Experiments on polynomial (chaos) approximation of maximum eigenvalue functions: Tutorial

Luca Fenzi[1] *& Wim Michiels* [2]

Description. *This tutorial describes the numerical experiments reported in the article,* Fenzi & Michiels (2018) "Polynomial (chaos) approximation of maximum eigenvalue functions: efficiency and limitations", *providing a template that can be modified for explorations of your own.*

The tutorial explores the polynomial approximation of smooth, non-differentiable and not even Lipschitz continuous benchmark functions in the univariate and bivariate cases. The analyzed functions arise from parameter eigenvalue problems; more in details, they are the real part of the rightmost eigenvalue (the so-called spectral abscissa).

The polynomial approximations are obtained by Galerkin and collocation approaches. In the Galerkin approach, the numerical approximation of the coefficients in the univariate case is achieved by extended (or composite) Trapezoidal and Simpson's rules or by Gauss and Clenshaw-Curtis quadrature rules. For the bivariate case, the coefficients are approximated by tensorial and non-tensorial Clenshaw-Curtis cubature rules, based on tensor-product Chebyshev grid and Padua points, respectively. The collocation approach interpolates the function on Chebyshev points in the univariate case, while for the bivariate case the interpolant nodes are given by tensor-product Chebyshev grid and Padua points.

Contents

[1] luca.fenzi@kuleuven.be, *Department of Computer Science, KU Leuven, Belgium.*
[2] wim.michiels@cs.kuleuven.be, *Department of Computer Science, KU Leuven, Belgium.*

1 Introduction

This tutorial documents the analysis performed in [6], in a similar fashion w.r.t. [10]. Most of the analysis can be carried out through the Chebfun software package. Additional MATLAB functions, used for the analysis of the bivariate case (Section 4), can be found at the end of this tutorial, in the Appendix A.

Everything here reported is fast to compute, in order that you can use it as a template to be modified for explorations of your own. The results of the simulations used in [6] are obtained with similar codes. The present tutorial follows the same structure of [6], and we refer to the definitions and ambients of [6] using the SMALL CAPS format style.

In what follows of this introduction, we explain how to produce this text with the desired layout. Moreover, we define acronyms and a function handle to estimate the rates of convergence, which are used in the upcoming sections.

Chebfun download Download Chebfun from the web site www.chebfun.org and install it in your MATLAB path as instructed there.

PCA_SA download Request the MATLAB script `PCA_SA` through the authors' emails. Publish this text with `publish('PCA_SA.m','latex')`. (`PCA_SA.tex` will appear in a subdirectory on your computer labeled `html`.)

If you want to use the same layout of [10], then run the Chebfun script `ATAPformats` before publishing this tutorial. For the layout of the figures, we set MATLAB to use LaTeX to render all text of the images.

```
set(0, 'DefaultTextInterpreter', 'LaTeX','DefaultAxesFontName', ...
    'LaTeX',  'DefaultAxesTickLabelInterpreter', 'LaTeX', ...
    'DefaultLegendInterpreter', 'LaTeX');
```

In addition, we define the following acronyms for the MATLAB plotting options (cf. *e.g.* `help plot` in MATLAB) and for the more common $x-$ and $y-$ labels of the present text.

```
LW='linewidth'; C='Color'; MS='MarkerSize';
P1='$P+1$'; M1='$M+1$'; C0='$|c_0-\tilde{c}_0^M|/|c_0|$';
AP='$\|\alpha-\alpha_P\|_\infty$'; P2='$P_d+1$';
TD='Total Degree'; MD='Maximal Degree';
```

The taxonomy of the spectral abscissa function is defined by the following terminology and highlighted by the usage of the corresponding colors:

```
cases={'SAE    ','MSSAEs ', 'MNSSAEs'};
col={[0,109,219]/255, [0,73,73]/255,[146,0,0]/255};
```

We define a function handle to estimate the convergence rate of the approximations $\{f_n\}_{n\in\mathbf{N}}$ w.r.t. f. If $\{f_n\}_{n\in\mathbf{N}}$ has an algebraic index of convergence r,

i.e. $f_n \sim \mathcal{O}(n^{-r})$ (cf. *e.g.* Section 2.3 in [3]), then r can be estimated by

$$r = \frac{\log(\|f - f_{n_1}\|_\infty) - \log(\|f - f_{n_2}\|_\infty)}{\log(n_2) - \log(n_1)}, \quad n_1 > n_2.$$

Hence, the function handle is defined by the errors `err`, which corresponds to $\|f - f_n\|_\infty$, and the vectors `nn`, with more than 2 ordered numbers $n \in \mathbf{N}$, where we want to estimate the convergence rates.

```
rates=@(err,nn) (log(err(nn(2:end)))-log(err(nn(1:end-1))))./...
    (log(nn(1:end-1))-log(nn(2:end)));
```

This function handle permits to empirically calculate the algebraic index of convergence r for both numerical and approximation errors.

The present version of this tutorial is produced by Chebfun v.5.7.0 and MATLAB R2016b. Ask the authors the script `PCA_SA.m`*, which produces this text through the MATLAB command* `publish`*.*

2 Parameter eigenvalue problem: Example 1

Consider Example 1 and the associated Figure 1, where the spectra of the following matrices are analyzed for $x \in [-1, 1]$.

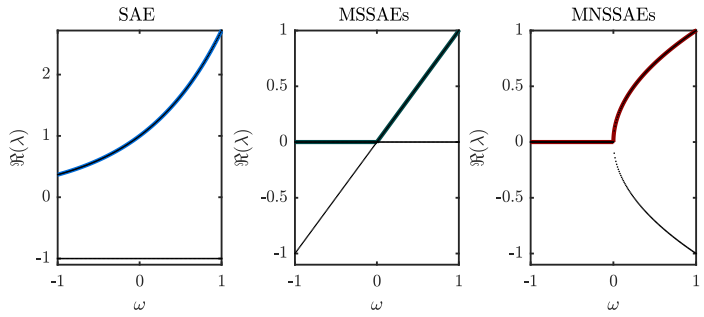
```
A{1}=@(x) [exp(x),0;0,-1];
A{2}=@(x) [x,0;0,0];
A{3}=@(x) [0,x;1,0];
```

The corresponding spectral abscissa functions are

```
x=chebfun('x');
alpha{1}=exp(x);          % SAE
alpha{2}=x*(x>0);         % MSSAE
alpha{3}=sqrt(x)*(x>0);   % MNSSAE
```

To visually see that the spectral abscissa functions, previously defined, are the real part of rightmost eigenvalues of the matrices, we plot the real part of the spectra and the associated spectral abscissa.

```
xx=-1:0.01:1; lambda=zeros(2,length(xx));
for k=1:3
    for i=1:length(xx)
        lambda(:,i)=real(eig(A{k}(xx(i))));
    end, subplot(1,3,k)
    plot(alpha{k},C,col{k},LW, 2);   hold on,
    plot(xx, lambda, '.k',MS, 0.75); hold off,
    title(cases{k}); ylim([-1.1, max(alpha{k})])
    xlabel('$\omega$'); ylabel('$\Re(\lambda)$');
end
```



3 Univariate polynomial approximation ($D = 1$)

We analyze the polynomial approximation up to order P of the spectral abscissa functions `alpha` with Galerkin and collocation approaches:

$$\alpha \approx \alpha_P(\omega) = \sum_{i=0}^{P} \tilde{c}_i p_i(\omega), \quad P + 1 = 100.$$

```
P=99;
```

3.1 Galerkin approach

Legendre polynomials are set as polynomial basis $\{p_i\}_{i=0}^{P}$. `L{i+1}` defines the ith Legendre polynomial, which is implemented in Chebfun by the command `legpoly(i)`.

```
L=cell(1,P); for i=0:P, L{i+1}=legpoly(i); end
```

The ρ-norms of the orthogonal basis, with $\rho(\omega) = 1/2$, are analytically known for the Legendre polynomial (cf. *e.g.* formula 22.2.10 in [1]). We consider the square of these norms, *i.e.* `Pi(i)`$= \|p_i\|_\rho^2$.

```
Pi=@(i) 1./(2*i+1);
```

The coefficients, `c(k,i+1)`$= c_i$ of $\alpha_P(\omega)$, and the ρ-inner products, `a(k,i+1)`$=$ $\langle$`alpha{k}`$, $`L{i + 1}`$\rangle_\rho$, are analytically evaluated by the formula given in Appendix A of [6], for `alpha{k}` with `k=1,2,3`.

```
a=zeros(3,P+1); c=zeros(3,P+1);
```

First of all, we compute the coefficients of the ρ-inner product for the SAE:

```
a(1,1)=(exp(1)-exp(-1))/2;c(1,1)=a(1,1)/Pi(0);
for j=2:P+1
   a(1,j)=(exp(1)+(-1)^j*exp(-1)-2*sum(c(1,j-1:-2:1)))/2;
   c(1,j)=a(1,j)/Pi(j-1);
end
```

Hence, we consider the MSSAEs and MNSSAEs cases:

```
for i=1:P+1
 if mod(i-1,2)==0,  j=(i-1)/2;
  a(2,i)=((-1)^j)*gamma(j-0.5)/(4*gamma(-0.5)*gamma(j+2));
  a(3,i)=((-1)^j)*gamma(j-1/4)*gamma(3/4)/...
         (4*gamma(-1/4)*gamma(j+7/4));
 else,               j=(i-2)/2;
  a(2,i)=1/6*(i==2);
  a(3,i)=((-1)^j)*gamma(j+1/4)*gamma(5/4)/...
         (4*gamma(1/4)*gamma(j+9/4));
 end
end
c(2,:)=a(2,:)./Pi(0:P); c(3,:)=a(3,:)./Pi(0:P); close;
```

MATLAB is not a symbolic software and the operations to compute `c` and `a` can be effected by computational errors. The SAE case, `alpha{1}`, is particularly effected by these errors, since the coefficients are evaluated by recursion (*i.e.* the computational error at iteration i is amplified in the following iterations $j > i$). To avoid the amplification of computational errors due to the recursion, we observe

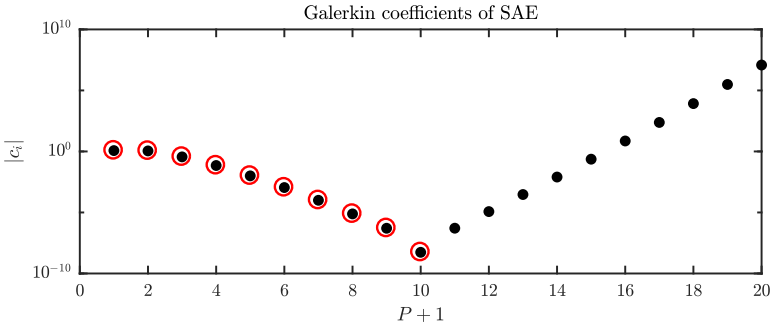
$$\|\alpha\|_\rho^2 = \sum_{i=0}^{\infty} c_i^2 \|p_i\|_\rho^2.$$

Hence, an upper bound on the coefficient c_j can be derived:

$$\|\alpha\|_\rho^2 - \sum_{i=0}^{j-1} c_i^2 \|p_i\|_\rho^2 = \sum_{i=j}^{\infty} c_i^2 \|p_i\|_\rho^2 \geq c_j^2 \|p_j\|_\rho^2,$$

and applied to the SAE case, where $\|\texttt{alpha}\{1\}\|_\rho^2 = (e^2 - e^{-2})/4$.

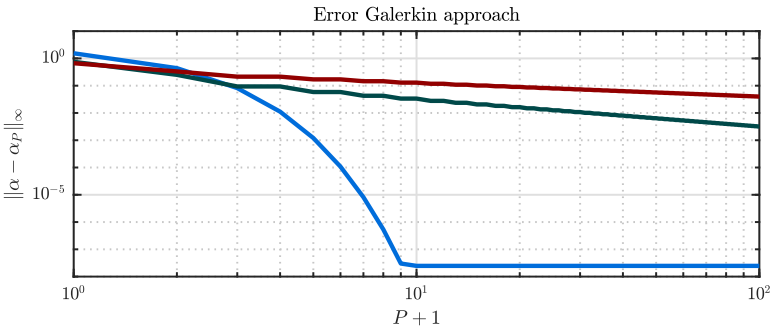
```
semilogy(1:20, abs(c(1,1:20)),'.k'); A12=(exp(2)-exp(-2))/4;
for j=2:P+1
    if Pi(j-1)*c(1,j)^2>A12-sum((c(1,1:j-1).^2).*Pi(0:j-2))
        c(1,j:end)=0;a(1,j:end)=0;break
    end
end, hold on
semilogy(1:j-1,abs(c(1,1:j-1)),'or',MS,8); hold off,xlabel(P1);
ylabel('$|c_i|$'); title('Galerkin coefficients of SAE');
```



3.1.1 Approximation error

The error of truncating the polynomial series up to order P (*i.e.* the approximation error) is considered, assuming that the coefficients (computed in Section 3.1) are not affected by any error. First the polynomial approximation `alphaP_G` is constructed, and then the error in ∞-norm, `error`, is considered. Through Chebfun, the L^∞ error for the SAE and MSSAEs cases is evaluated up to machine precision. For the MNSSAEs, the error can be correctly computed only in the interval $[-1, 0)$; for the interval $[0, 1]$ the ∞-norm of the error is approximated by the maximum error on 10^3 equidistant point in $[0, 1]$.

```
alphaP_G=cell(3,P+1);error=zeros(3,P+1); xxINF=linspace(0,1,1e3);
for k=1:3
 for i=1:P+1
  if i==1, alphaP_G{k,1}=c(k,1)*L{i};
  else, alphaP_G{k,i}=c(k,i)*L{i}+alphaP_G{k,i-1}; end
  if k<3, error(k,i)=norm(alpha{k}-alphaP_G{k,i},inf); else
   error(k,i)=max([norm(alphaP_G{k,i}*(x<=0),inf),...
     abs(alpha{3}(xxINF)-alphaP_G{k,i}(xxINF))]);
  end
 end,loglog(1:P+1,error(k,:),C,col{k},LW, 2); hold on
end, hold off, grid on, ylim([1e-8,10]);
xlabel(P1); ylabel(AP); title('Error Galerkin approach');
```



The convergence error plot is analogous to FIGURE 2. The relative errors $\|\alpha - \alpha_P\|_\infty / \|\alpha\|_\infty$ can be obtained by the previous analysis dividing the error of

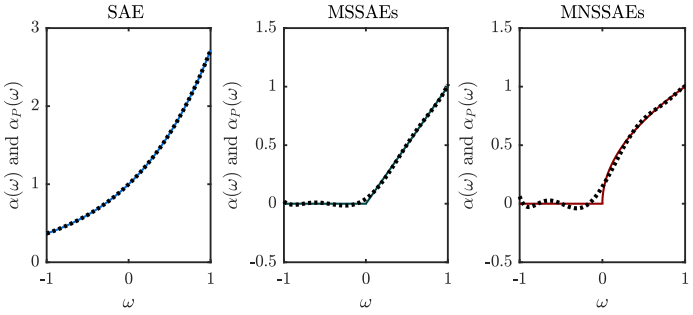
the SAE case by $\|\alpha\|_\infty = e$, *i.e.* `error(1,:)/exp(1)`. The convergence rates $\mathcal{O}(P^{-r})$ for MSSAEs and MNSSAEs cases are estimated by the median of the rates obtained by the previously defined function handle `rates`.

```
for k=2:3
    o=rates(error(k,:),2:2:P);
    fprintf([cases{k},' %4.8f\n'], median(o));
end
```

```
MSSAEs  1.01041762
MNSSAEs 0.50510554
```

The polynomial approximation α_P with P =`Pmax` obtained by Galerkin approach, can be compared w.r.t. the original spectral abscissa functions α.

```
Pmax=8;
for k=1:3, subplot(1,3,k),
    plot(alpha{k},C,col{k},LW, 1); hold on
    plot(alphaP_G{k,Pmax},':k',LW, 2); hold off,
    xlabel('$\omega$'); title(cases{k});
    ylabel('$\alpha(\omega)$ and $\alpha_P(\omega)$');
end
```



3.1.2 Numerical error

In this section, classical integration methods and interpolatory quadrature rules approximate the coefficients

$$c_i = \frac{1}{2\|p_i\|_\rho^2} \int_{-1}^{1} \alpha(\omega) p_i(\omega) \mathrm{d}\omega.$$

The computation of $\|p_i\|_\rho^2$ can be omitted in this analysis, since $\|p_i\|_\rho^2$ cancels out in the evaluation of the relative errors. W.l.g. only the first coefficient c_0 is analyzed, *i.e*

```
i=1;
```

The classical integration methods based on equally spaced points, here considered, are:

Extended Trapezoidal rule formula 25.4.2 in [1]

$$\int_{x_0}^{x_M} f(x)\mathrm{d}x = h\left(\frac{f_0}{2} + \sum_{i=1}^{M-1} f_i + \frac{f_M}{2}\right) - \frac{Mh^3}{12} f''(\xi),\ \xi \in (x_0, x_M).$$

Extended Simpson's rule formula 25.4.6 in [1]

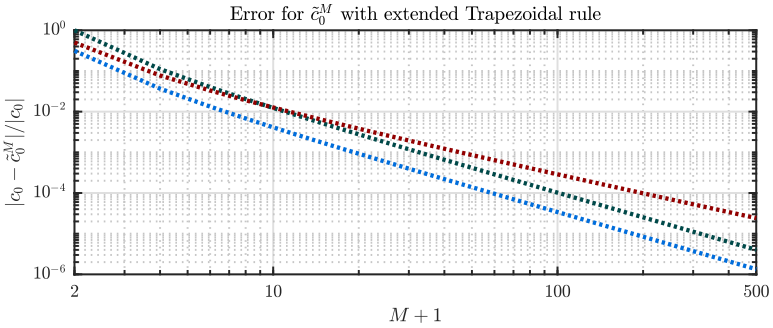
$$\int_{x_0}^{x_{2n}} f(x)\mathrm{d}x = \frac{h}{3}\left(f_0 + \sum_{i=1}^{n-1} 2f_{2i} + 4f_{2i-1} + f_{2n}\right) - \frac{nh^5}{90} f^{(4)}(\xi),\ \xi \in (x_0, x_{2n}).$$

Compute the numerical error introduced by extended Trapezoidal rule, `et`, using at most `M`+1 equally spaced points.

```
M=500; et=NaN(3,M);
for k=1:3
  for m=1:M, h=2/m; xx=-1:h:1; % m+1 points
    at=trapz(xx,alpha{k}(xx).*L{i}(xx))/2; et(k,m)=abs(at-a(k,i));
  end
end, close;
```

The slowest convergence rate of the error is, hence, achieved by

```
for k=1:3
 loglog((1:2:M)+1,et(k,1:2:M)/abs(a(k,i)),':',C,col{k},LW, 2);
 hold on
end, hold off, grid on, ylim([1e-6,1]);xlabel(M1); ylabel(C0)
title('Error for $\tilde{c}_0^M$ with extended Trapezoidal rule')
 set(gca, 'XLim', [2 M], 'XTick', [2,10, 100 M]);
```

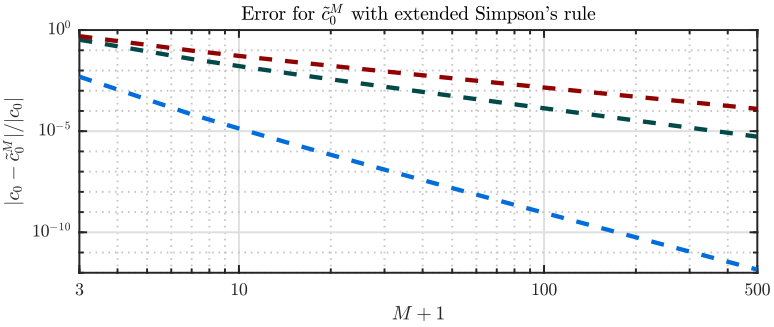


Hence we consider the numerical error due to the approximation of extended Simpson's rule, `es`.

```
es=NaN(3,floor((M+1)/2));  % Simpson's error
for k=1:3
 for m=2:2:M,  h=2/m; xx=-1:h:1; f=alpha{k}*L{i};
  as=h/6*(f(xx(1))+2*sum(f(xx(3:2:m-1)))+4*sum(f(xx(2:2:m)))+...
      f(xx(m+1)));  es(k,m/2)=abs(as-a(k,i));
 end
end
```

The slowest convergence rates are, hence, achieved by

```
for k=1:3
 loglog((2:4:M)+1,es(k,1:2:M/2)/abs(a(k,i)),'--',C,col{k},LW, 2);
 hold on
end, hold off,grid on, ylim([1e-12,1]); xlabel(M1); ylabel(C0)
title('Error for $\tilde{c}_0^M$ with extended Simpson''s rule')
 set(gca, 'XLim', [3 M], 'XTick', [3,10, 100 M]);
```



The last two convergence error plots furnish FIGURE 3. The corresponding orders of convergence r for these two classical integration methods, $\mathcal{O}(M^{-r})$, are given by:

```
disp('       Trapezoidal rule       Simpson''s rule');
for k=1:3
 ot=rates(et(k,:),1:2:M); os=rates(es(k,:),(2:4:M)/2);
 fprintf([cases{k},'   %10.8f %20.8f\n'], median(ot), median(os));
end
```

```
       Trapezoidal rule       Simpson's rule
SAE          1.99999786           3.99990845
MSSAEs       2.00000000           2.00000000
MNSSAEs      1.52883825           1.49999991
```

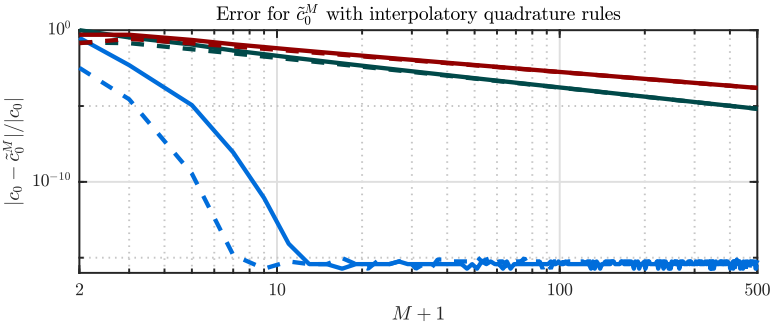
The interpolatory quadrature rules, here considered, are the Clenshaw-Curtis quadrature rule and the Gauss quadrature rule, based on the Chebyshev and Legendre points, respectively. The weights and the nodes of these quadrature rules can be computed by the Chebfun functions `chebpts` and `legpts`. In

addition, Clenshaw-Curtis quadrature rule can be handled by the command `sum` in Chebfun, which employ Fast Fourier Transform; this latter method is simple and faster when few integrands are involved [10], as in our case. Compute, hence, the numerical errors of Clenshaw-Curtis `ec` and Gauss Legendre `el` quadrature rules.

```
ec=NaN(3,M);   el=NaN(3,M);
for k=1:3
   for m=1:M
     ac=sum(chebfun(alpha{k}*L{i},m+1))/2;
     [n,w]=legpts(m+1);  al=w*(alpha{k}(n).*L{i}(n))/2;
     ec(k,m)=abs(ac-a(k,i));  el(k,m)=abs(al-a(k,i));
   end
end
```

FIGURE 4 considers only the slowest convergence rates, as the following code.

```
m_slow=[1,2:2:M];
for k=1:3
 loglog(m_slow+1,el(k,m_slow)/abs(a(k,i)),'--',C,col{k},LW, 2);
 hold on
 loglog(m_slow+1,ec(k,m_slow)/abs(a(k,i)),'-',C,col{k},LW, 2);
end, hold off,grid on, ylim([1e-16,1]); xlabel(M1); ylabel(C0);
title('Error for $\tilde{c}_0^M$ with interpolatory quadrature rules')
set(gca, 'XLim', [2 M], 'XTick', [2,10, 100 M]);
```



The convergence rates are analogous, and the SAE case convergences faster than $\mathcal{O}(M^{-r})$ for given natural numbers r. However, the interpolatory quadrature rules do not improve the orders of convergence for the non-smooth behaviors of the spectral abscissa functions.

```
disp('       Clenshaw-Curtis      Gauss-Legendre');
for k=2:3
 oc=rates(ec(k,:),m_slow);ol=rates(el(k,:),m_slow);
 fprintf([cases{k},'   %1.8f %20.8f\n'], median(oc), median(ol));
end
```

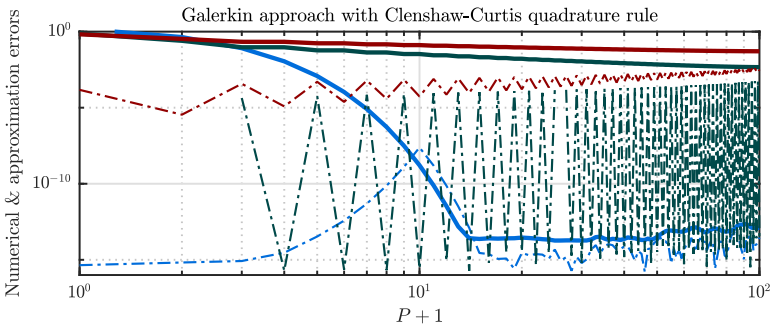
```
       Clenshaw-Curtis      Gauss-Legendre
MSSAEs    1.99998986           1.98808408
MNSSAEs   1.49999248           1.49105496
```

We compute the Galerkin polynomial approximation `alphaPM_G`, whose coefficients are approximated by Clenshaw-Curtis quadrature rules with `M`$+1 = 110$ points. Other than indicating the approximation errors of the polynomial approximation `err_PMa`, we consider also the numerical error `err_PMc`, *i.e.* $|c_i - \tilde{c}_i^M|$, due to the approximation of the coefficient $\tilde{c}_i^M$.

```
err_PMc=NaN(3,P+1); err_PMa=NaN(3,P+1); M=109;
for k=1:3, alphaPM_G=[];
 for p=1:P+1
  ac=sum(chebfun(alpha{k}*L{p},M+1))/(2*Pi(p-1));
  err_PMc(k,p)=abs(ac-c(k,p));
   if p==1, alphaPM_G=ac*L{p};
   else, alphaPM_G=ac*L{p}+alphaPM_G;
   end
   if k<3, err_PMa(k,p)=norm(alpha{k}-alphaPM_G,inf); else
     err_PMa(3,p)=max([norm(alphaPM_G*(x<=0),inf),...
           abs(alpha{3}(xxINF)-alphaPM_G(xxINF))]);
   end
 end
end
```

The numerical and approximation errors for Galerkin approach with Clenshaw-Curtis quadrature rule are presented.

```
for k=1:3
   loglog(1:P+1,err_PMc(k,:),'-.',C,col{k},LW, 1);  hold on
   loglog(1:P+1,err_PMa(k,:),'-',C,col{k},LW, 2);
end, hold off, grid on, ylim([1e-16,1]);
xlabel(P1);ylabel('Numerical \& approximation errors');
title('Galerkin approach with Clenshaw-Curtis quadrature rule')
```



Since the numerical errors do not dominate the approximation errors, the order of convergence are analogous to the ones previously obtained. In particular, for the non-smooth cases we have

```
for k=2:3
    o=rates(err_PMa(k,:),1:P-20);
    fprintf([cases{k},' %4.8f\n'], median(o));
end
```

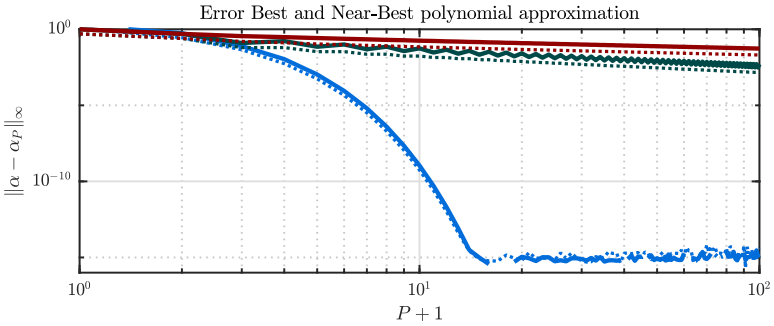
```
MSSAEs  1.08993006
MNSSAEs 0.47745746
```

Modifying this section, you can test the advice given in SECTION 4.1.3 of [6] for $M < 99$. Furthermore, you can test the convergence rates of the Galerkin polynomial approximations whose coefficients are given, for example, by extended Trapezoidal or Simpson's rules or by Gauss Legendre quadrature rule.

3.2 Collocation approach

The near-best polynomial approximation (*i.e.* the interpolant on Chebyshev points) and the best polynomial approximation in L^∞ sense can be easily evaluated by Chebfun with the functions `chebfun` and `minimax`, respectively. The first input of these MATLAB commands is the function that we want to approximate, while the second input is the number of interpolant point for `chebfun` and the degree of the best polynomial approximation for `minimax`.

```
alphaP_b=cell(3,P+1);  err_b=zeros(3,P+1);   % Best
alphaP_nb=cell(3,P+1); err_nb=zeros(3,P+1);  % Near-Best
for k=1:3
 for i=1:P+1
  alphaP_b{k,i}=minimax(alpha{k},i-1);
  alphaP_nb{k,i}=chebfun(alpha{k},i);
    if k<3, err_b(k,i)=norm(alpha{k}-alphaP_b{k,i},inf);
      err_nb(k,i)=norm(alpha{k}-alphaP_nb{k,i},inf);
    else
      err_b(k,i)=max([norm(alphaP_b{k,i}*(x<=0),inf),...
         abs(alpha{3}(xxINF)-alphaP_b{k,i}(xxINF))]);
      err_nb(k,i)=max([norm(alphaP_nb{k,i}*(x<=0),inf),...
         abs(alpha{3}(xxINF)-alphaP_nb{k,i}(xxINF))]);
    end
 end
 loglog(1:P+1,err_b(k,:),':',C,col{k},LW, 1.5); hold on
 loglog(1:P+1,err_nb(k,:),C,col{k},LW, 2);
end, hold off, xlabel(P1);ylabel(AP);ylim([1e-16,1]),grid on
title('Error Best and Near-Best polynomial approximation');
```



The previous image corresponds to FIGURE 5. The convergence rates are similar, and comparable with the results obtained by the Galerkin approach. Indeed, for the non-smooth cases, the empirical convergence rates are estimated by:

```
disp('          Near-Best        Best');
for k=2:3
 ob=rates(err_b(k,:),2:2:P); onb=rates(err_nb(k,:),2:2:P);
 fprintf([cases{k},'  %10.8f %14.8f \n'], median(ob), median(onb));
end
```

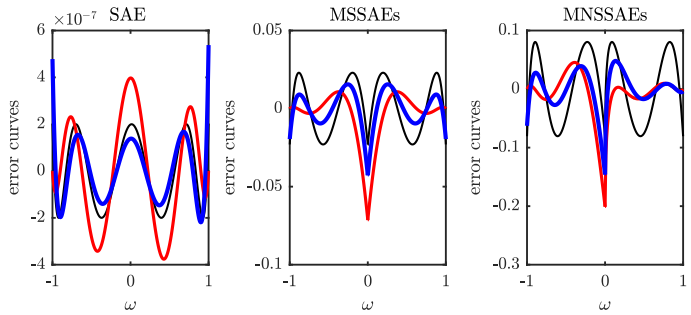
```
          Near-Best        Best
MSSAEs   1.04112886     1.02042229
MNSSAEs  0.51793510     0.51013556
```

3.3 Extra: analysis on the error function and on the mean

In this section, we exploit side aspects of the analysis conducted in the numerical experiments of [6].

First of all, we compare the error curves of Galerkin approach w.r.t. best and near-best approximations, such that the degree of the polynomial approximation is $P =$`Pmax`. The MATLAB warnings are disabled, since the error curves, computed by Chebfun, are not accurate up to machine precision.

```
for k=1:3, warning off; subplot(1,3,k);
    plot(alpha{k}-alphaP_b{k,Pmax},'-k',LW,1); hold on
    plot(alpha{k}-alphaP_nb{k,Pmax},'-r',LW,1.5);
    plot(alpha{k}-alphaP_G{k,Pmax},'-b',LW,2);
    xlabel('$\omega$'); ylabel('error curves'); title(cases{k});
end, warning on
```



Hence, we consider the corresponding errors in the 2-norm and in the ∞-norm. Only the SAE and MSSAEs cases are considered since they can easily computed by Chebfun.

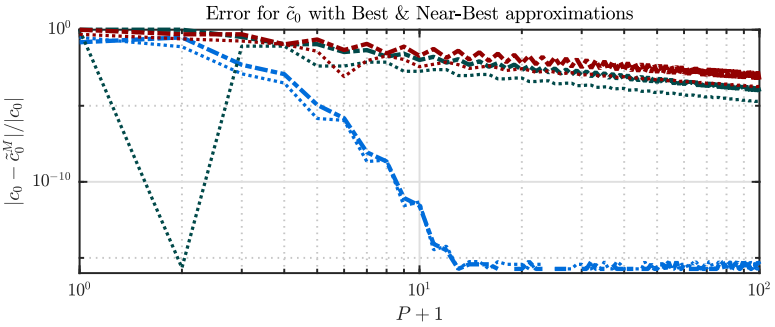
```
for k=1:2, b=alpha{k}-alphaP_b{k,Pmax};
    nb=alpha{k}-alphaP_nb{k,Pmax}; g=alpha{k}-alphaP_G{k,Pmax};
 fprintf([' *',cases{k},'  Inf-Norm         2-Norm\n']);
 fprintf('Best      %10.8e   %14.8e\n', norm(b,inf), norm(b,2));
 fprintf('Near-Best %10.8e   %14.8e\n', norm(nb,inf), norm(nb,2));
 fprintf('Galerkin  %10.8e   %14.8e\n', norm(g,inf), norm(g,2));
end
```

```
 *SAE      Inf-Norm         2-Norm
Best      1.99825278e-07   1.99433344e-07
Near-Best 3.97374478e-07   3.25491066e-07
Galerkin  5.37898493e-07   1.73986493e-07
 *MSSAEs   Inf-Norm         2-Norm
Best      2.29645310e-02   2.27521199e-02
Near-Best 7.14285714e-02   2.49210895e-02
Galerkin  4.27246094e-02   1.59471988e-02
```

Then, we compare the mean of the PC expansion associated to the best and near-best polynomial approximation. Indeed, the convergence rates of the numerical errors in Section 3.1.2 can be interpreted as the convergence rates of the mean, evaluated by the corresponding integration method. To this end, we transform the coefficients of the best and near-best polynomial approximations from the Chebyshev to the Legendre bases.

```
close; eb=zeros(3,P+1);  enb=zeros(3,P+1);
for k=1:3, for p=1:P+1
  cl=cheb2leg(chebcoeffs(alphaP_b{k,p}));
  eb(k,p)=abs(cl(1)-c(k,1))/abs(c(k,1));
  cl=cheb2leg(chebcoeffs(alphaP_nb{k,p}));
  enb(k,p)=abs(cl(1)-c(k,1))/abs(c(k,1));
           end,
    loglog(1:P+1,eb(k,:),':',C,col{k},LW, 1.5); hold on
```

```
    loglog(1:P+1,enb(k,:),'-.',C,col{k},LW, 2);
end, hold off, grid on, xlabel(P1); ylabel(C0);ylim([1e-16,1]);
title('Error for $\tilde{c}_0$ with Best \& Near-Best approximations')
```

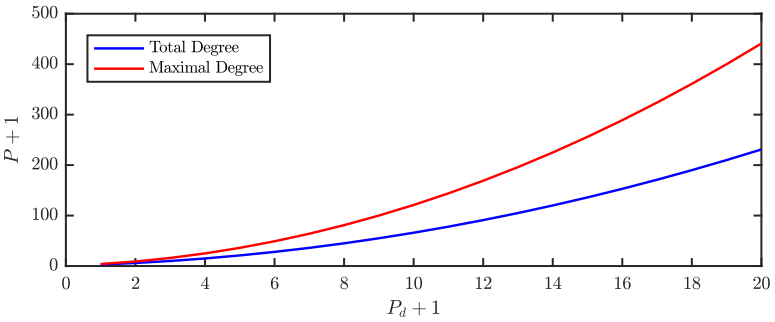


4 Bivariate polynomial approximation ($D = 2$)

In this section we consider the polynomial approximation of the spectral abscissa of parameter varying eigenvalue problems with $D = 2$.

The polynomial approximation is truncated up to degree P_d =`Pdmax`. The number of coefficients P grows exponentially w.r.t. the multivariate (total and maximal) degree P_d of the polynomial approximation.

```
Pdmax=19; pp=1:Pdmax+1;
plot(pp,(pp+1).*(pp+2)/2,'b',pp,(pp+1).^2,'r');
xlabel(P2); ylabel(P1); legend(TD,MD,'Location','northwest');
```



Set $P_d \in \mathbf{N}$, the maximal degree polynomial basis $\{p_i^{[m]}\}_{i=0}^P$ contains the total degree basis $\{p_i^{[t]}\}_{i=0}^P$, *i.e.* $\{p_i^{[t]}\}_{i=0}^P \subseteq \{p_i^{[m]}\}_{i=0}^P$. These bases are constructed by the inverse of an associated pairing function. The inverse of the Rosenberg-Strong pairing function, associated to the maximal degree polynomial basis, is first considered.

```
i1=zeros(1,(Pdmax+1)^2); i2=i1;
for i=0:length(i1)-1, t=floor(sqrt(i));
```

```
    if t>i-t^2, i1(i+1)=i-t^2; i2(i+1)=t;
    else,       i1(i+1)=t;     i2(i+1)=t^2+2*t-i;
    end
end
```

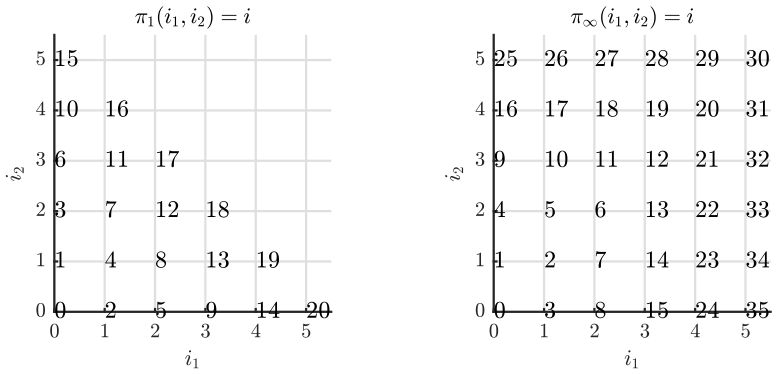
The inverse of the Cantor pairing function is associated to the total degree polynomial basis. However, we mainly consider the maximal degree polynomial basis, since $\{p_i^{[t]}\}_{i=0}^{P} \subseteq \{p_i^{[m]}\}_{i=0}^{P}$, and we map the ordering associated to the Cantor pairing to the one obtained by Rosenberg-strong pairing function, *i.e.*

$$\texttt{c2r}(i) = j, \quad \textit{where } \pi_1(i) = \pi_\infty(j).$$

```
i1c=zeros(1,nchoosek(Pdmax+2,2)); i2c=i1c; c2r=NaN(size(i1));
for i=0:length(i1c)-1
    w=floor((sqrt(8*i+1)-1)/2); t=(w^2+w)/2;
    i1c(i+1)=i-t;  i2c(i+1)=w-i1c(i+1);
    for j=0:length(i1)-1
        if i2c(i+1)==i2(j+1) && i1c(i+1)==i1(j+1)
            c2r(i+1)=j+1; break
        end
    end
end
```

To test the previous pairing functions, we construct a figure similar to FIGURE 6, which associates the numbers $\mathbf{N}\times\mathbf{N}$ to the corresponding value of the pairing functions. The construction follows the so-called *shell* of the pairing function [8], which is associated to the polynomial basis with degree equal to P_d.

```
for pd=0:5
    subplot(1,2,1);  axis square; grid on   % Total
    for j=(pd+1)*pd/2+1:(pd+1)*(pd+2)/2
        text(i1(c2r(j)),i2(c2r(j)),num2str(j-1))
    end,  title('$\pi_1(i_1,i_2)=i$');
    subplot(1,2,2); axis square; grid on    % Maximal
    for j=((pd)^2+1:(pd+1)^2)
        text(i1(j),i2(j),num2str(j-1))
    end,  title('$\pi_\infty(i_1,i_2)=i$')
end
s1=subplot(1,2,1); xlabel('$i_1$'); ylabel('$i_2$');
s2=subplot(1,2,2); xlabel('$i_1$'); ylabel('$i_2$');
set([s1,s2],'XLim',[0 5+.5],'XTick', 0:5,'YLim',[0 5+.5]);
```



Example 2. The delay parameter varying eigenvalue problems of EXAMPLE 2 can be linearized by the Infinitesimal generator approach given $(\omega_1, \omega_2) \in \mathbf{S}$. The MATLAB function `DelayedOscillator` (given in the Appendix A) discretizes the infinite dimensional linear eigenvalue problem associated to the oscillator with feedback delay system into a finite standard eigenvalue problem.

For our porpoise, we consider the discretization `IG`$= 20$ obtained by the method proposed by [2] (*i.e.* with `stru=0`), a similar result can be achieved by the approach of [7], setting `stru=1`. Moreover, we set the number of points `NT`^{2} for the approximation of the ∞-norm, and the linear transformation `Lt` $: [-1,1]^2 \to \mathbf{S}$.

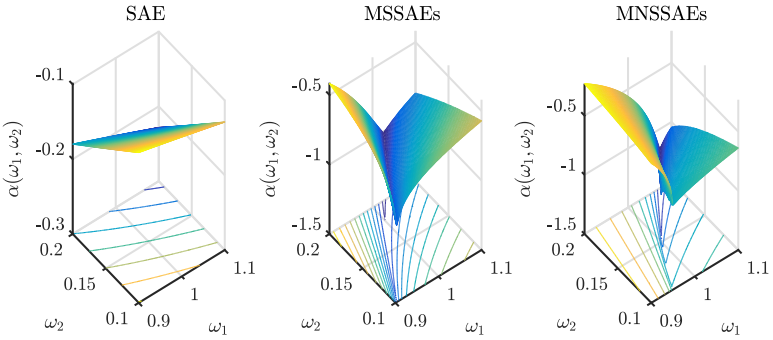
```
IG=20; stru=0; NT=100;
S1=[0.9,1.1]; Lt1=@(o1) (S1(2)-S1(1))*((o1+1)/2)+S1(1);
S2=[0.1,0.2]; Lt2=@(o2) (S2(2)-S2(1))*((o2+1)/2)+S2(1);
```

The approximation of the ∞-norm is achieved by `NT`^{2} evaluations of the spectral abscissa in the domain $\mathbf{S}$.

```
XT=linspace(-1,1,NT); YT=XT; O1=Lt1(XT); O2=Lt2(YT);
[OO1,OO2]=meshgrid(O1,O2); AlphaT=zeros(NT,NT,3);
for k=1:3
 for i=1:NT
  for j=1:NT
   [A,B]=DelayedOscillator(OO1(i,j),OO2(i,j),k,IG,stru);
   AlphaT(i,j,k)=max(real(eig(A,B)));
  end
 end
end, close;
```

The behaviors of the spectral abscissa functions vary w.r.t. the controllers of TABLE 1. The SAE case is a smooth bivariate function on $\mathbf{S}$, MSSAEs and MNSSAEs cases present non-differentiable and non-Lipschitz curves in the domain $\mathbf{S}$, respectively. The non-smooth behavior of MSSAEs is due to the crossing of two active eigenvalues, which do not overlap, while in the MNSSAEs a triple eigenvalue splits, as shown in Figure 3 of [5].

```
for k=1:3,  subplot(1,3,k); meshc(OO1,OO2,AlphaT(:,:,k));
  xlabel('$\omega_1$'); ylabel('$\omega_2$');
  title(cases{k});zlabel('$\alpha(\omega_1,\omega_2)$');
end
```



4.1 Galerkin approach

In this section, we compute the reference values of the coefficients and the corresponding polynomial approximations. Hence we consider the numerical errors and we analyze the advice given in Section 4.1.3 of [6].

4.1.1 Approximation error

The reference values are computed on `mpT` Padua points and `mcT` tensor product Chebyshev grid. The corresponding total number of points are `MpT`+1 and `McT`+1, respectively.

```
mpT=99; MpT=nchoosek(mpT+2,2)-1; mcT=99; McT=((mcT+1)^2)-1;
```

The tensorial Clenshaw-Curtis cubature rule is constructed by tensor product of the Clenshaw-Curtis quadrature rule on `mcT+1` Chebyshev nodes.

```
tensor=@(D1) [repmat(D1,length(D1),1),repelem(D1,length(D1))];
[X,W]=chebpts(mcT+1); WcT=prod(tensor(W'),2); WcT=WcT/sum(WcT);
XYT=tensor(X); XcT=XYT(:,1); YcT=XYT(:,2);
```

The non-tensorial Clenshaw-Curtis cubature rule is based on Padua points, implemented in Chebfun by the function `paduapts`. The weights associated to the Padua points are evaluated by `pdwtsMM` function (cf. [4]), included in Appendix A.

```
[XYT]=paduapts(mpT); XpT=XYT(:,1); YpT=XYT(:,2);
WpT=pdwtsMM(mpT); WpT=WpT/sum(WpT); close
```

We observe that `length(WpT)=MpT+1` and `length(WcT)=McT+1`.

Remark. *Modifying the tensor product formula given by the function handle* `tensor`*, it is possible to construct integration rules for arbitrary dimension* D*.*

We approximate the coefficients $c_i^{M^\star}$ associated to the bases $\{p_j^{[t]}\}_{j=0}^P$ and $\{p_j^{[m]}\}_{j=0}^P$ by the pairing functions π_1 and π_∞. The bases are constructed by univariate Legendre polynomials `L{i+1}`, defined in Section 3.1. We consider only $p_i \in \{p_i^{[m]}\}_{i=0}^P$, since $\{p_i^{[t]}\}_{i=0}^P \subseteq \{p_i^{[m]}\}_{i=0}^P$ can be evaluated through the function `c2r`.

The inner products $\langle \alpha, p_i \rangle_\rho$ are first approximated by non-tensorial Clenshaw-Curtis cubature rule, obtaining `apT(i+1)`.

```
apT=zeros(3,(Pdmax+1)^2);
for k=1:3, sa=zeros(MpT+1,1);
 for i=1: MpT+1
  [A,B]=DelayedOscillator(Lt1(XpT(i)),Lt2(YpT(i)),k,IG,stru);
  sa(i)=max(real(eig(A,B)));
 end
 for i=1:(Pdmax+1)^2
   apT(k,i)=sum(WpT.*sa.*L{i1(i)+1}(XpT).*L{i2(i)+1}(YpT));
 end
end
```

Hence, we approximate the coefficients by tensorial Clenshaw-Curtis cubature rule, obtaining `acT(i+1)`.

```
acT=zeros(3,(Pdmax+1)^2);
for k=1:3, sa=zeros(McT+1,1);
 for i=1: McT+1
  [A,B]=DelayedOscillator(Lt1(XcT(i)),Lt2(YcT(i)),k,IG,stru);
  sa(i)=max(real(eig(A,B)));
 end
 for i=1:(Pdmax+1)^2
  acT(k,i)=sum(WcT.*sa.*L{i1(i)+1}(XcT).*L{i2(i)+1}(YcT));
 end
end
```

We normalize the polynomial basis $\{p_i^{[m]}\}_{i=0}^P$ by $\|p_i\|_\rho^2$.

```
L2=zeros(NT,NT,(Pdmax+1)^2); [XXT,YYT]=meshgrid(XT,YT);
for i=1:(Pdmax+1)^2
 L2(:,:,i)=L{i1(i)+1}(XXT).*L{i2(i)+1}(YYT)/(Pi(i1(i))*Pi(i2(i)));
end
```

Remark. `L2` *is a tensor of dimension* `Nt`$\times$`Nt`$\times$`(Pdmax+1)`^{2}*, it is possible to store it only if* `Nt` *and* `Pdmax+1` *are small.*

At this point, we evaluate the polynomial approximation, obtained by reference values coefficients, and the corresponding error in ∞-norm. We start with maximal degree polynomial approximation, whose coefficients are evaluated by tensorial Clenshaw-Curtis cubature rules.

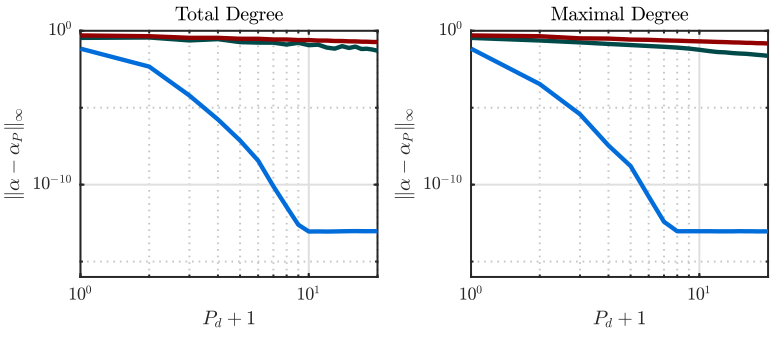
```
err_cT=zeros(3,Pdmax+1);
for k=1:3
    SA=acT(k,1)*ones(size(AlphaT(:,:,k)));
    err_cT(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
    for pd=1:Pdmax
        for j=((pd)^2+1:(pd+1)^2)
            SA=SA+acT(k,j)*L2(:,:,j);
        end
        err_cT(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
    end
end
```

We compute the error of the polynomial approximation with total degree P_d whose coefficients are approximated by non-tensorial Clenshaw-Curtis cubature rule.

```
err_pT=zeros(3,Pdmax+1);
for k=1:3
    SA=apT(k,1)*ones(size(AlphaT(:,:,k)));
    err_pT(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
    for pd=1:Pdmax
        for j=(pd+1)*pd/2+1:(pd+1)*(pd+2)/2
           SA=SA+apT(k,c2r(j))*L2(:,:,c2r(j));
        end
        err_pT(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
    end
end, close
```

The convergences of the errors can be shown by the following code.

```
for k=1:3
 subplot(1,2,1),loglog(pp,err_pT(k,:),C,col{k},LW,2);hold on
 subplot(1,2,2),loglog(pp,err_cT(k,:),C,col{k},LW,2);hold on
end, AX=[1,20,1e-16,1];
subplot(1,2,1),grid on,xlabel(P2);ylabel(AP);title(TD);axis(AX)
subplot(1,2,2),grid on,xlabel(P2);ylabel(AP);title(MD);axis(AX)
```



The non-smooth cases converge as $\mathcal{O}(P_d^{-r})$, where r is estimated by:

```
disp('        Total Degree     Maximal Degree');
for k=2:3
 ot=rates(err_pT(k,:),7:2:Pdmax); om=rates(err_cT(k,:),1:2:Pdmax);
 fprintf([cases{k},'  %10.8f %14.8f \n'], median(ot), median(om));
end
```

```
        Total Degree     Maximal Degree
MSSAEs   1.00450949     1.07162572
MNSSAEs  0.38729260     0.41083053
```

4.1.2 Numerical error

In this section, we analyze the numerical error introduced by non-tensorial and tensorial Clenshaw-Curtis cubature rules in the computation of the coefficient c_j, in the ordering associated to the pairing function π_1 and π_∞. For simplicity we consider `j=0`, since the first coefficient is independent from the multivariate degree, *i.e.* $c_0 = c_0^{[t]} = c_0^{[m]}$.

```
j=0; mm=1:50; Mp=(mm+1).*(mm+2)/2-1; Mc=(mm+1).^2-1;
```

Analogously to the previous Section 3.1.2, we consider the relative errors on the inner product. We start considering the numerical errors of the non-tensorial Clenshaw-Curtis cubature rule.

```
ep=zeros(3,length(mm));
for m=mm
    [XY]=paduapts(m); X=XY(:,1); Y=XY(:,2);
    W=pdwtsMM(m); W=W/sum(W); sa=zeros(Mp(m)+1,1);
    for k=1:3
      for i=1:Mp(m)+1
        [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
        sa(i)=max(real(eig(A,B)));
      end
      as=sum(W.*sa.*L{j+1}(X).*L{j+1}(Y));
      ep(k,m)=abs(as-apT(k,j+1));
    end
end
```

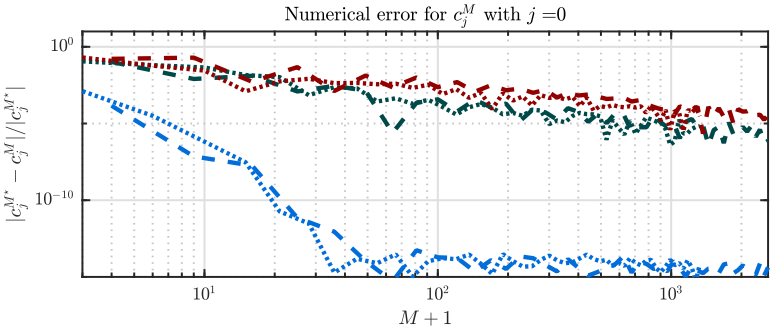
Hence, we consider the numerical errors of the tensorial Clenshaw-Curtis cubature rule.

```
ec=zeros(3,length(mm));
for m=mm
    [X,W]=chebpts(m+1); XY=tensor(X); X=XY(:,1); Y=XY(:,2);
    W=prod(tensor(W'),2); W=W/sum(W); sa=zeros(Mc(m)+1,1);
```

```
    for k=1:3
     for i=1:Mc(m)+1
      [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
      sa(i)=max(real(eig(A,B)));
     end
     as=sum(W.*sa.*L{j+1}(X).*L{j+1}(Y));
     ec(k,m)=abs(as-acT(k,j+1));
    end
end; close
```

The numerical errors for non-tensorial and tensorial Clenshaw-Curtis cubature rules are shown in the following figure, which corresponds to FIGURE 8 in [6].

```
for k=1:3
 loglog(Mp+1,ep(k,:)/abs(apT(k,j+1)),':',C,col{k},LW,2); hold on
 loglog(Mc+1,ec(k,:)/abs(acT(k,j+1)),'--',C,col{k},LW,2);
end, hold off, axis([Mp(1)+1,Mc(end)+1,1e-15,10]); xlabel(M1);
grid on,  ylabel('$|c_j^{M^\star}-c_j^{M}|/|c_j^{M^\star}|$')
title(['Numerical error for $c_j^M$ with $j=$',num2str(j)]);
```



The convergence rates in this case fluctuates a lot and it is not easy to empirally compute the convergence rates of these cubature rules. The estimation can be achieved by looking at the slope of the loglog plot.

4.1.3 Decoupling numerical and approximation errors

In this section, we first evaluate the polynomial approximations which follow the natural choices given in SECTION 4.1.3 in [6]. Then, as a counter check, we construct polynomial approximations which do not respect the advice, even though the number of points of the cubature rules is bigger than the number of approximated coefficients.

```
mp=[25, 30]; MP=(mp+1).*(mp+2)/2-1;
mc=[25, 15]; MC=(mc+1).^2-1;
```

We can observe that `mp(1)` and `mc(1)` represent natural choices for polynomial approximations with total and maximal degree up to `Pdmax`, respectively.

`mc(2)` and `mp(2)` do not satisfy the advice for polynomial approximations with total and maximal degree up to `Pdmax`, respectively. Indeed `mc(2)<Pdmax` and `mp(2)<2*Pdmax`, even though `MC(2)>nchoosek(Pdmax+2,2)-1` and `MP(2)>(Pdmax+1)`^{2}.

We fist consider the total degree polynomial approximations, constructed following the natural choice.

```
[XY]=paduapts(mp(1)); W=pdwtsMM(mp(1)); W=W/sum(W);
net=zeros(3,Pdmax+1); aet=net; X=XY(:,1); Y=XY(:,2);
for k=1:3, sa=zeros(MP(1)+1,1);
  for i=1:MP(1)+1
    [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
    sa(i)=max(real(eig(A,B)));
  end
   at=sum(W.*sa.*L{1}(X).*L{1}(Y)); net(k,1)=abs(at-apT(k,1));
   SA=at*ones(size(AlphaT(:,:,k)));
   aet(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
 for pd=1:Pdmax
    for j=(pd+1)*pd/2+1:(pd+1)*(pd+2)/2
      at=sum(W.*sa.*L{i1(c2r(j))+1}(X).*L{i2(c2r(j))+1}(Y));
      net(k,pd+1)=net(k,pd+1)+abs(at-apT(k,c2r(j)));
      SA=SA+at*L2(:,:,c2r(j));
    end
    aet(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
 end
end
```

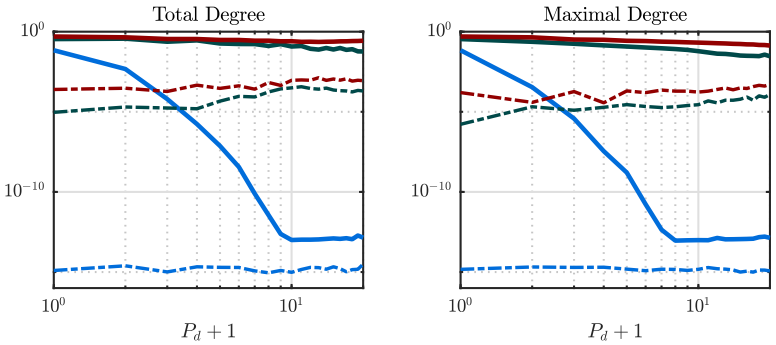
Hence, we evaluate the maximal degree polynomial approximation following the advice.

```
[X,W]=chebpts(mc(1)+1); XY=tensor(X); W=prod(tensor(W'),2);
nem=zeros(3,Pdmax+1); aem=nem; W=W/sum(W); X=XY(:,1); Y=XY(:,2);
for k=1:3, sa=zeros(MC(1)+1,1);
 for i=1:MC(1)+1
  [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
  sa(i)=max(real(eig(A,B)));
 end
  am=sum(W.*sa.*L{1}(X).*L{1}(Y)); nem(k,1)=abs(am-apT(k,1));
  SA=am*ones(size(AlphaT(:,:,k)));
  aem(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
  for pd=1:Pdmax
   for j=((pd)^2+1:(pd+1)^2)
    am=sum(W.*sa.*L{i1(j)+1}(X).*L{i2(j)+1}(Y));
    nem(k,pd+1)=nem(k,pd+1)+abs(am-apT(k,j)); SA=SA+am*L2(:,:,j);
   end
   aem(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
```

```
  end
end
```

The convergence of the approximation errors, obtained following the natural choices, is not affected by the numerical errors, as illustrated in the following panes, which correspond to FIGURE 7.

```
for k=1:3
 subplot(1,2,1), loglog(pp,aet(k,:),C,col{k},LW,2);
 hold on, loglog(pp,net(k,:),'-.',C,col{k},LW,1.5);
 subplot(1,2,2), loglog(pp,aem(k,:),C,col{k},LW,2);
 hold on, loglog(pp,nem(k,:),'-.',C,col{k},LW,1.5);
end
subplot(1,2,1),grid on, xlabel(P2); title(TD); axis(AX)
subplot(1,2,2),grid on, xlabel(P2); title(MD); axis(AX)
```



In a similar fashion we compute the total degree polynomial approximation, whose coefficients are computed via `mc(2)` tensorial Clenshaw-Curtis cubature rule.

```
[X,W]=chebpts(mc(2)+1); XY=tensor(X); W=prod(tensor(W'),2);
net=zeros(3,Pdmax+1); aet=net; W=W/sum(W); X=XY(:,1); Y=XY(:,2);
for k=1:3, sa=zeros(MC(2)+1,1);
 for i=1:MC(2)+1
  [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
  sa(i)=max(real(eig(A,B)));
 end
 at=sum(W.*sa.*L{1}(X).*L{1}(Y)); net(k,1)=abs(at-apT(k,1));
 SA=at*ones(size(AlphaT(:,:,k)));
 aet(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
 for pd=1:Pdmax
  for j=(pd+1)*pd/2+1:(pd+1)*(pd+2)/2
  at=sum(W.*sa.*L{i1(c2r(j))+1}(X).*L{i2(c2r(j))+1}(Y));
  net(k,pd+1)=net(k,pd+1)+abs(at-apT(k,c2r(j)));
  SA=SA+at*L2(:,:,c2r(j));
  end
```

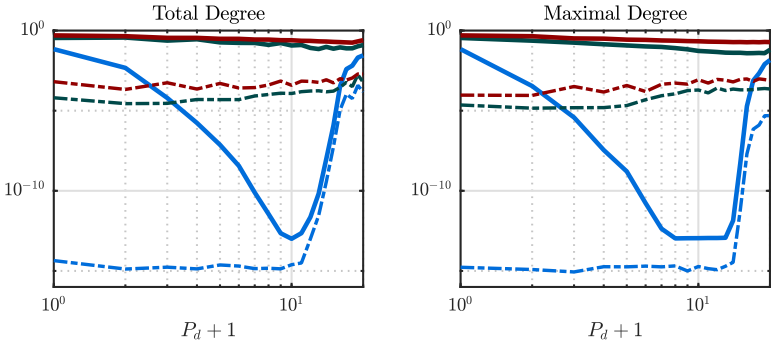
```
  aet(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
 end
end
```

Hence, we consider the maximal degree polynomial approximation, whose coefficients are computed via `mp(2)` non-tensorial Clenshaw-Curtis cubature rule.

```
[XY]=paduapts(mp(2)); W=pdwtsMM(mp(2)); W=W/sum(W);
nem=zeros(3,Pdmax+1); aem=nem; X=XY(:,1); Y=XY(:,2);
for k=1:3, sa=zeros(MP(2)+1,1);
 for i=1:MP(2)+1
  [A,B]=DelayedOscillator(Lt1(X(i)),Lt2(Y(i)),k,IG,stru);
  sa(i)=max(real(eig(A,B)));
 end
 am=sum(W.*sa.*L{1}(X).*L{1}(Y)); nem(k,1)=abs(am-apT(k,1));
 SA=am*ones(size(AlphaT(:,:,k)));
 aem(k,1)=max(max(abs(SA-AlphaT(:,:,k))));
 for pd=1:Pdmax
  for j=((pd)^2+1:(pd+1)^2)
  am=sum(W.*sa.*L{i1(j)+1}(X).*L{i2(j)+1}(Y));
  nem(k,pd+1)=nem(k,pd+1)+abs(am-apT(k,j));
  SA=SA+am*L2(:,:,j);
 end
 aem(k,pd+1)=max(max(abs(SA-AlphaT(:,:,k))));
 end
end; close
```

Analogously to FIGURE 9, the approximation error is dominated by the numerical errors of the cubature rules, as soon as the advice of SECTION 4.1.3 of [6] is not respected.

```
for k=1:3
 subplot(1,2,1), loglog(pp,aet(k,:),C,col{k},LW,2);
 hold on; loglog(pp,net(k,:),'-.',C,col{k},LW,1.5);
 subplot(1,2,2), loglog(pp,aem(k,:),C,col{k},LW,2);
 hold on; loglog(pp,nem(k,:),'-.',C,col{k},LW,1.5);
end
subplot(1,2,1),grid on, xlabel(P2); title(TD); axis(AX)
subplot(1,2,2),grid on, xlabel(P2); title(MD); axis(AX)
```



4.2 Collocation approach

The collocation approach in the bivariate case can be easily achieved by Chebfun. Let us consider the error of the interpolant on Padua points for total multivariate degree.

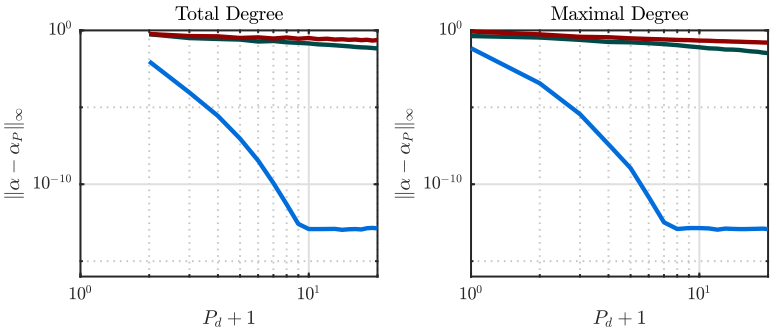
```
err_t=zeros(3,Pdmax);
for pd=1:Pdmax,  XY=paduapts(pd);
 for k=1:3,      sa=zeros(size(XY(:,1)));
  for i=1:length(sa)
   [A,B]=DelayedOscillator(Lt1(XY(i,1)),Lt2(XY(i,2)),k,IG,stru);
   sa(i)=max(real(eig(A,B)));
  end
  SA=chebfun2(sa, [-1 1 -1 1], 'padua');
  err_t(k,pd)=max(max(abs(SA(XXT,YYT)-AlphaT(:,:,k))));
 end
end
```

We analyze the error of the maximal degree polynomial interpolant on tensor product Chebyshev grid. (The construction of the grid differs from the one previously seen.)

```
err_m=zeros(3,Pdmax+1);
for pd=0:Pdmax, [XX, YY] = chebpts2(pd+1);
 for k=1:3,     sa=zeros((pd+1),(pd+1));
  for i=1:pd+1
   for j=1:pd+1
   [A,B]=DelayedOscillator(Lt1(XX(i,j)),Lt2(YY(i,j)),k,IG,stru);
   sa(i,j)=max(real(eig(A,B)));
   end
  end
  SA=chebfun2(sa);
  err_m(k,pd+1)=max(max(abs(SA(XXT,YYT)-AlphaT(:,:,k))));
 end
end, close
```

The convergence of the errors can be shown by the following code.

```
for k=1:3
 subplot(1,2,1),loglog(2:Pdmax+1,err_t(k,:),C,col{k},LW,2);hold on
 subplot(1,2,2),loglog(1:Pdmax+1,err_m(k,:),C,col{k},LW,2);hold on
end
subplot(1,2,1),grid on, xlabel(P2); ylabel(AP); title(TD); axis(AX)
subplot(1,2,2),grid on, xlabel(P2); ylabel(AP); title(MD); axis(AX)
```



The convergence rates are similar; in particular, for the non-smooth cases, the empirically convergence rates are estimated by:

```
disp('        Total Degree     Maximal Degree');
for k=2:3
 ot=rates(err_t(k,:),8:2:Pdmax); om=rates(err_m(k,:),1:2:Pdmax);
 fprintf([cases{k},'  %10.8f %14.8f \n'], median(ot), median(om));
end
```

```
        Total Degree     Maximal Degree
MSSAEs   0.99982882     1.06277303
MNSSAEs  0.31210863     0.47209949
```

Conclusions and Acknowledgments

Refer to [6] for further analyses and conclusion.

Acknowledgments. This work was supported by the project C14/17/072 of the KU Leuven Research Council, by the project G0A5317N of the Research Foundation-Flanders (FWO - Vlaanderen), and by the project UCoCoS, funded by the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie Grant Agreement No 675080.

References

[1] M. Abramowitz & I. A. Stegun, *Handbook of Mathematical Functions,* Dover, 1965.

[2] D. Breda, S. Maset & R. Vermiglio, *Stability of Linear Delay Differential Equations - A numerical approach with MATLAB,* Springer, 2015.

[3] J. P. Boyd, *Chebyshev and Fourier Spectral Methods,* Dover, 2001.

[4] M. Caliari, S. de Marchi, A. Sommariva, & M. Vianello, Padua2DM: fast interpolation and cubature at the Padua points in MATLAB/Octave, *Numerical Algorithms* 56 (2011), 45–60.

[5] L. Fenzi & W. Michiels, Robust stability optimization for linear delay systems in a probabilistic framework, *Linear Algebra and its Applications* 526 (2017), 1–26.

[6] L. Fenzi & W. Michiels, Polynomial (chaos) approximation of maximum eigenvalue function: efficiency and limitations, *arXiv,* 1804.03881 (2018).

[7] E. Jarlebring, K. Meerbergen & W. Michiels, A Krilov method for the delay eigenvalue problem, *SIAM Journal of Computer Science,* 32 (2010), 3278–3300.

[8] M. P. Szudzik, The Rosenberg-Strong pairing function, *arXiv,* 1706.04129 (2017).

[9] L. N. Trefethen, *Spectral Methods in MATLAB,* SIAM, 2000.

[10] L. N. Trefethen, *Approximation Theory and Approximation Practice,* Cambridge University Press, 2012.

A MATLAB functions for bivariate analysis

This appendix reports the additional MATLAB functions requested for the analysis of the bivariate case, Section 4. Before each function, the help is given.

`[A,B]=DelayedOscillator(nu,xi,RE,N,stru)` constructs the approximation of the delay eigenvalue problem associated to the oscillator with feedback delay, Example 2 in [6]. The non-linear finite dimensional eigenvalue problem is turned into a discretized version of the infinite dimensional linear problem, by the Infinitesimal Generator approach [2, 7]. The oscillator with feedback delay system is

$$\ddot{x}(t) = -\texttt{nu}^2 x(t) - 2\texttt{nu}\cdot\texttt{xi}\ \dot{x}(t) + \texttt{K(1)}\ x(t-1) + \texttt{K(2)}\ \dot{x}(t-1).$$

Input:

`nu` damping ratio, which is studied in $[0.9, 1.1]$.

`xi` angular frequency, which is studied in $[0.1, 0.2]$.

`RE` specifies the controller parameters and consequently the behavior of the Rightmost Eigenvalues, TABLE 1 in [6]. The SAE, MSSAEs and MNSSAEs cases can be obtained by setting `RE` equals to the numbers `1`, `2`, and `3`, respectively.

`N` discretization of the Infinitesimal Generator. It is an integer which defines the dimension of the final characteristic matrix, *i.e.* $2(\mathtt{N}+1)\times 2(\mathtt{N}+1)$.

`stru` structure of the discretized infinitesimal generator. Setting `stru=0`, the eigenvalue problem does not present a particular structure as in [2], while setting `stru=1`, we get a structured eigenvalue problem by the approach [7].

Output: `(A,B)` generalized eigenvalue problem (cf. *e.g.* `help eig` for further information).

```
function [A,B]=DelayedOscillator(nu,xi,RE,N,stru)
% Controller parameters - (Table 1 in [6])
if RE==1,      K=[0.2, 0.2];          % SAE
elseif RE==2,  K=[0.5105, -0.0918];   % MSSAEs
elseif RE==3,  K=[0.6179, -0.0072];   % MNSSAEs
else, error('MyComponent:incorrectType', ['Error.\n'...
 'RE must be a number\n 1 - SAE,\n 2 - MSSAEs,\n 3 - MNSSAEs.'])
end
% Definition of the oscillator with feedback delay system
n=2;            % Dimension of the system
h=2; TAU=[0,1]; % Delays
E=eye(n);       % Leading matrix
A=cell(1,h); A{1}=[0,1; -nu^2, -2*nu*xi];
             A{h}=[0,0;  K(1), K(2)];
% INFINITESIMAL GENERATOR APPROACH
if stru==0      % No stucture, approach in [2]
% Differentiation matrix D (pag 54 in [9])
  x=(TAU(h)/2)*(cos(pi*(0:N)/N)'-1); % Chebyshev nodes in [tau,0]
  c=[2; ones(N-1,1); 2].*(-1).^(0:N)';
  X=repmat(x,1,N+1);   dX=X-X';
  D=(c*(1./c)')./(dX+(eye(N+1))); D=D-diag(sum(D,2));
  DN=kron(D,eye(n));
% Definition of the eigenvalue problem (Example 5.1 in [2])
  AN=zeros(n*(N+1));  AN(1:n,1:n)=A{1}; AN(1:n,n*N+1:end)=A{2};
  AN(n+1:end,:)=DN(n+1:end,:); A=AN; B=eye(size(A));
elseif stru==1    % Structured eigenvalue problem [7]
% Band Matrix L_N (Section 2.3 in [7])
  L_N=zeros(N+1); L_N(2,1:3)=[2 0 -1];
  for i=3:1:N, L_N(i,i-1:i+1)=[1/(i-1) 0 -1/(i-1)];
  end
  L_N(N+1,N:N+1)=[1/N 0]; L_N=(TAU(h)/4)*L_N;
```

```
% Pi_N & Sigma_N matrices (Theorem 2.1 in [7].)
  Pi_N=kron(L_N,eye(n)); Pi_N(1:n,:)=kron(ones(1,N+1),E);
  Sigma_N=eye(n*(N+1));  Sigma_N(1:n,1:n)=zeros(n);
  for i=1:N+1, xx=(-2*TAU/TAU(h)+1);
 % Chebishev polynomial of degree i-1 evaluated in xx
   if i==1, CS=ones(1,h);
   elseif i==2, CS=xx;  C1=xx; C2=ones(1,h);
   else,        CS=2*xx.*C1-C2; C2=C1; C1=CS;
   end
   for j=1:h % First row block of Sigma_N
      Sigma_N(1:n,((i-1)*n+1):(i*n))=...
         Sigma_N(1:n,((i-1)*n+1):(i*n))+A{j}*CS(j);
   end
  end, A=Sigma_N; B=Pi_N; % Definition of the eigenvalue problem
else
 error('MyComponent:incorrectType',['Error.\nstru must be a',...
  'number\n  0 - No Structure,\n  1 - Structured eigenproblem.'])
end
end
```

`W= pdwtsMM(n)` computes the cubature weights `W` by Matrix Multiplication (MM) so that, if `Pad` is the matrix of Padua points computed in Chebfun by `Pad = paduadpts(n)`, then the cubature of a function `funct` is given by `W'*funct(Pad(:,1),Pad(:,2))`. The interested reader is referred to [4] for further information on the topic and on the algorithm.
Input: `n` interpolation degree.
Output: `W` cubature weights associated to the matrix `Pad` of Padua points.

```
function W= pdwtsMM(n)
if n == 0,  W = 4; % degree 0
else
  argn1=linspace(0,pi,n+1); argn2=linspace(0,pi,n+2);
  k=(0:2:n)'; l=(n-mod(n,2))/2+1; lp=(n+mod(n,2))/2+1;
% even-degree Chebyshev polynomials on the subgrids
  TE1=cos(k*argn1(1:2:n+1)); TE1(2:l,:)=TE1(2:l,:)*sqrt(2);
  TO1=cos(k*argn1(2:2:n+1)); TO1(2:l,:)=TO1(2:l,:)*sqrt(2);
  TE2=cos(k*argn2(1:2:n+2)); TE2(2:l,:)=TE2(2:l,:)*sqrt(2);
  TO2=cos(k*argn2(2:2:n+2)); TO2(2:l,:)=TO2(2:l,:)*sqrt(2);
% even,even moments matrix
  mom=2*sqrt(2)./(1-k.^2); mom(1)=2; [M1,M2]=meshgrid(mom);
  M0 = fliplr(triu(fliplr(M1.*M2)));
% interpolation weights matrices
  W1=2*ones(l)/(n*(n+1));       W1(:,1)=W1(:,1)/2;
  W2=2*ones(lp,lp-1)/(n*(n+1)); W2(1,:)=W2(1,:)/2;
  if mod(n,2)==0, i=n/2+1;
    M0(i,1) = M0(i,1)/2; W1(:,i)=W1(:,i)/2; W1(i,:)=W1(i,:)/2;
```

```
  else, i=(n+1)/2;
    W2(i+1,:)=W2(i+1,:)/2; W2(:,i)=W2(:,i)/2;
  end
% cubature weights as matrices on the subgrids.
  L1=W1.*(TE1'*M0*TO2)';  L2=W2.*(TO1'*M0*TE2)';
  if mod(n,2) == 0 % W=zeros(n/2+1,n+1);
    W(:,1:2:n+1)=L1; W(:,2:2:n+1)=L2; W=W(:);
  else
    W=[L1',L2']';                     W=W(:);
  end
end
end
```