

These notes are intended to give some background information and some help for 1B students starting their CATAM projects and Part 2 students who either did not do CATAM in 1B or did not do well in CATAM in 1B.

Some useful web references have been included in the notes.

If you think of anything else which it would be useful to cover in the notes then please let me know.

These notes are for the use of Cambridge undergraduates only as they include copywrite material.

There will be two lectures. The lectures will cover,

1. The practical benefit to you of taking the time to try the CATAM projects.
2. How to write a project report for academia or industry. (Transferable skill.)
3. Instructions more specific to CATAM write-ups.
4. Some examples of things to think about
5. Referencing and crediting sources.
6. Numerical approximations
7. Errors and their analysis.
8. Complexity.

1. The practical benefits of CATAM.

Gauss called mathematics the Queen of Sciences. I found this more recent quote online.

“I don't know if mathematics is the Queen of Sciences, but I think that C++ is the King's Whore.”
An Experimental Physicist.

You are free to use any language you wish in your CATAM projects but it will not be officially supported.

- Computer programming is a distinct skill to mathematics. A good mathematician might not have an aptitude for programming and vice versa.
- Computer programming is now used in so many different branches of mathematics, both pure and applied, that it is increasing difficult to argue, as some have in the past, that it has no place in a mathematics undergraduate degree.
- Computers are an incredibly valuable tool but they, like scalpels, are not safe in untrained hands.

The existence of CATAM is an acknowledgement of these opinions.

The stated objectives of CATAM as a course are necessarily quite modest. The aim is to provide some knowledge of how to use a standard high level package such as MATLAB to solve simple mathematical problems.

Ref: <http://www.mathworks.co.uk/products/matlab/index.html>

“MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.”

You can gain a great deal more but you will need to actively try to extract as much useful experience and knowledge as you can from the projects. CATAM is not going to make you into a competent programmer but if you make the effort then it should give you enough background to save your neck in the future.

1. The course is optional and the marks are added to your total Tripos score. These are effectively FREE marks in both 1B and Part 2.
2. If you are likely to find yourself on a class boundary in the Tripos then you SHOULD do CATAM as it may push you to the right side of the boundary. You may be sure that there are people who are not here today and who will not take this advice.
3. If you suffer from exam nerves and know that you will underachieve in the written papers then you should do CATAM.
4. If you found that you had serious time problems in the 1A exams and could not write down enough of the questions, which you knew how to do, in the time allowed then you should do CATAM.
5. If you are a failure risk then you should definitely do CATAM. There are no ordinary degrees awarded except in extenuating circumstances. So if you fail to be classed with honours then you fail.
6. If you go on to a PhD or if you go on to a technical position outside academia then it is very likely that you will need to learn a programming language and use a computer to solve a specific problem. CATAM will not give you enough programming experience but it should give you the confidence to know that you can learn whatever language is required and be able to correctly write a program to calculate the task demanded.
7. If you go on to join a software house or a computer manufacturer then you may be surprised to learn that they often prefer mathematics graduates to computer science graduates. Programming style has an aesthetic element. They like to teach you their own bad habits in writing software and would rather you did not import your own from your degree.

ASIDE: If you are thinking of a career option then please, please try to get a placement after 1B in the summer. Employers increasingly treat placements as extended interviews and most of the better opportunities are now offered to students on placements. You need to know before you sign the contract that this is a job you want and can handle.

The two parts of CATAM, 1B and Part 2 teach you different skills so you should do the available number of projects in each of the two years.

8. From the 1B projects you should gain some understanding and an intuitive feel for how mathematical results look when presented in the form of numerical data, whether in the form of tables or in graphs. The 1B CATAM projects are based on mathematics which should be familiar from other courses. In principle most of you should be able to solve the problems analytically. Of course you are not usually expected to solve problems analytically but you will be expected to have an understanding of the sort of solution you would expect. When you write up you should use the terminology from your other courses and you should reference course notes when this is useful. For example, if the CATAM problem is about RSA the coding algorithm then you can use your numbers and sets notes to help you with the project. If the CATAM project is about a dynamical system with an obvious resonance then you can use your 1A dynamics notes and your 1B Methods notes to help with the project.
9. From the Part 2 projects you should gain some understanding of how to turn numerical output and graphical information into a mathematical solution. In other words how to “read” the numerical data and corresponding graphical data and relate it to models. Most of the projects will be too long and too hard for you to solve analytically so you will be reliant on the output to work out what is happening. You should be able to recognise the difference between “real effects” described in the original problem and artefacts introduced by your solution method.
10. Solving a problem numerically involves rather more than finding a plausible looking algorithm in numerical recipes and forcing your problem into the selected form and slapping it onto your computer. I will mention some of the issues in these talks and they may prove useful when you have to comment on your results in specific CATAM projects.

2. How to write a project report for academia or industry.

A Transferable skill.

In general the form taken by your write-up or presentation depends on whether you are working in academia or outside academia. It also depends on your audience. We will assume that the report is intended for the team working on the problem and not for some other purpose within the company or university such as fundraising, public relations or outreach. Your write-up must be in a language which is precise enough to satisfy the “experts” but with enough explanation to be intelligible to any scientifically trained listener. Outside of academia your project write-up, the questions you ask and the answers which you give are likely to be very tightly specified. In a commercial setting you aim to answer all and only the questions raised in the project specification. You may indicate directions to follow up but the next contract must be in place before you continue your work. In academia the report is likely to be broader and more open, picking up links to other work and papers and suggesting the next stage of the research. You are expected to point out interesting tangential directions and refer to other work which might be relevant. In CATAM you are working in a university environment and your write-up should be styled like a research report in a university.

In any project, whether in academia or in industry, you are likely to be part of a team. In CATAM you are expected to work alone but your write-up should still be structured in a lucid and informative manner as if you were addressing the rest of the research team. You should assume that your audience is numerate and intelligent but not expert in your project.

In order to point out the sorts of questions which you should have in mind when commenting in a write-up we will begin by thinking about the issues involved in setting up and solving a problem. There are of course different types of problems which lead to slightly different sorts of issues. In essence there are only three or four steps. You are given some physical phenomenon to describe and explain or some mathematical situation to analyse. For example a differential equation to solve or an algebraic problem whose solutions you need to find and possibly classify.

1. Select a model or method to describe the situation. When a model is constructed in order to describe a physical problem, it tries to describe a, sometimes, complicated system in terms of a manageably small number of variables in a specified set of surroundings. It is important to have a clear list of the assumptions made in providing the model and a good understanding of their implications. In a CATAM project this step will be done for you, but you will need to be able to identify the assumptions and be able to comment on their implications for the model which you are analysing. You need to understand the circumstances in which you expect the model to be applicable. For example in the case of a differential equation you may choose to model a more complicated system by a simple growing exponential in a specified regime. A CATAM project of this sort will typically start with a proposed differential equation describing the physical situation; with a given model. You might get “style” marks for suggesting improvements or pointing out interesting implications. You will not get credit for a long digression which has not been requested in the question. Whenever you make a statement about the observed data you must also back it up with a justification. For example, you are given an exponential growth equation for some data and the data set. You are told that it describes the growth of a type of bacteria in a petri dish as a function of time. You plot the graph of the equation against the data set and discover that the data to the far right hand side of the plot is not more scattered but is diverging away from the theoretical curve. If the question asked you to comment on the fit then your answer should point out the deviation and if you have been given sufficient background in the question also suggest a reason for the deviation and the form of a correction. For example a decrease in the growth rate might indicate that the nutrient was running out. You should also point out that the fact that the scatter of the data has not increased suggests that the error is not due to some of the possible accuracy issues in this region. Remember that a systematic error could look like a deviation from your model, particularly if that error is proportional to population size, so you need to be cautious in your assertions.
2. You now put the model into a mathematical form. The important factors become variables and you check that the regime you are describing is reasonable mathematically. In other words you check your mathematical assumptions against the physical assumptions or the given data set. In 1B

projects you will be handed this step on a plate. In part two projects some of the questions may ask you to comment on the suitability of the given model or perhaps to compare the pros and cons of two different approaches.

3. If you have a continuous problem then you recast the mathematics into discrete algorithmic form. This involves making a suitable choice of algorithm. In most CATAM projects you will be told which numerical approximation to use but it is worth thinking about the implications of using the given method. The section on numerical approximations will make clear that one differential equation can be modelled in an infinite number of different ways and the best scheme to choose will depend on the questions to be answered and often on the expected form of the solution.
4. Select a platform and a language and then write the program to solve the problem. In CATAM you are advised to use MATLAB so you can be sure that all the projects will be amenable to choice. Although there is no requirement to use MATLAB unless you are already very competent with another language I would strongly recommend that you stick to MATLAB. Many of the programming problems which have seriously affected both the time taken and the eventual marks obtained in CATAM projects have stemmed from students using unsupported languages and not really having the competence to finish the projects without help.

Each one of these steps contributes to your solution technique so in each explanation you should be able to identify the effect on your solution of each stage. This will be made more explicit later when we look at some examples.

Whenever you write a program you must trade off several competing factors.

1. **Readability of the program**

Whether in the form of a code or an interactive session your program should have sufficient comment to be readable. For the purposes of CATAM this is especially important because this is your evidence that you have done the project yourself. The advantage outside of CATAM is that you can go back to the code and easily recall the workings. This means that you can reuse parts and easily modify parts as well. It also means that other team members or CATAM examiners have a reasonable chance of understanding what you have done.

2. **Generality of the program**

You may not want to write into the program explicit numbers and parameters which you will want to change in a later question. So read the whole project before you start and make your code as versatile as possible without introducing excessive complexity.

3. **Error catching**

You are told to assume in CATAM that the data has not been provided by an idiot but you might want to trap errors which occur as part of a numerical calculation for example division by small numbers. If you do this then you should comment in the write up. Remember that your CATAM code is not

read although it may be run so the computing marks are gained by evidence in your write-up that you have performed the required tasks.

4. Simplicity verses efficiency

When you are dealing with a problem where memory is plentiful and the program runs very quickly then you can afford to sacrifice efficiency in your code for transparency of the code. It is up to you to find the right balance. This balance will often depend on the details of the problem. You should justify the choices you have made as part of your write-up.

5. Verification of Code

The subject area of safety critical programming is a very important and very difficult branch of the subject. You never have to prove that your code works you merely check that it is likely that it does. For example when calculating solutions of some algebraic equation in modular arithmetic you can “check” your code by doing some similar examples which are easy enough to check by hand. It is a good idea to provide some sample output to prove that you have made these checks.

When you write-up you should explain how you wrote your program and how you tested that it works correctly. You should include some test data showing your checks. You should comment on any particular points of interest.

Some Things to Consider in Your Write-ups

I am not allowed to consider specific examples from the CATAM project files but I can provide a selection of examples of things which you might want to consider as part of your write-up.

ALGEBRAIC EQUATIONS AND FINDING ROOTS

The practise project demonstrates three different ways of finding the root of an equation. Each method is based on a technique and a bit of analysis which you have either already met in 1A or will meet in 1B. When you need to find a root or a turning point you should consider the most convenient way of getting at the answer. This means that you should start with some idea of the position and multiplicity of the root which you are trying to find. Once you have this information it might mean that you use a mixture of one or more methods in order to speed things up or you might use a very simple method because you already know that the search will be easy and short even though you have chosen an inefficient algorithm. You should explain in your write-up why you have used your chosen method if different options were plausible. You should also comment on any special features which you need to take into account.

RECASTING A DIFFERENTIAL EQUATION IN NUMERICAL FORM

In the 1B CATAM projects you will be given the numerical approximation to use and asked to comment on your data. It helps to know a little about the way in which these approximations are derived. For the specific properties of approximations in the projects you should use references and quote them. In the Part 2 projects there may

be opportunity to discuss the relative merits of one numerical approximation over another.

There are essentially two distinct approaches to numerical approximations for solving differential equations.

1. Finite difference methods
2. Finite element methods. (Finite difference method based on variation principle)

Since these talks are aimed mainly at the 1Bs I will only discuss the first of these but I will give some background on these methods and an example later on.

It is possible to calculate errors or optimal step lengths from the theory but it is also possible to calculate these explicitly numerically. If it is realistic to use both methods and check one against the other then that is a good strategy. In some situations you will have only one method available for practical reasons. You should always explain what you have done and why you have chosen your approach if there is more than one approach available.

In both industry and academia the different stages of setting up the problem are equally important and may involve collaboration of different people, with different training, in a team. This is obvious in an industrial or commercial setting but even in a university you might be the same mathematician in, for example, a civil engineering group working on optimising flows through a sewage drain system or traffic through London's busy road network. Since some of these people may be from completely different disciplines, your project report must be written in a way which is accessible to all.

3. Instructions more specific to CATAM write-ups.

1. First and foremost read the NEWS pages on the CATAM helpline.
2. Read the General Comments in the 1B instructions.
3. Before starting a project read the questions and answers related to that project.
4. Check that you are on the CATAM mail list.
5. If you chose not to try the sample project then read it and the model answer available online carefully.
6. In 1B try to pick projects in subjects which you like as they may be useful revision.
7. In part 2 there is no particular advantage or disadvantage to choosing short or long projects, but if you are unsure of your ability to finish a long project then two shorter projects may be a safer bet.
8. Some part 2 projects are based on 1B material. This is to allow you to get started over the summer. I know that you are likely to be working for part of the summer but your final year will also be disrupted with interviews and assessment days so it is worth getting an early start.

Presentation of your Write-up

Some of you are likely to have some help/advice in how to write-up your projects from family and friends who write reports as part of their jobs. This next section is intended to help those who do not have access to such informal help.

In CATAM most of the work is done for you. You are given the problem and told the algorithm to use. You are then encouraged to use the standard package of MATLAB to solve the problem. So you can be sure that the problem will be amenable to solution in this manner. A typical beta answer to a CATAM project will simply follow instructions correctly and produce a correct answer at the end. The CATAM questions allow you the opportunity to demonstrate that you understand what you are doing and appreciate both the limitations of the chosen method and the implications of your results. This is required to secure an alpha answer. This is particularly important in the Part 2 projects. The CATAM projects test your ability to solve and understand a problem using a computer as a tool.

1. Readability

I recommend that you word process your write-up. You are allowed to write up the projects by hand but word processing has the advantage that you can make changes easily and quickly. Some students choose to word process the main explanation part of their projects but write in the equations by hand. This is perfectly acceptable. No one expects perfect grammar but it is important to ensure that your answers to questions are clearly expressed, concise and detailed. Examiners are human and will do their best to give you as much credit as possible but if they cannot read your work or follow your reasoning then even if you have the right idea but have garbled the write-up they cannot give you credit. Do not blag and do not waffle as it will not improve your marks and may cost you some credit.

2. Maximising Marks

Try to structure your write-up in a way which will make it easy for examiners to mark. Be careful to indicate that every task requested in the project has been carried out. Suppose a project begins with Question 1.1 and then a Programming Task and then Question 1.2. Then in your write-up you should mirror these sub headings. You have a heading Question 1.1, then you have a sub heading Programming task for Question 1.2 and finally a sub heading for Question 1.2.

For each question you must make sure that you have answered the question with as much insight as well as detail as you can. If you are asked to draw a graph and comment on your results then you need to give a detailed explanation of the data and not just a description.

As a rule of thumb, ask the question

“Would a savvy 1B Lawyer or English Student be able to make these comments?”

If the answer is “Yes” then you have not said enough.

If a question asks you to comment on a data table it is often useful to draw a graph. Commenting on the general shape of a graph is usually NOT sufficient. You need to describe the form precisely and to relate it back to the mathematics.

Going back to our model of the growth of a bacterial culture in a petri dish, suppose the data looks like exponential growth which falls off rapidly after a certain time. You model the system as an exponential growth for a fixed time ignoring the correction required at later times. Writing simply that the graph shows that the population is increasing with time is not sufficient. You need to say more. For example, the original model predicts an exponential growth, from the graph, taking early times, we have a best fit with the coefficients $N(0)$ which represents the initial population and an exponent k , and you give both the two values and errors if you can. For later times the growth rate is reduced. This is the result of the numerical approximation which is a polynomial and does not reflect the original data where there is such a decay possibly due to lack of nutrient which was not included in the original model.

If you have anomalous data, for example, some statistical information with suspicious outlying points then you can comment on this and you can look at the effect of ignoring the contribution from them in a best fit. For every “feature” which you observe in your data you need to identify its origin. Is it a part of your original problem, is it something in your original differential equation for example a term which can be clearly seen to give rise to a resonance. Is it an artefact of your numerical solution, for example, because your numerical method is a polynomial modelling an exponential for large times from initial time zero. Is it an artefact of the final stage on the computer because in your method rounding errors begin to grow in the region. For every feature you need to look for echoes of it at each stage of your working. You should then find that you have a lot more to say about the questions.

Under the Programming Task subheading you should include the name of your program or session, which should also be a comment line at the top of your printout. You should also make any comments which you think are important. For example in a primality test program. Is it advantageous to check all non even smaller integers to see if they are factors of some given N or is it better to store the primes found and only check those to see if they divide N . Clearly the later is better for large N but for some purposes because computers are so fast and if the number being checked is sufficiently small it might not be worth the effort. You should justify any such decisions which you make. In this section you should also explain how you checked that your method was working correctly and provide sample test data if this is appropriate.

Arrange your write-up in an easy to read structure. Graphs should be between $\frac{1}{3}$ and $\frac{3}{4}$ of an A4 page except in exceptional circumstances. This is not compulsory but I would put graphical data which is part of the write-up into the script so that the examiner can read your solutions and look at your graphs in order without having to keep thumbing to the appendices. Graphical data which is in some way incidental would be better placed in an appendix. Similarly large data tables should be relegated to the appendices. I would label the appendices with the question number as well as a title. In your appendix your first line is a title. This is appendix A.12.1.1a for Question 1.1a. It is followed by the statement in which you give the title of the table “ This table shows the step lengths and errors for the range specified in question 1.1.a.” Then you reproduce the table. If you have a large data table but that data contained a few lines which were needed for the write up then I would reproduce those lines in the body of the script. I would write something like, “The 6 lines below

are an extract from table T.12.1.1 in the appendix for Question 1.1 which is the table showing the errors in the required range showing the optimal range of values for the step length h ." You should comment on your reasons for calling your choice "optimal". You might want to consider the following questions.

1. Restrictions due to method of subdivision of data
2. Restrictions due to rounding and global error.
3. Restrictions due to number of iterations or time.
4. Is there an optimal choice
5. What is a acceptable range of choices and why

I use a numbering system where my appendix is labelled by, A for appendix, then the project number, 12, followed by the question number 1.1. The table is then called T for table, followed by the project and question numbers and parts. You can use a different system if you wish

I would also place testing output for your programming in an appendix. So your write-up would say that you have tested that everything is working correctly and that sample test output can be found in appendix A12.1.1b. The actual output pages can then go into an appendix with the usual introductory lines at the top of the appendix.

If the questions are numbered 1.1a then I would follow the same pattern for the appendices and label them 1,1a(i) and 1.1(a)2 etc... If the questions are labelled 1.1(i) then I would label the appendices 1.1(i)a, etc... where the final symbol is there in case there is more than one appendix associated with a part of a given question.

Remember to check the instructions to make sure that you have left enough space on the front page and have sufficient margin space. Do not forget to number pages and label all diagrams, graphs and tables. Do not forget NOT to put your name on every page of your script. Remember that the instructions in the CATAM manual will include conventions needed for the electronic handling of your projects make absolutely certain that you have followed instructions.

Try to finish your project with time to spare so that you can leave it for a week and then read through your work. This is useful because when you are writing up everything is very fresh and familiar. If you leave the work and do something else for a couple of days you may find when you re-read the script that you need to fill out some of your explanations because you have forgotten to state some of the details.

Both in 1B and in Part 2 the projects are set to be worth roughly a full lecture course in total. Do not spend excessive time on the projects. In particular some of the Part 2 projects are very open ended so remember that once you have gained the available marks there is no more to be done.

Do not leave handing in your projects to five minutes from the deadline on the final day. Stop tinkering and get rid of them as soon as you can so that you can get on with your other work.

3. Credit Checking

In the first two projects you have a pro forma at the back to help you. When you have completed your write-up go through it with the marking scheme in front of you and check that you have written enough in your answer to each question to pick up all the available marks. The marks are ALL available within the write-up. So if the scheme says that there are $2M + 3C$ marks available in a question then you need to make two marks worth of correct mathematical statements and demonstrate in your write-up that you have made three marks worth of computing statements or results within the question. In later projects and particularly in Part 2 projects, particularly the more open ended projects, I suggest that you write your own mark scheme for each project and then check that you have said enough in the right places. You will need to write the mark scheme when you have more or less finished the project and write up as you will need to have some idea of how much work goes into each section.

4. References and Sources

You must always credit all the sources which you have used in your write-up. If you give proper credit then it is not plagiarism. It does not matter whether you put your references at the end of the document or as footnotes in the write up. You are likely to have different kinds of references. For example.

1. Lectures.
2. Supervisions.
3. Books.
4. Web based sources.
5. Private conversations.

Clearly you are expected to make use of the knowledge you have acquired through lectures and supervisions in Cambridge so you do not need to specifically mention them in your references unless you wish to emphasise a specific point.

The exception to this is in the case of help with a specific unsubmitted CATAM project. Some colleges arrange CATAM supervisions for their students. Much of the material covered is of the general form contained in these lectures. Supervisors are not allowed to give very specific help with the unsubmitted projects nor are they allowed to point out errors. Please do not embarrass your supervisor by trying to get them to give you excessive help with details of unsubmitted projects. They can tell you that your explanation needs more work. It might be too vague or there might be insufficient comment or there might be an error. When I supervised these projects I was often unwilling/unable to say which of these options was the case. If you feel that your supervisor or some other individual has given you too much help with the details of the project then you should declare the help.

When you refer to a book you do not need to give a detailed page reference but you might find it helpful to do so in case you want to check something in the future.

There is nothing wrong with using web based sources it is often a very quick and efficient way of finding information but whenever you credit a reference you should also consider the trustworthiness of that reference and make it clear that you have done so. Suppose you are asked to use your computer to count the number of primes below 3300. You can almost certainly find both lists and counting utilities online. If

you find a little demonstration on some teaching site at harvard.edu then you can give the reference and also comment that since your answer agrees with the Harvard answer then you are confident that it is correct. If you find a solution in some thread from a chat room then you need to be a bit more cautious. The best you can say is that the answer agrees with yours and that the probability of your both getting the same wrong answer is small. (Be careful though because in some circumstances students make the same sort of errors and these lead to the same results!) If you have time then you should try to find a better reference.

It is inevitable that students will compare results during informal conversations at tea. You should avoid any discussion which involves you in showing another student any of your work or which involves you in writing down on paper how to solve a given problem. Keep your work secure. Do not leave copies of write-ups or print outs lying around as there are some very unscrupulous people around. I would also keep both hard and electronic copies of all your work. The 1B projects are sometimes helpful when you start the Part 2 projects.

Floating Point Real Numbers and Small Numbers

This is in response to a question from a 1B student.

Integers can be represented exactly on a computer but the range is restricted. In some languages you can specify the type of a number and it might pay to work with integers only for some problems such as finding primes. The floating point representation of a real number allows a larger range of numbers to be represented than would be the case with fixed point or integer representations. The notation is the familiar scientific notation. Numbers are represented by an approximation to a fixed number of significant digits and scaled using an exponent.

$$\text{Significant digits} \times \text{base}^{\text{exponent}}$$

The base for the scaling is normally 2, 10 or 16. The term **FLOATING POINT** refers to the fact that the decimal point or binary point can "float"; that is, it can be placed anywhere relative to the significant digits of the number. Floating-point representation can support a much wider range of values, but the floating-point format needs slightly more storage to encode the position of the decimal or binary point, so when stored in the same space, floating-point numbers achieve their greater range at the expense of precision. The speed of floating-point operations is an important machine characteristic, especially in large-scale mathematical calculations. Single precision is comparable in its requirements to integer calculations, perhaps 4 bites, whereas double precision, not surprisingly, takes more storage. If storage is unlikely to be an issue than working with doubles keeps things simple.

Small Number Calculations

$$\mathbf{f(t + h)} = \sum_{r=0}^{\infty} \frac{\mathbf{h^r f^{(r)}(t)}}{\mathbf{r!}}$$

Finite Difference Methods

In the 1A Analysis a real analytic function was defined as a function which had a convergent Taylor series.

$$\mathbf{f}(t + \mathbf{h}) = \sum_{r=0}^{\infty} \frac{\mathbf{h}^r \mathbf{f}^{(r)}(t)}{r!}$$

From this we know that a real analytic function on the real line can be completely specified either by giving all the values on the line or by giving its Taylor series at a point. We can approximate the value of the function by truncating the series to obtain,

$$\mathbf{f}(t + \mathbf{h}) = \sum_{r=0}^N \frac{\mathbf{h}^r \mathbf{f}^{(r)}(t)}{r!} + \mathbf{R}_N(t, \mathbf{h})$$

Where the Lagrange form of the remainder is

$$\mathbf{R}_N(t, \mathbf{h}) = \frac{\mathbf{h}^{N+1} \mathbf{f}^{(N+1)}(\xi)}{(N+1)!} \quad t < \xi < t + \mathbf{h}$$

Finite difference methods are numerical methods for approximating the solutions to differential equations by using finite difference equations to approximate derivatives. In the numerical analysis the real line is replaced by a grid which is often, but not always, equally spaced. The parameter \mathbf{h} is called the step length and the derivatives in the Taylor series can be approximated and replaced by finite differences between points on the grid.

The **ERROR** in a method's solution is defined as the difference between its approximation and the exact analytical solution.

In finite difference methods there are two sources of error

1. Rounding Error
This is the loss of precision due to computer rounding of decimal quantities.
2. Truncation Error
This is the difference between the exact solution of the finite difference equation and the exact quantity ignoring the rounding error.

The **LOCAL TRUNCATION ERROR** is defined by $\mathbf{f}(t_k) - \mathbf{f}_k$ where $\mathbf{f}(t_k)$ is the exact value of the function at the point $t = t_k$ and \mathbf{f}_k is the numerical approximation at $t = t_k$.

This error is often expressed using the “big O” notation. Recall the definition

$$\mathbf{f}(\mathbf{t}) = \mathbf{O}(\mathbf{g}(\mathbf{t})) \text{ as } \mathbf{t} \rightarrow \mathbf{a} \text{ iff } \exists \delta, \mathbf{M} > \mathbf{0} : |\mathbf{f}(\mathbf{t})| \leq \mathbf{M}|\mathbf{g}(\mathbf{t})| \text{ for } |\mathbf{t} - \mathbf{a}| < \delta$$

If the function $\mathbf{g}(\mathbf{t})$ is non zero in a neighbourhood then we have an equivalent form

$$\mathbf{f}(\mathbf{t}) = \mathbf{O}(\mathbf{g}(\mathbf{t})) \text{ as } \mathbf{t} \rightarrow \mathbf{a} \text{ iff } \limsup_{\mathbf{t} \rightarrow \mathbf{a}} \left| \frac{\mathbf{f}(\mathbf{t})}{\mathbf{g}(\mathbf{t})} \right| < \infty \text{ for } |\mathbf{t} - \mathbf{a}| < \delta$$

The remainder term of a truncated Taylor series is convenient for analyzing the local truncation error.

Notation and terminology

http://en.wikipedia.org/wiki/Forward_difference

A **FORWARD DIFFERENCE** has the form $\Delta_h[\mathbf{f}](\mathbf{x}) = \mathbf{f}(\mathbf{x} + \mathbf{h}) - \mathbf{f}(\mathbf{x})$

A **BACKWARD DIFFERENCE** has the form $\nabla_h[\mathbf{f}](\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x} - \mathbf{h})$

A **CENTRAL DIFFERENCE** has the form $\delta_h[\mathbf{f}](\mathbf{x}) = \mathbf{f}\left(\mathbf{x} + \frac{\mathbf{h}}{2}\right) - \mathbf{f}\left(\mathbf{x} - \frac{\mathbf{h}}{2}\right)$

Depending on the application, the spacing \mathbf{h} may be variable or constant.

Relation with derivatives

The right derivative is defined by

$$\mathbf{f}^{(1)}(\mathbf{x}) = \lim_{\mathbf{h} \rightarrow 0} \frac{\mathbf{f}(\mathbf{x} + \mathbf{h}) - \mathbf{f}(\mathbf{x})}{\mathbf{h}}$$

Therefore the forward difference approximates the right derivative when \mathbf{h} is small.

$$\frac{\mathbf{f}(\mathbf{x} + \mathbf{h}) - \mathbf{f}(\mathbf{x})}{\mathbf{h}} = \frac{\Delta_h[\mathbf{f}](\mathbf{x})}{\mathbf{h}}$$

The error in this approximation can be derived from Taylor's theorem,

$$\frac{\Delta_h[\mathbf{f}](\mathbf{x})}{\mathbf{h}} - \mathbf{f}^{(1)}(\mathbf{x}) = \mathbf{O}(\mathbf{h}) \quad \text{as } \mathbf{h} \rightarrow \mathbf{0}$$

The left derivative is defined by

$$\mathbf{f}^{(1)}(\mathbf{x}) = \lim_{\mathbf{h} \rightarrow 0} \frac{\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x} - \mathbf{h})}{\mathbf{h}}$$

Therefore the backward difference approximates the left derivative when \mathbf{h} is small.

$$\frac{\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x} - \mathbf{h})}{\mathbf{h}} = \frac{\nabla_h[\mathbf{f}](\mathbf{x})}{\mathbf{h}}$$

The error in this approximation can be derived from Taylor's theorem,

$$\frac{\nabla_h[f](\mathbf{x})}{h} - f^{(1)}(\mathbf{x}) = O(h) \quad \text{as } h \rightarrow 0$$

The central difference yields a more accurate approximation, with an error proportional to square of the spacing.

$$\frac{\delta_h[f](\mathbf{x})}{h} - f^{(1)}(\mathbf{x}) = O(h^2) \quad \text{as } h \rightarrow 0$$

A problem with the central difference method, however, is that oscillating functions can yield zero derivative.

Higher-order differences

We can similarly obtain finite difference approximations to higher order derivatives.

2nd Order Central

Apply a central difference formula for the derivative of $f^{(1)}(\mathbf{x})$ at \mathbf{x} with a spacing of $\frac{h}{2}$, and then substitute into this the central difference formula for the derivative.

$$\frac{\delta_h^2[f](\mathbf{x})}{h^2} = \frac{f^{(1)}\left(\mathbf{x} + \frac{h}{2}\right) - f^{(1)}\left(\mathbf{x} - \frac{h}{2}\right)}{h^2} = \frac{f(\mathbf{x} + h) - 2f(\mathbf{x}) + f(\mathbf{x} - h)}{h^2}$$

$$f^{(2)}(\mathbf{x}) \approx \frac{\delta_h[f](\mathbf{x})}{h^2} = \frac{f(\mathbf{x} + h) - 2f(\mathbf{x}) + f(\mathbf{x} - h)}{h^2}$$

Similarly we can apply other difference formulae in a recursive manner.

2nd Order Forward

$$f^{(2)}(\mathbf{x}) \approx \frac{\Delta_h^2[f](\mathbf{x})}{h^2} = \frac{f(\mathbf{x} + 2h) - 2f(\mathbf{x} + h) + f(\mathbf{x})}{h^2}$$

The n^{th} -order forward, backward, and central differences are

$$f^{(n)}(\mathbf{x}) + O(\mathbf{h}) = \frac{\Delta_{\mathbf{h}}^n [f](\mathbf{x})}{\mathbf{h}^n} = \frac{1}{\mathbf{h}^n} \sum_{j=0}^n (-1)^j \binom{n}{j} f(\mathbf{x} + (\mathbf{n} - j)\mathbf{h})$$

$$f^{(n)}(\mathbf{x}) + O(\mathbf{h}) = \frac{\nabla_{\mathbf{h}}^n [f](\mathbf{x})}{\mathbf{h}^n} = \frac{1}{\mathbf{h}^n} \sum_{j=0}^n (-1)^j \binom{n}{j} f(\mathbf{x} - j\mathbf{h})$$

$$f^{(n)}(\mathbf{x}) + O(\mathbf{h}^2) = \frac{\delta_{\mathbf{h}}^n [f](\mathbf{x})}{\mathbf{h}^n} = \frac{1}{\mathbf{h}^n} \sum_{j=0}^n (-1)^j \binom{n}{j} f\left(\mathbf{x} + \left(\frac{\mathbf{n}}{2} - j\right)\mathbf{h}\right)$$

In the central difference formula the odd \mathbf{n} terms have non integer multiples of \mathbf{h} . This changes the grid size. We can get around this by taking averages of

$$\delta^n [f] \left(\mathbf{x} - \frac{\mathbf{h}}{2} \right) \text{ and } \delta^n [f] \left(\mathbf{x} + \frac{\mathbf{h}}{2} \right).$$

Higher-order differences can also be used to construct better approximations. The finite difference can be centered about any point by mixing forward, backward, and central differences.

Using Taylor Series we can make approximations, which sample an arbitrary number of points to the left and a possibly different number of points to the right of the center point, for any order of derivative. This is useful for differentiating a function near a grid edge.

This means that we have an infinite supply of numerical approximations to our original differential equation. In CATAM you are usually told which method to use but if you are asked to compare two different methods then remember that if you were choosing a method for yourself then you would be taking into account the following factors.

1. The required accuracy of the approximation.
2. Special features of the differential equation and how they would be reflected in a given numerical approximation.
3. The type of solution you expect.

Are you trying to model the surface of the Grand Canyon or the Cambridge fenlands? Are you interested in small scale or large scale features?

Is your data in the form of large amounts of height above sea level measurements or is it in the form of derivative and gradient information.

Using derivative data rather than functional values in your algorithm is often useful in numerical minimisation of functions. As usual there is a trade off. The algorithms with derivative data are faster but more complicated in general. You need to use your judgement in each particular case. In a CATAM project you might want to compare run times for two such algorithms and comment on the results.

A Naive Example Problem

When presented with a problem one might think that a numerical approximation can be obtained from the Taylor Series.

For example suppose you are given the equation

$$\frac{d^2 Y(t)}{dt^2} - k^2 Y(t) = 0 \quad 0 \leq t < \infty \quad k > 0$$

$$Y(0) = A \quad \left. \frac{dY}{dt} \right|_{t=0} = kA$$

We can numerically approximate this by having a grid on which we give values of our function or a combination of values and derivatives.

$$f(t+h) = f(t) + hf^{(1)}(t) + \frac{h^2 f^{(2)}(t)}{2!} + O(h^3)$$

$$f(t-h) = f(t) - hf^{(1)}(t) + \frac{h^2 f^{(2)}(t)}{2!} + O(h^3)$$

So we have

$$f^{(2)}(t) = \frac{f(t+h) + f(t-h) - 2f(t)}{h^2} + O(h)$$

and for the first derivative we are forced to use only forward terms so

$$f^{(1)}(t) = \frac{f(t+h) - f(t)}{h} + O(h^2)$$

Hence with step length h and $Y_n = Y(t_n)$ and $t_{n+1} = t_n + h$

$$Y_{n+1} - (2 + h^2 k^2) Y_n + Y_{n-1} = 0 \quad 0 \leq t < \infty \quad k > 0$$

$$Y_0 = A$$

$$\text{and } \left. \frac{dY}{dt} \right|_{t=0} = Ak \text{ so } Y^{(1)}_n = \frac{Y_{n+1} - Y_n}{h} + O(h^2) \text{ and } Y^{(1)}_0 = \frac{Y_1 - Y_0}{h}$$

$$\text{Solving } \lambda^2 - (2 + h^2 k^2) \lambda + 1 = 0 \text{ so } \lambda = \frac{(2 + h^2 k^2) \pm \sqrt{(2 + h^2 k^2)^2 - 4}}{2}$$

$$\lambda = \left(1 + \frac{h^2 k^2}{2} \right) \pm hk \sqrt{1 + \frac{h^2 k^2}{4}}$$

$$Y_n = a \left(\left(1 + \frac{h^2 k^2}{2} \right) + hk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)} \right)^n + b \left(\left(1 + \frac{h^2 k^2}{2} \right) - hk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)} \right)^n$$

$$Y_0 = A = a + b$$

$$Y^{(1)}_0 = kA = \frac{Y_1 - Y_0}{h}$$

$$b = \frac{(k+1)Ah - A \left(1 + \frac{h^2 k^2}{2} \right) - Ahk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)}}{-2hk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)}}$$

$$a = \frac{(k+1)Ah - A \left(1 + \frac{h^2 k^2}{2} \right) + Ahk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)}}{2hk \sqrt{\left(1 + \frac{h^2 k^2}{4} \right)}}$$

You now have to worry about the stability of this method before you can do anything with it.

Numerical Stability

If we ignore truncation and rounding errors then numerical algorithms are constructed to approach the correct solution in the limit when the step length goes to zero and the number of steps to infinity.

Depending on the computational method chosen the errors can be magnified or damped. If they are damped then the method is called **NUMERICALLY STABLE**. There are different definitions of numerical stability depending on the application.

Sometimes a calculation which can be carried out in several ways, all of which are algebraically equivalent in terms of ideal numbers, but which yield different results when carried out on computers.

Reducing the step size h reduces the truncation error of the difference approximation. As we have seen this error decreases more rapidly for central difference approximations and more slowly for forward and backward difference approximations. When h becomes too small, the difference approximations are taken for almost equal values of $f(x)$ at the two points. Any rounding error of computations of $f(x)$ is magnified by a factor of $1/h$. As a result, the rounding error grows with h for very small values of h . An optimal step size $h = h_{opt}$ can be computed from minimization of the sum of truncation and rounding errors.

It is also possible to identify an optimal range of values for \mathbf{h} in any given problem just by numerical trials. It then makes sense to choose a convenient \mathbf{h} within the given range. Note that for practical purposes it is sometimes useful to choose \mathbf{h} to be of a specific form, for example a multiple of 7 because that is convenient for the grid being used.

We will look at an example a little later which gives you three different difference methods for solving the heat equation. In each case there are advantages and disadvantages to using the specified method.

Explicit methods calculate the state of a system at a later time from the state of the system at the current time. We solve an equation of the form

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) = \mathbf{G}(\mathbf{f}(\mathbf{x}))$$

Implicit methods find a solution by solving an equation involving both the current state of the system and the later one. We solve an equation of the form

$$\mathbf{G}(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x} + \mathbf{h})) = \mathbf{0} \text{ to find } \mathbf{f}(\mathbf{x} + \mathbf{h})$$

Implicit methods require an extra computation and they can be much harder to implement. Implicit methods are used because there are many problems arising in practice for which the use of an explicit method requires impractically small time steps to keep the error in the result bounded. (For example see “stiff” problems.) For such problems, to achieve given accuracy, it takes much less computational time to use an implicit method with larger time steps. The choice of explicit or implicit method depends upon the problem to be solved.

Example 1:

http://en.wikipedia.org/wiki/Implicit_method

Consider the ordinary differential equation

$$\frac{dy}{dt} = -y^2 \quad y \in [0, a] \quad \text{with initial Condition } y(0) = 1$$

We can write down discrete versions of this equation. The simplest explicit method is the Forward Euler and the simplest implicit is the Backward Euler method.

Equi-spaced grid for \mathbf{t} : $\mathbf{t}_k = \frac{\mathbf{a}k}{\mathbf{n}}$ with step length $\mathbf{h} = \frac{\mathbf{a}}{\mathbf{n}}$ and $\mathbf{0} \leq \mathbf{k} \leq \mathbf{n}$

The numerical approximation of the analytic function $\mathbf{y}(\mathbf{t}_k)$ at the grid point $\mathbf{t} = \mathbf{t}_k$ will be denoted \mathbf{y}_k .

The Forward Euler method

The forward Euler method is the explicit method.

$$\left. \frac{dy}{dt} \right|_{t=t_k} \approx \frac{\Delta_hy}{h} = \frac{y_{k+1} - y_k}{h} = -y_k^2$$

Gives $y_{k+1} = y_k - hy_k^2$ for each $k = 0, \dots, n$

Backward Euler method

The Backward Euler method is the implicit method

$$\left. \frac{dy}{dt} \right|_{t=t_k} \approx \frac{\nabla_hy}{h} = \frac{y_{k+1} - y_k}{h} = -y_{k+1}^2$$

Gives $y_{k+1} + hy_{k+1}^2 = y_k$ for y_{k+1}

This is a quadratic equation, having one negative and one positive root. The positive root is picked because in the original equation the initial condition is positive, and then y at the next time step is given by

$$y_{k+1} = \frac{-1 + \sqrt{1 + 4hy_k}}{2h}$$

In the vast majority of cases, the equation to be solved when using an implicit scheme is much more complicated than a quadratic equation, and no exact solution exists. Then one uses root-finding algorithms, such as Newton's method.

Example 2: The heat equation in one dimension with homogeneous Dirichlet boundary conditions

http://en.wikipedia.org/wiki/Finite_difference_method

Differential Equation: $\frac{\partial U(\mathbf{x}, t)}{\partial t} = \frac{\partial^2 U(\mathbf{x}, t)}{\partial \mathbf{x}^2}$

Boundary Condition: $U(0, t) = U(1, t) = 0$

Initial Condition: $U(\mathbf{x}, 0) = U_0(\mathbf{x})$

METHOD (1) Explicit method solves

Equi-spaced grid for \mathbf{x} : $\mathbf{x}_0, \dots, \mathbf{x}_J$ with step length h

Equi-spaced grid for t : t_0, \dots, t_N with step length k

The numerical approximation of the analytic function $U(\mathbf{x}_j, t_n)$ at the grid point

(\mathbf{x}_j, t_n) will be denoted U_j^n .

Use a forward difference at time t_n

Use second-order central difference for the space derivative at position \mathbf{x}_j ("FTCS")

We get the recurrence equation:

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{h^2}$$

To apply this method we rewrite it in the form,

$$U_j^{n+1} = \left(1 - 2\frac{k}{h^2}\right)U_j^n + \frac{k}{h^2}U_{j-1}^n + \frac{k}{h^2}U_{j+1}^n$$

$U_0^n = 0$ and $U_J^n = 0$ are the boundary conditions.

So given the values at a time $\mathbf{t} = \mathbf{n}$ on the right hand side we can evolve to the next time step.

Numerically stable and convergent for $\frac{k}{h^2} \leq \frac{1}{2}$ with numerical errors

$$\Delta U = O(k) + O(h^2)$$

METHOD (2) Implicit method

Use a backward difference at time t_n

Use second-order central difference for the space derivative at position \mathbf{x}_j

We get the recurrence equation:

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{h^2}$$

To apply this method we rewrite it in the form,

$$U_j^n = \left(1 + 2\frac{k}{h^2}\right)U_j^{n+1} - \frac{k}{h^2}U_{j-1}^{n+1} + \frac{k}{h^2}U_{j+1}^{n+1}$$

$U_0^n = 0$ and $U_J^n = 0$ are the boundary conditions.

We solve this linear system of equations for U_j^n .

Numerically stable and convergent for $\frac{k}{h^2} \leq \frac{1}{2}$ with numerical errors

$$\Delta U = O(k) + O(h^2)$$

METHOD (3) Crank–Nicolson Method

Use a central difference at time $t_{n+1/2}$

Use second-order central difference for the space derivative at position x_j

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{1}{2} \left(\frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{h^2} + \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{h^2} \right)$$

We can obtain U_j^{n+1} from solving a system of linear equations

$$\left(2 + 2\frac{k}{h^2} \right) U_j^{n+1} - \frac{k}{h^2} U_{j-1}^{n+1} - \frac{k}{h^2} U_{j+1}^{n+1} = \left(2 - 2\frac{k}{h^2} \right) U_j^n + \frac{k}{h^2} U_{j-1}^n + \frac{k}{h^2} U_{j+1}^n$$

The scheme numerically stable and convergent.

The error is $\Delta U = O(k^2) + O(h^4)$ away from the boundaries but may rise to $\Delta U = O(k^2) + O(h^2)$ near the boundaries.

However, near the boundaries, the error is often $O(h^2)$ instead of $O(h^4)$.

Usually the Crank–Nicolson scheme is the most accurate scheme for small time steps. The explicit scheme is the least accurate and can be unstable, but is also the easiest to implement and the least numerically intensive. The implicit scheme works the best for large time steps.

Sources of Error

1. Your original model includes assumptions. These introduce errors. In CATAM projects we are given the model so with the exception of some part 2 projects which explicitly ask for comments on the difficulties or advantages of the model you can ignore these issues.
2. The next step might be turning a continuous system into a discrete system. I hope we have covered enough material about discrete methods for you to realise that these are a source of error and also that each problem must be considered on its own merits. Many CATAM projects ask questions which invite you to comment about the choice of numerical approximation or to compare different numerical approximations. If you check sources then you

will be able to say much more. In principle it is easy to take a Taylor series and use it to find a discrete analogue of a differential equation. It is harder to show that the resulting approximation actually works. You can always find discussion of the accuracy, stability and key features of the most commonly used algorithms in CATAM. You can quote these results but you can also verify that they are consistent with your data in your particular project.

- 3 This might be relevant in some Part 2 CATAM projects where you are explicitly making comparisons between different numerical algorithms. Ask yourself would a different approach be better.

COMPLEXITY

Some of you are conscientious enough to have gone online or looked in books or talked to compsci friends about the definition of complexity and you may now be very confused. CATAM takes a very simple minded approach to complexity and you really only need to follow instructions in each project and in the CATAM lectures which were given last term.

The CATAM definition of complexity is the number of arithmetic operations needed to carry out an algorithm. All the operations are given equal weight. This is hugely oversimplified but provides a good starting point from which to develop your experience. Do not be confused by things which you may hear from Computer Science students or read on the web. For the purposes of CATAM you are only interested in arithmetic operations and you give them all equal weight as instructed in the manual. You may find that the instructions for complexity calculations differ slightly from project to project always follow instructions.

I am going to give you a little background so that you have some idea of the issues involved and also being ignored. The first thing to recognise is that complexity is defined in different ways by different disciplines and therefore you need to be careful when talking to strangers!

The complexity analysis of numerical algorithms is an important part of the broader subject of computational complexity theory. Computational complexity provides theoretical estimates for the resources needed by any algorithm to solve a given computational problem.

The **COMPUTATIONAL TIME COMPLEXITY** of a problem is defined as the number of steps that it takes to solve a function as a function of the size of the input, which is usually measured in bits, using the most efficient algorithm.

The **COMPUTATIONAL SPACE COMPLEXITY** of a problem is defined as the volume of the memory used by the algorithm required to solve an instance of the problem as a function of the size of the input measured in bits and using the most efficient algorithm.

An axiomatic approach to these two ideas exists and allows the classification of problems into complexity classes such as polynomial P and non polynomial NP time.

- The travelling salesman problem can be solved in time $O(n^{2^n})$, where n is the size of the network to visit. As the size of the network of cities grows, the time needed to find the route grows more than exponentially.
- The binary search is said to run in a number of steps proportional to the logarithm of the length of the list being searched, $O(\log(n))$.

In theoretical analysis of algorithms it is common to estimate their complexity asymptotically. Asymptotic estimates are used because different implementations of the same algorithm may differ in efficiency. However the efficiencies of any two "reasonable" implementations of a given algorithm are related by a constant multiplicative factor called a **HIDDEN CONSTANT**.

All this seems and in fact is well defined. Clearly with a nice linear algorithm things will work well but we need to consider cases where the algorithm is a very complicated tree.

Intuitively we clearly need to define complexity as the average time taken through the algorithm. There are two things we can mean by time.

- The actual time taken to complete the calculation.
- The number of operations, steps, required to complete the algorithm.

Clearly the first of these is very dependent on the computer chip and is only really useful as a tool to compare different algorithms on the same computer. Many computers and programs have a function which will track the time and print out the result for you. This can be very useful provided you keep in mind the limitations. It is an excellent way of comparing different versions of an algorithm on the same machine. It is clearly useless as a comparison of different algorithms across different machines.

The second method is better because it is less dependent on the platform chosen. Here the difficulty is to assign weights to the operations and decide which operations to include. The operations of multiplication and division take much longer than simple addition and subtraction. In more detailed work you might find that operations have weights assigned to them and that a weighted sum is taken as the total.

Remember that different disciplines will have a different focus. For a computational scientist, looking a large data set it would make sense to add the time taken to write to external disc as an operation since this can be slow. Whereas if the data was held inside the computer then writing the data into an array held internally would be quick and would not need to be counted as an operation. The computer scientist is interested in the final step when the algorithm is implemented on a given computer. The size and nature of the data set would be a consideration. A mathematician would concentrate on the properties of the algorithm in the abstract and before a choice of computer is made. (Different steps in our solution process.)

An algorithm can be very linear and then the time taken through it is easy to calculate. Many algorithms have more of a tangled tree structure and then we are forced to talk about mean times through the algorithm. In such cases it often pays to look at worst

and best case scenarios. For example if you are given a list to sort and it is already in alphabetical order then the sort program to put it into alphabetical order will drop straight through and finish in a short time. If you are given the same list, maximally disordered with respect to your sorting algorithm, then it will take a lot longer for the program to run and sort the list. In some CATAM projects you are asked to consider worst and best case scenarios. It obviously makes sense to worry about the worst case if that might cause the program to fail on a particular computer. It is therefore in your interest to ask yourself in any particular project whether the structure of your algorithm is sufficiently complicated for it to be worth looking at the worst and best cases. As always in a given CATAM project you should use your common sense. Do not shoot off along a tangent just because worst and best cases were mentioned today but do make a comment if the problem merits a remark.

When you are calculating complexity it is often not necessary to count every detailed step. Most algorithms have a few start and finish steps and then some loops. If you know that the loops will be run many times within the algorithms then there is no point in counting the steps which are only carried out once. Since you are interested in “large n ” behaviour you can afford to ignore all but the dominant contribution. In your answer you should give the asymptotic form, that is the leading term only. I would include the hidden constant in the term unless your specific CATAM instructions tell you to ignore it but very often people do not include this constant because it is compiler dependent. Whichever you decide to do you must explain in your write-up why you have made this choice. In a CATAM calculation of complexity you should explain how you count the operations. If you are not sure whether to include an operation then ask the Catam Helpline. If that fails then ask around to find out how long the operation takes. This may not be easy to find out! Once you know then you can say in your write up that, for example, you are including checking for equality as an arithmetic operation because it takes a time which is comparable or greater than the standard operations of addition and subtraction. If you do not know then make an executive decision and justify it as best you can in your write up.

One further level of complication can arise, and this is one reason why I might include the hidden constant in my estimate of complexity. If you are inverting an n by n matrix once then you will have a contribution of n^3 from the Gauss-Jordan elimination due to the size of the matrix. If you are repeating the operation a million times in your implemented algorithm, which in real life is not an uncommon situation, then you will need to take $10^6 n^3$ which is a big difference if $n = 1000$. In other words you may have more than one “ n ” and you will need to string the contributions together. In this case we have mn^3 . This final consideration should not arise in CATAM.

REF: For nice list of complexities.

http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

REF: Example calculation source material, modified.

http://en.wikipedia.org/wiki/Analysis_of_algorithms

EXAMPLE: Evaluating run-time complexity

The run-time complexity for the worst-case scenario of a given algorithm can sometimes be evaluated by examining the structure of the algorithm and making some simplifying assumptions. Consider the following code.

```

Step 1  Read the first number n from the input vector
Step 2  If the number is greater than ten
Step 3      then print " Large"
Step 4      For i = 1 to n
Step 5          For j = 1 to i
Step 6              Print i*j
Step 7  Print "Finished."
```

In the algorithm above, steps 1, 2 and 7 will only be run once. For a worst-case evaluation, it should be assumed that step 3 will be run as well. Thus the total amount of time to run steps 1-3 and step 7 is:

$$T_1 + T_2 + T_3 + T_7.$$

We still need to evaluate the loops in steps 4, 5 and 6.

The outer loop test in step 4 will execute $(n + 1)$ times. The extra step is needed to terminate the for loop. This takes $T_4(n + 1)$ time.

The inner loop, on the other hand, is governed by the value of i , which iterates from 1 to n . On the first pass through the outer loop, j iterates from 1 to 1. The inner loop makes one pass, so running the inner loop body, step 6, takes T_6 time, and the inner loop test, step 5, takes $2T_5$ time. During the next pass through the outer loop, j iterates from 1 to 2. The inner loop makes two passes, so running the inner loop body, step 6, consumes $2T_6$ time, and the inner loop test, step 5, consumes $3T_5$ time.

Altogether, the total time required to run the inner loop body can be expressed as an

$$T_6 [1 + 2 + 3 + \dots + (n - 1) + n] = T_6 \left[\frac{1}{2}(n^2 + n) \right]$$

The total time required to run the inner loop test can be evaluated similarly.

$$T_5 [1 + 2 + 3 + \dots + (n - 1) + n + (n + 1)] - T_5 = T_5 \left[\frac{1}{2}(n^2 + 3n + 2) \right] - T_5$$

Therefore the total running time for this algorithm is:

$$f(n) = T_1 + T_2 + T_3 + T_7 + (n + 1)T_4 + \left[\frac{1}{2}(n^2 + n) \right] T_6 + \left[\frac{1}{2}(n^2 + 3n + 2) \right] T_5 - T_5$$

$$f(n) = \left[\frac{1}{2}(n^2 + n) \right] T_6 + \left[\frac{1}{2}(n^2 + 3n) \right] T_5 + (n + 1)T_4 + T_1 + T_2 + T_3 + T_7$$

Usually we can assume that the highest-order term in any given function dominates its rate of growth and thus defines its run-time order. In this example, n^2 is the highest-order term, so we conclude that $f(n) = O(n^2)$.

CLAIM:

$$\left[\frac{1}{2}(n^2 + n) \right] T_6 + \left[\frac{1}{2}(n^2 + 3n) \right] T_5 + (n+1)T_4 + T_1 + T_2 + T_3 + T_7 \leq cn^2, n \geq n_0$$

Proof

For positive integer n we have

$$\begin{aligned} & \left[\frac{1}{2}(n^2 + n) \right] T_6 + \left[\frac{1}{2}(n^2 + 3n) \right] T_5 + (n+1)T_4 + T_1 + T_2 + T_3 + T_7 \\ & \leq (n^2 + n)T_6 + (n^2 + 3n)T_5 + (n+1)T_4 + T_1 + T_2 + T_3 + T_7 \end{aligned}$$

Let k be a constant greater than or equal to $[T_1..T_7]$

$$\begin{aligned} T_6(n^2+n) + T_5(n^2+3n) + (n+1)T_4 + T_1 + T_2 + T_3 + T_7 & \leq k(n^2+n) + k(n^2+3n) + kn + 5k \\ & = 2kn^2 + 5kn + 5k \leq 2kn^2 + 5kn^2 + 5kn^2 = 12kn^2 \end{aligned}$$

Therefore

$$\left[\frac{1}{2}(n^2 + n) \right] T_6 + \left[\frac{1}{2}(n^2 + 3n) \right] T_5 + (n+1)T_4 + T_1 + T_2 + T_3 + T_7 \leq cn^2, n \geq n_0$$

for $c = 12k, n_0 = 1$

■

One final word of advice.

CATAM is about using your common sense as well as your mathematical training to understand a given project. You only have to demonstrate you thinking to do well.

Good Luck