Today

Last time

- Sparse grids

Today

- Review of *some* important topics

Announcements

- Be 10min earlier (at 5.00pm) in the room for the final exam next week
- Double check that all homeworks are entered correctly in brightspace

There are many opportunities for asking questions

Mon, Dec 5	office hour
Thu, Dec 8	office hour
Mon, Dec 12	office hour
Mon, Dec 12	recap of important topics of class
Thu, Dec 15	office hour (all HWs graded by Dec 15; no HW re-grading after Dec 17)
Mon, Dec 19	final exam

- Condition and stability
- Linear systems and linear least-squares problems
- Eigenproblems
- Iterative methods for linear systems and nonlinear systems
- Interpolation
- Quadrature

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation?

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation? $\rightsquigarrow$ change the problem)

Condition of a problem (cont'd)

- Absolute condition number at x is

$$\kappa_{\text{abs}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \leq \delta} \frac{\|f(x) - f(\hat{x})\|}{\|x - \hat{x}\|}$$

- Relative condition number at x is

$$\kappa_{\text{rel}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \leq \delta} \frac{\|f(x) - f(\hat{x})\| / \|f(x)\|}{\|x - \hat{x}\| / \|x\|}$$

- If f is differentiable in x, then

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\|,$$

where $\|f'(x)\|$ is the norm of the Jacobian $f'(x)$ in the operator norm

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\| = 1} \|Ax\|$$

Revisiting stability

An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} \in \mathcal{O}(u)\,,$$

where u is the roundoff unit

- Recall that, loosely speaking, the symbol $\mathcal{O}(u)$ means "on the order of the roundoff unit."
- By allowing $u \to 0$ (which is implied here by the $\mathcal{O}$), we consider an idealization of a computer (in practice, u is fixed). So what we mean is that the error should decrease in proportion to u or faster.

Stability + condition

Suppose a backward stable algorithm is applied to solve a problem $f : X \to Y$ with relative condition number κ. Then, the relative errors satisfy

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \in \mathcal{O}(\kappa(x)u)\,.$$

Condition of solving system of linear equations

Recall that we derived the condition number $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ of a matrix $\boldsymbol{A}$

Problem 1: fix A, consider $b \mapsto A^{-1}b$

$f: \mathbb{R}^n \to \mathbb{R}^n$, $f(b) = A^{-1}b$

$k_{abs} = \|A^{-1}\|$

$k_{rel} = \frac{\|b\|}{\|A^{-1}b\|} \|A^{-1}\| \leq \|A\| \|A^{-1}\| = k(A)$

Problem 2: fix b, $A \mapsto A^{-1}b$

$k_{rel} \leq \|A\| \|A^{-1}\| = k(A)$

Problem 1: Show that $\rho(A) \leq \|A\|$ for any induced matrix norm, where $\rho(A)$ is the spectral radius of A (the largest absolute eigenvalue).

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} \geq \frac{\|Ax_1\|}{\|x_1\|} = \frac{\|\rho(A)x_1\|}{\|x_1\|} = \rho(A)\frac{\|x_1\|}{\|x_1\|}$$

$$= \rho(A)$$

Solving systems of linear equations

Gauss elimination and LU factorization

multiply first row by ℓ_{i1} and subtract

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

$\leftarrow \ell_{21} = \dfrac{a_{21}^{(1)}}{a_{11}^{(1)}}$

$\leftarrow \ell_{31} = \dfrac{a_{31}^{(1)}}{a_{11}^{(1)}}$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} - \ell_{21} a_{11}^{(1)} = a_{21}^{(1)} - \frac{a_{21}^{(1)}}{a_{11}^{(1)}} a_{11}^{(1)} = 0 & a_{22}^{(1)} - \ell_{21} a_{12}^{(1)} & a_{23}^{(1)} - \ell_{21} a_{13}^{(1)} \\ 0 & a_{32}^{(1)} - \ell_{31} a_{12}^{(1)} & a_{33}^{(1)} - \ell_{31} a_{13}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} - \ell_{21} b_1^{(1)} \\ b_3^{(1)} - \ell_{31} b_1^{(1)} \end{bmatrix}$$

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures existance of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- Furthermore, pivoting (w.r.t. $\max |a_{ij}|$) leads to a backward stable algorithm.

- Once an LU factorization is available, solving a linear system is cheap:

$$\boldsymbol{Ax} = \boldsymbol{LUx} = \boldsymbol{L}(\boldsymbol{Ux}) = \boldsymbol{Ly} = \boldsymbol{b}$$

  or

$$\boldsymbol{PAx} = \boldsymbol{LUx} = \boldsymbol{L}(\boldsymbol{Ux}) = \boldsymbol{Ly} = \boldsymbol{Pb}$$

  - Solve for $\boldsymbol{y}$ using *forward substitution*
  - Solve for $\boldsymbol{x}$ by using *backward substitution* $\boldsymbol{Ux} = \boldsymbol{y}$

Recap: Costs

For forward [backward] substitution at step k there are $\approx k$ $[(n-k)]$ multiplications and subtractions plus a few divisions. The total over all n steps is

$$\sum_{k=1}^{n} k \in \mathcal{O}(n^2)$$

$\rightsquigarrow$ the number of floating-point operations (FLOPs) scales as $\mathcal{O}(n^2)$

For Gaussian elimination, at step k, there are $\approx (n-k)^2$ operations. Thus, the total scales as

$$\sum_{k=1}^{n} (n-k)^2 \in \mathcal{O}(n^3)$$

Summary:

- Directly applying Gaussian elimination (=LU + fwd/bwd) scales as $\mathcal{O}(n^3)$
- Computing LU decomposition scales as $\mathcal{O}(n^3)$
- Forward/backward substitution scales as $\mathcal{O}(n^2)$
- LU + forward/backward scales as $\mathcal{O}(n^3)$ $\rightsquigarrow$ can *reuse LU* for other b

Linear least-squares

Consider non-square matrices $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and rank(A) $= n$. Then the system

$$Ax = b$$

does, in general, not have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_x \|Ax - b\|^2 = \min_x \Phi(\boldsymbol{x})$$

$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$

Geometric sketch

$m \geq n$

$Ax = b$

Idea: project b onto the range of A

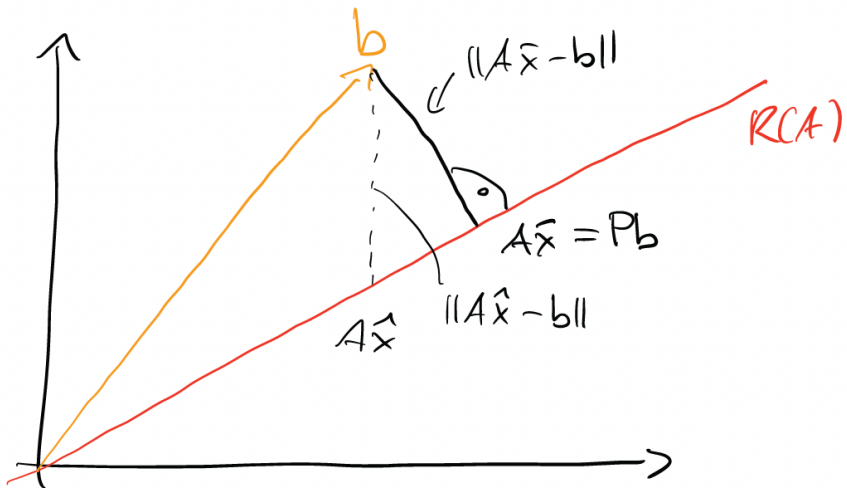
Pb

$Ax = Pb$

$\bar{x} = \arg\min_x \|Ax - b\|_2^2$

$\Updownarrow$

$A\bar{x} - b \perp R(A)$ ← column space of A



Linear least-squares problems

Now for the least-squares problem $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. The relative condition number κ in the Euclidean norm is bounded by

- With respect to perturbations in $\boldsymbol{b}$:

$$\kappa \leq \frac{\kappa_2(A)}{\cos(\theta)}$$

- With respect to perturbations in $\boldsymbol{A}$:

$$\kappa \leq \kappa_2(A) + \kappa_2(A)^2 \tan(\theta)$$

Small residual problems, small angle θ $\cos(\theta) \approx 1$, $\tan(\theta) \approx 0$: behavior similar to linear system.
Large residual problems, large angle θ $\cos(\theta) \ll 1$, $\tan(\theta) \approx 1$: behavior very different from linear system because $\kappa_2(A)^2$ shows up

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

$$\min_x \|Ax - b\|^2 = \min_x \|Q^T(Ax - b)\|^2 = \min_x \|b_1 - R_1 x\|^2 + \|b_2\|^2 = \min_x \| \begin{bmatrix} b_1 - R_1 x \\ b_2 \end{bmatrix} \|^2,$$

where $Q^T b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$.

Thus, the least squares solution is $x = R^{-1} b_1$ and the residual is $\|b_2\|$.

Problem 2: What is the following algorithm doing when applied to an $m \times n$ matrix A and is it a good idea to use it?

for $j = 1, \ldots, n$

- $v_j = A_{:,j}$
- $R_{1:j-1,j} = Q_{:,1:j-1}^T A_{:,j}$
- $v_j = v_j - Q_{:,1:j-1} R_{1:j-1,j}$
- $R_{jj} = \|v_j\|_2$
- $Q_{:,j} = v_j / R_{jj}$

Problem 2: What is the following algorithm doing when applied to an $m \times n$ matrix A and is it a good idea to use it?

for $j = 1, \ldots, n$

- $v_j = A_{:,j}$
- $R_{1:j-1,j} = Q_{:,1:j-1}^T A_{:,j}$
- $v_j = v_j - Q_{:,1:j-1} R_{1:j-1,j}$
- $R_{jj} = \|v_j\|_2$
- $Q_{:,j} = v_j / R_{jj}$

Computes the reduced QR decomposition with Gram-Schmidt. This is not modified Gram-Schmidt, so the columns in Q can be far from orthogonal.

Instead of directly computing

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

based on $\boldsymbol{a}_j$, the *modified* Gram-Schmidt procedure computes $\boldsymbol{v}_j$ iteratively

$$\begin{aligned}
\boldsymbol{v}_j^{(1)} &= \boldsymbol{a}_j, \\
\boldsymbol{v}_j^{(2)} &= \boldsymbol{v}_j^{(1)} - \boldsymbol{q}_1 \boldsymbol{q}_1^T \boldsymbol{v}_j^{(1)}, && \text{"subtract from } \boldsymbol{v}_j^{(1)} \text{ what is already in } \boldsymbol{q}_1\text{"} \\
\boldsymbol{v}_j^{(3)} &= \boldsymbol{v}_j^{(2)} - \boldsymbol{q}_2 \boldsymbol{q}_2^T \boldsymbol{v}_j^{(2)}, && \text{"subtract from } \boldsymbol{v}_j^{(2)} \text{ what is already in } \boldsymbol{q}_2\text{"} \\
&\vdots \\
\boldsymbol{v}_j &= \boldsymbol{v}_j^{(j)} = \boldsymbol{v}_j^{(j-1)} - \boldsymbol{q}_{j-1} \boldsymbol{q}_{j-1}^T \boldsymbol{v}_j^{(j-1)}
\end{aligned}$$

Computing a QR factorization with the modified Gram-Schmidt procedure is stabler than with the classical Gram-Schmidt procedure. However, even the modified Gram-Schmidt procedure can lead to vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$ that are far from orthogonal if the condition number of $\boldsymbol{A}$ is large (see, Golub et al., Matrix Computations, Section 5.2.9)

Eigenproblems

For a matrix $A \in \mathbb{C}^{n \times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Eigenproblems

For a matrix $A \in \mathbb{C}^{n\times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Let $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ be diagonalizable matrix and λ_1 be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Let $\boldsymbol{x}_0$ be an initial guess that is not orthogonal to the eigenspace of λ_1, then $\boldsymbol{x}_k$ obtained via the iterations

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \tag{1}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{z}_{k+1}/\|\boldsymbol{z}_{k+1}\|_2 \tag{2}$$

will converge to the normalized eigenvector of $\boldsymbol{A}$ corresponding to λ_1 for $k \to \infty$.

This process is called the power method. How did we proof convergence?

Problem 3: Computing eigenvectors

Problem 3: Starting with $p_0 = q_0 = 1$, define the iteration

$$\begin{aligned} p_{n+1} &= p_n + q_n \\ q_{n+1} &= p_{n+1} + p_n \end{aligned}$$

for $n = 0, 1, 2, \ldots$. The ratio q_n/p_n converges to $\sqrt{2}$ as $n \to \infty$.

Prove convergence of this algorithm using the power method. I.e., write the problem as an eigenvalue problem and show that the power method for this eigenvalue problem converges to an eigenvector and deduce from this eigenvector that $q_n/p_n \to \sqrt{2}$.

(*Hint:* $\det(A) = ad - bc$ for a 2×2 matrix with $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$ solves $x^2 + bx + c = 0$.)

$q_0 = P_0 = 1$

$$P_{n+1} = P_n + q_n$$
$$q_{n+1} = P_{n+1} + P_n = (P_n + q_n) + P_n = 2P_n + q_n$$

$$\begin{bmatrix} P_{n+1} \\ q_{n+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}}_{A} \begin{bmatrix} P_n \\ q_n \end{bmatrix}$$

Eigenvalues

$$\det(A - \lambda I) = \det\begin{pmatrix} 1-\lambda & 1 \\ 2 & 1-\lambda \end{pmatrix}$$
$$= (1-\lambda)(1-\lambda) - 1 \cdot 2$$
$$= \lambda^2 - 2\lambda - 1$$

roots:

$$\lambda_{1/2} = \frac{-(-2) \pm \sqrt{(-2)^2 - 4 \cdot (-1)}}{2}$$
$$= 1 \pm \sqrt{2}$$

dominant EV $|1 + \sqrt{2}| > |1 - \sqrt{2}|$

λ_1

$$0 = (A - \lambda_1 I)v = \begin{bmatrix} 1 - (1+\sqrt{2}) & 1 \\ 2 & 1 - (1+\sqrt{2}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$= \begin{bmatrix} -\sqrt{2} & 1 \\ 2 & -\sqrt{2} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \overset{!}{=} 0$$

$$\boxed{v_1 = \frac{1}{\sqrt{2}} v_2} \quad (\text{e})$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} p_0 \\ q_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$ not orthogonal

$$\boxed{q_{os} / p_{os}} = \frac{v_2}{v_1} = \frac{v_2}{\frac{1}{\sqrt{2}} v_2} = \sqrt{2}$$

Iterative methods for linear systems

Iterative solution of linear systems

Target problems: very large ($n = 10^5, 10^6, \ldots$), A is usually sparse and has specific properties.

To solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

we construct a sequence

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

of iterates that converges fast to the solution $\boldsymbol{x}$, where $\boldsymbol{x}_{k+1}$ can be cheaply computed from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ (e.g., one matrix-vector multiplication).

Thought experiment: If we can compute one iteration with cost $\mathcal{O}(n)$ (e.g., one matrix-vector multiplication with a sparse matrix) and need a constant $\mathcal{O}(1)$ number of iterations to reach desired precision, then we solve $Ax = b$ with costs $\mathcal{O}(n)$. Intuitively, we cannot do better than that because we solve for n quantities and thus need to touch each at least once.

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and with $\boldsymbol{G}$ invertible obtain that $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is a stationary point

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (need it to be invertible)?

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (need it to be invertible)?

Choose $Q = A^{-1}$, then our iteration becomes

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow A^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - A^{-1}A)\boldsymbol{x} + A^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow \boldsymbol{0} + \boldsymbol{x} = \boldsymbol{x} \end{aligned}$$

and we are done in just a single step

$$x_{k+1} = x$$

Thus, if we "know the solution" (in form of having the inverse A^{-1}) then no further work is needed here because we already did all the work when finding A^{-1}

The other extreme is setting $Q = I$, this leads to the Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

We have invested zero costs in finding Q and so we intuitively expect that $Q = I$ will require high costs in terms of number of iterations to converge in general, if it converges at all

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

  Jacobi uses $Q = D$ and Gauss-Seidel uses $Q = L + D$, which both can be computed quickly from A.
- **What property of A critically influences how quickly these relaxation methods converge?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

  Jacobi uses $Q = D$ and Gauss-Seidel uses $Q = L + D$, which both can be computed quickly from A.
- **What property of A critically influences how quickly these relaxation methods converge?**The condition number of A

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

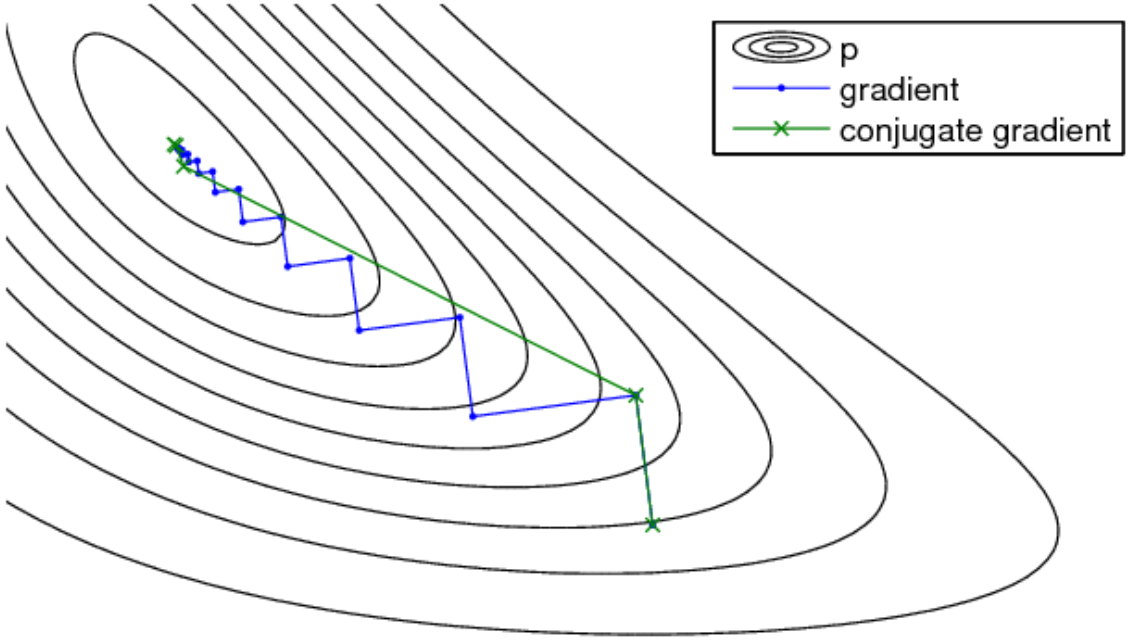
$$f(x) = \frac{1}{2} x^T A x - b^T x \,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, we have $f(x) = 0 \iff Ax = b$. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b = Ax - b = -r(x) = 0 \iff Ax = b$$



[Figure: Kuusela et al., 2009]

The convergence behavior of steepest descent in this context can be poor: we eventually get arbitrarily close to the minimum but we can always destroy something of the already achieved when applying the update $\rightsquigarrow$ can we find better search directions?

Conjugate gradient method

- All methods so far (relaxation, steepest descent) use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

Conjugate gradient method

- All methods so far (relaxation, steepest descent) use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

- The conjugate gradient (CG) method is a variation of steepest descent that *has a memory*.

- Let $p_1, \ldots, p_k$ be the directions up to step k, then CG uses the space

$$x_0 + \operatorname{span}\{p_1, \ldots, p_k\}, \qquad x_0 \text{ starting point}$$

  to find the next iterate x_k and thus

$$x_k = x_0 + \sum_{i=1}^{k} \alpha_i p_i$$

- (Recall that steepest descent uses only the search direction $p_k = r_{k-1} = -\nabla f(x_{k-1})$ to find the iterate x_k)

We want the following

a The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \text{span}(p_1, \ldots, p_k)} f(x)$$

c The step x_k can be calculated easily from x_{k-1}

We then worked hard to derive the CG algorithm based on these three conditions

What was a critical step in CG?

We want the following

a. The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b. We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \text{span}(p_1, \ldots, p_k)} f(x)$$

c. The step x_k can be calculated easily from x_{k-1}

We then worked hard to derive the CG algorithm based on these three conditions

What was a critical step in CG? $\rightsquigarrow$ Gram-Schmidt orthogonalization to find the search direction

It can be shown that for $k \geq 1$ and $e_j \neq 0, j < k$ it holds

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \|e_0\|_A$$

for spd matrices A. $\rightsquigarrow$ Trefethen & Bau

For steepest descent, if A is spd, we obtained

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y \rangle_A = x^T A y$ and $\| \cdot \|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

We keep finding we are limited by the condition number of A. What can be done about it (in Numerical Methods II)?

It can be shown that for $k \geq 1$ and $e_j \neq 0, j < k$ it holds

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \|e_0\|_A$$

for spd matrices A. $\rightsquigarrow$ Trefethen & Bau

For steepest descent, if A is spd, we obtained

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y \rangle_A = x^T A y$ and $\| \cdot \|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

We keep finding we are limited by the condition number of A. What can be done about it (in Numerical Methods II)? Multigrid, multi-level

Solving systems of nonlinear equations

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

Reformulation as fixed point method so that x^* is fixed point

$$x^* = \Phi(x^*)$$

Let $F : \mathbb{R}^n \to \mathbb{R}^n$, $n \geq 1$ and solve

$$F(\boldsymbol{x}) = 0.$$

Truncated Taylor expansion of F about starting point $\boldsymbol{x}^0$:

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}^0) + F'(\boldsymbol{x}^0)(\boldsymbol{x} - \boldsymbol{x}^0).$$

Hence:

$$\boldsymbol{x}^1 = \boldsymbol{x}^0 - F'(\boldsymbol{x}^0)^{-1} F(\boldsymbol{x}^0)$$

Newton iteration: Start with $\boldsymbol{x}^0 \in \mathbb{R}^n$, and for $k = 0, 1, \ldots$ compute

$$F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

Requires that $F'(\boldsymbol{x}^k) \in \mathbb{R}^{n \times n}$ is invertible.

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

  No, can converge to local optima

- You want to solve $F(x) = 0$ but you have access only to a scrambled F through $(G_k \circ F) = G_k(F(x))$, where k is the Newton iteration, and thus you solve a different problem $G_k(F(x)) = 0$ in each iteration. Derive another condition on G_k than G_k being the identity so that Newton converges to x^* with $F(x^*) = 0$ in a neighborhood of x^*. Provide a proof.

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

  No, can converge to local optima

- You want to solve $F(x) = 0$ but you have access only to a scrambled F through $(G_k \circ F) = G_k(F(x))$, where k is the Newton iteration, and thus you solve a different problem $G_k(F(x)) = 0$ in each iteration. Derive another condition on G_k than G_k being the identity so that Newton converges to x^* with $F(x^*) = 0$ in a neighborhood of x^*. Provide a proof.

  Newton is affine invariant. So as long as G_k is linear and the corresponding matrix is regular, it does not influence the Newton iterations. Proof is the same as for the affine invariance that we did in class.

Interpolation

Polynomial interpolation

Consider $n+1$ pairs $(x_i, y_i), i = 0, \ldots, n$ of a function f with

$$y_i = f(x_i)$$

Let now $\mathbb{P}_n$ be the set of all polynomials up to degree n over $\mathbb{R}$ so that we have for all $P \in \mathbb{P}_n$

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \,, \qquad a_n, \ldots, a_0 \in \mathbb{R}$$

We would like to find a $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

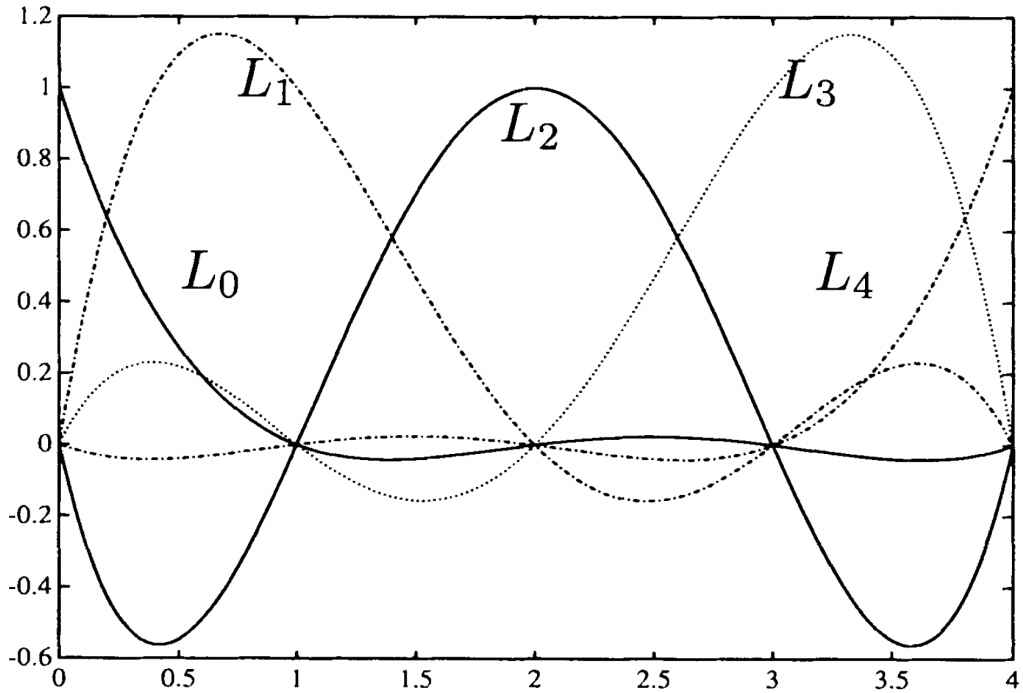
Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

Lagrange basis

The Lagrange polynomials $L_0, \ldots, L_n \in \mathbb{P}_n$ are uniquely defined for distinct $x_0, \ldots, x_n$

$$L_i(x_j) = \delta_{ij}\,, \qquad L_i \in \mathbb{P}_n\,.$$



Lagrange polynomials up to order $n = 4$ for equidistant $x_0, \ldots, x_4$. [Figure: Deuflhard]

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free but the cost of evaluating the polynomial is too high for practical computations

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

The leading coefficient a_n of the interpolation polynomial

$$P_f(x|x_0, \ldots, x_n) = a_n x^n + \cdots + a_0$$

is called the *n-th divided difference*, $[x_0, \ldots, x_n]f := a_n$.

The divided differences are the coefficients $c_0, \ldots, c_n$: The interpolation polynomial $P_f(\cdot|x_0, \ldots, x_n)$ for $x_0 \leq x_1 \leq \cdots \leq x_n$ is given by

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i] f \, \omega_i(x).$$

The following recurrence relation holds for $x_i \neq x_j$:

$$[x_0, \ldots, x_n] f = \frac{([x_0, \ldots, \hat{x_i}, \ldots, x_n] f - [x_0, \ldots, \hat{x_j}, \ldots, x_n] f)}{x_j - x_i}$$

which is helpful to compute the divided differences

Problem 5: Compute the polynomial p with $p(0) = 2, p(1) = 3, p(3) = 0$ in the Newton basis

x_i			
0	$[x_0]p = 2$		
1	$[x_1]p = 3$	$[x_0, x_1]p = \frac{[x_0]p - [x_1]p}{x_0 - x_1} = \frac{2-3}{0-1} = 1$	
3	$[x_2]p = 0$	$[x_1, x_2]p = \frac{[x_1]p - [x_2]p}{x_1 - x_2} = \frac{3-0}{1-3} = -\frac{3}{2}$	

$$[x_0, x_1, x_2]p = \frac{[x_0, x_1]p - [x_1, x_2]p}{x_0 - x_2} = \dots = -\frac{1}{6}$$

Newton-Cotes formulas for quadrature

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where $\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt \rightsquigarrow$ weights are unique
Quadrature formulas defined in this way are exact for polynomials $P \in \mathbb{P}_n$ of degree less than or equal to n

$$\hat{I}(P) = I(P_n(P)) = I(P), \qquad \text{for all } P \in \mathbb{P}_n$$

$$P(x) = 2 + 1x - \frac{5}{6}x(x-1)$$

$P(0) = 2$ ✓

$P(1) = 3$ ✓

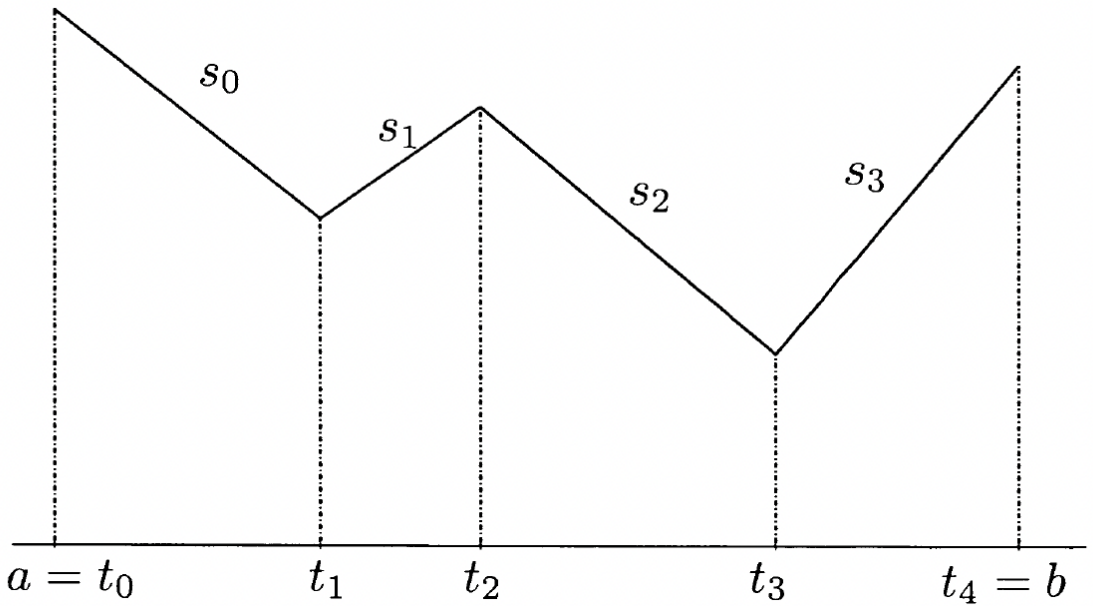
$P(3) = 0$ ✓

Trapezoidal sums

To avoid poorly conditioned problems, let us split the integration interval $[a, b]$ into n sub-intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, n$. Then consider the rule

$$\hat{I}(f) = \sum_{i=1}^{n} \hat{I}_{t_{i-1}}^{t_i}(f),$$

where $\hat{I}_{t_{i-1}}^{t_i}$ is a quadrature formula on the interval $[t_{i-1}, t_i]$.



We have seen already the trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

that has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

$\rightsquigarrow$ we can increase n (and thus decrease h) to reduce the error without increasing the degree of the underlying polynomial

Conclusions

- This was a very selective review of the topics that we covered!
- Be prepared to answer True/False questions to show your basic understand of topics that we discussed.
- Be prepared to answer questions that require 1-2 sentence explanations or short proofs.
- You should be able to do quick calculations such as, e.g., approximating an integral with the trapezoidal rule and computing the LU decomposition of small matrices and computing eigenvalues of a 2×2 matrix etc.
- Recapping basic linear algebra topics will be beneficial (e.g., computing determinant of a 2×2 matrix)