MAE 5032 High Performance Computing: Methods and Practices

Lecture 11: Numerical Analysis Basics

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

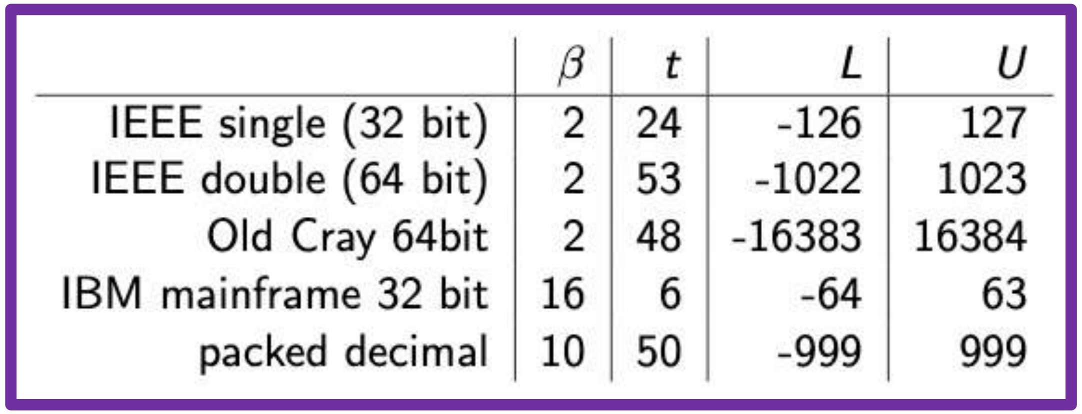
Floating-point arithmetic

Floating-point numbers

- Computers use a finite number of bits to represent numbers. Only a finite number of numbers can be represented.

$$x = \pm\left(\sum_{i=0}^{t-1} d_i \beta^{-i}\right)\beta^{e} = (1\times2^{-0} + 1\times2^{-1} + 0\times2^{-2} + \cdots 1\times2^{-23})\times2^{1}$$
$$\approx 1.5707964\times2 = 3.1415928$$

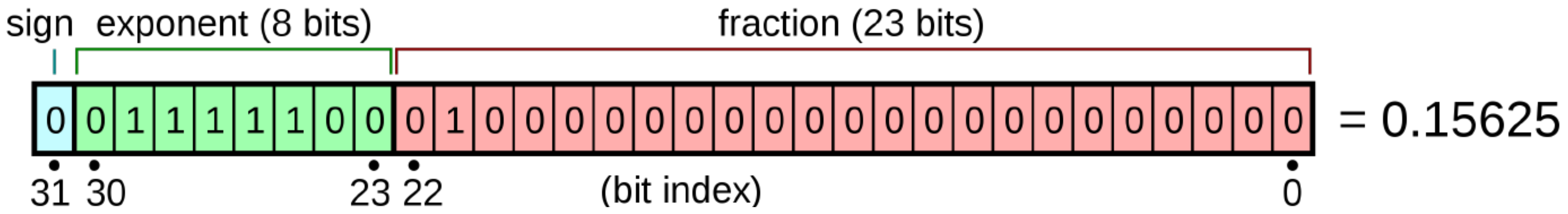
- One bit for sign (unsigned number exisits)
- β is the base of the number system (2,10,16, etc.)
- t is the significand precision
- $0 \le d_i \le \beta - 1$ is the digits of the significand
- $L \le e \le U$ is the signed exponent



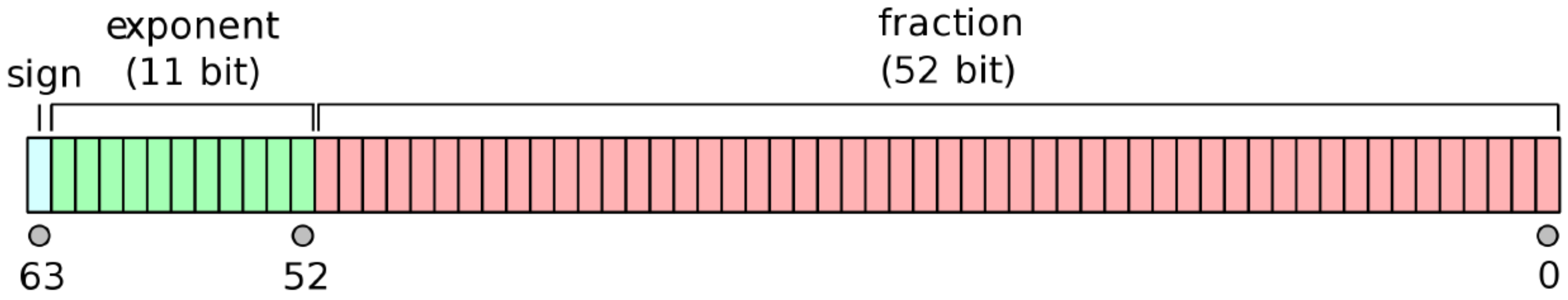
	β	t	L	U
IEEE single (32 bit)	2	24	-126	127
IEEE double (64 bit)	2	53	-1022	1023
Old Cray 64bit	2	48	-16383	16384
IBM mainframe 32 bit	16	6	-64	63
packed decimal	10	50	-999	999

- Underflow level β^{L}
- Overflow level $(1-\beta^{-t})\beta^{U+1}$

Floating-point numbers



IEEE single 32-bit



IEEE double 64-bit

Floating-point numbers

- **Normalized** number require that the **exp is not all zeros or all ones**.

- In binary system, the leading number of the significand is 1, thus we get one free bit for mantissa.

$$x = \pm \sum_{i=0}^{t} d_i \beta^{-i} \beta^{e} = (1\times 2^{-0} + 1\times 2^{-1} + 0\times 2^{-2} + \cdots 1\times 2^{-23})\times 2^{1}$$

$$\approx 1.5707964\times 2 = 3.1415928$$

- Exponent is coded as a biased value e = exp – bias.
  - exp: unsigned value of the exp field
  - bias: $2^{k-1} - 1 = U$ (single precision 127, double precision 1023).
  - e: single precision (-126,127); double precision (-1022, 1023).

Floating-point numbers

- **Normalized** number

float f = 15213.0 =$1.1101101101101 \times 2^{13}$

significand: 1.1101101101101

frac = 11011011011010000000000

exponent: e = 13 = 140 – 127

bias = 127

exp = 140 = 1001100

Result:

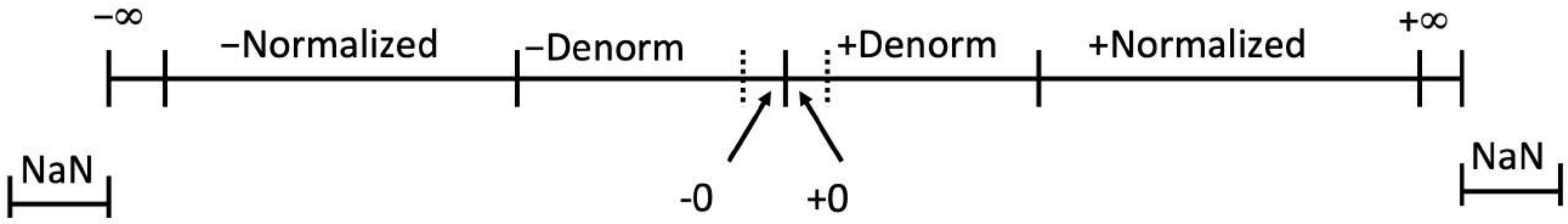
0	1001100	11011011011010000000000

Floating-point numbers

- **Denormalized** number
- condition: exp = 0…00
- exponent is e = 1– bias instead of 0 – bias
- significand codes with implied leading 0: M=0.x…xx
- cases:
  - exp = 0…00, frac = 0…00 represents zero value (there are +0.0 and -0.0)
  - exp = 0…00, frac $\neq 0..00$ represents values close to 0.0, equispaced.

Floating-point numbers

- **Special values**
- condition: exp = 1…11
- case: frac $= 0..00$ represents $\pm\infty$
- case: frac $\neq 0..00$ represents NaN, when no numeric value can be determined (e.g. sqrt(-1)).



Floating-point numbers

- 8-bit floating point representation
  - the sign bit is the most significant bit
  - the next four bits are the exponent, with a bias of 7
  - the last three bits are the frac

s	exp 4-bits	frac 3-bits

0	0000	000	⇨	e = -6 value = 0
0	0000	001	⇨	e = -6 value = 1/8*1/64 = 1/512
0	0000	010	⇨	e = -6 value = 2/8*1/64 = 2/512
0	0000	111	⇨	e = -6 value = 7/8*1/64 = 7/512
0	0001	000	⇨	e = -6 value = 1*1/64 = 8/512
0	0001	001	⇨	e = -6 value = 9/8*1/64 = 9/512

Floating-point numbers

- 8-bit floating point representation
  - the sign bit is the most significant bit
  - the next four bits are the exponent, with a bias of 7
  - the last three bits are the frac

s	exp 4-bits	frac 3-bits

0	0110	110	⇨	E = -1 value = 14/8 * 1/2 = 14/16	
0	0110	111	⇨	E = -1 value = 15/8*1/2 = 15/16	closest to 1 below
0	0111	000	⇨	E = 0 value = 8/8 * 1 = 1	
0	0111	001	⇨	E = 0 value = 9/8*1 = 9/8	closest to 1 above
0	1110	111	⇨	E = 7 value = 15/8*128 = 240	
0	1111	xxx	⇨	nan inf	

Rounding

	1.4	1.6	1.5	2.5	-1.5
Towards zero	1	1	1	2	-1
Round down	1	1	1	2	-2
Round up	2	2	2	3	-1
Nearest even	1	2	2	2	-2

Representation error

- Error between a number x and its floating-point representation $\tilde{x}$:
  - absolute $|x - \tilde{x}|$
  - relative $\frac{|x-\tilde{x}|}{|x|}$

$$\tilde{x} = \pm\left(\sum_{i=0}^{t-1} d_i \beta^{-i}\right)\beta^{e}$$
$$x = \pm\left(\sum_{i=0}^{\infty} d_i \beta^{-i}\right)\beta^{e}$$

- Equivalently, sometimes we say $\tilde{x} = x(1 \pm \varepsilon)$

- IEEE 754 standard gives different rounding rules, resulting in $\varepsilon \leq \beta^{-t}$

- The value of β^{-t} is called the machine precision. (Calculate it for a 64-bit double.)

Example: decimal numerical system (i.e. β=10), $t = 3$,
$x = 0.1256$, then $\tilde{x}_r = 0.126$ or $\tilde{x}_t = 0.125$.

Addition

- Steps for addition of floating-point numbers
  - align exponents
  - add significand
  - adjust exponent to normalize the result

Example: $123456.7 + 101.7654$ $= (1.234567\times10^5) + (1.017654\times10^2)$

$$= (1.234567\times10^5) + (0.001018\times10^5)$$

$$= 1.235585\times10^5$$

- We consider x_i and their floating-point number $\tilde{x}_i = x_i(1+\varepsilon_i)$

  To compute $s = x_1 + x_2$, the sum is represented as

$$\tilde{s} = (\tilde{x}_1 + \tilde{x}_2)\,(1+\varepsilon_3) = x_1(1+\varepsilon_1)\,(1+\varepsilon_3) + x_2(1+\varepsilon_2)\,(1+\varepsilon_3)$$

$$\approx x_1(1+\varepsilon_1+\varepsilon_3) + x_2(1+\varepsilon_2+\varepsilon_3) \approx s(1+2\varepsilon)$$

- Conclusion: Errors are added

Subtraction and associativity

Example: $123457.1467 - 123456.659 \approx (1.234571\times10^{5}) - (1.234567\times10^{5})$
$= 4.000000\times10^{-1}$

- The actual result is 4.877000×10^{-1}
- Relative error of the subtraction is about 20%.
- In extreme cases, all significant numbers can be lost due to cancellation.

Example: 7-digits decimal floating-point number to calculate (a+b)+c and a+(b+c), with a=1234.567, b=45.67834, c=0.0004.

a+b = 1280.24534, rounds to 1280.245; (a+b)+c = 1280.2454 rounds to 1280.245

(b+c) = 45.67874 rounds to 45.67874; a+(b+c) = 1280.24574 rounds to 1280.246

Because of this, compilers will not automatically reorder the operations for FP.

You need to enable “fast-math” options.

A toy problem

Evaluate $\sum_{n=1}^{10000} \frac{1}{n^2}$. The precise value is 1.644834 in a decimal numerical system (i.e. β=10) with $t = 7$.

First term is 1.000000, so the partial sum will be greater than 1. So, for the terms that $\frac{1}{n^2} < 10^{-6}$, their contribution to the sum will be ignored.

Floating point sum is 1.644725: 4 correct digits.

Solution: sum in reverse order.

Unstable algorithm

Consider the recurrence

$y_n = \int_0^1 \frac{x^n}{x+5} dx$. We may deduce that $y_n = \frac{1}{n} - 5y_{n-1}$.

The initial value of the recurrence is $y_0 = \log(6) - \log(5) = 1.82|322 \times 10^{-1}$.

Consider a decimal numerical system (i.e. β=10) with $t = 3$.

FP arithmetic	Correct value
$\tilde{y}_0 = 1.82 \times 10^{-1}$	$y_0 = 1.82 \times 10^{-1}$
$\tilde{y}_1 = 9.00 \times 10^{-2}$	$y_1 = 8.84 \times 10^{-2}$
$\tilde{y}_2 = 5.00 \times 10^{-2}$	$y_2 = 5.80 \times 10^{-2}$
$\tilde{y}_3 = 8.30 \times 10^{-2}$	$y_3 = 4.31 \times 10^{-2}$
$\tilde{y}_4 = -1.65 \times 10^{-1}$	$y_4 = 3.43 \times 10^{-2}$

Let $\tilde{y}_n = y_n + \varepsilon_n$. Then $\tilde{y}_n = \frac{1}{n} - 5\,\tilde{y}_{n-1} = \frac{1}{n} - 5y_{n-1} + 5\varepsilon_{n-1} = y_n + 5\varepsilon_{n-1}$.

$\varepsilon_n = 5\varepsilon_{n-1}$. Error grows exponentially.

Summary

- Arithmetic on computer is done based on floating-point numbers, and thus simple calculations like addition may lead to error.
- Mathematically equivalent operations may not remain equivalent on computers, which could affect parallel computations.
- Algorithms need to be carefully designed to control the floating-point error
- Stability analysis is needed for ALL operations on computers.

Reference: Introudction to High Performance Scientific Computing by Victor Eijkhout, Chapter 2.

Linear Algebra

A known unusable algorithm

To solve Ax = b, one may think of the Cramer's rule:

$$x_i = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1i-1} & b_1 & a_{1i+1} & \dots & a_{1n} \\ a_{21} & & \dots & & b_2 & & \dots & a_{2n} \\ \vdots & & & & \vdots & & & \vdots \\ a_{n1} & & \dots & & b_n & & \dots & a_{nn} \end{vmatrix} / |A|$$

The time complexity is O(n!)

Recall that on a single CPU of TaiYi, we can achieve around 76.8 Gflops.

This algorithm is too expensive! Solving a 20-by-20 matrix problem may take 1 year!

Direct method

To solve Ax = b, one may do the old Gaussian elimination method:

$$\begin{pmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 3 \end{pmatrix} x = \begin{pmatrix} 16 \\ 26 \\ -19 \end{pmatrix}$$

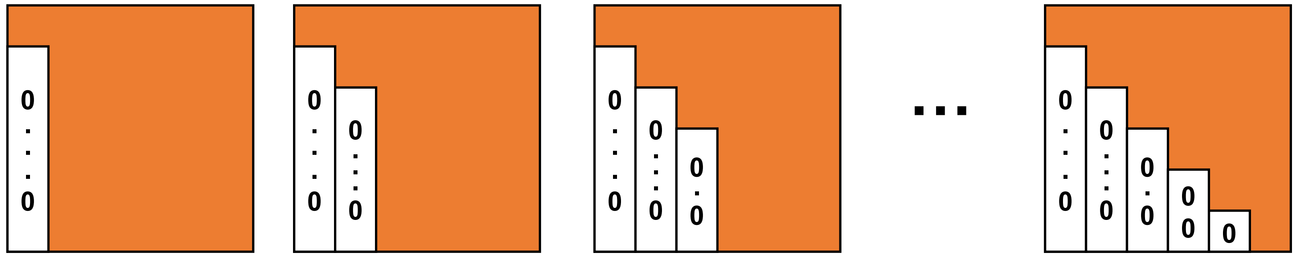
$$\left[\begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 12 & -8 & 6 & 26 \\ 3 & -13 & 3 & -19 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 0 & -4 & 2 & -6 \\ 0 & -12 & 2 & -27 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 0 & -4 & 2 & -6 \\ 0 & 0 & -4 & -9 \end{array}\right]$$

Algorithmic complexity is O(n^3).

Gaussian Elimination algorithm

- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

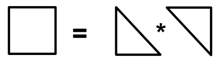
```
for each column i
zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    for each row j below row i
    for j = i+1 to n
        add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```



After i=1 After i=2 After i=3 After i=n-1

Gaussian Elimination algorithm: math kernel

- LU factorization: if the above algorithm completes, we get A = LU.



- Thus Ax=b becomes LUx=b.
  - ➢ We solve Ly=b first and solve Ux=y next.

- Solving Ax=b using GE:
  - ➢ Factorize A = L*U using GE (cost = 2/3 n^3 flops)
  - ➢ Solve L*y = b for y, using substitution (cost = n^2 flops)
  - ➢ Solve U*x = y for x, using substitution (cost = n^2 flops)

Roundoff control

Consider a system

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ 2 \end{bmatrix}.$$

The exact solution is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Perform Gaussian elimination

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & 1-\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ 1-\frac{1}{\varepsilon} \end{bmatrix}.$$

We can do a “back-substitution” by solving x_2 first and solve x_1 next:

$$x_2 = 1 \Longrightarrow x_1 = 1\ .$$

Roundoff control (cont.)

Suppose ε is smaller than the machine precision. $1 - \frac{1}{\varepsilon}$ becomes $-\frac{1}{\varepsilon}$, and $1 + \varepsilon$ becomes 1.

The Gaussian elimination

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 + \varepsilon \\ 1 - \frac{1}{\varepsilon} \end{bmatrix}$$

becomes

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & -\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{\varepsilon} \end{bmatrix}$$

We can do a “back-substitution” by solving x_2 first and solve x_1 next:

$$x_2 = 1 \Longrightarrow x_1 = 0 \ .$$

Machine round-off error has a dramatic impact on the results. We call this numerical instability.

Pivoting in Gaussian Elimination

- During the LU factorization, when the diagonal is small, swap the row (or column) to avoid division by small numbers.
  - ➢ This is referred to as pivoting in GE
  - ➢ We choose the largest possible pivot.
  - ➢ In fact, we always choose the largest possible value as the pivot in GE.
  - ➢ There is always a nonzero pivot if the matrix is non-singular.

Roundoff control with pivoting

Consider the system again,

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ 2 \end{bmatrix}.$$

Pivot the row

$$\begin{bmatrix} 1 & 1 \\ \varepsilon & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1+\varepsilon \end{bmatrix}.$$

Perform Gaussian elimination

$$\begin{bmatrix} 1 & 1 \\ 0 & 1-\varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1-\varepsilon \end{bmatrix}.$$

If ε is small,

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Back substitution: x_2 = 1 $\Longrightarrow x_1 = 1$.

1990 Nobel Prize in Economics

Our models strained the computer capabilities of the day [1950s]. I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were sparse in the matrix.

-- Harry Markowitz

Sparse matrices arise in many applications:

- ➢ simulating physics
- ➢ analyzing images
- ➢ web page ranking in search engines
- ➢ ……

Examples

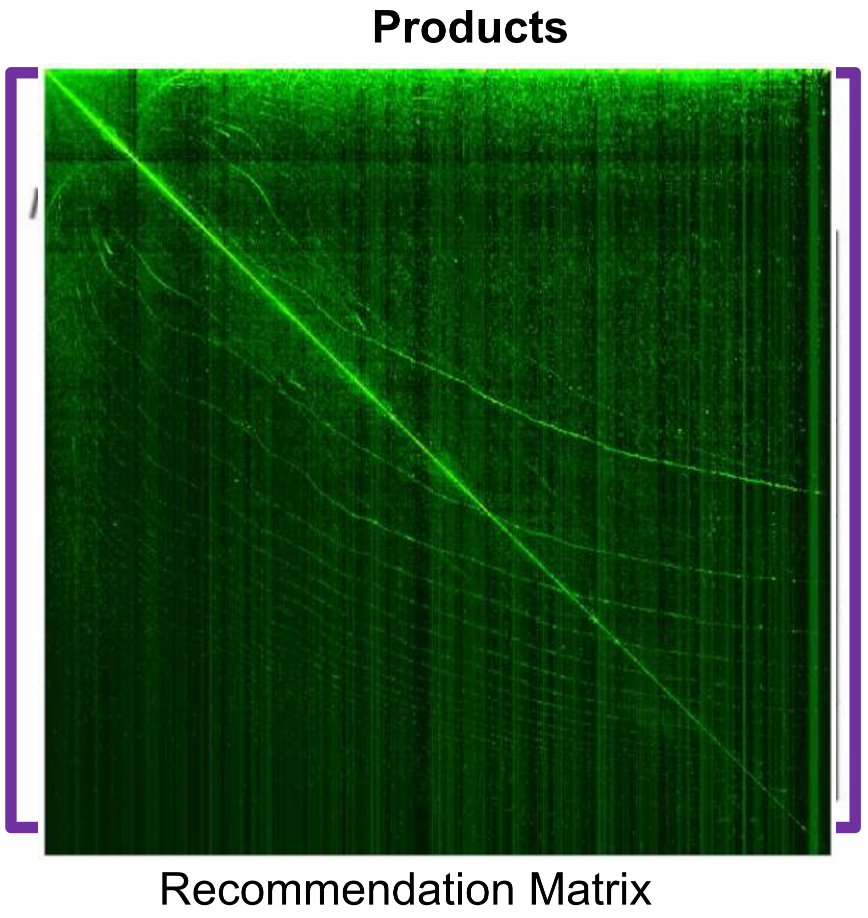
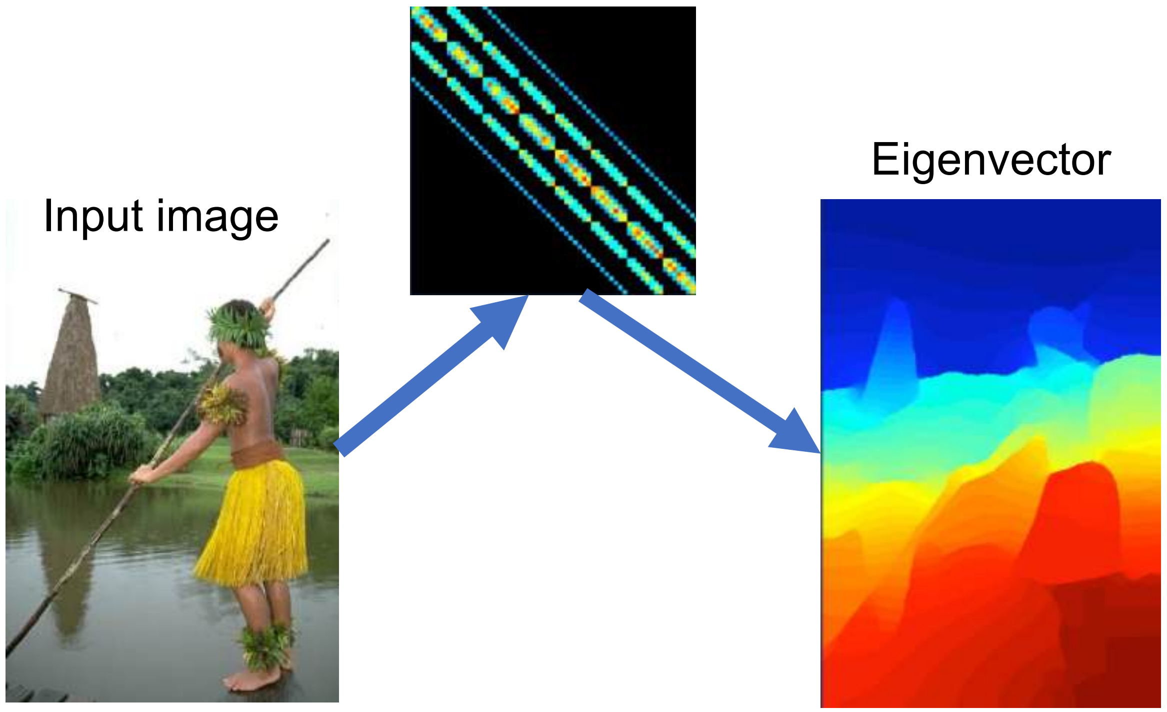


Image segmentation – identify the object boundaries in an image.

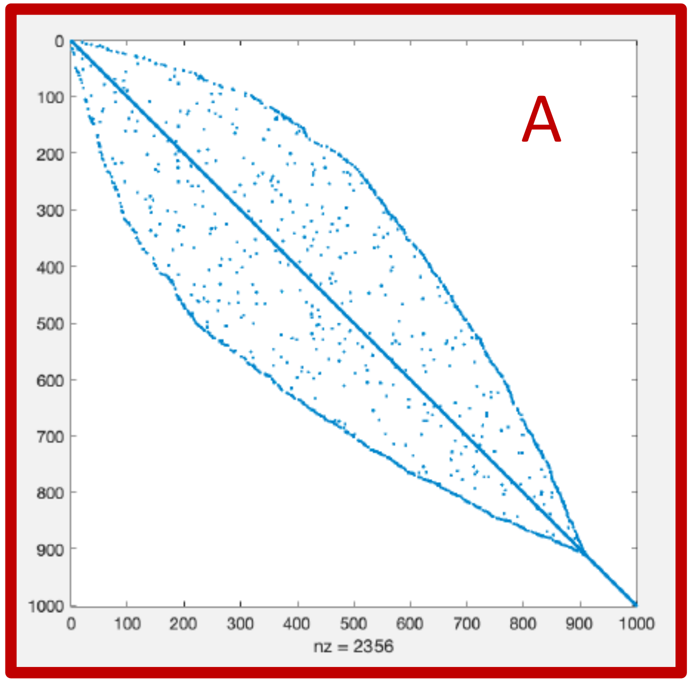
Find the eigenvalue of the affinity matrix.

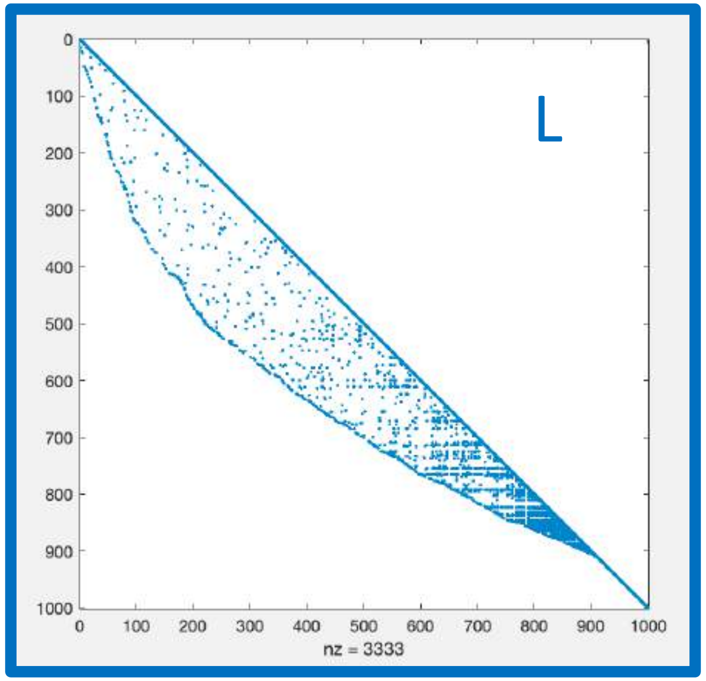
Efficient, High-quality image contour detection by Catanzaro, Su, et al. 2009.

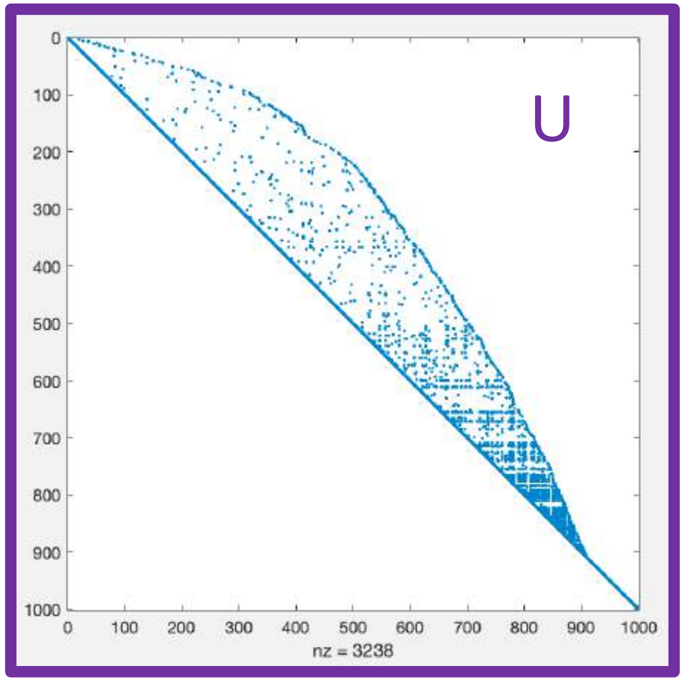


The fill-in phenomenon

- LU factorization of a sparse matrix typically leads to additional nonzeros. This phenomenon is called **fill-in**.
- The memory requirement can thus become a bottleneck if one wants to use direct method to solve a sparse matrix.



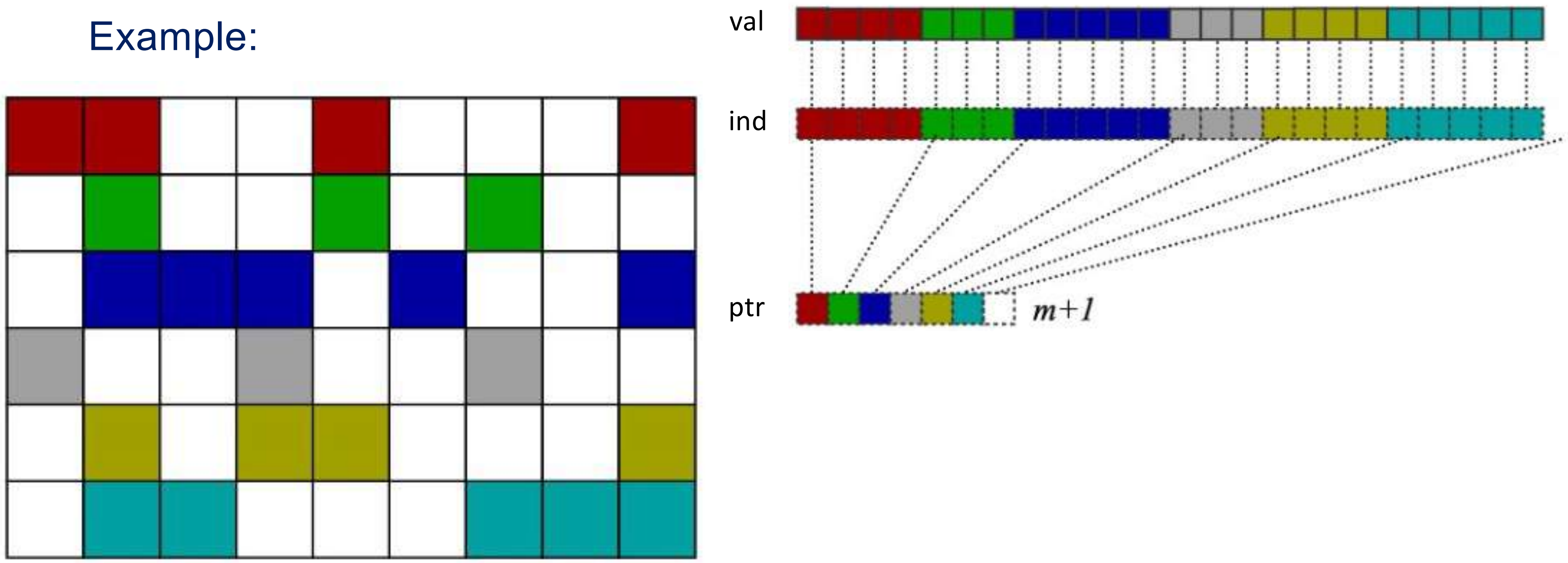




Data structure

- Use a proper data structure to represent the matrix by saving the nonzeros only.
- Compressed Row Storage (CRS) is the most widely used format.

Example:



Data structure

- Use a proper data structure to represent the matrix by saving the nonzeros only.
- Compressed Row Storage (CRS) is the most widely used format.

Example:

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

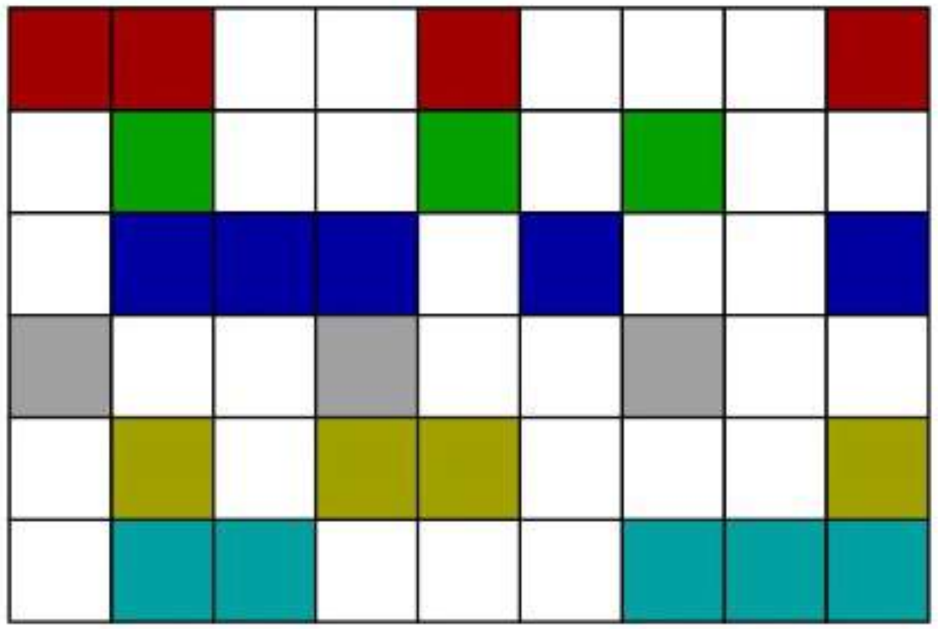
val	10	−2	3	9	3	7	8	7	3 ⋯ 9	13	4	2	−1
ind	0	4	0	1	5	1	2	3	0 ⋯ 4	5	1	4	5
ptr	0	2	5	8	12	16	19						

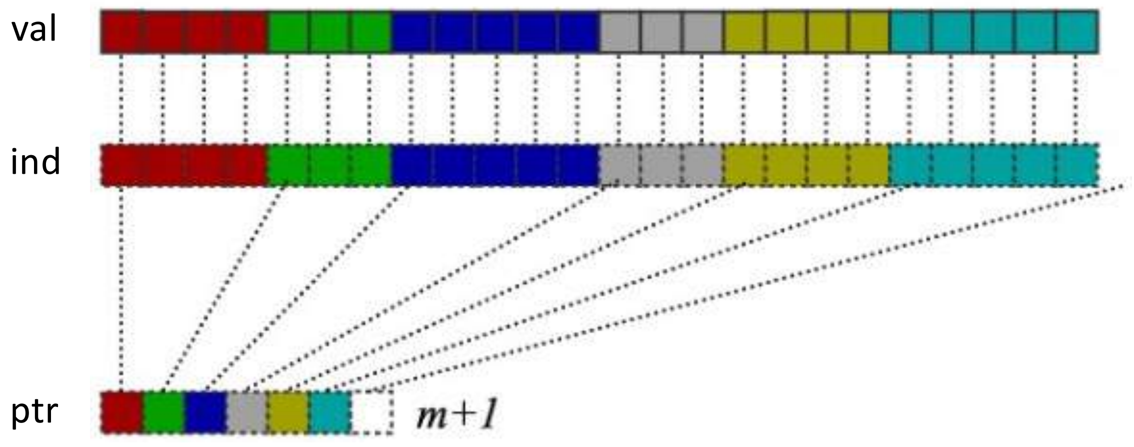
<u>Preallocation</u> of the memory space can be critical! Otherwise, the initial use of the matrix can be extremely slow.

Data structure

- Use a proper data structure to represent the matrix by saving the nonzeros only.
- Compressed Row Storage (CRS) is the most widely used format.

Example:





```
y(i) ← y(i) + A(i,j) x(j)
for each row i
    for k = ptr[i] to ptr[i+1]
            y[i] += val[k] * x[ind[k]]
```

Stationary iterative method

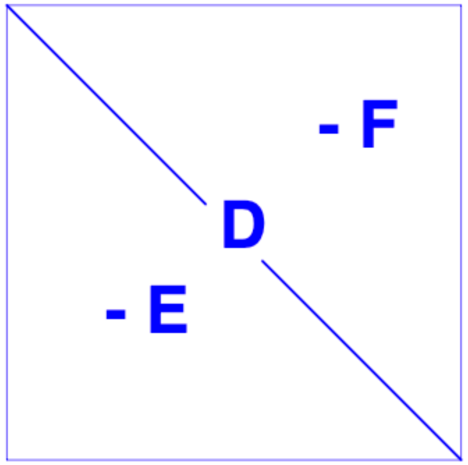
- To solve $Ax = b$ with the matrix A being a sparse matrix, we want to devise an iterative method

$$Mx_{k+1} = Nx_k + f.$$

such that the solution of $Ax = b$ also satisfies $\mathrm{M}x = Nx + f$.

- A lot iterations can be devised based on the decomposition $A = D - E - F$.
  - **Jacobi** iteration: $\mathrm{D}x_{k+1} = (E + F)x_k + b$.
  - **Gauss-Seidel** iteration: $(\mathrm{D} - \mathrm{E})x_{k+1} = Fx_k + b$.
  - **Successive Over-Relaxation (SOR)** iteration:

$$(\mathrm{D} - \omega\mathrm{E})x_{k+1} = (\omega F + (1 - \omega)D)x_k + \omega b$$



Stationary iterative method

- If the iteration converges, it converges to the linear system solution.

- In one iteration, in general, we only need to perform (1) matrix-vector multiplication; (2) perhaps solving a simpler matrix problem (usually can be achieved component-wisely). The cost of one iteration is at most O(n^2).

- Stopping criterion:
  - monitor the residual $||b - Ax_k||$
  - monitor the relative change $||x_{k+1} - x_k||$

- It is often the case that we can achieve desirable accuracy in less than n iteration, meaning the iterative method is rather competitive, when compared against direct method.

- Is the method guaranteed to converge? How fast does it converge?

  There are theories to support this. Often is problem-specific.

Krylov iterative method

- Aleksey N. Krylov (1863-1945) is a Russian naval engineer and applied mathematician.

- He published around 300 papers, topics include shipbuilding, magnetism, artillery, mathematics, astronomy, etc.

- He built the first machine in Russia for integrating ODEs.

- In 1931, he published a paper on what is no called the Krylov subspace and Krylov subspace methods.

Krylov iterative method

- We are considering solving $Ax = b$, in which A is **large**, **sparse**, and possibly with **irregular** structures.

- The idea is to solve a least-square problem: $min_{z \in K} ||b - Az||$

- Krylov subspace: $K_m(A, v_1) = span\{v_1, Av_1, \dots, A^{m-1}v_1\}$

- There are very fast algorithms that can locate the vector that minimizes $||b - Az||$, which are referred to as the Krylov method
  - ➢ Use Arnoldi to orthogonalize the subspace
  - ➢ Use QR-decomposition to find the minimizer (NOT Least Square Method!!!)
  - ➢ Include GMRES, Congujate Gradient (CG), BICGSTAB, MINRES, etc.
  - ➢ All have been implemented in free, open-source libraries, do not do it yourself.

Krylov iterative method

- We can modify the linear system by considering solving

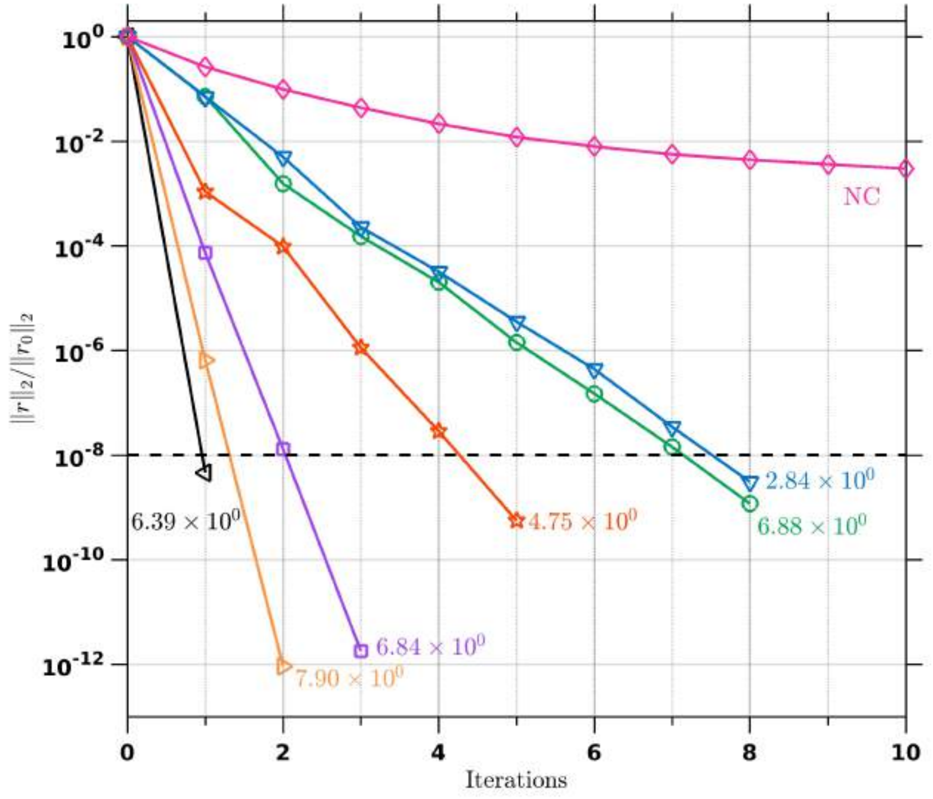
  $P_l A x = P_l b$,

  or $A P_r y = b$, with $x = P_r y$

  or $P_l\, A P_r x = P_l b$, with $x = P_r y$

- The matrices P_r and P_l are left and right **preconditioners**, which may **accelerate** the Krylov iteration.

Examples: Jacobi, ILU, etc.



Performance of different PCs

Summary

- Linear algebra problem is **ubiquitous**.

- Direct method such as LU factorization with pivoting can solve a problem in a very robust way.

  Its computational cost is O(n^3).

  For sparse matrices, the factorization will lead to a fill-in phenomenon, which could cost a lot memory space.

- Iterative method strives to solve the linear problem by matrix-vector multiplications iteratively.

  The cost per iteration is cheap O(n^2) at most and O(n) for very sparse matrices.

  Iteration drives the error down to certain prescribed tolerance.

  Preconditioner may accelerate the convergence.

  GMRES and CG are most commonly used method for solving linear systems.

Summary

Reference: Numerical Linear Algebra by L.N. Trefethen and D. Bau, III.

Differential equations

Initial value problem

- Consider the problem

$$\frac{du}{dt}(\mathrm{t}) = \mathrm{f}\big(\mathrm{t}, \mathrm{u}(\mathrm{t})\big),\ \mathrm{t} \in (0, T), \text{ with } u(0) = u_0.$$

- We may turn the continuous problem into a discrete one by looking at finite time steps: $t_0 = 0, t_1, t_2, \ldots\ t_N = T$. Here we assume the time step size is uniform: Δt.

- Taylor series

$$u(t + \Delta t) = u(t) + u'(t)\Delta t + u''(t)\frac{\Delta t^2}{2} + \cdots$$

- Use the following: $u'(t) = \frac{u(t+\Delta t) - u(t)}{\Delta t} + O(\Delta t)$, we get a discrete problem:

$$\frac{u_{k+1} - u_k}{\Delta t} = f(t_k, u_k)$$

- This is known as the Forward Euler or Explicit Euler method.
- First-order accurate.

Stability of the explicit Euler method

- Consider the problem

$$\frac{du}{dt}(\mathrm{t}) = \mathrm{f}\big(\mathrm{t}, \mathrm{u}(\mathrm{t})\big),\ \mathrm{t} \in (0, T),\ \text{with } u(0) = u_0.$$

Let $f = -\lambda u$. The exact solution is $u(t) = u_0 e^{-\lambda t}$. It is monotonically decreasing for positive λ.

$$u_{k+1} = u_k + \Delta t\, f(t_k, u_k) = (1 - \lambda\Delta t)\, u_k = \cdots = (1 - \lambda\Delta t)^{k+1} u_0$$

To have the discrete solution mimic its continuous counterpart,

$$|1 - \lambda\Delta t| < 1$$

which leads to $\Delta t < 2/\,\lambda$.

We call a method that has constraint on the selection of grid size to be conditionally stable.

Implicit Euler Method

- Consider the problem

$$\frac{du}{dt}(\mathrm{t}) = \mathrm{f}\big(\mathrm{t}, \mathrm{u}(\mathrm{t})\big),\ \ \mathrm{t} \in (0, T), \text{ with } u(0) = u_0.$$

- Taylor series

$$u(t - \Delta t) = u(t) - u'(t)\Delta t + u''(t)\frac{\Delta t^2}{2} + \cdots$$

- Use the following: $u'(t) = \frac{u(t) - u(t - \Delta t)}{\Delta t} + O(\Delta t)$

  We get a discrete problem:

$$\frac{u_{k+1} - u_k}{\Delta t} = f(t_{k+1}, u_{k+1})$$

- This is known as the Backward Euler or Implicit Euler method.

- First-order accurate.

Stability of the implicit Euler method

- Consider the problem

$$\frac{du}{dt}(\mathrm{t}) = \mathrm{f}\big(\mathrm{t}, \mathrm{u}(\mathrm{t})\big),\ \mathrm{t} \in (0, T), \text{ with } u(0) = u_0.$$

Let $f = -\lambda u$. The exact solution is $u(t) = u_0 e^{-\lambda t}$. It is monotonically decreasing for positive λ.

$$u_{k+1} = u_k + \Delta t\, f(t_{k+1}, u_{k+1}) \Rightarrow u_{k+1} = \frac{1}{1+\lambda\Delta t} u_k = \cdots = \left(\frac{1}{1+\lambda\Delta t}\right)^{k+1} u_0$$

To have the discrete solution mimic its continuous counterpart, there is no limit on the choice of time step size.
We call a method that has NO constraint on the selection of grid size to be unconditionally stable.

Explicit vs Implicit

- Consider the problem
  $\frac{du}{dt}(\mathrm{t}) = \mathrm{f}\big(\mathrm{t}, \mathrm{u}(\mathrm{t})\big),\ \mathrm{t} \in (0, T)$, with $u(0) = u_0$.

Let $f = u^2$.

Forward Euler: $u_{k+1} = u_k + \Delta t\, f(t_k, u_k) = u_k + u_k^2$.

Backward Euler: $u_{k+1} = u_k + \Delta t\, f(t_k, u_{k+1}) = u_k + u_{k+1}^2$

In implicit method, we need to solve equations to determine the solution.

There are other options: Trapezoidal/Crank-Nicholson rule, Runge-Kutta rule, etc.

Boundary value problem

- Consider the problem

  $$\frac{d^2u}{dx^2}(\mathrm{x}) = \mathrm{f}\left(\mathrm{x}, \mathrm{u}, \frac{du}{dx}\right), \; x \in (a, b), \text{ with } u(a) = u_a \text{ and } u(b) = u_b.$$

- Taylor series

  $$u(x + \Delta x) = u(x) + u'(x)\Delta x + u''(x)\frac{\Delta x^2}{2} + u'''(x)\frac{\Delta x^3}{6} + \cdots$$
  $$u(x - \Delta x) = u(x) - u'(x)\Delta x + u''(x)\frac{\Delta x^2}{2} - u'''(x)\frac{\Delta x^3}{6} + \cdots$$

- We can get: $u''(x) = \frac{u(x+\Delta x)-2u(x)+u(x-\Delta x)}{\Delta x^2} + O(\Delta x^2)$

  We get a discrete problem:

  $$\frac{u_{k+1}-2u_k+u_{k-1}}{\Delta x^2} = f(x_k, u_k, u_k')$$

Boundary value problem

- We get a discrete problem:

$$\frac{u_{k+1}-2u_k+u_{k-1}}{\Delta x^2} = f(x_k, u_k, u_k')$$

Written as a matrix problem:

$$\begin{bmatrix} 2 & -1 & \cdots & 0 \\ -1 & 2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \end{bmatrix} = -\Delta x^2 \begin{bmatrix} f_1 + u_a \\ f_2 \\ \vdots \\ \vdots \end{bmatrix}$$

Matrix properties: Very sparse (tri-diagonal), symmetric, positive definite.

Summary

- We can address the differential equations by replacing the differential operators by their difference counterpart.

- There are explicit and implicit method when marching in time.

- Typically, explicit method is conditionally stable, which poses a constraint on the choice of time step size.

- Implicit method can be either conditionally stable or unconditionally stable.

- Unconditionally stable algorithms is useful for delivering steady-state solutions.

- Accuracy analysis can be achieved by standard calculus techniques, such as the Taylor expansion.

- Accuracy can be verified by numerical results if you know the exact solution.

Summary

- Manufactured solution can be an effective way in designing code verifications.
- One may plot the error in a log-log plot against the mesh size, and the error shall be in a straight line with the slope being the accuracy.

Reference: 《微分方程数值解法》戴嘉尊，邱建贤。