Course Information

Instructor Kapil Hari Paranjape

Number MTH409

Title Computational Methods

Classes

- Theory Class on Tuesday, Wednesday Thursday 17:00-17:55 AB1A.
- Lab Sessions on Monday 14:00-17:00

Evaluation

- 40 marks for laboratory sessions/assignments/projects.
- 15x2 marks for mid-semester examinations.
- 30 marks for end-semester examination.

Course Outline

- Numerical Computation: Number representation, machine precision, round-off errors, truncating errors, random number generation, available numerical software.
- Linear Algebra: Solving systems of linear equations, finding inverses of matrices, Gauss-Jordan elimination, Gaussian elimination, LU decomposition, ill-conditioned systems, iterative methods (Jacob's method, Gauss-Seidel, Relaxation methods) and convergence; eigen values and eigen vectors, characteristic polynomial, power methods, Jacobi's method, QR method.
- Curve Fitting: Interpolation techniqes (Newton, Lagrange), difference formulas, cubic splines, method of least squares, two-dimensional in- terpolation.
- Root Finding: Bisection, False position, Newton-Raphson methods, contraction mapping methods, roots of polynomials.
- Numerical Differentiation and Integration: numerical differentiation, Newton-Cotes integration formulas, Romberg integration, Quadratures, improper integrals, multiple integrals.
- Differential Equations: Euler's method, Runge-Kutta methods, multi- step methods, Bulirsch-Stoer extrapolation methods, boundary value problems.
- PDEs: Elliptic equations, one-dimensional and two-dimensional parabolic and hyperbolic equations.

- Real-Life Examples: Google search engine, 1D and 2D simulations, weather forecasting.

Some books

- "Introduction to Numerical Methods" by Jeffrey R. Chasnov Based on course MTH3311 at Hong Kong University of Science and Technology, Copyright 2012, Jeffrey Robert Chasnov
- "Fundamental Numerical Methods and Data Analysis" by George W. Collins, II Available online
- "Lectures on Numerical Analysis" by Dennis Deturck and Herbert S. Wilf Copyright 2002, Dennis Deturck and Herbert Wilf.
- "Introduction to Numerical Analysis and Applications" by James F. Epperson, Copyright 2013, John Wiley and sons.
- "Introduction to Numerical Analysis (2nd Ed)" by F. B. Hildebrand, Copyright 1987, Dover Publications.
- "Applied Numerical Methods for Engineers" by Robert J. Schilling and Sandra L. Harris, Copyright 1999, Thomson-Brooks/Cole.
- Numerical Methods An online resource for many running programs.
- Numerical Analysis A brief introduction on Wikipedia.
- Holistic Numerical Methods An online resource for a lot of material.

Lab Rules

- Use any computer in the e-classroom. Login/Username: iiser and Password: iiser.
- Every assignment must be submitted in three parts: (a) Mathematical Explanation (b) Program (c) Results/conclusions. Submissions as Sage worksheets are also possible.
- The task of implementing programs belongs to the student. The instructor will assist to the extent possible.

Computational Mathematics

Mathematics without computation is like an orchestra without music. All mathematics *requires* computation; in some cases easy, in other cases a bit more complicated. So it may seem strange to have a course on "Computational Mathematics".

There are two types of computations that one may want to carry out. The first type can be called "exact" or "direct" computations. A typical example is the multiplication of two natural numbers. In such cases, the recipe for carrying out the computation has no need for a check-based iteration like a "while loop". Other than physical errors in the computation process, there is no likelihood of errors and thus we need not estimate errors. Finally, the verification of the correct-ness of the answer (if already given) can also be carried out by a similar exact recipe. (A more formal understanding of the notion of "exact" computation would be based on the notion of primitive recursive functions.)

However, in this course, we are primarily interested in computations that result in "approximate" answers. The procedure that we carry out will require iteration or recursion *until* an appropriate level of accuracy is reached. The result will have an error and must estimate the error, either before the calculation is carried out or as part of the calculation. Finally, the verification of the correct-ness may also require a similar calculation.

Instead of talking in generalities, let us look at an example. The familiar series for the number e is given by $\sum_{n=0}^{\infty} 1/n!$. Look up standard tables we see that $e = 2.718281828\ldots$. Let us try to make computational sense of these statements. First of all, we see that the latter statement can be written more mathematically as e lies in the (half-open) interval $[2.718281828, 2.718281829)$. Secondly, we need to *prove* that the above series converges to a number within this range. As usual, we define the partial sums $e_N = \sum_{n=0}^{N} 1/n!$ and note that this is an increasing sequence. Secondly, we note (by domination by a geometric series) that $e - e_N < (1/N!)(1 + 1/N) = b_N$. Now we can simultaneously calculate b_N along with e_N and as soon as $b_N < 10^{-10}$ we can stop our computation. Now, if we can now show that $e_N \in [2.7182818280, 2.7182818289)$ then we know that $e < e_N + b_N < 2.718281829$. In this case, we can even estimate the required N before doing the calculation.

There is a catch hidden in the above paragraph! Keeping track of a fraction like e_N (which has a large denominator) is a computationally expensive procedure as you will quickly see if you try to do it by hand. So it is much easier and quicker to *approximate* $1/n!$ by its decimal expansion and carry out the computation of e_N using these approximations. However, this means that we must also account for the errors introduced by such *imprecision* in the intermediate storage of numbers that we use in our calculation. The number of digits (or bits in the case of binary computation) that are stored is refered to as the *precision* of storage.

To see this more clearly, let us consider a series like $\sum_{n=0}^{\infty}(-1)^n/(n+1)$. Since this is an alternating series, the difference between its sum s and its N-th partial sum s_N is at most the size of the last term; in other words $|s - s_N| < 1/(N+1)$. So to get an answer that is correct upto 5×10^{-4} (roughly, but not quite(!) 3 places of decimal), we need to calculate about 50000 terms! We see that s is between $1/2$ and 1, so if we *only* keep 4 places of decimals in our storage, then in the last 40000 steps of our calculation we will be adding (or subtracting) a number that will *not* change the partial sum. In other words, our answer

is *guaranteed* to be wrong! Even if we keep 5 or 6 places of decimal in our storage, we are accumulating errors due to imprecision in storage quite rapidly. One way to solve this problem is to (a) store numbers with an exponent (for example, $1/40000$ is stored as 0.25×10^{-4}) and (b) start adding and subtracting will *small* numbers first! As we shall see there are still errors that creep in because of subtractions involving numbers that are close to each other; monotonic procedures are better!

As a result of limiting the precision of storage, we have put an upper limit on the accuracy of our answer. However, (and this is important!) this is *not* a lower limit as we have seen above. The above series is an example of an *unstable* computation. A stable computation may be thought of as one where we can keep some control over the errors that creep in due to imprecision in storage of numbers. We shall see how to gain this important feature where possible.

However, there will be situations where ensuring stability of a computation may not be possible. Non-linear systems often exhibit computational instability. In such cases, we must change the question that we are asking! Instead of requiring our computation to be point-wise correct, we can require that it be correct in the root-mean-square sense (also called the L^2-norm). In other words, we change the *sense* of approximation by giving a different metric of accuracy and try to achieve accuracy in this sense.

To summarise, this course is about correctly identifying a suitable notion of approximation for a given problem and then trying to achive that approximation through computation. Our computational resources are limited in terms of running time and storage space, so we need to devise procedures that are reasonably efficient in those terms *without* sacrificing mathematical correct-ness.