Appendix A
Generation of Multiwavelets

Algorithm 1 Generation of multiwavelets (mother-wavelets (2.14))

Start with the set of functions $\{f_k^1\}_{k=0}^{N_p}$, defined by

$$f_k^1(\xi) = \begin{cases} \xi^k, & \xi \in [-1, 0], \\ -\xi^k, & \xi \in [0, 1], \\ 0, & \text{otherwise.} \end{cases}$$

STEP 1: Orthogonalize w.r.t. the monomials $1, \ldots, \xi^{N_p}$ (Gram-Schmidt) to obtain $\{f_k^2\}_{k=0}^{N_p}$.
STEP 2:
  for $i \leftarrow 0$ to $N_p - 1$ **do**
    Make sure $\langle f_i^{i+1} \xi^{N_p+i} \rangle \neq 0$ (otherwise reorder).
    for $j = i + 1$ to N_0 **do**
      $w = \frac{\langle f_j^{i+2} \xi^{N_p+i} \rangle}{\langle f_i^{i+2} \xi^{N_p+i} \rangle}$
      $f_j^{i+3} \leftarrow f_j^{i+2} - w f_i^{i+2}$
    end for
  end for
STEP 3: Orthogonalize $\{f_i^{i+2}\}_{i=0}^{N_p}$ using G-S.
  for $i \leftarrow N_p$ to 0 **do**
    $\psi_i^W(\xi) \leftarrow$ Apply Gram-Schmidt to f_i^{i+2}.
  end for
  Output $\{\psi_i^W(\xi)\}_{i=0}^{N_p}$.

M.P. Pettersson et al., *Polynomial Chaos Methods for Hyperbolic Partial Differential Equations*, Mathematical Engineering,
DOI 10.1007/978-3-319-10714-1

Appendix B
Proof of Constant Eigenvectors of Low-Order MW Triple Product Matrices

B.1 Proof of Constant Eigenvectors of A

Proposition B.1. *The matrix $\boldsymbol{A}$ defined by (8.3) for Haar wavelets $\{\psi_j\}_{j=0}^{M}$ has constant eigenvectors for all $M+1=2^{N_r}$, $N_r \in \mathbb{N}$.*

Proof (Sketch of proof). We will use induction on the order M of wavelet chaos to show that the matrix $\boldsymbol{A}$ has constant eigenvectors for all orders M. In order to do this, we will need certain features of the structure of $\boldsymbol{A}$. To facilitate the notation, denote $\tilde{M} = M+1$. We can express $\boldsymbol{A}_{2\tilde{M}}$ in terms of the matrix $\boldsymbol{A}_{\tilde{M}}$. Two properties of the triple product $\langle \psi_i \psi_j \psi_k \rangle$ will be used to prove that $\boldsymbol{A}$ does indeed have the matrix structure presented.

Property 1. Let $i \in \{0, \ldots, \tilde{M}-1\}$, $j = k \in \{\tilde{M}, \ldots, 2\tilde{M}-1\}$ and let j' and j'' be the progenies of j. Then

$$\langle \psi_i \psi_j^2 \rangle = \langle \psi_i \psi_{j'}^2 \rangle = \langle \psi_i \psi_{j''}^2 \rangle.$$

Property 2. Consider the indices $i \in \{\tilde{M}, \ldots, 2\tilde{M}-1\}$, $j = k \in \{2\tilde{M}, \ldots, 4\tilde{M}-1\}$. Then

$$\langle \psi_i \psi_j^2 \rangle = \begin{cases} \tilde{M}^{1/2} & \text{if } j \text{ first progeny of } i \\ -\tilde{M}^{1/2} & \text{if } j \text{ second progeny of } i. \\ 0 & \text{otherwise} \end{cases}$$

M.P. Pettersson et al., *Polynomial Chaos Methods for Hyperbolic Partial Differential Equations*, Mathematical Engineering,
DOI 10.1007/978-3-319-10714-1

As an induction hypothesis, we assume that given $\boldsymbol{A}_{\tilde{M}}$ for some $\tilde{M} = 2^{N_r}$, $N_r \in \mathbb{N}$, the next order of triple product matrix $\boldsymbol{A}_{2\tilde{M}}$ can be written

$$\boldsymbol{A}_{2\tilde{M}} = \begin{bmatrix} \boldsymbol{A}_{\tilde{M}} & \boldsymbol{Q}_{\tilde{M}} \boldsymbol{D}_{\tilde{M}} \\ \boldsymbol{D}_{\tilde{M}} \boldsymbol{Q}_{\tilde{M}}^T & \boldsymbol{\Lambda} \end{bmatrix},$$

where $\boldsymbol{Q}_{\tilde{M}}$ is the matrix of constant eigenvectors of $\boldsymbol{A}_{\tilde{M}}$ satisfying $\left\| \boldsymbol{Q}_{\tilde{M}} \right\|_2^2 = \tilde{M}$, $\boldsymbol{D}_{\tilde{M}} = \operatorname{diag}(w_{\tilde{M}}, \ldots, w_{2\tilde{M}-1})$ and $\boldsymbol{\Lambda}$ is diagonal and contains the eigenvalues of $\boldsymbol{A}_{\tilde{M}}$. Then, we have that

$$\begin{bmatrix} \boldsymbol{A}_{\tilde{M}} & \boldsymbol{Q}_{\tilde{M}} \boldsymbol{D}_{\tilde{M}} \\ \boldsymbol{D}_{\tilde{M}} \boldsymbol{Q}_{\tilde{M}}^T & \boldsymbol{\Lambda} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \\ \pm \tilde{M}^{1/2} \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \boldsymbol{\Lambda} \pm \tilde{M}^{1/2} \boldsymbol{Q} \boldsymbol{D}_{\tilde{M}} \\ \tilde{M} \boldsymbol{D}_{\tilde{M}} \pm \tilde{M}^{1/2} \boldsymbol{\Lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \\ \pm \tilde{M}^{1/2} \boldsymbol{I} \end{bmatrix} (\boldsymbol{\Lambda} \pm \tilde{M}^{1/2} \boldsymbol{D}_{\tilde{M}}),$$

so the eigenvalues and eigenvectors of $\boldsymbol{A}_{2\tilde{M}}$ are given by $\boldsymbol{\Lambda} \pm \tilde{M}^{1/2} \boldsymbol{D}_{\tilde{M}}$ and $[\boldsymbol{Q}_{\tilde{M}}, \pm \tilde{M}^{1/2} \boldsymbol{I}]^T$, respectively. For the next order of expansion, $4\tilde{M}$, we have

$$\boldsymbol{A}_{4\tilde{M}} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{A}_{\tilde{M}} & \boldsymbol{Q}_{\tilde{M}} \boldsymbol{D}_{\tilde{M}} \\ \boldsymbol{D}_{\tilde{M}} \boldsymbol{Q}_{\tilde{M}}^T & \boldsymbol{\Lambda} \end{bmatrix} & \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \otimes [1,1] \\ \tilde{M}^{1/2} \boldsymbol{I} \otimes [1,-1] \end{bmatrix} \boldsymbol{D}_{2\tilde{M}} \\ \boldsymbol{D}_{2\tilde{M}} \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \otimes [1,1] \\ \tilde{M}^{1/2} \boldsymbol{I} \otimes [1,-1] \end{bmatrix}^T & \boldsymbol{\Lambda} \otimes \boldsymbol{I}_2 + \tilde{M}^{1/2} \boldsymbol{D}_{\tilde{M}} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \end{bmatrix}. \tag{B.1}$$

To see that this is indeed the structure of $\boldsymbol{A}_{4\tilde{M}}$, note that any nonzero matrix entry not already present in $\boldsymbol{A}_{2\tilde{M}}$, can be deduced using properties 1 and 2, and scaling the rows/columns by multiplication by the diagonal matrix $\boldsymbol{D}_{2\tilde{M}}$. The structure of $\boldsymbol{A}_{4\tilde{M}}$ follows from the construction of the Haar wavelet basis, but we do not give a proof here.

One can verify that $\boldsymbol{A}_{4\tilde{M}}$ given by (B.1) has the eigenvectors and eigenvalues

$$\boldsymbol{Q}_{4\tilde{M}} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{Q}_{\tilde{M}} \otimes [1,1] \\ \tilde{M}^{1/2} \boldsymbol{I}_{\tilde{M}} \otimes [1,-1] \end{bmatrix} \\ \pm (2\tilde{M})^{1/2} \boldsymbol{I}_{2\tilde{M}} \end{bmatrix},$$

$$\boldsymbol{\Lambda}_{4\tilde{M}} = \boldsymbol{\Lambda} \otimes \boldsymbol{I}_2 + \tilde{M}^{1/2} \boldsymbol{D}_{\tilde{M}} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \pm (2\tilde{M})^{1/2} \boldsymbol{D}_{2\tilde{M}},$$

so the eigenvectors are constant (but the eigenvalues are variable in the coefficients $(w_i)_j$ through $M_{\tilde{M}}$ and $M_{2\tilde{M}}$). The base cases $\tilde{M} = 1$, $\tilde{M} = 2$, can easily be verified, so by induction $\boldsymbol{A}_{\tilde{M}}$ has constant eigenvectors for all $\tilde{M} = 2^{N_r}$, $N_r \in \mathbb{N}$.

B.2 Eigenvalue Decompositions of A

B.2.1 Piecewise Constant Multiwavelets (Haar Wavelets)

B.2.1.1 $N_r = 2$

$$Q = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}, \quad \Lambda = diag\begin{bmatrix} u_0 + u_1 + \sqrt{2}u_2 \\ u_0 + u_1 - \sqrt{2}u_2 \\ u_0 - u_1 + \sqrt{2}u_3 \\ u_0 - u_1 - \sqrt{2}u_3 \end{bmatrix}$$

B.2.1.2 $N_r = 3$

$$Q = \frac{1}{\sqrt{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

$$\Lambda = diag\begin{bmatrix} u_0 + u_1 + \sqrt{2}u_2 + 2u_4 \\ u_0 + u_1 + \sqrt{2}u_2 - 2u_4 \\ u_0 + u_1 - \sqrt{2}u_2 + 2u_5 \\ u_0 + u_1 - \sqrt{2}u_2 - 2u_5 \\ u_0 - u_1 + \sqrt{2}u_3 + 2u_6 \\ u_0 - u_1 + \sqrt{2}u_3 - 2u_6 \\ u_0 - u_1 - \sqrt{2}u_3 + 2u_7 \\ u_0 - u_1 - \sqrt{2}u_3 - 2u_7 \end{bmatrix}$$

B.2.2 Piecewise Linear Multiwavelets

B.2.2.1 $N_r = 1$

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{\sqrt{3}+1}{4} & \frac{\sqrt{3}-1}{4} & \frac{\sqrt{3}+1}{4} & -\frac{\sqrt{3}-1}{4} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{\sqrt{3}-1}{4} & -\frac{\sqrt{3}+1}{4} & \frac{\sqrt{3}-1}{4} & -\frac{\sqrt{3}+1}{4} \end{bmatrix}, \quad \Lambda = diag\begin{bmatrix} u_0 - \frac{\sqrt{3}+1}{2}u_1 - u_2 - \frac{\sqrt{3}-1}{2}u_3 \\ u_0 + \frac{\sqrt{3}-1}{2}u_1 + u_2 - \frac{\sqrt{3}+1}{2}u_3 \\ u_0 + \frac{\sqrt{3}+1}{2}u_1 - u_2 + \frac{\sqrt{3}-1}{2}u_3 \\ u_0 - \frac{\sqrt{3}-1}{2}u_1 + u_2 - \frac{\sqrt{3}+1}{2}u_3 \end{bmatrix}$$

B.2.2.2 $N_r = 2$

$$\boldsymbol{Q}=\begin{bmatrix}
\frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} \\
\frac{\sqrt{14+3\sqrt{3}}}{8} & -\frac{\sqrt{14+3\sqrt{3}}}{8} & -\frac{\sqrt{14-3\sqrt{3}}}{8} & \frac{\sqrt{14-3\sqrt{3}}}{8} & \frac{\sqrt{3}+1}{8\sqrt{2}} & -\frac{\sqrt{3}+1}{8\sqrt{2}} & -\frac{\sqrt{3}-1}{8\sqrt{2}} & \frac{\sqrt{3}-1}{8\sqrt{2}} \\
-\frac{\sqrt{3}+1}{4\sqrt{2}} & -\frac{\sqrt{3}+1}{4\sqrt{2}} & -\frac{\sqrt{3}-1}{4\sqrt{2}} & -\frac{\sqrt{3}-1}{4\sqrt{2}} & \frac{\sqrt{3}-1}{4\sqrt{2}} & \frac{\sqrt{3}-1}{4\sqrt{2}} & \frac{\sqrt{3}+1}{4\sqrt{2}} & \frac{\sqrt{3}+1}{4\sqrt{2}} \\
\frac{\sqrt{3}+1}{8\sqrt{2}} & -\frac{\sqrt{3}+1}{8\sqrt{2}} & \frac{\sqrt{3}-1}{8\sqrt{2}} & -\frac{\sqrt{3}-1}{8\sqrt{2}} & -\frac{\sqrt{14-5\sqrt{3}}}{8} & \frac{\sqrt{14-5\sqrt{3}}}{8} & \frac{\sqrt{14+5\sqrt{3}}}{8} & -\frac{\sqrt{14+5\sqrt{3}}}{8} \\
0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\
0 & -\frac{\sqrt{3}-1}{4} & \frac{\sqrt{3}+1}{4} & 0 & 0 & -\frac{\sqrt{3}+1}{4} & \frac{\sqrt{3}-1}{4} & 0 \\
-\frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\
\frac{\sqrt{3}-1}{4} & 0 & 0 & -\frac{\sqrt{3}+1}{4} & \frac{\sqrt{3}+1}{4} & 0 & 0 & -\frac{\sqrt{3}-1}{4}
\end{bmatrix}$$

$$\boldsymbol{\Lambda} = diag\begin{bmatrix}
u_0 + \sqrt{\frac{14+3\sqrt{3}}{8}}u_1 - \frac{\sqrt{3}+1}{2}u_2 + \frac{\sqrt{3}+1}{4}u_3 - \sqrt{2}u_6 + \frac{\sqrt{3}-1}{\sqrt{2}}u_7 \\
u_0 - \sqrt{\frac{14+3\sqrt{3}}{8}}u_1 - \frac{\sqrt{3}+1}{2}u_2 - \frac{\sqrt{3}+1}{4}u_3 - \sqrt{2}u_4 - \frac{\sqrt{3}-1}{\sqrt{2}}u_5 \\
u_0 - \sqrt{\frac{14-3\sqrt{3}}{8}}u_1 - \frac{\sqrt{3}-1}{2}u_2 + \frac{\sqrt{3}-1}{4}u_3 + \sqrt{2}u_4 + \frac{\sqrt{3}+1}{\sqrt{2}}u_5 \\
u_0 + \sqrt{\frac{14-3\sqrt{3}}{8}}u_1 - \frac{\sqrt{3}-1}{2}u_2 - \frac{\sqrt{3}-1}{4}u_3 + \sqrt{2}u_6 - \frac{\sqrt{3}+1}{\sqrt{2}}u_7 \\
u_0 + \frac{\sqrt{3}+1}{4}u_1 + \frac{\sqrt{3}-1}{2}u_2 - \sqrt{\frac{14-5\sqrt{3}}{8}}u_3 + \sqrt{2}u_6 + \frac{\sqrt{3}+1}{\sqrt{2}}u_7 \\
u_0 - \frac{\sqrt{3}+1}{4}u_1 + \frac{\sqrt{3}-1}{2}u_2 + \sqrt{\frac{14-5\sqrt{3}}{8}}u_3 + \sqrt{2}u_4 - \frac{\sqrt{3}+1}{\sqrt{2}}u_5 \\
u_0 - \frac{\sqrt{3}-1}{4}u_1 + \frac{\sqrt{3}+1}{2}u_2 + \sqrt{\frac{14+5\sqrt{3}}{8}}u_3 - \sqrt{2}u_4 + \frac{\sqrt{3}-1}{\sqrt{2}}u_5 \\
u_0 + \frac{\sqrt{3}-1}{4}u_1 + \frac{\sqrt{3}+1}{2}u_2 - \sqrt{\frac{14+5\sqrt{3}}{8}}u_3 - \sqrt{2}u_6 - \frac{\sqrt{3}-1}{\sqrt{2}}u_7
\end{bmatrix}$$

Appendix C
Matlab Codes

The reference codes used to generate the results presented in Chaps. 5 and 6 are reported here. They can also be downloaded from the website http://extras.springer.com/2015/

C.1 Linear Transport

C.1.1 Main Code

C.1.1.1 advection_diffusion_main.m

```
% Advection-diffusion with stochastic viscosity coefficient, SBP-SAT
% implementation
% Time integration with 4th order Runge-Kutta

clear all;

% Choose mod == 'herm' for Hermite polynomials representing mu w. lognormal
% distribution, or choose mod == 'lege' for Legendre polynomials and
% uniform distribution of mu.

mod = 'lege';

% Spatial interval end points
left = 0.4;
right = 0.6;

% Problem input parameters
v = 2;
M = 1;
rho_0 = 0.5;

c1= 0.02;
c2= 0.01;

t = 0;
t0 = 0.005;
x0 = 0.5;


p = 4; % Number of gPC coefficients
m = 50; % Number of discretization points in space
T = 0.001; % End time
```


M.P. Pettersson et al., *Polynomial Chaos Methods for Hyperbolic Partial Differential Equations*, Mathematical Engineering,
DOI 10.1007/978-3-319-10714-1

```
order = 6; % Order of accuracy of SBP operators, should be 2, 4 or 6.

% Setup parameters determined as functions of other parameters
dx = (right-left)/(m-1); % Spatial step length
dt = 0.2*dx^2/v; % Time step, here dependent on (Delta x)^2 but changes with ratio viscosity/velocity
V = v*eye(m*p); % Advective velocity, here assumed spatially uniform
x = linspace(left,right,m);


I_p=eye(p);
I_m=eye(m);

old = 1;
new = 2;

u = zeros(p*m,2);
%Compute inner triple products of the chosen basis functions
if mod == 'lege'
    C = Legendre_chaos(p-1);
end
if mod == 'herm'
    C = Hermite_chaos(p-1);
end

%Difference operators Dx=P^(-1)Q

[D1,D2,BD,D,Pnorm] = SBP_operators(m,dx,order);
P_inv = inv(Pnorm);
D1 = kron(D1,eye(p));
D2 = kron(D2,eye(p));


%Initialization and forcing terms
u_init = zeros(m*p,1);
if t0>0
    if mod == 'herm'
        u(:,old) = init_Hermite(rho_0,c1,c2,v,t,t0,x,x0,m,p);
    end
    if mod == 'lege'
        u(:,old) = init_Legendre(rho_0,c1,c2,v,t,t0,x,x0,m,p);
    end
end

% This is a first order approximation of the delta function
if t0 == 0
    u((m-1)/2*p+1,old) = rho_0/(dx);
end


if mod == 'herm'
    B1 = mu_SG_lognormal(c1,c2,p);
end
if mod == 'lege'
    if p>1
        B1 = C(:,:,1)*c1+C(:,:,2)*c2/sqrt(3);
    end
    if p==1
        B1 = C(:,:,1)*c1;
    end
end

% Compute force terms
E_0 = sparse(zeros(m));
E_0(1,1) = 1;
E_M = sparse(zeros(m));
E_M(m,m) = 1;
Sig_I_L = -v/2*eye(p);
Sig_V_L = B1;
Sig_V_R = -B1;
force_left_1 = sparse(kron(P_inv,I_p)*kron(E_0,Sig_I_L));
force_left_2 = kron(P_inv*D',I_p)*kron(E_0,Sig_V_L);
force_right = sparse(kron(P_inv,I_p)*kron(E_M,Sig_V_R));

g0 = zeros(m*p,1);
g1 = zeros(m*p,1);


B = kron(I_m,B1);

%Iterate over time with fourth order Runge-Kutta until time T is reached

while (t<T)
    if T-t<dt
        dt = T-t;
    end
    t = t+dt;
    if mod == 'herm'
        [g0 g1] = bdy_cond_Hermite(rho_0,c1,c2,v,t,t0,x,x0,m,p);
    end
```

```
122     if mod == 'lege'
123         [g0 g1] = bdy_cond_Legendre(rho_0,c1,c2,v,t,t0,x,x0,m,p);
124     end
125
126     % Fourth order Runge-Kutta in time
127     k1 = dt*((-V*D1+B*D2)*u(:,old)+force_left_1*(u(:,old)-g0)+force_left_2*(u(:,old)-g0)+force_right*(kron(D,
            I_p)*u(:,old)-g1));
128     k2 = dt*((-V*D1+B*D2)*(u(:,old)+k1/2)+force_left_1*(u(:,old)+k1/2-g0)+force_left_2*((u(:,old)+k1/2)-g0)+
            force_right*(kron(D,I_p)*(u(:,old)+k1/2)-g1));
129     k3 = dt*((-V*D1+B*D2)*(u(:,old)+k2/2)+force_left_1*(u(:,old)+k2/2-g0)+force_left_2*((u(:,old)+k2/2)-g0)+
            force_right*(kron(D,I_p)*(u(:,old)+k2/2)-g1));
130     k4 = dt*((-V*D1+B*D2)*(u(:,old)+k3)+force_left_1*(u(:,old)+k3-g0)+force_left_2*((u(:,old)+k3)-g0)+
            force_right*(kron(D,I_p)*(u(:,old)+k3)-g1));
131
132     % Update the solution vector
133     u(:,new)=u(:,old)+1/6*(k1+2*k2+2*k3+k4);
134     u(:,old)=u(:,new);
135
136
137     % Plot the solution
138     for k=1:p
139         u_plot(:,k) = u(k:p:end,new);
140     end
141     plot(x,u_plot,'-*');
142     title(['gPC coefficients, t=' num2str(t,'%.4f')])
143     leg_strs = {};
144     for k=1:p
145         legstrs{k} = ['gPC coe. ' num2str(k-1)];
146     end
147
148     legend(legstrs);
149     drawnow;
150 end
151
152 u = u(:,new);
153
154 % Statistics are readily obtained from the gPC coefficients
155 u_num_exp = u(1:p:p*(m-1)+1);
156 u_num_var = zeros(m,1);
157
158 for i=1:m
159     u_num_var(i,1) = sum(u((i-1)*p+2:i*p).^2);
160 end
161
162 % Compute reference statistics based on numerical quadrature of the exact
163 % solution
164 if mod == 'herm'
165     [u_exp u_var] = statistics_adv_diff_lognorm(m,t,x,x0,c1,c2,rho_0,t0,v);
166 end
167 if mod == 'lege'
168     [u_exp u_var] = statistics_adv_diff_uniform_mu(m,t,x,x0,c1,c2,rho_0,t0,v);
169 end
170
171 % Plot the computed and reference expectation and variance as a function of
172 % space (at time T)
173
174 figure;
175 plot(x,u_num_exp,'-r',x,u_exp,'--k','LineWidth',1.5);
176 h_legend = legend('Num','Ref');;
177 set(h_legend,'FontSize',12);
178 title('Mean','FontSize',16);
179
180 figure;
181 plot(x,u_num_var,'-r',x,u_var,'--k','LineWidth',1.5);
182 h_legend = legend('Num','Ref');;
183 set(h_legend,'FontSize',12);
184 title('Variance','FontSize',16);
```

C.1.1.2 Legendre_chaos.m

```
function [C] = legendre_chaos(n)

% Compute Legendre chaos parameters

% Indata:
% n - Order of gPC

% Outdata:
% C - Three term inner products C(i,j,k) = E[Phi_i Phi_j Phi_k]


C = zeros(n+1,n+1,n+1);
for i = 0:n
    for j = 0:n
        for k = 0:n
            s = (i+k+j) / 2;
            if rem(i+k+j,2) == 1 || abs(i-j) > k || k > i+j
                C(i+1,j+1,k+1) = 0;
            else
                C(i+1,j+1,k+1) = sqrt((2*i+1)*(2*j+1)*(2*k+1))/(i+j+k+1)*A_for_lege(s-i)*A_for_lege(s-j)*
        A_for_lege(s-k)/A_for_lege(s);
            end
        end
    end
end
return
```

C.1.1.3 Hermite_chaos.m

```
function [C] = Hermite_chaos(n)

% Compute hermite chaos parameters

% Indata:
% n - Order of gPC

% Outdata:
% C - Three term inner products C(i,j,k) = E[Phi_i Phi_j Phi_k]


C = zeros(n+1,n+1,n+1);
for i = 0:n
    for j = 0:n
        for k = 0:n
            s = (i+k+j) / 2;
            if rem(i+k+j,2) == 1 || i > s || j > s || k > s
                C(i+1,j+1,k+1) = 0;
            else
                C(i+1,j+1,k+1) = factorial(i) * factorial(j) ...
                    *factorial(k) / ((factorial(s-i) * factorial(s-j) * factorial(s-k)) * sqrt(factorial(i)) *
        sqrt(factorial(j)) * sqrt(factorial(k)));
            end
        end
    end
end
return
```

C.1.1.4 A_for_lege.m

```
function [A] = A_for_lege(n)

% Auxiliary function for triple product matrix of Legendre polynomials

if n==0
    A = 1;
end
if n < 0
    A = 0;
end
if n >= 1
    A = 1;
    for j=1:n
        A = A*(2*j-1);
    end
    A = A/factorial(n);
end
```

C.1.2 Discretization Operators

C.1.2.1 SBP_operators.m

```
1
2  function [D1 D2,BS,S,H] = SBP_operators(n,dx,order)
3
4  % SBP operators of orders 2, 4, 6 and 8 for the first and second derivative.
5
6  % Indata:
7  % n - Number of spatial grid pts
8  % dx - Step size
9  % order - Order of accuracy (only for 2,4,6,8)
10
11 % Outdata:
12 % D1 - First derivative operator
13 % D2 - Second derivative operator (D = P^{-1}M)
14 % S - First derivative operator on boundaries
15 % BS - The boundary elements in the energy estimate
16 % H - The norm operator (denoted P in some papers)
17
18
19 e = ones(n,1);
20
21 if order==2
22
23     D1 = 1/dx*spdiags([-1/2*e 0*e 1/2*e],-1:1,n,n);
24     D1(1,1) = -1/dx;
25     D1(1,2) = 1/dx;
26     D1(1,3) = 0;
27     D1(n,n) = 1/dx;
28     D1(n,n-1) = -1/dx;
29     D1(n,n-2) = 0;
30
31     %%%%%%%
32     D2 = 1/(dx^2)*spdiags([1*e -2*e 1*e],-1:1,n,n);
33     D2(1,1) = 1/(dx^2);
34     D2(1,2) = -2/(dx^2);
35     D2(1,3) = 1/(dx^2);
36     D2(n,n) = 1/(dx^2);
37     D2(n,n-1) = -2/(dx^2);
38     D2(n,n-2) = 1/(dx^2);
39
40
41     H = dx*spdiags([e],0,n,n);
42     H(1,1) = dx*1/2;
43     H(n,n) = dx*1/2;
44
45     BS = (1/dx)*spdiags(zeros(size(e)),0,n,n);
46     BS(1,1) = 3/2/dx;
47     BS(1,2) = -2/dx;
48     BS(1,3) = 1/2/dx;
49     BS(n,n) = 3/2/dx;
50     BS(n,n-1) = -2/dx;
51     BS(n,n-2) = 1/2/dx;
52
53     S = (1/dx)*spdiags([e],0,n,n);
54     S(1,1) = -3/2/dx;
55     S(1,2) = 2/dx;
56     S(1,3) = -1/2/dx;
57
58     S(n,n) = 3/2/dx;
59     S(n,n-1) = -2/dx;
60     S(n,n-2) = 1/2/dx;
61
62 elseif order==4
63
64     D1 = 1/dx*spdiags([1/12*e -2/3*e 0*e 2/3*e -1/12*e],-2:2,n,n);
65     D1(1,1) = -24/17/dx;
66     D1(1,2) = 59/34/dx;
67     D1(1,3) = -4/17/dx;
68     D1(1,4) = -3/34/dx;
69     D1(1,5) = 0;
70     D1(1,6) = 0;
71     D1(2,1) = -1/2/dx;
72     D1(2,2) = 0;
73     D1(2,3) = 1/2/dx;
74     D1(2,4:6) = 0;
75     D1(3,1) = 4/43/dx;
76     D1(3,2) = -59/86/dx;
77     D1(3,3) = 0;
78     D1(3,4) = 59/86/dx;
79     D1(3,5) = -4/43/dx;
80     D1(3,6) = 0;
81     D1(4,1) = 3/98/dx;
82     D1(4,2) = 0;
```

```
83      D1(4,3) = -59/98/dx;
84      D1(4,4) = 0;
85      D1(4,5) = 32/49/dx;
86      D1(4,6) = -4/49/dx;
87      D1(4,7) = 0;
88      D1(n,n) = -D1(1,1);
89      D1(n,n-1) = -D1(1,2);
90      D1(n,n-2) = -D1(1,3);
91      D1(n,n-3) = -D1(1,4);
92      D1(n,n-4) = -D1(1,5);
93      D1(n,n-5) = -D1(1,6);
94      D1(n-1,n) = -D1(2,1);
95      D1(n-1,n-1) = -D1(2,2);
96      D1(n-1,n-2) = -D1(2,3);
97      D1(n-1,n-3) = -D1(2,4);
98      D1(n-1,n-4) = -D1(2,5);
99      D1(n-1,n-5) = -D1(2,6);
100     D1(n-2,n) = -D1(3,1);
101     D1(n-2,n-1) = -D1(3,2);
102     D1(n-2,n-2) = -D1(3,3);
103     D1(n-2,n-3) = -D1(3,4);
104     D1(n-2,n-4) = -D1(3,5);
105     D1(n-2,n-5) = -D1(3,6);
106     D1(n-3,n) = -D1(4,1);
107     D1(n-3,n-1) = -D1(4,2);
108     D1(n-3,n-2) = -D1(4,3);
109     D1(n-3,n-3) = -D1(4,4);
110     D1(n-3,n-4) = -D1(4,5);
111     D1(n-3,n-5) = -D1(4,6);
112
113
114     %%%%%%%
115     D2 = 1/(dx^2)*spdiags([-1/12*e 4/3*e -5/2*e 4/3*e -1/12*e],-2:2,n,n);
116     D2(1,1) = 2/(dx^2);
117     D2(1,2) = -5/(dx^2);
118     D2(1,3) = 4/(dx^2);
119     D2(1,4) = -1/(dx^2);
120     D2(2,1) = 1/(dx^2);
121     D2(2,2) = -2/(dx^2);
122     D2(2,3) = 1/(dx^2);
123     D2(2,4) = 0;
124     D2(3,1) = -4/43/(dx^2);
125     D2(3,2) = 59/43/(dx^2);
126     D2(3,3) = -110/43/(dx^2);
127     D2(3,4) = 59/43/(dx^2);
128     D2(3,5) = -4/43/(dx^2);
129     D2(4,1) = -1/49/(dx^2);
130     D2(4,2) = 0;
131     D2(4,3) = 59/49/(dx^2);
132     D2(4,4) = -118/49/(dx^2);
133     D2(4,5) = 64/49/(dx^2);
134     D2(4,6) = -4/49/(dx^2);
135     D2(n,n) = D2(1,1);
136     D2(n,n-1) = D2(1,2);
137     D2(n,n-2) = D2(1,3);
138     D2(n,n-3) = D2(1,4);
139     D2(n-1,n) = D2(2,1);
140     D2(n-1,n-1) = D2(2,2);
141     D2(n-1,n-2) = D2(2,3);
142     D2(n-1,n-3) = D2(2,4);
143     D2(n-2,n) = D2(3,1);
144     D2(n-2,n-1) = D2(3,2);
145     D2(n-2,n-2) = D2(3,3);
146     D2(n-2,n-3) = D2(3,4);
147     D2(n-2,n-4) = D2(3,5);
148     D2(n-3,n) = D2(4,1);
149     D2(n-3,n-1) = D2(4,2);
150     D2(n-3,n-2) = D2(4,3);
151     D2(n-3,n-3) = D2(4,4);
152     D2(n-3,n-4) = D2(4,5);
153     D2(n-3,n-5) = D2(4,6);
154
155     H = dx*spdiags(e,0,n,n);
156     H(1,1) = dx*17/48;
157     H(2,2) = dx*59/48;
158     H(3,3) = dx*43/48;
159     H(4,4) = dx*49/48;
160     H(n,n) = H(1,1);
161     H(n-1,n-1) = H(2,2);
162     H(n-2,n-2) = H(3,3);
163     H(n-3,n-3) = H(4,4);
164
165     S = (1/dx)*spdiags(e,0,n,n);
166     S(1,1) = -11/6/dx;
167     S(1,2) = 3/dx;
168     S(1,3) = -3/2/dx;
169     S(1,4) = 1/3/dx;
170     S(n,n) = 11/6/dx;
171     S(n,n-1) = -3/dx;
```

```
172     S(n,n-2) = 3/2/dx;
173     S(n,n-3) = -1/3/dx;
174
175     BS = (1/dx)*spdiags(zeros(size(e)),0,n,n);
176     BS(1,1) = 11/6/dx;
177     BS(1,2) = -3/dx;
178     BS(1,3) = 3/2/dx;
179     BS(1,4) = -1/3/dx;
180     BS(n,n) = 11/6/dx;
181     BS(n,n-1) = -3/dx;
182     BS(n,n-2) = 3/2/dx;
183     BS(n,n-3) = -1/3/dx;
184
185 elseif order==6
186     e = ones(n,1);
187     %%%%%
188     D1 = (1/(dx))*spdiags([-1/60*e 3/20*e -3/4*e 0*e 3/4*e -3/20*e 1/60*e],-3:3,n,n);
189
190     D1(1,1) = -21600/13649/dx;
191     D1(1,2) = 104009/54596/dx;
192     D1(1,3) = 30443/81894/dx;
193     D1(1,4) = -33311/27298/dx;
194     D1(1,5) = 16863/27298/dx;
195     D1(1,6) = -15025/163788/dx;
196     D1(1,7) = 0;
197     D1(1,8) = 0;
198     D1(2,1) = -104009/240260/dx;
199     D1(2,2) = 0;
200     D1(2,3) = -311/72078/dx;
201     D1(2,4) = 20229/24026/dx;
202     D1(2,5) = -24337/48052/dx;
203     D1(2,6) = 36661/360390/dx;
204     D1(2,7) = 0;
205     D1(2,8) = 0;
206     D1(3,1) = -30443/162660/dx;
207     D1(3,2) = 311/32532/dx;
208     D1(3,3) = 0;
209     D1(3,4) = -11155/16266/dx;
210     D1(3,5) = 41287/32532/dx;
211     D1(3,6) = -21999/54220/dx;
212     D1(3,7) = 0;
213     D1(3,8) = 0;
214     D1(4,1) = 33311/107180/dx;
215     D1(4,2) = -20229/21436/dx;
216     D1(4,3) = 485/1398/dx;
217     D1(4,4) = 0;
218     D1(4,5) = 4147/21436/dx;
219     D1(4,6) = 25427/321540/dx;
220     D1(4,7) = 72/5359/dx;
221     D1(4,8) = 0;
222     D1(5,1) = -16863/78770/dx;
223     D1(5,2) = 24337/31508/dx;
224     D1(5,3) = -41287/47262/dx;
225     D1(5,4) = -4147/15754/dx;
226     D1(5,5) = 0;
227     D1(5,6) = 342523/472620/dx;
228     D1(5,7) = -1296/7877/dx;
229     D1(5,8) = 144/7877/dx;
230     D1(5,9) = 0;
231     D1(6,1) = 15025/525612/dx;
232     D1(6,2) = -36661/262806/dx;
233     D1(6,3) = 21999/87602/dx;
234     D1(6,4) = -25427/262806/dx;
235     D1(6,5) = -342523/525612/dx;
236     D1(6,6) = 0;
237     D1(6,7) = 32400/43801/dx;
238     D1(6,8) = -6480/43801/dx;
239     D1(6,9) = 720/43801/dx;
240     D1(6,10) = 0;
241     D1(n,n) = -D1(1,1);
242     D1(n,n-1) = -D1(1,2);
243     D1(n,n-2) = -D1(1,3);
244     D1(n,n-3) = -D1(1,4);
245     D1(n,n-4) = -D1(1,5);
246     D1(n,n-5) = -D1(1,6);
247     D1(n,n-6) = -D1(1,7);
248     D1(n,n-7) = -D1(1,8);
249     D1(n-1,n) = -D1(2,1);
250     D1(n-1,n-1) = -D1(2,2);
251     D1(n-1,n-2) = -D1(2,3);
252     D1(n-1,n-3) = -D1(2,4);
253     D1(n-1,n-4) = -D1(2,5);
254     D1(n-1,n-5) = -D1(2,6);
255     D1(n-1,n-6) = -D1(2,7);
256     D1(n-1,n-7) = -D1(2,8);
257     D1(n-2,n) = -D1(3,1);
258     D1(n-2,n-1) = -D1(3,2);
259     D1(n-2,n-2) = -D1(3,3);
260     D1(n-2,n-3) = -D1(3,4);
```

```
261     D1(n-2,n-4) = -D1(3,5);
262     D1(n-2,n-5) = -D1(3,6);
263     D1(n-2,n-6) = -D1(3,7);
264     D1(n-2,n-7) = -D1(3,8);
265     D1(n-3,n) = -D1(4,1);
266     D1(n-3,n-1) = -D1(4,2);
267     D1(n-3,n-2) = -D1(4,3);
268     D1(n-3,n-3) = -D1(4,4);
269     D1(n-3,n-4) = -D1(4,5);
270     D1(n-3,n-5) = -D1(4,6);
271     D1(n-3,n-6) = -D1(4,7);
272     D1(n-3,n-7) = -D1(4,8);
273     D1(n-4,n) = -D1(5,1);
274     D1(n-4,n-1) = -D1(5,2);
275     D1(n-4,n-2) = -D1(5,3);
276     D1(n-4,n-3) = -D1(5,4);
277     D1(n-4,n-4) = -D1(5,5);
278     D1(n-4,n-5) = -D1(5,6);
279     D1(n-4,n-6) = -D1(5,7);
280     D1(n-4,n-7) = -D1(5,8);
281     D1(n-4,n-8) = -D1(5,9);
282     D1(n-5,n) = -D1(6,1);
283     D1(n-5,n-1) = -D1(6,2);
284     D1(n-5,n-2) = -D1(6,3);
285     D1(n-5,n-3) = -D1(6,4);
286     D1(n-5,n-4) = -D1(6,5);
287     D1(n-5,n-5) = -D1(6,6);
288     D1(n-5,n-6) = -D1(6,7);
289     D1(n-5,n-7) = -D1(6,8);
290     D1(n-5,n-8) = -D1(6,9);
291     D1(n-5,n-9) = -D1(6,10);
292
293     %%%%%
294     D2 = (1/(dx^2))*spdiags([1/90*e -3/20*e 3/2*e -49/18*e 3/2*e -3/20*e 1/90*e],-3:3,n,n);
295
296     D2(1,1) = 114170/40947/(dx^2);
297     D2(1,2) = -438107/54596/(dx^2);
298     D2(1,3) = 336409/40947/(dx^2);
299     D2(1,4) = -276997/81894/(dx^2);
300     D2(1,5) = 3747/13649/(dx^2);
301     D2(1,6) = 21035/163788/(dx^2);
302     D2(1,7) = 0;
303     D2(1,8) = 0;
304     D2(2,1) = 6173/5860/(dx^2);
305     D2(2,2) = -2066/879/(dx^2);
306     D2(2,3) = 3283/1758/(dx^2);
307     D2(2,4) =-303/293/(dx^2);
308     D2(2,5) = 2111/3516/(dx^2);
309     D2(2,6) = -601/4395/(dx^2);
310     D2(2,7) = 0;
311     D2(2,8) = 0;
312     D2(3,1) = -52391/81330/(dx^2);
313     D2(3,2) = 134603/32532/(dx^2);
314     D2(3,3) = -21982/2711/(dx^2);
315     D2(3,4) = 112915/16266/(dx^2);
316     D2(3,5) = -46969/16266/(dx^2);
317     D2(3,6) = 30409/54220/(dx^2);
318     D2(3,7) = 0;
319     D2(3,8) = 0;
320     D2(4,1) = 68603/321540/(dx^2);
321     D2(4,2) = -12423/10718/(dx^2);
322     D2(4,3) = 112915/32154/(dx^2);
323     D2(4,4) = -75934/16077/(dx^2);
324     D2(4,5) = 53369/21436/(dx^2);
325     D2(4,6) = -54899/160770/(dx^2);
326     D2(4,7) = 48/5359/(dx^2);
327     D2(4,8) = 0;
328     D2(5,1) = -7053/39385/(dx^2);
329     D2(5,2) = 86551/94524/(dx^2);
330     D2(5,3) = -46969/23631/(dx^2);
331     D2(5,4) = 53369/15754/(dx^2);
332     D2(5,5) = -87904/23631/(dx^2);
333     D2(5,6) = 820271/472620/(dx^2);
334     D2(5,7) = -1296/7877/(dx^2);
335     D2(5,8) = 96/7877/(dx^2);
336     D2(5,9) = 0;
337     D2(6,1) = 21035/525612/(dx^2);
338     D2(6,2) = -24641/131403/(dx^2);
339     D2(6,3) = 30409/87602/(dx^2);
340     D2(6,4) = -54899/131403/(dx^2);
341     D2(6,5) = 820271/525612/(dx^2);
342     D2(6,6) = -117600/43801/(dx^2);
343     D2(6,7) = 64800/43801/(dx^2);
344     D2(6,8) = -6480/43801/(dx^2);
345     D2(6,9) = 480/43801/(dx^2);
346     D2(6,10) = 0;
347     D2(n,n) = D2(1,1);
348     D2(n,n-1) = D2(1,2);
349     D2(n,n-2) = D2(1,3);
```

```
350     D2(n,n-3) = D2(1,4);
351     D2(n,n-4) = D2(1,5);
352     D2(n,n-5) = D2(1,6);
353     D2(n,n-6) = D2(1,7);
354     D2(n,n-7) = D2(1,8);
355     D2(n-1,n) = D2(2,1);
356     D2(n-1,n-1) = D2(2,2);
357     D2(n-1,n-2) = D2(2,3);
358     D2(n-1,n-3) = D2(2,4);
359     D2(n-1,n-4) = D2(2,5);
360     D2(n-1,n-5) = D2(2,6);
361     D2(n-1,n-6) = D2(2,7);
362     D2(n-1,n-7) = D2(2,8);
363     D2(n-2,n) = D2(3,1);
364     D2(n-2,n-1) = D2(3,2);
365     D2(n-2,n-2) = D2(3,3);
366     D2(n-2,n-3) = D2(3,4);
367     D2(n-2,n-4) = D2(3,5);
368     D2(n-2,n-5) = D2(3,6);
369     D2(n-2,n-6) = D2(3,7);
370     D2(n-2,n-7) = D2(3,8);
371     D2(n-3,n) = D2(4,1);
372     D2(n-3,n-1) = D2(4,2);
373     D2(n-3,n-2) = D2(4,3);
374     D2(n-3,n-3) = D2(4,4);
375     D2(n-3,n-4) = D2(4,5);
376     D2(n-3,n-5) = D2(4,6);
377     D2(n-3,n-6) = D2(4,7);
378     D2(n-3,n-7) = D2(4,8);
379     D2(n-4,n) = D2(5,1);
380     D2(n-4,n-1) = D2(5,2);
381     D2(n-4,n-2) = D2(5,3);
382     D2(n-4,n-3) = D2(5,4);
383     D2(n-4,n-4) = D2(5,5);
384     D2(n-4,n-5) = D2(5,6);
385     D2(n-4,n-6) = D2(5,7);
386     D2(n-4,n-7) = D2(5,8);
387     D2(n-4,n-8) = D2(5,9);
388     D2(n-5,n) = D2(6,1);
389     D2(n-5,n-1) = D2(6,2);
390     D2(n-5,n-2) = D2(6,3);
391     D2(n-5,n-3) = D2(6,4);
392     D2(n-5,n-4) = D2(6,5);
393     D2(n-5,n-5) = D2(6,6);
394     D2(n-5,n-6) = D2(6,7);
395     D2(n-5,n-7) = D2(6,8);
396     D2(n-5,n-8) = D2(6,9);
397     D2(n-5,n-9) = D2(6,10);
398
399     H = dx*spdiags([e],0,n,n);
400
401     H(1,1) = dx*13649/43200;
402     H(2,2) = dx*12013/8640;
403     H(3,3) = dx*2711/4320;
404     H(4,4) = dx*5359/4320;
405     H(5,5) = dx*7877/8640;
406     H(6,6) = dx*43801/43200;
407     H(n,n) = H(1,1);
408     H(n-1,n-1) = H(2,2);
409     H(n-2,n-2) = H(3,3);
410     H(n-3,n-3) = H(4,4);
411     H(n-4,n-4) = H(5,5);
412     H(n-5,n-5) = H(6,6);
413
414     BS = (1/dx)*spdiags([zeros(size(e))],0,n,n);
415
416     BS(1,1) = 25/12/dx;
417     BS(1,2) = -4/dx;
418     BS(1,3) = 3/dx;
419     BS(1,4) = -4/3/dx;
420     BS(1,5) = 1/4/dx;
421     BS(n,n) = BS(1,1);
422     BS(n,n-1) = BS(1,2);
423     BS(n,n-2) = BS(1,3);
424     BS(n,n-3) = BS(1,4);
425     BS(n,n-4) = BS(1,5);
426
427     S = (1/dx)*spdiags([e],0,n,n);
428
429     S(1,1) = -25/12/dx;
430     S(1,2) = 4/dx;
431     S(1,3) = -3/dx;
432     S(1,4) = 4/3/dx;
433     S(1,5) = -1/4/dx;
434     S(n,n) = BS(1,1);
435     S(n,n-1) = BS(1,2);
436     S(n,n-2) = BS(1,3);
437     S(n,n-3) = BS(1,4);
438     S(n,n-4) = BS(1,5);
```

```
439 elseif order==8
440     e = ones(n,1);
441
442     D = (1/(dx^2))*spdiags([-1/560*e 8/315*e -1/5*e 8/5*e -205/72*e 8/5*e -1/5*e 8/315*e -1/560*e],-4:4,n,n);
443     % eighth order standard central stencil
444
445     D(1,1) = 4870382994799/1358976868290/(dx^2);
446     D(1,2) = -893640087518/75498714905/(dx^2);
447     D(1,3) = 926594825119/60398971924/(dx^2);
448     D(1,4) = -1315109406200/135897686829/(dx^2);
449     D(1,5) = 39126983272/15099742981/(dx^2);
450     D(1,6) = 12344491342/75498714905/(dx^2);
451     D(1,7) = -451560522577/2717953736580/(dx^2);
452     D(1,8) = 0;
453     D(1,9) = 0;
454     D(1,10) = 0;
455     D(1,11) = 0;
456     D(1,12) = 0;
457     D(2,1) = 333806012194/390619153855/(dx^2);
458     D(2,2) = -154646272029/111605472530/(dx^2);
459     D(2,3) = 1168338040/33481641759/(dx^2);
460     D(2,4) = 82699112501/133926567036/(dx^2);
461     D(2,5) = -171562838/11160547253/(dx^2);
462     D(2,6) = -28244698346/167408208795/(dx^2);
463     D(2,7) = 11904122576/167408208795/(dx^2);
464     D(2,8) = -2598164715/31249532308 4/(dx^2);
465     D(2,9) = 0;
466     D(2,10) = 0;
467     D(2,11) = 0;
468     D(2,12) = 0;
469     D(3,1) = 7838984095/52731029988/(dx^2);
470     D(3,2) = 1168338040/5649753213/(dx^2);
471     D(3,3) = -88747895/144865467/(dx^2);
472     D(3,4) = 423587231/627750357/(dx^2);
473     D(3,5) = -43205598281/22599012852/(dx^2);
474     D(3,6) = 4876378562/1883251071/(dx^2);
475     D(3,7) = -5124426509/3766502142/(dx^2);
476     D(3,8) = 10496900965/39548272491/(dx^2);
477     D(3,9) = 0;
478     D(3,10) = 0;
479     D(3,11) = 0;
480     D(3,12) = 0;
481     D(4,1) = -94978241528/828644350023/(dx^2);
482     D(4,2) = 82699112501/157837019052/(dx^2);
483     D(4,3) = 1270761693/13153084921/(dx^2);
484     D(4,4) = -167389605005/118377764289/(dx^2);
485     D(4,5) = 48242560214/39459254763/(dx^2);
486     D(4,6) = -31673996013/52612339684/(dx^2);
487     D(4,7) = 43556319241/118377764289/(dx^2);
488     D(4,8) = -44430275135/552429566682/(dx^2);
489     D(4,9) = 0;
490     D(4,10) = 0;
491     D(4,11) = 0;
492     D(4,12) = 0;
493     D(5,1) = 1455067816/21132528431/(dx^2);
494     D(5,2) = -171562838/3018932633/(dx^2);
495     D(5,3) = -43205598281/36227191596/(dx^2);
496     D(5,4) = 48242560214/90567978 99/(dx^2);
497     D(5,5) = -52276055645/6037865266/(dx^2);
498     D(5,6) = 57521587238/90567978 99/(dx^2);
499     D(5,7) = -80321706377/36227191596/(dx^2);
500     D(5,8) = 8078087158/21132528431/(dx^2);
501     D(5,9) = -1296/299527/(dx^2);
502     D(5,10) = 0;
503     D(5,11) = 0;
504     D(5,12) = 0;
505     D(6,1) = 10881504334/327321118845/(dx^2);
506     D(6,2) = -28244698346/140280479505/(dx^2);
507     D(6,3) = 4876378562/9352031967/(dx^2);
508     D(6,4) = -10557998671/12469375956/(dx^2);
509     D(6,5) = 57521587238/28056095901/(dx^2);
510     D(6,6) = -278531401019/93520319670/(dx^2);
511     D(6,7) = 73790130002/46760159835/(dx^2);
512     D(6,8) = -137529995233/785570685228/(dx^2);
513     D(6,9) = 2048/103097/(dx^2);
514     D(6,10) = -144/103097/(dx^2);
515     D(6,11) = 0;
516     D(6,12) = 0;
517     D(7,1) = -135555328849/8509847458140/(dx^2);
518     D(7,2) = 11904122576/101307707835/(dx^2);
519     D(7,3) = -5124426509/13507694378/(dx^2);
520     D(7,4) = 43556319241/60784624701/(dx^2);
521     D(7,5) = -80321706377/81046166268/(dx^2);
522     D(7,6) = 73790130002/33769235945/(dx^2);
523     D(7,7) = -950494905688/303923123505/(dx^2);
524     D(7,8) = 239073018673/141830790969/(dx^2);
525     D(7,9) = -145152/670091/(dx^2);
526     D(7,10) = 18432/670091/(dx^2);
527     D(7,11) = -1296/670091/(dx^2);
```

```
528     D(7,12) = 0;
529     D(8,1) = 0;
530     D(8,2) = -2598164715/206729925524/(dx^2);
531     D(8,3) = 10496900965/155047444143/(dx^2);
532     D(8,4) = -44430275135/310094888286/(dx^2);
533     D(8,5) = 425162482/2720130599/(dx^2);
534     D(8,6) = -137529995233/620189776572/(dx^2);
535     D(8,7) = 239073018673/155047444143/(dx^2);
536     D(8,8) = -144648000000/51682481381/(dx^2);
537     D(8,9) = 8128512/5127739/(dx^2);
538     D(8,10) = -1016064/5127739/(dx^2);
539     D(8,11) = 129024/5127739/(dx^2);
540     D(8,12) = -9072/5127739/(dx^2);
541
542     D(n,n) = D(1,1);
543     D(n,n-1) = D(1,2);
544     D(n,n-2) = D(1,3);
545     D(n,n-3) = D(1,4);
546     D(n,n-4) = D(1,5);
547     D(n,n-5) = D(1,6);
548     D(n,n-6) = D(1,7);
549     D(n,n-7) = D(1,8);
550     D(n,n-8) = D(1,9);
551     D(n,n-9) = D(1,10);
552     D(n,n-10) = D(1,11);
553     D(n,n-11) = D(1,12);
554     D(n-1,n) = D(2,1);
555     D(n-1,n-1) = D(2,2);
556     D(n-1,n-2) = D(2,3);
557     D(n-1,n-3) = D(2,4);
558     D(n-1,n-4) = D(2,5);
559     D(n-1,n-5) = D(2,6);
560     D(n-1,n-6) = D(2,7);
561     D(n-1,n-7) = D(2,8);
562     D(n-1,n-8) = D(2,9);
563     D(n-1,n-9) = D(2,10);
564     D(n-1,n-10) = D(2,11);
565     D(n-1,n-11) = D(2,12);
566     D(n-2,n) = D(3,1);
567     D(n-2,n-1) = D(3,2);
568     D(n-2,n-2) = D(3,3);
569     D(n-2,n-3) = D(3,4);
570     D(n-2,n-4) = D(3,5);
571     D(n-2,n-5) = D(3,6);
572     D(n-2,n-6) = D(3,7);
573     D(n-2,n-7) = D(3,8);
574     D(n-2,n-8) = D(3,9);
575     D(n-2,n-9) = D(3,10);
576     D(n-2,n-10) = D(3,11);
577     D(n-2,n-11) = D(3,12);
578     D(n-3,n) = D(4,1);
579     D(n-3,n-1) = D(4,2);
580     D(n-3,n-2) = D(4,3);
581     D(n-3,n-3) = D(4,4);
582     D(n-3,n-4) = D(4,5);
583     D(n-3,n-5) = D(4,6);
584     D(n-3,n-6) = D(4,7);
585     D(n-3,n-7) = D(4,8);
586     D(n-3,n-8) = D(4,9);
587     D(n-3,n-9) = D(4,10);
588     D(n-3,n-10) = D(4,11);
589     D(n-3,n-11) = D(4,12);
590     D(n-4,n) = D(5,1);
591     D(n-4,n-1) = D(5,2);
592     D(n-4,n-2) = D(5,3);
593     D(n-4,n-3) = D(5,4);
594     D(n-4,n-4) = D(5,5);
595     D(n-4,n-5) = D(5,6);
596     D(n-4,n-6) = D(5,7);
597     D(n-4,n-7) = D(5,8);
598     D(n-4,n-8) = D(5,9);
599     D(n-4,n-9) = D(5,10);
600     D(n-4,n-10) = D(5,11);
601     D(n-4,n-11) = D(5,12);
602     D(n-5,n) = D(6,1);
603     D(n-5,n-1) = D(6,2);
604     D(n-5,n-2) = D(6,3);
605     D(n-5,n-3) = D(6,4);
606     D(n-5,n-4) = D(6,5);
607     D(n-5,n-5) = D(6,6);
608     D(n-5,n-6) = D(6,7);
609     D(n-5,n-7) = D(6,8);
610     D(n-5,n-8) = D(6,9);
611     D(n-5,n-9) = D(6,10);
612     D(n-5,n-10) = D(6,11);
613     D(n-5,n-11) = D(6,12);
614     D(n-6,n) = D(7,1);
615     D(n-6,n-1) = D(7,2);
616     D(n-6,n-2) = D(7,3);
```

```
617     D(n-6,n-3) = D(7,4);
618     D(n-6,n-4) = D(7,5);
619     D(n-6,n-5) = D(7,6);
620     D(n-6,n-6) = D(7,7);
621     D(n-6,n-7) = D(7,8);
622     D(n-6,n-8) = D(7,9);
623     D(n-6,n-9) = D(7,10);
624     D(n-6,n-10) = D(7,11);
625     D(n-6,n-11) = D(7,12);
626     D(n-7,n) = D(8,1);
627     D(n-7,n-1) = D(8,2);
628     D(n-7,n-2) = D(8,3);
629     D(n-7,n-3) = D(8,4);
630     D(n-7,n-4) = D(8,5);
631     D(n-7,n-5) = D(8,6);
632     D(n-7,n-6) = D(8,7);
633     D(n-7,n-7) = D(8,8);
634     D(n-7,n-8) = D(8,9);
635     D(n-7,n-9) = D(8,10);
636     D(n-7,n-10) = D(8,11);
637     D(n-7,n-11) = D(8,12);
638
639
640     H = dx*spdiags([e],0,n,n);
641
642     H(1,1) = dx*1498139/5080320;
643     H(2,2) = dx*1107307/725760;
644     H(3,3) = dx*20761/80640;
645     H(4,4) = dx*1304999/725760;
646     H(5,5) = dx*299527/725760;
647     H(6,6) = dx*103097/80640;
648     H(7,7) = dx*670091/725760;
649     H(8,8) = dx*5127739/5080320;
650     H(n,n) = H(1,1);
651     H(n-1,n-1) = H(2,2);
652     H(n-2,n-2) = H(3,3);
653     H(n-3,n-3) = H(4,4);
654     H(n-4,n-4) = H(5,5);
655     H(n-5,n-5) = H(6,6);
656     H(n-6,n-6) = H(7,7);
657     H(n-7,n-7) = H(8,8);
658
659     BS = (1/dx)*spdiags([zeros(size(e))],0,n,n);
660
661     BS(1,1) = 4723/2100/dx;
662     BS(1,2) = -839/175/dx;
663     BS(1,3) = 157/35/dx;
664     BS(1,4) = -278/105/dx;
665     BS(1,5) = 103/140/dx;
666     BS(1,6) = 1/175/dx;
667     BS(1,7) = -6/175/dx;
668     BS(n,n) = BS(1,1);
669     BS(n,n-1) = BS(1,2);
670     BS(n,n-2) = BS(1,3);
671     BS(n,n-3) = BS(1,4);
672     BS(n,n-4) = BS(1,5);
673     BS(n,n-5) = BS(1,6);
674     BS(n,n-6) = BS(1,7);
675
676
677     S = (1/dx)*spdiags([e],0,n,n);
678
679     S(1,1) = -4723/2100/dx;
680     S(1,2) = 839/175/dx;
681     S(1,3) = -157/35/dx;
682     S(1,4) = 278/105/dx;
683     S(1,5) = -103/140/dx;
684     S(1,6) = -1/175/dx;
685     S(1,7) = 6/175/dx;
686     S(n,n) = BS(1,1);
687     S(n,n-1) = BS(1,2);
688     S(n,n-2) = BS(1,3);
689     S(n,n-3) = BS(1,4);
690     S(n,n-4) = BS(1,5);
691     S(n,n-5) = BS(1,6);
692     S(n,n-6) = BS(1,7);
693
694 else
695     disp('Only order 2, 4, 6 or 8 implemented here.')
696 end
```

C.1.2.2 mu_SG_lognormal.m

```
function [B] = mu_SG_lognormal(c1,c2,P)

% Compute the stochastic Galerkin viscosity matrix B with normalized Hermite
% polynomials

% Indata:
% c1, c2 - Scaling parameters of shifted lognormal distribution
% P - Number of gPC terms to be retained

% Outdata:
% B - Viscosity matrix, [B]_{ij} = sum_{k}^{2P}<psi_i psi_j psi_k> mu_k

% For the 1D case, use twice as many basis functions as for the variables
% (see Proposition 1 in Chapter 5)

P2 = 2*P;
C = hermite_chaos(P2-1);

tol = 30;

% Recursively generate Hermite polynomials
basis_fun = cell(1,P2);
basis_fun{1} = @(xi) xi.^0;
basis_fun{2} = @(xi) xi;
for k=3:P2
    basis_fun{k} = @(xi) 1/sqrt(k-1)*xi.*basis_fun{k-1}(xi)-sqrt((k-2)/(k-1))*basis_fun{k-2}(xi);
end

visc_fun = @(x) c1+c2*exp(x);

B = zeros(P2);

for k=1:P2
    integ = @(x) 1/sqrt(2*pi).*exp(-x.^2/2).*basis_fun{k}(x).*visc_fun(x);
    visc_hc(k,1) = quadgk(integ,-tol,tol);
    B = B+C(:,:,k)*visc_hc(k,1);
end

B = B(1:P,1:P);
```

C.1.3 Boundary Treatment

C.1.3.1 bdy_cond_Dirichlet.m

```
function [g0] = bdy_cond_Dirichlet(rho_0,c1,c2,v,t,t0,x,x0,m,P)

% Generate left boundary data for Legendre polynomials and uniform viscosity

% Indata:
% rho_0 - Solution scaling parameter (assumed deterministic)
% c1,c2 - Scaling parameters of uniform viscosity
% v - Advective velocity
% t - Time
% t0 - Initial time
% x - Vector of spatial grid points
% x0 - Initial pulse location
% m - Number of spatial grid points
% P - Number of gPC coefficients to be computed

% Outdata:
% g0 - Dirichlet data, left boundary



u_init = zeros(m*P,1);

%Generate normalized Legendre polynomials recursively

basis_fun = cell(1,P);
basis_fun{1} = @(xi) xi.^0;
basis_fun{2} = @(xi) sqrt(3)*xi;
for k=3:P
    basis_fun{k} = @(xi) (sqrt(2*k-3)/(k-1)*xi.*basis_fun{k-1}(xi)-(k-2)/((k-1)*sqrt(2*k-5))*basis_fun{k-2}(xi
        ))*sqrt(2*k-1);
end
```

```
32
33 mu_fun = @(xi) c1+c2*xi;
34
35 g0 = zeros(m*P,1);
36
37 for k=1:P
38     integ = @(xi) 0.5.*basis_fun{k}(xi).*rho_0.*(1-erf((x(1)-(x0+v*(t+t0)))./sqrt(4*mu_fun(xi)*(t+t0))));
39     g0(k,1) = quad(integ,-1,1);
40 end
```

C.1.3.2 bdy_cond_Legendre.m

```
1  function [g0 g1] = bdy_cond_Legendre(rho_0,c1,c2,v,t,t0,x,x0,m,P)
2
3  % Compute Dirichlet data for the left boundary and Neumann data for the
4  % right boundary, assuming Legendre polynomials representation
5
6  % Indata:
7  % rho_0 - Solution scaling parameter (assumed deterministic)
8  % c1,c2 - Scaling parameters of uniform viscosity
9  % v - Advective velocity
10 % t - Time
11 % t0 - Initial time
12 % x - Vector of spatial grid points
13 % x0 - Initial pulse location
14 % m - Number of spatial grid points
15 % P - Number of gPC coefficients to be computed
16
17 % Outdata:
18 % g0 - Dirichlet data, left boundary
19 % g1 - Neumann data, right boundary
20
21
22 u_init = zeros(m*P,1);
23
24 %Legendre polynomials
25
26 basis_fun = cell(1,P);
27 basis_fun{1} = @(xi) xi.^0;
28 basis_fun{2} = @(xi) sqrt(3)*xi;
29 for k=3:P
30     basis_fun{k} = @(xi) (sqrt(2*k-3)/(k-1)*xi.*basis_fun{k-1}(xi)-(k-2)/((k-1)*sqrt(2*k-5))*basis_fun{k-2}(xi
        ))*sqrt(2*k-1);
31 end
32
33 mu_fun = @(xi) c1+c2*xi;
34
35 g0 = zeros(m*P,1);
36 g1 = zeros(m*P,1);
37
38 for k=1:P
39     integ_0 = @(xi) 0.5*basis_fun{k}(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp(-(x(1)-(x0+v*(t+t0))).^2.
        /(4*mu_fun(xi)*(t+t0)));
40     g0(k,1) = quad(integ_0,-1,1);
41     integ_1 = @(xi) 0.5*basis_fun{k}(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp(-(x(end)-(x0+v*(t+t0))).^2.
        /(4*mu_fun(xi)*(t+t0))).*(-(x(end)-(x0+v*(t+t0)))./(2*mu_fun(xi)*(t+t0)));
42     g1((m-1)*P+k,1) = quad(integ_1,-1,1);
43 end
```

C.1.3.3 bdy_cond_Hermite.m

```
1
2  function [g0 g1] = bdy_cond_Hermite(rho_0,c1,c2,v,t,t0,x,x0,m,P)
3
4  % Generate boundary data for Hermite polynomials and shifted lognormal viscosity
5
6  % Indata:
7  % rho_0 - Solution scaling parameter (assumed deterministic)
8  % c1,c2 - Scaling parameters of shifted lognormal viscosity
9  % v - Advective velocity
10 % t - Time
11 % t0 - Initial time
12 % x - Vector of spatial grid points
13 % x0 - Initial pulse location
14 % m - Number of spatial grid points
15 % P - Number of gPC coefficients to be computed
16
```

```
% Outdata:
% g0 - Dirichlet data, left boundary
% g1 - Neumann data, right boundary


tol = 15; % Replace infinite integration limit by sufficiently large real number

g0 = zeros(m*P,1);
g1 = zeros(m*P,1);

% Recursively generate Hermite polynomials
basis_fun = cell(1,P);
basis_fun{1} = @(xi) xi.^0;
basis_fun{2} = @(xi) xi;
for k=3:P
    basis_fun{k} = @(xi) 1/sqrt(k-1)*xi.*basis_fun{k-1}(xi)-sqrt((k-2)/(k-1))*basis_fun{k-2}(xi);
end

mu_fun = @(xi) c1+c2*exp(xi);

% Compute the gPC coefficients with numerical integration
for k=1:P
    integ = @(xi) 1/sqrt(2*pi).*exp(-xi.^2/2).*basis_fun{k}(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp(-(x
        (1)-(x0+v*(t+t0))).^2./(4*mu_fun(xi).*(t+t0)));
    g0(k,1) = quad(integ,-tol,tol);

    integ = @(xi) -((x(end)-(x0+v*(t+t0)))./(2*mu_fun(xi)*(t+t0))).*(1/sqrt(2*pi).*exp(-xi.^2/2).*basis_fun{k
        }(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp(-(x(end)-(x0+v*(t+t0))).^2./(4*mu_fun(xi).*(t+t0))));
    g1((m-1)*P+k,1) = quad(integ,-tol,tol);
end
```

C.1.4 Reference Solution

C.1.4.1 init_Hermite.m

```

function [u_init] = init_Hermite(rho_0,c1,c2,v,t,t0,x,x0,m,P)

% Generate initial function for Hermite polynomials and lognormal viscosity

% Indata:
% rho_0 - Solution scaling parameter (assumed deterministic)
% c1,c2 - Scaling parameters of lognormal viscosity
% v - Advective velocity
% t - Time
% t0 - Initial time
% x - Vector of spatial grid points
% x0 - Initial shock location
% m - Number of spatial grid points
% P - Number of gPC coefficients to be computed

% Outdata:
% u_init - gPC coefficients of the initial function evaluated at the spatial grid points


u_init = zeros(m*P,1);
tol = 20; % Replace integration limits of infinity by some sufficiently large number

% Generate normalized Hermite polynomials recursively

basis_fun = cell(1,P);
basis_fun{1} = @(xi) xi.^0;
basis_fun{2} = @(xi) xi;
for k=3:P
    basis_fun{k} = @(xi) 1/sqrt(k-1)*xi.*basis_fun{k-1}(xi)-sqrt((k-2)/(k-1))*basis_fun{k-2}(xi);
end

% Viscocity with shifted lognormal distribution
mu_fun = @(xi) c1+c2*exp(xi);

for k=1:P
    for j=1:m
        integ = @(xi) 1/sqrt(2*pi).*exp(-xi.^2/2).*basis_fun{k}(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp
        (-(x(j)-(x0+v*(t+t0))).^2./(4*mu_fun(xi).*(t+t0)));
        u_init((j-1)*P+k,1) = quadgk(integ,-tol,tol); % Integration over the real line replaced with finite
        interval
    end
end
```

C.1.4.2 init_Legendre.m

```
function [u_init] = init_Legendre(rho_0,c1,c2,v,t,t0,x,x0,m,P)

% Generate initial function for Hermite polynomials and uniformly distributed viscosity

% Indata:
% rho_0 - Solution scaling parameter (assumed deterministic)
% c1,c2 - Scaling parameters of uniformly distributed viscosity
% v - Advective velocity
% t - Time
% t0 - Initial time
% x - Vector of spatial grid points
% x0 - Initial pulse location
% m - Number of spatial grid points
% P - Number of gPC coefficients to be computed

% Outdata:
% u_init - gPC coefficients of the initial function evaluated at the spatial grid points


u_init = zeros(m*P,1);

%Generate normalized Legendre polynomials recursively

basis_fun = cell(1,P);
basis_fun{1} = @(xi) xi.^0;
basis_fun{2} = @(xi) sqrt(3)*xi;
for k=3:P
    basis_fun{k} = @(xi) (sqrt(2*k-3)/(k-1)*xi.*basis_fun{k-1}(xi)-(k-2)/((k-1)*sqrt(2*k-5))*basis_fun{k-2}(xi
        ))*sqrt(2*k-1);
end


mu_fun = @(xi) c1+c2*xi;

for k=1:P
    for j=1:m
        integ = @(xi) 0.5.*basis_fun{k}(xi).*rho_0./sqrt(4*pi*mu_fun(xi)*(t+t0)).*exp(-(x(j)-(x0+v*(t+t0))).
        ^2./(4*mu_fun(xi).*(t+t0)));
        u_init((j-1)*P+k,1) = quad(integ,-1,1);
    end
end
```

C.1.4.3 statistics_adv_diff_lognorm.m

```
function [u_mean u_var] = statistics_adv_diff_lognorm(m,t,x,x0,c1,c2,rho_0,t0,v)

% Compute mean and variance for the solution assuming lognormal viscosity

% Indata:
% m - Number of spatial grid points
% t - Time
% x - Vector of spatial grid points
% x0 - Initial pulse location
% c1,c2 - Scaling parameters of lognormal viscosity
% rho_0 - Solution scaling parameter (assumed deterministic)
% t0 - Initial time
% v - Advective velocity

% Outdata:
% u_mean - Mean solution evaluated on the spatial grid
% u_var - Variance of the solution evaluated on the spatial grid


mu0 = @(xi) c1+c2*exp(xi); % Shifted lognormal model for the viscosity (xi standard Gaussian)
tol = 20; % Threshold for replacement of infinite integration limits

% For each spatial grid point, compute mean and variance
for j=1:m
    u = @(xi) rho_0./(4*pi*mu0(xi)*(t+t0)).^0.5.*exp(-(x(j)-(x0+v*(t+t0))).^2./(4*mu0(xi)*(t+t0)));
    u_mean(j,1) = quad(@(xi) exp(-xi.^2/2)/sqrt(2*pi).*u(xi),-tol,tol);
    u_var(j,1) = quad(@(xi) exp(-xi.^2/2)/sqrt(2*pi).*(u(xi).^2-u_mean(j,1).^2),-tol,tol);
end
```

C.1.4.4 statistics_adv_diff_uniform_mu.m

```
function [u_mean u_var] = statistics_adv_diff_uniform_mu(m,t,x,x0,c1,c2,rho_0,t0,v)

% Compute mean and variance for the solution assuming uniformly distributed viscosity

% Indata:
% m - Number of spatial grid points
% t - Time
% x - Vector of spatial grid points
% x0 - Initial pulse location
% c1,c2 - Scaling parameters of uniformly distributed viscosity
% rho_0 - Solution scaling parameter (assumed deterministic)
% t0 - Initial time
% v - Advective velocity

% Outdata:
% u_mean - Mean solution evaluated on the spatial grid
% u_var - Variance of the solution evaluated on the spatial grid


mu0 = @(xi) c1+c2*xi;

% For each spatial grid point, compute mean and variance
for j=1:m
    u = @(xi) rho_0./(4*pi*mu0(xi)*(t+t0)).^0.5.*exp(-(x(j)-(x0+v*(t+t0))).^2./(4*mu0(xi)*(t+t0)));
    u_mean(j,1) = quad(@(xi) 0.5*u(xi),-1,1);
    u_var(j,1) = quad(@(xi) 0.5*(u(xi).^2-u_mean(j,1).^2),-1,1);
end
```

C.2 Non-linear Transport

C.2.1 Main Code

C.2.1.1 burgers_main.m

```


% Stochastic Galerkin formulation of Burgers' equation, one spatial dimension
% Finite difference discretization in space (summation by parts + SAT) for SG
% Weak imposition of boundary conditions
% Runge-Kutta (4th order accurate) in time
% The assigned fourth and second order dissipation operators => one-sided diff. op.


clear all;

left = 0; right = 1; % Spatial end points
p = 4;% Number of polynomial chaos terms (order+1)
m = 100; % Discretization points in space
T = 0.7; % End time of simulation
% Solution parameters
mean_left = 0.9;
mean_right = -1.1;
sig_h_left = 0.3; % Assumed to be positive for correct analytical solution
% Assume same standard deviation everywhere for comparison with analytical
% solution
x0 = 0.5; % Initial location of shock, x0 in [left, right]


dx = (right-left)/(m-1);
dt = 0.05*dx;

sig_h_right = sig_h_left;
sig_h = sig_h_left;

x = linspace(left,right,m);

% Analytical solution of the truncated problem for p=1,2, else the exact
% coefficients of the infinte expansion problem
if p == 1
    u_ref = exact_solution_determ(mean_left,mean_right,T,m,left,right,x0,x);
elseif p == 2
    u_ref = exact_solution_2x2(mean_left,mean_right,sig_h,T,m,left,right,x0,x);
else
    u_ref = exact_solution_p_inf(mean_left,mean_right,sig_h,T,m,left,right,x0,x,p);
end
```

```
42
43
44
45 I_p = eye(p);
46 I_m = eye(m);
47 old = 1;
48 new = 2;
49 u = zeros(p*m,2);
50 order = 4; % Order of accuracy of SBP operators - SHOULD MATCH DISSIPATION OPERATORS
51
52 %Compute inner triple products
53 % Here we assume Gaussian distribution of the states, which causes no
54 % problem in terms of numerical stability
55 [C] = Hermite_chaos(p-1);
56
57 %Difference operators (D1 1st derivative)
58 [D1 D2,BS,S,P] = SBP_operators(m,dx,order);
59
60 % Invert the norm matrix P (assuming it is diagonal)
61 for j=1:m
62     P_temp(j,j) = 1/P(j,j);
63 end
64 P_inv=sparse(P_temp);
65
66 kron_Dx = kron(D1,I_p);
67
68 %Initialization
69 u(:,old) = initial_conditions(m,p,C,x0,mean_left,sig_h_left,mean_right,sig_h_right,left,right);
70
71 %Compute force terms
72 E_1 = sparse(zeros(m));
73 E_1(1,1) = 1;
74 E_m = sparse(zeros(m));
75 E_m(m,m)=1;
76 [Sig_left,Sig_right] = penalty(p,u(1:p,1),u(p*(m-1)+1:p*m,1),C);
77 force_left = kron(I_m,I_p)*kron(P_inv,I_p)*kron(E_1,Sig_left);
78 force_right = kron(I_m,I_p)*kron(P_inv,I_p)*kron(E_m,Sig_right);
79
80 % Time dependent boundary conditions
81 t_tol = 1e-5; % Tolerance to avoid division with zero at t=0
82 if p == 1
83     [g_left g_right] = boundary_cond_determ(mean_left,mean_right,t_tol,x0,left,right,m);
84 elseif p == 2
85     [g_left g_right] = boundary_cond_2x2(mean_left,mean_right,sig_h,t_tol,x0,left,right,m);
86 else
87     [g_left g_right] = boundary_cond_p_inf(mean_left,mean_right,sig_h,t_tol,x0,left,right,p,m);
88 end
89
90 %Iterate over time with fourth order Runge-Kutta
91 t=0;
92 while t<T
93    if T-t<dt
94       dt = T-t; % Make sure we end at t=T
95    end
96     t = t+dt;% Update the time
97
98     % Compute system eigenvalues for dissipation. May be replaced by the
99     % approximate expression in Chapter 6.
100
101     max_ev = 0;
102     for i=0:length(u)/p-1
103         u_part = u(p*i+1:p*(i+1),1);
104         eig_temp = max(abs(eig(I_p*A_matrix(u_part,p,C))));
105         if eig_temp>max_ev
106             max_ev = eig_temp;
107         end
108     end
109
110     % These dissipation operators are chosen for the 4th order operators -
111     % need to be adapted for different order of accuracy
112
113     diss_2nd_ord = dissipation_2nd_der(m,p,P_inv,I_p,2*max_ev/(6*dx),dx);
114     diss_4th_ord = dissipation_4th_der(m,p,P_inv,I_p,2*max_ev/(24*dx),dx);
115
116     % Time-stepping by 4th order Runge-Kutta
117
118     F1 = flux_func(u(:,old),p,C);
119     k1 = dt*(-kron_Dx*F1+(diss_4th_ord+diss_2nd_ord+force_left+force_right)*u(:,old)-force_left*g_left-
        force_right*g_right);
120
121     F2 = flux_func(u(:,old)+k1/2,p,C);
122     k2 = dt*(-kron_Dx*F2+(diss_4th_ord+diss_2nd_ord+force_left+force_right)*(u(:,old)+k1/2)-force_left*g_left-
        force_right*g_right);
123
124     F3 = flux_func(u(:,old)+k2/2,p,C);
125     k3 = dt*(-kron_Dx*F3+(diss_4th_ord+diss_2nd_ord+force_left+force_right)*(u(:,old)+k2/2)-force_left*g_left-
        force_right*g_right);
126
127     F4 = flux_func(u(:,old)+k3,p,C);
```

```
128     k4 = dt*(-kron_Dx*F4+(diss_4th_ord+diss_2nd_ord+force_left+force_right)*(u(:,old)+k3)-force_left*g_left-
            force_right*g_right);
129
130     u(:,new) = u(:,old)+1/6*(k1+2*k2+2*k3+k4);
131     u(:,old) = u(:,new);
132
133     % Update penalties for imposition of boundary conditions
134     [Sig_left, Sig_right] = penalty(p,u(1:p,1),u(p*(m-1)+1:p*m,1),C);
135     force_left = kron(I_m,I_p)*kron(P_inv,I_p)*kron(E_1,Sig_left);
136     force_right = kron(I_m,I_p)*kron(P_inv,I_p)*kron(E_m,Sig_right);
137
138     % Update the time dependent boundary conditions
139
140     if p == 1
141         [g_left g_right] = boundary_cond_determ(mean_left,mean_right,t,x0,left,right,m);
142     elseif p == 2
143         [g_left g_right] = boundary_cond_2x2(mean_left,mean_right,sig_h,t,x0,left,right,m);
144     else
145         [g_left g_right] = boundary_cond_p_inf(mean_left,mean_right,sig_h,t,x0,left,right,p,m);
146     end
147
148     % Plot the coefficients
149
150     for k=1:p
151         u_plot(:,k) = u(k:p:end,new);
152     end
153     plot(x,u_plot,'-*');
154     title(['gPC coefficients, t=' num2str(t,'%.2f')])
155     leg_strs = {};
156     for k=1:p
157         legstrs{k} = ['gPC coe. ' num2str(k-1)];
158     end
159     legend(legstrs);
160     drawnow;
161 end
162
163 u=u(:,new);
164
165 for i=1:m
166     u_var(i,1) = sum(u((i-1)*p+2:i*p,old).^2);
167 end
168
169
170 % Plot the numerical and analytical solution for the truncated 2x2 problem
171
172 if p == 2
173     subplot(1,2,1);
174     plot(x,u(1:p:end),'-','LineWidth',2,'Color','r');
175     hold on;
176     plot(x,u_ref(:,1),'--','LineWidth',2,'Color','b')
177     legend('Numerical','Analytical, 2x2');
178     title('u_0','fontsize',14,'fontweight','b');
179
180     subplot(1,2,2);
181     plot(x,u(2:p:(m-1)*p+2),'-','LineWidth',2,'Color','r');
182     hold on;
183     plot(x,u_ref(:,2),'--','LineWidth',2,'Color','b')
184     legend('Numerical','Analytical, 2x2');
185     title('u_1','fontsize',14,'fontweight','b');
186 end
187
188 % Plot the numerical approximation of order p and the analytical
189 % coefficients
190
191 if p ~= 2
192     for k=1:p
193         figure;
194         plot(x,u(k:p:end),'-r',x,u_ref(:,k),'--b','LineWidth',2)
195         legend('Numerical','Analytical');%,'Analytical expected value');
196         title(['Solution gPC coefficient ' num2str(k-1)],'fontsize',14,'fontweight','b');
197     end
198 end
```

C.2.1.2 Hermite_chaos.m

```
1
2 function [C] = Hermite_chaos(n)
3
4 % Compute hermite chaos parameters
5
6 % Indata:
7 % n - Order of gPC
8
9 % Outdata:
10 % C - Three term inner products C(i,j,k) = E[Phi_i Phi_j Phi_k]
```

```
11
12
13 C = zeros(n+1,n+1,n+1);
14 for i = 0:n
15     for j = 0:n
16         for k = 0:n
17             s = (i+k+j) / 2;
18             if rem(i+k+j,2) == 1 || i > s || j > s || k > s
19                 C(i+1,j+1,k+1) = 0;
20             else
21                 C(i+1,j+1,k+1) = factorial(i) * factorial(j) ...
22                     *factorial(k) / ((factorial(s-i) * factorial(s-j) * factorial(s-k)) * sqrt(factorial(i)) *
       sqrt(factorial(j)) * sqrt(factorial(k)));
23             end
24         end
25     end
26 end
27 return
```

C.2.1.3 A_matrix.m

```
1
2 function [A] = A_matrix(u_loc,p,C)
3
4 % Compute the matrix A(u) of triple inner products,
5 % where A_{i,j} =  sum_{k=0}^{p}  int u_k psi_i psi_j psi_k dP
6
7 % Indata:
8 % u_loc - Vector of gPC coefficients of the argument u
9 % p - Number of gPC basis functions
10 % C - Precomputed inner triple products of the basis functions psi
11
12 % Outdata:
13 % A - matrix of sums of inner products
14
15
16 A = zeros(p);
17
18 for j=1:p
19     A = A + C(:,:,j)*u_loc(j);
20 end
```

C.2.2 Discretization Operators

C.2.2.1 SBP_operators.m

```
1
2 function [D1 D2,BS,S,H] = SBP_operators(n,dx,order)
3
4 % SBP operators of orders 2, 4, 6 and 8 for the first and second derivative.
5
6 % Indata:
7 % n - Number of spatial grid pts
8 % dx - Step size
9 % order - Order of accuracy (only for 2,4,6,8)
10
11 % Outdata:
12 % D1 - First derivative operator
13 % D2 - Second derivative operator (D = P^{-1}M)
14 % S - First derivative operator on boundaries
15 % BS - The boundary elements in the energy estimate
16 % H - The norm operator (denoted P in some papers)
17
18
19 e = ones(n,1);
20
21 if order==2
22
23     D1 = 1/dx*spdiags([-1/2*e 0*e 1/2*e],-1:1,n,n);
24     D1(1,1) = -1/dx;
25     D1(1,2) = 1/dx;
26     D1(1,3) = 0;
27     D1(n,n) = 1/dx;
28     D1(n,n-1) = -1/dx;
29     D1(n,n-2) = 0;
30
```

```
31      %%%%%%%
32      D2 = 1/(dx^2)*spdiags([1*e -2*e 1*e],-1:1,n,n);
33      D2(1,1) = 1/(dx^2);
34      D2(1,2) = -2/(dx^2);
35      D2(1,3) = 1/(dx^2);
36      D2(n,n) = 1/(dx^2);
37      D2(n,n-1) = -2/(dx^2);
38      D2(n,n-2) = 1/(dx^2);
39
40
41      H = dx*spdiags([e],0,n,n);
42      H(1,1) = dx*1/2;
43      H(n,n) = dx*1/2;
44
45      BS = (1/dx)*spdiags(zeros(size(e)),0,n,n);
46      BS(1,1) = 3/2/dx;
47      BS(1,2) = -2/dx;
48      BS(1,3) = 1/2/dx;
49      BS(n,n) = 3/2/dx;
50      BS(n,n-1) = -2/dx;
51      BS(n,n-2) = 1/2/dx;
52
53      S = (1/dx)*spdiags([e],0,n,n);
54      S(1,1) = -3/2/dx;
55      S(1,2) = 2/dx;
56      S(1,3) = -1/2/dx;
57
58      S(n,n) = 3/2/dx;
59      S(n,n-1) = -2/dx;
60      S(n,n-2) = 1/2/dx;
61
62  elseif order==4
63
64      D1 = 1/dx*spdiags([1/12*e -2/3*e 0*e 2/3*e -1/12*e],-2:2,n,n);
65      D1(1,1) = -24/17/dx;
66      D1(1,2) = 59/34/dx;
67      D1(1,3) = -4/17/dx;
68      D1(1,4) = -3/34/dx;
69      D1(1,5) = 0;
70      D1(1,6) = 0;
71      D1(2,1) = -1/2/dx;
72      D1(2,2) = 0;
73      D1(2,3) = 1/2/dx;
74      D1(2,4:6) = 0;
75      D1(3,1) = 4/43/dx;
76      D1(3,2) = -59/86/dx;
77      D1(3,3) = 0;
78      D1(3,4) = 59/86/dx;
79      D1(3,5) = -4/43/dx;
80      D1(3,6) = 0;
81      D1(4,1) = 3/98/dx;
82      D1(4,2) = 0;
83      D1(4,3) = -59/98/dx;
84      D1(4,4) = 0;
85      D1(4,5) = 32/49/dx;
86      D1(4,6) = -4/49/dx;
87      D1(4,7) = 0;
88      D1(n,n) = -D1(1,1);
89      D1(n,n-1) = -D1(1,2);
90      D1(n,n-2) = -D1(1,3);
91      D1(n,n-3) = -D1(1,4);
92      D1(n,n-4) = -D1(1,5);
93      D1(n,n-5) = -D1(1,6);
94      D1(n-1,n) = -D1(2,1);
95      D1(n-1,n-1) = -D1(2,2);
96      D1(n-1,n-2) = -D1(2,3);
97      D1(n-1,n-3) = -D1(2,4);
98      D1(n-1,n-4) = -D1(2,5);
99      D1(n-1,n-5) = -D1(2,6);
100     D1(n-2,n) = -D1(3,1);
101     D1(n-2,n-1) = -D1(3,2);
102     D1(n-2,n-2) = -D1(3,3);
103     D1(n-2,n-3) = -D1(3,4);
104     D1(n-2,n-4) = -D1(3,5);
105     D1(n-2,n-5) = -D1(3,6);
106     D1(n-3,n) = -D1(4,1);
107     D1(n-3,n-1) = -D1(4,2);
108     D1(n-3,n-2) = -D1(4,3);
109     D1(n-3,n-3) = -D1(4,4);
110     D1(n-3,n-4) = -D1(4,5);
111     D1(n-3,n-5) = -D1(4,6);
112
113
114     %%%%%%%
115     D2 = 1/(dx^2)*spdiags([-1/12*e 4/3*e -5/2*e 4/3*e -1/12*e],-2:2,n,n);
116     D2(1,1) = 2/(dx^2);
117     D2(1,2) = -5/(dx^2);
118     D2(1,3) = 4/(dx^2);
119     D2(1,4) = -1/(dx^2);
```

```
120     D2(2,1) = 1/(dx^2);
121     D2(2,2) = -2/(dx^2);
122     D2(2,3) = 1/(dx^2);
123     D2(2,4) = 0;
124     D2(3,1) = -4/43/(dx^2);
125     D2(3,2) = 59/43/(dx^2);
126     D2(3,3) = -110/43/(dx^2);
127     D2(3,4) = 59/43/(dx^2);
128     D2(3,5) = -4/43/(dx^2);
129     D2(4,1) = -1/49/(dx^2);
130     D2(4,2) = 0;
131     D2(4,3) = 59/49/(dx^2);
132     D2(4,4) = -118/49/(dx^2);
133     D2(4,5) = 64/49/(dx^2);
134     D2(4,6) = -4/49/(dx^2);
135     D2(n,n) = D2(1,1);
136     D2(n,n-1) = D2(1,2);
137     D2(n,n-2) = D2(1,3);
138     D2(n,n-3) = D2(1,4);
139     D2(n-1,n) = D2(2,1);
140     D2(n-1,n-1) = D2(2,2);
141     D2(n-1,n-2) = D2(2,3);
142     D2(n-1,n-3) = D2(2,4);
143     D2(n-2,n) = D2(3,1);
144     D2(n-2,n-1) = D2(3,2);
145     D2(n-2,n-2) = D2(3,3);
146     D2(n-2,n-3) = D2(3,4);
147     D2(n-2,n-4) = D2(3,5);
148     D2(n-3,n) = D2(4,1);
149     D2(n-3,n-1) = D2(4,2);
150     D2(n-3,n-2) = D2(4,3);
151     D2(n-3,n-3) = D2(4,4);
152     D2(n-3,n-4) = D2(4,5);
153     D2(n-3,n-5) = D2(4,6);
154
155     H = dx*spdiags(e,0,n,n);
156     H(1,1) = dx*17/48;
157     H(2,2) = dx*59/48;
158     H(3,3) = dx*43/48;
159     H(4,4) = dx*49/48;
160     H(n,n) = H(1,1);
161     H(n-1,n-1) = H(2,2);
162     H(n-2,n-2) = H(3,3);
163     H(n-3,n-3) = H(4,4);
164
165     S = (1/dx)*spdiags(e,0,n,n);
166     S(1,1) = -11/6/dx;
167     S(1,2) = 3/dx;
168     S(1,3) = -3/2/dx;
169     S(1,4) = 1/3/dx;
170     S(n,n) = 11/6/dx;
171     S(n,n-1) = -3/dx;
172     S(n,n-2) = 3/2/dx;
173     S(n,n-3) = -1/3/dx;
174
175     BS = (1/dx)*spdiags(zeros(size(e)),0,n,n);
176     BS(1,1) = 11/6/dx;
177     BS(1,2) = -3/dx;
178     BS(1,3) = 3/2/dx;
179     BS(1,4) = -1/3/dx;
180     BS(n,n) = 11/6/dx;
181     BS(n,n-1) = -3/dx;
182     BS(n,n-2) = 3/2/dx;
183     BS(n,n-3) = -1/3/dx;
184
185 elseif order==6
186     e = ones(n,1);
187     %%%%
188     D1 = (1/(dx))*spdiags([-1/60*e 3/20*e -3/4*e 0*e 3/4*e -3/20*e 1/60*e],-3:3,n,n);
189
190     D1(1,1) = -21600/13649/dx;
191     D1(1,2) = 104009/54596/dx;
192     D1(1,3) = 30443/81894/dx;
193     D1(1,4) = -33311/27298/dx;
194     D1(1,5) = 16863/27298/dx;
195     D1(1,6) = -15025/163788/dx;
196     D1(1,7) = 0;
197     D1(1,8) = 0;
198     D1(2,1) = -104009/240260/dx;
199     D1(2,2) = 0;
200     D1(2,3) = -311/72078/dx;
201     D1(2,4) = 20229/24026/dx;
202     D1(2,5) = -24337/48052/dx;
203     D1(2,6) = 36661/360390/dx;
204     D1(2,7) = 0;
205     D1(2,8) = 0;
206     D1(3,1) = -30443/162660/dx;
207     D1(3,2) = 311/32532/dx;
208     D1(3,3) = 0;
```

```
209     D1(3,4) = -11155/16266/dx;
210     D1(3,5) = 41287/32532/dx;
211     D1(3,6) = -21999/54220/dx;
212     D1(3,7) = 0;
213     D1(3,8) = 0;
214     D1(4,1) = 33311/107180/dx;
215     D1(4,2) = -20229/21436/dx;
216     D1(4,3) = 485/1398/dx;
217     D1(4,4) = 0;
218     D1(4,5) = 4147/21436/dx;
219     D1(4,6) = 25427/321540/dx;
220     D1(4,7) = 72/5359/dx;
221     D1(4,8) = 0;
222     D1(5,1) = -16863/78770/dx;
223     D1(5,2) = 24337/31508/dx;
224     D1(5,3) = -41287/47262/dx;
225     D1(5,4) = -4147/15754/dx;
226     D1(5,5) = 0;
227     D1(5,6) = 342523/472620/dx;
228     D1(5,7) = -1296/7877/dx;
229     D1(5,8) = 144/7877/dx;
230     D1(5,9) = 0;
231     D1(6,1) = 15025/525612/dx;
232     D1(6,2) = -36661/262806/dx;
233     D1(6,3) = 21999/87602/dx;
234     D1(6,4) = -25427/262806/dx;
235     D1(6,5) = -342523/525612/dx;
236     D1(6,6) = 0;
237     D1(6,7) = 32400/43801/dx;
238     D1(6,8) = -6480/43801/dx;
239     D1(6,9) = 720/43801/dx;
240     D1(6,10) = 0;
241     D1(n,n) = -D1(1,1);
242     D1(n,n-1) = -D1(1,2);
243     D1(n,n-2) = -D1(1,3);
244     D1(n,n-3) = -D1(1,4);
245     D1(n,n-4) = -D1(1,5);
246     D1(n,n-5) = -D1(1,6);
247     D1(n,n-6) = -D1(1,7);
248     D1(n,n-7) = -D1(1,8);
249     D1(n-1,n) = -D1(2,1);
250     D1(n-1,n-1) = -D1(2,2);
251     D1(n-1,n-2) = -D1(2,3);
252     D1(n-1,n-3) = -D1(2,4);
253     D1(n-1,n-4) = -D1(2,5);
254     D1(n-1,n-5) = -D1(2,6);
255     D1(n-1,n-6) = -D1(2,7);
256     D1(n-1,n-7) = -D1(2,8);
257     D1(n-2,n) = -D1(3,1);
258     D1(n-2,n-1) = -D1(3,2);
259     D1(n-2,n-2) = -D1(3,3);
260     D1(n-2,n-3) = -D1(3,4);
261     D1(n-2,n-4) = -D1(3,5);
262     D1(n-2,n-5) = -D1(3,6);
263     D1(n-2,n-6) = -D1(3,7);
264     D1(n-2,n-7) = -D1(3,8);
265     D1(n-3,n) = -D1(4,1);
266     D1(n-3,n-1) = -D1(4,2);
267     D1(n-3,n-2) = -D1(4,3);
268     D1(n-3,n-3) = -D1(4,4);
269     D1(n-3,n-4) = -D1(4,5);
270     D1(n-3,n-5) = -D1(4,6);
271     D1(n-3,n-6) = -D1(4,7);
272     D1(n-3,n-7) = -D1(4,8);
273     D1(n-4,n) = -D1(5,1);
274     D1(n-4,n-1) = -D1(5,2);
275     D1(n-4,n-2) = -D1(5,3);
276     D1(n-4,n-3) = -D1(5,4);
277     D1(n-4,n-4) = -D1(5,5);
278     D1(n-4,n-5) = -D1(5,6);
279     D1(n-4,n-6) = -D1(5,7);
280     D1(n-4,n-7) = -D1(5,8);
281     D1(n-4,n-8) = -D1(5,9);
282     D1(n-5,n) = -D1(6,1);
283     D1(n-5,n-1) = -D1(6,2);
284     D1(n-5,n-2) = -D1(6,3);
285     D1(n-5,n-3) = -D1(6,4);
286     D1(n-5,n-4) = -D1(6,5);
287     D1(n-5,n-5) = -D1(6,6);
288     D1(n-5,n-6) = -D1(6,7);
289     D1(n-5,n-7) = -D1(6,8);
290     D1(n-5,n-8) = -D1(6,9);
291     D1(n-5,n-9) = -D1(6,10);
292
293     %%%%%
294     D2 = (1/(dx^2))*spdiags([1/90*e -3/20*e 3/2*e -49/18*e 3/2*e -3/20*e 1/90*e],-3:3,n,n);
295
296     D2(1,1) = 114170/40947/(dx^2);
297     D2(1,2) = -438107/54596/(dx^2);
```

```
298     D2(1,3) = 336409/40947/(dx^2);
299     D2(1,4) = -276997/81894/(dx^2);
300     D2(1,5) = 3747/13649/(dx^2);
301     D2(1,6) = 21035/163788/(dx^2);
302     D2(1,7) = 0;
303     D2(1,8) = 0;
304     D2(2,1) = 6173/5860/(dx^2);
305     D2(2,2) = -2066/879/(dx^2);
306     D2(2,3) = 3283/1758/(dx^2);
307     D2(2,4) =-303/293/(dx^2);
308     D2(2,5) = 2111/3516/(dx^2);
309     D2(2,6) = -601/4395/(dx^2);
310     D2(2,7) = 0;
311     D2(2,8) = 0;
312     D2(3,1) = -52391/81330/(dx^2);
313     D2(3,2) = 134603/32532/(dx^2);
314     D2(3,3) = -21982/2711/(dx^2);
315     D2(3,4) = 112915/16266/(dx^2);
316     D2(3,5) = -46969/16266/(dx^2);
317     D2(3,6) = 30409/54220/(dx^2);
318     D2(3,7) = 0;
319     D2(3,8) = 0;
320     D2(4,1) = 68603/321540/(dx^2);
321     D2(4,2) = -12423/10718/(dx^2);
322     D2(4,3) = 112915/32154/(dx^2);
323     D2(4,4) = -75934/16077/(dx^2);
324     D2(4,5) = 53369/21436/(dx^2);
325     D2(4,6) = -54899/160770/(dx^2);
326     D2(4,7) = 48/5359/(dx^2);
327     D2(4,8) = 0;
328     D2(5,1) = -7053/39385/(dx^2);
329     D2(5,2) = 86551/94524/(dx^2);
330     D2(5,3) = -46969/23631/(dx^2);
331     D2(5,4) = 53369/15754/(dx^2);
332     D2(5,5) = -87904/23631/(dx^2);
333     D2(5,6) = 820271/472620/(dx^2);
334     D2(5,7) = -1296/7877/(dx^2);
335     D2(5,8) = 96/7877/(dx^2);
336     D2(5,9) = 0;
337     D2(6,1) = 21035/525612/(dx^2);
338     D2(6,2) = -24641/131403/(dx^2);
339     D2(6,3) = 30409/87602/(dx^2);
340     D2(6,4) = -54899/131403/(dx^2);
341     D2(6,5) = 820271/525612/(dx^2);
342     D2(6,6) = -117600/43801/(dx^2);
343     D2(6,7) = 64800/43801/(dx^2);
344     D2(6,8) = -6480/43801/(dx^2);
345     D2(6,9) = 480/43801/(dx^2);
346     D2(6,10) = 0;
347     D2(n,n) = D2(1,1);
348     D2(n,n-1) = D2(1,2);
349     D2(n,n-2) = D2(1,3);
350     D2(n,n-3) = D2(1,4);
351     D2(n,n-4) = D2(1,5);
352     D2(n,n-5) = D2(1,6);
353     D2(n,n-6) = D2(1,7);
354     D2(n,n-7) = D2(1,8);
355     D2(n-1,n) = D2(2,1);
356     D2(n-1,n-1) = D2(2,2);
357     D2(n-1,n-2) = D2(2,3);
358     D2(n-1,n-3) = D2(2,4);
359     D2(n-1,n-4) = D2(2,5);
360     D2(n-1,n-5) = D2(2,6);
361     D2(n-1,n-6) = D2(2,7);
362     D2(n-1,n-7) = D2(2,8);
363     D2(n-2,n) = D2(3,1);
364     D2(n-2,n-1) = D2(3,2);
365     D2(n-2,n-2) = D2(3,3);
366     D2(n-2,n-3) = D2(3,4);
367     D2(n-2,n-4) = D2(3,5);
368     D2(n-2,n-5) = D2(3,6);
369     D2(n-2,n-6) = D2(3,7);
370     D2(n-2,n-7) = D2(3,8);
371     D2(n-3,n) = D2(4,1);
372     D2(n-3,n-1) = D2(4,2);
373     D2(n-3,n-2) = D2(4,3);
374     D2(n-3,n-3) = D2(4,4);
375     D2(n-3,n-4) = D2(4,5);
376     D2(n-3,n-5) = D2(4,6);
377     D2(n-3,n-6) = D2(4,7);
378     D2(n-3,n-7) = D2(4,8);
379     D2(n-4,n) = D2(5,1);
380     D2(n-4,n-1) = D2(5,2);
381     D2(n-4,n-2) = D2(5,3);
382     D2(n-4,n-3) = D2(5,4);
383     D2(n-4,n-4) = D2(5,5);
384     D2(n-4,n-5) = D2(5,6);
385     D2(n-4,n-6) = D2(5,7);
386     D2(n-4,n-7) = D2(5,8);
```

```
387     D2(n-4,n-8) = D2(5,9);
388     D2(n-5,n) = D2(6,1);
389     D2(n-5,n-1) = D2(6,2);
390     D2(n-5,n-2) = D2(6,3);
391     D2(n-5,n-3) = D2(6,4);
392     D2(n-5,n-4) = D2(6,5);
393     D2(n-5,n-5) = D2(6,6);
394     D2(n-5,n-6) = D2(6,7);
395     D2(n-5,n-7) = D2(6,8);
396     D2(n-5,n-8) = D2(6,9);
397     D2(n-5,n-9) = D2(6,10);
398
399     H = dx*spdiags([e],0,n,n);
400
401     H(1,1) = dx*13649/43200;
402     H(2,2) = dx*12013/8640;
403     H(3,3) = dx*2711/4320;
404     H(4,4) = dx*5359/4320;
405     H(5,5) = dx*7877/8640;
406     H(6,6) = dx*43801/43200;
407     H(n,n) = H(1,1);
408     H(n-1,n-1) = H(2,2);
409     H(n-2,n-2) = H(3,3);
410     H(n-3,n-3) = H(4,4);
411     H(n-4,n-4) = H(5,5);
412     H(n-5,n-5) = H(6,6);
413
414     BS = (1/dx)*spdiags([zeros(size(e))],0,n,n);
415
416     BS(1,1) = 25/12/dx;
417     BS(1,2) = -4/dx;
418     BS(1,3) = 3/dx;
419     BS(1,4) = -4/3/dx;
420     BS(1,5) = 1/4/dx;
421     BS(n,n) = BS(1,1);
422     BS(n,n-1) = BS(1,2);
423     BS(n,n-2) = BS(1,3);
424     BS(n,n-3) = BS(1,4);
425     BS(n,n-4) = BS(1,5);
426
427     S = (1/dx)*spdiags([e],0,n,n);
428
429     S(1,1) = -25/12/dx;
430     S(1,2) = 4/dx;
431     S(1,3) = -3/dx;
432     S(1,4) = 4/3/dx;
433     S(1,5) = -1/4/dx;
434     S(n,n) = BS(1,1);
435     S(n,n-1) = BS(1,2);
436     S(n,n-2) = BS(1,3);
437     S(n,n-3) = BS(1,4);
438     S(n,n-4) = BS(1,5);
439 elseif order==8
440     e = ones(n,1);
441
442     D = (1/(dx^2))*spdiags([-1/560*e 8/315*e -1/5*e 8/5*e -205/72*e 8/5*e -1/5*e 8/315*e -1/560*e],-4:4,n,n);
443     % eighth order standard central stencil
444
445     D(1,1) = 4870382994799/1358976868290/(dx^2);
446     D(1,2) = -893640087518/75498714905/(dx^2);
447     D(1,3) = 926594825119/60398971924/(dx^2);
448     D(1,4) = -1315109406200/135897686829/(dx^2);
449     D(1,5) = 39126983272/15099742981/(dx^2);
450     D(1,6) = 12344491342/75498714905/(dx^2);
451     D(1,7) = -451560522577/2717953736580/(dx^2);
452     D(1,8) = 0;
453     D(1,9) = 0;
454     D(1,10) = 0;
455     D(1,11) = 0;
456     D(1,12) = 0;
457     D(2,1) = 3338060121194/390619153855/(dx^2);
458     D(2,2) = -154646272029/111605472530/(dx^2);
459     D(2,3) = 1168338040/33481641759/(dx^2);
460     D(2,4) = 82699112501/133926567036/(dx^2);
461     D(2,5) = -171562838/11160547253/(dx^2);
462     D(2,6) = -28244698346/167408208795/(dx^2);
463     D(2,7) = 11904122576/167408208795/(dx^2);
464     D(2,8) = -2598164715/312495323084/(dx^2);
465     D(2,9) = 0;
466     D(2,10) = 0;
467     D(2,11) = 0;
468     D(2,12) = 0;
469     D(3,1) = 7838984095/527310299888/(dx^2);
470     D(3,2) = 1168338040/5649753213/(dx^2);
471     D(3,3) = -88747895/144865467/(dx^2);
472     D(3,4) = 423587231/627750357/(dx^2);
473     D(3,5) = -43205598281/22599012852/(dx^2);
474     D(3,6) = 4876378562/1883251071/(dx^2);
475     D(3,7) = -5124426509/3766502142/(dx^2);
```

```
476     D(3,8) = 10496900965/39548272491/(dx^2);
477     D(3,9) = 0;
478     D(3,10) = 0;
479     D(3,11) = 0;
480     D(3,12) = 0;
481     D(4,1) = -94978241528/828644350023/(dx^2);
482     D(4,2) = 82699112501/157837019052/(dx^2);
483     D(4,3) = 1270761693/13153084921/(dx^2);
484     D(4,4) = -167389605005/118377764289/(dx^2);
485     D(4,5) = 48242560214/39459254763/(dx^2);
486     D(4,6) = -31673996013/52612339684/(dx^2);
487     D(4,7) = 43556319241/118377764289/(dx^2);
488     D(4,8) = -44430275135/552429566682/(dx^2);
489     D(4,9) = 0;
490     D(4,10) = 0;
491     D(4,11) = 0;
492     D(4,12) = 0;
493     D(5,1) = 1455067816/21132528431/(dx^2);
494     D(5,2) = -171562838/3018932633/(dx^2);
495     D(5,3) = -43205598281/36227191596/(dx^2);
496     D(5,4) = 48242560214/9056797899/(dx^2);
497     D(5,5) = -52276055645/6037865266/(dx^2);
498     D(5,6) = 57521587238/9056797899/(dx^2);
499     D(5,7) = -80321706377/36227191596/(dx^2);
500     D(5,8) = 8078087158/21132528431/(dx^2);
501     D(5,9) = -1296/299527/(dx^2);
502     D(5,10) = 0;
503     D(5,11) = 0;
504     D(5,12) = 0;
505     D(6,1) = 10881504334/327321118845/(dx^2);
506     D(6,2) = -28244698346/140280479505/(dx^2);
507     D(6,3) = 4876378562/9352031967/(dx^2);
508     D(6,4) = -10557998671/12469375956/(dx^2);
509     D(6,5) = 57521587238/28056095901/(dx^2);
510     D(6,6) = -278531401019/93520319670/(dx^2);
511     D(6,7) = 73790130002/46760159835/(dx^2);
512     D(6,8) = -137529995233/785570685228/(dx^2);
513     D(6,9) = 2048/103097/(dx^2);
514     D(6,10) = -144/103097/(dx^2);
515     D(6,11) = 0;
516     D(6,12) = 0;
517     D(7,1) = -135555328849/8509847458140/(dx^2);
518     D(7,2) = 11904122576/101307707835/(dx^2);
519     D(7,3) = -5124426509/13507694378/(dx^2);
520     D(7,4) = 43556319241/60784624701/(dx^2);
521     D(7,5) = -80321706377/81046166268/(dx^2);
522     D(7,6) = 73790130002/33769235945/(dx^2);
523     D(7,7) = -950494905688/303923123505/(dx^2);
524     D(7,8) = 239073018673/141830790969/(dx^2);
525     D(7,9) = -145152/670091/(dx^2);
526     D(7,10) = 18432/670091/(dx^2);
527     D(7,11) = -1296/670091/(dx^2);
528     D(7,12) = 0;
529     D(8,1) = 0;
530     D(8,2) = -2598164715/206729925524/(dx^2);
531     D(8,3) = 10496900965/155047444143/(dx^2);
532     D(8,4) = -44430275135/310094888286/(dx^2);
533     D(8,5) = 425162482/2720130599/(dx^2);
534     D(8,6) = -137529995233/620189776572/(dx^2);
535     D(8,7) = 239073018673/155047444143/(dx^2);
536     D(8,8) = -144648000000/51682481381/(dx^2);
537     D(8,9) = 8128512/5127739/(dx^2);
538     D(8,10) = -1016064/5127739/(dx^2);
539     D(8,11) = 129024/5127739/(dx^2);
540     D(8,12) = -9072/5127739/(dx^2);
541
542     D(n,n) = D(1,1);
543     D(n,n-1) = D(1,2);
544     D(n,n-2) = D(1,3);
545     D(n,n-3) = D(1,4);
546     D(n,n-4) = D(1,5);
547     D(n,n-5) = D(1,6);
548     D(n,n-6) = D(1,7);
549     D(n,n-7) = D(1,8);
550     D(n,n-8) = D(1,9);
551     D(n,n-9) = D(1,10);
552     D(n,n-10) = D(1,11);
553     D(n,n-11) = D(1,12);
554     D(n-1,n) = D(2,1);
555     D(n-1,n-1) = D(2,2);
556     D(n-1,n-2) = D(2,3);
557     D(n-1,n-3) = D(2,4);
558     D(n-1,n-4) = D(2,5);
559     D(n-1,n-5) = D(2,6);
560     D(n-1,n-6) = D(2,7);
561     D(n-1,n-7) = D(2,8);
562     D(n-1,n-8) = D(2,9);
563     D(n-1,n-9) = D(2,10);
564     D(n-1,n-10) = D(2,11);
```

```
565     D(n-1,n-11) = D(2,12);
566     D(n-2,n) = D(3,1);
567     D(n-2,n-1) = D(3,2);
568     D(n-2,n-2) = D(3,3);
569     D(n-2,n-3) = D(3,4);
570     D(n-2,n-4) = D(3,5);
571     D(n-2,n-5) = D(3,6);
572     D(n-2,n-6) = D(3,7);
573     D(n-2,n-7) = D(3,8);
574     D(n-2,n-8) = D(3,9);
575     D(n-2,n-9) = D(3,10);
576     D(n-2,n-10) = D(3,11);
577     D(n-2,n-11) = D(3,12);
578     D(n-3,n) = D(4,1);
579     D(n-3,n-1) = D(4,2);
580     D(n-3,n-2) = D(4,3);
581     D(n-3,n-3) = D(4,4);
582     D(n-3,n-4) = D(4,5);
583     D(n-3,n-5) = D(4,6);
584     D(n-3,n-6) = D(4,7);
585     D(n-3,n-7) = D(4,8);
586     D(n-3,n-8) = D(4,9);
587     D(n-3,n-9) = D(4,10);
588     D(n-3,n-10) = D(4,11);
589     D(n-3,n-11) = D(4,12);
590     D(n-4,n) = D(5,1);
591     D(n-4,n-1) = D(5,2);
592     D(n-4,n-2) = D(5,3);
593     D(n-4,n-3) = D(5,4);
594     D(n-4,n-4) = D(5,5);
595     D(n-4,n-5) = D(5,6);
596     D(n-4,n-6) = D(5,7);
597     D(n-4,n-7) = D(5,8);
598     D(n-4,n-8) = D(5,9);
599     D(n-4,n-9) = D(5,10);
600     D(n-4,n-10) = D(5,11);
601     D(n-4,n-11) = D(5,12);
602     D(n-5,n) = D(6,1);
603     D(n-5,n-1) = D(6,2);
604     D(n-5,n-2) = D(6,3);
605     D(n-5,n-3) = D(6,4);
606     D(n-5,n-4) = D(6,5);
607     D(n-5,n-5) = D(6,6);
608     D(n-5,n-6) = D(6,7);
609     D(n-5,n-7) = D(6,8);
610     D(n-5,n-8) = D(6,9);
611     D(n-5,n-9) = D(6,10);
612     D(n-5,n-10) = D(6,11);
613     D(n-5,n-11) = D(6,12);
614     D(n-6,n) = D(7,1);
615     D(n-6,n-1) = D(7,2);
616     D(n-6,n-2) = D(7,3);
617     D(n-6,n-3) = D(7,4);
618     D(n-6,n-4) = D(7,5);
619     D(n-6,n-5) = D(7,6);
620     D(n-6,n-6) = D(7,7);
621     D(n-6,n-7) = D(7,8);
622     D(n-6,n-8) = D(7,9);
623     D(n-6,n-9) = D(7,10);
624     D(n-6,n-10) = D(7,11);
625     D(n-6,n-11) = D(7,12);
626     D(n-7,n) = D(8,1);
627     D(n-7,n-1) = D(8,2);
628     D(n-7,n-2) = D(8,3);
629     D(n-7,n-3) = D(8,4);
630     D(n-7,n-4) = D(8,5);
631     D(n-7,n-5) = D(8,6);
632     D(n-7,n-6) = D(8,7);
633     D(n-7,n-7) = D(8,8);
634     D(n-7,n-8) = D(8,9);
635     D(n-7,n-9) = D(8,10);
636     D(n-7,n-10) = D(8,11);
637     D(n-7,n-11) = D(8,12);
638
639
640     H = dx*spdiags([e],0,n,n);
641
642     H(1,1) = dx*1498139/5080320;
643     H(2,2) = dx*1107307/725760;
644     H(3,3) = dx*20761/80640;
645     H(4,4) = dx*1304999/725760;
646     H(5,5) = dx*299527/725760;
647     H(6,6) = dx*103097/80640;
648     H(7,7) = dx*670091/725760;
649     H(8,8) = dx*5127739/5080320;
650     H(n,n) = H(1,1);
651     H(n-1,n-1) = H(2,2);
652     H(n-2,n-2) = H(3,3);
653     H(n-3,n-3) = H(4,4);
```

```
654     H(n-4,n-4) = H(5,5);
655     H(n-5,n-5) = H(6,6);
656     H(n-6,n-6) = H(7,7);
657     H(n-7,n-7) = H(8,8);
658
659     BS = (1/dx)*spdiags([zeros(size(e))],0,n,n);
660
661     BS(1,1) = 4723/2100/dx;
662     BS(1,2) = -839/175/dx;
663     BS(1,3) = 157/35/dx;
664     BS(1,4) = -278/105/dx;
665     BS(1,5) = 103/140/dx;
666     BS(1,6) = 1/175/dx;
667     BS(1,7) = -6/175/dx;
668     BS(n,n) = BS(1,1);
669     BS(n,n-1) = BS(1,2);
670     BS(n,n-2) = BS(1,3);
671     BS(n,n-3) = BS(1,4);
672     BS(n,n-4) = BS(1,5);
673     BS(n,n-5) = BS(1,6);
674     BS(n,n-6) = BS(1,7);
675
676
677     S = (1/dx)*spdiags([e],0,n,n);
678
679     S(1,1) = -4723/2100/dx;
680     S(1,2) = 839/175/dx;
681     S(1,3) = -157/35/dx;
682     S(1,4) = 278/105/dx;
683     S(1,5) = -103/140/dx;
684     S(1,6) = -1/175/dx;
685     S(1,7) = 6/175/dx;
686     S(n,n) = BS(1,1);
687     S(n,n-1) = BS(1,2);
688     S(n,n-2) = BS(1,3);
689     S(n,n-3) = BS(1,4);
690     S(n,n-4) = BS(1,5);
691     S(n,n-5) = BS(1,6);
692     S(n,n-6) = BS(1,7);
693
694 else
695     disp('Only order 2, 4, 6 or 8 implemented here.')
696 end
```

C.2.2.2 flux_func.m

```
1
2  function [flux] = flux_func(u,p,C)
3
4  % Flux function, yields the stochastic Galerkin flux f = 0.5*A(u)u
5
6  % Indata:
7  % u - Vector of solution variables (gPC coefficients)
8  % p - Number of gPC basis functions
9  % C - Triple product matrix
10
11 % Outdata:
12 % flux - Stochastic Galerkin flux function
13
14
15 flux = zeros(length(u),1);
16
17 for i=0:length(u)/p-1 % Loop over the spatial grid points
18     u_part=u(p*i+1:p*(i+1),1);
19     flux(i*p+1:(i+1)*p,1) = 0.5*A_matrix(u_part,p,C)*u_part;
20 end
```

C.2.2.3 dissipation_2nd_der.m

```
1
2  function [diss_op] = dissipation_2nd_der(m,p,P_inv,H_inv,const,dx)
3
4  % Dissipation operator corresponding to second derivative to a system of size m*p (space * PCE-coeff.)
5  % Global dissipation constant
6
7  % Indata:
8  % m - Number of spatial grid points
9  % p - Number of gPC coefficients
```

```
% P_inv - Inverse of SBP norm matrix P
% H_inv - Inverse of SG mass matrix (in our implementation it is always the identity matrix)
% const - Dissipation constant
% dx - Spatial grid size

% Outdata:
% diss_op - Discrete dissipation matrix

D = zeros(m)+diag(ones(m,1))-diag(ones(m-1,1),-1);
D(1,1) = -1;
D(1,2) = 1;
D = sparse(D);

B = const*eye(m*p);
B(1,1) = 0;
diss_op=-dx*kron(P_inv*D',eye(p))*B*kron(D,H_inv);
```

C.2.2.4 dissipation_4th_der.m

```

function [diss_op] = dissipation_4th_der(m,p,P2_inv,H_inv,const,dx)

% Dissipation operator corresponding to fourth order derivative to a system of size m*p (space * PCE-coeff.)
% Global dissipation constant

% Indata:
% m - Number of spatial grid points
% p - Number of gPC coefficients
% P2_inv - Inverse of SBP norm matrix P2
% H_inv - Inverse of SG mass matrix (in our implementation it is always the identity matrix)
% const - Dissipation constant
% dx - Spatial grid size

% Outdata:
% diss_op - Discrete dissipation matrix

dia=[1 -2 1];
D2=spdiags(ones(m,1)*dia,[-1:1],m,m);
D2(1,1:3)=dia;
D2(m,m-2:m)=dia;

B2 = const*eye(m);
B2(1,1) = 0;

diss_op=kron(-dx*P2_inv*D2'*B2*D2,H_inv);
```

C.2.3 Boundary Treatment

C.2.3.1 penalty.m

```

function [Sig_left,Sig_right]=penalty(p,u_bc_l,u_bc_r,C)

% Assign penalty matrix for conservative system

% Indata:
% p - Number of gPC coefficients (p=M-1)
% u_bc_l, u_bc_r - Left and right boundary values
% C - Matrices of inner triple products of gPC basis functions

% Outdata:
% Sig_left, Sig_right - Left and right penalty matrices (SAT)

if p>1
    A_l = (C(:,:,1)*u_bc_l(1)+C(:,:,2)*u_bc_l(2));
    A_r = (C(:,:,1)*u_bc_r(1)+C(:,:,2)*u_bc_r(2));
    %Decomposition of the system matrix according to the signs of the
    %eigenvalues
    [X_l,D_l] = eig(A_l);
    [X_r,D_r] = eig(A_r);
    for i=1:p
        if D_l(i,i)<0
            D_l(i,i) = 0;
        end
        if D_r(i,i)>0
```

```
26             D_r(i,i) = 0;
27         end
28     end
29     A_l = X_l*D_l*X_l';
30     A_r = X_r*D_r*X_r';
31
32     %Scaling with 0.5 for conservative systems
33     Sig_left = -1/2*A_l;
34     Sig_right = 1/2*A_r;
35
36 end
37 if p==1
38     Sig_left = -1/2*u_bc_l(1);
39     Sig_right = 1/2*u_bc_r(1);
40 end
```

C.2.3.2 boundary_cond_determ.m

```
1
2 function [g_left g_right] = boundary_cond_determ(mean_left,mean_right,t,x0,left,right,m)
3
4 % Compute boundary conditions for the deterministic Burgers' equation
5
6 % Indata:
7 % mean_left,mean_right - Left and right states
8 % t - Time
9 % x0 - Initial shock location
10 % left - Lower limit of spatial interval
11 % right - Upper limit of spatial interval
12 % m - Number of spatial grid points
13
14 % Outdata:
15 % g_left - Left boundary Dirichlet data
16 % g_right - Right boundary Dirichlet data
17
18
19 g_left = zeros(m,1);
20 g_right = zeros(m,1);
21
22 % Shock speed
23 s = (mean_left + mean_right)/2;
24
25 if x0+s*t < left
26    g_left(1) = mean_right;
27    g_right(end) = mean_right;
28 end
29
30 if x0+s*t >= left && x0+s*t <= right
31    g_left(1) = mean_left;
32    g_right(end) = mean_right;
33 end
34
35 if x0+s*t > right
36    g_left(1) = mean_left;
37    g_right(end) = mean_left;
38 end
```

C.2.3.3 boundary_cond_2x2.m

```
1
2 function [g_left g_right] = boundary_cond_2x2(mean_left,mean_right,sig_h,t,x0,left,right,m)
3
4 % Compute the boundary conditions of the 2x2 stochastic Galerkin form of
5 % Burgers' equation for the Riemann problem
6
7 % Indata:
8 % mean_left,mean_right - Mean (u_0) of the left and right states
9 % sig_h - Standard deviation (u_1), assumed uniform over the spatial domain
10 % t - Time
11 % x0 - Initial shock location
12 % left - Lower limit of spatial interval
13 % right - Upper limit of spatial interval
14 % m - Number of spatial grid points
15
16 % Outdata:
17 % g_left - Dirichlet condition for left boundary
18 % g_right - Dirichlet condition for right boundary
19
```

```
20
21 % Exact solution for the 2 x 2 case
22
23 s1 = (mean_left+mean_right)/2-sig_h; % Shock speed 1
24 s2 = (mean_left+mean_right)/2+sig_h; % Shock speed 2
25
26 g_left = zeros(2*m,1);
27 g_right = zeros(2*m,1);
28 g_left(1) = mean_left;
29 g_left(2) = sig_h;
30 g_right(2*(m-1)+1) = mean_right;
31 g_right(2*(m-1)+2) = sig_h;
32
33 % One wave propagating to the left, the other to the right
34 if s1<=0 && s2>=0
35     if t>(left-x0)/s1
36         g_left(1)=(mean_left+mean_right)/2
37         g_left(2) = (mean_left-mean_right)/2+sig_h;
38
39     end
40     if t>(right-x0)/s2
41         g_right(2*(m-1)+1) = (mean_left+mean_right)/2;
42         g_right(2*(m-1)+2) = (mean_left-mean_right)/2+sig_h;
43     end
44 end
45
46 % Both waves propagating to the left
47 if s1<=0 && s2<=0
48     if t>(left-x0)/s1 && t<(left-x0)/s2
49         g_left(1)=(mean_left+mean_right)/2;
50         g_left(2) = (mean_left-mean_right)/2+sig_h;
51
52     end
53     if t>(left-x0)/s2
54         g_left(1) = mean_right;
55         g_left(2) = sig_h;
56     end
57 end
58
59 % Both waves propagating to the right
60 if s1>=0 && s2>=0
61     if t>(right-x0)/s2 && t < (right-x0)/s1
62         g_right(2*(m-1)+1) = (mean_left+mean_right)/2;
63         g_right(2*(m-1)+2) = (mean_left-mean_right)/2+sig_h;
64     end
65     if t > (right-x0)/s1
66         g_right(2*(m-1)+1) = mean_left;
67         g_right(2*(m-1)+2) = sig_h;
68     end
69 end
```

C.2.3.4 boundary_cond_p_inf.m

```
1
2 function [g_left g_right] = boundary_cond_p_inf(mean_left,mean_right,sig_h,t,x0,left,right,p,m)
3
4 % Calculate time dependent boundary conditions for the first p coefficients of the infinite order expansion
5
6 % Indata:
7 % mean_left,mean_right - Left and right states
8 % sig_h - Standard deviation (uniform in space)
9 % t - Time
10 % x0 - Initial shock location
11 % left - Lower limit of spatial interval
12 % right - Upper limit of spatial interval
13 % p - Number of gPC coefficients to be computed
14 % m - Number of spatial grid points
15
16 % Outdata:
17 % g_left - Left boundary Dirichlet data for the vector of gPC coefficients
18 % g_right - Right boundary Dirichlet data for the vector of gPC coefficients
19
20
21 g_left = zeros(p*m,1);
22 g_right = zeros(p*m,1);
23
24 xi_l = (left-x0)./(sig_h*t)-(mean_left+mean_right)/(2*sig_h);
25 xi_r = (right-x0)./(sig_h*t)-(mean_left+mean_right)/(2*sig_h);
26 g_left(1) = mean_left + (mean_right-mean_left)*normcdf(xi_l,0,1);
27 g_right((m-1)*p+1) = mean_left + (mean_right-mean_left)*normcdf(xi_r,0,1);
28
```

```
g_left(2) = sig_h + (mean_left-mean_right)*exp(-xi_l^2/2)/sqrt(2*pi);
g_right((m-1)*p+2) = sig_h + (mean_left-mean_right)*exp(-xi_r^2/2)/sqrt(2*pi);
Psi_l(1:2) = [1 xi_l];
Psi_r(1:2) = [1 xi_r];

for k=3:p
    Psi_l(k) = xi_l.*sqrt(factorial(k-2)/factorial(k-1))*Psi_l(k-1) - (k-2)*sqrt(factorial(k-3)/factorial(k-1)).*Psi_l(k-2);
    Psi_r(k) = xi_r.*sqrt(factorial(k-2)/factorial(k-1))*Psi_r(k-1) - (k-2)*sqrt(factorial(k-3)/factorial(k-1)).*Psi_r(k-2);

    g_left(k) = (mean_left-mean_right)/sqrt(k-1)*exp(-xi_l^2/2)/sqrt(2*pi).*Psi_l(k-1);
    g_right((m-1)*p+k) = (mean_left-mean_right)/sqrt(k-1)*exp(-xi_r^2/2)/sqrt(2*pi).*Psi_r(k-1);
end
```

C.2.4 Reference Solution

C.2.4.1 initial_conditions.m

```

function [u_init] = initial_conditions(m,p,C,x0,mean_left,std_left,mean_right,std_right,left,right)

% Compute initial conditions (gPC coefficients) for the Riemann problem

% Indata:
% m - Number of spatial grid points
% p - Number of gPC coefficients to be computed
% C - Inner triple product matrices
% x0 - Initial shock location
% mean_left - Left mean state
% std_left - Standard deviation left state
% mean_right - Right mean state
% std_right - Standard deviation right state
% left - Lower limit of spatial interval
% right - Upper limit of spatial interval

% Output:
% u_init - Vector of initial gPC coefficients

u_init = zeros(m*p,1);

for i=1:p:p*(ceil(m*(x0-left)/(right-left))-1)+1
    u_init(i) = mean_left;
    if p>1
        u_init(i+1) = std_left;
    end
end
for (i=ceil(m*(x0-left)/(right-left))*p+1:p:p*(m-1)+1)
    u_init(i) = mean_right;
    if p>1
        u_init(i+1) = std_right;
    end
end
```

C.2.4.2 exact_solution_2x2.m

```

function [u_ref] = exact_solution_2x2(mean_left,mean_right,sig_h,T,m,left,right,x0,x)

% Compute the analytical solution of the 2x2 stochastic Galerkin Burgers'
% equation

% Indata:
% mean_left,mean_right - Mean (u_0) of the left and right states
% sig_h - Standard deviation (u_1), assumed uniform over the spatial domain
% T - Time
% m - Number of spatial grid points
% left - Lower limit of spatial interval
% right - Upper limit of spatial interval
% x0 - Initial shock location
% x - Vector of spatial grid points

% Outdata:
% u_ref - Analysical solution

```

```
20
21 s1 = (mean_left+mean_right)/2-sig_h; % Shock speed 1
22 s2 = (mean_left+mean_right)/2+sig_h; % Shock speed 2
23
24 u_ref = zeros(m,2);
25
26 for j=1:m
27     if x(j)< x0+s1*T
28         u_ref(j,1)=mean_left;
29         u_ref(j,2)=sig_h;
30     end
31     if x(j)>= x0+s1*T && x(j) < x0+s2*T
32         u_ref(j,1)= (mean_left+mean_right)/2;
33         u_ref(j,2) = (mean_left-mean_right)/2+sig_h;
34     end
35     if x(j)> x0+s2*T
36         u_ref(j,1)= mean_right;
37         u_ref(j,2) = sig_h;
38     end
39 end
```

C.2.4.3 exact_solution_determ.m

```
1
2 function [u_ref] = exact_solution_determ(mean_left,mean_right,T,m,left,right,x0,x)
3
4 % Compute the exact solution of the deterministic Burgers' equation
5
6 % Indata:
7 % mean_left,mean_right - Left and right states
8 % T - Time
9 % m - Number of spatial grid points
10 % left - Lower limit of spatial interval
11 % right - Upper limit of spatial interval
12 % x0 - Initial shock location
13 % x - Vector of spatial grid points
14
15 % Outdata:
16 % u_ref - Analytical solution
17
18
19 s = (mean_left + mean_right)/2; % Shock speed
20 u_ref = zeros(m,1);
21
22 for j=1:m
23     if x(j)< x0+s*T
24         u_ref(j,1)=mean_left;
25     end
26
27     if x(j)>= x0+s*T
28         u_ref(j,1)= mean_right;
29     end
30 end
```

C.2.4.4 exact_solution_p_inf.m

```
1
2 function [u_ref] = exact_solution_p_inf(mean_left,mean_right,sig_h,T,m,left,right,x0,x,p)
3
4 % Compute the analytical solution of the stochastic Burgers' equation with
5 % Hermite polynomials
6
7
8 % Indata:
9 % mean_left,mean_right - Mean (u_0) of the left and right states
10 % sig_h - Standard deviation (u_1), assumed uniform over the spatial domain
11 % T - Time
12 % m - Number of spatial grid points
13 % left - Lower limit of spatial interval
14 % right - Upper limit of spatial interval
15 % x0 - Initial shock location
16 % x - Vector of spatial grid points
17 % p - Number of gPC coefficients to be computed
18
19 % Outdata:
20 % u_ref - Analytical solution
21
```

```
22
23 y = zeros(m,1);
24 y(:,1)=(x-x0)./(sig_h*T)-(mean_left+mean_right)/(2*sig_h);
25
26 Psi_s = zeros(m,p);
27 Psi_s(:,1) = 1;
28 Psi_s(:,2) = y;
29
30
31 u_ref = zeros(m,p);
32 u_ref(:,1) = mean_left - (mean_left-mean_right)*normcdf(y,0,1);
33
34 if p>2
35     u_ref(:,2) = sig_h + (mean_left-mean_right)*exp(-y.^2/2)/sqrt(2*pi);
36     for k=3:p
37         Psi_s(:,k) = y.*sqrt(factorial(k-2)/factorial(k-1)).*Psi_s(:,k-1) - (k-2)*sqrt(factorial(k-3)/
        factorial(k-1)).*Psi_s(:,k-2);
38         u_ref(:,k) = (mean_left-mean_right)/sqrt(k-1)*exp(-y.^2/2)/sqrt(2*pi).*Psi_s(:,k-1);
39     end
40 end
```

Index

A
advection-diffusion equation, 24
artificial dissipation, 92–93, 96

B
boundary conditions, 90, 119
Burgers' equation, 7, 81

C
conservation, 165
conservation laws, 7

D
diagonalization, 52

E
energy estimates, 90
entropy function, 83
Euler equations, 159

H
hybrid scheme, 158

I
interface, 163–165

J
Jacobian, 90

K
Karhunen-Loève expansion, 12
Kronecker product, 63–64

M
manufactured solutions, 135
method of manufactured solutions, 136
MUSCL scheme, 31

N
normal modal analysis, 56–57

P
polynomial chaos, 4
pseudospectral projection, 26

Q
quadrature, 25

R
Roe average matrix, 132–135
Roe variable, 130

S
shock-capturing, 38
simultaneous approximation term, 6
stability, 7, 31

M.P. Pettersson et al., *Polynomial Chaos Methods for Hyperbolic Partial Differential Equations*, Mathematical Engineering,
DOI 10.1007/978-3-319-10714-1

stiffness, 55, 68
stochastic collocation, 25
stochastic Galerkin, 7
stochastic hyperbolic problems, 126
strong stability, 92
summation by parts, 6

T
triangular distribution, 138
two-phase flow, 150–151

U
uncertainties, 3

V
von Neumann analysis, 60

W
wavelets, 16
well-posedness, 7, 55, 88