Objectives of this Assignment

By now I assume that the following is true for all of you:

1. You have cleared-up the obvious cobwebs, patched the most glaring holes, and generally brush up on your Java/programming skills.
2. You have enough experience with Maple that you can now use it to prototype your algorithms.
3. You have developed a methodology and some basic automatisms and/or tools to use Maple efficiently. You probably have enough experience with it now that you frequently wish Maple developers bodily harm.

Our plans for today are:

To apply our knowledge and tools to the resolution of a simple practical problem.

1. The basics (long)

LU Factorization

To implement an efficient algorithm for performing the LU factorization of a matrix A.

To add some classical methods to our Matrix class, such as matrix inversion and determinant.

To apply our knowledge and tools to the resolution of a simple practical problem.

The Gaussian elimination (also called Gauss-Jordan elimination) algorithm that we use when solving an upper-triangular (matrix, so that the new SLE can be easily solved by (backward) substitution. We denote this matrix formally as $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$, and $b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$.

At step $i$, our SLE looks as follows:
\begin{align*}
&\text{thatis, } \\
&\begin{pmatrix}
A^1x^1 = b^1 & \iff & \begin{pmatrix}
A^1 & 0 \\
0 & b^1
\end{pmatrix} \\
A^1x^2 = b^2 & \iff & \begin{pmatrix}
A^1 & 0 \\
0 & b^2
\end{pmatrix}
\end{align*}

Of course, the operation we just applied can be expressed as a multiplication by a matrix on both
\begin{align*}
\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1
\end{pmatrix}
\begin{pmatrix}
x^1 \\
x^2 \\
\vdots \\
x^n
\end{pmatrix}
&= \\
\begin{pmatrix}
x^1 \\
x^2 \\
\vdots \\
x^n
\end{pmatrix}
\end{align*}

After this step, row \( q \) is now "final":
\begin{align*}
&\begin{pmatrix}
A^1 & 0 \\
0 & b^1
\end{pmatrix} \\
&\begin{pmatrix}
A^1 & 0 \\
0 & b^2
\end{pmatrix}
\end{align*}

If we set aside the issue of pivoting (i.e., of swapping rows or columns when the diagonal term \( a_{ii} \) is null or small), we have seen that if we want to cancel all the terms in column \( q \) that lie below the diagonal, we simply have to apply the following algorithm:
\begin{align*}
&\begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\
\end{pmatrix}
&\begin{pmatrix}
x^1 \\
x^2 \\
\vdots \\
x^n
\end{pmatrix}
&= \\
&\begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\
\end{pmatrix}
\begin{pmatrix}
x^1 \\
x^2 \\
\vdots \\
x^n
\end{pmatrix}
\end{align*}

In this expression, the matrix terms labelled \( u_{ij} \) and the right hand terms labelled \( \tilde{u}_{ij} \) with column \( q \leq j \), that is,
\begin{align*}
&\begin{pmatrix}
u_{q} \\
v_{q} \\
v_{q} \\
v_{q}
\end{pmatrix}
&= \\
&\begin{pmatrix} u_{1q} & \cdots & u_{1q} \\ \vdots & \ddots & \vdots \\ u_{1q} & \cdots & u_{1q} \\ u_{1q}
\end{pmatrix}
\begin{pmatrix}
x^1 \\
x^2 \\
\vdots \\
x^n
\end{pmatrix}
\end{align*}
You can consult your textbook, Pages 66-68, to get some details on the mathematical properties

\[
\begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 1 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 1
\end{pmatrix}
\]

where
It turns out that \( M \) is non-singular (you should be able to figure out why on your own) and that its inverse matrix is also lower-triangular with a unit diagonal. For example,

\[
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}^{-1} = \begin{pmatrix}
  1 & 0 & \cdots & 0 \\
  0 & 1 & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & 1
\end{pmatrix}
\]

The LU factorization algorithm

2.1. Maple implementation: Forward and backward substitution

This SLE can be solved in two easy steps:

\[
\mathbf{L} \mathbf{x} = \mathbf{b}
\]

\[ \mathbf{U} \mathbf{x} = \mathbf{L}^{-1} \mathbf{b} \]

When this holds, one is then left with \( \mathbf{L}^{-1} \mathbf{b} = \mathbf{U} \mathbf{x} \). But this is identical to the right-hand side \( \mathbf{y} \), where \( \mathbf{y} = \mathbf{U} \mathbf{x} \) which is easily done by forward substitution. The left-hand side is independent from the right-hand term \( \mathbf{b} \) and the original SLE can now be written as

\[
\begin{pmatrix}
  1 & 2 & \cdots & n \\
  0 & 1 & \cdots & 1 \\
  0 & 0 & 1 & 1 \\
  0 & 0 & 0 & 1
\end{pmatrix} = \begin{pmatrix}
  1 & 2 & \cdots & n \\
  0 & 1 & \cdots & 1 \\
  0 & 0 & 1 & 1 \\
  0 & 0 & 0 & 1
\end{pmatrix}
\]

2.2. Maple implementation: Forward and backward substitution

If you decide to do so, the name of your procedure (an, later, its C++/Java implementation) should reflect that. Of course, this speed gain would come at the cost of the ease of implementation. Note that in the case of the lower-triangular SLE, you may want to exploit the fact that you know that your matrix has a unit diagonal. However, without saying that you should thoroughly test your procedure to verify that they work as desired.
If we develop this equation to get an expression for each of the terms of $A$, we must differentiate.

One drawback of overwriting $\mathbf{V}$ is that you lose your input data. If you do this, better make sure

$$
\begin{pmatrix}
  w_{11} & w_{12} & \cdots & w_{1n} \\
  \vdots & \ddots & \ddots & \vdots \\
  w_{n1} & w_{n2} & \cdots & w_{nn}
\end{pmatrix}
$$

Although when we begin the algorithm none of the terms $w_{ij}$ is know, they are all computed

$\begin{pmatrix}
  \ell_{11} & \ell_{12} & \cdots & \ell_{1n} \\
  \vdots & \ddots & \ddots & \vdots \\
  \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn}
\end{pmatrix}$

and $n$ terms in one pass, working over the columns from 1 through $n$.

A careful look at these equations reveals something very interesting: It is possible to compute the

$$
\ell_{ij} = \ell_{1j} + \cdots + \ell_{ij-1} + \ell_{ij+1} + \cdots + \ell_{nn} \quad : i < j
$$

$$
\ell_{ij} = \ell_{1j} + \cdots + \ell_{ij-1} + \ell_{ij+1} + \cdots + \ell_{nn} \quad : i = j
$$

$$
\ell_{ij} = \ell_{1j} + \cdots + \ell_{ij-1} + \ell_{ij+1} + \cdots + \ell_{nn} \quad : i > j
$$

are between these cases:

If we develop this equation to get an expression for each of the terms of $\mathbf{V}$, $w_{ij}$ must differen-

The total number of steps in this factorization is about $\frac{n^2}{2}$.
Pivoting

So far, we have carefully avoided the pivoting issue. What do we do when the term $\hat{i}$ is too small? We swap rows, as usual? But this brings up two questions:

1. Since swapping rows is a very fast operation, why not do it all the time, and always choose the largest pivot in absolute value?

2. At what point in the algorithm can we perform the pivoting operation, we must be careful about one thing: we are (or could be) performing the factorization of $A$ before we know the right hand term $b$. Therefore, be careful when regarding the swapping operation.

Regarding the swapping operation, we must be careful about one thing: we are (or could be) performing the factorization of $A$ before we know the right hand term $b$. Therefore, be careful when regarding the swapping operation.

The simplest way to do this is to maintain an array of row indexes. In other words, we replace

$$\hat{i}$$

with the right hand term when we try to solve the SLE.

Thesimplestwayto do this istomaintainarrayofrowindexes. Inotherwords,we replacethe right hand term when we try to solve the SLE.

$$\hat{i}$$

The simplest way to do this is to maintain an array of row indexes. In other words, we replace the right hand term when we try to solve the SLE.

\[
\begin{align*}
\text{compute } &\frac{\hat{i}}{1} \\
\text{for } i & \text{ from } 1 \text{ to } n \text{ do} \\
\text{for all elements below (possibly new) diagonal} \\
& \text{set } (\hat{i} \in \mathbb{N}) \land (\hat{i} \neq \mathbb{n}) \text{ swap rows } \hat{i} \text{ and } \mathscr{A} \text{ swap rows } \hat{i} \text{ and } \mathbb{N} \\
& \text{determine such that } \omega = \frac{\hat{i}}{\mathbb{N}} \\
& \text{compute } i = \hat{i} - \omega \\
& \text{for } i \text{ from } 1 \text{ to } n \text{ do} \\
& \text{compute } \frac{\hat{i}}{1} = \hat{i} \\
& \text{for all elements above the diagonal} \\
& \text{compute } i = \hat{i} - \omega \\
& \text{for } i \text{ from } 1 \text{ to } n \text{ do} \\
& \text{compute } \frac{\hat{i}}{1} = \hat{i} \\
& \text{for all elements below the diagonal} \\
& \text{compute } i = \hat{i} - \omega \\
& \text{for } i \text{ from } 1 \text{ to } n \text{ do} \\
& \text{compute } \frac{\hat{i}}{1} = \hat{i}
\end{align*}
\]

This is somewhat intricate at first. But then when we look carefully at the expression for $\omega$, it is not hard to see why.

The answer to Question 1 is: Indeed, why not? This is exactly what we will do.

2. At what point in the algorithm can we compute the pivots and swap rows?

I. Since swapping rows is a very fast operation, why not do it all the time, and always choose small $\hat{i}$? We swap rows, as usual? But this brings up two questions:

So far, we have carefully avoided the pivoting issue. What do we do when the term $\hat{i}$ is too small? We swap rows, as usual? But this brings up two questions:

2.4 Pivoting
3.1 Solve a general SLE

This is a pretty straightforward application of the factorization, merely an integration of all your procedures into a general "solve the SLE" Maple procedure that takes as parameters a square $A \times A$ matrix $A$ and an $A \times 1$ column vector $b$, and solves the SLE $A x = b$.

3.2 Compute the determinant of a matrix

Having computed the LU factorization of a matrix $A$, you should have no many problems inverting it. The inversion computation can simply be posed as the resolution of $A$ different SLEs with the same matrix $A$, but different right hand terms. I will not explain how to get the determinant once you have the LU factorization; this one is almost embarrassingly easy. The only thing you have to be careful about is the sign of the determinant. This one is learned in your linear algebra course.

3.3 Compute the inverse of a matrix

And changes sign each time two rows are swapped. This may come as a surprise to you, but the most efficient way to compute the determinant of a matrix is to compute its LU factorization first, rather than apply the recursive expression you learned in your linear algebra course.

3.4 Count the condition number of a matrix

An easy 4 points of extra credit here. Of course, for this I want both condition numbers (for the $\| \cdot \|_\infty$ and $\| \cdot \|_2$ norms) and the $C++$/Java implementation.

4.1 Pass a curve through data points

Imagine that you have a set of $N$ data points of the form $(x_i, y_i)$ for $i = 1, \ldots, N$. The number of points is arbitrary. The only constraint they must respect is that no two points have the same $x$ coordinate. The range of the $x$ and $y$ coordinates of these points is at least 2, but could be arbitrarily large. The range of the $x$ and $y$ coordinates of these points is at least 2, but could be arbitrarily large. The number of these points is at least 2, but could be arbitrarily large.

This little application will give you an opportunity to reuse your $PolyJet$ class from Lab02 to

4 Test application

and the $C++$/Java implementation.
you saw in Lab02. High-degree polynomials are only useful on a very small range of values of x.

Derivatives in particular are very easy to compute, as you saw in Lab02. On the other hand, as

The standard approach to this problem is to impose a model for our function. One such model

not always in every objective.

If there is an infinity of solution, which one should we pick? First, there is no clear "best"

Figure 1: Our data as a list of coordinates and as geometric points.

Figure 2: Two different functions that pass through all our data points.

Figure 2: The graphs of two different functions that pass through all our data points.
4.2 The equations

Let us now consider a data point \( \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \) and a polynomial function of degree \( m \),

\[
\begin{align*}
\phi(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m \\
\text{where} \quad a_0, a_1, \ldots, a_m &\quad \text{are unknowns.}
\end{align*}
\]

You know from Lab 2 that each data point gives us one linear equation in terms of our unknowns \( a_i \). Let us now consider a data point \( \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \) belonging to the graph of \( f \).

\[
\begin{align*}
\sum_{i=0}^{m} \lambda_i \phi(\lambda_i) &= a_0 \lambda_0 + a_1 \lambda_1 + \cdots + a_m \lambda_m \\
\text{which is a linear} \quad a_0, a_1, \ldots, a_m \\
\text{equation in terms of the unknowns} \quad \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}.
\end{align*}
\]

Let us now consider a data point \( \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \) and a polynomial function of degree \( m \),

\[
\begin{align*}
\sum_{i=0}^{m} \lambda_i \phi(\lambda_i) &= a_0 \lambda_0 + a_1 \lambda_1 + \cdots + a_m \lambda_m \\
\text{which is a linear} \quad a_0, a_1, \ldots, a_m \\
\text{equation in terms of the unknowns} \quad \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}.
\end{align*}
\]
6 Evaluation

6.1 What to hand in

A: You should upload to the EnVision server a folder containing the following:

- A complete, cleaned-up CodeWarrior project folder:
  - Project folder (uploaded to EnVision server), per day late
  - Printed copy of the report, 1 day late

  Late penalties

- Project folder incomplete or not properly cleaned up: 0 pts

- Project folder incomplete or not properly cleaned up:
  - Report missing from the project folder
  - File of a well-commented Maple or muPad worksheet: 5 pts

B: You should hand in hard copies of your report and of your Maple or muPad worksheet.

6.2 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:

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Maple or muPad modeling</td>
<td>20 pts</td>
</tr>
<tr>
<td>Comments and analysis</td>
<td>10 pts</td>
</tr>
<tr>
<td>C++/Java code</td>
<td>20 pts</td>
</tr>
<tr>
<td>Good class design</td>
<td>10 pts</td>
</tr>
<tr>
<td>Accomplishes what was demanded</td>
<td>10 pts</td>
</tr>
<tr>
<td>Discussion and analysis of the results</td>
<td>20 pts</td>
</tr>
<tr>
<td>Discussion and analysis of the results</td>
<td>20 pts</td>
</tr>
<tr>
<td>General quality of the typing and presentation</td>
<td>10 pts</td>
</tr>
</tbody>
</table>

6.3 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 pts for that part

Late penalties

- Printed copy of the report, 1 day late: -5 pts
- Printed copy of the report, 1 day late: -5 pts

Printed copy of the report, 1 day late: -5 pts

Late penalties

- A well-commented Maple or muPad worksheet:
  - Project folder (uploaded to EnVision server), per day late
  - Printed copy of the report, 1 day late

Late penalties
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.