Biological Cybernetics: Non-Linear systems (http://www.reading.ac.uk/~shshawin/teaching/biocybernetics)

1 Outline

- Definition of linear and non-linear systems. System states.
- Ordinary differential equations and state systems
- Numerial integration
- Linearisation around an operating point
- Phase space, stable nodes, unstable nodes, saddle points
- Pendulum equations
- Optimization
- Machine intelligence
- Classification and confusion
- Naive Bayes, and Random tree classifiers
- Neural networks.
- Learning and adaptive systems (back propagation)

There may be additional information on the websites

(http://www.personal.reading.ac.uk/~shshawin/teaching/biocybernetics/01intro.html)

1.1 Assessment

Assessment is via examination and two assignment (see blackboard)

- Ying Zheng assignment
  - Opens 18th December 2020
  - Due 15th January 2021 (noon)
- William Harwin assignment
  - opens 19th Feb 2021
  - due 19th March 2021 (noon)

1.2 Interactive and drop-in sessions

Interactive and drop-in sessions will be held on Blackboard online interactive sessions (although we have the option to move to Microsoft Teams)
Exercises will be set for the interactive sessions,
Tutorials

- Matlab: Numerical integration, time domain and phase plane plots
- Matlab: Optimization
- Matlab: Machine learning and classification
- Matlab: Learning by back propogation and neural networks

1.3 Books:

The following books are some of the references used in the course

1. Steven H Strogatz, "Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering", 2018 UR Library call: **online https://www.reading.ac.uk/library/**

2. Ethem Alpaydin, "Introduction to machine learning", 2014 UR Library call: **online XX(1737603.1)**
3. Peter Flach, "Machine learning: the art and science of algorithms that make sense of data", 2012 UR Library call: **UR Call 006.31-FLA**
4. Gilbert Strang, "Computational science and engineering", 2007 UR Library call: **518-STR** *General maths book, also try Stroud and Cryer.*
5. K.A. Stroud and Dexter J. Booth., "Engineering mathematics", 2013 UR Library call: **LARGE–F 510.2462-STR**
6. Colin Walker Cryer, "A math primer for engineers", 2014 UR Library call: **XX(1764931.1)** *available online or to download. Appears to be good for background maths.*
7. Cleve B. Moler, "Numerical Computing with MATLAB", 2004 https://uk.mathworks.com/moler/index_ncm.html
8. William Gosling, "Helmsmen and Heroes: Control theory as a key to past and future", 1995 *Good background and an easy read.*
9. W. Ross Ashby, "An introduction to cybernetics", 1956 http://pespmc1.vub.ac.be/ASHBBOOK.html UR Library call: **003.5-ASH** *Mainly interesting because of the homeostat. Ashby's journals are also online (check Wikipedia external links).*
10. Norbert Wiener, "Cybernetics: or, Control and communication in the animal and the machine", 1961 UR Library call: **003.5-WIE** *First published 1948. Seen as the defining book on Cybernetics.*
11. J.R. Leigh, "Control Theory: a guided tour", 1992 UR Library call: **629.8312 LEI** *A good book for an alternate view of general control systems despite the numerous mistakes, (treat it as spot the typo).*
12. Martin Hargreaves, "Engineering Systems: Modelling and Control", 1996 UR Library call: **620.00151-HAR**
13. R Mitchell, W Browne, W. Harwin and K Warwick, "Cybernetics and its application", 2008 http://www.pearsoncustom.com UR Library call: **FOLIO–003.5-CYB** *possibly helpful but designed for a different course.*

1.4 Assumed background knowledge

- algebra, laws of association, commutation and distribution
- matrices addition, multiplication, inverse,
- differentiation, integration and Laplace transforms

2 What is Cybernetics

Cybernetics: The scientific study of control and communication in the animal and the machine.

Norbert Wiener 1948

... which I call Cybernétique, from the word $\kappa\upsilon\beta\epsilon\rho\nu\epsilon\tau\iota\chi\eta$ which at first (taken in a limited sense) is the art of governing a vessel but (even among the Greeks) extends to the art of governing in general.[1] André-Marie Ampère [2] 1838

The ancient Greek word kybernetikos ("good at steering"), refers to the art of the helmsman.

Latin word guberno/gubernãre that is the root of words for government comes from the same Greek word that gives rise to cybernetics.

[1]André-Marie Ampère (1838). *Essai sur la philosophie des sciences ou exposition analytique d'une classification naturelle de toute les connaissance humanines.* Paris, pp. 140–142. URL: https://archive.org/details/essaisurlaphilos00amp.

[2]Ampere referred also to the science of "people-to-people interactions" as cybernetics

2.1 The embodied brain (sense model plan act)

The posession of a brain allows humans, animals and (to some extent) robots to have a meaningful interaction in a complex environment. Intelligence may be observed from the way we react in this environment but (as shown in figure below), sense our environment and use this information to update internal models we have of the world. We react to this information via our understanding of the behaviour of our internal models and operate through muscles to enable us to alter our enviromnent to our benefit.

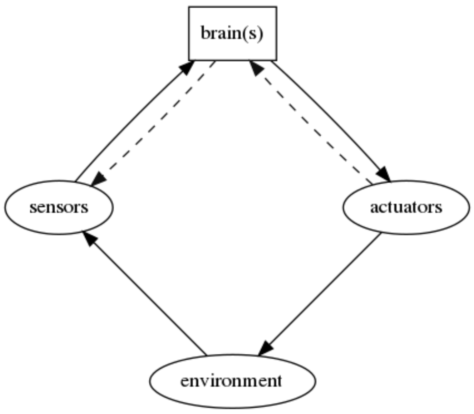


Figure 1: Embodied brain. Intelligence exists because of the environment. The brain can belong to a human, animal or robot

- Measurement
- Actuation (speakers and monitors are actuators in this context). Our muscles us to control our sensors (turn our heads towards sounds, glance at things that move past), as well as enable us to walk and move things.
- Communication. Humans and animals demonstrate communication that range from chemical messangers coveyed in blood etc, through to sounds and language.
- Control/intelligence. The brain is able to patch up information from poor sensors to gain some robust models. Think about how an experienced golfer manages to get the golfball onto the green.

Other relevant concepts

- Homeostasis and regulation (Covered by prof Zheng)
- Affordance (Pfeifer)
- Internal modelling and prediction (Wolpert)
- Motor babbling (Lipson)[3][4][5][6]
- Machine intelligence, Braitenberg vehicles and cognitive robotics

[3] Viktor Zykov, Josh Bongard, and Hod Lipson (2004). "Evolving dynamic gaits on a physical robot". In: *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*. vol. 4.

[4] JC Bongard, Victor Zykov, and Hod Lipson (2006a). "Automated synthesis of body schema using multiple sensor modalities". In: *Proc. of the Int. Conf. on the Simulation and Synthesis of Living Systems (ALIFEX)*. URL: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/LBP065.pdf.

[5] Josh Bongard, Victor Zykov, and Hod Lipson (2006b). "Resilient machines through continuous self-modeling". In: *Science* 314.5802, pp. 1118–1121. URL: http://sclab.yonsei.ac.kr/courses/11cognitive/11cognitive.files/referred/10.pdf.

[6] Josh Bongard and Hod Lipson (2005). "Automatic Synthesis of Multiple Internal Models Through Active Exploration". In: *American Association for Artificial Intelligence, Fall Symposiumon Reactive to Anticipatory Cognitive Embodied Systems*. URL: http://www.aaai.org/Papers/Symposia/Fall/2005/FS-05-05/FS05-05-004.pdf.

Highly recommended Ted Talk (https://www.ted.com/talks/daniel_wolpert_the_real_reason_for_brains) Daniel Wolpert. The real reason for brains. March 2014

2.1.1 Examples

- Izhikevich neurons
- Biomechanics of animals and humans
- Drug diffusion e.g. Pharmacokinetic two-compartment model
- Mass-spring-damper
- Inductor-capacitor-resistor
- control systems, robots etc
- weather, environmental pollutant diffusion
- A ship: We would like to control its direction by manipulating the rudder angle.
- A telephone channel: We would like to construct a receiver with good reproduction of the transmitted signal.
- A time series of data (e.g. sales): We would like to predict the future values.

2.2 Systems and models

A system is an object that we would like to study, control and affect the behavior.

A model is the knowledge of the properties of a system. It is necessary to have a model of the system in order to solve problems such as control, signal processing, system design. Sometimes the aim of modelling is to aid in design. Sometimes a simple model is useful for explaining certain phenomena and reality.

A model may be given in any one of the following forms:

1. Mental, intuitive or verbal models: Knowledge of the system's behavior is summarized in a person's mind.
2. Graphs and tables: e.g. the step response, Bode plot (sometimes called non-parametric)
3. Mathematical models: Here we confine this class of model to differential and difference equations.

3 Types of system

3.1 Definition of continuous time

A value $y(t)$ exists at all points values of t (or more usually all points in $t \geq 0$)

This should also be true of all derrivatives of y

- Examples, sin wave, step, dirac's delta

3.2 Continuous plant

Formed as output y(t) responding to input u(t) to a system g(t)

$$y(t) = \int_{-\infty}^{\infty} u(\tau)g(t-\tau)d\tau$$

This is convolution and can be considered graphically as follows[7]

1. Flip one of the two functions, say, $g(\tau)$, to get $g(-\tau)$;

[7] Ruye Wang (2011). *e101 Fall 2011*. Lecture notes, Harvey Mudd College. URL: http://fourier.eng.hmc.edu/e101/.

2. Slide this flipped function $g(t-\tau)$ along the τ axis as t goes from $-\infty$ to ∞;
3. Find the area of overlap between $u(\tau)$ and $g(t-\tau)$ at each moment t. This area corresponds to the convolution integral for y(t).

Laplace transformed versions

$$Y(s) = G(s)U(s)$$

Frequency domain and frequency response (Ying Zheng)

3.3 Sampled data systems/discrete-time system

A value $y(n)$ exists only at specific times, that is at all positive integer values of n. It may be convenient to occasionally include 0 or negative integers in n.

Tend to be defined in terms of a delay operator q^{-1} such that for a sequence $u(i)$ the operator delays each value by one timestep, i.e. $q^{-1}u(i) = u(i-1)$. The q operator is interchangeable with the z transform.

3.3.1 Questions

- see wk7interactive

3.4 Sampled

Formed as output y(n) responding to input u(n) to a system g(n) where n is an integer, i.e. the output does not exist for any time $t \neq n$

$$y(n) = \sum_{i=-\infty}^{\infty} u(i)g(n-i)$$

4 Definition of linear and non-linear systems.

4.1 Linear

If $y = f(u(t))$ is a system (or function) with an input $u(t)$ that can change with time t then the system can be considered as linear if

$$y' = f(au(t)) = ay$$

for some constant scalar value a.

This applies also to the vector form i.e. if $\underline{\mathbf{y}} = f(\underline{\mathbf{u}})$ then the system can be considered as linear if

$$a\underline{\mathbf{y}} = f(a\underline{\mathbf{u}})$$

[Draw as a block diagram]

4.2 Time invariant

If $y(t) = f(u(t))$ for input u at time t then the system can be considered as time variant if for some constant scalar value τ (or a) $y'(t) = f(u(t+\tau)) = y(t+\tau)$ for some constant scalar value a Again this applies also to the vector form

4.3 Generalised non-linear state equations

The concept of state space has two benefits, first it allows a system to be described as a vector of states such as position, velocity, voltage, current etc. Second it allows the equations (both linear and non linear) to be written as a single level of differentiation.

The general form for a state space with states $\underline{\mathbf{x}}(t)$, inputs $\underline{\mathbf{u}}(t)$ and outputs $y(t)$ is

$$\dot{\underline{\mathbf{x}}} = f(\underline{\mathbf{x}}, \underline{\mathbf{u}}, t) \tag{1}$$

$$\underline{\mathbf{y}} = g(\underline{\mathbf{x}}, \underline{\mathbf{u}}, t) \tag{2}$$

with input $\underline{\mathbf{u}}$, output $\underline{\mathbf{y}}$ and states $\underline{\mathbf{x}}$. All variables are assumed to be a function of t.

4.4 Generalised linear state equations

Linear state space equations are often written with the ABCD notation

$$\dot{\underline{\mathbf{x}}} = A\underline{\mathbf{x}} + B\underline{\mathbf{u}} \tag{3}$$

$$\underline{\mathbf{y}} = C\underline{\mathbf{x}} + D\underline{\mathbf{u}} \tag{4}$$

With constant matrices A, B, C, D

Often, in both linear and non-linear versions, the equations 2 and 4 are simplifed to $\underline{\mathbf{y}} = C\underline{\mathbf{x}}$ with C being an identity matrix.

5 Linearization

It is often. necessary to approximate a non-linear system with a locally equivalent linear system. Then many of the tools for linear systems can be used as long as things don't change too significantly.

Linearisation occurs around a stationary operating point, and is best explained with Taylor's theorem.

5.1 Example

Any function $f(x)$ can be approximated around an stationary operating point a as the Taylor's expansion so that

$$f(x) = f(a) + f'(a)(x - a) + \frac{1}{2}f''(a)(x - a)^2 ...$$

Note the notation where again f' means differentiating. We ignore higher orders so for a first order system we have

$$\dot{x} = f(x) \approx f(a) + f'(a)x - f'(a)a = c_1 + c_2 x$$

where $c_1 = f(a) - f'(a)a$ and $c_2 = f'(a)$

This same concept can be extended into vector forms, i.e. given $\dot{\underline{\mathbf{x}}} = f(\underline{\mathbf{x}})$ we can find a linear approximation where $\dot{\underline{\mathbf{x}}} = A\underline{\mathbf{x}}$

6 Matlab

Matlab ("matrix laboratory") is a programming language for numerical computing, that provides an extensive set of tools and libraries, a relatively readible structure, and good graphics in 2 and 3 dimensions. Matlab also provides a graphical language (simulink) for system analysis and a symbolic maths package based on Mupad. The numerical parts of this course will be based on Matlab and it is recommended that you use this for the second assignment.

6.1 Online Matlab courses

Choose one of the courses below to become more familiar with Matlab. Please let me have feedback on how useful it was

Matlab Onramp (2 hours) Mathworks.

(https://uk.mathworks.com/learn/tutorials/matlab-onramp.html)
(https://matlabacademy.mathworks.com/R2019b/portal.html?course=gettingstarted)

Matlab Fundamentals, Mathworks

(https://matlabacademy.mathworks.com/R2020a/portal.html?course=mlbe)

Learning Matlab (1h13m) Steven Moser, Linkedin learning

(https://www.linkedin.com/learning/learning-matlab-2/welcome?u=83824434)

MATLAB 2018 Essential Training (3h15m) Curt Frye, Linkedin learning

(https://www.linkedin.com/learning/matlab-2018-essential-training/use-matlab-for-data-cal
u=83824434)

6.2 Survey on Matlab courses (blackboard)

Once you have tried an online matlab course please go to the survey below and give your views.
(https://www.bb.reading.ac.uk/webapps/blackboard/content/launchAssessment.jsp?course_id=_155149_1&content_id=_5282250_1&mode=view)

6.3 Matlab alternatives

The university has a Matlab licence for all students. But many of the tools are available through other packages (albeit not as good). Some alernatives are

- Julia (https://julialang.org/)
- GNU Octave (https://www.gnu.org/software/octave/)
- Python with libraries such as numpy, matplotlib, scipy

6.4 Installing Matlab on personal computers

Method 1: AppsAnywhere: https://www.reading.ac.uk/internal/its/AppsAnywhere.aspx

Method 2: Create an account at mathworks.com using your student email address. (https://uk.mathworks.com/mwaccount/) (https://www.reading.ac.uk/internal/its/services/its-SerCat/MatlabStudents.aspx) Download MATLAB (Individual)

7 First and second order linear systems

7.1 Linear first order system

A first order system tends to be in the form

$$\frac{dy}{dt} + ky = 0$$

where k is a constant

The solutions are of the form $y = e^{at}$, so it is possible to solve by comparing coefficients.

Substitute the solution into the first order equation we get

$$ae^{at} + ke^{at} = 0$$

so $a = -k$ and the solution is $y = e^{-kt}$

It is also possible to get a solution with the Laplace transform in which case the first order equation has the Laplace transform

$$(sy - y_0) + ky = 0$$

with y_0 is the value of y at $t = 0$

7.1.1 Input to a first order system

A first order system can have an input driving the response. These are of the form

$$\frac{dy}{dt} + ky = u$$

where u is an input variable. The solution is slightly more complicated and for an arbitrary input the solution would be solved by numerical integration. Some specific inputs can be calculated so for a 'step change' the solution is of the form

$$y = U_0\left(1 - e^{-kt}\right)$$

where U_0 is the size of the step.

7.2 Linear second order system

Second order systems are widely used for modelling e.g.

- vibrations in structures (wobbly bridges, aircraft resonance,
- oscillations e.g. small angle pendulums, electronic oscillators (LCR circuits)
- response of a system to to an input e.g. car accelerometer (input) to car speed (output)

A general form is

$$\frac{d^2y}{dt^2} + \beta\frac{dy}{dt} + \omega^2 y = 0$$

where β (beta) and ω are constant values (damping and natural frequency). Sometimes it will be convenient to write $\beta = 2\zeta\omega$ i.e. use A state space form can be expressed by letting $x_1 \equiv y$ and $x_2 \equiv dy/dy$ so

$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\omega^2 & -\beta\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

i.e. $\dot{\underline{\mathbf{x}}} = A\underline{\mathbf{x}}$ We can check this by substituting back

$$\begin{bmatrix}\frac{dy}{dt}\\ \frac{d^2y}{dt^2}\end{bmatrix} = \begin{bmatrix}\dot{x_1}\\ \dot{x_2}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\omega^2 & -\beta\end{bmatrix}\begin{bmatrix}y\\ \frac{dy}{dt}\end{bmatrix}$$

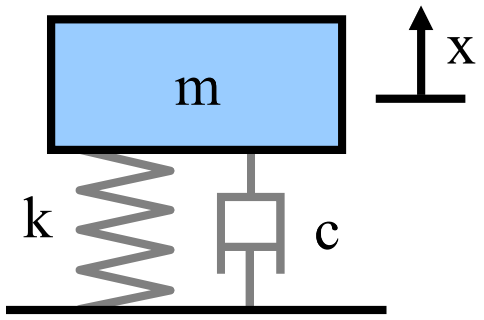


Figure 2: Mass m spring k damper c system. An input force f could also be applied to the mass. (image from wikimedia)

7.2.1 Second order system with an input

A mechanical example is a mass connected to a spring and damper

By equating forces on the mass m it is possible to derive the free running system as

$$m\ddot{x} + c\dot{x} + kx = 0$$

or the force driven system as

$$m\ddot{x} + c\dot{x} + kx = f \tag{5}$$

Equation (5) can also be written as

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = \frac{f}{m}$$

so it is then possible to see that $\omega^2 = \frac{k}{m}$ and $\beta = \frac{c}{m}$

7.3 Eigenvalues and Stability

Recall that for a square matrix A the Eigenvectors v are special vectors that when real have the same direction when transformed by A.

So if

$$v' = Av$$

then v is an Eigenvector if $v' = \lambda v$. i.e.

$$Av = \lambda v \tag{6}$$

The same is true if the Eigenvectors are complex, but it is may be harder to conceptulise a complex scaling value λ and complex direction of v. One additional cavet is that if the Eigenvalue is real and negative then v' will point in exactly the oposite direction to v, that is to say it now as negative magnitude.

From the equation (6) above we can compute a solution by noting that

$$(A - \lambda I)v = 0$$

and by realsing that for any non zero vector v the matrix $A - \lambda I$ must be singular and in which case the determinant must be zero hence

$$\left|A - \lambda I\right| = 0$$

The stability of the system is dictated by the Eigenvalues of the A matrix, more specifically whether the real parts of the Eigenvalues are greater than zero. For a second order system (where we use the substitution $\beta = 2\zeta\omega$) the the Eigenvalues are

$$\lambda = -\omega(\zeta \pm \sqrt{\zeta^2 - 1})$$

Now omega (ω) only effects the magnitude of the Eigenvalues, so we can map out the behaviour of the system based on the value of zeta (ζ).

zeta	lambda (λ)	behaviour
$\zeta < -1$	real positive numbers	exponentially increasing (unstable)
$-1 < \zeta < 0$	complex with positive real part	increasing oscillations (unstable)
$\zeta = 0$	imaginary (real part is zero)	stable oscillations
$0 < \zeta < 1$	complex with negative real part	decreasing oscillations (stable)
$\zeta > 1$	real negative numbers	exponentially decreasing (stable)

8 Numerical integration of ordinary differential equations

We will cover only *Initial value* problems, i.e we are interested in looking at the behaviour of a 'thing' as it evolves over time.

- Each state must have an initial value (even if it is 0)
- The 'system' can have an optional input, or can be only dependent on the initial values of the states
- Initial values determine state (position, voltage, etc) for all subsequent time.
- Often our models of the 'system' are sensitive to the inputs, our calibration of the model, and our measurements of the observable states.

8.1 State-space again

Tend to be set out for problems of the form

$$\dot{\underline{\mathbf{x}}} = f(t, \underline{\mathbf{x}})$$

Sometimes expressed with a mass matrix

$$M(t, \underline{\mathbf{x}})\dot{\underline{\mathbf{x}}} = f(t, \underline{\mathbf{x}})$$

Note: Numerical solvers will often adapt the step-size to the problem and can use additional information, such as problem constraints or higher order differentials. Solver also needs to know the initial value. System inputs can be included in the process.

Note2: Stiff solvers may be needed for problems where there is a strong coupling between state variables. (e.g. the Robertson's chemical reaction (use your favourite search engine to find out more))

8.2 Forward Euler

Suppose we know the continuous time state-space equations for a system.

$$\dot{x} = f(\underline{\mathbf{x}}) + g(\underline{\mathbf{u}})$$

For the moment we will consider f and g non-linear functions, although we can always revert them to being matrices A and B if they system is linear.

Approximate this as

$$\dot{\underline{\mathbf{x}}} \approx \frac{\underline{\mathbf{x}}(t+\Delta) - \underline{\mathbf{x}}(t)}{(t+\Delta) - t} = f(\underline{\mathbf{x}}(t)) + g(u(t))$$

the t in the denominator cancel so we can rearrange the equation to be

$$\underline{\mathbf{x}}(t+\Delta) = \underline{\mathbf{x}}(t) + \Delta f(\underline{\mathbf{x}}(t)) + \Delta g(u(t))$$

If we set Δ to be fixed this is then a sampled data system so we can effectively integrate the states. If we ignore the input for the moment, and accept that $\underline{\mathbf{x}}$ is a function of time, we get

$$x_{n+1} = x_n + \Delta f(x_n)$$

And if the system is linear then $f(\underline{\mathbf{x}})$ becomes $A\underline{\mathbf{x}}$ which is simply a fixed matrix so the equation becomes

$$x_{n+1} = x_n + \Delta A x_n$$

note: Forward Euler is known to be poor, but is often used in real-time software. Delta needs to be small to be effective, but this is then expensive in computer time.

challenge: use the q operator to rewrite this equation, and then see if you can sketch it out as a transfer function and as a code fragment.

8.3 Backwards Euler

If we look at the approximation for the Forward Euler case (and assuming it is linear with no input), we may be able to write it as

$$\frac{\underline{\mathbf{x}}(t) - \underline{\mathbf{x}}(t-\Delta)}{t - (t-\Delta)} = f\underline{\mathbf{x}}(t)$$

or

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{x}}(t-\Delta) + \Delta f \underline{\mathbf{x}}(t)$$

Again if Δ is constant we can consider this as a sampled data system.

$$x_n = x_{n-1} + \Delta f(t_n, x_n)$$

Effectively now are looking backwards in time for the state, but we have already computed the state! This may be possible for some systems but in most cases we will look for more sophisticated numerical integrators, such as Runge Kutta

8.4 Runge Kutta (RK4)

(Here we change from using $\underline{\mathbf{x}}$ as a vector of states to y that can also be a vector of states - that way we don't need to change the wikimedia figure!)

$\dot{y} = f(t, y)$ with initial values $y(0) = y_0$.

[Note the following is a variation of both the Wolfram Mathworld and the Wikipedia explanation and assumes that k_1, k_2, k_3 and k_4 are function gradients - this is not what they are in the Wikipedia article!]

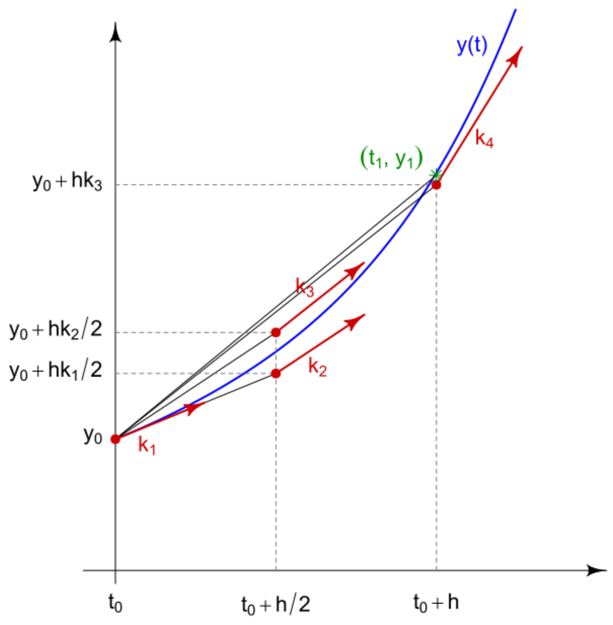


Figure 3: Gradients used to compute Runge Kutta RK4[1]
[1] Image from Wikimedia *Runge Kutta methods*

For a positive step size h compute in order the following values of the function to be integrated (i.e. the function gradients)

You now have 4 gradients as shown in

1. $k_1 = f(t_0, y_0)$
2. $k_2 = f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1)$ i.e use k_1 to locate a point near the true curve a half step away. This requires increment t_0 by $\frac{h}{2}$ and increment y_0 by $\frac{h}{2}k_1$
3. $k_3 = f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2)$ i.e. use k_2 to make another half step estimate
4. $k_3 = f(t_0 + h, y_0 + \frac{h}{k}_3)$ i.e. use k_3 to make a full step estimate.

figure 3. The function estimate used as the result is then

$$y_1 = y_0 + h\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{7}$$
$$t_1 = t_0 + h \tag{8}$$

See(8)

Repeat for the new values of t_1 and y_1 so in general

$$y_{n+1} = y_n + h\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9}$$
$$t_{n+1} = t_n + h \tag{10}$$

Where k_1, k_2, k_3 and k_4 are defined by figure 3

9 Phase planes

- stable node/focus
- unstable node
- saddle point

- stable limit cycle
- unstable limit cycle
- two versions of a semi-stable limit cycle
- a centre

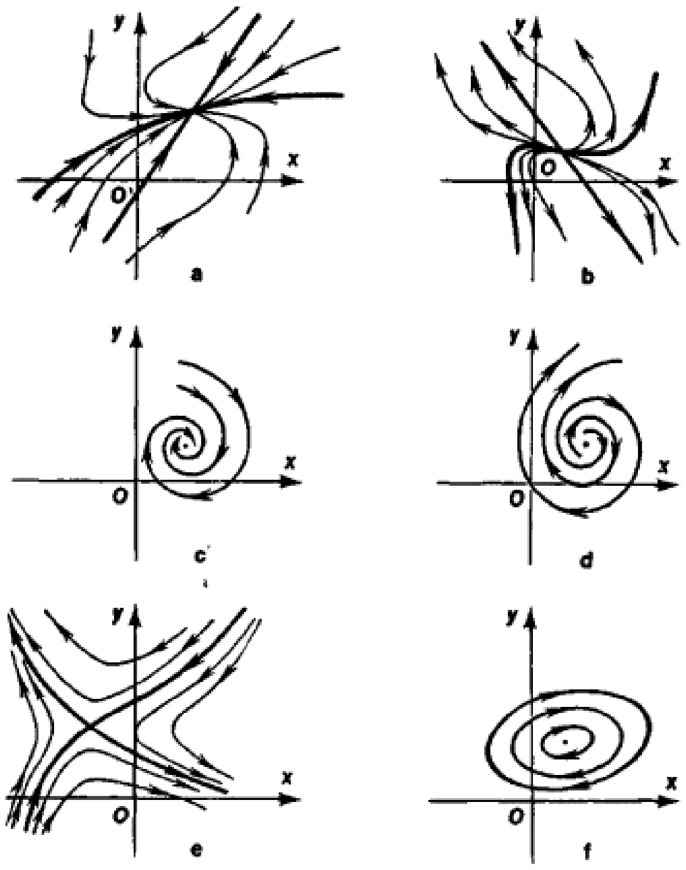


Figure 4: Phase trajectories in the neighbourhood of the following types of singular points: (a) a stable node, (b) an unstable node, (c) a stable focus, (d) an unstable focus, (e) a saddle point, (f) a centre [2]

[2] `https://encyclopedia2.thefreedictionary.com/Phase+Plane+Analysis`

9.1 Example of a simple phase plane

If we assume the system

$x = A\sin(\omega t)$ the system 'velocity' is $\dot{x} = A\omega\sin(\omega t)$

Consider some points (taking ωt as a single angle value)

ωt	$\cos(\omega t)$	x	$\dot{x}$
0	1	0	ωA
$45°$ ($\pi/4$)	0.707 ($\frac{1}{\sqrt{2}}$)	$0.7A$	$0.7\omega A$
$90°$ ($\pi/2$)	0	A	0
$135°$ ($3\pi/4$)	-0.707 ($-\frac{1}{\sqrt{2}}$)	$0.7A$	-$0.7\omega A$
$180°$ (π)	-0.707 ($-\frac{1}{\sqrt{2}}$)	$0.7A$	-$0.7\omega A$

If we now assume that $A = 1$ and $\omega = 1$ we can draw a graph of x against $\dot{x}$

This is a phase plane graph, in our case we are plotting a variable against its differential, but it could be two variables linked in some other way.

9.2 The non-linear not-damped pendulum equations

A simple model of biped walking is based on an 'inverted pendulum'. The assumption is that during the period you have one foot on the ground your knee is 'locked' and your dynamics are similar to a pendulum that can swing in two dimensions Kajita et al. 2001.

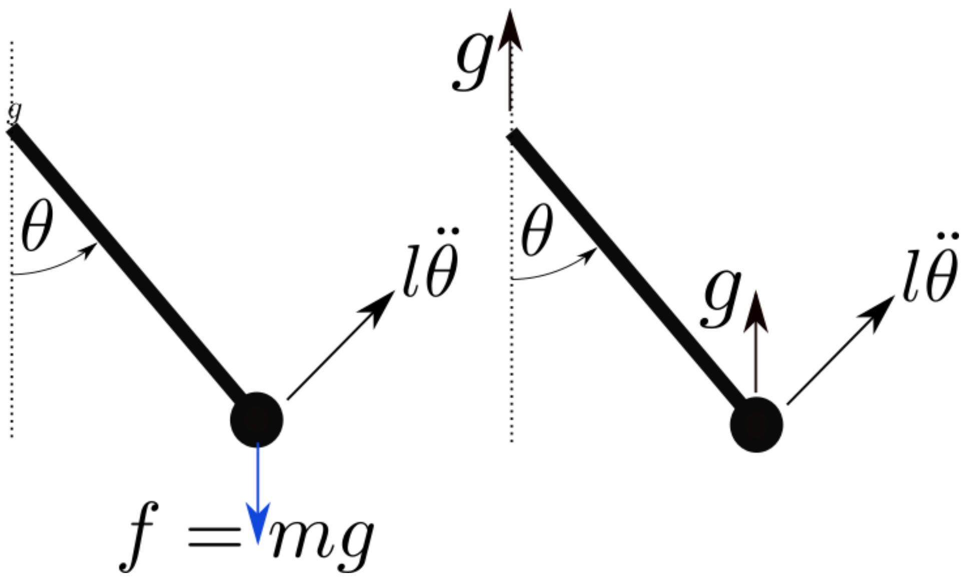


Figure 5: Left subfigure shows forces on a pendulum. Right subfigure considers the equivalent system in terms of accelerations

By analysing the forces on a pendulum (left figure), or equivalently the accelerations (right figure) we can derive the pendulum equations. Either way the non-linear pendulum equation is then

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

This can also be written in state space form as

$$\dot{\theta} = \omega \tag{11}$$

$$\dot{\omega} = -\frac{g}{l}\sin\theta \tag{12}$$

Where we introduce the additional state variable

$$\omega = \frac{d\theta}{dt}$$

We can find a numerical solution to the above state-space equation using 'ode' integrators such 'Euler' or 'Runge-Kutta'.

There is also an analytic approach that can be used to draw and understand the phase plane.

This uses a refactoring of a second differential. If we write $\dot{x} = \frac{dx}{dt} = v$

$$\ddot{x} = \frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} = \frac{dv}{dx}v$$

If we replace x and v in the above equation for θ and ω we can substitute that into the non-linear pendulum equation so

$$\omega\frac{d\omega}{d\theta} = -\frac{g}{l}\sin(\theta)$$

Rewrite to get

$$\omega d\omega = -\frac{g}{l}\sin(\theta)d\theta$$

and integrate with the limits of each integral corresponding.

$$\int_{\omega(0)}^{\omega(t)} \omega d\omega = -\int_{\theta(0)}^{\theta(t)} \frac{g}{l}\sin(\theta)d\theta$$

Once the integration is done and the limits evaluated the equation becomes

$$\frac{1}{2}\left(\omega^2(t) - \omega^2(0)\right) = \frac{g}{l}\left(\cos\theta(t) - \cos\theta(0)\right)$$

So it is then possible to rearrange to compute $\omega(t)$ from the initial conditions $\omega(0)$ and $\theta(0)$ and the

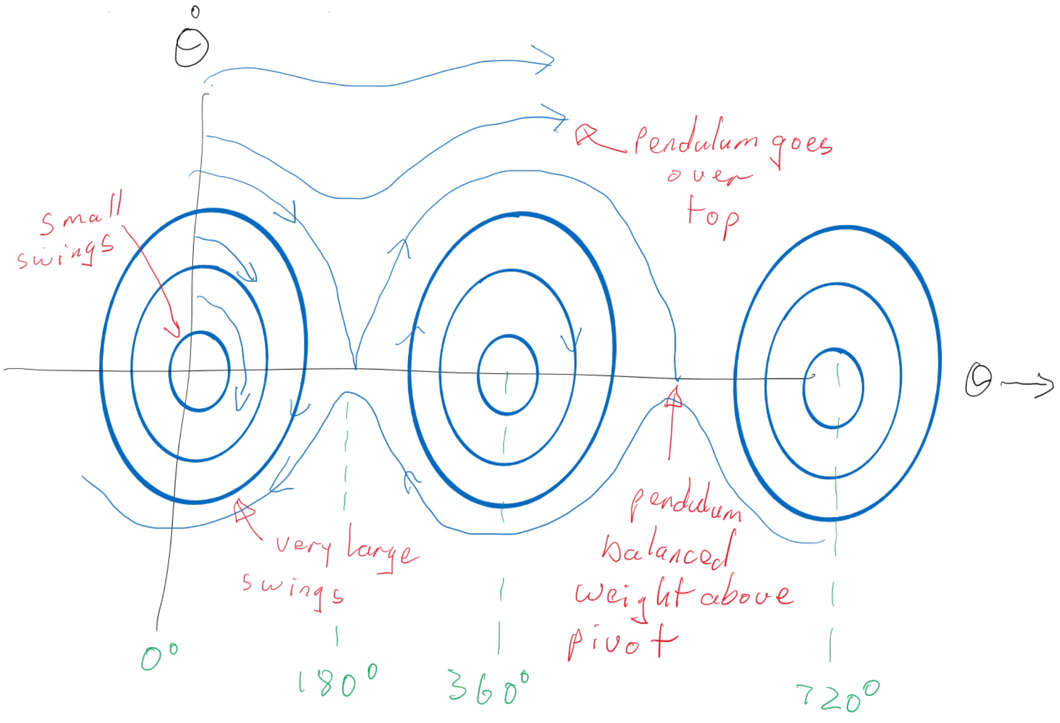


Figure 6: The phase plane of a frictionless pendulum

current angle $\theta(t)$ That is

$$\omega(t) = \pm\sqrt{\frac{g}{l}\left(\cos(\theta(t)) - \cos(\theta(0))\right) + \omega^2(0)}$$

Since this is no longer time dependent we can write it as

$$\omega = \pm\sqrt{\frac{g}{l}\left(\cos\theta - \cos\theta_0\right) + \omega_0^2} \tag{13}$$

Challenge is to plot ω vs θ for a variety of angles

9.3 The non-linear damped pendulum equations

$$\ddot{\theta} = -\frac{g}{l}\sin\theta - \frac{B}{m}\dot{\theta} \tag{14}$$

where l and m the pendulum length and mass, in this case the constant B is known as damping (inherent energy loss) of the system, and g is gravitational acceleration.

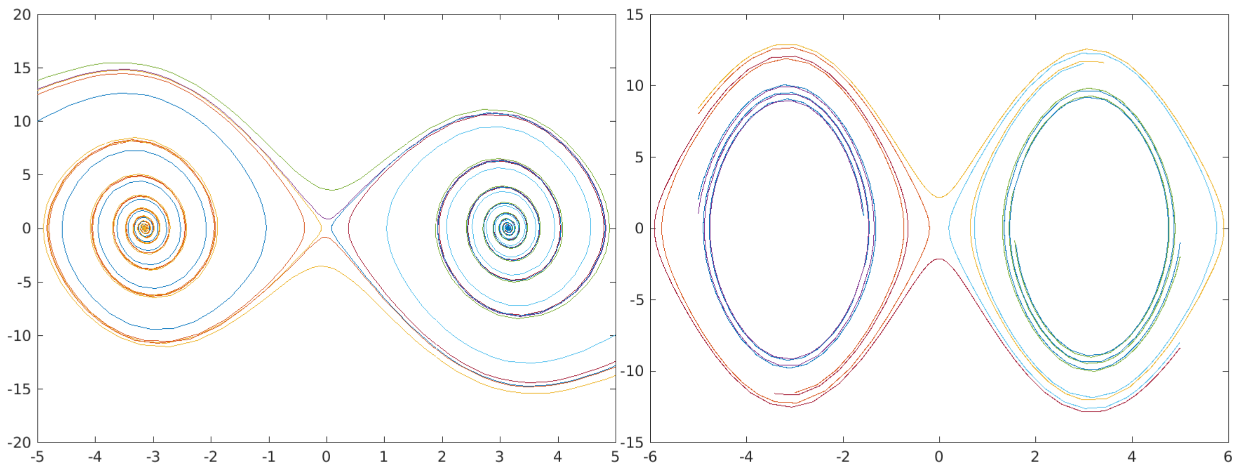


Figure 7: A '1 second' pendulum, length=248 mm. High damping (Left), low damping (right)

9.3.1 Special case

Closed form solution if $B = 0$

Assuming $\dot{x} = v$ Can use the relationship

$$\frac{d^2x}{dt^2} = \frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} = \frac{dv}{dx}v$$

Now do the equivalent calculation for $\dot{\theta} = \omega$ and solve equation 14

10 Phase plane gradient

If we have a state space model we can choose any two states and map out the state gradients with all other states fixed.

If we assume the general form of a non-linear model with two states of the form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

If our phase space has axes for x_1 (horizontal) and x_2 (vertical) we can, for every point on the x_1, x_2 plane, compute the gradient of the trajectory.

This gradient is $\frac{dx_2}{dx_1}$ and we can expand it so

$$\frac{dx_2}{dx_1} = \frac{dx_2}{dx_1}\frac{dt}{dt} = \frac{dx_2}{dt}\frac{dt}{dx_1} = \frac{dx_2}{dt}/\frac{dx_1}{dt} = \frac{\dot{x}_2}{\dot{x}_1}$$

10.1 van der Pol oscillators

First studied by the Dutch engineer/physicist Balthasar van der Pol, the equations emerge from an analysis of vacuum tube oscillators. Later used by Mary Cartwright [8] and J.E. Littlewood to explore chaos. The van der Pol equation (with zero input) expressed as a second order single equation is

$$\frac{d^2y}{dt^2} - \mu(1 - y^2)\frac{dy}{dt} + y = 0$$

where μ is a constant value that controls behaviour.

Some observations are:

[8] First woman to get a first in Maths from Oxford, to win RS Sylvester medal, to be elected to the RS Council, to be president of the London Mathematical Society

when $\mu \to 0$ the equations are a simple harmonic oscillator.
if $y^2 < 1$ the equations are locally unstable
if $y^2 > 1$ the equations are locally stable
if $y^2 = 1$ the equations are locally an undying oscillations (sin wave motion)

We can express the van der Pol equations in a state space form

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & \mu(1-x_1^2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Although this looks like a linear system this is not the case. Why? (are all the values in the A matrix constants?)

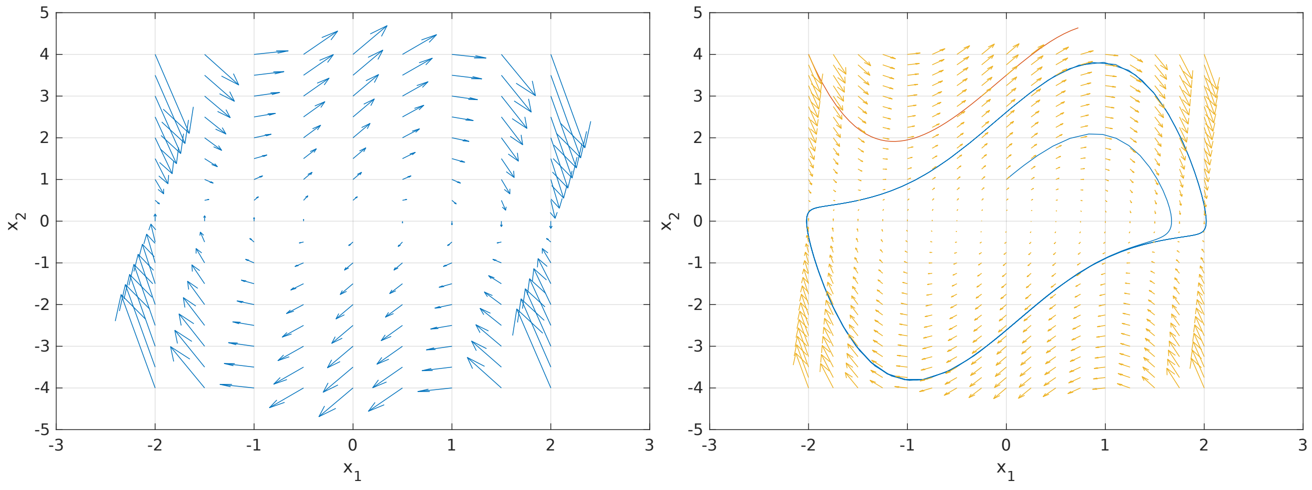


Figure 8: local gradients in the phase space of the van der Pol equation. Note change of behaviour at $x_1 = \pm 1$

Local gradients can be calculated at all points in the phase plane since

$$\frac{dx_2}{dx_1} = \frac{dx_2}{dt}\frac{dt}{dx_1} = \frac{\dot{x}_2}{\dot{x}_1}$$

So for the van der Pol oscillator the local gradient for any point can be computed as

$$\frac{\dot{x}_2}{\dot{x}_1} = \frac{x_2}{-x_1 + \mu(1-x_1^2)x_2}$$

The above conditions are now

if $x_1^2 < 1$ the equations are locally unstable
if $x_1^2 > 1$ the equations are locally stable
if $x_1^2 = 1$ the equations are locally an undying oscillations (sin wave motion)

11 Some examples of non-linear systems models

There are many non-linear system models, using a variety of techniques to model dynamics. The following is a sample of a few that are based on non-linear differential equations.

11.1 Izhikevich Neuron models

This is an efficient spiking model of a Neuron proposed by Eugene Izhikevich in his paper "Simple Model of Spiking Neurons" IEEE Transactions on Neural Networks (2003) 14:1569- 1572

Izhikevich uses a simple Euler integrator.

$$\dot{v} = ev^2 + fv + g - u + I \tag{15}$$

$$\dot{u} = a(bv - u) \tag{16}$$

v represents the membrane potential, u represents the neuron recovery (related to the flux of potassium post firing) I is treated as an input current and either excites or inhibits the neuron.

if $v \geq 30$, then v resets so $v = c$ and u resets to $u + d$.

a, b, c, d, e, f and g are constants that depend on the type of neuron. Typically $e = 0.04$, $f = 5$, $g = 140$. The parameters a, b, c, d are chosen to represent differing neurons as per figure 11.1

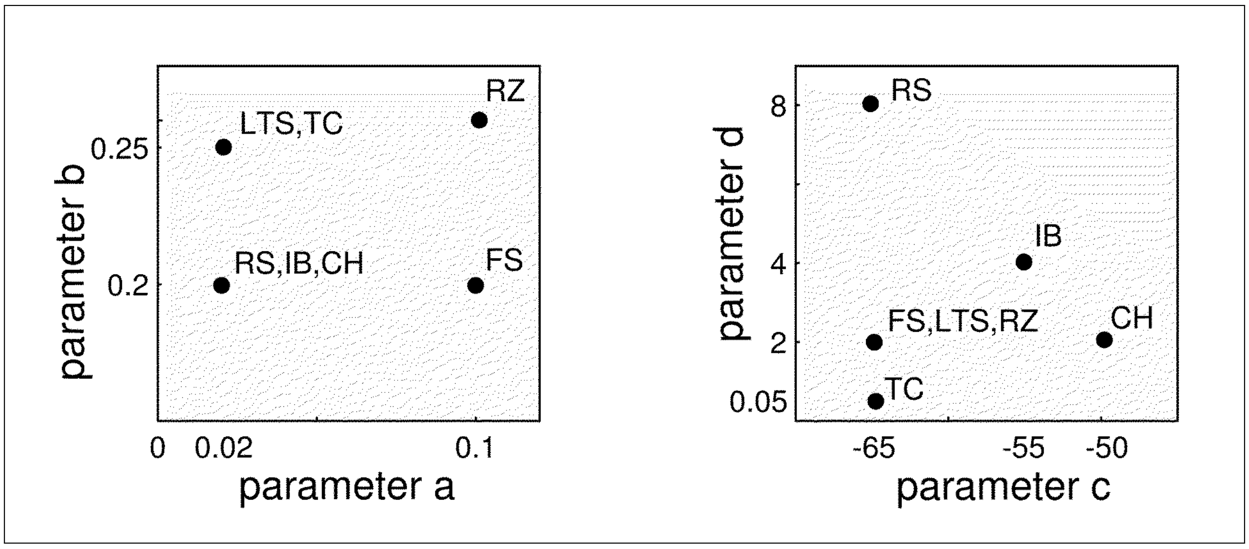


Figure 9: Parameter values for a, b, c, d to simulate Low threshold spiking (LTS), Thalmo-cortical (TC), Regular spiking (RS), Intrinsic bursting (IB), Fast spiking (FS), Chattering (CH) and Resonator (RZ) - See Izhikevich 2003 and www.izhikevich.com

(https://www.izhikevich.org/publications/spikes.htm)
(https://courses.cs.washington.edu/courses/cse528/07sp/izhi1.pdf)

11.2 Lorenz equations

The Lorenz equations were the first evidence of chaotic system. They were originally a description of the convection in clouds, but are now seen as one of the classic examples of a system that exhibits chaotic and predictable behaviour, depending on the parameter values of the model.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x), \tag{17}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y, \tag{18}$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z. \tag{19}$$

Classically Lorenz equations used the values $\sigma = 10, \beta = 8/3$ and $\rho = 28$. when the system is chaotic.

11.3 Population models

The Lotka–Volterra model are a very simple two species population model describing prey that expand their population exponentially, and predators that expand their population as a function of the number of available prey.

$$\frac{dx}{dt} = \alpha x - \beta xy,$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$

If x are the pray and y are the predators, then the constants in the above equation can be considered to mean the following.

- α this is the exponential rate of increase of the in the absence of predators. It contains both the birth rate and the death rate and is assumed to be greater than 1, otherwise the prey will die out even if there are no predators.
- β this is the harvesting rate of the predators. The more predators there are the more of the available prey that can be eaten.
- δ is the ability the predators have the sustain life by eating prey.
- γ the rate the predators die off from starvation if there are no prey to eat.

These equations exhibit classic limit cycles. In the absence of predators the prey population increases. As more prey become available, the predictor population increases. Too many predators eat the prey and the prey population decreases. Since there are fewer prey to eat the predators die of starvation.

11.4 Epidemic models

The classic epidemic model is known as the SIRs modelDitsworth 2019. It is an example of a compartmental model where drugs, people, things etc are considered to move at specific rates between different compartments. In the case of the SIRS epidemic model these compartments are *Susceptible*, *Infected* and *Removed.*

In the case of an epidemic *Removed* can be either because the person has recovered or has died. Thus the progression of the population is from Susceptible through to infected, and finally either recovered or died, but either way, no longer effecting the epidemic.

There are two constants that drive the model, the contact rate (`C` or β) that controls the movement from *Susceptible* S to *Infected* and the healing rate `H` or *gamma* that controls the movement from *Infected* to *Removed.*

The SIR model is then

$$\frac{dS}{dt} = -\mathtt{C}SI \tag{20}$$
$$\frac{dI}{dt} = \mathtt{C}SI - \mathtt{H}I \tag{21}$$
$$\frac{dR}{dt} = \mathtt{H}I = \dot{S} - \dot{I} \tag{22}$$

The R number is the reproduction number and $R = \frac{\mathtt{C}}{\mathtt{H}}$ is the expected number of cases that will be generated per infected person in the population. Thus if a person with the disease infects more than 1 other person then the epidemic will escalate, if a person with the disease infects less than one other person the disease will die away (we can either allow fractional people, or consider this as say 100 people infecting on average R*100 other people). [9] The initial value of R is sometimes known as R_0. The management of an epidemic can be considered as finding ways to to reduce the R value, ideally so that it is less than 1. Thus measures for covid19 at the moment include keeping distance, washing hands wearing face-masks. Track and trace effectively reduces the contact rate C and hence R. When a vaccine becomes available this can considered a way of removing people from S. Alternatively it can be considered as reducing `C`. As more is learned about the management of the disease there remains an additional way to reduce R by increasing `H`. Thus it is possible to influence the course of the epidemic as shown in the figure below (`http:sir4.m`) .

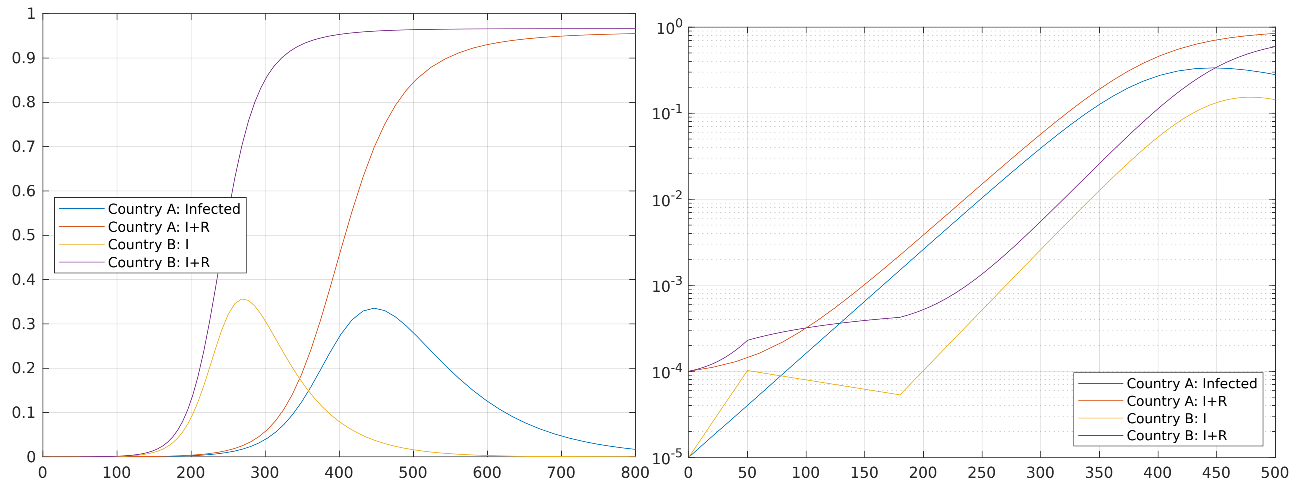


Figure 10: Left subfigure: SIR model for two countries with different R0 values. Right subfigure: SIR model in log form. Country B imposes a lock-down between days 50 and 180.

12 Model fitting

Please note in the following section we will use both the underscore/overarrow ($\underline{\mathbf{v}}$) and bold face $\boldsymbol{\phi}$ to represent vectors.

If we have some data and a proposed model for the data the challenge is to find the parameters for that model that best represent the data. For example if you have a graph that shows the height of a person in relation to their age, you could fit a variety of models

The figure below shows scatter plots for height-age data of young women in the !Kung San culture collected in late 1960s. The linear model is of the form

$$y = mx + c \tag{23}$$

The non-linear model is of the form

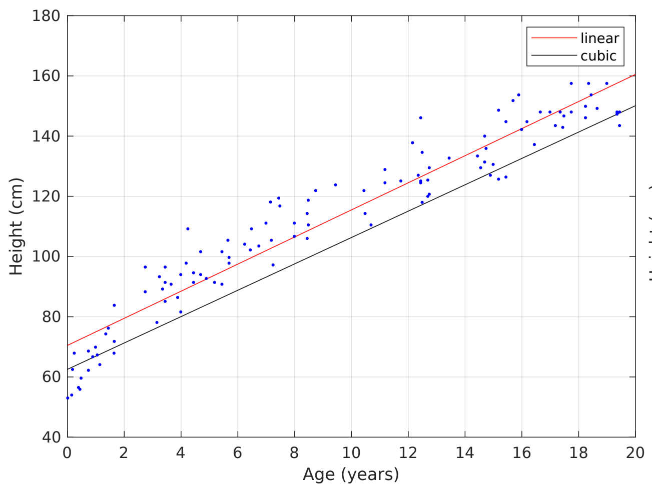
$$y = m_1x^2 + m_2x + c \tag{24}$$

Which model best represents the data? It may be that if we are considering individual women below the age of 20 we can use a linear regression, however if we would like a model to fit a larger population we will need to use more sophisticated models of the data, depending on this model we may still be able to fit the model with the technique of least squares regression (first proposed by Gauss in 1822).

13 Least squares regression

We can write a more general form for the equations (23) and (24) by putting the parameters (m_1, c etc) into a vector variable, and creating a matrix of independent parameters X. In the case of equation (24) this can be written as

[9] see also (`https://en.wikipedia.org/wiki/Basic_reproduction_number`)



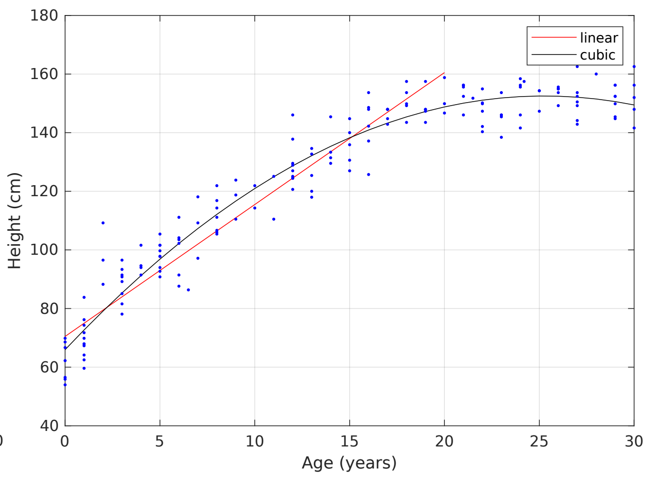


Figure 11: Left subfigure: Age-height scatterplot with fitted models for !Kung San women less than 20. Right subfigure: Age-height scatterplot with fitted models for !Kung San women less than 30. (Data collected by Nancy Howell (1967-69) and archived at (`https://tspace.library.utoronto.ca/handle/1807/10395`))

$$y = \begin{bmatrix} x^2 & x & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ c \end{bmatrix} = \begin{bmatrix} x^2 & x & 1 \end{bmatrix} \boldsymbol{\theta} = \phi^T \boldsymbol{\theta}$$

where $\boldsymbol{\theta}$ contains the constant values (model parameters) that we would like to fit to our data.

We can now consider our model for every data point model, it may not fit precisely so we can add a variable e to account for this difference.

The simplest parametric model is linear regression.

$$y_i = \hat{y}_i + e_i = \boldsymbol{\phi}_i^T \boldsymbol{\theta} + e_i \tag{25}$$

The variable $\hat{y}_i$ is known as y-hat and is the models estimate of the true result y_i. ϕ_i is a vector of known quantities, and $\boldsymbol{\theta}$ is a vector of unknown parameters.

We can expand the above equation for each data point, this will result in a matrix calculation that we can then solve for the parameters in $\boldsymbol{\phi}$. So

$$\begin{aligned} y_1 &= \hat{y}_1 + e_1 = \phi_1^T \boldsymbol{\theta} + e_1 \\ y_2 &= \hat{y}_2 + e_2 = \phi_2^T \boldsymbol{\theta} + e_2 \\ y_3 &= \hat{y}_3 + e_3 = \phi_3^T \boldsymbol{\theta} + e_3 \\ &\vdots \\ y_n &= \hat{y}_n + e_n = \phi_n^T \boldsymbol{\theta} + e_n \end{aligned}$$

Forming this into a matrix we can write

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \phi_1^T \\ \phi_2^T \\ \phi_3^T \\ \vdots \\ \phi_n^T \end{bmatrix} \boldsymbol{\theta} + \begin{bmatrix} e_1 \\ y_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix}$$

or as vectors and matrices this becomes

$$\mathbf{y} = X\boldsymbol{\theta} + \mathbf{e}$$

Note that $\mathbf{e}$ is modelling error and this is usually assumed to be a random variable with zero mean.

Given a model form and a data set, we wish to calculate the parameter $\boldsymbol{\theta}$. One way is to minimise the errors between the actual y_i and the model prediction $\hat{y}_i$.

13.1 Example 1:

We wish to fit a model of the form $y(t) = au(t) + b + e(t)$ with the following data

t	1	2	3	4	5
$u(t)$	1	2	4	5	7
$y(t)$	1	3	5	7	8

We need to form the following vectors and matrices $\boldsymbol{\theta} = [a, b]^T$, $\boldsymbol{\phi}(t) = [u(t), 1]^T$. $X = \begin{bmatrix} \phi_1^T \\ \phi_2^T \\ \phi_3^T \\ \vdots \\ \phi_n^T \end{bmatrix}$

$$\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \\ \hat{y}_5 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 4 & 1 \\ 5 & 1 \\ 7 & 1 \end{pmatrix} \boldsymbol{\theta} = X\boldsymbol{\theta}$$

13.1.1 Challenge

What is the correct form of the vector $\mathbf{y}$, where $\hat{\mathbf{y}} + \mathbf{e} = \mathbf{y}$

13.2 Example 2:

We wish to fit a model of the form $y_i = b_2 u_i^2 + b_1 u_i + b_0 + e_i$
$\boldsymbol{\theta} = [b_2, b_1, b_0]^T$, $\boldsymbol{\phi}_i = [u_i^2, u_i, 1]^T$.

$$\begin{aligned} \hat{\mathbf{y}} &= \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{N-1} \\ \hat{y}_N \end{pmatrix} \\ &= \begin{pmatrix} u_1^2 & u_1 & 1 \\ u_2^2 & u_2 & 1 \\ \vdots & \vdots & \vdots \\ u_{N-1}^2 & u_{N-1} & 1 \\ u_N^2 & u_N & 1 \end{pmatrix} \boldsymbol{\theta} \\ &= X\boldsymbol{\theta} \end{aligned}$$

13.3 Derivation of the least squares algorithm

For $i = 1, ..., N$,

$$e_i = y_i - \hat{y}_i$$

In vector form

$$\mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_{N-1} \\ e_N \end{pmatrix} = \mathbf{y} - \hat{\mathbf{y}}$$

The sum of squared errors is known as SSE and is

$$\mathrm{SSE} = \sum_{i=1}^{N} e_i^2 = \mathbf{e}^T\mathbf{e}$$

The model parameter vector $\boldsymbol{\theta}$ is derived so that the distance between these two vectors is the smallest possible, i.e. SSE is minimised.

$$\begin{aligned} \mathbf{e}^T\mathbf{e} =& (\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}}) \\ =& (\mathbf{y}^T - \hat{\mathbf{y}}^T)(\mathbf{y} - \hat{\mathbf{y}}) \end{aligned}$$

We can make the substitutions $\hat{\mathbf{y}} = X\boldsymbol{\theta}$ and $(X\boldsymbol{\theta})^T = \boldsymbol{\theta}^T X^T$ into the above equation then expand so

$$\begin{aligned} \mathbf{e}^T\mathbf{e} =& (\mathbf{y}^T - \boldsymbol{\theta}^T X^T)(\mathbf{y} - X\boldsymbol{\theta}) \\ =& \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\theta}^T X^T\mathbf{y} + \boldsymbol{\theta}^T X^T X\boldsymbol{\theta} \end{aligned}$$

Note that the *sum of squares error* is a scalar value that will always be positive or zero. We would like to find the smallest sum of squares error, that is

$$\mathrm{SSE} = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\theta}^T X^T\mathbf{y} + \boldsymbol{\theta}^T X^T X\boldsymbol{\theta}$$

and the SSE has the minimum when

$$\frac{\partial(\mathrm{SSE})}{\partial\boldsymbol{\theta}} = \mathbf{0}$$

Differentiating the equation is relatively straight forward, even though these are matrices.

$$\begin{aligned} \frac{\partial(SSE)}{\partial\boldsymbol{\theta}} &= -2[\frac{\partial(\mathbf{y}^T X\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}]^T + \frac{\partial(\boldsymbol{\theta}^T X^T X\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} \\ &= -2X^T\mathbf{y} + 2X^T X\boldsymbol{\theta} \\ &= -2X^T(\mathbf{y} - X\boldsymbol{\theta}) = \mathbf{0} \end{aligned}$$

yielding

$$\hat{\boldsymbol{\theta}} = [X^T X]^{-1} X^T\mathbf{y} \tag{26}$$

13.4 Numerical solutions

The least squares equation (26) has several methods for finding a numerical solution these include

- Direct computation of Theta=inv(X'*X)*X'*Y
- Left matrix divide Theta=X\Y
- Singular value decomposition of X, i.e. [U,S,V] = svd(X) where X = U **S**V' and S is diagonal, while U and V are unitary. This is essentially a method of doing a matrix inverse for any non-square or square matrix. Depending on the dimension the answer is either the inverse, a least squares solution, or an incomplete solution.

14 Optimisation

Optimisation is the process of adjusting to a situation. Evolution results in animals well suited to their environment. A sports person such as an archer can optimise parameters such as their bow size and strength, their strength to draw back the string, the ability to maintain position by regulating breathing etc, as well as their aim. When presented with data it is often not possible to use least squares techniques to understand and fit a model so an alternative technique is to use a mathematical technique called optimisation.

Optimisation requires an evaluation of cost and a method to change the 'input' parameters (states/configurations etc).

- Determine a cost metric and a cost function.
- Minimise the cost by changing the parameters that determining this cost
- Stop when cost is deemed acceptable, or when you run out of resources (e.g. computing time).
- The relationship between the parameters and the cost is not necessarily constant. In this case optimisation might continue forever.

Other points

- A function stands in for a model of the world
- maximisation is the same as minimisation, but with the sign of the cost function reversed.

Examples:

- Travelling salesman problem, given that you need to visit a number of cities, what route should you take to minimise the distance travelled
- Chess playing machines (including deep blue), given the state of the board, what move should you (the computer) make that increases the probability of winning.
- Bongard/Lipson starfish, what muscles should be activated to allow movement.
- Evolution e.g.Stewart 2004/ Learning. Demonstrates that the cost function may need to change as the situation changes.

15 Function minimisation (mathematical optimisation)

Given a function $f(\underline{\mathbf{u}})$ what vector $\underline{\mathbf{u}}$ gives the smallest value for f. That is, find $\underline{\mathbf{u}}_0$ such that $f(\underline{\mathbf{u}}_0) < f(\underline{\mathbf{u}}_{not\ 0})$

15.1 Constraint minimisation.

Challenge now is to find the minimum value of a function given additional constraints (e.g. the vector should always be positive)

find **min** $f(\underline{\mathbf{u}})$ such that a number of constraint equations are also satisfied. for example $g(\underline{\mathbf{x}}) = c$ or $g(\underline{\mathbf{x}}) > c$

Lagrange multipliers are a technique that can be used for constrained minimisation.

15.2 Data modelling

Suppose we are given some data relating some an independent vector of variables $\underline{\mathbf{x}}$ to some dependent variable y. We would like to fit a model with the parameters $\underline{\mathbf{u}}$ so that the estimate of y can be made with

$$\hat{y} = h(\underline{\mathbf{x}}, \underline{\mathbf{u}})$$

We can define a metric such as

$$f(\underline{\mathbf{u}}) = \sum_{\text{all data points } i} (y_i - \hat{y}_i)^2$$

so

$$f(\underline{\mathbf{u}}) = \sum_{\text{all data points } i} (y_i - h(\underline{\mathbf{x}}_i, \underline{\mathbf{u}}))^2$$

We can now use an 'out-of-the-box' solver to find our best set of parameters for $\underline{\mathbf{u}}$

15.3 Established techniques

Some examples are

- Simulated annealing
- Nelder-Mead Simplex Method (e.g. Matlab `fminsearch`)
- Gradient descent (e.g. Matlab `fminunc`)
- Genetic algorithms
- Constrained minimisation (e.g. Matlab `fmincon` and `fminbnd`)

15.3.1 Simulated annealing

RefsIngber 1995 and Mathworks link (`https://uk.mathworks.com/help/gads/how-simulated-annealing-` `html`)

- Uses the concept of temperature to control the minimisation process
- Less likely to get caught in a local minimum.
- Requires
  - a starting state
  - rule to map the starting state to a new state
  - a cost function
  - a possible termination condition
- Can be computationally intensive

1. Set the starting state $\underline{\mathbf{x}}_i = \underline{\mathbf{x}}_{\texttt{start}}$ and the starting temperature T.

2. Generate a trial point $\underline{\mathbf{x}}_{i+1}$ at some distance from the current state according to some 'temperature'. For example choose a direction to move from a normal N(0,1) distribution (Gaussian) that is, mean 0, variance 1. Scale this direction by either the 'temperature' (fast) or square root of 'temperature' (Boltzman).
3. Possibly constrain the optimisation to stay within some boundary.
4. Compute the cost $C_{i+1} = f(\underline{\mathbf{x}}_{i+1})$
5. If $C_{i+1} < C_i$ then set the state to $\underline{\mathbf{x}}_{i+1}$
6. If $C_{i+1} > C_i$ then set the state to $\underline{\mathbf{x}}_{i+1}$ with a probability of

$$p = \frac{1}{1 + e^{(C_{i+1} - C_i)/T}}$$

   i.e. more likely to change to a worse state if the temperature is high and the cost difference is low.
7. Progressively lower the 'temperature', and stop when T gets sufficiently close to zero. Various algorithmic tricks can be used to improve the outcome.

Is this guaranteed to get to the global minimum?

15.3.2 Nelder-Mead Simplex Method

The following is intended to outline the concept of minimisation using the Nelder-Mead Simplex Method. Fortunately others have encoded the algorithm, see Press et al. 1992 and (`https://uk.mathworks.com/help/matlab/math/optimizing-non-linear-functions.html`)

- Uses a simplex (polytope), i.e. triangle in 2D, tetrahedron in 3D, 5-cell in 4D.
- That is for n dimensions the simplex will have $n + 1$ vertices.
- Also known as the amoeba method, presumably because the simplex grows and shrinks, and tries to squeeze though small spaces.
- Algorithm used by Matlab `fminsearch`

Need to determine where to make the next function (cost) evaluation

The simplex moves towards the minimum by *reflection, expansion, contraction (inside and outside) and shrinking.*

Assuming the initial $n + 1$ vertices are $\underline{\mathbf{x}}_1 \ldots \underline{\mathbf{x}}_{n+1}$ Steps are then one algorithm (of many possible variants) is

1. Evaluate the function at the nodes and identify the largest $\underline{\mathbf{x}}_{\texttt{worst}}$, and the smallest $\underline{\mathbf{x}}_{\texttt{best}}$
2. Calculate the mean of all vertices other than the worst, i.e.

$$\underline{\mathbf{x}}_m = \frac{1}{n} \sum_{i \neq \texttt{worst}} \underline{\mathbf{x}}_i$$

   This gives a vector to a point in the surface opposite to $\underline{\mathbf{x}}_{\texttt{worst}}$ (**Xg** in figure 12). If $\underline{\mathbf{p}}$ is the vector from $\underline{\mathbf{x}}_{\texttt{worst}}$ to $\underline{\mathbf{x}}_m$, that is $\underline{\mathbf{p}} = \underline{\mathbf{x}}_m - \underline{\mathbf{x}}_{\texttt{worst}}$, then it is possible to locate a point $\underline{\mathbf{x}}_r = \underline{\mathbf{x}}_m + p$. Evaluate the function at this point to get $f(\underline{\mathbf{x}}_r)$
3. *Reflection* is when $f(\underline{\mathbf{x}}_{\texttt{best}}) < f(\underline{\mathbf{x}}_r) < f(\underline{\mathbf{x}}_m)$. Replace $\underline{\mathbf{x}}_{\texttt{worst}}$ with $\underline{\mathbf{x}}_r$, and either terminate or restart the algorithm.
4. If $f(\underline{\mathbf{x}}_r) < f(\underline{\mathbf{x}}_{\texttt{best}})$ then try *Expansion.* Locate the point $\underline{\mathbf{x}}_s = \underline{\mathbf{x}}_m + 2\underline{\mathbf{p}}$, then is $f(\underline{\mathbf{x}}_s) < f(\underline{\mathbf{x}}_r)$ replace $\underline{\mathbf{x}}_{\texttt{worst}}$ with $\underline{\mathbf{x}}_s$. Terminate or restart.
5. If $f(\underline{\mathbf{x}}_r) \geq f(\underline{\mathbf{x}}_{\texttt{second worst}})$ try a *Contraction.* Find $\underline{\mathbf{x}}_{c1} = \underline{\mathbf{x}}_m + \frac{1}{2}\underline{\mathbf{p}}$. If $f(\underline{\mathbf{x}}_{c1}) < f(\underline{\mathbf{x}}_r)$ swap $\underline{\mathbf{x}}_{\texttt{worst}}$ for $\underline{\mathbf{x}}_{c1}$ (Contraction outside), otherwise go to step 7 and try *Shrinking* the simplex.
6. If $f(\underline{\mathbf{x}}_r) > f(\underline{\mathbf{x}}_{\texttt{worst}})$ try to *Contract inside* the simplex. Find $\underline{\mathbf{x}}_{c2} = \underline{\mathbf{x}}_m - \frac{1}{2}\underline{\mathbf{p}}$. If $f(\underline{\mathbf{x}}_{c2} < f(\underline{\mathbf{x}}_{\texttt{worst}}$ then use that point, otherwise go to step 7 and try shrinking.
7. If all else fails then keep $\underline{\mathbf{x}}_{\texttt{best}}$ and *Shrink all* the other points towards $\underline{\mathbf{x}}_{\texttt{best}}$ by 50 %.

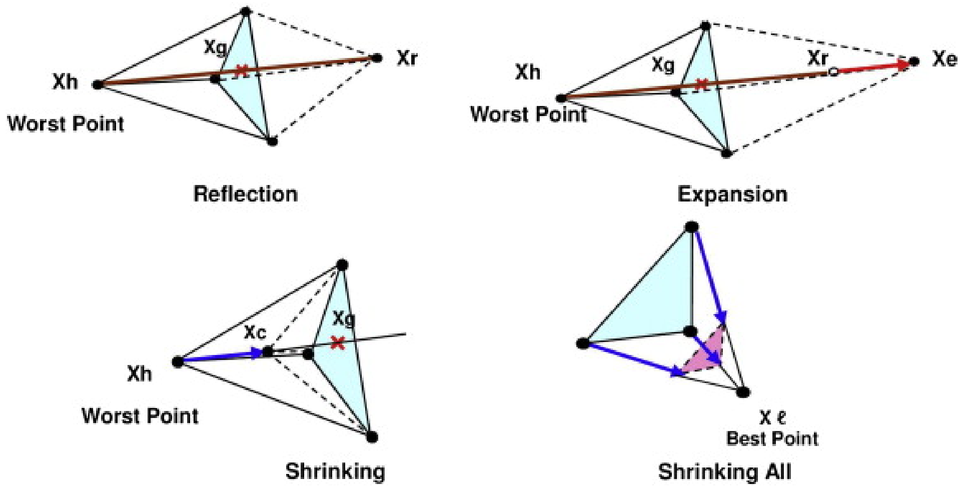


Figure 12: Nelder Mead Simplex operations. From Maehara and Shimoda (https:doi.org/10.1016/j.applthermaleng.2013.08.021)

15.3.3 gradient descent

This is a variant of the Newton–Raphson method

$$\underline{\mathbf{x}}_{n+1} = \underline{\mathbf{x}}_n - k\nabla f(\underline{\mathbf{x}}_n)$$

where in three dimensions

$$\underline{\mathbf{g}} = \nabla f = \begin{bmatrix} \frac{df}{dx} & \frac{df}{dy} & \frac{df}{dz} \end{bmatrix}^T$$

Using $f = f(\underline{\mathbf{x}}_n)$

k is a small constant so that in most cases

$$f(\underline{\mathbf{x}}_n) \geq f(\underline{\mathbf{x}}_{n+1})$$

Nabla can be though of as a differentiating operator working on f so

$$\nabla = \begin{bmatrix} \frac{d}{dx} & \frac{d}{dy} & \frac{d}{dz} \end{bmatrix}^T$$

Evidently this is only going to work if you can compute a gradient (Nabla) function. This is the method used to train multilayer perceptrons (neural networks).

16 Animal learning

Norbert Weiner (author of ‘Cybernetics’) identified two phenomena that characteristic living systems:

- the power to learn.
- the power to reproduce.

Why is artificial intelligence so concerned with chess machines when *Elephants don’t play chess?* (Title of a paper by Rodney Brooks, MIT, 1990, http:doi.org/10.1016/S0921-8890(05)80025-9)

17 Resilient machines (2006)

Key paper is Bongard, Zykov and Lipson 2006Bongard, Zykov, and Lipson 2006b other papers are lipson Bongard, Zykov, and Lipson 2006a

The robot is the starfish. It has a fixed morphology consisting of

Figure 13: Question: why do we (animals) have brains whereas plants do not? Daniel Wolpert's answer - Motor chauvinism - to move!

- 8 joint sensors,
- 2 tilt sensors and
- 8 joint actuators.

- Most robotic systems get the humans to construct a mathematical model of the robot kinematics and/or dynamics and then plan movement against that model.
- Approach is expensive (time to construct a valid model), and requires calibration
- Robot does not have an explicit model of itself
  - Methods like SLAM model and adapt to change the environment
  - Very few robots allow adapt to changes to their own morphology
- Premise, create multiple internal models and use a system identification like approach to select the best
- Don't freeze the system identification process during use
  - Rather look for disagreements between model and sensors (recall Kalman filters?)
  - When disagreement is high, re-initiate the 'model generation and evaluation' cycles.

(`https://www.creativemachineslab.com/evolutionary-self-modeling.html`)

Some info on the neural network based learning of movements is covered inBongard and Lipson 2005

Internal model done with the Open Dynamics Engine (www.ode.org)

17.1 SUPPLEMENTAL MATERIAL

17.1.1 Video

The 'self modelling video has the following parts':

Part I three cycles of model synthesis and action synthesis (Fig. 1A-C).
Part II locomotion synthesis using the best self-model (Fig. 1D).
Part III the physical robot executing the best behaviour (Fig. 1E).
Part IV a sample experiment after the robot suffers damage. The robot is shown alternating self-modelling with exploratory action (Fig. 1A-C); then, the best compensatory gait is shown running on the self- model (Fig. 1D), after which it is executed by the physical robot (Fig. 1E).

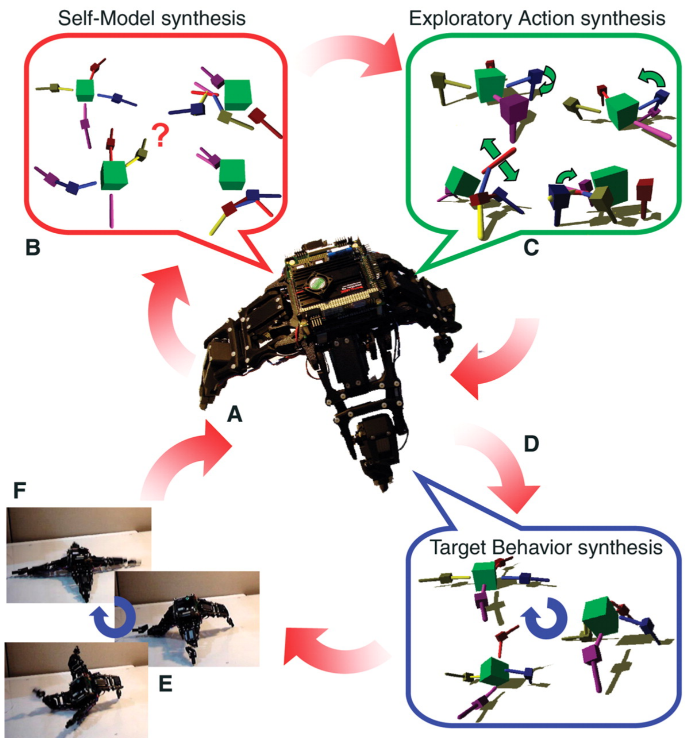


Figure 14: Start at A, cycle A-C around 16 times to build the model. Evaluate walking in simulation, Best simulation is implemented on the robot. FromBongard, Zykov, and Lipson 2006bfig 1

Part V some other compensatory gaits,

video:Bongard and Lipsen Starfish self modelling (`https://www.youtube.com/embed/ehno85yI-sA`)
video:Bongard and Lipsen Starfish locomotion (`https://www.youtube.com/embed/x579QKA6fkY`)

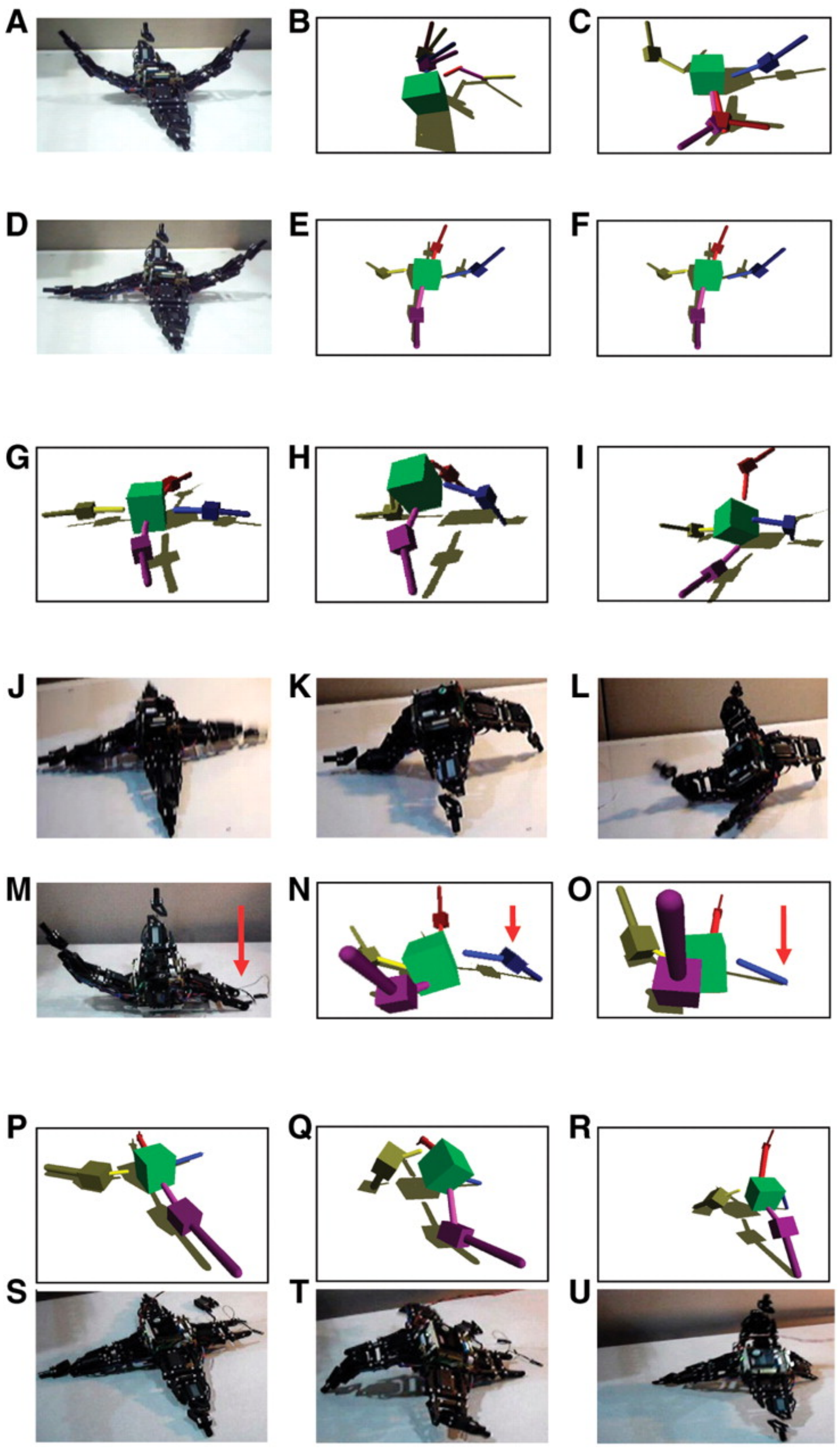


Figure 15: A and D try random actions, B,C,E,F try to fit a model. G,H,I try to walk in simulation. J,K,L try the walk on the robot. M Inflict damage, N,O, large errors therefore re-initiate modelling (this time only the limb lengths get changed), P,Q,R evaluate a new gait on model, S,T,U evaluate new gait on robot. FromBongard, Zykov, and Lipson 2006bfig 2

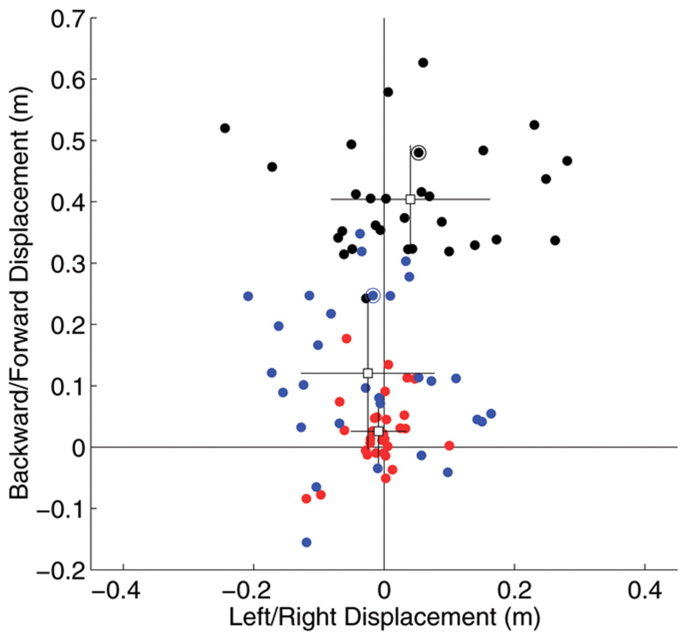


Figure 16: Evaluation of the ability the robot has to use its internal represent to survive in the real world. Red dots = random behaviour, Black dots = evaluation of movement from internal model, Blue dots = actual movement. FromBongard, Zykov, and Lipson 2006bfig 3

18 Machine Intelligence

Intelligence types, (intelligence is what we decide it is!)

- Play chess
- Infer answers. Turing test, chatbots, 20 Questions games, http://20q.net/
- Recognise patterns

18.1 State-of-the-art

- Supervised learning (trained with known classes) (classification and regression)
  - Linear classifiers - decision boundaries are linear weighted sum of inputs
  - Decision trees - decision boundaries are a hierarchical set of binary linear classifiers
  - K nearest neighbours
  - Probability methods
    - Naive Bayes
    - Hidden Markov models
  - Genetic algorithms
  - Multilayered perceptrons
    - Radial basis functions
    - Deep learning
    - Backpropogation neural networks
  - Support vector machines (SVMs)
- Unsupervised learning
  - k-means clustering and cluster analysis
  - Kohonen nets/ self organising maps
- Reinforcement learning

  - Q learning
  - Monte Carlo methods

18.2 XOR example

Predictor 1	Predictor 2	Response
0	0	0
0	1	1
1	0	1
1	1	0

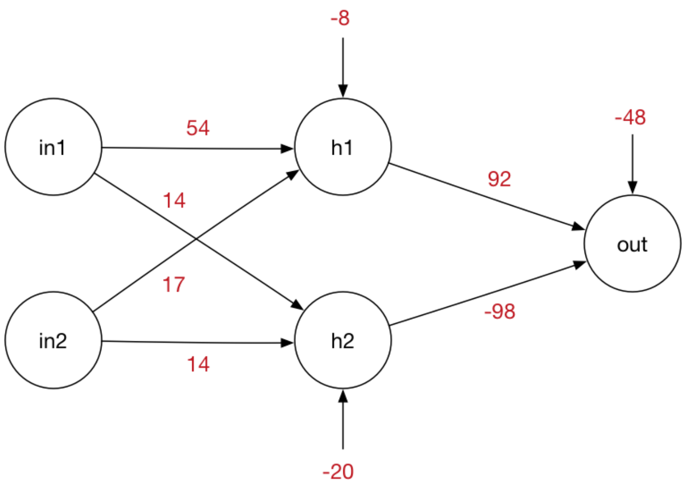


Figure 17: XOR neural net

(http://machinethink.net/blog/the-hello-world-of-neural-networks/)

The xor problem can be solved with a neural network with one hidden layer of two nodes and one output node.

The traditional non-linear activation function is the sigmoid function

$$S(x) = \frac{1}{(1 + \exp(-x))}$$

Each hidden node is computed as the linear sum of the inputs (including a constant input of 1), and the weights $h_{1\text{in}} = w_1^T \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \\ 1 \end{bmatrix}$ and $h_{2\text{in}} = w_2^T \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \\ 1 \end{bmatrix}$

These two equations can be combined into a single matrix calculation for the hidden layers

$$\begin{bmatrix} h_{1\text{in}} \\ h_{2\text{in}} \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \end{bmatrix} \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \\ 1 \end{bmatrix} = W_2 \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \\ 1 \end{bmatrix}$$

The outputs of the two hidden layers are thus

$$H_{\text{out}} = S\left(W_2 \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \\ 1 \end{bmatrix} \right)$$

Repeating for the output layer gets

$$\text{OUT} = S\left(W_1 \begin{bmatrix} H_{\text{out}} \\ 1 \end{bmatrix}\right)$$

This can be done in matlab as

```
>> S=@(x) 1./(1+exp(-x)); % set up the sigmoid function
>> W2=[54 14 -8;17 14 -20]; % Hidden layer weights
>> W1=[92 -98 -48]; % output layer weights
>> in=[0;0;1]; % trial input (0,0) % in=[1;0;1]; trial input (1,0)
>> H1=S(W2*in); % compute hidden layer outputs
>> Zm=S(W1*[H1;1]); % compute output layer
```

18.3 Neural network surface to solve the XOR problem

The following gives one of many surfaces that can solve the xor problem with a suitable threashold.

18.4 Evaluation

18.4.1 Holdout

- Split data into a training and a test set
- Use models from trained data to assess accuracy of test set

18.4.2 k-fold, (good for small data problems)

- Partition data into k sets
- For each i in k, train on the data excluding set i
- Test on set i
- Aggregate the results

18.4.3 Resubstitution

- Train on all the data
- Test on all the data

18.4.4 Binary classifications

Consider a test with two outcomes, (e.g. True/false, positive/negative, 1/0, yes/no) that tries to measure a hidden state.

Example: Suppose we have a test for a disease and use it to test an individual. That person may or may not have the disease, and the test tries to give a positive result if the person has the disease and negative if not. There are four post test (posterior) possibilities.

- True positive (TP): The test is positive for the disease and the individual has the disease :-)
- True negative (TN): The test is negative for the disease and the individual does not have the disease :-)
- False positive (FP): The test is positive for the disease and the individual does not have the disease :-(
- False negative (FN): The test is negative for the disease and the individual has the disease :-(

If we apply this test across a population, and we have a subsequent confirmation of the underlying truth we can compute these as rates, ie the percentage of individuals in each category.

> For another example, consider a pregnancy test. At the time of the test there is a level of uncertainty as to whether the person might be actually be pregnant. Only once there is confirmation from a more reliable test later on can the correct state be established. These could be a combination of factors (able to hear a foetal heart beat, see a baby in an ultrasound scan, no birth, no medical indication of a foetus)

Several measures of the quality of the test are possible two in particular are

High specificity means that if the test is positive the actual state is likely to be positive.
High sensitivity means that if the test is negative the actual state is likely to be negative.

Sensitivity (a value expected to be between 0.5 and 1) is identical to the True Positive Rate (TPR)
Specificity (also a value expected to be between 0.5 and 1) is identical to the True Negative Rate (TNR)

		Predicted class			
		Predicted positive	Predicted negative	TPR	NPV
True class	Actually positive	A	d	$\frac{A}{A+d}$	$\frac{B}{B+d}$
	Actually negative	f	B		
	PPV	$\frac{A}{A+f}$			
	SPEC	$\frac{B}{B+f}$			

		Predicted Class			
		Predicted positive	Predicted negative		
True class	Actual positive	TP (A)	FN (d)	TPR (Sensit)	FNR
	Actual negative	FP (f)	TN (B)	FPR	TNR (Specif)
		PPV	FOR		
		FDR	NPV		

A is the number of correct predictions that an instance is positive. **True positive**
d is the number of incorrect of predictions that an instance is negative. **False negative** (Type II error)
f is the number of incorrect predictions that an instance is positive. **False positive** (Type I error)
B is the number of correct predictions that an instance is negative. **True negative**

Total population (number of tests) is $A + B + d + f = \Sigma$
TPR, True positive rate, also called **Recall**, **Sensitivity**, **Probability of detection** . Likelihood of a correct (true) positive prediction given the state is actually positive TRP = $\frac{A}{A+d}$
TNR, True negative rate, Specificity (**SPEC**). Likelihood of a correct prediction of *negative* or *no* given that the state is negative. SPC = 1 − FPR = $\frac{B}{B+f}$
FPR, False Positive Rate. **Fall out**, **Probability of false alarm** : Likelihood of a (false) positive prediction given the truth is negative FPR = 1 − SPEC = $\frac{f}{B+f}$
Accuracy : (ACC) Likelihood (Probability) of a correct prediction $ACC = \frac{A+B}{\Sigma}$
Misclassification rate : Likelihood of a false prediction = $\frac{d+f}{\Sigma} = 1 - ACC$
Precision : Likelihood of a correct (true) positive state given a positive prediction = $\frac{A}{A+f}$
Prevalence : Likelihood of *yes* in the sample = $\frac{A+d}{\Sigma}$
FOR : False omission rate 1-NPV
NPV : = $\frac{B}{B+d}$
PPV : Positive predictive value, also positive predictive rate. $\frac{A}{A+f} = \frac{TP}{(TP+FP)}$
NPV : Negative predictive value. NPV = $\frac{B}{d+B} = \frac{TN}{(TN+FN)}$

TPR	sensitivity, recall, power, probability of detection	1-FNR
FPR	false alarms, type I errors, fall-out, prob of false alarms	1-SPEC
TNR	SPEC (specificity)	1-FPR
FNR	type II errors	1-TPR

18.4.5 Multiclass confusion matrix

Machine learning techniques tend to have large numbers of classes with the success rates tending to fall as the number of classes increases. For example in trying to recognise hand writing it may be easy to distinguish a written O and a X, it may be harder to identify O against letters such as D, P, B, Q, R.

The following example is a confusion matrix that tries to classify a person's activity into three categories, sitting, standing and supine.

This is a broad generalisation that may fail if the person is not sitting, not standing and not supine (perhaps they are walking, jumping, hopping, prone, doing head stands, etc).

It may also fail because the sensor has a poor 'view' of the posture, e.g. a wrist mounted IMU such as those found in fitness devices and smart watches.

It may also fail because the classification algorithm is not very good.

		Predicted class				
		Sitting	Standing	Walking	TPR	NPV
True class	Sitting	A	d	e	$\frac{A}{A+d+e}$	$\frac{B+C+g+j}{(B+C+g+j)+d+e}$
	Standing	f	B	g		
	Walking	h	j	C		
	PPV	$\frac{A}{A+f+h}$				
	SPEC	$\frac{B+C+g+j}{(B+C+g+j)+f+h}$				

If A,B,C,d,e,f,g,h,j are the numbers in each category then for each of the (true) classes we can calculate

TPR, Recall, Sensitivity : The True positive rate. For example, for the true class 'Sitting' the TPR is $\frac{A}{A+d+e}$ (Each row value is divided by the row sum) and similar for the other classes

NPV : Negative predictive value. Measures how often 'not in the class' is correctly predicted and is ideally near 1. For Sitting the NPV = $\frac{B+C+g+j}{d+e+B+C+g+j}$

For each predicted class we can calculate

PPV : The Positive predictive values or positive predictive rate. Similar to precision. The measure for the prediction 'Sitting' is $\frac{A}{A+f+h}$

Specificity :Given a prediction class, how often is it correct. For the class 'Sitting' SPEC=$\frac{B+C+g+j}{f+h+B+C+g+j}$

Accuracy : Total correct predictions over total number of predictions. This measure is subjective to class imbalance, i.e. one class can dominate the results.

(see Beleites, C.; Salzer, R. Sergo, V. Validation of soft classification models using partial class memberships: An extended concept of sensitivity & co. applied to grading of astrocytoma tissues, Chemom Intell Lab Syst, 122, 12 - 22 (2013). DOI: 10.1016/j.chemolab.2012.12.003)

How to calculate precision and recall in a 3 x 3 confusion matrix (`https://stats.stackexchange.com/questions/91044/how-to-calculate-precision-and-recall-in-a-3-x-3-confusion-matrix`)

Example 1 Imagine you are a consultant and your patient has just received a test result. Supose the result is positive, how confident should you and your patient be in that result. Equally suppose the result is negative, can this result be trusted?

Answer, look at PPV and FDR

Example 2 An E.M.G. controller for a prosthetic hand is used to classify the users intentions as 'open the hand' 'close the hand' 'lock'. The following confusion matrix has emerged from tests.

		Predicted class				
		open	close	lock	TPR	NPV
True class	open	47	2	1		
	close	1	42	7		
	lock	5	18	27		
	PPV					
	SPEC					

where

Table 1: Common notation for the 2-way binary matrix and for the case of 'sitting' in the 3-way confusion matrix

	2-way	3-way
Σ	$A+B+d+f$	$A+B+C+d+e+f+g+h+j$
TPR	$\frac{A}{A+d}$ (Prevalence)	$\frac{A}{A+d+e}$
NPV	$\frac{B}{B+d}$	$\frac{B+C+g+j}{(B+C+g+j)+d+e}$
PPV	$\frac{A}{A+f}$ (precision)	$\frac{A}{A+f+h}$
SPEC	$\frac{B}{B+f}$ FPR=1-SPEC	$\frac{B+C+g+j}{(B+C+g+j)+f+h}$
Accuracy	$\frac{A+B}{(A+B)+c+d}$ Misclassification rate = 1 - Accuracy	$\frac{A+B+C}{\Sigma}$

TPR= True Positive Rate
NPV= Negative Predictive Value
PPV= Positive Predictive Value
SPEC= Specificity

Calculate as a percentage, TPR, NPV, PPV, SPEC, and accuracy

Terminology

18.4.6 Receiver operating characteristic (ROC) curves

An example. We can either consider a two class classification problem, or a test of a population. A two class classification problem might be distinguishing ducks from geese based on their swimming speed. The test might be the size of a response in people with the disease, vs the size of a response in people who do not have the disease.

Figure 18 shows an example where the results have an equal variance but different mean values.

TP/ TPR indicates true positive of the test, or correct identification of (say) a duck.

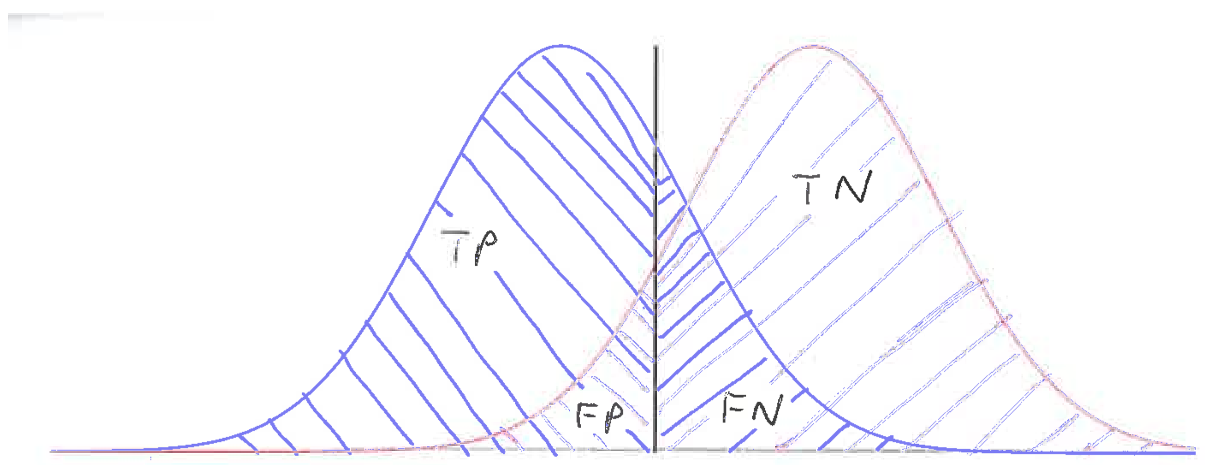
FN/ FNR indicates the rate that the test gives a negative result even though the person has the condition. Or incorrectly identifying a duck as a goose.

TN/ TNR indicates correctly identifying the person doesn't have the condition. Or correct identification of (say) a goose.

FP/ FPR indicates the rate that the test gives a positive result even though the person does not have the condition. Or incorrectly identifying a goose as a duck.

An ROC curve plots, for each class, a FPR (false positive rate, i.e. 1-TNR or 1-specificity) on the horizontal axis, and a TPR (true positive rate/sensitivity) on the vertical axis, for differing threshold levels of acceptance.

Good performance are indicated by high TPR and low FPR so can be seen by how close the curve approaches the top left corner (0,1). Chance results are indicated by points on a 45 degree line



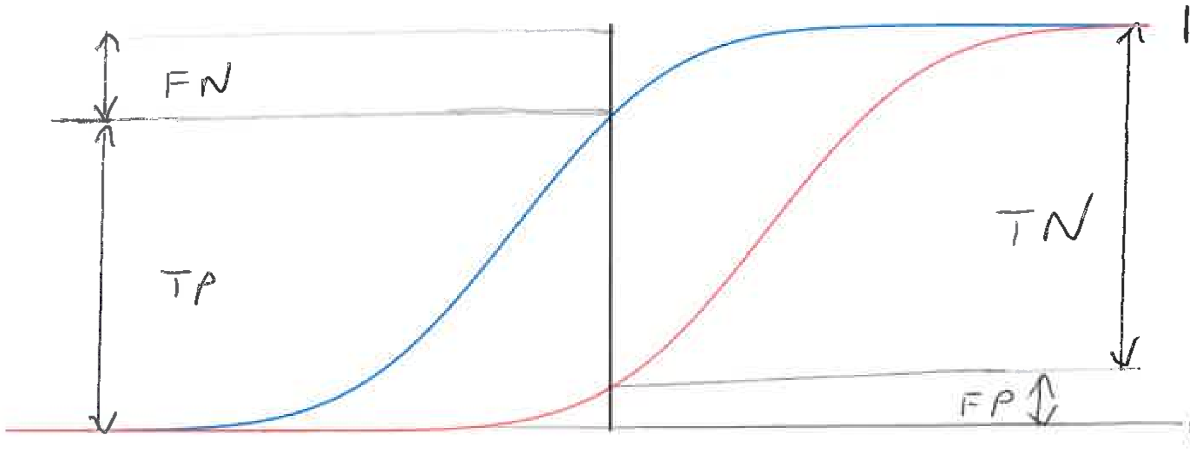


Figure 18: Upper diagram: pdf for binary classifier. Lower diagram: cdf for binary classification

between (0,0) and (1,1). Thus the area under the curve can be considered as an overall measure of the quality of the classifier.

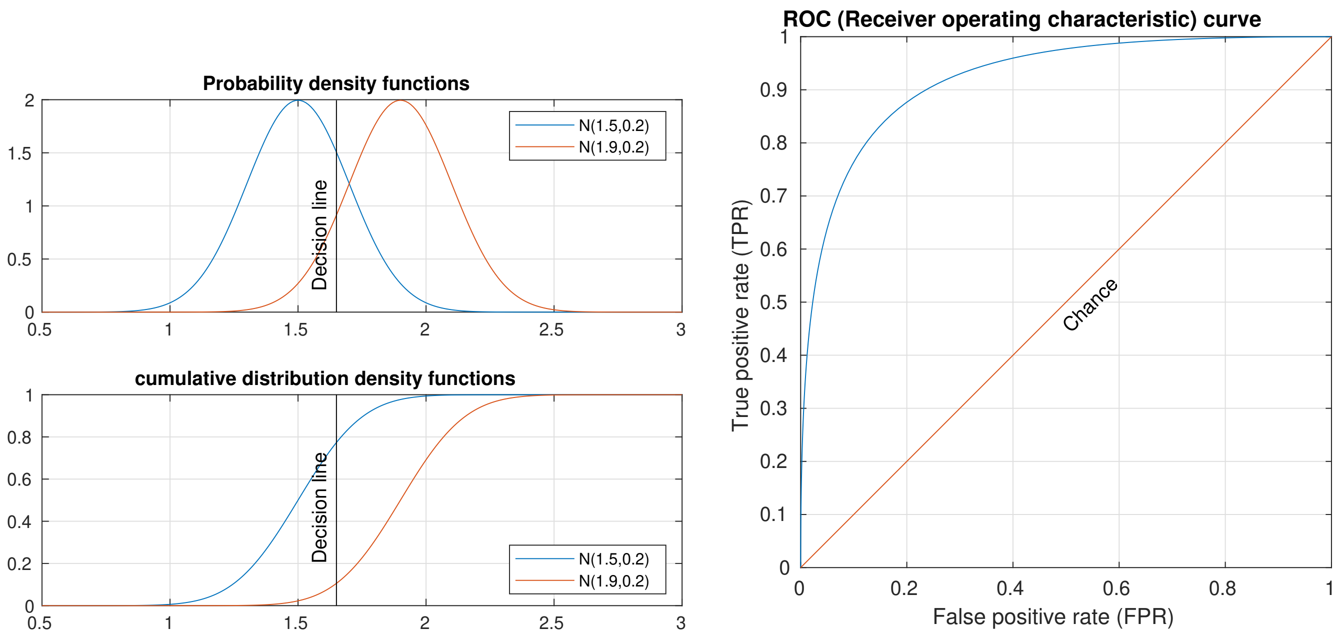


Figure 19: Distributions and ROC curve for two varaibles with identical variance. The location of the decision line gives a trade off between getting the decision right (TPR), and not getting the decision wrong (FPR)

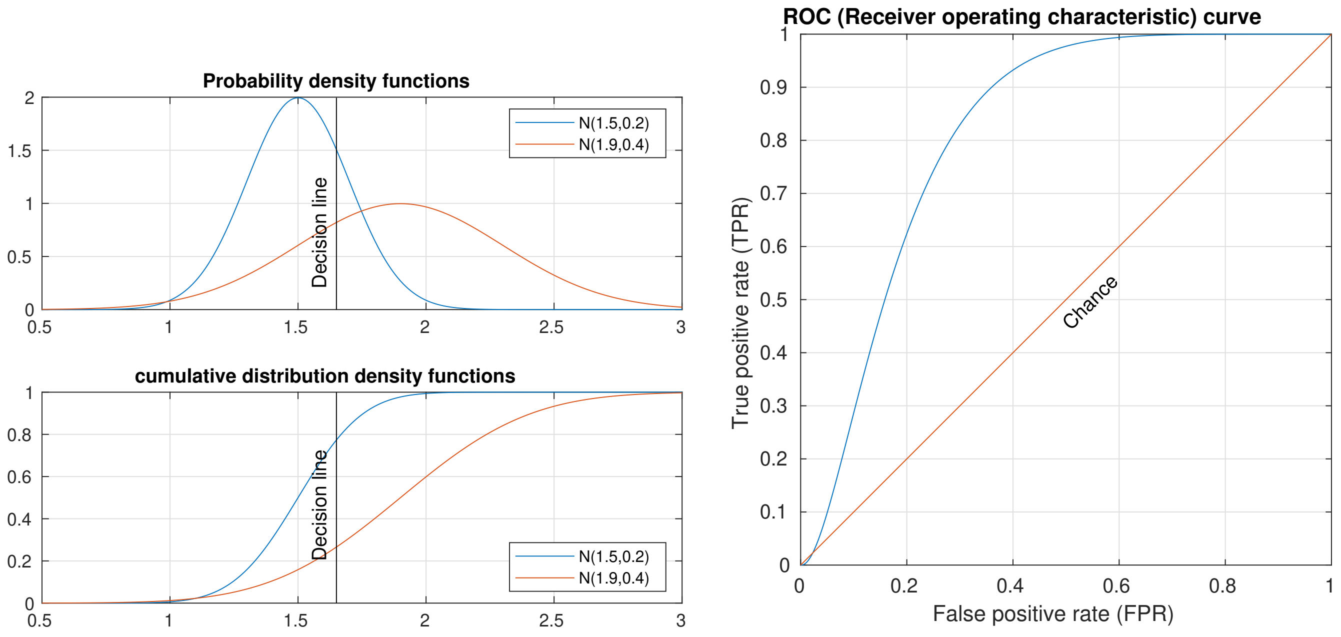


Figure 20: Distributions and ROC curve for two varaibles with different mean and variance. CDF provides an easy way to compute the ROC curve.

19 Linear classifier

Problems

- How to distinguish people and pets
- How to tell cats from dogs
- How to classify heart sounds

19.1 Normal distribution

The normal distribution is an idealisation of a bar chart that gives the ratio of things in each ¡it¿bin.¡noit¿ For many experiments, as the *bins* get smaller the histogram look more and more like a Normal (Gaussian) distribution.

μ	mu
σ	sigma
Σ	Sigma
$P(a < x < b)$	probability that an event that depends on x lies between the values a and b

- It is a compact representation of data
- Used to describe a population (of things, animals, test results etc)

Examples that might have a normal distribution

- weight of quarter pound hambergers
- length of cats, height of people
- the total when throwing n dice (as n tends to infinity $\mu = 3.5n$, $\sigma^2 = \frac{17.5}{6}n$)

Counter examples

- result of two coin tosses
  - Side problem (Monty Hall problem)

19.2 Probability density functions

The probability that some variable X lies between a and b is

$$P(a < X < b) = \int_a^b f(x)dx$$

where $f(x)$ is the probability densityfunction. In interesting problems the probability $P(-\infty < x < \infty) = 1$, ie the event must occur at some point.

For a normal distribution

$$\mathcal{N}(\mu, \sigma^2) = f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{27}$$

Where μ is the mean (sometimes called the expected value) and σ is the standard deviation.

or in vector form

$$\mathcal{N}(\underline{\mu}, \underline{\mathbf{\Sigma}}) = \frac{1}{\sqrt{(2\pi)^k |\underline{\mathbf{\Sigma}}|}} e^{\left(-\frac{1}{2}(\underline{\mathbf{x}}-\underline{\mu})^{\mathrm{T}} \underline{\mathbf{\Sigma}}^{-1} (\underline{\mathbf{x}}-\underline{\mu})\right)}$$

The matrix $\underline{\mathbf{\Sigma}}$ is the symmetric covariance matrix. If the vector has only one element, then we work out that $\underline{\mathbf{\Sigma}}$ would be $2\sigma^2$.

19.3 Example (1D) Pets and people

How to classify people and pets. Guess at means and standard devisions

missing image (tex) `lc11_02.png`

Figure 21: Algorithm to distinguish pets from people

```
>> muppl=1.700; sigppl=.400; % mean and standard deviation of people (a guess)
>> mupets=.300; sigpets=.100;  % mean and standard deviation of pets (a guess)
```

First create some data (10 people and 10 pets)

```
>> people=randn(10,1)*sigppl+muppl;
>> pets=randn(10,1)*sigpets+mupets;
>> ht=1; plot(pets,ht*ones(size(pets)),'x',people,ht*ones(size(people)),'o')
>> legend({'pets','people'}); xlabel('length (m)'); pause
```

How does this look as a propability density function (PDF)?

```
>> x=0:max(people)/100:max(people);
>> Npeople=exp(-(x-muppl).^2/(2*sigppl^2))/sqrt(2*pi*sigppl^2);
>> Npets=exp(-(x-mupets).^2/(2*sigpets^2))/sqrt(2*pi*sigpets^2);
>> plot(pets,ht*ones(size(pets)),'x',people,ht*ones(size(people)),'o',x,Npets,x,Npeople)
>> legend({'pets','people'}); xlabel('length (m)');ylabel('probability density');pause
```

The probability is the integral of the PDF. We can plot the cumulitive probability function cdf(x) as the probability that the event has occured somewhere between $-\infty$ and x

```
>> plot(pets,ht*ones(size(pets)),'x',people,ht*ones(size(people)),'o',x,cumsum(Npets),...
>>  x,cumsum(Npeople))
>> xlabel('length (mm)'); ylabel('probability pet/person is less than height x');pause
```

19.4 Example (2D) Iris species

Using the Fisher Iris data set[10] on the petal and sepal widths and lengths for three species of iris, *setosa, versicolor* and *virginica.*

Can we identify an algorithm to classify the species based on measurements of two parameters?

The `meas` columns are sepal width, sepal length, petal width, petal length. Species are in the order

```
>> setosa=1:50;versi=51:100;virg=101:150;
>> t = readtable('fisheriris.csv','format','%f%f%f%f%C');
>> meas=t{:,1:4};
>> plot(meas(setosa,3),meas(setosa,4),'x',meas(versi,3),meas(versi,4),'o',meas(virg,3),...
>>  meas(virg,4),'d')
>> legend({'setosa','versicolor','virginica'},'Location','southeast')
>> title('petal width vs petal length');xlabel('petal width (cm)');
>> ylabel('petal length (cm)'); pause
```

[10] R.A. Fisher, Use of multiple measurements in taxonimic problems, 1936

A linear classifier needs to have the class means of each group. These are sufficient to attempt a classification. While we are calculating the means we can also calculate the covariance.

```
>> muset=mean(meas(setosa,:)); covset=cov(meas(setosa,:));
>> muversi=mean(meas(versi,:)); covversi=cov(meas(versi,:));
>> muvirg=mean(meas(virg,:)); covvirg=cov(meas(virg,:));
>> muall=[muversi;muset;muvirg];
```

We can use the Voronoi method to work out boundaries based on means (linear discriminant analysis)

```
>> plot(meas(setosa,3),meas(setosa,4),'x',meas(versi,3),meas(versi,4),'o',meas(virg,3),mea
>> hold on; voronoi(muall(:,3),muall(:,4)); hold off; pause
```

Can see how well it does on other pairs, e.g. sepal length and petal length

```
>> plot(meas(setosa,2),meas(setosa,4),'x',meas(versi,2),meas(versi,4),'o',meas(virg,2),mea
>> hold on; voronoi(muall(:,2),muall(:,4)); hold off;
>> legend({'setosa','versicolor','virginica'})
>> xlabel('sepal length (cm)'); ylabel('petal length (cm)');pause
```

missing image (tex) `lc11_06.png`

Figure 22: Best linear boundaries between classes

Probably easiest to use libraries and programmes such as Matlab to extend this to more than 2 dimensions. Matlab has a linear classifier fitcdiscr that does much of the work needed.

```
>> d = fitcdiscr(t,'Species');
>> [cm,order]=confusionmat(t.Species,predict(d,t))
```

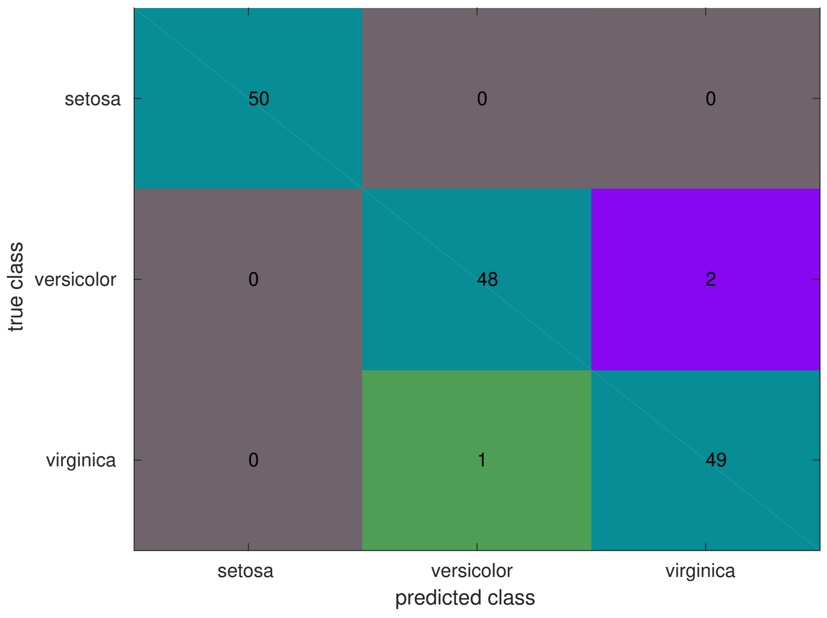


Figure 23: Confusion matrix for a linear classifier applied to Fisher Iris data

20 Decision trees

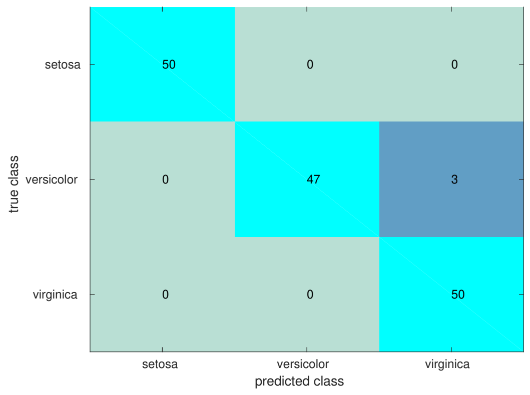
- Part of machine learning/data mining
- Computes a set of rules
- Related to stream data mining where rules are constantly reconsidered (Fred Stahl)
- Used for classification and regression
- Benefit that the rules are understandable by humansSanjeevi 2017
- Can be applied to categorical data

Training done by computing the gain in entropy (information gain) and choosing a rule that results in the highest information gain. This process is then applied at the next level of the tree until there is no information gain if a new rule is introduced.

A decision tree for the Fisher iris data can be computed with the Matlab *fitctree* command. Using all measurements as predictors we get the following rules and results.

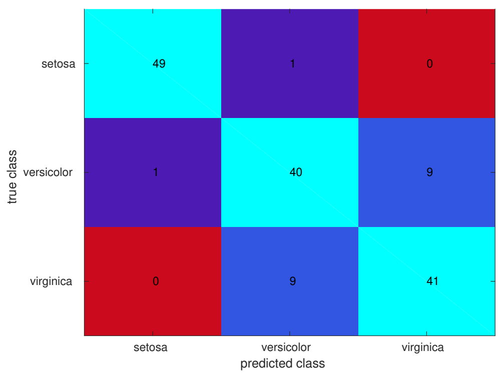
```
    Decision tree for classification
    1  if PL<2.45 then node 2 elseif PL>=2.45 then node 3 else setosa
    2  class = setosa
    3  if PW<1.75 then node 4 elseif PW>=1.75 then node 5 else versicolor
    4  if PL<4.95 then node 6 elseif PL>=4.95 then node 7 else versicolor
    5  class = virginica
    6  if PW<1.65 then node 8 elseif PW>=1.65 then node 9 else versicolor
    7  class = virginica
    8  class = versicolor
    9  class = virginica
```

```
>> t4 = fitctree(meas, species,'PredictorNames',{'SL' 'SW' 'PL' 'PW'});
>> [cm4,order4]=confusionmat(species,predict(t4,meas))
>> drawconfusion(cm4,[],order4)
>> view(t4,'mode','graph')
```



missing image (tex) dt4a

Figure 24: Confusion matrix and Decision tree for Fisher Iris data predicting from petal length (PL) and petal width (PW)



missing image (tex) `dt2.png`

Figure 25: Confusion matrix and decision trees for Fisher Iris predicting from sepal length (SL) and sepal width (SW)

Potential issues with decision trees include

- Sensitive to the data change
- Can result in excessively complex sets of rules. (pruning needed)
- Could be adapted to allow humans to introduce rules

21 Least squares recap

- Fits a linear model to data
- Model is of the form $y = w_0 + w_1u_1 + w_2u_2 ... w_nu_n$ for a model with one output and several inputs
- With multiple measurements of y and u we can form a matrix equation of $Y = U\underline{\mathbf{w}}$ where Y contains all the output (response) values, U contains all the input (predictor) values, and $\underline{\mathbf{w}}$ contains all the weighting.
  - One column of U can be set to 1 to enable estimation of w_0
  - If there are l exemplars then $Y_{l\times 1} = U_{l\times n}\underline{\mathbf{w}}_{n\times 1}$
- A least squares solution is then $\underline{\mathbf{w}} = (U^TU)^{-1}U^TY$
  - Check this with the matrix dimensions (the inverse must be a square matrix)

21.1 A method to find the best values of $\underline{\mathbf{w}}$

$$SSE = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T\mathbf{y} - 2\underline{\mathbf{w}}^TU^T\mathbf{y} + \underline{\mathbf{w}}^TU^TU\underline{\mathbf{w}}$$

SSE has the minimum when

$$\frac{\partial(SSE)}{\partial\underline{\mathbf{w}}} = \mathbf{0}$$

So first differentiate SSE with respect to $\underline{\mathbf{w}}$
Note:

- Differentiating matrix variables is much like differentiating scalar variables but retaining the matrix orders.
- Error is a scalar value so that the things added together after differentiation must also be a scalar

$$\begin{aligned}\frac{\partial(SSE)}{\partial\underline{\mathbf{w}}} &= -2[\frac{\partial(\mathbf{y}^TU\underline{\mathbf{w}})}{\partial\underline{\mathbf{w}}}]^T + \frac{\partial(\underline{\mathbf{w}}^TU^TU\underline{\mathbf{w}})}{\partial\underline{\mathbf{w}}}\\ &= -2U^T\mathbf{y} + 2U^TU\underline{\mathbf{w}}\\ &= -2U^T(\mathbf{y} - U\underline{\mathbf{w}}) = \mathbf{0}\end{aligned}$$

yielding

$$\hat{\underline{\mathbf{w}}} = [U^TU]^{-1}U^T\mathbf{y}$$

21.2 Another method to find the best values of $\underline{\mathbf{w}}$ (Singular value decomposition)

In 1965 Gene Golub[11] published an efficient decomposition of any matrix into its Singular values. That is for any matrix X there are three matricies that can be used to recreate it of the form

$$X = U_uDV^T$$

with the properties that D is diagonal, and U and V are unitary, which means that $U_u^TU_u = I$ and $V^TV = I$ We are looking for a solution to the over determined equation

$$Y = X\underline{\mathbf{w}}$$

[11]search for 'Professor SVD' by Cleve Moler 2006

note: since svd maths uses U to indicate a unitary matrix, we are temporarily using X to represent our matrix of input values

So

$$\begin{aligned}
Y &= U_u D V^T \underline{\mathbf{w}} \\
U_u^T Y &= D V^T \underline{\mathbf{w}} \\
D^{-1} U_u^T Y &= V^T \underline{\mathbf{w}} \\
V D^{-1} U_u^T Y &= \underline{\mathbf{w}}
\end{aligned}$$

Since the inverse of a diagonal matrix is simply the inverse of its elements, the singular value provides another mechanism to determine the best parameters in $\underline{\mathbf{w}}$ to use for the linear model.

Matlab provides a function to compute these three matrices, see

```
>> help svd
```

22 Multilayer perceptrons

The concept of an artificial neuron first explored by Warren McCulloch and Walter Pitts in 1943McCulloch and Pitts 1943.

They assumed neurons were

- either firing (1) or at rest (0)
- had multiple inputs and a single output
- inputs to the neuron were weighted
  - positive values could be considered excitatory
  - negative values could be considered inhibitory

missing image (tex) `img18.jpg`

Other representations of neurons

- Spiking ANNs (artificial neural networks)
- Realistic models
  - Hodgkin Huxley models
- Leaky integrate and fire
- Neural field theory (Neurons as a continuum)

MLP's consider these neurons as a succession of layers (possibly akin to the layers in the neocortical brain (5-6 layers))

22.1 Non-linear output functions

Key concept are the non-linear output elements. They allow complex decision boundaries to be established.

- Often seeking functions that have a finite bound on their output, e.g. has a value of 1 as input gets very large.

22.1.1 Step (also called the Heavyside function)

Output Range $0 \leq f(x) \leq 1$ (finite bounds)

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

22.1.2 Sigmoid

Output Range $0 \leq f(x) \leq 1$ (finite bounds)

$$f(x) = \sigma x = \frac{1}{1 + e^{-x}}$$

differential is $\frac{d}{dx}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x))$

22.1.3 Linear

Output Range $-\infty \leq f(x) \leq \infty$

$$f(x) = x$$

This is evidently the only linear output function. Often used when the MLP needs to give an output over a large range of values.

22.1.4 Relop

Output Range $0 \leq f(x) \leq \infty$ (finite bound as input tents to negative infinity)

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Easy to calculate so widely used in deep learning systems

For example in tortured c this can be written as

```
f=x<0?0:x
```

22.1.5 Gaussian

Output Range $0 \leq f(x) \leq 1$ (finite bounds)

$$f\left(x\right) = e^{-(\varepsilon x)^2}$$

Use in Radial Basis Function (RBF) networks. Note the similarity to the probability density function of a normal distribution (see Linear classifiers equation 27)

22.1.6 Hyperbolic tangent

Output Range $-1 \leq f(x) \leq 1$ (finite bounds)

$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Differential is $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$

Other functions

- error functions (differentiate to get a Gaussian)
- piece wise polynomials,
- rectifier function

22.2 Neuron (mathematical) function

If the neuron has n inputs, and for a particular evaluation each input has a value u_i then the neuron output y is calculated as

$$y = f\left(\sum_{i=1}^{n} w_i u_i\right)$$

There are n weights each with a value w_i, and the function $f(.)$ is often the sigmoid.

This is best considered as two parts

1. Compute the sum of the weighted inputs.
2. Apply the appropriate non-linear output function $f(x)$

The neuron can also be expressed as a matrix calculation. If there are m records we wish to show to the neuron, to compute m outputs we can create a vector $\underline{\mathbf{y}}$ that is $m \times 1$ (m rows and 1 column). Then for m different examples of the n inputs we can form an $m \times n$ matrix U and a vector of weights

$$\underline{\mathbf{w}} = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}$$

The calculation then becomes

$$\underline{\mathbf{y}} = f\left(U\underline{\mathbf{w}}\right)$$

22.2.1 Constant offsets

We need a constant offset. (a.k.a. fitting a line to some points as $y = mx + c$)

We can either put this in explicitly i.e. as a sum

$$y = f\left(\sum_{i=1}^{n} w_i u_i + c\right)$$

and as a matrix calculation

$$\underline{\mathbf{y}} = f\left(U\underline{\mathbf{w}} + \underline{\mathbf{c}}\right)$$

Alternatively we can simply add one more input to the neuron and set the input to that node as a constant value (e.g. 1).

Mostly we will do the latter.

22.3 Example: Logic functions implemented as ANN

Logic gates revision: AND, OR, NAND, NOR, XOR, XNOR

Example: build an artificial neuron to do the AND function.

$$\underline{\mathbf{y}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, U = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

22.3.1 What weights should we use?

Lets assume the sigmoid non-linear function, then any value over say 4 will map to something near 1, and any value less than 4 will map to 0. We can find the 'best least squares' fit to this data from the linear equation

$$\underline{\mathbf{s}} = \begin{bmatrix} -4 \\ -4 \\ -4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \underline{\mathbf{w}}$$

where $\underline{\mathbf{s}}$ is the output after the summation. Note that the right most column is the column that is always set to 1 to provide the neuron with an offset.

First learn the weights

```
>> U=[0 0 1;0 1 1; 1 0 1; 1 1 1]
>> s=[-4 -4 -4 4]'
>> w=inv(U'*U)*U'*s
```

Then test the network

```
>> sig=@(x) 1./(1+exp(-x));
>> sig(U*w)
```

22.4 A least squares solution

The above uses a useful equation, the least squares solution to a linear model.

Used in 'over determined solutions' e.g. linear regression.

If we have the error between what we would like and what we get as

$$\underline{\mathbf{e}} = \underline{\mathbf{y}} - U\underline{\mathbf{w}}$$

We can compute the overall magnitude of the error as $\sum e^2 = \underline{\mathbf{e}}^T\underline{\mathbf{e}}$ If we can find the differential of this function with respect to the weights $\underline{\mathbf{w}}$ we can then find the values of $\underline{\mathbf{w}}$ where the error is a global minimum.

22.4.1 Questions

What if there is some variability about the inputs? Perhaps we have data of the form

u_1	u_2	t
0	.1	.1
.15	-.1	0
.7	.8	1.1
.8	-.1	.05
0.3	.99	.89
.4	.5	.5

What about xor? - we can't get the equations to work!

We need to include at least one hidden layer of neurons.
(Deep networks may have 9 or more hidden layers)

23 Training MLPs

Now almost any network uses the back propagation algorithm[12]. Algorithm adapts the weights so as to reduce the classification errors. (Often called gradient descent)

Algorithm has two parts

- Forward propagation of inputs through the network to compute the error between the predicted and the desired output.
- Back propagation of these errors allowing weights at each layer to be adjusted.
- This requires that the two parts of the node (input summation and sigmoid function) can be differentiated. (this makes the Heaviside function unusable for this mechanism of training.)

Simulated Annealing is an alternative that would allow non-differentiable non linearities, but it is less efficient when compared to the back propagation algorithm.

24 Common acronyms

MLP	Multilayer perceptron
ANN	Artificial neural network
RBF	Radial basis function

25 foo

26 Feature creation and selection

Machine learning and statistics considers

inputs	predictors	independent variables
outputs	responses	dependent variables

output variables are usually either *categorical* or *regressors*

Features may be needed for machine learning when

- there are limited examples of data
- there is a need to understand the data
- there is a potential to utilise knowledge about the data

Sometimes this is called ‘shallow learning’.

Feature selection can reduce dimensionality of data.

Features should not be

- Redundant: i.e. duplicates information or has a high correlation with other features
  - e.g. Annual income and tax code, shoe size and foot length.
- Noisy: i.e. other non relevant information overpowers the information
- Irrelevant: contain little or no pertinent information.
  - e.g. Your student number is not a good predictor of your exam result.

[12]David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1986a). “Learning internal representations by error propagation”. In: *Parallel distributed processing : explorations in the microstructure of cognition*. Ed. by James L. McClelland David E. Rumelhart. Vol. 1. MIT Press. Chap. 8; David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science. URL: `http://www.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf`; David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1986b). “Learning representations by back-propagating errors”. In: *nature* 323.6088, p. 533.

26.1 Examples

26.1.1 static

- Images: ideally features that are invariant to lighting, object rotation
  - Image recognition may want invariants on some facial features (wearing spectacles, sporting a moustache, makeup) and not others (eye colour, ear shape)
  - Viola and Jones 'Robust real-time object detection' 2001, proposed a large feature space for face recognition, algorithm widely used in cameras, Facebook, etc
- Diagnosis or estimations of medical risk

26.1.2 time varying

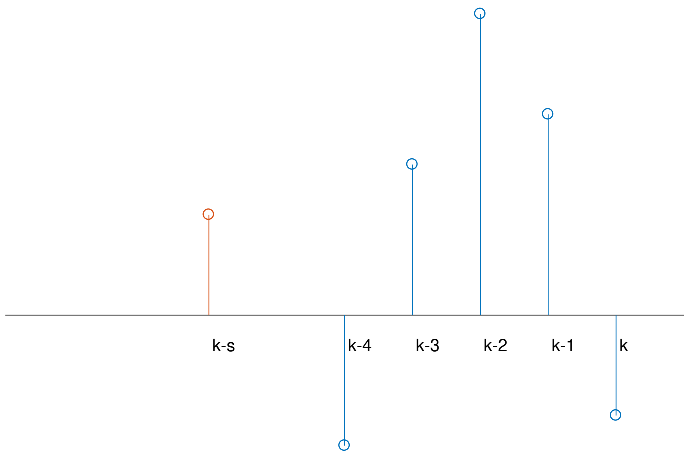
- Speech: Features might determine 'voiced' and 'unvoiced' speech, the response might be the text versions of the words spoken.
- Inertial measurements: Features might be signal statistics, outputs might be activities
- Netflix recommendations/Amazon 'other people brought...'
- Financial data (stock markets, exchange rates etc). Features might be the 3 month and 6 month price, responses might be 'buy/sell'

Note the remainder of these notes are for completeness. They will not be examined, but are relevent to the final lab.

27 Generalised digital filters

Interesting problems are time varying, we can use 'digital filters' to help us to create useful features.

- Almost all data either comes as a set of numbers, or can be converted to a set of numbers.
  - In electronics this is done with an *analogue to digital converter*
- We can consider digital data where we have knowledge at equally spaced points in time. (If this is not the case, we may be able to resample the data so that it is)
- If each sample is separated by Δ seconds then if we are at sample k when we get to time t we have $t = k\Delta$



- Mostly looking at filters that operate on the s previous values of the signal. This is sometimes called the filter span. This span usually refers to the previous input values `nb` (see below)
- We may want to only compute a value or a feature every m samples.
- It is not uncommon in machine learning to set $m = s$ or $m = s/2$

27.1 Example 1

Consider a digitised sin wave where

$$z_k = \sin(k\Delta\omega)$$

where ω is some constant value, (when Δ is in seconds, then $\omega = 2\pi f$ where f is the frequency in Hertz.

For example.

```
>> Delta=.1;omega=1*(2*pi);stem(0:Delta:2,sin((0:Delta:2)*omega));xlabel('time (s)')
```

We can construct a filter that is the differential of y, that is

$$y_k = (z_k - z_{k-1})\frac{1}{\Delta}$$

i.e. the change in y over change in time. so

$$y_k = \Big(\sin(k\Delta\omega) - \sin((k-1)\Delta\omega)\Big)\frac{1}{\Delta}$$

$$=$$

$$= \Big(\sin(k\Delta\omega)(1-\cos(\Delta\omega)) + \cos(k\Delta\omega)\sin(\Delta\omega)\Big)\frac{1}{\Delta}$$

So as Δ gets smaller we can get the small angle approximation to show that

$$\frac{y_k - y_{k-1}}{\Delta} \approx \frac{1}{\Delta}\frac{d}{dk} y_k = \cos(k\Delta\omega)\omega$$

28 Linear filters

The example above is an example of a linear digital filter. The general form of the filter is

$$\begin{aligned} y_k =& b_0 x_k + b_1 x_{k-1} + ... + b_{\mathtt{nb}} x_{k-\mathtt{nb}} \dots \\ & - a_1 y_{k-1} - a_2 y_{k-2} ... - a_{\mathtt{na}} y_{k-\mathtt{na}} \end{aligned}$$

where there are `nb` coefficients for the input x_k (and its previous values) and `na` coefficients for the previous values of y_k.

The filter defined above is causal since we can only compute y_k at time $k\Delta$ from previous values of y_k and the current and previous values of the input x_k

A special input called a digital impulse function is defined as

$$x_k = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{if } k \neq 0 \end{cases}$$

From this we can define two types of linear filter, FIR and IIR

28.1 Finite impulse response filters (FIR)

If we put a digital impulse into a FIR the output will go to zero after `nb` samples.

- Moving average
- Frequency response (freqz)

28.2 Infinite impulse response (IIR)

If we put a digital impulse into a FIR the output will get never get to zero.
In some cases the output will get smaller, and in some cases larger.

For the simple IIR $y_k = x_k - a_1 y_{k-1}$, under what conditions does y_k get larger/smaller?

- Which behaviour is preferred?

28.3 moving average (mean) filter

A common, simple, and non ideal filter is the weighted mean. If values are changing due to noise, then the general trend can be seen by computing a mean over the previous `nb` samples.

Finite impulse response filter

$$y_k = \left(x_k + x_{k-1} + ... + x_{k-\mathtt{nb}}\right)\frac{1}{\mathtt{nb}}$$

Infinite impulse response variants also possible, for example.

$$y_k = \frac{y_{k-1}}{2} + \left(x_k + x_{k-1} + ... + x_{k-\mathtt{nb}}\right)\frac{1}{2\mathtt{nb}}$$

28.4 WEIGHTED MOVING AVERAGE (MEAN) FILTER

In a moving average filter, to ensure that the filter does not scale the estimate of the original signal requires for the weights to sum to 1, so for example a weighted FIR moving average filter could be

$$y_k = 0.5x_k + 0.2x_{k-1} + 0.2x_{k-2} + 0.08x_{k-3} + 0.02x_{k-4}$$

You can see that the sum of coefficients is 1.

This has the advantage that more weight is given to recent values resulting in less lag between the true signal and the filtered signal.

29 NON LINEAR FILTERS

Non linear filters are widely used in machine learning. Although they loose the property that allows filters to be reordered without changing the result, they may have properties that work well for a particular data type and sometimes provide an intuative insite into the statistics of the signal.

Examples are

29.1 MEDIAN FILTER

$$y_k = \text{median}(x_k, x_{k-1}, ..., x_{k-\mathtt{nb}})$$

29.2 SIGNAL VARIANCE

If y'_k is the output of the moving average filter at sample k then the output of the signal variance filter is

$$y_k = \left((u_k - y'_k)^2 + (u_{k-1} - y'_k)^2 + ... + (x_{k-\mathtt{nb}} - y'_k)^2\right)\frac{1}{\mathtt{nb} - 1}$$

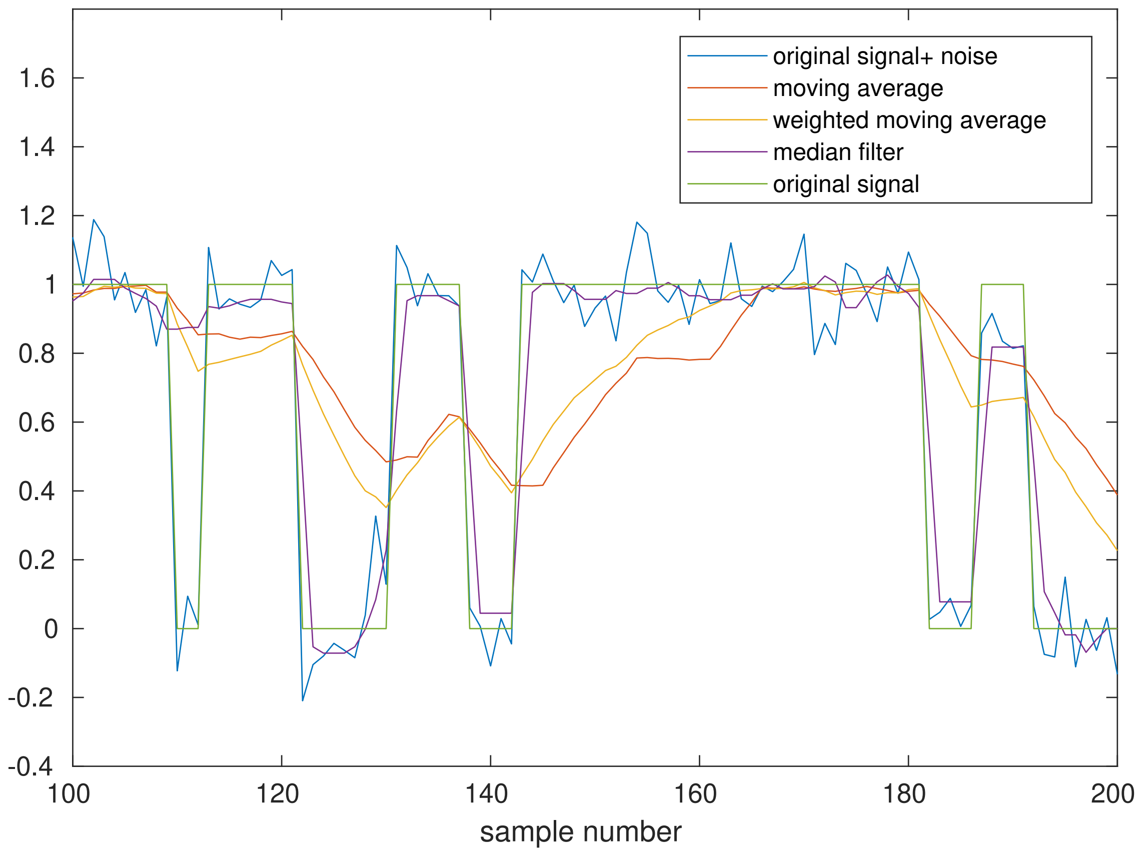


Figure 26: Filter responses for moving average, weighted moving average and median filters. The weighted moving average responds faster and more accurately when compared to the true mean moving average filter. The median filter responds even faster but is not a linear filter.

30 Data reduction

Computing multiple features on a time-series icreases dimensionality. To reduce the dimensionality, often the filter is not computed for every time sample, rather the algorithm waits m samples before recomputing the output.

Window length s,
Window overlap m

Most digital filters assume $m = 1$ Digital filters to compute classes for machine recognition may use $m = s$ (no overlap) or $m = s/2$ (50%) overlap.

2016 PhysioNet/Computing in Cardiology Challenge.

https://www.physionet.org/challenge/2016/

31 Heart signals

Electro cardio gram
https://www.physionet.org/challenge/2016/figure1.png https://archive.physionet.org/challenge/2016/figure
Validation set https://www.physionet.org/physiobank/database/challenge/2016/validation.zip
heartbeat electrocardiogram (ECG) data from the PhysioNet 2017 Challenge Download the structured dataset : https://www.dropbox.com/s/ilaofyb6h6m5sr6/ECGTable.mat?dl=0