

# CMSC 31100 Week 2: Numerical Stability

Lecturer: Ridg Scott

October 8, 2006

## Numerical Computation

For human beings, closed form solutions in mathematics may be preferred. But for digital computer, numerical solutions are better. The shift from closed form solutions to numerical ones guides people to consider the problem of numerical stability. Of course, besides in numerical computation, the concept of stability lies in many aspects in computer science.

## Numerical Stability

There are about two kinds of error in numerical computation,

- Error from floating point
- Error from computation models

This lecture mainly considers the second.

Example: roots for the following equations

$$(1) \quad 0 = 1 - 2bx + x^2$$

$$(2) \quad 0 = \frac{1}{y^2} - 2b\frac{1}{y} + 1$$

Solution for equation (1)

$$(3) \quad x_1 = b - \sqrt{b^2 - 1}$$

$$(4) \quad x_2 = b + \sqrt{b^2 - 1}$$

Solution for equation (2)

$$(5) \quad y_1 = \frac{1}{x_1}$$

$$(6) \quad y_2 = \frac{1}{x_2}$$

Because of floating point error, it is obvious that when  $b$  is large enough,  $b \approx \sqrt{b^2 - 1}$ . This will result in  $x_1 \approx 0$ , which means  $y_1 \rightarrow \infty$ . So in this problem,  $y_1$  is not a stable solution. In the example, when  $b$  is relatively small, the numerical solution will be stable. This shows that for numerical solution, closed form solution directly from the problem might not be the best choice. The same situation also happens in linear equation with singular matrix.

## Differential Equation

Suppose a function is defined by a differential equation

$$(7) \quad \frac{du}{dt} = f(u, t)$$

For example,  $f(u, t) = -u^2$ , and  $u(0) = a$ . Then the closed form solution is

$$(8) \quad u(t) = \frac{1}{\frac{1}{a} + t}$$

From the definition of derivative, we have a straightforward solution

$$(9) \quad u_{i+1} = u_i + hf(u_i, t_i)$$

This is called implicit **Euler** method.

An improved method uses high order to reduce floating point error. In general, this method can be written as

$$(10) \quad a_0 u_n + a_1 u_{n-1} + \dots + a_k u_{n-k} = hf(u_n, t_n)$$

When  $f(u_i, t_i) = 0$ ,  $i > n$ , we have

$$(11) \quad a_0 u_n + a_1 u_{n-1} + \dots + a_k u_{n-k} = 0$$

However, this does not necessarily imply  $\forall i, i > n \quad u_i = 0$ .

### Why?

Suppose there exists  $u_n = \xi^n$ , then we have the following equation

$$(12) \quad p_k(\xi) = a_0 \xi^n + a_1 \xi^{n-1} + \dots + a_k \xi^{n-k} = 0$$

If  $|\xi| > 1$ , when  $n$  increase,  $u_n = \xi^n$  will go up exponentially. This will make the algorithm unstable.

If  $|\xi| < 1$ ,  $u_n = \xi^n$  will go to zero rapidly, and the solution will be stable.

When  $|\xi| = 1$ , we have to examine each of the roots including complex ones.

## Backward Differentiation Formula

$$(13) \quad \sum_{i=0}^k a_i u_{n-i} = \sum_{j=1}^k \frac{1}{j} \nabla^j u_n$$

For example,

$$(14) \quad \nabla^2 u_n = \nabla(\nabla u_n) = \nabla(u_n - u_{n-1}) = (u_n - u_{n-1}) - (u_{n-1} - u_{n-2}) = u_n - 2u_{n-1} + u_{n-2}$$

The condition that  $|\xi| > 1$  for the roots of  $p_k(\xi) = 0$  restricts the order of BDF regarding stability.