Supplementary Information II

Blocky Bromination of Syndiotactic Polystyrene via Post-Polymerization Functionalization in the Heterogeneous Gel State

Kristen F. Noble,[a] Alexandria M. Noble[b] Samantha J. Talley[a] and Robert B. Moore*[a]
[a]Department of Chemistry and Macromolecules Innovation Institute, Virginia Tech, Blacksburg, VA 24061
[b]Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA 24061

```
% This code simulates copolymers with specified degrees of functionalization
%and random or blocky microstructures and then calculates (1) the sequence
%length and frequency of consecutive units; (2) the probability that a unit
%exists in a crystallizable segment; and (3) the prevalence of unique triad
%sequences in the copolymer chain
% This code was created using MATLAB® R2017a programming software

% Authors: Alexandria M. Noble & Kristen F. Noble

%% Variables
chain_length = 1442;  % number of units in one chain
amt = 0.55;  % fraction of inaccessible styrene units (for a random copolymer
      amt = 0)
keep_size = 5;  % units in the inaccessible block (Note: for the purpose of
      demonstrating how the inaccessible block is created the below code is
      written for a keep_size of 5, however the code can be modified for any
      desired keep_size by following the format of
      Block_InaccessibleFraction.m, shown below. This work used a %keep_size
      of 53.
r = 1000;  % defines the iterative process that generates r number of chains
ts = 3;  % defines that triad sequences will be counted (Note: the code can
      be modified for any desired sequence (e.g., pentads, heptads, etc.) by
      following the format of TriadSequenceCounting.m, shown below.
l = 26;  % units in one crystallizable segment
percent_functionalization = transpose(0.0:0.02:0.40);  % Simulated degrees of
      functionalization (0-40% at intervals of 2%)

% The copolymer chain will contain 1's to represent styrene (s) units and 0's
      to represent brominated (b) styrene units.

%% Outputs
keep = [];  % stores the fixed styrene indices to create the inaccessible
      fraction
chain_matrix = [];  % stores all chains for one degree of functionalization
store_avg = []; % stores the average prevalence for each degree of
      functionalization
store_sd = []; % stores the standard deviation of the prevalences for each
      degree of functionalization
P = [];  % stores the probability that a 1 exists in a segment length of j
      consecutive 1 units
```

```
store_P = [];  % stores the probability that a 1 exists in a crystallizable
      segment for each chain
Table = [];  % stores the average and standard deviation of the probability
      that a 1 exists in a crystallizable segment for each degree of
      functionalization

for vt = 1:size(percent_functionalization,1)  % for each degree of
      functionalization
    store_P = [];
    for uu = 1:r  % generates r number of chains and performs the following
      on each chain

        chain = transpose(ones(1,chain_length));  % creates a chain of 1's of
      length chain_length
        q = round(amt*floor(chain_length));  % defines the number of
      inaccessible 1's by the predefined amount (amt)

        fixed_styrene = randsample(chain_length,q);  % selects the
      inaccessible 1's by random chance

%% run Blocky_InaccessibleFraction.m  % code below
```

%% <u>Blocky_InaccessibleFraction.m</u>

```
% This code establishes the monomer units in the inaccessible fraction
% Note that for the purpose of demonstrating how the inaccessible block is
%created the below code is written for a keep_size of 5, however the code can
%be modified for any desired keep_size by following the format below. %This
%work used a keep_size of 53.
% This code was created using MATLAB® R2017a programming software

% Authors: Alexandria M. Noble & Kristen F. Noble
```

%% -- Begin Code Blocky_InaccessibleFraction -- %%

```
        keep = [];  % stores the fixed styrene indices to create the
      inaccessible fraction

        for k = 1:size(fixed_styrene,1)  % repeat until k is the size of the
      fixed styrene matrix (based on the predetermined percent
      functionalization and chain length)
            while size(keep,1)< q  % while the size of the keep matrix is
      less than the number of inaccessible 1's (q) established above
                if keep_size == 5
                    if fixed_styrene(k)-1<1  % prevents the styrene index
      from being less than 1 (if fixed_styrene = 1 then keep 1,2,3)
                        keep =
      [keep;fixed_styrene(k);fixed_styrene(k)+1;fixed_styrene(k)+2];
                    elseif fixed_styrene(k)-1<2
                        keep = [keep;fixed_styrene(k)-
      1;fixed_styrene(k);fixed_styrene(k)+1;fixed_styrene(k)+2];
                    elseif fixed_styrene(k)+1 > chain_length  % prevents the
      styrene index from being greater than the chain length (if chain_length
      = 1442 and fixed_styrene = 1442 then keep 1440,1441,1442)
                        keep = [keep;fixed_styrene(k)-2;fixed_styrene(k)-
      1;fixed_styrene(k)];
                    elseif fixed_styrene(k)+2 > chain_length
```

```
                    keep = [keep;fixed_styrene(k)-2;fixed_styrene(k)-
1;fixed_styrene(k);fixed_styrene(k)+1];
                else
                    keep = [keep;fixed_styrene(k)-2;fixed_styrene(k)-
1;fixed_styrene(k);fixed_styrene(k)+1;fixed_styrene(k)+2];
                end
            end
        end
        keep = unique(keep);
    end
```

%% -- End Code Blocky_InaccessibleFraction -- %%

```
  q2 = round(percent_functionalization(vt)*floor(chain_length));  %
gives integer value for the percent functionalization (e.g., 40%*length
of chain)
  I2 = transpose(1:1:chain_length);  % converts a row of integer values
into a column

  xr = [];  % specifies which indices are in the accessible fraction
based on indices in the inaccessible fraction
  for n = 1:size(keep,1)
      xr = [xr;find(I2 == keep(n))];
  end

  I2(xr) = [];  % stores the indices not in keep

  rand_bromine = randsample(I2,q2);  % selects at random without
replacement indices in the chain

  chain(rand_bromine,1) = 0;  % changes specified indices in
rand_bromine to zeros
  chain_matrix(:,uu) = chain;

  x1 = find(chain == 0);  % identify bromines
  x2 = find(chain == 1);  % identify indices where chain is equal to 1

  chain_f(x1,1)={'b'};  % change all 0's in the chain to 'b' for
bromine
  chain_f(x2,1)={'s'};  % change all 1's in the chain to 's' for
styrene

  %% run TriadSequenceCounting.m  %code below
```

%% <u>TriadSequenceCounting.m</u>

```
% This code indexes the triad combinations in the chain, calculates the
%frequency and prevalence of the unique triad sequences, and creates a matrix
%of the average and standard deviation of the triad sequence prevalences
%calculated from all generated chains
% Note: this code can be modified for any sequence length (e.g., pentad,
heptad, %etc.)
% This code was created using MATLAB® R2017a programming software

% Authors: Alexandria M. Noble & Kristen F. Noble
```

```
            %% -- Begin Code TriadSequenceCounting -- %%
  tpseq = {};  % stores the triad sequences identified in the loop
below
  for i = 1:chain_length  % loop that repeats based on the chain length
      if i <= chain_length-(ts-1)  % if the step (i) is less than or
equal to the (chain length)-ts-1
          indices =  i:1:i+(ts-1);  % then indices are selected as
shown
          if ts == 3
              tpseq = [tpseq;
[char(chain_f(indices(1))),char(chain_f(indices(2))),char(chain_f(indic
es(3)))], {indices}]; % makes triad sequences from the chain
          end
      end
  end

  if ts == 3  % if calculating the prevalence of triad sequences
      k = 8;  % there are 8 possible sequences

      C = {'sss'; 'ssb'; 'sbs'; 'sbb'; 'bss'; 'bsb'; 'bbs'; 'bbb'};  %
creates an array of the 8 possible triad sequences
      C = table(C,C,zeros(size(C,1),1),zeros(size(C,1),1));  % creates
a table to store the triad sequences, their frequency, and their
prevalence
      C.Properties.VariableNames = {'SeqA','SeqB','Freq','Theo_Prev'};

      for i = 1:size(C,1)  % for each triad sequence in column SeqA,
identify the unique triad sequences in the chain and count their
frequency (e.g., C.SeqA(i))
          flipped = {fliplr(C.SeqA{i,1})};  % flip C.SeqA in order to
count forward and reverse variants of the sequence

          if strcmp(C.SeqA(i),flipped)  % if SeqA equals the flipped
SeqA (e.g., SeqA = 'sss')
              x = find(strcmp(C.SeqA(i),tpseq(:,1))); % identify the
indices in tpseq where SeqA(i) occurs
              C.SeqB(i) = C.SeqA(i);  % SeqA == SeqB
              C.Freq(i) = size(x,1);  % obtain the size of x, which is
the frequency of each of the symmetric triad sequences without double
counting

          else  % if SeqA does not equal the flipped SeqA (SeqA(i) is
asymmetric)
              C.SeqB(i) = flipped; % populate column SeqB with the
appropriate variant from SeqA (e.g., 'ssb' = 'bss')
              x = size(find(strcmp(C.SeqA(i),tpseq(:,1))),1)+...
                  size(find(strcmp(flipped,tpseq(:,1))),1); % obtain
the size of tpseq for the locations of C.SeqA(i) and the variant in
C.SeqB(i) and add the two size functions together to determine the
overall frequency
              C.Freq(i) = x ;  % records the overall frequency of
asymmetric SeqA(i) and its variant
          end
      end
  end
```

```
  C.Theo_Prev = C.Freq/size(tpseq,1);  % calculates the prevalence of
each triad sequence as the ratio of the frequency to the total number
of triads in the chain (size of tpseq)

  prev(:,uu) = C{:,4};  % stores the prevalences for each triad
sequence from all generated chains
  avg_prev = mean(prev,2);  % calculates the average of the prevalences
  sd_prev = std(prev,[],2);  % calculates the standard deviation of the
prevalence
```

%% -- End Code TriadSequenceCounting -- %%

```
% Length and frequency of consecutive units

  string = sprintf('%d',chain);  % Converts the chain matrix into a
string with no spaces

  t1=textscan(string,'%s','delimiter','0','multipleDelimsAsOne',1);  %
reads the consecutive 1's from the string using 0's as delimiters (to
calculate the segment length of consecutive bromine, the delimiter
should be changed from 0 to 1)
  s = t1{:};  % computes the length of each consecutive segment in the
string
  data = cellfun('length',s);  % assigns the length of each segment
(e.g., 111 = 3, 111111 = 6)

  [number_times segment_length] = hist(data, 1:(chain_length));  %
makes matrices of the segment_lengths and their frequencies in the
string

  Table1 = transpose([segment_length;number_times]);  % converts the
above arrays into one matrix of segment length and frequency

% Calculate the probability that a 1 is from a segment of at least the
crystallizable segment

  j = Table1(:,1);  % segment lengths of consecutive styrene units
  P = [];  % stores the probability that a 1 exists in a segment length
of j consecutive 1 units

  for t = 1:size(j,1)  % for each segment length
      S = Table1(:,1).*Table1(:,2);  % calculates the number of 1’s in
each segment length (segment length*frequency)
      wj = (1-percent_functionalization(vt))*(S/(chain_length*(1-
percent_functionalization(vt))));  % probability that a unit chosen at
random is a 1 and is a member of a sequence of j consecutive units

      if j(t) < l  % if the segment length is less than that of the
defined crystallizable segment (l) then P = 0
          P = [P; 0];  % Probability based on Flory, P. J., Theory of
Crystallization in Copolymers. T. Faraday Soc. 1955, 51 (0), 848-857.
      elseif j(t) >= l
```

```
            P = [P;((j(t)-l+1)/j(t))*wj(t)];  % if the segment length is
  at least that of the defined crystallizable segment (l) then calculates
  probability based on Flory, P. J., Theory of Crystallization in
  Copolymers. T. Faraday Soc. 1955, 51 (0), 848-857.

        end
    end

    sumP = sum(P);  % sums all of the P’s to determine the probability
  that a unit chosen at random is a 1 from a crystallizable segment
    store_P = [store_P;sumP];  % stores the probability that a 1 exists
  in a crystallizable segment for each chain

end

store_avg = [store_avg, avg_prev];  % stores the average of the
  prevalences (rows) for each degree of functionalization (columns)
store_sd = [store_sd, sd_prev];  % stores the standard deviation of the
  prevalences (rows) for each degree of functionalization (columns)
p1 = 100.*(mean(store_P));  % calculates the average of the store_P’s for
  each degree of functionalization
p2 = 100.*(std(store_P));  % calculates the standard deviation of the
  store_P's for each degree of functionalization
Table = [Table; percent_functionalization(vt).*100,p1,p2];  % stores the
  degree of functionalization and the average and standard deviation of
  the probability that a 1 exists in a crystallizable segment

end
```