Spectral Element Library - CHQZ_lib Release 1.0 [1]

Paola Gervasio[2]

September, 21 2007

[1]CHQZ2: C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, *S*pectral Methods. Fundamentals in Single Domains. Springer Verlag, Berlin Heidelberg New York, 2006. CHQZ3: C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, *S*pectral Methods. Evolution to Complex Geometries and Applications to Fluid Dynamics Springer Verlag, Berlin Heidelberg New York, 2007.

[2]Department of Mathematics, University of Brescia, 25133 Brescia (Italy). `gervasio@ing.unibs.it`

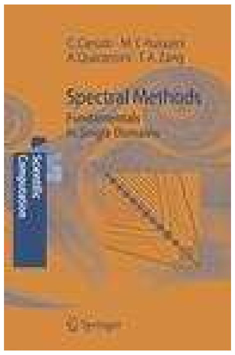
Chapter 1

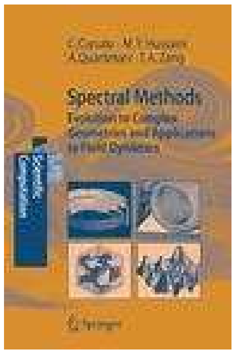
Introduction

Functions developed in this library provide the numerical solution of some simple boundary value problems in 1D, 2D and 3D geometries by either Spectral Methods with Galerkin-Numerical Integration (G-NI), in the case of single domain formulation, or Spectral Element Methods with Numerical Integration (SEM-NI), in the case of multi-domain formulation. Extrema eigenvalues of discrete operators are also computed in some cases.

The present library includes matlab files we have used to produce some numerical results published in the books:

C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, *Spectral Methods. Fundamentals in Single Domains* Springer Verlag, Berlin Heidelberg New York, 2006;





C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, *Spectral Methods. Evolution to Complex Geometries and Applications to Fluid Dynamics* Springer Verlag, Berlin Heidelberg New York, 2007.

Here is a summary of the features of the library:

1. Basic functions
   - Computation of nodes and weights of Legendre-Gauss-Lobatto (LGL), Chebyshev-Gauss-Lobatto (CGL), Legendre-Gauss (LG), Chebyshev-Gauss (CG) quadrature formulas.
   - Computation of 1st and 2nd Legendre/Chebyshev derivative matrices.
   - Legendre Transform.

  - Interpolation routines from either LGL or LG grid to another grid.
  - Evaluation and plot of Legendre, Lagrange and boundary-adapted modal basis functions.

2. Numerical solution of 1D Boundary Value Problems:
   - non periodic Burgers equation ($u_t + uu_x - \nu u_{xx} = 0$ in Ω, $\forall t > 0$, b.c. on $\partial\Omega$ and i.c. at $t = 0$),
   - scalar linear hyperbolic equation ($u_t + \beta u_x = 0$ in Ω, $\forall t > 0$, b.c. on $\partial\Omega$ and i.c. at $t = 0$),
   - second order elliptic equation ($-u'' + \beta u' + \gamma u = f$ in Ω, b.c. on $\partial\Omega$),
   - FEM preconditioners for spectral matrices for elliptic self-adjoint second-order equations ($-u''+\gamma u = f$ in Ω, b.c. on $\partial\Omega$),
3. 2D Boundary Value Problems on rectangular geometries:
   - diffusion-reaction problems ($-\nu\Delta u + \gamma u = f$ in Ω, b.c. on $\partial\Omega$),
   - additive Schwarz preconditioner (with overlap and coarse mesh) for diffusion-reaction problems
   - Neumann-Neumann and Balancing Neumann-Neumann preconditioners for the Schur complement matrix associated to diffusion-reaction problems
4. 3D Boundary Value Problems on parallelepiped geometries:
   - diffusion-reaction problems ($-\nu\Delta u + \gamma u = f$ in Ω, b.c. on $\partial\Omega$).

1.1 Download

The library can be downloaded from http://dm.ing.unibs.it/gervasio. It is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should receive a copy of the GNU General Public License along with this library; if not, write to the Free Software Foundation Inc, 59 Temple Pl. - Suite 330, Boston, MA 02111-1307, USA.

1.2 Requirements

1. Matlab 7.2.sp (R14) or above, since handle functions are used to evaluate mathematical functions. The Symbolic Toolbox is used to set input data, but it is not needed to solve differential problems. Previous releases of Matlab could be used, provided that handle functions are replaced by calls to either `eval` or `feval` and that you replace calls to Symbolic Toolbox functions with exact derivates, computed at hand.

2. Operating system: any.

1.3 How to install

1. Download the latest CHQZ_lib.zip on the website:
   http://dm.ing.unibs.it/gervasio

2. Extract files in your Matlab Repository /home/foo/matlab/ :
   `unzip CHQZ_lib_1.0.zip`

3. Add all CHQZ_lib_1.0 subdirectories in your Matlab path:

```
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_0
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_1
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_2
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_3
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Basis\_functions
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Hyperbolic_1d
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Burgers
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_1d
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_2d
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_2d/Schur
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_2d/Schwarz
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_3d
```

4. Ready to use.

Remark 1.3.1 You can add to your Matlab path only directories needed by a specific script (or function). For example, if you want to call a function belonging to subdirectory `CHQZ_lib_1.0/Src/Elliptic_2d/Schur`, you may add only directories:

```
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_0
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_2
addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_2d/Schur
```

Functions of directory `Level_0` are always needed.
To call functions for 1D problems, you have to add directory `Level_1` and the directory the function belongs to;
to call functions for 2D problems, you have to add directories `Level_2` and the directory the function belongs to;
to call functions for 3D problems, you have to add directories `Level_3` and the directory the function belongs to.

1.4 On-line documentation

On-line documentation has been produced by using M2HTML (Copyright ©2003 Guillaume Flandin `Guillaume@artefact.tk`). Open the file `/home/foo/matlab/CHQZ_lib_1.0/Doc/Html/index.html` by either a web-browser or matlab editor.

Chapter 2

A simple example

We want to solve the 2D Poisson problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \tag{2.1}$$

where $\Omega = (-2,2) \times (0,1)$, $g(x,y) = \sin(\pi x) cos(\pi y)$, $f(x,y) = -2\pi^2 \sin(\pi x) cos(\pi y)$. We choose a discretization of Ω in 4×2 rectangular elements while the polynomial degrees are $N_x = 12$ (along x-direction) and $N_y = 6$ (along y-direction).

1. Start matlab
2. add some directories to matlab path:

   ```
   addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_0
   addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Level_2
   addpath /home/foo/matlab/CHQZ_lib_1.0/Src/Elliptic_2d
   ```

3. set input data (you can modify the file `Elliptic_2d/call_lap_2d`):

   ```
   uex=@(x,y)[sin(pi*x).*cos(pi*y)]; % exact solution =  boundary data
   uex_x=@(x,y)[pi*cos(pi*x).*cos(pi*y)]; % du/dx
   uex_y=@(x,y)[-pi*sin(pi*x).*sin(pi*y)]; % du/dy
   f=@(x,y)[2*pi^2*sin(pi*x).*cos(pi*y)];  % r.h.s
   g=@(x,y)[sin(pi*x).*cos(pi*y)]; % Dirichlet boundary data
   h=@(x,y)[pi*sin(pi*x).*sin(pi*y);pi*cos(pi*x).*cos(pi*y);...
   -pi*sin(pi*x).*sin(pi*y);-pi*cos(pi*x).*cos(pi*y)]; % Neumann boundary data
   gam=0;  % coefficient of the term of order zero.
   xa=-2;xb=2;   % Omega=(xa,xb) x (ya,yb)
   ya=0;yb=1;
   cb='nndd'  % For boundary conditions cb(i)='d' ---> Dirichlet on side i
              % cb(i)='n' ----> Neumann on side i
   nex=4;ney=2; nx=12; ny=6;
   param=zeros(20,1);  % "help lap_2d" for a complete description of param
   param(1)=1;    % 1=SEM-NI,  2= Patching
   param(2)=2;    % 0=no reordering, 1=CM ordering, 2=AMD ordering
   param(3)=3;    % 1= solve linear system by Cholesky fact.
                  % 2= compute extrema eigenvalues of A
                  % 3 solve by Schur complement
                  % 4= compute extrema eigenvalues of the Schur complement
   ```

```
param(4)=1;   % computes errors
param(5)=1;    % 0 exact norms, 1= discrete norms
param(6)=nx*2;   % nq for LG quadrature formulas
param(7)=1;    % 0 =absolute errors, 1=relative errors
param(8)=2;    % 0 no plot, 1 mesh, 2 surf, 3 contour
param(9)=(nx+1); % nodes used to plot numerical solution
gammax=[]; gammay=[]; % if SEM decomposition is not uniform:
                      % they are the arrays with intefaces positions
                      % along x- and y- directions, respectively
```

4. call the function `lap_2d`

```
[xy,un,D,param]=lap_2d(xa,xb,ya,yb,gam,uex,uex_x,uex_y,f,g,h,cb,...
           nex,nx,ney,ny,gammax,gammay,param);
```

5. print the errors

```
fprintf('nx=%d,nex=%d,err_inf=%11.4e, err_h1=%11.4e,err_l2=%11.4e \n',...
    nx,nex,param(29),param(30),param(31))
```

The 2-indexes arrays `xy` contains coordinates of the nodes of the mesh used, while `un` contains numerical solution. `param(29), param(30), param(31)` are the relative errors between exact and numerical solution with respect to $L^\infty(\Omega)$-norm, $H^1(\Omega)$-norm and $L^2(\Omega)$-norm, respectively. A figure has been generated with the plot of the numerical solution, in particular the command `surf` has been used.

Chapter 3

Notations

From now on, we will use the following notations:

CGL	Chebyshev-Gauss-Lobatto
CG	Chebyshev-Gauss
LGL	Legendre-Gauss-Lobatto
LG	Legendre-Gauss
G-NI	Galerkin formulation with Numerical Integration
SEM	Spectral Element Method
SEM-NI	Spectral Element Method with Numerical Integration

3.1 Functions setting

All mathematical functions, such as exact solution, right hand side, variable coefficients, are referred by following the function_handle syntax. If you want to define the function $f(x) = \sqrt{\pi x + 2}$, the matlab instruction do to this is:

```
f=@(x)[sqrt(pi*x+2)];
```

Matlab instruction to evaluate the function f at $x = 4$ is

```
y=f(4);
```

If you want to define the function $u(x, y) = \sin(\pi x)\cos(\pi y)$ the matlab instruction do to this is:

```
u=@(x,y)[sin(pi*x).*cos(pi*y)];
```

Matlab instruction to evaluate the function u at $(x, y) = (0.5, 0.3)$ is

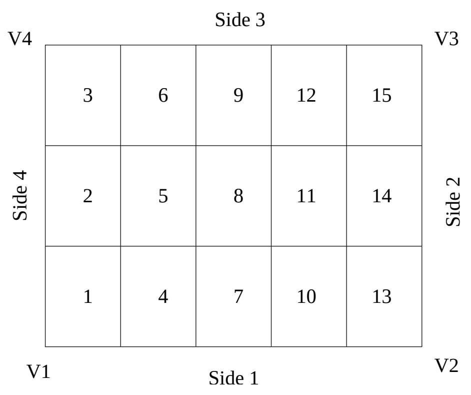
```
z=u(0.5,0.3);
```

3.2 1D b.v.p. defaults

For 1D problems, nodes are ordered from left to right. Spectral elements are ordered from left to right, too. Only uniform decompositions are provided, i.e. the elements have equal size and equal number of nodes.

In the following list we show the correspondence between commonly used variables and their meaning.

`xa`	left extreme of Ω
`xb`	right extreme of Ω
`ne`	number of spectral elements in Ω



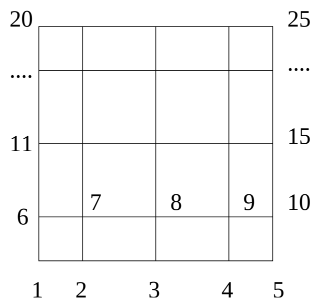


Figure 3.1: Elements ordering for 2D boundary value problems (left), nodes ordering inside each spectral element (right).

`nx`	polynomial degree in each spectral element (the same in each element)
`npdx`	number of nodes in each spectral element (`npdx=nx+1`)
`noe`	global number of nodes in Ω
`xy`	column array with global nodes coordinates in Ω
`x`	column array with `npdx` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$
`w`	column array with `npdx` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1, 1]$
`dx`	first derivative LGL (or CGL, or LG, or CG) matrix
`nov`	two indexes array whose size is `(npdx,ne)` which implements the restriction/extension map from element local-meshes to global mesh. i.e. `ig=nov(il,m)`: `ig` is the global-mesh index associated to node `il` of element `m`.
`cb`	two elements character array for setting boundary conditions on $\partial\Omega$. The character `cb(1)` is associated to the left extreme of Ω, while `cb(2)` is associated to the right extreme of Ω. `cb(i)='d'` stands for Dirichlet boundary conditions at extreme i, while `cb(i)='n'` stands for Neumann boundary conditions at extreme i.
`param`	row arrow containing scalar input parameters used for selecting an approach instead of another. Some times `param` is used to pass scalar outputs, such number of CG iterations, errors, extrema eigenvalues

3.3 2D b.v.p. defaults

For 2D problems, only rectangular geometries $\Omega = (x_a, x_b) \times (y_a, y_b)$ are provided.
The nodes are ordered following local element by element order. The elements are ordered first along y-direction, then along x-direction, as shown in Fig. 3.1. On the contrary, inside each element, the nodes are ordered first along x-direction, then along y-direction (lexicographical order).

Uniform and non-uniform decompositions are provided, i.e. the elements may have different sizes, nevertheless the local number of nodes along single directions are the same in each spectral element. You can chose two different numbers of nodes along x and y directions.
The sides of Ω are sorted as shown in Fig. 3.1.
The vertexes of both Ω and each spectral element are counterclockwise ordered, starting from the bottom-left vertex.

In the following list we show the correspondence between commonly used variables and their meaning.

`nex`	number of spectral elements in Ω along x-direction
`ney`	number of spectral elements in Ω along y-direction
`ne`	global number of spectral elements in Ω
`xx`	2-indexes array of size `(4,ne)`. `xx(1:4,m)=[x_V1_m;x_V2_m;x_V3_m;x_V4_m]`, where `x_Vi_m` denotes the abscissa of vertex i in element m.
`yy`	2-indexes array of size `(4,ne)`. `yy(1:4,m)=[y_V1_m;y_V2_m;y_V3_m;y_V4_m]`, where `y_Vi_m` denotes the ordinate of vertex i in element m.
`nx`	polynomial degree along x-direction in each spectral element (the same in each element)
`npdx`	number of nodes along x-direction in each spectral element (`npdx=nx+1`)
`ny`	polynomial degree along y-direction in each spectral element (the same in each element)
`npdy`	number of nodes along y-direction in each spectral element (`npdy=ny+1`)
`noe`	global number of nodes in Ω
`xy`	two-indexes array (of length `noe`) with global nodes coordinates (x, y) in Ω
`x`	column array with `npdx` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$
`wx`	column array with `npdx` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1, 1]$
`dx`	first x-derivative LGL (or CGL, or LG, or CG) matrix
`y`	column array with `npdy` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$
`wy`	column array with `npdy` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1, 1]$
`dy`	first y-derivative LGL (or CGL, or LG, or CG) matrix
`ldnov`	number of nodes in each spectral element
`nov`	two indexes array whose size is `(ldnov,ne)` which implements the restriction/extension map from element local-meshes to global mesh. i.e. `ig=nov(il,m)`: `ig` is the global-mesh index associated to node `il` of element `m`.
`cb`	four elements character array for setting boundary conditions on $\partial\Omega$. Each character of `cb` is associated to a side of $\partial\Omega$ (`cb(i)` is associated to side i of Ω) and `cb(i)='d'` stands for Dirichlet boundary conditions on Side i, while `cb(i)='n'` stands for Neumann boundary conditions on Side i. A Dirichlet boundary condition dominates Neumann boundary condition at vertexes.

`ifro`	Column array of length `noe` with values `0, 1, -1, 31`.
	`ifro(i)=0` means that node `i` is internal to Ω
	`ifro(i)=1` means that node `i` is on $\partial\Omega$ and a Dirichlet boundary condition is imposed at node `i`
	`ifro(i)=-1` means that node `i` is internal to Ω and it belongs to interface Γ between spectral elements
	`ifro(i)=31` means that node `i` is on $\partial\Omega$ and a Neumann boundary condition is imposed at node `i`

`param`	row arrow containing scalar input parameters used for selecting an approach instead of another. Some times `param` is used to pass scalar outputs, such number of CG iterations, errors, extrema eigenvalues

3.4 3D b.v.p. defaults

For 3D problems, only parallelepiped geometries $\Omega = (x_a, x_b) \times (y_a, y_b) \times (z_a, z_b)$ are provided.
The nodes are ordered following local element by element order. The elements are ordered first along z-direction, then along y-direction and finally along x-direction. Inside each element nodes are ordered first along x-direction, then along y-direction and finally along z-direction.

Uniform and non-uniform decompositions are provided, i.e. the elements may have different sizes, nevertheless the local number of nodes along single directions are the same in each spectral element. You can chose different numbers of nodes along x, y and z directions.
The vertexes of both Ω and each spectral element are counterclockwise ordered, starting from the bottom-left vertex, first on bottom face and then on top face.

In the following list we show the correspondence between commonly used variables and their meaning.

`nex`	number of spectral elements in Ω along x-direction
`ney`	number of spectral elements in Ω along y-direction
`nez`	number of spectral elements in Ω along z-direction
`ne`	global number of spectral elements in Ω
`xx`	2-indexes array of size `(8,ne)`. `xx(1:8,m)=[x_V1_m;x_V2_m;x_V3_m;x_V4_m; x_V5_m;x_V6_m;x_V7_m;x_V8_m]`, where `x_Vi_m` denotes the abscissa of vertex i in element m.
`yy`	2-indexes array of size `(8,ne)`. `yy(1:8,m)=[y_V1_m;y_V2_m;y_V3_m;y_V4_m; y_V5_m;y_V6_m;y_V7_m;y_V8_m]`, where `y_Vi_m` denotes the ordinate of vertex i in element m.
`zz`	2-indexes array of size `(8,ne)`. `zz(1:8,m)=[z_V1_m;z_V2_m;z_V3_m;z_V4_m; z_V5_m;z_V6_m;z_V7_m;z_V8_m]`, where `z_Vi_m` denotes the third coordinate of vertex i in element m.
`nx`	polynomial degree along x-direction in each spectral element (the same in each element)
`npdx`	number of nodes along x-direction in each spectral element (`npdx=nx+1`)
`ny`	polynomial degree along y-direction in each spectral element (the same in each element)
`npdy`	number of nodes along y-direction in each spectral element (`npdy=ny+1`)
`nz`	polynomial degree along z-direction in each spectral element (the same in each element)
`npdz`	number of nodes along z-direction in each spectral element (`npdz=nz+1`)
`noe`	global number of nodes in Ω
`xyz`	two-indexes array (of length `noe`) with global nodes coordinates (x, y, z) in Ω
`x`	column array with `npdx` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$
`wx`	column array with `npdx` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1, 1]$
`dx`	first x-derivative LGL (or CGL, or LG, or CG) matrix
`y`	column array with `npdy` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$
`wy`	column array with `npdy` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1, 1]$
`dy`	first y-derivative LGL (or CGL, or LG, or CG) matrix
`z`	column array with `npdz` LGL (or CGL, or LG, or CG) nodes on the reference interval $[-1, 1]$

`wz`	column array with `npdz` LGL (or CGL, or LG, or CG) weights on the reference interval $[-1,1]$
`dz`	first z-derivative LGL (or CGL, or LG, or CG) matrix
`ldnov`	number of nodes in each spectral element
`nov`	two indexes array whose size is `(ldnov,ne)` which implements the restriction/extension map from element local-meshes to global mesh. i.e. `ig=nov(il,m)`: `ig` is the global-mesh index associated to node `il` of element `m`.
`ifro`	Column array of length `noe` with values `0, 1`. `ifro(i)=0` means that node `i` is internal to Ω `ifro(i)=1` means that node `i` is on $\partial\Omega$ and a Dirichlet boundary condition is imposed at node `i`
`param`	row arrow containing scalar input parameters used for selecting an approach instead of another. Some times `param` is used to pass scalar outputs, such number of CG iterations, errors, extrema eigenvalues

Chapter 4

Structure of the library

The library is organized in several directories:

- CHQZ_lib_1.0/Src/Level_0
- CHQZ_lib_1.0/Src/Level_1
- CHQZ_lib_1.0/Src/Level_2
- CHQZ_lib_1.0/Src/Level_3
- CHQZ_lib_1.0/Src/Basis_functions
- CHQZ_lib_1.0/Src/Burgers
- CHQZ_lib_1.0/Src/Eigenvalues_1d
- CHQZ_lib_1.0/Src/Hyperbolic_1d
- CHQZ_lib_1.0/Src/Elliptic_1d
- CHQZ_lib_1.0/Src/Elliptic_2d
- CHQZ_lib_1.0/Src/Elliptic_2d/Schur
- CHQZ_lib_1.0/Src/Elliptic_2d/Schwarz
- CHQZ_lib_1.0/Src/Elliptic_3d

4.1 `Level_0` directory

The directory `Level_0` consists of a number of functions to

- generate quadrature nodes and weights,
- assemble derivative matrices,
- interpolate data and functions from LGL (or LG) grids to other grids,
- evaluate Legendre transform,
- evaluate Legendre, Chebyshev (and all other Jacoby) polynomials, Lagrange polynomials, boundary adapted Legendre polynomials.

These functions have been written according to notations, identities and formulas reported in [1, Ch.1, Ch. 2] and in [3, Ch. 4].

These functions are called by many routines of CHQZ_lib_1.0.

The following is a list of the functions currently supported with a brief explanation.

`chebyshev_pol`
: Plots Chebyshev polynomials for $n = 0, ..., 4$

`[d]=der2cgl(x,np)`
: Spectral (Chebyshev Gauss Lobatto) second derivative matrix

`[d]=der2lgl(x,np)`
: Spectral (Legendre Gauss Lobatto) second derivative matrix

`[d]=dercgl(x,np)`
: Spectral (Chebyshev Gauss Lobatto) derivative matrix

`[d]=derlg(x,np)`
: Spectral (Legendre Gauss) derivative matrix

`[d]=derlgl(x,np)`
: Spectral (Legendre Gauss Lobatto) derivative matrix

`[a]=intlag_cgl(x_cgl, x_new)`
: Computes matrix `a` to evaluate 1D Lagrange interpolant at CGL

`[a]=intlag_lg(x_lg, w_lg, x_new)`
: Computes matrix `a` to evaluate 1D Lagrange interpolant at LG

`[a]=intlag_lgl(x_lgl, x_new)`
: Computes matrix `a` to evaluate 1D Lagrange interpolant at LGL

`[p,pd] = jacobi_eval(x,n,alpha,beta)`
: Evaluates Jacobi polynomial $P_n(\alpha, \beta)$ and its first derivative at x

`jacobi_pol`
: Script for plotting some Jacobi polynomials for $n = 4$

`[x,flag] = jacobi_roots(n,alpha,beta)`
: Computes the n zeros of the Jacoby polynomial $P_n(\alpha, \beta)(x)$

`legendre_pol`
: Plots Legendre polynomials for $n = 0, ..., 4$

`[uk]=legendre_tr_coef(x,u)`

Computes Discrete Legendre Transform coefficients

`[u_int]=legendre_tr_eval(x,u,x_int)`
: Evaluates Discrete Legendre Transform

`[a]=legendre_tr_matrix(x)`
: Computes matrix `a` to evaluate Discrete Legendre Transform

`[p] = pnleg (x, n)`
: Evaluates Legendre polynomial of degree n

`[p1,p] = pnleg1 (x, n)`
: Evaluates the first derivative of Legendre polynomial of degree n

`[p2,p1,p] = pnleg2 (x, n)`
: Evaluates the second derivative of Legendre polynomial of degree n

`[p] = pnleg_all(x,n)`
: Evaluates Legendre polynomials, from degree 0 to n

`test`
: Script for testing all functions of this directory

`[x,w] = xwcg(np,a,b)`
: Computes nodes and weights of the Chebyshev-Gauss quadrature formula

`[x,w] = xwcgl(np,a,b)`
: Computes nodes and weights of the Chebyshev-Gauss-Lobatto quadrature formula.

`[x,w] = xwlg(np,a,b)`
: Computes nodes and weights of the Legendre-Gauss quadrature formula.

`[x,w] = xwlgl(np,a,b)`
: Computes nodes and weights of the Legendre-Gauss-Lobatto quadrature formula.

A dependency-graph for this directory is shown in Fig. 4.1

4.2 `Level_1` directory

The directory `Level_1` consists of a number of functions to

- built and assemble SEM-NI mass and stiffness matrices related to 1D boundary value problems
- generate 1D SEM mesh structures
- evaluate errors in L^∞-, H^1-, L^2-norms between numerical and exact solution of 1D boundary value problems

These functions have been written according to notations, identities and formulas reported in [1, Ch. 3].

They are called by several functions of directories `Burgers, Eigenvalues_1d, Elliptic_1d, Hyperbolic_1d`.

The following is a list of the functions currently supported with a brief explanation.

`[A]=ad_1d_se(npdx,ne,nov,nu,beta,wx,dx,jacx)`
: Assembles 1D global SEM-NI matrix associated to the advection diffusion operator $-\nu u'' + \beta u'$ with constant ν and β

`[A]=ad_1d_sp(nu,beta,wx,dx,jacx)`
: Computes 1D local SEM-NI matrix associated to the advection diffusion operator $-\nu u'' + \beta u'$ with constant ν and β

`[A]=adr_1d_se(npdx,ne,nov,nu,b,gam,wx,dx,jacx)`
: Assembles 1D global SEM-NI element matrix associated to the advection-diffusion-reaction operator $-(\nu u' + b(x)u)' + \gamma u$, in divergence form, with constant ν and γ.

`[A]=adr_1d_sp(wx,dx,jacx,nu,b,gam)`
: Computes 1D local SEM-NI matrix associated to the advection-diffusion-reaction operator $-(\nu u' + b(x)u)' + \gamma u$, in divergence form, with ν and γ constants.

`[nov]=cosnov_1d(npdx,ne,nov)`
: Constructs the 1D (local mesh $\rightarrow$ global mesh) map

`[A]=ell_1d_se(npdx,ne,nov,nu,beta,gam,wx,dx,jacx)`
: Assembles 1D global SEM-NI matrix associated to the advection-diffusion-reaction operator $-\nu u'' + \beta u' + \gamma u$

`[A]=ell_1d_sp(nu,beta,gam,wx,dx,jacx)`
: Computes 1D local SEM-NI matrix associated to the advection-diffusion-reaction operator $-\nu u'' + \beta u' + \gamma u$

`[err_inf,err_h1,err_l2]=errors_1d(nx,ne,xa,xb,un,uex,uexx,param)`
: Computes errors for 1D boundary value problems

`[A,M]=matrices_1d(xa,xb,cb,ne,nx)`
: Assembles SEM-NI stiffness and mass matrices for 1D b.v.p.

`[xx,jacx,xy,ww]=mesh_1d(xa,xb,ne,npdx,nov,x,wx)`
: Generates uniform 1D spectral elements mesh

`[err_h1]=normah1_1d(fdq, nq, errtype, u, uex, uexx, x, wx, dx, xx, jacx, xy,nov)`
: Computes H1-norm in 1D domains

`[err_l2]=normal2_1d(fdq, nq, errtype, u, uex, x, wx, xx, jacx, xy,nov)`
: Computes L2-norm in 1D domains

`[ha]=plot_sem_1d(fig,ne,x,wx,xx,jacx,xy,nov,un,n_int)`
: Plots SEM-NI solution of 1D boundary value problems

`[uex,uexx,ff,nu,beta,gam]=setfun_adr_1d`
: Sets functions and coefficients for calling `adr_1d`

`[uex,uexx,ff,nu,beta,gam]=setfun_ell_1d`
: Sets functions and coefficients for calling `ell_1d` and `ellprecofem_1d`

`[uex,uexx,ff,nu,gam]=setfun_lap_1d`
: Sets functions and coefficients for calling `lap_1d` and

`[A]=stiff_1d_se(npdx,ne,nov,wx,dx,jacx)`
: Assembles 1D global stiffness SEM-NI matrix $(\varphi_j', \varphi_i')_\Omega$

`[A]=stiff_1d_sp(w,d,jac)`

Computes 1D local stiffness matrix $(\varphi'_j, \varphi'_i)_{N,\Omega_m}$

A dependency-graph for this directory is shown in Fig. 4.2

4.3 `Level_2` directory

The directory `Level_2` consists of a number of functions to

- built and assemble SEM-NI mass and stiffness matrices related to 2D boundary value problems
- generate 2D SEM mesh structures
- evaluate errors in L^∞-, H^1-, L^2-norms between numerical and exact solution of 2D boundary value problems
- plot numerical solution of 2D boundary value problems

These functions have been written according to notations, identities and formulas reported in [1, Ch. 3].

These functions are called by several functions of directories `Elliptic_2d, Elliptic_2d/Schur, Elliptic_2d/Schwarz`.

The following is a list of the functions currently supported with a brief explanation.

`[nov]=cosnov_2d(npdx,nex,npdy,ney)`

Constructs the 2D (local mesh → global mesh) map

```
[err_inf,err_h1,err_l2]=errors_2d(x,wx,dx,xx,jacx,y,wy,dy,yy,jacy,
     xy,ww,nov,un,uex,uex_x,uex_y,param)
```

Computes errors for 2D boundary value problems

`[lbor,lint,lintint,lgamma,ifro]=liste(ifro,nov)`

Assembles lists of internal, boundary, interface nodes

`[lbor,lint,lintint,lgamma]=liste1(ifro)`

Assembles lists of internal, boundary, interface nodes (similar to liste)

```
[xx,yy,jacx,jacy,xy,ww,ifro]=mesh2d(xa,xb,ya,yb,cb,nex,ney,npdx,npdy,
     nov,x,wx,y,wy,gammax,gammay)
```

Constructs uniform 2D SEM mesh on rectangular domain $\Omega = (x_a, x_b) \times (y_a, y_b)$

```
[err_h1]=normah1_2d(fdq,nq,errtype,x,wx,dx,xx,jacx,y,wy,dy,yy,jacy,
     xy,ww,nov,un,u,uex,uex_x,uex_y);
```

Computes H1-norm in 2D domains

```
[err_l2]=normal2_2d(fdq,nq,errtype,x,wx,xx,jacx,y,wy,yy,jacy,
     xy,ww,nov,un,u,uex)
```

Computes L2-norm in 2D domains

`[ha]=plot_sem_2d(fig,command,nex,ney,x,xx,jacx,y,yy,jacy,xy,ww,nov, u,n_int)`

Plots SEM numerical solution of 2D boundary value problems

`[uex,uexx,uexy,ff,gam]=setfun_lap_2`

Sets functions and coefficients for calling `lap_2d`

`[A]=stiff_2d_se(npdx,nex,npdy,ney,nov,wx,dx,jacx,wy,dy,jacy)`

Assembles 2D global stiffness SEM-NI matrix $(\nabla\varphi_j, \nabla\varphi_i)_\Omega$

`[A]=stiff_2d_sp(wx,dx,jacx,wy,dy,jacy)`

Computes 2D local stiffness SEM-NI matrix $(\nabla\varphi_j, \nabla\varphi_i)_N$

A dependency-graph for this directory is shown in Fig. 4.3

4.4 `Level_3` directory

The directory `Level_3` consists of a number of functions to

- built and assemble SEM-NI mass and stiffness matrices related to 3D boundary value problems
- generate 3D SEM mesh structures
- evaluate errors in L^∞-, H^1-, L^2-norms between numerical and exact solution of 3D boundary value problems

These functions are called by functions of directories `Elliptic_3d`.

The following is a list of the functions currently supported with a brief explanation.

`[err_inf,err_h1,err_l2]=errors_3d(x,wx,dx,xx,jacx,y,wy,dy,yy,jacy, z,wz,dz,zz,jacz,`

`xyz,ww,nov,un,uex,uex_x,uex_y,uex_z,param)`

Computes errors for 3D boundary value problems

`[xx,yy,zz,jacx,jacy,jacz,xyz,ww,ifro,nov]=mesh3d(xa,xb,ya,yb,za,zb, nex,ney,nez,`
`npdx,npdy,npdz,x,wx,y,wy,z,wz,gammax,gammay,gammaz)`

Computes uniform 3D Spectral element mesh on parallelepiped $\Omega = (x_a, x_b) \times (y_a, y_b) \times (z_a, z_b)$

`[err_h1]=normah1_3d(fdq,nq,errtype,x,wx,dx,xx,jacx,y,wy,dy,yy,jacy,`
`z,wz,dz,zz,jacz,xyz,ww,nov,un,u,uex,uex_x,uex_y,uex_z)`

Computes H1-norm in 3D domains

`[err_l2]=normal2_3d(fdq,nq,errtype,x,wx,xx,jacx,y,wy,yy,jacy,`
`z,wz,dz,zz,jacz,xyz,ww,nov,un,u,uex)`

Computes L2-norm in 3D domains

`[uex,uexx,uexy,uexz,ff,gam]=setfun_lap_3d`

Sets functions and coefficients for calling `lap_3d`

`[A]=stiff_3d_se(npdx,nex,npdy,ney,npdz,nez,nov,wx,dx,jacx,wy,dy,jacy,wz,dz,jacz)`

Assembles 3D global stiffness SEM matrix $(\nabla\varphi_j, \nabla\varphi_i)_\Omega$

`[A]=stiff_3d_sp(wx,dx,jacx,wy,dy,jacy,wz,dz,jacz)`

Computes 3D local stiffness SEM-NI matrix $(\nabla\varphi_j, \nabla\varphi_i)_N$

These functions have been written according to notations, identities and formulas reported in [1, Ch. 3].
A dependency-graph for this directory is shown in Fig. 4.4

4.5 `Basis_functions` directory

The directory `Basis_functions` consists of a number of functions to

- plot 1D polynomial basis functions: Lagrange, modal Legendre, boundary-adapted modal Legendre polynomials
- plot 2D polynomial basis functions: Lagrange, boundary-adapted modal polynomials

These functions have been written according to notations, identities and formulas reported in [1, Ch. 1, Ch. 2].
The following is a list of the functions currently supported with a brief explanation.

`[dlnp1]=derpol_legendre(n,ln,dln,dlnm1)`

Recursive construction of first derivative of Legendre polynomials, formula (2.3.19), pag. 77, [1]

`plot_2dlagrange`

Plots 2D Lagrange polynomials, formula (1.2.55), pag. 17, [1] (tensorial product), produces part of Fig. 2.13, pag. 100 [1]

`plot_2dmodal`

Plots 2D modal boundary adapted polynomials, formula (2.3.31), pag. 82, [1] (tensorial product), produces part of Fig. 2.13, pag. 100 [1]

`plot_lagrange`

Plots 1D Lagrange polynomials, formula (1.2.55), pag. 17, [1], produces part of Fig. 2.12, pag. 83 [1]

`plot_legendre`

Plots 1D Legendre polynomials, formula (2.3.2), pag. 75, [1], produces Fig. 2.12, pag. 83 [1]

`plot_modal`

Plots 1D modal boundary-adapted polynomials, formula (2.3.31), pag. 82, [1], produces Fig. 2.12, pag. 83 [1]

`[lnp1]=pol_legendre(n,ln,lnm1)`

Recursive construction of Legendre basis function, formula (2.3.19), pag. 77, [1] `[etak]=pol_modal(k,lk,lkm2)`

Recursive construction of modal basis function, formula (2.3.31), pag. 82, [1]

`test`

call all callable functions of directory `Basis_functions`

A dependency-graph for this directory is shown in Fig. 4.5

4.6 `Burgers` directory

The directory `Burgers` consists of a number of functions to approximate the solution of non periodic Burgers equation [1, Sect. 3.1],

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 \qquad \text{in } \Omega, \qquad \forall t > 0 \tag{4.1}$$

that satisfies the no-flux boundary conditions

$$\frac{1}{2}u^2 - \nu\frac{\partial u}{\partial x} = 0 \quad \text{at } x = \pm 1, \qquad \forall t > 0,$$

and a suitable initial condition at $t = 0$. The approximation is based on the Legendre Galerkin with Numerical Integration (G-NI) method [1, Sect. 3.3.5].

The following is a list of the functions currently supported with a brief explanation.

`call_npbur`

Script to set input data and to call `nonperiodic_burgers`

`[u1]=ee(tt,deltat,f,u0,dx,A,w,visc,uex,uex1,bc)`

Performs one step of Explicit Euler scheme for 1D parabolic problems

`[f]=fburgers(tt,deltat,u0,A,dx,w,visc,uex,uex1,bc)`

Defines the non perodic Burgers function for `nonperiodic_burgers`

`[u,err]=nonperiodic_burgers(xa,xb,t0,T,visc,nx,deltat,tscheme,bc)`

Numerical solution of non periodic Burgers equation (4.1)

`[u1]=rk2(t,deltat,f,u0,dx,A,w,visc,uex,uex1,bc)`

Performs one step of explicit 2nd order Runge-Kutta scheme for 1D parabolic problems

`[u1]=rk4(t,deltat,f,u0,dx,A,w,visc,uex,uex1,bc)`

Performs one step of explicit 4th order Runge-Kutta scheme for 1D parabolic problems

A dependency-graph for this directory is shown in Fig. 4.6

4.7 `Eigenvalues_1d` directory

The directory `Eigenvalues_1d` consists of a number of functions to numerically compute eigenvalues of first and second derivative matrices of Legendre G-NI, Legendre collocation, Chebyshev collocation approaches [1, Sect. 4.3].

The following is a list of the functions currently supported with a brief explanation.

`fig4_10`

Legendre collocation first-derivative eigenvalues computation and plot. Script to produce Fig 4.10, pag. 203 [1]

`fig4_12`
: Legendre collocation/ G-NI / generalized G-NI first-derivative eigenvalues computation and plot. Script to produce Fig 4.12 (top-left) and Fig 4.13 (top-left) pag. 204 [1]

`fig4_14`
: Legendre collocation first-derivative eigenvalues and spectral condition number. Script to produce Fig 4.14, pag. 206 [1]

`fig4_7`
: Extreme eigenvalues of Legendre G-NI stiffness matrices for the 2nd order derivative operator. Script to produce Fig 4.7, pag. 199 [1]

`fig4_8`
: Chebyshev collocation first-derivative eigenvalues computation and plot. Script to produce Fig 4.8, pag. 201 [1]

`[d,A]=lgl_eig(nx,nu,pbl)`
: Computes eigenvalues of first/second order spectral derivative matrices: collocation/G-NI, LGL nodes, 1D mono domain.

A dependency-graph for this directory is shown in Fig. 4.7

4.8 `Hyperbolic_1d` directory

The directory `Hyperbolic_1d` consists of a number of functions to approximate the solution of the linear scalar hyperbolic problem [1, Sect. 3.7],

$$\frac{\partial u}{\partial t} + \beta\frac{\partial u}{\partial x} = 0 \qquad \text{in } \Omega = (-1,1), \qquad \forall t > 0 \tag{4.2}$$

with constant $\beta > 0$, satisfying the inflow condition $u(-1,t) = u_L(t), \quad \forall t > 0$, and a suitable initial condition at $t = 0$. The approximation is based on either strong collocation approach, or the Legendre Galerkin with Numerical Integration (G-NI) method, or the penalty approach, or the staggered-grid method. The discretization in time is based on the explicit fourth order Runge-Kutta method.

Also the stationary counterpart of (4.2) has been taken into account.

The following is a list of the functions currently supported with a brief explanation.

`call_hyp`
: Script to set input data and call `scalar_hyp` and `stag_scalar_hyp`

`call_stat_hyp`
: Script to set input data and call `stat_scalar_hyp`

`[x,u,err,Psi,Phi]=scalar_hyp(xa,xb,t0,T,beta,uex,u0,ul,nx,deltat,param)`
: Numerical solution of scalar linear hyperbolic equations, formula (3.7.1a), pag. 145, [1]

`[int]=simpcx(xx,yy)`
: Composite Simpson Quadrature Formula

`[x,u,err,Psi,Phi]=stag_scalar_hyp(xa,xb,t0,T,beta,uex,u0,ul,nx,deltat)`
: Numerical solution of scalar linear hyperbolic equations, formula (3.7.1a), pag. 145, [1] by staggered grids method (pag. 149, [1])

`[x,u,err]=stat_scalar_hyp(xa,xb,beta,f,uex,ul,nx,param)`
: Numerical solution of a stationary scalar linear hyperbolic equation, formula (3.7.1a), pag. 145, [1]

A dependency-graph for this directory is shown in Fig. 4.8

4.9 `Elliptic_1d` directory

The directory `Elliptic_1d` consists of a number of functions to approximate the solution of 1D linear elliptic second order problems following SEM-NI approach [1, Ch. 4], [2, Ch. 5]:
problem (1.2.52)-(1.2.53), pag. 17, [1]

$$\begin{cases} -(\nu u' + \beta(x)u)' + \gamma u = f & x_a < x < x_b \\ \text{Neumann or Dirichlet b.c.} & \text{at } x = x_a, x = x_b, \end{cases} \tag{4.3}$$

with constants $\nu > 0$ and $\gamma \geq 0$;

$$\begin{cases} -\nu \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + \beta \frac{\mathrm{d}u}{\mathrm{d}x} + \gamma u = f & x_a < x < x_b \\ \text{Neumann or Dirichlet b.c.} & \text{at } x = x_a, x = x_b \end{cases} \tag{4.4}$$

with constants $\nu > 0$, $\beta \in \mathbb{R}$ and $\gamma \geq 0$;

$$\begin{cases} -\nu \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + \gamma u = f & x_a < x < x_b \\ \text{Neumann or Dirichlet b.c.} & \text{at } x = x_a, x = x_b \end{cases} \tag{4.5}$$

with constants $\nu > 0$ and $\gamma \geq 0$.

Optimal preconditioners, based on Finite Element Methods have been developed for problem (4.4) [1, Sect. 4.4.2]. Two functions solving problem (4.4) are present in this directory: `ell_1d` and `ellprecofem_1d`. In the former one the linear system is solved by a direct method by using the backslash command of matlab, in the second one, the linear system is solved by preconditioned (with FEM preconditioners) CG/BiCGStab methods.

The following is a list of the functions currently supported with a brief explanation.

`[xy,un,err_inf,err_l2,err_h1,der]=adr_1d(xa,xb,nu,beta,gam, uex,uexx,ff,cb,ne,p,nx,param)`

Numerical solution of the 1D boundary value problem (4.3) by SEM-NI approach.

`call_adr_1d`

Script to call `adr_1d` and to produce data of figure 1.4, pag. 21, [1]

`call_ell_1d`

Script to call `ell_1d` and to produce data of figure 4.17, pag. 207, [1]

`call_ellprecofem_1d`

Script to call `ellprecofem_1d`.

`call_lap_1d`

Script to call `lap_1d`.

`[xy,un]=ell_1d(xa,xb,nu,beta,gam,ff,cb,ub,ne,nx)`

Numerical solution of the 1D elliptic boundary value problem (4.4) by SEM-NI approach.

`[xy,un,A,M,AFE,MFE,MFEd,d,kappa,param]=ellprecofem_1d(xa,xb,nu, beta,gam, ff,cb,ub,ne,nx,param)`

Numerical solution of the 1D elliptic boundary value problem (4.4) by SEM-NI approach. The linear system is solved by FEM-preconditioned CG (or BiCGStab) method (see [1, Table 4.6, pag. 221]).

`[AFE,MFE,MFEd]=femp1_preco_1d(npdx,ne,nov,jacx,xy,nu,beta,gam,param)`

Assembles P1 (stiffness and mass) matrices for 2-nd order 1D b.v.p. (4.4)

`[xy,un,err_inf,err_l2,err_h1]=lap_1d(xa,xb,nu,gam, uex,uexx,ff,cb,ne,nx,param)`

Numerical solution of the 1D Poisson boundary value problem (4.5) by SEM-NI approach.

`[Al,Ml,Mld]=matricesp1_1d(nu,beta,gam,jacx,param)`

Constructs P1 local mass and stiffness matrices in $[-1, 1]$

`[nov_p1,nov_p1_g]=nov_p1_1d(nov)`

Construct P1-FEM (local-P1 mesh $\rightarrow$ global-P1 mesh (=local spectral element)) map

`[u,iter,err]=pbcgstab_vdv(A, b, u, tol, maxit ,P,preco)`

Preconditioned BiCGStab method with FEM preconditioners (see [1, pag. 513], [4]).

`[x,iter,res]=precg(A, b, x0, tol, itmax,P,preco)`

Preconditioned Conjugate Gradient method with FEM preconditioners

`[AFE,MFE,MFEd]=precofem_1d_se(AFE,MFE,MFEd,ne,xy,nov,nu,beta,gam,param)`

Assembles P1 matrices (stiffness, mass, discrete mass) on macro spectral elements

A dependency-graph for this directory is shown in Fig. 4.9

4.10 `Elliptic_2d` directory

The directory `Elliptic_2d` consists of a number of functions to approximate the solution of the 2D linear elliptic second order equation

$$\begin{cases} -\Delta u + \gamma u = f & \text{in } \Omega \\ \text{Dirichlet or Neumann b.c.} & \text{on } \partial\Omega \end{cases} \tag{4.6}$$

with constant $\gamma \geq 0$, by following the SEM-NI approach [1, Ch. 4], [2, Ch. 5]. The interface Schur complement approach has been also considered, but without preconditioner. The implementation of both Neumann-Neumann and balancing Neumann-Neumann preconditioners for the interface Schur complement is realized in the `Elliptic_2d/Schur` directory.

The following is a list of the functions currently supported with a brief explanation.

`call_lap_2d`

Script for pre- and post- processing `lap_2d`

`[xy,un,D,param]=lap_2d(xa,xb,ya,yb,gam,uex,uex_x,uex_y, ff,g,h,cb,`
`nex,nx,ney,ny,gammax,gammay,param)`

Numerical solution of the 2D b.v.p. (4.6)

`[A,f]=patch_se(A,f,ifro,nov,dx,jacx,dy,jacy)`

Calls `patch_sp` to impose strong continuity of normal derivatives across interfaces

`A=patch_sp(dx,jacx,dy,jacy)`

Imposes strong continuity of normal derivatives across interfaces

`plot_mesh`

Script for plotting SEM mesh on a rectangle

A dependency-graph for this directory is shown in Fig. 4.10

4.11 `Elliptic_2d/Schwarz` directory

The directory `Elliptic_2d/Schwarz` consists of a number of functions to approximate the solution of the 2D linear elliptic second order equation (4.6) by SEM-NI approach and by exploiting the additive Schwarz preconditioner with coarse correction [1, Ch. 4], [2, Ch. 5, Ch. 6]. Eigenvalues are also computed to measure efficiency of Schwarz preconditioner.

The following is a list of the functions currently supported with a brief explanation.

`call_eig_schwarz_2d`
: Script file for pre and post processing `eig_schwarz_2d`

`call_schwarz_2d`
: Script for pre and post processing `schwarz_2d`

`[nove,nvle]=cosnovenew(nx,nex,ny,ney,nov,ifro,nlevel)`
: Construction of restriction maps for extended elements

`[param]=eig_schwarz_2d(xa,xb,ya,yb,gam,cb,nex,nx,ney,ny,gammax,gammay,param)`
: Eigenvalues computation for the matrix associated to 2D b.v.p. (4.6), either with Schwarz preconditioner or without preconditioner

`[listaint,listadir]=liste2(ifro)`
: Assembles lists of internal and boundary nodes (similar to liste)

`[r0t]=matr0t(nx,ny,xy,nov, novc,noec,lista_coarse)`
: Constructs matrix R_H^T referred in (6.3.21), pag. 373 [2]

`[lista_coarse,novc,novcg,jacxe,jacye]=meshq1_coarse(npdx,npdy,nov,xy)`
: Construction of structures for the coarse mesh, for Schwarz preconditioner

`[novl,jacx,jacy]=meshq1_ie(nov,nvl,xy,ipar)`
: Construction of extended Q1 mesh, for Schwarz preconditioner

`[p_unity]=partition_e(nove,nvle,noe)`
: Unity partition for the extended mesh, for Schwarz preconditioner

```
[z]=precoasc(r,param,noei,lint,p_unity,xy,ww,nov,x,wx,y,wy,xx,jacx,yy,jacy,
     Aq1,wwq1,linte,nove,nvle,Ac,Acb,wwc,r0t,lista_coarse,
     noec,novc,lintc,ldirc)
```
: Solves the linear system $P_{as}\mathbf{z} = \mathbf{r}$ where P_{as} is the additive Schwarz preconditioner

`[p]=reorder(xyl,nex,ney)`
: Reordering of the array of nodes of extended element

```
[xy,un,param]=schwarz_2d(xa,xb,ya,yb,gam, uex,uex_x,uex_y,ff,g,h,cb,
     nex,nx,ney,ny,gammax,gammay,param)
```
: Numerical solution of the 2D boundary value problem (4.6) by using additive Schwarz preconditioner with coarse mesh

```
[u,iter,res]=schwarz_pbcgstab(u0, f, param,p_unity,xy,ww,A,nov,noei, lint,
     x,wx,y,wy,xx,jacx,yy,jacy,Aq1,wwq1,linte,nove,nvle,Ac,Acb,wwc,r0t,
     lista_coarse,noec,novc,lintc,ldirc)
```
: BiCGStab method with additive Schwarz preconditioner with overlap and coarse mesh

```
[u,iter,res]=schwarz_pcg(u0, f, param,p_unity,xy,ww,A,nov,noei, lint,
     x,wx,y,wy,xx,jacx,yy,jacy,Aq1,wwq1,linte,nove,nvle,Ac,Acb,wwc,r0t,
     lista_coarse,noec,novc,lintc,ldirc)
```
: Conjugate Gradiente method with additive Schwarz preconditioner with overlap and coarse mesh

`[Aq1,Abq1,wwq1,linte,ldire,nove]=stiffq1(ifro,nov,xy,nove,nvle)`
: Constructs local stiffness Q1 matrices on extended elements, for Schwarz preconditioner

`[Ac,Acb,wwc,lista_coarse,noec,novc,lintc,ldirc]=stiffq1H(nx,nex,ny,ney,`
`xy,nov,ifro)`

Construction of stiffness Q1 matrix on the coarse grid, for Schwarz preconditioner

`[A,ww]=stiffq1_se(ipar,ifro,nov,wx,dx,jacx,wy,dy,jacy)`

Assembles Q1 stiffness local matrices on extended elements, for Schwarz preconditioner

A dependency-graph for this directory is shown in Fig. 4.11

4.12 `Elliptic_2d/Schur` directory

The directory `Elliptic_2d/Schur` consists of a number of functions to approximate the solution of the 2D linear elliptic second order equation (4.6) by SEM-NI approach and by exploiting the interface Schur complement matrix [2, Ch. 6]. The interface Schur complement matrix could be preconditioned by either Neumann-Neumann or balancing Neumann-Neumann preconditioner. Eigenvalues are also computed to measure efficiency of this approach.

The following is a list of the functions currently supported with a brief explanation.

`call_eig_schur_2d`

Script for pre and post processing `eig_schur_2d`

`call_eig_schur_2d_file`

Script for pre and post processing `eig_schur_2d`. It produces files for Fig. 6.19 [2]

`call_schur_2d`

Script for pre and post processing `schur_2d`

`[novg]=cosnovg(xyi,noei,ifroi,lgamma,ldnov,novi,nvli)`

Constructs matrix `novg` which implements operators R_{Γ_m} (for $m = 1, ..., M$), the restriction operator from the vector of coefficient unknowns related to the nodes of Γ to only those associated with $\Gamma_m = \Gamma \cap \partial\Omega_m$ (see [2], pag. 394)

`[novi,nvli]=cosnovi(nov,ifro,lint)`

Constructs matrix `novi` which implements operators R_m, the restriction operator from the vector of coefficient unknowns related to the nodes of $\overline{\Omega}$ to the vector of coefficient unknowns related to the nodes of $\overline{\Omega}_m$

`[Rgamma]=cosrgam(novg,LGG,ne,ngamma)`

Computes matrix R_Γ for Schur complement preconditioners. If Γ is the interface (union of interfaces between sub domains) and $\Gamma_m := \partial\Omega_m \cap \Gamma$, then $(R_\Gamma)_{mj} := 1/n_j$ if $x_j \in \Gamma_m$, $(R_\Gamma)_{mj} := 0$ otherwise, where $n_j =$ is the number of sub domains x_j belongs to.

`[param]=eig_schur_2d(xa,xb,ya,yb,gam,cb,nex,nx,ney,ny,gammax,gammay,param)`

Eigenvalues computation for interface (preconditioned) Schur complement matrix

`[un]=local_solver(Amm,AGm,Lmm,LGG,novi,nvli,nov,novg,lint,lgamma,ugamma,f,ub)`

Solution of local problems after knowledge of u on the interface

`[D]=partition(Rgamma)`

Computes the diagonal weighting matrix D relative to interface unknowns

`[PSH]=pinv_sigma(AGG,Amm,AGm,LGG,novg,Rgamma)`

Computes the pseudo inverse of Σ_H

`[xy,un,param]=schur_2d(xa,xb,ya,yb,gam,uex,uex_x,uex_y,ff,g,h,cb,`
`nex,nx,ney,ny,gammax,gammay,param)`

Numerical solution of the 2D b.v.p. (4.6) by the interface Schur complement approach

`[Sigma]=schur_assemb(AGG,Amm,AGm,LGG,novg,lint,lgamma,param)`

Assembles global Schur complement matrix

```
[AGG,Amm,AGm,Lmm,LGG,Am,f]=schur_loc(ifro,nov,wx,dx,jacx,wy,dy,jacy,
     nvli,gam,f,ub,param)
```

Computes local matrices and lists for implementing the Schur method

```
[Sigma,PNN]=schur_matrix(ifro,nov,wx,dx,jacx,wy,dy,jacy,nvli,gam,novg,lint,
     lgamma,D,Rgamma,param)
```

Computes Schur complement matrix Sigma and its preconditioner

`[v]=schur_mxv(x,AGG,Amm,AGm,LGG,novg,ne)`

Computes matrix vector product, where the matrix is the interface Schur complement

```
[x,iter,res]=schur_pcg(x0, b, tol, maxit,param,AGG,Amm,AGm,LGG,Am,
     nvli,novg,D,Rgamma,PSH)
```

Preconditioned conjugate gradient to solve the Schur complement system

`[z]=schur_precobnn(r,ne,nvli,novg,D,LGG,Am,Rgamma,PSH,AGG,Amm,AGm)`

Solves the system $(P_b^{NN})^{-1}\mathbf{z} = \mathbf{r}$ where P_b^{NN} is the Balancing Neumann-Neumann preconditioner for Schur complement matrix

`[z]=preconnl(r,ne,nvli,novg,D,LGG,Am)`

Solves the system $(P^{NN})^{-1}\mathbf{z} = \mathbf{r}$ where P^{NN} is the Neumann-Neumann preconditioner for Schur complement matrix

A dependency-graph for this directory is shown in Fig. 4.12

4.13 `Elliptic_3d` directory

The directory `Elliptic_3d` consists of a number of functions to approximate the solution of the 3D linear elliptic second order equation (4.6) by SEM-NI approach.

The following is a list of the functions currently supported with a brief explanation.

`call_lap_3d`

Script for pre- and post-processing `lap_3d`

```
[xyz,un,D,param]=lap_3d(xa,xb,ya,yb,za,zb,gam,uex,uex_x,uex_y,uex_z,ff,
     nex,nx,ney,ny,nez,nz,gammax,gammay,gammaz,param)
```

Numerical solution of the 3D b.v.p. (4.6) where Ω is a parallelepiped.

A dependency-graph for this directory is shown in Fig. 4.13

4.14 Dependency graphs

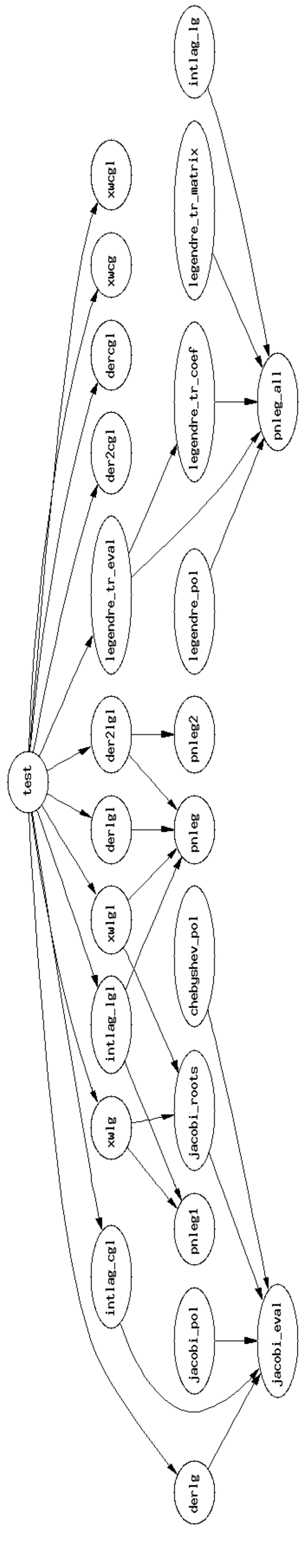


Figure 4.1: Dependency-graph for directory Level_0

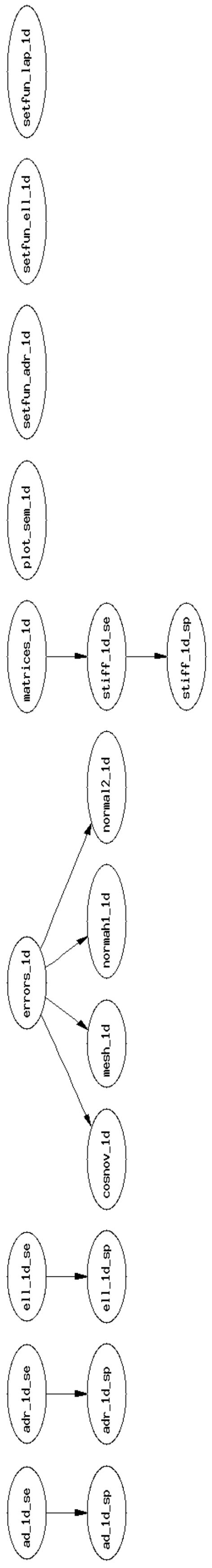


Figure 4.2: Dependency-graph for directory Level_1

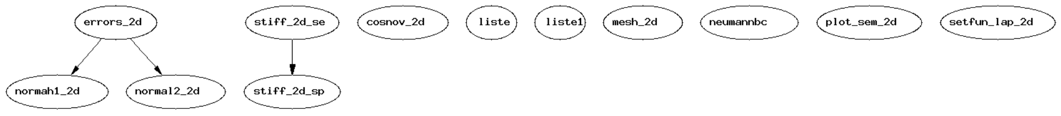


Figure 4.3: Dependency-graph for directory Level_2

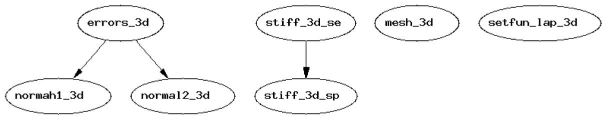


Figure 4.4: Dependency-graph for directory Level_3

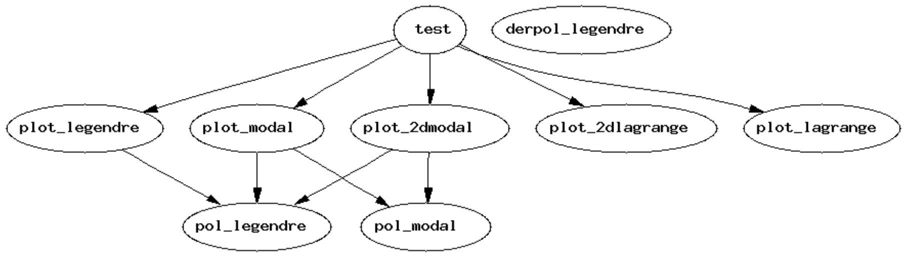


Figure 4.5: Dependency-graph for directory `Basis_functions`

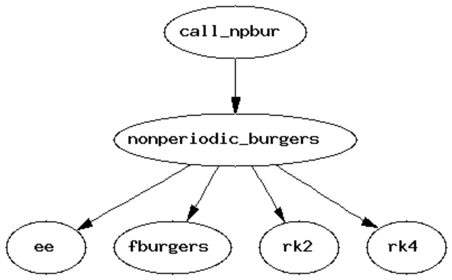


Figure 4.6: Dependency-graph for directory `Burgers`

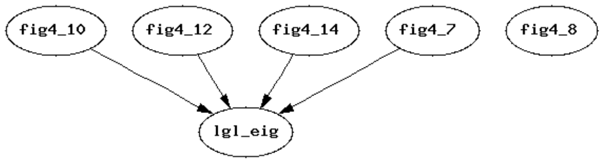


Figure 4.7: Dependency-graph for directory `Eigenvalues_1d`

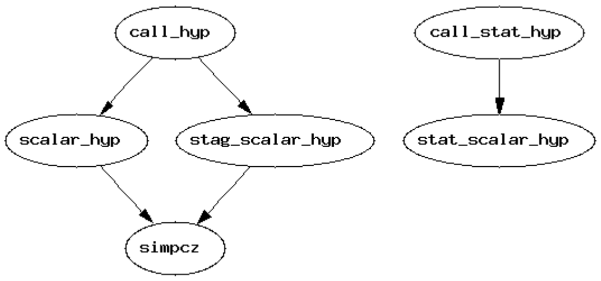


Figure 4.8: Dependency-graph for directory `Hyperbolic_1d`

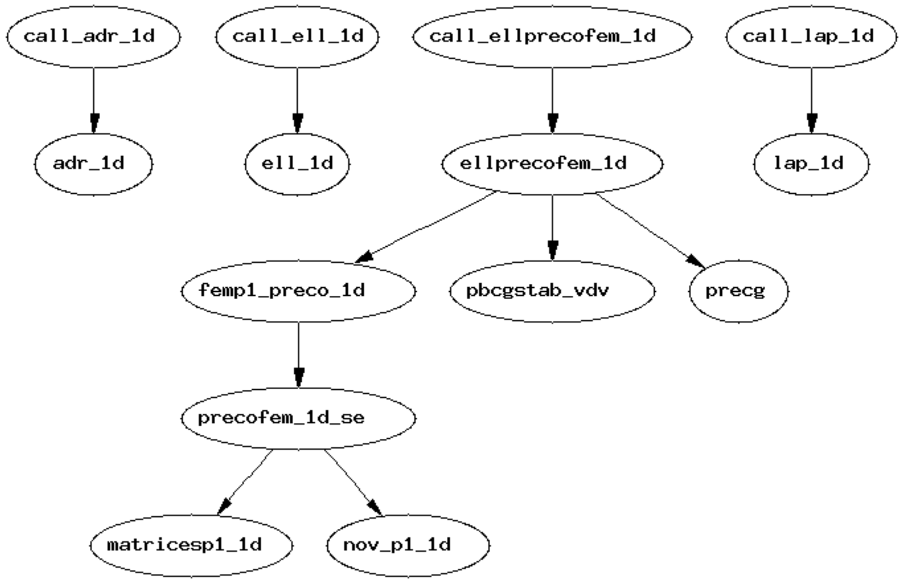


Figure 4.9: Dependency-graph for directory `Elliptic_1d`

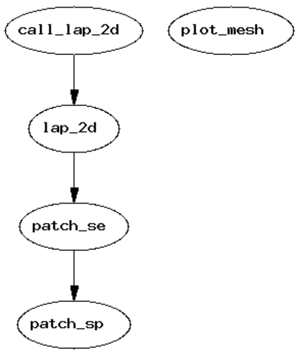


Figure 4.10: Dependency-graph for directory `Elliptic_2d`

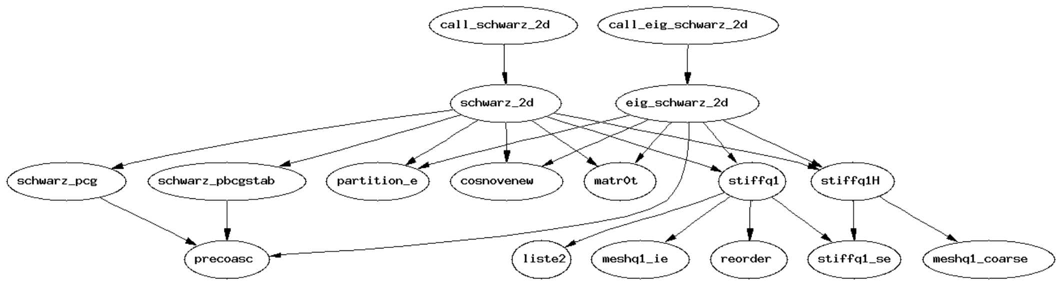


Figure 4.11: Dependency-graph for directory `Elliptic_2d/Schwarz`

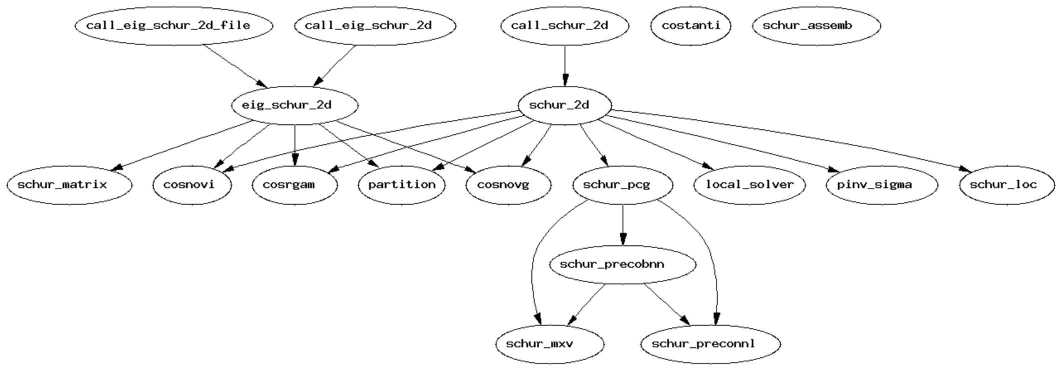


Figure 4.12: Dependency-graph for directory `Elliptic_2d/Schur`

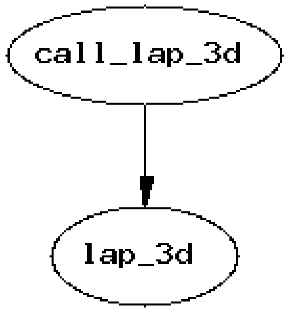


Figure 4.13: Dependency-graph for directory `Elliptic_3d`

Bibliography

[1] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods. Fundamentals in Single Domains* . Springer, Heidelberg, 2006.

[2] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods. Evolution to Complex Geometries and Application s to Fluid Dynamics* . Springer, Heidelberg, 2007.

[3] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations.* Springer Verlag, Heidelberg, 1994.

[4] Henk A. van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics.* Cambridge University Press, Cambridge, 2003.