ECE 204 *Numerical methods*

A three-page course summary

Course by Douglas Wilhelm Harder

Engineers use computers to process data and to make decisions based on that data. Interestingly enough, engineers prefer, where possible, using integers as opposed to floating-point arithmetic in deployed systems: integer arithmetic uses less power and is faster. Thus, if it is known that non-zero values of significance are bounded between $[2^m, 2^n - 2^m]$ where $n - m \leq 4$, 8, 16, 32 or 64, and where the minimum tolerable absolute error is 2^m engineers will use a 4-, 8-, 16-, 32- or 64-bit, respectively, integer N to represent the value $N \times 2^m$. If integers are used in this manner, the critical issues are overflow and underflow, and this is exactly the cause of the failure of the first Ariane 5 launch. However, if errors such as this can be avoided, such a representation is very efficient.

However, that is not what this course is about. There are times when such representations are unacceptable, and thus, it is necessary to have a better representation of real numbers. For example, when simulating a system that is described by differential equations, or finding a value when the minimum acceptable precision does not allow the above integer representation of real numbers. For this, we need a different representation of real numbers, and the format we introduce is the floating-point representation, implemented as `float` and, more preferably, `double` in C++ and other programming languages. This representation is superior to the representation discussed above, but it still has its weaknesses, and we will discuss these weaknesses. Throughout this course, we will be solving mathematical problems, and at each step, we will take precautions to avoid the weaknesses of this representation.

Thus, after we define some terms such as absolute error, relative error, percent relative error, accuracy, precision, and significant digits, we proceed to describe the floating-point representation. We start by using an equivalent but simpler six-decimal-digit representation that allows us to explore the issues of this representation. Next, we look at the binary double-precision floating-point representation and observe how the weaknesses we saw with our decimal-equivalent format translate to this representation as well. We will observe that $x + (y + z)$ may no longer equal $(x + y) + z$, that $x + y$ may equal x even if y is not zero, and that if x and y are approximately equal, then the floating-point calculation $x - y$ may be a very good approximation of the actual difference.

Thus, the next chapter introduces seven *tools* that we will use to approximate solutions to mathematical problems while using floating-point numbers. These include using appropriate weighted averages, iteration, linear algebra, interpolation, Taylor series, bracketing and the intermediate-value theorem. We discuss each of these seven tools, and where possible we use previously introduced tools to approximate solutions to problems described by subsequent tools, such as using iteration to approximate solutions to systems of linear equations with Jacobi's method. These tools will be used either to derive algorithms or analyze those algorithms.

We then proceed by discussing the various sources of error, and describing those errors that we will address in this course, specifically, those errors associated with the use of the floating-point representation. We also introduce the four categories of mathematical problems that we will address in this course, including:

1. approximating the value of a real- or vector-valued expression;
2. approximating the solution to an algebraic equation or system of algebraic equations, including both linear equations (using linear algebra) and non-linear equations;
3. approximating the solution to an analytic equation or system of analytic equations, specifically, ordinary or partial differential equations or systems of ordinary or partial differential equations, together with either initial or boundary values; and
4. approximating the solution to unconstrained optimization problems.

For the first topic, we will begin by seeing how to efficiently evaluate a polynomial using Horner's rule, and it will be a technique you would probably never have considered in secondary school. Next, we will assume we are periodically reading a signal and we would like to approximate the value of that signal between readings, approximate the rate of change, or derivative, of that signal, and approximate the integral of that signal. We will do this using two approaches: using interpolating polynomials under the assumption that the readings of the signal are relatively noise free, and then using least-squares best-fitting polynomials if there is assumed to be significant noise in the read signal. The least-squares best-fitting polynomial approach can be best described as follows: suppose that we have two or three linearly independent vectors $\mathbf{u}_1$ and $\mathbf{u}_2$, or $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$, and we have a target vector $\mathbf{v}$ such that $\mathbf{v}$ is not a linear combination of either the two or three vectors. Our question is, "what is the linear combination of either the two or three vectors that is *closest to* the vector $\mathbf{v}$?"

For the second topic, we start by observing that it is generally possible to solve $ax = b$ for x, a single linear equation in a single unknown, but that something as trivial as the solutions to a quadratic equation cannot be reliably approximated using floating-point arithmetic while using the quadratic formula you saw in secondary school. Having removed what misplaced faith we may have had in simple arithmetic, we proceed to look at six different methods for approximating the solution to a single non-linear equation in a single unknown: Newton's method, the bisection method, the bracketed secant method (my name for the false-position method), the secant method, Muller's method, and inverse quadratic interpolation. Instead of solving a non-linear equation, we will instead always convert a non-linear equation into a root-finding problem: $f(x) = 0$. Each of these methods will use a different set of tools and assumptions. This will be concluded by the observation that in general we will use either Newton's method if information about the derivative is available, the secant method if information about the derivative is not available, but we do not have a bound on the solution, either the bisection, the bracketed secant method or inverse quadratic interpolation if the root is bracketed, or Muller's method if we are trying to find the root of a polynomial where that root may be complex. Next, we look at additional techniques for speeding up approximating the solution to a system of linear equations using the Gauss-Seidel method and the method of successive over-relaxation. We conclude by using a generalization of Newton's method to approximate a solution to a system of n non-linear equations in n unknowns while asking what it would take to perform a similar generalization of the secant method.

For the third topic, we look at approximating solutions to analytic equations. Now, analytic equations tend to include differential, integral, and integro-differential equations; however, engineers prefer to convert all such problems into ones using ordinary differential equations (the same way we convert a non-linear equation into a root-finding problem). We will look at both ordinary and partial differential equations:

> For ordinary differential equations, we start by observing that once we have not only the value of functions at points but also the derivatives at those points, we can now use better suited splines as opposed to interpolating polynomials. Next, we look at initial-value problems, first solving a single ordinary differential equation with a single initial value. We do so by using three techniques including Euler's, Huen's and the classic 4^{th}-order Runge-Kutta method. We then observe the weaknesses of such techniques and thus look for adaptive techniques, were we begin with a novel use of both Euler and Huen's method, and then describe the Dormand-Prince method. Having looked at all of these, we observe that these same techniques can be used to approximate a system of 1^{st}-order initial-value problems, and we can also convert higher-order initial-value problems and systems of higher-order initial-value problems into systems of 1^{st}-order initial-value problems. Next, we look at approximating solutions to boundary-value problems, first by using the shooting method that uses iteration and the previous initial-value problem solvers, and then by introducing finite-difference methods that convert ordinary-differential equations into finite-difference equations using the approximations of the derivatives we saw in the last topic. We also introduce Neumann boundary conditions and the special case of insulated boundary conditions.
>
> For partial differential equations, we look at approximating solutions to the heat equation and the wave equation but only in one spatial dimension. We then look at approximating solutions to Laplace's equation in two and three spatial dimensions, and then discuss how we can approximate solutions to the heat and wave equations in two and three dimensions.

For the fourth, and final, topic, we look at unconstrained optimization problems. To begin, we convert any optimization problem to a minimization problem. We then see that if we have information about the derivative of the function we are optimizing, the problem is equivalent to a root-finding problem on the derivative, so Newton's method; however, if information about the derivative is not available, we need to use different approaches, including the golden-ratio search and successive parabolic interpolations. For unconstrained optimization problems of two or more variables, we describe the Hooke-Jeeves method when no information about the partial derivatives is available, a generalization of Newton's method if information about the partial derivatives and second partial derivatives are available, and gradient descent when the partial derivatives can be approximated.

This concludes the course, where we described the problems with using a floating-point approximation of real numbers, where we introduced seven tools to approximate solutions to mathematical problems, and then looked at four categories of mathematical problems where we used the seven tools over and over again to find approximates to these problems.