Chapter 2

Discretization of some simple PDE's

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.

2.1 Finite volume discretization in 1D

We consider solving 1D PDE's in space-time and boundary value ODE's. The model problem we discuss is

$$\frac{d}{dx}\left(m\frac{du}{dx}\right) = q \tag{2.1}$$
$$u(0) = u_{\mathrm{BC}}; \quad u'(1) = 0$$

The conductivity u is assumed to be piecewise constant with possible jumps.

Before we discuss numerical methods for the solution of the problem it is important to discuss the properties of the solution. This is particularly important if m has jumps. Note that in this case using the product rule to open the brackets is wrong! Since m has jumps, $u'(x)$ must also has a jump. These jumps compensate for the jumps in m such that the product is smooth. Therefore $u''(x)$ is not well defined and one should avoid opening the brackets.

A different way to look at the same system is to write it in first order form

$$\frac{d}{dx}J = q \tag{2.2a}$$
$$m^{-1}J - \frac{du}{dx} = 0 \tag{2.2b}$$
$$u(0) = 1; \quad J(1) = 0$$

Where we introduced the "flux" $J(x)$. Note that J is smooth, that is, it has one derivative (at least). One may wonder why to divide the second equation in m. This will be explained next. The equation Eq. (2.2b) is also refers to as the constitutive relation.

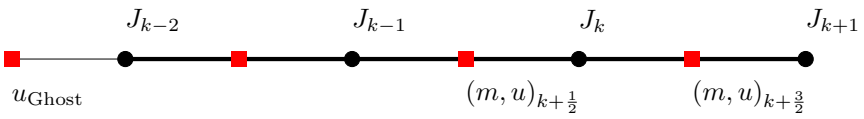


Figure 2.1. *1D grid for cell centered discretization*

Consider now a grid, in the interval $[0,1]$ made out of n intervals determined by the nodal points $0 = x_1 < x_2 < \ldots < x_{n+1} = 1$. We also note the the size of each cell as $h_{k+\frac{1}{2}} = x_{k+1} - x_k$. Consider also the staggered grid in cell centered $[x_{\frac{3}{2}} = \frac{1}{2}(x_1 + x_2), \ldots, x_{n+\frac{1}{2}} = \frac{1}{2}(x_n + x_{n+1})]$.

We have two options when discretizing the system. We can place u or J in cell-center or we can place u in the nodes. Here we use the cell-centered discretization. Nodal discretization can be derived in a similar manor.

Consider the case that J is discretized on the nodes as demonstrated in Figure 2.1. This is a natural discretization for the first equation Eq. (2.2a). Integrating the equation over a cell $[x_k, x_{k+1}]$ we have

$$\int_{x_k}^{x_{k+1}} J' dx = \int_{x_k}^{x_{k+1}} q$$

The left hand side can be integrated analytically. For the right hand side we use the midpoint method

$$J_{k+1} - J_k = h_{k+\frac{1}{2}} q_{k+\frac{1}{2}}. \tag{2.3}$$

Next, we need to integrate the constitutive relation equation

$$m^{-1} J = \frac{du}{dx}.$$

It is important to see that if we place J on the nodes then u is naturally discretized in cell-centers. We can integrate the right hand side analytically obtaining

$$\int_{x_{k-\frac{1}{2}}}^{x_{k+\frac{1}{2}}} m^{-1} J \, dx = u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}}.$$

To integrate the left hand side, one requires to be a bit more careful. If u is a smooth function then one can simply use the midpoint method obtaining the approximation

$$\frac{1}{2}(h_{k+\frac{1}{2}} + h_{k-\frac{1}{2}}) m_k^{-1} J_k + \mathcal{O}(h) = u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}}.$$

Note that if the grid is not uniform then we obtain only first order since J is not exactly in the middle of the cell.

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

Assume next that m is discontinuous but that it is piecewise smooth, that is in any interval $[x_k, x_{k+1}]$ m is smooth but it may have jumps between cells. In this case the midpoint method breaks and has to resort to a different approach. One such approach is to use a one sided approximation to the integral setting

$$\int_{x_{k-\frac{1}{2}}}^{x_{k+\frac{1}{2}}} m^{-1} J \, dx = \int_{x_{k-\frac{1}{2}}}^{x_k} m^{-1} J \, dx + \int_{x_k}^{x_{k+\frac{1}{2}}} m^{-1} J \, dx$$

Each of the integrals involves smooth functions and thus can be approximated by

$$\int_{x_{k-\frac{1}{2}}}^{x_k} m^{-1} J \, dx = \frac{1}{2} m^{-1}_{k-\frac{1}{2}} J_k h_{k-\frac{1}{2}} + \mathcal{O}(h)$$
$$\int_{x_k}^{x_{k+\frac{1}{2}}} m^{-1} J \, dx = \frac{1}{2} m^{-1}_{k+\frac{1}{2}} J_k h_{k+\frac{1}{2}} + \mathcal{O}(h)$$

Putting it all together we obtain that

$$\int_{x_{k-\frac{1}{2}}}^{x_{k+\frac{1}{2}}} m^{-1} J \, dx = \frac{1}{2}(m^{-1}_{k-\frac{1}{2}} h_{k-\frac{1}{2}} + m^{-1}_{k+\frac{1}{2}} h_{k+\frac{1}{2}}) J_k + \mathcal{O}(h) \tag{2.4}$$

Combining Eq. (2.3) and Eq. (2.4) and rearranging terms we obtain

$$\frac{J_{k+1} - J_k}{h_{k+\frac{1}{2}}} = q_{k+\frac{1}{2}} \tag{2.5a}$$
$$\frac{1}{2}(m^{-1}_{k-\frac{1}{2}} h_{k-\frac{1}{2}} + m^{-1}_{k+\frac{1}{2}} h_{k+\frac{1}{2}}) J_k = u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}} \tag{2.5b}$$

It is possible to further eliminate J from the system and obtain a system for u alone

$$h^{-1}_{k+\frac{1}{2}} \left\{ \left(\frac{1}{2}(m^{-1}_{k+\frac{3}{2}} h_{k+\frac{3}{2}} + m^{-1}_{k+\frac{1}{2}} h_{k+\frac{1}{2}}) \right)^{-1} (u_{k+\frac{3}{2}} - u_{k+\frac{1}{2}}) - \right.$$
$$\left. \left(\frac{1}{2}(m^{-1}_{k-\frac{1}{2}} h_{k-\frac{1}{2}} + m^{-1}_{k+\frac{1}{2}} h_{k+\frac{1}{2}}) \right)^{-1} (u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}}) \right\} = q_{k+\frac{1}{2}}$$

The discretization is simplified when we have a uniform mesh $h = h_{\frac{3}{2}} = \ldots = h_{n+\frac{1}{2}}$. In this case we have that

$$\frac{J_{k+1} - J_k}{h} = q_{k+\frac{1}{2}} \tag{2.6a}$$
$$\frac{1}{2}(m^{-1}_{k-\frac{1}{2}} + m^{-1}_{k+\frac{1}{2}}) J_k = \frac{u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}}}{h}. \tag{2.6b}$$

It is then common to define the harmonic average of m

$$m_k = (m_{k-\frac{1}{2}}^{-1} + m_{k+\frac{1}{2}}^{-1})^{-1}$$

and to rewrite the equation for u as

$$h^{-2}\left(m_{k+1}(u_{k+\frac{3}{2}} - u_{k+\frac{1}{2}}) \; - m_k(u_{k+\frac{1}{2}} - u_{k-\frac{1}{2}})\right) = q_{k+\frac{1}{2}}$$

which has the same form as for smooth m.

This is the time to pause and discuss the proposed discretization so far. First, it is important to realize that given smooth coefficients and uniform grid the discretization is second order accurate for **both** m and J. It is possible to show that in fact our discretization is second order even for non uniform mesh as long as the mesh is smooth. The discretization is reduced to first order accuracy when discontinuous coefficients are present. Obtaining second order discretization for the case of discontinuous coefficients is challenging and requires more accurate integration techniques for equation Eq. (2.2b). Such integration techniques lead to a wider stencil for J and thus it is not easy to obtain a system for u alone.

To complete the discussion we need to consider the implementation of boundary conditions. Since J is on the boundary a boundary condition on J is straight forward to implement. A boundary condition on u is slightly more difficult to implement. To do that we use a ghost point *outside* of our mesh. Then, boundary condition on u reads

$$\frac{1}{2}(u_{\text{Ghost}} + u_{\frac{3}{2}}) = u_{\text{BC}}$$

and therefore

$$u_{\text{Ghost}} = 2u_{\text{BC}} - u_{\frac{3}{2}}.$$

Using this equation we see that the only equation that needs modification is the equation for the first cell in Eq. (2.6b) which is rewritten as

$$\frac{1}{4}(m_{\frac{3}{2}}^{-1} + m_{\frac{1}{2}}^{-1})\, J_1 = \frac{u_{\frac{3}{2}} - u_{\text{BC}}}{h}.$$

The main difficulty when using the formula is that $m_{\frac{1}{2}}$ is outside of the grid and therefore is unknown. There are two options. In some cases one has an analytic expression for m and therefore m can be estimated directly. In more complex cases second order extrapolation is need to evaluate $m_{\frac{1}{2}}$. We note that if $m_{\frac{1}{2}}$ is extrapolated using a lower order then some loss of accuracy at the boundary could be observed.

Some difficulties arise when working with a purely Neumann problem. Recall that in this case the solution is not well defined. This is because any solution of the form $u + \text{const}$ solves the ODE. The discretization of the purely Neumann problem leads to a difference matrix where the vector $e = (1, 1, \ldots, 1)^\top$ is in its null

space. This implies that the linear system has one zero eigenvalue and therefore it is singular.

First, note that in order to have a solution at all, the right hand side must not have a part in the null space. For the continuous problem this implies that

$$\int_0^1 q(x)\,dx = 0.$$

which translates to

$$e^\top q = 0.$$

There are a number of ways to obfuscate the problem. Since u is defined up to a constant we can set the constant arbitrarily. For example, we can define $u_1 = 0$. Another more common approach is to define the constant by demanding that

$$\int_0^1 u(x)\,dx = 0$$

this leads to the following constraint

$$h\sum_i u_i = 0$$

that is added to the system.

2.2 1D in practice

We now give our attention to the actual coding of the 1D problem. To do that it is useful to think about the linear operations as matrices and the fields as vectors, using sparse linear algebra. Notice that we have the following matrices to be generated

- A difference matrix from cells to nodes, D_c^n : cell centers $\rightarrow$ nodes
- A difference matrix from nodes to cells, D_n^c : nodes $\rightarrow$ cell centers
- An averaging matrix from cells to nodes, A_n^c : cell center $\rightarrow$ nodes

We derive the operators for the case of uniform grid and the extension to nonuniform mesh is straight forward.

To start, the following code generates the 1D difference from nodes to cell centers in the interval $[0, 1]$ assuming no boundary conditions on the nodes.

```
e = ones(n+1,1); h = 1/n;
Dn2c = 1/h*spdiags([-e,e],[0,1],n,n+1);
```

Similarly, it is easy to generate a finite difference matrix from cell-centers to nodes

```
Dc2n = 1/h*spdiags([-e,e],[-1,0],n+1,n);
```

Note that as long as we use "standard" boundary conditions we have that

$$D_c^n = -(D_n^c)^\top.$$

This fits the inner product for two functions that vanish at the boundary

$$\int_0^1 u(x)v'(x)\,dx = -\int_0^1 u(x)'v(x)\,dx$$

The averaging matrix from cell-centers to nodes has a very similar structure. Using this matrix we can generate a diagonal matrix of $m(\cdot)$ for the cell centered discretization of m. We first compute the harmonic average of m and then invert it to obtain the matrix that corresponds to the operator $m(\cdot)$.

```
Ac2n = 1/2*spdiags([e,e],[-1,0],n+1,n);
S        = sdiag(1./(Ac2n*(1./u)));
```

Note that for this matrix we use nearest neighbor extrapolation for m on the boundary and therefore only $\mathcal{O}(h)$ can be achieved using this discretization.

The right hand side is composed of two parts. First, there is the integrals of q over the cells. Second, there are contributions from the boundary conditions. To obtain the contributions from the boundaries we define a matrix B that multiplies the boundary data and generates a vector of the appropriate size. For the 1D case the matrix is of size $(n+1)\times 2$ and it can be written as

```
B = sparse(n+1,2);
B(1,1) = 1/h; B(end,end) = -1/h;
```

Multiplying the matrix with the boundary conditions generates the appropriate right hand side.

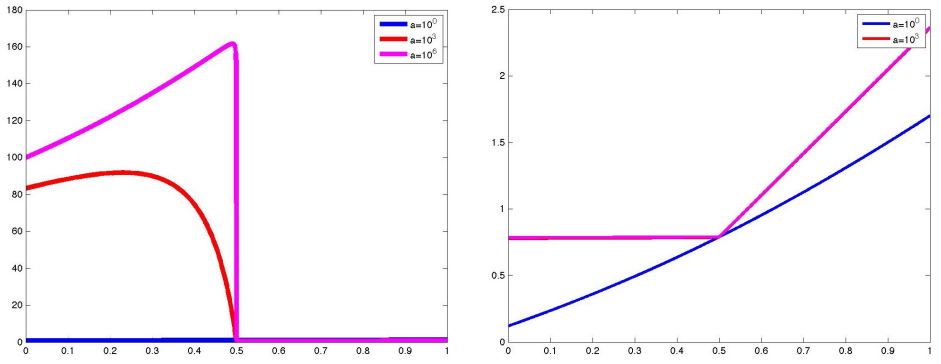
Putting it all together we obtain the following linear system for u and J

$$\begin{pmatrix} \operatorname{diag}(A_n^c m^{-1})) & D_n^c \\ D_c^n & 0 \end{pmatrix} \begin{pmatrix} J \\ u \end{pmatrix} = \begin{pmatrix} Bu_{\text{bc}} \\ q \end{pmatrix}$$

where for a vector s, we have that $s^{-1} = [s_1^{-1}, \ldots, s_n^{-1}]^\top$. This system is referred as a saddle point system that would be discussed later. For now we note that since the $(1,1)$ block is invertible and diagonal it is easy to eliminate J and obtain a system for y alone

$$D_n^c \operatorname{diag}(A_n^c m^{-1}))^{-1} D_n^c u = q + D_n^c \operatorname{diag}(A_n^c m^{-1}))^{-1} Bu_{\text{bc}}$$

Note that assuming Derichlet boundary conditions, the system is symmetric positive definite (SPD). For the Neumann boundary conditions the system is only positive semi-definite (PSD). The system is tridiagonal and can be easily solved and u to be recovered.



$m(x)$ $u(x)$

Figure 2.2. *The conductivity and field for the 1D test case*

2.3 Testing the code

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 appropriately. In many cases, developing a "good enough" test can be as complicated as writing the code, however, testing in necessary if we want to use the code and make some conclusions.

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.

Our test is composed of functions in $[0, 1]$. Choosing a smooth function for m,

$$m(x) = \frac{\exp(x)}{\arctan(a(x - \frac{1}{2})) + b}.$$

This choice of m allows us to test different scenarios. For large a, say $a = 10^9$, m is practically discontinuous while for small a, say, $a = 1$, m is very smooth. We would like to generate a problem with the characteristics and complexities of a realistic problem. Choosing

$$u'(x) = \arctan(a(x - \frac{1}{2})) + b$$

implies that $u'(x)$ also has a jump for large a and that mu' is smooth. Using Maple and integrating $u'(x)$ we obtain

$$u(x) = \arctan(ax - \frac{1}{2}a)x - \frac{1}{2}\arctan(ax - \frac{1}{2}a) - \frac{1}{2a}\log((ax - \frac{1}{2}a)^2 + 1) + bx.$$

We plot $u(x)$ and $m(x)$ in Figure 2.2.

Note that $u(x)$ has a "kink" at $\frac{1}{2}$ for large a. We now generate and solve the system on a sequence of grids ranging from 8 cells to 512 cells. We generate the discrete operators and compare three quantities. First, we compare the truncation errors that is, we input the exact solution (evaluated at the appropriate points) into the equations and recored the errors. Second, we solve the system for the fields and compare the analytic solution to the computed solution.

The code is as follows

```
kk = 1;
for n = [8,16,32,64,128,256,512]

    h = 1/n;
    % nodal grid
    tN = linspace(0,1,n+1); tN = tN(:);
    % cell centered grid
    tC = tN(1:end-1) + diff(tN)/2; tC = tC(:);
    % Boundary points
    tB = [tN(1); tN(end)];

    % define the fields and fluxes
    a = 1e0; b = pi/2+0.01;
    u     = @(t)(atan(a*t - 1/2*a).*t - 1/2*atan(a*t - 1/2*a) ...
              -1/(2*a)*log((a*t - 1/2*a).^2  + 1) + b*t);
    up    = @(t)(atan(a*(t-0.5)) + b);
    m = @(t)(1./up(t).*exp(t));
    J     = @(t)(up(t).*m(t));
    rhs   = @(t)(exp(t));
```

```
    % compute operators
    e = ones(n+1,1);
    Dn2c = 1/h*spdiags([-e,e],[0,1],n,n+1);
    Dc2n = 1/h*spdiags([-e,e],[-1,0],n+1,n);
    Ac2n = 1/2*spdiags([e,e],[-1,0],n+1,n);
    s    = Ac2n*(1./sigma(tC));

    % correct for conductivity outside of the grid
    % (for second order at the boundary)
    s(1)   = (1/m(h/2) + 1/m(-h/2))/4;
    s(end) = (1/m(1-h/2) + 1/m(1+h/2))/4;
    S      = sdiag(1./s);

    % Boundary condition matrix
    B = sparse(n+1,2); B(1,1) = 1/h; B(end,end) = -1/h;

    % check truncation error
    r1 = Dc2n*u(tC) - S*J(tN) - B*u(tB);
    r2 =  Dn2c * J(tN) - rhs(tC);

    % now solve and compare solution error
    A = Dn2c*S*Dc2n;
    b = rhs(tC) + Dn2c*S*B*u(tB);

     uN = A\b; uA = u(tC);
    r3 = uA-uN;

    % print some info
    rho(kk,1:4) = [h,norm(r1,'inf'),norm(r2,'inf'),norm(r3,'inf')]; kk = kk+1;
    fprintf('%3.2e     %3.2e     %3.2e     %3.2e\n',...
                   h,norm(r1,'inf'),norm(r2,'inf'),norm(r3,'inf'));
end

% check convergence rate
fprintf('\n\n\n Convergence summary\n\n');
fprintf('h           |r1|_inf    |r2|_inf    |error|_inf\n\n');

fprintf('%3.2e    %3.2e    %3.2e    %3.2e\n',(rho(1:end-1,:)./rho(2:end,:))')
```

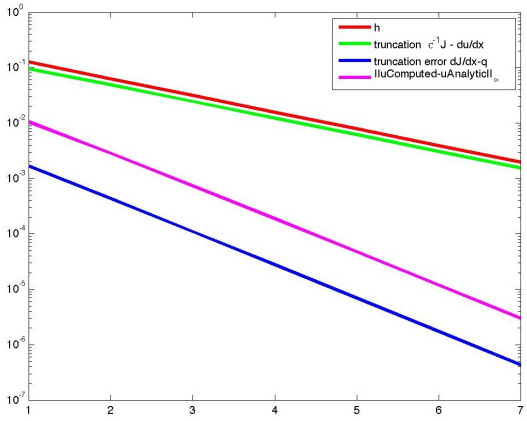


Figure 2.3. *Convergence results for the 1D case*

Convergence curves for the problem are plotted in Figure 2.3. For this example we have chosen $a = 10^9$ thus we recover the solution for a discontinuous conductivity. Note that just as the theory predicts, the truncation error for the discretization of the equation $m^{-1}J - u'$ is linear (behaves as h while the convergence for the equations $J' = q$ is of order h^2. A more surprising observation is that although the first equation is only $\mathcal{O}(h)$ accurate (at one point), the overall solution is actually $\mathcal{O}(h^2)$. Thus, the fact that we have an inaccuracy over a small manifold does not heart the overall accuracy.

To summarize the testing stage of code writing we stress that one cannot trust the code if it did not go through robust testing that shows that the code yields the theoretical accuracy. Building an appropriate test can be rather complicated (it can take longer than programming the original code!) but it is mandatory. Unfortunately, there is much numerical code that is not thoroughly tested. Far reaching conclusions such as global warming, nuclear storage and more have been done in the past based on "gently" tested codes. One should be always a bit skeptic about the code and continue to evaluate it with different use and applications.

2.4 Finite Volume Discretization in 2 and 3D

We now quickly extend the discretization to 3 and 3D. The equivalent equation in 2 and 3D to the equation we have analyzed in 1D is

$$\nabla \cdot m\nabla u = q \tag{2.7}$$

or in first order form

$$\nabla \cdot \vec{J} = q \tag{2.8a}$$

$$m^{-1}\vec{J} - \nabla u = 0 \tag{2.8b}$$

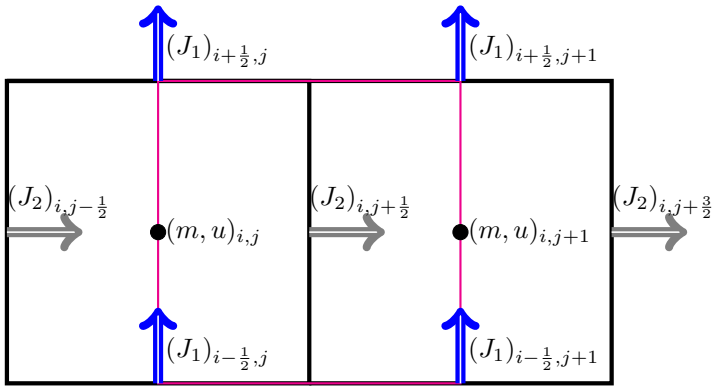


Figure 2.4. *Staggered grid in 2D*

Note that in this case the flux $\vec{J}$, is a vector, $\vec{J} = [J_1, \ldots, J_d]$ where $d = 2, 3$ is the dimension of the problem. Equation Eq. (2.8) is referred to as the flux-balance equation while Eq. (2.8b) is referred to as the constitutive equation. Boundary conditions are either on u, that is

$$u_{\partial\Omega} = u_{\text{BC}}$$

or on $\vec{J}$, that is

$$(\vec{J} \cdot \vec{n})_{\partial\Omega} = J_{\text{BC}}.$$

Before we discretize the equations we note a few properties of the continuous variables. First, u is differentiable once in each direction. If m is discontinuous then u is not differentiable twice in each direction and derivatives such as $u_{x_1x_1}$ and $u_{x_2x_2}$ may not exist. On the other hand, since does exist we observe that ∇u may not be smooth in the normal direction but it is smooth in the tangential direction. Thus we expect u to have a "kink" normal to an interface but be smooth otherwise.

Second, the vector field $\vec{J}$ is smooth in the normal direction, that is J_1 is differentiable in the x_1 direction and J_2 is differentiable in the x_2 direction. On the other hand, $\vec{J}$ may not be differentiable in the tangential direction, that is J_1 may not be differentiable in the x_2 direction and J_2 may not be differentiable in the x_1 direction.

We derive the discretization in 2D and as we see next, the extension to 3D is straight forward. Consider the cell-center discretization of u on a uniform grid of size h. The question which rise is, where should we discretize $\vec{J}$? It is straight forward to see that a natural discretization for $\vec{J}$ is on cell faces. This is demonstrated in Figure 2.4. Thus, we have introduced a **staggered grid** for $\vec{J}$.

Just as in 1D there are two equations we integrate upon. We start with the flux-balance equation. We integrate the equation in a box that its center is

$[(x_1)_i, (x_2)_j]$

$$\int_{\Omega_{ij}} \nabla \cdot \vec{J}\, dx_1\, dx_2 = \int_{\Omega_{ij}} q\, dx_1\, dx_2.$$

Integrating the right hand side we obtain

$$\int_{\Omega_{ij}} q\, dx_1\, dx_2 = q_{ij}h^2 + \mathcal{O}(h^2).$$

Next, we use the divergence theorem to integrate the left hand side

$$\int_{\Omega_{ij}} \nabla \cdot \vec{J}\, dx_1\, dx_2 = \int_{\partial\Omega_{ij}} \vec{J} \cdot \vec{n}\, ds = \tag{2.9}$$
$$h((J_1)_{i+\frac{1}{2},j} - (J_1)_{i-\frac{1}{2},j}) + h((J_2)_{i,j+\frac{1}{2}} - (J_2)_{i,j-\frac{1}{2}}) + \mathcal{O}(h^2)$$

We now need to integrate the constitutive relation equation. This equation is actually two equations in 2D

$$u_{x_1} = m^{-1}J_1 \quad u_{x_2} = m^{-1}J_2.$$

We integrate the first equation over a box around J_1 (see Figure 2.4) and the second equations over a box around J_2. It is crucial to recall that u_{x_1} may be discontinuous at point $[(x_1)_i, (x_2)_{j+1}$ and therefore, the equation does not mean much pointwise. This is *exactly* the same issue we had in 1D and we resolve it in the same way.

$$\int_{(x_2)_{i-\frac{1}{2}}}^{(x_2)_{i+\frac{1}{2}}} \int_{(x_1)_j}^{(x_1)_{j+1}} u_{x_1}\, dx_1\, dx_2 = \int_{(x_2)_{i-\frac{1}{2}}}^{(x_2)_{i+\frac{1}{2}}} \int_{(x_1)_j}^{(x_1)_{j+1}} m^{-1}J_1\, dx_1\, dx_2$$

Integrating the lest hand side we use exact integration in x_1 and the midpoint method in x_2 to obtain

$$\int_{(x_2)_{i-\frac{1}{2}}}^{(x_2)_{i+\frac{1}{2}}} \int_{(x_1)_j}^{(x_1)_{j+1}} u_{x_1}\, dx_1\, dx_2 = h(u_{i+ij} - u_{ij}) + \mathcal{O}(h^2)$$

Integrating the left hand side over x_1 is done in the same way as in the previous section in 1D and we use the midpoint methods in the x_2 direction to obtain

$$\int_{(x_2)_{i-\frac{1}{2}}}^{(x_2)_{i+\frac{1}{2}}} \int_{(x_1)_j}^{(x_1)_{j+1}} m^{-1}J_1\, dx_1\, dx_2 = \frac{1}{2}(J_1)_{i,j+\frac{1}{2}}(m_{ij}^{-1} + m_{ij+1}^{-1}) + \mathcal{O}(h)$$

Note again that the $\mathcal{O}(h)$ assumes a discontinuous m for a smooth m $\mathcal{O}(h^2)$ is obtained.

Defining

$$m_{ij+\frac{1}{2}} = 2\left((m_{ij}^{-1} + m_{ij+1}^{-1})\right)^{-1}$$
$$m_{i+\frac{1}{2},j} = 2\left((m_{ij}^{-1} + m_{i+j}^{-1})\right)^{-1}$$

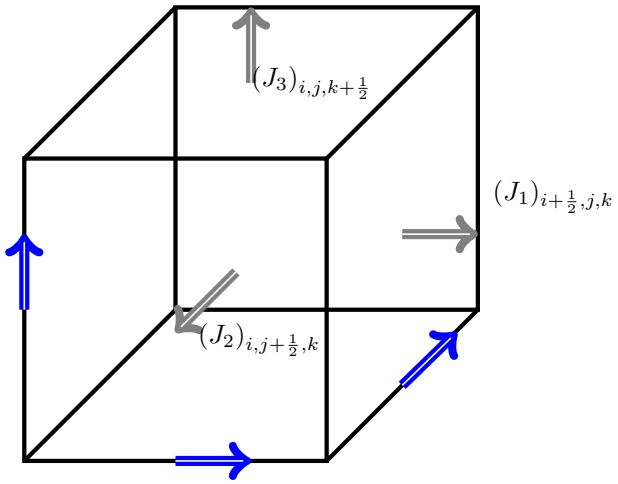


Figure 2.5. *Staggered discretization in 3D*

We can eliminate $\vec{J}$ from the equations and obtain an equation for u alone

$$\frac{m_{ij+\frac{1}{2}}(u_{ij+1}-y_{ij})-m_{ij-\frac{1}{2}}(u_{ij}-u_{ij-1})+m_{i+\frac{1}{2},j}(u_{i+1j}-y_{ij})-m_{i-\frac{1}{2},j}(u_{ij}-u_{i-1,j})}{h^2}=q_{ij} \tag{2.10}$$

Boundary conditions are handled exactly as per in 1D by adding ghost points. Since the discretization is derived by combining 1D discretizations in x_1 and x_2 boundaries in x_1 and x_2 are dealt in the same way we dealt with 1D problems.

2.5 Problems in 3D

We now extend the discussion above into more general problems and to 3D. In 3D there are three main operators of interest that can be used to discretize most linear PDE's; the gradient, the divergence and the curl. We now discuss the discretization of these operators in 3D.

The extension of the staggered discretization proposed in 2D to 3D is slightly more involved. In principle, we have 4 types of variables. First, for scalar fields we have cell-centered and nodal discretization. This is similar to the 2D case. A slight departure from the 2D case is the separation between vector fields that are face and edge variables which "live" in different location on the grid. While face variables are assumed to be normal to the cell face edge variables are aligned with the cell edge. This is demonstrated in Figure 2.5. To derive differential operators we look at particular discretizations and use the appropriate integral rule.

To derive a discretization of the divergence we look at cell face vectors and using the divergence theorem obtain that

$$\int_{\Omega_{ijk}} \nabla \cdot \vec{J}\, dx\, dy\, dz = \int_{\partial\Omega_{ijk}} \vec{J} \cdot \vec{n}\, ds = \tag{2.11}$$

$$h^2 \left((J_1)_{i+\frac{1}{2},j,k} - (J_1)_{i-\frac{1}{2},j,k} + (J_2)_{i,j+\frac{1}{2},k} - (J_2)_{i,j-\frac{1}{2},k} + (J_3)_{i,j,k+\frac{1}{2}} - (J_3)_{i,j,k-\frac{1}{2}} \right) + \mathcal{O}(h^2)$$

To discretize the gradient we consider first a cell center discretization of a scalar function y. A short central difference which average on cell-faces yield

$$\begin{aligned}
(u_{x_1})_{i+\frac{1}{2},j,k} &= \frac{u_{i+1,j,k} - u_{i,j,k}}{h} + \mathcal{O}(h^2) \\
(u_{x_2})_{i,j+\frac{1}{2},k} &= \frac{u_{i,j+1,k} - u_{i,j,k}}{h} + \mathcal{O}(h^2) \\
(u_{x_3})_{i,j,k+\frac{1}{2}} &= \frac{u_{i,j,k+1} - u_{i,j,k}}{h} + \mathcal{O}(h^2)
\end{aligned}$$

Material averaging mirrors the one we had in 2D and can be summarized as follows

- We consider material averaging through integration.
- Using the same arguments as in 2D, harmonic averaging is needed for the first order Poisson equation if u is in cell centers and $\vec{J}$ is on the faces.

2.6 Matrix representation in 2 and 3D

In order to program problems in 2 and 3D it is useful to consider the matrix representation of the operators. As we see next, the matrices can be obtained by looking at the 1D case. For the simplicity of the discussion we assume that we discretize the functions on a uniform mesh with equal spacing and equal number of points in all directions.

A key property that we heavily use in our derivation is as follows. Let Y be a 2D array, the discretization of the function $u(x_1, x_2)$. We also define (with some abuse of notation) the vector

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

That is, if U is an $N \times M$ matrix then

$$u(k) = U(i,j) \quad k = (M-1) \times i + j$$

Then

$$\mathrm{vec}(AUB^\top) = (B \otimes A)u \tag{2.12}$$

where $\otimes$ is the kronecker product of matrices

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

To see how this can be used in order to generate a matrix representation of say u_{x_1} where y is a discretization of a 2D function. Consider the 2D version of u, that is the matrix $U_{ij} = u((x_1)_i, (x_2)_j)$. Let D be the 1D derivative matrix

$$D = \frac{1}{h}\begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}$$

Let us look at the product

$$DU = \frac{1}{h}\begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}\begin{pmatrix} U_{11} & U_{12} & \cdots & \\ U_{21} & \ddots & & \\ & & U_{n-1,n} & U_{nn} \end{pmatrix} =$$
$$\frac{1}{h}\begin{pmatrix} U_{21}-U_{11} & U_{22}-U_{12} & \cdots & \\ U_{31}-U_{21} & \ddots & & \\ & & U_{n-1,n}-U_{n-2,n} & U_{nn}-U_{n-1,n} \end{pmatrix}$$

If we organize the directions such that x_1, x_2 is the same as the i, j directions on the matrix then we can write the difference matrix which corresponds the the operator ∂_{x_1} in 2D as

$$\partial_{x_1} \approx D_{x_1} = I \otimes D$$

It is easy to verify that the difference matrix in 2D for the x_2 direction can be written as

$$\partial_{x_2} \approx D_{x_2} = D \otimes I$$

Combining them together we obtain a discrete representation for the gradient in 2D

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

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

The extension to 3D is rather straight forward

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

To code this we first write a small code for the 1D case. The code can deal with either 1D cell-centered discretization with Neumann BC or with 1D Nodal discretization with Derichlet boundary conditions.

```
function[D] = ddx(n,opt)

h = 1/n;
switch opt
      case  'nodal'   % second order approximation on nodes with  Derichlet BC
               D = 1/h*spdiags([-ones(n,1),ones(n,1)],-1:0,n,n-1);
               D(1,1) = 2*D(1,1); D(end,end) = D(end,end)*2;
      case  'cell centered'  % second order approximation on cells with Newman BC
                D = 1/h*spdiags([-ones(n,1),ones(n,1)],0:1,n-1,n);

     otherwise
           error('opt = nodal or cell centered')
end
```

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)

D1 = kron(speye(n3),kron(speye(n2),ddx(n1,'nodal')));
D2 = kron(speye(n3),kron(ddx(n2,'nodal'),speye(n1)));
D3 = kron(ddx(n3,'nodal'),kron(speye(n2),speye(n1)));

% DIV from faces to cell-centers
DIV = [D1  D2 D3];
```

and the gradient is the transpose of the divergence.

The discretization is not complete without material averaging functions. We assume that material properties are given in cell centers and that they need to get averaged to the faces Again, we use kronecker products to obtain the averaging matrices. First, we consider 1D averaging

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

A = spdiags(0.5*ones(n+1,2),-1:0,n+1,n);
A([1,end]) = 1;
```

Using the averaging matrices we generate an equivalent to the mass matrix. For example, a mass matrix that multiplies face variables can be generated by the following lines

```
function[Mass] = getFaceMassMatrix(m1,m2,m3)

[n1,n2,n3] = size(m1);
% averaging matrices cell centers -> faces
Acf1 = kron(speye(n3),kron(speye(n2),getCCtoNode(n1)));
Acf2 = kron(speye(n3),kron(getCCtoNode(n2),speye(n1)));
Acf3 = kron(getCCtoNode(n3),kron(speye(n2),speye(n1)));

Mass = blkdiag(sdiag(Acf1*m1),sdiag(Acf2*m2),sdiag(Acf3*m3));
```

Note that in the above matrix we have allowed for anisotropy, that is, we have used three different materials for the three different directions.

Finally, we use the "machinery" above to discretize the Helmholtz equation in 3D Recall that the equation can be written as

$$\nabla \cdot (\rho + \frac{\beta}{i\omega})^{-1} \nabla p + (\omega^2 \kappa - i\omega\alpha) p = \omega^2 s$$

```
function[H] = get3DHelmholtz(kappa,rho,omega,n,L)
% Fourier transform of
% [rho      ][U]    - [-beta    -grad ][U] +  [fu]
% [    kappa][p]_t  - [-div     -alpah][p]    [ft]

h = [L(1)/n(1) L(2)/n(2) L(3)/n(3)];

% Grad and div
grad = getGradientMatrix(n,h);
%%%% PML Attenuation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% number of PML cells
nbc = 5;

alpha = 3e-9*ones(n(1),n(2),n(3));
alpha(nbc+1:end-nbc, nbc+1:end-nbc, nbc+1:end-nbc) = 0;
alpha(nbc+1:end-nbc, nbc+1:end-nbc, 1:end-nbc) = 0;

betax = alpha/2.*rho./kappa;
betay = betax;
betaz = betax;

betax(nbc+1:end-nbc,:,:) = 0;
betay(:,nbc+1:end-nbc,:) = 0;
betaz(:,:,1:end-nbc) = 0;

%%%%%%%%%%%%%%%%%%%%%%%%
% generate material properties matrices
rho = getMassMatrix(rho,rho,rho);
kappa = spdiags(kappa(:),0,n(1)*n(2)*n(3),n(1)*n(2)*n(3));

alpha = spdiags(alpha(:),0,n(1)*n(2)*n(3),n(1)*n(2)*n(3));
beta = getFaceMassMatrix(betax,betay,betaz);

H = grad'* ((rho  + (1i*omega)^(-1)*beta)\ grad) - omega^2*kappa + 1i*omega*alpha;
```

2.7 Discretizing the wave equation

We now discuss the discretization of the wave equation in time. We do not discuss time discretization methods but rather summarize a "standard" solution techniques that is commonly used for the wave equation. We discretize the wave equation in space time

$$\Delta t^{-1}\kappa(p^{n+1} - p^n) = G^\top u^n - \alpha p^n \tag{2.13a}$$
$$\Delta t^{-1}\rho(u^{n+1} - u^n) = G p^{n+1} - \beta u^n \tag{2.13b}$$

which can be solved for p^{n+1} and u^{n+1} using the previous values p^n and u^n. Since κ and ρ are diagonal matrices this calculation can be done pointwise[2]. In the equation above G is the discretization of the gradient and $G^\top$ is the discretization of the divergence. The parameters α and β are PML parameters that attenuate the wave over layers that are close to the boundary.

Note that we can write this as a system

$$\begin{pmatrix} \Delta t^{-1}\kappa & 0 \\ -G & \Delta t^{-1}\rho \end{pmatrix} \begin{pmatrix} p \\ u \end{pmatrix}^{n+1} - \begin{pmatrix} \Delta t^{-1}\kappa - \alpha & -G^\top \\ 0 & \Delta t^{-1}\rho - \beta \end{pmatrix} \begin{pmatrix} p \\ u \end{pmatrix}^{n} = 0 \tag{2.14}$$

There are some important points when considering the wave equation in the context of optimization. One interesting aspect is stability of the solution of the

[2]For finite element these are matrices and the computation can be more difficult

problem. It is well known that for the above system to be stable one requires to have the CFL condition that reads

$$\frac{\kappa}{\rho}\Delta t \leq \Delta x$$

When solving the forward problem it is easy to determine an appropriate Δt but for the inverse problem, we do not have κ and thus may need to adjust Δt when κ is changing. Changing Δt while using some optimization algorithm may lead to other complications (that are discussed later) and thus care must be taken when designing algorithms for the solution of the problem.

2.8 A note about Finite Element discretization

In the above we discussed a finite volume discretization of some simple PDE's. Another common way to discretize such PDE's is by finite element methods. We now shortly review finite elements for the elliptic problem and discuss some of the implementation issues that are special to parameter identification.

We consider the DC resistivity equation in weak form of finding a minimizer to the functional

$$\mathcal{J}(u) = \int_\Omega \frac{1}{2}(\nabla u)^\top m(\nabla u) - uq \, dV$$

Assuming that u is divided into elements and that

$$u(x) = \sum_e u^e(x)$$

and that on each element it has the form

$$u_e = \sum_j u_j^e \varphi_j^e(x)$$

where u_j^e are the coefficients and $\varphi_j^e(x)$ are some basis function over the element.

When substituting u into the integral we obtain a discrete representation of the integral.

$$\mathcal{J}_h(u) = \sum_e m_e \frac{1}{2}(u^e)^\top A_e u^e - (q^e)^\top M_e u_e$$

where we abuse the notation for u between continuous and discrete and assumed that m is piecewise constant over each element. The matrix A_e is the local stiffness matrix and it is given by

$$(A_e)_{ij} = \int_{\Omega_e} \nabla\varphi_i(x) \cdot \nabla\varphi_j(x) \, dV \tag{2.15}$$

Differentiating $\mathcal{J}$ with respect to u we obtain a discretization to the forward problem

$$A(m)u = q$$

where

$$A(m) = \sum_e m_e A_e$$

The latter representation is important for optimization and will play a role later in the discussion. In particular, we note that assuming some finite element geometry (a mesh) changing the coefficients on the mesh can be done easily if we save the element matrices. Most FEM codes tend to assemble the stiffness matrix and not keep the individual, element stiffness matrices. In parameter estimation the parameter m changes from iteration to iteration. Reassembling the matrix can take some time and therefore it is recommended to keep the individual stiffness matrices.

2.9 A note about solving the linear systems

The discretization of the problems discussed above give rise to linear systems to be solved. The numerical solution of linear systems is a huge topic by itself and we do not intend to cover it in this course. We will heavily use direct factorization methods, iterative methods such as preconditioned conjugate gradient (PCG) and other Krylov methods. We refer the reader to [9, 3, 6, 13] for more details.

In this section we discuss some specific aspects of the solution techniques that are common to parameter estimation problems.

First, and most important, in many parameter estimation problems the PDE is solved many times. For example, in impedance tomography the linear system has to be solved for at least $N_s \times 2$ where N_s is the number of sources, and in many cases many more times than that (this is due to the estimation of the gradients and Hessians). For many problems N_s is a rather large number and could easily reach to a few thousands. For such cases the linear systems are solved tens or even hundred's of thousands of times! For these applications, if possible, direct factorization is, by far, the best way to deal with the linear system. For problems in 1D and for most problems in 2D direct factorization methods can be easily used however, for problems in 3D and 4D (space-time) it is not always possible to factorize the systems. Recently, parallel direct solvers have been available, for example MUMPS [1] and superLU [7]. Given the appropriate computational resources, these codes can solve problems with millions of unknowns by factorization. For problems in 3D such solvers are preferable.

For many problems, the memory requirements of direct methods is a prohibitive factor and iterative methods must be used. Iterative methods typically require a good preconditioner which is rather problem dependent. For the examples in the course we use some rather basic preconditioners such as Jacobi and Gauss-Seidel. We emphasize that these are not necessarily good proconditioners but they are useful and very easy to code. Optimal preconditioners are typically based on multigrid methods but they can be difficult to program and implement.

For large problems with multiple right hand sides when iterative methods are used it is sometimes possible to combine direct and iterative methods. For example, using ILU as a preconditioner with a very low drop-tolerance or domain decomposition with very large domains, where the LU factorization on each domain

is saved. Designing preconditioners for the forward problem has to be done for each problem individually and we do not discuss this further here.

2.10 Problems for chapter 2

1. Write a code to solve Poisson equation with variable coefficients in 3D and provide testing to see that the code is working appropriately

2. A forth order finite difference operator in 1D has the form

$$(u_x)_{i+\frac{1}{2}} = \frac{1}{h}\left(\frac{1}{24}u_{i-1} - \frac{9}{8}u_i + \frac{9}{8}u_{i+1} - \frac{1}{24}u_{i+2}\right)$$

   which, for nodal discretization, can be discretized as

```
D = 1/h*spdiags([1/24*ones(n+1,1),-9/8*ones(n+1,1), ...
                 9/8*ones(n+1,1),-1/24*ones(n+1,1)],-1:2,n,n+1);
```

   Use this 1D discretization to generate a 3D discretization for the Poisson equation with variable smooth coefficients. Show that your discretization is indeed 4^{th} order.

3. In some cases it is easier to work directly with the discretization of the second derivative.

   (a) Derive a 1D second order approximation to the operator ∂_{xx} with Derichlet BC

   (b) The Stokes system for $(\vec{u}, p) = (u_1, u_2, p)$ reads

$$\Delta u_1 + p_x = f_1 \qquad \Delta u_2 + p_y = f_2 \quad \nabla \cdot \vec{u} = 0$$

   Discretizing $\vec{u}$ on cell faces and p in cell centers, derive a second order discrete approximation to the system. Code and test it. This famous discretization is refers to as Marker And Cell (MAC).