Due date: Maple: Monday, February 06, at the beginning of the lab
C++/Java: Monday, February 13, at the beginning of the lab

1 What this Lab is About

The main objective of this lab is for you to get used (again?) to Maple, its awkward syntax and interface, and less-than-perfect help system. I also want you to get used to importing output data from a C++/Java application back into Maple for plotting, verification, and comparison.

2 Reminder: Derivatives

In this lab, we will only consider functions \( f : I \to \mathbb{R} \), with \( I \subseteq \mathbb{R} \), that is, functions of a single real variable that take their values in the set of real numbers. I will assume that the function \( f \) that I talk about is \( C^\infty \) around \( x_0 \), that is, for any order \( n \), the \( n \)th derivative of \( f \), \( f^{(n)} \) is defined and continuous at \( x_0 \).

Since I only posted my class notes last Thursday or Friday, I have repeated here parts of these notes relative to function derivatives and Taylor expansions, so that this first lab assignment is more or less “self-contained.” I will not do so for future labs.

2.1 Derivative of a function

First definition

We say that \( f \) is \textit{derivable} at \( x_0 \) iff the function

\[
I \setminus \{x_0\} \to \mathbb{R} \\
x \mapsto \frac{f(x) - f(x_0)}{x - x_0}
\]

has a limit as \( x \) tends toward \( x_0 \). This limit is called the derivative of \( f \) at \( x_0 \) and is noted \( f'(x_0) \).

We say that \( f \) is derivable on \( I \) iff \( f \) is derivable at all points of \( I \).

The interpretation of this definition is that the derivative of \( f \) at \( x_0 \) is the local change rate of \( f \) at \( x_0 \), or the slope of its graph at \((x_0, f(x_0))\).
2.2 Linear approximation of a function

If there exists a neighborhood $V$ of $x_0$ (an open interval of $\mathbb{R}$ that contains $x_0$) and a number $l \in \mathbb{R}$ such that

$$\forall x \in V, f(x) = f(x_0) + l \cdot (x - x_0) + o(x - x_0),$$

then we will say that $f$ is derivable\(^1\) at $x_0$. We call $l$ the derivative of $f$ at $x_0$ and note it $f'(x_0)$.

This definition is very interesting, because it says that if $f$ is derivable at $x_0$, then it can be approximated by a linear function on a small neighborhood around $x_0$. In other words, if we know that $f$ is derivable at $x_0$, then all we need to build a reasonable approximation of $f$ around $x_0$ are the values of $f(x_0)$ and $f'(x_0)$.

2.3 Check the validity of this approximation

Create a Maple worksheet and define a few functions. The syntax for defining a function in Maple is

$$> f := x -> x * \cos(x);$$

You can plot this function on an interval (for example $[-3, 3]$), by executing

$$> \text{plot}(f(x), x=-3..3);$$

Then you can define and plot the derivative of your function, alone or together with $f$.

$$> df := x -> \text{diff}(f(x), x);$$
$$> \text{plot}(df(x), x=-3..3);$$
$$> \text{plot}([f(x), df(x)], x=-3..3);$$

Finally, you can define the linear approximation of your function around a point $x_0$.

$$> lf := x -> f(x_0) + \text{eval}(df(y), y=x_0) \cdot (x-x_0);$$

Now you should plot $f$ and this linear approximation around the $x_0$ that you chose and see how good the approximation is.

In fact, Maple has a package named student that does some of the work for you (in the linear case). Just try

$$> \text{with(student)};$$
$$> \text{showtangent}(f(x), x_0);$$

This package does some neat things, but not Taylor expansions, which we will do next.

\(^1\)In fact, this is the 1-dimensional version of the definition of the differentiability of $f$ at $x_0$, which we will use when we deal with optimization. When dealing with functions $\mathbb{R} \rightarrow \mathbb{R}$, derivability and differentiability are, for all purpose, equivalent notions.
2.4 Try this with a few functions

Maple 1.
Check the validity of your linear approximation for the following functions at a few well-chosen
points. Zoom in (by changing the range of your plot) to get a better feel for what is going on.

\begin{align*}
f_1(x) &= \cos \left( \frac{x^5 - 1}{1 + x^3} \right), \\
f_2(x) &= \cos x \frac{1 - \cos 2x}{(2 + \sin x)^2}, \\
f_3(x) &= \cos x \frac{1 - \cos 2x}{(1 + \sin x)^2}, \\
f_4(x) &= x \sin \left( \frac{1}{x} \right).
\end{align*}

Report 1. Note that some of the above functions exhibit some pathological points, where the linear
approximation does not seem to be doing a very good job of approximating our function. When this
happens, try to explain why this is the case.

3 Experimenting with the Taylor Expansion

3.1 The Taylor-Young formula

Let \( f \) be \( n \) times derivable at \( a \). Then

\[
\lim_{t \to a, t \neq a} \frac{1}{(t - a)^n} \left[ f(t) - f(a) - \sum_{k=1}^{n} \frac{(t - a)^k}{k!} f^{(k)}(a) \right] = 0.
\]  

(1)

where \( f^{(k)} \) denotes the \( k \)-th derivative of \( f \).

Another way to write this would be that, on a small neighborhood \( V \) of \( a \), we would have

\[
\forall t \in V, \quad f(t) = f(a) + \sum_{k=1}^{n} \frac{(t - a)^k}{k!} f^{(k)}(a) + o((t - a)^n).
\]

3.2 The Taylor-Lagrange formula

This formula is interesting because it is not a “local” formula, unlike Taylor-Young since it applies
over any interval.

Let \( f \) be a function defined over \([a, b]\) and \( n \) times derivable over \((a, b)\). Then there exists \( c \in (a, b)\)
such that

\[
f(b) = f(a) + \sum_{k=1}^{n-1} \frac{(b - a)^k}{k!} f^{(k)}(a) + \frac{(b - a)^n}{n!} f^{(n)}(c).
\]  

(2)
### 3.3 The \( n \)-jet of a function \( f \) around a point \( x_0 \)

The \( n \)-jet of function \( f \) at \( x_0 \) is defined by taking the terms of the Taylor expansion of \( f \) at \( x_0 \) up to degree \( n \) only, that is, by ignoring all terms of degree higher than \( n \):

\[
j^n f : \mathbb{R} \longrightarrow \mathbb{R} \quad x \longmapsto j^n f(x) = f(x_0) + \sum_{k=1}^{n} \frac{1}{k!} f^{(k)}(x_0) \cdot (x - x_0)^k.
\]

So, for example, the 0-jet of \( f \) at \( x_0 \) is simply the constant function that always returns \( f(x_0) \). The 1-jet of \( f \) at \( x_0 \) is the linear approximation of \( f \) around \( x_0 \) that we studied in Section 1.

Intuitively, we would expect that, as we increase the order of the jet, the approximation gets better. This is correct, but again, only on a small neighborhood around \( x_0 \).

### 3.4 Plot the jets

**Maple 2.**

You should write a Maple procedure that does the following, assuming that a function \( f : \mathbb{R} \longrightarrow \mathbb{R} \) has been properly defined:

- Take as parameters:
  - a function \( f \),
  - a value for \( x_0 \),
  - a (small) number \( h \) indicating the size of the interval \( (x_0 - h, x_0 + h) \) on which the study should take place,
  - an integer \( n > 0 \) indicating the maximum jet order that should be considered.

- Plot \( f \) over the interval \([x_0 - h, x_0 + h]\).

- For each \( k, 0 \leq k \leq n \), plot together \( f(x) \) and \( j^n f(x) \) over the interval \([x_0 - h, x_0 + h]\). Use a different color for \( f \) and the \( k \)-jet.

**Maple 3.** Run your procedure for each of the following functions and chosen values of \( x_0 \):

\[
\begin{align*}
g_1(x) &= \frac{x^5 - 1}{5x^2 + 1}, \\
g_2(x) &= \cos(\sin x), \\
g_3(x) &= x^3 \sin \left( \frac{1}{x} \right),
\end{align*}
\]

Experiment with different values of \( h \) and \( n \) to get a better idea of what is going on.
3.5 What about the residual term?

If we look back at two forms of the Taylor formula, we see that the second formula (Taylor-Lagrange) tells us something more interesting about the difference between $f$ and its $n$-jet. The Taylor-Young expression tells us that this difference, at $x_0 + \epsilon$ is $o(\epsilon^n)$, that is, “negligible compared to $\epsilon^n$.” In effect, the Taylor-Young formula tells us more or less that near $x_0$, $f$ is equal to its $n$-jet, plus something really small (of course the exponent $n$ gives some idea of how small that term is). The Taylor-Lagrange is much more informative: It tells us in particular about the minimum and maximum possible value of this small term.

Maple 4.

Now, you should be convinced by now that Taylor-Young is indeed correct, but what about Taylor-Lagrange? Can you think of a way to verify graphically that this formula is correct for the different functions $f$ that we have experimented with in the previous section?

Let me try to elucidate a bit what I want here. There are a number of situations in mathematics, applied mathematics, and even engineering (in the large), where we are just happy with the knowledge that a solution exists, without feeling compelled to precisely determine what this solution is. We know that there is a solution, we know a range for that solution, we know that if we really wanted to, we could compute it (not to say that this would necessarily be easy and fast, just that we could do it), but we are happy just benefiting from the consequences of the existence of such a solution. Sometimes, for example, such a solution exists because we have done something (for example, adjust some parameters) to put ourselves under conditions where the solution exists, and it is nice to have a way to verify that we have done the right adjustments, and that a solution indeed exists. Very often the verification of the existence of a solution is much easier/cheaper/faster than the computation of the solution itself.

Now, this being said, look again at our Taylor-Lagrange formula 2. We are saying that, $a$ and $b$ having been chosen, there always exists a $c$ between $a$ and $b$ so that this equation is verified. The question is: Can you verify this for a few examples. Note: This is a verification, not a proof!

4 Estimating the Derivative at a Point

4.1 Linear approximation

Since the definition of the derivative of $f$ at a point $x$ is

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

We could simply use the value taken by the right side for a “small” value of $h$ as an approximate value for $f'(x)$:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \text{ for some “small” } h.$$  

As we discussed in class, the problem with this approximation is that, if $h$ is too large or too small, then the value computed is incorrect. In the first case, the problem is mostly due to “truncation error” (due to the algorithm). When $h$ becomes too small, however, the rounding error becomes dominant.
Maple 5.
Pick your favorite 1D function and use Maple to plot the absolute and relative error of the above approximation against the value of $h$.

### 4.2 An easy improvement

We know from the Taylor-Lagrange formula (and we have verified it in the preceding sections of this lab) that there exists a $z \in [x, x + h]$ such that

$$f(x + h) = f(x) + h f'(x) + \frac{h^2}{2} f''(z).$$

From the above formula, we can see that

$$f'(x) = \frac{f(x + h) - f(x)}{h} + \frac{h}{2} f''(z),$$

which tells us that the error of our linear approximation is $O(h)$. Indeed $f''(z)$ is a constant in that equation, and therefore the error term is a linear function of $h$. We could have used a “small oh” notation instead, to say that our error term is $o(1)$, which means that the error term is negligible (for a small $h$) relative to constant terms. Indeed, we can get the error term to be as small as we want by taking down the value of $h$.

We can get the error term to be as small as we want... but it does not get there very fast. $O(h)$ is not bad, but it is not great either. What we would prefer is to get an error term that is $O(h^2)$, or even higher. It turns out that this is in fact quite easy to do, but for this we will need to increase first the degree of our expansion. We know that there exists a $z_1 \in [x, x + h]$ such that

$$f(x + h) = f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(z_1).$$

We can obtain a similar expression on the left side of $x$: There exists a $z_2 \in [x - h, x]$ such that

$$f(x - h) = f(x) - h f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(z_2).$$

By subtraction we obtain

$$f(x + h) - f(x - h) = 2h f'(x) + \frac{h^3}{6} \left( f'''(z_1) - f'''(z_2) \right),$$

which gives us the following revised approximation for $f'(x)$:

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} + \frac{h^2}{6} \left( f'''(z_1) - f'''(z_2) \right).$$

Have we gained anything in the process? We have gained a lot: The error for this approximation is now $O(h^2)$!

Maple 6.
Pick your favorite 1D function and use Maple to plot the absolute and relative error of the above approximation against the value of $h$. 
4.3 Further improvements: Richardson extrapolation

Why stop with $O(h^2)$? We can push the same trick a lot further. Let us write the $n$th degree Taylor-Young (local, approximate) expansion for $f(x + h)$ and $f(x - h)$:

$$
\begin{align*}
  f(x + h) &= f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \cdots + \frac{h^n}{n!} f^{(n)}(x) + O(h^{n+1}), \\
  f(x - h) &= f(x) - h f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \cdots + (-1)^n \frac{h^n}{n!} f^{(n)}(x) + O(h^{n+1}).
\end{align*}
$$

By subtraction, we obtain

$$
\begin{align*}
  f(x + h) - f(x - h) &= 2h f'(x) + \frac{h^2}{3} f''(x) + \cdots + \frac{2h^{2k+1}}{(2k + 1)!} f^{(2k+1)}(x) + O(h^{2k+3}),
\end{align*}
$$

where I have used the form $2k + 1$ to indicate that only odd terms remain in this expression. Note that the “big Oh” terms don’t “cancel out,” instead, the order remains the same: $O(h^k) - O(h^k) = O(h^k)$. To convince yourself that this is correct, think for example of the difference $5h^2 - 3h^2$. Both terms in the difference are $O(h^2)$, and so is the difference. Note, however, that since even terms do cancel out, we “jump” to the next odd term for our “big Oh” error term. We can now write the expansion of $f'(x)$ as follows:

$$
\begin{align*}
  f'(x) &= \frac{1}{2h} (f(x + h) - f(x - h)) - \frac{h^2}{6} f''(x) + \cdots + \frac{h^{2k}}{(2k + 1)!} f^{(2k+1)}(x) + O(h^{2k+2}).
\end{align*}
$$

Now here comes the main trick (pay close attention): All the terms $f^{(i)}(x)$ are constants in this expression. They only depend on the function $f$ and the point $x$ where we want to calculate $f'(x)$. The term $h$, on the other hand, is a “free variable” in this equation, since it can take any value in a small interval about 0. To make this point even more explicit, let me define

$$
\phi(h) = \frac{1}{2h} (f(x + h) - f(x - h)),
$$

so that we can write

$$
\begin{align*}
  f'(x) &= \phi(h) + a_2 h^2 + a_4 h^4 + a_6 h^6 + \cdots
\end{align*}
$$

What the above expression makes clear (I hope) is that we have, on the left side, the constant term that we want to evaluate. On the right side we have a function of $h$ that we can easily compute, and an error term that has the form of a Taylor expansion containing only even powers of $h$.

Now, the very interesting property of this type of expression is that by evaluating $\phi$ for different values of $h$ we can manage to get increasingly better approximations of $f'(x)$. Try this:

$$
\begin{align*}
  \phi(h) &= f'(x) - a_2 h^2 - a_4 h^4 - a_6 h^6 - \cdots, \\
  \phi(h/2) &= f'(x) - a_2 (h/2)^2 - a_4 (h/2)^4 - a_6 (h/2)^6 - \cdots.
\end{align*}
$$

We observe that

$$
\begin{align*}
  \phi(h) - 4\phi \left( \frac{h}{2} \right) &= -3f'(x) - \frac{3a_2}{4} h^4 - \frac{15a_6}{16} h^6 - \cdots
\end{align*}
$$
We divide by 3 and re-arrange some terms to get
\[
\frac{4}{3} \phi \left( \frac{h}{2} \right) - \frac{1}{3} \phi(h) = f'(x) + \frac{1}{4} a_4 h^4 + \frac{5}{16} a_6 h^6 + \cdots \tag{3}
\]
We are down to a $O(h^4)$ error term! Even better: we can keep applying the same trick over and over to improve the order of the error term. For this, we simply define
\[
\psi(h) = \frac{4}{3} \phi \left( \frac{h}{2} \right) - \frac{1}{3} \phi(h).
\]
Then we have from Equation 3
\[
\psi(h) = f'(x) + b_4 h^4 + b_6 h^6 + \cdots \\
\psi(h/2) = f'(x) + b_4(h/2)^4 + b_6(h/2)^6 + \cdots
\]
Here again, we can combine these two expressions to get
\[
\psi(h) - 16 \psi \left( \frac{h}{2} \right) = -15 f'(x) - \frac{7}{8} b_6 h^6 - \cdots
\]
This time we reach $O(h^6)$:
\[
\frac{16}{15} \psi \left( \frac{h}{2} \right) - \frac{1}{15} \psi(h) = f'(x) + \frac{7}{120} b_6 h^6 + \cdots
\]

4.4 The algorithm

The algorithm proceeds by evaluating the elements $D(n, m)$ of a triangular array $D$:
\[
D[0, 0] \\
D[1, 0] \ D[1, 1] \\
D[2, 0] \ D[2, 1] \ D[2, 2] \\
\vdots \quad \vdots \quad \vdots \quad \vdots \\
D[N, 0] \ D[N, 1] \ D[N, 2] \ \cdots \ D[N, N]
\]
The $D[n, m]$ are successive approximations of $f'(x)$, and $F[N, N]$ is the best among them. The error term corresponding to $D[n, m]$ is
\[
\epsilon(n, m) = O \left( \frac{h^{2(m+1)}}{2^{2n(m+1)}} \right).
\]
You get to decide on suitable values for $N$ and $h$. This is something that you will do by experimentation (trial and error). The important thing is to get a feel for what is going on.

The computation of the terms $D[n, m]$ is very simple:
\[
D[n, 0] = \phi \left( \frac{h}{2^n} \right), \text{ for } n = 0 \ldots N, \\
D[n, m] = D[n, m-1] + \frac{1}{4^m - 1} (D[n, m-1] - D[n-1, m-1]).
\]

Maple 7.
Implement the Richardson extrapolation algorithm and test it for different functions $f$ and different points $x$. Evaluate the error in terms of $h$ and $N$. 

4.5 Second derivative

If you understand the Richardson algorithm, you should have no major problem adapting it to the calculation of a function’s second derivative. The conditions are the same as above: we have a function defined over some interval $I \subseteq \mathbb{R}$, $f : I \rightarrow \mathbb{R}$ that we can evaluate at any point of $I$. What we want is to be able to compute a good approximation of $f''(x)$ for any $x \in I$.

I am not asking you to develop an algorithm (although, if you manage to do so under a form similar to that for $f'$, this will be worth 7 points of extra credit). Using the Taylor expansion of $f$ around $x$ and a method inspired from that we followed for $f'$, propose a formula for computing $f''(x)$ such that the error term is at least $O(h^5)$.

Just as we did for $f'$, you will develop increasingly accurate approximating formulas for $f''(x)$. For each of these formulas, don’t forget to provide the order of the approximation.

Maple 8.

*Use Maple to verify the validity and quality of your approximations.*

5 C++/Java Implementation

For this assignment, you will not be asked to implement the jets, only the basic function class and a few of its subclasses (all hard-coded).

5.1 1.2 Packages (Java) and Namespaces (C++)

Over the semester, we are going to implement a nice little library. I don’t want to overburden you with the issue of producing .jar, .lib, or .dll files yet. For this assignment, you will simply compile your test application against your source code, as you did in CSC211 and CSC212. Still, it is better to take a few early steps that are going to facilitate our transition to a binary library, later on.

5.1.1 In Java

For the Java implementation, this means that we are going to define packages. Rather than developing a single big package, we are going to develop several small packages with a name hierarchy, just the way the JDK does. Examples of packages we are going to develop are:

- csc350Lib.linalg.*
  - csc350Lib.linalg.base will implement basic matrix and vector classes,
  - csc350Lib.linalg.sle will implement classes taking care of the solution of SLEs,
  - csc350Lib.linalg.eigen will deal with eigensystems,
  - csc350Lib.linalg.lls will deal with Linear Least Squares problems,
- csc350Lib.calc.*
- `csc350Lib.calc.base` will implement basic calculus function classes (part of your job in this assignment),
- `csc350Lib.calculus.snle` will implement classes taking care of the solution of systems of nonlinear equations,
- `csc350Lib.calc.optim` will deal with optimization problems,
- `csc350Lib.calc.ode` will deal with Ordinary Differential Equation problems,

- `csc350Lib.random.*`, if we have the time, would deal with random number generation.

One important thing to keep in mind is that, although the naming convention would seem to reflect a hierarchy of some sort, Java does not in fact enforce any such thing: `csc350Lib.calculus.snle` and `csc350Lib.calculus` are just viewed as two identifiers, and Java does not consider the former to be a "sub-package" of the latter (such a notion does not exist in Java). On the other hand, the "wild card" character * gives the appearance of a hierarchy, by allowing us to load a complete ”parent package” (or so it seems):
```java
import csc350Lib.calc.*;
```

I still cannot understand what on Earth possessed the developers of Java to make such a decision, but it is very likely that it is another “hack” of the javac compiler that found its way into the specifications or guidelines of the language.

Still, even if Java itself does not do it that way, it would be a good idea if you did store the source files of your library in a folder hierarchy that reflects the package hierarchy that we envision when we read the names of the packages.

5.1.2 In C++

For the C++ implementation, we are going to define different namespaces. It is very obvious from the syntax of namespace definitions that there is no hierarchy of namespaces. In C++ the namespace is the means through which one can reduce the scope of a class. We will define the following namespaces:

- Linear algebra namespaces
  - `csc350Lib_linalg_base` will implement basic matrix and vector classes,
  - `csc350Lib_linalg_sle` will implement classes taking care of the solution of SLEs,
  - `csc350Lib_linalg_eigen` will deal with eigensystems,
  - `csc350Lib_linalg_l1s` will deal with Linear Least Squares problems,

- Calculus namespaces
  - `csc350Lib_calc_base` will implement basic calculus function classes (part of your job in this assignment),
  - `csc350Lib_calculus_snle` will implement classes taking care of the solution of systems of nonlinear equations,
- csc350Lib_calc_optim will deal with optimization problems,
- csc350Lib_calc_ode will deal with Ordinary Differential Equation problems,

- Other stuff
  - csc350Lib_random_stuff to come, if we have the time, would deal with random number generation.

As for the Java implementation, it would be a good idea if you did store the source files of your library in a folder hierarchy that reflects the hierarchy that we envision when we read the names of the namespaces.

## 5.2 The `Function1D` abstract class

You are going to re-use this class throughout the semester, so make sure that you implement it correctly. For example, in a couple of weeks, you will write a “nonlinear solver” class that finds the zeros of a function that has been passed as an argument (pointer to a `Function1D` object in C++, simply a reference to that object in Java).

Complete specifications for this class and its subclasses have been posted on the course’s “Handouts” web page.

## 5.3 Validation of the C++/Java implementation

It is not sufficient to write code that compiles and does not crash when executed. We must also verify that this code also computes the values it is suppose to compute, that is, correct values.

For that, the best is to compare the values computed by Maple and that computed by your code. In the coming weeks, we will do more of that and discuss how many test calculations are needed. For this assignment, all I want is for you to put into place the mechanisms for this testing, and in particular the I/O of text files in C++/Java and in Maple.

### Maple 9.

*Read in a Maple worksheet the output of your C++/Java calculations and verify that they are correct.*

Information and demo code on file I/O in C++, Java, and Maple will be posted on the “Handouts” page tonight.

## 6 Evaluation

### 6.1 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):

- csc350Lib_calc_optim
- csc350Lib_calc_ode
- csc350Lib_random_stuff

- Other stuff
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.

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:

**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

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 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).