# Algorithms

April 4, 2023

## 0.1 STAT 207: Algorithms and Programs

### 0.1.1 Zhe Fei (zhe.fei@ucr.edu)

### 0.1.2 Algorithms

Loosely speaking, algorithm is a method or a set of instructions for doing something.

Algorithms are sometimes distinguished as

- Numerical
- Seminumerical
- Nonnumerical

The program is the set of computer instructions that implement the algorithm.

- A poor implementation can render a good algorithm useless.
- A good implementation will preserve the algorithm's accuracy and efficiency, and will detect data that are inappropriate for the algorithm.

A robust algorithm is applicable over a wide rand of data to which it is applied.

A robust program, which is more important, is one that will detect input data that are inappropriate either for the algorithm or for the implementation in the given program.

Two most important aspects of a computer algorithm:

- Accuracy
- Efficiency

**Error in Numerical Computations** The basic source of error in numerical computations is the inability to work with the reals.

- absolute error, $|\tilde{r} - r|$
- relative error, $|\tilde{r} - r|/|r|$

What if $r$ is not a single real number? For example, in statistical data analysis, the numerical result, ˜r, may consist of estimates of several regression coefficients, various sums of squares and their ratio, and several other quantities.

$$\Delta(\tilde{r}, r),$$

where $\Delta(\cdot, \cdot)$ is a nonnegative, real-valued function.

If $r$ is a vector, the measure may be based on some norm, $\|\tilde{r} - r\|$.

- For a given algorithm, the amount of error to expect or some bound on the error.
- Alternatively, an upper bound on the error.
- Or, an estimate of an "average error" based on some assumed probability distribution of the data comprising the problem.

**Order of Error** In general, a function $f(t)$ is said to be of order $g(t)$ at $t_0$, written $O(g(t))$ ("big O of $g(t)$"), if there exists a positive constant $M$ such that

$$|f(t)| \leq M|g(t)| \quad \text{as } t \to t_0.$$

This is the order of convergence of one function to another function at a given point. Notice that this is pointwise convergence; we compare the functions near the point $t_0$.

For example, if $\tilde{f}(t)$ is a finite series approximation to $f(t)$ using $k$ terms, we may express the error as $O(h(k))$ for some function $h(k)$. Typical orders of errors due to the approximation may be

- $O(1/k)$
- $O(1/k^2)$
- $O(1/k!)$

The special case of convergence to the constant zero is often of interest. A function $f(t)$ is said to be "little o of $g(t)$" at $t_0$, written $o(g(t))$, if

$$f(t)/g(t) \to 0, \quad \text{as } t \to t_0.$$

If the function $f(t)$ approaches 0 at $t_0$, $g(t)$ can be taken as a constant and $f(t)$ is said to be $o(1)$.

A function $f(t)$ is said to be $\Omega(g(t))$ ("big omega of $g(t)$") if there exists a positive constant $m$ such that

$$|f(t)| \geq m|g(t)| \quad \text{as } t \to t_0.$$

Likewise, a function $f(t)$ is said to be "little omega of $g(t)$" at $t_0$, written $\omega(g(t))$, if

$$g(t)/f(t) \to 0, \quad \text{as } t \to t_0.$$

Usually the limit on $t$, that is, $t_0$, in order expressions is either 0 or $\infty$.

### 0.1.3 Algorithms and Data

The performance of an algorithm may depend on the data.

Heuristically, data for a given problem are ill-conditioned if small changes in the data may yield large changes in the solution.

- Condition of Data
- Robustness of Algorithms
- Stability of Algorithms

### 0.1.4 Efficiency

The efficiency of an algorithm refers to its usage of computer resources:

- processing units
- memory (storage)

A limiting factor for the time the processing units are in use is the number and type of operations required.

In numerical computations, the most important types of computation are usually the **floating-point operations**. The actual number of floating-point operations divided by the number of seconds required to perform the operations is called the **flops (floating-point operations per second)** rate.

**Measuring Efficiency** Often, instead of the exact number of operations, we use the *order* of the number of operations in terms of the measure of problem size.

If $n$ is some measure of the size of the problem, an algorithm has order $O(f(n))$ if, as $n \to \infty$, the number of computations $\to cf(n)$, where $c$ is some constant that does not depend on $n$.

- to multiply two $n \times n$ matrices in the obvious way requires $O(n^3)$ multiplications and additions;
- to multiply an $n \times m$ matrix and an $m \times p$ matrix requires $O(nmp)$ multiplications and additions.

In the latter case, $n, m$, and $p$ are all measures of the size of the problem.

### 0.1.5 Recursion

Recurrence relations are ubiquitous in computational statistics and probability.

**Binomial Coefficients**

Let $\binom{n}{k}$ be the number of subsets of size $k$ from a set of size $n$. Pascal's triangle is the recurrence scheme specified by

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k},$$

with the boundary conditions $\binom{n}{0} = \binom{n}{n} = 1$.

**Number of Partitions of a Set**

Let $B_n$ be the number of partitions of a set with $n$ elements. By a partition we mean a division of the set into disjoint blocks.

Starting with $B_0 = 1$, the $B_n$ satisfy the recurrence relation

$$\begin{aligned} B_{n+1} &= \sum_{k=0}^{n} \binom{n}{k} B_{n-k} \\ &= \sum_{k=0}^{n} \binom{n}{k} B_k. \end{aligned}$$

**Horner's Method**

Suppose we want to evaluate the polynomial

$$p(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n$$

for a particular value of x. A naive alrogithm would require $3n - 1$ operations. On the other hand, we can write

$$\begin{aligned} p(x) =& x(a_0x^{n-1} + a_1x^{n-2} + ... + a_{n-1}) + a_n \\ =& xb_{n-1}(x) + a_n. \end{aligned}$$

And do the same for $b_{n-1}(x)$, a complete recursive scheme is

$$\begin{aligned} b_0(x) =& a_0 \\ b_k(x) =& xb_{k-1}(x) + a_k, \quad k = 1, 2, .., n. \end{aligned}$$

This scheme, known as Horner's method, requires only $n$ multiplications and $n$ additions to compute $p(x) = b_n(x)$.

- Horner's method can be modified to produce the derivative $p'(x)$ as well as $p(x)$. This modification is useful, for instance, in searching for a root of $p(x)$ by Newton's method.

**An Unstable Recurrence**

Not all recurrence relations are numerically stable. Consider the integrals

$$y_n = \int_0^1 \frac{x^n}{x+a} dx.$$

The recurrence can be derived as

$$y_n = 1/n - ay_{n-1}.$$

With the initial value $y_0 = \ln \frac{1+a}{a}$. We can compute the sequence with certain precision as below.

[16]:
```
import math

a = 10
y = math.log((1+a)/a)

print(f"y_0 = {y:.4f}")

for n in range(1, 10):
    y = 1/n - a*round(y,5)
    print(f"y_{n} = {y:.4f}")
```

```
y_0 = 0.0953
y_1 = 0.0469
y_2 = 0.0310
y_3 = 0.0233
y_4 = 0.0167
y_5 = 0.0330
y_6 = -0.1633
y_7 = 1.7762
y_8 = -17.6366
y_9 = 176.4771
```

For $n$ moderately large, most of the mass of the integral occurs near $x = 1$. Thus, to a good approximation

$$y_{n-1} \approx \frac{1}{1+a}\int_0^1 x^{n-1}dx = \frac{1}{(1+a)n}.$$

And

$$y_n \approx \frac{1}{n} - a\frac{1}{(1+a)n} = \frac{1}{n}\left(1 - \frac{a}{1+a}\right).$$

We lose precision whenever we subtract two numbers of the same sign and comparable magnitude.

- We must exercise caution in using recurrence relations involving subtraction.
- Fortunately, many recurrences in probability theory arise by conditioning argumentsand consequently entail only addition and multiplication of nonnegative numbers.

### 0.1.6 Improving Efficiency

There are many ways to attempt to improve the efficiency of an algorithm, a brief overview here:

- Divide and Conquer: *NAS 1.10 Quick Sort Algorithm*
- Convolutions: If $f$ and $g$ are PDFs of stochastically independent random variables $U$ and $V$, then $f * g$ is the PDF of $U + V$.
- Discrete Transforms: discrete Fourier transform (DFT), fast Fourier Transform (FFT). Widely used in signal processing, image processing, etc.
- Greedy Methods: design each step to be as efficient as possible, without regard to what future steps may be part of the algorithm.

### 0.1.7 Iterations and Convergence

- "Iterative" algorithms are methods in which groups of computations form successive approximations to the desired solution.
- "Converge", and the various derivatives of this root word, refers to a condition in the progress of an algorithm in which the values no longer change.
- Convergence criterion or stopping criterion:

$$\Delta(x^{(k)}, x^{(k-1)}) \leq \epsilon.$$

There are typically three reasons to terminate an algorithm.

- It is known that the problem is solved.
- The iterations converge, that is, the computed values quit changing.
- The number of iterations is too large, or the problem appears to diverge.

**Rate of Convergence** The *convergence ratio* of the sequence $x^{(k)}$ to a constant $x_0$ is

$$\lim_{k\to\infty} \frac{\Delta(x^{(k+1)}, x_0)}{\Delta(x^{(k)}, x_0)}$$

if this limit exists.

- If the convergence ratio is greater than 0 and less than 1, the sequence is said to converge *linearly*.
- If the convergence ratio is 0, the sequence is said to converge *superlinearly*.
- The convergence rate is often a function of $k$, say $g(k)$. The convergence is then expressed as an order in $k$, $O(g(k))$.

### 0.1.8 Programming

"Programming is only learned by programming." - Gentle, *Computational Statistics*

High-level languages for readily implementations:

- R: `tidyverse`, `ggplot`
- Matlab
- Python: `numpy`, `sklearn`

Low-level languages when developing new algorithms:

- Rcpp
- Cython
- Julia

[ ]:

```

```