1. Introduction. Computer Precision

NUMERICAL ANALYSIS. Prof. Y. Nishidate (323-B, nisidate@u-aizu.ac.jp)
http://web-int.u-aizu.ac.jp/∼nisidate/na/

What Is Numerical Analysis

Numerical analysis is a way to do higher mathematics problems on a computer, a technique widely used by scientists and engineers to solve their problems. A major advantage for numerical analysis is that a numerical answer can be obtained even when a problem has no ”analytical” solution. Further, the only mathematical operations required are addition, subtraction, multiplication, and division plus the making of comparisons. Since these simple operations are the only functions that computers can perform, computers and numerical analysis make a perfect combination. For example, the following integral, which gives the length of one arch of the curve of $y = \sin(x)$, has no closed form solution:

$$\int_0^{\pi} \sqrt{1+\cos^2(x)}dx$$

Numerical analysis can compute the length of this curve by standard methods that apply to essentially any integrand; there is never a need to make special substitutions nor to do integration by parts to get the result.

It is important to realize that a numerical analysis solution is always numerical. Analytical methods usually give a result in terms of mathematical functions that can then be evaluated for specific instances. There is thus an advantage of the analytical result, the exact solution as a function become apparent. This is not the case for purely numerical results, though numerical results can plot some of the behavior of the solution.

Another important distinction is that the result from numerical analysis is an approximation, but results can be made as accurate as desired (there are limitations to the achievable level of accuracy, because of the way that computers do arithmetic). To achieve high accuracy, huge number of arithmetic operations must be carried out, but computers do them so rapidly without ever making mistakes that this is no significant problem. The analysis of computer errors and the other sources of error in numerical methods is a critically important part of the study of numerical analysis.

Here are some of the operations that numerical analysis can do and that are presented in this course:

- Errors of floating-point computations. Function evaluations.
- Finding the zero of a function.
- Linear algebra. Vectors and matrices. Linear equations.
- Matrix inversion. Matrix eigenvalues.
- Methods for sparse matrices.
- Interpolation. Curve fitting.
- Cubic spline interpolation and fitting to scattered data.
- Random Number Generation and Monte Carlo Method.
- Numerical differentiation and integration.
- Ordinary differential equations.
- Partial differential equations.
- Finite difference method.
- Finite element method. Discretization. Elements.
- Finite element method. Assembly. Solution.

Computer Languages

Traditionally numerical computations were done in FORTRAN and C. Later, object-oriented programming (OOP) has been welcomed for its benefits of reliability, easy debugging and code reuse. C++ and Java are the most familiar examples in our curriculum. We are going to use Java in exercise computing because of its multi-platform nature and ease of memory management. However, if you like the other programming language you can present exercise problems in it.

Object-Oriented Concepts

Three properties are considered essential for object-oriented software:

1. data encapsulation
2. class hierarchy and inheritance
3. polymorphism

Data encapsulation is the fact that each object hides its internal structure from the rest of the system. Data encapsulation is in fact a misnomer since an object usually chooses to expose some of its data. Instead of the word ”encapsulation” it is possible to use the expression *hiding the implementation*, a more precise description of what is usually understood by data encapsulation. Hiding the implementation is a crucial point because an object, once fully tested, is guaranteed to work ever after. It ensures an easy maintainability of applications because the internal implementation of an object can be modified without impacting the application, as long as the public methods are kept identical.

Class hierarchy and inheritance is the keystone implementation of any object-oriented system. A class is a description of all properties of all objects of the same type. These properties can be structural (static) or behavioral (dynamic). Static properties are mostly described with instance variables. Dynamic properties are described by methods. Inheritance is the ability to derive the properties of an object from those of another. The class of the object from which another object is deriving its properties is called the superclass. A powerful technique offered by class hierarchy and inheritance is the overloading of some of the behavior of the superclass.

Polymorphism is the ability to manipulate objects from different classes, not necessarily related by inheritance, through a common set of methods. Polymorphism deals with decoupling in terms of types. The polymorphic method call allows one type to express its distinction from another, similar type, as long as they are both derived from the same base type. This distinction is expressed through differences in behavior of the methods that you can call through the base class.

Data Types in Java

Primitive type	**Size**	**Minimum**	**Maximum**
boolean	-	-	-
char	16-bit	0	$2^{16}-1$
byte	8-bit	-128	+127
short	16-bit	-2^{15}	$+2^{15}$-1
int	32-bit	-2^{31}	$+2^{31}$-1
long	64-bit	-2^{63}	$+2^{63}$-1
float	32-bit	$\pm 1.4 \cdot 10^{-45}$	$\pm 3.4 \cdot 10^{38}$
double	64-bit	$\pm 4.9 \cdot 10^{-324}$	$\pm 1.8 \cdot 10^{308}$

Floating Point Representation

In present-day computers, a floating-point number is represented as

$$m \cdot r^e$$

where the radix r is a fixed number, generally 2. On some exotic machines, however, the radix can be 10 or 16. Thus, each floating-point number is represented in two parts: an integral part called the mantissa m and an exponent e. This approach is quite familiar to people using large quantities (e.g., astronomers) or studying the microscopic world (e.g., microbiologists). Of course, the natural radix for people is 10. For example, the average distance from Earth to the sun expressed in kilometers is written as $1.4959787 \cdot 10^8$.

In the case of radix 2, the number 18446744073709551616 is represented as $1 \cdot 2^{64}$. Quite a shorthand compared to the decimal notation! IEEE standard floating-point numbers use 23 bits for the mantissa (about 7 decimal digits) in single precision; they use 52 bits (about 15 decimal digits) in double precision. One important property of floating-point number representation is that the relative precision of the representation-that is, the ratio between the precision and the number itself – is the same for all numbers except, of course, for the number 0.

Finding the Numerical Precision of a Computer

Some numerical algorithms are carried until the estimated precision of the result becomes smaller than a given value, called the *desired precision*. Since an application can be executing on different hardware, the desired precision is best determined at run time.

Let us determine the parameters of the floating-point representation of a particular computer relevant for numerical computations. These parameters are the following:

radix The radix of the floating-point representation—that is, r

machinePrecision The largest positive number that when added to 1 yields 1

negativeMachinePrecision The largest positive number that when subtracted from 1 yields 1

smallestNumber The smallest positive number different from 0

largestNumber The largest positive number that can be represented in the machine

defaultNumericalPrecision The relative precision that can be expected for a general numerical computation

smallNumber A number that can be added to some value without noticeably changing the result of the computation

Computing the radix

Computing the radix r is done in two steps. First, one computes a number equivalent of the machine precision (see the next paragraph) assuming the radix is 2. Then, one keeps adding 1 to this number until the result changes. The number of added 1s is the radix.

Computing the machine precision

The machine precision is computed by finding the largest integer n such that

$$(1 + r^{-n}) - 1 \neq 0$$

This is done with a loop over n. The quantity $\epsilon_+ = r^{-(n+1)}$ is the machine precision.

Negative machine precision

The negative machine precision is computed by finding the largest integer n such that

$$(1 - r^{-n}) - 1 \neq 0$$

Computation is made as for the machine precision. The quantity $\epsilon_- = r^{-(n+1)}$ is the negative machine precision. If the floating-point representation uses two-complement to represent negative numbers, the machine precision is larger than the negative machine precision.

Smallest and largest numbers

To compute the smallest and largest numbers, one first computes a number whose mantissa is full. Such a number is obtained by building the expression

$$f = 1 - r \cdot \epsilon_-$$

The smallest number is then computed by repeatedly dividing this value by the radix until the result produces an underflow. The last value obtained before an underflow occurs is the smallest number. Similarly, the largest number is computed by repeatedly multiplying the radix until an overflow occurs. The last value obtained before an overflow occurs is the largest number.

Default numerical precision

defaultNumericalPrecision is an estimate of relative precision that can be expected for a general numerical computation. This value of the default numerical machine precision is defined as the square root of the machine precision:

$$\epsilon = \sqrt{\epsilon_+}$$

For example, one should consider that two numbers a and b are equal if the relative difference between them is less than the default numerical machine precision:

$$\frac{|a - b|}{\max(|a|, |b|)} < \epsilon$$

Small number

smallNumber contains a value that can be added to some number without noticeably changing the result of the computation. In general, an expression of the type $\frac{0}{0}$ is undefined. In some particular case, however, one can define a value based on a limit. For example, the expression $\frac{\sin x}{x}$ is equal to 1 for x = 0. For algorithms, where such an undefined expression can occur (of course, after making sure that the ratio is well defined numerically), adding a small number to the numerator and the denominator can avoid the division by zero exception and can obtain the correct value. This value of the small number has been defined as the square root of the smallest number that can be represented on the machine.