unless you specify that it is sparse. You can convert a full matrix to sparse format using the command `A = sparse(A)`, and a sparse matrix to full format using the command `A = full(A)`.

When you type `x = A \ b` where A is $n \times n$, MATLAB chooses different algorithms depending on the type of A. If A is full it uses the standard methods for general matrices. If A is sparse, it uses an algorithm that takes advantage of sparsity. In our application, the matrix $I + \lambda D^T D$ is sparse (in fact tridiagonal), so if we make sure to define it as a sparse matrix, the normal equations will be solved much more quickly than if we ignore the sparsity.

The command to create a sparse zero matrix of dimension $m \times n$ is `A = sparse(m,n)`. The command `A = speye(n)` creates a sparse $n \times n$-identity matrix. If you add or multiply sparse matrices, the result is automatically considered sparse. This means you can solve the normal equations (26) by the following MATLAB code (assuming λ and x_{cor} are defined):

```
D = sparse(999,1000);
D(:,1:999) =  -speye(999);
D(:,2:1000) =  D(:,2:1000) + speye(999);
xhat = (speye(1000) + lambda*D'*D) \ xcor;
```

Solve the least squares problem (25) with the vector x_{cor} defined in `lsdenoising.m`, for three values of λ: $\lambda = 1$, $\lambda = 100$, and $\lambda = 10000$. Plot the three reconstructed signals $\hat{x}$. Discuss the effect of λ on the quality of the estimate $\hat{x}$.

9.4 *Regularized least squares image deblurring.* This exercise is on the image deblurring problem in §15.3.3 of the textbook. The purpose is to develop a fast method for solving the regularized least-squares problem on page 321:

$$\text{minimize } \|Ax - y\|^2 + \lambda(\|D_{\mathrm{v}}x\|^2 + \|D_{\mathrm{h}}x\|^2). \tag{27}$$

Notation. A black-and-white image of size $n \times n$ is represented as an $n \times n$ matrix X with X_{ij} the intensity of pixel i, j, or as an n^2-vector x. We use column-major order when converting a matrix X to a vector x:

$$x = \begin{bmatrix} X_{1:n,1} \\ X_{1:n,2} \\ \vdots \\ X_{1:n,n} \end{bmatrix}.$$

In MATLAB the conversion can be done by the command `x = X(:)` or, equivalently, `x = reshape(X, n^2, 1)`. To convert an n^2-vector x to an $n \times n$ matrix X we use `X = reshape(x, n, n)`.

In (27), the vectors x and y have length n^2. The vector y is given and represents an observed, noisy and blurred image Y of size $n \times n$. The variable x is the reconstructed $n \times n$ image X in vector form.

The following notation will be used to express the matrices A, D_{v}, D_{h} in (27). As in exercise 6.9, we use $T(v)$, with v an n-vector, to denote the $n \times n$ circulant matrix with v as its first column:

$$T(v) = \begin{bmatrix} v_1 & v_n & v_{n-1} & \cdots & v_3 & v_2 \\ v_2 & v_1 & v_n & \cdots & v_4 & v_3 \\ v_3 & v_2 & v_1 & \cdots & v_5 & v_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_{n-1} & v_{n-2} & v_{n-3} & \cdots & v_1 & v_n \\ v_n & v_{n-1} & v_{n-2} & \cdots & v_2 & v_1 \end{bmatrix}.$$

We also define $T(U)$ for an $n \times n$ matrix U. If the columns of U are u_1, u_2, ..., u_n, then $T(U)$ is the $n^2 \times n^2$

matrix

$$T(U) = \begin{bmatrix} T(u_1) & T(u_n) & T(u_{n-1}) & \cdots & T(u_3) & T(u_2) \\ T(u_2) & T(u_1) & T(u_n) & \cdots & T(u_4) & T(u_3) \\ T(u_3) & T(u_2) & T(u_1) & \cdots & T(u_5) & T(u_4) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ T(u_{n-1}) & T(u_{n-2}) & T(u_{n-3}) & \cdots & T(u_1) & T(u_n) \\ T(u_n) & T(u_{n-1}) & T(u_{n-2}) & \cdots & T(u_2) & T(u_1) \end{bmatrix}.$$

This structure is called *block-circulant with circulant blocks* (BCCB). Each block $T(u_i)$ is a circulant $n \times n$ matrix. The matrix $T(U)$ is block-circulant because each block column is a circular downward shift of the previous block column.

Image blurring. A linear, spatially invariant blurring operation is defined by a set of coefficients P_{kl}, with k and l integers ranging from $-r$ to r. We call P (or, more accurately, the function that maps a pair (k, l) to the coefficient P_{kl}) the *point spread function* (PSF). The integer r is the *width* of the PSF. We will assume that $2r + 1 \leq n$ (usually, $r \ll n$). The blurring operation defined by the PSF P is a two-dimensional convolution. It transforms an $n \times n$ image X into a blurred $n \times n$ image Y defined by

$$Y_{ij} = \sum_{k=-r}^{r} \sum_{l=-r}^{r} P_{kl} X_{i-k,j-l}, \quad i = 1, \ldots, n, \; j = 1, \ldots, n. \tag{28}$$

Note that the sum in this definition references some values of $X_{i-k,j-l}$ with indices outside the interval $[1, n]$. To fully specify the blurring operation we therefore need to make assumptions about the image X outside the frame. These assumptions are called *boundary conditions*. In this exercise we will use *periodic* boundary conditions. Periodic boundary conditions assume that the image X is repeated periodically outside the frame. One way to write this is to replace the convolution (28) with

$$Y_{ij} = \sum_{k=-r}^{r} \sum_{l=-r}^{r} P_{kl} \bar{X}_{i-k,j-l}, \quad i = 1, \ldots, n, \quad j = 1 \ldots, n, \tag{29}$$

where $\bar{X}$ denotes the larger, bordered image

$$\bar{X} = \begin{array}{|c|c|c|} \hline X & X & X \\ \hline X & X & X \\ \hline X & X & X \\ \hline \end{array}$$

and the indices i, j in $\bar{X}_{ij}$ run from $-n+1$ to $2n$. For example, if $n = 3$, $r = 1$, the convolution with periodic boundary conditions gives

$$\left[\begin{array}{c} Y_{11} \\ Y_{21} \\ Y_{31} \\ \hline Y_{12} \\ Y_{22} \\ Y_{32} \\ \hline Y_{13} \\ Y_{23} \\ Y_{33} \end{array}\right] = \left[\begin{array}{ccc|ccc|ccc} P_{00} & P_{-1,0} & P_{1,0} & P_{0,-1} & P_{-1,-1} & P_{1,-1} & P_{01} & P_{-1,1} & P_{11} \\ P_{10} & P_{00} & P_{-1,0} & P_{1,-1} & P_{0,-1} & P_{-1,-1} & P_{11} & P_{01} & P_{-1,1} \\ P_{-1,0} & P_{10} & P_{00} & P_{-1,-1} & P_{1,-1} & P_{0,-1} & P_{-1,1} & P_{11} & P_{01} \\ \hline P_{01} & P_{-1,1} & P_{11} & P_{00} & P_{-10} & P_{10} & P_{0,-1} & P_{-1,-1} & P_{1,-1} \\ P_{11} & P_{01} & P_{-1,1} & P_{10} & P_{00} & P_{-1,0} & P_{1,-1} & P_{0,-1} & P_{-1,-1} \\ P_{-1,1} & P_{11} & P_{01} & P_{-1,0} & P_{10} & P_{00} & P_{-1,-1} & P_{1,-1} & P_{0,-1} \\ \hline P_{0,-1} & P_{-1,-1} & P_{1,-1} & P_{01} & P_{-1,1} & P_{11} & P_{00} & P_{-1,0} & P_{10} \\ P_{1,-1} & P_{0,-1} & P_{-1,-1} & P_{11} & P_{01} & P_{-1,1} & P_{10} & P_{00} & P_{-1,0} \\ P_{-1,-1} & P_{1,-1} & P_{0,-1} & P_{-1,1} & P_{11} & P_{01} & P_{-1,0} & P_{10} & P_{00} \end{array}\right] \left[\begin{array}{c} X_{11} \\ X_{21} \\ X_{31} \\ \hline X_{12} \\ X_{22} \\ X_{32} \\ \hline X_{13} \\ X_{23} \\ X_{33} \end{array}\right].$$

(In this example, $2r + 1$ happens to be equal to n, while in general $2r + 1 \leq n$.)

We note that the matrix on the right-hand side is a BCCB matrix. More generally, if we write the convolution with periodic boundary conditions (29) in matrix form as $y = Ax$, then A is the $n^2 \times n^2$ matrix $A = T(B)$,

where B is the following $n \times n$ matrix constructed from the PSF coefficients.

$$B = \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0r} & 0 & \cdots & 0 & P_{0,-r} & \cdots & P_{0,-1} \\ P_{10} & P_{11} & \cdots & P_{1r} & 0 & \cdots & 0 & P_{1,-r} & \cdots & P_{1,-1} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ P_{r0} & P_{r1} & \cdots & P_{rr} & 0 & \cdots & 0 & P_{r,-r} & \cdots & P_{r,-1} \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ P_{-r,0} & P_{-r,1} & \cdots & P_{-r,r} & 0 & \cdots & 0 & P_{-r,-r} & \cdots & P_{-r,-1} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ P_{-1,0} & P_{-1,1} & \cdots & P_{-1,r} & 0 & \cdots & 0 & P_{-1,-r} & \cdots & P_{-1,-1} \end{bmatrix}.$$

As we will see, periodic boundary conditions are very convenient mathematically. Although the assumption of periodicity is not realistic, it is an acceptable approximation if $r \ll n$.

Vertical and horizontal differencing. Next we discuss the matrices D_{v} and D_{h} in (27). The vector $D_{\mathrm{v}}x$ is the image obtained by subtracting from X the image X shifted up over one pixel, so that

$$\|D_{\mathrm{v}}x\|^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(X_{ij} - X_{i+1,j})^2.$$

Here too, the sum references values of X_{ij} outside the frame (namely, the values $X_{n+1,j}$ are needed when $i = n$). To be consistent with the boundary conditions used for the blurring operation, we use periodic boundary conditions and assume $X_{n+1,j} = X_{1j}$. With this assumption, D_{v} is an $n^2 \times n^2$ matrix

$$D_{\mathrm{v}} = \begin{bmatrix} D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{bmatrix},$$

with all blocks of size $n \times n$ and D defined as the $n \times n$ matrix

$$D = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \\ -1 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}.$$

Note that D_{v} is a BCCB matrix: $D_{\mathrm{v}} = T(E)$ where E is the $n \times n$ matrix

$$E = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \\ -1 & 0 & \cdots & 0 \end{bmatrix}. \tag{30}$$

(The matrix E is zero, except for the elements 1, -1 in the first column.)

The last term $\|D_{\rm h}x\|^2$ in (27) is a penalty on the horizontal differences in the image: $D_{\rm h}$ is defined by

$$\|D_{\rm h}x\|^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(X_{ij} - X_{i,j+1})^2.$$

We again use periodic boundary conditions and define $X_{i,n+1} = X_{i1}$. This gives

$$D_{\rm h} = \begin{bmatrix} I & -I & 0 & \cdots & 0 & 0 & 0 \\ 0 & I & -I & \cdots & 0 & 0 & 0 \\ 0 & 0 & I & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & I & -I & 0 \\ 0 & 0 & 0 & \cdots & 0 & I & -I \\ -I & 0 & 0 & \cdots & 0 & 0 & I \end{bmatrix}.$$

This is a BCCB matrix: $D_{\rm h} = T(E^T)$ with E the matrix defined in (30).

Two-dimensional discrete Fourier transform. The two-dimensional DFT of a complex $n \times n$ matrix U is the complex $n \times n$ matrix

$$V = WUW, \tag{31}$$

where W is the DFT matrix (10). The inverse two-dimensional DFT maps an $n \times n$ matrix V to the matrix

$$U = \frac{1}{n^2}W^H V W^H. \tag{32}$$

In MATLAB the two-dimensional DFT and its inverse are implemented in the functions `fft2` and `ifft2`. We use `V = fft2(U)` to evaluate (31) and `U = ifft2(V)` to evaluate (32). The complexity is roughly $n^2 \log n$.

The two-dimensional DFT can also be interpreted as a matrix-vector product. Suppose u and v are the matrices U and V converted to n^2-vectors (in column-major order). Then the relation $V = WUW$ can be written as

$$v = \widetilde{W}u, \qquad \widetilde{W} = \begin{bmatrix} W_{11}W & W_{12}W & \cdots & W_{1n}W \\ W_{21}W & W_{22}W & \cdots & W_{2n}W \\ \vdots & \vdots & & \vdots \\ W_{n1}W & W_{n2}W & \cdots & W_{nn}W \end{bmatrix}. \tag{33}$$

The matrix $\widetilde{W}$ is $n^2 \times n^2$ and has n block rows and columns. The i,j block of $\widetilde{W}$ is $W_{ij}W$ where W_{ij} is the i,j element of W. (The operation that constructs $\widetilde{W}$ from W is known as a *Kronecker product* and written $\widetilde{W} = W \otimes W$.) Using (33) and the property $W^HW = (1/n)I$ one can show that

$$\widetilde{W}^H\widetilde{W} = n^2 I.$$

Therefore $\widetilde{W}^{-1} = (1/n^2)\widetilde{W}^H$ and $\widetilde{W}\widetilde{W}^H = n^2 I$.

It is important to keep in mind that we never need the matrix $\widetilde{W}$ explicitly, since we use the `fft2` and `ifft2` functions for the matrix-vector multiplications with $\widetilde{W}$. The product $v = \widetilde{W}u$ can be evaluated by combining `fft2` and the `reshape` function:

```
v = reshape( fft2( reshape( u, n, n ) ), n^2, 1);
```

Similarly, the matrix-vector product $u = \widetilde{W}^{-1}v = (1/n^2)\widetilde{W}^H v$, can be computed as

```
u = reshape( ifft2( reshape( v, n, n ) ), n^2, 1);
```

Finally, we need the two-dimensional counterpart of the factorization result for circulant matrices proved in exercise 6.9. The result is as follows. Suppose U is an $n \times n$ matrix and u is the corresponding n^2-vector (with the elements of U in column-major order). Then the $n^2 \times n^2$ BCCB matrix $T(U)$ can be factored as

$$T(U) = \frac{1}{n^2}\widetilde{W}^H \,\mathbf{diag}(\widetilde{W}u)\widetilde{W}. \tag{34}$$

Assignment.

(a) The normal equations for (27) are

$$(A^TA + \lambda D_{\mathrm{v}}^T D_{\mathrm{v}} + \lambda D_{\mathrm{h}}^T D_{\mathrm{h}})x = A^T y.$$

Assume $A = T(B)$, $D_{\mathrm{v}} = T(E)$, $D_{\mathrm{h}} = T(E^T)$ are BCCB matrices, where B is a given $n \times n$ matrix and E is the $n \times n$ matrix defined in (30). Use the factorization (34) to derive a fast algorithm (with order $n^2 \log n$ complexity) for solving the normal equations. Implement the fast algorithm in MATLAB or Octave.

(b) Download the file `deblur.mat` on the course website and load it in MATLAB or Octave (`load deblur`). The file contains two matrices Y and B. The matrix Y is the blurred image and can be displayed using the command `imshow(Y)`. The matrix B defines the blurring matrix $A = T(B)$. (The image is from the USC-SIPI Image Database at http://sipi.usc.edu/database.)

Test your deblurring code with several values of λ, plot the reconstructed image for each value (using `imshow(X)`), and determine the value of λ that gives the best result (visually, in your judgment). It is best to search for a good value of λ over a large range, by using a series of values evenly spaced on a logarithmic scale, for example, $\lambda = 10^{-6}, 10^{-5}, 10^{-4}, \ldots$.

Julia users will need the packages `FFTW`, `ImageView`, and `MAT`. The code

```
using MAT, ImageView
f = matopen("deblur.mat");
Y = read(f, "Y");
B = read(f, "B");
imshow(Y)
```

imports the matrices Y, B and displays the blurred image Y. The 2-dimensional DFT and inverse DFT are computed using the functions `fft` and `ifft` in the `FFTW` package, *i.e.*, the same functions as for the 1-dimensional DFT and inverse DFT (see homework 4). When applied to a matrix, `fft` and `ifft` compute the 2-dimensional DFT and inverse DFT.

9.5 Suppose A is an $m \times n$ matrix with columns $a_1, \ldots, a_n$, and $\tilde{A}$ is the $m \times n$ matrix with the de-meaned vectors $\tilde{a}_i = a_i - \mathbf{avg}(a_i)\mathbf{1}$ as its columns.

(a) Show that $\tilde{A} = HA$ where H is the $m \times m$ matrix $H = I - (1/m)\mathbf{1}\mathbf{1}^T$. Verify that $H^2 = H$.

(b) Let $\hat{t}$, $\hat{x}$ be the solution of the regularized least squares problem

$$\text{minimize} \quad \left\| \begin{bmatrix} \mathbf{1} & A \end{bmatrix} \begin{bmatrix} t \\ x \end{bmatrix} - b \right\|^2 + \lambda\|x\|^2$$

where $\lambda > 0$. The variables are the scalar t and n-vector x. Show that $\hat{x}$ is also the solution of the regularized least squares problem

$$\text{minimize} \quad \|\tilde{A}x - b\|^2 + \lambda\|x\|^2.$$

An application is the regularized data fitting problem of lecture 10, page 7. Excluding the constant feature from the regularization term (the first least squares problem in the assignment) is equivalent to a least squares problem with the centered data matrix and a regularization term on all parameters (the second problem in the assignment).