**Mathematical Approach for Iterative Coupling of External Models**

***K.W. Little, RTI International***

***August 2012***

This paper presents a numerical technique for coupling external models where some outputs of one model are inputs to the other and vice-versa. “External” means that the models are distinct (separate computer code) from each other and, when either runs, the other is unaware of those results, i.e. each is a black box with respect to the other. This approach would work for multiple (> 2) models as well. We consider first the general problem, then present a Newton’s method technique for coupling the models, then include a simplified numerical example for a proof-of-concept.

1. ***General Problem***

Consider a coupled system, e.g. groundwater/surface water flow, as in Figure 1. W1 is the external input (external to the coupled system) to subsystem 1 and W2 is the external input to subsystem 2. Let a\*S1 represent a subsystem 1 output that is an input to subsystem 2. Let b\*S2 represent a subsystem 2 output that is an input to subsystem 1. “a” and “b” are parameters. S1 is the state variable representing the status (flow, loading, concentration, etc.) of subsystem 1 (including the effect of the coupled input/outputs) and S2 is the state variable representing the status of subsystem 2

bS2

aS1

S1

S2

W1

W2

Subsystem 2

Subsystem 1

**Figure 1. Coupled System**

The mass balance equations for subsystems 1 and 2 are

 (1)

 (2)

***1.1 Holistic Solution***

Assume that we were developing a new model from scratch for the coupled system. We would “know” (have access to) the functional forms of equations 1 and 2. Equations 1 and 2 are two equations (linear in this simple example) in 2 unknowns. The unknowns are the state variables, S1 and S2. The inputs, W1 and W2 are assumed known, and the coupling parameters, a and b are assumed known.

The new model would simply calculate a simultaneous solution of equations 1 and 2 using a simultaneous linear solver. If we let:

W1 = 10

W2 = 20

a = b = 1

then the holistic, simultaneous solution (achieved by algebraically manipulating the 2 equations in this case) is

S1 = 40/3

S2 = 50/3

***1.2 The Separate-Models’ Problem***

Suppose now that you don’t have access to the functional forms of the models for systems 1 and/or 2. All you have is computer code for each subsystem that, given each subsystem’s individual inputs, generates the outputs. The computer code is a black box. Under this scenario, let’s represent subsystem 1’s output as

S1= f1(W1,aS2) (3)

And subsystem 2’s output as

S2 = f2(W2,bS1) (4)

Where “f1” and “f2” represent the (probably unknown) functional forms of the respective models. Thus, if you knew the value of aS2 (in addition to W1), then you would run model 1 and generate S1. If you knew aS1 (in addition to W2), then you would run model 2 and generate S2. The problem of course is that you don’t know these values. Even if you know the parameter a and b values, you still don’t know S2 (for running model 1) or S1 (for running model 2).

How can you achieve the true (holistic) solution with only the 2, uncoupled, executable-code-only models?

***1.3 Newton’s Method Iterative Approach***

Newton’s method is an iterative technique for finding the “roots” of one or more simultaneous, nonlinear equations. It can be used to solve our Separate-Models Problem. In this discussion, we assume we have computer programs represented by equations (3) and (4) and have developed a separate, “manager”-type program that can fire off both models (i.e., equations (3) and (4)), individually and iteratively.

Expressing equations (3) and (4) in a form suitable for Newton’s method we have

g1(S1,S2) = S1 – f1(W1,S2) = 0 (5)

g2(S1,S2) = S2 - f2(W2,S1) = 0 (6)

where “g” denotes the implicit (set equal to zero) functional forms of equations (3) and (4). We are seeking the roots of equations (5) and (6), i.e. the values of S1 and S2 that results in zeros for the implicit functions.

It might be useful to reflect for a moment on why satisfying equations (5) and (6) is equivalent to solving our holistic problem (equations (1) and (2)). For the holistic problem we are solving two (linear) equations in 2 unknowns. Assuming the problem is well-formed (the equations are linearly independent) the solution is unique; there is only one combination of S1 and S2 that will satisfy both equations. Consider the system of equations given by equations (5) and (6). These are the same equations as in the holistic model; the difference is that we are representing the (probably unknown) functional forms as generic functions, implemented via computer code, and expressing them implicitly, i.e. set equal to zero. Assuming that the black box computer codes “know” the correct functional forms for f1 and f2, even if we don’t, then equations (5) and (6) are mathematically equivalent to equations (1) and (2), albeit simply expressed in a different form. There is a unique solution to equations (5) and (6) and it is the same solution as for the equations of the holistic model. Newton’s method is simply a different way to arrive at this solution than the simple linear solver used in the holistic model.

The essential difference in this exercise and a classical, multi-variable, root-finding problem is that the complete functional forms of equations (5) and (6) are not known. However, we can certainly evaluate S1 and S2 given “guesses” for the unknown input values using the computer codes, so lack of knowledge of the exact functional forms and the lack of “access” to them is not a particular issue, with one exception. Newton’s method requires the evaluation of the Jacobian matrix (matrix of partial derivatives of g1 and g2 with respect to S1 and S2). With knowledge of the functional forms of g1 and g2, we could calculate analytical derivatives and use those. Without knowledge, we must use *estimates* of the Jacobian elements based on finite difference approximations to those analytical derivatives. (Newton’s method with Jacobian derivatives approximated by finite differences is sometimes called the Broyden method[[1]](#endnote-1).)

The general Newton procedure is:

Step 1) Known data are W1, W2, a, and b. Assume a convergence criterion, ε.

Step 2) Start with a guess for S1 and S2, S1\* and S2\*.

Step 3) Run model 1, get

S1 = f1(W1,S2\*)

Step 4) Run model 2, get

S2 = f2(W2,S1\*)

Step 5) Evaluate the convergence criterion:

Are both

S1 - f1(W1,S2\*) < ε?

S2 - f2(W2,S1\*) < ε?

Step 6) If yes, stop. Solution obtained. If no, calculate the “Newton step” using the estimated Jacobian (details are not shown here), update the guesses for S1\* and S2\* and go to Step 3. Repeat until convergence is obtained.

**1.4 Comments**

* For dynamic problems, the above procedure is repeated at each time step.
* The separate submodels can be linear or nonlinear, explicit or implicit.
* This iterative procedure will add computational effort relative to a holistic model that can solve the separate models internally and simultaneously.
* We are using the iterative Newton’s method here, not necessarily because we are dealing with nonlinear problems, but rather because the functional forms of the submodels are “unknown”, or at least unavailable, to us.

1. **Simple Dynamic Example using Feedback Method**

A numerical example is presented here for a coupled, dynamic, water quality system to provide a proof-of-concept for the iterative procedure. From this point forward, we refer to the coupled, iterative, Newton’s method procedure as the “feedback” method, because we are relying on feedback from each external model to solve the larger holistic problem. This example is simple, yet fully captures the problem posed by coupling more complex, independent computer programs. The 2-system interaction shown in Figure 1 for this example represents the kinetic interaction between two coupled water quality variables in a completely mixed lake. “Subsystem 1” describes state variable 1 (chemical 1) and “Subsystem 2” describes state variable (chemical) 2. Both chemicals 1 and 2 have external inputs. Chemical 1 partially degrades to chemical 2 via 1st order kinetics. Simultaneously, chemical 2 partially degrades to chemical 1 via 1st order kinetics. Thus, the concentration of chemical 1 over time is determined by its mass input rate, the volume and flow rate of the lake, the mass loss to chemical 2, and the mass gain from chemical 2. The concentration of chemical 2 is analogously determined by its inputs, flow/volume, and losses/gains to/from chemical 1.

The mass balance, differential equations for chemicals 1 and 2 are

 (7)

 (8)

Where V and Q are the volume and volumetric flow rate of the lake; S1 and S2 are the concentrations of chemicals 1 and 2, respectively; W1 and W2 are the external loadings of chemicals 1 and 2, respectively; and k1 and k2 are 1st order rate coefficients. (“a and b” in Figure 1 are here equivalent to k1V and k2V.) Thus, the change in chemical mass over time (the left-hand side) equals the external mass loading minus the 1st order loss to the other chemical plus the 1st order gain from the other chemical minus the mass advected out of the lake in the outflow. The units on each term in the equations are mass/time, e.g. grams/day.

Most environmental models involving differential equations (e.g., ModFlow) use approximate, numerical methods to solve those equations. Accordingly, we will use an (implicit) numerical technique (specifically the “back time” finite difference method) to solve equations (7) and (8). To do so, we discretize the derivative in time so that equations (7) and (8) can be rewritten in their approximate form as

 (9)

 (10)

Where the indices “t” and “t+1” denote two consecutive instants in time separated by the finite difference time step, Δt. The algorithm that solves equations (9) and (10) begins with initial conditions (t = 0) for each chemical and advances the solution to the next time step, t = 1. The known concentrations at t = 1 are then advanced to t = 2, and so on until the desired total simulation period is covered.

Let’s assign the following values to the system parameters:

V = 100 m3

Q = 10 m3/day

W1 = W2 = 10 g/day

k1 = k2 = 1/day

Δt = 1 day

S10 = S20 = 0 g/m3 (initial conditions)

Because we are using constant parameters, and have assigned equal values to chemical 1 and 2’s loadings, initial conditions, and 1st order kinetic parameters, the steady-state solution to the coupled system of equations is intuitive. In other words, because the loss of chemical from chemical 1 to chemical 2 is matched by the gain from chemical 2, the steady-state solution for each chemical will simply be the continuous loading divided by the flow rate, or



We have made these simplifying assumptions so that the following numerical solutions can be validated.

Before implementing the feedback approach using Newton’s method to solve this simultaneous system, we first used the Generic Environmental Model[[2]](#endnote-2) (GEM) to solve the *complete* system (equations 9 and 10) in time. This, of course, is the *idealized* situation that would result if the modeler had full knowledge of the equations comprising systems 1 and 2, and had the time and resources to implement the fully interacting solution, i.e. solving equations 9 and 10 simultaneously within the same computer code. (This is what we are trying to avoid and is the whole purpose of this paper.) Nonetheless, we do so here to demonstrate the validity of the solution technique to reproduce the known steady-state solution (1 g/m3) and to serve as a validation benchmark for the feedback approach which follows. We call this fully-integrated model the “*holistic*” model.

Figure 2 is a time series plot for S1 and S2 using the above parameters for 60 time steps using a one-day time step.

**Figure 2. Dynamic Solution for Holistic Model with Equal Kinetics**

What we see is that, indeed, the holistic, numerical model runs the solution out to our known steady-state solution of 1.0 g/m3. The time to steady-state is approximately 50 days. Both S1 and S2 are shown on the plot, but are indistinguishable from each other because they are identical due to their identical loadings and kinetics as noted above.

Now consider a slightly more interesting dynamic solution with unequal kinetics between the two chemicals, and a non-zero initial condition. We now assume k2 = 0.5/day, i.e. one-half the value of k1, and that S1 and S2 have an initial concentration of 5 g/m3. (We don’t have an “intuitive” solution for the steady-state concentrations under this scenario.) Running the GEM for this scenario results in the time series plots shown in Figure 3. S1 immediately begins to decrease from its initial condition and reaches a steady-state concentration that is somewhat less than S1, because S2 is losing mass to S1 faster than it is gaining mass from S1. In contrast, S2 exhibits a slight increase in concentration from its initial condition for a few days and then decreases in a 1st order manner until it reaches a steady-state greater than S1. Because chemical 1 is losing mass to chemical 2 at twice the rate it is gaining mass from chemical 2, we would expect to see chemical 2 have higher concentrations for all time periods and, indeed, that is what we see.

**Figure 3. Dynamic Solution for Holistic Model with Unequal Kinetics**

With that background, let’s implement the feedback approach that is our goal. Recapping the earlier discussion, the feedback solution assumes that the holistic solution (software capable of simulating the entire coupled system simultaneously) is not available. What is available are two independent models. The first simulates subsystem 1 (chemical 1) given its external input, W1, and the input from subsystem 2, i.e. the “loading” k2VS2 for our simple example. The second model simulates subsystem 2 (chemical 2) given its external input, W2, and the input from subsystem 1, k1VS1.

Two Visual Basic programs, S1.vbp and S2.vbp, were written that simulate subsystems 1 and 2, respectively, given these inputs. S1.vbp is independent of S2.vbp and vice-versa. Their only “communication” is through each model’s input file. However, at any time step t, note that the user of S1.vbp would not know the value of S2t, and thus would not know the value of the loading term k2VS2t. The analogous is true for the user of S2.vbp with respect to the S1t and the loading term k1VS1t. S1.vbp solves equation (9) given V,Q,k1,k2,W1 and some “guess” for S2t+1. S2.vbp solves equation (10) given V,Q,k1,k2,W2 and some guess for S1t+1. We *do* assume that the user knows the values of the two kinetic coefficients, k1 and k2. That assumption is not unreasonable for persons wishing to couple different environmental software models. We then compiled S1.vbp and S2.vbp into executable-only codes, S1.exe and S2.exe, to simulate typical applications where the user attempting coupling may not have access to the source code.

From the perspective of a modeler who does not have access to the mathematical equations comprising model 1, the equation representing model 1 is

 (12)

where f1 is simply a generic representation of S1.exe for a single time step. Similarly, the system 2 equation that the modeler is aware of is

 (13)

Where f2 represents S2.exe and its associated set of inputs for a single time step. Equations (12) and (13) are perfectly legitimate coupled mathematical relationships. At any time step, these are two coupled equations in two unknowns, S1t+1 and S2t+1. Under the feedback technique, the Newton algorithm iterates between the two equations to find the roots of the two implicit equations for each time step, starting with guesses for the two unknown concentrations. The implicit equations being solved are

 (14)

 (15)

The solution to these equations (the roots) are fully equivalent to finding the solution to the equations (whatever they might be) that comprise the (assumed unavailable) holistic model.

The Jacobian matrix, required by Newton’s method, is the matrix of partial derivatives at time step t (where the unknowns are at t+1):

 (16)

In general, our user of S1.exe and S2.exe will not be able to calculate the analytical derivatives to form the Jacobian matrix because he or she does not know the underlying functional forms of the equations. However, as previously mentioned, the elements of the Jacobian can be estimated numerically using finite difference approximations.

The GEM software was used in this case as the high-level, “manager” program that implements the feedback method and couples S1.exe and S2.exe by firing them off iteratively (by “shelling out”) within each Newton iteration. (In this application[[3]](#endnote-3), the GEM knew nothing about the internal structure of S1 and S2. It simply executed Newton’s method using the information provided by S1.exe and S2.exe.) The Jacobian matrix was estimated numerically at each iteration by running S1.exe and S2.exe twice for each element of the Jacobian (4 elements), once as a benchmark and again after perturbing the appropriate parameter (S1 or S2) slightly. The initial “guess”, to get Newton’s method started for the first time step, was 5.0, the initial condition for both chemicals. A stopping criterion of 10-15 was used to satisfy convergence at each time step. A time step of 1 day was used and 60 time steps were made.

Results of the feedback method (“S1\_Feedback” and “S2\_Feedback”) are shown in Figure 4, along with the holistic solution for comparison. Although there are slight differences between the holistic and feedback results, they are very comparable demonstrating the proof-of-concept of the method.

**Figure 4. Results of Feedback Method for Example Dynamic Problem**

We noted during the solution that Newton’s method converged after a single iteration at each time step. This is because the underlying problem is linear (even if the user and/or the GEM does not know that) and Newton’s method should converge to the roots in a single iteration for linear problems. Thus, based on this example, the increased computational burden posed by the feedback procedure appears to result only from the overhead of having to “shell” out to fire the external models, S1.exe and S2.exe, and to write their input files and read their output files at each time step. It would seem that we are not having to pay a computational penalty for the feedback procedure itself.

We are not sure why there are slight differences in the feedback versus holistic solutions in Figure 4. Theoretically, the two methods should result in identical results because, as argued earlier, we are solving the same problem albeit using two different methods. To investigate this issue, we used the GEM to run the holistic problem again, but now using the GEM’s nonlinear (Newton’s method) option to solve that holistic problem, even though the problem is linear. This time the GEM has full knowledge of the functional forms comprising the overall system and is *not* using S1.exe and S2.exe. Using the same stopping criterion (10-15) and initial guess values (5.0), the Newton method again converged after a single iteration at each time step and the results were identical to those from the feedback method. Thus, the slight differences in Figure 4 apparently result from using two different mathematical techniques (the linear solver versus Newton’s method) and not from the feedback procedure itself. This issue will be investigated further.

1. Broyden, C.G. (October 1965). “A Class of Methods for Solving Nonlinear Simultaneous Equations”. Mathematics of Computation (American Mathematical Society) 19 (92): 577 – 593. [↑](#endnote-ref-1)
2. K.W. Little, “Environmental Fate and Transport Analysis with Compartment Modeling”, CRC Press, June 2012. [↑](#endnote-ref-2)
3. This used the GEM’s “Equation Solver” mode. [↑](#endnote-ref-3)