```
clear;clc;clf;

tic();
%       ** Biomolecular degradation and renewal simulations **
%
%   - Calculates and displays steady state distributions of P and D for biomolecular ensembles performing work and undergoing renewal.
%   - Configurable to sweep through a range of renewal rates to calculate P and energetic ROI for each renewal rate.
%   - Configurable to loop through a range of 'm' and 'P_ren_max' values.
%   - Results can be saved as Matlab data output or Excel file.
%   - Movie of P or D frequency distributions
%
%   Consult manuscript entitled "A Multidisciplinary Approach to Explain Biological Aging and Longevity" for derivations and
%   detailed description of formulae.
%
%   Author: Brett Augsburger,PhD
%   Date: 7/14/2017
%   Programmed in MATLAB R2017A

%   Functionality to consider in future:
%   - add phiD
%   - add ability to use curve-fit performance/time data instead of two points

% Biomolecular degradation and performance parameters (time units arbitrary)
tdeg50 = 105;       % Time to degrade to 50% performance while producing max work
% Specify *either* tdeg75, tdeg25 or k (ONLY ONE). Set unused parameters to NaN.
tdeg75 = 38;       % Time to degrade to 75% performance
tdeg25 = NaN;       % Time to degrade to 25% performance
k = NaN;
Dp50 = 0.02;    % Needs to be low enough that P is very near 0 at D=1
w_dc = 1.0;     % Percentage of time average biomolecule spends performing work

% Logistic function fitting settings
fitType = 4;    % 3: 3-parameter fit, 4: 4-parameter fit
delta = 350.2;   % Required for 4-parameter fit (not used for 3-parameter)

nResult_threshold = 1e-5;%2e-1;%2.25e-1;%1e-9;   % Fit error
Cinit = 2;      % Initial C for 4-parameter fit
kInit = 15;     % Initial k for 3-parameter fit
calc_f_d = false;   % Set to 'false' if there are problems with symbolic solve of f_d (4-param fit only), otherwise set to 'true' (faster)

% Biomolecular renewal parameters
m = 0.30;       % Worst m*100% of population randomly chosen from on renewal event
    % If m_array is non-empty will sweep through specified m values separately and save files for each
m_array = [];   %[0.01 0.05 0.10 0.15 0.20 0.25 0.30 0.40 0.50 0.60 0.70 0.80 0.90 0.99];
P_ren_max = 1;  % Recognition limit (i.e. maximum performance) of molecule for renewal, set to 1 for no limit
    % If P_ren_max_array is non-empty will sweep through specified P_ren_max values separately and save files for each
P_ren_max_array = [];   %[0.10 0.15 0.20 0.25 0.30 0.40 0.50 0.60 0.70 0.80 0.90 1];
% Specify either Dstatic_mult or tdeg50_static (but not both) and set Dstatic_type accordingly
Dstatic_type = 1;   % 0: Static deg. as % of max dynamic deg, 1: Static deg. by tdeg50_static
Dstatic_mult = 0;   % Maximum static degradation rate (as % of peak D at P=1)
tdeg50_static = inf;   % Only for Dstatic_type = 1. Time to degrade to 50%P under non-working conditions.

Ntot = 1e9;         % Total number of molecules of interest in system
Ndotrc_min = 2.2e5;   % Min renewal rate in molecules per unit time. Also spec's value to use for single analysis.
Ndotrc_max = 3e8;   % Maximum renewal rate
num_Ndotrc_datapoints = 100; % Renewal rate sweep- if set to 1, will display different plots than multirate analysis

% Simulation parameters
ts_init = 1e-0;     % Initial (i.e. maximum) timestep
ts_min = 1e-5;      % Minimum timestep to use
t_stop = 60e5;      % Max simulation time (not CPU time, will stop before this if convergence occurs)
convergence_threshold_1 = 0.03;   % Convergence threshold based on Eq. 14
convergence_threshold_2 = 1e-5;   % Convergence threshold based on change in distribution with time

Ds_min_step = 1e-5; % Minimum degradation step size to use
Ds_min_num = 2e2;   % Minimum total number of degradation steps
useOptimizedDspacing = 0;   % Use optimized D spacing? (Y/N; 1/0)
Ds_min_scalar = 4;  % Max difference between largest and smallest Ds (optimized D spacing mode only)
Ds_dec_mult = 2;    % Multiplier for decreasing Ds if unable to converge on solution
ts_dec_mult = 4;    % Multiplier for decreasing time step (must be integer)

% Output settings (not used with m or P_ren_max sweeps)
captureMovie = 1;   % Capture movie of P or D frequency distributions? (Y/N; 1/0)
movieType = 1;      % 1 - P versus N freq, 2 - D versus N freq
saveMovie = 1;      % Save captured movie to file? (Y/N; 1/0)
write_to_xls_file = 0;  % Output data to excel file? (Y/N; 1/0)
save_matlab_variables_to_file = 1;  % Output final variables to Matlab data file? (Y/N; 1/0)

% End configuration settings

if (~isempty(m_array) && ~isempty(P_ren_max_array))
    error('m_array and P_ren_max_array are both non-empty. Specify only one of the two (other should be empty).');
end

% Determine P-D curve
syms D_off_sym C_sym Dp2nd_sym k_sym D_sym P_sym Dscale_sym Ddot_sym Ds_sym Dscale_sym
if ~isnan(tdeg75) && isnan(tdeg25) && isnan(k)
    P_at_2nd_point = 0.75;
    tdegX = tdeg75;
```

```
elseif ~isnan(tdeg25) && isnan(tdeg75) && isnan(k)
    P_at_2nd_point = 0.25;
    tdegX = tdeg25;
elseif ~isnan(k) && isnan(tdeg75) && isnan(tdeg25)
    P_at_2nd_point = 0.75; % Arbitrary for this case
else
    error('Specify either k, tdeg75, or tdeg25 but only one of these');
end

% Fit to either 4-parameter or 3-parameter logistic function
if (fitType == 4)   % 4-parameter logistic function
    % NOTE: We are going to vary 'C' to find best fit instead of 'k' like we do with 3-parameter logistic
    % Driving function: P = C*(1+(k*(D-D_off)^delta))^-1
    eqI = D_off_sym-Dp50+k_sym^-1*(2*C_sym-1)^(1/delta);
    eqII=(C_sym-1)^(1/delta)+k_sym*D_off_sym;        % Pzero_offset - should be 0
    eqIII = Dp2nd_sym-D_off_sym-k_sym^-1*(C_sym*(P_at_2nd_point^-1)-1)^(1/delta);  % 2nd data point
    soleq = solve(eqI,eqII,eqIII,[D_off_sym,k_sym,Dp2nd_sym]);
    f_D_off = matlabFunction(soleq.D_off_sym);
    f_k = matlabFunction(soleq.k_sym);
    f_Dp2nd = matlabFunction(soleq.Dp2nd_sym);  % D at either P75 or P25 (if specified)
    f_Pzero_offset = matlabFunction(eqII, 'vars', {C_sym,D_off_sym,k_sym});

    % Dstatic is L and Dscale is K*w_dc from Eqs. 10 & 11 of manuscript
    if (Dstatic_type == 0)
        % Dstatic specified as % of peak D at max P
        Ddot_static = 0;    % This parameter will not be used so set to zero
        eqXI = int((Dscale_sym*(Dstatic_mult+(1-Dstatic_mult)*C_sym*(1+(k_sym*(D_sym-D_off_sym))^delta)^-1))^-1,D_sym);
        eqXII = subs(eqXI,D_sym,0); % Solve for D = 0 so we can calculate definite integral
        f_tdeg = matlabFunction(simplify(eqXI-eqXII),'vars', {D_sym Dscale_sym C_sym k_sym D_off_sym});  % Time to degrade from D=0 to D=D_sym

        solveforDdot = solve(Ddot_sym-Dscale_sym*(Dstatic_mult+(1-Dstatic_mult)*P_sym)/Ds_sym,Ddot_sym);
        f_Ddot = matlabFunction(solveforDdot,'vars',{Dscale_sym,Ds_sym,P_sym});
    else
        % Dstatic determined by tdeg50_static
        Ddot_static = Dp50/tdeg50_static;   % TO DO: Changes needed here if we want to implement phiD
        eqXXI = int((Ddot_static+Dscale_sym*C_sym*(1+(k_sym*(D_sym-D_off_sym))^delta)^-1)^-1,D_sym);
        eqXXII = subs(eqXXI,D_sym,0); % Solve for D = 0 so we can calculate definite integral
        f_tdeg = matlabFunction(simplify(eqXXI-eqXXII),'vars', {D_sym Dscale_sym C_sym k_sym D_off_sym});  % Time to degrade from D=0 to D=D_sym
        eqXXIII = simplify(eqXXI-eqXXII)-tdeg50;    % Use this to solve for Dscale

        solveforDdot = solve(Ddot_sym-(Ddot_static+Dscale_sym*P_sym)/Ds_sym,Ddot_sym);
        f_Ddot = matlabFunction(solveforDdot,'vars',{Dscale_sym,Ds_sym,P_sym});
    end

    eqXIII = P_sym-C_sym*(1+(k_sym*(D_sym-D_off_sym))^delta)^-1;   %P_sym == C_sym*(1+(k_sym*(D-D_off_sym)^delta))^-1
    solveforP = solve(eqXIII,P_sym);
    f_P = matlabFunction(solveforP, 'vars',{D_sym,C_sym,D_off_sym,k_sym});   % Find P given D,C,D_off, and k
    if (calc_f_d ~= false)  % We can choose to calculate this another way if this produces problems
        assume(C_sym ~= 0 & P_sym ~= 0 & k_sym ~= 0);
        solveforD = solve(eqXIII,D_sym);
    %     solveforDnum = vpa(solveforD);
        solveforDnum = vpa(solveforD(1));
        f_D = matlabFunction(solveforDnum(1), 'vars',{P_sym,C_sym,D_off_sym,k_sym});   % Find D given P,C,D_off, and k
    end
    assume([C_sym P_sym k_sym],'clear');

    Dscale = 1; % Set to 1 for now. Will calculate correct value after finding D_off, C and k.

    % Fit P/D curve. Find C if given tdeg75 or tdeg25
    fprintf('Calculating best fit...\n');
    if ~isnan(k) && isnan(tdeg75) && isnan(tdeg25)  % k specified
        error('tdeg75 or tdeg25 must be specified for 4-parameter logistic P(D) function');
    elseif (~isnan(tdeg75) || ~isnan(tdeg25)) && isnan(k)    % tdeg75 specified - Find best k, C and D_off
        C = Cinit;   % initial guess, C should never be less than 1 with 4-parameter logistic function
        Cinc = 0.1;  % initial increment
        D_off = f_D_off(C);
        Dp2nd = f_Dp2nd(C);
        k = f_k(C);

        if (Dstatic_type ~= 0)
            eqXIV = subs(eqXXIII,[D_sym k_sym C_sym D_off_sym],[Dp50 k C D_off]);
            Dscale = double(solve(eqXIV,Dscale_sym));
        end

        tdeg50_calc_unsc = f_tdeg(Dp50,Dscale,C,k,D_off);
        tdegX_calc_unsc = f_tdeg(Dp2nd,Dscale,C,k,D_off);
        nResult = tdeg50_calc_unsc/tdeg50 - tdegX_calc_unsc/tdegX;

        lastGuess = NaN;
        guessCount = 0;
        corDir = 1; % Used to flip directions of C increments
        nResult_last = nResult;
        while ((abs(nResult) > nResult_threshold) && ~isnan(nResult))
            if (((nResult_last > 0)  && (nResult_last < nResult)) || ((nResult_last < 0)  && (nResult_last > nResult)))
                corDir = corDir*(-1);
            end
            nResult_last = nResult;
            if (nResult*corDir < 0)
                if (lastGuess == 0); Cinc = Cinc/2; guessCount = 0; end
                if (guessCount >= 10); Cinc = Cinc*2; guessCount = 0; end
                C = C+Cinc;
```

```
                lastGuess = 1;
                guessCount = guessCount + 1;
            elseif (nResult*corDir > 0)
                if (lastGuess == 1); Cinc = Cinc/2; guessCount = 0; end
                if (guessCount >= 10); Cinc = Cinc*2; guessCount = 0; end
                C = C-Cinc;
                lastGuess = 0;
                guessCount = guessCount + 1;
            end
            if (C < 1); error('Problem fitting curve: C < 1. Try adjusting simulation parameters.'); end
            D_off = f_D_off(C);
            Dp2nd = f_Dp2nd(C);
            k = f_k(C);
            if (Dstatic_type ~= 0)
                eqXIV = subs(eqXXIII,[D_sym k_sym C_sym D_off_sym],[Dp50 k C D_off]);
                Dscale = double(solve(eqXIV,Dscale_sym));
            end
            tdeg50_calc_unsc = f_tdeg(Dp50,Dscale,C,k,D_off);
            tdegX_calc_unsc = f_tdeg(Dp2nd,Dscale,C,k,D_off);
            nResult = tdeg50_calc_unsc/tdeg50 - tdegX_calc_unsc/tdegX;
        end
        Pzero_offset=f_Pzero_offset(C,D_off,k);  % Should be zero since we are calculating exact solution
        if (~isreal(Pzero_offset) || ~isreal(C) || isnan(nResult)); error('Values for C and D_off cannot be determined'); end
    % Check Pzero_offset for sanity
    if (abs(Pzero_offset) > 1e-3); error('|Pzero_offset| exceeds 1e-3'); end
    clear CInc corDir Dp50_off_arr nResult_last Pzero_offset_arr lastGuess a b
    end
elseif fitType==3   % 3-parameter logistic function
    eqI = D_off_sym-Dp50+k_sym^-1*log(2*C_sym-1);
    eqII = 1-C_sym*(1+exp(-k_sym*D_off_sym)).^-1;    % Pzero_offset - should be 0
    eqIII = Dp2nd_sym-D_off_sym-k_sym^-1*log(C_sym/P_at_2nd_point-1);
    soleq = solve(eqI,eqII,eqIII,[D_off_sym C_sym Dp2nd_sym],'ReturnConditions',true);
    f_D_off = matlabFunction(soleq.D_off_sym);
    f_C = matlabFunction(soleq.C_sym);
    f_Dp2nd = matlabFunction(soleq.Dp2nd_sym);  % D at either P75 or P25 (if specified)
    f_Pzero_offset = matlabFunction(eqII, 'vars', {C_sym,D_off_sym,k_sym});

    % Dstatic is L and Dscale is K*w_dc from Eqs. 10 & 11 of manuscript
    if (Dstatic_type == 0)
        % Dstatic specified as % of peak D at max P
        Ddot_static = 0;    % This parameter will not be used so set to zero
        eqXI = int((Dscale_sym*(Dstatic_mult+(1-Dstatic_mult)*C_sym*(1+exp(k_sym*(D_sym-D_off_sym)))^-1))^-1,D_sym);
        eqXII = subs(eqXI,D_sym,0); % Solve for D = 0 so we can calculate definite integral
        f_tdeg = matlabFunction(simplify(eqXI-eqXII),'vars', {D_sym Dscale_sym C_sym k_sym D_off_sym});  % Time to degrade from D=0 to D=D_sym

        solveforDdot = solve(Ddot_sym-Dscale_sym*(Dstatic_mult+(1-Dstatic_mult)*P_sym)/Ds_sym,Ddot_sym);
        f_Ddot = matlabFunction(solveforDdot,'vars',{Dscale_sym,Ds_sym,P_sym});
    else
        % Dstatic determined by tdeg50_static
        Ddot_static = Dp50/tdeg50_static;   % TO DO: Changes needed here if we want to implement phiD
        eqXXI = int((Ddot_static+Dscale_sym*C_sym*(1+exp(k_sym*(D_sym-D_off_sym)))^-1)^-1,D_sym);
        eqXXII = subs(eqXXI,D_sym,0); % Solve for D = 0 so we can calculate definite integral
        f_tdeg = matlabFunction(simplify(eqXXI-eqXXII),'vars', {D_sym Dscale_sym C_sym k_sym D_off_sym});  % Time to degrade from D=0 to D=D_sym
        eqXXIII = simplify(eqXXI-eqXXII)-tdeg50;    % Use this to solve for Dscale

        solveforDdot = solve(Ddot_sym-(Ddot_static+Dscale_sym*P_sym)/Ds_sym,Ddot_sym);
        f_Ddot = matlabFunction(solveforDdot,'vars',{Dscale_sym,Ds_sym,P_sym});
    end
    eqXIII = P_sym-C_sym*(1+exp(k_sym*(D_sym-D_off_sym))).^-1;   %P = C*(1+exp(k*(D-D_off)))^-1
    solveforP = solve(eqXIII,P_sym);
    f_P = matlabFunction(solveforP, 'vars',{D_sym,C_sym,D_off_sym,k_sym});   % Find P given D,C,D_off, and k
    solveforD = solve(eqXIII,D_sym);
    f_D = matlabFunction(solveforD, 'vars',{P_sym,C_sym,D_off_sym,k_sym});   % Find D given P,C,D_off, and k

    Dscale = 1; % Set to 1 for now. Will calculate correct value after finding D_off, C and k.

    % Fit P/D curve. Find k if given tdeg75 or tdeg25
    fprintf('Calculating best fit...\n');
    if ~isnan(k) && isnan(tdeg75) && isnan(tdeg25)  % k specified - Find C and D_off
        D_off = f_D_off(k);
        C = f_C(k);
        Pzero_offset=f_Pzero_offset(C,D_off,k);  % Should be zero since we are calculating exact solution
        if (~isreal(Pzero_offset) || ~isreal(C)); error('Values for C and D_off cannot be calculated'); end
    elseif (~isnan(tdeg75) || ~isnan(tdeg25)) && isnan(k)    % tdeg75 specified - Find best k, C and D_off
        k = kInit;  % initial guess, set high to make sure we find largest root
        kInc = 1;   % initial increment
        D_off = f_D_off(k);
        Dp2nd = f_Dp2nd(k);
        C = f_C(k);

        if (Dstatic_type ~= 0)
            eqXIV = subs(eqXXIII,[D_sym k_sym C_sym D_off_sym],[Dp50 k C D_off]);
            Dscale = double(solve(eqXIV,Dscale_sym));
        end

        tdeg50_calc_unsc = f_tdeg(Dp50,Dscale,C,k,D_off);
        tdegX_calc_unsc = f_tdeg(Dp2nd,Dscale,C,k,D_off);
        nResult = tdeg50_calc_unsc/tdeg50 - tdegX_calc_unsc/tdegX;

        lastGuess = NaN;
        while (abs(nResult) > nResult_threshold)
```

```
            if (nResult > 0)
                if (lastGuess == 0); kInc = kInc/2; end
                if (P_at_2nd_point > 0.5)
                    k = k+kInc;
                else
                    k = k-kInc;
                end
                lastGuess = 1;
            elseif (nResult < 0)
                if (lastGuess == 1); kInc = kInc/2; end
                if (P_at_2nd_point > 0.5)
                    k = k-kInc;
                else
                    k = k+kInc;
                end
                lastGuess = 0;
            end
            D_off = f_D_off(k);
            Dp2nd = f_Dp2nd(k);
            C = f_C(k);
            if (Dstatic_type ~= 0)
                eqXIV = subs(eqXXIII,[D_sym k_sym C_sym D_off_sym],[Dp50 k C D_off]);
                Dscale = double(solve(eqXIV,Dscale_sym));
            end
            tdeg50_calc_unsc = f_tdeg(Dp50,Dscale,C,k,D_off);
            tdegX_calc_unsc = f_tdeg(Dp2nd,Dscale,C,k,D_off);
            nResult = tdeg50_calc_unsc/tdeg50 - tdegX_calc_unsc/tdegX;
        end
        Pzero_offset=f_Pzero_offset(C,D_off,k);  % Should be zero since we are calculating exact solution
        if (~isreal(Pzero_offset) || ~isreal(C)); error('Values for C and D_off cannot be determined'); end
    end
    % Check Pzero_offset for sanity
    if (abs(Pzero_offset) > 1e-3); error('|Pzero_offset| exceeds 1e-3'); end
    clear kInc Dp50_off_arr Pzero_offset_arr lastGuess a b
else
    error('Specify either 3- or 4-parameter fit');
end
fprintf('Datafit succeeded...\n');

if (Dstatic_type == 0)
    Dscale = f_tdeg(Dp50,Dscale,C,k,D_off)/(tdeg50/w_dc);
else
    eqXIV = subs(eqXXIII,[D_sym k_sym C_sym D_off_sym],[Dp50 k C D_off]);
    Dscale = double(solve(eqXIV,Dscale_sym))*w_dc;
end

clear D_off_sym C_sym Dp2nd_sym k_sym D_sym P_sym Dscale_sym Ddot_sym Ds_sym Dscale_sym...
    eqI eqII eqIII soleq eqXI eqXII eqXIII eqXXI eqXXII eqXXIII solveforP solveforD solveforDnum solveforDdot eqXIV

ts_init_c = ts_init;    % Start with default ts_init but correct if necessary
if(useOptimizedDspacing == 0)   % Use linear D spacing
    Ds_max_def = max(Ds_min_step,(Ddot_static+Dscale)*ts_init_c);   % Works with both Dstatic_type options
    % **** Now calculate default values for D and P vectors ****
    D_def = linspace(0,1,floor(1/Ds_max_def));
    if (length(D_def) < Ds_min_num)
        D_def = linspace(0,1,Ds_min_num);
    end
    Ds_def = diff(D_def);
    Ds_def(end+1)=Ds_def(end);

    ts_init_c = Ds_def(1)/(Ddot_static+Dscale); % Corrects ts for Ds spacing, works with both Dstatic_type options
else    % Optimized D vector spacing
    % Configure D vector so that a molecule shifts exactly one position right per time step
    % NOTE: Ignores Ds_min_num
    if(ts_init_c*(Ddot_static+Dscale)<Ds_min_step); ts_init_c = (Ddot_static+Dscale); end
    Ds_min_step_temp = max(ts_init_c*(Ddot_static+Dscale)/Ds_min_scalar,Ds_min_step);
    D_def=zeros(1,round(Ds_min_step_temp^-1)+1);
    i=1;
    while (D_def(i) < 1)
        P_d = f_P(D_def(i),C,D_off,k);
        Ddot_delta = f_Ddot(Dscale,ts_init_c^-1,P_d);
        if (Ddot_delta < Ds_min_step_temp)
            Ddot_delta = Ds_min_step_temp;
        end
        if ((D_def(i)+Ddot_delta)> 1); break; end
        D_def(i+1) = min(1,D_def(i)+Ddot_delta);
        i = i+1;
    end
    D_def=D_def(1:i); % Truncate unused vector positions
    clear i P_d Ddot_delta Ds_min_step_temp

    Ds_def=diff(D_def);
    Ds_def(end+1)=Ds_def(end);
end
P_def = f_P(D_def,C,D_off,k);
Ddot_def = f_Ddot(Dscale,Ds_def,P_def);     % Scaled for degradation step

% Check to see if any value in Ddot is greater than 1. If so, something went wrong.
if any(Ddot_def.*ts_init_c-1 > 1e-6); error('Ddot cannot be greater than 1. Decrease ts or increase Ds.'); end

norm1 = normpdf(D_def,Dp50/2,0.01); % Start with a normal/Gaussian distribution of biomolecules across D values (this isn't critical)
```

```
Nd_def = Ntot*norm1/(length(norm1)-1);
clear norm1
%Nd_def(1:length(D_def)) = Ntot/length(D_def);   % Init distribution flat works fine too but may require a few more compute cycles

% Uncomment the following two lines and set Ndotrc_min = 0 to test degradation calculations and distributions:
% Nd_def = zeros(1,length(D_def));    % TEST
% Nd_def(1:1) = Ntot/1;   % TEST

if (num_Ndotrc_datapoints == 1)
    Ndotrc_array(1) = Ndotrc_min;
else
    Ndotrc_array = logspace(log10(Ndotrc_min),log10(Ndotrc_max),num_Ndotrc_datapoints);
end

D_common = 0:0.001:1;
P_from_D_common = min(1,f_P(D_common,C,D_off,k));
P_common = D_common;
if ((fitType == 3) || (calc_f_d ~= false))
    D_from_P_common = min(1,f_D(P_common,C,D_off,k));
else % Calculate D's from P's by interpolating
    D_from_P_common = min(1,interp1(P_from_D_common,D_common,P_common,'spline'));
end

% Vector of degradation times
tdeg = f_tdeg(D_from_P_common,Dscale,C,k,D_off);

% Output for informational purposes
fprintf('\ntdeg75 = %.4f\ntdeg25 = %.4f\ntdeg10 = %.4f\n\n',tdeg(751),tdeg(251),tdeg(101));
%

% Setup loop count for m or P_ren_max sweeping
if (isempty(m_array) && isempty(P_ren_max_array))
    m_or_P_ren_max_loopcount_max = 1;
elseif (~isempty(m_array))
    m_or_P_ren_max_loopcount_max = length(m_array);
    dateStamp_mSweep = datestr(now,'mm-dd-yyyy__HH_MM_SS');
elseif (~isempty(P_ren_max_array))
    m_or_P_ren_max_loopcount_max = length(P_ren_max_array);
    dateStamp_mSweep = datestr(now,'mm-dd-yyyy__HH_MM_SS');
end

for m_or_P_ren_max_loopcount=1:m_or_P_ren_max_loopcount_max
    Ndotrc = Ndotrc_array(1);
    Ndotrc_loopcount = 1;
    converged = false;
    Ntotf_sweep = zeros(1,length(Ndotrc_array));
    converged_sweep = zeros(1,length(Ndotrc_array));
    connerr_sweep = zeros(length(Ndotrc_array),1000);
    results = zeros(length(Ndotrc_array),3);
    D_max_result = 0;

    results_2 = zeros(3,length(D_common),length(Ndotrc_array));

    if (~isempty(m_array))
        m = m_array(m_or_P_ren_max_loopcount);
        fprintf('\nPerforming simulation for m = %.3f\n\n',m);
    elseif (~isempty(P_ren_max_array))
        P_ren_max = P_ren_max_array(m_or_P_ren_max_loopcount);
        fprintf('\nPerforming simulation for P_ren_max = %.3f\n\n',P_ren_max);
    end % otherwise just uses specified single value for m and P_ren_max

    % Calculate D corresponding to P_ren_max. If P_ren_max sweeping, recalc every time. Otherwise only calculate once.
    if ((~isempty(m_array) && m_or_P_ren_max_loopcount == 1) || ~isempty(P_ren_max_array) ||...
        (isempty(m_array) && isempty(P_ren_max_array)))
        if ((fitType == 3) || (calc_f_d ~= false))
            D_at_P_ren_max = max(0,f_D(P_ren_max,C,D_off,k));
        else
            D_at_P_ren_max = max(0,interp1(P_from_D_common,D_common,P_ren_max,'spline'));
        end
    end

    % Ndotrc iteration
    for Ndotrc_loopcount=1:length(Ndotrc_array)  % Might speed up with parfor if parallel tb is available
        Ndotrc = Ndotrc_array(length(Ndotrc_array)+1-Ndotrc_loopcount);    % Do highest renewal rates first
        % Ndotrc = Ndotrc_array(Ndotrc_loopcount);

        Ds = Ds_def;
        Ddot = Ddot_def;
        P = P_def;

        if ((Ndotrc_loopcount > 1) && converged == true)   % Start with Nd from previous Ndotrc (not compatible with parfor)
           Dold=D;
           D = D_def;
           Nd = max(0,interp1(Dold,Nd,D,'spline'));
           Nd = Nd/(sum(Nd)/Ntot);
        else
           D = D_def;
           Nd = Nd_def;
        end
        x = length(Nd);
        ts = ts_init_c;
```

```
    Ndprev = zeros(length(Nd),1);
    concheckcount = 1;
    tloopcount = 3; % Start at 3 to avoid immediately checking convergence
    converged = false;

    connerr = zeros(6,1000);

    t = 0;
    Nd_old = Nd;
    degVals = zeros(1,length(Nd));
    concheckcount_last_ts_change = 0;

    % Time step loop
    while (t < t_stop)
        t = t + ts;

        % Renewal calcs
        % Find x (degradation state corresponding to start of worst m*100%). Interpolate to get best estimate.
        num = sum(Nd(floor(x):end)) - (x-floor(x))*Nd(floor(x));

        Ntot_t = sum(Nd);
        if (m == 1) % No need to calculate x for totally random renewal
            x = 1;
            D_at_x = 0;
        else
            if (num < m*Ntot_t) && ((num+m*sum(Nd(1:floor(x))))-m*Ntot_t >= -1e-2)
                x = floor(x);
                num = sum(Nd(round(x):end));  % Don't need remainder for now
                while ((num < m*Ntot_t) && (x > 1) && ((num+m*sum(Nd(1:round(x)-1)))-m*Ntot_t >= -1e-2))
                    x = x-1;
                    num = num+Nd(round(x));
                    if (x < 1); error('Unexpected error while calculating x'); end
                end
                if ((num+m*sum(Nd(1:round(x)-1)))-m*Ntot_t < -1e-2)
                    x = round(x+1);
                else    % Normally come here
                    % Now interpolate
                    x = min(round(x)+1,round(x)+(num-m*Ntot_t)/Nd(round(x)));
                end
            elseif (num > m*Ntot_t)
                x = ceil(x);
                num = sum(Nd(round(x):end));
                while ((num > m*Ntot_t) && (x < length(Nd)))
                    num = num-Nd(round(x));
                    x = x+1;
                    if (x > length(Nd)); error('Unexpected error while calculating x'); end
                end
                % Now interpolate
                x = x-1;
                num = num+Nd(round(x));
                x = round(x)+(num-m*Ntot_t)/Nd(round(x));
            end
            if ((x < 1) || (x >= length(Nd))); error('x outside limits'); end
%           D_at_x = interp1(D,x);
            D_at_x = D(round(x))+mod(x,1)*(D(ceil(x))-D(floor(x))); % faster than interp1
        end

        % Check to see if there are sufficient numbers of recognizable deterioriated molecules below P_ren_max
        if ((D_at_x < D_at_P_ren_max) && (m ~= 1))
            x = min(length(D),interp1(D,1:length(D),D_at_P_ren_max));
            D_at_x = interp1(D,x);
            x_limited = 1;  % P_ren_max limiting is active
        else
            x_limited = 0;
        end

        % Calculated average D of renewed molecules
        Dren_avg = max(0,(sum(D(ceil(x):end).*Nd(ceil(x):end))+D(floor(x))*Nd(floor(x))*(ceil(x)-x))/(sum(Nd(ceil(x):end))+Nd(floor(x))*(ceil(x)-x)));

        % Smooth renewal curve by converting to logistic function:
        % Function is centered at D(x) with 0.75 point at (Dren_avg-D(x))/2
        % r75/r25 must be at least r75_dist % * D(x) away from 0 and 1 (max/min D values)
        r75_dist = 0.95;  % in percent / 100
        r75 = min([(D_at_x+Dren_avg)/2 D_at_x*(2-r75_dist) D_at_x+(1-D_at_x)*(1-r75_dist)]);

        q = abs(-log(1/3)/(r75-D_at_x));
        renPr = 1./(1+exp(-q*(D-D_at_x)));

        Ndotrc_t = Ndotrc*(Ntot_t/Ntot);  % Correct renewal rate for drifts in total number of molecules

        % Calc vector of renewal rates (renRates units -> # mol per unit time)
        if(x_limited == 0)
            renRates = Nd.*renPr;   % Normal case without P_ren_max limiting
            renRates = renRates*Ndotrc_t/sum(renRates);    % Renewal rates by D in molecules per unit time
            renRates(1) = -Ndotrc_t;   % Renewed molecules go to D=0
            renRates(isnan(renRates)) = 0;
        else
            % Come here if P_ren_max has been exceeded
            numRen_below_P_ren_max = sum(Nd(floor(x):end)) - (x-floor(x))*Nd(floor(x));
%           numRen_below_P_ren_max = sum(Nd.*renPr);
```

```
            scalar = Ndotrc_t*numRen_below_P_ren_max/(m*Ntot_t)^2;
            j = ((Ndotrc_t/scalar)-sum(Nd.*renPr))/(sum(Nd)-sum(Nd.*renPr));
            if (j < 1)
                renRates = (j.*Nd+(1-j)*renPr.*Nd)*scalar;
%                 Dren_avg = sum(j.*Nd.*D)*scalar/Ndotrc_t+(numRen_below_P_ren_max/Ntot_t)*(1-j)*Dren_avg;
                Dren_avg = sum(renRates(2:end).*D(2:end))/Ndotrc_t;
            else
                renRates = Nd.*Ndotrc_t/sum(Nd);
                Dren_avg = sum(D.*Nd)/Ntot_t;
            end
            renRates(1) = renRates(1)-Ndotrc_t;   % Renewed molecules go to D=0
            renRates(isnan(renRates)) = 0;

            clear j scalar numRen_below_P_ren_max;
        end

        Ndlast = Nd;

        % Calculate vector for degradation
        degVals(2:length(Nd))=Nd(1:end-1).*Ddot(1:end-1);
        degVals(1) = 0;

        % Calculate distribution for next time point
        Nd = max(0,Nd+(degVals-Nd.*Ddot-renRates)*ts);

        % Check whether the solution has converged sufficiently
        if (mod(tloopcount,100) == 0)    % Check every 100 time steps
            Ndprev = Nd;
        elseif (mod(tloopcount,100) == 1)   % A few different methods of looking at convergence
            connerr(1,concheckcount) = (sqrt((sum((Ndprev-Nd).^2)))/sqrt(sum(Ndprev.^2)))/ts;
            connerr(5,concheckcount) = max(Nd)/mean(Ds);    % TO DO - check this
            connerr(8,concheckcount) = sum(Ndlast.*Ddot.*Ds)/(Dren_avg*Ndotrc_t);    % Convergence per Eq. 14 in manuscript
%             connerr(8,concheckcount) = sum(Ndlast.*Ddot.*Ds)/sum(renRates(2:end).*D(2:end));    % Convergence per Eq. 14 in manuscript
            % Check convergence using two methods (both must pass)
            if ((connerr(1,concheckcount) <= convergence_threshold_2) && (abs(connerr(8,concheckcount)-1) <= convergence_threshold_1))
                converged = true;
                break
            end
            if((concheckcount - concheckcount_last_ts_change) >= 20)
                connerr(2,concheckcount) = mean(connerr(1,concheckcount-19:concheckcount));
                connerr(3,concheckcount) = (mean(connerr(1,concheckcount-19:concheckcount))-connerr(1,concheckcount))/connerr(1,concheckcount);%/ts;
                connerr(4,concheckcount) = mean(diff(connerr(1,concheckcount-19:concheckcount))./connerr(1,concheckcount-19:concheckcount-1));%/ts;
                connerr(6,concheckcount) = mean(diff(connerr(5,concheckcount-19:concheckcount))./connerr(5,concheckcount-19:concheckcount-1))/ts;
                connerr(7,concheckcount) = mean(diff(connerr(1,concheckcount-10:concheckcount))./connerr(1,concheckcount-10:concheckcount-1));
                % Decrease time step because we are not converging
                if (((connerr(4,concheckcount) > -3e-3) && (connerr(7,concheckcount) > -3e-3)) || ((concheckcount-concheckcount_last_ts_change) >= 400/ts))
                    if (ts <= ts_min); fprintf('ts_min reached before convergence.'); break
                    elseif (Ds(1)-Ds_min_step <1e-9); fprintf('Ds_min_step reached before convergence.'); break
                    end
                    concheckcount_last_ts_change = concheckcount;
                    ts = ts/ts_dec_mult;
                    if (ts < ts_min); ts = ts_min; end
                    % **** Reinit D, Nd, Ds, P, and Ddot ****
                    Dold = D;
                    if (useOptimizedDspacing == 0)   % Using linear D spacing
                        Ds_max = max(Ds_min_step,Ds(1)/ts_dec_mult);
                        D = linspace(0,1,floor(Ds_max^-1));
                        Ds=[diff(D) 0];
                        Ds(end)=Ds(end-1);    % one short because of using diff.
                        ts = Ds(1)/(Ddot_static+Dscale); % Corrects ts for Ds spacing
                    else
                        % Using optimized spacing
                        if(ts*(Ddot_static+Dscale)<Ds_min_step); ts = Ds_min_step/(Ddot_static+Dscale); end
                        Ds_min_step_temp = max(ts*(Ddot_static+Dscale)/Ds_min_scalar,Ds_min_step);
                        D=zeros(1,round(Ds_min_step_temp^-1)+1);
                        i=1;
                        while (D(i) < 1)
                            P_d = f_P(D(i),C,D_off,k);
                            Ddot_delta = f_Ddot(Dscale,ts^-1,P_d);
                            if (Ddot_delta < Ds_min_step_temp)
                                Ddot_delta = Ds_min_step_temp;
                            end
                            if ((D(i)+Ddot_delta)> 1); break; end
                            D(i+1) = min(1,D(i)+Ddot_delta);
                            i = i+1;
                        end
                        D=D(1:i); % Truncate unused vector positions
                        Ds=[diff(D) 0];
                        Ds(end)=Ds(end-1);
                        clear i P_d Ddot_delta Ds_min_step_temp
                    end
                    Nd = max(0,interp1(Dold,Nd,D,'spline'));
                    Nd = Nd/(sum(Nd)/Ntot);
                    P = f_P(D,C,D_off,k);
                    Ddot = f_Ddot(Dscale,Ds,P);     % Scaled for degradation step
                    [a, index] = min(abs(Dold(round(x))-D));
                    x = min(index,length(D)); % Estimate
                    if any(Ddot.*ts-1 > 1e-6); error('Ddot cannot be greater than 1. Decrease ts or increase Ds.'); end
                    %fprintf('Decreasing ts to %.2e.\n',ts);
                    clear Dold a index;
                end
```

```
        end
        concheckcount = concheckcount+1;
    elseif (mod(tloopcount,100) == 2) % Make sure pop # isn't drifting too far
        Ntoti = sum(Nd);
        Ntoterr = (Ntoti-Ntot)/Ntot;
        if (abs(Ntoterr) >= 0.01)   % Rescale to correct back to Ntot
            Nd = (1-(Ntoti-Ntot)/Ntoti)*Nd;
        end
        if (abs(Ntoterr) >= 0.10)
            % Current sim settings are not optimal. Try decreasing Ds and starting over.
            if (Ds(1) > Ds_min_step)
                Dold = D;
                Ds_max = max(Ds_min_step,Ds(1)/Ds_dec_mult);
                if ((Ds(1)-Ds_max) < 1e-9); fprintf('Minimum Ds step reached. Halting calculation.\n'); break; end
                if (useOptimizedDspacing == 0)   % Use linear D spacing
                    D = linspace(0,1,floor(Ds_max^-1));
                    Ds = [diff(D) 0];
                    Ds(end) = Ds(end-1);
                    ts = Ds(1)/(Ddot_static+Dscale);
                else
                    % Using optimized D spacing
                    ts = Ds_max/(Ddot_static+Dscale);
                    Ds_min_step_temp = max(ts*(Ddot_static+Dscale)/Ds_min_scalar,Ds_min_step);
                    D = zeros(1,round(Ds_min_step_temp^-1)+1);
                    i = 1;
                    while (D(i) < 1)
                        P_d = f_P(D(i),C,D_off,k);
                        Ddot_delta = f_Ddot(Dscale,ts^-1,P_d);
                        if (Ddot_delta < Ds_min_step_temp)
                            Ddot_delta = Ds_min_step_temp;
                        end
                        if ((D(i)+Ddot_delta)> 1); break; end
                        D(i+1) = min(1,D(i)+Ddot_delta);
                        i = i+1;
                    end
                    D=D(1:i); % Truncate unused vector positions
                    clear i P_d Ddot_delta Ds_min_step_temp
                    Ds=[diff(D) 0];
                    Ds(end)=Ds(end-1);
                end
                if (abs(Ntoterr) >= 3)
                    Nd = max(0,interp1(D_def,Nd_def,D,'spline'));
                else
                    Nd = max(0,interp1(Dold,Nd,D,'spline'));
                end
                Nd = Nd/(sum(Nd)/Ntot);
                P = f_P(D,C,D_off,k);
                Ddot = f_Ddot(Dscale,Ds,P);

                x = length(Nd);
                if any(Ddot.*ts-1 > 1e-6); error('Ddot cannot be greater than 1. Decrease ts or increase Ds.'); end
                Ndprev = zeros(length(Nd),1);
                concheckcount = 1;
                tloopcount = 3;
                connerr = zeros(6,1000);
                t = 0;
                Nd_old = Nd;
                degVals = zeros(1,length(Nd));
                concheckcount_last_ts_change = 0;
                fprintf('Ntoterr exceeded threshold. Rerunning with decreased Ds_max (%.2e).\n',Ds(1));
                clear Ds_max_old Ds_max Dold
                continue;
            else
                fprintf('Problem! Ntoterr exceeded threshold and Ds_min_step also reached.\n');
                break;
            end
        end
    end
    tloopcount = tloopcount + 1;
end
% END TIME STEP LOOP

if (t >= t_stop); fprintf('Calculation halted. Maximum simulation time exceeded.\n'); end

Ntotf = sum(Nd);    % Number of molecules after sim. Will be different than Ntot
                    % due to accumulation of small numerical errors.

D_avg = sum(D.*Nd)/Ntotf;
P_avg = f_P(D_avg,C,D_off,k);

% Diagnostics
Ntotf_sweep(Ndotrc_loopcount) = Ntotf;
converged_sweep(Ndotrc_loopcount) = converged;
connerr_sweep(Ndotrc_loopcount, 1:length(connerr(1,:))) = connerr(8,:);

% Results
results(Ndotrc_loopcount, 1:3) = [Ndotrc, D_avg, P_avg];
fprintf('Calculation for %d molecules per unit time completed. Converged = %d\n',Ndotrc, converged);

% For plotting Frequency as a function of D
FreqByD = max(0,interp1(D,Nd,D_common,'spline'));
```

```
    FreqByD = length(FreqByD)*FreqByD./sum(FreqByD);
    results_2(1,1:length(D_common),Ndotrc_loopcount) = D_common;
    results_2(2,1:length(D_common),Ndotrc_loopcount) = P_from_D_common;
    results_2(3,1:length(D_common),Ndotrc_loopcount) = FreqByD;

    % For plotting Frequency as a function of P
    FreqByP = max(0,interp1(D,Nd,D_from_P_common,'spline'));
    transformToP = [diff(D_from_P_common) 0];
    FreqByP = transformToP.*FreqByP;
    FreqByP = length(FreqByP)*FreqByP./sum(FreqByP);
    results_2(4,1:length(P_common),Ndotrc_loopcount) = P_common;
    results_2(5,1:length(P_common),Ndotrc_loopcount) = D_from_P_common;
    results_2(6,1:length(P_common),Ndotrc_loopcount) = FreqByP;

    if (D(find(Nd(:)<1e-1,1)) > D_max_result); D_max_result = D(find(Nd(:)<1e-1,1)); end
end
clear degVals num Ndprev renPr renRates Ntot_t tloopcount x_limited;

if (min(converged_sweep) == 1); fprintf('\nAll results converged\n\n');
else; fprintf('\nConvergence problem. Check results.\n\n');
end

% Renewal rate and Work rate / Renewal rate
results(:,4)=100*results(:,1)./Ntot;    % Renewal rate in % of total population per unit time
results(:,5)=100*w_dc*results(:,3)./results(:,4);    % Work rate (% of max possible) / (renewal rate in % of total population per unit time)

toc();

% Make some graphs and a movie, save Matlab data, export results to an Excel file
dateStamp = datestr(now,'mm-dd-yyyy__HH_MM_SS');
if ((num_Ndotrc_datapoints > 1) && isempty(m_array) && isempty(P_ren_max_array))
    ROI_P60 = interp1(results(:,3),results(:,5),0.6,'spline');
    ROI_P80 = interp1(results(:,3),results(:,5),0.8,'spline');
    fprintf('\nROI = %.3f @ P = 0.80\nROI = %.3f @ P = 0.60\n',ROI_P80,ROI_P60);
    fprintf('ROI is %.2f%% greater at P = 0.60 compared to P = 0.8\n\n',100*(ROI_P60-ROI_P80)/ROI_P80);
    [maxROI, maxROI_index] = max(results(:,5));
    fprintf('Maximum ROI is %.4f @ P = %.4f\n\n',maxROI,results(maxROI_index,3));

    set(gcf,'Position',[330,100,1450,870]);
    subplot(2,2,1);
    plot(D,P,'r');
    xlim([0 1]);
    ylim([0 1]);
    title('Degradation State versus Biomolecular Performance');
    xlabel('Degradation State (D)');
    ylabel('Performance(P)');
    subplot(2,2,2);
    plot(results(:,3),results(:,4)) % P by Renewal Rate (%/unit time)
    avgRenRate = sum(abs(diff(results(:,3))).*(results(2:end,4)))/(max(results(2:end,3))-min(results(2:end,3)));
    xlim([0 1]);
    ylim([0 min(avgRenRate*4,max(results(:,4)))]);
    set(gca,'xdir','reverse');
    title('Biomolecular Peformance vs Renewal Rate');
    xlabel('Performance (P)');
    ylabel({'Renewal Rate'; '(% renewed / unit time)'});
    subplot(2,2,3);
    plot(tdeg,P_common)
    xlim([0 tdeg(find(P_common(:)>0.05,1))]);
    ylim([0 1]);
    title('Performance Degradation with Time');
    xlabel('Time');
    ylabel('Biomolecular Performance');
    subplot(2,2,4);
    plot(results(:,3),results(:,5)) % P by Renewal Rate (%/unit time)
    text1 = sprintf('%.4f @ P = %.4f \\rightarrow',maxROI,results(maxROI_index,3));
    text(results(maxROI_index,3),maxROI,text1,'HorizontalAlignment','right')
    clear text1;
    set(gca,'xdir','reverse');
    xlim([0 1]);
    title('ROI from Renewal');
    xlabel('Performance (P)');
    ylabel({'Work Rate / Renewal Rate'; '((% of max possible) / (% renewed / unit time))'});
    if (write_to_xls_file == 1)
        sim_parameters_1 = {'tdeg50',tdeg50;'tdeg75',tdeg75;'tdeg25',tdeg25;'w_dc',w_dc;'fitType',fitType;
            'nResult_threshold',nResult_threshold;'Cinit',Cinit;'kInit',kInit;'calc_f_d',calc_f_d};
        sim_parameters_2 = {'k',k;'Dp50',Dp50;'C',C;'D_off',D_off;'delta',delta;'K',Dscale/w_dc;'K/Dp50',Dscale/(w_dc*Dp50);'L',Ddot_static};
        sim_parameters_3 = {'m',m;'P_ren_max',P_ren_max;'Dstatic_type',Dstatic_type;'Dstatic_mult',Dstatic_mult;'tdeg50_static',...
            tdeg50_static;'Ntot',Ntot;'Ndotrc_min',Ndotrc_min;'Ndotrc_max',Ndotrc_max;'numNdotrc_datapoints',num_Ndotrc_datapoints};
        sim_parameters_4 = {'ts_init',ts_init;'ts_min',ts_min;'t_stop',t_stop;'convergence_threshold_1',convergence_threshold_1;...
            'convergence_threshold_2',convergence_threshold_2;'Ds_min_step',Ds_min_step;'Ds_min_num',Ds_min_num;...
            'useOptimizedDspacing',useOptimizedDspacing;'Ds_min_scalar',Ds_min_scalar;'Ds_dec_mult',Ds_dec_mult;'ts_dec_mult',ts_dec_mult};
        filename = strcat('output_data_',dateStamp,'.xlsx');
        fprintf('Writing .xls output file: %s\n',filename);
        xlswrite(filename,sim_parameters_1,1,'A1')
        xlswrite(filename,sim_parameters_2,1,'E1')
        xlswrite(filename,sim_parameters_3,1,'I1')
        xlswrite(filename,sim_parameters_4,1,'M1')
        A = {'Ndotrc','D_avg','P_avg','Ren Rate (%/time)','ROI'};
        xlswrite(filename,A,1,'A17:E17')
        xlswrite(filename,results,1,'A18')
        fprintf('Write completed...\n');
```

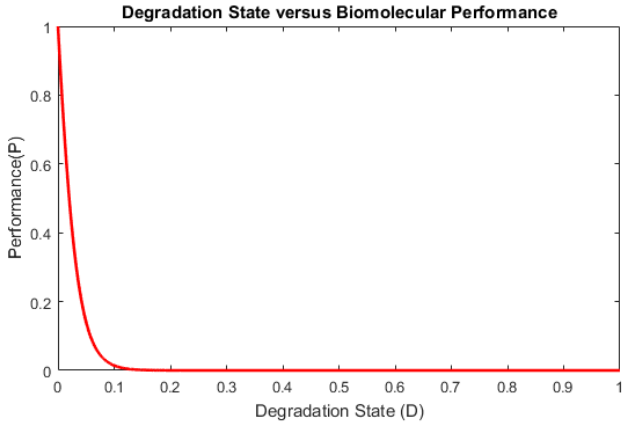
```
            clear sim_parameters_1 sim_parameters_2 sim_parameters_3 sim_parameters_4 A
        end
        if (save_matlab_variables_to_file == 1)
            filename = strcat('output_data_',dateStamp);
            fprintf('Writing Matlab data output file: %s\n',filename);
            save(filename);
            fprintf('Write completed...\n');
        end
        if (captureMovie == true)
            F(length(Ndotrc_array)) = struct('cdata',[],'colormap',[]);    %#ok<SAGROW> % Movie frames
            figure;
            set(gcf,'Position',[850,100,610,890]);
            rect = get(gcf,'Position');
            rect(1:2) = [0 0];
            dim = [.55 .5 .3 .3];
            for movieLoop=1:length(Ndotrc_array)
                if (movieType == 1)     % P vs Frequency
                    plot(results_2(4,:,movieLoop),results_2(6,:,movieLoop),'r');
                    ylim([0 min(30,max(max(results_2(6,:,:))))]);
                    xlim([0 1]);
                    set(gca,'xdir','reverse');
                    str = sprintf('Renewal Rate:\n%.4f\n(%% renewed / unit time)',results(movieLoop,4));
                    annotation('textbox',dim,'String',str,'FitBoxToText','on','HorizontalAlignment','center');
                    xlabel('Biomolecular Performance (P)');
                    ylabel('Frequency');
                elseif (movieType == 2) % D vs Frequency
                    plot(results_2(1,:,movieLoop),results_2(3,:,movieLoop),'r');
                    ylim([0 max(max(results_2(3,:,:)))]);
                    xlim([0 D_max_result]);
                    str = sprintf('Renewal Rate:\n%.4f\n(%% renewed / unit time)',results(movieLoop,4));
                    annotation('textbox',dim,'String',str,'FitBoxToText','on','HorizontalAlignment','center');
                    xlabel('Equivalent Degradation State (D)');
                    ylabel('Frequency');
                else
                    fprintf('Invalid movie type specified. Movie not created.\n');
                    close; clear F;
                    break;
                end
                F(:,movieLoop) = getframe(gcf,rect);
                clf;
            end
            if exist('F','var')
                if (saveMovie == 1)
                    filename = strcat('movie_',dateStamp);
                    vidObj = VideoWriter(filename,'MPEG-4'); %#ok<TNMLP>
                    vidObj.Quality = 100;
                    vidObj.FrameRate = 10;
                    open(vidObj);
                    writeVideo(vidObj, F);
                    close(vidObj);
                    winopen(strcat(filename,'.mp4'));
                end
                close; clear F;
            end
        end
        clear dim rect str
    elseif ((num_Ndotrc_datapoints > 1) && (~isempty(m_array) || ~isempty(P_ren_max_array)))   % m or P_ren_max sweeping
        if (~isempty(m_array))
            label_suffix = sprintf('_m_%.3f',m);
        else
            label_suffix = sprintf('_P_ren_max_%.3f',P_ren_max);
        end
        filename = strcat('output_data_',dateStamp_mSweep,label_suffix,'.mat');
        clear label_suffix
        fprintf('Saving Matlab data output file: %s\n',filename);
        save(filename);
        fprintf('Save completed...\n');
    else    % Outputs for single Ndotrc simulation
        fprintf('D_avg = %.3f, P_avg = %.3f\n',D_avg,P_avg);
        set(gcf,'Position',[760,100,980,843]);
        subplot(2,2,1);
        plot(D_common,FreqByD,'r');
        title('Biomolecular Distribution across Degradation States');
        xlabel('Equivalent Degradation State (D)');
        ylabel('Frequency');
        subplot(2,2,3);
        plot(P_common,FreqByP,'r');
        xlim([0 1]);
        set(gca,'xdir','reverse');
        title('Biomolecular Distribution across Perfomance');
        xlabel('Biomolecular Performance');
        ylabel('Frequency');
        subplot(2,2,2);
        plot(D,P,'r');
        ylim([0 1]);
        title('Degradation State versus Biomolecular Performance');
        xlabel('Degradation State (D)');
        ylabel('Biomolecular Performance');
        subplot(2,2,4);
        plot(tdeg,P_common)
        xlim([0 tdeg(find(P_common(:)>0.05,1))]);
```

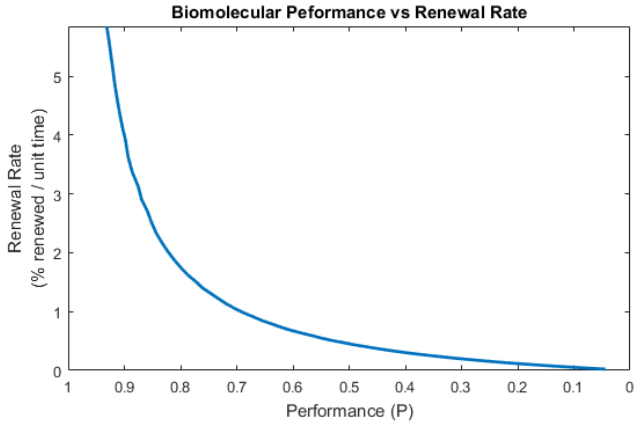
```
        ylim([0 1]);
        title('Performance Degradation with Time');
        xlabel('Time');
        ylabel('Biomolecular Performance');
        if (write_to_xls_file == 1)
            sim_parameters_1 = {'tdeg50',tdeg50;'tdeg75',tdeg75;'tdeg25',tdeg25;'w_dc',w_dc;'fitType',fitType;
                'nResult_threshold',nResult_threshold;'Cinit',Cinit;'kInit',kInit;'calc_f_d',calc_f_d};
            sim_parameters_2 = {'k',k;'Dp50',Dp50;'C',C;'D_off',D_off;'delta',delta;'K',Dscale/w_dc;'K/Dp50',Dscale/(w_dc*Dp50);'L',Ddot_static};
            sim_parameters_3 = {'m',m;'P_ren_max',P_ren_max;'Dstatic_type',Dstatic_type;'Dstatic_mult',Dstatic_mult;'tdeg50_static',...
                tdeg50_static;'Ntot',Ntot;'Ndotrc',Ndotrc_min;};
            sim_parameters_4 = {'ts_init',ts_init;'ts_min',ts_min;'t_stop',t_stop;'convergence_threshold_1',convergence_threshold_1;...
                'convergence_threshold_2',convergence_threshold_2;'Ds_min_step',Ds_min_step;'Ds_min_num',Ds_min_num;...
                'useOptimizedDspacing',useOptimizedDspacing;'Ds_min_scalar',Ds_min_scalar;'Ds_dec_mult',Ds_dec_mult;'ts_dec_mult',ts_dec_mult};
            filename = strcat('output_data_1Nrc_',dateStamp,'.xlsx');
            fprintf('Writing output file: %s\n',filename);
            xlswrite(filename,sim_parameters_1,1,'A1')
            xlswrite(filename,sim_parameters_2,1,'E1')
            xlswrite(filename,sim_parameters_3,1,'I1')
            xlswrite(filename,sim_parameters_4,1,'M1')

            xlsA = {'D_avg',D_avg;'P_avg',P_avg};
            xlswrite(filename,xlsA,1,'A15')
            xlsB = {'D_common','FreqByD','','P_common','FreqByP','','D','P','','tdeg','P_common'};
            xlswrite(filename,xlsB,1,'A18')
            xlsC = [D_common; FreqByD]';
            xlsD = [P_common; FreqByP]';
            xlsE = [D; P]';
            xlsF = [tdeg; P_common]';
            xlswrite(filename,xlsC,1,'A19')
            xlswrite(filename,xlsD,1,'D19')
            xlswrite(filename,xlsE,1,'G19')
            xlswrite(filename,xlsF,1,'J19')
            fprintf('Write completed...\n');
            clear sim_parameters_1 sim_parameters_2 sim_parameters_3 sim_parameters_4 xlsA xlsB xlsC xlsD xlsE xlsF
        end
        if (save_matlab_variables_to_file == 1)
            filename = strcat('output_data_1Nrc_',dateStamp);
            fprintf('Writing Matlab data output file: %s\n',filename);
            save(filename);
            fprintf('Write completed...\n');
        end
    end
    clear dateStamp filename
end
clear dateStamp_mSweep
```

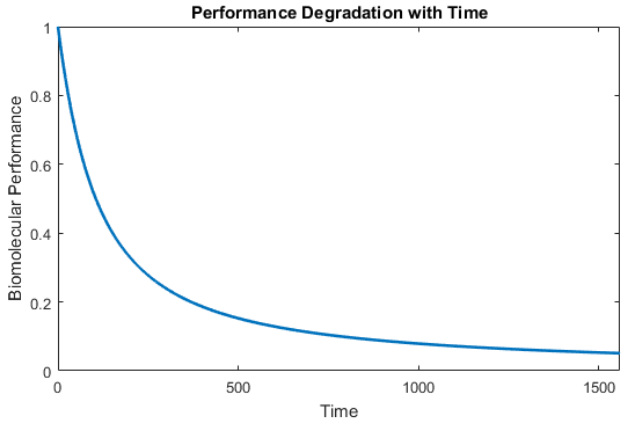
```
Calculating best fit...
Datafit succeeded...

tdeg75 = 38.0002
tdeg25 = 284.3678
tdeg10 = 783.2077

Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 3.000000e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 2.789059e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 2.592949e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 2.410629e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 2.241129e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (3.39e-05).
Calculation for 2.083546e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Calculation for 1.937044e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Calculation for 1.800843e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Calculation for 1.674219e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Calculation for 1.556498e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
```

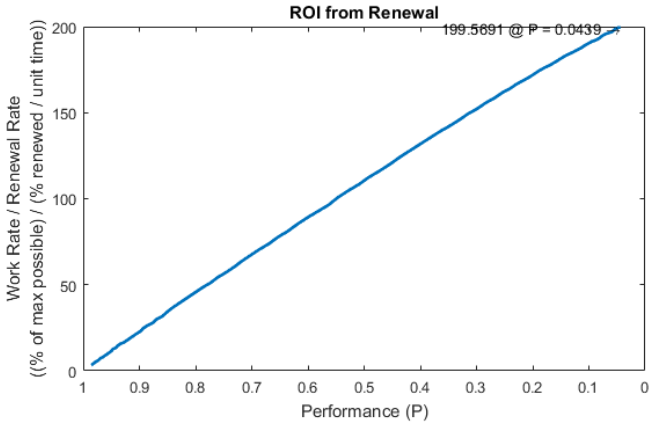
```
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (6.78e-05).
Calculation for 1.447055e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Calculation for 1.345307e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Calculation for 1.250713e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Calculation for 1.162771e+08 molecules per unit time completed. Converged = 1
Calculation for 1.081012e+08 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Calculation for 1.005002e+08 molecules per unit time completed. Converged = 1
Calculation for 9.343365e+07 molecules per unit time completed. Converged = 1
Ntoterr exceeded threshold. Rerunning with decreased Ds_max (1.36e-04).
Calculation for 8.686397e+07 molecules per unit time completed. Converged = 1
Calculation for 8.075623e+07 molecules per unit time completed. Converged = 1
Calculation for 7.507796e+07 molecules per unit time completed. Converged = 1
Calculation for 6.979894e+07 molecules per unit time completed. Converged = 1
Calculation for 6.489111e+07 molecules per unit time completed. Converged = 1
Calculation for 6.032837e+07 molecules per unit time completed. Converged = 1
Calculation for 5.608645e+07 molecules per unit time completed. Converged = 1
Calculation for 5.214280e+07 molecules per unit time completed. Converged = 1
Calculation for 4.847644e+07 molecules per unit time completed. Converged = 1
Calculation for 4.506788e+07 molecules per unit time completed. Converged = 1
Calculation for 4.189898e+07 molecules per unit time completed. Converged = 1
Calculation for 3.895290e+07 molecules per unit time completed. Converged = 1
Calculation for 3.621398e+07 molecules per unit time completed. Converged = 1
Calculation for 3.366763e+07 molecules per unit time completed. Converged = 1
Calculation for 3.130033e+07 molecules per unit time completed. Converged = 1
Calculation for 2.909949e+07 molecules per unit time completed. Converged = 1
Calculation for 2.705339e+07 molecules per unit time completed. Converged = 1
Calculation for 2.515117e+07 molecules per unit time completed. Converged = 1
Calculation for 2.338269e+07 molecules per unit time completed. Converged = 1
Calculation for 2.173856e+07 molecules per unit time completed. Converged = 1
Calculation for 2.021004e+07 molecules per unit time completed. Converged = 1
Calculation for 1.878900e+07 molecules per unit time completed. Converged = 1
Calculation for 1.746787e+07 molecules per unit time completed. Converged = 1
Calculation for 1.623964e+07 molecules per unit time completed. Converged = 1
Calculation for 1.509777e+07 molecules per unit time completed. Converged = 1
Calculation for 1.403619e+07 molecules per unit time completed. Converged = 1
Calculation for 1.304925e+07 molecules per unit time completed. Converged = 1
Calculation for 1.213171e+07 molecules per unit time completed. Converged = 1
Calculation for 1.127868e+07 molecules per unit time completed. Converged = 1
Calculation for 1.048563e+07 molecules per unit time completed. Converged = 1
Calculation for 9.748348e+06 molecules per unit time completed. Converged = 1
Calculation for 9.062904e+06 molecules per unit time completed. Converged = 1
Calculation for 8.425657e+06 molecules per unit time completed. Converged = 1
Calculation for 7.833217e+06 molecules per unit time completed. Converged = 1
Calculation for 7.282434e+06 molecules per unit time completed. Converged = 1
Calculation for 6.770378e+06 molecules per unit time completed. Converged = 1
Calculation for 6.294327e+06 molecules per unit time completed. Converged = 1
Calculation for 5.851749e+06 molecules per unit time completed. Converged = 1
Calculation for 5.440290e+06 molecules per unit time completed. Converged = 1
Calculation for 5.057762e+06 molecules per unit time completed. Converged = 1
Calculation for 4.702132e+06 molecules per unit time completed. Converged = 1
Calculation for 4.371507e+06 molecules per unit time completed. Converged = 1
Calculation for 4.064130e+06 molecules per unit time completed. Converged = 1
Calculation for 3.778365e+06 molecules per unit time completed. Converged = 1
Calculation for 3.512694e+06 molecules per unit time completed. Converged = 1
Calculation for 3.265703e+06 molecules per unit time completed. Converged = 1
Calculation for 3.036079e+06 molecules per unit time completed. Converged = 1
Calculation for 2.822601e+06 molecules per unit time completed. Converged = 1
Calculation for 2.624133e+06 molecules per unit time completed. Converged = 1
Calculation for 2.439620e+06 molecules per unit time completed. Converged = 1
Calculation for 2.268081e+06 molecules per unit time completed. Converged = 1
Calculation for 2.108604e+06 molecules per unit time completed. Converged = 1
Calculation for 1.960340e+06 molecules per unit time completed. Converged = 1
Calculation for 1.822501e+06 molecules per unit time completed. Converged = 1
Calculation for 1.694354e+06 molecules per unit time completed. Converged = 1
Calculation for 1.575217e+06 molecules per unit time completed. Converged = 1
Calculation for 1.464458e+06 molecules per unit time completed. Converged = 1
Calculation for 1.361486e+06 molecules per unit time completed. Converged = 1
Calculation for 1.265755e+06 molecules per unit time completed. Converged = 1
Calculation for 1.176755e+06 molecules per unit time completed. Converged = 1
Calculation for 1.094013e+06 molecules per unit time completed. Converged = 1
Calculation for 1.017088e+06 molecules per unit time completed. Converged = 1
Calculation for 9.455731e+05 molecules per unit time completed. Converged = 1
Calculation for 8.790863e+05 molecules per unit time completed. Converged = 1
Calculation for 8.172744e+05 molecules per unit time completed. Converged = 1
Calculation for 7.598087e+05 molecules per unit time completed. Converged = 1
Calculation for 7.063836e+05 molecules per unit time completed. Converged = 1
Calculation for 6.567151e+05 molecules per unit time completed. Converged = 1
Calculation for 6.105390e+05 molecules per unit time completed. Converged = 1
Calculation for 5.676096e+05 molecules per unit time completed. Converged = 1
Calculation for 5.276988e+05 molecules per unit time completed. Converged = 1
Calculation for 4.905943e+05 molecules per unit time completed. Converged = 1
Calculation for 4.560988e+05 molecules per unit time completed. Converged = 1
Calculation for 4.240287e+05 molecules per unit time completed. Converged = 1
Calculation for 3.942136e+05 molecules per unit time completed. Converged = 1
Calculation for 3.664950e+05 molecules per unit time completed. Converged = 1
Calculation for 3.407253e+05 molecules per unit time completed. Converged = 1
Calculation for 3.167676e+05 molecules per unit time completed. Converged = 1
```

```
Calculation for 2.944945e+05 molecules per unit time completed. Converged = 1
Calculation for 2.737875e+05 molecules per unit time completed. Converged = 1
Calculation for 2.545364e+05 molecules per unit time completed. Converged = 1
Calculation for 2.366390e+05 molecules per unit time completed. Converged = 1
Calculation for 2.200000e+05 molecules per unit time completed. Converged = 1

All results converged

Elapsed time is 43.456379 seconds.

ROI = 45.671 @ P = 0.80
ROI = 89.242 @ P = 0.60
ROI is 95.40% greater at P = 0.60 compared to P = 0.8

Maximum ROI is 199.5691 @ P = 0.0439

Writing Matlab data output file: output_data_09-07-2017__11_01_37
Write completed...
```









Published with MATLAB® R2017a