```
% ### EXwalker2D.m ###        07.15.15 CB

% Simulates a 2-D Brownian walker
% - polar coords. for computing a step
% - step size is unit value with normally distrib. val.
% - direction-wise, allows for either (via A.Pbias):
% o step direction is uniformally distributed over circle
% o Gaussian-like directional bias (kinda kludgy, but works)
% - [IN PROGRESS] allow for a (circular) boundary condition (via
A.Pbound)
% that either reflects or is periodic

clear
% =======
walkerNum= 200;  % # of walkers to compute
steps= 500; % # of steps to take by walker
A0= 0.1;    % limiting stochastic factor (re 1) for unit step size
(A0=0 --> unit radial steps, A0>0 introduce Gaussian variance)
% ---
A.Pbias= 0;       % boolean to create a directional bias
A.alpha= 0.2;     % bias factor [0,1] --> small (~0.1 means stronger
bias)
A.offset= 0.75;    % offset direction for bias[cyc]
% ---
A.force1D= 0;       % boolean to force angle to be 0 or pi (thus
making this 1-D)
% ---
A.Pbound= 1;       % boolean to create circular boundary (i.e.,
walkers constrained)
A.bndR= 20;         % radius of bounding wall (re origin)
A.boundType= 0;     % boundary condtion (req. A.Pbound=1):
0-"hard" (reflecting), 1-periodic
% ---
axLim= 25;  % bounds for plotting (Fig.66)
kk= 1;  % particle ID to visualize a single walker (Fig.66)
animate= 1;     % boolean to turn on/off movie for an individual
walker (Fig.66)
numWplot= 5;    % # of walkers to plot individual (r^2) paths (Fig.4)
% =======

% ---
% if a constrained walk, force bounding condition (A.bndR) to be much
% larger than mean unit step size (helps avoid some coding headaches
below)
if (A.Pbound==1 && A.bndR <= 5), disp('Make larger bounding
condition'); end

for m= 1:walkerNum
    % walker m initially at origin [i.e., cartesian (0,0)]
    P(m).coord(1,:)= [0 0];
```

```
    for nn=2:steps
        % ----
        P(m).A(nn)= 1+ A0*randn(1);     % (radial) size of nn'th step
for m'th walker
        if (P(m).A(nn)<0),   P(m).A(nn)=0;    end     % make zero size
step if negative (introduces bias?)
        % ----
        % direction of nn'th step for m'th walker (allows possibility
of bias)
        if A.Pbias==0
            P(m).theta(nn)= rand(1)*2*pi;   % no bias
        else
            if (m==1 && nn==2), disp('Radial bias in effect');  end
            P(m).theta(nn)= (A.offset + A.alpha.*randn(1))*2*pi;
% w/ radial bias
        end
        % ----
        % constrain angle such that movement is essentially 1-D
        if A.force1D==1
            P(m).theta(nn)= round(P(m).theta(nn)/(2*pi))*pi;
        end
        % ----
        % update re last position and store away in Cartesian and
radial coords.
        P(m).coord(nn,:)=
[P(m).coord(nn-1,1)+P(m).A(nn)*cos(P(m).theta(nn))
P(m).coord(nn-1,2)+P(m).A(nn)*sin(P(m).theta(nn))];
        P(m).rsq(nn)= P(m).coord(nn,1)^2 + P(m).coord(nn,2)^2;  % new
radial position (squared)
        P(m).phi(nn)= atan2(P(m).coord(nn,2),P(m).coord(nn,1));    %
angle of new position re origin
        % ----
        % if constrained, check that new coords. aren't past wall
(otherwise "reflect")
        if A.Pbound==1
            if (m==1 && nn==2 && A.boundType==0), disp('Circular hard/
reflecting boundary in effect');  end
            if (m==1 && nn==2 && A.boundType==0), disp('Circular
periodic boundary in effect');  end
            temp1= sqrt(P(m).rsq(nn));  % dummy to reduce re-
computation
            if temp1 >= A.bndR
                if A.boundType==0
                    % ### HARD REFLECTION ###
                    % angle stays the same, only radius changes (and
in a simple way)
                    temp2= 2*A.bndR- temp1;     % reflected radial
length
                    %disp([temp1 P(m).A(nn) P(m).theta(nn)
```

```
P(m).phi(nn) temp2]);  % for debugging
                elseif A.boundType==1
                    % ### PERIODIC B.C. ###
                    % both radius changes and angle flips 180
                    temp2= 2*A.bndR- temp1;     % reflected radial
length
                    P(m).phi(nn)= mod(P(m).phi(nn)+pi,2*pi);
                end
                P(m).rsq(nn)= temp2^2;      % squared version
                % revised Cartesian version
                P(m).coord(nn,1)= temp2*cos(P(m).phi(nn));
                P(m).coord(nn,2)= temp2*sin(P(m).phi(nn));
            end
        end
        % ----
        % determine MSD
        P(m).time(nn)= nn;  % "time" is simply the step number (can
rescale as needed)
        %P(m).MSD(nn)= sqrt(P(m).coord(nn,1)^2 + P(m).coord(nn,2)^2);
% radial position (not squared)
        %P(m).MSD(nn)= P(m).coord(nn,1)^2 + P(m).coord(nn,2)^2;  %
squared to get the "S" in MSD
        P(m).MSD(nn)= P(m).rsq(nn);  % note that this is the radial
position squared (hence "S" in MSD)
    end
end

% --------
% compute mean MSD (across all walkers)  --> KLUDGE (better way to do
this sans loops??)
for nn=1:steps
    for m= 1:walkerNum
        val(m)= P(m).MSD(nn);
    end
    meanMSD(nn)= mean(val);
end

% --------
% plot vals. for a (specified) individual walker
if 1==1
    figure(1); clf;
    subplot(211); plot(P(kk).coord(:,1),P(kk).coord(:,2),'k.-');
    xlabel('x'); ylabel('y'); grid on; hold on; title('Walker
position')
    axis([-axLim axLim -axLim axLim])
    % --- (plot a bounding circle)
    if A.Pbound==1
        th= 0:pi/50:2*pi; xunit= A.bndR*cos(th); yunit=
A.bndR*sin(th); h66= plot(xunit, yunit,'r-');
    end
```

```
    % --- (plot MSD for an individual walker)
    subplot(212); plot(P(kk).time,P(kk).MSD,'k-');
    xlabel('Time'); ylabel('Radial displacement (squared)'); grid on;
hold on;
end

% --------
% plot MSD for the ensemble
figure(2); clf;
plot(P(m).time,meanMSD,'k-');
xlabel('Time'); ylabel('MSD'); grid on; hold on;
% if constrained, visualize effective bounding limit
if (A.Pbound==1), h2B=
stem(A.bndR^2,max(meanMSD),'r--','LineWidth',1);
    legend(h2B,'Bounding radius (squared)','Location','SouthEast');
end

% --------
% plot distribution of angular values (polar histogram)
if 1==0
    figure(3); clf;
    % == (single walker) directions taken for each step for an
individual walker
    subplot(221); h3= rose(P(kk).theta,30);
    set(h3,'LineWidth',1.5); x = get(h3,'Xdata'); y = get(h3,'Ydata');
g=patch(x,y,'y');
    title('All steps for a single walker'); grid on; hold on;
    % == (all walkers) directions taken for all steps of all walkers
    subplot(223); h3= rose([P(:).theta],30);
    set(h3,'LineWidth',1.5); x = get(h3,'Xdata'); y = get(h3,'Ydata');
g=patch(x,y,'y');
    title('All steps for all walkers'); grid on; hold on;
    % == (all walkers) final position for all walkers [KLUDGE: not
sure how to do sans loop]
    for mm=1:numel(P)   bank(mm)= P(mm).phi(end);   end
    subplot(224); h3= rose(bank,floor(numel(P)/15));
    set(h3,'LineWidth',1.5); x = get(h3,'Xdata'); y = get(h3,'Ydata');
g=patch(x,y,'y');
    title('Final ang. position of all walkers'); grid on; hold on;
end

% --------
% plot time course (or r^2) for several walkers? (see also Fig.1B)
if 1==1
    figure(4); clf;
    for n=1:numWplot
        hh= 0.8*n/numWplot;  % shading factor (to discern different
traces)
        plot(P(n).time,P(n).MSD,'-','Color',hh*[1 1 1]); grid on; hold
on;
```

```
    end
    leg= plot(P(m).time,meanMSD,'r--','LineWidth',2);   % also plot
ensemble MSD
    xlabel('Time'); ylabel('Radial displacement (squared)');
    title(['Bounding limit= ',num2str(A.bndR),'(squared=
',num2str(A.bndR^2),')']);
    legend(leg,'ensemble MSD','Location','NorthWest');
end

% --------
% movie for an individual walker
if animate==1
    figure(66); clf; axis([-axLim axLim -axLim axLim]); grid on; hold
on;
    for nn=2:steps
        % --- (plot a bounding circle)
        if A.Pbound==1
            th= 0:pi/50:2*pi; xunit= A.bndR*cos(th); yunit=
A.bndR*sin(th); h66= plot(xunit, yunit,'r-');
        end
        % --- (plot/update the track)
        %plot(P(kk).coord(nn,1),P(kk).coord(nn,2),'ko-');
        plot([P(kk).coord(nn-1,1) P(kk).coord(nn,1)],
[P(kk).coord(nn-1,2) P(kk).coord(nn,2)],'k.-');
        pause(0.04);  % {0.04}
    end
end
```