Direct Solvers Review

- We present a brief overview of direct solvers, a.k.a., Gaussian Elimination (*GE*).

- These differ from iterative solvers in that they terminate in a finite number of steps. (Technically, conjugate gradient iteration also terminates in a finite number of steps—but we rarely need to take that many steps to have a converged solution.)

- We will see that direct solvers are advantageous for systems coming from low-dimensional PDEs in $\mathbb{R}^d$ (i.e., $d = 1$ or 2), but generally not competitive for $d > 2$. For $d = 2$, the winning approach is largely determined by the condition number of the system matrix.

- Direct methods also form the basis for some preconditioning strategies known as ILU methods, which are based on incomplete *LU* factorizations.

- We'll start with GE for general (i.e., *dense*) matrices so that we internalize the central ideas.
  Key take-aways for this section:
  - pivoting, for stability
  - performance using block-based (BLAS3) algorithms
  - performance gains for banded systems

- We'll start with Gil Strang's perspective on the *geometry of linear systems*.

- Consider the 6×6 system,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}. \tag{1}$$

- We can view A as a set of 6 column vectors, $\underline{a}_j$, $j = 1:6$,

$$\begin{bmatrix} | & | & | & | & | & | \\ \underline{a}_1 & \underline{a}_2 & \underline{a}_3 & \underline{a}_4 & \underline{a}_5 & \underline{a}_6 \\ | & | & | & | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}. \tag{2}$$

- The matrix-vector product, $A\underline{x}$, is a linear combination of the columns of A,

$$\left[\begin{array}{cccccc} | & | & | & | & | & | \\ | & | & | & | & | & | \\ x_1\underline{a}_1 + & x_2\underline{a}_2 + & x_3\underline{a}_3 + & x_4\underline{a}_4 + & x_5\underline{a}_5 + & x_6\underline{a}_6 \\ | & | & | & | & | & | \\ | & | & | & | & | & | \end{array}\right] = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}. \tag{3}$$

- The following notation is a bit more consistent.

$$\left[\begin{array}{cccccc} | & | & | & | & | & | \\ | & | & | & | & | & | \\ \underline{a}_1 x_1 + & \underline{a}_2 x_2 + & \underline{a}_3 x_3 + & \underline{a}_4 x_4 + & \underline{a}_5 x_5 + & \underline{a}_6 x_6 \\ | & | & | & | & | & | \\ | & | & | & | & | & | \end{array}\right] = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}. \tag{4}$$

- The unknowns to be found are the column coefficients, x_j.
- This geometric *column* perspective, *Find a linear combination of the vectors,* $\underline{a}_j$*, with coefficients* x_j *such that* $A\underline{x} = \underline{b}$, is quite distinct from the *row* perspective, which views each equation as a describing a hyperplane of dimension $n-1$ embedded in $\mathbb{R}^n$ and seeking the intersection point of these n hyperplanes.

- **A key idea** in linear algebra that is central to iterative methods is that *every matrix-vector product is a linear combination of the columns of that matrix.*

- Consider an $m \times n$ matrix, V. The matrix-vector product $V\underline{y}$ is

$$\underline{z} \;=\; V\underline{y} \;=\; \underline{v}_1 y_1 + \underline{v}_2 y_2 + \cdots + \underline{v}_n y_n. \tag{5}$$

- **Q:** What can we say about the vector $\underline{z}$ in the following expression?

$$\underline{z} \;=\; V(V^T A V)^{-1} V^T \underline{y} \tag{6}$$

  A: We can say that $\underline{z}$ lies in the *column space of* V, which is also known as the *range of* V, denoted as $\mathcal{R}(V)$.

  That is, $\underline{z}$ is a linear combination of the columns of V. Always.

- We explore the implications of this fact in through geometric interpretations of linear systems in the following examples.

The Geometry of Linear Equations[1]

- Example, 2×2 system:

$$\left.\begin{array}{rcl} 2x - y &=& 1 \\ x + y &=& 5 \end{array}\right\} \iff \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

- Can look at this system by *rows* or *columns*.
- We will do both.

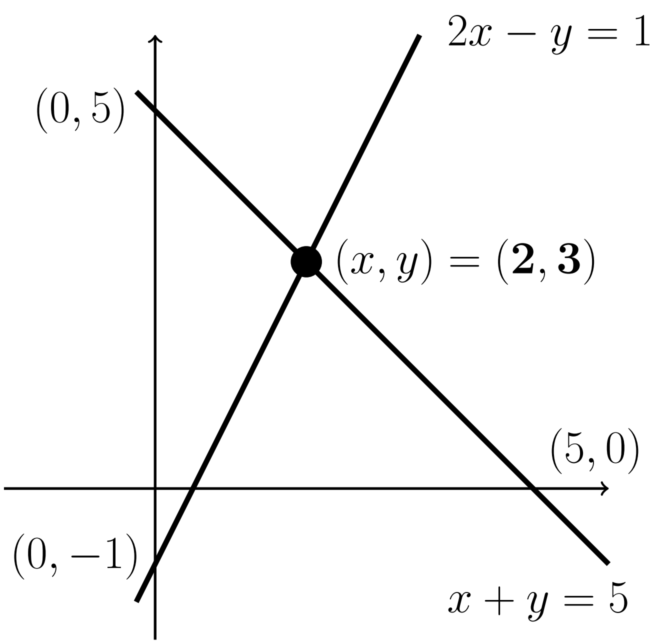
[1]Gilbert Strang: *Linear Algebra and Its Applications*

Row Form

- In the 2×2 system, each equation represents a line:

$$\begin{aligned} 2x - y &= 1 \qquad \text{line 1} \\ x + y &= 5 \qquad \text{line 2} \end{aligned}$$

- The intersection of the two lines gives the unique point $(x, y) = (2, 3)$, which is the solution.



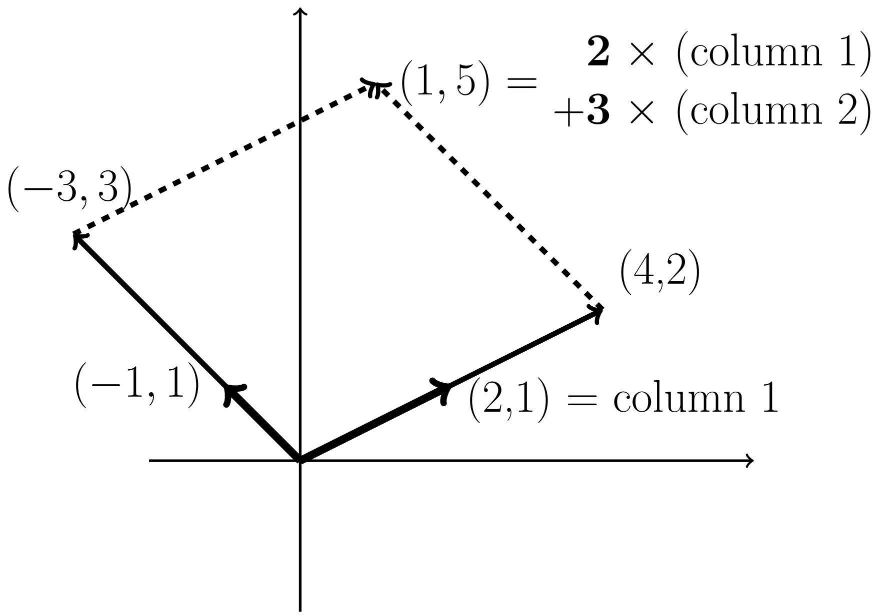
- We remark that the system is relatively *ill-conditioned* if the lines are close to being parallel, that is, if the smallest subtended angle is close to 0.

Column Form

- The second (and more important) geometry is column based.
- Here, we view the system of equations as *one vector equation*:

$$\textbf{Column form} \qquad x\begin{bmatrix}2\\1\end{bmatrix} + y\begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}1\\5\end{bmatrix}.$$

- The problem is to find coefficients, x and y, such that the combination of vectors on the left equals the vector on the right.



- In this case, the system is *ill-conditioned* if the column vectors are nearly parallel.

  If these vectors are separated by an angle θ, it's relatively easy to show that the condition number scales as $\kappa \sim \frac{2}{\theta}$ as $\theta \longrightarrow 0$.

Row Form: A Case with n=3.

Three planes:

$$\begin{aligned} 2u + v + w &= 5 \\ 4u - 6v \qquad &= -2 \\ -2u + 7v + 2w &= 9 \end{aligned}$$

- Each equation (*row*) defines a plane in $\mathbb{R}^3$.
- The first plane is $2u + v + w = 5$ and it contains points ($\frac{5}{2}$,0,0) and (0,5,0) and (0,0,5).
- It is determined by three points, provided they do not lie on a line.
- Changing 5 to 10 would shift the plane to be parallel this one, with points (5,0,0) and (0,10,0) and (0,0,10).

Row Form: A Case with n=3, cont'd.

- The second plane is $4u - 6v = -2$.
- It is vertical because it can take on any w value.
- The intersection of this plane with the first is a *line.*
- The third plane, $-2u + 7v + 2w = 9$ intersects this line at a point, $(u, v, w) = (1, 1, 2)$, which is the solution.
- In n dimensions, the solution is the intersection point of n hyperplanes, each of dimension $n - 1$.   A bit confusing.

Column Vectors and Linear Combinations

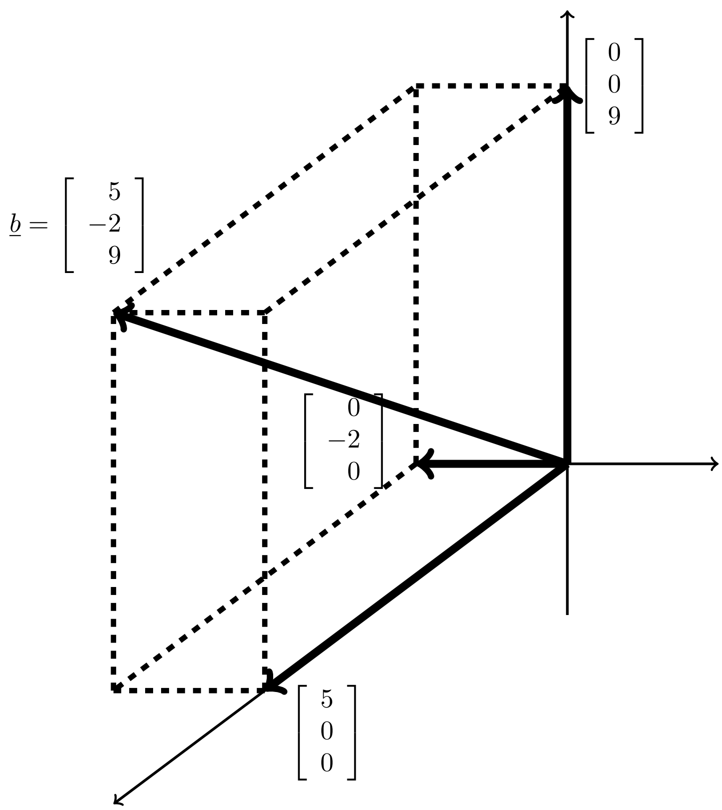
- The preceding system in $\mathbb{R}^3$ can be viewed as the vector equation

$$u \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + v \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + w \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix} = \underline{b}.$$

- Our task is to find the multipliers, u, v, and w.
- The vector $\underline{b}$ is identified with the point (5,-2,9).
- We can view $\underline{b}$ as a list of numbers, a point, or an arrow.
- For $n > 3$, it's probably best to view it as a list of numbers.

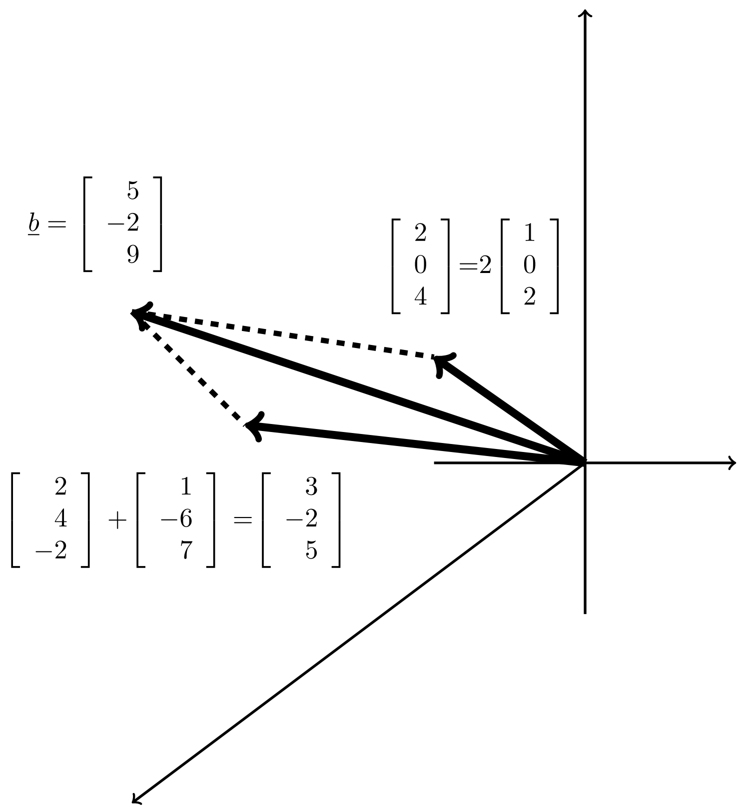
Vector Addition Example

$$\begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 9 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}.$$

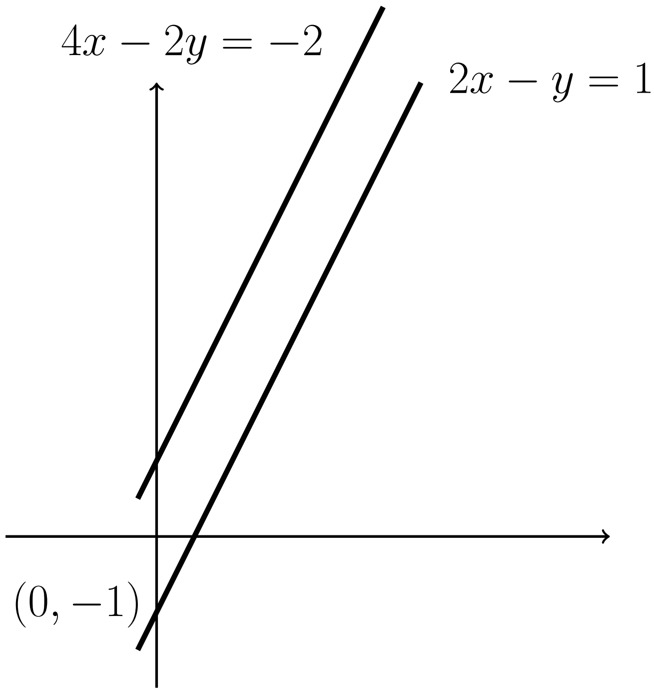


Linear Combination

$$\mathbf{1}\begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + \mathbf{1}\begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + \mathbf{2}\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}.$$



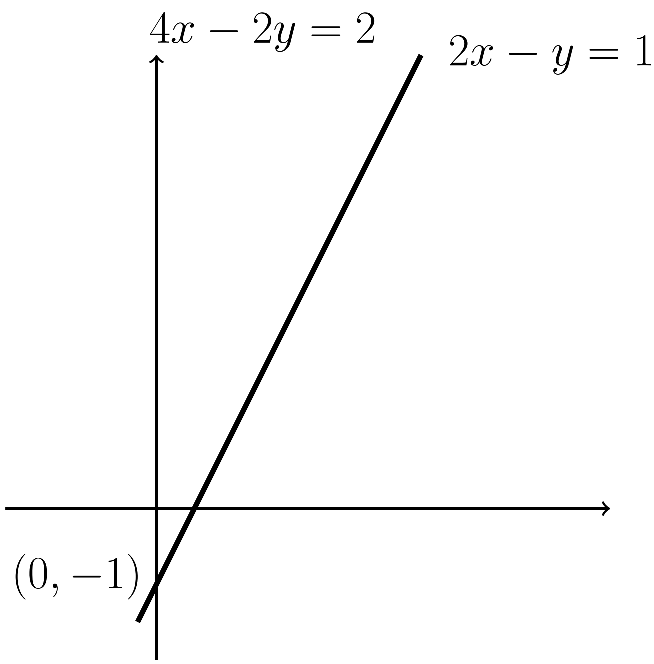
The Singular Case: Row Picture



$$\begin{aligned} 2x - y &= 1 \\ 4x - 2y &= -2 \end{aligned}$$

- No solution.

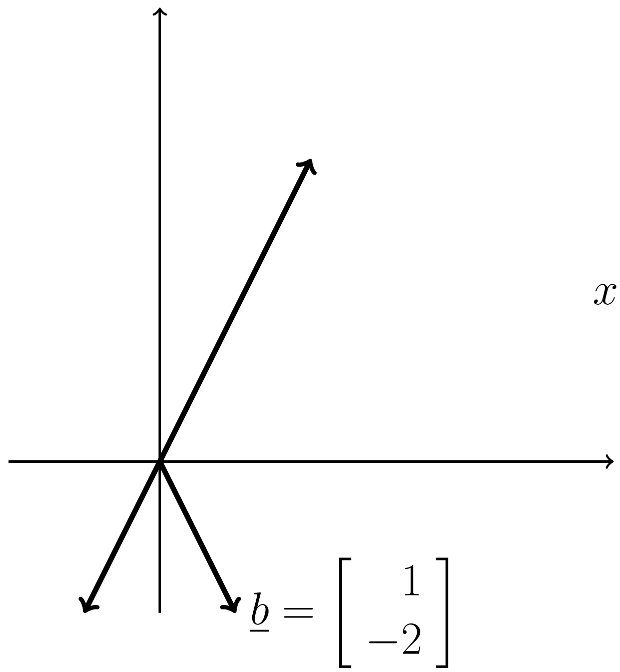
The Singular Case: Row Picture



$$\begin{aligned} 2x - y &= 1 \\ 4x - 2y &= 2 \end{aligned}$$

- Infinite number of solutions.

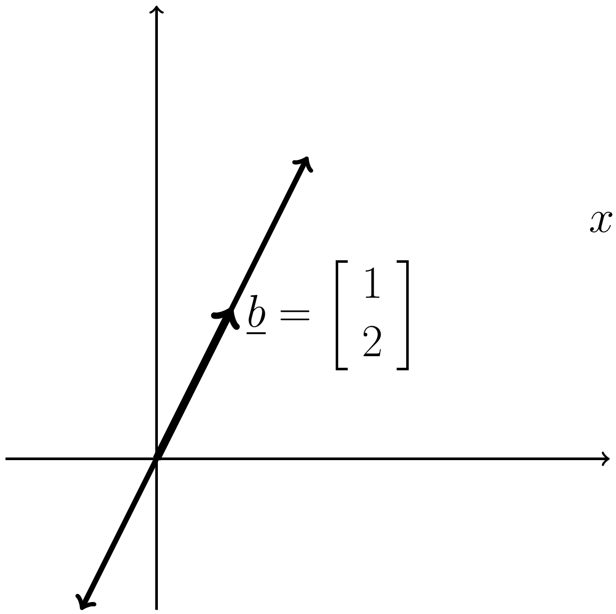
The Singular Case: Column Picture



$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- No solution.

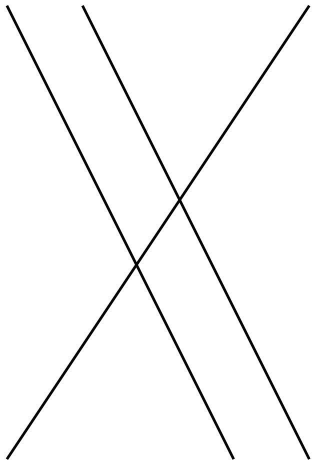
The Singular Case: Column Picture



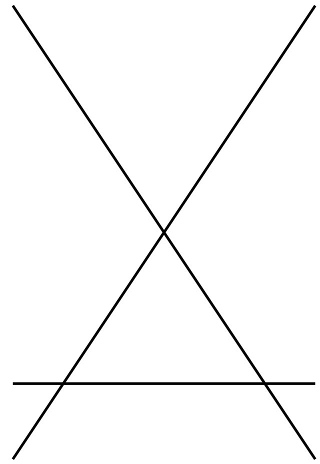
$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- Infinite number of solutions. ($\underline{b}$ coincident with $\underline{a}_1$ and $\underline{a}_2$.)

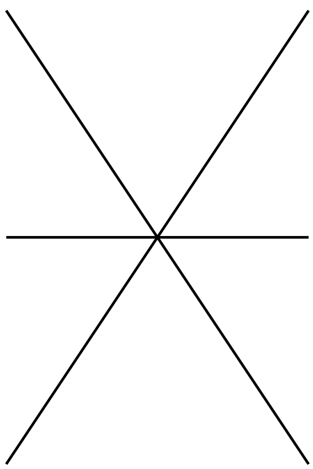
Singular Case: Row Picture with n=3



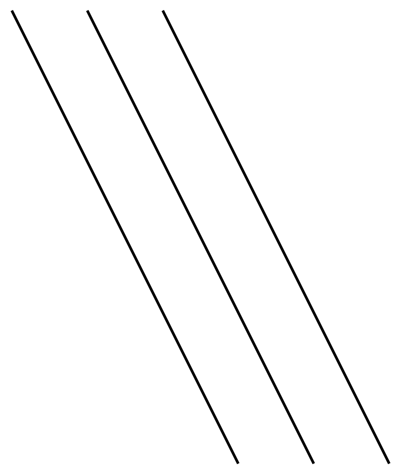
(a) two parallel planes



(b) no intersection



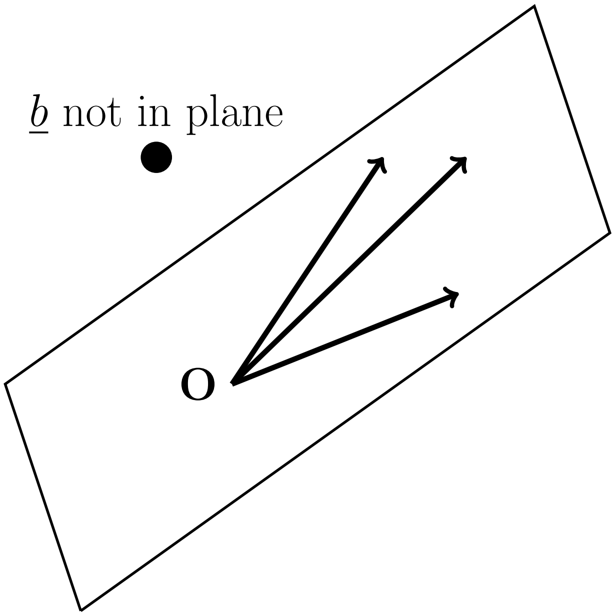
(c) line of intersection

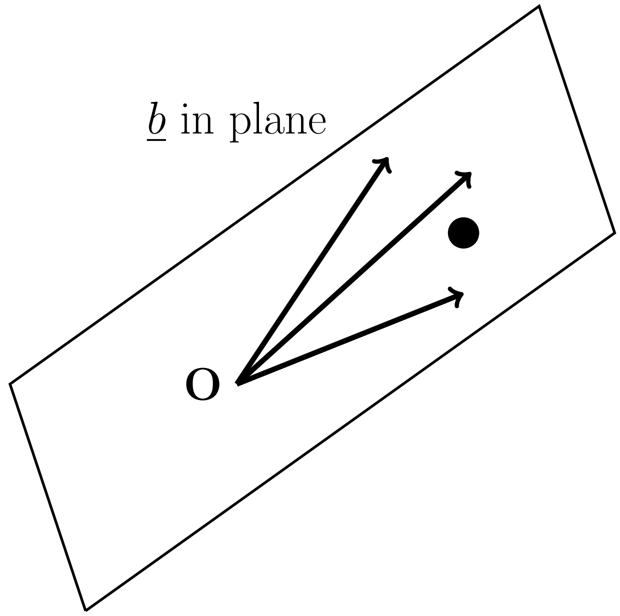


(d) all planes parallel

End-on view of 3 planes.

Singular Case: Column Picture with n=3





- In this case, the three columns of the system matrix lie in the same plane.

$$\text{Example:} \quad u \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + v \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + w \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \underline{b}.$$

- On the left, $\underline{b}$ is not in the plane $\longrightarrow$ *no solution.*
- On the right, $\underline{b}$ is in the plane $\longrightarrow$ *an inifnite number of solutions.*
- Our system is *solvable* (we can get to any point in $\mathbb{R}^3$) for **any** $\underline{b}$ if the three columns are *linearly independent.*

Gaussian Elimination = LU Factorization

Triangular Solves Example

- Upper- or lower-triangular systems are straightforward to solve.
- Consider the following upper-triangular system governing the unknown, $\underline{x} = [x_1\ x_2\ x_3]^T$.

$$\begin{array}{rcrcrcl} 1 \cdot x_1 & + & 2 \cdot x_2 & + & 3 \cdot x_3 & = & 16 \\ & & 4 \cdot x_2 & + & 5 \cdot x_3 & = & 14 \\ & & & & 6 \cdot x_3 & = & 12 \end{array} \tag{7}$$

- To solve this, we use the well-known *backward substitution* approach of working from the bottom equation (which is trivial) up to the first equation.

- From the bottom, we have

$$x_3 \;=\; 12/6 = \;2\;. \tag{8}$$

- Next up, we can find x_2 as

$$4 \cdot x_2 \;=\; 14 \;-\; 5 \cdot x_3 \;=\; 14 - 5 \cdot 2 \;=\; 4, \tag{9}$$

  so $x_2 = 1$.

- Finally, from the first equation, we have:

$$1 \cdot x_1 \;=\; 16 \;-\; 3 \cdot x_3 \;-\; 2 \cdot x_2 \;=\; 16 \;-\; 3 \cdot 2 \;-\; 2 \cdot 1 \;=\; 8. \tag{10}$$

- Note that we can permute the rows of this system without changing the answer:

$$\begin{array}{rcrcrcl} & & & & 6\cdot x_3 & = & 12 \\ & & 4\cdot x_2 & + & 5\cdot x_3 & = & 14 \\ 1\cdot x_1 & + & 2\cdot x_2 & + & 3\cdot x_3 & = & 16 \end{array} \tag{11}$$

- We can also permute the columns:

$$\begin{array}{lclclcl} 6\cdot x_3 & & & & & = & 12 \\ 5\cdot x_3 & + & 4\cdot x_2 & & & = & 14 \\ 3\cdot x_3 & + & 2\cdot x_2 & + & 1\cdot x_1 & = & 16 \end{array} \tag{12}$$

  Here, nothing has changed, save for the positions on the page.

- The equations and, hence the solution, are the same.
  The solution process follows in precisely the same way as before.

- We conclude that solving a lower-triangular system is essentially the same as solving an upper-triangular system.

  One starts with the trivial entry, computes that value and subtracts a multiple of it from the RHS for the next equation.

  This process is repeated as each unknown (x_3, x_2, etc.) becomes known.

A More General Example

- For more general systems, the convention is to effect a sequence of transformations such that the result is an equivalent *upper triangular system.*

- Because we work in finite-precision arithmetic, "equivalent" must be tempered by the expectation that there will be round-off errors.

- Good (i.e., *stable*) algorithms, however, will mitigate these round-off errors to the extent possible.

- In general, if the condition number of the system matrix is 10^k, we can expect to lose k digits of accuracy.

- For example, if we are working in FP64, we have 16 digits of accuracy in the representation of most numbers. If the condition number of the system matrix is 10^5, we can expect only 11 digits of accuracy in the final result.

- **Q:** For the same system, what accuracy should we expect if working in
  - FP32?
  - FP16?

- The transformation of a general matrix to upper triangular form is known as *Gaussian Elimination* and it is equivalent to what is known as *LU* factorization.

- Equivalence-preserving operations used in Gaussian elimination include
  - row interchanges
  - column interchanges (relatively rare; used only for "full pivoting")
  - addition of a multiple of another row to a given row

  Notice that we do not include "multiplication of a row by a constant" because, while valid for any nonzero constant, it is generally not needed for Gaussian elimination.

- We have already seen how row/column interchanges can transform a system from lower-triangular form to upper-triangular form and can understand that reversing that procedure would take us back to our targeted upper-triangular form.

- Let's now look at the row-addition process for a more general example.

Generating Upper Triangular Systems: LU Factorization

- Example:

$$\begin{bmatrix} 1 & 2 & 3 & & \\ & 4 & 4 & 6 & 1 \\ & 8 & 8 & 9 & 2 \\ & 6 & 1 & 3 & 3 \\ & 4 & 2 & 8 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}$$

- First column is already in upper triangular form.
- Eliminate second column:

$$\begin{aligned} \text{row}_3 &\longleftarrow \text{row}_3 - \frac{8}{4} \times \text{row}_2 \\ \text{row}_4 &\longleftarrow \text{row}_4 - \frac{6}{4} \times \text{row}_2 \\ \text{row}_5 &\longleftarrow \text{row}_5 - \frac{4}{4} \times \text{row}_2 \end{aligned} \qquad \begin{bmatrix} 1 & 2 & 3 & & \\ & 4 & 4 & 6 & 1 \\ & & 0 & -3 & 0 \\ & & -5 & -6 & \frac{3}{2} \\ & & -2 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ -4 \\ -2 \\ 0 \end{bmatrix}$$

- $a_{22} = 4$ is the *pivot*
- row_2 is the *pivot row*
- $l_{32} = \frac{8}{4}$, $l_{42} = \frac{6}{4}$, $l_{52} = \frac{4}{4}$, is the *multiplier column.*

- Notice that neither row_1 nor row_2 is modified in this process.
  - row_1 is already in upper triangular form.
  - row_2 is the pivot row, which is unchanged.

Generating Upper Triangular Systems: LU Factorization

- Augmented form. Store $\underline{b}$ in $A(:, n+1)$:

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & 8 & 8 & 9 & 2 & 4 \\ & 6 & 1 & 3 & 3 & 4 \\ & 4 & 2 & 8 & 4 & 4 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

This Case. ***General Case.***

$$\text{pivot} = 4 \qquad = a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$\text{pivot row} = [\ 4\ \ 6\ \ 1 \mid 4\ ] \qquad = \underline{r}_k^T = a_{kj},\ j = k+1, \ldots, n\,[+b_k\,]$$

$$\text{multiplier column} = \frac{1}{4}\begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix} \qquad = \underline{c}_k = \frac{a_{ik}}{a_{kk}},\ i = k+1, \ldots, n$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

Next Step: $k = k + 1$

- We now move to eliminate the next column, $k = 3$.

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

- Here, we have diffiulty because the nominal pivot, a_{33} is zero.
- The remedy is to exchange rows with one of the remaining two, since the order of the equations is immaterial.
- For numerical stability, we choose the row that maximizes $|a_{ik}|$.
- This choice ensures that all entries in the multiplier column are less than one in modulus.

- **Q:** From a performance standpoint, should we explicitly swap rows? Or just use a pointer?

Next Step: $k = k + 1$

- After switching rows, we have

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & 0 & -3 & 0 & -4 \\ & & -2 & 2 & 3 & 0 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & 0 & -3 & 0 & -4 \\ & & 0 & 4\frac{2}{5} & 2\frac{2}{5} & \frac{4}{5} \end{array}\right]$$

$$\begin{aligned} \text{pivot} &= -5 \\ \text{pivot row} &= \left[\begin{array}{cc|c} -6 & \frac{3}{2} & -2 \end{array}\right] \\ \text{multiplier column} &= \frac{1}{-5}\begin{bmatrix} 0 \\ -2 \end{bmatrix} \end{aligned}$$

Code for the general case, without pivoting:

As derived, in *row* form:

```
for k = 1 : min(m, n)
    piv = a_kk
    for i = k+1 : m
        a_ik = a_ik/piv
        for j = k+1 : n
          a_ij = a_ij - a_ik * a_kj
        end
    end
end
```

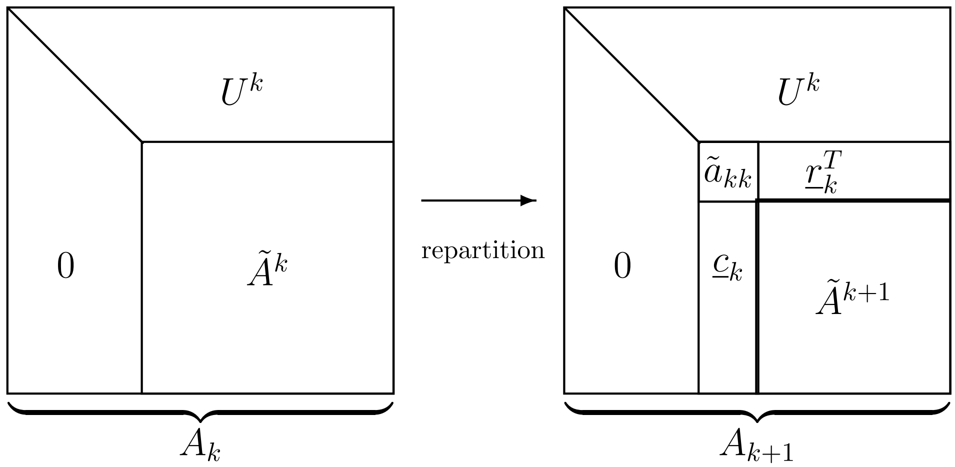
Better memory access (much faster):

```
for k = 1 : min(m, n)
    piv = a_kk
    for i = k+1 : m          % put multiplier column
        a_ik = a_ik/piv      % in lower part of A
    end
    for j = k+1 : n          % Ã^{k+1} = Ã^{k+1} - c_k r_k^T
    for i = k+1 : m
        a_ij = a_ij - a_ik * a_kj
    end
    end
end
```

- Remarkably, L is now resident in the overwritten lower part of A.
- To retrieve L and U, we use the following:

```
l = min(m, n);   L = zeros(m,l);   U = zeros(l,n);
for k = 1 : l
    L(k : end, k) = A(k : end, k);   L(k, k) = 1;
    U(k, k : end) = A(k, k : end);
end
```

Illustration of Basic Update Step:



- A_k is the reduced form of A at the start of step k.
- $\tilde{A}^k$ is the active submatrix A^k starting at row k, col k.
- After identifying the

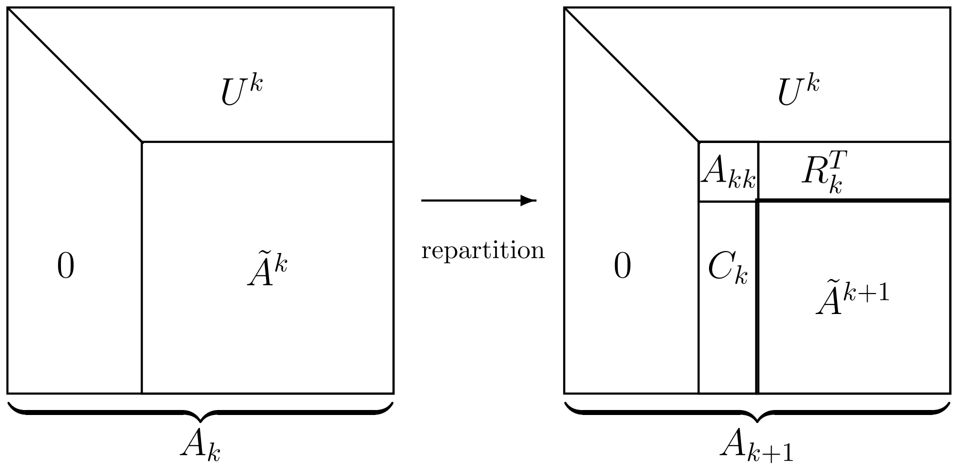
$$\begin{aligned} \text{pivot,} \quad & a_{kk} \\ \text{pivot row,} \quad & \underline{r}_k^T = a_{k:}, \text{ and} \\ \text{multiplier column,} \quad & \underline{c}_k = a_{:k}/a_{kk}, \end{aligned}$$

  the rank-one update step reads:

$$\tilde{A}^{k+1} \;=\; \tilde{A}^{k+1} \;-\; \underline{c}_k\, \underline{r}_k^T.$$

- The memory footprint of each successive submatrix is $(n-1)^2$, $(n-2)^2$, ... 1.
- This matrix must be pulled into cache $n-1$ times.
- The total number of memory references (of *non-cached* data) is $\approx \frac{1}{3}n^3$, and the total work $\approx \frac{2}{3}n^3$ ops (one "+" and "*" for each submatrix entry).
- Recall that non-cached memory accesses slow ($\approx 20\times$) compared to an `fma`.
- This observation suggests the idea of **block factorizations** that exploit `BLAS3` matrix-matrix products.
- This is the essential difference between LinPack and LaPack, with the latter being about $20\times$ faster.

Illustration of Block-Update:



- Here, A_{kk} is a $b \times b$ block, where $b \approx 64$ is the block size.
- In this case, the update step is

$$\tilde{A}^{k+1} \;=\; \tilde{A}^{k+1} \;-\; C_k\, A_{kk}^{-1} R_k^T.$$

- Since $A_{kk}^{-1} \;=\; (L_{kk}\, U_{kk})^{-1} \;=\; U_{kk}^{-1}\, L_{kk}^{-1}$, we can rewrite the update step as

$$\begin{aligned} R_k^T &= L_{kk}^{-1}\, R_k^T \\ C_k &= C_k U_{kk}^{-1} \\ \tilde{A}^{k+1} &= \tilde{A}^{k+1} \;-\; C_k\, R_k^T. \end{aligned}$$

- The advantage of the block strategy is that we reduce by a factor of b the number of times that $\tilde{A}^{k+1}$ is dragged into cache from main memory and that the principal work, computation of $C_k\, R_k^T$, is cast as a fast matrix-matrix product.

Matlab Code for LU, with and without Blocking:

```
function [L,U]=plu(A);

%  Unpivoted LU factorization

m=size(A,1);
n=size(A,2);
K=min(m,n);

U=A(1:K,:);
L=zeros(m,K);

for k=1:K;

   piv=U(k,k);             %% pivot
   row=U(k,k:end)';        %% pivot row
   col=U(k+1:end,k)/piv;   %% multiplier column

   U(k+1:end,k:end) = U(k+1:end,k:end)-col*row';

   L(k+1:end,k)     = col;
   L(k,k)           = 1;

end;
```

```
function [L,U]=blu(A,b);

%  Unpivoted Block-LU factorization
%  Blocksize = b

m=size(A,1);
n=size(A,2);
K=min(m,n);

U=A;
L=0*A;

for k=1:b:K; l=k+b-1; l=min(l,K);

   P=U(k:l,k:l);       [PL,PU] = plu(P); %% pivot
   R=U(k:l,k+b:end);  R=PL\R;           %% pivot row
   C=U(k+b:end,k:l);  C=C/PU;           %% multiplier column

   U(k+b:end,k+b:end)  = U(k+b:end,k+b:end) - C*R;

   U(k:l,k+b:end) = R;  U(k:l,k:l) = PU; U(k+b:end,k:l)=0;
   L(k+b:end,k:l) = C;  L(k:l,k:l) = PL;

end;
```

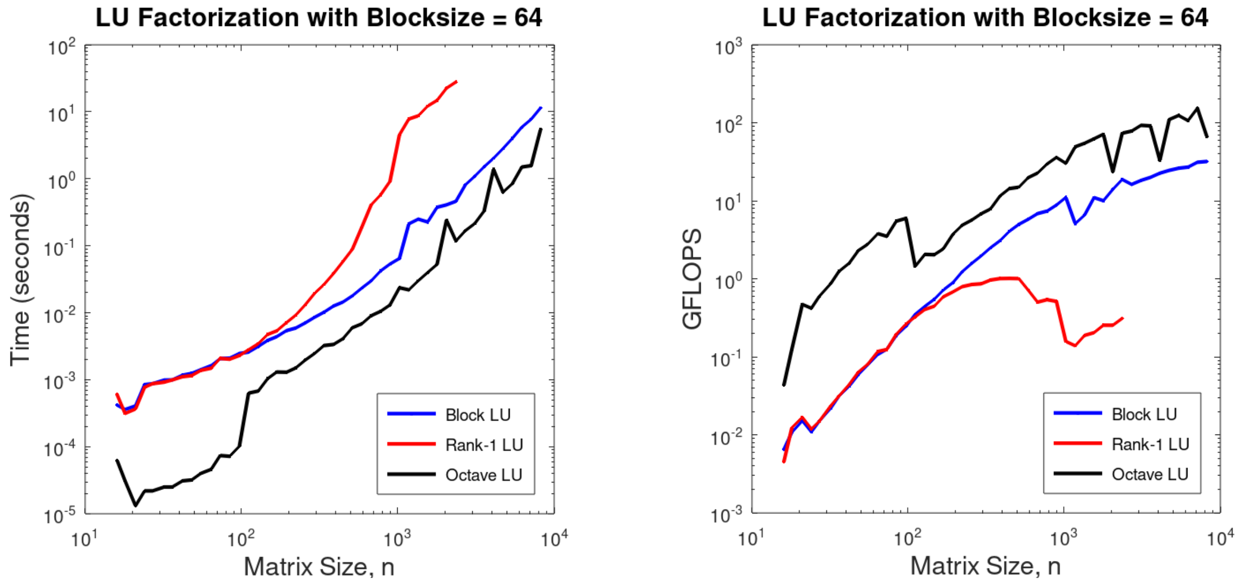


Figure 1: Time and GFLOPS for unblocked rank-1-based LU factorization (red) and blocked LU factorization with blockize $b = 64$ (blue) vs. matrix size, n. For large n, there is a $40\times$ difference in performance between Block-LU and Rank-1 LU. The default Octave LU gains another factor of 5 for large n, and a factor of 70 for $n < 100$. The results show that the dense-matrix factor times for $n = 8192$ are about 6 seconds for Octave when using multiple cores on an M1-based Macbook Pro.

- Importantly, the number of operations is $b(n-k)^2$ `fma`'s for the work-intensive matrix-matrix products, while the number of memory references is only $(n-k)^2$, which yields a b-fold increase in *computational intensity* (the ratio of flops to bytes).

- For this reason, LU factorization of large matrices can often realize close to the theoretical peak performance of a machine.

  (Some argue that this so-called Linpack performance number, which is used to score the machines in the Top 500 list, is inflated and artificial. Personally, I view it as an existence proof. The counter-argument is that vendors focus solely on the Linpack benchmark to the detriment of real applications.)

Banded Solves

- Banded system solves are common in PDE solvers and other systems where there is multidimensional locality.

- Saad provides the following definition:

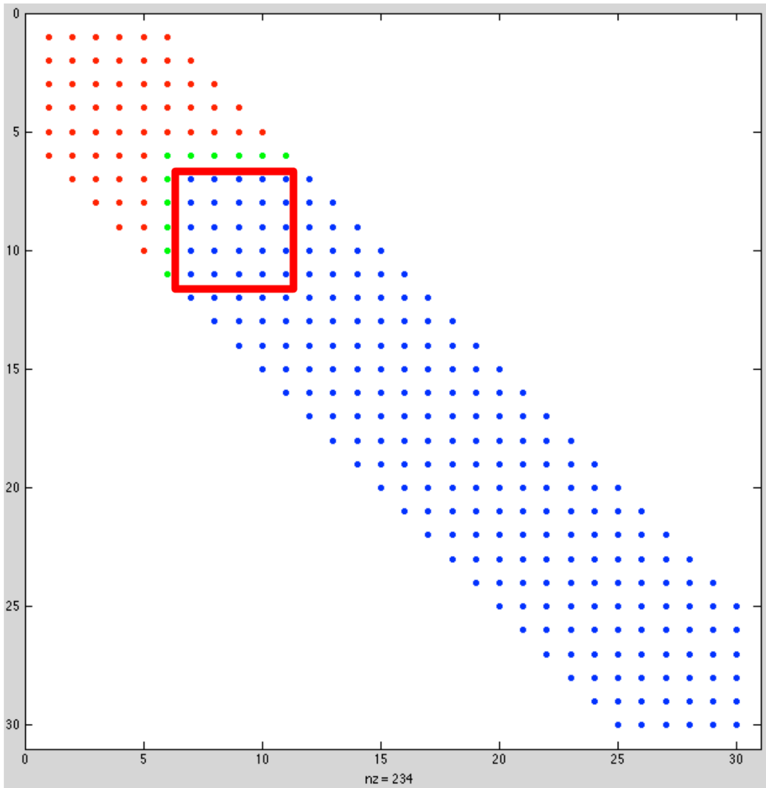
  Banded matrices: $a_{ij} \neq 0$ only if $i - m_l \leq j \leq i + m_u$, where m_l and m_u are two nonnegative integers.

  The number $m_l + m_u + 1$ is called the bandwidth of A.

- Frequently, $m_l = m_u$, even if A is not symmetric.

- We will use $b = m_u = m_l$ for the matrix bandwidth (or sometimes β), which is about half the value used by Saad.

The figure below illustrates the data layout for a banded matrix with matrix bandwidth b (=5, in this case).

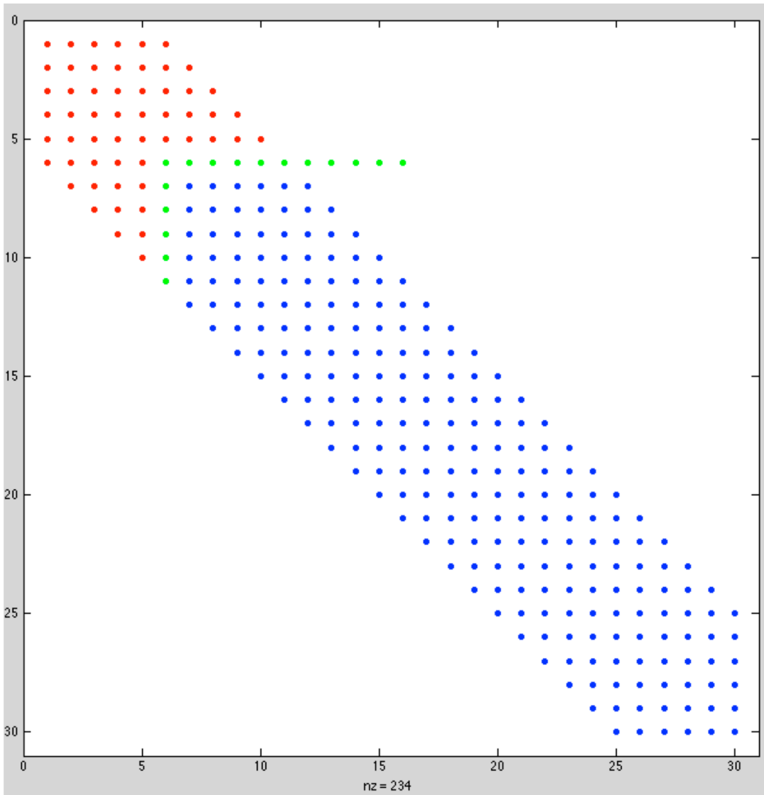


- Red indicates LU factors already computed.
- Green indicates the pivot, pivot row, and multiplier column.
- Blue is the section that remains to be factored.
- And the red box indicates the current active submatrix.

- **Q:** Assuming that we don't pivot, how much work is required to factor this banded matrix?

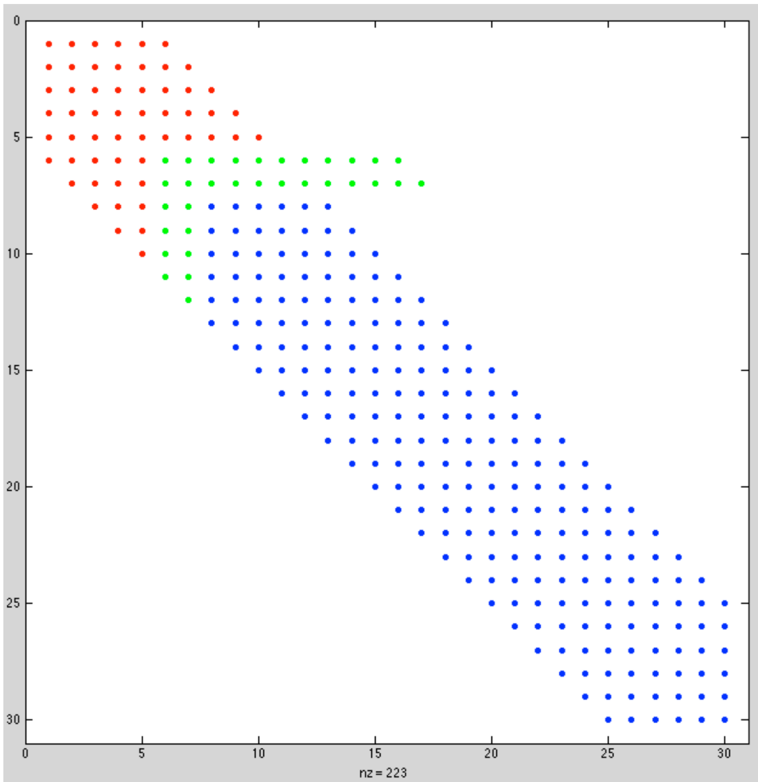
- Pivoting can pull a row that has $2b$ nonzeros to the right of the diagonal up into the pivot row.

- U can end up with bandwidth $2b$.



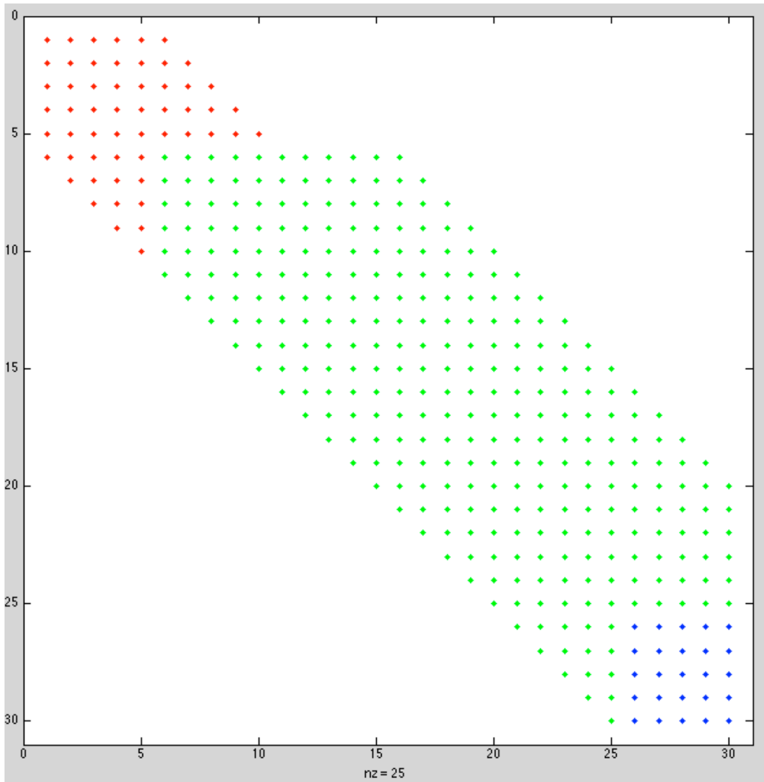
- Pivoting can pull a row that has $2b$ nonzeros to the right of the diagonal up into the pivot row.

- U can end up with bandwidth $2b$.



- Pivoting can pull a row that has $2b$ nonzeros to the right of the diagonal up into the pivot row.

- U can end up with bandwidth $2b$.



- Questions to think about:
  - What is the max storage required to solve a banded matrix with bandwidth b?
  - What is the work to compute the LU factors?
  - What is the work to solve the system, once L and U are known?

    The solve is executed as:

    $$\begin{aligned} Solve \quad L\underline{y} &= \underline{b} \\ Solve \quad U\underline{x} &= \underline{y} \end{aligned}$$

  - What is the cost of a tridiagonal solve?