1 Operator Splitting

In general we will be searching for methods of solving problems containing terms where the courant condition is too restrictive for practical computation; typical in this regard is the inclusion of diffusion or thermal conduction which contain second derivatives in space implying that time steps must scale as $\Delta t \propto 1/(\Delta x)^2$. In this case we can not solve the problems explicitly but rather solve (at least part of) the problem implicitly to avoid the time step restrictions imposed by the inclusion of these operators. We follow Numerical Recipes and consider an initial value problem of the form

$$\frac{\partial u}{\partial t} = Lu$$

where $L$ is some operator containing e.g. the MHD equations and including other terms such as radiative losses and thermal conduction. $L$ is not necessarily linear, but suppose that it can be written as a linear sum of $m$ pieces which act additively on $u$,

$$Lu = L_1 u + L_2 u + \ldots + L_m u$$

Let us assume that for each of these operators we know a differencing scheme, $U$, for updating the variables $u$ from timestep $t$ to $t + \Delta t$, valid if that were the only operator on the right hand side. Symbolically we would write

$$u^{n+1} = U_1(u^n, \Delta t)$$
$$u^{n+1} = U_1(u^n, \Delta t)$$
$$\ldots$$
$$u^{n+1} = U_m(u^n, \Delta t)$$

One form of operator splitting is to go from timestep $t$ (or $n$) to $t + \Delta t$ ($n+1$) by the following steps:

$$u^{n+1(1/m)} = U_1(u^n, \Delta t)$$
$$u^{n+1(2/m)} = U_2(u^{n+1(1/m)}, \Delta t)$$
$$\ldots$$

For example our combined advection-diffusion equation

$$\frac{\partial u}{\partial t} = -v \frac{\partial u}{\partial x} + D \frac{\partial^2 u}{\partial x^2}$$

might be solved by using an explicity scheme for the advective term combined with an implicit scheme for the diffusion term.
1.1 Relaxation methods for Boundary Value Problems

When solving the implicit part of an operator split problem as outlined above we will in general find ourselves facing problems where it is necessary to invert a matrix. An identical problem arises when one attempts to solve elliptic or elliptic-type problems using a relaxation technique; i.e. when one rewrites the problem

\[ \mathcal{L} u = \rho \quad \text{to} \quad \frac{\partial u}{\partial t} = \mathcal{L} u - \rho \]

with \( \mathcal{L} \) and elliptic operator \( \rho \) some right hand side. The relaxation technique consists of making a guess to the final solution \( u \) and letting this solution relax to its equilibrium as \( t \to \infty \).

Let us see how this is implemented in a two dimensional diffusion problem

\[ \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \rho. \]

If we discretize using FTCS differencing we get

\[ u_{j,l}^{n+1} = u_{j,l}^n + \frac{\Delta t}{h^2} \left( u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^n \right) - \frac{h^2}{4} \rho_{j,l}. \]  

The largest timestep we can take for this problem while remaining stable in one dimension is if \( \Delta t/h^2 \leq \frac{1}{2} \), for two dimensional problems it is if \( \Delta t/h^2 \leq \frac{1}{4} \). Assume now we take the longest possible timestep and set \( \Delta t = h^2/4 \). Our equation then becomes

\[ u_{j,l}^{n+1} = \frac{1}{4} \left( u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n \right) - \frac{h^2}{4} \rho_{j,l}, \]

which shows that the algorithm consists of using the average of \( u \) at its four nearest neighbor points on the grid, iterating until convergence. This is in fact the famous Jacobi’s method which is not in much practical use because it converges much too slowly, but contains much pedagogical insight which we will use in understanding how the multigrid method works.

Another classical method is the Gauss-Seidel method which is very similar to Jacobi’s method, but where one takes the updated \( u \)’s into account as soon as they are available,

\[ u_{j,l}^{n+1} = \frac{1}{4} \left( u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^{n+1} + u_{j,l-1}^{n+1} \right) - \frac{h^2}{4} \rho_{j,l}, \]

assuming we are proceeding along rows incrementing \( j \) for fixed \( l \). The difference here is that the averaging is done “in place” instead of being “copied” from an earlier timestep to a later one.

Now look at Jacobi and/or Gauss-Seidel in terms of matrix splitting, changing notation from \( u \) to \( x \) to conform with standard matrix notation

\[ A x = b. \]

Now split \( A \) as

\[ A = L + D + U \]

where \( D \) is the diagonal part of \( A \), \( L \) is the lower triangle of \( A \) with zeroes on the diagonal, and \( U \) is the upper triangle of \( A \) with zeroes on the diagonal. In the Jacobi method we write

\[ D \cdot x^{(r)} = -(L + U) \cdot x^{(r-1)} + b \]

For our problem \( 1 \) \( D \) is the identity matrix.

What is the rate of convergence of the Jacobi method? The matrix \( -D^{-1} \cdot (L + U) \) is the iteration matrix which maps one set of \( x \)’s into the next. The iteration has eigenvalues, each which reflects the amplitude of

\[ \text{The calculation of eigenvalues from the Jacobi and other methods is for the time being outside the scope of our text.} \]
a particular eigenmode of residual is suppressed during one iteration. The rate of convergence is set by the rate of the slowest decaying eigenmode, i.e. the factor with the largest modulus, which is called the spectral radius, $\rho_s$.

The number of iterations $r$ required to reduce the overall error by a factor $10^{-p}$ is estimated by

$$r \approx \frac{p \ln 10}{(- \ln \rho_s)}.$$  

For example equation 1 on a $J \times J$ grid, the asymptotic formula for large $J$ is

$$\rho_s \simeq 1 - \frac{\pi^2}{2J^2}.$$  

The number of iterations required to reduce the error by a factor $10^{-p}$ is thus

$$r \approx \frac{2pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{2}pJ^2$$

i.e. the number of iterations is proportional to the number of mesh points, $J^2$.

The Gauss-Seidel method, equation 2, corresponds to the matrix decomposition

$$(L + D) \cdot x^{(r)} = -U \cdot x^{(r-1)} + b$$

which can be quickly ascertained as correct by writing this expression out in component form. One can show that the spectral radius is just the square of the spectral radius of the Jacobi method. Therefore,

$$\rho_s \simeq 1 - \frac{\pi^2}{J^2}$$

$$r \approx \frac{pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{4}pJ^2$$

which is a factor two improvement over the Jacobi method.

### 1.2 The multigrid method

Multigrid methods can solve elliptical PDE’s discretized on $N$ grid points on only $O(N)$ operations, i.e. much more rapidly than the Jacobi and Gauss-Seidel methods outlined above. This increase in speed comes at a cost of significantly greater algorithm complexity, but since the method can be applied very generally to a wide set of elliptic or elliptic-type problems, even non-linear problems, the increased complexity is well worth the programming effort.

To reiterate, we wish to solve a problem of the form

$$\mathcal{L}u = f \quad \text{or} \quad \mathcal{L}_h u_h = f_h$$

where $\mathcal{L}$ is a elliptic operator and the right hand equation indicates the discretized version of this problem on a grid of gridsize $h$. Let us now call our current approximation to the solution $\tilde{u}_h$ and the exact (discretized) solution $u_h$. The error, or correction is then defined as $v_h \equiv u_h - \tilde{u}_h$, and the residual (or defect) is given by

$$d_h = \mathcal{L}_h \tilde{u}_h - f_h$$

Since $\mathcal{L}_h$ (for now) is linear, we can write

$$\mathcal{L}_hv_h = -d_h$$
Our goal is thus to find $v_h$. In the methods described above the path to this was to make some sort of simplified version of the operator $\hat{L}_h \approx L_h$ such that we could solve for $\tilde{v}_h$:

$$\hat{L}_h \tilde{v}_h = -d_h.$$ 

as was the case for the Jacobi and Gauss-Seidel methods, where the approximate correction found is used to generate a new approximation to the solution

$$\bar{u}^\text{new}_h = \bar{u}_h + \tilde{v}_h,$$

and then iterating until the desired accuracy is obtained.

As opposed to this procedure, in the multigrid method one rather seeks a coarser version of the operator $L_h$: $L_H$ defined on a coarser grid such that e.g. $H = 2h$. Presumably, since there are fewer points on the coarser grid, it will be easier to solve the problem there. In order to make use of the coarser grid we will need operators to move variables from one grid to the other. The restrctor or injector operator moves variables from the fine grid to the coarse grid,

$$d_H = R d_h,$$

while the interpolation or prolongation operator moves variables in the other direction, from coarse to fine

$$\bar{v}_h = P \bar{v}_H$$

The operators $R$ and $P$ should both be linear. Finally it remains to update our current approximation to the solution

$$\bar{u}^\text{new}_h = \bar{u}_h + \bar{v}_h$$

We thereby have the basis of an algorithm for finding the solution to our (linear) problem:

1. Compute the defect on the fine grid from $d_h = L_h - f_h$
2. Restrict the defect to the coarse grid $d_H = R d_h$
3. Solve the problem exactly on the coarse grid $\bar{v}_H = L^{-1}_H d_H$
4. Prolong the coarse solution to the fine grid $\bar{v}_h = P \bar{v}_H$
5. Compute the next approximation $\bar{u}^\text{new}_h = \bar{u}_h + \bar{v}_h$

The reason this method works is indicated in figure 1 where we have plotted simulated error $v_h$ data as a function of position (grid point) for the fine grid. The error is comprised of many different spatial frequencies, the highest frequency and lowest frequency component are plotted. Since the Gauss-Seidel (and Jacobi) method essentially work by averaging nearest neighbors it is very efficient at removing the high frequency error. On the other hand the low frequency error will take on order the number of grid points to remove, as discussed in the previous section. The “two grid” (and multigrid) technique work by taking advantage of this: when the problem is moved to a coarser grid, previously low frequency errors become high frequency errors and can be effectively removed by Gauss-Seidel smoothing. The full two-grid algorithm thus work as follows

1. Pre-smooth at the fine grid level using a few Gauss-Seidel iterations.
2. Inject the error to a coarse grid and solve the problem on this grid
3. Post-smooth the prolonged solution on the fine grid and calculate a new approximation $\bar{u}^\text{new}_h$.

The extension of the two grid algorithm to the multigrid algorithm is then simply to recursively invoke a coarser grid until a grid coarse enough to solve the problem exactly or through very many (cheap) iterations of the smoothing operator.
Figure 1: Simulated error $v_h = u_h - \tilde{u}_h$ (thick line) is seen to be comprised of many (spatial) frequencies, plotted are the lowest (dashed line) and highest (dotted line). Since it essentially works by averaging nearest neighbors, a Gauss-Seidel iteration will remove the highest frequency component very rapidly while the low frequency component will take of order $O(N)$, where $N$ is the number of grid points, to remove.

1.2.1 Prolongation and Injection operators

As indicated above the first (and only?) smoothing method to be tried is Gauss-Seidel since it usually leads to a good convergence rate. The exact form of the Gauss-Seidel depends on the ordering chosen for the grid points. For typical second-order problems such as the diffusion term shown above it is often best to use so-called red-black ordering, making one pass through the mesh updating the “even” points and another pass updating the “odd” points. When quantities are strongly coupled along a specific dimension as opposed to another one should try to relax the line along that dimension simultaneously. Line relaxation for nearest-neighbor coupling involves solving a tridiagonal system, which is still efficient. Relaxing odd and even lines on successive passes is called zebra relaxation and is usually preferred over simple line relaxation.

A short notation for prolongation and injection operators is to give their symbol. The symbol of $P$ is found by considering $v_H$ to be 1 at some mesh point $(x, y)$, zero everywhere else and then asking for the values of $Pv_H$. For example using bilinear interpolation as a prolongation operator we find

$$P = \begin{bmatrix} 1 \frac{1}{4} \frac{1}{4} \frac{1}{2} \frac{1}{4} \frac{1}{4} \frac{1}{4} \frac{1}{4} \frac{1}{4} \frac{1}{2} \frac{1}{4} \frac{1}{4} \frac{1}{2} \frac{1}{4} \end{bmatrix}$$

The symbol for the injection operator $R$ is defined by considering $v_h$ to be defined everywhere on the fine grid, and then asking what is $Rv_h$ as a linear combination of these values. One possible choice for $R$ is straight injection, $R = [1]$, but this has proved a less than good in practice. It turns out that the best choice for $R$ is to make it the adjoint operator to $P$. To define the adjoint, define the scalar product of two grid
functions \( u_h \) and \( v_h \) for mesh size \( h \) as
\[
(u_h | v_h)_h \equiv h^2 \sum_{x,y} u_h(x, y)v_h(x, y)
\]

Then the adjoint to \( P \) denoted \( P^\dagger \), is defined by
\[
\langle u_H | P^\dagger v_h \rangle_H = \langle P | v_h \rangle_h
\]

If now \( P \) is bilinear interpolation, choose \( u_H = 1 \) at \((x, y)\) zero otherwise, and set \( P^\dagger = R \) in equation 3 and set \( H = 2h \). Then
\[
(Rv_h)_{(x,y)} = \frac{1}{4} v_h(x, y) + \frac{1}{8} v_h(x + h, y) + \frac{1}{16} v_h(x + h, y + h) + \cdots
\]

so the symbol for the interpolation operator is
\[
R = \begin{bmatrix}
\frac{1}{16} & \frac{1}{8} & \frac{1}{8} \\
\frac{1}{8} & \frac{1}{2} & \frac{1}{8} \\
\frac{1}{8} & \frac{1}{8} & \frac{1}{16}
\end{bmatrix}
\]

A general rule when \( P^\dagger = R \) and \( H = 2h \) is that \( R \) is \( \frac{1}{4} \) of the transpose of symbol \( P \). This choice of \( R \) is called \textit{full weighting}. Another choice for \( R \) is \textit{half weighting}:
\[
R = \begin{bmatrix}
0 & \frac{1}{8} & 0 \\
\frac{1}{8} & \frac{1}{2} & \frac{1}{8} \\
0 & \frac{1}{8} & 0
\end{bmatrix}
\]

Note that \( R \) and \( P \) should enforce the boundary condition for the problem being solved.

1.2.2 Non-linear Multigrid

Now let us look into a non-linear elliptic equation, which we write symbolically as
\[
L(u) = 0
\]

where we have moved an explicit source term into the operator on the left hand side. After discretization we will find that during the multigrid solution we will need to consider a situation with a non-zero right hand side generated as a result of the method:
\[
L_h(u_h) = f_h.
\]

We can directly apply the multigrid approach to solving this equation as long as we have a suitable \textit{non-linear} relaxation method to smooth the high frequency errors, plus a procedure for approximating corrections on a coarser grid.

Given an approximation \( \tilde{u}_h \) to the solution we seek a smooth correction \( v_h \) to solve the equation
\[
L_h(\tilde{u}_h + v_h) = f_h.
\]

To find \( v_h \) note that
\[
L_h(\tilde{u}_h + v_h) - L_h(\tilde{u}_h) = f_h - L_h(\tilde{u}_h) = -d_h
\]

A few non-linear relaxation iterations will smooth the right hand side and we can transfer the left hand side to a coarse grid
\[
L_H(u_H) - L_H(R\tilde{u}_h) = -Rd_h \quad \text{or, we solve} \quad L_H(u_H) = L_H(R\tilde{u}_h) - Rd_h
\]
on the coarse grid. Note also that this is how the non-zero right hand sides appear. Now assume that the approximate coarse grid solution is $\tilde{u}_H$, then the coarse-grid correction is

$$\tilde{v}_H = \tilde{u}_H - R\tilde{u}_h$$

and

$$\tilde{u}^\text{new}_h = \tilde{u}_h + P(\tilde{u}_H - R\tilde{u}_h)$$

In general $PR \neq 1$, so $\tilde{u}^\text{new}_h \neq P\tilde{u}_H$: the interpolation error comes only from the correction, not from the full solution $\tilde{u}_H$. This algorithm (called FAS for Full Approximation Storage Algorithm) is very similar to the linear multigrid algorithm. The only difference is that both the defect $d_h$ and the relaxed approximation $u_h$ have to be injected to the coarser grids and where it is equation 4 this is solved by recursive invocation of the algorithm.

**The Dual Viewpoint.** Another way of looking at the non-linear multigrid algorithm is to consider the local truncation error defined as

$$\tau \equiv L(u) - f_h \quad \text{or} \quad L(u) = f_h + \tau$$

Figure 2: Temperature from a Bifrost simulation at four multigrid levels. In this case the equation solved is $\partial e/\partial t + \nabla \cdot F_c = 0$, with $F_c = -\kappa_0 T^{5/2} \nabla T$. The displayed data represents the temperature on one processor; after it has been injected onto the fourth level the temperature from all processors is gathered and solved as a single system (see text).
where \( u \) is the exact solution of the original continuum equation. We see that \( \tau \) can be regarded as the correction to \( f_h \) so that the solution of the fine grid equation will be the exact solution \( u \).

Define now the relative truncation error

\[
\tau_h \equiv L_H(Ru_h) - RL_h(u_h)
\]

or since \( L_h(u_h) = f_h \), this can be written

\[
L_H(u_H) = f_H + \tau_h,
\]

i.e. \( \tau_h \) is the correction to \( f_H \) that makes the solution of the coarse grid equation equal to the fine grid solution. We cannot compute \( \tau_h \), but we do have an approximation from using \( \tilde{u}_h \) in equation 5.

\[
\tau_h \simeq \tilde{\tau}_h \equiv L_H(R\tilde{u}_h) - RL_h(\tilde{u}_h).
\]

But then replacing \( \tau_h \) by \( \tilde{\tau}_h \) in equation 6 we get

\[
L_H(u_H) = L_H(R\tilde{u}_h) - R\tilde{d}_h
\]

which is just the coarse grid equation 4.

There are therefore two complementary viewpoints for the relation between the fine and the coarse grids:

- The coarse grid is used to accelerate convergence of the smooth portion of the fine grid residual.
- The fine grid is used to compute corrections to the coarse grid solution so that one attains fine grid accuracy on the coarse grid.