Due date: Maple: Monday, February 28, at the beginning of the lab  
C++/Java: Monday, March 07, at the beginning of the lab.

1 What this Lab is About

Our plans for this assignment are

- To learn how to solve systems of linear equations with Maple, evaluate the results, and check out what went wrong;
- Implement simple Matrix and Vector classes in C++ or Java, and verify that you get the desired results by comparing them with what you get in Maple.

2 Linear Algebra in Maple

Maple has several mutually incompatible linear algebra packages with significantly overlapping functionalities, and different naming conventions. It is a bit of a mess, so be careful what you use.

For this assignment, I think that you are better off using the LinearAlgebra package. I know that most of you used linalg in MTH 215, but I think that the latter package is more limited. It certainly seems to pose more problems when mixed with procedures (judging by student feedback from the last three years).

2.1 The basics

Here is some simple Maple code to create and multiply vectors and matrices.

```maple
> A := Matrix([[1, -2],[3, 4]]);
> b := Vector([4, 2]);
> d := A &* b;
> B := A &* A;
> v := [5, 10, -7, 8, 2];
> V := Vector(v);
```

The &* symbol stands for matrix and vector product. Now if you want to see the result in matrix form, you must `evalm()` the expression.
An interesting thing to notice is that our vectors are both row and column vectors. In other words, you could write $d \times B$ as well (of course, the dimensions have to match). Maple uses the context to decide if a 1D vector should be seen as an $n \times 1$ or a $1 \times n$.

Note that this can lead to problems. What does $d \times d$ compute? How would you get the other form?

2.2 Norm of a matrix

First, you should read the part of the notes that deals with norms. Next, determine which norm is implemented by default by the function `Norm` of the `LinearAlgebra` package. Implementing the other norm is a one-line affair.

2.3 Determinants

2.3.1 Recursive computation

You are all familiar with the technique of computation of a matrix' determinant by development along a row (or a column). The main quality of this technique is that it is very straightforward, but in practice you should almost never use it in a program because it is simply too inefficient. I just give the definition here as a reminder.

\textbf{Definition 1 (cofactor).} 
Given an $n \times n$ matrix $A = [a_{i,j}]_{i,j=1}^{n}$, we note $A_{k,l}$ the submatrix of $A$ that is obtained by eliminating from $A$ its $k$th row and its $l$th column. We then call cofactor of $a_{k,l}$ the determinant

$$D_{k,l} = (-1)^{k+l} \det (A_{k,l}).$$

Then you know that the determinant of $A$ can be computed as follows (note that this is really a property, not a definition):

\begin{align*}
\det(A) & = \sum_{i=1}^{n} a_{1,i} D_{1,i}, \\
& = \sum_{j=1}^{n} a_{j,1} D_{j,1}.
\end{align*}

The first equation corresponds to a development along the first column, while the second equation corresponds to a development along the first row.

2.3.2 Iterative computation

The other “naive” way to compute the determinant is to apply directly the definition. It is possible that you may not have seen that one in your linear algebra class, so I have to give another definition first.
Definition 2 (permutation).
Given a sequence of $n$ elements, we call permutation of that sequence any reordering of its elements. It can be shown that any permutation $\sigma$ is the combination of a series of swappings of two elements in the sequence. In addition, the combinations of swappings that produce a given permutation are either all odd or all even. We define $\varepsilon(\sigma)$, the signature of a permutation $\sigma$ as

$$
\varepsilon(\sigma) = \begin{cases} 
-1, & \text{if } \sigma \text{ is an odd permutation;} \\
1, & \text{if } \sigma \text{ is an even permutation.}
\end{cases}
$$

Definition 3 (determinant).
Let us consider the integers between 1 and $n$ and let us call $\mathcal{P}_n$ the set of permutations of these numbers. Then we can define the determinant of an $n \times n$ matrix $A = [a_{ij}]_{i,j=1..n}$ as

$$
\det(A) = \sum_{\sigma \in \mathcal{P}_n} \varepsilon(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}
$$

(3)

$$
= \sum_{\sigma \in \mathcal{P}_n} \varepsilon(\sigma) \prod_{j=1}^{n} a_{\sigma(j)j}.
$$

(4)

Maple 1. Implement this iterative algorithm in Maple. Use the determinant function of Maple to verify your result.

Next week, we will see how determinants are usually computed in practice. The above algorithm, though, is still fairly efficient, certainly vastly superior to the recursive algorithm.

2.4 Inverse of a Matrix [extra credit 5 points]

In class the other day I mentioned briefly a technique to compute the inverse of a matrix by using determinants of subparts of that matrix. Here is that theorem (it is using the definition of the cofactors):

Theorem 1.
Let $A$ be an $n \times n$ matrix $A = [a_{ij}]_{i,j=1..n}$ and let $D_{k,i}$, $k, i = 1 \ldots n$, be the cofactors of the $a_{k,i}$. Then if $A$ is not singular we have

$$
A^{-1} = \frac{1}{\det A} \left( [D_{k,i}]_{k,i=1..n} \right)^T
= \frac{1}{\det A} \left( \begin{array}{cccc}
D_{1,1} & D_{1,2} & \cdots & D_{1,n} \\
D_{2,1} & D_{2,2} & \cdots & D_{2,n} \\
& & \ddots & \vdots \\
D_{n,1} & D_{n,2} & \cdots & D_{n,n}
\end{array} \right)^T.
$$

(5)

Maple 2. Implement this algorithm in Maple (using your determinant computation function). Use the matrix inverse function of Maple to verify your result.
3 A First Look at Systems of Linear Equations

3.1 LinearSolve

You can use the function LinearSolve to solve systems of linear equations in Maple. At this point, you don’t need to specify a particular technique for solving the linear system: Maple’s default algorithm will do.

3.2 First example

First, try to solve a simple equation \( A \cdot x = b \). For example, you can use the following SLE:

\[
\begin{pmatrix}
3 & 4 & -1 & 2 \\
3 & 4 & 3 & 3 \\
-3 & -6 & 4 & -1 \\
-6 & -2 & -3 & -5
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix}
=
\begin{pmatrix}
-1 \\
10 \\
0 \\
11
\end{pmatrix}.
\]

3.3 Another example

Try again, this time with the following matrix:

\[
B = \begin{pmatrix}
1 & 1 + \varepsilon \\
1 & 1
\end{pmatrix}.
\]

Solve this problem for different (small) values of \( \varepsilon \). What do you observe? Use again the other technique for solving the equation you used with the previous example, and compare your results.

Maple 3. Compute the norm of \( B \) and its condition number as functions of \( \varepsilon \). Plot the graphs of these functions. Get the expression of the solution as a function of \( \varepsilon \). Make some sensitivity analysis based on what you know about floating point computations.

I remind you (if you have read the notes, that is) that the condition number of a matrix is

\[
\text{cond}(A) = \|A\| \|A^{-1}\|.
\]

By convention, if \( A \) is singular then \( \|A^{-1}\| = \infty \). Obviously, the value of the condition number depends on the norm chosen (the same norm is used for \( A \) and \( A^{-1} \)), but a matrix that is ill-conditioned with one norm will not be well-conditioned with another. If you look at the definition of the norm, you will see that the condition number is the ratio of the maximum amount of stretching and of the minimum amount of stretching that the matrix affords.
4 Algorithmic Solutions of an SLE Problem

4.1 General solution of an SLE

This one is really simple. Since you know how to compute $A^{-1}$ if $A$ is not singular, than you can solve any SLE $A \cdot x = b$ by computing $x = A^{-1} \cdot b$.

In practice, however, this technique is just not practical: computationally inefficient, very sensitive to noise, unable to handle ill-conditioned matrices.

4.2 Cramer’s rule [extra credit: 7 points]

This algorithm works also better in mathematics than in the real world, but since we have implemented a procedure to compute determinants, we might as well put it to some use. Given the following $m \times m$ SLE

$$A \cdot x = b,$$

we can define $D_i$ to be the $m \times m$ matrix equal to $A$ except for the $i$th column that has been replaced by the right-side term $b$:

$$D_i = \begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1m} \\
a_{21} & a_{22} & \cdots & a_{2,i-1} & b_2 & a_{2,i+1} & \cdots & a_{2m} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{m,i-1} & b_m & a_{m,i+1} & \cdots & a_{mm}
\end{pmatrix},$$

$$= \left( \begin{array}{cccc}
A_{11} & A_{12} & \cdots & A_{1,i-1} \\
A_{21} & A_{22} & \cdots & A_{2,i-1} \\
\vdots & \vdots & \ddots & \vdots \\
A_{m1} & A_{m2} & \cdots & A_{m,i-1}
\end{array} \right) \begin{pmatrix} b \\ A_{i+1} \\ \vdots \\ A_{m} \end{pmatrix}.$$

**Theorem 2 (Cramer’s rule).**

The variables of the SLE can be computed by application of the ratio

$$x_i = \frac{\det D_i}{\det A}.$$

**Maple 4.** Write a Maple procedure solveCramer that receives as parameters a square matrix $A$ and a column vector $b$, and returns the solution to the SLE $A \cdot x = b$.

4.3 Iterative improvement of the solution

This subsection is just copied from the notes.

Having somehow solved the SLE $A \cdot x = b$, we obtain a solution $\hat{x}$, which we hope to the close to the real solution $x$. Let us call $\delta x$ the absolute error of our solution: $\delta x = \hat{x} - x$. If we multiply our solution $x$ by the SLE’s matrix, $A$, we get

$$A \cdot \hat{x} = A \cdot (x + \delta x) = A \cdot x + A \cdot \delta x.$$

Since $A \cdot x = b$, we see that

$$A \cdot \delta x = A \cdot \hat{x} - b.$$
This defines a new SLE whose matrix is still \( A \). The right-side term is known, and therefore we can solve the SLE to get an estimate of the error term \( \delta x \). Having done this, we can get a better approximation of the correct solution by computing

\[
\hat{x} = \hat{x} - \delta x.
\]

We can therefore improve our solution by calculating an estimate of the error term and subtracting it from the solution \( \hat{x} \). It is highly recommended to apply this extra step at least once because

- It improves solutions that were already good,
- It can sometimes salvage completely incorrect solution (by at least putting them back in the right range of values),
- The additional computation time is small compared to that to compute the first solution. Just keep in mind that the matrix of the iterative improvement step is the same as that of the original SLE. The algorithmic complexity of decomposing the matrix is \( O(n^3) \), while that of solving the SLE once the matrix has been decomposed is only \( O(n^2) \). the cost of performing the \( O(n^2) \) operation twice is therefore negligible compare to that of the entire operation.

It is possible to apply this step more than once, twice being generally considered to be sufficient. It is also possible to keep applying the iteration until the solution doesn’t change anymore.

**Maple 5.** Implement the incremental improvement algorithm. Your procedure should take as parameters the matrix \( A \) and the right-side term of the SLE, \( b \), the current approximate solution \( \hat{x} \), a maximum number of iterations \( n \), and a tolerance \( \varepsilon \). Your iteration should stop when the relative change in the solution is below the tolerance or the max number of iterations has been reached.

Please note that this incremental improvement technique can and should be applied regardless of the algorithm used to solve the SLE.

## 5 Iterative Solution of an SLE

The following algorithms should not be confused with the incremental improvement technique we have just discussed. The incremental improvement technique just make small adjustments to a column vector that is already in the neighborhood of the solution. The iterative algorithms we are about to see make no such assumption.

The idea behind these techniques is to compute the solution to an SLE as the limit of a converging sequence. I don’t want you either to memorize any of these techniques. What is important is that you understand what we are trying to do.

### 5.1 The Basic algorithm: Jacobi iteration

Let us start from our standard SLE

\[
Ax = b, \tag{6}
\]
now let us decompose matrix $A$ into the sum of three matrices $L$, $D$, and $U$ such that $L$ is strictly lower-diagonal (diagonal elements are all zero), $D$ is diagonal, and $U$ is strictly upper-diagonal. Now we can rewrite Eq. (6) as

$$Dx = -(L + U)x + b,$$

or, if if multiply on both sides of the equality sign,

$$x = -D^{-1}(L + U)x + D^{-1}b.$$ (7)

So what have we gained by changing Eq. (6) into Eq. (7)? Simply it offers up the possibility to try to compute $x$ as the limit of the following recursive sequence:

$$x^{(k+1)} = (-D^{-1}(L + U)) x^{(k)} + D^{-1}b,$$ (8)

where the superscripts indicate the order in the iteration. Why should it work? Does it always work? Even if it does work, what is so good about that? Let me answer the last question first. Since $D$ is so easy to invert, we can compute $D^{-1}(L + U)$ and $-D^{-1}b$ once and for all, and all we have to do in our iteration is repeatedly multiply a matrix and a vector and add two vectors.

5.1.1 Develop some intuition in 1D

You’re going to answer the first two questions. To do this, you should build up some intuition for the problem first by studying the 1D case:

$$x^{k+1} = ax^k + b.$$  

Try this for different values of $a$ and $b$. It should be pretty obvious for you to propose a criterion for guaranteed convergence of this equation.

5.1.2 Back to the SLE

Can you now think of a criterion to guarantee that the iteration will work with your SLE?

Maple 6. Implement the Jacobi iteration algorithm in Maple.

N.B. Maple has a function named `LUDecomposition` and some of you might be tempted to use it. However, this function computes the LU decomposition of a matrix, not its LUD-decomposition. The LUD decomposition is summation-based:

$$A = L + D + U,$$

while the LU decomposition is product-based:

$$A = LU.$$
5.2 Gauss-Seidel iteration

The Gauss-Seidel iteration uses the same “trick” as the Jacobi iteration, except that it does not pass the L matrix on the right side of the equation as in Eq. 7. Instead we write:

\[(D + L)x = -Ux + b,\]  

so that our iteration now becomes

\[(D + L)x^{(k+1)} = -Ux^{(k)} + b,\]  

that is, a lower-triangular SLE. Instead of inverting the matrix on the left side, it is much preferable to solve it by forward substitution.

So why is this good, when it looks more complicated than the Jacobi version. To see this you have to express both equations at the level of the components of the x unknown vector.

For the Jacobi iteration

\[x_i^{(k+1)} = \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right].\]

For the Gauss-Seidel iteration

\[x_i^{(k+1)} = \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right].\]

Note that the Gauss-Seidel algorithms uses elements of the \(k+1\)th iterations to compute other elements of the \(k+1\)th iteration, whereas the Jacobi iteration uses only elements of the \(k\)th iteration. We say that Jacobi iteration is a simultaneous correction method, while Gauss-Seidel is a successive correction method.

Maple 7. Do the same for Gauss-Seidel as you did with Jacobi (think of a criterion for convergence and check with a few examples). Now that you have seen both algorithms in operation, can you think of criteria for halting the iteration?

5.3 One more iterative technique [extra credit: 6 pts]

This one is just for your information, you don’t have to do anything with it for this lab, but we may use it in a few weeks when we talk about partial differential equations. This modified version of the Gauss-Seidel iteration is called successive over-relaxation, or SOR. The idea here is just to add \(x_i^{(k)}\) in one place of the Gauss-Seidel iteration and remove it in another place. We obtain

\[x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right] = x_i^{(k)} + \delta_i^{(k)}.\]

So what we’re doing is just adding a correcting term to the current estimate \(x_i^{(k)}\). Obviously this makes it easier to check if the iteration is converging.
Maple 8. Implement the SOR algorithm.

6 C++ or Java Implementation

You are going to implement Matrix and Vector classes that do the following:

- Create vectors and arrays of any dimensions;
- Add, subtract, and multiply them (if the dimensions match);
- Transpose matrices and vectors;
- Compute the $N_1$, $N_2$, and $N_\infty$ norms for vectors and the $N_1$ and $N_\infty$ norms for matrices.

7 Evaluation

7.1 What to hand in

Please read this part carefully. You will be penalized (see next subsection) if you don’t follow the rules and the TA ends up wasting time converting files just to be able to evaluate your work.

**A:** You should upload the following (the procedure to follow will be clarified by the end of this week):

1. Next week: the file of a well-commented Maple worksheet. Please make

2. Two weeks from now: A complete, cleaned-up CodeWarrior, gcc (mingw or cygwin), or Eclipse project folder.

   - Complete means that everything needed to compile and execute the project is there: source files, project file or make file, header files (.h files) and precompiled header source files (.pch files) if you program in C++. Of course, your test data files should be there as well, but the TA might use other data sets to test your code.

   - Cleaned-up means that files resulting from the compilation should be removed: .exe files, .sym files, .o object files (in C++), .class files (in Java), precompiled header binary files (if in C++). If you use CodeWarrior, the entire `<name of project>` data folder should be deleted as well (close the project before you throw away this folder).

3. Two weeks from now: Your report as a Word, HTML, or Acrobat (.pdf) document.

**B:** You should hand in printouts of your (complete) Maple worksheet and of your report on the day the corresponding part of the assignment is due.
7.2 Point distribution

The maximum number of points is 100, but extra points could be awarded for excellent aspects of the project or report. The point distribution for this assignment is as follows:

**Maple Modelling**
- Accomplishes what was demanded: 20 pts
- Comments and analysis: 10 pts

**C++/Java Code**
- Accomplishes what was demanded: 20 pts
- Good class design: 10 pts
- General quality & readability: 10 pts

**Report**
- Discussion and analysis of the results: 20 pts
- General quality of the writing and presentation: 10 pts

7.3 Various point penalties

Hopefully we won’t have to apply many of these:

- Project folder incomplete or not properly cleaned up: -5 pts
- Report file missing from the project folder: -5 pts
- Maple file missing: 0 for that part

**Late penalties**
- Printed copy of the report, 1 day late: -5 pts
- Project folder (uploaded to EnVision server), per day late: -5%

If you submit a project late, then it is your responsibility to notify the TA (with CC. to me) that the project is finally available for download on the EnVision server. If you fail to do so, then the “late penalty clock” will keep ticking until the TA gets around to checking your folder on the EnVision server and notices your project. Unless asked explicitly to do so, do not mail your project folder as an attachment.

As I explained in class, I will post sometimes partial solutions to the assignments, to make sure that nobody gets too far behind. Obviously, it will be impossible to get any point on a part after a solution for it has been posted (this does not affect the late penalty count).