Project 1

PHYS2G03

Project: Proposal Due October 22nd

1. Choose a research area of interest to you
2. Identify a system or object in that area that is modeled computationally
3. Find out how people model that system or object in practice
4. Try to find a simplified model for that system or object that still has the important characteristics of that system
5. Propose some numerical experiments to try with your model

Sources of Ideas

- Courses / Professors
- Science news articles
- Popular science magazines
- Popular science books
- Web pages

Keys to a good idea

1. Can you find a simple mathematical model for objects or systems in that research area? e.g. a system of 2-3 ODEs
2. Can you do numerical experiments with the model?
3. Can you identify features / behaviours of the object or system that the model should exhibit to test in your numerical experiments?

Good simple models

- Ideally: a small set of differential equations or a similar mathematical system that involves **updates**, e.g..

moving an object or two under gravity

molecules moving in a box

populations of animals: predator / prey

a population affected by disease: infected, susceptible, cured

Solving the structure of a star or planet

Neural networks (e.g. Hopfield network)

Bad choices

- A set of equations where you plug in numbers to get a fixed answer, e.g. an equation for lift given airplane parameters

- Linear algebra, such as inverting a matrix

- Cryptography where you code up an algorithm to encrypt a message

- Calculating geometry -- optics

The biggest issue is projects that are not models – they are like a homework assignment to code up an algorithm. Some are too simple (e.g. sets of equations).   Some can’t be experimented with. Some are too complicated to code up.

Good simple models: Attributes

- Mathematical expression(s) where a physical meaning can be attributed to the variables

  e.g. population, orbit position, concentration

- Physically meaningful parameters you can play with and change

  e.g. food availability, mass, diffusion rate

- Some sort of process required to get solutions from initial values

  e.g. Stepping forward in time or from case to case, testing many cases, updating a state

The great solved problem

- You don’t have to find an open question
- You can select a problem considered solved at this point – it is easier to check that your program works!
- e.g. Many circuits can be solved with linear algebra exactly. Using the exact solution would be a bad project. BUT – the exact solution can be used to test that your integrator program is working correctly.

Your model must be a Mathematical Model

- Your model must be a mathematical description of a system
- For example, it could calculate the position of an object in two dimensions over time: x(t), y(t) starting from known positions, x(0) and y(0) at t=0
- By default, projects should be aiming to solve a **system of differential equations**. Another allowed case is Hopfield neural networks. Most other things require special permission – *before* you submit it as a proposal.

Standard models for scientific computing

A system of **ordinary differential equations (ODEs)** can predict x(t) and y(t) given some initial values for x and y at time t=0

e.g. $dx/dt = a\,y + c$

$dy/dt = b\,x^2 + f\,y + \sin(t)$

This is a non-linear system of 2 equations that can be solved for x(t) and y(t).

Solving these analytically is hard. Solving them *approximately,* with a program, is easy.

Your model

- In your project, x and y could be numbers of wolves and rabbits, positions of planets, projectiles, angles of a pendulum, numbers of infected people …

For a computer more than 2 equations is also fairly easy, e.g.

- wolves, rabbits, lettuce: 3 equations,
- one planet: 6 equations, 2 planets: 12 equations

We can help

- You probably don’t have experience solving differential equations numerically
- We will help you devise way to do it.

e.g. For $dx/dt = x^2$

Finite difference:

$$x(t2) = x(t1) + x(t1)*x(t1)*(t2-t1)$$

Accurate if $|t2-t1|$ isn’t too large

$dt = t2-t1$ is the time step

We will help

- Talk to us about your model
- You wont be penalized if your proposed model is too hard to implement in practice:

  After we read the proposal we will help find a more appropriate one

Project overview

1. Find a mathematical model, usually a set of ordinary differential equations (in the end write up some background on it too)
2. Write a program that runs the model
3. Test that your program is a correct implementation of the model **(verify)**
4. Test the model with different cases **(validate)**
5. Critique it, discuss its usefulness

You will not be penalized for an imperfect model – you just need to explore it and show where it works and where it fails. Imperfect models usually make for better projects because you can explore when they do or don't work.

Validation vs. Verification

- You should **verify** that your program does what you intended it to

  e.g. it integrates your differential equation accurately (exact solutions are useful here)

- You still need to show if the equation is a **valid** model for your system. The original equations may not be realistic in some cases

  e.g. Does the population increase in the model increase when a real population would increase?

Verifying a program

You can verify your program works:

- You can match the figures in the paper, textbook or website you got the model from
- It gives the known answer for easy cases (e.g. circular orbits)
- Different methods give the same answer
- The answer at time t2 doesn’t change if you repeat with smaller timesteps (dt)
- momentum and energy are conserved (if that is applicable)

Validating a model

- A model should display the behaviour of the real physical system:

  e.g. Closed systems should conserve energy, simple orbits should repeat, populations should increase in good conditions

Even if the program is correct, the model may do poorly in some cases. That is ok – you just document that in your critique

Program Verification vs. Model Validation

- Some tests may indicate a failure of verification and validation so they really test both

  e.g. An isolated system should conserve momentum. If your program results don't it could be a failure of the program (verification) or the model could be bad (validation)

Simple cases with known answer are a good way to tell which is at fault

Some examples:
Bacterial Growth

parameters: food level, toxin removal rate

ODEs solved for population vs. time

Tests:

- Verification: Is the equation solved correctly by the program? (e.g. changing dt gives the same answer)
- Validation: Does more food promote more growth? Does overcrowding inhibit growth?

Some examples: Diffusion of heat

parameters: diffusion rate, initial temperatures

This is fairly complex problem – a PDE needs to be solved for temperature in time and space (ask for permission in advance)

Tests:

- Is total energy conserved?
- Do sharp profiles soften?
- Does temperature flow from hot to cold?

Some examples:
Orbits

parameters: masses, initial locations

This is a set of ODEs – 6 per object, 3 for position and 3 for velocity

Tests:

- Verification: Is angular momentum, energy, linear momentum conserved?
- Validation: Does the system evolve like an real observed one?

Some examples:
Cars on a highway

parameters: speed limit, special conditions, e.g. lights, accidents

Note this is a PDE: solve for density of cars in time and space (ask for permission)

Tests:

- Verification: Do you conserve the total number of cars?
- Validation: Do cars pile up if there's an accident? Do they spread out when a light turns green?

Some examples:

The structure of a White Dwarf

parameters: mass, composition

ODE: solve for density vs. radius (not time)

Tests:

- Verification: Does pressure force balance gravity? Is the answer the same for smaller dr?
- Validation: Does the white dwarf get smaller for large masses approaching 1.4 times the mass of the Sun?

Some examples:
Neural Networks

parameters: dataset, number of neurons

Modify the network to reproduce a training dataset

e.g. Hopfield network with Hebbian learning

Tests:

- Verification: Is it stable on the training data?
- Validation: Is it able to guess correctly from incomplete data?

Detailed example: Climate and chaos

Introduction:

Climate models show long term variation

- Long periods of hot and cold (e.g. ice ages)
- Long lived or transient structures: e.g. Gulf Stream, El Ninio

Detailed example:
Climate and chaos

Methods:

Climate modeling – how is it done?

Weather models based on fluid dynamics equations plus recipes for small scale physics such as clouds

Methods: Finite difference, Spectral Methods

Special needs: Large scale parallel computing such as Earth Simulator

Detailed example:
Climate and chaos

Climate Behaviours:

Weather patterns stay within broad limits

But: Intrinsic unpredictability

- Weather is not periodic
- Weather can flip from a long term pattern to a very different pattern (e.g. ice ages)

Detailed example:
Climate and chaos

Simplified model:

Lorenz Attractor

Simplified model system:

A convection cell like the cell carrying hot air from the equator toward the poles

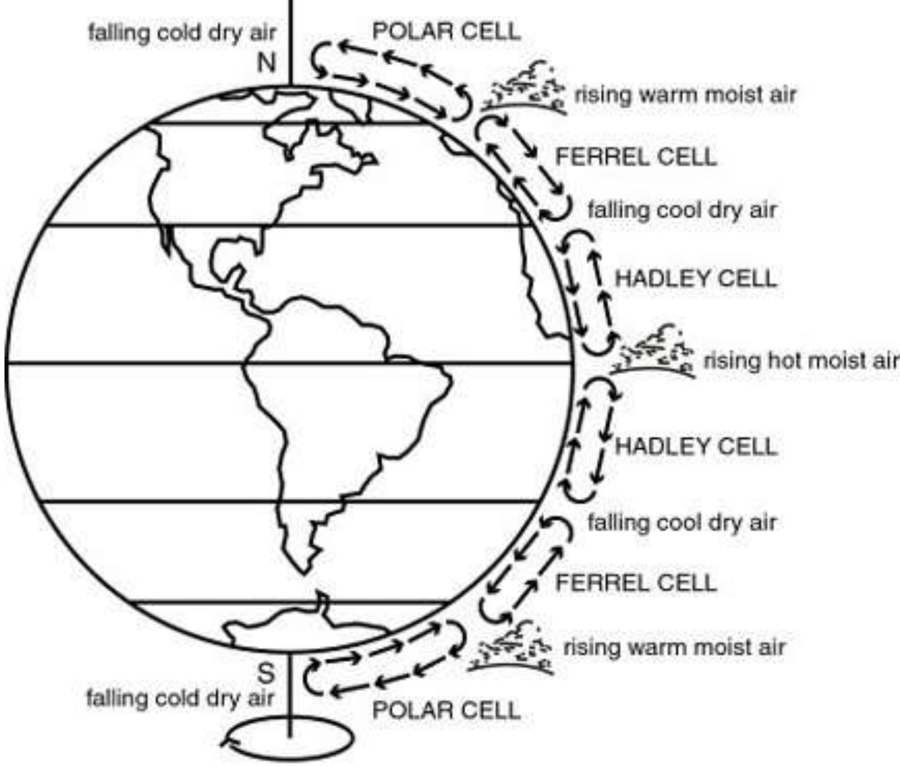


Figure 2.09
Global three-cell convection

Detailed example:
Climate and chaos

Simplified model:

Lorenz Attractor

Variables:

- X – speed of fluid motion
- Y – current temperature difference
- Z – temperature gradient linearity

Detailed example:
Climate and chaos

Simplified model:

Lorenz Attractor

Equations:

$$dX/dt = a (Y-X)$$

$$dY/dt = X (b-Z) - Y$$

$$dZ/dt = X Y - c Z$$

Parameters: e.g. $a = 10$, $b = 28$, $c = 8/3$

Detailed example:
Climate and chaos

Simplified model:

Lorenz Attractor

Key Parameter: Rayleigh number (b),

a combination of density, viscosity, diffusion and temperature gradients:

It measures susceptibility to convection

For the simple model b > 1 means convection

Varying b lets us see how the simple Lorenz attractor can model complex behaviour we associate with climate systems

Detailed example: Climate and chaos

Simplified model:

Lorenz Attractor

Raleigh number parameter: b

0 - 1	No convection
1 - 24.74	Stable convection
24.74 +	Unstable convection: Chaotic – unpredictable evolution

Detailed example:
Climate and chaos

Key References:

- Chaos and Weather prediction – New Scientist, Nov, 1989
- http://en.wikipedia.org/wiki/Lorenz_system
- http://mathworld.wolfram.com/LorenzAttractor.html
- Lorenz, E.N. "Deterministic Nonperiodic Flow", J. Atmos. Sci. 20, 130-141, 1963
- "Numerical Soln. of PDEs: Finite Difference Methods", Smith

Detailed example:
Climate and chaos

Program:

/home/2G03/lorenz

/home/2G03/lorenz/lorenz.cpp   C++ code

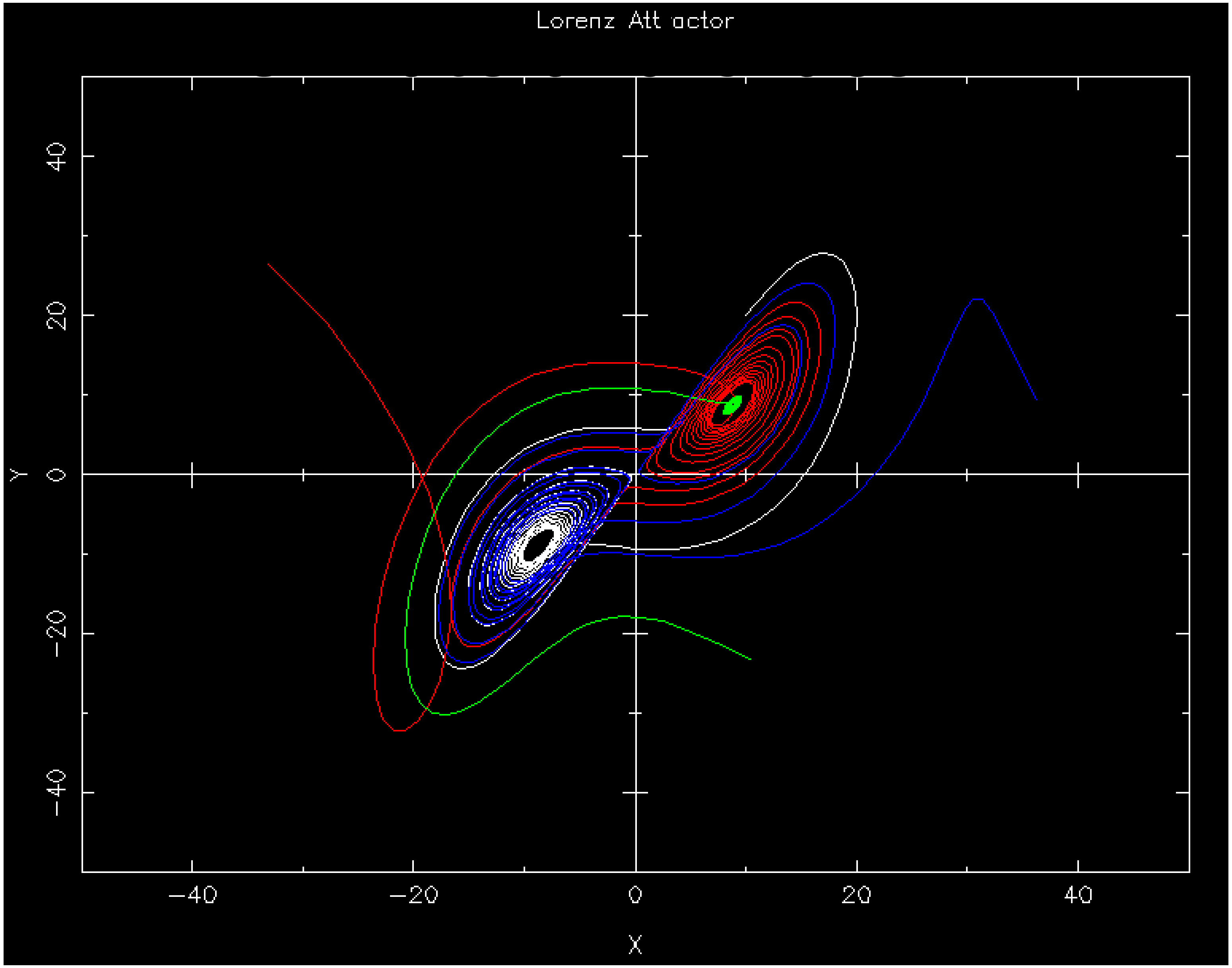
make lorenz

run it:

/home/2G03/lorenz/lorenz

Uses pgplot library to plot (we will explore that more in class and homework)

Detailed example:



Detailed example:
Climate and chaos

Note:

- The standard Lorenz attractor by itself is a bit simple for a 2G03 project – only one interesting parameter b
- Too far removed from a real system to easily extend or comment on how valid it is
- The Lorenz attractor is a bare bones system of ODEs – designed to show that varying one parameter can lead to chaotic behaviour
- **It is best to see lorenz.cpp as similar to the code you might produce for a project on a different system (e.g. ODEs)**

Your Project: Explorable models

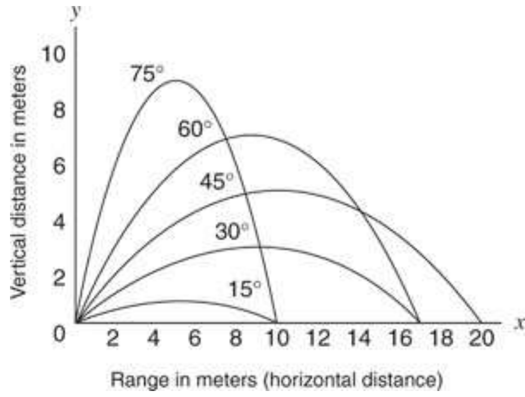
We’d like models that are still closely connected to a real system

- Models where you can explore if it is a good or bad model of that real system
- Models that can be explored in multiple ways (not just one parameter changing)
- Models than can be modified or extended in physically meaningful ways:
  - Tweaking a model could mean changing parameters, adding variables or extra equations
  - Those changes mean something: new planets, new predators, different conditions, etc..
  - Does the model stay valid, improve, get worse?

Monte Carlo

- Some systems are complicated to explore – such as all possible orbits.
- You could systematically try a large range of parameters.   Another approach is to use random numbers to explore possible initial states or cases and see what happens (The name Monte Carlo refers to gambling/chance – the random element).
- Monte Carlo is a way to explore otherwise simple project ideas

Monte Carlo
Example: Projectiles



- Projectile motion is simple (too simple)
- Calculating many different trajectories is not
- Find out which trajectories are *interesting*
- Interesting means what scores (trajectory of a ball in game), what most often hits a target (given a limit on the accuracy)
- What makes the trajectory interesting is related to the goal of the project

cf. Worms game

Monte Carlo
Example: Projectiles

- Projectile motion is simple (too simple)
- Calculating many different trajectories is not
- Find out which trajectories are *interesting*
- Interesting means what scores (trajectory of a ball in game), what most often hits a target (given a limit on the accuracy)
- What makes the trajectory interesting is related to the goal of the project

Monte Carlo
Example: Simple Orbits

- Calculating an orbit around one object is too simple
- Calculating many orbits is not
- Find out which orbits are *interesting*
- Interesting could mean dangerous (Earth crossing asteroids), productive (building planets), strange (horseshoe orbits), …
- What makes the orbits interesting is related to the scientific goal of the project