Chapter 2

The Finite Volume method in 1D and its application to Maxwell's equations

In this chapter, we discuss a finite volume discretization for some simple problems. As we see next, the 1D is in the core of 2 and 3D computations and therefore we give much attention to the 1D problem where things are much simpler. Our derivation follows mimetic finite volume methods presented in [48, 49, 47, 46] which can be thought of as an extension to the famous Yee discretization [83] used in most finite difference/volume discretizations of Maxwell's equations.

At this point we would like to note that although finite element methods (FEM) are also common and can be used for the discretization, we chose to not discuss them in this exposition, for the sake of brevity. We would like to note that in spite of the (sometimes) heated discussion about the advantages and disadvantages of FEM compared with FV the methods are very similar, especially if they are both derived from the weak form. We do not believe that one technique is superior to the other but that this is really a question of personal taste.

2.1 Deriving the discretization

Maxwell's equations in 1D medium can be written as

$$\begin{aligned} i\omega b &= -\partial_z e, \\ s &= \partial_z(\mu^{-1}b) - \sigma(z)e, \end{aligned}$$

where $e = \widehat{\vec{E}}_x$ and $b = \widehat{\vec{B}}_y$.

We rewrite the equations in weak form as

$$i\omega(b, f) = -(\partial_z e, f), \tag{2.1a}$$

$$(s, w) = -(\mu^{-1}b, \partial_z w) - (\sigma(z)e, w), \tag{2.1b}$$

where f is in the same functional space as b and w is in the same functional space as e. Note the minus sign in the inner product $(\mu^{-1}b, \partial_z w)$ that results from the integration by parts.

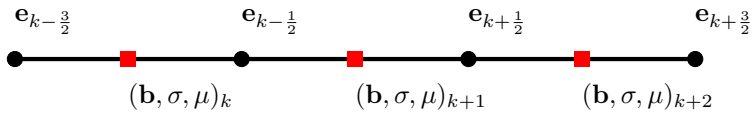

Figure 2.1. *1D grid for nodal discretization*

To discretize the system, we consider a simple 1D mesh plotted in Figure 2.1 The functions e and b are discretized by grid functions $\mathbf{e}$ and $\mathbf{b}$. Here we face our first dilemma. At which points should we discretize the fields? To answer this question, consider the inner product on a cell with uniform material properties, that is an interval $[x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}]$. A natural way to discretize the derivative of a quantity u that is discretized on the nodes is

$$(\partial_z u)_{x_k} \approx h_k^{-1}(\mathbf{u}_{k+\frac{1}{2}} - \mathbf{u}_{k-\frac{1}{2}}) + \mathcal{O}(h^2).$$

Recall - Order: We say that v_h converge to u in order h^p if $|v_h - u| = Ch^p$ where C is independent of h

Since the derivatives in the weak form operate on the electric field, e, it makes sense to discretize e on the nodes of our mesh and since $\partial_z e$ "lives" in the same place where the magnetic flux, b, "lives", this implies that b should be discretized on the cell-center of our mesh leading to the well known **staggered grid**.

To complete the discretization we note that we need to compute 2 inner products. One involves a cell centered variable and the other involved a nodal variable. The derivation inner products for variables that live in the cell center is straight forward. Consider two grid functions $\mathbf{b}$ and $\mathbf{f}$ discretized on cell centers. Using the midpoint rule we obtain

$$(b, f) \approx \sum_k h_k \mathbf{b}_k \mathbf{f}_k + \mathcal{O}(h^2).$$

For the discretization of inner products that involve with nodal variables, one must be a little bit more careful. Consider the inner product $(\sigma e, w)$ and a nodal discretization of the functions by grid functions $\mathbf{e}$ and $\mathbf{w}$. A second order accurate discretization of this product can be written as

$$(\sigma e, w) \approx \sum_k \frac{h_k \sigma_k}{4}(\mathbf{e}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}})(\mathbf{w}_{k+\frac{1}{2}} + \mathbf{w}_{k+\frac{1}{2}}) + \mathcal{O}(h^2).$$

That is, we average e and w to the cell center and then compute the inner product. However this inner product is **inappropriate** to use. When considering an inner product we need to note that the inner product is not only converging to the true

value but also that it does not contain non-trivial null spaces. Consider the inner product $(\sigma w, w)$ for a nontrivial w and a positive σ. Clearly, this product must be greater than zero. However, the above inner product does not yield this property! Consider the vector $\mathbf{w} = (-1, 1, -1, \ldots)$. Then clearly this approximation yield $(\sigma w, w) \approx 0$ and this is clearly a wrong approximation for a highly oscillating function. An inner product with the same accuracy but with no null space can be obtained by considering the approximation

$$(\sigma e, w) \approx \sum_k \frac{h_k \sigma_k}{4} (\mathbf{e}_{k+\frac{1}{2}} \mathbf{w}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}} \mathbf{w}_{k+\frac{1}{2}}) + \mathcal{O}(h^2). \tag{2.2}$$

This phenomena is a fundamental observation in discretization techniques and it leads to the following principle for many numerical methods

> When some function $f(w)$ needs to be averaged/interpolated and grid function $\mathbf{w}$ is used for the discretization it is better to first compute (pointwise) $f(\mathbf{w})$ and only then average/interpolate

This principle will be used later when we discretize the inverse problem as well.

2.2 Matrix Representation

Matrix representation of the equations allow us to rewrite the equations in a much simpler and more apparent form. To this end, we consider first the discretization of the gradient of the nodal function e. The matrix form of the gradient is simply

$$\mathsf{GRAD}\ \mathbf{e} = \begin{pmatrix} h_1^{-1} & & & \\ & h_2^{-1} & & \\ & & \ddots & \\ & & & h_n^{-1} \end{pmatrix} \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \vdots \\ \\ \mathbf{e}_{n+1} \end{pmatrix},$$

that can be written in matrix form as

$$e_z \approx \mathbf{L}^{-1} \mathbf{G} \mathbf{e},$$

where $\mathbf{L} = \mathrm{diag}(\mathbf{h})$ contains the length of the each edge and $\mathbf{G}$ is a matrix with values of ± 1 that represents the topology of our mesh. As we will see later, this representation is common for 3D as well as 1D and therefore is very useful.

Next, consider the inner products. It is straight forward to verify that the cell center inner products have the form

$$(b, f) \approx \mathbf{b}^\top \mathbf{M}^f \mathbf{f} \quad \text{and} \quad (\mu^{-1} b, f) \approx \mathbf{b}^\top \mathbf{M}_\mu^f \mathbf{f},$$

where $\mathbf{M}^f_\mu = \text{diag}(\mathbf{h} \odot \mu^{-1})$ and $\mathbf{M}^f = \text{diag}(\mathbf{h})$. To express the nodal averaging in matrix form we note that

$$\sum_k \frac{h_k \sigma_k}{4}(\mathbf{e}_{k+\frac{1}{2}}\mathbf{w}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}}\mathbf{w}_{k+\frac{1}{2}}) = (\mathbf{h} \odot \sigma)^\top(\mathbf{A}_v(\mathbf{e} \odot \mathbf{w})) = \mathbf{w}^\top \text{diag}(A_v^\top(\mathbf{h} \odot \sigma))\,\mathbf{e},$$

where $\odot$ is the point wise Hadamard product and $\mathbf{A}_v$ is an averaging matrix from nodes to cell center

$$\mathbf{A}_v = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & & \\ & \ddots & \ddots & \\ & & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

We define the matrix

$$\mathbf{M}^e_\sigma = \text{diag}(A_v^\top(\mathbf{h} \odot \sigma)).$$

Using the above matrices we obtain the following discretization for 1D Maxwell's equations in weak form

$$\begin{aligned} i\omega\, \mathbf{f}^\top \mathbf{M}^f \mathbf{b} &= -\mathbf{f}^\top \mathbf{M}^f \mathbf{L}^{-1} \mathbf{G}\mathbf{e}, \\ \mathbf{w}^\top \mathbf{M}^e \mathbf{s} &= -\mathbf{w}^\top \mathbf{G}^\top \mathbf{L}^{-1} \mathbf{M}^f_\mu \mathbf{b} - \mathbf{w}^\top \mathbf{M}^e_\sigma \mathbf{e}. \end{aligned}$$

These equations are discretized for arbitrary grid functions $\mathbf{w}$ and $\mathbf{f}$ and therefore we must have that

$$i\omega\, \mathbf{b} = -\mathbf{L}^{-1}\mathbf{G}\mathbf{e}, \tag{2.3a}$$

$$\mathbf{M}^e \mathbf{s} = -\mathbf{G}^\top \mathbf{L}^{-1} \mathbf{M}^f_\mu \mathbf{b} - \mathbf{M}^e_\sigma \mathbf{e}. \tag{2.3b}$$

We can now eliminate $\mathbf{b}$ from the equations obtaining an equation for $\mathbf{e}$ only

$$(\mathbf{G}^\top \mathbf{L}^{-1} \mathbf{M}^f_\mu \mathbf{L}^{-1} \mathbf{G} + i\omega \mathbf{M}^e_\sigma)\mathbf{e} = -i\omega \mathbf{M}^e \mathbf{s}. \tag{2.4}$$

2.3 Implementation of boundary conditions

In the above we did not discuss the implementation of boundary conditions. Homogeneous boundary conditions for b are incorporated into the discretization. This can be seen from the inner product

$$(\partial_z \mu^{-1} b, w) = -(\mu^{-1} b, \partial_z w) + (\mu^{-1} b w)|_0^L,$$

where we have assumed that the domain of integration is $[0, L]$. Previously, we have dropped the boundary conditions implying $b = 0$ at the ends of the interval. In the case that the boundary conditions are not homogeneous we need to incorporate the conditions in the discretization. In matrix vector form, this condition can be written as

$$(\mu^{-1} b w)|_0^L = \mathbf{b}_{\text{bc}}^\top (\mathbf{B}\mathbf{w}),$$

where the matrix $\mathbf{B}$ extracts the boundary elements in $\mathbf{w}$ and $\mathbf{b}_{\text{bc}}$ are the known boundary conditions. For the simple 1D case here we have

$$\mathbf{B} = \begin{pmatrix} -1 & 0 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}.$$

This implies that equation (2.1b) in the weak form is changed to

$$\mathbf{M}^e\mathbf{s} = -\mathbf{G}^\top\mathbf{L}^{-1}\mathbf{M}^f_\mu\mathbf{b} - \mathbf{M}^e_\mu\mathbf{e} + \mathbf{B}^\top\mathbf{b}_{\text{bc}},$$

and that the boundary conditions play the role of a source.

Next we discuss the implementation of the boundary conditions on e, that are straight forward to implement. We rewrite the system that was obtained without boundary conditions on $\mathbf{e}$ as

$$\mathbf{A}\mathbf{e} = \mathbf{q}.$$

The boundary condition on $\mathbf{e}$ implies that we know two elements in $\mathbf{e}$, $\mathbf{e}_1$ and $\mathbf{e}_{\text{end}}$. We can partition the unknowns as $\mathbf{e} = [\mathbf{e}_I^\top, \mathbf{e}_B^\top]^\top$ where $\mathbf{e}_B$ are the boundary unknowns and $\mathbf{e}_I$ are the internal nodes. We can further partition the matrix as

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{II} & \mathbf{A}_{IB} \\ \mathbf{A}_{BI} & \mathbf{A}_{BB} \end{pmatrix}.$$

Since the boundary conditions are known the values of $\mathbf{e}_B$ are known and we can reduce the system to obtain the equation for the known boundary conditions

$$\mathbf{A}_{II}\mathbf{e}_I = \mathbf{q}_I - \mathbf{A}_{IB}\mathbf{e}_B.$$

Finally, boundary conditions of the form $\alpha e + b = c$ can be applied and this is left as an problem to the reader.

2.4 1D in practice

We now give our attention to the actual implementation of the 1D problem using matlab. Notice that we have the following matrices to be generated

- A difference matrix from cells to nodes, $\mathbf{G}$: nodes $\rightarrow$ cell centers
- Averaging matrices from nodes to cells, $\mathbf{M}^f$: nodes $\rightarrow$ cell center
- Diagonal matrices with edge length.

To start, the following code generates the 1D difference from nodes to cell centers on a mesh with n cells

```
function[G] = ddx(n)
% [G] = ddx(n)
%
G = spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);
```

Next, the diagonal matrix with the volumes on its diagonal can be generated by the simple command

```
function[V] = sdiag(v)
% [V] = sdiag(n)
%
V= spdiags(v(:),0,numel(v),numel(v));
```

Finally, the averaging matrix can be obtained by using a similar structure to the derivative matrix

```
function[Av] = ave(n)
% Av = ave(n)
%
Av = spdiags(ones(n+1,1)*[1/2 1/2],[0,1],n,n+1);
```

Using these building blocks we can now quickly write a program that discretizes Maxwell's equations in 1D. We now discuss testing our code by showing how to setup a problem.

2.5 Programmer note: Testing the code - The method of fictitious sources

The first rule in any code writing is that there must be bugs somewhere and therefore testing the code is crucial. In fact, we should not consider the code as "correct" unless it was tested and retested. In many cases, developing a "good enough" test can be as complicated as writing the code, nonetheless, testing is necessary if we want to use the code and make some conclusions from the results.

In our case here, we have developed a second order discretization for sufficiently smooth problems. Assuming no bugs exist in the theoretical development of the discretization, we would like to verify that the code behaves like the theory predicts.

The method to test the code is referred to as the method of fictitious sources. The idea is to choose functions $e(x), b(x), \mu(x)$ and $\sigma(x)$ and then "plug" them into the PDE obtaining some nontrivial right hand side. The right hand side can be thought of as a fictitious source. Next, we discretize the PDE and the fictitious source on a mesh and try to recover the known solution. The "trick" is to choose the fields $e(x)$, $b(x)$ and the parameter functions $\sigma(x)$ and $\mu(x)$ such that the test is indicative of our numerical method.

Here we use a rather simple test. We choose smooth functions for σ, μ, e and b on the interval $[0, 1]$. We make sure that $b(0) = b(1) = 0$ that are the simple boundary conditions we test. To make the test somewhat challenging we use a random mesh for our discretization. We then solve a sequence of problem with a

decreasing mesh-size and setup the linear system on each of these meshes, solving the problem and recording the solution error.

The code for setting up the problem is as follows

```
% test 1D EM
clear all;
close all;

%% setup the analytic example
omega = 1;
sigma = @(x) x.^2+1;
mu    = @(x) cos(x) + 2;

b     = @(x) x.*(x-1).*mu(x);  % note b(0) = b(1) = 0;
e     = @(x) cos(2*pi*x);

% The system
% 1i*w*b + e'           = s1
% (1/mu * b)' - sigma*e = s2

% fictitious source
s1 = @(x) 1i*omega*b(x) - 2*pi*sin(2*pi*x);
s2 = @(x) 2*x-1 - sigma(x).*e(x);
```

Using the functions we can now run the following testing

mesh points	L2 b	L2 e	L_∞ b	L_∞ e
8	1.01e-02	3.79e-02	3.80e-02	5.42e-02
16	1.78e-03	6.38e-03	1.18e-02	1.37e-02
32	2.30e-04	9.39e-04	2.79e-03	3.19e-03
64	3.29e-05	2.01e-04	6.58e-04	9.97e-04
128	1.00e-05	3.71e-05	2.24e-04	2.26e-04
256	1.90e-06	7.54e-06	6.82e-05	6.09e-05
512	3.37e-07	1.47e-06	1.58e-05	1.53e-05
1024	5.64e-08	2.47e-07	4.15e-06	4.04e-06
2048	1.00e-08	4.28e-08	9.96e-07	9.99e-07

Table 2.1. *Errors in the recovered solutions of b and e.*

```
%% Run numerical setting

for n = [8  16  32  64  128  256 512  1024   2048]
  % setup a random nodal mesh
  h = rand(n,1)*0+1; L = sum(h); h = h/L;
  x = [0; cumsum(h)];
  % cell-centered mesh
  xc = x(1:end-1) + diff(x)/2;

  % the sources
  sh = s1(xc);
  se = s2(x);

  % the linear system
  nc = length(sigma(xc));

  G    = ddx(nc);
  Av   = ave(nc);
  Linv = sdiag(1./h);
  Mmu  = sdiag(h./mu(xc));
  Msig = sdiag(Av'*(sigma(xc).*h));
  M    = sdiag(Av'*h);

  % set the matrix system
  % [1i*w         d/dz] [b] - [s1]
  % [1/mu d/dz  -sigma] [e] - [s2]

  A = [1i*omega*speye(n)  Linv*G; ...
       -G'*Linv'*Mmu      -Msig];
  bb = [sh; M*se];


  eh = A\bb;
  subplot(2,1,1)
  plot(xc,real(eh(1:n)),xc,real(b(xc)),'r');
  subplot(2,1,2)
  plot(x,real(eh(1+n:end)),x,real(e(x)),'r');

  fprintf('L2 error  %3.2e %3.2e  Linf error %3.2e  %3.2e\n', ...
          norm(Linv\(real(eh(1:n) -b(xc)))), ...
          norm(Linv\(0.5*real(eh(1+n:end-1)+eh(2+n:end))-e(xc))), ...
          norm(real(eh(1:n) -b(xc)),'inf'), ...
          norm(real(eh(1+n:end)-e(x)),'inf'));

  pause(0.5)

end
```

The convergence is presented in Table 2.1 Note that even though the mesh is random, the overall convergence is $\mathcal{O}(h^2)$ where h is the averaged mesh size.

For every code that is written, similar (and even more robust) tests must be performed before we pass on to the stage of trying to solve a realistic problem. Unfortunately, far reaching conclusions on issues such as global warming and nuclear storage have been reached in the past based on "gently" tested codes. One should be always a bit skeptic about the code and continue to evaluate it. Remember, the usual phrase is:

the only code without bugs is a code that is not being used.

2.6 The 1D MT problem

Finally, we use our discretization to solve the 1D magnetotelluric problem

$$i\omega b = -\partial_z e, \quad 0 = \partial_z(\mu^{-1}b) - \sigma(z)e, \quad b(0) = 1 \quad b(-\infty) = 0.$$

The problem we face when discretizing the differential equation is the boundary condition at $-\infty$. Since discretized problem can be only expressed on a finite domain we need to limit the computations to an interval $[-L, 0]$ and the question is, how should we choose L. There are two approaches to resolve this issue. First, it is possible to discretize the problem on a large enough domain $[-L, 0]$ such that $b(-L) \approx 0$. This approach can be highly inefficient since L can be large. A better approach is to note that assuming that σ and μ are constant for $z < -L$, the solution has the form

$$e = c\exp(ikz) \quad \text{where} \quad k = -\sqrt{i\omega\mu\sigma} \quad \text{and} \quad b = -ikc\omega^{-1}\exp(ikz). \quad (2.5)$$

Therefore for $z < -L$ the equation $i\omega b + e_z = 0$ imply that

$$\omega b - ke = 0.$$

This replaces the boundary condition at $-\infty$ with a boundary condition at $z = -L$ and we can use a much smaller domain for the discretization. To deal with this boundary condition one needs to use ghost points (see exercises). In the script below we avoid ghost points and set the boundary conditions at a depth of 3 times the skin depth. We compute the electric field on the surface and extract its absolute value and phase as a function of frequency.

```
% setup and test the MT problem in 1D
clear all;
close all;

omega = logspace(1,4,32);
mu    = 4*pi*1e-7;

% setup the basic mesh
sig0 = 1e-2;
skin = sqrt(2./omega/mu/sig0(end));


for j = 1:length(omega)

  % setup a deep enough mesh
  L  = 3*skin(j);

  h = diff(linspace(0,2*skin(j),2049));
  h = h(:);
  while 1,
      h = [h;h(end)*1.1];
      if sum(h) > L, break; end
  end

  n   = length(h);
  sig = ones(n,1) * sig0;

  % mesh
  z = [0; cumsum(h)];
  % cell-centered mesh
  zc = z(1:end-1) + diff(z)/2;

  % the linear system

  G    = ddx(n);
  Av   = ave(n);
  Linv = sdiag(1./h);
  Mmu  = sdiag(h./mu);
  Msig = sdiag(Av'*(sig.*h));
  M    = sdiag(Av'*h);

  % set the matrix system
  % [1i*w         d/dz] [b] - [s1]
  % [1/mu d/dz  -sigma] [e] - [s2]

  A = [1i*omega(j)*speye(n)  Linv*G; ...
        -G'*Linv'*Mmu      -Msig];

  % seup boundary conditions b(0) = 1
  bb = -A(2:end,1);
  A  = A(2:end,2:end);

  % solve
  be = A\bb;
  b = [1;be(1:n-1)];
  e = be(n+1:end);

  % extract MT data
  d(j) = e(1);


end

subplot(2,1,1)
semilogx(omega,1./omega(:)*mu .* abs(d(:)).^2)
title('apparent resistivity')
subplot(2,1,2)
semilogx(omega,angle(d))
title('phase')
```

The MT problem in 1D has an analytical solution (see previous chapter). Thus, the code can be verified and checked for accuracy. The MT data are the electric field in the x-direction divided by the magnetic field in the y-direction on

the surface. If we define $b = \vec{B}_y$ on the surface as 1 then the data are simply $e = \vec{E}_x$. It is possible to use the ratio between the electric field and the magnetic flux in order to estimate the conductivity. Using the analytic solution (2.5) the apparent resistivity is defined

$$\rho_a = \frac{\mu}{\omega} \left| \frac{e}{b} \right|^2 .$$

A first test for any MT code is that if one uses a homogeneous space then the apparent resistivity is retrieved. This can be seen in the example above.

2.7 Exercises

1. Derive the discretization for the mixed boundary condition $\alpha e + b = c$. Hint: use the inner product

   $$(\partial_z b, w) = -(b, \partial_z w) + (bw)_0^{\text{end}},$$

   and

   $$b_0^{\text{end}} = c_0^{\text{end}} - \alpha e_0^{\text{end}}.$$

2. Discretize the MT problem using your mixed boundary condition. Report on the accuracy when simulating data.

3. Build your own example for the fictitious sources. Use a solution that has sharper derivatives and comment about the accuracy obtained.

Chapter 3

Finite Volume Discretization in 3D

In this chapter we expand the 1D discussion into 2D and 3D. As we see next, most of the discussion is straight forward extension of the 1D case. We start by discussing the discretization of differential operators followed by the inner products on an orthogonal, tensor mesh. We then discuss the discretization of the operators and inner products on a logically orthogonal mesh. Such meshes give greater flexibility when working on problems with complex geometry.

The first part of this chapter is a straight forward description of the Yee method [83] as been also explained in [35]. The second part of this chapter is more advanced and is based on [48, 49, 47, 46].

Just as in 1D, the discretization on any mesh (regular or irregular) is made from two parts. First, the discretization of the differential operators is needed and second, the discretization of inner products is approximated. As we see next, the discretization of the differential operators is straight forward while the discretization of inner products is easy on regular mesh and requires some attention on any other mesh.

3.1 Discretization of differential operators on a tensor mesh

The extension of the staggered discretization proposed in 1D to 2D and 3D is slightly more involved. In principle, we have 4 types of variables (see Figure 3.1). First, scalar fields can be placed on cell-centered or on nodes. Next, we need to consider faces and edge variables that "live" on faces and edges respectively. While face variables are assumed to be normal to the cell face, edge variables are aligned with the cell edge.

The physical interpretation of face variables are fluxes, since they represent a flux of some sort into or out of a cell. The edge variables are fields as they represent some path around the surface. The physical meaning of these variables explains our representation of Maxwell's equations in terms of a field $\vec{E}$ (that is discretized on

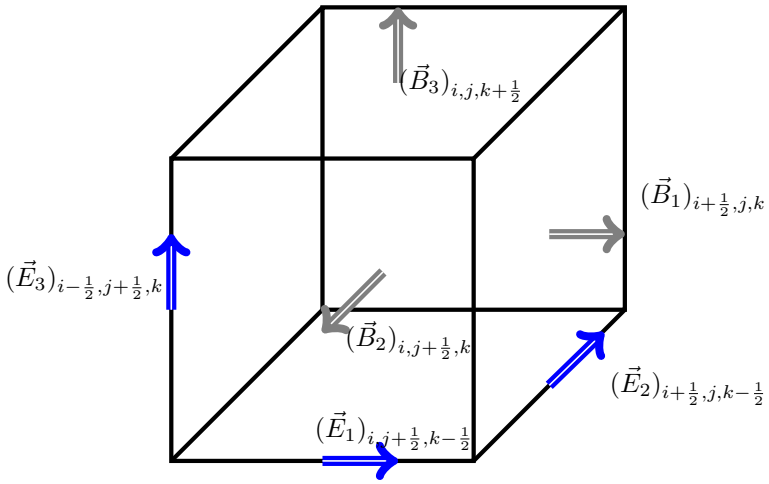

Figure 3.1. *Staggered discretization in 3D*

the edges) and a flux $\vec{B}$ (that is discretized on the faces). As we see next, cell-center and nodal scalars also have continuous variables that correspond to their physical meaning. A typical mesh in 3D is plotted in Figure 3.1.

We now use simple integration to discretize the differential operators on a staggered mesh. The basic idea is to use the integral definition of each differential operator. These definitions are

$$\nabla \cdot \vec{B} = \lim_{\Omega \to 0} \frac{\int_{\partial\Omega} \vec{B} \cdot \vec{n} \, dS}{\int_{\Omega} dV} \tag{3.1a}$$

$$(\nabla \times \vec{E}) \cdot \vec{n} = \lim_{S \to 0} \frac{\oint_{\partial S} \vec{E} \cdot d\vec{\ell}}{\int_{S} da} \tag{3.1b}$$

$$(\nabla \varphi) \cdot (\vec{x}_2 - \vec{x}_1) = \lim_{|\vec{x}_2 - \vec{x}_1| \to 0} \frac{\varphi(\vec{x}_2) - \varphi(\vec{x}_1)}{|\vec{x}_2 - \vec{x}_1|} \tag{3.1c}$$

The DIV

To derive a discretization of the divergence we simply apply the integral form to obtain

$$\nabla \cdot \vec{B} \approx \frac{1}{V_{ijk}} \int_{\Omega_{ijk}} \nabla \cdot \vec{B} \, dV = \frac{1}{\Omega_{ijk}} \int_{\partial\Omega_{ijk}} \vec{B} \cdot \vec{n} \, ds \approx \tag{3.2}$$

$$V^{-1} \left(\sum_{\text{in/out flux}} (\vec{B} \cdot \vec{n}_{\text{in}}) \Delta s_{\text{in}} - (\vec{B} \cdot \vec{n}_{\text{out}}) \Delta s_{\text{out}} \right)$$

Assuming a tensor mesh where $\vec{B}$ is on its faces using a grid function $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ and denoting the area of each face as a face vector as $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$

$$\nabla \cdot \vec{B} \approx V_{ijk}^{-1} \Big((\mathbf{a}_1 \mathbf{b}_1)_{i+\frac{1}{2},j,k} - (\mathbf{a}_1 \mathbf{b}_1)_{i-\frac{1}{2},j,k} + (\mathbf{a}_2 \mathbf{b}_2)_{i,j+\frac{1}{2},k} - (\mathbf{a}_2 \mathbf{b}_2)_{i,j-\frac{1}{2},k} + (\mathbf{a}_3 \mathbf{b}_3)_{i,j,k+\frac{1}{2}} - (\mathbf{a}_3 \mathbf{b}_3)_{i,j,k-\frac{1}{2}} \Big) \tag{3.3}$$

Similar to the 1D case, we separate the operator into a part that contain the geometry and the part that contains the mesh parameters. We set

$$\mathbf{F} = \text{diag}(\mathbf{a}) \quad \text{and} \quad \mathbf{V} = \text{diag}(\mathbf{v})$$

where $\mathbf{a}$ and $\mathbf{v}$ are vectors of face sizes and cell volumes, then we obtain that

$$\mathsf{DIV} = \mathbf{V}^{-1}\mathbf{D}\mathbf{F}$$

where $\mathbf{D}$ is a matrix of ± 1 that is has the mesh topology.

Note that we somewhat abuse notation to express the vectors$\mathbf{v}$, $\mathbf{a}$ and $\mathbf{b}$ both as 3D grid functions and vectors, that is we do not distinguish between $\mathbf{v}$ and $\mathsf{vec}(\mathbf{v})$ and the meaning should be understood from the context. We continue using this notation next.

The CURL

To discretize the curl we use Gauss rule

$$\int_{\mathcal{S}} \nabla \times \vec{E} \cdot dS = \oint_{\Gamma} \vec{E} \cdot d\ell$$

where $\mathcal{S}$ is any surface closed with a path Γ. Using the tensor mesh and Gauss's rule, for example, on the x faces (i.e. faces that their normal is parallel to the x axis we obtain the first component of the **curl**

$$(\nabla \times \vec{E})_1 \approx \frac{1}{|S|} \int_{\mathcal{S}} \nabla \times \vec{E} \cdot dS =$$
$$\frac{1}{\mathbf{a}_{i+\frac{1}{2},j,k}} \Big((\ell_3 \mathbf{e}_3)_{i+\frac{1}{2},j+\frac{1}{2},k} - (\ell_3 \mathbf{e}_3)_{i+\frac{1}{2},j-\frac{1}{2},k} + (\ell_2 \mathbf{e}_2)_{i+\frac{1}{2},j,k-\frac{1}{2}} - (\ell_2 \mathbf{e}_2)_{i+\frac{1}{2},j,k+\frac{1}{2}} \Big)$$

where $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ is the discretization of the edge electric field and ℓ is the length of the edges. Similar expressions are obtained by integrating on the y and z faces.

Separating the operator into a part that contain the geometry and the part that contains the mesh parameters we set

$$\mathbf{F} = \text{diag}(\mathbf{a}) \quad \text{and} \quad \mathbf{L} = \text{diag}(\ell)$$

where $\mathbf{a}$ and ℓ are vectors of face area and edge length, and we let $\mathbf{C}$ be the matrix with ± 1 on its diagonals then we obtain that

$$\mathsf{CURL} = \mathbf{F}^{-1}\mathbf{C}\mathbf{L}.$$

The GRAD

In a similar way we can discretize the gradient of a nodal function. Let φ be such a function then we have that $\nabla\varphi$ can be approximated on cell edges by the following short differences

$$(\varphi_x)_{i,j+\frac{1}{2},k+\frac{1}{2}} \approx \frac{\varphi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \varphi_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{\ell_{i,j+\frac{1}{2},k+\frac{1}{2}}}$$
$$(\varphi_y)_{i+\frac{1}{2},j,k+\frac{1}{2}} \approx \frac{\varphi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \varphi_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}}{\ell_{i+\frac{1}{2},j,k+\frac{1}{2}}}$$
$$(\varphi_z)_{i+\frac{1}{2},j+\frac{1}{2},k} \approx \frac{\varphi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \varphi_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}}{\ell_{i+\frac{1}{2},j+\frac{1}{2},k}}$$

Separating the discretization of the operator into a part that contain the geometry and the part that contains the mesh parameters we obtain

$$\mathsf{GRAD} = \mathbf{L}^{-1}\mathbf{G}. \tag{3.4}$$

where again $\mathbf{G}$ is a matrix that contains the ± 1 and the mesh topology.

Properties of the discrete operators

It is possible to verify that for the matrices that contain the geometry of the mesh we have

$$\mathbf{CG} = 0 \quad \mathbf{DC} = 0$$

It is also possible to verify that $\mathbf{G}$ spans the nontrivial null space of $\mathbf{C}$.

This observation has important consequences known as the mimetic properties of the discrete linear operators. The CURLis an edge to face **curl** operator, the GARD is a nodal gradient to edge operator and DIV is a face to cell-center divergence operator. It is straight forward to show (involves a bunch of indices) that

$$\begin{aligned}
&\mathsf{CURL}\ \mathsf{GRAD} = 0\\
&\mathsf{DIV}\ \mathsf{CURL} = 0\\
&\mathsf{CURL}^\top\ \mathsf{CURL}\ +\ \mathsf{GRAD}\ \mathsf{GRAD}^\top\ =\ -\vec{\Delta}_{\text{Edges}}\\
&\mathsf{CURL}\ \mathsf{CURL}^\top\ +\ \mathsf{DIV}^\top\ \mathsf{DIV} = -\vec{\Delta}_{\text{Faces}}
\end{aligned}$$

where $\vec{\Delta}$ is the discretization of the vector Laplacian on the Face/Edge variables.

These properties are often refer to as **mimicking** properties as they mimic the properties of the continuous differential operators. In many physical simulations such as fluid flow and electromagnetics these properties are essential as they keep some of the conservation properties of the underling PDE. For example, the above discretization applied to Maxwell's keeps the magnetic field divergence free, which can be crucial in many physical simulations.

3.2 Inner Products on tensor mesh

Our discretization is based on Maxwell's equations in weak form. The equations are presented in weak form and therefore we need to discretize 3 different weak forms, of cell center, face and edge variables. We now discuss the discretization of each inner product.

3.2.1 Cell center inner product

Maybe the simplest discretization is of the cell inner product. Using a midpoint approximation we obtain that if u and q are functions discretized in cell centers by the vectors $\mathbf{u}$ and $\mathbf{q}$ then

$$(u, q) = \int_\Omega u(\vec{x})q(\vec{x})\ dV \approx \sum(\mathbf{vuq})_{i,j,k} = \mathbf{u}^\top \mathbf{V}\mathbf{q}$$

3.2.2 Face variables inner product

For face variables inner products we note that we discretize a vector inner product. More specifically

$$(\vec{B}, \vec{F}) = \int_\Omega \vec{B}(\vec{x})^\top \vec{F}(\vec{x})\ dV = \int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x}) + \vec{B}_2(\vec{x})\vec{F}_2(\vec{x}) + \vec{B}_3(\vec{x})\vec{F}_3(\vec{x})\ dV$$

To compute the product in the of the $\vec{x}_1$ components we note that this is equivalent to the 1D case and therefore‘

$$\int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x})\ dV \approx \sum \frac{\mathbf{v}_{i,j,k}}{2}\left((\mathbf{b}_1\mathbf{f}_1)_{i+\frac{1}{2},j,k} + (\mathbf{b}_1\mathbf{f}_1)_{i-\frac{1}{2},j,k})\right)$$

This can be expressed in matrix form as

$$\int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x})\ dV \approx \mathbf{v}^\top \mathbf{A}_{f1}(\mathbf{b}_1 \odot \mathbf{f}_1) = \mathbf{f}_1^\top \operatorname{diag}(\mathbf{A}_{f1}^\top \mathbf{v})\mathbf{b}_1$$

where $\mathbf{b}_1$ is the discretization of $\vec{B}_1$ on the faces in the $\vec{x}_1$ direction. Similarly

$$\int_\Omega \vec{B}_2(\vec{x})\vec{F}_2(\vec{x})\ dV \approx \mathbf{f}_2^\top \operatorname{diag}(\mathbf{A}_{f2}^\top \mathbf{v})\mathbf{b}_2 \quad \text{and} \quad \int_\Omega \vec{B}_3(\vec{x})\vec{F}_3(\vec{x})\ dV \approx \mathbf{f}_3^\top \operatorname{diag}(\mathbf{A}_{f3}^\top \mathbf{v})\mathbf{b}_3$$

The sparse matrices $\mathbf{A}_{fj}$ average face variables in the j-th direction into the cell centers and it contains only $\frac{1}{2}$ where the faces are averaged.

Finally, we put it all together to obtain

$$(\vec{B}, \vec{F}) \approx \mathbf{f}^\top \mathbf{M}_f \mathbf{b} \tag{3.5}$$

where

$$\mathbf{M}_f = \operatorname{diag}\begin{pmatrix} \mathbf{A}_{f1}^\top \mathbf{v} \\ \mathbf{A}_{f2}^\top \mathbf{v} \\ \mathbf{A}_{f3}^\top \mathbf{v} \end{pmatrix}$$

Note that if we need to compute an inner product with weighting of material propery it is straight forward.

$$(\mu^{-1}\vec{B}, \vec{F}) \approx \mathbf{f}^\top \mathbf{M}_{\mu^{-1}f} \mathbf{b} \tag{3.6}$$

where

$$\mathbf{M}_{\mu^{-1}f} = \text{diag}\begin{pmatrix} \mathbf{A}_{f1}^\top(\mathbf{v} \odot \mu^{-1}) \\ \mathbf{A}_{f2}^\top(\mathbf{v} \odot \mu^{-1}) \\ \mathbf{A}_{f3}^\top(\mathbf{v} \odot \mu^{-1}) \end{pmatrix}$$

3.2.3 Edge variables inner product

It is straight forward to derive the discretization for the edge inner product in a similar way to the face inner products. The main difference that each cell in the mesh has 4 edges in each direction and therefore the averaging is done twice, yielding weights of $\frac{1}{4}$ on each edge. This yields the following approximation to the inner product of an edge function

$$(\vec{E}, \vec{W}) \approx \mathbf{w}^\top \mathbf{M}_f \mathbf{e} \tag{3.7}$$

where

$$\mathbf{M}_e = \text{diag}\begin{pmatrix} \mathbf{A}_{e1}^\top \mathbf{v} \\ \mathbf{A}_{e2}^\top \mathbf{v} \\ \mathbf{A}_{e3}^\top \mathbf{v} \end{pmatrix}$$

where the matrix $\mathbf{A}_{ej}$ averages the edges in the j-th direction into cell centers.

Similarly weighted inner products are easy to compute

$$(\sigma\vec{E}, \vec{W}) \approx \mathbf{e}^\top \mathbf{M}_{\sigma e} \mathbf{w} \tag{3.8}$$

where

$$\mathbf{M}_{\sigma e} = \text{diag}\begin{pmatrix} \mathbf{A}_{e1}^\top(\mathbf{v} \odot \sigma) \\ \mathbf{A}_{e2}^\top(\mathbf{v} \odot \sigma) \\ \mathbf{A}_{e3}^\top(\mathbf{v} \odot \sigma) \end{pmatrix}$$

3.3 Discretization on logically orthogonal grids

Logically orthogonal mesh can be thought of as a regular mesh that has been distorted. In 2D, the mesh is a collection of $(n_1 + 1) \times (n_2 + 1)$ points that define $n_1 \times n_2$ cells. Each point on the mesh has 4 connected neighbors. A 2D logically orthogonal mesh is plotted in Figure 3.2. In 3D, each point is connected to 6 other points.

The discretization on this mesh follows the same principles of orthogonal mesh and the main complication is the computation of the geometrical equivalents to the ones discussed on regular mesh. We now discuss each of the components at some depth. Before we do that we recall some basic vector and geometry identities and refer to Figure 3.3

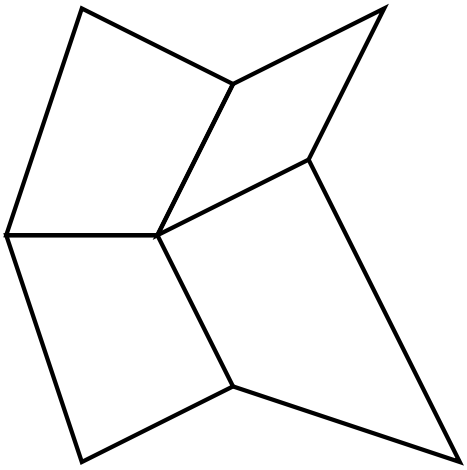

Figure 3.2. *Logically orthogonal mesh*

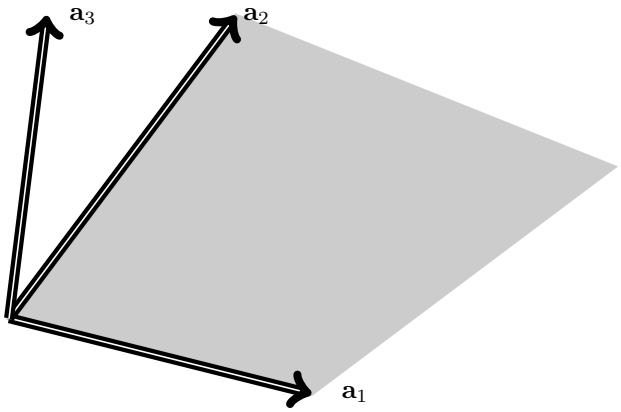

Figure 3.3. *The area of a parallelogram and the volume of a parallelepiped*

- Two vectors, $\mathbf{a}_1$ and $\mathbf{a}_2$ in 3D space generate a plane (as long as they are not parallel to each other). An orthogonal line to the plane, that is, the normal, can be found by the cross product

$$\mathbf{n} = \mathbf{a}_1 \times \mathbf{a}_2$$

- The parallelogram obtained by $\mathbf{a}_1$ and $\mathbf{a}_2$ has the area of

$$a = |\mathbf{a}_1 \times \mathbf{a}_2|$$

- Given three vectors $\mathbf{a}_1, \mathbf{a}_2$ and $\mathbf{a}_3$ in 3D space, the volume of the parallelepiped

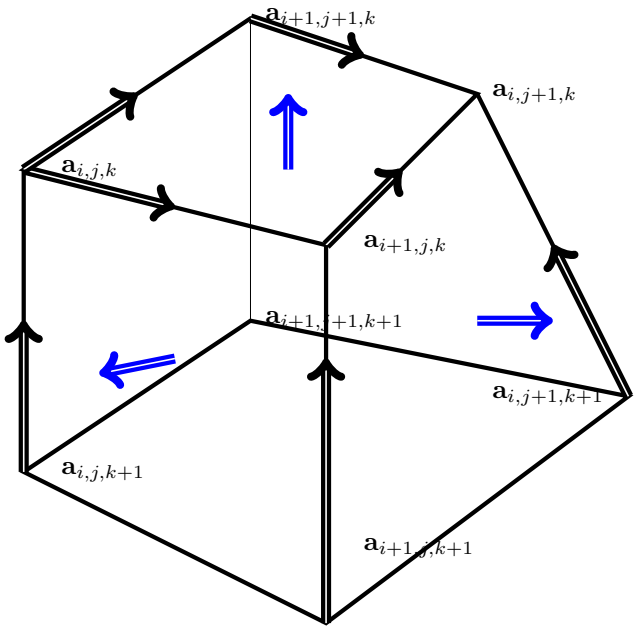

Figure 3.4. *A control volume*

that is generated by these vectors is

$$v = |(\mathbf{a}_1 \times \mathbf{a}_2) \cdot \mathbf{a}_3|$$

Using these simple geometrical formulas we can now discuss the discretization on logically orthogonal mesh. To this end, we assume that the mesh is defined by three 3D arrays $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ such that the nodes of each deformed cube on the mesh corresponds to the entries in these arrays. We note the (i, j, k) node by $\mathbf{a}_{i,j,k} = (\mathbf{X}_{i,j,k}, \mathbf{Y}_{i,j,k}, \mathbf{Z}_{i,j,k})$. The volume $v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}$ is located between the nodes

$$\{\mathbf{a}_{i,j,k}, \mathbf{a}_{i+1,j,k}, \mathbf{a}_{i,j+1,k}, \mathbf{a}_{i+1,j+1,k}\mathbf{a}_{i,j,k+1}, \mathbf{a}_{i+1,j,k+1}, \mathbf{a}_{i,j+1,k+1}, \mathbf{a}_{i+1,j+1,k+1}\}$$

Unlike orthogonal mesh, the discretization of the different components is a delicate matter. Just like in the case of orthogonal mesh, we can think of nodal variables, edge variables, face variables and cell-center variables.

Nodal Variables: Nodal variables are a discretization of a scalar field, say potential, onto the nodes of the grid that is, given a function φ and its grid function $\boldsymbol{\varphi}$ we have that $\boldsymbol{\varphi} = \{\varphi(\mathbf{a}_{i,j,k}), i = 1, \dots (n_1 + 1), j = 1, \dots (n_2 + 1), k = 1, \dots (n_3 + 1)\}$.

Edge Variables: To define edge variables we first define the edges. Each volume has 12 edges. The coordinates of the edges can be obtained by using the nodes of the mesh. We define the edge coordinate $\mathbf{a}_{i+\frac{1}{2},j,k}$ as

$$\mathbf{a}_{i+\frac{1}{2},j,k} = \mathbf{a}_{i+1,j,k} - \mathbf{a}_{i,j,k}$$

and the unit vector

$$\hat{\mathbf{a}}_{i+\frac{1}{2},j,k} = \frac{\mathbf{a}_{i+\frac{1}{2},j,k}}{|\mathbf{a}_{i+\frac{1}{2},j,k}|}.$$

Now, assume we discretize a field $\vec{E}$ on the edges of our non-orthogonal grid and let $\mathbf{e}$ be the grid function that represents $\vec{E}$. The field $\vec{E}$ is a vector field and therefore, its discretization $\mathbf{e}$ is a projection of $\vec{E}$ onto the discretization point in the direction of the edge. For example, the field $\mathbf{e}(\mathbf{a}_{i+\frac{1}{2},j,k})$ is defined as

$$\mathbf{e}(\mathbf{a}_{i+\frac{1}{2},j,k}) = \vec{E} \cdot \hat{\mathbf{a}}_{i+\frac{1}{2},j,k}.$$

that is, the field on the $i+\frac{1}{2}, j, k$ edge projected onto the edge direction. The vector field $\mathbf{e}$ is therefore the collection of the projection of fields onto the edges

$$\mathbf{e} = [\mathbf{e}_1^\top, \mathbf{e}_2^\top, \mathbf{e}_3^\top]^\top$$

where

$$\begin{aligned}
\mathbf{e}_1 &= \{\vec{E} \cdot \hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, , i = 1, \dots n_1, j = 1, \dots (n_2+1), k = 1, \dots (n_3+1)\} \\
\mathbf{e}_2 &= \{\vec{E} \cdot \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, , i = 1, \dots (n_1+1), j = 1, \dots n_2, k = 1, \dots (n_3+1)\} \\
\mathbf{e}_3 &= \{\vec{E} \cdot \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}}, , i = 1, \dots (n_1+1), j = 1, \dots (n_2+1), k = 1, \dots n_3\}
\end{aligned}$$

Face Variables: A more complicated issue is the discretization of face variables. The main problem here is that in general, there is no linear surface that goes through 4 points. Therefore, there is neither a unique point to define the face variable, nor a unique normal to the surface. To define the center of the face we use the average of the 4 nodes that define the face, that is, the face location $\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k}$ is defined as

$$\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}(\mathbf{a}_{i,j,k} + \mathbf{a}_{i+1,j,k} + \mathbf{a}_{i,j+1,k} + \mathbf{a}_{i+1,j+1,k})$$

The direction of the face is defined as an average direction of normals for each of the corners. First, we consider the 4 normalized edges that define the face

$$\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, \hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k}, \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k}$$

Each two edges generate a plane. Thus we consider four different planes, each with its own normal that represents the normal at the corner of each node. Averaging, these normals we obtain an average normal to the face

$$\begin{aligned}
\mathbf{n}_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}\Big(&\hat{\mathbf{a}}_{i+\frac{1}{2},j,k} \times \hat{\mathbf{a}}_{i,j+\frac{1}{2},k} + \hat{\mathbf{a}}_{i+\frac{1}{2},j,k} \times \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k} + \\
&\hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k} \times \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k} + \hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k} \times \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}\Big)
\end{aligned} \tag{3.9}$$

Similar to the electric field, if a magnetic flux, $\vec{B}$, is discretized on the face of the mesh then we imply that

$$\mathbf{b}(\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k}) = \vec{B} \cdot \mathbf{n}_{i+\frac{1}{2},j+\frac{1}{2},k}.$$

Cell centered variables: Finally, a cell-center variable $\boldsymbol{\varphi}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}$ is a discretization of scalar field located at the point

$$\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{8} \sum_{m,n,p=0}^{1} \mathbf{a}_{i+m,j+n,k+p}$$

3.3.1 Differential Operators

The discretization in this case follows the same lines as the one on orthogonal mesh and we use the approximations

$$\nabla \cdot \vec{B} = \frac{\int_{\partial V} \vec{B} \cdot \vec{n}\, dS}{\int_V dV} \approx \mathbf{V}^{-1}\mathbf{D}\mathbf{F}\mathbf{b} = \mathsf{DIV}\mathbf{b} \tag{3.10a}$$

$$(\nabla \times \vec{E}) \cdot \vec{n} = \frac{\int_{\partial S} \vec{E} \cdot d\ell}{\int_S dS} \approx \mathbf{F}^{-1}\mathbf{C}\mathbf{L}\mathbf{e} = \mathsf{CURL}\mathbf{e} \tag{3.10b}$$

where, just as in regular mesh, $\mathbf{D}$ and $\mathbf{C}$ are matrices that have entries of ± 1 and $\mathbf{F}$, $\mathbf{V}$ and $\mathbf{L}$ contain the approximate face area, volume and edge length of each face, cell and edge. Since the mesh topology is identical to the tensor mesh, the matrices $\mathbf{D}$ and $\mathbf{C}$ are identical to the ones obtained on a regular mesh. The main difference between logical orthogonal mesh to a tensor mesh and the main difficulty when working with logically orthogonal mesh is the computation of the face area, the volumes and edge lengths. We now discuss the computation of the edge length, face area and cell-volume in detail.

Computation of edge length

The computation of edge length is straight forward. Consider two points on a cube $\mathbf{a}_{i,j,k} = (\mathbf{X}_{i,j,k}, \mathbf{Y}_{i,j,k}, \mathbf{Z}_{i,j,k})$ and $\mathbf{a}_{i+1,j,k} = (\mathbf{X}_{i+1,j,k}, \mathbf{Y}_{i+1,j,k}, \mathbf{Z}_{i,j,k})$. The length of the edge between these two points can be computed as

$$\ell_{i+\frac{1}{2},j,k} = \|\mathbf{a}_{i+1,j,k} - \mathbf{a}_{i,j,k}\|.$$

Computation of face area

Similar to the discussion on face variables, The computation of the area of each face is **not** a straight forward calculation, this is because, given four points in space, they do not generally lie in the same plane.

One way to overcome the difficulty is to determine a unique plane by for example, seeking the one with the minimal curvature. A different approach that we

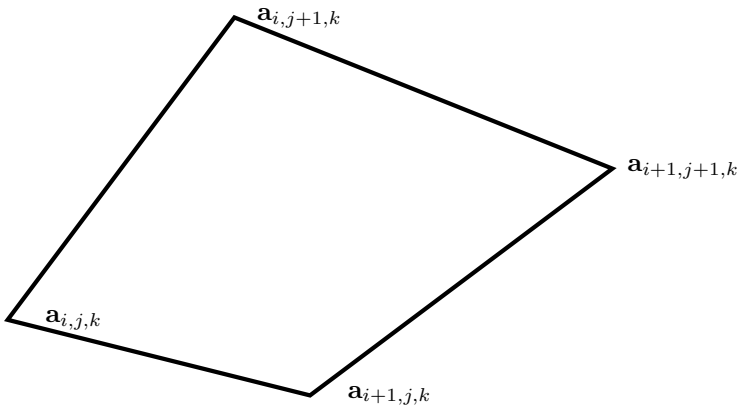

Figure 3.5. *A plane in 3D*

have used to define the face variables was to use an "average" plane to represent the face and its area. Concept, consider the 4 points in Figure 3.5. It is possible to define (at least) 4 different planes using these points. We define the plane $P_{i,j,k}$ by the points $\mathbf{a}_{i+1,j,k}, \mathbf{a}_{i,j,k}, \mathbf{a}_{i,j+1,k}$. Similarly, the plane $P_{i+1,j,k}$ is defined by the points $\mathbf{a}_{i,j,k}, \mathbf{a}_{i+1,j,k}, \mathbf{a}_{i+1,j+1,k}$. The plane $P_{i,j+1,k}$ is defined by $\mathbf{a}_{i,j,k}, \mathbf{a}_{i,j+,k}, \mathbf{a}_{i+1,j+1,k}$ and finally, the plane $P_{i+1,j+1,k}$ is defined by $\mathbf{a}_{i+1,j,k}, \mathbf{a}_{i+1,j+,k}, \mathbf{a}_{i,j+1,k}$. To compute the area of the face we average the four areas obtained by the four planes

$$f_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}\Big(|\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i+1,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j+1,k} \times \mathbf{a}_{i+1,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j+1,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}|\Big)$$

Computation of cell volume

Since cell faces are not uniquely defined the volume of each cell is not well defined either. Once again, it is possible to define it using some definitions of the surfaces however, it is also possible to compute it as the average of volumes. Indeed, we can consider every corner of the deformed cube as a corner of a parallelepiped that is defined by the three edges that emerge from that corner. Defining the volume around the (i, j, k) node as

$$v_{i,j,k} = |(\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}) \cdot \mathbf{a}_{i,j,k+\frac{1}{2}}|$$

we can approximate the volume as

$$V_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \approx \frac{1}{8} \sum_{\ell,n,m=0,1} v_{i+\ell,j+n,k+m}$$

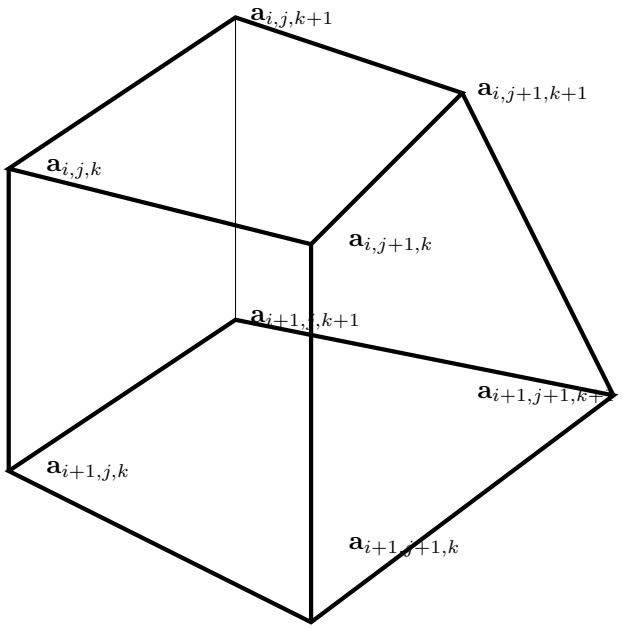

Figure 3.6. *A control volume*

3.3.2 Computation of inner products

The computational of inner products is the most involved part when considering the discretization on non-orthogonal grids. Before we go into details we review the general idea behind the computation of such a product. Consider the case that the vector fields $\vec{W}$ and $\vec{E}$ are given in non-orthogonal coordinates and we need to compute the inner product

$$(\vec{W}, \vec{E}) = \int_\Omega \vec{E} \cdot \vec{W} \; dv = \int_\Omega (\vec{E}_1 \vec{W}_1 + \vec{E}_2 \vec{W}_2 + \vec{E}_3 \vec{W}_3) \, dv$$

Clearly, if we are given orthogonal components of the fields we can use the formula above in a straight forward manor. If on the other hand the components are not orthogonal then we need to do an extra step. Consider that the components on the non-orthogonal coordinate system that is given by

$$\hat{\vec{E}}_1 = \mathbf{a}_1 \cdot \vec{E} \quad \hat{\vec{E}}_2 = \mathbf{a}_2 \cdot \vec{E} \quad \hat{\vec{E}}_3 = \mathbf{a}_3 \cdot \vec{E}$$

where $\mathbf{a}_i$ are direction vectors that transform the field from the orthogonal system to the non-orthogonal one. This can be written as

$$\begin{pmatrix} \hat{\vec{E}}_1 \\ \hat{\vec{E}}_2 \\ \hat{\vec{E}}_3 \end{pmatrix} = \begin{pmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ - & \mathbf{a}_3^\top & - \end{pmatrix} \begin{pmatrix} \vec{E}_1 \\ \vec{E}_2 \\ \vec{E}_3 \end{pmatrix} \quad \text{or} \quad \hat{\vec{E}} = \mathbf{A}\vec{E}$$

The above implies that if we are given $\hat{\vec{E}}$ we can obtain $\vec{E}$ by simply inverting $\mathbf{A}$ or

$$\vec{E} = \mathbf{A}^{-1}\hat{\vec{E}} \quad \text{and} \quad \vec{W} = \mathbf{A}^{-1}\hat{\vec{W}}$$

The inner product can therefore be written as

$$(\vec{E}, \vec{W}) = \int_\Omega \hat{\vec{W}}^\top \mathbf{A}^{-\top}\mathbf{A}^{-1}\hat{\vec{E}}\, dv$$

Equipped with this machinery we can now discuss the discretization of edge and face inner products.

The discretization of edge inner products

To discuss the discretization of the edge inner product we use the notation in Figure 3.7. Consider the (i, j, k) node and the three edges that emerge from this node, noted as $(i+\frac{1}{2}, j, k), (i, j+\frac{1}{2}, k), (i, j, k+\frac{1}{2})$. These 3 edges form a non-orthogonal coordinate system with the unit vectors $\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}}$. The discretization of the vectors fields $\vec{E}$ and $\vec{W}$ on these edges is such that

$$\mathbf{e}_{i+\frac{1}{2},j,k} = \vec{E}_{i+\frac{1}{2},j,k}\cdot\hat{\mathbf{a}}_{i+\frac{1}{2},j,k} \quad \mathbf{e}_{i,j+\frac{1}{2},k} = \vec{E}_{i,j+\frac{1}{2},k}\cdot\hat{\mathbf{a}}_{i,j+\frac{1}{2},k} \quad \mathbf{e}_{i+\frac{1}{2},j,k} = \vec{E}_{i+\frac{1}{2},j,k}\cdot\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}$$

We can approximate the fields in cartesian coordinates around the (i, j, k) nodes as

$$\begin{pmatrix}(\mathbf{e}_{i,j,k})_x\\(\mathbf{e}_{i,j,k})_y\\(\mathbf{e}_{i,j,k})_z\end{pmatrix} = \begin{pmatrix}- & \hat{\mathbf{a}}_{i+\frac{1}{2},j,k} & -\\- & \hat{\mathbf{a}}_{i,j+\frac{1}{2},k} & -\\- & \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}} & -\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{e}_{i+\frac{1}{2},j,k}\\\mathbf{e}_{i,j+\frac{1}{2},k}\\\mathbf{e}_{i,j,k+\frac{1}{2}}\end{pmatrix}$$

Similarly,

$$\begin{pmatrix}(\mathbf{w}_{i,j,k})_x\\(\mathbf{w}_{i,j,k})_y\\(\mathbf{w}_{i,j,k})_z\end{pmatrix} = \begin{pmatrix}- & \hat{\mathbf{a}}^\top_{i+\frac{1}{2},j,k} & -\\- & \hat{\mathbf{a}}^\top_{i,j+\frac{1}{2},k} & -\\- & \hat{\mathbf{a}}^\top_{i,j,k+\frac{1}{2}} & -\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{w}_{i+\frac{1}{2},j,k}\\\mathbf{w}_{i,j+\frac{1}{2},k}\\\mathbf{w}_{i,j,k+\frac{1}{2}}\end{pmatrix}$$

With some abuse of notation we set

$$\mathbf{e}^{i,j,k} = \begin{pmatrix}\mathbf{e}_{i+\frac{1}{2},j,k} & \mathbf{e}_{i,j+\frac{1}{2},k} & \mathbf{e}_{i,j,k+\frac{1}{2}}\end{pmatrix} \quad \mathbf{w}^{i,j,k} = \begin{pmatrix}\mathbf{w}_{i+\frac{1}{2},j,k} & \mathbf{w}_{i,j+\frac{1}{2},k} & \mathbf{w}_{i,j,k+\frac{1}{2}}\end{pmatrix}$$

and

$$\mathbf{A}^{i,j,k} = \begin{pmatrix}- & \hat{\mathbf{a}}^\top_{i+\frac{1}{2},j,k} & -\\- & \hat{\mathbf{a}}^\top_{i,j+\frac{1}{2},k} & -\\- & \hat{\mathbf{a}}^\top_{i,j,k+\frac{1}{2}} & -\end{pmatrix}$$

Using this notation we can approximate the dot product $\vec{E}\cdot\vec{W}$ around the (i, j, k) node as

$$(\vec{E}\cdot\vec{W})_{i,j,k} \approx (\mathbf{w}^{i,j,k})^\top(\mathbf{A}^{i,j,k})^{-\top}(\mathbf{A}^{i,j,k})^{-1}\mathbf{e}^{i,j,k}$$

Similar approximation can be obtained around every node in our deformed cube, thus we obtain 8 approximations to the dot product $\vec{E}\cdot\vec{W}$, each around a different

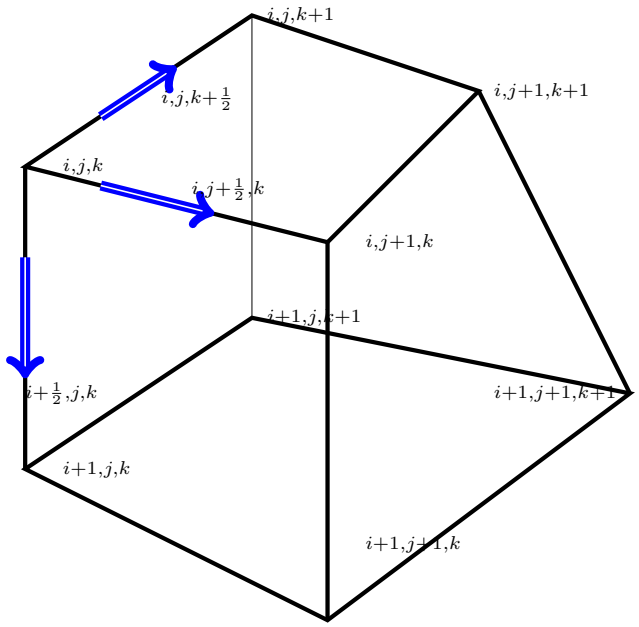

Figure 3.7. *Edges for the computation of an inner product*

node. To compute the inner product, that is, the integral over the volume of the dot product, we average the 8 corners and multiply by the volume

$$(\vec{E},\vec{W}) = \int_V \vec{E}\cdot\vec{W}\,dv \approx \tag{3.11}$$
$$\frac{v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{8} \sum_{n,m,\ell=0,1} (\mathbf{w}^{i+n,j+m,k+\ell})^\top (\mathbf{A}^{i+n,j+m,k+\ell})^{-\top} (\mathbf{A}^{i+n,j+m,k+\ell})^{-1} \mathbf{e}^{i+n,j+m,k+\ell}.$$

The discretization of face inner products

The face inner product is done in a very similar way to the edge inner product. Considering the normals $\hat{\mathbf{n}}_{i,j+\frac{1}{2},k+\frac{1}{2}}, \hat{\mathbf{n}}_{i+\frac{1}{2},j,k+\frac{1}{2}}, \hat{\mathbf{n}}_{i+\frac{1}{2},j+\frac{1}{2},k}$. We can associate them with the (i,j,k) node to approximate the dot product between a vector field $\vec{B}$ and $\vec{F}$ as

$$(\vec{B}\cdot\vec{F})_{i,j,k} \approx (\mathbf{b}^{i,j,k})^\top (\mathbf{N}^{i,j,k})^{-\top} (\mathbf{N}^{i,j,k})^{-1} \mathbf{f}^{i,j,k}$$

where

$$\mathbf{b}^{i,j,k} = \begin{pmatrix} \mathbf{b}_{i,j+\frac{1}{2},k+\frac{1}{2}} & \mathbf{b}_{i+\frac{1}{2},j,k+\frac{1}{2}} & \mathbf{b}_{i+\frac{1}{2},j+\frac{1}{2},k} \end{pmatrix} \quad \mathbf{f}^{i,j,k} = \begin{pmatrix} \mathbf{f}_{i,j+\frac{1}{2},k+\frac{1}{2}} & \mathbf{f}_{i+\frac{1}{2},j,k+\frac{1}{2}} & \mathbf{f}_{i+\frac{1}{2},j+\frac{1}{2},k} \end{pmatrix}$$

and

$$\mathbf{N}^{i,j,k} = \begin{pmatrix} - & \hat{\mathbf{n}}^{\top}_{i,j+\frac{1}{2},k+\frac{1}{2}} & - \\ - & \hat{\mathbf{n}}^{\top}_{i+\frac{1}{2},j,k+\frac{1}{2}} & - \\ - & \hat{\mathbf{n}}^{\top}_{i+\frac{1}{2},j+\frac{1}{2},k} & - \end{pmatrix}$$

and the inner product is approximated as

$$(\vec{F},\vec{B}) = \int_V \vec{F}\cdot\vec{B}\,dv \approx \tag{3.12}$$
$$\frac{v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{8} \sum_{n,m,\ell=0,1} (\mathbf{f}^{i+n,j+m,k+\ell})^{\top} (\mathbf{N}^{i+n,j+m,k+\ell})^{-\top} (\mathbf{N}^{i+n,j+m,k+\ell})^{-1} \mathbf{b}^{i+n,j+m,k+\ell}$$

Matrix representation of inner products

Similar to the case of regular mesh, it is possible to represent the inner products in matrix operations. Here, we do not explain this in detail and we only note that the inner product over the domain Ω can be approximated as

$$(\vec{E},\vec{W}) \approx \mathbf{e}^{\top}\mathbf{M}_e\mathbf{w}$$

where

$$\mathbf{M}_e = \frac{1}{8}\sum_i \mathbf{P}_i^{\top}\mathbf{T}_i^{\top}\,\mathrm{diag}(\mathbf{v},\mathbf{v},\mathbf{v})\mathbf{T}_i\mathbf{P}_i$$

where $\mathbf{P}_i$ is a matrix that takes the fields that are associated with the i^{th} corner and $\mathbf{T}_i$ contains the geometrical edge factors (the inverse of a block 3×3 matrices that transform from one coordinate system to the other).

Similarly

$$(\vec{B},\vec{F}) \approx \mathbf{b}^{\top}\mathbf{M}_f\mathbf{f}$$

where

$$\mathbf{M}_f = \frac{1}{8}\sum_i \mathbf{Q}_i^{\top}\mathbf{N}_i^{\top}\,\mathrm{diag}(\mathbf{v},\mathbf{v},\mathbf{v})\mathbf{N}_i\mathbf{Q}_i$$

and again $\mathbf{Q}_i$ extract the fields associated with the nodes while $\mathbf{N}_i$ contains the face geometry.

Extension of the discretization to anisotropy

One of the nicer consequences of this approach is the ease in which anisotropy can be handled. Assuming symmetric positive definite matrix functions, Σ and Ξ, we can compute inner products of the form

$$(\vec{E},\Sigma\vec{W}) \quad \text{and} \quad (\vec{F},\Xi\vec{B})$$

Assuming discretization, Σ and Ξ are tensors that are defined at each cell, that is, the tensor Σ and Ξ are discretized on the mesh as

$$\Sigma_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} = \begin{pmatrix} \sigma^{11}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{12}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{13}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ \sigma^{12}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{22}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{23}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ \sigma^{13}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{23}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{33}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \end{pmatrix}$$

It is straight forward to follow the arguments above and observe that the inner product in this case is changed into

$$\mathbf{M}_{e,\sigma} = \frac{1}{8}\sum_i \mathbf{P}_i^\top \mathbf{T}_i^\top \mathbf{V}_\sigma \mathbf{T}_i \mathbf{P}_i$$

where

$$\mathbf{V}_\sigma = \mathrm{diag}(\mathbf{v},\mathbf{v},\mathbf{v})^{\frac{1}{2}} \begin{pmatrix} \mathrm{diag}(\boldsymbol{\sigma}^{11}) & \mathrm{diag}(\boldsymbol{\sigma}^{12}) & \mathrm{diag}(\sigma^{13}) \\ \mathrm{diag}(\boldsymbol{\sigma}^{12}) & \mathrm{diag}(\boldsymbol{\sigma}^{22}) & \mathrm{diag}(\sigma^{23}) \\ \mathrm{diag}(\boldsymbol{\sigma}^{13}) & \mathrm{diag}(\boldsymbol{\sigma}^{23}) & \mathrm{diag}(\sigma^{33}) \end{pmatrix} \mathrm{diag}(\mathbf{v},\mathbf{v},\mathbf{v})^{\frac{1}{2}}$$

3.4 Programming the operators using Kronecker products

In order to program problems in 3D it is useful to consider the matrix representation of the operators. As we see next, for tensor meshes the matrices can be obtained by looking at the 1D case. We assume that we discretize the functions on a mesh with spacing vectors $\mathbf{h}_1, \mathbf{h}_2$ and $\mathbf{h}_3$ and mesh size $n_1 \times n_1 \times n_3$. That is, our mesh is not uniform but orthogonal. We leave the code for the non-orthogonal mesh as an exercise to the reader.

A key property that we heavily use in our derivation is as follows. Let $\mathbf{U}$ be a 2D array, the discretization of the function $u(x, y)$ on the nodes of our mesh. We also define the vector

$$\mathbf{u} = \mathrm{vec}(\mathbf{U})$$

That is, if $\mathbf{U}$ is an $(n_1 + 1) \times (n_2 + 1)$ matrix then $\mathbf{u}$ is $(n_1 + 1) \times (n_2 + 1)$ vector with entries

$$\mathbf{u}(k) = \mathbf{U}(i, j) \quad k = (i - 1) \times (n_1 + 1) + j$$

The following identity connects a linear operation on $\mathbf{U}$ to $\mathbf{u}$

$$\mathrm{vec}(\mathbf{A}\ \mathbf{U}\mathbf{B}^\top) = (\mathbf{B} \otimes \mathbf{A})\mathbf{u} \tag{3.13}$$

where $\otimes$ is the kronecker product of matrices

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ & \ddots & \\ a_{n1}\mathbf{B} & \dots & a_{nn}\mathbf{B} \end{pmatrix}$$

To see how this can be used in order to generate a matrix representation of say u_x consider the 2D version of u, that is the matrix $\mathbf{U}_{ij} = u(x_i, y_j)$. Let $\mathbf{D}_1$ be the 1D difference matrix in the x-direction

$$\mathbf{D}_1 = \begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}$$

Let us look at the product

$$\mathbf{D}_1\mathbf{U} = \frac{1}{h}\begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}\begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} & \dots & \\ \mathbf{U}_{21} & \ddots & & \\ & & \mathbf{U}_{n,n+1} & \mathbf{U}_{n+1,n+1} \end{pmatrix} =$$
$$\frac{1}{h}\begin{pmatrix} \mathbf{U}_{21}-\mathbf{U}_{11} & \mathbf{U}_{22}-\mathbf{U}_{12} & \dots & \\ \mathbf{U}_{31}-\mathbf{U}_{21} & \ddots & & \\ & & \mathbf{U}_{n+1,n}-\mathbf{U}_{n,n} & \mathbf{U}_{n+1,n+1}-\mathbf{U}_{n,n+1} \end{pmatrix}$$

If we organize the directions such that x, y is the same as the i, j directions on the matrix then we can write the difference matrix which corresponds to the (scaled) operator ∂_x in 2D as

$$\partial_x \approx \mathbf{D}_x = \mathbf{I}_2 \otimes \mathbf{D}_1$$

where D_1 is a 1D $(n_1)\times(n_1+1)$ finite difference matrix and $\mathbf{I}_2$ is a $(n_2+1)\times(n_2+1)$ identity matrix. It is easy to verify that the difference matrix in 2D for the y direction can be written as

$$\partial_y \approx \mathbf{D}_y = \mathbf{D}_2 \otimes \mathbf{I}_1$$

where $\mathbf{D}_2$ is a 1D $n_2\times(n_2+1)$ finite difference matrix and $\mathbf{I}_1$ is a $(n_1+1)\times(n_1+1)$ identity matrix.

3.4.1 Programming Linear Differential Operators

Combining derivatives in different directions we obtain a discrete representation for the gradient in 2D on a uniform mesh with mesh size 1

$$\nabla \approx \begin{pmatrix} \mathbf{I}_2 \otimes \mathbf{D}_1 \\ \mathbf{D}_2 \otimes \mathbf{I}_1 \end{pmatrix}$$

Thus, if we can program the 1D difference matrix and we have a kronecker product we can easily obtain a matrix representation in 2D.

The extension to 3D is rather straight forward

$$\nabla \approx \begin{pmatrix} \mathbf{I}_3 \otimes \mathbf{I}_2 \otimes \mathbf{D}_1 \\ \mathbf{I}_3 \otimes \mathbf{D}_2 \otimes \mathbf{I}_1 \\ \mathbf{D}_3 \otimes \mathbf{I}_2 \otimes \mathbf{I}_1 \end{pmatrix}$$

To code this we use the code for the 1D case (from the previous chapter). This is a single matlab line which we insert into the code as an inline function. Given the 1D code we can now quickly get the differential operators by combining the derivatives in the different directions and kronecker products

```
function[Div] = getFaceDivergenceMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

D1 = kron(speye(n3),kron(speye(n2),ddx(n1)));
D2 = kron(speye(n3),kron(ddx(n2),speye(n1)));
D3 = kron(ddx(n3),kron(speye(n2),speye(n1)));
% DIV from faces to cell-centers
Div = [D1 D2 D3];
```

Similarly, we can get the gradient on the nodes

```
function[Grad] = getNodalGradientMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

G1 = kron(speye(n3+1),kron(speye(n2+1),ddx(n1)));
G2 = kron(speye(n3+1),kron(ddx(n2),speye(n1+1)));
G3 = kron(ddx(n3),kron(speye(n2+1),speye(n1+1)));
% grad on the nodes
Grad = [G1; G2; G3];
```

And the more complex **curl** matrix

```
function[Curl] = getEdgeCurlMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

nfx = (n1+1)*n2*n3; nfy = n1*(n2+1)*n3; nfz = n1*n2*(n3+1);
nex = n1*(n2+1)*(n3+1); ney = (n1+1)*n2*(n3+1); nez = (n1+1)*(n2+1)*n3;


Dyz = kron(ddx(n3),kron(speye(n2),speye(n1+1)));
Dzy = kron(speye(n3),kron(ddx(n2),speye(n1+1)));

Dxz = kron(ddx(n3),kron(speye(n2+1),speye(n1)));
Dzx = kron(speye(n3),kron(speye(n2+1),ddx(n1)));

Dxy = kron(speye(n3+1),kron(ddx(n2),speye(n1)));
Dyx = kron(speye(n3+1),kron(speye(n2),ddx(n1)));


% curl on the edges
Curl = [sparse(nfx,nex)          Dyz            -Dzy; ...
        -Dxz             sparse(nfy,ney)         Dzx; ...
        Dxy                      -Dyx     sparse(nfz,nez)];
```

The above operators are on a uniform mesh with mesh-size 1. To use a general orthogonal mesh we need to generate the volumes, face areas and edges of the mesh. This can also be obtained from the vector of edge lengths $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ by using kronecker products as follows

```
function[V,F,L] = getMeshGeometry(h1,h2,h3)

n1 = length(h1); n2 = length(h2); n3 = length(h3);
V = kron(sdiag(h3),kron(sdiag(h2),sdiag(h1)));

F = sdiag([diag(kron(sdiag(h3),kron(sdiag(h2),speye(n1+1)))); ...
           diag(kron(sdiag(h3),kron(speye(n2+1),sdiag(h1)))); ...
           diag(kron(speye(n3+1),kron(sdiag(h2),sdiag(h1))))]);

L = sdiag([diag(kron(speye(n3+1),kron(speye(n2+1),sdiag(h1)))); ...
          diag(kron(speye(n3+1),kron(sdiag(h2),speye(n1+1)))); ...
          diag(kron(sdiag(h3),kron(speye(n2+1),speye(n1+1))))]);
```

setting

```
Div = V\(Div*F); Grad = L\Grad; Curl = F\(Curl*L);
```

The mimetic properties of the operators can be verified by the following lines of code

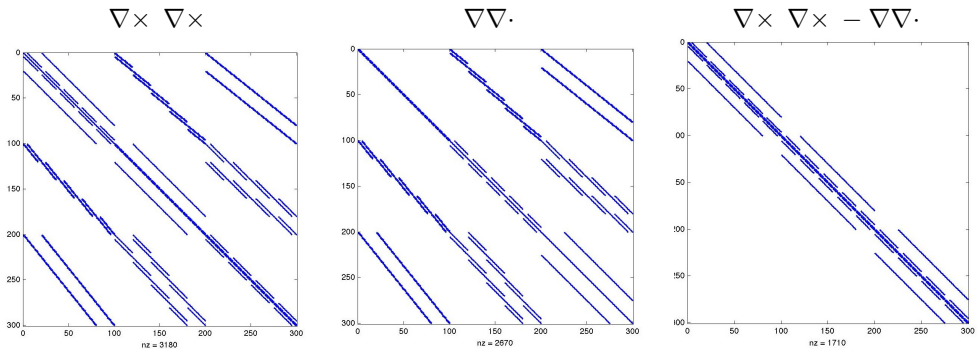

Figure 3.8. *The discrete* $\nabla\times\ \nabla\times\ -\nabla\nabla\cdot$

```
G = getNodalGradientMatrix(n1,n2,n3);
C = getEdgeCurlMatrix(n1,n2,n3);
figure(1); spy(C*G)
figure(2); spy(C'*C + G*G')
```

We can also check the mimetic properties of the formula

$$\nabla\times\ \nabla\times\ -\nabla\nabla\cdot = -\vec{\Delta}$$

This is demonstrated in Figure 3.8

Combining the operators we can now generate simple discretizations for differential equations with constant coefficients. For example, to discretize the Helmholtz equation $\Delta u + k^2 u$ we use the following code

```
G = getNodalGradientMatrix(n1,n2,n3);
H = G'*G + k^2*speye(size(G,2));
```

3.4.2 Programming the inner product matrices

The discretization is not complete without the discretization of the inner products. Similar to the 1D case we use a combination of the trapezoidal and midpoint methods for the calculation of the integral. To do that we need to generate interpolation matrices from faces/edges/nodes to cell centers. We use the 1D interpolation matrix from nodes to cell centers from the previous chapter as an inline function and using this 1D discretization we can build an interpolation from faces to cell centers

```
function[Afc] = getFaceToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

A1 = kron(speye(n3),kron(speye(n2),av(n1)));
A2 = kron(speye(n3),kron(av(n2),speye(n1)));
A3 = kron(av(n3),kron(speye(n2),speye(n1)));
% average from faces to cell-centers
Afc = [A1 A2 A3];
```

and similarly interpolating from the edges to cell center.

```
function[Aec] = getEdgeToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

A1 = kron(av(n3),kron(av(n2),speye(n1)));
A2 = kron(av(n3),kron(speye(n2),av(n1)));
A3 = kron(speye(n3),kron(av(n2),av(n1)));
% average from edge to cell-centers
Aec = [A1 A2 A3];
```

and from the nodes to cell center.

```
function[Anc] = getNodalToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

Anc = kron(av(n3),kron(av(n2),av(n1)));
```

Using the averaging we can now discretize the mass matrices in (3.6) and (3.8) directly as

```
% compute the cell volume
V = kron(sdiag(h3),kron(sdiag(h2),sdiag(h1)));
Afc = getFaceToCellCenterMatrix(n1,n2,n3);
Aec = getEdgeToCellCenterMatrix(n1,n2,n3);

Mf = sdiag(Afc'*(V*(1./mu(:))));
Me = sdiag(Aec'*(V*(sigma(:))));
```

The discretization of the mass matrices completes the building blocks needed to discretize Maxwell's equations (and many other equations as well). In the next chapter we discuss the discretization using the tools we build in this chapter.

In the above we have generated different computer codes for the discretization of the **div**, **grad**, **curl** and the averaging operators. It is possible to "lump" the codes together into one file that computes all the differential operators together. We have done so in the file getLinearOps.m. In the rest of the book we either all each operator separately or generate all the operators within a single call, depending on the context.

3.5 Exercises

1. The Stokes system from the pressure and velocity of fluid is

$$\vec{\Delta}\vec{u} + \nabla p = q \qquad \nabla \cdot \vec{u} = 0$$

   (a) Show that the system is equivalent to the system

$$\nabla \times \vec{w} + \nabla p = q \qquad \nabla \times \vec{u} = \vec{w} \qquad \nabla \cdot \vec{u} = 0$$

   where $\vec{w}$ is the vorticity

   (b) Assume that $\vec{u}$ is a field that is discretized on the edges. Where would $\vec{w}$ is naturally discretized? Where is p naturally discretize?

(c) Suggest a discretization to the system using the tools built in this chapter.

2. Extend the discretization on tensor mesh to include tensor conductivity. What is the order of the accuracy of your discretization.