```
% AmirHosein Sadeghimanesh
% 2022 January
%
% This script is the same as the script
% "Matlab_bistable_autoregulatory_motif_PSS_from_sampling.m" with the only
% difference that we rescale the parameter region to the unit rectangle,
% i.e. [0,1]^2, compute the PSS and then rescale it back. We do this to
% reduce the numerical errors happening in the optimization algorithms of
% YALMIP + SeDuMi.
% Again note that for the PSS of different degrees, you just need to change
% the value of 'd' below. In Figure 7 of the paper we used d = 2, 6, 10.
%
% PART 1
%
n = 2;
syms x [1, n]
B=[0.0, 1.0; 0.0, 1.0];
d = 2;
[p, c] = multPoly(n, x, d);
f = intOverB(p, n, x, B);
%
% PART 2
%
K = [0.000862,1.122156;
    0.000772,1.195306;
    0.000720,1.261000;
    0.000991,1.083385;
    0.000952,1.041393;
    0.000629,1.262206;
    0.000630,1.341161;
    0.000547,1.370650;
    0.000612,1.319884;
    0.000752,1.200114;
    0.000848,1.108466;
    0.000894,1.053006;
    0.000646,1.256657;
    0.000949,1.032261;
    0.000925,1.085609;
    0.000808,1.056900;
    0.000641,1.249726;
    0.000755,1.254697;
    0.000598,1.214788;
    0.000862,1.008877;
    0.000915,1.134443;
    0.000523,1.501984;
    0.000603,1.304372;
    0.000526,1.455036;
    0.000698,1.162353;
    0.000648,1.234009;
    0.000573,1.299606;
    0.000519,1.301665;
    0.000774,1.065382;
    0.000500,1.451561;
    0.000700,1.222773;
    0.000904,1.085566;
```

```
    0.000717,1.204874;
    0.000918,1.118868;
    0.000851,1.039567;
    0.000554,1.262855;
    0.000762,1.077291;
    0.000591,1.398213;
    0.000885,1.059792;
    0.000516,1.500313;
    0.000571,1.305487;
    0.000597,1.293478;
    0.000869,1.111391;
    0.000594,1.380029;
    0.000650,1.264536;
    0.000624,1.306416;
    0.000505,1.506003;
    0.000603,1.309134;
    0.000788,1.127426;
    0.000519,1.436538;
    0.000708,1.239370;
    0.000848,1.173154;
    0.000930,1.103855;
    0.000955,1.113499;
    0.000939,1.110050;
    0.000662,1.297071;
    0.000815,1.187189;
    0.000875,1.157290;
    0.000667,1.186704;
    0.000712,1.125051;
    0.000775,1.081421;
    0.000518,1.481606;
    0.000898,1.108248;
    0.000550,1.434035;
    0.000655,1.266879;
    0.000523,1.379208;
    0.000575,1.267634;
    0.000939,1.054433;
    0.000566,1.420775;
    0.000677,1.294160;
    0.000807,1.151845;
    0.000818,1.201988;
    0.000688,1.261712;
    0.000899,1.017230;
    0.000955,1.106314;
    0.000801,1.046122;
    0.000545,1.454359;
    0.000525,1.384763];
K(:,1) = 2000*K(:, 1)-1;
K(:, 2) = K(:, 2)/2;
a = partSOSFitting(n, x, p, c, f, B, K);
disp(a)
% Saving the coefficient vector of the PSS polynomial in a txt file.
folder = 'C:\Home\PSS\Codes\Bistable_autoregulatory_motif'; % replace this directory↙
to the directory of the folder you are using.
baseFileName = "PSS_via_sampling_degree_" + d + "_output_vector_a.txt";
fullFileName = fullfile(folder, baseFileName);
```

```
a_vec_file = fopen(fullFileName, 'w');
fprintf(a_vec_file, 'The coefficient vector of the polynomial p of the PSS↙
respresentation.\n\n');
fprintf(a_vec_file, 'a: ');
for i = 1:length(a)
    fprintf(a_vec_file, '%f,', a(i));
end
fclose(a_vec_file);
%
% PART 3
%
if n == 2
    fig = figure;
    fig.Units = 'pixels';
    fig.Position(1:2) = [100, 100];
    fig.Position(3:4) = [540, 460];
    % The main plot.
    fcontour(subs(p, c, a), [B(1, 1), B(1, 2), B(2, 1), B(2, 2)], 'LevelList', [0, 1],↙
'Fill', 'on')
    axis([0 1 0 1])
    xticks([0.0 0.2 0.4 0.6 0.8 1.0])
    xticklabels([0.0005 0.0006 0.0007 0.0008 0.0009 0.0010])
    yticks([0.0 0.2 0.4 0.6 0.8 1.0])
    yticklabels([0 0.5 1 1.5 2])
    xlabel('$k_3$', 'interpreter', 'latex', 'FontName', 'Times New Roman', 'FontSize',↙
18)
    ylabel('$k_8$', 'interpreter', 'latex', 'FontName', 'Times New Roman', 'FontSize',↙
18)
    set(get(gca, 'ylabel'), 'rotation', 0)
    ax = gca;
    ax.XRuler.Exponent = 0;
    % colormap to only have three colors, below 0, between 0 and 1, and
    % above 1.
    cMap=[
        0.57, 0.88, 1;
        1, 1, 0];
    colormap(cMap); % generating the colormap for the colorbar.
    hold on
    x_list = zeros(size(K, 1), 1);
    y_list = zeros(size(K, 1), 1);
    for i = 1:size(K, 1)
        x_list(i) = K(i, 1);
        y_list(i) = K(i, 2);
    end
    scatter(x_list, y_list, 40, [1, 0.5, 0], 'filled');
    hold off
else
    disp("The dimension is higher than 2.")
end
%
%%%%%%%%%%%%
% Functions %
%%%%%%%%%%%%
%
% Generating a matrix that each of its rows is an exponent vector of a
```

```
% monomial of degree at most d in n variables.
% The monomials are ordered with respect to the graded lexicographic
% monomial order.
%
function vMtx = allMonomials(n, d)
    % if isinteger(n)==0 || isinteger(d)==0 || n<=0 || d<0
    %     error("The input arguments are not valid. The first argument must be a positive integer and the second argument must be a nonnegative integer.");
    % end
    vMtx = zeros(nchoosek(d+n, n), n);
    for idx = 1:1:nchoosek(d+n, n)-1
        vMtx(idx+1, :) = nxtMonomial(vMtx(idx, :));
    end
end
%
% Generating the next expnent vector of the next monomial in the graded
% lexicographic order.
%
function v = nxtMonomial(v, n)
    % if nargin>1
    %     if isinteger(n)==0 || n<1 || n~=length(v)
    %         error("The second input argument must be a positive integer equal to the length of the first input argument.");
    %     end
    % else
    %     n=length(v);
    % end
    if nargin == 1
        n = length(v);
    end
    if n == 1
        v(1) = v(1)+1;
        return
    end
    idx1 = 0;
    for idx2 = n:-1:1
        if v(idx2) ~= 0
            idx1 = idx2;
            break;
        end
    end
    if idx1 == 0
        v(1) = 1;
        return
    end % so there is a first nonzero index.
    if idx1 ~= n
        v(idx1) = v(idx1)-1;
        v(idx1+1) = v(idx1+1)+1;
        return
    end % here we know idx1=n, so no point in keeping idx1 for saving a fixed known number which is already saved at some variable.
    idx1 = 0;
    for idx2 = n-1:-1:1
        if v(idx2) ~= 0
            idx1 = idx2;
```

```
            break;
        end
    end
    if idx1 == 0
        v(1) = v(n)+1;
        v(n) = 0;
        return
    end % so there is a second nonzero index.
    tmpMem = v(n);
    v(n) = 0;
    v(idx1) = v(idx1)-1;
    v(idx1+1) = v(idx1+1)+tmpMem+1;
end
%
% The following function receives an integer, a symbol and another integer,
% respectively n, x and d. Then returns a polynomial in n variables of
% total degree d with all monomials. It also returns a second output which
% is a vector of coefficients of this polynomial.
%
function [p, c] = multPoly(n, x, d)
    %syms x [1,n] % remove x from the input arguments and ncomment this
    %line if you want the function generate the variables as x1 ... xn
    %itself.
    Mtx = allMonomials(n, d);
    c = str2sym(arrayfun(@(idx) "c" + join("_" + Mtx(idx,:), ""), 1:size(Mtx, 1)));
    p = sum(arrayfun(@(idx) c(idx).*prod(x.^Mtx(idx,:)), 1:size(Mtx, 1)));
end
%
% The following function receives a polynomial, an integer, a symbol and a
% hyperrectangle, respectively called p, n, x and B. Then it returns the
% n-dimensional integral of p in x over B.
%
function f = intOverB(p, n, x, B)
    f = p;
    for idx = n:-1:1
        f = int(f, x(idx), B(idx, :));
    end
end
%
% The following function is the SOS + linear optimization formulating the
% PSS polynomial with the help of YALMIP and SeDuMi.
%
function PSSCoeffs = partSOSFitting(n, x, p, c, f, B, K) %#ok<STOUT>
    % The following string should not be produced after the spdvar xi's,
    % otherwise you may get a strange string with x61 or x82 etc.
    tmpStr = "Goal=[sos(p";
    tmpStr = tmpStr + join(arrayfun(@(idx) "-s" + idx + "B*(" + string(x(idx)) + "-("↙
+ B(idx, 1) + "))*((" + B(idx ,2)+ ")-" + string(x(idx)) + ")", 1:n), "") + "),\n";
    tmpStr = tmpStr + join(arrayfun(@(idx) "sos(s" + idx + "B)", 1:n), ",\n") + ",\n";
    tmpStr = tmpStr + join(arrayfun(@(idx) ...
        string(subs(p, x, K(idx, :))) + ">=1", 1:size(K, 1)), ",\n") + "];";
    % back to the expected order of the lines.
    for idx = 1:n
        eval("sdpvar " + string(x(idx)));
    end
```

```
    for idx = 1:length(c)
        eval("sdpvar " + string(c(idx)));
    end
    eval("p=" + string(p));
    eval("F=" + string(f));
    for idx1 = 1:n
        eval("[s" + idx1 + "B,c" + idx1 + "B]=polynomial([" + join(arrayfun(@(idx2)↙
string(x(idx2)), 1:n)) + "],0)");
    end
    eval(sprintf(tmpStr));
    eval("solvesos(Goal, F, [], ["+ ...
        join(arrayfun(@(idx) "c" + idx + "B;", 1:n), "") + ...
        join(arrayfun(@(idx) string(c(idx)), 1:length(c)), ";") + "])");
    eval("PSSCoeffs=[" + join(arrayfun(@(idx) "value(" + string(c(idx)) + ")", 1:↙
length(c)), ",") + "]");
end
%
% End of the file.
```