Handout **July, 2007**

Contents

prepared by Zhaojun Bai, bai@cs.ucdavis.edu

1 Introduction

1. Scientific computing and computational science

   Scientific computing (numerical computing) is about the design and analysis of algorithms and engineering software for solving mathematical problems.

   Computational science involves innovative and essential use of high performance computation, and/or the development of computational technologies to advance knowledge or capabilities in scientific and engineering disciplines. A necessary element in computational science is a strong, close tie to an application discipline. Research in computation is inherently multidisciplinary and includes, for example, environmental modeling, simulation of complex physical systems that generate energy, semiconductor design, modeling DNA sequences and protein structure, and the simulation and analysis of flow through geologic structures. Ref: [DOE's computational science graduate fellowship program].

2. Algorithms as a technology, computational simulation as the third pillar of science

3. General strategy: *to replace a difficult problem with an easier one that has the same solution, or at least a closely related solution.*

   For example, solve $Ax = b$. If we can write $A = LU$, where L and U are lower and upper triangular matrices, respectively, then it is equivalent to solve $Ly = b$ for y and $Ux = y$ for x, can be easily computed by forward and back substitution. We will discuss this in more detail when we study the solution of linear system of equations.

4. Approximation and error are the facts of life in computational science.

   Sources of errors:

   modeling
   data uncertainty,
   truncation (discretization),
   rounding in finite precision arithmetic
   ...

   For example, let $f : \mathbf{R} \to \mathbf{R}$

   $$x \to f(x)$$

   We have an inexact input $\hat{x}$, and approximate function $\hat{f}$

   $$\begin{aligned} \text{total error} &= \hat{f}(\hat{x}) - f(x) \\ &= [\hat{f}(\hat{x}) - f(\hat{x})] + [f(\hat{x}) - f(x)] \\ &= \text{computational errors} + \text{propagated data errors} \\ &= \underline{\text{trunction/rounding}} + \underline{(\text{conditioning}) \times (\text{data error})} \end{aligned}$$

5. Absolute error and relative error

   Let $\hat{x}$ be an approximation of x. Then the *absolute error* is defined by

   $$\text{abserr}(x) = |\hat{x} - x|,$$

and the *relative error* (assume that x is a nonzero number) is defined by

$$\text{relerr}(x) = |\rho| := |\hat{x} - x|/|x|.$$

By the definition of the relative error, $\hat{x} = x(1 + \rho)$

Relative error is independent of scaling.

Rule of Thumb: if the relative error is approximately 10^{-d}, then x and $\hat{x}$ agree to about d significant digits, and conversely.

6. Forward error and backward error

   Suppose that an approximation $\hat{y}$ to $y = f(x)$ is computed. How should we measure the "quality" of $\hat{y}$?

   Ideally, we would like to have the relative *forward error* $\text{relerr}(y) = |y - \hat{y}|/|y| = \text{tiny}$.

   Instead, we can ask "for what set of data have we actually solved our problem?" That is, for what Δx, do we have $\hat{y} = f(x + \Delta x)$?

   $|\Delta x|$ (or $\min |\Delta|$ if there are many such Δx) is called *backward error*.

   Two main motivations for using backward error:

   - interprets errors as being equivalent to perturbations in the data,
   - reduces the question of bounding or estimating the forward error to perturbation theory, for which many problems is well understood (and only has to be developed once, for the given problem, and not for each method.)

7. An algorithm for computing $y = f(x)$ is called *(backward) stable* if, for any x, it produces a computed $\hat{y}$ with a small backward error, that is, $\hat{y} = f(x + \Delta x)$ for some small Δx.

8. Conditioning of problems: the relationship between forward and backward errors for a problem is governed by the conditioning of the problem, that is, the sensitivity of the solution to perturbation in the data.

   Example: compute $y = f(x)$. Let the computed results in terms of backward error $\hat{y} = f(x + \Delta x)$. Then the absolute error is

   $$\hat{y} - y = f(x + \Delta x) - f(x) = f'(x)\Delta x + O\left((\Delta x)^2\right).$$

   Correspondingly, the relative error is given by

   $$\frac{\hat{y} - y}{y} = \frac{x \cdot f'(x)}{f(x)} \left(\frac{\Delta x}{x}\right) + O((\Delta)^2).$$

   where

   $$\kappa_f(x) = \left|\frac{x \cdot f'(x)}{f(x)}\right|$$

   The quantity $\kappa_f(x)$ is called the *condition number of f at x*. It measures approximately how much the relative backward error in x is magnified by evaluating of f at x.

Rule of Thumb:

$$|\text{relative forward error}| \leq (\text{condition number}) \times |\text{relative backward error}|.$$

The computed solution to an ill-conditioned (i.e., large condition number) problem can have a large forward error, even for small backward error!

9. Some desirable qualities of numerical software

   - reliability
   - robustness
   - accuracy
   - efficiency (speed)
   - maintainability (structure modules, documentation, ... easy to modify)
   - portability
   - usability (easy of use)
   - applicability (functionality)
   - ...

   Tradeoffs among the different desirable qualities are common. Different priority for different users.

10. Three programming paradigms for exploiting other experts' software

    - Traditional software libraries and packages
    - Scientific computing environment: a much easier-to-use environment, but at the cost of some performance.
    - Templates for assembling complicated algorithms out of simpler building blocks.

    Netlib (`www.netlib.org`) is a valuable resource for free numerical software to solve all kinds of computational problems.

2 Floating-Point Arithmetic

1. Floating point representation (scientific notation) of numbers, for example,

$$\begin{array}{cc} - & 3.1416 \times 10^{1} \leftarrow \text{exponent} \\ \uparrow & \uparrow \quad \uparrow \\ \text{sign} & \text{significant} \ \ \text{base} \end{array}$$

   For representation on the computer, we prefer base 2 to base 10. The floating point representation of a nonzero binary number x is of the form

$$x = \pm\, b_0.b_1b_2\cdots b_{p-1} \times 2^E, \tag{1}$$

   (a) It is *normalized*, i.e., $b_0 = 1$ (*the hidden bit*)

   (b) *Precision* ($= p$) is the number of bits in the significant (including the hidden bit).

   (c) *Machine epsilon* $\epsilon = 2^{-(p-1)}$, the gap between the number 1 and the smallest floating point number that is greater than 1.

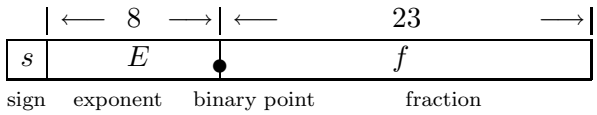
   (d) The *unit in the last place*, $\text{ulp}(x) = 2^{-(p-1)} \times 2^E = \epsilon \times 2^E$.
   If $x > 0$, then $\text{ulp}(x)$ is the gap between x and the next larger floating point number. If $x < 0$, then $\text{ulp}(x)$ is the gap between x and the smaller floating point number (larger in absolute value).

   Special numbers: 0, -0, ∞, $-\infty$, NaN = "Not a Number".

2. IEEE 754 floating point standard (IEEE 1985) essentials:

   - consistent representation of floating point numbers by all machines adopting the standard;
   - correctly rounded floating point operations, using various rounding modes;
   - consistent treatment of exceptional situation such as division by zero.

3. IEEE single format takes 32 bits long (= 4 bytes):



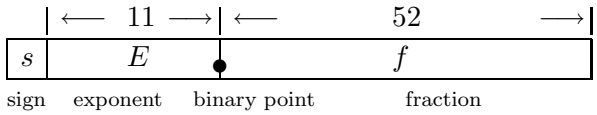
   It represents $(-1)^s \cdot (1.f) \times 2^{E-127}$ (note that the leading 1 in the fraction need not be stored explicitly, because it is always 1. This *hidden bit* accounts for the"1." here).

   Special repsentations for 0, $\pm\infty$ and NaN:

zero	=	±	00000000	00000000000000000000000
$\pm\infty$	=	±	11111111	00000000000000000000000
NaN	=	±	11111111	otherwise

   The range of positive normalized numbers is from $N_{\min} = 1.00\cdots0 \times 2^{-126} = 2^{-126} \approx 1.2 \times 10^{-38}$ to $N_{\max} = 1.11\cdots1 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}$.

4. IEEE double format takes 64 bits long (=8 bytes):



   It represents $(-1)^s \cdot (1.f) \times 2^{E-1023}$.

   Special repsentations for 0, $\pm\infty$ and NaN.

   The range of positive normalized numbers is from $N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308}$ to $N_{\max} = 1.11\cdots1 \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}$.

5. IEEE extended format, with at least 15 bits available for the exponent and at least 63 bits for the fractional part of the significant. (Pentium has 80-bit extended format)

6. Precision and machine epsilon of the IEEE formats

Format	Precision p	Machine epsilon $\epsilon = 2^{-p-1}$
single	24	$\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
double	53	$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
extended	64	$\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$

7. Rounding

   Let a positive real number x is in the normalized range, i.e., $N_{\min} \le x \le N_{\max}$, and write in the normalized form

$$x = (1.b_1b_2\cdots b_{p-1}b_pb_{p+1}\ldots) \times 2^E,$$

   Then the closest floating point number less than or equal to x is

$$x_- = 1.b_1b_2\cdots b_{p-1} \times 2^E,$$

   i.e., x_- is obtained by *truncating*. The next floating point number bigger than x_- is

$$x_+ = ((1.b_1b_2\cdots b_{p-1}) + (0.00\cdots01)) \times 2^E,$$

   therefore, also the next one that bigger than x.

   If x is negative, the situtation is reversed.

   Correctly rounding modes:

   - *round down*: $\text{round}(x) = x_-$;
   - *round up*: $\text{round}(x) = x_+$;
   - *round towards zero*: $\text{round}(x) = x_-$ of $x \ge 0$; $\text{round}(x) = x_+$ of $x \le 0$;
   - *round to nearest*: $\text{round}(x) = x_-$ or x_+, whichever is nearer to x, except that if $x > N_{\max}$, $\text{round}(x) = \infty$, and if $x < -N_{\max}$, $\text{round}(x) = -\infty$. In the case of tie, i.e., x_- and x_+ are the same distance from x, the one with its least significant bit equal to zero is chosen

When the *round to nearest* (IEEE default rounding mode) is in effect,

$$\text{abserr}(x) = |\text{round}(x) - x| \leq \frac{1}{2}\text{ulp}(x).$$

and

$$\text{relerr}(x) = \frac{|\text{round}(x) - x|}{|x|} \leq \frac{1}{2}\epsilon.$$

IEEE single: The maximum relative representation error is $\frac{1}{2} \cdot 2^{1-24} = 2^{-24} \approx 5.96 \cdot 10^{-8}$

IEEE double: The maximum relative representation error is $\frac{1}{2} \cdot 2^{-52} \approx 1.11 \times 10^{-16}$.

8. Correctly rounded floating point operations

   IEEE rules are as follows: if x and y are correctly rounded floating point numbers, then

$$\begin{aligned} \text{fl}(x+y) &= \text{round}(x+y) = (x+y)(1+\delta) \\ \text{fl}(x-y) &= \text{round}(x-y) = (x-y)(1+\delta) \\ \text{fl}(x \times y) &= \text{round}(x \times y) = (x \times y)(1+\delta) \\ \text{fl}(x/y) &= \text{round}(x/y) = (x/y)(1+\delta) \end{aligned}$$

   where for the *round to nearest*,

$$|\delta| \leq \frac{1}{2}\epsilon.$$

   IEEE standard also requires that correctly rounded remainder, and square root operations be provided.

9. IEEE standard response to exceptions

Event	Example	Set result to
Invalid operation	$0/0$, $0 \times \infty$	NaN
Division by zero	Finite nonzero/0	$\pm\infty$
Overflow	$\|x\| > N_{\max}$	$\pm\infty$ or $\pm N_{\max}$
underflow	$x \neq 0, \|x\| < N_{\min}$	± 0, $\pm N_{\min}$ or subnormal
Inexact	whenever $\text{fl}(x \circ y) \neq x \circ y$	correctly rounded value

10. Itanium Chip

    In 2000, Intel announced its new IA-64 Itanium chip. The IA-64 complies with the IEEE standard, and its floating point registers support the 80-bit extended format. In many aspects, the IA-64 departs radically from the Intel's previous chips. Most significantly, as far as floating point is concerned, is the fact that it has 128 floating point registers (compared with 8 on Pentium). Another significant change is that the IA-64 includes a *fused multiply-add instruction* (FMA). The FMA computes the correctly rounded value

$$\text{round}(a \times b + c).$$

11. Further Reading

    The following article based on lecture notes of Prof. W. Kahan of the University of California at Berkeley provides an excellent review of IEEE float point arithmetics.

D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 18(1):5–48, 1991.

The following recent published textbook gives a broad overview of numerical computing, with special focus on the IEEE standard for binary floating point arithmetic.

M. Overton. Numerical computing with IEEE floating point arithemetic. SIAM, Philadelphia, 2001. ISBN 0-89871-482-6. Student price $20.00 directly from `www.siam.org`.

To know more about IA-64, and particularly on floating point arithmetic aspects, see

P. Markstein. IA-64 and Elementary Functions. Prentice Hall PTR. 2000

M. Cornea, J. Harrison, and P. Tang, Scientific and Engineering Computation on Itanium Processors, Intel press, to appear.

3 Floating Point Arithmetic Error Analysis

1. Let $\hat{x}$ and $\hat{y}$ be the floating point numbers and that

$$\hat{x} = x(1+\tau_1) \quad \text{and} \quad \hat{y} = y(1+\tau_2), \quad \text{for } |\tau_i| \le \tau \ll 1,$$

where τ_i could be the relative errors in the process of "collecting/getting" the data from the original source or the previous operations.

Question: how do the four basic arithmetic operations behave?

(a) Addition and subtraction

$$\begin{aligned}
\text{fl}(\hat{x}+\hat{y}) &= (\hat{x}+\hat{y})(1+\delta), \qquad |\delta| \le \frac{1}{2}\epsilon \\
&= x(1+\tau_1)(1+\delta) + y(1+\tau_2)(1+\delta) \\
&= x + y + x(\tau_1 + \delta + O(\tau\epsilon)) + y(\tau_2 + \delta + O(\tau\epsilon)) \\
&= (x+y)\left(1 + \frac{x}{x+y}(\tau_1 + \delta + O(\tau\epsilon)) + \frac{y}{x+y}(\tau_2 + \delta + O(\tau\epsilon))\right) \\
&\equiv (x+y)(1+\hat{\delta}),
\end{aligned}$$

where $\hat{\delta}$ can be bounded as follows:

$$|\hat{\delta}| \le \frac{|x|+|y|}{|x+y|}\left(\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)\right).$$

Three possible cases:

i. If x and y have the same sign, i.e., $xy > 0$, then $|x+y| = |x| + |y|$; this implies

$$|\hat{\delta}| \le \tau + \frac{1}{2}\epsilon + O(\tau\epsilon) \ll 1.$$

Thus $\text{fl}(\hat{x}+\hat{y})$ approximates $x+y$ well.

ii. If $x \approx -y \Rightarrow |x+y| \approx 0$, then $(|x|+|y|)/|x+y| \gg 1$; this implies that $|\hat{\delta}|$ could be nearly or much bigger than 1. Thus $\text{fl}(\hat{x}+\hat{y})$ may turn out to have nothing to do with the true $x+y$. This is so called *catastrophic cancellation* which happens when a floating point number is subtracted from another nearly equal floating point number. Cancellation causes relative errors or uncertainties already presented in $\hat{x}$ and $\hat{y}$ to be magnified.

EXAMPLE. Computing $\sqrt{n+1} - \sqrt{n}$ straightforward causes substantial loss of significant digits for large n

n	$\text{fl}(\sqrt{n+1})$	$\text{fl}(\sqrt{n})$	$\text{fl}(\text{fl}(\sqrt{n+1}) - \text{fl}(\sqrt{n}))$
1.00e+10	1.00000000004999994e+05	1.00000000000000000e+05	4.99999441672116518e-06
1.00e+11	3.16227766018419061e+05	3.16227766016837908e+05	1.58115290105342865e-06
1.00e+12	1.00000000000050000e+06	1.00000000000000000e+06	5.00003807246685028e-07
1.00e+13	3.16227766016853740e+06	3.16227766016837955e+06	1.57859176397323608e-07
1.00e+14	1.00000000000000503e+07	1.00000000000000000e+07	5.02914190292358398e-08
1.00e+15	3.16227766016838104e+07	3.16227766016837917e+07	1.86264514923095703e-08
1.00e+16	1.00000000000000000e+08	1.00000000000000000e+08	0.00000000000000000e+00

Catastrophic cancellation can sometimes be avoided if a formula is properly reformulated. In the present case, one can compute $\sqrt{n+1} - \sqrt{n}$ almost to full precision by using the equality

$$\sqrt{n+1} - \sqrt{n} = \frac{1}{\sqrt{n+1} + \sqrt{n}}.$$

n	$\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n}))$
1.00e+10	4.999999999875000e-06
1.00e+11	1.581138830080237e-06
1.00e+12	4.999999999998749e-07
1.00e+13	1.581138830084150e-07
1.00e+14	4.999999999999987e-08
1.00e+15	1.581138830084189e-08
1.00e+16	5.000000000000000e-09

In fact, one can show that $\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n})) = (\sqrt{n+1}-\sqrt{n})(1+\delta)$, where $|\delta| \le 5\epsilon + O(\epsilon^2)$ (try it!)

EXAMPLE: Consider the function

$$f(x) = \frac{1-\cos x}{x^2} = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

Note that $0 \le f(x) < 1/2$ for all $x \neq 0$.
Compare the computed values for $x = 1.2 \times 10^{-5}$ using the above two expressions (assume that the value of $\cos x$ rounded to 10 significant figures).

iii. In general, if $(|x|+|y|)/|x+y|$ is not too big, $\mathrm{fl}(\hat{x}+\hat{y})$ provides a good approximation to $x+y$.

(b) Multiplication and Division are very well-behaved.

$$\begin{aligned}
\mathrm{fl}(\hat{x} * \hat{y}) &= (\hat{x} \times \hat{y})(1+\delta) = xy(1+\tau_1)(1+\tau_2)(1+\delta) \equiv xy(1+\hat{\delta}_\times), \\
\mathrm{fl}(\hat{x}/\hat{y}) &= (\hat{x}/\hat{y})(1+\delta) = (x/y)(1+\tau_1)(1+\tau_2)^{-1}(1+\delta) \equiv xy(1+\hat{\delta}_\div),
\end{aligned}$$

where

$$\hat{\delta}_\times = \tau_1 + \tau_2 + \delta + O(\tau\epsilon), \quad \hat{\delta}_\div = \tau_1 - \tau_2 + \delta + O(\tau\epsilon).$$

Thus $|\hat{\delta}_\times| \le 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)$ and $|\hat{\delta}_\div| \le 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)$.

2. Forward and backward error analysis

We illustrate the basic idea through a simple example. Consider the computation of an inner product of two vector x, $y \in \mathcal{R}^3$

$$x^T y \stackrel{\mathrm{def}}{=} x_1y_1 + x_2y_2 + x_3y_3,$$

assuming already x_i's and y_j's are floating point numbers. It is likely that $\mathrm{fl}(x \cdot y)$ is computed in the following order.

$$\mathrm{fl}(x^T y) = \mathrm{fl}(\,\mathrm{fl}(\mathrm{fl}(x_1y_1) + \mathrm{fl}(x_2y_2)) + \mathrm{fl}(x_3y_3)\,).$$

Adopting the floating point arithmetic model, we have

$$\begin{aligned}
\mathrm{fl}(x^T y) &= \mathrm{fl}(\,\mathrm{fl}(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2)) + x_3y_3(1+\epsilon_3)\,) \\
&= \mathrm{fl}(\,(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2))(1+\delta_1) + x_3y_3(1+\epsilon_3)\,) \\
&= (\,(x_1y_1(1+\epsilon_1) + x_2y_2(1+\epsilon_2))(1+\delta_1) + x_3y_3(1+\epsilon_3)\,)(1+\delta_2) \\
&= x_1y_1(1+\epsilon_1)(1+\delta_1)(1+\delta_2) + x_2y_2(1+\epsilon_2)(1+\delta_1)(1+\delta_2) \\
&\quad + x_3y_3(1+\epsilon_3)(1+\delta_2),
\end{aligned}$$

where $|\epsilon_i| \le \frac{1}{2}\epsilon$ and $|\delta_j| \le \frac{1}{2}\epsilon$.

Now there are two ways to interpret the errors in the computed $\mathrm{fl}(x^T y)$:

(a) We have

$$\mathrm{fl}(x^T y) = x^T y + E,$$

where $E = x_1 y_1(\epsilon_1 + \delta_1 + \delta_2) + x_2 y_2(\epsilon_2 + \delta_1 + \delta_2) + x_3 y_3(\epsilon_3 + \delta_2) + O(\epsilon^2)$. It implies that

$$|E| \leq \frac{1}{2}\epsilon(3|x_1 y_1| + 3|x_2 y_2| + 2|x_3 y_3|) + O(\epsilon^2) \leq \frac{3}{2}\epsilon \cdot |x|^T |y| + O(\epsilon^2).$$

This bound on E tells the worst case difference between the exact $x \cdot y$ and its computed value. Such an error analysis is so-called *Forward Error Analysis.*

(b) We can also write

$$\mathrm{fl}(x^T y) = \hat{x}^T \hat{y} = (x + \Delta x)^T (y + \Delta y),$$

where[1]

$$\begin{array}{lllllll} \hat{x}_1 &=& x_1(1+\epsilon_1), & \hat{y}_1 &=& y_1(1+\delta_1)(1+\delta_2) &\equiv\ y_1(1+\hat{\delta}_1), \\ \hat{x}_2 &=& x_2(1+\epsilon_2), & \hat{y}_2 &=& y_2(1+\delta_1)(1+\delta_2) &\equiv\ y_2(1+\hat{\delta}_2), \\ \hat{x}_3 &=& x_3(1+\epsilon_3), & \hat{y}_3 &=& y_3(1+\delta_2) &\equiv\ y_3(1+\hat{\delta}_3). \end{array}$$

It can be seen that $|\hat{\delta}_1| = |\hat{\delta}_2| \leq \epsilon + O(\epsilon^2)$ and $|\hat{\delta}_3| \leq \frac{1}{2}\epsilon$. This says the computed value $\mathrm{fl}(x^T y)$ is the *exact* inner product of a slightly perturbed $\hat{x}$ and $\hat{y}$. Such an error analysis is so-called *Backward Error Analysis.*

3. Further Reading

A classical book on error analysis, where the notion of backward error analysis is invented, is

J.H. Wilkinson. Rounding Errors in Algebraic Process. Prentice-Hall, Englewood, NJ, 1964. Reprinted by Dover, New York, 1994.

A contemporary treatment of error analysis and its applications to numerical analysis is

N.J. Higham, Accuracy and stability of Numerical Algorithms. second edition, SIAM, Philadelphia, 2002.

[1]There are many ways to distribute factors $(1 + \epsilon_i)$ and $(1 + \delta_j)$ to x_i and y_j. In this case it is even possible to make either $\hat{x} \equiv x$ or $\hat{y} \equiv y$.

4 Vector and Matrix Norms

1. A **vector norm** on $\mathcal{C}^n$ is a mapping that maps each $x \in \mathcal{C}^n$ to a real number $\|x\|$, satisfying

   (a) $\|x\| > 0$ for $x \neq 0$, and $\|0\| = 0$ (positive definite property)

   (b) $\|\alpha x\| = |\alpha| \, \|x\|$ for $\alpha \in \mathcal{C}$ (absolute homogeneity)

   (c) $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)

   - Commonly used vector p-norms: Let $x = (x_1, x_2, \cdots, x_n)^T$.

     $$\|x\|_p \stackrel{\text{def}}{=} \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \qquad 1 \leq p < \infty.$$

     Most commonly used vector norms:

     $$\begin{aligned} \|x\|_1 &= \sum_{i=1}^{n} |x_i|, \quad \text{“Manhattan” or “taxi cab” norm} \\ \|x\|_2 &= \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}, \quad \text{Euclidean length} \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i|. \end{aligned}$$

   - The *geometry* of the closed unit “ball”: $\{x \in \mathcal{C}^2 : \|x\| \leq 1\}$ corresponding to each norm.
   - Norm equivalence: Let $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ be any two vector norms. There are constants $c_1, c_2 > 0$ such that

     $$c_1 \|\cdot\|_\alpha \leq \|\cdot\|_\alpha \leq c_2 \|\cdot\|_\alpha$$

     for examples, it can be easily proved that

     $$\begin{array}{lll} \|x\|_1 \leq \sqrt{n}\|x\|_2, & \|x\|_2 \leq \|x\|_1, & \|x\|_1 \leq n\|x\|_\infty, \\ \|x\|_\infty \leq \|x\|_1, & \|x\|_2 \leq \sqrt{n}\|x\|_\infty, & \|x\|_\infty \leq \|x\|_2. \end{array}$$

   - Cauchy-Schwarz inequality:

     $$|x^H y| \leq \|x\|_2 \|y\|_2.$$

     Hölder inequality:

     $$|x^H y| \leq \|x\|_p \|y\|_q,$$

     where $1 \leq p, q < \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$.

2. A **matrix norm** on $\mathcal{C}^{m \times n}$ is a mapping that maps each $A \in \mathcal{C}^{m \times n}$ to a real number $\|A\|$, satisfying

   (a) $\|A\| > 0$ for $A \neq 0$, and $\|0\| = 0$ (positive definite property)

   (b) $\|\alpha A\| = |\alpha| \, \|A\|$ for $\alpha \in \mathcal{C}$ (absolute homogeneity)

(c) $\|A+B\| \le \|A\| + \|B\|$ (triangle inequality)

Commonly used matrix norms:

- **The Frobenius Norm $\|\cdot\|_{\mathrm{F}}$:** For $A = (a_{ij}) \in \mathcal{C}^{m\times n}$,

$$\|A\|_{\mathrm{F}} \stackrel{\mathrm{def}}{=} \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2} = \sqrt{\mathrm{tr}(A^H A)}.$$

- **The induced (operator) norm $\|\cdot\|$:** Any vector norm $\|\cdot\|$ that is generic induced a (matrix) norm on $\mathcal{C}^{m\times n}$, denoted by the same notation,

$$\|A\| \stackrel{\mathrm{def}}{=} \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

  Then it can be verified that $\|A\|$ so defined is indeed a norm on $\mathcal{C}^{m\times n}$;
  - $\|A\|_1$, $\|A\|_2$, and $\|A\|_\infty$ are frequently used induced norm, induced by the vector 1, 2, and ∞-norms respectively.

$$\begin{aligned} \|A\|_1 &= \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}| = \text{ max absolute column sum,} \\ \|A\|_2 &= \sqrt{\text{the largest eigenvalue of } A^*A} = \text{the largest singular value of } A, \\ \|A\|_\infty &= \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| = \text{ max absolute row sum.} \end{aligned}$$

  - Useful property: $\|Ax\| \le \|A\|\,\|x\|$.
  - "Geometry" of $\|A\|$: the maximal factor by which A can "strech" a vector.
- $\|A\|_2^2 \le \|A\|_1 \|A\|_\infty$.

5 Frequently Used Matrix Decompositions

1. **LU decomposition** (Gaussian Elimination). If A is nonsingular, then there exist permutations P, a unit lower triangular matrix L, and a nonsingular upper triangular matrix U such that
$$PA = LU.$$
Special cases:
   (a) **Cholesky decomposition.** A matrix A is symmetric positive definite if and only if there exists a unique nonsingular upper triangular matrix R, with positive diagonal entries, such that
   $$A = R^T R.$$
   (b) **LDL$^{\mathrm{T}}$ factorization** If $A^T = A$ is nonsingular, then there exists a permutation P, a unit lower triangular matrix L, and a block diagonal matrix D with 1-by-1 and 2-by-2 blocks such that
   $$PAP^T = LDL^T.$$
   Applications:
   - Solve $Ax = b$.
   - ...

2. **QR decomposition** (Gram-Schmidt orthogonalization). Let A be m-by-n with $m \geq n$. Suppose that A has full column rank. Then there exist a unique m-by-n orthogonal matrix Q ($Q^TQ = I$) and a unique n-by-n upper triangular matrix R with positive diagonal $r_{ii} > 0$ such that
$$A = QR.$$
   Applications:
   - Find an orthonormal basis of the subspace spanned by the columns of A.
   - Solve the linear least squares problem $\min_x \|Ax - b\|_2$.

3. **Schur decomposition**. Let A be of order n. Then there is an $n \times n$ unitary matrix U ($U^HU = I$) such that
$$A = UTU^H,$$
   where T is upper triangular. By appropriate choice of U, the eigenvalues of A, which are the diagonal elements of T, may be made to appear in any order.
   Applications:
   - Compute eigenvalues and eigenvectors of A.
   - ...

4. **Singular Value Decomposition (SVD).** Let A be an m-by-n matrix with $m \geq n$. Then we can write
$$A = U\Sigma V^T,$$

where U is m-by-n orthogonal matrix ($U^TU = I_n$) and V is n-by-n orthogonal matrix ($V^TV = I$), and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. (If $m < n$, the SVD can be defined by considering A^T).

The columns $u_1, u_2, \ldots, u_n$ of U are called *left singular vectors* of A. The columns $v_1, v_2, \ldots, v_n$ of V are called *right singular vectors*. The $\sigma_1, \sigma_2, \ldots, \sigma_n$ are called singular values.

Applications:

- Suppose that A is m-by-n with $m \geq n$ and has full rank, with $A = U\Sigma V^T$ being A's SVD. Then the pseudo-inverse can also be written as

  $$A^\dagger \equiv (A^TA)^{-1}A^T = V\Sigma^{-1}U^T.$$

  (If $m < n$, then $A^\dagger = A^T(AA^T)^{-1}$)

- Suppose that

  $$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0,$$

  Then the rank of A is r. The range space of A is $\text{span}(u_1, u_2, \cdots, u_r)$. and the null space of A is $\text{span}(v_{r+1}, v_{r+2}, \ldots, v_n)$.

- $\|A\|_2 = \sigma_1 (\equiv \sigma_{\max})$

- Let A be $m \times n$ with $m \geq n$. Then
  (a) eigenvalues of A^TA are σ_i^2, $i = 1, 2, \ldots, n$. The corresponding eigenvectors are the right singular vectors v_i, $i = 1, 2, \ldots, n$.
  (b) eigenvalues of AA^T are σ_i^2, $i = 1, 2, \ldots, n$ and $m - n$ zeros. The left singular vectors u_i, $i = 1, 2, \ldots, n$ are corresponding eigenvectors for the eigenvalues σ_i^2. One can take any $m - n$ other orthogonal vectors that are orthogonal to $u_1, u_2, \ldots, u_n$ as the eigenvectors for the eigenvalues 0.

- Optimal rank-k approximation:

  $$\min_{\substack{B\,:\,m\times n \\ \text{rank}(B)=k}} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1},$$

  where $A_k = U\Sigma_k V^T$, $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0)$
  Note that A_k can be written in a compact form as

  $$A_k = U_k \widehat{\Sigma}_k V_k^T,$$

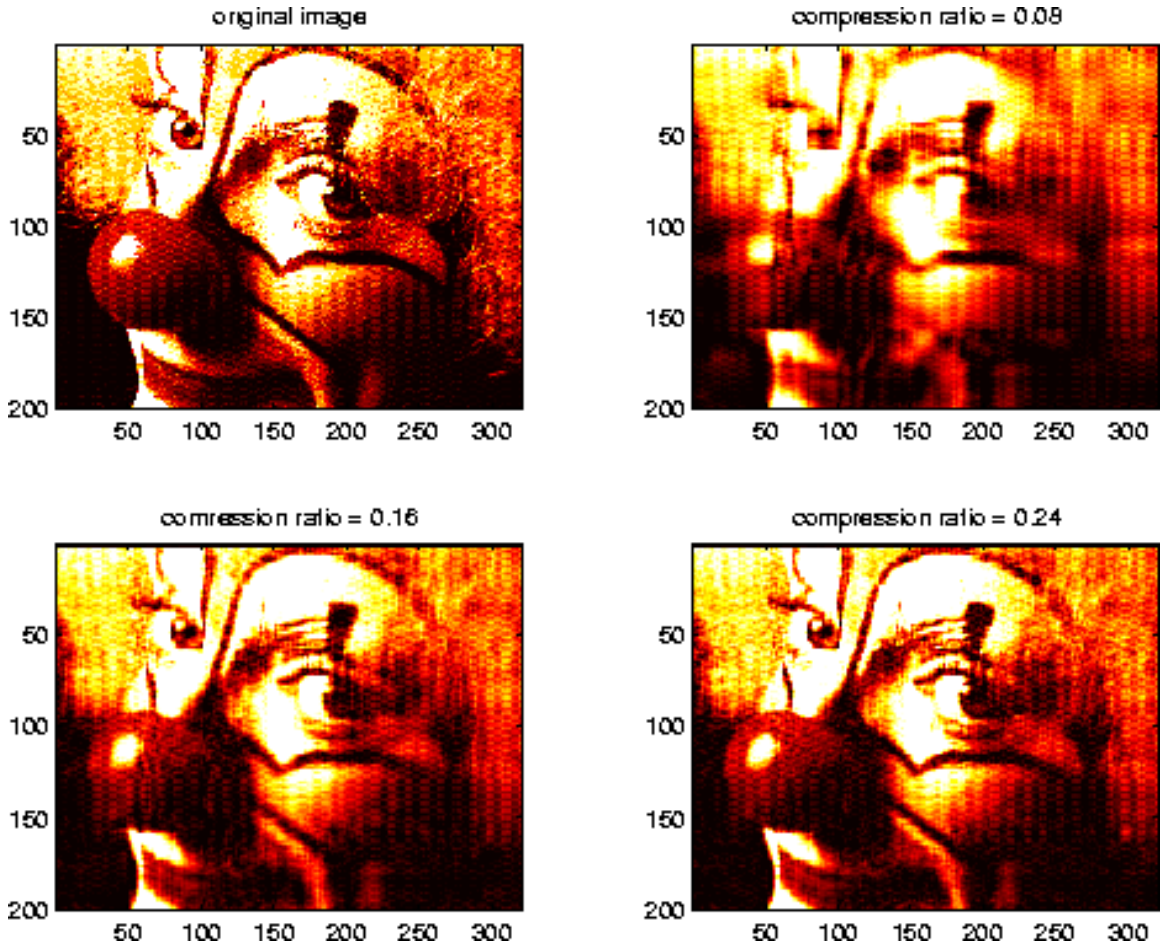
  where U_k and V_k are the first k columns of U and V, respectively, $\widehat{\Sigma}_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k)$.
  Therefore, A_k is represented by $mk + k + nk = (m + n + 1)k$ elements, in contrast, A is represented by mn elements.
  Application: image compression.

  $$\text{compression ratio} = \frac{(m + n + 1)k}{mn}$$

Matlab's M-scripts:

```
>> load clown.mat;
>> [m,n]=size(X);
>> figure(1);
>> colormap(map);
>> imag(X);
>> [U,S,V]=svd(X);
>> k = 20;
>> X20 = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
>> figure(2);
>> colormap(map);
>> image(X20);
>> compression_ratio = (m+n)*k/(m*n)
```

Output:



6 Iterative Linear Solvers – Basics

1. The landscape of linear system solvers

	Direct ($A = LU$)	Iterative ($u = Av$)
Nonsymmetric	pivoting LU	GMRES,...
Symm.pos.def.	Cholesky	Conjugate Gradient

2. A general framework for iterative projection methods

   Given the linear system of equations

$$Ax = b$$

   where A is an $n \times n$ real matrix, b and n vector. The basic idea of *projection techniques* is to extract an approximate solution from subspaces of $\mathcal{R}^n$. Let $\mathcal{W}$ and $\mathcal{V}$ be two m-dimensional subspaces of $\mathcal{R}^n$, and x_0 is a "good" initial guess of the solution, then the *projection technique* is to

$$\text{Find } \widetilde{x} \in x_0 + \mathcal{W} \text{ such that } b - A\widetilde{x} \perp \mathcal{V}. \tag{2}$$

   Write $\widetilde{x} = x_0 + z$, $z \in \mathcal{W}$ and define initial residual $r_0 = b - Ax_0$. Notice that $b - A\widetilde{x} = b - A(x_0 + z) = r_0 - Az$. Then the formulation (2) is equivalent to

$$\text{Find } z \in \mathcal{W} \text{ such that } r_0 - Az \perp \mathcal{V}. \tag{2a}$$

   This is a basic projection step in its most general form. Most standard techniques use a succession of such projections. Typically, a new projection step uses a new pair of subspaces $\mathcal{W}$ and $\mathcal{V}$ (updated from the previous step) and an initial guess x_0 equal to the most recent approximation. This simple framework is common to many numerical computing.

   Definition 1 *If $\mathcal{W} = \mathcal{V}$, then it is called* orthogonal projection *method. Otherwise, it is called* oblique projection *method.*

   Matrix Representation Let $V = [v_1, v_2, \ldots, v_m]$ be an $n \times m$ matrix whose columns form a basis of $\mathcal{V}$, and similarly $W = [w_1, w_2, \ldots, w_m]$ an $n \times m$ matrix whose columns form a basis of $\mathcal{W}$. Then any approximation solutions in $x_0 + \mathcal{W}$ can be written as

$$\widetilde{x} = x_0 + z = x_0 + Wy, \quad \text{i.e.,} \quad z = Wy,$$

   and orthogonality implies $V^T(r_0 - Az) = 0$; thus

$$V^T AWy = V^T r_0 \quad \Rightarrow \quad y = (V^T AW)^{-1} V^T r_0,$$

   provided $V^T AW$ is invertible. Putting it all together, we have

$$\widetilde{x} = x_0 + W(V^T AW)^{-1} V^T r_0.$$

   Now, we have a prototype projection method:

0. Let x_0 be an initial approximation
1. Iterate until convergence:
2. Select a pair of subspaces $\mathcal{V}$ and $\mathcal{W}$
3. Generate basis matrices V and W for $\mathcal{V}$ and $\mathcal{W}$
4. $r_0 \leftarrow b - Ax_0$
5. $y \leftarrow (V^T AW)^{-1} V^T r_0$
6. $x_0 \leftarrow x_0 + Wy$

There are two important remarks:

1. In many practical algorithms to be discussed below, the matrix $V^T AW$ does not have to be formed explicitly. It is available as a by-product of Steps 2 and 3, e.g., the Arnoldi process.
2. The method is defined only when $V^T AW$ is nonsingular, which is not guaranteed to be true even when A is nonsingular.

The following theorem states that there are two important special cases where the nonsingularity of $V^T AW$ is guaranteed.

Theorem 1 *Let A, $\mathcal{W}$, and $\mathcal{V}$ satisfy either one of the following two conditions:*

(a) A is symmetric positive definite (SPD) and $\mathcal{W} = \mathcal{V}$, or

(b) A is nonsingular, and $\mathcal{V} = A\mathcal{W}$.

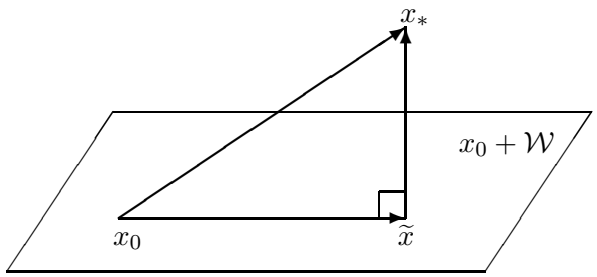
Then the matrix $V^T AW$ is invertible for any basis matrices W and V of $\mathcal{W}$ and $\mathcal{V}$, respectively.

Optimality We now discuss two important optimality results that are satisfied by the approximate solutions in some cases.

Theorem 2 *Assume that A is SPD and that $\mathcal{V} = \mathcal{W}$. Then a vector $\widetilde{x}$ is the result of (2) if and only if*

$$\|x_* - \widetilde{x}\|_A = \min_{x \in x_0 + \mathcal{W}} \|x_* - x\|_A,$$

where $\|x_* - x\|_A = \sqrt{(x_* - x)^T A (x_* - x)}$, and x_* is the exact solution to $Ax = b$.



PROOF: Notice that $(A(\cdot), \cdot)$ is an inner product on $\mathcal{R}^n$ since A is SPD. Thus $\|x_* - x\|_A$ over all possible $x \in x_0 + \mathcal{W}$ is minimized at $\widetilde{x}$ if and only if $x_* - \widetilde{x} \perp_A \mathcal{W}$, i.e.,

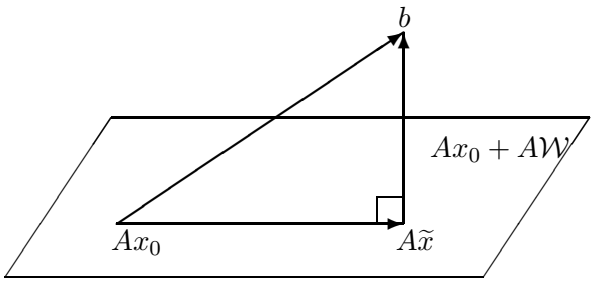
$$(A(x_* - \widetilde{x}), w) = (b - A\widetilde{x}, w) = 0 \quad \text{for any } w \in \mathcal{W} = \mathcal{V}.$$

This is (2). ∎

The corresponding methods are *steepest descent method* and *conjugate gradient (CG) method*.

Theorem 3 *Let A be an arbitrary square matrix and assume $\mathcal{V} = A\mathcal{W}$. Then a vector $\widetilde{x}$ is the result of (2) if and only if*

$$\|b - A\widetilde{x}\|_2 = \min_{x \in x_0 + \mathcal{W}} \|b - Ax\|_2.$$



PROOF: $\|b - Ax\|_2$ over all possible $x \in x_0 + \mathcal{W}$ is minimized at $\widetilde{x}$ if and only if $b - A\widetilde{x} \perp A\mathcal{W}$, i.e.,

$$(b - A\widetilde{x}, v) = 0 \quad \text{for any } v \in A\mathcal{W} = \mathcal{V}.$$

This is (2). ∎

The corresponding methods are *minimal residual residual (MR) method* and *generalized minimal residual (GMRES) method*.

3. **One-Dimensional Projection Processes** are defined when

$$\mathcal{W} = \text{span}\{w\} \quad \text{and} \quad \mathcal{V} = \text{span}\{v\},$$

where w and v are two vectors. In this case, the new approximation takes form

$$x \leftarrow x + z = x + \alpha w$$

and the condition (2a) implies $v^T(r - Az) = v^T(r - \alpha Aw) = 0$, and thus

$$\alpha = \frac{v^T r}{v^T Aw}.$$

Following are two popular choices of w and v.

(a) STEEPEST DESCENT This is for *SPD* A, and at each step $v = w = r$, the residual vector. This yields

STEEPEST DESCENT ALGORITHM:
1. Pick an initial guess x_0
2. Until convergence for $k = 0, 1, 2, \ldots$ do
3. $\quad r_k = b - Ax_k$
4. $\quad \alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$
5. $\quad x_{k+1} = x_k + \alpha_k r_k$
6. EndDo

Alternatively, we can view that each step of the above iteration minimizes

$$f(x) \stackrel{\text{def}}{=} \|x_* - x\|_A^2 = (x_* - x)^T A(x_* - x),$$

over all vectors of the form $x - \alpha(\nabla f(x))$, where $\nabla f(x)$ is the gradient of f at x. Recall that the negative of the gradient direction is locally the direction that yields the fastest rate of decrease for f.

Now we show that the convergence of the steepest descent is guaranteed when A is SPD.

Theorem 4 *Let A be SPD, and let $\lambda_{\min}$ and $\lambda_{\max}$ be its smallest and largest eigenvalues respectively. Then for the* Steepest Descent Algorithm

$$\|x_* - x_{k+1}\|_A \leq \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right) \|x_* - x_k\|_A = \left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right) \|x_* - x_k\|_A,$$

where x_* is the exact solution to $Ax = b$. $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ is the condition number of A.

Thus the Steepest Descent Algorithm *converges for any initial guess.*

(b) Minimal Residual (MR) Iteration This is for matrix A not necessary symmetric but $A + A^T$ is *SPD*[2]. At each step take $w = r$ and $v = Ar$.

Minimal Residual (MR) Iteration:
1. Pick an initial guess x_0
2. Until convergence for $k = 0, 1, 2, \ldots$ do
3. $\quad r_k = b - Ax_k$
4. $\quad \alpha_k = \frac{r_k^T A r_k}{r_k^T A^T A r_k}$
5. $\quad x_k = x_k + \alpha_k r_k$
6. EndDo

Alternatively, we can view that each step of the MR iteration minimizes

$$f(x) \stackrel{\text{def}}{=} \|r\|_2^2 = \|b - Ax\|_2^2$$

over all vectors of the form $x - \alpha r$.

Theorem 5 *Assume that $A + A^T$ is* SPD*, and let $\mu = \lambda_{\min}\left(\frac{A+A^T}{2}\right)$, and $\sigma = \|A\|_2$. Then for the MR iteration*

$$\|r_{k+1}\|_2 \leq (1 - \mu^2/\sigma^2)^{1/2} \|r_k\|_2.$$

Thus the MR iteration converges for any initial guess.

[2]This is equivalent to say that A is positive definite. A real matrix A said to be positive definite if $u^T Au > 0$ for any $0 \neq u \in \mathcal{R}$. It can be shown that if A is real positive definite, then A is nonsingular, in addition, $u^T Au \geq \lambda_{\min}\left(\frac{1}{2}(A + A^T)\right) u^T u$.

7 Iterative Linear Solvers – Krylov subspace methods I

1. **Krylov Subspace**. Given a vector $v \in \mathcal{R}^n$, the *Krylov subspace* is defined as

$$\mathcal{K}_m \equiv \mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}.$$

Note that if $x \in \mathcal{K}_m$, then $x = p(A)v$, where $p(A)$ is a polynomial of degree not exceeding $m-1$.

2. **Arnoldi procedure** is an algorithm for building an orthogonal basis of the Krylov subspace $\mathcal{K}_m(A, v)$ using a modified Gram-Schmidt orthogonalization process.[3]

ARNOLDI PROCEDURE – MODIFIED GRAM-SCHMIDT VERSION
1. $v_1 = v/\|v\|_2$
2. for $j = 1, 2, \ldots, m$
3. compute $w = Av_j$
4. for $i = 1, 2, \ldots, j$
5. $h_{ij} = v_i^T w$
6. $w := w - h_{ij} v_i$
7. end for
8. $h_{j+1,j} = \|w\|_2$
9. If $h_{j+1,j} = 0$, **Stop**
10. $v_{j+1} = w/h_{j+1,j}$
11. endfor

Proposition 1 *Assume that the Arnoldi procedure does not stop before the m-th step. Then the vectors* $\{v_1, v_2, \ldots, v_m\}$ *form an orthonormal basis of the Krylov subspace* $\mathcal{K}_m(A, v)$

Denote $V_m = [v_1, v_2, \ldots, v_m]$ and $H_m = (h_{ij})$ = Upper Hessenberg. Then in the matrix form, the Arnoldi procedure can be expressed in the following governing relations:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$$

and $V_m^T V_m = I_m$ and $V_m^T A V_m = H_m$. This is often referred to as an order-m Arnoldi decomposition.

Furthermore, if we denote $V_{m+1} = [V_m, v_{m+1}]$ and $\widehat{H}_m = \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix}$, i.e., $\widehat{H}_m$ is a $m+1$ by m upper triangular matrix, then an order-m Arnoldi decomposition can also be written in the following compact form

$$AV_m = V_{m+1} \widehat{H}_m.$$

Remarks:

- The Arnoldi procedure breaks down when $h_{j+1,j} = 0$ for some j. It is easy to see that if the Arnoldi procedure breaks down at step j (i.e. $h_{j+1,j} = 0$), then $\mathcal{K}_j$ is invariant subspace of A.

[3] Warning: Some care must be taken to insure that the vectors v_j remain orthogonal to working accuracy in the presence of rounding error. The usual technique is called reorthogonalization.

- Note that the matrix A is only referenced via the matrix-vector multiplication Av_j. Therefore, it is ideal for large sparse or large dense structure matrices. Any sparsity or structure of a matrix can be exploited in the matrix-vector multiplication.
- The main storage requirement is $(m+1)n$ for storing Arnoldi vectors $\{v_i\}$ plus the storage requirement for the matrix A in question or the required matrix-vector multiplication.
- The primary arithmetic cost of the procedure is the cost of m matrix-vector products plus $2m^2n$ for the rest. It is common that the matrix-vector multiplication is the dominant cost.

3. **Full Orthogonalization Method (FOM).** Given an initial guess x_0 to the original linear system $Ax = b$, we now consider an orthogonal projection method with

$$\mathcal{W} = \mathcal{V} = \mathcal{K}_m(A, r_0)$$

where $r_0 = b - Ax_0$. Following the general framework of a projection method, an approximate x_m is sought from the affine subspace $x_0 + \mathcal{K}_m$ of dimension m by imposing the Galerkin condition

$$b - Ax_m \perp \mathcal{K}_m.$$

With $v_1 = r_0/\|r_0\|_2$, Arnoldi procedure generates an orthogonal basis V_m of $\mathcal{K}_m$ and furthermore,

$$V_m^T A V_m = H_m \quad \text{and} \quad V_m^T r_0 = V_m^T(\beta v_1) = \beta e_1,$$

where $\beta = \|r_0\|_2$. As a result, the approximate solution is given by

$$x_m = x_0 + V_m y_m = x_0 + \beta V_m H_m^{-1} e_1.$$

Unfortunately, FOM fails if H_m is singular! In practice, GMRES method to be discussed below is a more robust variant of FOM.

The residual vector of x_m is given as the following

$$\begin{aligned} b - Ax_m &= b - A(x_0 + V_m y_m) \\ &= r_0 - AV_m y_m \\ &= \beta v_1 - V_m H_m y_m - h_{m+1,m} v_{m+1}(e_m^T y_m) \\ &= V_m(\beta e_1 - H_m y_m) - h_{m+1,m} v_{m+1}(e_m^T y_m) \\ &= -h_{m+1,m} v_{m+1}(e_m^T y_m). \end{aligned}$$

Hence

$$\|b - Ax_m\|_2 = h_{m+1,m}|e_m^T y_m|$$

Therefore, we observe that an elegant feature of the method is that the residual norm can be cheaply computed. *There is no need to form the approximate solution x_m until its accuracy is satisfied.*

Restarted FOM: As m increases, the computational cost increases at least as $O(m^2n)$. The memory cost increases as $O(mn)$. For large n this limits the largest value of m that can be used. The popular remedy is to restart the algorithm periodically for a fix m.

RESTARTED FOM:

1. compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$
2. call Arnoldi procedure with A, v_1 and m
3. compute $y_m = H_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$
4. test for convergence, if satisfied, then **Stop**
5. set $x_0 := x_m$ and go to 1.

4. The **Generalized Minimum Residual (GMRES) method**[4] based on taking

$$\mathcal{W} = \mathcal{K}_m(A, r_0) \quad \text{and} \quad \mathcal{V} = A\mathcal{W} = A\mathcal{K}_m(A, r_0).$$

We have two ways to derive the GMRES method.

- The first way exploits the optimality property. Note that any vector x in $x_0 + \mathcal{K}_m$ can be written as $x = x_0 + V_m y$, where y is an m-vector. Define

$$J(y) = \|b - Ax\|_2 = \|b - A(x_0 + V_m y)\|_2 \tag{3}$$

Then using the Arnoldi decomposition, we have

$$\begin{aligned} b - Ax &= b - A(x_0 + V_m y) = r_0 - AV_m y \\ &= \beta v_1 - V_{m+1}\widehat{H}_m y = V_{m+1}(\beta e_1 - \widehat{H}_m y). \end{aligned}$$

Since the column vectors of V_{m+1} are orthonormal, then

$$J(y) = \|b - A(x_0 + V_m y)\|_2 = \|\beta e_1 - \widehat{H}_m y\|_2.$$

Therefore, the GMRES approximation x_m is the unique vector $x_m = x_0 + V_m y$, where y the solution of the least squares problem

$$\min_y \|\beta e_1 - \widehat{H}_m y\|_2.$$

This least squares problem is inexpensive to compute since m is typically small.

- The second way to derive the GMRES algorithm is to use the framework of the projection technique. This is to be presented at the class.

Breakdown of GMRES: Since the least squares problem always has solution, the only possibility of the breakdown of the GMRES is in the Arnoldi procedure when $h_{j+1,j}$ at some step j. However, in this case, the residual norm of x_j is zero, $b - Ax_j = 0$. x_j is the exact solution. This is called *lucky breakdown*. In fact, we have

Proposition 2 *Let A be a nonsingular matrix. Then the GMRES algorithm breaks down at step j, i.e., $h_{j+1,j} = 0$, if and only if the approximate solution x_j is exact.*

Restarting GMRES method: Similar to the FOM, the GMRES method becomes impractical when m is large because of the growth of memory and computational requirements as m increases. These requirements are identical with those of FOM. As with FOM, in the practical application of the GMRES method, it is restarted periodically for a fix m.

[4] Y. Saad and M. H. Schultz. GMRES: a Generalized Minimal RESidual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing, Vol.7, pp.856–869, 1986.

5. Convergence of GMRES. We wish to establish a result to provide an upper bound on the convergence rate of the GMRES iterates. Unfortunately, because of the complication of non-Hermitian matrices and their spectral distribution, it is not possible to prove a simple result, but can get pretty close for practical use. First, we have the following lemma to characterize the approximate solution by the GMRES method:

Lemma 1 *Let x_m be the approximate solution obtained from the m-th step of the GMRES algorithm, and let $r_m = b - Ax_m$. Then x_m is of the form*

$$x_m = x_0 + q_m(A)r_0$$

and

$$\|r_m\|_2 = \|(I - Aq_m(A))r_0\|_2 = \min_{q \in \mathcal{P}_{m-1}} \|(I - Aq(A))r_0\|_2.$$

PROOF: This is true because x_m minimizes the 2-norm of the residual in the affine subspace $x_0 + \mathcal{K}_m$, the optimality property of the projection technique. Recall that $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m - 1$.

Proposition 3 *Assume that A is diagonalizable matrix and let $A = V\Lambda V^{-1}$ where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues. Define*

$$\epsilon^{(m)} = \min_{p \in \mathcal{P}_m, p(0)=1} \max_{1 \leq i \leq n} |p(\lambda_i)|.$$

Then the residual norm satisfies the inequality

$$\|r_m\|_2 \leq \kappa_2(V)\epsilon^{(m)}\|r_0\|_2.$$

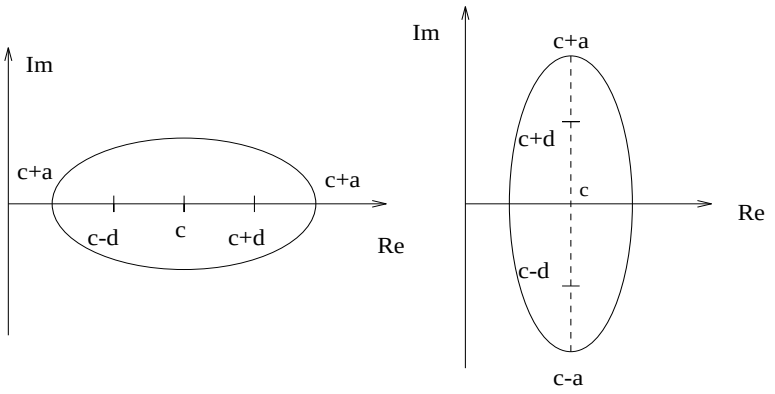
where $\kappa_2(V) = \|V\|_2\|V^{-1}\|_2$.

The results of approximation theory on near-optimal Chebyshev polynomials in the complex plane can now be used to obtain an upper bound for $\epsilon^{(m)}$. This is stated in the following corollary.

Corollary 1 *Assume that all the eigenvalues of A are located in the ellipse $E(c, d, a)$ which excludes the origin. Then*

$$\|r_m\|_2 \leq \kappa_2(V)\frac{T_m(a/d)}{T_m(c/d)}\|r_0\|_2 \lesssim \kappa_2(V)\left(\frac{a + \sqrt{a^2 - d^2}}{c + \sqrt{c^2 - d^2}}\right)^m \|r_0\|_2.$$

The follow plots show the spectrum of A is contained in the ellipses $E(c, d, a)$ with center c, focal distance d and major semi axis a. The left plot is for the case of real d and the right plot is for the case of purely imaginary d.



Since the condition number $\kappa_2(V)$ is typically not known and can be very large, results are of limited practical interest. They can be useful one when it is known that the matrix is nearly normal, in which case, $\kappa_2(V) \approx 1$.

8 Iterative Linear Solvers – Krylov subspace methods II

1. The **symmetric Lanczos procedure** can be viewed as a simplification of Arnoldi's procedure when A is symmetric.

   By an order-m Arnoldi decomposition, we know that

   $$H_m = V_m^T A V_m.$$

   If A is symmetric, then H_m becomes symmetric tridiagonal. This simple observation leads to the following procedure to compute an orthonormal basis Q_m of Krylov subspace $\mathcal{K}_m(A, v)$ when A is symmetric[5]

   LANCZOS PROCEDURE
   1. $v_1 = v/\|v\|_2$, set $\beta_1 = 0$, $v_0 = 0$
   2. for $j = 1, 2, \ldots, m$
   3. $\quad w = Av_j - \beta_j v_{j-1}$
   4. $\quad \alpha_j = v_j^T w$
   5. $\quad w := w - \alpha_j v_j$
   8. $\quad \beta_{j+1} = \|w\|_2$
   9. $\quad$ If $\beta_{j+1} = 0$, then *stop*
   10. $\quad v_{j+1} = w/\beta_{j+1}$
   11. endfor

   Note that only three vectors must be saved in the inner loop of the procedure. This is referred as a *three-term recurrence*.

   If we denote $V_m = [v_1, v_2, \ldots, v_m]$ and $T_m = \text{tridiag}(\beta_j, \alpha_j, \beta_{j+1})$. Then in the matrix form, the Lanczos procedure can be expressed in the following so-called *order−m Lanczos decomposition*:

   $$AV_m = V_m T_m + \beta_{m+1} v_{m+1} e_m^T$$

   and furthermore, $V_m^T V_m = I_m$ and $V_m^T A V_m = T_m$.

   Remarks

   - The computed Lanczos vectors $\{v_i\}$ are orthogonal in exact arithmetic. In the presence of finite precision, it starts losing such orthogonality rapidly with the increase of j. (The same phenomenon is also observed in the Arnoldi procedure, but it's not as severe as in the Lanczos procedure).

     There has been much research devoted to understanding the effect of loss of the orthogonality, and finding ways to either recover the orthogonality, or to at last diminish its effects. The best reference is [B. N. Parlett, The Symmetric Eigenvalue Problem, SIAM Press, 1998].

2. The **Conjugate Gradient (CG) method** is one of the best known iterative techniques for solving sparse SPD linear system.[6]

[5] Note that we change the notation $\alpha_j = h_{jj}$ and $\beta_{j+1} = h_{j-1,j}$, comparing with the Arnoldi procedure.

[6] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Standards, 49:409–436, 1952.

There are several ways to derive this way. In terms of our familiar subspace projection technique, we can describe the CG method in one sentence:

> the CG is a realization of an orthogonal projection technique onto the Krylov subspace $\mathcal{K}_m(A, r_0)$, where $r_0 = b - Ax_0$ with initial guess x_0.

Therefore, it is mathematically equivalent to FOM. However, because A is SPD, some simplifications resulting from the three-term Lanczos recurrence will lead to an elegant algorithm.

By the analogue of FOM, with an initial guess x_0, the approximate solution obtained from an orthogonal projection method onto $x_0 + \mathcal{K}_m(A, r_0)$ is given by

$$x_m = x_0 + V_m y_m \tag{4}$$

where y_m is the solution of the tridiagonal system

$$T_m y_m = \beta e_1,$$

where $\beta = \|r_0\|_2$.

Now, let's try to compute the solution of the tridiagonal system progressively along with the Lanczos procedure. For doing so, let's write the LU factorization of T_m as $T_m = L_m U_m$, i.e. the Gaussian elimination *without pivoting*:

$$T_m = L_m U_m = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_m & 1 \end{pmatrix} \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & \eta_{m-1} & \beta_m \\ & & & & \eta_m \end{pmatrix}$$

Then x_m is given by

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1} (\beta e_1)$$

Let $P_m = V_m U_m^{-1}$ and $z_m = L_m^{-1}(\beta e_1)$, then

$$x_m = x_0 + P_m z_m.$$

Note that p_m, the last column of P_m, can be computed from previous p_i's and v_m by the simple update

$$p_m = \eta_m^{-1}[v_m - \beta_m p_{m-1}], \tag{5}$$

where β_m is a scalar computed from the Lanczos algorithm, while η_m results from the m-th Gaussian elimination step on the tridiagonal matrix, i.e.,

$$\begin{aligned} \lambda_m &= \beta_m / \eta_{m-1} \\ \eta_m &= \alpha_m - \lambda_m \beta_m. \end{aligned}$$

In addition, it can be shown that

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix}$$

where $\zeta_m = -\lambda_m \zeta_{m-1}$. As a result, x_m can be updated at each step as

$$x_m = x_0 + [P_{m-1},\ p_m] \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} = x_0 + P_{m-1} z_{m-1} + \zeta_m p_m = x_{m-1} + \zeta_m p_m.$$

This gives the following algorithm, which is the direct version of the Lanczos algorithm:

DIRECT LANCZOS METHOD

1. compute $r_0 = b - Ax_0$, $\beta := \zeta_1 := \|r_0\|_2$, and $v_= r_0/\beta$,
2. set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3. for $m = 1, 2, \ldots,$ do
4. $\quad w := Av_m - \beta_m v_{m-1}$ and $\alpha_m = v_m^T w$
5. $\quad$ If $m > 1$ then compute $\lambda_m = \beta_m/\eta_{m-1}$ and $\zeta_m = -\lambda_m \zeta_{m-1}$
6. $\quad \eta_m = \alpha_m - \lambda_m \beta_m$
7. $\quad p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1})$
8. $\quad x_m = x_{m-1} + \zeta_m p_m$
9. $\quad$ If x_m has converged, then Stop
10. $\quad w := w - \alpha_m v_m$
11. $\quad \beta_{j+1} = \|w\|_2$ and $v_{m+1} = w/\beta_{m+1}$
12. endfor

The direct Lanczos method and the solution (4) are mathematically equivalent. However, since Gaussian elimination *without pivoting* is being used implicitly to solve the triangular system $T_m y_m = \beta e_1$, the direct version may be more prone to numerical instability.

Regarding the residual vectors $\{r_i\}$ and the vectors $\{p_i\}$, we have the followings.

Proposition 4

(a) The residual vectors $\{r_i\}$ are orthogonal to each other, i.e., $r_j^T r_i = 0$ for $i \neq j$.

(b) the vectors $\{p_i\}$ form an A-conjugate set, i.e., $p_j^T A p_i = 0$ for $i \neq j$.

A consequence of the above proposition is that a version of the algorithm can be derived by directly imposing the orthogonality and conjugacy conditions. This gives the Conjugate Gradient (CG) algorithm. We now drive this. Let express the vector x_{j+1} as

$$x_{j+1} = x_j + \alpha_j p_j$$

Therefore, the residual vectors must satisfy the recurrence

$$r_{j+1} = b - Ax_{j+1} = b - A(x_j + \alpha_j p_j) = r_j - \alpha_j A p_j. \tag{6}$$

If the r_j's are to be orthogonal, i.e., $r_j^T r_{j+1} = 0$, then it gives

$$\alpha_j = \frac{r_j^T r_j}{r_j^T A p_j}$$

Also, from (5), it is known that the next search direction p_{j+1} is a linear combination of r_{j+1} and p_j, and with proper rescaling the p vectors approximately, it can be written as

$$p_{j+1} = r_{j+1} + \beta_j p_j.$$

Thus a first consequence of the above relation is that

$$r_j^T A p_j = (p_j - \beta_{j-1} p_{j-1})^T A p_j = p_j^T A p_j.$$

i.e.,

$$\alpha_j = \frac{r_j^T r_j}{p_j^T A p_j}.$$

By imposing A-conjugacy $p_{j+1}^T A p_j = 0$, we have

$$\beta_j = -\frac{p_j^T A r_{j+1}}{p_j^T A p_j}$$

Note that from (6), $Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j)$ and therefore

$$\beta_j = \frac{1}{\alpha_j} \frac{(r_{j+1} - r_j)^T r_{j+1}}{p_j^T A p_j} = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$$

Putting these relations together gives the following CG algorithm

CONJUGATE GRADIENT (CG) METHOD

1. compute $r_0 = b - Ax_0$ and $p_0 := r_0$
2. for $j = 0, 1, 2, \ldots,$ until convergence do
3. $\quad \alpha_j = \frac{r_j^T r_j}{p_j^T A p_j}$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j A p_j$
6. $\quad \beta_j = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$
7. $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
8. endfor

Note that the scalars α_j and β_j here are different from those of the direct Lanczos method. In addition to the matrix A, four vectors of storage are required: x, p, Ap and r.

3. An alternative derivation of the CG method is presented in the following excellent paper (pdf file is available at the class website)

   Jonathan Shewchuk, *An Introduction to Conjugate Gradient Method Without the Agonizing Pain.* 1994 (64 pages)

4. From the optimality of the projection technique, we know that the approximate solution obtained from the m-th step of the CG algorithm minimizes the A-norm of the error in the affine subspace $x_0 + \mathcal{K}_m(A, r_0)$. Since $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m - 1$, we conclude the following lemma which characterizes the approximate solution x_m:

Lemma 2 *Let x_m be the approximate solution obtained from the m-th step of the CG algorithm, and let $d_m = x_* - x_m$ where x_* is the exact solution of $Ax = b$. Then x_m is of the form*

$$x_m = x_0 + q_m(A)r_0$$

where q_m is a polynomial of degree $m-1$ such that

$$\|(I - Aq_m(A))d_0\|_A = \min_{q \in \mathcal{P}_{m-1}} \|(I - Aq(A))d_0\|_A$$

From this, we have the following theorem.

Theorem 6 *Let x_m be the approximate solution obtained from the m-th step of the CG algorithm, and x_* is the exact solution of $Ax = b$. Then,*

$$\|x_* - x_m\|_A \leq \frac{1}{T_m(1+2\eta)} \|x_* - x_0\|_A, \tag{7}$$

where T_m is the Chebyshev polynomial of degree m, and $\eta = \lambda_{\min}/(\lambda_{\max} - \lambda_{\min})$. $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of A.

A slightly different formulation of inequality can be derived. Using the relation

$$T_m(t) = \frac{1}{2}\left[\left(t + \sqrt{t^2-1}\right)^m + \left(t + \sqrt{t^2-1}\right)^{-m}\right] \geq \frac{1}{2}\left(t + \sqrt{t^2-1}\right)^m.$$

Then

$$T_m(1+2\eta) \geq \frac{1}{2}\left(1 + 2\eta + \sqrt{(1+2\eta)^2 - 1}\right)^m = \frac{1}{2}\left(1 + 2\eta + 2\sqrt{\eta(\eta+1)}\right)^m.$$

Now notice that

$$\begin{aligned} 1 + 2\eta + 2\sqrt{\eta(\eta+1)} &= (\sqrt{\eta} + \sqrt{\eta+1})^2 = \frac{(\sqrt{\lambda_{\min}} + \sqrt{\lambda_{\max}})^2}{\lambda_{\max} - \lambda_{\min}} \\ &= \frac{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}} = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \end{aligned}$$

where κ is the condition number of A, $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$. Substituting into the inequality (7) yields

$$\|x_* - x_m\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m \|x_* - x_0\|_A.$$

This bound is similar to that of the steepest descent algorithm except that the condition number of A is now replaced by its square root. CG method could be of order of magnitudes faster than the steepest descent algorithm. For example, let $\kappa = 10^3$, if one wants

$$\left(\frac{k-1}{k+1}\right)^{m_1} = \left(\frac{\sqrt{k}-1}{\sqrt{k}+1}\right)^{m_2} = 10^{-2}$$

then it means that the steepest descent algorithm needs to take $m_1 \approx 2300$ iterations to reach the same level of accuracy as $m_2 \approx 73$ iterations of the CG method.

The above analysis using the condition number does not explain all the important convergence behavior of CG. In fact, the entire distribution of eigenvalues of A is important, not just the ratio of the largest to the smallest one. If the largest and smallest eigenvalues of A are few in number (or clustered closely together), then CG will converge much more quickly than the above analysis based just on A's condition number would indicate. Any important fact is that the behavior of CG in floating point arithmetic can differ significantly from its behavior in exact arithmetic[7].

[7] A. Greenbaum and Z. Strakos, Predicting the behavior of finite precision Lanczos and conjugate gradient computations, SIMAX, 13:121-137, 1992.

A. Greenbaum, Iterative Methods for Solving Linear Systems, SIAM, Philadelphia, 1997.

9 Preconditioning Techniques

1. In the convergence analysis of CG and GMRES algorithms, we saw that the convergence rate of the CG depends on the condition number of A, or more generally the distribution of A's eigenvalues. Other Krylov subspace methods have the similar property.

   Preconditioning means replacing the system $Ax = b$ with the modified systems

   $$M^{-1}Ax = M^{-1}b. \tag{8}$$

   If M is SPD, then one can precondition symmetrically and solve the modified linear system

   $$L^{-1}AL^{-T}y = L^{-1}b, \quad x = L^{-T}y, \tag{9}$$

   where $M = LL^T$. The matrix L could be the Cholesky factor of M or any other matrix satisfying $M = LL^T$.

   The desired *preconditioner* M should be chosen so that

   (a) $M^{-1}A$ or $L^{-1}AL^{-T}$ is "well-conditioned" or approximates "the identity matrix",

   (b) linear systems with coefficient matrix M are easy to solve.

   A careful, problem-dependent choice of M can often make the condition number of the modified system much smaller than the condition number of the original one, and thus accelerate convergence dramatically. Indeed, a good preconditioner is often necessary for an iterative method to converge at all, and much current research in iterative methods is directed at finding better preconditioners.

   More specifically, a good preconditioner M depends on the iterative method being used.

   - For CG and related methods, one would like the condition number of the symmetrically preconditioned matrix $L^{-1}AL^{-T}$ to be close to one, in order for the error bound based on the Chebyshev polynomial to be small, or alternatively, has few extreme eigenvalues.
   - For GMRES, a preconditioned matrix that is close to normal and whose eigenvalues are tightly clustered around some point away from the origin would be good, but other properties might also suffice to define a good preconditioner.

2. Preconditioned Conjugate Gradient.

   If the CG algorithm is applied directly to the symmetric preconditioned system (9), the iterative kernels satisfy

   $$\begin{aligned} y_{j+1} &= y_j + \alpha_j \hat{p}_j \\ \hat{r}_{j+1} &= \hat{r}_j - \hat{\alpha}_j A\hat{p}_j \\ \hat{p}_{j+1} &= \hat{r}_{j+1} + \hat{\beta}_j \hat{p}_j \end{aligned}$$

   with

   $$\hat{\alpha}_j = \frac{\hat{r}_j^T \hat{r}_j}{\hat{p}_j^T L^{-1}AL^{-T}\hat{p}_j} \quad \text{and} \quad \hat{\beta}_j = \frac{\hat{r}_{j+1}^T \hat{r}_{j+1}}{\hat{r}_j^T \hat{r}_j}.$$

Defining

$$x_j = L^{-T} y_j, \quad r_j = L\hat{r}_j, \quad p_j = L^{-T}\hat{p}_j.$$

We obtained the following preconditioned CG algorithm for $Ax = b$.

PRECONDITIONED CONJUGATE GRADIENT (PCG)

1. compute $r_0 = b - Ax_0$, solve $Mz_0 = r_0$ and $p_0 := z_0$
2. for $j = 0, 1, 2, \ldots,$ until convergence do
3. $\quad \alpha_j = (r_j^T z_j)/(p_j^T A p_j)$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j A p_j$
6. $\quad$ solve $Mz_{j+1} = r_{j+1}$
7. $\quad \beta_j = (r_{j+1}^T z_{j+1})/(r_j^T z_j)$
8. $\quad p_{j+1} = z_{j+1} + \beta_j p_j$
9. endfor

3. Preconditioned GMRES.

The GMRES applies to the modified system (8) is straightforward.

PRECONDITIONED GMRES

1. compute $r_0 = M^{-1}(b - Ax_0)$, $\beta = \|r_0\|_2$ and $v_1 := r_0/\beta$
2. for $j = 0, 1, 2, \ldots, m$ do
3. $\quad$ compute $w := M^{-1} A v_j$
4. $\quad$ for $i = 1, 2, \ldots, j$ do
5. $\quad\quad h_{ij} = v_i^T w$
6. $\quad\quad w := w - h_{ij} w_i$
7. $\quad$ end do
8. $\quad$ compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$
9. end do
10. let y_m be the solution of $\min_y \|\beta e_1 - \widehat{H}_m y\|_2$
11. $x_m = x_0 + V_m y_m$
12. If satisfied, **Stop**, else set $x_0 := x_m$ and GOTO 1.

Note that in the above algorithm, $V_m = [v_1, v_2, \ldots, v_m]$ and $\widehat{H}_m$ is a $(m+1) \times m$ upper triangular matrix with the entries h_{ij} computed at steps (4) and (8).

4. Preconditioning Techniques.

The reliability and robustness of iterative techniques, when dealing with various applications, often depends much more on the quality of the preconditioner than on the particular Krylov subspace methods used. Finding a good preconditioner to solve a given sparse linear system is oftne viewed as a combination of art and science. Preconditioners can be divided roughly into three categories:

I. Preconditioners designed for general classes of matrices; e.g. Jacobi, Gauss-Seidel, SOR, incomplete LU factorization, incomplete Cholesky decomposition, approximate inverse.

II. Preconditioners designed for broad classes of underlying problems; e.g. elliptic partial differential equations (such as Poisson equation). Examples are multigrid and domain decomposition preconditioners.

III. Preconditioners designed for a specific matrix or underlying problem; e.g. for the transport equation.

5. ILU Factorization Preconditioners.

Except for diagonal matrices, the solution of the linear system with coefficient matrix M requires that we have a suitable decomposition of M. In many instances this will be an LU decomposition. The idea of an incomplete LU preconditioner is to perform an abbreviated (sparse) form of Gaussian elimination of A and to declare the production of the resulting factors to be M. Since M is by construction already factorized, system involving M will be easy to solve.

Let us first introduce a **sparsity set** $\mathcal{Z}$ to control the patterns of zeros. Specifically, let $\mathcal{Z}$ be a set of ordered pairs of integers from $\{1, 2, \ldots, n\}$ containing no pairs of the form (i, i). An incomplete LU factorization of A is a decomposition of the form

$$A = LU + E, \tag{10}$$

where L is unit lower triangular, and U is upper triangular, and L, U and E have the following properties

(a) If $(i, j) \in \mathcal{Z}$ with $i > j$, then $\ell_{ij} = 0$,

(b) If $(i, j) \in \mathcal{Z}$ with $i < j$, then $u_{ij} = 0$,

(c) If $(i, j) \notin \mathcal{Z}$, then $e_{ij} = 0$.

In other words, the elements of L and U are zero on the sparsity set $\mathcal{Z}$, and off the sparsity set the decomposition reproduces A.

It is instructive to consider two extreme cases. (1) If the sparsity $\mathcal{Z}$ set is empty, we get the LU decomposition of A, i.e., we are using A as a preconditioner. (2) If $\mathcal{Z}$ is everything except diagonal pairs of the form (i, i), then we are effectively using the diagonal of A as a preconditioner.

Let us consider an ILU algorithm to generate L and U rowwise. Suppose we have computed the first $k-1$ rows of L and U, and we wish to compute the kth row. Write the first k rows of (10) in the form

$$\begin{bmatrix} A_{11} & A_{1k} \\ a_{k1}^T & a_{kk}^T \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ l_{1k}^T & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{1k} \\ 0 & u_{kk}^T \end{bmatrix} + \begin{bmatrix} R_{11} & E_{1k} \\ e_{k1}^T & e_{kk}^T \end{bmatrix}.$$

we need to compute l_{1k}^T and u_{kk}^T. Multiplying out, we find that

$$l_{1k}^T U_{11} + e_{k1}^T = a_{k1}^T \tag{11}$$

and

$$u_{kk}^T + e_{kk}^T = a_{kk}^T - l_{1k}^T U_{1k}$$

We then can solve these two systems in order:

$$\underbrace{\ell_{k1}, \ell_{k2}, \ldots, \ell_{k,k-1}}_{l_{1k}^T}, \ \underbrace{\nu_{kk}, \nu_{k,k+1}, \ldots, \nu_{k,n}}_{u_{kk}^T}.$$

Suppose that we have computed $\ell_{k1}, \ell_{k2}, \ldots, \ell_{k,j-1}$. If (k,j) is in the sparsity set, we simply set $\ell_{kj} = 0$. If (k,j) is not in the sparsity set, then $r_{kj} = 0$, and the equation (11) gives

$$\alpha_{kj} = \sum_{i=1}^{k-1} \ell_{ki}\nu_{ij} + \ell_{kj}\nu_{jj},$$

from which we get

$$\ell_{kj} = \frac{\alpha_{kj} - \sum_{i=1}^{k-1} \ell_{ki}\nu_{ij}}{\nu_{jj}}.$$

The key observation here is that it does not matter how the values of the preceding ℓ's and ν's were determined. If ℓ_{kj} is defined in this way, then when we compute LU, its (k,j)-element will be α_{kj}. Thus we set ℓ's and ν's to zero on the sparsity set without interfering with the values of LU off the sparsity set. A similar procedure applies to the determination of $\nu_{kk}, \nu_{k,k+1}, \ldots, \nu_{k,n}$.

```
INCOMPLETE_LU_FACTORIZATION(A, 𝒦)
1.  for k = 1 to n
2.      for j = 1 to k − 1
3.          if ((k, j) ∈ 𝒵)
4.              L(k, j) = 0
5.          else
6.              L(k, j) = (A(k, j) − L(k, 1 : j − 1) ∗ U(1 : j − 1, j))/U(j, j)
7.          end if
8.      end for j
9.      for j = k to n
10.         if ((k, j) ∈ 𝒵)
11.             U(k, j) = 0
12.         else
13.             U(k, j) = (A(k, j) − L(k, 1 : k − 1) ∗ U(1 : k − 1, j)
14.         end if
15.     end for j
16. end for k
```

The algorithm can be carried to completion provided the quantities $U(j,j)$ are all nonzero, in which case the decomposition is unique. Whether or not the $U(j,j)$ are nonzero will depend on the matrix in question. For the following two classes of matrices, the algorithm always works.

(a) If A is nonsingular diagonally dominant matrix, then A has an incomplete LU factorization for any sparsity set $\mathcal{Z}$.

Note: A matrix A of order n is *diagonally dominant* if

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, \qquad \text{for } i = 1, 2, \ldots, n.$$

It is strictly diagonally dominant if strictly inequality holds for all j.

It can be shown that *A strictly diagonally dominant matrix is nonsingular*. Be aware that diagonal dominance alone does not imply either nonsingularity or singularity. For examples, let

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & -1 \\ 1 & 2 & 4 \end{pmatrix}.$$

Then A is nonsingular. On the other hand, B is singular.

(b) The incomplete LU factorization also exists for any M-matrix.

Note: A matrix is said to be an M-matrix if it satisfies the following properties:

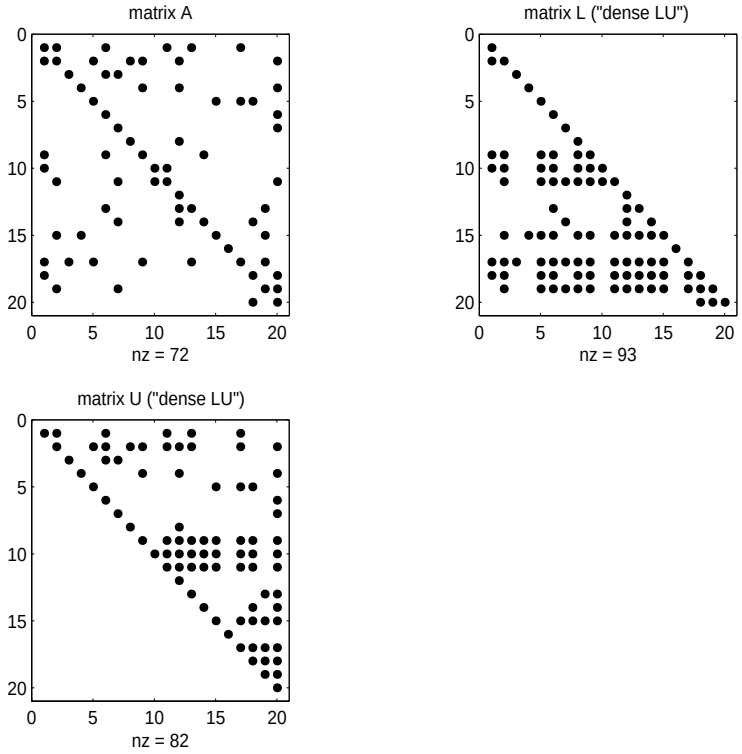
(1) $a_{ii} > 0$ for $i = 1, \ldots n$,

(2) $a_{ij} \leq 0$ for $i \neq j, i, j = 1, \ldots n$,

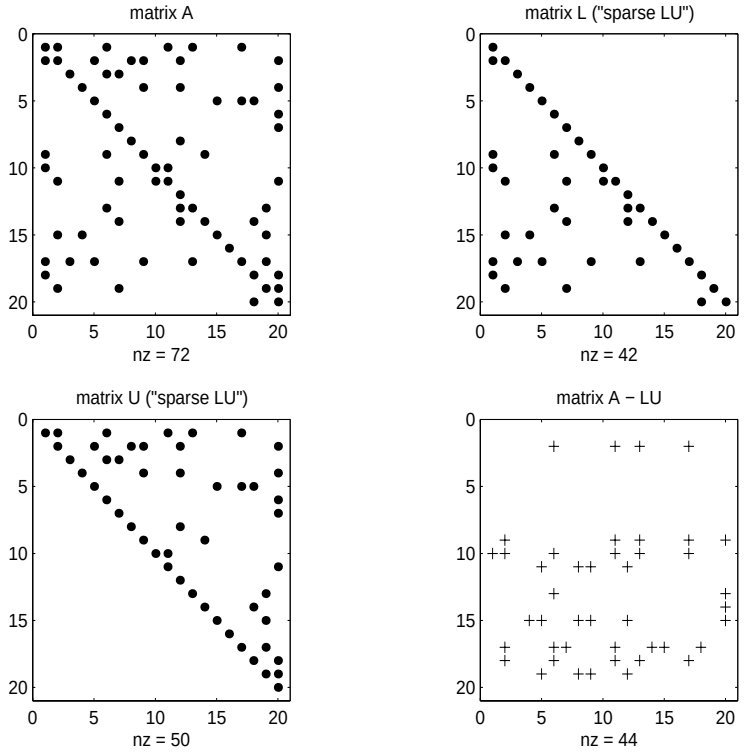
(3) A is nonsingular and

(4) A^{-1} is a nonnegative matrix (all entries are nonnegative).

6. The following figure shows the LU factorization of a sparse 20 by 20 matrix



Then an ILU factorization (and E-factor) of the same matrix.



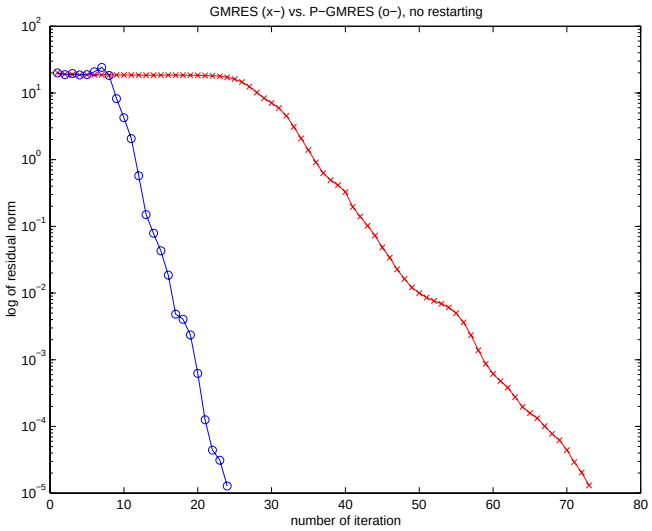
7. Block preconditioner is a popular technique for block-tridiagonal matrices arising from the discretization of elliptic problems, such as Poisson's equation. It can be also be generalized to other sparse matrices. For example, the matrix arises in the solution of 2D Poisson's equation has the form

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix}$$

where T is a symmetric tridiagonal matrix, with diagonal entres all 4, and off diagonal entries all -1. In this case, a natural preconditioner is

$$M = \text{diag}(T, T, \ldots, T).$$

8. The following figure shows the convergence history of GMRES with and without preconditioning for solving a linear system of equations arising from a discretization of a model convection-diffusion equation. The preconditioner used here is ILU(0), i.e., ILU factorization with the same sparsity pattern of A.



9. Iterative methods in Matlab

functions	methods
`pcg`	Preconditioned Conjugate Gradients Method.
`gmres`	Generalized Minimum Residual Method.
`bicg`	BiConjugate Gradients Method.
`bicgstab`	BiConjugate Gradients Stabilized Method.
`cgs`	Conjugate Gradients Squared Method.
`minres`	Minimum Residual Method.
`qmr`	Quasi-Minimal Residual Method.
`symmlq`	Symmetric LQ Method.

Preconditioners:

functions	preconditioners
`luinc`	Incomplete LU factorization.
`cholinc`	Incomplete Cholesky factorization.

10. Further Reading

- Yousef Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003
- A. Greenbaum, Iterative Methods for Solving Linear Systems. SIAM, 1997.
- H. van der Vorst, Iterative Krylov Methods for Large Linear Systems, Cambridge Univ. Press, 2003
- R. Barrett *et al*, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994 (linked to our class website)

10 Algebraic Eigenvalue Problems

1. Let $A \in \mathcal{C}^{n\times n}$.

   (a) A scalar λ is an *eigenvalue* of an $n \times n$ A and a nonzero vector $x \in \mathcal{C}^n$ is a corresponding *(right) eigenvector* if

   $$Ax = \lambda x.$$

   $\mathcal{L}_{A,\lambda} \stackrel{\text{def}}{=} \{x : Ax = \lambda x\}$ is an *eigenspace* of A.

   (b) A nonzero vector y such that

   $$y^H A = \lambda y^H$$

   is a left eigenvector.

   (c) The set $\lambda(A)$ of all eigenvalues of A is called the *spectrum* of A.

   (d) $p_A(\lambda) \stackrel{\text{def}}{=} \det(\lambda I - A)$, a polynomial of degree n, is called *characteristic polynomial* of A.

   The following is a list of properties straightforwardly from the definition

   (a) λ is A's eigenvalue $\Leftrightarrow \lambda I - A$ is singular $\Leftrightarrow \det(\lambda I - A) = 0 \Leftrightarrow p_A(\lambda) = 0$.

   (b) There is at least one eigenvector x associated with A's eigenvalue λ; in the other word, the dimension $\dim(\mathcal{L}_{A,\lambda}) \geq 1$.

   (c) $\mathcal{L}_{A,\lambda}$ is a subspace, i.e., it has the following two properties:

   i. $x \in \mathcal{L}_{A,\lambda} \Rightarrow \alpha x \in \mathcal{L}_{A,\lambda}$ for all $\alpha \in \mathcal{C}$.

   ii. $x_1, x_2 \in \mathcal{L}_{A,\lambda} \Rightarrow x_1 + x_2 \in \mathcal{L}_{A,\lambda}$.

   (d) Suppose A is real. λ is A's eigenvalue $\Leftrightarrow$ conjugate $\bar{\lambda}$ is also A's eigenvalue.

   (e) A is singular $\Leftrightarrow 0$ is A's eigenvalue.

   (f) If A is upper (or lower) triangular, then its eigenvalues consist of its diagonal entries.

2. Let $Ax_i = \lambda_i x_i$, $x_i \neq 0$ for $i = 1, 2, \ldots k$, and $\lambda_i \neq \lambda_j$ for $i \neq j$. Then $x_1, x_2, \ldots, x_k$ are linearly independent (proof by induction)

3. $A \in \mathcal{C}^{n\times n}$ is *simple* if it has n linearly independent eigenvectors; otherwise it is *defective*.

   Examples

   (a) I and any diagonal matrices is simple. $e_1, e_2, \ldots, e_n$ *are* n linearly independent eigenvectors.

   (b) $\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ is simple. It has two different eigenvalues -1 and 5. By the fact that each eigenvalue corresponds to at least one eigenvector, it must have 2 linearly independent eigenvectors.

   (c) If $A \in \mathcal{C}^{n\times n}$ has n different eigenvalues, then A is simple.

   (d) $\begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}$ is defective. It has two repeated eigenvalues 2, but only one eigenvector $e_1 = (1, 0)^T$.

4. An *invariant subspace* of A is a subspace $\mathcal{V}$ of $\mathcal{R}^n$, with the property that $v \in \mathcal{V}$ implies that $Av \in \mathcal{V}$. We also write this as $A\mathcal{V} \subseteq \mathcal{V}$.

   Examples:

   (1) The simplest, one-dimensional invariant subspace is the set span(x) of all scalar multiples of an eigenvector x.

   (2) Let $x_1, x_2, \ldots, x_m$ be any set of independent eigenvectors with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$. Then $\mathcal{X} = \text{span}(\{x_1, x_2, \ldots, x_m\})$ is an invariant subspace.

5. Let A be n-by-n, let $V = [v_1, v_2, \ldots, v_m]$ be any n-by-m matrix with linearly independent columns, and let $\mathcal{V} = \text{span}(V)$, the m-dimensional space spanned by the columns of V. Then $\mathcal{V}$ is an invariant subspace if and only if there is an m-by-m matrix B such that

$$AV = VB.$$

   In this case the m eigenvalues of B are also eigenvalues of A.

6. Similarity transformations: $n \times n$ matrices A and B are <u>*similar*</u> if there is an $n \times n$ non-singular matrix P such that $B = P^{-1}AP$. We also say A is *similar* to B, and likewise B is similar to A; P is a *similarity transformation*. A is *unitarily similar* to B if P is unitary.

7. Suppose that A and B are similar: $B = P^{-1}AP$.

   (a) A and B have the same eigenvalues. In fact $p_A(\lambda) \equiv p_B(\lambda)$.

   (b) $Ax = \lambda x \Rightarrow B(P^{-1}x) = \lambda(P^{-1}x)$.

   (c) $Bw = \lambda w \Rightarrow A(Pw) = \lambda(Pw)$.

8. Schur decomposition or Schur canonical form: Let A be of order n. Then there is an $n \times n$ unitary matrix U ($U^HU = I$) such that

$$A = UTU^H,$$

   where T is upper triangular. By appropriate choice of U, the eigenvalues of A, which are the diagonal elements of T, may be made to appear in any order.

   Exercise: The following exercises develop an analogue of the Schur decomposition for real matrices:

   (a) Let the columns of X form an orthonormal basis for an invariant subspace of A. Let $AX = XL$, and let (X, Y) be unitary. Show that

$$\begin{pmatrix} X^H \\ Y^H \end{pmatrix} A \begin{pmatrix} X & Y \end{pmatrix} = \begin{pmatrix} L & H \\ 0 & M \end{pmatrix}$$

   (b) Let A be real, and let λ be a complex eigenvalue of A with eigenvector $x + iy$. Show that the space spanned by x and y is an invariant subspace of A.

   (c) (Real Schur Decomposition). Show that if A is real, there is an orthogonal matrix U such that U^TAU is block triangular with 1×1 and 2×2 blocks on its diagonal. The 1×1 blocks contain the real eigenvalues of A, and the eigenvalues of the 2×2 blocks are the complex eigenvalues of A.

9. The Power method

THE POWER METHOD
Given an initial vector x_0,
$i = 0$
repeat
  $y_{i+1} = Ax_i$
  $x_{i+1} = y_{i+1}/\|y_{i+1}\|_2$ (approximate eigenvector)
  $\mu_{i+1} = x_{i+1}^H A x_{i+1}$ (approximate eigenvalue)
  $i = i + 1$
until convergence

Practical stopping criterion

$$|\mu_{i+1} - \mu_i| \leq tol \cdot |\mu_i|.$$

Example. Let

$$A = \begin{bmatrix} -261 & 209 & -49 \\ -530 & 422 & -98 \\ -800 & 631 & -144 \end{bmatrix}$$

Then $\lambda(A) = \{10, 4, 3\}$. Let $x_0 = e_1$, by the power method, we have

i	1	2	3	$\cdots$	10
μ_i	994.49	13.0606	10.07191	$\cdots$	10.0002

Convergence analysis: Assume $A = S\Lambda S^{-1}$ with

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \qquad |\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|.$$

It can be shown that

(a) $x_i = \frac{A^i x_0}{\|A^i x_0\|}$ converges to $s_1/\|s_1\|$, where $s_1 = Se_1$ as $i \to \infty$.

(b) μ_i converges to λ_1 as $i \to \infty$.

(c) Convergence rate depends on $\frac{|\lambda_2|}{|\lambda_1|}$

Drawback: if $\frac{|\lambda_2|}{|\lambda_1|}$ is close to 1, then the PM could be very slow convergent or doesn't converge at all.

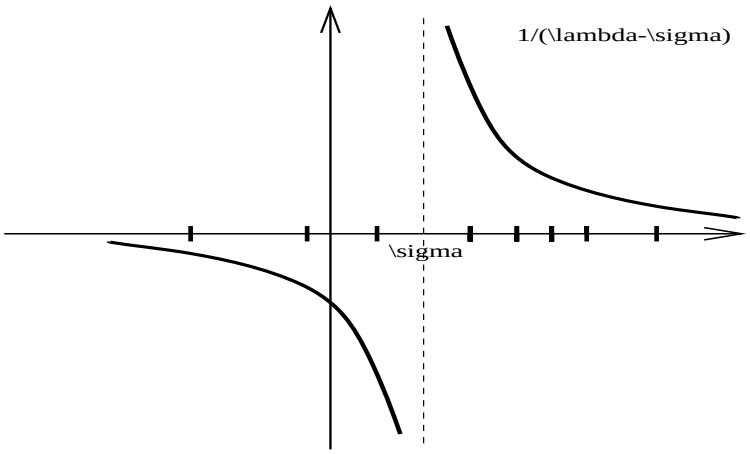
10. Inverse iteration: (a) overcome the drawbacks of the power method (slow convergence) (b) find an eigenvalue closest to a particular given number (called *shift*): σ

Observation: if λ is an eigenvalue of A, then

(1) $\lambda - \sigma$ is an eigenvalue of $A - \sigma I$,

(2) $\frac{1}{\lambda - \sigma}$ is an eigenvalue of $(A - \sigma I)^{-1}$.

Shift-and-invert spectral transformation:



Pseudo-code:

INVERSE ITERATION
Given an initial vector x_0 and a shift σ
$i = 0$
repeat
  $y_{i+1} = (A - \sigma I)^{-1} x_i$
  $x_{i+1} = y_{i+1} / \|y_{i+1}\|_2$ (approximate eigenvector)
  $\mu_{i+1} = x_{i+1}^H A x_{i+1}$ (approximate eigenvalue)
  $i = i + 1$
until convergence

Convergence analysis: Assume $A = S\Lambda S^{-1}$ with $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and λ_k is the eigenvalue cloest to the shift σ. It can be shown that

(a) x_i converges to $s_k/\|s_k\|$, where $s_k = Se_k$ $i \to \infty$.

(b) μ_i converges to λ_k $i \to \infty$.

(c) Convergence rate depends on $\max_{j \neq k} \frac{|\lambda_k - \sigma|}{|\lambda_j - \sigma|}$.

Advantage: the advantage of inverse iteration over the power method is the ability to converge to any desired eigenvalue (the one nearest to the shift σ). By choosing σ very close to a desired eigenvalue, the method converges very quickly and thus not be as limited by the proximity of nearby eigenvalues as is the original power method. The method is particularly effective when we have a good approximation to an eigenvalue and want only its corresponding eigenvector.

Drawbacks: (a) expensive in general: solving $(A - \sigma I)y_{i+1} = x_i$ for y_{i+1}. One LU factorization of $A - \sigma I$ is required. (b) Only compute one eigenpair .

11 Eigensolvers based Iterative Subspace Projection I

1. General framework -Orthogonal Projection Methods

   Let A be an $n \times n$ complex matrix and $\mathcal{K}$ be an m-dimensional subspace of $\mathcal{C}^n$. An orthogonal projection technique seeks an approximate eigenpair

   $$(\tilde{\lambda}, \tilde{u}) \quad \text{with} \quad \tilde{\lambda} \in \mathcal{C} \text{ and } \tilde{u} \in \mathcal{K}.$$

   by imposing the following Galerkin condition:

   $$A\tilde{u} - \tilde{\lambda}\tilde{u} \perp \mathcal{K} \ , \tag{12}$$

   or, equivalently,

   $$v^T(A\tilde{u} - \tilde{\lambda}\tilde{u}) = 0, \quad \forall \ v \in \mathcal{K}. \tag{13}$$

   To translate this into a matrix problem, assume that an orthonormal basis $\{v_1, v_2, \ldots, v_m\}$ of $\mathcal{K}$ is available. Denote $V = [v_1, v_2, \ldots, v_m]$, and let $\tilde{u} = Vy$ Then, equation (13) becomes

   $$v_j^T(AVy - \tilde{\lambda}Vy) = 0, \quad j = 1, \ldots, m.$$

   Therefore, y and $\tilde{\lambda}$ must satisfy

   $$B_m y = \tilde{\lambda} y \tag{14}$$

   with

   $$B_m = V^H A V.$$

   The eigenvalues $\tilde{\lambda}_i$ of B_m are called *Ritz value* values, and the vectors Vy_i are called *Ritz vector*. This procedure A is known as the *Rayleigh-Ritz procedure*:

   Rayleigh-Ritz Procedure

   (a) Compute an orthonormal basis $\{v_i\}_{i=1,\ldots,m}$ of the subspace $\mathcal{K}$. Let $V = [v_1, v_2, \ldots, v_m]$.
   (b) Compute $B_m = V^H A V$;
   (c) Compute the eigenvalues of B_m and select the k desired ones $\tilde{\lambda}_i, i = 1, 2, \ldots, k$, where $k \leq m$.
   (d) Compute the eigenvectors $y_i, i = 1, \ldots, k$, of B_m associated with $\tilde{\lambda}_i, i = 1, \ldots, k$, and the corresponding approximate eigenvectors of A, $\tilde{u}_i = Vy_i, i = 1, \ldots, k$.

   The numerical solution of the $m \times m$ eigenvalue problem in steps (c) and (d) can be treated by standard algorithms for solving small dense eigenvalue problems. Another important note is that in step (d) one can replace eigenvectors by Schur vectors to get approximate Schur vectors $\tilde{u}_i$ instead of approximate eigenvectors. Schur vectors y_i can be obtained in a numerically stable way and, in general, eigenvectors are more sensitive to rounding errors than are Schur vectors.

2. The Rayleigh-Ritz procedure and optimality:

   As we know, the simplest eigenvalue problem is to compute just the largest eigenvalue in absolute value, along with its eigenvector. The power method is the simplest algorithm suitable for this task. Starting with a given x_0, k iterations of the power

method produce a sequence of vectors $x_0, x_1, x_2, \ldots, x_k$. It is easy to see that these vectors span a *Krylov Subspace*:

$$\text{span}\{x_0, x_1, x_2, \ldots, x_k\} = \mathcal{K}_{k+1}(A, x_0) = \text{span}\{x_0, Ax_0, A^2x_0, \ldots, A^kx_0\}.$$

Now, rather than taking x_k as out approximate eigenvector, it is natural to ask for the "best" approximate eigenvector in $\mathcal{K}_{k+1}(A, x_0)$. We will see that the best eigenvector (and eigenvalue) approximations from $\mathcal{K}_{k+1}(A, x_0)$ are much better than x_k alone.

Let $Q = [Q_k, Q_u]$ be any n-by-n orthogonal matrix, where Q_k is n-by-k, and Q_u is n-by-$(n-k)$. In practice, the column of Q_k will be computed by the Lanczos algorithm and span a Krylov subspace. But for now, we do not care where we get Q.

Let

$$T = Q^TAQ = [Q_k, Q_u]^TA[Q_k, Q_u] = \begin{bmatrix} Q_k^TAQ_k & Q_k^TAQ_u \\ Q_u^TAQ_k & Q_u^TAQ_u \end{bmatrix} \equiv \begin{bmatrix} T_k & T_{uk} \\ T_{ku} & T_u \end{bmatrix}$$

When $k = 1$, T_k is just called the Rayleigh quotient. So far $k > 1$, T_k is called a generalization of the Rayleigh quotient.

Definition 2

- *The* Rayleigh-Ritz procedure *is to approximate the eigenvalues of A by the eigenvalues of $T_k = Q_k^TAQ_k$.*
- *These approximations are called the* Ritz values.
- *Let $T_k = V\Lambda V^T$ be the eigendecomposition of T_k. The corresponding eigenvector approximations are the columns of Q_kV and are called* Ritz vectors.

The Ritz values and Ritz vectors are considered *optimal* approximations to the eigenvalues and eigenvectors of A. This is justified by the following theorem.

Theorem 7 *The minimum of $\|AQ_k - Q_kR\|_2$ over all k-by-k symmetric matrices R is attained by $R = T_k$, in which case, $\|AQ_k - Q_kT_k\|_2 = \|T_{ku}\|_2$.*

PROOF: Let $R = T_k + Z$, to proof the theorem, we just want to show that $\|AQ_k - Q_kR\|_2$ is minimized when $Z = 0$. This is shown by the following sequence of derivation:

$$\begin{aligned}
\|AQ_k - Q_kR\|_2^2 &= \lambda_{\max}\left[(AQ_k - Q_kR)^T(AQ_k - Q_kR)\right] \\
&= \lambda_{\max}\left[(AQ_k - Q_k(T_k + Z))^T(AQ_k - Q_k(T_k + Z))\right] \\
&= \lambda_{\max}\Big[(AQ_k - Q_kT_k)^T(AQ_k - Q_kT_k) - ((AQ_k - Q_kT_k)^T(Q_kZ) \\
&\qquad -(Q_kZ)^T(AQ_k - Q_kT_k) + (Q_kZ)^T(Q_kZ)\Big] \\
&= \lambda_{\max}\Big[(AQ_k - Q_kT_k)^T(AQ_k - Q_kT_k) - (Q_k^TAQ_k - T_k)Z \\
&\qquad -Z^T(Q_k^TAQ_k - T_k) + Z^TZ\Big] \\
&= \lambda_{\max}\left[(AQ_k - Q_kT_k)^T(AQ_k - Q_kT_k) + Z^TZ\right] \\
&\geq \lambda_{\max}\left[(AQ_k - Q_kT_k)^T(AQ_k - Q_kT_k)\right] \\
&= \|AQ_k - Q_kT_k\|_2^2
\end{aligned}$$

Furthermore, it is easy to compute the minimum value

$$\|AQ_k - Q_kT_k\|_2 = \|(Q_kT_k + Q_uT_{ku}) - Q_kT_k\|_2 = \|Q_uT_{ku}\|_2 = \|T_{ku}\|_2.$$

∎

Corollary 2 *Let $T_k = V\Lambda V^T$ be the eigendecomposition of T_k. The minimum of $\|AP_k - P_kD\|$ over all n-by-k orthogonal matrices P_k where $\mathrm{span}(P_k) = \mathrm{span}(Q_k)$ and over all diagonal D is also $\|T_{ku}\|_2$ and is attained by $P_k = Q_kV$ and $D = \Lambda$.*

PROOF: If we replace Q_k with Q_kU in the above proof, where U is another orthogonal matrix, then the columns of Q_k and Q_kU span the same space, and

$$\|AQ_k - Q_kR\|_2 = \|AQ_kU - Q_kRU\|_2 = \|A(Q_kU) - (Q_kU)(U^TRU)\|_2.$$

These quantities are still minimized when $R = T_k$, and by choosing $U = V$ so that U^TT_kU is diagonal. ∎

3. The Lanczos algorithm for finding a few eigenpairs of a symmetric matrix A combines the Lanczos process for building a Krylov subspace with the Raleigh-Ritz procedure. First, let us recall that the Lanczos process will generate an orthonormal basis of a Krylov subspace:

$$\mathcal{K}_k(A, v) \stackrel{\text{def}}{=} \mathrm{span}\{v, Av, \ldots, A^{k-1}v\} = \mathrm{span}\{q_1, q_2, \ldots, q_k\},$$

and yield a fundamental relation

$$AQ_k = Q_kT_k + f_ke_k^T, \quad f_k = \beta_kq_{k+1} \tag{15}$$

where $T_k = Q_k^TAQ_k = \mathrm{tridiag}(\beta_j, \alpha_j, \beta_{j+1})$. Let μ be an eigenvalue of T_k and y be a corresponding eigenvector y, i.e.,

$$T_ky = \mu y, \quad \|y\|_2 = 1.$$

Apply y to the right of (15) to get

$$A(Q_ky) = Q_kT_ky + f_k(e_k^Ty) = \mu(Q_ky) + f_k(e_k^Ty).$$

$\{\mu\}$ are *Ritz values*, and $\{Q_ky\}$ are *Ritz vectors*.

Convergence test:

- If $f_k(e_k^T y) = 0$ for some k, then the associated Ritz value μ is an eigenvalue of A with the corresponding eigenvector $Q_k y$.
- In general, it is unlikely that $f_k(e_k^T y) = 0$, but we hope that the residual norm $\|f_k(e_k^T y)\|_2$ may be small; and when this happens we expect that μ is going to be a good approximate to A's eigenvalue. Indeed, we have

  Lemma 3 *Let H be (real) symmetric, and $Hz - \mu z = r$ and $z \neq 0$. Then*

  $$\min_{\lambda \in \lambda(H)} |\lambda - \mu| \leq \|r\|_2 / \|z\|_2.$$

  PROOF: Let $H = U\Lambda U^T$ be the eigen-decomposition of H. Then $Hz - \mu z = r$ yields

  $$(H - \mu I)z = r \quad \Rightarrow \quad U(\Lambda - \mu I)U^T z = r \quad \Rightarrow \quad (\Lambda - \mu I)(U^T z) = U^T r.$$

  Notice that $\Lambda - \mu I$ is diagonal. Thus

  $$\|r\|_2 = \|U^T r\|_2 = \|(\Lambda - \mu I)(U^T z)\|_2 \geq \min_{\lambda \in \lambda(H)} |\lambda - \mu| \, \|U^T z\|_2 = \min_{\lambda \in \lambda(H)} |\lambda - \mu| \|z\|_2,$$

  as expected. ∎

  The following corollary is a consequence of above Lemma 3.

  Corollary 3 *There is an eigenvalue λ of A such that*

  $$|\lambda - \mu| \leq \|f_k(e_k^T y)\|_2 = |\beta_k| \cdot |e_k^T y|.$$

In summary, we have the following Lanczos algorithm in the simplest form:

LANCZOS ALGORITHM for finding eigenvalues and eigenvectors of $A = A^T$:

1. $q_1 = v/\|v\|_2$, $\beta_0 = 0$; $q_0 = 0$;
2. for $j = 1$ to k, do
3. $\quad w = Aq_j$;
4. $\quad \alpha_j = q_j^T w$;
5. $\quad w = w - \alpha_j q_j - \beta_{j-1} q_{j-1}$;
6. $\quad \beta_j = \|w\|_2$;
7. $\quad$ if $\beta_j = 0$, quit;
8. $\quad q_{j+1} = w/\beta_j$;
9. $\quad$ Compute eigenvalues and eigenvectors of T_j
10. $\quad$ Test for convergence
11. EndDo

Caveat: All the discussion in this lecture is under the assumption of exact arithmetic. In the presence of finite precision arithmetic, the numerical behaviors of the Lanczos algorithm could be significantly different. For example, in finite precision arithmetic, the orthogonality of the computed Lanczos vectors $\{q_j\}$ is lost when j is as small as

10 or 20. The simplest remedy (and also the most expensive one) is to implement the the full reorthogonalization, namely after the step 5, do

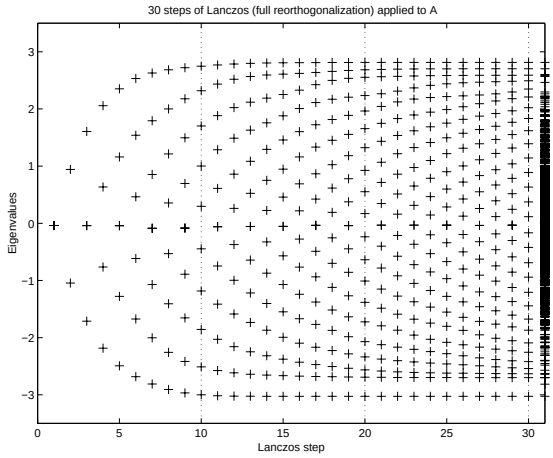
$$w = w - \sum_{i=1}^{j-1} (w^T q_i) q_i.$$

This is called the Lanczos algorithm with full reorthogonalization. (Sometimes, it may be needed to execute *twice*). A more elaborate scheme, necessary when convergence is slow and several eigenvalues are sought, is to use the selective orthogonalization.

Example We illustrate the Lanczos algorithm by a running an example, a 1000-by-1000 diagonal matrix A, most of whose eigenvalues were chosen randomly from a normal Gaussian distribution. To make the plot easy to understand, we have also sorted the diagonal entries of A from largest to smallest, so $\lambda_i(A) = a_{ii}$ with the corresponding eigenvector e_i. There are a few extreme eigenvalues, and the rest cluster near the center of the spectrum. The starting Lanczos vector v has all equal entries.

There is no loss in generality in experimenting with a diagonal matrix, since running the Lanczos algorithm on A with starting vector $q_1 = v/\|v\|_2$ is equivalent to running the Lanczos algorithm on $Q^T A Q$ with starting vector $Q^T q_1$.

The following figure illustrates convergence of the Lanczos algorithm for computing the eigenvalues of A. In this figure, the eigenvalues of each T_k are shown plotted in column k, for $k = 1, 2, 3, \ldots, 30$, with the eigenvalues of A plotted in an extra column at the rightmost column. The column k has k "+"s, one marking each eigenvalues of T_k.



We observe that:

- Extreme eigenvalues, i.e., the largest and smallest ones, converge first, and the interior eigenvalues converge last.
- Convergence is monotonic, with the ith largest (smallest) eigenvalues of T_k increasing (decreasing) to the ith laregst (smallest) eigenvalue of A, provided that the Lanczos algorithm does not stop prematurely with some $\beta_k = 0$.

The best reference to study the observation in theory is the book by B. N. Parlett, "The Symmetric Eigenvalue Problem", reprinted by SIAM, 1998.

12 Eigensolvers based Iterative Subspace Projection II

1. The Arnoldi algorithm for finding a few eigenpairs of a general matrix A combines the Arnoldi process for building a Krylov subspace with the Raleigh-Ritz procedure.

   First, let us recall that the Arnoldi process generates an orthonormal basis of a Krylov subspace:

   $$\mathcal{K}_k(A, v) \stackrel{\text{def}}{=} \text{span}\{v, Av, \ldots, A^{k-1}v\} = \text{span}\{q_1, q_2, \ldots, q_k\},$$

   and yield a fundamental relation

   $$AQ_k = Q_k H_k + h_{k+1,k} q_{k+1} e_k^T. \tag{16}$$

   Any decomposition of the form (16), where H_k is Hessenberg, $Q_k^H Q_k = I$, and $Q_k^H q_{k+1} = 0$, is called an *Arnoldi decomposition of length k*. If H_k is unreduced and $h_{k+1,k} \neq 0$, the decomposition is uniquely determined by the starting vector v (This is commonly called implicit Q Theorem).

   Since $Q_k^H q_{k+1} = 0$, we have

   $$H_k = Q_k^T A Q_k.$$

   Thus, H_k is called the generalized Rayleigh Quotient corresponding to Q_k. Let μ be an eigenvalue of H_k and y be a corresponding eigenvector y, i.e.,

   $$H_k y = \mu y, \quad \|y\|_2 = 1.$$

   Then the corresponding Ritz pair is $(\mu, Q_k y)$. Applying y to the right of (16), the residual vector of $(\mu, Q_k y)$ is given by

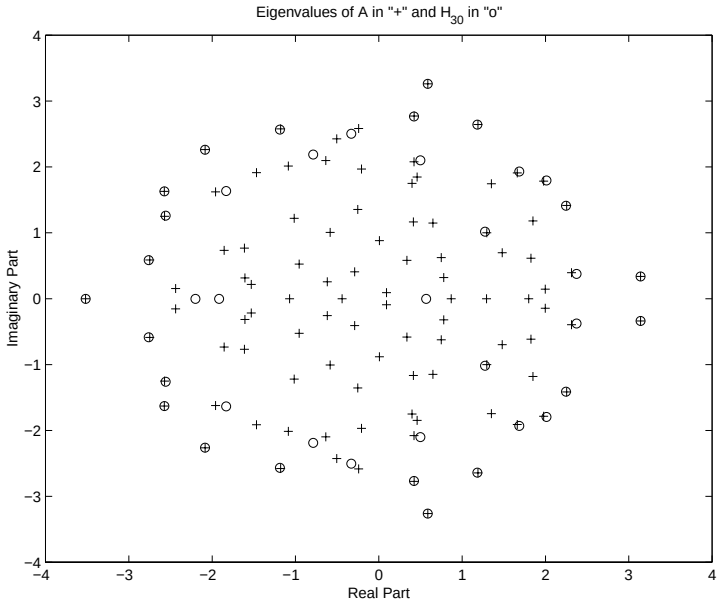
   $$A(Q_k y) - \mu(Q_k y) = h_{k+1,k} q_{k+1}(e_k^T y).$$

   Using the backward error interpretation, we know that $(\mu, Q_k y)$ is an exact eigenpair of $A + E$, where $\|E\|_2 = |h_{k+1,k}| \cdot |e_k^T y|$. This gives us a criterion for accepting the Ritz pair $(\mu, Q_k y)$ as approximate eigenpair of A. [8]

   Arnoldi's Method
   1. Choose a starting vector v
   2. Generate the Arnoldi decomposition of length k by the Arnoldi process
   3. Compute the Ritz pairs and decide which are acceptable
   4. If necessary, increase k and repeat

   Example. We illustrate the above simplest Arnoldi algorithm by a running a 100-by-100 random sparse matrix A with approximately 1000 normally distributed nonzero entries, $A =$ `sprandn(100,100,0.1)`. All entries of the starting vector v are 1. The following figure illustrates typical convergence behavior of the Arnoldi algorithm for computing the eigenvalues. In the figure, "+" are the eigenvalues of matrix A (computed by `eig(full(A))`). and the "o" are the eigenvalues of upper Hessenberg matrix H_{30} (computed by `eig(`H_{30}`)`).

[8] Note that because of non-symmetry of A, we generally do not have the nice forward error estimation as discussed in the Lanczos algorithm for symmetric eigenproblem. But a similar error bound involving the condition number of the corresponding eigenvalue exists.



We observe that *Exterior eigenvalues converge first.* This is the typical convergence phenomenon of the Arnoldi algorithm (in fact, all Krylov subspace based methods). There is a general theory for the convergence analysis of the Arnoldi algorithm.

The Arnoldi algorithm has two nice aspects:

(a) The matrix H_k is already in Hessenberg form, so that we can immediately apply the QR algorithm to find its eigenvalues.

(b) After we increase k, say $k+p$, we only have to orthogonalize p vectors to compute the $(k+p)$th Arnoldi decomposition. The work we have done previously is not thrown away.

Unfortunately, the algorithm has its drawbacks:

- If A is large we cannot increase k indefinitely, since Q_k requires $n \times k$ memory locations to store.
- We have little control over which eigenpairs the algorithm finds. In a given application we will be interested in a certain set of eigenpairs. For example, eigenvalues lying near the imaginary axis. There is nothing in the algorithm to force desired eigenvectors into the subspace or the discard undesired ones.

2. We show how to implicitly restart the algorithm with a new Arnoldi decomposition in which (in exact arithmetic) the unwanted eigenvalues have been purged from H_k. We begin by asking how to cast an undesired eigenvalue out of an unreduced Hessenberg matrix H. Let μ be an eigenvalue of H, then by the QR decomposition of $H - \mu I$, we determine an orthogonal matrix U such that

$$R = U^H(H - \mu I).$$

is upper triangular. Now $H - \mu I$ is singular, and hence R must have a zero on its diagonal. Because H is unreduced, that zero cannot occur at a diagonal position other

than the last. Consequently, the last row of R is zero and so is the last row of RU. Hence

$$\hat{H} = RU + \mu I = \begin{pmatrix} \hat{H}_* & \hat{h} \\ 0 & \mu \end{pmatrix} = U^H H U.$$

In other words, the eigenvalue μ has been found exactly, and the similarity transformation of H with U has deflated the eigenvalue μ.

In the presence of rounding error we cannot expect the last element of R to be nonzero. This means that the matrix $\hat{H}$ will have the form

$$\begin{pmatrix} \hat{H}_* & \hat{h} \\ \hat{h}_{k,k-1} e_{k-1}^T & \hat{\mu} \end{pmatrix}$$

We are now going to show how to use the transformation U to reduce the size of the Arnoldi decomposition. The first step is to note that $U = P_{12} P_{23} \cdots P_{n-1,n}$, where $P_{i,i+1}$ is a rotation in the $(i, i+1)$-plane. Consequently, U is Hessenberg and can be partitioned in the form

$$U = \begin{pmatrix} U_* & u \\ u_{k,k-1} e_{k-1}^T & u_{k,k} \end{pmatrix}.$$

From the relation

$$AQ_k = Q_k H_k + h_{k+1,k} q_{k+1} e_k^T,$$

we have

$$AQ_k U = Q_k U (U^T H_k U) + h_{k+1,k} q_{k+1} e_k^T U.$$

If we partition

$$\hat{Q}_k = Q_k U = \begin{pmatrix} \hat{Q}_{k-1} & \hat{q}_k \end{pmatrix}$$

Then

$$A \begin{pmatrix} \hat{Q}_{k-1} & \hat{q}_k \end{pmatrix} = \begin{pmatrix} \hat{Q}_{k-1} & \hat{q}_k \end{pmatrix} \begin{pmatrix} \hat{H}_* & \hat{h} \\ \hat{h}_{k,k-1} e_{k-1}^T & \hat{\mu} \end{pmatrix} + h_{k+1,k} q_{k+1} \begin{pmatrix} q_{k,k-1} e_{k-1}^T & q_{k,k} \end{pmatrix}.$$

Computing the first $k-1$ columns of this partition, we get

$$A\hat{Q}_{k-1} = \hat{Q}_{k-1} \hat{H}_* + \left(\hat{h}_{k,k-1} \hat{q}_k + h_{k+1,k} q_{k,k-1} q_{k+1} \right) e_{k-1}^T \tag{17}$$

The matrix $\hat{H}_*$ is Hessenberg. The vector $\hat{h}_{k,k-1}\hat{q}_k + h_{k+1,k} q_{k,k-1} q_{k+1}$ is orthogonal to the columns of $\hat{Q}_{k-1}$. Hence (17) is an Arnoldi decomposition of length $k-1$. With exact computation, the eigenvalue μ is presented in $\hat{H}_*$.

The process may be repeated to remove other unwanted values from H. If the matrix is real, complex eigenvalues can be removed two at a time via an implicit double shift. The key observation here is that Q is zero below its second subdiagonal element, so that truncationing the last two columns and adjusting the residual results in an Arnoldi decomposition. Once un-desired eigenvalues have been removed from H, the Arnoldi decomposition may be expanded to one of order k and the process repeated.

3. There are two important additions to the algorithm that are beyond the scope of this lecture.

- First, as Ritz pairs converge they can be locked into the decomposition. The procedure amounts to computing an Arnoldi decomposition of the form

$$A \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{pmatrix} + h_{k+1,k} q_{k+1} e_k^T$$

  When this is done, one can work with the part of decomposition corresponding to U_2, thus saving multiplications by A. (However, care must be taken to maintain orthogonality to the columns of Q_1.)

- The second addition concerns unwanted Ritz pairs. The restarting procedure will tend to purge the unwanted eigenvalues from H. But the columns of U may have significant components along the eigenvectors corresponding to the purged pairs, which will then reappear as the decomposition is expanded. If certain pair are too persistent, it is best to keep them around by computing a block diagonal decomposition of the form

$$A \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} H_{11} & 0 \\ 0 & H_{22} \end{pmatrix} + \eta q_{k+1} e_k^T,$$

  where H_{11} contains the unwanted eigenvalues. This insures that U_2 has negligible components along the unwanted eigenvectors. We can then compute an Arnoldi decomposition by reorthognalizing the relation

$$AQ_2 = H_{22} Q_2 + \eta q_{k+1} e_m^T,$$

  where m is the order of H_2.

4. Further reading

   - The best reference on symmetric Lanczos method, theory and practice, is the book by B. N. Parlett, The Symmetric Eigenvalue Problem, reprinted with some revision by SIAM Press, 1998.
   - MATLAB function `eigs` is an implementation of the implicitly restarted Arnoldi algorithm for finding a few eigenpairs. The starting reference is D. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, SIMAX, Vol. 13, pp.357–385, 1992.
   - In the recent years, eigensolvers with preconditioning techniques are studied extensively. The best starting paper is [G.L.G. Sleipen and H.A. van der Vorst, A Jacobi-Davidson iteration method for linear eigenvalue problems, SIMAX, 17:401–425,1996.]
   - A complete treatment of eigenvalue problems is [Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra. A. Ruhe and H. van der Vorst eds. SIAM, 2000].

13 Poisson Solvers I

1. Poisson's equation in one dimension.

$$-\frac{d^2v(x)}{dx^2} = f(x), \qquad 0 < x < 1 \tag{18}$$

with Dirichlet boundary conditions:

$$v(0) = v(1) = 0, \tag{19}$$

where $f(x)$ is a given function and $v(x)$ is the unknown function to be computed.

This boundary value problem can be viewed as modeling the displacement of an elastic bar or cord in continuum mechanics, the temperature distribution in a heat conducting bar and others.

2. Let us *discretize* the problem by trying to compute an approximate solution $N+2$ evenly spaced point x_i between 0 and 1:

$$0 = x_0 < x_1 < x_2 < \cdots < x_N < x_{N+1} = 1$$

and

$$x_i = x_0 + ih = ih, \qquad h = \frac{1}{N+1}.$$

Denote $v_i = v(x_i)$ and $f_i = f(x_i)$.

Using the three-point centered difference approximation, at $x = x_i$, we have

$$-\frac{d^2v(x)}{dx^2} = \frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} + \tau_i,$$

where the truncation error $\tau_i = O(h^2)$ (assuming $v(x)$ is smooth enough). Therefore at $x = x_i$, $0 < i < N+1$, we have

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i + h^2 \tau_i$$

and $v_0 = v_{N+1} = 0$.

In matrix notation, let $v = [v_1, v_2, \ldots, v_N]^T$ and $\bar{\tau} = [\tau_1, \tau_2, \ldots, \tau_N]^T$. Then we have

$$T_N\, v = h^2 f + h^2 \bar{\tau} \tag{20}$$

where

$$T_N = \text{tridiag}(-1, 2, -1).$$

To solve this equation, let us ignore $\bar{\tau}$, since it is expected to be small compared to f, then we have

$$T_N\, \hat{v} = h^2 f, \tag{21}$$

where $\hat{v}$ is an approximation of v.

3. The tridiagonal matrix T_N has the following special properties

   - The eigenvalues of T_N are $\lambda_j = 2(1 - \cos\frac{\pi j}{N+1})$.

- The eigenvectors are z_j, where the kth entry $z_j(k) = \sqrt{\frac{2}{N+1}}\sin(\frac{\pi kj}{N+1})$ and $\|z_j\|_2 = 1$.

Thus, we have the eigendecomposition

$$T_N = Z_N \Lambda_N Z_N^T,$$

where $Z_N = [z_1, z_2, \ldots, z_N]$ (Z is orthogonal) and $\Lambda_N = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$. The largest eigenvalue of T_N is λ_N and for large N,

$$\lambda_N = 2(1 - \cos\frac{N\pi}{N+1}) \approx 4$$

and the smallest eigenvalue is λ_1 and for large N,

$$\lambda_1 = 2(1 - \cos\frac{\pi}{N+1}) \approx 2\left(1 - \left(1 - \frac{\pi^2}{2(N+1)^2}\right)\right) = \frac{\pi^2}{(N+1)^2}.$$

Therefore, T_N is symmetric positive definite. The condition number of T_N is

$$\mathrm{cond}(T_N) = \|T_N\|_2\|T_N^{-1}\|_2 = \frac{\lambda_N}{\lambda_1} \approx \frac{4(N+1)^2}{\pi^2} = \mathcal{O}(h^{-2}).$$

4. Now we can bound the error $v - \hat{v}$, subtracting equation (21) from equation (20), we have

$$v - \hat{v} = h^2 T_N^{-1}\,\bar{\tau}.$$

By taking norm, we have

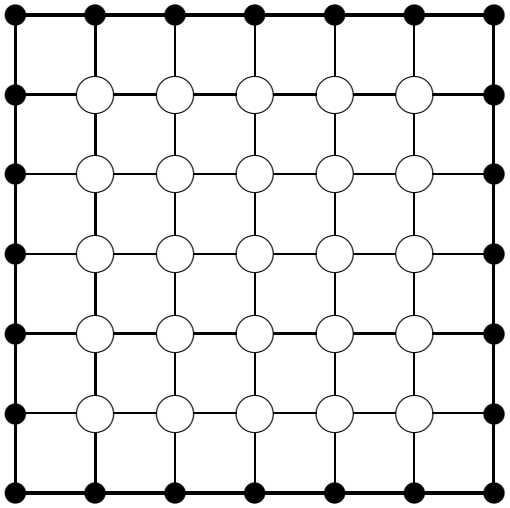
$$\|v - \hat{v}\|_2 \le h^2\|T_N^{-1}\|_2\|\bar{\tau}\|_2 \approx h^2\frac{(N+1)^2}{\pi^2}\|\bar{\tau}\|_2 = O(\|\bar{\tau}\|_2) = O(h^2),$$

assuming that $v(x)$ is sufficient smooth (the required derivatives are bounded).

5. The Poisson's equation in 2 dimensions.

$$\begin{aligned} -\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2} &= f(x,y) \quad \text{for } (x,y) \in \Omega, \\ v(x,y) &= \phi(x,y) \quad \text{for } (x,y) \in \partial\Omega, \end{aligned}$$

where $\Omega = (0,1) \times (0,1)$, the unit square and $\partial\Omega$ is its boundary. To discretize the differential equation, the domain Ω is covered with a grid of step size $h = 1/(N+1)$ as follows.



This is an example grid with $N = 5$. The values $v(x,y)$ at the boundary grid points ● is given by $\phi(x,y)$, and the values $v(x,y)$ at interior grid points ◯ are to be sought.

Each grid point (x_i, y_j) have the representation

$$x_i = ih \quad \text{and} \quad y_j = jh \quad \text{for } i,\, j = 0, 1, \ldots, N+1.$$

Those points with one of i and j being $i = 0$ or $N+1$ are the *boundary grid points*; all other points are the *interior grid points*. We seek approximations to $v(x_i, y_j)$ for all the interior grid points. Write

$$v_{ij} = v(x_i, y_j), \quad f_{ij} = f(x_i, y_j), \quad \text{and} \qquad \phi_{ij} = \phi(x_i, y_j).$$

To this end, we do approximately at each interior grid point:

$$\begin{aligned} -\frac{\partial^2 v}{\partial x^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i-1\,j} + 2v_{ij} - v_{i+1\,j}}{h^2}, \\ -\frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i\,j-1} + 2v_{ij} - v_{i\,j+1}}{h^2}. \end{aligned}$$

Adding these approximations we have

$$-\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} = \frac{-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1}}{h^2} + \tau_{ij}$$

where τ_{ij} is a truncation error. By Taylor expansion, it is easy to show that it is at the order of h^2, $O(h^2)$. Ignoring the truncation errors, we arrive at the linear equations in the unknowns v_{ij},

$$-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1} = h^2 f_{ij}, \tag{22}$$

for $1 \le i, j \le N$. The left-hand side of which is 4 times the v at the point subtracting the v at the four neighbor points. This is called *5-point centered difference* or *5-point stencil*.

Notice that

$$v_{0j} = \phi_{0j}, \quad v_{0\,N+1} = \phi_{0\,N+1}, \quad v_{i0} = \phi_{i0}, \quad v_{i\,N+1} = \phi_{i\,N+1}$$

are known and the unknowns are for $0 < i, j < N+1$; so there are N^2 of them. So care should be taken in putting the equations (22) into the familiar form $Ax = b$ of a linear system for the grid points that are neighbors of boundary grid points. We shall now do so.

Collect all v_{ij} to form an $N \times N$ matrix V whose (i,j)th entry is v_{ij}: $V = (v_{ij})$. Define an $N \times N$ matrix $\widetilde{F}$ by

$$h^2(\widetilde{F})_{ij} = \begin{cases} h^2 f_{ij}, & \text{for } 2 \le i, j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1}, & \text{for } 2 \le i \le N-1 \text{ and } j = 1, \\ h^2 f_{ij} + \phi_{i\,j+1}, & \text{for } 2 \le i \le N-1 \text{ and } j = N, \\ h^2 f_{ij} + \phi_{i-1\,j}, & \text{for } i = 1 \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i+1\,j}, & \text{for } i = N \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i-1\,j}, & \text{for } (i,j) = (1,1), \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,1), \\ h^2 f_{ij} + \phi_{i-1\,j} + \phi_{i\,j+1}, & \text{for } (i,j) = (1,N), \\ h^2 f_{ij} + \phi_{i\,j+1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,N). \end{cases}$$

It can be verified that the (22) becomes

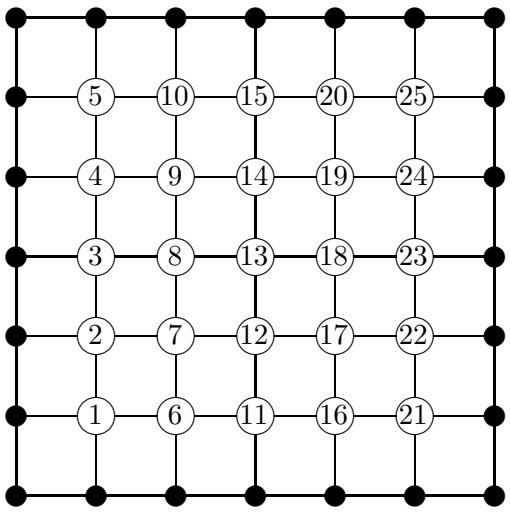
$$T_N \cdot V + V \cdot T_N = h^2 \widetilde{F}, \tag{23}$$

where $T_N = \text{tridiag}(-1, 2, -1)$.

6. Lexicographic (Natural) Ordering: system (23) is still not in the familiar form $Ax = b$ because all the unknowns are compactly stored into a matrix. To reorganize equations (22) in a way that leads to the $Ax = b$ form, we need to arrange v_{ij} into a column vector. A natural way would be arranging one column of V on top of another, i.e., defining a N^2-dimensional vector v as (in MATLAB-like notation)

$$v = [V(:,1); V(:,2); \ldots; V(:,N)] \equiv \text{vec}(V).$$

Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Define also N^2-dimensional vector $\widetilde{f}$ from the matrix $\widetilde{F}$ analogously. The system (23) becomes

$$Av = h^2 \widetilde{f}, \tag{24}$$

where

$$A = \begin{pmatrix} T_N + 2I_N & -I_N & & & & \\ -I_N & T_N + 2I_N & -I_N & & & \\ & \ddots & \ddots & \ddots & & \\ & & -I_N & T_N + 2I_N & -I_N & \\ & & & -I_N & T_N + 2I_N & \end{pmatrix} \equiv T_{N \times N}.$$

In fact, A is the Kronecker products of T_N and I_N:

$$A = I_N \otimes T_N + T_N \otimes I_N \equiv T_{N \times N}.$$

Definition:

(a) Let $A = (a_{ij})$ be $m \times n$ and $B = (b_{ij})$ be $p \times q$, then the *Kronecker Product* of A and B are defined as

$$A \otimes B = (a_{ij}B) = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix},$$

an $(mp) \times (nq)$ matrix.

(b) Let X be m by n. Then $\mathrm{vec}(X)$ is defined to be a column vector of size $m \cdot n$ made of the columns of X stacked atop one another from left to right.

Basic properties of Kronecker product:

(a) Let A be m-by-m and B be n-by-n, and X and C are $m \times n$. Then
$\mathrm{vec}(AX) = (I_n \otimes A) \cdot vec(X)$
$\mathrm{vec}(XB) = (B^T \otimes I_m) \cdot vec(X)$

(b) Assume AC and BD are well defined, then
$(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$

(c) If A and B are invertible, $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

(d) $(A \otimes B)^T = A^T \otimes B^T$

Proposition 5 *Let $T_N = Z_N \Lambda_N Z_N^T$ be the eigendecomposition of the tridiagonal matrix T_N. Then the eigendecomposition of $T_{N\times N}$ is given by*

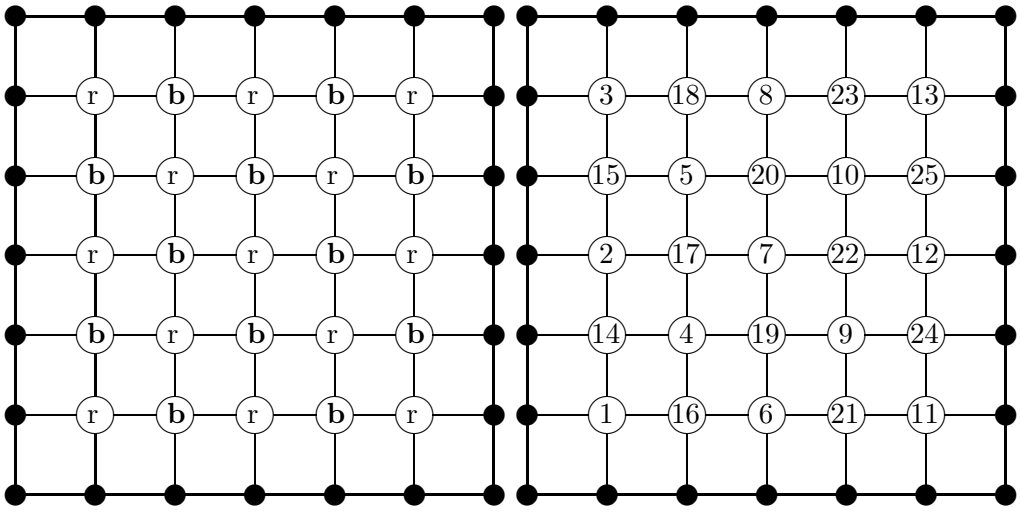
$$T_{N\times N} = (Z_N \otimes Z_N)(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N)(Z_N \otimes Z_N)^T.$$

From the proposition, it shows that eigenvalues λ_{ij} of the Poisson matrix $T_{N\times N}$ are given by

$$\lambda_{i\,j} \stackrel{\text{def}}{=} \lambda_i + \lambda_j = 2(2 - \cos i\pi h - \cos j\pi h) \tag{25}$$

$i, j = 1, 2, \ldots, N$, where λ_i and λ_j are the eigenvalues of T_N. Note that $h = 1/(N+1)$.

7. Red-Black Ordering: first color all nodes by either *red* or *black* in such a way that no neighbor nodes share the same color; and then enumerate all nodes with one color and then all nodes with the other. Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Let $v_{\rm rb}$ and $\widetilde{f}_{\rm rb}$ be the N^2-dimensional vectors obtained from V and $\widetilde{F}$ with this red-black ordering. The system (23) becomes

$$A_{\rm rb} v_{\rm rb} = h^2 \widetilde{f}_{\rm rb}, \quad A_{\rm rb} = \begin{pmatrix} D_r & B \\ B^T & D_b \end{pmatrix}, \tag{26}$$

both D_r and D_b are diagonal matrices with all diagonal entries being 4. B is a sparse matrix with nonzero entries -1 (the details of the structure of B is not important for us now).

Notice that $A_{\rm rb}$ is consistently ordered and has eigenvalues given by (25).

8. Poisson's equations in 3 Dimensions.

$$\begin{aligned} -\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2} - \frac{\partial^2 v}{\partial z^2} &= f(x,y,z) \quad \text{for } (x,y,z) \in \Omega, \\ v(x,y,z) &= \phi(x,y,z) \quad \text{for } (x,y,z) \in \partial\Omega, \end{aligned}$$

where $\Omega = (0,1) \times (0,1) \times (0,1)$, the unit cubic and $\partial\Omega$ is its boundary

Using a 7-point centered finite difference on a cubic grid of step size $h = 1/(N+1)$. Then with natural ordering, it leads to the linear system of equations $Ax = b$, where

$$A = T_{N\times N\times N} = T_N \otimes I_N \otimes I_N + I_N \otimes T_N \otimes I_N + I_N \otimes I_N \otimes T_N$$

It can be shown that A's eigenvalues are all possible triple sum of the eigenvalues of T_N and the eigenvector matrix is $Z_N \otimes Z_N \otimes Z_N$.

9. Poisson's equation in higher dimensions is represented analogously.

14 Poisson Solvers II

1. Block cyclic reduction (BCR) is a fast method for the Poisson model problem. We will describe a simple but numerically unstable version of the BCR algorithm, then point out a reference for a stable implementation.

2. Write the 2D Poisson's model problem as

$$\begin{bmatrix} A & -I & & \\ -I & A & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix},$$

where $A = T_N + 2I$ and I is an $N \times N$ identity matrix. x_i and b_i are N-vectors.

For simplicity we assume that N is **odd**. We use block Gaussian elimination to combine three consecutive sets of equations

$$\begin{array}{rl} & [\ -x_{j-2} + Ax_{j-1} - x_j \qquad\qquad\qquad = b_{j-1}\ ], \\ +A & [\ \qquad -x_{j-1} + Ax_j - x_{j+1} \qquad = b_j\ ], \\ + & [\ \qquad\qquad -x_j + Ax_{j+1} - x_{j+2} = b_{j+1}\ ], \end{array}$$

Thus eliminating x_{j-1} and x_{j+1}

$$-x_{j-2} + (A^2 - 2I)x_j - x_{j+2} = b_{j-1} + Ab_j + b_{j+1}.$$

Doing this for every set of three consecutive equations yields two sets of equations:

- one for the x_j with j **even**

$$\begin{bmatrix} A^{(1)} & -I & & \\ -I & A^{(1)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 + Ab_2 + b_3 \\ b_3 + Ab_4 + b_5 \\ \vdots \\ b_{N-2} + Ab_{N-1} + b_N \end{bmatrix}, \tag{27}$$

  where

$$A^{(1)} = A^2 - 2I \equiv \left(A^{(0)}\right)^2 - 2I,$$

- one set of equations for the x_j with j **odd**,

$$\begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 + x_2 \\ b_3 + x_2 + x_4 \\ \vdots \\ b_N + x_{N-1} \end{bmatrix}. \tag{28}$$

  This set of equations can be solved directly after solving the equation (27) for x_j with j even.

Note that equation (27) has the same form as the original problem, so we may repeat this process recursively. For example, at the next step we get

$$\begin{bmatrix} A^{(2)} & -I & & \\ -I & A^{(2)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(2)} \end{bmatrix} \begin{bmatrix} x_4 \\ x_8 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}, \tag{29}$$

where

$$A^{(2)} = \left(A^{(1)}\right)^2 - 2I,$$

and

$$\begin{bmatrix} A^{(1)} & & & \\ & A^{(1)} & & \\ & & \ddots & \\ & & & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_6 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}. \tag{30}$$

We repeat this until only one equation is left, which we solve another way. Therefore, the algorithm consists of three steps:

(a) Reduction (see equations (27) and (29))

(b) Solve $A^{(k)}x^{(k)} = b^{(k)}$ where assuming $N = 2^{k+1} - 1$.

(c) Back-solver (see equations (28) and (30))

3. It can be shown that the cost of this algorithm is $O(N^2 \log_2 N)$.

4. The simple BCR approach has two drawbacks:

   (a) It is numerically unstable because $A^{(r)}$ grows quickly:

   $$\|A^{(r)}\| \sim \|A^{(r-1)}\| \approx 4^{2^r},$$

   so in computing $b_j^{(r+1)}$, the $b_{2j\pm1}^{(r)}$ are lost in roundoff.

   (b) $A^{(r)}$ has bandwidth $2^r + 1$ if $A^{(0)} = A$ is tridiagonal, so it can be dense and thus more expensive to multiply or solve.

5. A numerically stable and efficient algorithm can be found in

   - B. Buzbee, G. Golub and C. Nielson , On the direct method for solving Poisson's equation, SIAM J. Numer. Anal. Vol. 7, pp.627–656, 1970.

15 Poisson Solvers III

1. Solving the Poisson's equation using eigendecomposition/FFT.

   We recall the formulation of the 2D Poisson's equation

   $$T_N \cdot V + V \cdot T_N = h^2 F.$$

   Let $T_N = Z\Lambda Z^T = Z \cdot \mathrm{diag}(\lambda_j) \cdot Z^T$ be the eigendecomposition of T_N. Then the last equation becomes

   $$\Lambda \cdot \widetilde{V} + \widetilde{V} \cdot \Lambda = h^2 \widetilde{F}.$$

   where $\widetilde{V} = Z^T V Z$ and $\widetilde{F} = Z^T F Z$. It is easy to see that the (j,k) entry of this equation is

   $$\lambda_j \tilde{v}_{jk} + \tilde{v}_{jk} \lambda_k = h^2 \tilde{f}_{jk},$$

   which can be solved for $\tilde{v}_{jk}$ to get

   $$\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}.$$

   This yields the first version of our algorithm:

   (a) Compute $\widetilde{F} = Z^T F Z$

   (b) For all j and k, compute $\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}$

   (c) Compute $V = Z \widetilde{V} Z^T$

   The cost of step (b) is $3N^2$, and the cost of steps (a) and (b) is four matrix-matrix multiplications by Z and $Z^T (= Z)$, which is $8N^3$ using a conventional algorithm. In the following, we show how multiplication by Z is essentially the same as computing a *discrete Fourier transform*, which can be done in $O(N^2 \log_2 N)$ operation. Note that if $N = 2^{20} = 1,048,576$, then $\log_2 N = 20$.

2. The Discrete Fourier Transform (DFT) of an N-vector a is the vector

   $$b = \Phi a,$$

   where $\Phi = (\phi_{jk})$ is N-by-N matrix defined as follows:

   $$\phi_{jk} = \omega^{jk}, \quad \text{for} \quad j,k = 0,1,\ldots,N-1$$

   where

   $$\omega = \exp\left(\frac{-2\pi i}{N}\right) = \cos\frac{2\pi}{N} - i \sin\frac{2\pi}{N},$$

   a principal Nth root of unity.

   The Inverse Discrete Fourier Transform (IDFT) of b is the vector

   $$a = \Phi^{-1} b.$$

3. Properties:

(a) $\frac{1}{\sqrt{N}}\Phi$ is a symmetric unitary matrix, i.e.,

$$\Phi^{-1} = \frac{1}{N}\Phi^* = \frac{1}{N}\bar{\Phi}.$$

(b) If $b = \Phi a$ and $a = [a_0, a_1, \ldots, a_{N-1}]$, then the kth component of b is

$$b_k = \sum_{j=0}^{N-1} a_j \omega^{kj}.$$

This can be view as the value of the polynomial $a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$, i.e.,

$$b_k = (\Phi a)_k = a(\omega^k).$$

In other words,

the DFT is polynomial evaluation at the points $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

Conversely, the IDFT is polynomial interpolation, producing the coefficients of a polynomial given its values at $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

(c) Both the DFT and IDFT are just matrix-vector multiplications and can be straightforwardly implemented in $2N^2$ operations.

Fast Fourier Transform (FFT) is a fast way to multiply by Φ. Instead of $2N^2$, it will require only about $\frac{3N}{2} \cdot \log_2 N$ operations.

(d) The DFT and IDFT are closely related the Fourier transform and its inverse in continuous case.

4. We have seen that to solve the discrete Poisson's model problem by the eigendecomposition of T_N requires the ability to multiply by the N-by-N matrix Z, whose the (j, k) entry is

$$z_{jk} = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(j+1)}{N+1}\right),$$

where for the convenient of notation, we number rows and columns from 0 to $N-1$ in this section.

Now consider the $(2N+2)$-by-$(2N+2)$ DFT matrix Φ, whose j, k entry is

$$\exp\left(\frac{-2\pi ijk}{2N+2}\right) = \exp\left(\frac{-\pi ijk}{N+1}\right) = \cos\frac{\pi jk}{N+1} - i\sin\frac{\pi jk}{N+1}.$$

Thus the N-by-N matrix Z consists of $-\sqrt{\frac{2}{N+1}}$ times the imaginary part of the second through $(N+1)$st rows and columns of Φ. So if we can multiply efficiently by Φ using the FFT, then we can multiply efficiently by Z. In practice, we can modify the FFT to multiply by Z directly. This is called the *Fast Sine Transform*.

5. Convolutions

Definition 3 *If* $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ *and* $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ *are* $2N$*-vectors, then the discrete* convolution *of* a *and* b *is defined as*

$$a * b \equiv c = [c_0, \ldots, c_{2N-1}]^T,$$

where $c_k = \sum_{j=0}^{k} a_j b_{k-j}$.

To illustrate the use of the discrete convolution, consider the polynomial multiplication. Let $a(x) = \sum_{k=0}^{N-1} a_k x^k$ and $b(x) = \sum_{k=0}^{N-1} b_k x^k$ be degree-$(N-1)$ polynomials. Then their product

$$c(x) \equiv a(x) \cdot b(x) = \sum_{k=0}^{2N-1} c_k x^k,$$

where the coefficients c_k are given by the discrete convolution.

One purpose of the Fourier transform is to convert the convolution into multiplication.

Theorem 8 *Let* $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ *and* $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ *be vectors of dimension* $2N$*, and let* $c = a * b = [c_0, \ldots, c_{2N-1}]^T$*. Then*

$$(\Phi c)_k = (\Phi a)_k \cdot (\Phi b)_k.$$

PROOF. Recall the important observation (b) in the above item 2, if $\widetilde{a} = \Phi a$, then the kth entries of $\widetilde{a}$ is $\widetilde{a}_k = \sum_{j=0}^{2N-1} a_j \omega^{kj}$, the value of the polynomial $a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$, i.e.,

$$\widetilde{a}_k = a(\omega^k).$$

Similarly, $\widetilde{b} = \Phi b$ means that $\widetilde{b}_k = \sum^{N-1} b_j \omega^{kj} = b(\omega^k)$, and $\widetilde{c} = \Phi c$ means that $\widetilde{c}_k = \sum^{2N-1} c_j \omega^{kj} = c(\omega^k)$. Therefore

$$(\Phi a)_k \cdot (\Phi b)_k = \widetilde{a}_k \cdot \widetilde{b}_k = a(\omega^k) \cdot b(\omega^k) = c(\omega^k) = \widetilde{c}_k = (\Phi c)_k.$$

6. We now derive the FFT via its interpretation as polynomial evaluation. The goal is to evaluate $a(x) = \sum_{k=0}^{N-1} a_k x^k$ at $x = \omega^j$ for $0 \le j \le N-1$. For simplicity we will assume $N = 2^m$. Now write

$$\begin{aligned} a(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots) \\ &= a_{\text{even}}(x^2) + x a_{\text{odd}}(x^2). \end{aligned}$$

Thus, the evaluation of $a(x)$ is **divided** into evaluating two polynomials a_{even} and a_{odd} of degree $\frac{N}{2} - 1$ at $(\omega^j)^2$, $0 \le j \le N-1$.

Moreover, because of

$$\omega^{2j} = \omega^{2(j+\frac{N}{2})},$$

there are really just $\frac{N}{2}$ points ω^{2j} for $0 \le j \le \frac{N}{2} - 1$.

In other words, evaluating a polynomial of degree $N - 1 = 2^m - 1$ at all N points ω^j $(0 \le j \le N-1)$ is the same as evaluating two polynomials of degree $\frac{N}{2} - 1$ at all $\frac{N}{2}$ points[9], and then combining the results with N multiplications and additions. This can be done *recursively*!

[9] those are the $\frac{N}{2}$th roots of the unity.

FFT ALGORITHM
function $\widetilde{a} = \mathrm{FFT}(a, N)$
    if $N = 1$
        return $\widetilde{a} = a$
    else
        $\widetilde{a}_{\mathrm{even}} = \mathrm{FFT}(a_{\mathrm{even}}, N/2)$
        $\widetilde{a}_{\mathrm{odd}} = \mathrm{FFT}(a_{\mathrm{odd}}, N/2)$
        $\omega = e^{-2\pi i/N}$
        $z = [\omega^0, \omega^1, \ldots \omega^{N/2-1}]$
        return $\widetilde{a} = [\widetilde{a}_{\mathrm{even}} + z. * \widetilde{a}_{\mathrm{odd}},\ \widetilde{a}_{\mathrm{even}} - z. * \widetilde{a}_{\mathrm{odd}}]$
    end if

where $.*$ means componentwise multiplication of arrays (as in MATLAB), and have used the fact that $\omega^{j+N/2} = -\omega^j$.

7. Matlab script

```
function y = ffttx(x)
%FFTTX  Textbook Fast Finite Fourier Transform.
%    FFTTX(X) computes the same finite Fourier transform as FFT(X).
%    The code uses a recursive divide and conquer algorithm for
%    even order and matrix-vector multiplication for odd order.
%    If length(X) is m*p where m is odd and p is a power of 2, the
%    computational complexity of this approach is O(m^2)*O(p*log2(p)).

x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0
   % Recursive divide and conquer
   k = (0:n/2-1)';
   w = omega .^ k;
   u = ffttx(x(1:2:n-1));
   v = w.*ffttx(x(2:2:n));
   y = [u+v; u-v];
else
   % The Fourier matrix.
   j = 0:n-1;
   k = j';
   F = omega .^ (k*j);
   y = F*x;
end
```

8. Let the cost of this algorithm be denoted $C(N)$. Then we see that

$$C(N) = 2C\left(\frac{N}{2}\right) + \frac{3N}{2},$$

assuming that the powers if ω are precomputed and stored in tables. To solve this recurrence write

$$\begin{aligned} C(N) &= 2C\left(\frac{N}{2}\right) + \frac{3N}{2} \\ &= 4C\left(\frac{N}{4}\right) + 2 \cdot \frac{2N}{2} \\ &= 8C\left(\frac{N}{8}\right) + 3 \cdot \frac{2N}{2} \\ &= \cdots \\ &= \log_2 N \cdot \frac{3N}{2}. \end{aligned}$$

Note that $C(1) = 0$.

To compute the FFT of each column (or each row) of an N-by-N matrix therefore costs $\log_2 N \cdot \frac{3N}{2} \cdot N$. This is the complexity of the FFT method for solving the 2D Poisson's model problem.