

```
% AmirHosein Sadeghimanesh
% 2020 February, modified 2021 August
%
% This script contains computation of PSS representation of the
% multistationarity region of the example in Section 5.2 of the paper with
% respect to the rectangular representation in Figure 8f. In paper we used
%  $d = 2, 4$  depicted in Figures 8g and 8h.
%
% 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 = 2;
[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 SOS optimization by YALMIP and SEDUMI.
%
K = [3.75, 2.5, 5.0, 3.75;
     1.25, 1.25, 1.875, 1.875;
     1.875, 1.875, 2.5, 2.5;
     2.5, 1.875, 3.125, 2.5;
     3.125, 1.875, 3.75, 2.5;
     2.5, 2.5, 3.125, 3.125;
     3.125, 2.5, 3.75, 3.125;
     3.125, 3.125, 3.75, 3.75;
     3.75, 3.75, 4.375, 4.375;
     4.375, 3.75, 5.0, 4.375;
     4.375, 4.375, 5.0, 5.0];
a = SOSFitting(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_rectangular_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
representation.\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.
    for idx = 1:size(K, 1)
        hold on
        fill([K(idx, 1), K(idx, 3), K(idx, 3), K(idx, 1), K(idx, 1)], [K(idx, 2), K
(idx, 2), K(idx, 4), K(idx, 4), K(idx, 2)], [1, 0.5, 0]);
        end
        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: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 optimization formulating the
% PSS polynomial with the help of YALMIP and SeDuMi.
%
function PSSCoeffs = SOSFitting(n, x, p, c, f, B, K) %#ok<STOUT>
    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 + "]=polynomial([" + join(arrayfun(@(idx2) ↵
string(x(idx2)), 1:n)) + "],0)");
    end
    for idx1 = 1:size(K, 1)
        for idx2 = 1:n
            eval("[s" + idx2 + "K" + idx1 + ",c" + idx2 + "K" + idx1 + "]=polynomial↵
([" + join(arrayfun(@(idx3) string(x(idx3)), 1:n)) + "],0)");
        end
    end
    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(@(idx1) ...
        "sos(p-1" + join(arrayfun(@(idx2) ...
            "-s" + idx2 + "K" + idx1 + "*(" + string(x(idx2)) + "-(" + K(idx1, idx2) + ")")↵
*("(" + K(idx1, n+idx2) + ")-" + string(x(idx2)) + ")", 1:n), "") + "],\n" + ...
        join(arrayfun(@(idx2) "sos(s" + idx2 + "K" + idx1 + ")", 1:n), ",\n"), 1:size↵
(K, 1)), ",\n") + "]"";

```

```
eval(sprintf(tmpStr));
eval("solvesos(Goal, F, [], [" + ...
    join(arrayfun(@(idx1) "c" + idx1 + "B;", 1:n), "") + ...
    join(arrayfun(@(idx1) ...
        join(arrayfun(@(idx2) "c" + idx2 + "K" + idx1 + ";", 1:n), ""), 1:size(K, 1)),
    "")) + ...
    join(arrayfun(@(idx1) string(c(idx1)), 1:length(c)), ";") + "]"");
eval("PSSCoeffs=[" + join(arrayfun(@(idx) "value(" + string(c(idx)) + ")", 1:
length(c)), ",") + "]"");
end
%
% End of the file.
```