CME 302: NUMERICAL LINEAR ALGEBRA
FALL 2005/06
LECTURE 5

GENE H. GOLUB

1. Perturbation Theory

Suppose we want to solve

$$A\mathbf{x} = \mathbf{b}.$$

We actually have an approximation $\boldsymbol{\xi}$ such that

$$\mathbf{x} = \boldsymbol{\xi} + \mathbf{e}.$$

The question is, how can we use norms to bound the relative error in ξ? We define the *residual* $\mathbf{r}$ by

$$\mathbf{r} = \mathbf{b} - A\boldsymbol{\xi} = A(\mathbf{x} - \boldsymbol{\xi}) = A\mathbf{e}$$

Note that $\mathbf{r} = \mathbf{0}$ if $A\mathbf{x} = \mathbf{b}$. From the relations $\|\mathbf{r}\| \leq \|A\|\|\mathbf{e}\|$ and $\|\mathbf{e}\| \leq \|A^{-1}\|\|\mathbf{r}\|$, we obtain

$$\frac{\|\mathbf{r}\|}{\|A\|} \leq \|\mathbf{e}\| \leq \|A^{-1}\|\|\mathbf{r}\|.$$

It follows that the relative error is bounded as follows:

$$\frac{\|\mathbf{r}\|}{\|A\|\|\mathbf{x}\|} \leq \frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq \frac{\|A^{-1}\|\|\mathbf{r}\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

since $\|A\|\|\mathbf{x}\| \geq \|\mathbf{b}\|$. The quantity

$$\kappa(A) = \|A^{-1}\|\|A\|$$

is called the *condition number* of A. The condition number serves as a measure of how perturbation in the data of the problem $A\mathbf{x} = \mathbf{b}$ affects the solution.

How does a perturbation in A affect the solution $\mathbf{x}$? To answer this question, we define $A(\epsilon)$ to be a function of the size of the perturbation ϵ, with $A(0) = A$. Starting with

$$A(\epsilon)A^{-1}(\epsilon) = I$$

and differentiating with respect to epsilon yields

$$A(\epsilon)\frac{dA^{-1}(\epsilon)}{d\epsilon} + \frac{dA(\epsilon)}{d\epsilon}A^{-1}(\epsilon) = 0$$

or

$$\frac{dA^{-1}(\epsilon)}{d\epsilon} = -A^{-1}(\epsilon)\frac{dA(\epsilon)}{d\epsilon}A^{-1}(\epsilon).$$

Now, suppose that $\mathbf{x}(\epsilon)$ satisfies

$$(A + \epsilon E)\mathbf{x}(\epsilon) = \mathbf{b}.$$

Date: October 26, 2005, version 1.0.
Notes originally due to James Lambers. Minor editing by Lek-Heng Lim.

Using Taylor series, we obtain

$$\begin{aligned}\mathbf{x}(\epsilon) &= (A+\epsilon E)^{-1}\mathbf{b} \\ &= \mathbf{x}(0) + \epsilon \left.\frac{d\mathbf{x}(\epsilon)}{d\epsilon}\right|_{\epsilon=0} + O(\epsilon^2) \\ &= \mathbf{x}(0) + \epsilon\left(-A^{-1}(\epsilon)\frac{dA(\epsilon)}{d\epsilon}A^{-1}(\epsilon)\right)\mathbf{b} + O(\epsilon^2) \\ &= \mathbf{x}(0) + \epsilon(-A^{-1}EA^{-1})\mathbf{b} + O(\epsilon^2) \\ &= \mathbf{x}(0) + \epsilon(-A^{-1}E)\mathbf{x} + O(\epsilon^2)\end{aligned}$$

Taking norms, we obtain

$$\|\mathbf{x}(\epsilon) - \mathbf{x}(0)\| \leq |\epsilon|\|A^{-1}\|^2\|E\|\|\mathbf{b}\| + O(\epsilon^2)$$

from which it follows that the relative perturbation in $\mathbf{x}$ is

$$\frac{\|\mathbf{x}(\epsilon) - \mathbf{x}\|}{\|\mathbf{x}\|} \leq |\epsilon|\|A^{-1}\|\|A\|\frac{\|E\|}{\|A\|} + O(\epsilon^2) \leq \kappa(A)\rho + O(\epsilon^2)$$

where $\rho = \|\epsilon E\|/\|A\|$ is the relative perturbation in A.

Since the exact solution to $A\mathbf{x} = \mathbf{b}$ is given by $\mathbf{x} = A^{-1}\mathbf{b}$, we are also interested in examining $(A+E)^{-1}$ where E is some perturbation. Can we say something about $\|(A+E)^{-1} - A^{-1}\|$? We assume that $\|A^{-1}E\| = r < 1$. We have

$$A + E = A(I + A^{-1}E) = A(I - F), \quad F = -A^{-1}E$$

which yields

$$\|(I-F)^{-1}\| \leq \frac{1}{1-r}.$$

From

$$\begin{aligned}(A+E)^{-1} - A^{-1} &= (I + A^{-1}E)^{-1}A^{-1} - A^{-1} \\ &= (I + A^{-1}E)^{-1}(A^{-1} - (I + A^{-1}E)A^{-1}) \\ &= (I + A^{-1}E)^{-1}(-A^{-1}EA^{-1})\end{aligned}$$

we obtain

$$\|(A+E)^{-1} - A^{-1}\| \leq \frac{1}{1-r}\|A^{-1}\|^2\|E\|$$

or

$$\frac{\|(A+E)^{-1} - A^{-1}\|}{\|A^{-1}\|} \leq \frac{1}{1-r}\kappa(A)\frac{\|E\|}{\|A\|}.$$

2. Inexact Computation

2.1. **Number representation.** Real numbers can be represented using *floating-point* notation: a *floating-point representation* such as

$$y = \pm d_1 \cdots d_s d_{s+1} \cdots 10^e$$

A real number may not necessarily have a unique floating-point representation, as the following examples indicate:

$$\begin{aligned}y &= \pm 0.899\cdots 9\cdots \times 10^0 \\ &= \pm 0.90\cdots 0\cdots \times 10^0\end{aligned}$$

or even worse,

$$\begin{aligned}y &= +0.99\cdots 9 \times 10^0 \\ &= +0.10\cdots 0 \times 10^1\end{aligned}$$

How can we represent y? We can use a *chopped representation*

$$\tilde{y} = \pm .d_1 \cdots d_s \times 10^3$$

or possibly

$$\tilde{y} = \pm .d_1 \cdots d_{s-1} \bar{d}_s \times 10^{\bar{e}}$$

where

$$\bar{d}_s = \begin{cases} d_s & d_{s+1} < 5, \\ (d_s + 1) \bmod 9 & d_{s+1} > 5, \\ ? & d_{s+1} = 5. \end{cases}$$

where the value of $\bar{d}_s$ when $d_s = 5$ depends on what convention is used for rounding. Note that e can change if $d_{s+1} > 5$. We often write $\tilde{y} = fl(y)$ where $fl(\cdot)$ means "floating-point representation of y".

Floating point numbers can be represented in base β as

$$y = \pm .d_1 \cdots d_s \times \beta^e$$

where $m \le e \le M$ and the digits d_j satisfy

$$1 \le d_1 \le \beta - 1, \quad 0 \le d_j \le \beta - 1, \quad j = 2, \ldots, s.$$

This is a *normalized floating-point number*. The sequence of *significant* digits $d_1 \cdots d_s$ is called the *mantissa*, and the number e is called the *exponent*.

Suppose $s = 1$, $m = -1$, and $M = 1$. Then the representable numbers are

$$\begin{array}{llll} +0.1 \times 10^{-1} & +0.2 \times 10^{-1} & \cdots & +0.9 \times 10^{-1} \\ +0.1 \times 10^{0} & +0.2 \times 10^{0} & \cdots & +0.9 \times 10^{0} \\ +0.1 \times 10^{1} & +0.2 \times 10^{1} & \cdots & +0.9 \times 10^{1} \end{array}$$

and 27 negative numbers. Note that the distribution is not uniform. In the previous example, the representable numbers are

$$-9, -8, \ldots, -1, -0.9, -0.8, \ldots, -0.1, -0.09, \ldots, -0.01, 0, 0.1, \ldots$$

Note that there are large gaps between integers.

Now, suppose that $s = 3$, and that we multiply two numbers $y_1 = 0.999$ and $y_2 = 0.999$. The exact product is $y_1 \times y_2 = 0.998001$, but the computed product is $fl(y_1 \times y_2) = 0.998$. We will write $fl(y_1 \mathsf{op} y_2)$ to indicate some numerical calculation.

On the IBM 360, we have base 16 (hexadecimal). In general,

$$\begin{aligned} y &= (\pm .d_1 \cdots d_s)_\beta \times \beta^e, \\ &= \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_s}{\beta^s} \right) \times \beta^3, \qquad \beta = 16, \end{aligned}$$

with $1 \le d_1 \le \beta - 1$ and $0 \le d_j \le \beta_1$ for $2 \le j \le s$. For the IBM 360, floating-point numbers are represented using $s = 6$, $m = -64$ and $M = 63$, while double-precision floating-point numbers are represented using $s = 14$, $m = -64$, and $M = 63$. If, for the result of any operation, $e < -64$, then *underflow* has occurred. On the other hand, the scenario $e > 63$ is called *overflow*.

2.2. **Roundoff Error in Arithmetic Operations.** We need to consider the error arising from computing the results of arithmetic expressions. In general,

$$z = fl(x \mathsf{op} y) = (x \mathsf{op} y)(1 + \delta)$$

where $|\delta| \le \beta^{-(s-1)} \equiv \mathsf{u}$ when results are truncated, or $|\delta| \le \frac{1}{2}\beta^{-(s-1)}$ when results are rounded. Therefore the relative error in $fl(x \mathsf{op} y)$ is

$$\left| \frac{fl(x \mathsf{op} y) - (x \mathsf{op} y)}{x \mathsf{op} y} \right| = |\delta| \le \mathsf{u}$$

where u is known as the *unit roundoff*.

We can use this idea sequentially to bound the error in more complex computations. Suppose we want to compute

$$s_n = x_1 + x_2 + \cdots + x_n.$$

If we use the numerical algorithm

$$\begin{aligned} s_0 &= 0 \\ s_1 &= s_0 + x_1 \\ s_2 &= s_1 + x_2 \\ &\vdots \\ s_n &= s_{n-1} + x_n \end{aligned}$$

the corresponding computer algorithm is

$$\begin{aligned} \sigma_0 &= 0 \\ \sigma_1 &= fl(\sigma_0 + x_1) \\ \sigma_2 &= fl(\sigma_1 + x_2) \\ &\vdots \\ \sigma_n &= fl(\sigma_{n-1} + x_n) \end{aligned}$$

Expanding the expressions for the σ_i, we obtain

$$\sigma_1 = fl(\sigma_0 + x_1) = (\sigma_0 + x_1)(1 + \delta_1)$$
$$\sigma_2 = fl(\sigma_1 + x_2) = (\sigma_1 + x_2)(1 + \delta_2) = \sigma_1(1 + \delta_2) + x_2(1 + \delta_2) = (\sigma_1 + x_1)(1 + \delta_1)(1 + \delta_2) + x_2(1 + \delta_2)$$
$$\vdots$$
$$\sigma_n = x_1(1 + \delta_1)\cdots(1 + \delta_n) + x_2(1 + \delta_2)\cdots(1 + \delta_n) + \cdots + x_n(1 + \delta_n) = \sum_{j=1}^{n} x_j \prod_{k=j}^{n}(1 + \delta_k) = \sum_{j=1}^{n} x_j(1 + \eta_j)$$

where

$$(1 + \eta_j) = \prod_{k=j}^{n}(1 + \delta_k) = x_1 + \cdots + x_n + \sum_{j=1}^{n} \eta_j x_j.$$

In summary,

$$\left| \sigma_n - \sum_{i=1}^{n} x_i \right| \le \sum_{j=1}^{n} |\eta_j||x_j|, \quad |\eta_j| \le (n - j + 1)\mathsf{u} + O(\mathsf{u})^2$$

if $0 < x_{i_1} \le x_{i_2} \le \cdots \le x_{i_n}$. It would seem to imply that we should add the numbers in that order since the numbers are weighted. A better procedure is to add the numbers in pairs, resulting in the bound

$$|1 + \eta_i| \le 1 + p\mathsf{u}$$

where $n = 2^p$. In this case, the error is uniformly distributed.

For the product of n numbers $p = x_1 \cdots x_n$, we obtain the computed results

$$\begin{aligned} \Pi_1 &= fl(x_1 x_2) = (x_1 x_2)(1 + \epsilon_1) \\ \Pi_2 &= fl(\Pi_2 x_3) = (x_1 x_2 x_3)(1 + \epsilon_1)(1 + \epsilon_2) \\ &\vdots \\ \Pi_n &= fl(\Pi_{n-1} x_n) = (x_1 \cdots x_n)(1 + \epsilon_1)\cdots(1 + \epsilon_{n-1}) = p(1 + \eta_{n-1}) \end{aligned}$$

where

$$|\eta_{n-1}| \le (n - 1)\mathsf{u} + O(\mathsf{u}^2).$$

2.3. **Common Computations.** It is frequently necessary to compute the *inner product*

$$s = \sum_{i=1}^{n} u_i v_i$$

Now

$$fl(u_i \times v_i) = (u_i \times v_i)(1 + \gamma_i), \quad |\gamma_i| \leq \mathsf{u}.$$

Therefore, if we wish to compute

$$fl(s) = \sum_{j=1}^{n} x_j y_j (1 + \eta_j)$$

and $(1 + \eta_j)$ will depend on how the computation is performed. If we do serial addition, then

$$(1 + \eta_j) = \prod_{k=j}^{n} (1 + \delta_k)(1 + \gamma_j).$$

If we do pairwise addition and $n = 2^p$, then

$$(1 + \eta_j) = \left(\sum_{k=1}^{p} (1 + \delta_{i_{k_j}}) \right) (1 + \gamma_j).$$

In computing inner products one must be quite careful; otherwise the underflow or overflow problem can be quite serious.

Suppose one wishes to compute

$$s = \sum_{i=1}^{n} x_i^2.$$

It is quite possible that overflow will occur. One solution is to scale. Suppose

$$|x_{\pm}| \geq |x_j|, \quad j = 1, 2, \ldots, n$$

and $x_{\pm} = \alpha \beta^p$. Then one should compute

$$s = \left[\sum_{i=1}^{n} \left(\frac{x_i}{\beta_p} \right)^2 \right] \beta^{2p}.$$

This requires two passes; there are better ways of performing the calculation.

We can build up analysis of many problems. This has been done extensively for linear algebra by J. H. Wilkinson. The analysis can be automated; R. Moore has used interval arithmetic for determining bounds on the solution.

It is a general principle that one should add terms of a similar sign. Suppose we wish to compute

$$s^2 = \sum_{i=1}^{n} (x_i - \bar{x})^2, \quad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{2.1}$$

It is well known that

$$s^2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2 \tag{2.2}$$

and this formula is frequently used. But it is very bad, especially if the x_i's are large but s^2 is small. Then, you can be subtracting two large numbers. Formula (2.1) is more stable but requires two passes. First, you must compute the mean and then the quantity s^2. We could try writing

$$s^2 = \sum_{i=1}^{n} (x_i - \nu)^2 + \eta(\nu^2 - \bar{x}^2)$$

where ν is some approximation to $\bar{x}$.

3. IEEE floating point numbers

These notes are due to Lieven Vandenberghe of UCLA.

3.1. Floating point numbers with base 10.

- **Notation.**

$$x = \pm(.d_1 d_2 \cdots d_n)_{10} \cdot 10^e$$

  - $.d_1 d_2 \cdots d_n$ is the *mantissa* (d_i integer, $0 \le d_i \le 9$, $d_1 \neq 0$ if $x \neq 0$)
  - n is the *mantissa length* (or *precision*)
  - e is the *exponent* ($e_{\min} \le e \le e_{\max}$)
- **Interpretation.**

$$x = \pm(d_1 10^{-1} + d_2 10^{-2} + \cdots + d_n 10^{-n}) \cdot 10^e$$

- **Example.** (with $n = 7$)

$$\begin{aligned} 12.625 &= +(.1262500)_{10} \cdot 10^2 \\ &= +(1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 6 \cdot 10^{-3} + 2 \cdot 10^{-4} + 5 \cdot 10^{-5} + 0 \cdot 10^{-6} + 0 \cdot 10^{-7}) \cdot 10^2 \end{aligned}$$

  used in pocket calculators
- **Properties.**
  - a finite set of numbers
  - unequally spaced distance between floating point numbers varies
    - the smallest number greater than 1 is $1 + 10^{-n+1}$
    - the smallest number greater than 10 is $10 + 10^{-n+2}, \ldots$
  - largest positive number:

$$+(.999 \cdots 9)_{10} \cdot 10^{e_{\max}} = (1 - 10^{-n}) 10^{e_{\max}}$$

  - smallest positive number:

$$x_{\min} = +(.100 \cdots 0)_{10} \cdot 10^{e_{\min}} = 10^{e_{\min} - 1}$$

3.2. Floating point numbers with base 2.

- **Notation.**

$$x = \pm(.d_1 d_2 \cdots d_n)_2 \cdot 2^e$$

  - $.d_1 d_2 \cdots d_n$ is the *mantissa* ($d_i \in \{0, 1\}$, $d_1 = 1$ if $x \neq 0$)
  - n is the *mantissa length* (or *precision*)
  - e is the *exponent* ($e_{\min} \le e \le e_{\max}$)
- **Interpretation.**

$$x = \pm(d_1 2^{-1} + d_2 2^{-2} + \cdots + d_n 2^{-n}) \cdot 2^e$$

- **Example.** (with $n = 8$):

$$\begin{aligned} 12.625 &= +(.11001010)_2 \cdot 2^4 \\ &= +(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 0 \cdot 2^{-8}) \cdot 2^4 \end{aligned}$$

  used in almost all computers
- **Properties.**
  - a finite set of unequally spaced numbers
  - largest positive number:

$$x_{\max} = +(.1 \cdots 1)_2 \cdot 2^{e_{\max}} = (1 - 2^{-n}) 2^{e_{\max}}$$

  - smallest positive number:

$$x_{\min} = +(.100 \cdots 0)_2 \cdot 2^{e_{\min}} = 2^{e_{\min} - 1}$$

- in practice, the number system includes *subnormal numbers*: unnormalized small numbers ($d_1 = 0, e = e_{\min}$), and the number 0

3.3. **IEEE floating point standard.** Specifies two binary floating point number formats:

- **IEEE standard single precision:**

$$n = 24, \quad e_{\min} = -125, \quad e_{\max} = 128$$

  requires 32 bits: 1 sign bit, 23 bits for mantissa, 8 bits for exponent.
- **IEEE standard double precision:**

$$n = 53, \quad e_{\min} = -1021, \quad e_{\max} = 1024$$

  requires 64 bits: 1 sign bit, 52 bits for mantissa, 11 bits for exponent; used in almost all modern computers

3.4. **Machine precision.**

- **Definition.** The *machine precision* of a binary floating point number system with mantissa length n is defined as

$$\epsilon_M = 2^{-n}$$

- **Example.** IEEE standard double precision ($n = 53$):

$$\epsilon_M = 2^{-53} \approx 1.1102 \cdot 10^{-16}$$

- **Interpretation.** $1 + 2\epsilon_M$ is the smallest floating point number greater than 1:

$$(.10\cdots01)_2 \cdot 2^1 = 1 + 2^{1-n} = 1 + 2\epsilon_M.$$

3.5. **Rounding error.** A floating-point number system is a finite set of numbers; all other numbers must be rounded

- **Notation.** $fl(x)$ is the floating-point representation of x
- **Rounding rules** used in practice:
  - numbers are rounded to the nearest floating-point number
  - in case of a tie: round to the number with least significant bit 0 ("round to nearest even")
- **Example.** numbers $x \in (1, 1 + 2\epsilon_M)$ are rounded to 1 or $1 + 2\epsilon_M$:
  - $fl(x) = 1$ if $1 < x \leq 1 + \epsilon_M$
  - $fl(x) = 1 + 2\epsilon_M$ if $1 + \epsilon_M < x < 1 + 2\epsilon_M$
- gives another interpretation of ϵ_M: numbers between 1 and $1 + \epsilon_M$ are indistinguishable from 1

3.6. **Rounding error and machine precision.** General result (no proof):

$$\frac{|fl(x) - x|}{|x|} \leq \epsilon_M.$$

- machine precision gives a bound on the relative error due to rounding
- number of correct (decimal) digits in $fl(x)$ is roughly

$$-\log_{10} \epsilon_M$$

  i.e., about 15 or 16 in IEEE precision
- fundamental limit on accuracy of numerical computation

Department of Computer Science, Gates Building 2B, Room 280, Stanford, CA 94305-9025
E-mail address: golub@stanford.edu