1 Introduction

The methods for solving matrix equations that we have seen already are called “direct” methods because they involve an algorithm that is guaranteed to terminate in a predictable number of steps. There are alternative methods, called “iterative” because they involve a repetitive algorithm that terminates when some specified tolerance is met. In this lab, we will focus on the conjugate gradient method applied to matrices arising from elliptic partial differential equations. The conjugate gradient method is often presented as a direct method because in exact arithmetic it terminates after \( n \) steps for an \( n \times n \) matrix. In computer arithmetic, however, the termination property is lost and is irrelevant in many cases because the iterates often converge in far fewer than \( n \) steps.

Iterative methods are most often applied to matrices that are “sparse” in the sense that their entries are mostly zeros. If most of the entries are zero, the matrices can be stored more efficiently than the usual form by omitting the zeros, and it is possible to take advantage of this efficient storage using iterative methods but not so easily with direct methods. Further, since partial differential equations often give rise to large sparse matrices, iterative methods are often used to solve systems arising from PDEs.

You will see the conjugate gradient method (regarded as an iterative method), and will deploy it using matrices stored in their usual form and also in compact form. You will see examples of the rapid convergence of the method and you will solve a matrix system stored in compact form that is so large that you could not even construct and store the matrix in its usual form in central memory, let alone invert it.

You may find it convenient to print the pdf version of this lab rather than the web page itself. This lab will take three sessions.

2 Poisson equation matrices

In two dimensions, the Poisson equation can be written as

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho
\]

where \( u \) is the unknown function and \( \rho \) is a given function. The simplest discretization of this equation is based on the finite difference expression for a second derivative

\[
\frac{d^2 \phi}{d\xi^2} = \frac{\phi(\xi + \Delta \xi) - 2\phi(\xi) + \phi(\xi - \Delta \xi)}{\Delta \xi^2} + O(\Delta \xi)
\]

Suppose that the unit square \([0,1] \times [0,1]\) is broken into \((N+1)^2\) smaller, nonoverlapping squares, each of side \( h = 1/(N+1) \). Each of these small squares will be called “elements” and their corner points will be called “nodes.” The combination of all elements making up the unit square will be called a “mesh.” An example of a mesh is shown below.
The nodes in Figure 1 have coordinates \((x, y)\) given by
\[
\begin{align*}
x &= jh & \text{for } j = 0, 1, \ldots, 11 \\
y &= kh & \text{for } k = 0, 1, \ldots, 11
\end{align*}
\] (3)

You will be generating the matrix associated with the Poisson equation during this lab. Using (2) twice in (1) at a mesh node \((x, y)\) yields the expression
\[
\frac{u(x + h, y) + u(x, y + h) + u(x - h, y) + u(x, y - h) - 4u(x, y)}{h^2} = \rho(x, y).
\]

Denoting the point \((x, y)\) as the “center” point, the point \((x + h, y)\) as the “right” point, \((x - h, y)\) as the “left” point, \((x, y + h)\) as the “above” point, and \((x, y - h)\) as the “below” point yields the expression
\[
\frac{1}{h^2}u_{\text{right}} + \frac{1}{h^2}u_{\text{left}} + \frac{1}{h^2}u_{\text{above}} + \frac{1}{h^2}u_{\text{below}} - 4\frac{1}{h^2}u_{\text{center}} = \rho_{\text{center}}.
\] (4)

Some authors denote these five points as “center”, “north”, “east”, “south”, and “west”, referring to the compass directions.

The matrix equation (system of equations) can be generated from (4) by numbering the nodes in some way and forming a vector from all the nodal values of \(u\). Then, (4) represents one row of a matrix equation.
\( \Pi u = \rho \). It is immediate that there are at most five non-zero terms in each row of the matrix \( \Pi \), no matter what the size of \( N \). The diagonal term is \(-\frac{1}{h^2}\), the other four, if present, are each equal to \( \frac{1}{h^2} \), and all remaining terms are zero. We will be constructing such a matrix in Matlab during this lab.

The equation in the form (4) yields the “stencil” of the differential matrix, and is sometimes illustrated graphically as in Figure 2 below. Other stencils are possible and result in other difference schemes.

In order to construct matrices arising from the Poisson differential equation, it is necessary to give each node at which a solution is desired a number. We will be considering only the case of Dirichlet boundary conditions (\( u = 0 \) at all boundary points), so we are only interested in interior nodes as illustrated in Figure 1. These nodes will be counted consecutively, starting with 1 on the lower left and increasing up and to the right to 100 at the upper right, as illustrated in Figure 3.

**Exercise 1:**

(a) Write a script m-file, named `meshplot.m` that, for \( N=10 \), generates two vectors, \( x(1:N^2) \) and \( y(1:N^2) \) so that the points with coordinates \((x(n), y(n))\) for \( n=1, \ldots, N^2 \) are the interior node points illustrated in Figures 1, 2 and 3 and defined in (3). You can use the following outline

\[
N=10; \\
h=1/(N+1); \\
\]
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{node_numbering.png}
\caption{Node numbering}
\end{figure}

\begin{verbatim}
n=0;
\% initialize x and y as column vectors
x=zeros(N^2,1);
y=zeros(N^2,1);
for j=1:N
    for k=1:N
        \% xNode and yNode should be between 0 and 1
        xNode=???
yNode=???
n=n+1;
x(n)=xNode;
y(n)=yNode;
    end
end
plot(x,y,'o');
\end{verbatim}

(b) Send me your plot. It should look similar to Figure 2, but without the stencil.

(c) The following function m-file will print, for \textbf{n} and \textbf{N} given, the index (subscript) value associated
with the point immediately above the given point, or the word “none” if there is no such point.

```
function tests(n)
% function tests(n) prints index numbers of points near the
% one numbered n

N=10;
if mod(n,N)~=0
    nAbove=n+1
else
    nAbove='none'
end

Copy this code to a file named tests.m and test it by applying it with n=75 to find the number of the point above the one numbered 75. Test it also for n=80 to see that there is no point above the one numbered 80.

(d) Add similar code to print the index nBelow of the point immediately below the given point. Use it to find the number of the point below the one numbered 75, and also apply it to a point that has no interior point below it.

(e) Add similar code to print the index nLeft of the point immediately to the left of the given point. Use it to find the number of the point to the left of the one numbered 75, and also apply it to a point that has no interior point to its left.

(f) Add similar code to print the index nRight of the point immediately to the right of the given point. Use it to find the number of the point to the right of the one numbered 75, and also apply it to a point that has no interior point to its right.

You are now ready to construct a matrix A derived from the Poisson equation with Dirichlet boundary conditions on a uniform mesh. In this lab we are considering only one of the infinitely many possibilities for such a matrix. The matrix Π defined above turns out to be negative definite, so the matrix A we will use in this lab is its negative, \( A = -\Pi \)

**Exercise 2:**

(a) Recall that there are no more than five non-zero entries in the matrix A and that they are the values \(4/h^2\), and \(-1/h^2\) repeated four times. (Recall that A is the negative of the matrix Π above.) Use the following skeleton, and your work from the previous exercise, to write a function m-file to generate the matrix A.

```
function A=poissonmatrix(N)
% A=poissonmatrix(N)
% more comments

% your name and the date

h=1/(N+1);
A=zeros(N-2,N-2);
for n=1:N-2
    % center point value
    A(n,n) = 4/h^2;

    % "above" point value
    if mod(n,N)~=0
        nAbove=n+1;
    end

    % "below" point value
    if mod(n,N)~=N
        nBelow=n-1;
    end

    % "left" point value
    if n>1
        nLeft=n-1;
    end

    % "right" point value
    if n2
        nRight=n+1;
    end

end
```
A(n,nAbove)=-1/h^2;
end

% "below" point value
if ???
  nBelow= ???;
  A(n,nBelow)=-1/h^2;
end

% "right" point value
if ???
  nRight= ???;
  A(n,nRight)=-1/h^2;
end

% "left" point value
if ???
  nLeft= ???;
  A(n,nLeft)=-1/h^2;
end
end

In the following parts of the exercise, take N=10.

(b) It may not be obvious from our construction, but the matrix A is symmetric. Consider A(k,ell) for k=n and ell=nBelow, and also for k=nBelow and ell=n (n is “above” nBelow). Think about it. Check that norm(A-A','fro') is essentially zero. If it is not zero, follow these instructions to fix your poissonmatrix function.

i. Set nonSymm=A-A', and choose one set of indices k,ell for which nonSymm is not zero. You can use the variable browser (double-click on nonSymm in the “variables” pane) or the max function in the following way:
   [row]=max(nonSymm);
   [col,ell]=max(row);
   [maxval,k]=max(nonSymm(:,ell));

ii. Look at A(k,ell) and A(ell,k). The most likely case is that one of these is zero and the other is not. Use your tests.m code to check which of them should be non-zero, and then look to see why the other is not.

(c) The Gershgorin circle theorem (see, for example, your class notes or the interesting Wikipedia article http://en.wikipedia.org/wiki/Gershgorin_circle_theorem) says that the eigenvalues of a matrix A are contained in the union of the disks in the complex plane centered at the points z_n = A_{n,n}, and having radii r_n = \sum_{k \neq n} |A_{n,k}|. By this theorem, our matrix A is non-negative definite. A more delicate proof exists that A is positive definite. For a test vector v=rand(100,1), compute v'*A*v. This quantity should be positive. If it is not, fix your poissonmatrix function.

(d) Positive definite matrices must have positive determinants. Check that det(A) is positive. If it is not, fix your poissonmatrix function.

(e) A simple calculation shows that the functions

\[ e_{k,\ell}(x, y) = \sin(k \pi x) \sin(\ell \pi y) \]
are eigenfunctions of the Dirichlet problem for Poisson’s equation on the unit square for each choice of $k$ and $\ell$, with eigenvalues $\lambda_{k,\ell} = -(k^2 + \ell^2)\pi^2$. It turns out that, for each value of $k$ and $\ell$, the vector $E = \sin(k\pi x) \cdot \sin(\ell\pi y)$ is also an eigenvector of the matrix $A$, where $x$ and $y$ are the vectors from Exercise 1. The eigenvalue is $L = 2(2 - \cos(k\pi h) - \cos(\ell\pi h))/h^2$, where $h = y(2) - y(1)$. Choose $k = 1$ and $\ell = 2$ and show that $L \cdot E = A \cdot E$. If this is not true for your matrix $A$, go back and fix $\text{poissonmatrix}$.

(f) It can be shown that $A$ is an “M-matrix” (see, for example, Varga, *Matrix Iterative Analysis*) because the off-diagonal entries are all negative (or are zero) and the diagonal entry exceeds the negative of the sum of the off-diagonal entries. It turns out that M-matrices have inverses all of whose entries are strictly positive. Use Matlab’s $\text{inv}$ function to find the inverse of $A$ and show that its smallest entry (and hence each entry) is positive. (It might be small, but it must be positive.)

(g) Finally, you can check your construction against Matlab itself. The Matlab function $\text{gallery}('\text{poisson}', N)$ returns a matrix that is $h^2 \cdot A$. Compute $\text{norm}(\text{gallery}('\text{poisson}', N)/h^2 - A, 'fro')$ to see if it is essentially zero. (Note: The $\text{gallery}$ function returns the Poisson matrix in a special compressed form that is different from the usual form and also not the same as the one discussed later in this lab.) If this test fails, fix $\text{poissonmatrix}$.

You now have a first good matrix to use for the study of the conjugate gradient iteration in the following section. In the following exercise you will generate another.

**Exercise 3:**

(a) Copy your to $\text{poissonmatrix.m}$ another one called $\text{anothermatrix.m}$ (or use “Save As” on the “File” menu.)

(b) Modify the comments and some statements in the following way:

\[
A(n,n) = 8*n; \\
A(n,nAbove) = -2*n-1; \\
A(n,nBelow) = -2*n+1; \\
A(n,nRight) = -2*n-N; \\
A(n,nLeft) = -2*n+N;
\]

This matrix has the same structure as that from $\text{poissonmatrix}$ but the values are different.

(c) This matrix is also symmetric. Check that, for $N=12$, $\text{norm}(A-A', 'fro')$ is essentially zero. **Debug hint:** If this norm is *not* essentially zero for $N=12$, but *is* essentially zero for $N=10$, then you have the number 10 coded in your function somewhere. Try searching for “10”. (If you have that error in $\text{anothermatrix.m}$ you probably have the same error in $\text{poissonmatrix.m}$.)

(d) This matrix is positive definite, just as the Poisson matrix is. For $N=12$ and one test vector $v = \text{rand}(N^2,1)$; check that $v' \cdot A \cdot v$ is positive.

(e) Plot the structure of the matrices and see they are the same, using the following commands

```
spy(poissonmatrix(12), 'b*'); % blue spots
hold on
spy(anothermatrix(12), 'y*'); % yellow spots cover all blue
spy(poissonmatrix(12), 'r*'); % red spots cover all yellow
```

Please include this plot with your summary.

(f) What is the result of the following calculation?

\[
x = ((1:N^2)^2)'; \\
\text{sum}(A\times x);
\]
The conjugate gradient method is an important iterative method and one of the first examples of methods based on Krylov spaces. Krylov spaces are vector spaces spanned by the set \( \{ x, Ax, A^2x, \ldots \} \) for some choice of \( x \) and \( A \).

One very important point to remember about the conjugate gradient method (and other Krylov methods) is that only the matrix-vector product is required. One can write and debug the method using ordinary matrix arithmetic and then apply it using other storage methods, such as the compact matrix storage used below.

A second very important point to remember is that the conjugate gradient algorithm requires the matrix to be positive definite. You will note that the algorithm below contains a step with \( p^{(k)} \cdot Ap^{(k)} \) in the denominator. This quantity is guaranteed to be nonzero only if \( A \) is positive (or negative) definite. If \( A \) is negative definite, multiply the equation through by (-1) as we have done with the Poisson equation.

The conjugate gradient algorithm can be described in the following way. Starting with an initial guess vector \( x^{(0)} \),

\[
\begin{align*}
r^{(0)} &= b - Ax^{(0)} \\
\text{For } k = 1, 2, \ldots, m \\
\rho_{k-1} &= r^{(k-1)} \cdot r^{(k-1)} \\
\text{if } \rho_{k-1} \text{ is zero, stop: the solution has been found.} \\
\text{if } k \text{ is 1} \\
p^{(1)} &= r^{(0)} \\
\text{else} \\
\beta_{k-1} &= \rho_{k-1} / \rho_{k-2} \\
p^{(k)} &= r^{(k-1)} + \beta_{k-1}p^{(k-1)} \\
q^{(k)} &= Ap^{(k)} \\
\gamma_k &= p^{(k)} \cdot q^{(k)} \\
\text{if } \gamma_k \text{ is zero, stop: the algorithm has failed.} \\
\alpha_k &= \rho_{k-1} / \gamma_k \\
x^{(k)} &= x^{(k-1)} + \alpha_kp^{(k)} \\
r^{(k)} &= r^{(k-1)} - \alpha_kq^{(k)}
\end{align*}
\]

This algorithm is essentially the same as the one given in class, where the vectors \( p \) are called \( d \). (The two "if" tests guard against dividing by zero.)

In the following exercise you will first write a function to perform the conjugate gradient algorithm. You will test it without using hand calculations.

Exercise 4:

(a) Write a function m-file named `cgm.m` with signature

\[
\text{function } x = \text{cgm}(A, b, x, m)
\]

\[
\% x = \text{cgm}(A, b, x, m)
\]

\[
\% more comments
\]

\[
\% your name and the date
\]

that performs \( m \) iterations of the conjugate gradient algorithm for the problem \( A*x = b \), with starting vector \( x \). Do not create vectors for the variables \( \alpha_k, \beta_k, \) and \( \rho_k \), instead use simple Matlab scalars \( \alpha, \beta, \rho_{k1}, \) and \( \rho_{k2} \). Similarly, denote the vectors \( r^{(k)}, p^{(k)}, q^{(k)}, \) and \( x^{(k)} \) by the Matlab variables \( r, p, q, \) and \( x \), respectively. For example, part of your loop might look like
rhoKm1=0;
for k=1:m
    rhoKm2=rhoKm1;
    rhoKm1=dot(r,r);
    ???
    r=r-alpha*q;
end

(b) Use N=5, b=(1:N^2)', A=poissonmatrix(N), as initial guess, use the perturbed exact solution x=(A\b)+500*eps*rand(N^2,1), and take one step. Call the result y. You should get that norm(y-x) is very small. (If you cannot converge when you start from the exact solution, you will never converge!) (If your result is not small, look at your code. \( \rho_{k-1} \) should be very small, and \( \gamma_k \) should be larger than \( \rho_{k-1} \), and \( \alpha_k \) should also be very small. The resulting \( \|r(k)\| \) should be very small.)

(c) Since the set of vectors \( \{x, Ax, A^2x, \ldots, A^{M-1}x\} \) span the whole space, the conjugate gradient algorithm terminates after \( M=N^2 \) steps on an \( M \times M \) matrix. As a second test, choose \( N=5 \), \( x_{\text{Exact}}=\text{rand}(N^2,1) \) and \( A=\text{poissonmatrix}(N) \). Set \( b=A*x_{\text{Exact}} \) and then use your function, starting from the zero vector, to solve the system in \( N^2 \) steps. If you call your solution \( x \), show that \( \text{norm}(x-x_{\text{Exact}}) \) is essentially zero.

(d) You know that your routine solves a system of size \( M \) in \( M \) steps. Choose \( x_{\text{Exact}}=\text{rand}(625,1) \), the matrix \( A=\text{poissonmatrix}(25) \), and \( b=A*x_{\text{Exact}} \). How close would the solution \( x \) calculated using \( \text{cgm} \) on \( A \) be if you started from the zero vector and took only one hundred steps? (Use the relative 2-norm.)

The point of the last part of the previous exercise is that conjugate gradients serves quite well as an iterative method, especially for large systems. Regarding it as an iterative method, however, requires some way of deciding when to stop before doing \( M \) iterations on an \( M \times M \) matrix. Assuming that \( \|b\| \neq 0 \) a reasonable stopping criterion is when the relative residual error \( \|b - Ax^{(k)}\|/\|b\| \) is small. (If \( \|b\| = 0 \), then \( b \) is trivial, and so is the solution.) Without an estimate of the condition number of \( A \), this criterion does not bound the solution error, but it will do for our purpose. A simple induction argument will show that \( r^{(k)} = b - Ax^{(k)} \), so it is not necessary to perform a matrix-vector multiplication to determine whether or not to stop the iteration. Conveniently, the quantity \( \rho_{k-1} \) that is computed at the beginning of the loop is the square of the norm of \( r^{(k-1)} \), so the iteration can be terminated \textit{without any extra calculation}.

In the following exercise, you will modify your \text{cgm.m} file by eliminating the parameter \( m \) in favor of a specified tolerance on the relative residual.

Exercise 5:

(a) Make a copy of your \text{cgm.m} function and call it \text{cg.m}. Change its name on the signature line, but leave the parameter \( m \) for the moment. Modify it by computing \( \|b\| \) and calling it \text{normBsq}, before the loop starts. Then, \textit{temporarily} add a statement following the computation of \text{rhoKm1} to compute the quantity \text{relativeResidual=sqrt(rhoKm1/normBsq)}. Follow it with the commands \text{semilogy(k,relativeResidual,'*');hold on}; so that you can watch the progression of the relative residual.

(b) As in the last part of the previous exercise, choose \( x_{\text{Exact}}=\text{rand}(625,1) \), the matrix \( A=\text{poissonmatrix}(25) \), and \( b=A*x_{\text{Exact}} \). Run it for 200 iterations starting from \text{zeros}(625,1). You will have to use the command \text{hold off} after the function is done. Please include the plot with your summary. You should observe uneven but rapid convergence to a small value long before 200 < 625 iterations. Estimate from the plot how many iterations it would take to meet a convergence criterion of \( 10^{-10} \).

(c) Change the signature of your \text{cg} function to
\[ [x,k] = \text{cg}(A,b,x,\text{tolerance}) \]

so that \( m \) is no longer input. Replace the value \( m \) in the \texttt{for} statement with \texttt{size(A,1)} because you know convergence should be reached at that point. Compute \( \text{targetValue} = \text{tolerance}^2 \times \| \text{normBsq} \| \). Replace the \texttt{semilogy} plot with the exit commands

\[
\text{if } \rho_{k+1} < \text{targetValue} \\
\text{return} \\
\text{end}
\]

There is no longer any need for \texttt{relativeResidual}.

(d) As in the last part of the previous exercise, choose \( \text{xExact} = \text{rand}(625,1) \), the matrix \( A = \text{poissonmatrix}(25) \), and \( b = A \times \text{xExact} \). Start from \( \text{zeros}(625,1) \). Run your changed \texttt{cg} function with a tolerance of \( 1.0 \times 10^{-10} \). How many iterations does it take? Run it with a tolerance of \( 1.0 \times 10^{-14} \). How many iterations does it take this time?

(e) To see how another matrix might behave, choose \( \text{xExact} = \text{rand}(625,1) \), the matrix \( A = \text{anothermatrix}(25) \), and \( b = A \times \text{xExact} \). How many iterations does it take to reach a tolerance of \( 1.e^{-10} \)?

4 Compact storage

A matrix \( A \) generated by either \texttt{poissonmatrix} or \texttt{anothermatrix} has only five non-zero entries in each row. In fact, it has only five non-zero diagonals: the main diagonal, the super- and sub-diagonals, and the two diagonals that are \( N \) columns away from the main diagonal. All other entries are zero. It is prudent to store \( A \) in a much more compact form by storing only the diagonals and none of the other zeros. Further, \( A \) is symmetric, so it is only necessary to store \textit{three} of the five diagonals. While not all matrices have such nice and regular structure as these, it is very common to encounter matrices that are sparse (mostly zero entries). In the exercises below, you will be using a very simple compact storage, based on diagonals, but you will get a flavor of how more general compact storage methods can be used.

\textbf{Remark}: Since \texttt{poissonmatrix} contains only two different numbers, it can be stored in an even more efficient manner, but we will not be taking advantage of this strategy in this lab.

Since there are only three non-zero diagonals, you will see how to use a \( M \times 3 \) rectangular matrix to store a matrix that would be \( M \times M \) if ordinary storage were used. Further, since the matrices from \texttt{poissonmatrix} and \texttt{anothermatrix} are always \( N^2 \times N^2 \), we will only consider the case that \( M = N^2 \). The “main diagonal” is the one consisting of entries \( A_{k,k} \), the “superdiagonal” is the one consisting of entries \( A_{k,k+1} \) and non-zero diagonal farthest from the main diagonal consists of entries \( A_{k,k+N} \), where \( N^2 = M \). For the purpose of this lab, we will call the latter diagonal the “far diagonal.”

Conveniently, Matlab provides a function to extract these diagonals from an ordinary square matrix and to construct an ordinary square matrix from diagonals. It is called \texttt{diag} and it works in the following way. Assume that \( A \) is a square matrix with entries \( A_{k,l} \). Then

\[
\text{diag}(A) = A_{k,k} \\
\text{diag}(A,n) = A_{k,k+n}
\]

where \( n \) can be negative, so \texttt{diag} returns a column vector whose elements are diagonal entries of a matrix. Given a vector \( v \) with entries \( v_k \), then \( A = \text{diag}(v,n) \) returns a matrix \( A \) consisting of all zeros except that \( A_{k,k+n} = v_k \), for \( k = 1, 2, \ldots, \text{length}(v) \). You must be careful to have the lengths of the diagonals and vectors correct.

In the following exercise, you will rewrite \texttt{poissonmatrix} and \texttt{anothermatrix} to use storage by diagonals.

\textbf{Exercise 6}: 

(a) Copy your anothermatrix.m file to one called anotherdiags.m (or use “Save as”). Change the name on the signature line and change the comments to indicate the resulting matrix will be stored by diagonals.

(b) The superdiagonal corresponds to nAbove and the far diagonal corresponds to nRight. Modify the code so that the matrix A has three columns, the first column is the main diagonal, the second column is the superdiagonal, and the third column is the far diagonal. The superdiagonal will have one fewer entry than the main diagonal, so leave a zero as its last entry. The far diagonal will have N fewer entries than the main diagonals, so leave zeros in the last N positions.

(c) Your function should pass the following test

```matlab
N=10;
A=anothermatrix(N);
Adiags=anotherdiags(N);
size(Adiags,2)-3 % should be zero
norm(Adiags(:,1)-diag(A)) % should be zero
norm(Adiags((1:N^2-1),2)-diag(A,1)) % should be zero
norm(Adiags((1:N^2-N),3)-diag(A,N)) % should be zero
```

If all four numbers are not zero, fix your code. If you are debugging, it is easier to use N=3.

(d) Repeat the steps to write poissondiags.m by starting from poissonmatrix.m. Be sure to test your work.

If you look at the conjugate gradient algorithm, you will see that the only use made of the matrix is to multiply the matrix by a vector. If we plan to store a matrix by diagonals, we need to write a special “multiplication by diagonals” function.

Exercise 7:

(a) Start off a function m-file named multdiags.m in the following way

```matlab
function y=multdiags(A,x)
% y=multdiags(A,x)
% more comments
% your name and the date

M=size(A,1);
N=round(sqrt(M));
if N~=N^2
    error('multdiags: matrix size is not a squared integer.')
end
if size(A,2) ~=3
    error('multdiags: matrix does not have 3 columns.')
end

% the diagonal product
y=A(:,1).*x;
% the superdiagonal product
for k=1:M-1
    y(k)=y(k)+A(k,2)*x(k+1);
end
% the subdiagonal product
```
\( \text{the far diagonal product} \)
???
\( \text{the far subdiagonal product} \)
???

(b) There are three more groups of lines needed in order to include all five diagonals. Fill them in.

(c) Choose \( N=3 \), \( x=\text{ones}(9,1) \), and compare the results using \text{multdiags} to multiply anotherdiags by \( x \) with the results of an ordinary multiplication by the matrix from anothermatrix. The results should be the same. (Debugging hint: All the \( x \) are the same, so the only places your code might be wrong in this test are in the subscripts of \( A \).)

(d) Choose \( N=3 \), \( x=(10:18)' \), and compare the results using \text{multdiags} to multiply anotherdiags by \( x \) with the results of an ordinary multiplication by the matrix from anothermatrix. The results should be the same.

(e) Choose \( N=10 \), \( x=\text{rand}(N^2,1) \), and compare the results using \text{multdiags} to multiply anotherdiags by \( x \) with the results of an ordinary multiplication by the matrix from anothermatrix. The results should be the same.

5 Conjugate gradient by diagonals

Now is the time to put conjugate gradients together with storage by diagonals. This will allow you to handle much larger matrices than using ordinary square storage.

Exercise 8:

(a) Copy your cg.m file to cgdiags.m. Modify each product of \( A \) times a vector to use \text{multdiags}. There are only two products: one before the \text{for} loop and one inside it. You do not need to change any others.

(b) Solve the same problem twice, once using anothermatrix and cg and again using anotherdiags and cgdiags. In each case, use \( N=3 \), \( b=\text{ones}(N^2,1) \), \text{tolerance}=1e-10, and start from \text{zeros}(N^2,1). You should get exactly the same number of iterations and exactly the same answer in the two cases.

(c) Solve a larger problem twice, with \( N=10 \) and \( b=\text{rand}(N^2,1) \). Again, you should get the same answer in the same number of iterations.

(d) Construct a much larger problem to which you know the solution using the following sample code

\[
\begin{align*}
N&=400; \\
x\text{Exact}&=\text{rand}(N^2,1); \\
A\text{diags}&=\text{poissondiags}(N); \\
b&=\text{multdiags}(A\text{diags},x\text{Exact}); \\
tic;[y,n]=\text{cgdiags}(A\text{diags},b,\text{zeros}(N^2,1),1.e-6);toc
\end{align*}
\]

How many iterations did it take? How much time did it take?

(e) You can compute the relative norm of the error because you have both the computed and exact solutions. What is it? How does it compare with the tolerance? Recall that the tolerance is the relative residual norm and may result in the error being either larger or smaller than the tolerance.

In the previous exercise, you saw that the conjugate gradient works as well using storage by diagonals as using normal storage. You also saw that you could solve a matrix of size \( 1.6 \cdot 10^5 \), stored by diagonals. This matrix is actually quite large, and you would not be able to even construct it, let alone solve it by a direct method, using ordinary storage.
Exercise 9:

(a) Roughly, how many entries are in the Adiags from the previous exercise? If one number takes eight bytes, how many bytes of central memory does Adiags take up?

(b) Roughly, what would the total size of poissonmatrix(N) (for N=400) be? If one number takes eight bytes of central memory, how many bytes of central memory would it take to store that matrix? It costs about US$20 to purchase 1GB of RAM memory (2^{30} or roughly 10^9 bytes). How much would it cost to purchase enough memory to store that matrix? In your estimation, is this amount more or is it less than the University paid for each entire computer in GSCC?

(c) In Lab 3, Exercise 2, you found that the time it takes to solve a matrix equation using ordinary Gaussian elimination is proportional to $M^3$, where $M$ is the size of the matrix, and that it takes about $T = 6.8$ seconds to solve a matrix equation with $M = 2560$. Find $C$ so that $T = CM^3$ holds.

(d) How long would it take to use Gaussian elimination to solve a matrix equation using poissonmatrix(N) (N=400)? Please convert your result from seconds to days. How does this compare with the time it actually took to solve such a matrix equation in the previous exercise?

Exercise 10: Matlab has a built-in compact storage method for general sparse matrices. It is called “sparse storage.” It is not difficult to see how it works by looking at the following example. As you saw above, the gallery('poisson',N) constructs a matrix that is $h^2$ times the matrix from poissonmatrix. Use the following command to print out a sparse matrix

\[ A = \text{gallery('poisson',5)} \quad \% \text{no semicolon!} \]

and you should be able to see how it is stored. From what you just printed what do you think $A(15,20)$, $A(16,20)$ and $A(20,20)$ are?