numbers to decide in which direction the photon will proceed after each scattering event. We must also use a random number to find out where along the present path the next scattering event will take place. We repeat this process for many tens of thousands (millions?) of such photons, until we have a good statistical sample of paths that the photons will follow.

The advantage of Monte Carlo methods is that they are easy to understand, since they actually *simulate* the motion of photons. Also, it is easy to implement complicated microphysics into Monte Carlo methods.

We will encounter various versions of Monte Carlo methods throughout this lecture. They are all based on the same basic principles, but they are sufficiently diverse that we will not devote a separate chapter to them. Instead we will discuss them separately for dust thermal continuum transfer (Chapter 5), light scattering off small particles (Chapter 6) and line transfer (Chapter 7).

In this section we will give an introduction to the method, applied here to the simple multiple-isotropic-scattering problem. In most of what follows we assume that we have a pure scattering problem, i.e. $\epsilon_\nu = 0$, except if we explicitly state otherwise.

4.2.1 Finding the location of the next scattering event

Suppose we follow a photon that is, at present, located at position $\mathbf{x}$ in direction $\mathbf{n}$. How do we find out where it will have its next scattering event (assuming $\epsilon_\nu = 0$)? Because scattering is not a deterministic process, we cannot predict exactly where this will be, but we know that it follows a Poisson distribution. In our Monte Carlo approach we want to use a random number generator to find the optical depth τ that the photon will travel until the next scattering event. We know that this must have a probability distribution

$$p(\tau) = e^{-\tau} \tag{4.22}$$

This can be achieved by drawing a random number ξ that is uniformly distributed between 0 and 1, and computing the τ according to:

$$\tau = -\ln(\xi) \tag{4.23}$$

Sometimes you find the formula $\tau = -\ln(1-\xi)$ in the literature, which is equivalent. It is slightly more correct, because typical random number generators generate $\xi \in [0, 1\rangle$, meaning, in theory, they could draw $x = 0$, in which case the $\ln(\xi)$ fails. However, if you have a good random number generator, then the period is so tremendously large, that the chance of ever drawing exactly 0 is negligible.

Once we have fixed the optical depth where the next scattering event takes place, we must travel along the ray cell-by-cell until we arrive at this point. Each segment of ray corresponds to some $\Delta\tau$ (see the figure of Subsection 3.8.7). Every time we traverse such a segment of ray, we subtract $\Delta\tau$ from τ, i.e. we evaluate $\tau \leftarrow \tau - \Delta\tau$. As we pass along the ray, τ decreases steadily, until we arrive at a segment of ray where $\Delta\tau$ is larger than the remaining τ. This is the segment where the next scattering event will happen. The precise location will be a fraction of $\tau/\Delta\tau$ times the length of the segment from its start.

It can also happen that we will never arrive at our next scattering event because the photon packet escapes from the cloud. This is then the end of the journey for the photon packet, and we will go on to trace the next photon packet.

If we have, in addition to scattering, also absorption ($\epsilon_\nu > 0$), then we must also treat these absorption events and allow new photons to be generated through thermal emission. This is not entirely trivial, so we defer it to Chapter 5.

4.2.2 Drawing a random scattering direction

Once a scattering event happens, a new direction must be drawn. One way to obtain a random direction is to draw two uniformly distributed random numbers between 0 and 1, let us call them ξ_1 and ξ_2, and to compute θ and ϕ from them according to:

$$\theta = \text{acos}(2\xi_1 - 1) \qquad , \qquad \phi = 2\pi\xi_2 \tag{4.24}$$

It is important to understand that $\theta = \pi\,\xi_1$ would give a wrong distribution! The arccosine comes from the fact that in a random distribution of directions, $\mu = \cos\theta$ is uniformly distributed.

Another way to draw a random direction would be to draw three random numbers between 0 and 1 ξ_1, ξ_2 and ξ_3, compute $n_x = 2\xi_1 - 1$, $n_y = 2\xi_2 - 1$ and $n_z = 2\xi_3 - 1$ and compute $n = \sqrt{n_x^2 + n_y^2 + n_z^2}$. If $n > 1$ we reject it, and go back to drawing three new uniform random numbers and repeat the procedure. If $n \leq 1$, on the other hand, we retain it and normalize the vector through $n_x \leftarrow n_x/n, n_y \leftarrow n_y/n$ and $n_z \leftarrow n_z/n$.

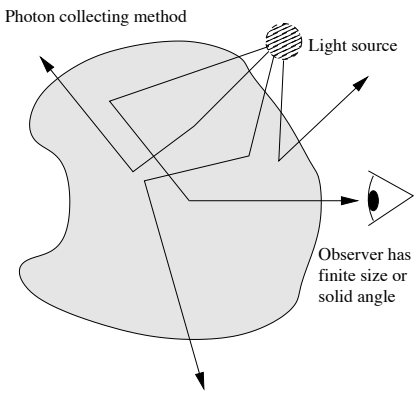
4.2.3 The meaning of a "photon packet"

A star sends out of the order of 10^{45} photons per second. It is clear that we cannot, in our computer model, follow each of these photons. When we model a "photon" in our Monte Carlo method, we actually tacitly assume that it represents many photons at once. One can see this as if we follow a whole packet of photons instead of just one: a "photon packet". The approximation we make is that we assume that all these photons in a single photon packet follow the same path.

If we look a bit closer, however, we see that the photon packet picture can be a bit confusing for the problem of stationary radiative transfer we consider here. Intuitively a packet would contain a certain number of photons, but in reality it contains a certain number of photons *per second*. The star emits 10^{45} photons *per second*, meaning that if our Monte Carlo simulation involves 10^5 photon packets, then each packet represents 10^{40} photons *per second*. In other words: a Monte Carlo photon packet does not represent a total energy, but represents a total luminosity (=energy/second). If we model the Sun with 10^5 photon packets, then each packet represents a luminosity of 10^{-5} solar luminosities.

Note that if we would use the Monte Carlo method for time-dependent radiative transfer (see Section 3.6) then a photon packet indeed represents an energy, or equivalently, a number of photons.

4.2.4 How to make an image using Monte Carlo - a preview



Modeling how the photon packets move through the cloud is not yet the full story. In the end we want to predict what we would observe if we look the cloud. This is not trivial. Usually the observer is tiny compared to the object of interest, or is located "at infinity" in a well-defined direction. The chance that a photon packet will find its way from the source to the observer is small. Most photons will escape to outer space in arbitrary directions, and miss the observer.

One brute-force way to overcome the "problem of the small observer" is by artificially "enlarging" the observer to enhance the chance of capturing a photon packet. For an observer at infinity one could assign a solid angle $\Delta\Omega$ around the direction of the observer, and collect all photons that end up in that direction. The larger we choose $\Delta\Omega$, the more photon packets we collect, but the less accurate the result since we thereby average over outgoing direction. To obtain a good result our observer needs to collect a large number of photon packets, so that the noise on the result is small. This noise is called *Monte Carlo noise*, and is a typical drawback of Monte Carlo methods. The only way to reduce this noise is to increase the number of photon packets we use,

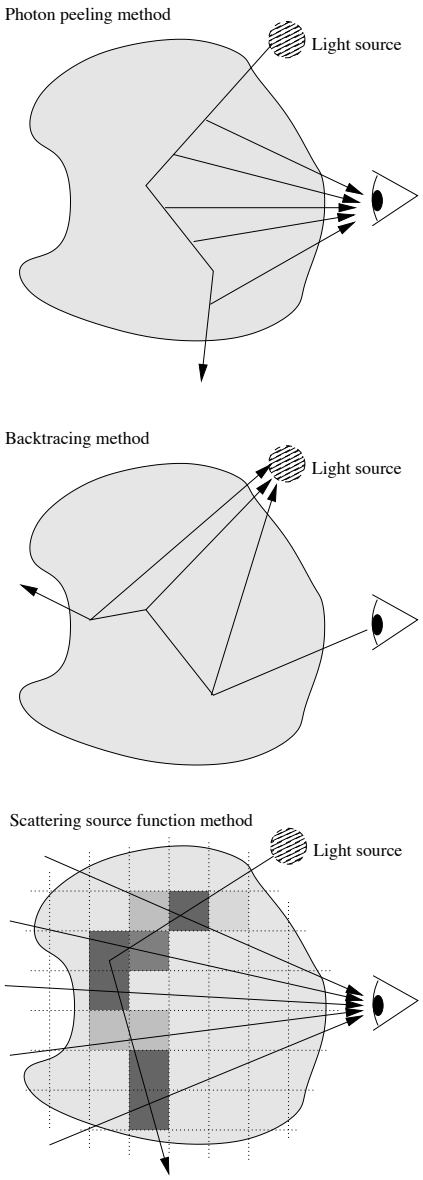
so that each photon packet corresponds to a smaller contribution to the result. There are many other subtleties with this "photon collection method", but we will defer this to later chapters.

A related method, which is often used is to continuously "peel off" radiation from a photon packet as it passes through the cloud, and compute what the observer would see from this, including the extiction from each of these points to the observer. This is a somewhat costly method but it works well.

Another approach is to reverse the problem. We start our photon packets at the observer and trace the packets back along their rays. The scattering events are then no longer "where to?" but "from where?" events. This technique is often used in 3-D computer graphics rendering, but also in many scientific applications. However, if our source of photons is small, we encounter the "problem of the small source". So we now have the opposite problem: how to ensure that we end up at the source? One method, often employed, is to "force" the photon packet toward the source at the final (i.e. in reality the first) scattering event. But how do we know if a scattering event is the final (first) one? In fact, we don't know that. So what we can do is to calculate at every scattering event what the chance is that a photon from the source will scatter into the ray. We continue this backtracing procedure until the next (i.e. in reality previous) scattering event would be outside of our cloud. If our problem also includes thermal emission/absorption (i.e. $\epsilon_\nu > 0$), and if this thermal emission far outshines any small sources, then the "problem of the small source" is usually not so accute since then the cloud itself is the source. Also, a non-zero ϵ_ν means that we do not necessarily have to backtrace a photon until we leave the cloud again, because in addition to scattering events we can also encounter an emission event: this means that we have then arrived at the origin of the photon, and our path ends (or better: it starts).

A final approach is the scattering source function approach. We follow each photon packet as it moves through the cloud. As it does so, we add an appropriate contribution of this photon packet to the scattering emissivity function j_ν of each cell it passes. The length of the ray segment will be taken into account, so that if a photon packet passes right through the middle of a cell, its contribution to that cell will be larger than if it just "cuts a corner". Once we have launched all photon packets, the function $j_\nu(\mathbf{x})$ is known throughout the cloud. We can now render an image by integrating the formal transfer equation along a pre-defined set of rays all ending at the observer. In the next chapters we will discuss this method of the scattering source function in more detail, including the appropriate formulas.

This list of methods is not exhaustive. A host of ideas have been tried out over the many years. Each has its advantages and disadvantages. But the above list is, I think, a reasonble overview of the most commonly used methods. Many special methods can be considered versions of the above.



4.3 Discrete Ordinate methods (a short note)

The opposite of Monte Carlo Methods is *discrete ordinate methods*. These methods solve the problem by dividing all coordinates, including the angles and the frequency, into discrete grid points or grid cells. In a way they are "cleaner" than Monte Carlo methods because they do not introduce Monte Carlo noise. But this can give a false sense of comfort, because also discrete ordinate methods introduce errors due to limited resolution of θ and ϕ - and these errors do not show up as conspicuous noise but will remain hidden in a reasonable-looking answer. In that sense Monte Carlo methods will raise the alarm automatically if not enough computer power has been used, while discrete ordinate methods will give an answer that might look reasonable, but might be very wrong nonetheless. If done well, however, discrete ordinate methods can have advantages over Monte Carlo methods, and produce indeed "cleaner" results. But "doing it well" can be very challenging, and may require some experience from

the modeller.

There is no "discrete ordinate method" as such. It is a *class* of methods in which, in addition to x, y, z, ν also θ and ϕ are discretized. This introduces several subtleties that require special care. But these issues are perhaps hard to understand without some of the background of the methods and their applications that employ the discrete ordinate approach. We will therefore discuss them in later chapters, and focus in this chapter on examples in 1-D.

4.4 Lambda Iteration (LI) and Accelerated Lambda Iteration (ALI)

One of the cornerstones of radiative transfer theory is a very simple iteration method called *Lambda Iteration* (LI). It sounds fancier than it is. In fact, it is presumably the iteration scheme that you would intuitively develop yourself if someone would ask you to solve the coupled set of equations, Eqs.(4.1, 4.2), without you having any prior knowledge of radiative transfer algorithms. It goes like this:

1. Make an initial guess for the mean intensity $J_\nu(\mathbf{x})$ (for instance, take it zero).
2. Integrate the formal transfer equation along a large number of rays, such that close to every point $\mathbf{x}$ a sufficient number of rays pass by that we can make an approximate calculation of the mean intensity $J_\nu(\mathbf{x})$.
3. Compute $J_\nu(\mathbf{x})$ at all locations, thereby computing the scattering emissivity $j_\nu(\mathbf{x})$.
4. Go back to 2, until $J_\nu(\mathbf{x})$ appears to be converged.

It is essentially a scheme that alternates between a global calculation (the transfer equation) and a local calculation (computing the scattering emissivity). That's all there is to it! Indeed, this algorithm is so simple, at least from a conceptual viewpoint (a numerical implementation is another matter altogether), that it has been re-invented over and over again in the many different fields of physics and engineering, and thereby goes by different names. We will, however, stick to the Lambda Iteration name, for reasons that will become clear later.

Note that we have actually already "invented" this method ourselves in the text above, perhaps even without being aware of it. Remember that in Section 4.1.1 we talked about "chicken-egg cycles". Well, each "chicken-egg cycle" is in fact one Lambda Iteration cycle.

This brings us to a physical interpretation of Lambda Iteration: After the first iteration we have treated single scattering. After the second iteration we have, in addition to single scattering, also treated double-scattering. After the N-th iteration we have treated all multiple scattering trajectories up to, and including, N-scattering.

This means that we can in fact get a rough prediction for the convergence of this method. If we have $\epsilon = 0$ (only scattering) and we have a cloud of $\tau \gg 1$, then this means that we can expect on average τ^2 scattering events before a photon escapes the cloud (assuming the light source is somewhere inside the cloud). This means that we would need *at least* $N_{\rm iter} = \tau^2$ iterations (presumably more) before we can be sure that convergence is reached. If we have a moderate optical depth of, say, $\tau = 100$ this would require at least 10000 iterations.

This shows the main drawback of Lambda Iteration: for scattering problems with zero or very low absorption, and for moderate to high optical depth, Lambda Iteration can converge extremely slowly. In fact, the slowness of convergence can sometimes be so extreme that often people have mistakenly thought that convergence has been reached even though it was not the case by far!

Of course, we have taken quite an extreme example: that of zero absorption. If $\epsilon > 0$ the number of scattering events experienced by a single photon will be reduced because the photon will eventually get absorbed. An estimate of the number of scattering events experienced by a single photon, for the case of $\epsilon > 0$, is:

$$\langle N_{\text{scatterings}} \rangle \simeq \min\left(\tau_{\text{scat}}^2, \frac{1}{\epsilon}\right) \tag{4.25}$$

For Lambda Iteration to converge we need

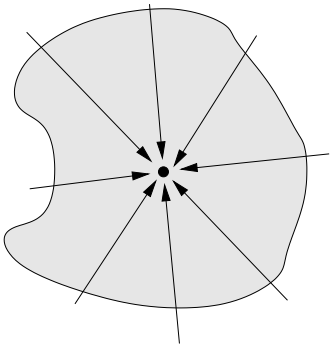
$$N_{\text{iter}} \gg \langle N_{\text{scatterings}} \rangle \tag{4.26}$$

Given these somewhat pessimistic estimates, it may seem that it is hopeless to pursue Lambda Iteration any further. There are two reasons why we will resist this thought: (1) There are not many alternatives that are as easy to program and work any better, and (2) there are clever techniques to dramatically speed up the convergence of Lambda Iteration. We will introduce two techniques of acceleration that can be used together: (a) *approximate operator acceleration*, also often called *Accelerated Lambda Iteration*, and (b) *Ng-acceleration*.

4.4.1 The Lambda Operator

So what is this "Lambda Operator"? It is best described as a task: The task of computing the mean intensity J at some point $\mathbf{x}$ knowing what the source function $S_\nu(\mathbf{x}')$ is for all $\mathbf{x}'$. Of course in addition to $S_\nu(\mathbf{x}')$, we also have to know $\alpha_\nu(\mathbf{x}')$ and any light sources that may be present.

"Partial" Lambda Operator



To find the mean intensity at *all* points $\mathbf{x}$ we have to apply it to each point. One could call this the "full" Lambda Operator, in contrast to the "partial" Lambda Operator that only calculates J_ν at one point (though these are not an official terms; I just invented them for clarification).

So rather than a formal, abstract, operator, we can regard the Lambda Operator as a C++ or F90 subroutine: You give it $S_\nu(\mathbf{x}')$ for all $\mathbf{x}'$, tell it where you want to know J, and it will compute it for you. That is, the "partial" Lambda Operator subroutine will do so. The "full" Lambda Operator subroutine will in fact compute it everywhere in one go. In practice it will be clear from the context whether we mean the "partial" or the "full" Lambda Operator.

With this new concept we can now write:

$$J_\nu = \Lambda[S_\nu] \tag{4.27}$$

The square brackets here mean that it is not a local operation, but involves integrals over the entire domain. Note that for every frequency ν this Lambda Operator is different, because the $\alpha_\nu(\mathbf{x}')$ is different.

Using Eq. (4.27) we can now rewrite the equation for the source function including scattering and absorption (Eq. 4.19) as:

$$S_\nu = \epsilon_\nu B_\nu(T) + (1 - \epsilon_\nu)\Lambda[S_\nu] \tag{4.28}$$

This is the mathematically abstract way of writing the "chicken or egg" problem of multiple scattering in the presence of thermal emission/absorption.

Using Eq. (4.28) the Lambda Iteration process can be mathematically expressed as:

$$S_\nu^{n+1} = \epsilon_\nu B_\nu(T) + (1 - \epsilon_\nu)\Lambda[S_\nu^n] \tag{4.29}$$

where n is the iteration counter. We have now arrived at a mathematically clean way of formulating the Lambda Iteration process.

4.4.2 Worked-out example: The two-stream approximation for 1-D problems

Although the topic of developing accurate and efficient Lambda Operator subroutines will be deferred to a later Chapter, it is useful to work out a very simple example. Consider a 1-D plane parallel atmosphere (see Section 3.7). We put gridpoints at positions $z_{1/2}, \cdots, z_{N_z+1/2}$, i.e. we have $N_z + 1$ gridpoints. For the angle μ we take just *two* gridpoints:

$$\mu_- = -\frac{1}{\sqrt{3}} \qquad \text{and} \qquad \mu_+ = +\frac{1}{\sqrt{3}} \tag{4.30}$$

This is called the *two-stream approximation*. Of course, this two-point sampling of the entire angular span of $-1 \le \mu \le +1$ is extremely sparse, and one might be worried that this will cause this Lambda Operator to be very inaccurate. However, it turns out that the two-stream approximation is surprisingly accurate! In Section 4.5 we will find out why, and what the reason for the strange numbers $\pm 1/\sqrt{3}$ is.

We can now construct a "full" Lambda Operator subroutine in the following way. We first integrate the formal transfer equation of a single ray (Eq. 3.22) with $\mu = \mu_+ = 1/\sqrt{3}$ from the bottom of the atmosphere to the top. While doing so, we store the intensity I_+ along the way at each grid point – call these $I_{+,i+1/2}$. Next we integrate the formal transfer equation of a single ray with $\mu = \mu_- = -1/\sqrt{3}$ from the top to the bottom. While doing that we store the intensity I_- at each grid point – call these $I_{-,i+1/2}$. We now loop over all grid points again and calculate

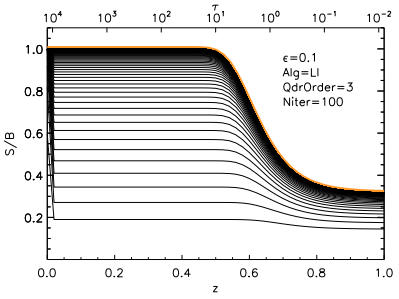
$$J_{i+1/2} = \frac{1}{2}\left(I_{-,1/2} + I_{+,1/2}\right) \tag{4.31}$$

That's it.

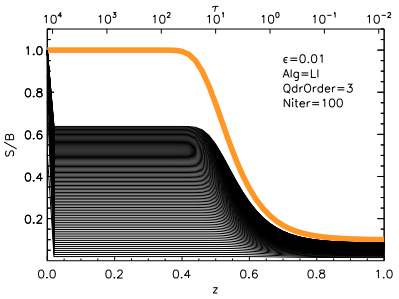
Now let us see how the Lambda Iteration method behaves in this two-stream approximation. Let us define the following dimensionless multiple scattering problem. We have a 1-D plane parallel atmosphere with vertical coordinate z ranging from $z = 0$ to $z = 1$. The total extinction coefficient $\alpha(z)$ is defined as

$$\alpha(z) = 10^{5-6z} \tag{4.32}$$

so that $\alpha(0) = 10^5$ and $\alpha(1) = 10^{-1}$, and in between it is an exponential decay. The Planck function is taken to be $B = 1$ everywhere. We have a constant photon destruction probability ϵ. At the $z = 0$ boundary we set $I_+ = J = B$ while at the $z = 1$ boundary we set $I_- = 0$, i.e. we have a free outflow of radiation and no inflow.



Let us see how the method works for the case $\epsilon = 10^{-1}$, i.e. not a very challenging problem. In the margin figure you can see the convergence. The thick red line is the converged solution. It shows that we clearly need $\gg 10$ iterations to get convergence, even for this rather simple test case.



Now let us apply the method to slightly nastier problem: $\epsilon = 10^{-2}$, which is still not nearly among the worst cases. Again the result is shown in the margin figure. You can see that after 100 iteration the solution has not even nearly converged yet. Many hundreds of iterations are, in fact, necessary. This demonstrates the weakness of Lambda Iteration. Clearly we must find ways to improve the convergence. These methods will be discussed below: ALI and Ng acceleration.

4.4.3 Lambda Operator as a matrix

For what follows, it will be useful to go one level further in the mathematical abstraction of the concept of Lambda Operator. Suppose we divide the cloud up in $N = N_x \times N_y \times N_z$ grid cells. Or almost[1] equivalently, we place $N = N_x \times N_y \times N_z$ grid

[1] As we shall see later, there is a fundamental difference between grid cells and grid points. For now we will brush over this difference, to not over-complicate things at this point.

points (sampling points) in the cloud. Let us give each grid point a unique ID number:

$$i = i_x + (i_y - 1)N_x + (i_z - 1)N_x N_y \tag{4.33}$$

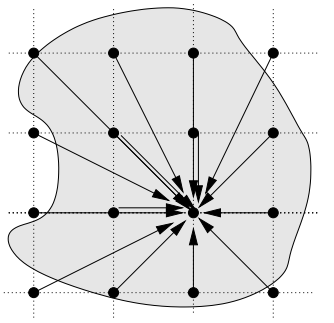
where $i_x \in [1, N_x]$, $i_y \in [1, N_y]$ and $i_z \in [1, N_z]$. The ID i is now $i \in [1, N]$.

The "partial" Lambda Operator for cell i is now a function of the source function S_j of all cells j (including $j = i$ itself). Moreover, it is a *linear* function, because the formal transfer equation is linear in the source function. We can thus formally write:

$$J_i = \Lambda_i[S] = \sum_{j=1}^{N} \Lambda_{ij} S_j \tag{4.34}$$

When we apply our Lambda Operator as a subroutine, we never really calculate the coefficients Λ_{ij} explicitly. We just integrate the transfer equation along rays and integrate the intensities over solid angle. However, deep down the mathematical truth is that we calculate the sum shown in Eq. (4.34).

One row of the 16x16 Lambda Operator matrix



If we regard the sequence of numbers $(S_1, \cdots, S_N)$ as a vector in an N-dimensional linear space, and we do the same for $(J_1, \cdots, J_N)$, then Λ_{ij} are the elements of an $N \times N$ matrix. It is the matrix coupling every grid point to every grid point. The act of applying the Lambda operator to S is then a matrix multiplication:

$$\begin{pmatrix} J_1 \\ \vdots \\ J_N \end{pmatrix} = \begin{pmatrix} \Lambda_{11} & \cdots & \Lambda_{1N} \\ \vdots & \ddots & \vdots \\ \Lambda_{N1} & \cdots & \Lambda_{NN} \end{pmatrix} \begin{pmatrix} S_1 \\ \vdots \\ S_N \end{pmatrix} \tag{4.35}$$

The "partial Lambda Operator" is one row of this matrix, while the "full Lambda Operator" is the complete matrix.

In most circumstances it is utterly unpractical to explicitly calculate the matrix elements of the Lambda Operator. Consider a 3-D radiative transfer problem on a $100 \times 100 \times 100$ grid. This means that $N = 10^6$ and that we have 10^{12} matrix elements. Just the storage of this matrix is already practically impossible, let alone doing any reasonable calculations with it.

The reason why we introduce the Lambda Operator Matrix is twofold: (1) it shows even better the formal mathematical nature of the operator, and the linear nature of Eq. (4.28), and (2) it will allow us to develop a powerful "boosted version" of Lambda Iteration called *approximate operator acceleration* or *accelerated lambda iteration*.

4.4.4 Accelerated Lambda Iteration

Suppose that we have an infinitely big and infinitely powerful computer, so that we do not have to worry about the size of the Λ_{ij} matrix, nor about the computing time to manipulate it. Let us revisit the radiative transfer equation in the form Eq. (4.28):

$$S_\nu = \epsilon_\nu B_\nu(T) + (1 - \epsilon_\nu)\Lambda[S_\nu] \tag{4.36}$$

We can write this in vector/matrix form (where we drop the index ν for notational convenience):

$$\mathbf{S} = \epsilon \mathbf{B} + (1 - \epsilon)\Lambda \mathbf{S} \tag{4.37}$$

The objective of the radiative transfer problem is to solve for $\mathbf{S}$. The classical Lambda Iteration scheme is to start with some guess $\mathbf{S}^0$, and then iterate

$$\mathbf{S}^{n+1} = \epsilon \mathbf{B} + (1 - \epsilon)\Lambda \mathbf{S}^n \tag{4.38}$$

until convergence. But if we have infinite computing power, we might as well solve Eq. (4.37) with brute force, using linear algebra. For that we rewrite Eq. (4.37) in the form:

$$[1 - (1 - \epsilon)\Lambda]\mathbf{S} = \epsilon \mathbf{B} \tag{4.39}$$

The left hand side is the product of a matrix M given by

$$M = [1 - (1 - \epsilon)\Lambda] \tag{4.40}$$

and the vector $\mathbf{S}$. Eq. (4.39) is thus the typical standard form of a matrix equation:

$$M\mathbf{S} = \epsilon\,\mathbf{B} \tag{4.41}$$

which is in fact a huge set of coupled linear equations. This can be solved by inverting the matrix:

$$\mathbf{S} = \epsilon\, M^{-1}\,\mathbf{B} \tag{4.42}$$

or by making use of standard software libraries for solving matrix equations to solve $\mathbf{S}$ straight from Eq. (4.39).

Of course this exercise is entirely hypothetical, because even for relatively small grids this would require huge computing time. But for understanding what follows it is important to understand this idea.

The idea behind *Accelerated Lambda Iteration* is to construct an *approximate operator* Λ^* that has the following properties:

- Λ^* should be easy to store in a computer
- The matrix equation $M^*\mathbf{S} = \epsilon\mathbf{B}$ with $M^* = 1 - (1 - \epsilon)\Lambda^*$ should be relatively easy to solve
- Λ^* should be an approximation of the full Λ
- At gridpoints i where it is difficult to assure that Λ^* is a sufficiently good approximation of Λ (for instance in optically thin regions), Λ^* should have the property of approaching $\Lambda^*_{ij} \to 0\ \forall\, j$.

Typically, in contrast to the full Lambda Operator Λ, the approximate operator Λ^* is *not* just a subroutine, but is in the form of truly tangible matrix elements. However, the matrix Λ^*_{ij} should be a *sparse matrix*, i.e. most of its elements should be zero, because otherwise we would need our "infinitely powerful computer" to handle it. We will discuss some typical choices of Λ^* in a minute, but let us first see how we can use such an approximate operator to speed up the convergence of the Lambda Iteration.

What we do is split the full Lambda Operator in two parts:

$$\Lambda = (\Lambda - \Lambda^*) + \Lambda^* \tag{4.43}$$

Inserting this into Eq. (4.37) yields

$$\mathbf{S} = \epsilon\mathbf{B} + (1 - \epsilon)(\Lambda - \Lambda^*)\mathbf{S} + (1 - \epsilon)\Lambda^*\mathbf{S} \tag{4.44}$$

We now bring only the easy-to-solve matrix to the other side, and leave the difficult-to-solve matrix on the right-hand side:

$$[1 - (1 - \epsilon)\Lambda^*]\mathbf{S} = \epsilon\mathbf{B} + (1 - \epsilon)(\Lambda - \Lambda^*)\mathbf{S} \tag{4.45}$$

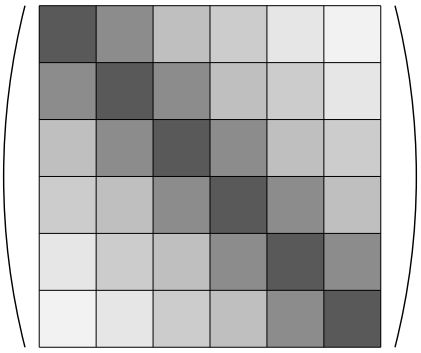
This still leaves a contribution of $\mathbf{S}$ on the right-hand side, so we cannot directly solve this. So we now re-introduce the Lambda Iteration scheme:

$$[1 - (1 - \epsilon)\Lambda^*]\mathbf{S}^{n+1} = \epsilon\mathbf{B} + (1 - \epsilon)(\Lambda - \Lambda^*)\mathbf{S}^n \tag{4.46}$$

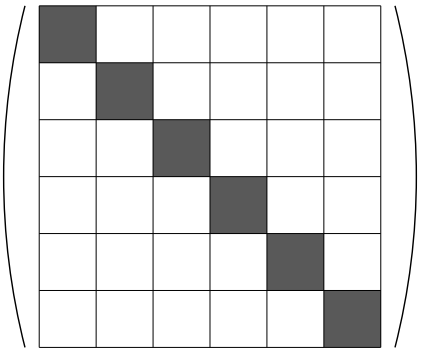
which is now so-to-speak a $(\Lambda - \Lambda^*)$-Iteration scheme. For each iteration step we solve the above matrix equation. Slightly more explicitly, we define the matrix

$$M^* = [1 - (1 - \epsilon)\Lambda^*] \tag{4.47}$$

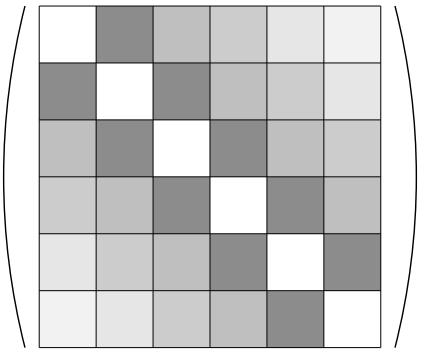
Full Lambda Operator matrix (6x6)



Local Approximate Operator



Λ–$\Lambda*$ for local approximate operator



and our matrix equation, to be solved at each iteration step, becomes

$$M^* \mathbf{S}^{n+1} = \epsilon \mathbf{B} + (1-\epsilon)(\Lambda - \Lambda^*)\mathbf{S}^n \tag{4.48}$$

Since M^* is a sparse matrix, it can be easily stored in a computer. And there are many libraries of subroutines to solve sparse matrix equations, even for vectors with millions of components. So this is a doable task. In each iteration step we must thus compute $\Lambda\,\mathbf{S}^n$, for which we use the Lambda Operator subroutine, and we must compute $\Lambda^*\mathbf{S}^n$, for which we use the sparse matrix Λ^*. We now iterate this procedure until convergence.

This method is called *approximate operator acceleration* of the Lambda Iteration procedure. The full iteration scheme is called *Accelerated Lambda Iteration*, or ALI. It was originally proposed by Cannon (1973, Astrophysical Journal, 185, 621).

It turns out that, if we make a clever choice of Λ^*, we can dramatically accelerate the convergence.

So: What constitutes a good choice of Λ^*? The simplest choice is:

$$\Lambda^* = \mathrm{diag}(\Lambda) \tag{4.49}$$

This is called a *local operator*, because it involves only the matrix elements that couple each grid point i with itself ($j = i$). The nice thing about such a local operator is that it it produces an M^* matrix that is trivial to invert, as it is a diagonal matrix. Loosely one can say that Λ^* is the part of the Λ operator that deals with photons that scatter twice or multiple times within the same cell. It is, so to speak, the cell "self-coupling". This may sound like a let-down: We do lots of effort to treat *non-local* transfer of radiation, and as an approximation of Λ we take the self-coupling part of the radiative transfer. However, it is this self-coupling at high optical depths (i.e. when the optical depth of a grid cell τ_i is much larger than 1) that causes the excessively slow convergence of Lambda Iteration. Loosely expressed: with Lambda Iteration we follow photons as they scatter hundreds of times *in a single cell*: Since we are not interested anyway in what a photon does at sub-cell resolution, this is a complete waste of computer time. Choosing $\Lambda^* = \mathrm{diag}(\Lambda)$ means that we extract this self-coupling from the iteration scheme and instead solve the self-coupling part analytically.

Let us investigate more concretely how this works. As we shall show later (Section 4.4.5), the expression for $\mathrm{diag}(\Lambda)$ in the limit that $\Delta\tau \gg 1$ is:

$$\mathrm{diag}(\Lambda) \simeq 1 - \frac{2}{\Delta\tau^2} \qquad (\text{for} \quad \Delta\tau \gg 1) \tag{4.50}$$

We take $\Lambda^* = \mathrm{diag}(\Lambda)$. With this, the matrix M^* (Eq. 4.47) becomes

$$M^* = 1 - (1-\epsilon)\Lambda^* = 1 - (1-\epsilon)\left(1 - \frac{2}{\Delta\tau^2}\right) \simeq \epsilon + \frac{2}{\Delta\tau^2} \tag{4.51}$$

for $\Delta\tau \gg 1$ and $\epsilon \ll 1$. Eq. (4.48) then becomes

$$\mathbf{S}^{n+1} = \left(\epsilon + \frac{2}{\Delta\tau^2}\right)^{-1} \left(\epsilon \mathbf{B} + (1-\epsilon)(\Lambda - \Lambda^*)\mathbf{S}^n\right) \tag{4.52}$$

which is a factor of $1/(\epsilon + 2/\Delta\tau^2)$ speed-up in convergence, which can be a pretty large boosting factor.

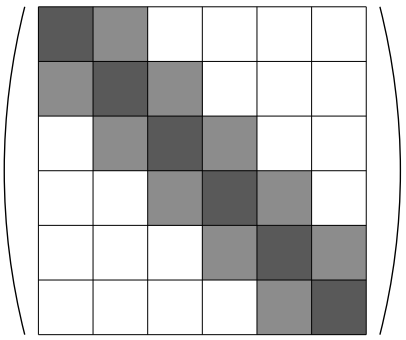
The remaining operator, $(\Lambda - \Lambda^*)$ transports photons over distances of *at least* one grid spacing. We thus no longer waste time following the photon's sub-cell scatterings: we can concentrate our iteration scheme on the real deal: transport over large distances. This is why the ALI method converges much faster than classical Lambda Iteration.

The local approximate operator method dates back to Olson, Auer & Buchler (1986, J. Quant. Spectros. Radiat. Transfer 35, 431), and is often referred to as the OAB method.

There is, however, a drawback of the local approximate operator: it becomes less effective the finer the grid spacing is. This can be easily understood. The smaller the grid cell for the same physical object, the smaller the optical depth of each cell, while the total optical depth of the object stays the same (we simply have more grid cells). Since the boosting factor $1/(\epsilon + 2/\Delta\tau^2)$ becomes smaller as $\Delta\tau$ becomes smaller, the ALI method becomes less effective.

To overcome this drawback one can introduce a *tridiagonal operator*, where not only the self-coupling of each gridpoint is included, but also the coupling to the direct neighbors. This kind of approximate operator tends to overcome the problem of the grid cell size, because it essentially reduces to the diffusion operator for high optical depth (see Section 4.5 for the diffusion approximation). It does, however require the use of sparse matrix equation solver libraries (see Section 4.7).

Tridiagonal Approx Operator (1–D RT)



4.4.5 Worked-out example: ALI for two-stream transfer in 1-D

Let us work out an example in a 1-D plane-parallel atmosphere using the two-stream approximation (see Sections 3.7 and 4.4.2). Let us compute the diagonal of the Lambda Operator. Since we do not have the Lambda Operator in matrix form, we have to ask ourselves the question: how does each gridpoint affect itself? We have to write down the transfer equation in exactly the way our integration algorithm integrates it.

As we will find out soon (see Section 4.4.8 below), we must use the third-order quadrature formula (Section 3.8.4), otherwise the converged solution will be wrong. Let us focus on grid point $i + 1/2$, and its neighbors $i - 1/2$ and $i + 3/2$. We get (using Eqs. 3.38, 3.39):

$$I_{+,i+1/2} = I_{+,i-1/2}\, e^{-\Delta\tau_i} + u_{+,i+1/2} S_{i-1/2} + v_{+,i+1/2} S_{i+1/2} + w_{+,i+1/2} S_{i+3/2} \quad (4.53)$$

$$I_{-,i+1/2} = I_{-,i+3/2}\, e^{-\Delta\tau_{i+1}} + u_{-,i+1/2} S_{i+3/2} + v_{-,i+1/2} S_{i+1/2} + w_{-,i+1/2} S_{i-1/2} \quad (4.54)$$

where

$$\Delta\tau_i = \sqrt{3}\,\alpha_i\,(z_{i+1/2} - z_{i-1/2}) \quad (4.55)$$

(and likewise for $\Delta\tau_{i+1}$). From this we can compute $J_{i+1/2}$:

$$\begin{aligned} J_{i+1/2} &= \frac{1}{2}\,(I_{-,i+1/2} + I_{+,i+1/2}) \\ &= \frac{1}{2} I_{+,i-1/2}\, e^{-\Delta\tau_i} + \frac{1}{2} I_{-,i+3/2}\, e^{-\Delta\tau_{i+1}} + (u_{+,i+1/2} + w_{-,i+1/2}) S_{i-1/2} \\ &\quad + (v_{+,i+1/2} + v_{-,i+1/2}) S_{i+1/2} + (w_{+,i+1/2} + u_{-,i+1/2}) S_{i+3/2} \end{aligned} \quad (4.56)$$

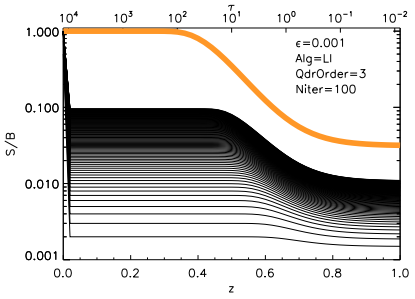
In principle we should now continue to work out $I_{+,i-1/2}$ in terms of $I_{+,i-3/2}$ and $I_{-,i+3/2}$ in terms of $I_{-,i+5/2}$. But for high optical depth ($\Delta\tau \gg 1$) these extra terms will vanish, as they will be proportional to $e^{-\Delta\tau}$. And since we anyway need Λ^* only for such high-$\Delta\tau$ cases, this omission will not be problematic at all. So let us omit these terms, and thus make the computation of the Λ^* substantially easier:

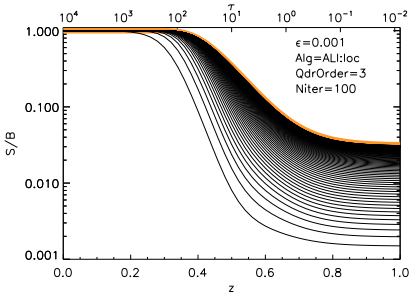
$$\Lambda^*_{i+1/2,i-1/2} = \tfrac{1}{2}(u_{+,i+1/2} + w_{-,i+1/2}) \quad (4.57)$$

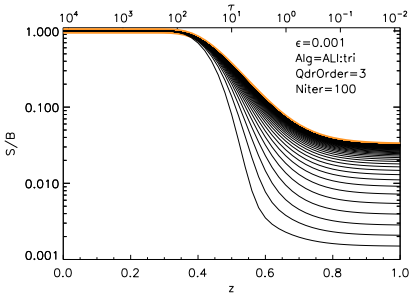
$$\Lambda^*_{i+1/2,i+1/2} = \tfrac{1}{2}(v_{+,i+1/2} + v_{-,i+1/2}) \quad (4.58)$$

$$\Lambda^*_{i+1/2,i+3/2} = \tfrac{1}{2}(w_{+,i+1/2} + u_{-,i+1/2}) \quad (4.59)$$

and all other elements of Λ^* are zero. This would be our full tridiagonal approximate operator. If we put $\Lambda^*_{i+1/2,i-1/2} = 0$ and $\Lambda^*_{i+1/2,i+3/2} = 0$ we obtain our local approximate operator. Since we calculate $u_{+,i+1/2}$, $v_{+,i+1/2}$ and $w_{+,i+1/2}$ anyway during our upward integration, and $u_{-,i+1/2}$, $v_{-,i+1/2}$ and $w_{-,i+1/2}$ during our downward integration, it is easy to construct these matrix elements.







Most ALI-based codes use local operators, because diagonal matrices are trivial to invert. If we include the off-diagonal elements the solution of Eq. (4.48) at each iteration step becomes a non-local problem. This is more challenging.

In the margin you see results for our standard test problem (see Section 4.4.2), this time with $\epsilon = 10^{-3}$. You can see that, while Lambda Iteration (LI) goes nowhere, ALI with a local approximate operator does a good job and ALI with a tridiagonal approximate operator converges even faster.

Let us now study the behavior of our approximate operator for $\Delta\tau \to \infty$. Let us, for convenience, assume that $\Delta\tau_i = \Delta\tau_{i+1}$ and call this $\Delta\tau$. With $e^{-\Delta\tau} \to 0$, Eqs. (3.43, 3.44, 3.45) become:

$$e_0 \to 1 \quad , \quad e_1 \to \Delta\tau - 1 \quad , \quad e_2 \to \Delta\tau^2 - 2\Delta\tau + 2 \tag{4.60}$$

and Eqs. (3.40, 3.41, 3.42) become:

$$u = e_0 + \frac{e_2 - 3\Delta\tau e_1}{2\Delta\tau^2} = \frac{1}{\Delta\tau^2} + \frac{1}{2\Delta\tau} \tag{4.61}$$

$$v = \frac{2\Delta\tau e_1 - e_2}{\Delta\tau^2} = 1 - \frac{2}{\Delta\tau^2} \tag{4.62}$$

$$w = \frac{e_2 - \Delta\tau e_1}{2\Delta\tau^2} = \frac{1}{\Delta\tau^2} - \frac{1}{2\Delta\tau} \tag{4.63}$$

We thus get

$$\Lambda^*_{i+1/2,i-1/2} = \frac{1}{\Delta\tau^2} \tag{4.64}$$

$$\Lambda^*_{i+1/2,i+1/2} = 1 - \frac{2}{\Delta\tau^2} \tag{4.65}$$

$$\Lambda^*_{i+1/2,i+3/2} = \frac{1}{\Delta\tau^2} \tag{4.66}$$

The matrix M^* (Eq. 4.47) thus becomes

$$M^*_{i+1/2,i-1/2} = -(1-\epsilon)\frac{1}{\Delta\tau^2} \tag{4.67}$$

$$M^*_{i+1/2,i+1/2} = 1 - (1-\epsilon)\left(1 - \frac{2}{\Delta\tau^2}\right) \tag{4.68}$$

$$M^*_{i+1/2,i+3/2} = -(1-\epsilon)\frac{1}{\Delta\tau^2} \tag{4.69}$$

If we take $\epsilon \to 0$, then this becomes

$$M^*_{i+1/2,i-1/2} = -\frac{1}{\Delta\tau^2} \tag{4.70}$$

$$M^*_{i+1/2,i+1/2} = \epsilon + \frac{2}{\Delta\tau^2} \tag{4.71}$$

$$M^*_{i+1/2,i+3/2} = -\frac{1}{\Delta\tau^2} \tag{4.72}$$

We will see in Section 4.5 that the part proportional to $1/\Delta\tau^2$ is in fact mathematically the same as a diffusion operator. In other words, the reason why a tridiagonal approximate operator works so well in the extremely high optical depth limit is because, in this limit, it actually treats the radiative transfer as a diffusion problem, which is indeed the correct thing to do in the optically thick limit. In other words: ALI with a tridiagonal operator splits the problem into a diffusion part and the long-range part. The diffusion part is solved directly while the long-range part is solved iteratively.

Inverting a tridiagonal matrix, or better, solving a tridiagonal matrix equation, is not trivial. We will discuss standard numerical methods for this in Section 4.7.

4.4.6 Linear convergence of LI and ALI algorithms

Although ALI converges already substantially faster than classical LI in optically thick problems, both ALI and LI algorithms are linearly converging algorithms. The danger of such kind of convergence is that it might lead to progressively slower convergence as the iteration sequence proceeds, and that it might be hard to know if we have converged or not.

This can be understood by looking at the iteration scheme as a successive application of a matrix multiplication. Let us focus on the Lambda Iteration sequence (Eq. 4.38),

$$\mathbf{S}^{n+1} = \epsilon\mathbf{B} + (1-\epsilon)\Lambda\mathbf{S}^n \tag{4.73}$$

and slightly rewrite it into the form:

$$\mathbf{S}^{n+1} = \mathbf{A} + \Psi\mathbf{S}^n \tag{4.74}$$

with the vector $\mathbf{A} = \epsilon\mathbf{B}$ and the matrix Ψ defined as $\Psi = (1-\epsilon)\Lambda$. We can also bring the ALI iteration scheme (Eq. 4.48) into the form of Eq. (4.74) by defining the vector $\mathbf{A}$ as $\mathbf{A} = \epsilon(M^*)^{-1}\mathbf{B}$ and the matrix Ψ as $\Psi = (1-\epsilon)(M^*)^{-1}(\Lambda - \Lambda^*)$. Eq. (4.74) is therefore the general form of the LI and ALI iteration schemes.

Suppose now that $\mathbf{S}$ is the actual solution we wish to find:

$$\mathbf{S} = \lim_{n\to\infty} \mathbf{S}^n \tag{4.75}$$

This means that $\mathbf{S}$ obeys

$$\mathbf{S} = \mathbf{A} + \Psi\mathbf{S} \tag{4.76}$$

We can then write Eq. (4.74) into the form

$$(\mathbf{S}^{n+1} - \mathbf{S}) = \Psi(\mathbf{S}^n - \mathbf{S}) \tag{4.77}$$

Since we do not know $\mathbf{S}$ in advance, this equation has no practical use. But it does give us mathemtical insight: it shows us that the convergence is essentially a successive application of the matrix Ψ:

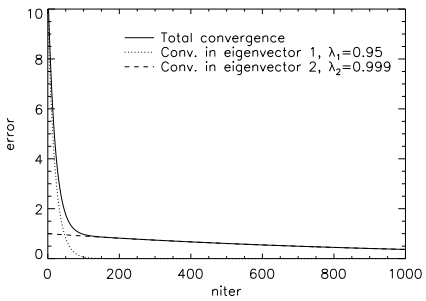
$$(\mathbf{S}^n - \mathbf{S}) = \Psi^n(\mathbf{S}^0 - \mathbf{S}) \tag{4.78}$$

where $\mathbf{S}^0$ is the initial guess of the iteration procedure. Eq. (4.78) is the mathematical way of expressing *linear convergence*.

Now, the matrix Ψ has a set of Eigenvalues λ_i. Fast convergence is achieved when all $\lambda_i \ll 1$. If only *one* or more of these eigenvalues is very close to 1, then the convergence can be extremely slow. The number of iterations required would be

$$N_{\rm iter} \gg \frac{1}{\min(1-\lambda_i)} \tag{4.79}$$

So here is the reason why linear convergence can be treacherous: Initially convergence may be rapid, because one sees the rapid convergence along the eigenvectors belonging to λ_i that are $\ll 1$. As we continue the iteration the solution may thus be already converged along those eigenvectors, but not along eigenvectors with $\lambda_i \simeq 1$. So the slowing down of changes in S^{n+1} might not be due to real convergence, but due to the last remaining non-converged eigenvectors having eigenvalues *very* close to 1. To take a trivial and extreme example: Suppose our matrix Ψ has only two distinct eigenvalues: $\lambda_1 = 0.1$ and $\lambda_2 = 0.999999$. After 4 iterations we may be tempted to call the solution converged, because $|\mathbf{S}^4 - \mathbf{S}^3| \lesssim 10^{-4}$. However, in reality we need at least 10^6 iterations to get convergence because $1/(\lambda_2 - 1) = 10^6$. This phenomenon is called *false convergence*.



What we should learn from this is that there is always a bit of experience required to know if a problem has converged or not. What may help you to make this judgement is to use the estimation we made in Section 4.4 which, for LI reads

$$N_{\rm iter} \gg \min\left(\tau^2, \frac{1}{\epsilon}\right) \tag{4.80}$$

and for ALI with a local operator reads

$$N_{\rm iter} \gg \min\left(\tau^2, N_x^2, N_y^2, N_z^2, \frac{1}{\epsilon}\right) \tag{4.81}$$

where N_x, N_y and N_z are the number of grid points in the three directions.

4.4.7 Ng-acceleration

Although ALI already helps dramatically in speeding up convergence, there is a very simple method to speed it up even more. Suppose we have done three iteration steps: starting from $\mathbf{S}^{n-3}$, to $\mathbf{S}^{n-2}$, $\mathbf{S}^{n-1}$ and $\mathbf{S}^{n}$, and now we want to find the next one: $\mathbf{S}^{n+1}$. Instead of doing another (A)LI step we can use the sequence $\mathbf{S}^{n-3}$, $\mathbf{S}^{n-2}$, $\mathbf{S}^{n-1}$ and $\mathbf{S}^{n}$ to try to predict where this is going.

The tips of the three vectors $\mathbf{S}^{n-3}$, $\mathbf{S}^{n-2}$ and $\mathbf{S}^{n-1}$ span up a 2-dimensional surface in the linear source function space. This surface is defined by all $\mathbf{S}^*$ that obey

$$\mathbf{S}^* = (1-a-b)\mathbf{S}^{n-1} + a\mathbf{S}^{n-2} + b\mathbf{S}^{n-3} \tag{4.82}$$

for arbitrary a and b. We ask now ourselves, can we find a value for a and b such that if we insert $\mathbf{S}^*$ into the iteration equation Eq. (4.74)

$$\begin{aligned}\mathbf{S}^{**} &= \mathbf{A} + \Psi\mathbf{S}^* \\ &= (1-a-b)\mathbf{S}^{n} + a\mathbf{S}^{n-1} + b\mathbf{S}^{n-2}\end{aligned} \tag{4.83}$$

the magnitude-squared of the difference between $\mathbf{S}^*$ and its next iteration $\mathbf{S}^{**}$

$$|\mathbf{S}^{**} - \mathbf{S}^{*}|^2 \tag{4.84}$$

is minimal? Note that we introduced a norm here, which is defined as

$$|\mathbf{V}|^2 = \sum_i W_i V_i^2 \tag{4.85}$$

where W_i are weight coefficients that you can choose as you see fit. Likewise the inner product is defined as

$$\mathbf{U}\cdot\mathbf{V} = \sum_i W_i U_i V_i \tag{4.86}$$

It turns out that if you go through the algebra of minimizing $|\mathbf{S}^{**} - \mathbf{S}^{*}|^2$ you find that there is indeed a minimum: The corresponding values of a and b are:

$$a = \frac{C_1B_2 - C_2B_1}{A_1B_2 - A_2B_1} \quad , \quad b = \frac{C_2A_1 - C_1A_2}{A_1B_2 - A_2B_1} \tag{4.87}$$

where

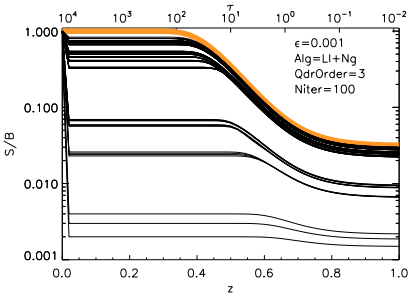
$$\begin{aligned} A_1 &= \mathbf{Q}_1\cdot\mathbf{Q}_1 \quad , \quad B_1 = \mathbf{Q}_1\cdot\mathbf{Q}_2 \quad , \quad C_1 = \mathbf{Q}_1\cdot\mathbf{Q}_3 \\ A_2 &= \mathbf{Q}_2\cdot\mathbf{Q}_1 \quad , \quad B_2 = \mathbf{Q}_2\cdot\mathbf{Q}_2 \quad , \quad C_2 = \mathbf{Q}_2\cdot\mathbf{Q}_3 \end{aligned} \tag{4.88}$$

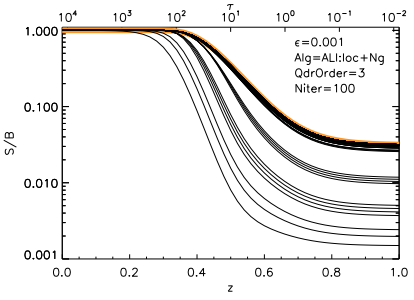
with (finally):

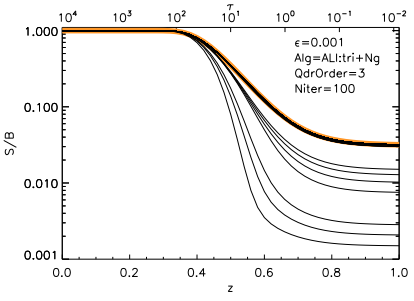
$$\mathbf{Q}_1 = \mathbf{S}^n - 2\mathbf{S}^{n-1} + \mathbf{S}^{n-2} \tag{4.89}$$

$$\mathbf{Q}_2 = \mathbf{S}^n - \mathbf{S}^{n-1} - \mathbf{S}^{n-2} + \mathbf{S}^{n-3} \tag{4.90}$$

$$\mathbf{Q}_3 = \mathbf{S}^n - \mathbf{S}^{n-1} \tag{4.91}$$







For the next iteration step we now choose:

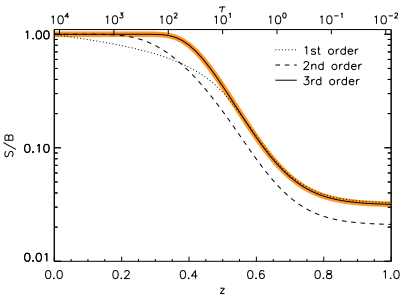
$$\mathbf{S}^{n+1} = (1 - a - b)\mathbf{S}^n + a\mathbf{S}^{n-1} + b\mathbf{S}^{n-2} \tag{4.92}$$

It turns out that this method dramatically improves the convergence! This is called *Ng-acceleration*, named after the author of the first paper detailing this method: Ng (1974, J. Chem. Phys. 61, 2680). This Ng acceleration method is in fact applicable to any linearly converging problem.

Note that before we apply this technique again we must do three "normal" iteration steps again, otherwise not enough information is present for this method to work. Also it is essential to apply Ng acceleration to the *entire* vector $\mathbf{S}$, not to each gridpoint separately.

In the margin you see results for our standard test problem (see Section 4.4.2) with $\epsilon = 10^{-3}$. They are exactly the same as the examples in the margin figures in Section 4.4.5, but now with Ng-acceleration added.

4.4.8 The need for the 3rd order quadrature



We already alluded to it before: that it is important to use the 3rd order integration formula of Section 3.8.4, and not the 2nd or 1st order ones. So let us find out for ourselves if this is indeed necessary. We take again our standard test setup (Section 4.4.2), but this time we set $\epsilon = 10^{-5}$. In the margin figure you can see the result for 1st, 2nd and 3rd order integration, compared to the real solution. You can clearly see that the 1st and 2nd order integration give wrong results. They are fully converged solutions, so it is not a matter of false convergence. It is because the 1st and 2nd order integration do not have the right properties for use in (A)LI schemes at very high optical depth. They do not reproduce the diffusion regime and they do not conserve radiative flux properly. Therefore energy is "lost" as it tries to diffuse upward.

You can also convince yourself of the need of 3rd order integration by working out the $\Lambda_{i+1/2,i-1/2}$, $\Lambda_{i+1/2,i+1/2}$ and $\Lambda_{i+1/2,i+3/2}$ for 1st and 2nd order quadrature. You will find that in the diffusion limit $\Delta\tau \to \infty$ the correct diffusion operator (Eqs. 4.64, 4.65, 4.66) is not reproduced. It is therefore impossible for such an operator to find the correct solution at high optical depth.

4.4.9 Alternative to 3rd order quadrature: Feautrier's method

The main reason to use 3rd order quadrature is that the source function S is described as a 2nd order polynomial, which we have shown to be important to get the diffusion right. An other method that ensures this is Feautrier's method, which is very popular in the stellar atmospheres radiative transfer community. In this method we do not integrate in one direction along a ray, but we integrate in both directions simultaneously. Suppose $I_{(+)}$ is the intensity pointing in positive s direction and $I_{(-)}$ pointing in negative s direction. The formal transfer equations are:

$$\frac{dI_{(+)}}{ds} = \alpha(S - I_{(+)}) \tag{4.93}$$

$$\frac{dI_{(-)}}{ds} = -\alpha(S - I_{(-)}) \tag{4.94}$$

We can define

$$I_e = \frac{1}{2}\left[I_{(+)} + I_{(-)}\right] \tag{4.95}$$

$$I_o = \frac{1}{2}\left[I_{(+)} - I_{(-)}\right] \tag{4.96}$$

This leads to

$$\frac{dI_e}{ds} = -\alpha I_o \tag{4.97}$$

$$\frac{dI_o}{ds} = -\alpha(I_e - S) \tag{4.98}$$

These two can be combined to

$$\frac{1}{\alpha}\frac{d}{ds}\left(\frac{1}{\alpha}\frac{dI_e}{ds}\right) = I_e - S \tag{4.99}$$

If we define τ along the ray such that $d\tau = \pm\alpha ds$ (you can choose the sign as you wish), then we obtain

$$\frac{d^2 I_e}{d\tau^2} = I_e - S \tag{4.100}$$

This is a *two-point boundary value problem*. As boundary conditions we put $I_{(+)} = 0$ at the small-s side (left side) and $I_{(-)} = 0$ at the large-s side (right side). This translates, with the above equations, into the following boundary conditions:

$$\frac{dI_e}{ds} = \alpha I_e \qquad \text{(for left boundary)} \tag{4.101}$$

$$\frac{dI_e}{ds} = -\alpha I_e \qquad \text{(for right boundary)} \tag{4.102}$$

Eq. (4.100) and its boundary conditions is very similar to the 1-D diffusion equation we will discuss in Section 4.5.4. This is the reason that Feaurtrier's method can reproduce the diffusion limit so well. The equations are written into a form that is similar to the diffusion equation, *while still retaining the full angle-dependence of the radiation field*. The method of solution is identical to the one we will discuss in Section 4.5.4. A tiny preview, in case you can't wait: It basically involves translating Eq. (4.100) into the form of a matrix equation, and using a special procedure to solve it. But let us defer this discussion to Section 4.5.4.

4.5 The moment equations of radiative transfer

So far we have always explicitly treated the angular dependence of the radiation field by following photon paths (in the Monte Carlo method) or by solving the tranfer equation along a fixed set of angles (μ_i, ϕ_i) (in discrete ordinate methods). But as we already discussed in Section 2.5, we can also use the angular moments of the radiation field at every given point. This is perhaps the more elegant way of reducing the angular resolution than simply choosing a limited number of discrete directions (μ_i, ϕ_i).

4.5.1 Deriving the first two moment equations

To remind you, the first three moments are (see Section 2.5):

$$J_\nu = \frac{1}{4\pi}\oint I_\nu(\mathbf{n})\, d\Omega \tag{4.103}$$

$$\mathbf{H}_\nu = \frac{1}{4\pi}\oint I_\nu(\mathbf{n})\, \mathbf{n}\, d\Omega \tag{4.104}$$

$$\mathcal{K}_\nu = \frac{1}{4\pi}\oint I_\nu(\mathbf{n})\, \mathbf{n}\,\mathbf{n}\, d\Omega \tag{4.105}$$

and we could go on to ever higher-rank tensors, but the first three are the most important ones. Also to remind you, the formal radiative transfer equation is (see Section 3.2):

$$\mathbf{n}\cdot\nabla I_\nu(\mathbf{x},\mathbf{n}) = j_\nu(\mathbf{x}) - \alpha_\nu(\mathbf{x}) I_\nu(\mathbf{x},\mathbf{n}) \tag{4.106}$$

If we, at some given point $\mathbf{x}$, integrate Eq. (4.106) over 4π angle, and divide by 4π, we obtain

$$\frac{1}{4\pi}\nabla \cdot \oint I_\nu(\mathbf{x},\mathbf{n})\,\mathbf{n}\,d\Omega = \frac{1}{4\pi}\oint j_\nu(\mathbf{x})\,d\Omega - \alpha_\nu(\mathbf{x})\frac{1}{4\pi}\oint I_\nu(\mathbf{x},\mathbf{n})d\Omega \tag{4.107}$$

We recognize $\mathbf{H}_\nu$ on the left hand side and $\mathbf{J}_\nu$ on the right hand side:

$$\nabla \cdot \mathbf{H}_\nu(\mathbf{x}) = j_\nu(\mathbf{x}) - \alpha_\nu(\mathbf{x})J_\nu(\mathbf{x}) \tag{4.108}$$

This is the first moment equation. It says that if the emission exceeds the absorption, the divergence of the flux must be positive: a very physically meaningful statement! However, Eq. (4.108) is not a "closed set of equations", because it is 1 equation for 4 unknowns: J_ν, $H_{x,\nu}$, $H_{y,\nu}$, $H_{z,\nu}$.

Now let us go back to Eq. (4.106) and multiply it by the directional vector $\mathbf{n}$, and *then* integrate over 4π angle and divide by 4π. We obtain

$$\frac{1}{4\pi}\nabla \cdot \oint I_\nu(\mathbf{x},\mathbf{n})\,\mathbf{n}\,\mathbf{n}\,d\Omega = \frac{1}{4\pi}\oint j_\nu(\mathbf{x})\,\mathbf{n}\,d\Omega - \alpha_\nu(\mathbf{x})\frac{1}{4\pi}\oint I_\nu(\mathbf{x},\mathbf{n})\,\mathbf{n}\,d\Omega \tag{4.109}$$

Note that $\mathbf{n}\,\mathbf{n}$ is a second rank tensor. Since j_ν has no angle-dependency, and since

$$\oint \mathbf{n}\,d\Omega = 0 \tag{4.110}$$

the first term on the right hand side of Eq. (4.109) vanishes. Furthermore we recognize $\mathcal{K}$ on the left-hand side and $\mathbf{H}$ on the right-hand side:

$$\nabla \cdot \mathcal{K}_\nu(\mathbf{x}) = -\alpha_\nu(\mathbf{x})\mathbf{H}_\nu(\mathbf{x}) \tag{4.111}$$

This is the second moment equation. It is a vectorial equation, so it can be regarded as three equations. Together, Eqs. (4.108, 4.111) are 4 equations for 10 unkowns: J_ν, $H_{x,\nu}$, $H_{y,\nu}$, $H_{z,\nu}$, $K_{xx,\nu}$, $K_{xy,\nu}$, $K_{xz,\nu}$, $K_{yy,\nu}$, $K_{yz,\nu}$, $K_{zz,\nu}$ (the other components of K are, by symmetry of the tensor, not independent). Again our set of equations is not closed.

We could continue to ever higher-rank tensor moments, but we will always end up with more unknowns than equations, i.e. a non-closed system. In principle, if we go to infinte-rank tensor moments, we get a system of equations that is mathematically equivalent to, and equally complete as, the full angle-dependent radiative transfer equations. It is very similar (and in fact mathematically equivalent) to doing a spherical harmonic expansion: If we go to infinitely high l, m we get essentially a complete description of a function on a sphere.

One of the main advantages of the moment equations is that they are manifestly energy conservative. This is the reason that this form of the radiative transfer equations (and related methods such as the diffusion approximation) is typically used in radiation hydrodynamics calculations.

4.5.2 Eddington Approximation (Diffusion approximation)

In practice, however, we must truncate the system somewhere. In other words: we must introduce a *closure equation*. This can either be an assumption about the angular dependence of the radiation field, or we can actually calculate this angular dependence. Let us first discuss the first option. The simplest assumption one can make is the *Eddington approximation*:

$$K_{ij,\nu} = \frac{1}{3}\delta_{ij}J_\nu \tag{4.112}$$

where δ_{ij} is the Kronecker-delta. This form of the $\mathcal{K}$ tensor is valid for an isotropic radiation field (exercise: prove this, starting from Eq. 4.105), and is in fact also a reasonably good approximation when the radiation field has a small deviation from

isotropy. Mathematically the Eddington approximation is equivalent to truncating the spherical harmonics expansion of the radiation field at $l = 2$, i.e. putting all components with $l \geq 2$ to zero and retaining only the components $(l, m) = (0, 0)$, $(1, -1)$, $(1, 0)$, $(1, +1)$.

If we insert Eq. (4.112) into Eq. (4.111) we obtain

$$\frac{1}{3}\nabla J_\nu(\mathbf{x}) = -\alpha_\nu(\mathbf{x})\mathbf{H}_\nu(\mathbf{x}) \tag{4.113}$$

Now Eqs. (4.108, 4.113) form a closed set: four equations with four unknowns. Interestingly, we can even reduce this to one equation with one unknown by dividing Eq. (4.113) by $\alpha_\nu(\mathbf{x})$ and taking the divergence on both sides:

$$\frac{1}{3}\nabla \cdot \left(\frac{1}{\alpha_\nu(\mathbf{x})}\nabla J_\nu(\mathbf{x})\right) = -\nabla \cdot \mathbf{H}_\nu(\mathbf{x}) \tag{4.114}$$

Now the right-hand side of this equation is (apart from a minus sign) identical to the left-hand side of the first moment equation, Eq. (4.108). This means that with Eq. (4.114) we can rewrite Eq. (4.108) as

$$\frac{1}{3}\nabla \cdot \left(\frac{1}{\alpha_\nu(\mathbf{x})}\nabla J_\nu(\mathbf{x})\right) = \alpha_\nu(\mathbf{x})J_\nu(\mathbf{x}) - j_\nu(\mathbf{x}) \tag{4.115}$$

This is a second-order partial differential equation (PDE). We have thus created one second order PDE out of two coupled first order PDEs. This is the equation for the *diffusion approximation of radiative transfer*. The diffusion coefficient is $1/\alpha_\nu(\mathbf{x})$. We will discuss numerical methods for solving diffusion equations in Section 4.7.

We can see from the second moment equation in the form Eq. (4.113) that the radiative flux is proportional to the gradient in the mean intensity:

$$\mathbf{F}_\nu(\mathbf{x}) = -\frac{4\pi}{3\alpha_\nu(\mathbf{x})}\nabla J_\nu(\mathbf{x}) \tag{4.116}$$

which is exactly what you would indeed expect for diffusion.

Let us apply the diffusion approximation equation (Eq. 4.115) to the mixed scattering and thermal absorption/emission problem (see Eq. 4.19). So let us first write Eq. (4.115) in the form with a source function (omitting, for notational convenience, the $(\mathbf{x})$):

$$\frac{1}{3}\nabla \cdot \left(\frac{1}{\alpha_\nu}\nabla J_\nu\right) = \alpha_\nu[J_\nu - S_\nu] \tag{4.117}$$

Then we insert Eq. 4.19, and we obtain:

$$\frac{1}{3}\nabla \cdot \left(\frac{1}{\alpha_\nu}\nabla J_\nu\right) = \alpha_\nu^{\rm abs}[J_\nu - B_\nu(T)] \tag{4.118}$$

What we see here is that the scattering has completely vanished from the right hand side. It is only still present in the left hand side, because $\alpha_\nu = \alpha_\nu^{\rm abs} + \alpha_\nu^{\rm scat}$.

4.5.3 Variable Eddington Tensor method

One of the main advantages of the diffusion approximation is that it reduces the entire radiative transfer problem to an elliptic scalar partial differential equation, for which standard efficient numerical solution techniques exist (Section 4.7). It completely makes LI, ALI iteration or Monte Carlo modeling unnecessary. A diffusion solution is not just a solution to the formal transfer equation, but to the full transfer equation. The drawback is that the diffusion approximation is, well, an approximation.

Can we combine the accuracy of the full angular-dependent radiation transfer methods with the numerical efficiency of diffusion-type methods? The answer is: in many cases yes.

The *Variable Eddington Tensor* (VET) method is one such approach[2]. The idea is to write

$$K_{ij,\nu} = f_{ij,\nu} J_\nu \tag{4.119}$$

so that $f_{ij,\nu}$ is a dimensionless version of the $\mathcal{K}$ tensor. This $f_{ij,\nu}$ is the variable Eddington tensor. Putting it to $f_{ij,\nu} = (1/3)\delta_{ij}$ is the Eddington approximation. But let us not do that for now, and try to find an equation similar to the diffusion equation (Eq. 4.115) but now with $f_{ij,\nu}$ kept arbitrary. We get:

$$\nabla_i \cdot \left(\frac{1}{\alpha_\nu(\mathbf{x})} \nabla_j \Big[f_{ij,\nu}(\mathbf{x}) J_\nu(\mathbf{x}) \Big] \right) = \alpha_\nu(\mathbf{x}) J_\nu(\mathbf{x}) - j_\nu(\mathbf{x}) \tag{4.120}$$

where $\nabla_i = \partial/\partial_i$. When solving Eq. (4.120) for $J_\nu(\mathbf{x})$ we keep $f_{ij,\nu}(\mathbf{x})$ fixed. This equation is an elliptic equation just like Eq. (4.115), and can be solved using the same techniques as those used for solving diffusion-type equations. If we apply this to the absorption/scattering case we obtain

$$\nabla_i \cdot \left(\frac{1}{\alpha_\nu(\mathbf{x})} \nabla_j \Big[f_{ij,\nu}(\mathbf{x}) J_\nu(\mathbf{x}) \Big] \right) = \alpha_\nu^{\rm abs}(\mathbf{x}) [J_\nu(\mathbf{x}) - B_\nu(\mathbf{x})] \tag{4.121}$$

The remaining question is: How do we determine the variable Eddington tensor $f_{ij,\nu}(\mathbf{x})$? The way this is done in the Variable Eddington tensor method is by doing the following iteration process:

1. Start with the Eddington approximation $f_{ij,\nu} = (1/3)\delta_{ij}$
2. Solve Eq. (4.121) to obtain $J_\nu(\mathbf{x})$
3. Use Eq. (4.19) to compute the source function $S_\nu(\mathbf{x})$
4. Do a formal transfer calculation and compute $f_{ij,\nu}(\mathbf{x})$. This step is like the Lambda Operator, but now to compute $f_{ij,\nu}(\mathbf{x})$ instead of $J_\nu(\mathbf{x})$.
5. Go back to 2, until $J_\nu(\mathbf{x})$ has converged

This method converges extremely rapidly. Sometimes just 4 iterations are enough! Of course, step 4 remains numerically costly, but the cost is similar to the cost of a Lambda Operator.

This extremely rapid speed of the convergence is, however, essentially dependent on the fact that in going from Eq. (4.120) to Eq. (4.121) we have essentially removed the scattering source terms from the right-hand-side. If we would do the iteration with Eq. (4.120), i.e. keeping $j_\nu(\mathbf{x})$ fixed while solving the VET equation Eq. (4.120), the convergence would be as slow as Lambda Iteration. The reason for this is that in that case solving Eq. (4.120) would be nothing more than a fancy way of applying a Lambda Operator.

Considering the extremely fast convergence rate, the VET method shown here appears to be a superior method to most others. However, while it is easy to derive the equations for the case of isotropic scattering, it is quite another matter to derive similarly fast converging equations for more complex problems. Also, for multi-dimensional problems step 4 is not so trivial either and can for some type of problems be very time-consuming. And finally, adding new physics to a radiative transfer code based on the VET method can be hard. A VET code is thus difficult to maintain and upgrade. These are presumably some of the reasons why the VET method is not so much in use in current codes.

[2] In 1-D models it is often called by the more familiar name *Variable Eddington Factor* (VEF) method

4.5.4 Worked-out example: Moment methods in 1-D problems

Let us work things out explicitly for a 1-D plane parallel problem. The moment equations are then (omitting (z) and ν for notational convenience):

$$\frac{dH}{dz} = j - \alpha J = \alpha\,(S - J) \tag{4.122}$$

$$\frac{dK}{dz} = -\alpha\, H \tag{4.123}$$

With the Eddington approximation, and after inserting the expression for S for an absorption/scattering problem, we arrive at the diffusion equation for absorption/scattering:

$$\frac{1}{3}\frac{d}{dz}\left(\frac{1}{\alpha}\frac{dJ}{dz}\right) = \alpha^{\rm abs}(J - B) \tag{4.124}$$

At this point it can be convenient to switch to optical depth τ as our coordinate. In fact, you will find that in most literature on 1-D plane-parallel models of e.g. stellar atmospheres the optical depth τ is indeed chosen as coordinate, not the vertical height z. For most of this lecture we will use, however, z, because this lecture also involves 3-D radiative transfer, which does not allow using optical depth as a coordinate. However, here we will make an exception, and switch to τ as a coordinate. *We will choose the τ along the slanted downward ray with $\mu = -1/\sqrt{3}$ as our coordinate.*[3]

$$\tau(z) = \sqrt{3}\int_z^\infty \alpha(z)dz \tag{4.125}$$

Eq. (4.124) then becomes

$$\frac{d^2J}{d\tau^2} = \epsilon(J - B) \tag{4.126}$$

Now let us discretize this on a grid. We get

$$\frac{1}{\Delta\tau_{i+1/2}}\left(\frac{J_{i+3/2} - J_{i+1/2}}{\Delta\tau_{i+1}} - \frac{J_{i+1/2} - J_{i-1/2}}{\Delta\tau_i}\right) = \epsilon(J_{i+1/2} - B_{i+1/2}) \tag{4.127}$$

If we write all the Js and Bs as vectors $\mathbf{J}, \mathbf{B}$ we can express this equation in the form of a matrix equation:

$$M\mathbf{J} = \epsilon\mathbf{B} \tag{4.128}$$

with M a tridiagonal matrix with the following non-zero elements:

$$M_{i+1/2,i-1/2} = -\frac{1}{\Delta\tau_{i+1/2}\Delta\tau_i} \tag{4.129}$$

$$M_{i+1/2,i+1/2} = 1 + \frac{1}{\Delta\tau_{i+1/2}\Delta\tau_i} + \frac{1}{\Delta\tau_{i+1/2}\Delta\tau_{i+1}} \tag{4.130}$$

$$M_{i+1/2,i+3/2} = -\frac{1}{\Delta\tau_{i+1/2}\Delta\tau_{i+1}} \tag{4.131}$$

We will discuss numerical methods for solving this set of equations in Section 4.7.

4.5.5 Relation between diffusion and the two-stream approximation

There is a surprising relation between the two-stream approximation and the diffusion approximation. If we write

$$J = \tfrac{1}{2}(I_+ + I_-) \qquad , \qquad H = \tfrac{1}{2\sqrt{3}}(I_+ - I_-) \tag{4.132}$$

[3] In the Rybicki & Lightman book, and in most literature, the τ is taken as the perfectly vertical optical depth, but since we have so far always used τ as being along the rays, it would lead to unnecessary confusion.

The formal transfer equations for I_- and I_+ are:

$$\frac{1}{\sqrt{3}}\frac{dI_+}{dz} = j - \alpha I_+ = \alpha(S - I_+) \tag{4.133}$$

$$-\frac{1}{\sqrt{3}}\frac{dI_-}{dz} = j - \alpha I_- = \alpha(S - I_-) \tag{4.134}$$

If we add/subtract these equations and divide by 2 we get:

$$\frac{dH}{dz} = j - \alpha J = \alpha(S - J) \tag{4.135}$$

$$\frac{1}{\sqrt{3}}\frac{dJ}{dz} = -\sqrt{3}\alpha H \tag{4.136}$$

These two equations can be combined to

$$\frac{1}{3}\frac{d}{dz}\left(\frac{1}{\alpha}\frac{dJ}{dz}\right) = \alpha(J - S) \tag{4.137}$$

in which we recognize the diffusion equation. This means that the two-stream approximation is mathematically equivalent to the Eddington approximation.

At this point it also becomes clear why we made the choice of $\mu = \pm 1/\sqrt{3}$ as the angles of the rays: It ensures the right radiative diffusion rates in the limit of high optical depth.

Note that, while the two-stream and diffusion approximations are mathematically equivalent, in practice we use the two-stream approximation as a Lambda Operator in the LI and ALI iteration schemes, while the diffusion approximation is typically used in the form of Eq. (4.118) in which the scattering part of the right-hand side has been removed so that we get the final answer in one go, and no iteration is required. They are therefore mathematically related, but in numerical practice different.

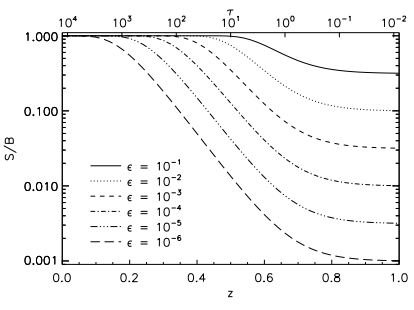
We can use the equivalence of two-stream and diffusion approximation to formulate self-consistent boundary conditions (see the book by Rybicki & Lightman). If we have, for instance, a semi-infinite atmosphere, with a open boundary at the top but infinitely progressing downward, the boundary condition at the top is that $I_- = 0$. Inserting this into Eq. (4.132) yields the following boundary condition at the top:

$$H = \frac{1}{\sqrt{3}}J \tag{4.138}$$

If we would, instead, have a lower open boundary we would get a minus sign in this equation.

4.5.6 Worked-out example: Solutions of the diffusion equation in 1-D

With the equivalence of the two-ray and diffusion approximations, we can argue that we have already seen examples of solutions of the diffusion equation in 1-D, namely those that were shown in the section on ALI methods. This is indeed the case. Let us nevertheless revisit these examples and discuss how they behave as we vary ϵ. You can see the results in the margin figure. What you see is that the location where the solution approaches the thermal equilibrium value $S = B$ shifts to higher and higher optical depths as ϵ increases, and that this depth goes proportional to $1/\sqrt{\epsilon}$. We call this the *thermalization depth*. This behavior can be traced back to the form of the diffusion equation (Eq. 4.126):



$$\frac{d^2J}{d\tau^2} = \epsilon(J - B) \tag{4.139}$$

If we define the *effective optical depth* $\tau_{\rm eff}$ as

$$\tau_{\rm eff} = \sqrt{\epsilon}\,\tau \tag{4.140}$$

then the diffusion equation becomes

$$\frac{d^2J}{d\tau_{\rm eff}^2} = J - B \tag{4.141}$$

So in units of $\tau_{\rm eff}$ the thermalization depth is always around $\tau_{\rm eff} \simeq 1$, so in units of τ it is around $\tau \simeq \sqrt{\epsilon}$.

4.6 Thermal radiative transfer

We have so far always taken the example of isotropic scattering in a medium with known temperature. Many real radiative transfer problems can be considered mathematically similar to this problem, even if they are much more complex in the details. In the next chapters we will discuss various such problems in detail, and discuss their solution methods there. However, one general kind of problem will appear again and again, and it may therefore be useful to give a little preview here: The problem of *thermal radiative transfer*, i.e. how radiative transfer can transport heat, and affect the temperature throughout our object of interest. This will be one of the main topics in the chapter on radiative transfer in dusty media (Chapter 5) as well as in the chapters on stellar atmospheres (Chapter 10) and planetary atmospheres (Chapter 9). But let us have a glimpse at this problem already now.

4.6.1 Thermal radiative transfer in a grey stellar atmosphere

Let us consider a simple 1-D plane-parallel model of a stellar atmosphere. We assume zero albedo ($\epsilon = 1$, i.e. $\alpha_\nu = \alpha_\nu^{\rm abs}$). We will employ the moment equations with the Eddington approximation (Section 4.5.2, i.e. the diffusion approximation), which is equivalent to the two-stream approximation (Section 4.4.2). The first two moment equations, in the Eddington approximation ($K_\nu = J_\nu/3$) are:

$$\frac{dH_\nu}{dz} = \alpha_\nu\,(B_\nu(T) - J_\nu) \tag{4.142}$$

$$\frac{1}{3}\frac{dJ_\nu}{dz} = -\alpha_\nu\,H_\nu \tag{4.143}$$

Now let us assume that the opacity is grey, i.e. α_ν does not depend on ν. We can integrate Eqs. (4.142, 4.143) over frequency:

$$\frac{dH}{dz} = \alpha\,(B(T) - J) \tag{4.144}$$

$$\frac{1}{3}\frac{dJ}{dz} = -\alpha\,H \tag{4.145}$$

where

$$J = \int_0^\infty J_\nu d\nu \tag{4.146}$$

$$H = \int_0^\infty H_\nu d\nu \tag{4.147}$$

$$B(T) = \int_0^\infty B_\nu(T) d\nu = \frac{\sigma_{\rm SB}}{\pi}T^4 \tag{4.148}$$

Next we are going to assume that the gas and the radiation field are always in *radiative equilibrium* with each other:

$$J = B(T) = \frac{\sigma_{\rm SB}}{\pi}T^4 \tag{4.149}$$

If we insert this into Eq. (4.144) then we obtain that

$$H = \text{constant} \tag{4.150}$$

This is a natural thing in a stellar atmosphere: the atmosphere is responsible for producing the radiative flux that cools the star. The radiative flux at the stellar surface is typically a known property of the star:

$$F = 4\pi H = \sigma_{\rm SB} T_{\rm eff}^4 \tag{4.151}$$

where $T_{\rm eff}$ is the *effective temperature* of the star. For a given stellar type you can look up the effective temperature in tables. For our radiative transfer problem, $T_{\rm eff}$ is thus a *given quantity*. And with Eq. (4.150) we know that it is constant throughout the atmosphere. This is great because Eq. (4.145) thus becomes trivial to integrate. Let us define the *vertical downward optical depth* $\hat{\tau}$:

$$\hat{\tau}(z) = \int_z^\infty \alpha dz \tag{4.152}$$

(the $\hat{}$ is to distinguish it from the τ along the slanted rays, Eq. 4.125, which had an extra $\sqrt{3}$ in there). We can now write Eq. (4.145):

$$\frac{dJ}{d\hat{\tau}} = 3H = \frac{3}{4\pi}\sigma_{\rm SB} T_{\rm eff}^4 \tag{4.153}$$

This integrates trivially to

$$J = \frac{3}{4\pi}\sigma_{\rm SB} T_{\rm eff}^4 \hat{\tau} + C \tag{4.154}$$

with C an integration constant to be determined from the boundary condition. The boundary condition at $\hat{\tau} = 0$ can be obtained from the two-stream approximation (Eq. 4.138):

$$\frac{1}{4\pi}\sigma_{\rm SB} T_{\rm eff}^4 = H = \frac{1}{3}\frac{dJ}{d\hat{\tau}} = \frac{1}{\sqrt{3}} J \tag{4.155}$$

Now inserting Eq. (4.149) finally gives $C = \sqrt{3} T_{\rm eff}^4/4$, so that the final solution is:

$$T^4(\hat{\tau}) = \frac{3}{4} T_{\rm eff}^4 \left(\hat{\tau} + \frac{1}{\sqrt{3}}\right) \qquad \text{(Two-stream)} \tag{4.156}$$

Note that in some derivations in the literature not the two-stream approximation is used for the boundary condition, but the assumption that the intensity for $\mu > 0$ is constant and for $\mu < 0$ is zero. In that case one would obtain

$$T^4(\hat{\tau}) = \frac{3}{4} T_{\rm eff}^4 \left(\hat{\tau} + \frac{2}{3}\right) \qquad \text{(Classic)} \tag{4.157}$$

which is a slightly more "classic" result (the two numbers differ only by 15%). Since both are approximations anyway, it is hard to say which is better. The main thing we should learn from this result is that with some simple physical considerations we can compute a zeroth-order model of a stellar atmosphere. Another thing we can learn is that if we use the Eddington-Barbier estimation for the intensity (Section 3.5.3), we will get the intensity belonging to a temperature $T = T_{\rm eff}$.

4.6.2 Mathematical similarity with scattering

Now, in which respect is this thermal radiative transfer problem similar to the isotropic scattering problem? Consider the isotropic scattering problem with $\epsilon = 0$. In that case we had the condition that any radiation that is extincted through scattering will also be "re-emitted" into another direction:

$$j_\nu^{\rm scat} = \alpha_\nu^{\rm scat} J_\nu \tag{4.158}$$

or equivalently:

$$S_\nu^{\rm scat} = J_\nu \tag{4.159}$$

It is a way to express "what comes in must go out". This is the same in our thermal radiative transfer problem, where we have:

$$S^{\rm emis} = J \tag{4.160}$$

where the source function $S^{\rm emis} = B(T) = (\sigma_{\rm SB}/\pi)T^4$. The main difference is that Eq. (4.160) is a frequency-integrated equation, while Eq. (4.159) is a monochromatic equation. Other than that, from a mathematical standpoint, they are similar.

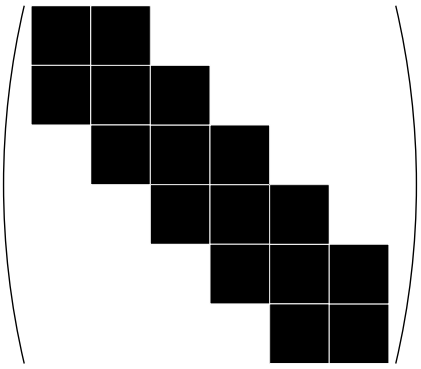
4.7 Solving diffusion-type (i.e. tridiagonal) equations numerically

We have seen that many of the above methods require the solution of a diffusion-type equation or, equivalently, a tridiagonal matrix equation of the type

$$M\mathbf{y} = \mathbf{r} \tag{4.161}$$

In this section we will give a brief overview of methods to solve matrix equations of this kind.

Band–diagonal matrix for 6 pt 1–D grid



The most important feature of this type of matrices is that they are *sparse*, meaning that only a tiny fraction of their elements are non-zero. This allows us to store even huge matrices in computer memory, because we do not need to reserve computer memory for all the zero elements. For 1-D problems there is even another nice aspect: only the diagonal and the side-diagonal elements are non-zero. This will allow for a very simple solution method.

Let us write the matrix with integer indices: $M_{i,j}$. The diagonal is $M_{i,i}$, the lower side-diagonal is $M_{i,i-1}$ and the upper one is $M_{i,i+1}$. One can see that each row has three elements, except for the first one and the last one. This allows us to perform a *forward elimination, backward substitution* procedure, where we start with the first row:

$$M_{1,1}y_1 + M_{1,2}y_2 = r_1 \tag{4.162}$$

we write y_1 in terms of y_2:

$$y_1 = \frac{r_1 - M_{1,2}y_2}{M_{1,1}} \equiv t_1 + c_1 y_2 \tag{4.163}$$

with $t_1 = r_1/M_{1,1}$ and $c_1 = -M_{1,2}/M_{1,1}$. Now we go to the next row

$$M_{2,1}y_1 + M_{2,2}y_2 + M_{2,3}y_3 = r_2 \tag{4.164}$$

We can now insert Eq. (4.163) and obtain

$$(M_{2,1}c_1 + M_{2,2})y_2 + M_{2,3}y_3 = r_2 - M_{2,1}t_1 \tag{4.165}$$

We write y_2 in term of the rest:

$$y_2 = \frac{r_2 - M_{2,1}t_1 - M_{2,3}y_3}{M_{2,1}c_1 + M_{2,2}} = t_2 + c_2 y_3 \tag{4.166}$$

and we repeat this procedure until the end. There we again encounter a row with just two elements, which closes the system. We can now work our way back to substitute the values to obtain our solution of y. See the "Numerical Recipes" book for more details. Their subroutine is called `tridag()`.

For multi-dimensional grids the corresponding matrix is more complex. In addition to the $M_{i,i\pm1}$ side bands, the matrix has also $M_{i,i\pm N_x}$ and (for 3-D) $M_{i,i\pm N_x N_y}$ sidebands, where $N_{x,y,z}$ is the number of gridpoints in x,y,z-direction (see Section 4.4.3 for how to

put 2-D and 3-D problems into matrix form). Now this forward elimination-backward-substitution procedure no longer works. Solving problems of this kind is much harder and solving a tridiagonal problem. There are, however, many libraries with standard methods available on the web. Most of these methods are based *Krylov subspace methods*, which are iterative methods. We will, however, not dwell on this further, since this is a topic on its own. Again, we refer to the "Numerical Recipes" book for details.

Band–diagonal matrix for 6x6 2–D grid