```
% AmirHosein Sadeghimanesh
% 2020 February, modified 2021 August
%
% This script contains the computations for finding the PSS representation
% of degree 2 of the multistationarity region of the example in Section 5.2
% of the paper, which is depicted in Figure 8i.
%
% This file use the sampling representation found by the script
% "Matlab_section_5_2_sampling_from_rectangular.m".
%
% PART 1
%
% Computing the target function for the optimization problem of PART 2.
%
n = 2;
syms x [1, n]
B = [0.0, 5.0; 0.0, 5.0];
d = 4;
[p, c] = multPoly(n, x, d);
f = intOverB(p, n, x, B);
%
% PART 2
%
% Computing the coefficients of the polynomial p of the PSS
% representation using optimization by YALMIP and SEDUMI having only 1 SOS
% constraints, the rest of constraints are linear inequalities.
%
K = [4.681933,3.420334;
    4.452327,2.730243;
    4.496514,2.874921;
    3.917654,2.765752;
    4.868677,2.589316;
    4.053108,2.567193;
    4.302153,2.516604;
    4.871489,2.745823;
    3.866713,2.884209;
    4.320072,2.627087;
    4.994237,2.915116;
    4.121684,2.577557;
    4.122805,2.557939;
    4.381785,3.451782;
    4.538837,2.612365;
    3.851078,3.471551;
    4.881418,3.167215;
    3.886443,3.532261;
    4.172622,2.867466;
    4.682892,2.512921;
    3.810559,3.334895;
    4.504335,3.157628;
    4.662137,3.384067;
    4.726721,2.859971;
    4.615665,3.195837;
    4.245651,2.576988;
    4.725219,2.921980;
    4.509832,3.426568;
```

```
3.881017,2.659860;
4.436925,3.106537;
4.863095,3.498700;
4.667926,2.564165;
3.841107,2.610659;
4.747939,3.678760;
4.604644,2.665104;
4.653406,2.637942;
3.896866,3.300897;
4.161018,3.317265;
4.686414,3.228982;
4.675040,2.793534;
1.709348,1.856624;
1.791831,1.303897;
1.479023,1.480749;
1.678143,1.623714;
1.743352,1.479783;
1.378767,1.304167;
1.732459,1.378547;
1.492670,1.594862;
1.393096,1.651213;
1.552800,1.344903;
2.363707,1.937879;
2.058791,2.023358;
2.206795,1.932187;
2.128322,1.940529;
1.945177,2.365267;
2.057231,2.252208;
2.477764,2.145303;
2.309220,2.348812;
2.145401,2.284686;
1.943597,2.458600;
2.617163,2.041362;
2.998644,2.179752;
2.980599,2.122504;
2.670587,1.898272;
2.920809,2.143478;
2.782337,2.256161;
2.537127,2.072382;
2.982951,2.310271;
2.578333,1.956345;
2.557720,1.879888;
3.389443,2.284733;
3.576827,2.207006;
3.193011,2.269854;
3.204062,1.958940;
3.186621,1.963767;
3.230157,1.997656;
3.323425,2.072768;
3.260977,2.031901;
3.683077,2.314515;
3.472336,1.990271;
2.632519,2.548342;
3.071125,2.941697;
2.848618,2.695893;
```

```
2.603877,2.889061;
3.117459,2.606520;
2.661120,2.748000;
2.546247,2.927560;
2.751493,3.114272;
2.751365,2.887920;
2.596481,2.738341;
3.225709,2.973820;
3.669444,2.719235;
3.553460,2.683843;
3.456643,3.020265;
3.498431,2.709570;
3.312016,2.782870;
3.389154,2.724754;
3.473949,2.964091;
3.390209,2.768347;
3.203045,2.515271;
3.306366,3.323450;
3.533556,3.723085;
3.709832,3.411179;
3.275299,3.602436;
3.599580,3.587905;
3.589805,3.191200;
3.550975,3.414538;
3.257602,3.186574;
3.639734,3.234381;
3.227231,3.541242;
4.308993,4.072849;
4.189189,3.845994;
4.345911,4.088053;
4.174834,3.772852;
4.255752,4.217887;
3.825117,4.078153;
3.953646,4.091531;
3.999300,4.009433;
3.862961,3.909617;
3.762835,4.327297;
4.783562,4.332883;
4.477195,4.325686;
4.871661,4.110871;
4.650022,3.911009;
4.844966,3.892918;
4.415117,4.229581;
4.794501,4.197008;
4.776288,4.011905;
4.619226,4.260088;
4.573392,4.259087;
4.868171,4.907665;
4.691023,4.772288;
4.969309,4.652478;
4.412512,4.916719;
4.769493,4.596921;
4.998127,4.515107;
4.782782,4.753119;
4.617028,4.463867;
```

```
    4.390709,4.638195;
    4.490063,4.828610];
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\Section_5_2'; % 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
%
% Plotting the PSS approximation of the multistationarity region.
%
if n == 2
    fig = figure;
    fig.Units = 'pixels';
    fig.Position(1:2) = [100, 100];
    fig.Position(3:4) = [460, 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([B(1, 1), B(1, 2), B(2, 1), B(2, 2)])
    xticks(B(1, 1):(B(1, 2)-B(1, 1))/5:B(1, 2))
    yticks(B(2, 1):(B(2, 2)-B(2, 1))/5:B(2, 2))
    xlabel('$k_7$', '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.
```