to the true area, which is indicated by the diagonal lines.

The resulting recursion formula is called **Euler's method**:

$$x_{n+1} = x_n + h f_n. \tag{11.5}$$

Example 1:
Write a program to solve

$$\frac{dx}{dt} = x - tx^2,$$

with initial condition $x(0) = 1$ using Euler's method and time step $h = 0.01$. Plot the solution for $0 < t < 10$ using Gnuplot.

Solution: To achieve the aim, we do the following:

1. The first task consists in finding the necessary recursion formula. In this case application of Eq. 11.5 yields

   $$x_{n+1} = x_n + h(x_n - t_n x_n^2).$$

   Starting from the initial condition $t_0 = 0$, $x_0 = 1$, the successive values of t_n are: $t_0 = 0$, $t_1 = h$, $t_2 = 2h$ etc; the corresponding values of x_n are: $x_0 = 1.0$, $x_1 = u_0 + h(x_0 - t_0 x_0^2)$, $x_2 = u_1 + h(x_1 - t_1 x_1^2)$, $x_3 = u_2 + h(x_2 - t_2 x_2^2)$, etc.

2. The second task consists in writing, compiling and testing the following Fortran program *euler2.f90*:

```
!************************************************************
program euler2
! Aim: to solve dx/dt=x-t*x*x
implicit none
integer :: step,steps
real :: t,x,told,xold,fold,h,tfinal

! Output file
open(unit=7,file='out.dat')
! Get parameters from the screen
print*,'Enter initial time t'
read*,t
print*,'Enter initial value x'
read*,x
print*,'Enter final time t'
read*,tfinal
print*,'Enter number of time steps'
```

```
read*,steps
h=tfinal/steps   ! Time step

! Time loop
do step=1,steps
! Update old values
    told=t
    xold=x
! Evaluate RHS of equation at the old time
    call get_f(told,xold,fold)
! Evolve t and x by one timestep
    t=h*step
    x=xold+h*fold
! Print values on the screen and on the output file
    print*,t,x
    write(unit=7,fmt="(2e12.4)") t,x
enddo ! Close time loop

! Finis
close(unit=7)
end program euler2

!************************************************
subroutine get_f(t,x,f)
! Aim: to get RHS of equation
implicit none
real :: x,t,f
f=x-t*x*x
end subroutine get_f
!************************************************
```

Note the following features:

(a) Only two values of $x(t)$ are actually needed for time stepping: the "new" value x_{n+1} and the "old" value x_n. Thus the program has only two variables, x and *xold*. Think of x and *xold* as "boxes" which contain numbers. At the beginning the number $x_0 = 1$ is put into the box x. Then the time loop starts. At the first iteration ($step = 1$) what is in the box x (the number x_0), is put into the box *xold*; then x_1 is calculated and put into the box x, overwriting the previous content. At the second iteration ($step =$

2) the number contained in the box x, which is now x_1, is put into the box *xold* (again, overwriting the previous content of the box); then x_2 is calculated and put into the box x (overwriting the previous content). At each iteration the values of t and x are written in the file *out.dat*. Each time step produces one row with two numbers, t and x.

(b) The right-hand-side function f in defined in a subroutine rather than in the main program. This simplifies the program and makes it easier to change it to solve another differential equation.

(c) The initial values of t and x, the final time and the number of time steps are input from the screen for convenience. The time step h is defined as the final time divided by the number of time steps.

3. The third task consists in actually running the program. We choose initial condition $t = 0$, $x = 1$, input the final time $t = 10$, and the total number of steps, 1000; this means that the time step is $h = 10/1000 = 0.01$; the file *out.dat* will contain 1000 rows. When the program has run, to make sure that the output is OK, we use an editor (eg *Notepad*) to check that the file *out.dat* contains t and x, arranged in 2 columns and 1000 rows.

4. The fourth task consists in plotting the data contained in the file *out.dat* using the plotting program *Gnuplot*. At Gnuplot's prompt we can simply type

```
plot 'out.dat'
```

The graph will appear on the screen. If we prefer to draw a line rather than to mark the points, we type

```
plot 'out.dat' w l
```

where *w l* means *with line*. A better way to proceed is to use *Notepad* to create a small file called *gnu* to be loaded into Gnuplot; an example is given below:

```
#-------------------------------------------------------
# program gnu
# to run this program type
#      gnuplot
# then at the prompt of gnuplot type
#      load "gnu-euler2"
# Use the symbol # to blank out lines

# To avoid having a legenda which tells which files are plotted
#set nokey

# to make postscript file
 set terminal postscript
 set output "euler2.ps"

#to make a title
#set title "x vs t"

# to label the axes
 set xlabel "t"
 set ylabel "x"

plot "out.dat"  with l
#-------------------------------------------------------
```

(note that the symbol hash denotes a comment for **Gnuplot**, in the same way as we used exclamation marks in Fortran 90). This way to proceed is useful is we want to use many plotting commands. In the Gnuplot program above, the picture is saved in a postscript file to be printed later (see Fig. 11.2), rather than put on the screen.

11.3 System of equations

It is very easy to generalise Euler's method and solve two or more coupled equations. Consider the system of equations for $x(t)$ and $y(t)$:

$$\frac{dx}{dt} = f(x, y),$$
$$\frac{dy}{dt} = g(x, y),$$

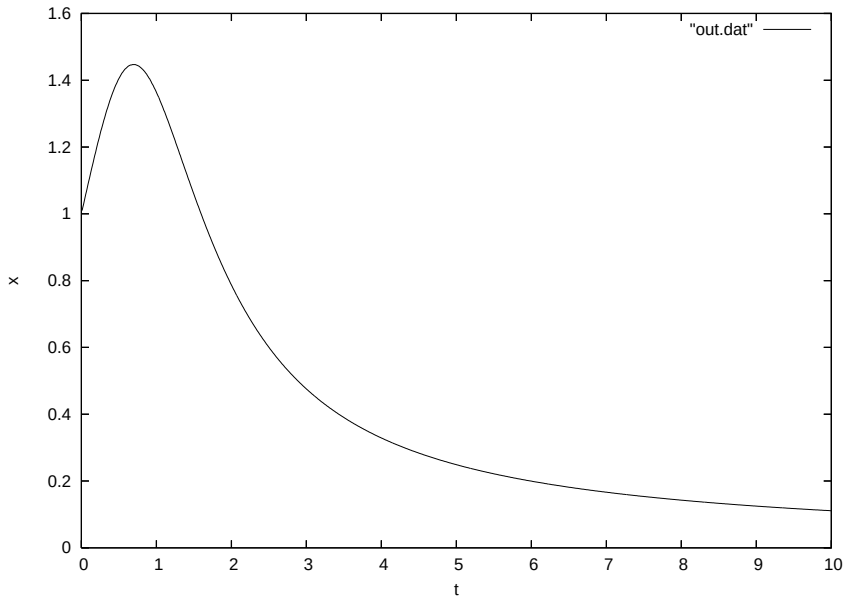


Figure 11.2: Plot of x vs t produced by the Gnuplot program *gnu*.

where the functions f and g are prescribed and the initial conditions are $x(0) = x_0$ and $y(0) = y_0$. Let $t_n = nh$ $(n = 0, 1, 2, \cdots)$ where h is the time step. Call x_n, y_n, f_n and g_n the values of xm y, f and g at each time step t_n. At the n^{th} step we compute *together*

$$x_{n+1} = x_n + hf_n, \qquad y_{n+1} = y_n + hg_n,$$

Example 2:
Compute the solution of the following system of equations

$$\frac{dx}{dt} = x - y - x(x^2 + y^2), \tag{11.6}$$
$$\frac{dy}{dt} = x + y - y(x^2 + y^2), \tag{11.7}$$

with initial condition $x(0) = y(0) = 0.1$ at $t = 0$, using time step $h = 0.01$, in the time interval $0 < t < 10$. Make two graphs of the solution: the first graph to show x and y versus t, the second graph to show y versus x, State the nature of the solution.

Solution: We modify the existing program *euler2.f90* and write the following new program *euler3.f90*. Note that the right hand sides of the equations are given in a single subroutine for simplicity.

```
!************************************************************
program euler3
! Aim: To solve
!      dx/dt=f=x-y-x*(x*x-y*y)
!      dy/dt=g=x+y-y*(x*x+y*y)
implicit none

integer :: n,nn
real :: t,told,x,xold,y,yold,fold,gold,h,tt

! Open a file in which to save the results
open(unit=7,file='out.dat')

! Initial condition
t=0.0
x=0.1
y=0.1

! Input parameters
h=0.01     ! Time step
nn=10000    ! Number of time steps

! Time Loop
do n=1,nn
!  Update old values
    told=t
    xold=x
    yold=y

!  Evaluate the RHS of both equations
    call get_rhs(told,xold,yold,fold,gold)

!  Evolve t,x and y by one timestep
    t=h*n
    x=xold+h*fold
    y=yold+h*gold

!  Output to file
    write(unit=7,fmt="(3e12.4)") t,x,y
enddo ! Close time loop
```

```
! Tidy up
close(unit=7)
end program euler3

!*******************************************
subroutine get_rhs(t,x,y,f,g)
! Calculate the RHS of both equations
implicit none
real :: t,x,y,f,g
f=x-y-x*(x*x+y*y)
g=x+y-y*(x*x+y*y)
end subroutine get_rhs
!*******************************************
```

The output file *out.dat* contains the solution in the form of three columns t, x, y. The data can be plotted using *Gnuplot*. Since there are three variables, x, y, z, we can plot them in various ways:

To plot x (the second column) versus t (the first column) we use the command *using 1:2*, or the abbreviation *u 1:2*, which means using the first and second column:

```
plot 'out.dat' u 1:2 w l
```

where *w l* means *with line* (to draw a line between the data rather than marking the points).

To plot y (the third column) versus t (the first column) we use the command *u 1:3*.

To plot y (the third column) versus x (the second column) we use the command *u 2:3*.

Finally, we can also make a 3-dim graph: type

```
splot 'out.dat' w l
```

We can grab the picture with the prompt and rotate it to look at it from any direction.

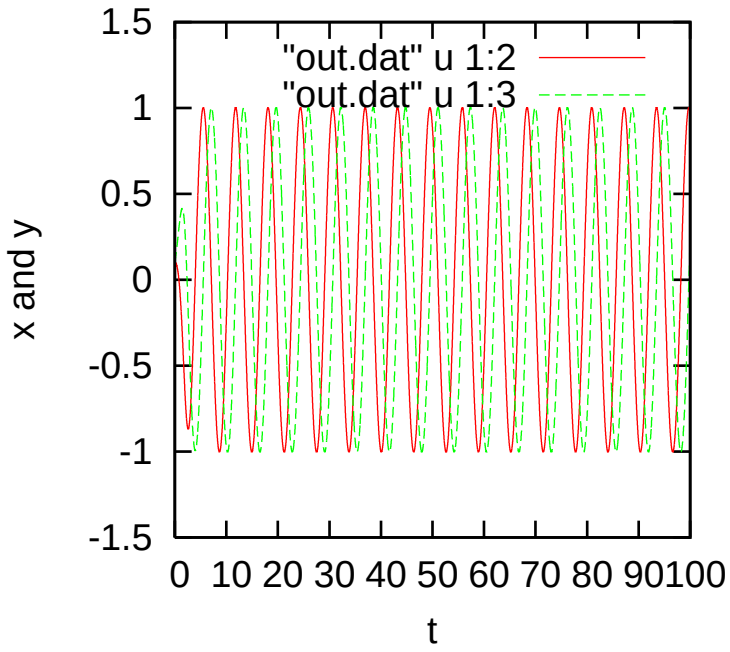


Figure 11.3: Graph of the solution x vs t and y vs t of Eq. 11.7 computed by the Fortran program *euler3.f90* and plotted using the Gnuplot program *gnu-euler3a*.

As said before, it is often convenient to collect the Gnuplot commands in a small file. The following files *gnu-euler3a* and *gnu-euler3b* have been used to make Fig. 11.3 and Fig. 11.4 respectively. The last figure shows that the nature of the solution is a limit cycle. Look carefully at *gnu-euler3a* to see how to plot two curves in the same graph.

```
#-----------------------------------------------------------
# program gnu-euler3a
# to run this program type
#      gnuplot
# then at the prompt of gnuplot type
#      load "gnu-euler3a"
# Use the symbol # to blank out lines

# To avoid having a legenda which tells which files are plotted
#set nokey

# to make postscript file
```

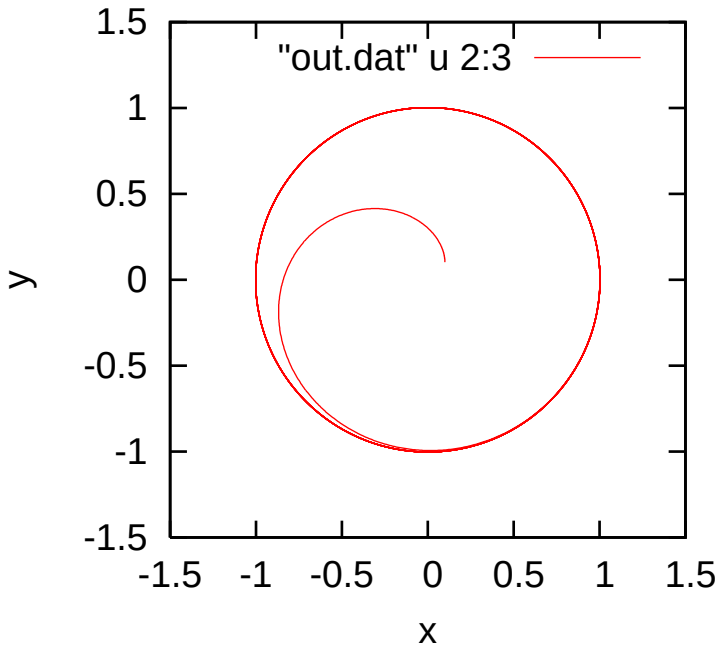


Figure 11.4: Graph of the solution x vs y of Eq. 11.7 computed by the Fortran program *euler3.f90* and plotted using the Gnuplot program *gnu-euler3b*.

```
 set terminal postscript color
 set output "euler3a.ps"

 set size square
 set size 0.5,0.5

#to make a title
#set title "x and y vs t"

# to label the axes
 set xlabel "t"
 set ylabel "x and y"

plot "out.dat" u 1:2  with l,\
     "out.dat" u 1:3 w l
#-------------------------------------------------------

#-------------------------------------------------------
# program gnu-euler3b
# to run this program type
```

```
#      gnuplot
# then at the prompt of gnuplot type
#      load "gnu-euler3b"
# Use the symbol # to blank out lines

# To avoid having a legenda which tells which files are plotted
set nokey

# to make postscript file
 set terminal postscript color
 set output "euler3b.ps"

 set size square
 set size 0.5,0.5

#to make a title
#set title "y vs x"

# to label the axes
 set xlabel "x"
 set ylabel "y"

plot "out.dat" u 2:3  with l
#-------------------------------------------------------------
```

11.4 Accuracy

Consider the equation $x' = f(x)$. From Taylor's Theorem, the value of the function x at the point $t_{n+1} = t_n + h$ is:

$$x(t_n + h) = x(t_n) + x'(t_n)h + \frac{1}{2!}x''(\xi)h^2,$$

where $t_n \leq \xi \leq t_n + h$. Since x satisfies the differential equation $x' = f$ where $x' = dx/dt$, then

$$x(t_n + h) = x(t_n) + f(x_n, t_n)h + \frac{1}{2!}x''(\xi)h^2,$$

which is

$$x_{n+1} = x_n + f_n h + \frac{1}{2!}x''(\xi)h^2 = x_n + f_n h + \mathcal{O}(h^2),$$

Compare the above result against the Euler method:

$$x_{n+1} = x_n + f_n h,$$

It is apparent that the Euler method has a *local error* (the measure of the accuracy of a single step) which is proportional to h^2.

In general, a recursion formula to solve a differential equation is said to be *of order* k if the local error is $\mathrm{O}(h^{k+1})$. We conclude that the order of the Euler method is one.

Chapter 12

Solutions

1. **Solution**

```
program hello2
! Greet the codemaster
  print*,'Good morning Anna'
end program
```

2. **Solution**

```
program trigo
!
! Input x and compute sin(x)+cos(x)
!
  implicit none

  real::x,y

! Request an x value
  print *,'Enter x'
  read *,x

! Calculate the function and display the result
  y=sin(x)+cos(x)
  print *,y
end program trigo
```

3. **Solution**

```
program irs
!
! Calculate income tax payable for a given income
! with the tax brackets:
!      0 to 17850      at 15%
!      17850 to 43150  at 28%
!
  implicit none

  real :: income,tax

! Ask the user their income
  print*,'Enter income: '
  read*,income

! Calculate tax or suggest further action
  if (income<0.0) then
     print*,'income cannot be negative '
  else if (income==0.0) then
     tax=0.0
     print *,' no income, no tax'
  else if (income>0.0.and.income<=17850) then
     tax=0.15*income
     print *,'low tax bracket: tax =',tax
  else if (income>17850.and.income<=43150) then
     tax=0.15*income+0.28*(income-17850)
     print *,'high tax bracket: tax =',tax
  else if (income>43150) then
     print *,' hire an accountant'
  end if
end program irs
```

4. **Solution**

```
program divide
!
! Test if a is number is divisible by another number
!
  implicit none
```

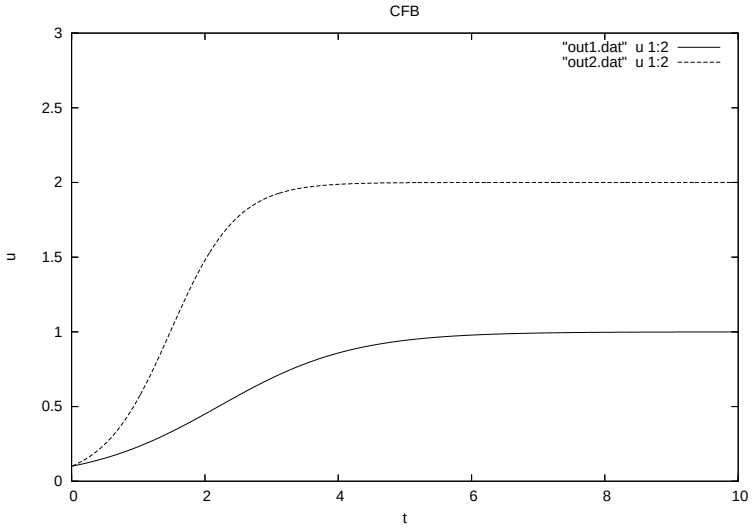
```
  integer :: n,m,k

  print*,' Enter first number'
  read*,n

  print*,' Enter second number'
  read*,m

! Calculate the remainder on division (n modulo m)
  k=mod(n,m)

! Test for divisibility
  if (k==0) then
      print*,n,' can be divided by ',m
  end if
end program divide
```

5. **Solution**

```
program out2
! Calculate something and write the number to a file
  implicit none

  real :: x

  open(unit=7,file='out2.dat')

! Calculate and output it to unit 7
  x=4.0*atan(1.0)
  write(unit=7,fmt="(e10.2)") x

  close(unit=7)
end program out2
~
```

6. **Solution**

The program *euler4.f9* is listed below. It was run with $u(0) = 0.1$, $tt = 10$, $nn = 1000$, for two values of α: $\alpha = 1$ and $\alpha = 2$. The solutions corresponding to these two values of α are plotted below.




```
program euler4
!
! To find the solution of du/dt=alpha*u-u*u
!  at given final time final, given initial
!   condition u(0) and given number of time
!   steps nn
!
  implicit none

  integer :: n,nn
  real :: t,u,told,uold,fold,h,tt,alpha

! Open a file to which we can write our results
  open(unit=7,file='out.dat')

! Get run parameters from the user
  print*,'Enter initial value u(0)'
  read*,u
  print*,'Enter final time t'
  read*,tt
  print*,'Enter number of time steps nn'
  read*,nn
  print*,'Enter alpha'
  read*,alpha
```

```
  h=tt/nn
  t=0.0
! Time loop
  do n=1,nn
! Update 'old' values from 'last' step
    told=t
    uold=u
! Evaluate RHS of differential equation at told
    call get_f(alpha,uold,fold)
! Evolve t and u by one timestep
    t=h*n
    u=uold+h*fold
! Print some debug information and write the
! result to the file for plotting
    print*,t,u
    write(unit=7,fmt="(2e12.4)") t,u
  enddo

! Finished with the output file
  close(unit=7)
end program

!***********************************************
  subroutine get_f(alphau,f)
!
! Calculate right handside (f) of differential
! equation at a given time t and u
!
    implicit none

    real :: u,alpha,f

    f=alpha*u-u*u
  end subroutine
!***********************************************
```

Bibliography

[1] W.S. Brainard, C.J. Goldberg, and J.C. Adams. *Programmer's Guide to Fortran 90*. Springer, 1995.

[2] P.G. Drazin. *Nonlinear Systems*. Oxford University Press, 1983.

[3] J. Gleich. *Chaos: making a new science*. Minerva, 1997.

[4] S.H. Strogatz. *Nonlinear Dynamics and Chaos*. Westview, 1994.