```
function [true_state, BLS_info] = BLS_spring( x, Y, W, x_bar, W_bar )
%BLS_spring Batch least squares solution for spring problem in StatOD book.
%   Use Batch Least Squares algorithm to determine initial state when given
%   some measurements.  There are TODOs where I'd like to genericize this
%   function.
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

num_dim = length(x);
% x0 = x_bar;
x0 = [4.0; 0.2];
for ii = 1:4
    % Setup for this iteration
    STM = eye(num_dim);
    lam = W_bar;
    N=W_bar*x_bar;
    state = [x0; reshape(STM,num_dim*num_dim,1)];
    RMS_accum = [0;0];
    y_hist = zeros(2,1,11);
    H_hist = zeros(2,2,11);

    for measurement = 1:length(Y)
```

Integrate

```
        if measurement ~= 1
            % default to a tenth of a timestep...
            dt = 0.1; % s
            times = 0:dt:1; %Assumes 1s measurements...
            ode_opts = odeset('RelTol', 1e-13, 'AbsTol', 1e-20);
            [T,Xout] = ode45(@hw5_spring_deriv, times, state, ode_opts);
            STM = reshape(Xout(end,num_dim+1:end),num_dim, num_dim);
        end
```

Observations

Function handles would be good here, making it useful for any dynamics for any number of dimensions... G = [compute_range(state); compute_range_rate(state)]; H_tilda = [dRange(state); dRangeRate(state)]; TODO: Handle this a little better I think...

```
        if measurement ~= 1
            x = Xout(end,1);
            v = Xout(end,2);
        else
            x = x0(1);
            v = x0(2);
        end
        % TODO: Not in ideal algorithm...
        h = 5.4; % m
        rho = sqrt(x*x+h*h);
        rho_dot = x*v/rho;
```

```
            G = [rho; rho_dot];
            H_tilda = [x/rho 0; (v/rho - x*x*v/(rho*rho*rho)) x/rho];
            y = Y(:,measurement) - G;
            H = H_tilda*STM;

            % Accumulate
            lam = lam + H'*W*H;
            N = N + H'*W*y;
            if measurement ~= 1
                state = [Xout(end,1:2)'; reshape(STM,num_dim*num_dim,1)];
            end

            % Accumulate residuals
            y_hist(:,:,measurement) = y;
            H_hist(:,:,measurement) = H;

        end
        x_hat = lam\N;

        % Determine RMS
        for measurement = 1:length(Y)
            epsilon = y_hist(:,:,measurement) - H_hist(:,:,measurement)*x_hat;
            %This is per measurement type, not overall RMS error...
            RMS_accum = RMS_accum + epsilon.*epsilon;
        end
        RMS = sqrt(RMS_accum/11);

        % Test for convergence
        if ii ~= 4 %Later a convergence thing can go in here.
            x_bar = x_bar-x_hat;
            x0 = x0 + x_hat;
        end
    end

    true_state = x0 + x_hat;
    BLS_info.RMS = RMS;
    BLS_info.P0 = inv(lam);
    BLS_info.xhat = x_hat;
    end
```

Published with MATLAB® R2013b