Sparse block matrices in Matlab

- Constructing sparse block matrices
  - Sparse matrices
  - Block matrices
- Solving large sparse linear systems
  - LU factorization
  - Conjugate gradient

Sparse matrices

- sparse(m,n)
  - All zero sparse mxn matrix
- sparse(A)
  - Converts full matrix A to sparse
- speye(m,n)
  - Sparse matrix with ones on the main diagonal
- spalloc(m,n,nz)
  - Allocates storage for an mxn matrix with nz nonzero entries.
  - Since reallocation is expensive it is a good idea to allocate storage for a matrix before building it.

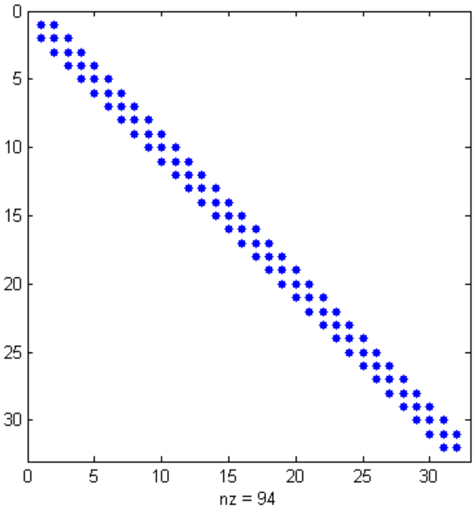
Sparse matrices

- spdiags(B, d, m, n)
  - Form a sparse mxn matrix whose diagonals, d, are the columns of B.
  - In d
    - 0 is the main diagonal
    - Positive values are super diagonals
    - Negative values are subdiagonals
- Example: second central difference matrix
  - e = ones(4,1);
  - A = spdiags([e, -2*e, e], [-1, 0, 1], 4, 4);

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}$$

Sparse matrices

- spy(A)
  - Visualize the sparsity structure of the matrix
- Example: 1D second central difference matrix
  - n = 32;
  - e = ones(n,1);
  - A = spdiags([e, -2*e, e], [-1, 0, 1], n, n);
  - spy(A);



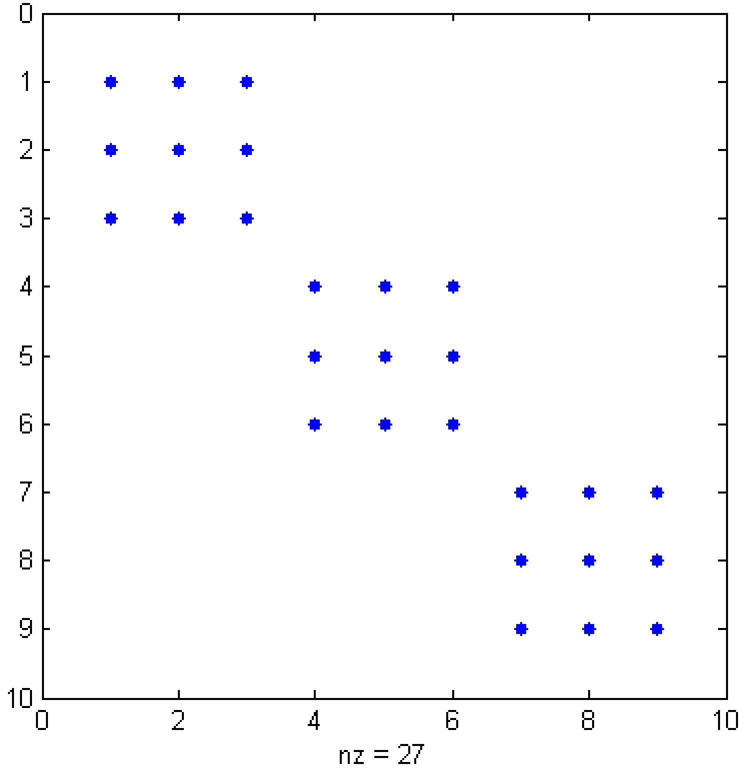
Block Matrices

- Sometimes it is useful to specify a matrix block-by-block.
- M = blkdiag(a,b,...)

$$M = \begin{bmatrix} a_{11} & \cdots & a_{1\mathrm{n}} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_{11} & \cdots & b_{1\mathrm{n}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

blkdiag example

- e = ones(3,3);
- A = blkdiag(e,e,e);
- spy(A);



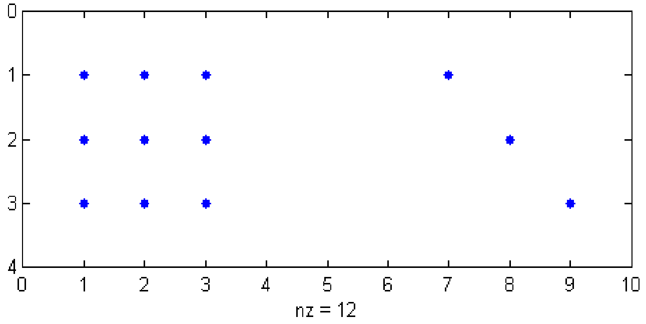
Matrix concatenation

- horzcat(a1, a2, a3,...)
  - Concatenate matrices horizontally
- vertcat(a1, a2, a3,...)
  - Concatenate matrices vertically

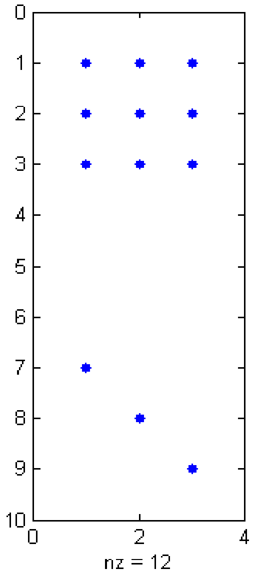
Matrix concatenation

- e = ones(3,3);
- z = zeros(3,3);
- I = eye(3,3);
- A = horzcat(e, z, I);
- B = vertcat(e, z, I);

- spy(A);



- spy(B);



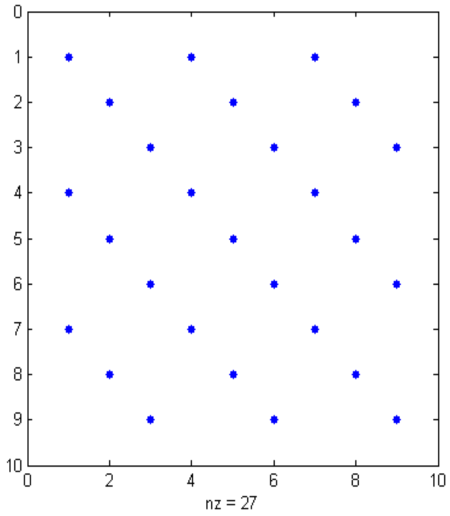
Kronecker tensor product

- K = kron(X,Y);
  - if X is mxn and Y is pxq then K is mp x nq

$$K = \begin{bmatrix} X_{11} Y & \cdots & X_{1\mathrm{n}} Y \\ \vdots & \ddots & \vdots \\ X_{n1} Y & \cdots & X_{nn} Y \end{bmatrix}$$

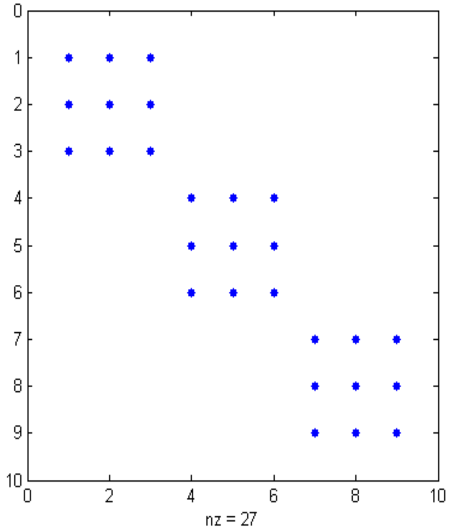
Kronecker tensor product example

- spy(kron(X,Y));



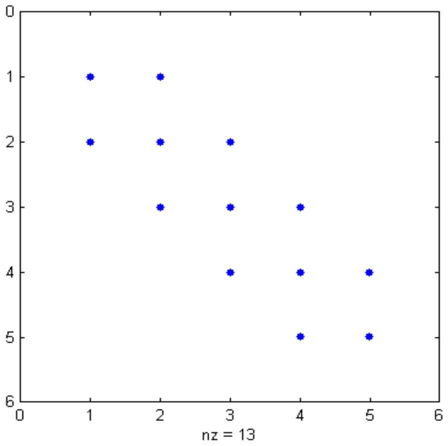
- X = ones(3,3);
- Y = eye(3,3);

- spy(kron(Y, X));

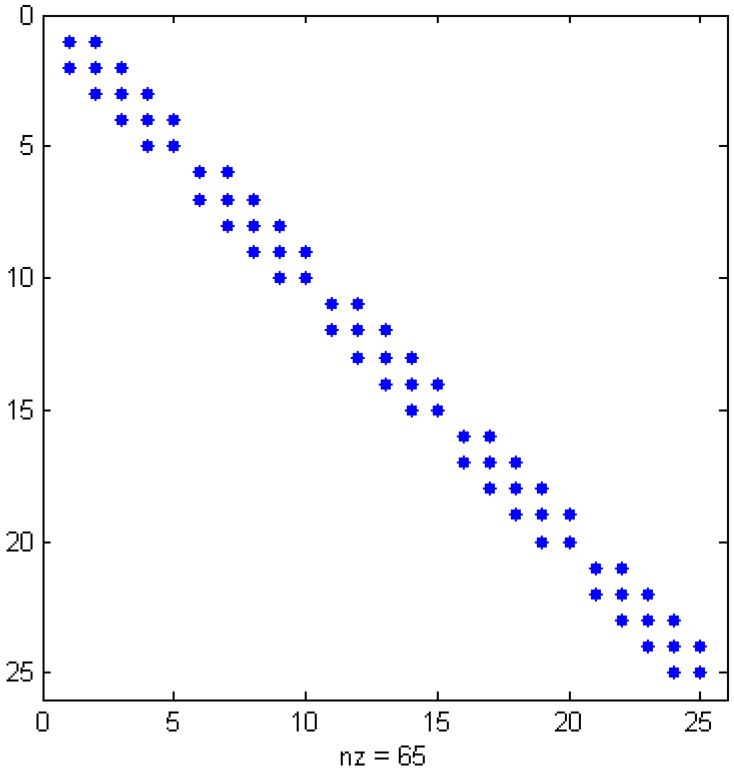


Using kron to create a 2D Laplacian matrix

- Boundary conditions:zeros outside image domain
- First create 1D second central difference matrix for x-direction
  - n1 = size(I,1);
  - e1 = ones(n1,1);
  - I1 = speye(n1, n1);
  - D1xx = spdiags([e1 -2*e1 e1], [-1 0 1], n1, n1);
  - spy(D1xx);



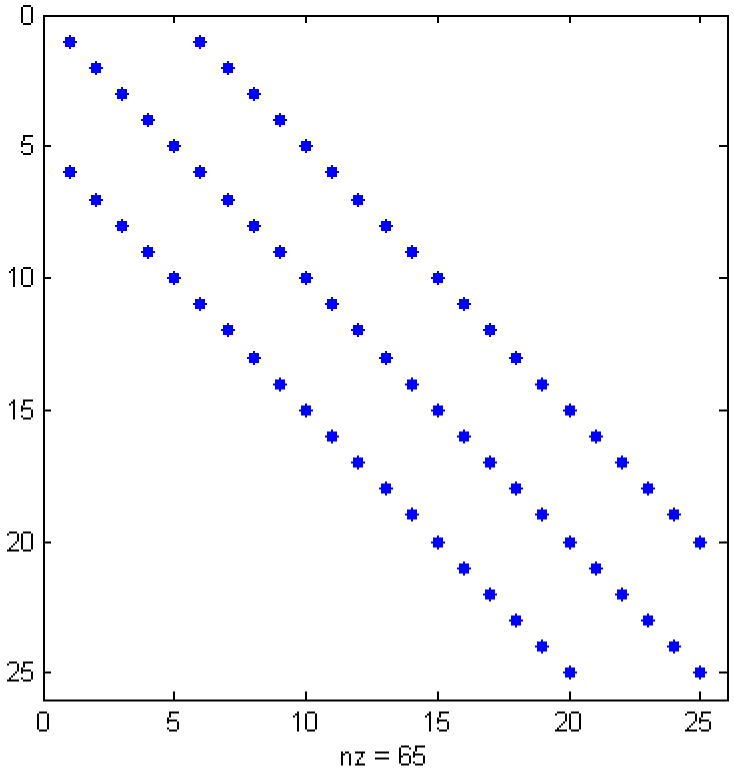
- Then create the 2D second central difference matrix
- I2 = speye(n2, n2);
- D2xx = kron(I2, D1xx);
- spy(D2xx);



Using kron to create a 2D Laplacian matrix

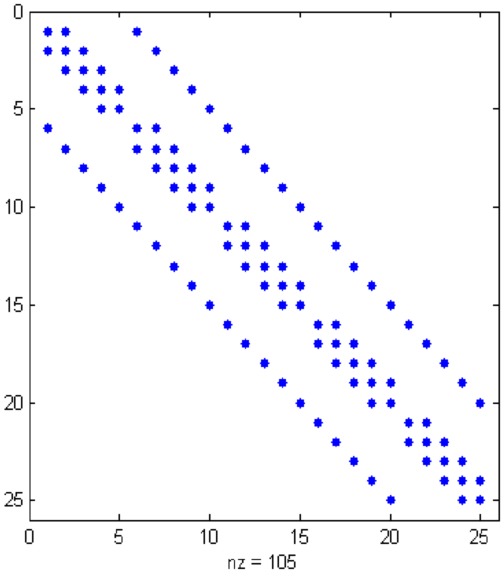
- Create 1D second central difference matrix for y-direction
  - n2 = size(I,1);
  - e2 = ones(n2,1);
  - I2 = speye(n2, n2);
  - D1yy = spdiags([e2, -2*e2 e2], [-1 0 1], n2, n2);
- Then create the 2D second central difference matrix
  - D2yy = kron(D1yy, I1);

- spy(D2yy);

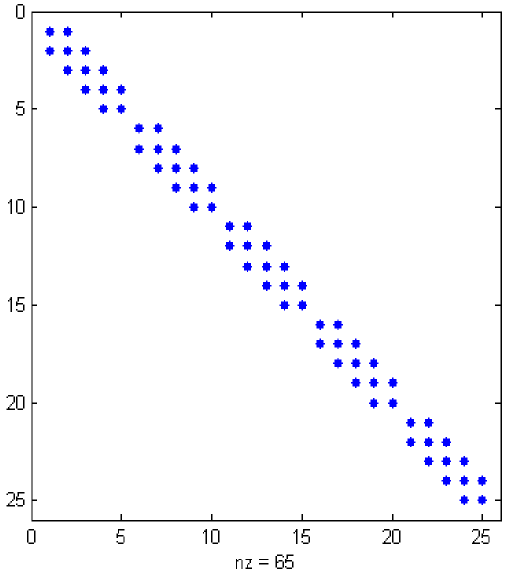


2D Laplacian Matrix

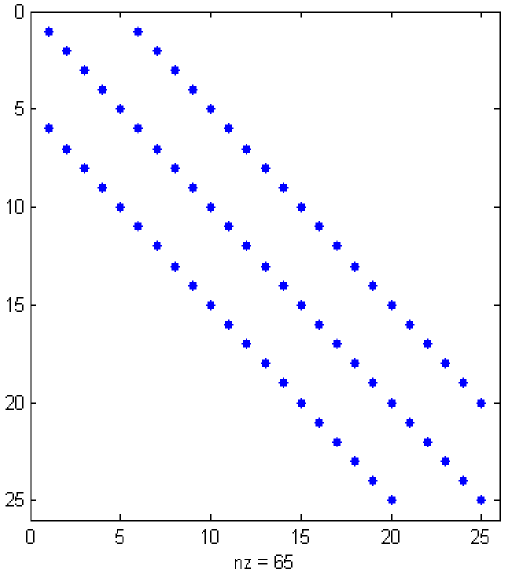
- Compute 2D Laplacian matrix
    - L = D2xx+D2yy;



=

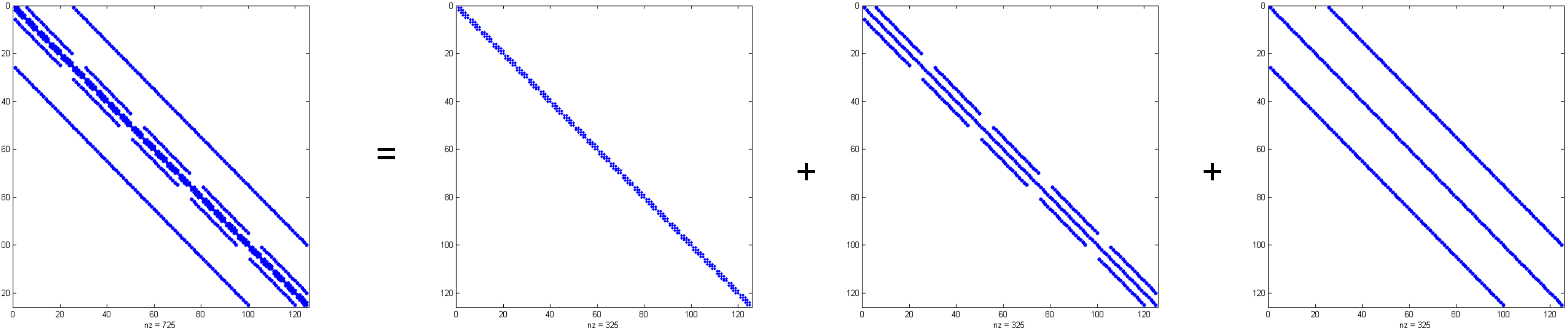


+



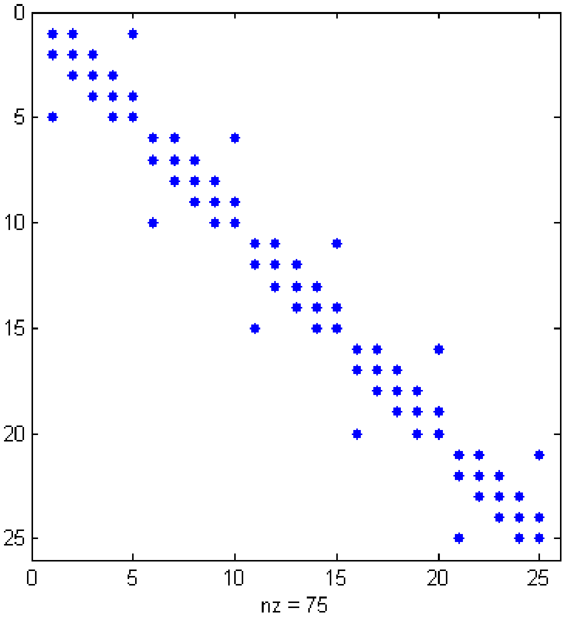
In 3D...

- D3xx = kron(I3, kron(I2, D1xx));
- D3yy = kron(I3, kron(D1yy, I1));
- D3zz = kron(kron(D1zz, I2), I1);
- L = D3xx+D3yy+D3zz

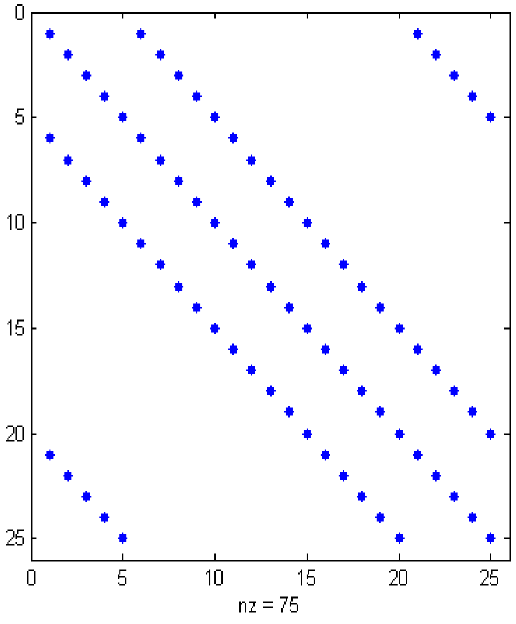


Imposing other boundary conditions

- Periodic boundary conditions
  - D1xx = D1xx + spdiags([e1 e1], [-n1+1 n1-1], n1, n1);
  - D2xx = kron(I2, D1xx);
  - D1yy = D1yy + spdiags([e2 e2], [-n2+1 n2-1], n2, n2);
  - D2yy = kron(D1yy, I1);



D2xx



D2yy

Solving linear systems

- Solve for x in $Ax=b$
- Inversion

$$x=A^{-1}b$$

  - Not practical for large or ill-conditioned matrices
- Other direct methods
  - LU factorization
- Iterative methods
  - Conjugate gradient (CG) methods

LU factorization

- This may be what happens when you type 'x = A\b' in Matlab
    - Check mldivide help for details
- LU decomposition is a form of Gaussian elimination
- Permits the linear system to be solved by back substitution
- If the matrix A does not change in every iteration you can factorize the matrix once, then only perform the back substitution each iteration

LU factorization

- A = LU
  - L is lower triangular (all superdiagonals are 0)
  - U is upper triangular (all subdiagonals are 0)

$$L=\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \qquad U=\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1\mathrm{n}} \\ 0 & u_{22} & u_{23} & \cdots & u_{2\mathrm{n}} \\ 0 & 0 & u_{33} & \ddots & u_{3\mathrm{n}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

  - For symmetric A you can find the Cholesky decomposition
    - A = LL^T

Solving by LU factorization

- Replace A with L times U

$$A\,x = b$$
$$L\,U\,x = b$$

- Solve in 2 steps
  - Let $y = U\,x$
  - Solve $Ly = b$
  - Then solve $Ux = y$

Solving triangular linear systems

- Easy, just back substitution
  - Proceed row-by-row
  - Solve for one unknown per row

$$L=\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

$$y_1=\frac{b_1}{l_{11}}$$

$$y_2=\frac{b_2-l_{21}y_1}{l_{22}}$$

...

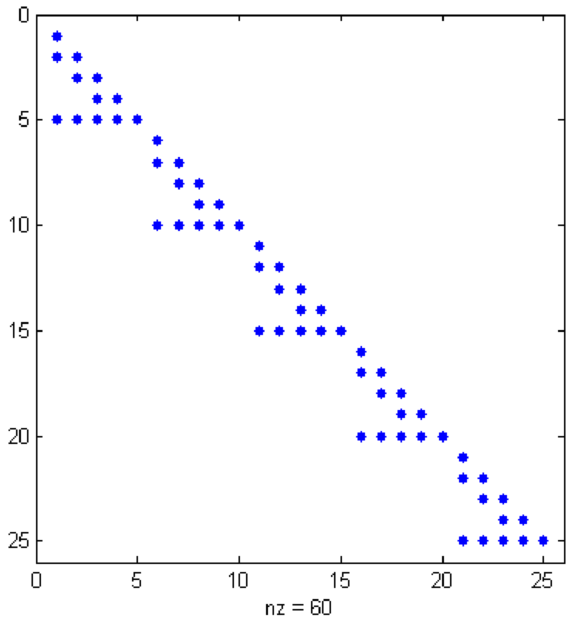
Solving by LU decomposition in Matlab

- To solve Ax = b
- Decompose
    - `[L,U] = lu(A);`
- Backsubstitute
    - `x = U\L\b;`

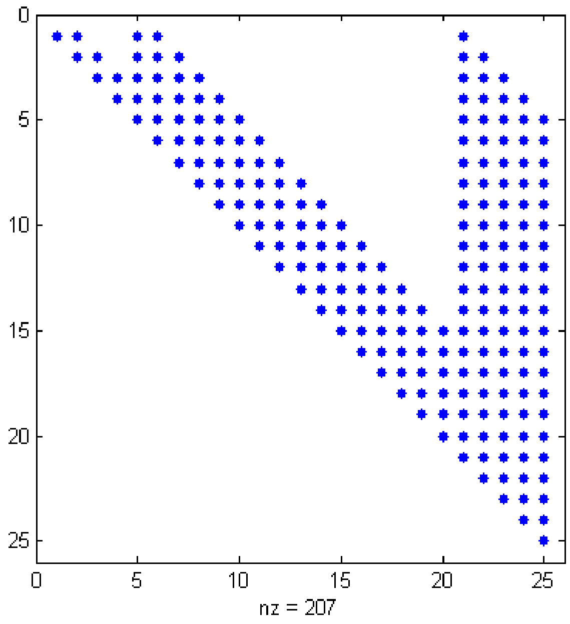
Sparse LU

- If A is sparse then L and U are usually sparse also
  - For the 2D Laplacian matrix:

L =



U =

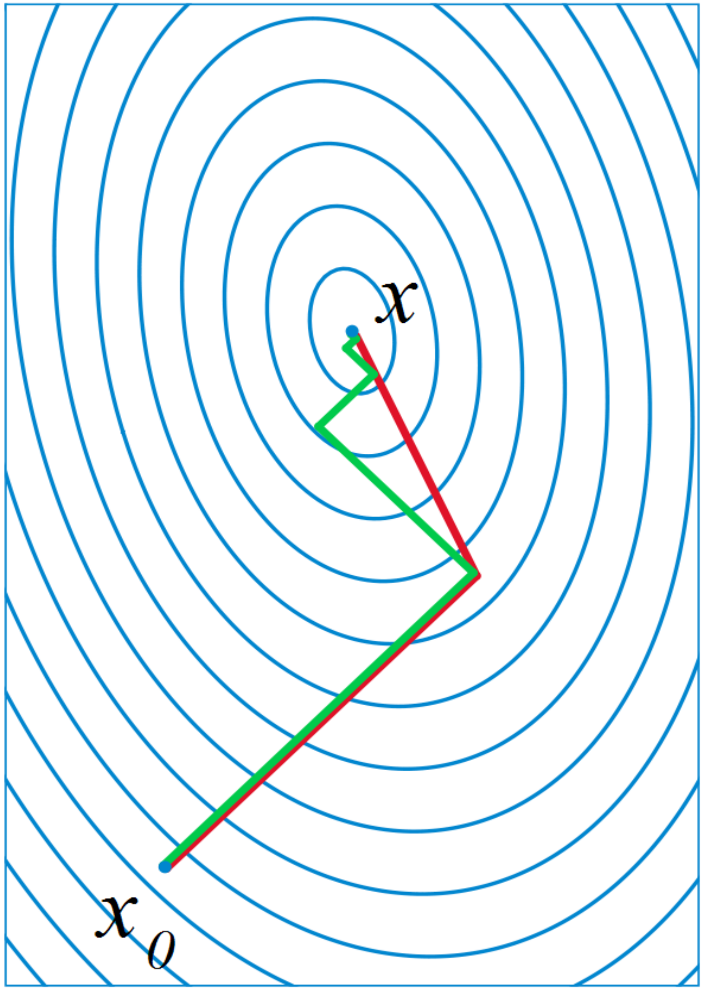


Conjugate gradient

- Iterative method
- Only requires matrix-vector multiplications, vector-vector operations
  - Can be very efficient when matrix is sparse.
- Can solve symmetric positive-definite systems
- See JR Shewchuk, “An introduction to the conjugate gradient method without the agonizing pain” for more details

Conjugate gradient

- Green lines: iterations of gradient descent.
    - Subsequent search directions , v, are perpendicular
    - $v_i^T v_{i+1} = 0$
- Red lines: iterations of conjugate gradient method.
    - In CG methods the search directions are conjugate
    - $v_i^T A v_{i+1} = 0$



Conjugate gradient variants in Matlab

- Preconditioned CG (symmetric A)
  - x = pcg(A,b,tol,maxit,M)
- Biconjugate gradients (square A – not req'd to be symmetric)
  - x = bicg(A,b,tol,maxit,M)
- CG squared (a variant of bicg)
  - x = cgs(A,b,tol,maxit,M)
- Biconjugate gradients stabilized method (another variant of bicg)
  - x = bicgstab(A,b,tol,maxit,M)

  - See Matlab help for details on differences in computational cost and convergence speed

Preconditioning

- The matrix M specified in the Matlab functions is a preconditioner

$$M^{-1} A x = M^{-1} b$$

- If A is ill-conditioned, choose M such that $M^{-1}A$ is well conditioned
- M must be symmetric and positive definite for PCG
- Ideally, $M^{-1} = A^{-1}$