```
function [F, T] = transientFinDiffFuncNonLin(param, readings, tOffset, ...
reading1, readingF, offsets2, amb1, Pin, eq, iceEnd, blackRod, moistRod)
    %Finite difference calculations for aluminum rod transients
    %param meanings: 1=k, 2=kc, 3=eps, 4=kcEnd, 5=epsEnd, 6=c. Parameters
    %that are not used in a particular scenario are
    %ignored.
    %Other parameters:
    %readings: Data from sensors
    %tOffset: starting time for heating
    %reading1: starting reading for heating
    %readingF: final reading to use
    %offsets2: secondary offset to ensure all sensors are identical
    %initially
    %amb1: whether the ambient pin is 1 (if false, 6)
    %Pin: input power to rod
    %eq: whether kc and eps are equal inside and at end of rod
    %iceEnd: whether the end of the rod is 0 degrees C
    %blackRod: for black rod experiment: shuffles sensors
    %Return values:
    %F: a matrix containing the errors in each reading
    %T: a matrix containing temperatures for the sensor locations.

    %%Temp sensor calibration stuff
    factors = [1.79 1.81 1.53 1.46 2.06 2];
    offsets = [4.14 2.25 0.16 3.64 -0.35 0];

    %%shuffle sensors for black rod
    if(blackRod)
        factors = factors([2 3 5 4 1 6]);
        offsets = offsets([2 3 5 4 1 6]);
    end

    sensorPos = [1 6 9 12 19];

    %%Set up constants
    sigma = 5.670e-8; % W /^-2 K^-4
    c = param(6); %specific heat capacity
    rho = 2700; %kg/m^3

    L = 0.305; %length
    r = 0.0111; %radius
    A = pi*r^2;

    %%Handle case for differing ambient pins
    if(amb1)
        Tamb = mean(squeeze(readings(1, 1, 1:35)))/2; %ambient temperature
        factors = [2 factors(1:5)];
        offsets = [0 offsets(1:5)];
    else
        Tamb = mean(squeeze(readings(1, 6, 1:35)))/2;
    end
```

```
nstepsT = ceil(readings(3, 6, readingF));

time = nstepsT; %seconds

nstepsX = 19;

dt = time / nstepsT;
dx = L / nstepsX;

m = dx*A*rho;

T = zeros(nstepsT + 1, nstepsX + 1); %K
P = zeros(nstepsT + 1, nstepsX + 1); %W

T(1, :) = Tamb;
P(:, 1) = Pin;

%%Iteration step
for i = 1:(nstepsT)
    for j = 1:(nstepsX + 1)
        if j == nstepsX + 1
            S = A;
            Pcond = 0;
            if(eq)
                eps = param(3);
                kc = param(2);
            else
                eps = param(5);
                kc = param(4);
            end
            if(iceEnd) Tamb = 0; end;%(ice water)
        else
            S = dx*pi*r*2;
            Pcond = -param(1)*A*(T(i, j) - T(i, j+1))/dx;
            eps = param(3);
            kc = param(2); %represents heat loss through insulation
            %actually probably represents conduction
            if(iceEnd) Tamb = T(1, 1); end; %(ice water)
        end
        Pconv = -S * (T(i, j) - Tamb) * kc;
        Prad = -eps * S * sigma * ((T(i, j) + 273.15)^4 - (Tamb+273.15)^4);
        Ptot = Pcond + Pconv + Prad + P(i, j);
        if(j >= 12 && j < 19 && moistRod)
            Ptot = Pcond + Pconv/2 + Prad - param(7) / 6 + P(i, j);
        end
        P(i, j+1) = -Pcond;
        T(i+1, j) = Ptot*dt/(c*m) + T(i, j);
    end
end
rng = reading1:readingF;

%%Calculate errors
for i = 1:5
    j = i+amb1;
```

```
        times = squeeze(readings(3, j, rng));
        values = squeeze(readings(1, j, rng));
        %if(i == 4); j = 3; elseif(i == 3); j=4; else j = i; end
        F(j-amb1, :) = interp1((1:length(T(:,i)))+tOffset, ...
            T(:,sensorPos(i)), times)-(values-offsets(j))/factors(j)+offsets2(j);
    end

    %%Plot residuals
    p = plot(F');
    title(param);

    clrs = 'rbgmk';
    for i=1:numel(p)
        set(p(i), 'color', clrs(i));
    end
    pause(.01);
end
```

Published with MATLAB® R2013a