8.5 A numerical example

Perhaps the best way of seeing how to use the MATLAB ODE solvers is to construct a simple model and work with it. We hope the application here will clarify how we use the tools, and incidentally, something about how the ocean works. Additionally, we will mention that in this example model we are going to use **x** as our unknown variable, i.e. the vector of phosphorus concentrations in time. We do this for two reasons: first, it is fairly standard in the modeling literature to use **x** for the thing you want to know; and second, we want to make the derivation of this model consistent with our usage of the same model in later chapters.

8.5.1 An example: a two-box global ocean phosphate model

We start with a two-box model of the global ocean made popular by Wally Broecker among others (Broecker and Peng, 1982), where we have two reservoirs (boxes) representing a surface layer and the deep ocean (see Fig. 8.5). In this model the global ocean cycle of phosphorus is relatively simple, with phosphorus added to the surface ocean by

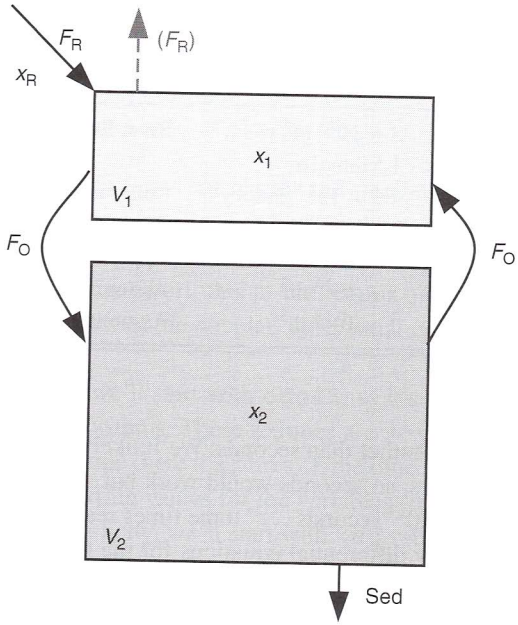

Figure 8.5 A two-box global ocean phosphate model showing water fluxes. In the implementation of this model in the text, we use the following parameters: box volumes $V_1 = 3 \times 10^{16}\,\mathrm{m}^3$; $V_2 = 1 \times 10^{18}\,\mathrm{m}^3$; river water flux $F_R = 3 \times 10^{13}\,\mathrm{m}^3\,\mathrm{y}^{-1}$; and overturning water flux $F_O = 6 \times 10^{14}\,\mathrm{m}^3\,\mathrm{y}^{-1}$.

river inflow and removed by sedimentation in the deep ocean. For the purposes of this model, phosphorus predominantly exists in the ocean in the form of inorganic phosphate, although it spends time within organic pools (ignored here for the moment). There is a net overturning of water in our ocean model, because in the real ocean, water sinks to depth in the polar regions and is balanced by an equivalent upwelling elsewhere. In fact, to balance the river input of water, there must be an equivalent amount of evaporation from the surface of the ocean (river flow must be balanced hydrologically by rainfall that finds its ultimate source in evaporation from the ocean surface). Since flowing water carries phosphate and evaporating water does not, our model river is a source of phosphate to the surface box.

To model the system, we need to put numbers on the river phosphate concentration, water fluxes, and box volumes (concentration changes occur as a result of material fluxes divided by volumes). The phosphate fluxes associated with overturning water and river inflow are given by the water fluxes times the concentrations of phosphate in the water. Table 8.2 gives the appropriate numbers for our toy model.

Note that we have striven to maintain a consistent system of units. This is vital in a model (or any ocean) calculation. We've chosen m^3 for volume, $\mathrm{m}^3\,\mathrm{y}^{-1}$ for a water flux, mmol for phosphorus amount, and mmol m^{-3} for phosphorus concentration. The residence time of phosphorus in the oceans is probably of order tens of thousands of years, so we've

Table 8.2 *Global two-box phosphorus model parameters*

Name	Symbol	Flux or volume	Description
River water flux	F_R	3×10^{13} m^3 y^{-1}	River flux $\sim$ 1 Sv = 10^6 m^3 s^{-1}
P conc. in river water	x_R	1.5 mmol m^{-3}	
River P flux		4.5×10^{13} mmol y^{-1}	Conc. $\times$ water flux
Overturning water flux	F_O	60×10^{13} m^3 y^{-1}	Overturning $\sim$20 Sv = 20×10^6 m^3 s^{-1}
Surface area of oceans		3×10^{14} m^2	Approximately 70% of Earth's area
Surface box volume	V_1	3×10^{16} m^3	Assuming 100 m depth
Deep box Volume	V_2	1×10^{18} m^3	Assuming 3300 m depth

chosen years as our time unit rather than seconds. We'll likely be integrating our ODEs for thousands to millions of years, so seconds would work but would be a little inconvenient (try saying, "integrating for 10^{13} seconds ..." three times really quickly).

So let's set up the ordinary differential equations for the two reservoirs. The equations are:

$$\frac{dx_1}{dt} = \frac{(F_R x_R - F_O x_1 + F_O x_2)}{V_1} \tag{8.41}$$

$$\frac{dx_2}{dt} = \frac{(F_O x_1 - F_O x_2 - \text{Sed})}{V_2} \tag{8.42}$$

where x_1 and x_2 are the phosphorus concentrations in boxes 1 and 2 respectively at time t. Notice that the phosphorus mass budget is maintained because when we take an amount of phosphorus out of one box, we put it in the other; we say these two ODEs are coupled. In (8.42) Sed is the sedimentary phosphate flux in mmol y^{-1}. Inasmuch as we don't quite know what to do with this flux just yet, we'll arbitrarily set it to 1% of the downwelling flux of phosphorus from the surface box (not a very clever choice, but it's a start). Now the system of equations is given by:

$$\frac{dx_1}{dt} = \frac{(F_R x_R - F_O x_1 + F_O x_2)}{V_1} \tag{8.43}$$

$$\frac{dx_2}{dt} = \frac{(F_O x_1 - F_O x_2 - 0.01 F_O x_1)}{V_2} \tag{8.44}$$

We now need to code this in MATLAB in order to use the ODE solver. We write a MATLAB function, which we'll call `phos2.m`.

```
function dxdt=phos2(t,x)

V=[3 100] * 1e16;          % volume of reservoirs in m3
dxdt=zeros(2,1);           % Initialize output as a column vector
FO = 6e14;                 % overturning water flux in m3 per year
FR = 3e13;                 % river water flux in m3 per year
```

```
xR = 1.5;                % river water P concentration in mmol per m3
Sed = 0.01*FO*x(1);      % sedimentary loss of P in deep box

dxdt(1) = (FR*xR - FO*x(1) + FO*x(2))/V(1);
dxdt(2) = (FO*x(1) - FO*x(2) - Sed)/V(2);
```

Notice how we've split out the terms for clarity and flexibility, so if we want to change the model in the future, it'll be easier (e.g. to change the water overturning flux we need to edit just one number in the fourth line instead of four numbers in the eighth and ninth lines). Note also that, as a `function`, this m-file returns a column vector of two values corresponding to the derivatives of the coupled differential equations. Now let's integrate the model.

Following our own advice we'll start with `ode45`, as it's generally the most accurate, efficient and stable of the algorithms. These routines are very easy to use. All we need to do is to specify the function name (here it's `phos2`), the time span for integration (let's say a mere 10^4 years), and the starting values for the variables (remember, it's an initial value problem). For the sake of simplicity, we'll start with zero values. Note also that the initial value and the output of the function must be column vectors.

But why 10^4 years? Well, to make things simple we decided to set the initial concentrations to zero, and let the river fill the ocean with phosphate. That being the case, we know from geochemistry that we will need to let the model run for at least the equivalent of one to several water overturning residence times before the model will show any detectable change. We can easily put bounds on the residence time of water in the deep model box with respect to overturning equal to $V_2/F_O \sim 1.67 \times 10^3$ years and the residence time of the whole model with respect to river inflow equal to $(V_1 + V_2)/x_R \sim 3.43 \times 10^4$ years. Based on these back of the envelope calculations, 10^4 years is a good compromise, long enough to allow the model to do something without committing to a very long integration. So we type the following:

```
x0=[0 0]';
[t,X]=ode45('phos2',1e4,x0);
plot(t,X)
```

Note that `ode45`, like the other integrators, returns two things: a column vector of the times, `t`, in the integration steps, and a multi-column matrix of the variable outputs, `X` (one column per variable, each row corresponds to a time step). Additionally, since it is a matrix of many rows by two columns, `X` is written capitalized. Also, we've chosen to integrate only for a shortish period of time, as an exploratory step. This is a prudent, standard approach.

In plotting the results, we see that the model has not reached equilibrium after such a short time (in Fig. 8.6a the concentrations are still increasing and have not leveled off yet). No real surprise there, so we run it again with a longer integration time span:

```
x0=[0 0]';
[t,X]=ode45('phos2',1e6,x0);
plot(t,X)
```

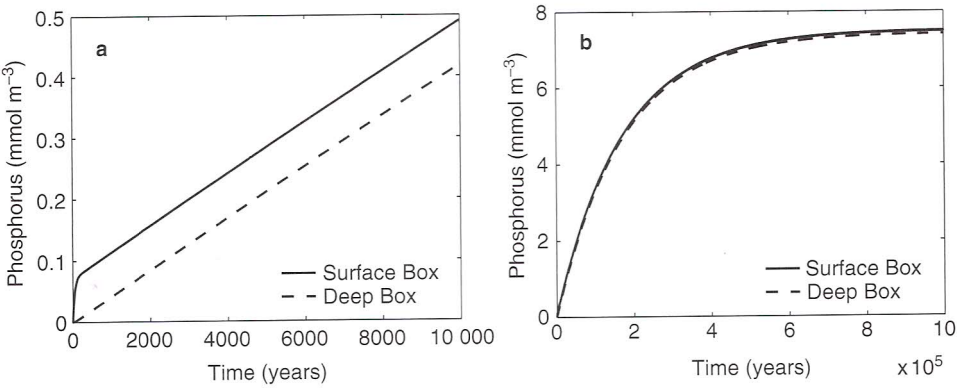

Figure 8.6 Results of our first two-box global phosphate model integration, (a) $t_f = 10^4$ y and (b) $t_f = 10^6$ y.

Figure 8.6b shows the results. Not surprisingly, given the simple structure, the model has the two boxes with rather similar concentrations. The surface box is slightly higher than the lower box. The sign of the difference is also not surprising, as the input is in the surface box, and the sink is in the lower box. The size of the difference makes sense too; the removal rate is about 1% per box turnover, which gives a timescale large relative to the overturning timescale ($V_2/F_O \sim 1670$ years) so that they should be very similar in concentration.

This, however, is not realistic as it disagrees with two things we know about the ocean. First, biological production occurs in the surface ocean, using up phosphate (plus other nutrients) and resulting in a particulate flux to the deep water. Most of this particulate flux is respired in the deep ocean but a small fraction of it ends up in the sediments. Second, as a consequence surface phosphorus concentrations in the real ocean are generally much lower than those in deep waters. Thus our model is not well formulated. Let's recast the equations in a slightly more realistic fashion.

$$\frac{dx_1}{dt} = \frac{(F_R x_R - F_O x_1 + F_O x_2 - \text{Production})}{V_1} \tag{8.45}$$

$$\frac{dx_2}{dt} = \frac{(F_O x_1 - F_O x_2 + 0.99 \times \text{Production})}{V_2} \tag{8.46}$$

$$\text{Production} = \frac{(x_1 V_1)}{\tau_x} \tag{8.47}$$

What we've done is assumed that phosphorus is removed from the surface box at a rate proportional to the amount of phosphorus in the box, $x_1 V_1$ (not a bad idea, actually, since biological production is largely nutrient limited). We've characterized this removal rate with a time-constant (τ_x) that might be visualized as the lifetime of phosphorus in the surface ocean due to biological uptake. In the deep box, this loss magically reappears (by oxidation of the falling particles), but it is not 100% efficient as some particles make it to the sediments and are lost from the system. In fact, this results in the system approaching

steady state, i.e. $d\mathbf{x}/dt = 0$ ($\mathbf{x}$ is a two-element array representing the phosphorus concentration in both boxes), as the sedimentary loss must balance the river input. So we rewrite the m-file to add the biology (calling it phos2b.m):

```
function dxdt=phos2b(t,x)

V=[3 100] * 1e16;          % volume of reservoirs in m3
dxdt=zeros(2,1);           % Initialize output as a column vector
FO = 6e14;                 % overturning water flux in m3 per year
FR = 3e13;                 % river water flux in m3 per year
xR = 1.5;                  % river water P concentration in mmol per m3
Tau = 100;                 % 100 year residence time of P in the surface
Prodtvy = x(1)*V(1)/Tau;   % Production
ReminEff = 0.99;           % Remineralization efficiency

dxdt(1) = (FR*xR - FO*x(1) + FO*x(2) - Prodtvy)/V(1);     % surface box
dxdt(2) = (FO*x(1) - FO*x(2) + ReminEff*Prodtvy)/V(2);    % deep
```

and we integrate it with:

```
[t,X]=ode45('phos2b',3e6,x0);
```

to get Fig. 8.7 which shows the surface box has lower concentrations. Did you notice that the model took a little longer to run than the previous experiment (perhaps half again as long)? We increased the integration period by a factor of three (1×10^6 to 3×10^6 years) so

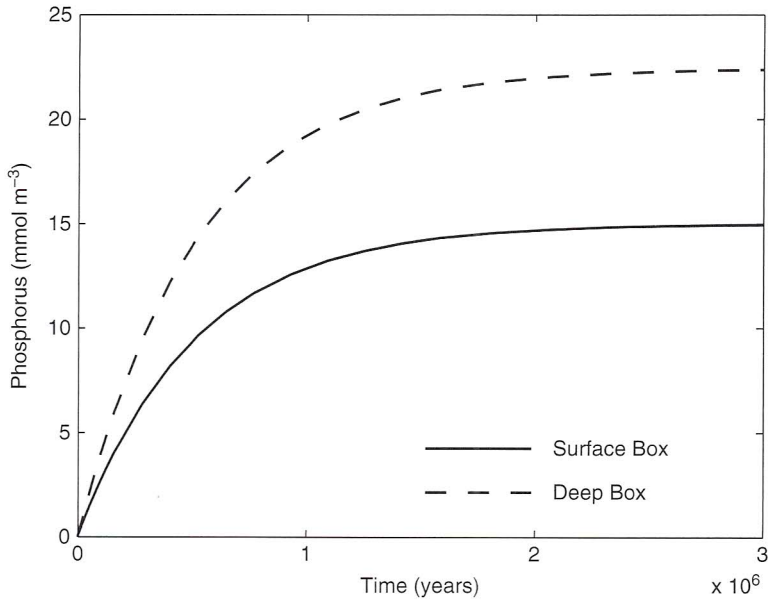

Figure 8.7 Our two-box global phosphorus model with more realistic biological removal of phosphorus from the surface box.

a half increase in execution time seems reasonable. Doing a `whos` shows that the number of time steps was larger, which fits (i.e. the computer took longer because it computed more time steps).

Now we suspect that the residence time of phosphate in the surface ocean is in fact a lot shorter than 100 years. Try running the experiment again, this time setting τ_x to 1 year. What you'll find when you do this is that `ode45` takes forever to run the code. In fact, we had to kill the program with the task manager (Windows) or with `kill -9 PID` (UNIX). We finally got it to complete the integration by putting it on a really fast computer and letting it churn away for a long time. What happened? We didn't increase the computational load in the equations significantly. If you do a `whos` after running the code for a 3×10^6 year run, you find $\sim 3.7 \times 10^6$ time steps (vs. $\sim 110\,000$ for the 100 year time-constant case). It appears that the integrator has to choose smaller time steps for stability. Why `ode45` needs to take so many more steps is the topic of the next section.