Briefing Notes—Freighter Parameter Estimation

(c) UBC Vancouver, March 2011

IDEA 1: *Boat Physics*

Recall the experimental setup detailed in Physical Lab 3. To simplify notation, let's define

$M = M_{\text{boat}} + M_{\text{cargo}}$, ... the total mass being dragged through the water, and

$m = M_{\text{drive}}$, ... the total mass of the pulley, platform, and driving weights.

(In most cases, M is significantly larger than m.) The manual for Physical Lab 3 explains why the acceleration of the boat along the tank, denoted a, is given by

$$a = \left(\frac{2m}{4M+m}\right) g - \frac{4}{4M+m} F_{\text{drag}}(v). \tag{$*$}$$

To save subscripts later, we will write F instead of F_{drag} from here on.

Average Drag. Experts expect a drag force proportional to the square of the speed, so they complete the specification of $(*)$ using the generic form below. It involves a scalar $K > 0$ not yet known:

$$F(v) = Kv^2.$$

To give F the correct units (Newtons) and to bring in some parameters related to the experiment itself, we expand K in terms of some new dimensionless value $C > 0$:

$$F(v) = \tfrac{1}{2}\rho C S v^2. \tag{2}$$

Here S is the wetted surface area of the boat (in m^2) and ρ is the density of water. A precise theoretical model for the boat's motion will be available as soon as we know C, the "drag coefficient". *Our goal here is to estimate C and related quantities, and then to compare the motion predicted by $(*)$ with the motion measured in the tow-tank.*

Rearranging line (2) gives

$$C = \frac{2F(v)}{\rho S v^2}. \tag{3}$$

We can, in principle, evaluate C from any measured pair of speed v and force $F(v)$. The steady-state speed—call this v_s—is a good choice for this, because the drag force $F(v_s)$ must match the tension force in the tow line, and we know that. Details below.

In the Physical Lab writeup, plugging $v = v_s$ into (3) produces a value of C denoted there by $C_{D,\text{avg}}$. Matlab can help us estimate the value of C for an individual boat, and then to compare the idealized motion calculated from (2) with the observed motion in the lab.

A Variable Drag Coefficient. Maybe we can match the measured motion of the boat better by allowing the drag coefficient itself to depend on the speed. For example, instead of using a *constant* C in equation (2), we might pick some exponent p and try a *function* of the form $C = \alpha + \beta v^p$. The constants α, β, and p would be tuning parameters that we could adjust to help the theoretical solution of $(*)$ match the data collected in the lab. With this form, the total drag force becomes

$$F(v) = \tfrac{1}{2}\rho[\alpha + \beta v^p] S v^2, \tag{4}$$

and equation (3) says

$$C(v) = \alpha + \beta v^p = \frac{2F(v)}{\rho S v^2}. \tag{5}$$

We can still evaluate the right side when $v = v_s$, but now we can use the various steady speeds v_s from different towing experiments to find several values for the function C in (5). We can use those to estimate α, β, and p.

An Aesthetic Issue. The drag coefficient C must be dimensionless, so each term added to produce it must be dimensionless. Parameter α is fine, but the term βv^p offends against good taste because it can only make sense if β carries the units of $(m/s)^{-p}$. For reasons both aesthetic and theoretical, it would be nice to replace this term with one like $\gamma(v/\sqrt{gL})^p$, where L is the length of the boat. Knowing β would be equivalent to knowing γ, because g and L are known and

$$\beta v^p = \gamma \left(\frac{v}{\sqrt{gL}}\right)^p \iff \beta = \frac{\gamma}{(gL)^{p/2}},$$

but γ has the advantage of being dimensionless. Weighing against this choice is the added complication the dimensionless setup would create. For this occasion only, we have opted to put this important general principle second and simplicity of matlab implementation first.

Reconciling Math with Engineering. Boat experts expect the drag force in $(*)$ to be a sum of three velocity-dependent drag forces, as follows.[1]

Total Drag:	$F(v) = F_f(v) + F_s(v) + F_w(v),$	is the sum of
Form Drag:	$F_f(v) = \frac{1}{2}\rho S v^2 \left(\frac{AC_D}{S}\right),$	
Skin Friction Drag:	$F_s(v) = \frac{1}{2}\rho S v^2 \left(\frac{1.327}{\sqrt{\mathrm{Re}(v)}}\right),$	where $\mathrm{Re}(v) \overset{\text{def}}{=} L\lvert v\rvert/\nu,$
Wave-Making Drag:	$F_w(v) = \frac{1}{2}\rho S v^2 \Big(C_w(v)\Big),$	where $C_w(v) \overset{\text{def}}{=} 60\left(\frac{v}{\sqrt{gL}}\right)^p, \ p \approx 4.$

Note that the Reynolds Number, Re, is actually a function of speed. For this experiment, it's probably a big number: using $\nu \approx 10^{-6}$ m^2/s, boat length $L \approx 0.5$ m, and speed $v \approx 0.2$ m/s, we have $\mathrm{Re} = Lv/\nu \approx 10^4$. Thus the *denominator* defining the skin friction drag contains the big number $\sqrt{\mathrm{Re}} \approx 10^2$, and we might expect F_s to be negligible for this experiment. We will soon make the approximation $F_s \approx 0$ (or, informally, send $\mathrm{Re} \to \infty$)—although it might be smart to double-check that this is reasonable at some point in our work.

Plugging the boat-builders' drag formulas into line (5) juxtaposes the form for $C(v)$ suggested here with the form conjectured earlier:

$$C(v) = \alpha + \beta v^p = \frac{2}{\rho S v^2} F(v) = \frac{AC_D}{S} + \frac{1.327}{\sqrt{\mathrm{Re}(v)}} + 60\left(\frac{v}{\sqrt{gL}}\right)^p.$$

These match pretty well if we consider the limit of $\mathrm{Re} \to \infty$ and admit some considerable uncertainty in the reliability of the number 60 in the formulation of the wave-making drag. In fact, offline conversations with the suppliers of function F_w suggest a huge willingness to negotiate not only the precise value of that coefficient, but even the choice of exponent p. We propose to hold a little contest between the choices $p = 1, 2, 3, 4$.

IDEA 2: *Extracting Parameters from Terminal Velocity*

In the lab, the boat accelerates from rest and rapidly reaches a speed at which the drag forces balance the tension in the tow-line. In this case, the net force on the boat is zero, so it travels at a

[1] These expressions assume $v > 0$. To get a sign that opposes the motion even when $v < 0$, replace v^2 with $v\lvert v\rvert$ throughout.

steady speed. Let's call this speed v_s. By definition, when $v = v_s$ we get $a = 0$: substitution in $(*)$ reveals

$$F(v_s) = \frac{mg}{2}.$$

Using this in line (5) gives

$$C(v_s) = \alpha + \beta v_s^p = \frac{mg}{\rho S v_s^2}. \tag{$**$}$$

When p is known and we plug in the known experimental values, line $(**)$ turns into a linear equation for the unknown parameters α and β. To make this clearer still, let's rewrite it as

$$x + cy = b, \qquad \text{where} \qquad x \overset{\text{def}}{=} \alpha,\ y \overset{\text{def}}{=} \beta,\ c \overset{\text{def}}{=} v_s^p,\ b \overset{\text{def}}{=} \frac{mg}{\rho S v_s^2}. \tag{$\ddagger$}$$

Each towing experiment provides via $(\ddagger)$ the equation of a straight line passing through the point (α, β) of the (x, y)-plane. We expect the intersection point to satisfy both $\alpha > 0$ and $\beta > 0$.

Cargo Mass Influence. The boat's wetted surface area S, projected frontal surface area A, and effective length L should be the same for all three light-cargo trials, because nothing about the boat changes. We can hope for some consistency between the equations labelled $(\ddagger)$ in this case, and optimistically predict a meaningful intersection point (α, β).

Reloading the boat with heavier cargo will push it deeper into the water. Perhaps the values of S, A, and L will change, and perhaps this will influence appropriate values for the values (α, β). We suggest analyzing the two cargo regimes independently; later comparison of the outcomes will be interesting.

IDEA 3: *Overdetermined Linear Systems*

Suppose p is known, e.g., $p = 2$. Imagine making n towing experiments with the light cargo setup, calculating the coefficients, and generating n equations:

$$x + c_i y = b_i, \qquad i = 1, 2, \ldots, n. \tag{6}$$

Geometrically, each one describes a straight line in the (x, y)-plane. If the drag force postulated in (4)–(5) is correct, each of these lines should pass through the point $(x, y) = (\alpha, \beta)$ we are looking for. But in truth lines (4)–(5) are little better than wishful thinking, and if $n > 2$ the experimental lines will not all go through the same point. We need some plausible way to predict a reasonable point of approximate intersection. Matlab has a sensible approach built-in.

To set it up, we write the system (6) in vector-matrix form, taking $n = 3$ for simplicity:

$$\begin{bmatrix} 1 & c_1 \\ 1 & c_2 \\ 1 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{7}$$

Writing `A` for the coefficient matrix on the left and `b` for the column vector on the right, the Matlab command `A\b` computes the "least-squares solution vector" for the overdetermined linear system $A\mathbf{x} = \mathbf{b}$. This is the best mathematical prediction of the point (α, β). (Read the help on the backslash operator for a fuller explanation.) Please don't use it without thinking, though: see Idea 5, below.

IDEA 4: *Multi-Pane Plots in Matlab*

Matlab's `subplot` command provides an easy way to generate multi-pane plots whose horizontal axes line up precisely. The numbering scheme is consistent with matrix labelling: to create a single figure with mn sets of coordinate axes arranged in an grid having m rows and n columns, and then to focus your drawing commands on the axes numbered p, say `subplot(m,n,p)`. So if you want to create a single figure with two plotting zones, stacked one above the other, and you want to draw in the top zone, say

```
subplot(2,1,1);
```
(†)

All normal plot commands can be used, and they all apply to the top sketch in the two-pane figure. To switch to drawing in the bottom zone, just give the command

```
subplot(2,1,2);
```
(‡)

You can change the focus between zones whenever you like by re-entering one of (†) or (‡). Say `help subplot` if you need more details.

Sometimes you need to adjust the interval shown on the horizontal axis of the current plot. You can read this interval into a 2-element vector by saying `get(gca,'XLim')`. Conversely, if you want to insist that the horizontal axis runs from -2 to 17, put these limits into a 2-element vector and use it as follows:

```
set(gca,'XLim',[-2,17]);
```

The vertical axis can be interrogated and adjusted in the same way: just change `XLim` to `YLim`.

IDEA 5: *Determination, Persistence, and Progress*

You may find that the three lines described in Idea 2 don't come anywhere near having a reasonable intersection point, and the least-squares solution produced by Matlab's backslash operator is just nonsense. (For example, a solution with either $\alpha < 0$ or $\beta < 0$ is simply unusable.) If this happens to you, it's time for some creative, but honest, fabrication. Just pick some reasonable positive numbers for the (α, β)-coordinates of the intersection point and use them. This is sad, but it will let you demonstrate some Matlab skills and keep making progress instead of getting stuck. Truth in science is essential, though, so if you have to resort to simply making something up, you must clearly say so in your lab writeup. (Being honest about bad data will not cost you any marks; making stuff up and lying to conceal the fact will cost you many marks (and possibly your soul)!)

IDEA 6: *Global Variables in Matlab*

Here's some of what Matlab has to say in response to "`doc global`":

> "`global X Y Z` defines `X`, `Y`, and `Z` as global in scope.
>
> Ordinarily, each MATLAB function, defined by an M-file, has its own local variables, which are separate from those of other functions, and from those of the base workspace. However, if several functions, and possibly the base workspace, all declare a particular name as global, they all share a single copy of that variable. Any assignment to that variable, in any function, is available to all the functions declaring it global."

In our boat study, the physical parameters of the boat are needed in every script and function we create. To make it easy to apply the same tools to different configurations and scenarios, it's nice to have just one reference copy of each parameter, and to share that value with all functions that need it. (The alternative would require checking each occurrence of every parameter in every script and every function, every time you wanted to adjust one!) Global variables are good for this.

Global variables have disadvantages, too: if some function changes the value of a global variable, all the other functions that use it will be affected. If the change was unwanted, it can be very hard to figure out which one of all the functions currently active contains the error. Successful use of global variables requires strict limits on where they are allowed to change, and clear identifiers discouraging accidental overwriting. To help with this, it's common to identify global variables as special and draw attention to them aggressively by giving them `LONG_NAMES_IN_CAPS`. Then we agree that no ordinary function will ever change the value of a global variable: we will use a special `setup` script to initialize the global variables, and all other scripts and functions will only read, never overwrite, global values. The scripts supplied with the lab follow this practice.

IDEA 7: *Predicting motion with* `ode45`

Here is a quick refresher on `ode45`, one of Matlab's many built-in differential equation solvers. The `ODE` in its name stands for "Ordinary Differential Equation," and the `45` advertises the style of algorithm used inside: the so-called "an explicit Runge-Kutta $(4, 5)$-formula". A fuller introduction to this command was provided in the Prelab for Computer Lab 4 of MECH 221. You can still find that on Vista, if the short summary below is not enough.

To coax Matlab into solving the second-order scalar initial-value problem

$$\ddot{x}(t) = a(x(t), \dot{x}(t)), \qquad x(0) = x_0,\ \dot{x}(t) = v_0,$$

we convert it to this first-order initial-value problem whose unknown is a *column vector*, namely $\mathbf{u}(t) = (x(t), v(t))$:

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ a(x(t), v(t)) \end{bmatrix}, \qquad \begin{bmatrix} x(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ v_0 \end{bmatrix}.$$

This has the generic form of $\dot{\mathbf{u}}(t) = \mathbf{F}(t, \mathbf{u}(t))$, $\mathbf{u}(0) = \mathbf{u}_0$, that Matlab's function `ode45` is designed for. To put `ode45` to work, you must define a function (possibly named `F`) that takes a scalar input `t` and a 2-element column vector input `u`, so that `F(t,u)` returns the 2-element column vector `[u(2);accel(u(1),u(2))]`.

If you want to start your theoretically-ideal boat with initial position 0 and initial velocity 0, then you should define the inital vector

$$\texttt{u0} = [0; 0]$$

To track the boat from time 0 to time 35, you could define the vector variable `Tint = [0,35]`. The simplest solution command would then be

$$[\texttt{TlistUlist}] = \texttt{ode45}(\texttt{@F}, \texttt{Tint}, \texttt{u0}); \tag{13}$$

In response, Matlab would return a column vector `Tlist` containing some number N times between `t0` and `t1` and a corresponding $N \times 2$ matrix you can interpret as a pair of 2 tall column vectors. The first column, `Ulist(:,1)`, lists the values of x at the times in `Tlist`; the second column, `Ulist(:,2)` returns the computed values of v at the times in `Tlist`. In the basic form of line (13), the number N and the precise times that end up in Tlist are chosen automatically by Matlab. Typically they are not evenly spaced: Matlab has smart methods that position the nodes for the best compromise between efficiency and accuracy.

To gain complete control over the list of times returned by `ode45`, decide in advance which instants you are interested in. Create a vector `MyTlist` whose elements are the times t_k at which you want to know the values of $\mathbf{u}(t_k)$. Then simplyl use the long vector `MyTlist` instead of the short vector `Tint` in line (13) above. That is, say

$$[\texttt{TlistUlist}] = \texttt{ode45}(@\texttt{F}, \texttt{MyTlist}, \texttt{u0}); \tag{14}$$

The output vector `Tlist` should be a column containing an exact copy of `MyTlist`, so that the corresponding columns in `Ulist` are the desired instantaneous values of $\mathbf{u}$.