8 A pseudo-spectral solution to the Stokes Problem

8.1 The Method

8.1.1 Generalities

We are interested in setting up a pseudo-spectral method for the following Stokes Problem

\[
\begin{cases}
\Delta \mathbf{u} - \sigma \mathbf{u} - \nabla p = f \quad \text{in } \Omega \\
\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \\
\mathbf{u} \mid_{\Gamma} = \mathbf{u}_\Gamma \quad \text{on } \Gamma := \partial \Omega
\end{cases}
\]  

where \( \sigma \geq 0 \), \( \Omega \subset \mathbb{R}^2 \) is a bounded domain, \( \mathbf{u} : \bar{\Omega} \to \mathbb{R}^2 \) is the velocity field and \( p : \tilde{\Omega} \to \mathbb{R} \) is the pressure.

It is worth to stress that this particular instance of the Stokes Problem may arise when one deals with a Stokes evolution equation and uses numerical integration: a problem like \( (S-I) \) must be solved at every time step.

First we observe that \( \nabla \cdot \Delta \mathbf{u} = \Delta \nabla \cdot \mathbf{u} = \Delta(0) = 0 \) thus the first two equations in \( (S-I) \) imply \( \Delta p = \nabla \cdot f \). Also we introduce the Helmholtz operator \( H_\sigma := \Delta - \sigma \text{Id} \) of parameter \( \sigma \) so we can re-write \( (S-I) \) in the form

\[
\begin{cases}
H_\sigma \mathbf{u} - \nabla p = f \quad \text{in } \Omega \\
\Delta p = \nabla \cdot f \quad \text{in } \Omega \\
\mathbf{u} \mid_{\Gamma} = \mathbf{u}_\Gamma \quad \text{on } \Gamma := \partial \Omega \\
\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Gamma
\end{cases}
\]  

\( (S-II) \)

It is rather important to note that (at least at this stage, i.e., no discretization procedure performed so far) the latter formulation implies the first one. For (let us reason in term of classical solutions for simplicity), set \( Q := \nabla \cdot \hat{\mathbf{u}} \), where \( \hat{\mathbf{u}} \) solves \( (S-II) \). It follows that \( Q \) solves

\[
\begin{cases}
\Delta Q = \sigma Q \quad \text{in } \Omega \\
Q = 0 \quad \text{on } \Gamma
\end{cases}
\]

thus \( \nabla \cdot \hat{\mathbf{u}} = Q \equiv 0 \) in \( \Omega \), that is \( \hat{\mathbf{u}} \) is divergence-free and thus solves \( (S-I) \) as well.

8.1.2 Rough Chebyshev-Chebyshev discretization

We focus on the very simple case of \( \Omega = ]-1, 1[^2 \). This leads to use Chebyshev polynomials as basis functions. Let us denote by \( \mathbb{P}_N^2 \) the space of tensor product polynomials of degree at most \( N \) in each of the two variables separately.
We will use the Chebyshev-Lobatto quadrature nodes

\[ x_i = \cos \left( \frac{\pi i}{N} \right), \quad i = 0, 1, \ldots, N \]

\[ y_j = \cos \left( \frac{\pi j}{N} \right), \quad j = 0, 1, \ldots, N \]

and define the following computational domains

\[ \Omega_N := \{(x_i, y_j) : (i, j) \in \{1, 2, \ldots, N - 1\} \times \{1, 2, \ldots, N - 1\}\}, \]

\[ \overline{\Omega}_N := \{(x_i, y_j) : (i, j) \in \{0, 1, \ldots, N\} \times \{0, 1, \ldots, N\}\}, \]

\[ \overline{\Omega}_N^{\text{int}} := \{(x_i, y_j) \in \overline{\Omega}_N : (i, j) \notin \{0, N\} \times \{0, N\}\}, \]

\[ \Gamma_N := \{(x_i, y_j) : (i, j) \in \{0, N\} \times \{0, 1, \ldots, N\} \cup \{0, 1, \ldots, N\} \times \{0, N\}\}, \]

\[ \Gamma_N^{\text{int}} := \{(x_i, y_j) \in \Gamma_N : (i, j) \notin \{0, N\} \times \{0, N\}\}, \]

corresponding respectively to

- interior nodes
- all nodes
- all nodes but the corners
- boundary nodes
- all boundary nodes but the corners.

The resulting discretized problem becomes: find \( u^N \in \mathbb{P}_2^N \times \mathbb{P}_2^N \) and \( p^N \in \mathbb{P}_2^N \) such that, denoting by \( f^N \) the interpolating polynomial of \( f \) on \( \overline{\Omega}_N \), we have

\[
\begin{cases}
H_\sigma u^N - \nabla p^N = f^N & \text{in } \Omega_N \\
\Delta p^N = \nabla \cdot f^N & \text{in } \Omega_N \\
\left. u^N \right|_\Gamma = u_\Gamma & \text{on } \Gamma_N^{\text{int}} \\
\nabla \cdot u^N = 0 & \text{on } \Gamma_N^{\text{int}}
\end{cases}
\]

(S-CC)

We stress that, as usual in PS methods, the derivatives are taken in the interpolation sense, that is one first interpolates than takes derivatives.
8.1.3 A variational crime

In this discretization procedure a remarkable phenomena comes into play: even if any classical solution \( u \) of (S-II) is a divergence free vector field, its approximation \( u^N \) does not need to fulfil the same property. This is a consequence of imposing the differential equations in the strong sense point-wise. Remember that, when showing that a solution to (S-II) needs to be divergence free, we used that the laplacian of an identically vanishing function is zero. Here equations are satisfied only on a finite sets of points so we can’t perform the same reasoning: in general \( \nabla \cdot u_N \neq 0 \) on \( \Omega \).

Recall that the divergence free condition represent the conservation of the mass, hence trying to preserve such a condition seems very reasonable. More concretely, as often occurs, the lack of physical meaning of the discretized equation (S-CC) boils down to numerical instability in solving the steady state problem (S-I) and even worse behaviour of the final numerical solution if (S-I) arises as a intermediate problem in the time integration of an evolution equation. We need to overcome this problem.

We introduce a numerical flux \( B^N \in \mathbb{P}_2^N \times \mathbb{P}_2^N \) such that its normal component at the boundary is precisely the normal component of the residual, more precisely we replace the problem (S-CC) with

\[
\begin{align*}
H_\sigma u^N - \nabla p^N &= f^N & \text{in } \Omega_N \\
\Delta p^N &= \nabla \cdot f^N - \nabla \cdot B^N & \text{in } \Omega_N \\
B^N &= 0 & \text{in } \Omega_N \\
u|_\Gamma &= u_\Gamma & \text{on } \Gamma_N^{\text{int}} \\
B^N \cdot \nu &= (H_\sigma u^N - \nabla p^N - f^N) \cdot \nu & \text{on } \Gamma_N^{\text{int}}.
\end{align*}
\]

(S-CC’)

Here \( \nu \) is the outer unit normal.

At this stage it is not clear why this variational crime should enforce the divergence free condition on \( u_N \), indeed this will surface out in a while when presenting the Influence Matrix Method that we use to solve (S-CC’).

8.1.4 Influence Matrix

We split (S-CC’) in two sub problems. More precisely, we assume that

\[ u_N = \tilde{u}^N + \bar{u}^N, \quad p_N = \tilde{p}^N + \bar{p}^N, \quad B^N = \tilde{B}^N + \bar{B}^N, \]

where each \( \tilde{\cdot} \) and \( \bar{\cdot} \) function solves respectively the \( \tilde{\cdot} \) or the \( \bar{\cdot} \) problem below.
\[
\begin{aligned}
\Delta \tilde{p}^N &= -\nabla f^N \quad \text{in } \Omega_N, \\
\tilde{p}^N &= 0 \quad \text{on } \Gamma_{\text{int}}^N, \\
H_\sigma \tilde{u}^N &= f^N + \nabla \tilde{p}^N \quad \text{in } \Omega_N, \\
\tilde{u}^N &= u_\Gamma \quad \text{on } \Gamma_{\text{int}}^N, \\
\tilde{B}^N &= 0 \quad \text{in } \Omega_N, \\
\tilde{B}^N \cdot \nu &= g^N := (H_\sigma \tilde{u}^N - f^N - \nabla \tilde{p}^N) \cdot \nu \quad \text{on } \Gamma_{\text{int}}^N, \\
\tilde{B}^N \cdot \tau &= 0 \quad \text{on } \Gamma_{\text{int}}^N,
\end{aligned}
\]

(\text{\textasciitilde problem I})

(\text{\textasciitilde problem II})

(\text{\textasciitilde problem III})

Here \(\tau\) is the unit tangent.

\[
\begin{aligned}
\Delta \bar{p}^N &= -\nabla (\tilde{B}^N + \bar{B}^N) \quad \text{in } \Omega_N, \\
H_\sigma \bar{u}^N &= \nabla \bar{p}^N \quad \text{in } \Omega_N, \\
\bar{u}^N &= 0 \quad \text{on } \Gamma_{\text{int}}^N, \\
\nabla \cdot \bar{u}^N &= -\nabla \cdot \tilde{u}^N \quad \text{on } \Gamma_{\text{int}}^N, \\
\bar{B}^N \cdot \nu &= (H_\sigma \bar{u}^N - \nabla \bar{p}^N) \nu \quad \text{on } \Gamma_{\text{int}}^N.
\end{aligned}
\]

(\text{\textbar problem})

First, notice that the \(\text{\textasciitilde}\) problems I, II and III can be considered separately in this order. We compute \(\tilde{p}^N\) solving (\text{\textasciitilde problem I}), then we compute the gradient of \(\tilde{p}^N\), we plug it in (\text{\textasciitilde problem II}) and we compute \(\tilde{u}^N\) by solving it; finally \(\nabla \cdot \tilde{B}^N\) is computed and stored.

Recall that the solution to our final problem consists in finding \(u^N\) and \(p^N\), while the numerical flux is an additional variable whose tangential or normal values we are going to compute only when these are required in order to compute \(u^N\) or \(p^N\).

To understand that (\text{\textbar problem III}) leads to compute \(\nabla \cdot \bar{B}^N\) it is sufficient to re-write the problem in a more explicit form by Lagrange interpolation. Since \(\bar{B}^N = (\bar{B}_1^N, \bar{B}_2^N) \in \mathbb{P}^N \times \mathbb{P}^N\) we have

\[
\bar{B}_1^N(x, y_j) = \sum_{h=0}^N \ell_h(x) \bar{B}_1^N(x_h, y_j) = \ell_0(x) \bar{B}_1^N(x_0, y_j) + \ell_N(x) \bar{B}_1^N(x_N, y_j)
\]

\[
\ell_N(-x) \bar{B}_1^N(x_0, y_j) + \ell_N(x) \bar{B}_1^N(x_N, y_j) - \ell_N(-x) g^N(x_0, y_j) + \ell_N(x) g^N(x_N, y_j)
\]

(8.1)

\[
\bar{B}_2^N(x_i, y) = \sum_{k=0}^N \ell_k(y) \bar{B}_2^N(x_i, y_k) = \ell_0(y) \bar{B}_2^N(x_i, y_0) + \ell_N(y) \bar{B}_2^N(x_i, y_N)
\]

\[
\ell_N(-y) \bar{B}_2^N(x_i, y_0) + \ell_N(y) \bar{B}_2^N(x_i, y_N) - \ell_N(-y) g^N(x_i, y_0) + \ell_N(y) g^N(x_i, y_N).
\]

(8.2)
Here we used that at any point \((x_h, y_k) \in \Gamma^\text{int}_{N}\) we have

\[
\mathbf{B}^N \cdot \nu = \begin{cases} 
-B^N_1 & h = 0 \\
B^N_1 & h = N \\
-B^N_2 & k = 0 \\
B^N_1 & k = N
\end{cases}
\]

and the fact that \(\mathbf{B}^N\) vanishes at any interior point. Using (8.1), (8.2) and the symmetries of \(\Omega_N\) we get

\[
\nabla \cdot \mathbf{B}^N(x_i, y_j) = \ell'_{N}(x_{N-i})g^N(x_0, y_j) + \ell'_{N}(x_i)g^N(x_N, y_j) + \ell'_{N}(y_{N-j})g^N(x_i, y_0) + \ell'_{N}(y_j)g^N(x_i, y_N). \tag{8.3}
\]

Finally, let us notice that, for any \(l \in \{1, 2, \ldots, N-1\}\), we have

\[
\ell'_N(x_l) = \frac{d}{dx} \left( \prod_{j=0}^{N-1} \frac{x - x_j}{x_N - x_j} \right) \bigg|_{x=x_l} = \sum_{h=0}^{N-1} \prod_{j \neq h,N} \left( x_l - x_j \right) \prod_{j \neq N} \left(x_N - x_j\right),
\]

where we denoted by \(\ell'^{(l)}_N\) the fundamental Lagrange interpolating polynomial relative to the \(N\)-th node \(x_N\) built on the nodes \(X^{(l)}_N := \{x_0, x_1, \ldots, x_{l-1}, x_{l+1}, \ldots, x_N\}\). We warn the reader that this new Lagrange basis is no more preserving the symmetries of the starting one, since \(X^{(l)}_N\) is no more symmetric. Indeed one can prove that \(\ell'^{(N-l)}_N(x_{N-l}) = \ell'^{(l)}_N(x_l)\).

Using the above formula for the derivatives of the Lagrange basis equation (8.3) becomes

\[
\nabla \cdot \mathbf{B}^N(x_i, y_j) = \frac{\ell'^{(N-l)}_N(x_{N-i})}{x_N - x_{N-i}} g^N(x_0, y_j) + \frac{\ell'^{(l)}_N(x_l)}{x_N - x_i} g^N(x_N, y_j) + \frac{\ell'^{(N-j)}_N(y_{N-j})}{y_N - y_{N-j}} g^N(x_i, y_0) + \frac{\ell'^{(j)}_N(y_j)}{y_N - y_j} g^N(x_i, y_N) \tag{8.4}
\]

\[
= c_{N-i}g^N(x_0, y_j) + c_{i}g^N(x_N, y_j) + c_{N-j}g^N(x_i, y_0) + c_{j}g^N(x_i, y_N).
\]

Now we consider \((-\text{problem})\). We use a quite unusual method, termed superposition method. In this technique each function \(\phi\) of the set \(\{\mathbf{p}^N, \mathbf{u}^N, R^N := \mathbf{B}^N \cdot \nu, \mathbf{B}\}\) is expanded in the form

\[
\phi = \sum_{l=1}^{2L} \xi_l \phi_l, \quad L := \text{Card} \Gamma^\text{int}_{N} = 2(2N - 2), \tag{8.5}
\]

159
but the coefficients are fixed: what varies are the $\phi_l$ themselves.

More precisely, we consider two families of problems, each of them of $L$ problems. For $l = 1, 2, \ldots, L$ solve

$$
\begin{align*}
\Delta \bar{p}^N_l &= 0 \quad \text{in } \Omega_N \\
\bar{p}^N_l(\eta_m) &= \delta_{l,m} \eta_m \in \Gamma^\text{int}_N
\end{align*}
$$

(8.6)

$$
\begin{align*}
\Delta \bar{u}^N_l &= \nabla \bar{p}^N_l \quad \text{in } \Omega_N \\
\bar{u}^N_l &= 0 \quad \text{on } \Gamma^\text{int}_N
\end{align*}
$$

(8.7)

$$
B^N_l = 0 \quad \text{in } \Omega
$$

(8.8)

For $l = L + 1, L + 2, \ldots, 2L$ solve

$$
\begin{align*}
\Delta \bar{p}^N_l &= 0 \quad \text{in } \Omega_N \\
B^N_l \cdot \nu(\eta_m) &= \delta_{L+m,l} \eta_m \in \Gamma^\text{int}_N
\end{align*}
$$

(8.9)

$$
\begin{align*}
\Delta \bar{p}^N_l &= -\nabla B^N_l \cdot \nu \quad \text{in } \Omega_N \\
\bar{p}^N_l &= 0 \quad \text{in } \Omega_N \\
H^\sigma \bar{u}^N &= \nabla \bar{p}^N \quad \text{in } \Omega_N \\
\bar{u}^N &= 0 \quad \text{on } \Gamma^\text{int}_N
\end{align*}
$$

(8.10)

(8.11)

Again, one first solves (8.6) to determine $\bar{p}^N_l$, $l \in \{1, \ldots, L\}$, computes the gradient of the pressure and solves (8.7) by plugging it into the equation. This leads to compute $\bar{u}^N_l$, $l \in \{1, \ldots, L\}$, then we compute its laplacian and finally the values $R^N_l = (\Delta \bar{u}^N_l - \nabla \bar{p}^N_l) \cdot \nu$ at the nodes in $\Gamma^\text{int}_N$.

Now consider the case $l > L$. It is worth to stress that (8.9) fully characterize the values of $\nabla \cdot B^N_l$ at points of $\Omega_N$, this is a consequence of the tensor product construction of both the domain and the functions space we took into account: notice that the normal derivatives are polynomial of only one variable. In order to get convinced is convenient to draw a picture of $\Omega_3$ (the easiest example), pick any $l$, $m$ and write the values of $B^N_l$ given by (8.9) and try to compute $\nabla \cdot B^N_l(x_2, y_2)$ (the only node in $\Omega_3$).

Indeed in order to compute $\nabla \cdot B^N_l$ starting by (8.9) it is sufficient to apply equation (8.4).

$$\nabla \cdot B^N_l(x_i, y_j) = c_{N-i}\delta\{(x_0, y_j) = \eta_{L-L}\} + c_i\delta\{(x_N, y_j) = \eta_{L-L}\} + c_{N-j}\delta\{(x_i, y_0) = \eta_{L-L}\} + c_j\delta\{(x_i, y_N) = \eta_{L-L}\}$$

(8.12)

After determining the value of $\nabla \cdot B^N_l$ at $\Omega_N$ by solving (8.9), we plug it into (8.10), we solve it and we compute $\bar{p}^N_l$ and $R^N_l(\eta_m)$ for $l = L + 1, L + 2, \ldots, 2L$, $m = 1, \ldots, L$.

Finally we compute the gradient of $\bar{p}^N_l$, we plug it in (8.11) and we solve it to determine $\bar{u}^N_l$ and $R^N_l$ for $l > L$. 

160
Now note that any set of functions \( \{ \tilde{p}^N, \tilde{u}^N, R^N, B \} \) defined by a superposition of the form (8.5), that is for any \( \xi \in \mathbb{R}^{2L} \), satisfy the first three equations in (8.1) problem; we want to determine \( \xi \) by imposing the remaining two equations.

\[
\begin{align*}
\sum_{l=1}^{2L} \xi_l \nabla \cdot \tilde{u}^N_l(\eta_m) &= -\nabla \cdot \tilde{u}^N(\eta_m) & \eta_m \in \Gamma^N_{\text{int}} \\
\sum_{l=1}^{2L} \xi_l (B^N_l \cdot \nu - R^N_l)(\eta_m) &= (\tilde{B}^N \cdot \nu)(\eta_m) & \eta_m \in \Gamma^N_{\text{int}}
\end{align*}
\]  \hspace{1cm} (8.13)

This system may be written in the form \( I^N \xi = c \), where the matrix

\[
I^N := \begin{bmatrix}
\nabla \cdot \tilde{u}^N_1(\eta_1) & \nabla \cdot \tilde{u}^N_2(\eta_1) & \ldots & \ldots & \nabla \cdot \tilde{u}^N_2L(\eta_1) \\
\nabla \cdot \tilde{u}^N_1(\eta_2) & \nabla \cdot \tilde{u}^N_2(\eta_2) & \ldots & \ldots & \nabla \cdot \tilde{u}^N_2L(\eta_2) \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
(B^N_1 \cdot \nu - R^N_1)(\eta_1) & (B^N_2 \cdot \nu - R^N_2)(\eta_1) & \ldots & \ldots & (B^N_2L \cdot \nu - R^N_2L)(\eta_1) \\
(B^N_1 \cdot \nu - R^N_1)(\eta_2) & (B^N_2 \cdot \nu - R^N_2)(\eta_2) & \ldots & \ldots & (B^N_2L \cdot \nu - R^N_2L)(\eta_2) \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
(B^N_1 \cdot \nu - R^N_1)(\eta_L) & (B^N_2 \cdot \nu - R^N_2)(\eta_L) & \ldots & \ldots & (B^N_2L \cdot \nu - R^N_2L)(\eta_L)
\end{bmatrix}
\]

is the Influence Matrix and

\[
c = \begin{bmatrix}
-\nabla \cdot \tilde{u}^N(\eta_1) \\
-\nabla \cdot \tilde{u}^N(\eta_2) \\
\vdots \\
-\nabla \cdot \tilde{u}^N(\eta_L) \\
(\tilde{B}^N \cdot \nu)(\eta_1) \\
(\tilde{B}^N \cdot \nu)(\eta_2) \\
\vdots \\
(\tilde{B}^N \cdot \nu)(\eta_L)
\end{bmatrix}.
\]

Unfortunately a problem may arise when solving the influence matrix system: in general the matrix \( I^N \) may be not invertible; experimentally four zero singular values appear. Two situations may occur: if \( c \) lies in the image of \( I^N \) the system has solution(s), otherwise there exists no solution.

### 8.2 Implementation

The solution of the Stokes problem as proposed in the last subsection is quite complicated. In order to implement it as a software, it is convenient to split it in four steps

1. Computation of nodes, Lagrange basis and differentiation matrices,
S2 Solution of the problems,
S3 Solution of 2L problems arising from the problems,
S4 Assembling the influence matrix and solving the final linear system.

We briefly describe each step below.

8.2.1 Step One

We aim to compute the vector \( C := (c_h)_{h=1,...,N-1} \) defined in (8.4); note that this is done working in only one variable due to the structure of equation (8.4). To ensure a stable computation we use a Chebyshev-Vandermonde approach. Let

\[
V^N := [T_j(x_i)]_{0 \leq j,i \leq N}, \quad \tilde{V}_i^N := [V_{i,j}]_{0 \leq j \leq N-1, i \neq l}, \quad v_l := (V_l,j)_{0 \leq j \leq N-1}.
\]

We compute Lagrange polynomials coefficients by the interpolation conditions and then evaluate it on the desired point, that is

\[
\ell^{(l)}_N(x_l) = v_l(\tilde{V}_l^N)^{-1}e_{N-1}, \quad e_{N-1} = (0, \ldots, 0, 1)^T \in \mathbb{R}^N.
\]

Obviously the solution \( a_l = (\tilde{V}_l^N)^{-1}e_{N-1} \) of \( \tilde{V}_l^N a = e_{N-1} \) should not be computed by the inversion of the matrix, a suitable linear solver must be chosen. Note that

\[
v_l(\tilde{V}_l^N)^{-1}e_{N-1} = \langle (\tilde{V}_l^N)^{-T}v_l^T, e_{N-1} \rangle
\]

hence we need to compute only the last component of the solution \( b_l \) of the linear system \( (V_l^N)^T b_l = v_l^T \); note the transpose sign, due to the fact that \( v_l \) by definition, is a row. We can use the QR algorithm, if we compute the factorization \( (V_l^N)^T = QR \), then the last component \( b_l(N) \) of \( b_l \) is given by

\[
b_l(N) = \langle ((Q_{:,N})^T, v_l^T) = \frac{Q_{:,N}, v_l^T}{R_{N,N}} \rangle.
\]

Here \( R_{N,N} \) is the element of \( R \) of indexes \( N, N \) and \( Q_{:,N} \) is the last column of \( Q \).

Finally

\[
c_h = \frac{\ell^{(h)}_N(x_h)}{1 - \cos \left( \frac{h\pi}{2N} \right)} = \frac{\ell^{(h)}_N(x_h)}{2 \sin^2 \left( \frac{h\pi}{2N} \right)} = \frac{Q_{:,N}, v_l^T}{2R_{N,N} \sin^2 \left( \frac{h\pi}{2N} \right)}.
\]

All these coefficients as well as the matrix \( V^N \) need to be pre-computed and stored for future use. For instance

\[
D_5 = \begin{bmatrix}
0 & 1 & 0 & 3/2 & 0 & 5/2 \\
0 & 0 & 4 & 8 & 0 \\
0 & 0 & 0 & 6 & 10 \\
0 & 0 & 0 & 0 & 8 \\
0 & 0 & 0 & 0 & 10 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

162
The partial differentiation operator can be represented in the space of frequencies by Chebyshev differentiation matrices. These matrices can be derived by the following equation

\[
\frac{d^p T_k(x)}{dx^p} = 2^p k \left( \sum_{m \geq 0, k - p - m \text{ even}}^{k-p} \right)' \left( \frac{(k + p - m - 2)/2}{(k + p - m)/2} \right) \frac{[(k + p - m - 2)/2]!}{[(k + p - m)/2]!} T_m(x).
\]

Here the prime symbol on the sum means that the 0 term in the sum needs to be halved, when appearing. The formula simplifies considerably if we take into account the case \( p = 1 \)

\[
\frac{dT_k(x)}{dx} = 2k \left( \sum_{m \geq 0, k - p - m \text{ even}}^{k-p} \right)' T_m(x).
\] (8.14)

It follows by (8.14) that for any function \( v \in \mathbb{P}^N \times \mathbb{P}^N \) that can be written as

\[
v(x, y) = \sum_{0 \leq i, j \leq N} c_{i,j} [v] T_i(x) T_j(y)
\]

we have

\[
\partial_x v(x, y) = \sum_{0 \leq i, j \leq N} c_{i,j} [v] \partial_x T_i(x) T_j(y) = \sum_{j=0}^{N} \left( \sum_{m=0}^{N-1} \right)' \sum_{i=m+1,i-1-m \text{ even}}^{N} 2i c_{i,j} [v] T_m(x) T_j(y)
\]

\[
= \sum_{0 \leq m, j \leq N} c_{m,j} [\partial_x v] T_m(x) T_j(y),
\]

\[
\partial_y v(x, y) = \sum_{0 \leq i, j \leq N} c_{i,j} [v] T_i(x) \partial_y T_j(y) = \sum_{j=0}^{N} \left( \sum_{m=0}^{N-1} \right)' \sum_{i=m+1,i-1-m \text{ even}}^{N} 2j c_{i,j} [v] T_i(x) T_m(j)
\]

\[
= \sum_{0 \leq i, m \leq N} c_{i,m} [\partial_y v] T_i(x) T_m(y).
\]

We rewrite the above equations in a matrix form, where the conditions on the indexes are encoded in the coefficients

\[
c_{i,j} [\partial_x v] = \sum_{0 \leq h \leq N} d_{i,h}^N c_{h,j} [v], \quad c_{i,j} [\partial_x v] = D^N c_{i,j} [v],
\] (8.15)

\[
c_{i,j} [\partial_y v] = \sum_{0 \leq k \leq N} d_{k,j}^N c_{k,i} [v], \quad c_{i,j} [\partial_y v] = (D^N c_{i,j} [v]^T)^T.
\] (8.16)

Finally, let us fix the standard lexicographical ordering for the basis, that is

\[
\phi_1(x, y) := T_0(x) T_0(y) < \phi_2(x, y) := T_0(x) T_1(y) < \cdots < T_0(x) T_N(y) < T_1(x) T_0(y) < \cdots < T_N(x) T_N(y) =: \phi_{(N+1)^2}(x, y).
\]
We denote by $\mathcal{F}^N[v] : C([-1,1]^2) \rightarrow \mathbb{P}^N \times \mathbb{P}^N$, $\mathcal{F}^N[v] = [c_{ij}[v]]_{0 \leq i,j \leq N}$ the analysis operator and by $\mathcal{E}$ its inverse, the synthesis. We also need the operators $I, J : \mathbb{R}^{(N+1)^2} \rightarrow M_{N+1 \times N+1}(\mathbb{R})$

where $I(v)$ is the $N + 1 \times N + 1$ matrix whose $(i + 1, j + 1)$-element is $v_{(N+1)i+j+1}$, while $J(v) = I(v)^T$; note that $I^{-1}$ corresponds to the matlab (:) operator.

we get $\forall v \in \mathbb{P}^N \times \mathbb{P}^N$

$$\partial_x v = \mathcal{E}^N (D^N \mathcal{F}^N[v]),$$

$$\partial_y v = \mathcal{E}^N ((D^N \mathcal{F}^N[v]^T)^T).$$

(8.17) (8.18)

Analogous formulas for the higher order derivatives can be derived simply iterating these last equations. All these matrices are termed differentiation matrices: they need to be computed once for all and stored to be used many times in the next steps.

Since we are imposing both the differential and the boundary equations in a strong sense we need to consider the grid values of functions in place of the formal sum computed by $\mathcal{E}^N$. Hence we fix a numbering of the interpolation nodes where the first $L$ nodes are in $\Gamma^N$, we identify any function $v \in \mathbb{P}^N \times \mathbb{P}^N$ with $v^N := [v(\eta_m)]_{0 \leq m \leq (N+1)^2}$ and we replace $\mathcal{E}^N$ in equation (8.17) with the evaluation operator $V_N \in GL_{(N+1)^2}(\mathbb{R})$ that simply is the Vandermonde matrix of order $(N + 1)^2$ computed at the points of $\Omega_N$ with the above mentioned ordering for the nodes and We have

$$V_N := [\phi_j(\eta_i^1, \eta_i^2)]_{1 \leq i,j \leq (N+1)^2}.$$

Moreover the analysis can be represented as

$$\mathcal{F}^N = I \circ V_N^{-1}.$$

While the evaluation operator acting on coefficients arranged in a matrix is $V_N I^{-1}$. Thus equations (8.17) and (8.18) leads to the equations for the differentiation in the physical space.

$$(\partial_x v^N(\eta_m)) = V_N I^{-1} D^N I (V_N^{-1} v^N),$$

$$(\partial_y v^N(\eta_m)) = V_N J^{-1} D^N J (V_N^{-1} v^N).$$

(8.19) (8.20)

We can find $\tilde{D}_N^I, \tilde{D}_N^J \in M_{(N+1) \times (N+1)}(\mathbb{R})$ representing the linear operations $I^{-1} D^N I(\cdot)$ and $J^{-1} D^N J(\cdot)$ respectively, so we get

$$(\partial_x v^N(\eta_m)) = V_N \tilde{D}_N^I V_N^{-1} v^N,$$

$$(\partial_y v^N(\eta_m)) = V_N \tilde{D}_N^J V_N^{-1} v^N.$$
Similar relations for higher order partial derivatives are easily obtained.

\[
\begin{align*}
(\partial^2_{xx} v^N(\eta_m)) &= V_N(\tilde{D}_N^I)^2 V_N^{-1} v^N, \\
(\partial^2_{yy} v^N(\eta_m)) &= V_N(\tilde{D}_N^J)^2 V_N^{-1} v^N.
\end{align*}
\] (8.21) (8.22)

Also we will need to compute the divergence of scalar and vector fields, to this aim we use the operators

\[
\text{div}_N v^N := V_N(\tilde{D}_N^I + \tilde{D}_N^J) v^N
\]
and

\[
\text{Div}_N v^N := V_N(\tilde{D}_N^I v^N_1 + \tilde{D}_N^J v^N_2).
\]

8.2.2 Steps two and three

If we go back to Subsubsection 8.1.4, we can easily check that the steps II and III consists of solving a large number of differential problems, each of the form

\[
\begin{align*}
\begin{cases}
H_\sigma v^N &= f \\
v^N &= g
\end{cases} \quad \in \Omega_N \\
&\quad \text{on } \Gamma^\text{int}_N
\end{align*}
\] (8.23)

where \(f, g, \sigma\) may vanish. Note that we formulated this last equation in the scalar form: each vectorial differential problem in Subsubsection (8.1.4) needs to be first re-written in this form.

Now we recall that we can represent, using (8.21) and (8.22), the evaluation of the Helmotz operator \(H_\sigma\) on \(\Omega_N\) by a composition of matrices. More precisely, denoting by \(V_N^\Omega\) the matrix \(V_N\) where the first 4\(N\) rows (relative to boundary nodes) have been removed, we have

\[
H_\sigma v^N = V_N^\Omega \left( (\tilde{D}_N^I)^2 V_N^{-1} + (\tilde{D}_N^J)^2 V_N^{-1} - \sigma I \right) v^N.
\]

Thus the above system is equivalent to

\[
\begin{pmatrix}
V_N^\Omega \left( (\tilde{D}_N^I)^2 + (\tilde{D}_N^J)^2 \right) V_N^{-1} - \sigma I \\
V_N^T
\end{pmatrix}
\begin{pmatrix}
v_N(\eta_1) \\
v_N(\eta_{N+1})^2
\end{pmatrix} =
\begin{pmatrix}
f_N \\
g_N
\end{pmatrix}
\] (8.24)

Here \(V_N^T\) is the matrix obtained starting by \(V_N\) and deleting all rows not corresponding to knots of \(\Gamma^\text{int}_N\).

It is clear that the (rather large and) very sparse matrix \(L_N := V_N^\Omega \left( (\tilde{D}_N^I)^2 + (\tilde{D}_N^J)^2 \right) V_N^{-1}\) must be computed once for all and stored.

The solution to the system may be carried out in several ways, the QR algorithm may be a good choice but iterative methods should be considered as well due to the sparsity of the system.

Let us recall that the (~problem I) and (~problem II) leads to one scalar and one vector problem, while the ~\(l\)-problems consists of one scalar and one vector problem each. Hence
the total number of system of the type (8.24) are $6L+3 = 24N-21$ but we should consider
the evaluations of the divergence of the numerical flux (which have explicit solution, see (??)) as well. The possibility of calculating the solutions by parallel computing should be considered.

8.2.3 Step four

To assemble the influence matrix we use the discrete divergence operator $\text{Div}_N$ in order
to represent the divergence acting on $\mathbb{P}^N \times \mathbb{P}^N$. The matrix representation of $\text{Div}_N$ has
been already computed and stored at the preprocessing stage step I.

We already mentioned that the influence matrix may be not invertible. We present two
methods to cope with this issue, the first one, due to . . . , is a spectral regularizing technique
that leads to recover a true solution if there exists one. The second is a mixed approach
between exact solution and least squares approximation.

**Regularization method.** We compute the factorization

$$I^N = U^T \Sigma V, \quad \Sigma = \| \sigma, \quad \sigma = (\sigma_1, \ldots, \sigma_{2L-4}, 0, 0, 0)$$

and we replace $I^N$ by the regularized matrix

$$I^N_\lambda := U^T \Sigma_\lambda V, \quad \Sigma_\lambda = \| \sigma_\lambda, \quad \sigma_\lambda = (\sigma_1, \ldots, \sigma_{2L-4}, \lambda, \lambda, \lambda, \lambda)$$

Then we solve the system $I^N_\lambda \xi = c$. Note that, if a "true" solution $\hat{\xi}$ of the original system
exists, then $\hat{\xi}$ is a solution of the regularized system as well.

**Mixed approach.** We divide the influence matrix $I^N \in M_{2L \times 2L}(\mathbb{R})$ in two sub-matrices

$$A, B \in M_{L \times L}(\mathbb{R})$$

We solve exactly the problem $A \xi = (c_1, \ldots, c_L)^T$. To do that we use the QR factorization

$$A = R^T Q^T = \begin{bmatrix} R^T_1 \\ \odot \end{bmatrix} Q^T.$$
We get
\[\boldsymbol{\xi}^1 := (\xi_1, \ldots, \xi_M)^T = QR_1^{-T}c_1, \quad M := \text{Rank } A.\]

Then we split the matrix \( B \) as \( B = [B_1, B_2] \) with \( B_1 \in M_{L,M}(\mathbb{R}) \) thus the condition \( B\xi = c_2 \) is equivalent to
\[B_1\xi^1 + B_2\xi^2 = c_2 \iff B_2\xi^2 = c_2 - B_1\xi^1.\]

This last equation is finally solved with respect to the variable \( \xi^2 \) in the least squares sense.