Numerical Linear Algebra
Lecture 5

Estimating Condition Numbers

To compute a practical error bound based on a bound (see Lecture 3)

$$\|\delta x\| = \|A^{-1}r\| \leq \|A^{-1}\|\|r\|$$

we need to estimate $||A^{-1}||$. This is also enough to estimate the condition number $k(A) = ||A^{-1}|| \cdot ||A||$, since $||A||$ is easy to compute. One approach is to compute A^{-1} explicitly and compute its norm. However, this would cost $2n^3$, more than the original $\frac{2}{3}n^3$ for Gaussian elimination. It is a fact that most users will not bother to compute error bounds if they are expensive.
So instead of computing A^{-1} we will devise a much cheaper algorithm to *estimate* $||A^{-1}||$.

Estimating Condition Numbers

Such an algorithm is called a *condition estimator* and should have the following properties:

1. Given the L and U factors of A, it should cost $O(n^2)$, which for large enough n is negligible compared to the $\frac{2}{3}n^3$ cost of GEPP.
2. It should provide an estimate which is almost always within a factor of 10 of $||A^{-1}||$. This is all one needs for an error bound which tells you about how many decimal digits of accuracy that you have.

Estimating Condition Numbers

- There are a variety of such estimators available. We choose one to solve $Ax = b$.
- This estimator is guaranteed to produce only a lower bound on $||A^{-1}||$, not an upper bound.
- It is almost always within a factor of 10, and usually 2 to 3, of $||A^{-1}||$.
- The algorithm estimates the one-norm $||B||_1$ of a matrix B, provided that we can compute Bx and $B^T y$ for arbitrary x and y. We will apply the algorithm to $B = A^{-1}$, so we need to compute $A^{-1}x$ and $A^{-T}y$, i.e., solve linear systems. This costs just $O(n^2)$ given the LU factorization of A.

The algorithm was developed in:
W. W. Hager. Condition estimators. SIAM J. Sci. Statist. Comput., 5:311-316, 1984.
N. J. Higham. A survey of condition number estimation for triangular matrices. SIAM Rev., 29:575-596, 1987.
N. J. Higham. Experience with a matrix norm estimator. SIAM J. Sci. Statist. Comput., 11:804-809, 1990.
with the latest version in [N. J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. ACM Trans. Math. Software].

Recall that $||B||_1$ is defined by

$$||B||_1 = \max_{x \neq 0} \frac{||Bx||_1}{||x||_1} = \max_j \sum_{i=1}^{n} |b_{ij}|.$$

It is easy to show that the maximum over $x \neq 0$ is attained at $x = e_{j_0}[0, \ldots, 0, 1, 0, \ldots, 0]^T$. (The single nonzero entry is component j_0, where $\max_j \sum_i |b_{ij}|$ occurs at $j = j_0$.)
Searching over all e_j, $j = 1, \ldots, n$, means computing all columns of $B = A^{-1}$; this is too expensive. Instead, since $||Bx||_1 = \max_{||x||_1 \leq 1} ||Bx||_1$, we can use *hill climbing* or *gradient ascent* on $f(x) \equiv ||Bx||_1$ inside the set $||x||_1 \leq 1$. $||x||_1 \leq 1$ is clearly a convex set of vectors, and $f(x)$ is a convex function, since $0 \leq \alpha \leq 1$ implies $f(\alpha x + (1-\alpha)y) = ||\alpha Bx + (1-\alpha)By||_1 \leq \alpha||Bx||_1 + (1-\alpha)||By||_1 = \alpha f(x) + (1-\alpha)f(y)$.

Doing gradient ascent to maximize $f(x)$ means moving x in the direction of the gradient $\nabla f(x)$ (if it exists) as long as $f(x)$ increases. The convexity of $f(x)$ means $f(y) \geq f(x) + \nabla f(x) \cdot (y - x)$ (if $\nabla f(x)$ exists). To compute $\nabla f(x)$ we assume all $\sum_j b_{ij} x_j \neq 0$ in $f(x) = \sum_i \sum_j |b_{ij} x_j|$ (this is almost always true). Let $\zeta_i = sign(\sum_j b_{ij} x_j)$, so $\zeta_i = \pm 1$ and $f(x) = \sum_i \sum_j \zeta_i b_{ij} x_j$. Then $\frac{\partial f}{\partial x_k} = \sum_i \zeta_i b_{ik}$ and

$\nabla f = \zeta^T B = (B^T \zeta)^T$.

In summary, to compute $\nabla f(x)$ takes three steps: $\omega = Bx$, $\zeta = sign(\omega)$ and $\nabla f(x) = \zeta^T B$.

ALGORITHM *Hager's condition estimator returns a lower bound* $||\omega||_1$ *on* $||B||_1$*:*

choose any x *such that* $||x||_1 = 1$ $\quad$ $/^* \, e.g. \, x_i = \frac{1}{n} \, ^*/$

repeat

$\omega = Bx$, $\zeta = sign(\omega)$, $z = B^T \zeta$, $\quad$ $/^* \, z^T = \nabla f \, ^*/$

if $||z||_\infty \leq z^T x$ *then*

$\quad$ *return* $||\omega||_1$

else

$\quad$ $x = e_j$ *with 1 at the place* j *where* $|z_j| = ||z||_\infty$

end if

end repeat

Implementation in Matlab

```
x=(1/length(B))*ones(length(B),1);
iter=1;
while iter < 1000
w=B*x; xi=sign(w); z = B'*xi;
if max(abs(z)) <= z'*x
break
else
x= (max(abs(z))== abs(z));
end
iter = iter + 1;
end
LowerBound = norm(w,1);

end
```

THEOREM 1. *When* $||\omega||_1$ *is returned,* $||\omega||_1 = ||Bx||_1$ *is a local maximum of* $||Bx||_1$.
2. *Otherwise,* $||Be_j||$ *(at end of loop)* $> ||Bx||$ *(at start), so the algorithm has made progress in maximizing* $f(x)$.
Proof.
1. In this case, $||z||_\infty \leq z^T x$ (*). Near x, $f(x) = ||Bx||_1 = \sum_i \sum_j \zeta_i b_{ij} x_j$ is linear in x so $f(y) = f(x) + \nabla f(x) \cdot (y - x) = f(x) + z^T(y - x)$, where $z^T = \nabla f(x)$. To show x is a local maximum we want $z^T(y - x) \leq 0$ when $||y||_1 = 1$. We compute

$$\begin{aligned} z^T(y-x) &= z^T y - z^T x = \sum_i z_i \cdot y_i - z^T x \leq \sum_i |z_i| \cdot |y_i| - z^T x \\ &\leq ||z||_\infty \cdot ||y||_1 - z^T x = \underbrace{||z||_\infty - z^T x}_{see(*)} \leq 0. \end{aligned}$$

2. In this case $||z||_\infty > z^T x$. Choose $\widetilde{x} = e_j \cdot sign(z_j)$, where j is chosen so that $|z_j| = ||z||_\infty$. Then

$$\begin{aligned} f(\widetilde{x}) &= f(x) + \nabla f \cdot (\widetilde{x} - x) = f(x) + z^T(\widetilde{x} - x) \\ &= f(x) + z^T\widetilde{x} - z^T x = f(x) + |z_j| - z^T x > f(x), \end{aligned}$$

where the last inequality is true by construction. □

Remarks

Higham [FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation; Experience with a matrix norm estimator] tested a slightly improved version of this algorithm by trying many random matrices of sizes 10,25,50 and condition numbers $k = 10, 10^3, 10^6, 10^9$; in the worst case the computed k underestimated the true k by a factor .44. A different condition estimator is available in Matlab as rcond. The Matlab routine cond computes the exact condition number $||A^{-1}||_2||A||_2$, it is much more expensive than rcond.

Estimating the Relative Condition Number

We can apply the Hager's algorithm to estimate the relative condition number $k_{CR}(A) = || \, |A^{-1}| \cdot |A| \, ||_\infty$ or to evaluate the bound $|| \, |A^{-1}| \cdot |r| \, ||_\infty$. We can reduce both to the same problem, that of estimating $|| \, |A^{-1}| \cdot g \, ||_\infty$, where g is a vector of nonnegative entries. To see why, let e be the vector of all ones. From definition of norm, we see that $||X||_\infty = ||Xe||_\infty$ if the matrix X has nonnegative entries. Then

$$|| \, |A^{-1}| \cdot |A| \, ||_\infty = || \, |A^{-1}| \cdot |A|e \, ||_\infty = || \, |A^{-1}| \cdot g \, ||_\infty,$$

where $g = |A|e$.

Here is how we estimate $|| \, |A^{-1}| \cdot g \, ||_\infty$. Let $G = diag(g_1, \ldots, g_n)$; then $g = Ge$. Thus

$$\begin{aligned} || \, |A^{-1}| \cdot g \, ||_\infty &= || \, |A^{-1}| \cdot Ge \, ||_\infty = || \, |A^{-1}| \cdot G \, ||_\infty = \\ &= || \, |A^{-1}G| \, ||_\infty = ||A^{-1}G||_\infty. \end{aligned} \tag{2.12}$$

The last equality is true because $||Y||_\infty = || \, |Y| \, ||_\infty$ for any matrix Y. Thus, it suffices to estimate the infinity norm of the matrix $A^{-1}G$. We can do this by applying Hager's algorithm to the matrix $(A^{-1}G)^T = GA^{-T}$, to estimate $||(A^{-1}G)^T||_1 = ||A^{-1}G||_\infty$ (see definition of norm).

Practical Error Bounds

We present two practical error bounds for our approximate solution $\widetilde{x}$ of $Ax = b$. For the first bound we use inequality $||\widetilde{x} - x||_\infty \leq ||A^{-1}||_\infty \cdot ||r||_\infty$ to get

$$error = \frac{||\widetilde{x} - x||_\infty}{||\widetilde{x}||_\infty} \leq ||A^{-1}||_\infty \cdot \frac{||r||_\infty}{||\widetilde{x}||_\infty}, \tag{2.13}$$

where $r = A\widetilde{x} - b$ is the residual. We estimate $||A^{-1}||_\infty$ by applying Algorithm to $B = A^{-T}$, estimating $||B||_1 = ||A^{-T}||_1 = ||A^{-1}||_\infty$ (see definition of norm).
Our second error bound comes from the inequality:

$$error = \frac{||\widetilde{x} - x||_\infty}{||\widetilde{x}||_\infty} \leq \frac{|||A^{-1}| \cdot |r|||_\infty}{||\widetilde{x}||_\infty}. \tag{2.14}$$

We estimate $|||A^{-1}| \cdot |r|||_\infty$ using the algorithm based on equation (2.12).

What Can Go Wrong

- Error bounds (2.13) and (2.14) are not guaranteed to provide bounds in all cases in practice.
- First, the estimate of $||A^{-1}||$ from Algorithm (or similar algorithms) provides only a lower bound, although the probability is very low that it is more than 10 times too small.
- Second, there is a small but non-negligible probability that roundoff in the evaluation of $r = A\hat{x} - b$ might make $||r||$ artificially small, in fact zero, and so also make our computed error bound too small. To take this possibility into account, one can add a small quantity to $|r|$ to account for it: the roundoff in evaluating r is bounded by

$$|(A\hat{x} - b) - fl(A\hat{x} - b)| \leq (n+1)\varepsilon(|A| \cdot |\hat{x}| + |b|), \qquad (2.15)$$

  so we can replace $|r|$ with $|r| + (n+1)\varepsilon(|A| \cdot |\hat{x}| + |b|)$ in bound (2.14) or $||r||$ with $||r|| + (n+1)\varepsilon(||A|| \cdot ||\hat{x}|| + ||b||)$ in bound (2.13).
- Third, roundoff in performing Gaussian elimination on very ill-conditioned matrices can yield such inaccurate L and U that bound (2.14) is much too low.

Improving the Accuracy of a Solution

We have just seen that the error in solving $Ax = b$ may be as large as $k(A)\varepsilon$. If this error is too large, what can we do? One possibility is to rerun the entire computation in higher precision, but this may be quite expensive in time and space. Fortunately, as long as $k(A)$ is not too large, there are much cheaper methods available for getting a more accurate solution.

Improving the Accuracy of a Solution

To solve any equation $f(x) = 0$, we can try to use Newton's method to improve an approximate solution x_i to get $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. Applying this to $f(x) = Ax - b$ yields one step of iterative refinement:

$$\begin{aligned} r &= Ax_i - b \\ &\text{solve } Ad = r \text{ for } d \\ x_{i+1} &= x_i - d \end{aligned}$$

If we could compute $r = Ax_i - b$ exactly and solve $Ad = r$ exactly, we would be done in one step, which is what we expect from Newton applied to a linear problem. Roundoff error prevents this immediate convergence. The algorithm is interesting and of use precisely when A is so ill-conditioned that solving $Ad = r$ (and $Ax_0 = b$) is rather inaccurate.

Suppose that r is computed in double precision and $k(A) \cdot \varepsilon < c \equiv \frac{1}{3n^3 g+1} < 1$ where n is the dimension of A and g is the pivot growth factor. Then repeated iterative refinement converges with

$$\frac{||x_i - A^{-1}b||_\infty}{||A^{-1}b||_\infty} = O(\varepsilon).$$

Note that the condition number does not appear in the final error bound. This means that we compute the answer accurately independent of the condition number, provided that $k(A)\varepsilon$ is sufficiently less than 1.(In practice, c is too conservative an upper bound, and the algorithm often succeeds even when $k(A)\varepsilon > c$.)

For partial pivoting of $n \times n$ matrices $g \leq 2^{n-1}$. The classical definition used by Wilkinson is (k is the number of permut.):

$$g(A) := \frac{\max_{i,j,k} |a_{ij}|^{(k)}}{\max_{i,j} |a_{ij}|}.$$

Another definition for LU decomposition of A is:

$$g(A) := \frac{\||L| \cdot |U|\|_\infty}{\|A\|_\infty}.$$

Single Precision Iterative Refinement

THEOREM.
Suppose that r is computed in single precision and

$$||A^{-1}||_\infty \cdot ||A||_\infty \cdot \frac{\max_i(|A| \cdot |x|)_i}{\min_i(|A| \cdot |x|)_i} \cdot \varepsilon < 1.$$

Then one step of iterative refinement yields x_1 such that $(A + \delta A)x_1 = b + \delta b$ with $|\delta a_{ij}| = O(\varepsilon)|a_{ij}|$ and $|\delta b_i| = O(\varepsilon)|b_i|$. In other words, the componentwise relative backward error is as small as possible. For example, this means that if A and b are sparse, then δA and δb have the same sparsity structures as A and b, respectively.

For a proof, see

N. J. Higham. Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia, PA, 1996.

M. Arioli, J. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. SIAM J. Matrix Anal. AppL, 10:165-190, 1989.

R. D. Skeel. Scaling for numerical stability in Gaussian elimination. Journal of the ACM, 26:494-526, 1979.

R. D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. Math. Comp., 35:817-832, 1980.

R. D. Skeel. Effect of equilibration on residual size for partial pivoting. SIAM J. Numer. Anal, 18:449-454, 1981.

Single precision iterative refinement and the error bound (2.14) are implemented in LAPACK routines like sgesvx.

Equilibration

- There is one more common technique for improving the error in solving a linear system: **equilibration**. This refers to choosing an appropriate diagonal matrix D and solving $D\mathbf{A}\mathbf{x} = Db$ instead of $Ax = b$. D is chosen to try to make the condition number of DA smaller than that of A.
- For instance, choosing d_{ii} to be the reciprocal of the two-norm of row i of A would make DA nearly equal to the identity matrix, reducing its condition number from 10^{14} to 1.
- It is possible to show that choosing D this way reduces the condition number of DA to within a factor of $\sqrt{n}$ of its smallest possible value for any diagonal D [A. Van Der Sluis. Condition numbers and equilibration of matrices. Numer. Math., 14:14-23, 1969].
- In practice we may also choose two diagonal matrices D_{row} and D_{col} and solve $(D_{row}AD_{col})\bar{x} = D_{row}b$, $x = D_{col}\bar{x}$ and thus $D_{row}Ax = D_{row}b$.

Special Linear Systems

It is important to exploit any special structure of the matrix to increase speed of solution and decrease storage. We will consider only real matrices:

- s.p.d. matrices,
- symmetric indefinite matrices,
- band matrices,
- general sparse matrices,
- dense matrices depending on fewer than n^2 independent parameters.

Real Symmetric Positive Definite Matrices

Recall that a real matrix A is s.p.d. if and only if $A = A^T$ and $x^T Ax > 0$ for all $x \neq 0$. In this section we will show how to solve $Ax = b$ in half the time and half the space of Gaussian elimination when A is s.p.d.

PROPOSITION.

1. *If X is nonsingular, then A is s.p.d. if and only if $X^T AX$ is s.p.d.*
2. *If A is s.p.d. and H is any principal submatrix of A($H = A(j:k, j:k)$ for some $j \leq k$), then H is s.p.d.*
3. *A is s.p.d. if and only if $A = A^T$ and all its eigenvalues are positive.*
4. *If A is s.p.d., then all $a_{ii} > 0$, and $\max_{ij} |a_{ij}| = \max_i a_{ii} > 0$.*
5. *A is s.p.d. if and only if there is a unique lower triangular nonsingular matrix L, with positive diagonal entries, such that $A = LL^T$. $A = LL^T$ is called the Cholesky factorization of A, and L is called the Cholesky factor of A.*

Proof.

1. **If X is nonsingular, then A is s.p.d. if and only if X^TAX is s.p.d.**

$\Rightarrow$ When X is nonsingular and A is s.p.d. implies that for any $Xx \neq 0$ for all $x \neq 0$, so $x^TX^TAXx > 0$ for all $x \neq 0$. We can see that $x^TX^TAXx = x^TX^TAXx > 0$ what implies X^TAX is s.p.d.

$\Leftarrow$ When X^TAX is s.p.d. then for all $x \neq 0$ we have that $x^TX^TAXx > 0$ and $X^TAX = (X^TAX)^T = XA^TX^T = X^TA^TX$. Thus, $A = A^T$. We observe that $x^TX^TAXx = x^TX^TAXx > 0$ what means that A is s.p.d.

2. **If A is s.p.d. and H is any principal submatrix of A($H = A(j:k, j:k)$ for some $j \leq k$), then H is s.p.d.**
Suppose first that $H = A(1:m, 1:m)$. Then given any m-vector y, the n-vector $x = [y^T, O]^T$ satisfies $y^T H y = x^T A x$. So if $x^T A x > 0$ for all nonzero x, then $y^T H y > 0$ for all nonzero y, and so H is s.p.d. If H does not lie in the upper left corner of A, let P be a permutation so that H does lie in the upper left corner of $P^T A P$ and apply Part 1.

3. **A is s.p.d. if and only if $A = A^T$ and all its eigenvalues are positive.**
$\Rightarrow$ Let A is s.p.d., then $A = A^T$ and for all $x \neq 0$ $x^T Ax > 0$.
Let X be the real, orthogonal eigenvector matrix of A so that $X^T AX = \bigwedge$ is the diagonal matrix of real eigenvalues λ_i. Since $x^T \bigwedge x = \sum_i \lambda_i x_i^2$ then by part 1 since A is s.p.d. then $X^T AX$ is s.p.d. and thus $0 < x^T X^T AX x = x^T \bigwedge x = \sum_i \lambda_i x_i^2$. Thus, $\sum_i \lambda_i x_i^2 > 0$ only when each $\lambda_i > 0$.

$\Leftarrow$ Let $A = A^T$ and all eigenvalues λ_i of A are postive. Let X be the real, orthogonal eigenvector matrix of A so that $X^T AX = \bigwedge$ is the diagonal matrix of real eigenvalues λ_i. Consider $x^T \bigwedge x = \sum_i \lambda_i x_i^2$. Since all eigenvalues λ_i of A are postive then $x^T \bigwedge x = \sum_i \lambda_i x_i^2 > 0$. We observe then that $\bigwedge$ is positive definite. From $x^T \bigwedge x > 0$ it follows that $x^T X^T AX x = x^T \bigwedge x > 0$ and thus, $X^T AX$ is s.p.d. By part 1, A is also s.p.d.

4. **If A is s.p.d., then all $a_{ii} > 0$, and $\max_{ij} |a_{ij}| = \max_i a_{ii} > 0$.** Let e_i be the ith column of the identity matrix. Then $e_i^T A e_i = a_{ii} > 0$ for all i. If $|a_{kl}| = \max_{ij} |a_{ij}|$ but $k \neq l$, choose $x = e_k - sign(a_{kl}) e_l$. Then $x^T A x = a_{kk} + a_{ll} - 2|a_{kl}| \leq 0$, contradicting positive-definiteness.

5. A is s.p.d. if and only if there is a unique lower triangular nonsingular matrix L, with positive diagonal entries, such that $A = LL^T$. $A = LL^T$ is called the Cholesky factorization of A, and L is called the Cholesky factor of A.

Suppose $A = LL^T$ with L nonsingular. Note that $\|L^T x\|_2^2 = (L^T x)^T L^T x = (x^T L)(L^T x)$. Then $x^T A x = (x^T L)(L^T x) = \|L^T x\|_2^2 > 0$ for all $x \neq 0$, so A is s.p.d. If A is s.p.d., we show that L exists by induction on the dimension n. If we choose each $l_{ii} > 0$, our construction will determine L uniquely. If $n = 1$, choose $l_{11} = \sqrt{a_{11}}$, which exists since $a_{11} > 0$. As with Gaussian elimination, it suffices to understand the block 2-by-2 case.

Write

$$
\begin{aligned}
A &= \begin{bmatrix} a_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix} \\
&= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{A_{12}^T}{\sqrt{a_{11}}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{A}_{22} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{A_{12}}{\sqrt{a_{11}}} \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & A_{12} \\ A_{12}^T & \tilde{A}_{22} + \frac{A_{12}^T A_{12}}{a_{11}} \end{bmatrix},
\end{aligned}
$$

so the $(n-1)$-by-$(n-1)$ matrix $\tilde{A}_{22} + \frac{A_{12}^T A_{12}}{a_{11}}$ is symmetric and thus, A is also symmetric.

We note to the previous slide (here we write how we can obtain elements in the last matrix on the previous slide) that

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix} = LL^T = \\ &= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ y & \tilde{L}_{22} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & y^T \\ 0 & \tilde{L}_{22}^T \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & \sqrt{a_{11}} y^T \\ \sqrt{a_{11}} y & yy^T + \tilde{L}_{22}\tilde{L}_{22}^T \end{bmatrix}, \end{aligned}$$

and thus $A_{12} = \sqrt{a_{11}}y$ such that we can find $y = \frac{A_{12}}{\sqrt{a_{11}}}$ and $A_{22} = yy^T + \tilde{L}_{22}\tilde{L}_{22}^T = \tilde{A}_{22} + \frac{A_{12}^T A_{12}}{a_{11}}$ with $\tilde{A}_{22} = \tilde{L}_{22}\tilde{L}_{22}^T$.

By Part 1 above, $\begin{bmatrix} 1 & 0 \\ 0 & \tilde{A}_{22} \end{bmatrix}$ is s.p.d, so by Part 2 $\tilde{A}_{22}$ is s.p.d.

Thus by induction there exists an $\tilde{L}$ such that $\tilde{A}_{22} = \tilde{L}\tilde{L}^T$ and

$$
\begin{aligned}
A &= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{A_{12}^T}{\sqrt{a_{11}}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{L}\tilde{L}^T \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{A_{12}}{\sqrt{a_{11}}} \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{A_{12}^T}{\sqrt{a_{11}}} & \tilde{L} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{A_{12}}{\sqrt{a_{11}}} \\ 0 & \tilde{L}^T \end{bmatrix} \equiv LL^T. \ \square
\end{aligned}
$$

We may rewrite this induction as the following algorithm.

ALGORITHM *Cholesky algorithm:*

for $j = 1$ *to* n
$l_{jj} = (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{1/2}$
for $i = j + 1$ *to* n
$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk})/l_{jj}$
end for
end for

If A is not positive definite, then (in exact arithmetic) this algorithm will fail by attempting to compute the square root of a negative number or by dividing by zero; this is the cheapest way to test if a symmetric matrix is positive definite.

Number of operations in Cholesky algorithm

In Cholesky algorithm L can overwrite the lower half of A. Only the lower half of A is referred to by the algorithm, so in fact only $n(n+1)/2$ storage is needed instead of n^2. The number of flops is

Number of operations in Cholesky algorithm

$$= \sum_{j=1}^{n} \left(2j + \sum_{i=j+1}^{n} 2j \right) = \frac{1}{3} n^3 + O(n^2). \tag{1}$$

The number of operations for LU decomposition is $\frac{2}{3} n^3 + O(n^2)$.

Stability of Cholesky algorithm

Pivoting is not necessary for Cholesky to be numerically stable. We show this as follows. The same analysis as for Gaussian elimination reveals that we will have similar formula for error E in Cholesky decomposition as in the LU decomposition:

$$A = LL^T + E,$$

where error in Cholesky decomposition will be bounded as

$$|E| \leq n\epsilon |L| \cdot |L^T|.$$

Taking norms we get

$$\|E\| \leq n\epsilon \| \, |L| \, \| \cdot \| \, |L^T| \, \|.$$

We can rewrite expression above as

$$\|E\| \leq n\epsilon \|L\| \cdot \|L^T\|. \tag{2}$$

Thus, in formula (2) we have obtained error estimate in decomposition $A = L \cdot L^T$.

Stability of Cholesky algorithm

To show how the error in Gaussian elimination fo solution $Ax = b$ can be obtained via usiang Cholesky decomposition $A = L \cdot L^T$ in it, we again solve $L \underbrace{L^T x}_{y} = b$ via $Ly = b$ and $L^T x = y$. Solving $Ly = b$ gives as a computed solution $\hat{y}$ such that $(L + \delta L)\hat{y} = b$ where $|\delta L| \leq n\varepsilon |L|$. The same is true for $(L^T + \delta L^T)\hat{x} = \hat{y}$ with $|\delta L^T| \leq n\varepsilon |L^T|$. Combining both estimates into one we get

$$
\begin{aligned}
b = (L + \delta L)\hat{y} &= (L + \delta L)(L^T + \delta L^T)\hat{x} \\
&= (LL^T + L\delta L^T + \delta L L^T + \delta L \delta L^T)\hat{x} \\
&= (A \underbrace{-E + L\delta L^T + \delta L L^T + \delta L \delta L^T}_{\delta A})\hat{x} \\
&= (A + \delta A)\hat{x}.
\end{aligned}
$$

Stability of Cholesky algorithm

Recall

$$\delta A = -E + L\delta L^T + \delta L L^T + \delta L \delta L^T.$$

Now we combine all bounds for $E, \delta L^T, \delta L$ and use triangle inequality to get

$$\begin{aligned}
|\delta A| &\leq | - E + L\delta L^T + \delta L L^T + \delta L \delta L^T| \\
&\leq |E| + |L| \cdot |\delta L^T| + |\delta L| \cdot |L^T| + |\delta L| \cdot |\delta L^T| \\
&\leq n\varepsilon |L| \cdot |L^T| + n\varepsilon |L| \cdot |L^T| + n\varepsilon |L| \cdot |L^T| + n^2\varepsilon^2 |L| \cdot |L^T| \\
&\approx 3n\varepsilon |L| \cdot |L^T|.
\end{aligned}$$

Stability of Cholesky algorithm

Assuming that $\| |X| \| = \|X\|$ is true (as before for Frobenius, infinity, one-norms but not for two-norms) we obtain

$$||\delta A|| \leq 3n\varepsilon ||L|| \cdot ||L^T||. \tag{3}$$

Thus, from (3) follows that the computed solution $\tilde{x}$ satisfies $(A + \delta A)\tilde{x} = b$ with $|\delta A| \leq 3n\varepsilon |L| \cdot |L^T|$. But by the Cauchy-Schwartz inequality and proposition (part 4) we have that for every entry (i, j) of $|L| \cdot |L^T|$ we can write estimate

$$\begin{aligned} (|L| \cdot |L^T|)_{ij} &= \sum_k |l_{ik}| \cdot |l_{jk}| \\ &\leq \sqrt{\textstyle\sum_k l_{ik}^2}\sqrt{\textstyle\sum_k l_{jk}^2} \\ \leq \sqrt{a_{ii}} \cdot \sqrt{a_{jj}} &\leq \max_{ij} |a_{ij}| = \max_i a_{ii} > 0. \end{aligned}$$

Cholesky algorithm
for j = 1 to n
$l_{jj} = (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{1/2}$
for i = j + 1 to n
$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk})/l_{jj}$
end for
end for

Stability of Cholesky algorithm

Then applying this estimate to all n entries of $|L| \cdot |L^T|$ we have

$$\| \, |L| \cdot |L^T| \, \|_\infty \leq n \|A\|_\infty. \tag{4}$$

Substituting (4) into (3) we get the following estimate

$$\|\delta A\|_\infty \leq 3n^2 \varepsilon \|A\|_\infty. \tag{5}$$

From it follows that Cholesky is stable when

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq 3n^2 \varepsilon, \tag{6}$$

what means that Cholesky is "more stable" than LU. Recall that for LU we have obtained that Gaussian elimination is stable if

$$\|\delta A\| \leq 3n\varepsilon \|L\| \|U\| \leq 3n\varepsilon \|A\|. \tag{7}$$

Symmetric Indefinite Matrices

- The question of whether we can still save half the time and half the space when solving a symmetric but indefinite (neither positive definite nor negative definite) linear system naturally arises. It turns out to be possible, but a more complicated pivoting scheme and factorization is required.
- If A is nonsingular, one can show that 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 $PAP^T = LDL^T$.
- To see why 2-by-2 blocks are needed in D, consider the matrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. This factorization can be computed stably, saving about half the work and space compared to standard Gaussian elimination.

Band Matrices

A matrix A is called a *band matrix* with *lower bandwidth* b_L, and *upper bandwidth* b_U if $a_{ij} = 0$ whenever $i > j + b_L$ or $i < j - b_U$:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1,b_U+1} & & 0 \\ \vdots & & & a_{2,b_U+2} & \\ a_{b_L+1,1} & & & & \ddots \\ & a_{b_L+2,2} & & & a_{n-b_U,n} \\ & & \ddots & & \vdots \\ 0 & & & a_{n,n-b_L} & \cdots \quad a_{n,n} \end{bmatrix}.$$

Band matrices arise often in practice and are useful to recognize because their L and U factors are also "essentially banded", making them cheaper to compute and store. We consider LU factorization without pivoting and show that L and U are banded in the usual sense, with the same band widths as A.

Example of a bandmatrix

The matrix A which appears after discretization of Laplacian in the Poisson's equation $-\triangle u = f$ is a bandmatrix. After discretization of laplacian we should solve system in the form $Au = b$. The vector b has the components $b_{i,j} = h^2 f_{i,j}$. The explicit elements of the matrix A are given by the following block matrix

$$A = \left(\begin{array}{c|c|c} A_N & -I_N & \\ \hline -I_N & \ddots & \ddots \\ \hline & \ddots & \ddots & -I_N \\ \hline & & -I_N & A_N \end{array}\right)$$

with blocks A_N of order N given by

$$A_N = \begin{pmatrix} 4 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 4 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 4 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & -1 & 4 \end{pmatrix},$$

Example: ODE

Example

Consider the ordinary differential equation (ODE) $y''(x) - p(x)y'(x) - q(x)y(x) = r(x)$ on the interval $[a, b]$ with boundary conditions $y(a) = \alpha$, $y(b) = \beta$. We also assume $q(x) \geq \underline{q} > 0$. This equation may be used to model the heat flow in a long, thin rod, for example. To solve the differential equation numerically, we *discretize* it by seeking its solution only at the evenly spaced mesh points $x_i = a + ih$, $i = 0, \ldots, N + 1$, where $h = (b - a)/(N + 1)$ is the mesh spacing. Define $p_i = p(x_i)$, $r_i = r(x_i)$, and $q_i = q(x_i)$.

Example

We need to derive equations to solve for our desired approximations $y_i \approx y(x_i)$, where $y_0 = \alpha$ and $y_{N+1} = \beta$. To derive these equations, we approximate the derivative $y\prime(x_i)$ by the following *finite difference approximation*:

$$y\prime(x_i) \approx \frac{y_{i+1} - y_{i-1}}{2h}.$$

(Note that as h gets smaller, the right-hand side approximates $y\prime(x_i)$ more and more accurately.) We can similarly approximate the second derivative by

$$y\prime\prime(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}.$$

Inserting these approximations into the differential equation yields

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - p_i \frac{y_{i+1} - y_{i-1}}{2h} - q_i y_i = r_i, \quad 1 \leq i \leq N.$$

Example

Multiplying by $-h^2/2$ we get:

$$-\frac{y_{i+1}}{2} + y_i - \frac{y_{i-1}}{2} + p_i\frac{y_{i+1}h}{4} - p_i\frac{y_{i-1}h}{4} + q_i y_i\frac{h^2}{2} = -\frac{-r_i h^2}{2}$$

Rewriting this as a linear system we get $Ay = b$, where

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad b = \frac{-h^2}{2}\begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} + \begin{bmatrix} (\frac{1}{2} + \frac{h}{4}p_1)\alpha \\ 0 \\ \vdots \\ 0 \\ (\frac{1}{2} - \frac{h}{4}p_N)\beta \end{bmatrix},$$

Example

and recalling

$$-\frac{y_{i+1}}{2} + y_i - \frac{y_{i-1}}{2} + p_i\frac{y_{i+1}h}{4} - p_i\frac{y_{i-1}h}{4} + q_i y_i\frac{h^2}{2} = -\frac{-r_i h^2}{2}$$

we have

$$A = \begin{bmatrix} a_1 & -c_1 & & \\ -b_2 & \ddots & \ddots & \\ & \ddots & \ddots & -c_{N-1} \\ & & -b_N & a_N \end{bmatrix}, \quad \begin{aligned} a_i &= 1 + \tfrac{h^2}{2} q_i, \\ b_i &= \tfrac{1}{2}[1 + \tfrac{h}{2} p_i], \\ c_i &= \tfrac{1}{2}[1 - \tfrac{h}{2} p_i]. \end{aligned}$$

Note that $a_i > 0$ and also $b_i > 0$ and $c_i > 0$ if h is small enough. This is a nonsymmetric *tridiagonal* system to solve for y. We will show how to change it to a symmetric positive definite tridiagonal system, so that we may use *band Cholesky* to solve it.

Example

Choose $D = diag(1, \sqrt{\frac{c_1}{b_2}}, \sqrt{\frac{c_1 c_2}{b_2 b_3}}, \ldots, \sqrt{\frac{c_1 c_2 \cdots c_{N-1}}{b_2 b_3 \cdots b_N}})$. Then we may change $Ay = b$ to $(\underbrace{DAD^{-1}}_{\tilde{A}})\underbrace{(Dy)}_{\tilde{y}} = \underbrace{Db}_{\tilde{b}}$ or $\tilde{A}\tilde{y} = \tilde{b}$, where

$$\tilde{A} = \begin{bmatrix} a_1 & -\sqrt{c_1 b_2} & & \\ -\sqrt{c_1 b_2} & a_2 & -\sqrt{c_2 b_3} & \\ & -\sqrt{c_2 b_3} & \ddots & \\ & \ddots & \ddots & -\sqrt{c_{N-1} b_N} \\ & & -\sqrt{c_{N-1} b_N} & a_N \end{bmatrix}.$$

It is easy to see that $\tilde{A}$ is symmetric, and it has the same eigenvalues as A because A and $\tilde{A} = DAD^{-1}$ are *similar*. We will use the next theorem to show it is also positive definite.

Gershgorin's Theorem

THEOREM (Gershgorin's) *Let B be an arbitrary matrix. Then the eigenvalues λ of B are located in the union of the n disks*

$$|\lambda - b_{kk}| \leq \sum_{j \neq k} |b_{kj}|.$$

Proof. Given λ and $x \neq 0$ such that $Bx = \lambda x$, let $1 = ||x||_\infty = x_k$ by scaling x if necessary. Then $\sum_{j=1}^{N} b_{kj} x_j = \lambda x_k = \lambda$, so $\lambda - b_{kk} = \sum_{\substack{j=1 \\ j \neq k}}^{N} b_{kj} x_j$, implying

$$|\lambda - b_{kk}| \leq \sum_{j \neq k} |b_{kj} x_j| \leq \sum_{j \neq k} |b_{kj}||x_j| \leq \sum_{j \neq k} |b_{kj}| = R_k. \quad \square$$

Examples of using Gershgorin's circle theorem

Example

Example 1.
Use the Gershgorin circle theorem to estimate the eigenvalues of

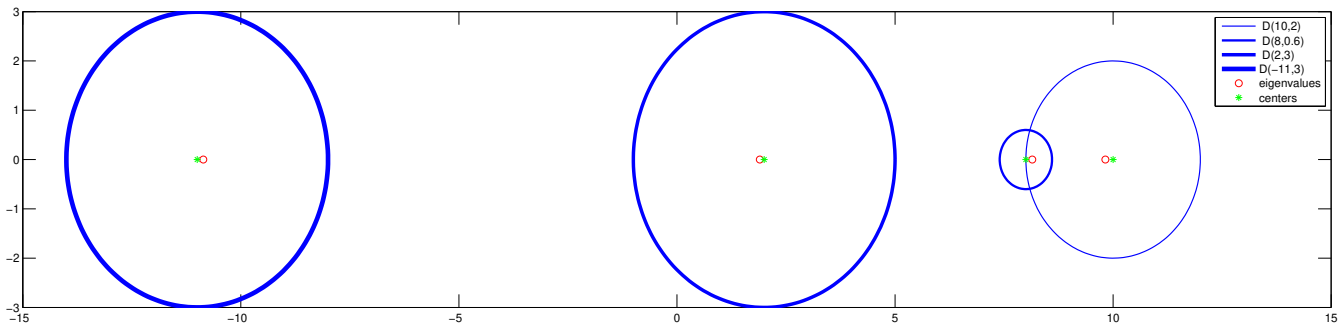
$$A = \begin{bmatrix} 10 & -1 & 0 & 1 \\ 0.2 & 8 & 0.2 & 0.2 \\ 1 & 1 & 2 & 1 \\ -1 & -1 & -1 & -11 \end{bmatrix}.$$

Starting with row one, we take the element on the diagonal, a_{ii} as the center for the disc. We then take the remaining elements in the row and apply the formula:

$$\sum_{j \neq i} |a_{ij}| = R_i$$

to obtain the following four discs:
$D(10, 2), D(8, 0.6), D(2, 3),$ and $D(-11, 3)$
The eigenvalues are: 9.8218, 8.1478, 1.8995, -10.86

Example



Example 1

In example 1 the eigenvalues are: 9.8218, 8.1478, 1.8995, -10.86

Examples of using Gershgorin's circle theorem

Example

Example 2.
Use the Gershgorin circle theorem to estimate the eigenvalues of

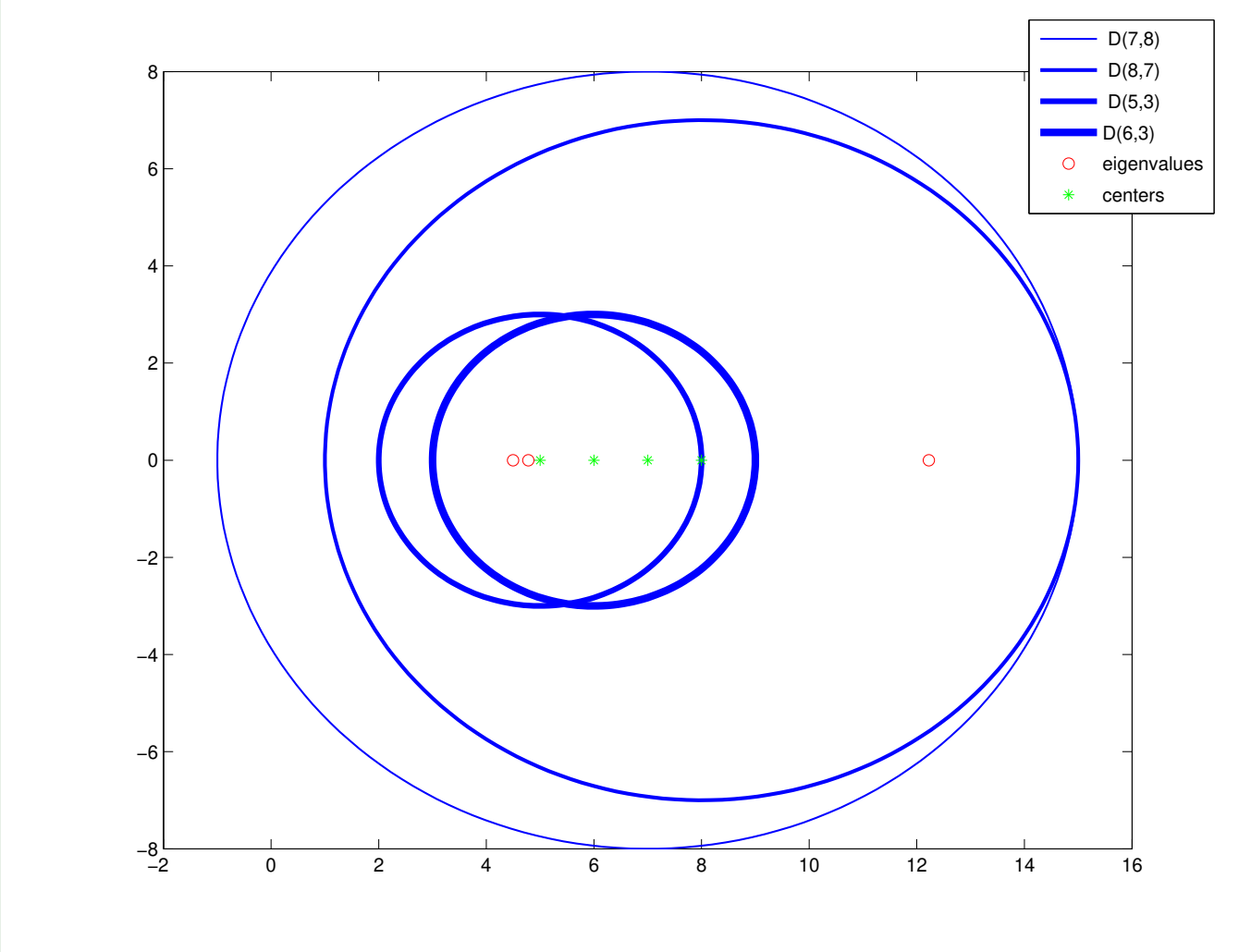
$$A = \begin{bmatrix} 7 & 5 & 2 & 1 \\ 2 & 8 & 3 & 2 \\ 1 & 1 & 5 & 1 \\ 1 & 1 & 1 & 6 \end{bmatrix}.$$

Starting with row one, we take the element on the diagonal, a_{ii} as the center for the disc. We then take the remaining elements in the row and apply the formula:

$$\sum_{j \neq i} |a_{ij}| = R_i$$

to obtain the following four discs:
$D(7,8), D(8,7), D(5,3), D(6,3)$
The eigenvalues are: 12.2249 + 0.0000i; 4.4977 + 0.6132i; 4.4977 - 0.6132i; 4.7797 + 0.0000i;

Example



Example 2

Example: ODE (continuation)

Example

- If h is so small that for all i, $|\frac{h}{2}p_i| < 1$, then

$$|b_i|+|c_i| = \frac{1}{2}\left(1+\frac{h}{2}p_i\right)+\frac{1}{2}\left(1-\frac{h}{2}p_i\right) = 1 < 1+\frac{h^2}{2}\underline{q} \leq 1+\frac{h^2}{2}q_i = a_i.$$

  Therefore all eigenvalues of A lie inside the disks centered at $a_i = 1 + h^2 q_i/2 \geq 1 + h^2\underline{q}/2 > 1$ with radius 1; We can conclude that all eigenvalues must have positive real parts.

- Since $\tilde{A}$ is symmetric, its eigenvalues are real and hence positive, so $\tilde{A}$ is positive definite. Its smallest eigenvalue is bounded below by $\underline{q}h^2/2$.

- Thus, it can be solved by Cholesky.

Example: solution of Poisson's equation

The model problem is the following Dirichlet problem for Poisson's equation:

$$\begin{aligned} -\triangle u(x) &= f(x) \quad \text{in} \quad \Omega, \\ u &= 0 \text{ on } \partial\Omega. \end{aligned} \tag{8}$$

Here $f(x)$ is a given function, $u(x)$ is the unknown function, and the domain Ω is the unit square $\Omega = \{(x_1, x_2) \in (0,1) \times (0,1)\}$. To solve numerically (8) we first discretize the domain Ω with $x_{1\,i} = ih_1$ and $x_{2\,j} = jh_2$, where $h_1 = 1/(n_i - 1)$ and $h_2 = 1/(n_j - 1)$ are the mesh sizes in the directions x_1, x_2, respectively, n_i and n_j are the numbers of discretization points in the directions x_1, x_2, respectively. In this example we choose $n_i = n_j = n$ with $n = N + 2$, where N is the number of inner nodes in the directions x_1, x_2, respectively.

Example: solution of Poisson's equation

Indices (i, j) are such that $0 < i, j < n + 1$ and are associated with every global node n_{glob} of the finite difference mesh. Global nodes numbers n_{glob} in two-dimensional case can be computed as:

$$n_{glob} = j + n_i(i - 1). \tag{9}$$

or using the following loop (here, *ni* is number of points in x direction, *nj* - number of points in y direction):

```
for  i=1:ni
for j=1:nj
n_g  = j + ni*(i - 1).
end
end
```

We use the standard finite difference discretization of the Laplace operator Δu in two dimensions and obtain discrete laplacian $\Delta u_{i,j}$:

$$\Delta u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}, \qquad (10)$$

where $u_{i,j}$ is the solution at the discrete point (i, j). Using (10), we obtain the following scheme for solving problem (8):

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) = f_{i,j}, \qquad (11)$$

where $f_{i,j}$ are the value of the function f at the discrete point (i, j). Then (11) can be rewritten as

$$-(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) = h^2 f_{i,j}, \qquad (12)$$

or in the more convenient form as

$$-u_{i+1,j} + 4u_{i,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = h^2 f_{i,j}. \qquad (13)$$

System (13) can be written in the form $Au = b$. The vector b has the components $b_{i,j} = h^2 f_{i,j}$. The explicit elements of the matrix A are given by the following block matrix

$$A = \left(\begin{array}{c|c|c} A_N & -I_N & \\ \hline -I_N & \ddots & \ddots \\ & \ddots & \ddots & -I_N \\ \hline & & -I_N & A_N \end{array}\right)$$

with blocks A_N of order N given by

$$A_N = \begin{pmatrix} 4 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 4 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 4 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & -1 & 4 \end{pmatrix},$$

which are located on the diagonal of the matrix A, and blocks with the identity matrices $-I_N$ of order N on its off-diagonals. The matrix A is symmetric and positive definite and we can use the LU factorization algorithm without pivoting.

Suppose, that we have discretized the two-dimensional domain Ω as described above with $N = n_i = n_j = 3$. We present the schematic discretization via the global nodes numbering for all $1 \leq i, j < n + 1$

$$n_{glob} = j + n_i(i - 1).$$

in the following scheme:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \Longrightarrow \begin{pmatrix} n_1 & n_2 & n_3 \\ n_4 & n_5 & n_6 \\ n_7 & n_8 & n_9 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}. \tag{14}$$

Then the explicit form of the block matrix A will be:

$$A = \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{array}\right).$$

Example 8.2: Gaussian elimination for solution of Poisson's equation

We illustrate the numerical solution of problem (8). We define the right hand side $f(x)$ of (8) as

$$f(x_1, x_2) = A_f \exp\left(-\frac{(x_1 - c_1)^2}{2s_1^2} - \frac{(x_2 - c_2)^2}{2s_2^2}\right)\frac{1}{a(x_1, x_2)}, \tag{15}$$

The coefficient $a(x_1, x_2)$ in (15) is given by the following Gaussian function:

$$a(x_1, x_2) = 1 + A \exp\left(-\frac{(x_1 - c_1)^2}{2s_1^2} - \frac{(x_2 - c_2)^2}{2s_2^2}\right), \tag{16}$$

Here A, A_f are the amplitudes of these functions, c_1, c_2 are constants which show the location of the center of the Gaussian functions, and s_1, s_2 are constants which show spreading of the functions in x_1 and x_2 directions.
We produce the mesh with the points $(x_{1\,i}, x_{2\,j})$ such that $x_{1\,i} = ih, x_{2\,j} = jh$ with $h = 1/(N+1)$, where N is the number of the inner points in x_1 and x_2 directions. The linear system of equations $Au = f$ is solved then via the LU factorization of the matrix A without pivoting.

Example 8.2: solution of Poisson's equation via LU

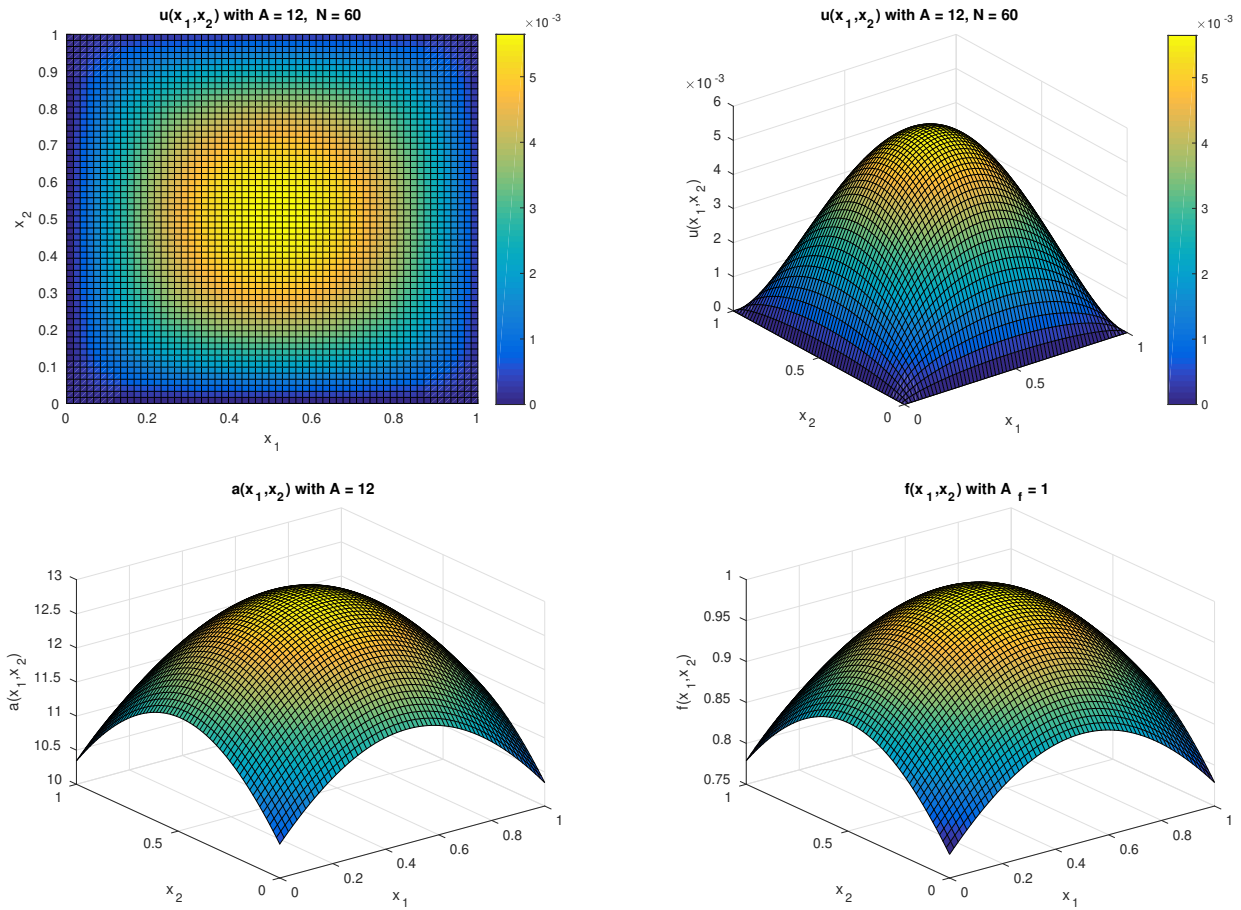


Figure: Solution of Poisson's equation (8) with $f(x_1, x_2)$ as in (15) and $a(x_1, x_2)$ as in (16).

Example 8.4.4: solution of Poisson's equation using Cholesky factorization

$$f(x_1, x_2) = 1 + 10e^{\left(-\frac{(x_1-0.25)^2}{0.02} - \frac{(x_2-0.25)^2}{0.02}\right)} + 10e^{\left(-\frac{(x_1-0.75)^2}{0.02} - \frac{(x_2-0.75)^2}{0.02}\right)} \tag{17}$$

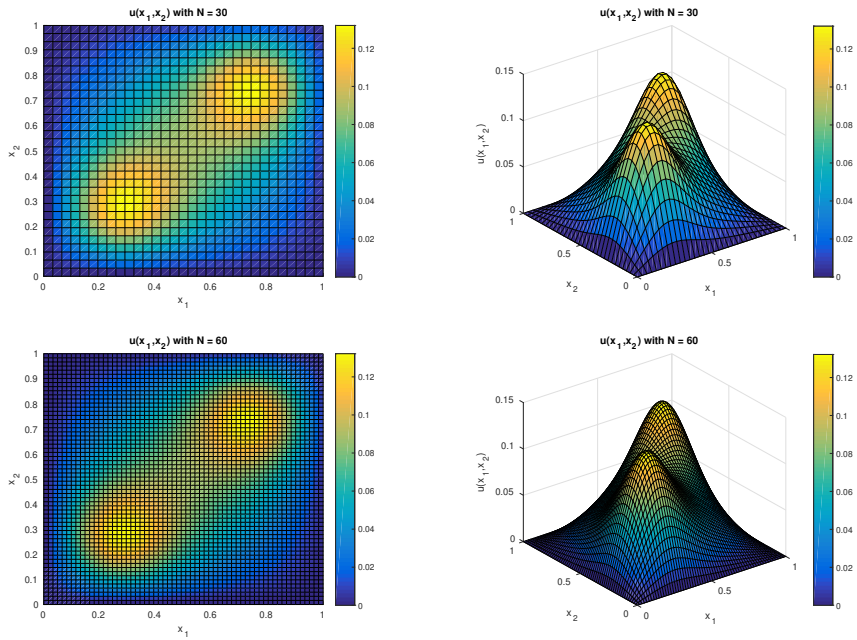


Figure: Solution of Poisson's equation (8) with $f(x_1, x_2)$ as in (17).