DIAGNOSTIC 5 SOLUTION

MODELING AND SIMULATION

Figure 1: Cartoon from climateprogress.org.

reaction_rate

`reaction_rate` takes an index (between 0 and 12) and a vector of concentrations, and returns the reaction rate of the given reaction.

```
function res = reaction_rate(i, X)
    % compute the rate of the (i)th reaction,
    % given the vector of concentrations (X)

    % kcats are in (1/seconds)
    kcat = [14.48 0.16 0.36 -Inf 0.16 0.36 -Inf 0.85 0.13 0.27 1.07 1.01];

    % Kms are in (Mol/L)
    Km = [37.17 0.46 0.09 -Inf 0.46 0.09 -Inf 0.25 1210 360 1300 1300];

    % which enzyme and substrate pertain to each reaction?
    enzyme = [1 1 1 1 1 1 2 2 2 3 3];
    substrate = [2 2 2 2 3 3 3 3 4 5 5 4];

    % enzyme concentrations in Mol/L
    E = [9.8e-5 1e-4 3.02e-2];

    if i == 0
        Vmax = 0.005;
        Km = 250;
        rate = Vmax * X(1) / (Km + X(1));
    else
        j = enzyme(i);
        k = substrate(i);
        rate = kcat(i) * E(j) * X(k) / (Km(i) + X(k));
    end
    res = rate;
end
```

The vectors `enzyme` and `substrate` indicate which enzyme and substrate are involved in each reaction. This way of using vectors makes it possible to avoid lots of `if` statements.

rate_func

rate_func has the usual interface for functions that work with ode45. It takes time and a state vector as inputs and returns the derivative of the state vector.

The state vector contains the concentrations of the six substrates, in this order: $S_{unavailable}$, $S_{available}$, D, M, MT, G.

```
function res = rate_func(t, X)
    % compute the rate of change for each of the six substrates,
    % given the vector of concentrations (X)
    r0 = reaction_rate(0, X);
    for i=1:12
        r(i) = reaction_rate(i, X);
    end

    % since rate constants are not available for two of the
    % reactions, we have to fudge
    r(4) = 0.05 * r(3);
    r(7) = 0.05 * r(6);

    % the following rates are in mMol/L / minute (or second?)
    Sun = -r0;
    Savail = r0 - r(1) - r(2) - r(3) - r(4);
    D = r(1) - r(5) - r(6) - r(7) - r(8);
    M = r(3) + r(6) + r(10) + r(11) - r(9) - r(12);
    MT = r(2) + r(5) - r(10) - r(11);
    G = r(4) + r(7) + r(8) + r(9) + r(10) + r(11) + r(12);

    % pack the results into a column vector
    res = [Sun Savail D M MT G]';
end
```

Notice that the return value is a *column* vector, which makes ode45 happy.

corn

corn uses the "Events" option of ode45 to stop when the concentration of starch reaches a given final value. You can read about options in Section 10.1 of the cat book.

```
function res = corn()
    % run a simulation of the enzyme action in fuel alcohol production
    % and return the time (in hours) to reduce the unavailable starch
    % concentation to 0.01 (mMol/L)
    options = odeset('Events', @event_func);

    tend = 50 * 60 * 60;                    % 50 hours in seconds
    Sinit = 0.1512;                         % mMol/L
    Sinit = Sinit / 1000;                   % now Mol/L
    [T, M] = ode45(@rate_func, [0,tend], [Sinit 0 0 0 0 0], options);
    plot(T, M)

    % return the end time in hours
    res = T(end) / 60 / 60;
end

function [value, isterminal, direction] = event_func(t, X)
    % check whether the concentration of unavailable starch has
    % reached its final value (in Mol/L)
    value = X(1) - 0.01e-3;
    isterminal = 1;
    direction = -1;
end
```

A more general version of corn would take the final concentration as an input variable, which could be passed to event_func as an input variable, but the mechanism is a little awkward. The easiest alternative is to next event_func inside corn so it has direct access to the variables defined in corn.