On solving a Cosmic Ray equation

Introduction

In the theory of cosmic ray modulation in the heliosphere, i.e. the region around the sun that is dominated by the solar wind plasma (which itself represents the expanding outer part of the sun's atmosphere), there occurs the following hydrodynamical version of a cosmic ray (CR) transport- or energy equation:

$$\nabla \cdot (\overset{\leftrightarrow}{\kappa}\ \nabla p) + (-\vec{v}_{sw} - \gamma \vec{v}_{dr}) \cdot \nabla p + [-\gamma(\nabla \cdot \vec{v}_{sw})]p = 0$$

The quantities in this equation are:

$p(\vec{r})$	$=$	pressure or energy density of the cosmic rays
$\vec{v}_{sw}$	$=$	velocity of the solar wind
$\vec{v}_{dr}$	$=$	drift velocity of the cosmic rays
γ	$=$	constant polytropic index of the cosmic ray gas
$\overset{\leftrightarrow}{\kappa}$	$=$	the diffusion tensor

The above equation describes the diffusion and drift of the cosmic ray particles in the interplanetary magnetic field and their convection and adiabatic deceleration due to the expanding solar wind plasma. The pressure indicates the cosmic ray intensity at a position $\vec{r}$. More about the physics of CR is found in astronomical literature [**?**].

Analytical treatment

Using spherical polar coordinates

$$x = r\cos\phi\sin\theta \quad ; \quad y = r\sin\phi\sin\theta \quad ; \quad z = r\cos\theta$$

one obtains the most convenient analytical formulation of the problem.
In all subsequent cases, the drift velocity $\vec{v}_{dr}$ is zero and the velocity of the solar

wind is radial & constant: $\vec{v}_{sw} = v_0\vec{e}_r$. Hence the term $-\gamma(\nabla \cdot \vec{v}_{sw})$ can be evaluated as $-\gamma v_0 \nabla \cdot \vec{e}_r = -\gamma v_0 (2/r)$.

Here comes a compilation of (fictive) one-dimensional analytical solutions of the CR equation. It is remarked that *this* version of the theory actually matches reasonably well with (parts of) the numerical solutions. So I have confidence that *both* theory and code are correct, at last.

MODE=1: $\kappa = radial = r$; $v_0 = 0$

The Partial Differential Equation (P.D.E.) is reduced, for κ equal to the scalar r, and for v_0 equal to zero, to the following equation:

$$\frac{d}{dr} r \frac{dp}{dr} + 2\frac{dp}{dr} = 0$$

$$\left(\frac{d}{dr} + \frac{2}{r}\right) r \frac{d}{dr} p = 0$$

Using a basic formula of Operator Calculus $\frac{d}{dx} + f(x) = e^{-\int f(x)dx} \frac{d}{dx} e^{+\int f(x)dx}$ with $exp(\int 2/r\, dr) = r^2$:

$$\left(\frac{1}{r^2}\frac{d}{dr} r^2 r\right) \frac{d}{dr} p = 0$$

The general solution is $p = C/r^2$. Boundary conditions are chosen spherically symmetric: $p(RI) = 0$ and $p(RO) = 1$. Then the solution is:

$$p(r) = \frac{1/RI^2 - 1/r^2}{1/RI^2 - 1/RO^2}$$

It has been seen that this solution agrees VERY well with the numerical values.

MODE=2: $\kappa = unity$; $v_0 = 0$

The P.D.E. is reduced, for κ equal to the unity tensor, and for v_0 equal to zero, to the Laplace equation in spherical coordinates. The boundary conditions are also chosen spherically symmetric: $p(RI) = 0$ and $p(RO) = 1$. Then the solution is:

$$p(r) = \frac{1/RI - 1/r}{1/RI - 1/RO}$$

Since it is well known that the potential field of a point source goes like $1/r$. It has been seen that this solution also agrees VERY well with the numerical values.

MODE=3: $\kappa = radial = r$

The P.D.E. is reduced to an ordinary differential equation:

$$\frac{d}{dr} r \frac{dp}{dr} + (2 - v_0)\frac{dp}{dr} - \gamma v_0 \frac{2}{r} p = 0$$

Here $v_0 = 0.5984$ and $\gamma = 1.663151$. This is a differential equation of type Euler in $p(r)$, a and b constant:

$$r^2 p" + (a+1) r p' + bp = 0$$

Where $a = 2 - v_0$; $b = -2\gamma v_0$. The general solution is, in case $\lambda_1 \neq \lambda_2$:

$$p(r) = C_1 r^{\lambda_1} + C_2 r^{\lambda_2} \qquad C_1, C_2 \text{ arbitrary}$$

$\lambda_{1,2}$ are the solutions of the characteristic equation:

$$\lambda^2 + a\lambda + b = 0$$

Substituting herein the numerical values of v_0 and γ yields:

$$\lambda_1 = 0.8745031 \qquad ; \qquad \lambda_2 = -2.2761030$$

The constants $C_{1,2}$ are determined by the boundary conditions, as before. The result is:

$$p(r) = \frac{r^{\lambda_1} RI^{\lambda_2} - r^{\lambda_2} RI^{\lambda_1}}{RO^{\lambda_1} RI^{\lambda_2} - RO^{\lambda_2} RI^{\lambda_1}}$$

Which also matches VERY well with the numerical solution.

MODE=4: $\kappa = unity = 1$; $\gamma = 0$

If we assume that $v_0 \neq 0$, and specialize for $\gamma = 1$, then CR reduces to the following ordinary differential equation:

$$\frac{d^2 p}{dr^2} + (2/r - v_0)\frac{dp}{dr} - v_0 \frac{2}{r} p = 0$$

The solution of this differential equation can be found again by employing Operator Calculus (which is kind of routine for me if I want to find "exact" solutions):

$$\left(\frac{d}{dr} + \frac{2}{r}\right)\left(\frac{d}{dr} - v_0\right) p = 0$$

Using the basic formula $\frac{d}{dx} + f(x) = e^{-\int f(x)dx} \frac{d}{dx} e^{+\int f(x)dx}$:

$$\left(r.\frac{1}{r^2}\frac{d}{dr} r^2\right)\left(e^{v_0 r}\frac{d}{dr} e^{-v_0 r}\right) p = 0$$

Systematic integration gives:

$$\frac{d}{dr}e^{-v_0 r}p = C_1\frac{e^{-v_0 r}}{r^2}$$

$$p(r) = C_1 e^{v_0 r}\int\frac{e^{-v_0 r}}{r^2}dr + C_2 e^{v_0 r}$$

So there exist the following two elementary solutions:

$$p_1(r) = e^{v_0 r}\int\frac{e^{-v_0 r}}{r^2}dr$$

$$p_2(r) = e^{v_0 r}$$

The integral can be worked out further by partial integration, with $t = -v_0 r$:

$$\int\frac{e^t}{t^2}dt = -\frac{e^t}{t} + \int\frac{e^t}{t}dt$$

Giving for the first solution:

$$p_1(r) = \frac{1}{v_0 r} + e^{v_0 r}.Ei(-v_0 r)$$

$$Ei(x) = \int_{-\infty}^{x}\frac{e^t}{t}dt$$

If you don't believe this, you can substitute back and check out with MAPLE:

```
p:=1/(v0*r)+exp(v0*r)*Ei(-v0*r);
r*diff(diff(p,r),r)+(2-v0*r)*diff(p,r)-2*v0*p;
simplify(");
quit;
```

Together with the boundary conditions, the solution finally becomes:

$$p(r) = \frac{p_1(RI)p_2(r) - p_2(RI)p_1(r)}{p_1(RI)p_2(RO) - p_2(RI)p_1(RO)}$$

Which in turn can be compared with numerical results. The latter is somewhat difficult for two reasons: the exponential integral function Ei(x) is unknown to Fortran, and the analytical result has a very steep gradient for the nominal value of v_0. The first problem can be adressed by invoking the NAG Fortran Library: the exponential integral function Ei(x) is calculated by a function S13AAF(X,IFAIL). The second problem can be adressed by lowering the value of v_0 by a factor of 100.

MODE=5: $\kappa = unity = 1$; $\gamma = 0$

One more analytical solution of the CR equation can be constructed, resulting in a total of 5 exact solutions so far.

If we assume that κ is the unity tensor, $v_0 \neq 0$, and specialize for $\gamma = 0$ instead of $\gamma = 1$, then the CR Confusion (:-) P.D.E. reduces to the following ordinary differential equation:

$$\frac{d^2p}{dr^2} + (2/r - v_0)\frac{dp}{dr} = 0$$

The solution of this differential equation can be found again by Operator Calculus:

$$\left[\frac{d}{dr} + \left(\frac{2}{r} - v_0\right)\right]\frac{d}{dr}p = 0$$

Using the basic formula $\frac{d}{dx} + f(x) = e^{-\int f(x)dx}\frac{d}{dx}e^{+\int f(x)dx}$:

$$\left(\frac{1}{r^2}e^{v_0 r}\frac{d}{dr}r^2 e^{-v_0 r}\right)\frac{d}{dr}p = 0$$

Systematic integration gives:

$$p(r) = C_1 \int \frac{e^{v_0 r}}{r^2}dr + C_2$$

The integral can be worked out as in the previous case, giving the solution, apart from constants:

$$p_1(r) = -\frac{e^{v_0 r}}{v_0 r} + Ei(v_0 r)$$

If you don't believe this, you can substitute back and check out with MAPLE:

```
p:=-exp(v0*r)/(v0*r)+Ei(v0*r);
diff(diff(p,r),r)+(2/r-v0)*diff(p,r);
simplify(");
quit;
```

Together with the boundary conditions, the solution finally becomes:

$$p(r) = \frac{p_1(RI) - p_1(r)}{p_1(RI) - p_1(RO)}$$

Which in turn can be compared with numerical results. It should be emphasized, however, that the analytical, and *hence* also the numerical solution, is *only* known for the case $v_0 < 0$. It is *just zero* in other cases!

Implementation

The five analytical solutions have been implemented for testing purposes as the Fortran function EXAKT(R), where R is a radial distance from the origin. The solution is selected by an index MODE, which is ranging from 1,..,5 and resides in the common block /TEST/. MODE is also the index which determines what coefficients to calculate in subroutine COEFFS. Needless to say that the choice of the exact solution and the choice of the coefficients should lead to a match between the exact and the numerical solutions respectively.

Under normal circumstances, COEFFS contains the entire physics of the problem. Given the convection-difusion equation as such, together with its boundary conditions and the geometry of the calculation domain, only the coefficients are yet to be determined, by proper physical modelling. The relationship between the parameters of COEFFS and the (test) physics of CR is as follows:

$$\overset{\leftrightarrow}{\kappa} = \begin{bmatrix} a_{xx} & a_{xy} & a_{zx} \\ a_{xy} & a_{yy} & a_{yz} \\ a_{zx} & a_{yz} & a_{zz} \end{bmatrix} \qquad = \text{symmetric}$$

$$-v_0 \frac{x}{r} = a_x \qquad - v_0 \frac{y}{r} = a_y \qquad - v_0 \frac{z}{r} = a_z$$

$$-\gamma v_0 \frac{2}{r} = a_0$$

With our test examples, the κ tensor can be *unity* or *radial*. In the latter case, the coefficients are calculated by:

$$a_{xx} = \frac{x^2}{r} \qquad a_{yy} = \frac{y^2}{r} \qquad a_{zz} = \frac{z^2}{r}$$

$$a_{xy} = \frac{xy}{r} \qquad a_{yz} = \frac{yz}{r} \qquad a_{zx} = \frac{zx}{r}$$

Numerical treatment

Leaving out non-essentials, the convection-diffusion equation for CR reads as follows:

$$\nabla \cdot (\overleftrightarrow{\kappa} \, \nabla p) + \vec{v} \cdot \nabla p + a_0 p = 0$$

The quantities in this equation are:

$p(\vec{r})$	$=$	pressure or energy density of the cosmic rays
$\overleftrightarrow{\kappa}$	$=$	the diffusion tensor
$\vec{v}$	$=$	total velocity $-\vec{v}_{sw} - \gamma \vec{v}_{dr}$
a_0	$=$	coefficient $-\gamma(\nabla \cdot \vec{v}_{sw})$

There are several factors which make the CR equation unmanageble by available packages, despite of the fact that Convection-Diffusion equations like this are quite common in for example Fluid Dynamics. First there is the symmetric tensor κ, which as such could have been accounted for by a Structural Mechanics program [**?**]. But in our case this tensor has to be combined with convection terms. These convection terms, together with diffusion, could have been handled by any common Computational Fluid Dynamics (CFD) code [**?**]. But to my knowledge there exists no CFD code which implements a full tensor in its diffusion modelling. Moreover, due to the geometry of the problem, the density decreases with distance like r^{-2} resulting - with the condition of mass conservation - into a velocity field with *non-vanishing* divergence. It is essentially the latter feature which also gives rise to a multiplicative a_0 term; this complicates the CR model even further.

The above is not to say that something very special would be needed in order to solve the problem. On the contrary. We shall see that very conventional Finite Element and Finite Volume techniques are quite sufficient. The important thing, however, is that they must be used *in combination*. This is a basic idea I have been advertizing several times in the past, and will continue to do so in future. The idea is called: "Unified Numerical Approximations" [**?**].

Linear Tetrahedron

It is strongly advised to read about the 2-D case first, before proceeding to the more difficult 3-D algebra. It's in the appendix "Triangle Isoparametrics". Let's consider the simplest non-trivial finite element shape in 3-D, which is a *linear tetrahedron*. Function behaviour is approximated inside such a tetrahedron by a *linear* interpolation between the function values at the vertices, also called nodal points. Let T be such a function, and x, y, z coordinates, then:

$$T = A.x + B.y + C.z + D$$

Where the constants A, B, C, D are yet to be determined. Substitute $x = x_k$, $y = y_k$, $z = z_k$ with $k = 0, 1, 2, 3$. Start with:

$$T_0 = A.x_0 + B.y_0 + C.z_0 + D$$

Clearly, the first of these equations can already be used to eliminate the constant D, once and forever:

$$T - T_0 = A.(x - x_0) + B.(y - y_0) + C.(z - z_0)$$

Then the constants A , B , C are determined by:

$$\begin{array}{l} T_1 - T_0 = A.(x_1 - x_0) + B.(y_1 - y_0) + C.(z_1 - z_0) \\ T_2 - T_0 = A.(x_2 - x_0) + B.(y_2 - y_0) + C.(z_2 - z_0) \\ T_3 - T_0 = A.(x_3 - x_0) + B.(y_3 - y_0) + C.(z_3 - z_0) \end{array}$$

Three equations with three unknowns. A solution can be found:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} T_1 - T_0 \\ T_2 - T_0 \\ T_3 - T_0 \end{bmatrix}$$

It is concluded that A, B, C and hence $(T - T_0)$ must be a linear expression in the $(T_k - T_0)$:

$$T - T_0 = \xi.(T_1 - T_0) + \eta.(T_2 - T_0) + \zeta.(T_3 - T_0)$$

$$= \begin{bmatrix} \xi & \eta & \zeta \end{bmatrix} \begin{bmatrix} T_1 - T_0 \\ T_2 - T_0 \\ T_3 - T_0 \end{bmatrix}$$

See above:

$$= \begin{bmatrix} \xi & \eta & \zeta \end{bmatrix} \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

See above:

$$= T - T_0 = \begin{bmatrix} x - x_0 & y - y_0 & z - z_0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

Hence:

$$\begin{array}{l} x - x_0 = \xi.(x_1 - x_0) + \eta.(x_2 - x_0) + \zeta.(x_3 - x_0) \\ y - y_0 = \xi.(y_1 - y_0) + \eta.(y_2 - y_0) + \zeta.(y_3 - y_0) \\ z - z_0 = \xi.(z_1 - z_0) + \eta.(z_2 - z_0) + \zeta.(z_3 - z_0) \end{array}$$

But also:

$$T - T_0 = \xi.(T_1 - T_0) + \eta.(T_2 - T_0) + \zeta.(T_3 - T_0)$$

Therefore the *same* expression holds for the function T as well as for the coordinates x, y, z. This is called an *isoparametric* transformation. It is remarked without proof that the *local coordinates* ξ, η, ζ within a tetrahedron can be interpreted as sub-volumes, spanned by the vectors $\vec{r}_k - \vec{r}_0$ and $\vec{r} - \vec{r}_0$ where $\vec{r} = (x, y, z)$ and $k = 1, 2, 3$.
Reconsider the expression:

$$T - T_0 = \xi.(T_1 - T_0) + \eta.(T_2 - T_0) + \zeta.(T_3 - T_0)$$

Partial differentiation to ξ , η , ζ gives:

$$\partial T/\partial\xi = T_1 - T_0 \quad ; \quad \partial T/\partial\eta = T_2 - T_0 \quad ; \quad \partial T/\partial\zeta = T_3 - T_0$$

Therefore:

$$T = T(0) + \xi\frac{\partial T}{\partial\xi} + \eta\frac{\partial T}{\partial\eta} + \zeta\frac{\partial T}{\partial\zeta}$$

This is part of a Taylor series expansion around node (0). Such Taylor series expansions are very common in Finite Difference analysis. Now rewrite as follows:

$$T = (1 - \xi - \eta - \zeta).T_0 + \xi.T_1 + \eta.T_2 + \zeta.T_3$$

Here the functions $(1-\xi-\eta-\zeta), \xi, \eta, \zeta$ are called the *shape functions* of a Finite Element. Shape functions N_k have the property that they are unity in one of the nodes (k), and zero in all other nodes. In our case:

$$N_0 = 1 - \xi - \eta - \zeta \quad ; \quad N_1 = \xi \quad ; \quad N_2 = \eta \quad ; \quad N_3 = \zeta$$

So we have two representations, which are allmost trivially equivalent:

$$\begin{array}{ll} T = T_0 + \xi.(T_1 - T_0) + \eta.(T_2 - T_0) + \zeta.(T_3 - T_0) & : \text{Finite Difference} \\ T = (1 - \xi - \eta - \zeta).T_0 + \xi.T_1 + \eta.T_2 + \zeta.T_3 & : \text{Finite Element} \end{array}$$

What kind of terms can be discretized at the domain of a linear tetrahedron? In the first place, the function $T(x, y, z)$ itself, of course. But one may also try on the first order partial derivatives $\partial T/\partial(x, y, z)$. We find:

$$\partial T/\partial x = A \quad ; \quad \partial T/\partial y = B \quad ; \quad \partial T/\partial z = C$$

Using the expressions which were found for A, B, C:

$$\begin{bmatrix} \partial T/\partial x \\ \partial T/\partial y \\ \partial T/\partial z \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} T_1 - T_0 \\ T_2 - T_0 \\ T_3 - T_0 \end{bmatrix}$$

It is seen from this formula that one must determine the inverse of the above matrix first. Then add up the rows of the inverted matrix and provide the sum with a minus sign, in order to find the coefficients belonging to T_0. The result is a 3×4 *Differentiation Matrix*, which represents the gradient operator $\partial/\partial(x, y, z)$ for the function values $T_{0,1,2,3}$ at a linear tetrahedron.

Diffusion part

Consider the three-dimensional Laplace-like term, which is defined in Cartesian coordinates by:

$$\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} + \frac{\partial Q_z}{\partial z}$$

where:

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \end{bmatrix} = \begin{bmatrix} \kappa_{xx} & \kappa_{xy} & \kappa_{xz} \\ \kappa_{yx} & \kappa_{yy} & \kappa_{yz} \\ \kappa_{zx} & \kappa_{zy} & \kappa_{zz} \end{bmatrix} \begin{bmatrix} \partial p/\partial x \\ \partial p/\partial y \\ \partial p/\partial z \end{bmatrix}$$

Which is essentially the first term of the CR equation. Suppose the contribution is valid in a domain D with boundary S. According to the so called Galerkin method, the contribution is multiplied by an arbitrary function f and then integrated over the Domain of interest. Since the function f is completely arbitrary (well, continuous at least), this is supposed to be equivalent to the original problem:

$$\iiint f(x,y,z) \left(\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} + \frac{\partial Q_z}{\partial z} \right) dx.dy.dz$$

The advantage of the Galerkin formulation is that we are able now to convert second order derivatives into first order derivatives. To see how this works, let us recall Green's theorem, or partial integration for triple integrals, by which the following expression can be substituted for the Galerkin integral:

$$\oint\!\!\oint f.Q_n \, dS \quad - \quad \iiint \left(Q_x \frac{\partial f}{\partial x} + Q_y \frac{\partial f}{\partial y} + Q_z \frac{\partial f}{\partial z} \right) dx.dy.dz$$

The first of these terms incorporates boundary conditions at the surface S. The boundary integral is *zero* in case the normal derivative $Q_n = 0$.

For this reason $Q_n = 0$ is called a *natural* boundary condition: it is fulfilled *automatically* if the first term in the above formulation is simply discarded. With the CR problem, such a natural boundary condition occurs at the $z = 0$ plane (at least in the case of interest for us). Due to symmetry, the gradient of the pressure normal to it must vanish:

$$\left(\frac{\partial p}{\partial z} = 0 \right)_{z=0}$$

It can be shown that the $\overleftrightarrow{\kappa}$ tensor reads as follows, when expressed in spherical coordinates:

$$\begin{bmatrix} \kappa_{rr} & 0 & \kappa_{r\phi} \\ 0 & \kappa_{\theta\theta} & 0 \\ \kappa_{\phi r} & 0 & \kappa_{\phi\phi} \end{bmatrix}$$

At $z = 0$ the vector $\vec{e}_\theta$ coincides with $-\vec{e}_z$ and the normal $\vec{n}$. Therefore:

$$(Q_n = 0)_{z=0} \quad \Longleftrightarrow \quad \overset{\leftrightarrow}{\kappa}\, \nabla p \cdot \vec{n} = \kappa_{\theta\theta} \frac{\partial p}{\partial \theta} = 0 \quad \Longleftrightarrow \quad \left(\frac{\partial p}{\partial z} = 0 \right)_{z=0}$$

Hence the natural boundary condition at $z = 0$ is fulfilled if and only if the symmetry condition is fulfilled. Implementation of the boundary condition would have been a zillion times more difficult if this were not the case!

The second term is an integral for the *bulk* material. Substitute pressure fluxes herein and watch out for the minus sign:

$$-\iiint \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{bmatrix} \begin{bmatrix} \kappa_{xx} & \kappa_{xy} & \kappa_{xz} \\ \kappa_{yx} & \kappa_{yy} & \kappa_{yz} \\ \kappa_{zx} & \kappa_{zy} & \kappa_{zz} \end{bmatrix} \begin{bmatrix} \partial p/\partial x \\ \partial p/\partial y \\ \partial p/\partial z \end{bmatrix} dx.dy.dz$$

Note that the second order derivatives have been removed indeed. It is possible to handle second order problems with first order (linear) finite elements only.

If the integration domain is subdivided into finite elements E, then the above integral is splitted up as a sum of integrals over these elements. Integration is always carried out numerically, by using *integration points* [?]. A little bit of innovation is involved in recognizing that it doesn't make much difference if the integration points are deliberately chosen in such a way that evaluation is as easy as possible: we always select them at the vertices of any elements involved. It can be shown that elements which are integrated in such a way are in fact *superpositions of linear tetrahedra*. Linear tetrahedra are so to speak the ultimate 3-D elements and there's no need for anything else most of the time. At such linear tetrahedra, differentiations $\partial/\partial(x, y, z)$ are given as a matrix operation, with the *Differentiation Matrices* found in an earlier stage. Here is the symbolic representation for the element-matrix contribution belonging to the diffusion term, using differentiation matrices $\partial/\partial(x, y, z)$:

$$-\sum_{\text{tetrahedra}} \frac{\Delta}{n} \begin{bmatrix} \partial/\partial x & \partial/\partial y & \partial/\partial z \end{bmatrix} \begin{bmatrix} \kappa_{xx} & \kappa_{xy} & \kappa_{xz} \\ \kappa_{yx} & \kappa_{yy} & \kappa_{yz} \\ \kappa_{zx} & \kappa_{zy} & \kappa_{zz} \end{bmatrix} \begin{bmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{bmatrix}$$

Here n = number of element-brick vertices, Δ = determinant of tetrahedron. A formula which can be traced back easily in the Fortran listing of the ELEMENT subroutine. In [?] it is suggested that a tensor should be represented as a diagonal matrix, thus making it necessary to find eigenvalues and -vectors in the first place. The above analysis shows that such a procedure is too much of the good. (Well, I didn't believe it at first sight, but experience shows that the coefficients obtained for the element matrix are always identical with both methods.)

Convection part

There are several paths to understanding with respect to proper discretization of the convective terms in the CR equation. The first thing to keep in mind is that Finite Volume methods are to be preferred and that techniques of upwinding according to this method are to be considered as common practice. There will be no dispute about the pros and cons of this approach. Now it is well known that, with Finite Volume methods, rectangular (equidistant) grids are the easiest to work with. Having adopted the idea that tetrahedral (sub)elements are necessary and sufficient, the only thing we have to do is to transform the coordinates of the tetrahedral element into a rectangular coordinate system and then apply the Finite Volume tradition within these new coordinates.

Let $(0, 1, 2, 3)$ be the numbering of the vertices of an arbitrary tetrahedron. Let (u, v, w) be the velocity components in the global (x, y, z) coordinates. Let (U, V, W) the components of the velocity, as seen in the local coordinate system spanned by the sides $\overline{01}$, $\overline{02}$, $\overline{03}$ of the tetrahedron. Then the following relationship holds:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = U \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \\ z_1 - z_0 \end{bmatrix} + V \begin{bmatrix} x_2 - x_0 \\ y_2 - y_0 \\ z_2 - z_0 \end{bmatrix} + W \begin{bmatrix} x_3 - x_0 \\ y_3 - y_0 \\ z_3 - z_0 \end{bmatrix}$$

The inverse of this transformation can be written as follows:

$$\begin{bmatrix} U & V & W \end{bmatrix} = \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}^{-1}$$

The transformed scheme for convection now simply reads:

$$U(p_1 - p_0) + V(p_2 - p_0) + W(p_3 - p_0)$$

In the chapter "Linear Tetrahedron" the following formula was derived:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} T_1 - T_0 \\ T_2 - T_0 \\ T_3 - T_0 \end{bmatrix}$$

Together with:

$$\partial T/\partial x = A \quad ; \quad \partial T/\partial y = B \quad ; \quad \partial T/\partial z = C$$

Combining this with the above, it is inferred that:

$$\begin{bmatrix} U & V & W \end{bmatrix} \begin{bmatrix} p_1 - p_0 \\ p_2 - p_0 \\ p_3 - p_0 \end{bmatrix} =$$

$$\begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} p_1 - p_0 \\ p_2 - p_0 \\ p_3 - p_0 \end{bmatrix} =$$

$$\begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} \partial p/\partial x \\ \partial p/\partial y \\ \partial p/\partial z \end{bmatrix} = u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z}$$

Just meaning that the theory is quite consistent with respect to the proposed coordinate transformation.

In order to achieve proper weighting of the convection contribution as compared with the contribution due to diffusion, it must be examined how the scheme fits into the same Galerkin procedure as was applied to diffusion:

$$\iiint f \left(u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} \right) dx.dy.dz$$

Again, the integration is carried out numerically, with the integration points located at the vertices of the finite element brick. The result is:

$$\frac{1}{n} \sum_{k=1}^{n} |\Delta_k| \, (\text{convex})_k$$

Where "convex" are the convective terms, discretized according to the above, n =number of brick vertices, Δ_k =determinant of tetrahedron at vertex (k).

So far so good. What has been forgotten so far is just one more thing: *upwinding*. Let's recall the transformed scheme for convection:

$$U(p_1 - p_0) + V(p_2 - p_0) + W(p_3 - p_0) = Up_1 + Vp_2 + Wp_3 - (U + V + W)p_0$$

When applying the conventional Finite Volume upwind technique [?], only velocity components (U, V, W) which have a *negative* sign are allowed to contribute to the scheme. There is a small additional complication, which can be explained as follows. It is assumed that the local grid is orthogonal in the first place. Attention is paid to the velocity component U only, which means in effect that we have a one-dimensional problem. Brainless application of the Galerkin method, without upwinding, would have resulted in a scheme like:

$$\frac{U_2 - U_1}{2} + \frac{U_3 - U_2}{2} = \frac{U_3 - U_1}{2} \qquad \text{instead of} \qquad (U_2 - U_1)$$

The infamous & wrong zero-diagonal result is on the left. The correct upwind scheme is on the right. It is seen that a factor 2 in the denominator would give insufficient weight to the latter. Hence we must apply $2\times$ these terms.

Remaining parts

The last term to be discretized is the multiplicative factor a_0. Due to the fact that everything is integrated at the element vertices, this represents no problem whatsoever. Simply add terms a_0 times Δ_k/n to the diagonal of the element matrix under construction.

The entire local discretization is contained in *one* Fortran subroutine, called ELEMENT. Two building blocks ("bricks") are implemented herein: a prism with 6 vertices and a hexahedron with 8 vertices. Both are ultimately built up from (mutually overlapping) tetrahedra, which are located with their "origins" 0 at the vertices of the bricks. A step in determining the Differentiation Matrices is the inversion of a 3×3 matrix by subroutine DINV3.

Needed for the element matrices are two things: local geometry information and the coefficients of the CR equation, which also depend upon the coordinates. Calculating the geometry for a hemisphere is elementary. It is done by a piece of in-line code in the MAIN program. Calculating the coefficients of the CR equation is done in the ELEMENT subroutine itself: subroutine COEFFS.

The local element matrices are assembled into a large a-symmetrical (banded) global matrix by the traditional finite element Assembly procedure. What's needed for such a procedure is the so called connectivity or topology of the mesh. We have decided also to keep the connectivity information in-line. Use has been made of a Fortran statement function $nr(i, j, k)$, which basically is an association between the finite element (nr) and the finite difference (i, j, k) numbering schemes, thus combining (again) the best of both worlds.

There are two types of Boundary Conditions associated with CR. The "natural" type B.C. which is applicable for the plane $z = 0$ was discussed already in the section "Diffusion part". Dirichlet type boundary conditions occur at the inner and the outer hemi-spheres, which furthermore delimit the integration domain. Pressure values must be prescribed there. In the demonstration model, values 0 and 1 are adopted, respectively.

Direct outcore solver

After the asymmetrical linear system of equations has been assembled, it must be solved. A direct solution method has been implemented in the first place. It's still contained in a main program called "Direct.f". I have devised an out-core version of the LU decomposition technique for asymmetric matrices: subroutines VEGEN and TERUG. The bookkeeping, which is inevitably associated with storing coefficients in a band matrix, is taken care of by the Fortran statement function lok(i,j). It's used in all main programs, as well as in the two subroutines mentioned.

With such a direct solution method, Dirichlet boundary conditions are usually implemented by adding a BIG number to the diagonal of the global matrix and adding the same BIG number, multiplied by the actual (pressure) value, to the right hand side (load vector). A certain inaccuracy is introduced herewith, which can be diminished by making BIG still bigger. At the expence, however, of making worse the so-called *condition number* of the global matrix. A value of 10^6 seems to be reasonable compromise, and is contained as such in the 'Direct' program.

With direct methods, it turns out that huge amounts of memory and computation time are required for solving even a moderate problem. Let NN be the number of unknowns and NB be the bandwidth. Then memory requirements as a whole will increase as NN*(2*NB+1). This is taken care of a great deal by putting most of the equation coefficients on temporary disk: only (2*NB+1)**2 coefficients need to be kept in core memory at a time. Computing times, however, increase even more with increasing dimensions. It can easily be inferred from the loop structure in 'vegen' that they are roughly proportional to NN*NB**2. Let's compare the following two cases. The first one was actually run, and its computing time measured: it turned out to be 6 minutes.

```
 Unknowns :        2535      : case 1
 Bandwidth:         192      : case 1
 Unknowns :       20335      : case 2
 Bandwidth:         620      : case 2
```

The computing time can be estimated for the second case, at hand of the first one: NN*NB**2 = $(20335/2535) * (620/192)^2 = 83.646 \times$ slower. First run = 6 minutes, second one = 83.646 x 6 = 502 minutes = 8.4 hours for the solver (vegen.f) alone.

The overall structure of the asymmetrical band matrix can be visualized with a separate Fortran program called 'PlotIt.f'. In order to use it, you have to change the 'Direct' MAIN program in at least two places:

```
c     , access='direct',recl=4*(2*NB+1),disp='delete')  BECOMES:
      , access='direct',recl=4*(2*NB+1)) ! File will be on disk
INSERT:
      stop 'effe'
JUST BEFORE:
*
*     SOLVATION :-)
```

The MAIN program then will leave on disk the big matrix just after Assembly. Now run "PlotIt" and send "fort.7" to a PostScript printer. And ... don't forget to restore the original MAIN program!

It is easily conceived from the picture that the observed inefficiency must be associated with fill-in of the very many zeros within the banded storage. The PostScript-file (making up the figure) represents the matrix $S(i, j)$ of the equations to be solved in our program. The x-direction corresponds to the index (i), the y-direction (with its origin at the top) corresponds to the index

(j). The non-zero entries in $S(i,j)$ are respresented by a black dot. As can be seen, most of the matrix is blank, that is: filled with zeros. (It's a so called SPARSE matrix.) During pivoting (in "vegen") the whole area between the two "boundary lines" will suffer from "fill-in", which means that it will be "painted black" with additional non-zero numbers. All these will take part in the computational process. Crunch, crunch ...

```
Unknowns :        2535
Bandwidth:         192
Incore ..:      633460
Outcore .:     3903900

  0.7083941    % of Full
  4.664361     % of Banded
  39 = maximum number coefficients / row
```

The frustrating thing is that less than 5 percent of the MegaBytes needed is filled with non-zero coefficients, just after Assembly. Then pivoting comes in (with "vegen") and fills in this space for more than 85 percent!

Iterative solver

Instead of a Direct Solver, as has been used so far, there is the possibility of implementing an Iterative Solver. Due to lack of experience, I hesitated to do it for a long time. But now I've finished a few programs, and the result has been surprising to me! Iterative methods, as compared to a direct method, seem to be FAR better for the bigger problems! Our method is called "Gauss- Seidel with successive over-relaxation". It's nothing new, actually ...

Crucial part of the code is the Gauss pre-processor, which in fact is nothing but a large ("Topology ...") Sorting & Searching program. The "Gauss" program is surprisingly fast, nevertheless, because I've made an *explicit* use of the problem's structure. This has been a very dedicated programming exercise. It also means that you *can't easily change* the existing topology of the of the calculation domain, being a half-sphere.

The following serves as background info. The merging algorithm of "inbreng.f" was developed because I was in the need of sorting out a huge collection of Midi(music) files. The accompanying BASIC program was published on the Net. Now I've known for a long time that using iterative methods for finite element problems involves kind of a sorting procedure, which seemed *prohibiting* at first sight. But then suddenly it occurred to me that I could use ... the same algorithm as with the Midi files! Which turned out to be very efficient, for quite another purpose. The original BASIC program, called 'midiot.bas' is included herewith. If you are not a MIDIot, then it's useless. If you are, then you'd better stop collecting everything from the net, so that you don't have to use it anyway :-)

As has been stated previously, the iterative method needs some pre-processing first. This is the "Gauss" code. All of the Geometry is contained in this part. The PreProcessor writes (quite a lot of) data, called "Gauss.dat" to disk. After "Gauss", guess what, "Seidel" must be invoked *several times.* This program also contains the Boundary Conditions, and a couple of iteration parameters, which can be adjusted. "Seidel" leaves it's "Seidel.dat" data on disk for restart. Hence you must do "Gauss Seidel Seidel Seidel ..." until convergence occurs.

```
  MOST = number of iterations within 1 run
  EPS  = criterion for convergence
  OO   = over-relaxation factor
```

I have already made a choice for these numbers. Especially the "OO" factor, usually denoted as ω, is rather critical. A good choice will accelerate the iterations a great deal. Quoted without permission: "In general, it is not possible to compute in advance the value of ω that is optimal with respect to the rate of convergence [...] Frequently, some heuristic estimate is used, such as $\omega = 2 - O(h)$ where h is the mesh spacing of the discretization of the underlying physical domain." Be aware of the possibility that the value of ω probably must be changed as soon as the physics is altered in subroutine COEFFS. The ω in the program was determined by experiment: simply by looking at the convergence with different values of it. We think that's the best strategy anyway. The ω number *must* be in the interval [0,2], thus: $0 \leq \omega < 2$, but people say it's better to over-estimate than to under-estimate it. (Thus, instead of "1.82" you might try "1.83" but not "1.81" ;-)

An advantage of iterative methods is also that you can see what the solution is going to look like before obtaining full "accurate" results. For example, first choose EPS = 1.E-4 and then refine the solution with EPS=1.E-5.

It's a slow mental process, but it gradually dawns to me that I have made the wrong choice by implementing a direct solver in the first version of the CR program. Following below is a couple of measurements. No use has been made of Convex 3840 vector-processing; hence pure scalar performance, as will be the case with a moderate workstation (such as a RS-6000 machine).

```
Grid = 18 x 18 x 12 :

Direct method: 404.5 real   359.1 user + 15.8 sys =  374.9

Iterative method:

 Gauss:        14.6 real    13.8 user     0.2 sys
Seidel:        19.9 real    16.9 user     0.6 sys
                            ----            ---
                            30.7    +     0.8     =  31.1 total
```

```
Comparison = 374.9 / 31.1 =  12  times faster !!!


Grid = 35 x 35 x 24 :

Direct method:

 Pivot ............................*** CPU time limit exceeded
 because the CPU time limit of our long queue is 5 hours.
 By counting dots we see that the direct solver has done only
 28/72 'th part of the work to be done. By extrapolation we
 get an estimate of  > 13  hours. This number is also obtained
 by multiplying the result of the 18x18x12 grid with 2^7 :
 128 x 374.9 /3600 = 13.33 hours.

Iterative method:

 Gauss:      130.7 real   121.1 user  2.1 sys
Seidel:      559.2 real   515.5 user  8.4 sys
                          -----       ----
                          636.6   +  10.5  =  647.1  total

Hence the improvement in performance comes close to a factor
of ... 75 !!!
```

So, iterative methods are really sensational! I get the unbelievable result that computations will be completed in 10 minutes instead of 10 hours! Also the amount of core memory needed should be hardly a problem for a moderate workstation: approximately 5 MB for a 35x35x24 grid.

Appendix

How we met ... on the Net

Collaboration between the authors of this paper wouldn't even have come into existence without the Internet. Over a hundred E-mails have been exchanged during this little project. The correspondence was started with a poster in one of the Internet newsgroups. Here is the original message:

```
From sci.math.num-analysis Thu Mar 18 08:41:34 1993
From: fichtner@acs.ucalgary.ca (Horst Fichtner)
Subject: 3-D PDE on non-rectangular domain
Organization: The University of Calgary, Alberta

I want to solve numerically an elliptical PDE on a
three-dimensional domain.
```

Rest of message deleted. The request was answered on 93 Mar 19, as follows:

```
From: Han.deBruijn@rc.tudelft.nl (Han de Bruijn)

I am in the process to develop such a code. If you describe
to me in some more detail what your problem is, [ ... ]
```

The collaboration then started by "setting the stage": how to explain the physics of CR to a non-astronomer in the first place? It also became clear that the CR problem could not be solved by a kind of "general" code. Hence it was decided to develop a quite *dedicated* computer-program instead. The exchange of mathematics "at a distance" was accomplished in an efficient manner with help of LaTeX. The following splitting of responsibilities has proven to be effective. A numerical method for solving convection-diffusion equations of the CR type can be developed, quite independently of whatever physical details are actually involved. Because the physics is mainly in the coefficients of the equation, the latter can be separated a great deal from the equation itself and separately coded. Modular programming is not luxury, but merely a necessity when people with such different backgrounds are going to collaborate! At last the program, with exception of the "coeffs" routine, was "shipped", by e-mail of course, from Delft University to the University of Maryland in the beginning of May 1995:

```
From horst@aix.umd.edu Sat May  6 21:54:25 1995
Subject: Nice performance...
```

This concerned the version with the direct solver in it. The iterative programs were sent a couple of months later. And the following response was received:

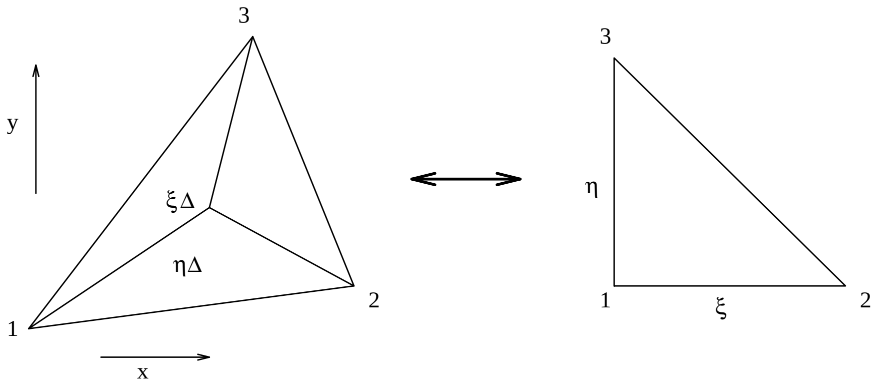
```
From hfichtne@wam.umd.edu Fri Oct  6 22:10:44 1995
Subject: Flabbergasted...
```

Triangle Isoparametrics

Isoparametric transformation is the standard approach where the Finite Element Method relies on when it has to deal with curved geometries: its strong point. Let's consider first the simplest non-trivial finite element shape in two dimensions, the linear triangle, and see where isoparametric transformations actually come from. Function behaviour is approximated inside such a triangle by a *linear* interpolation between the function values at the vertices, also called nodal points. Let T be such a function, and x, y coordinates, then:

$$T = A.x + B.y + C$$

Where the constants A, B, C are yet to be determined.



Substitute $x = x_k$ and $y = y_k$ with $k = 1, 2, 3$:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} C \\ A \\ B \end{bmatrix}$$

Start with:

$$T_1 = A.x_1 + B.y_1 + C$$

Clearly, the first of these equations can already be used to eliminate the constant C, once and forever:

$$T - T_1 = A.(x - x_1) + B.(y - y_1)$$

Then the constants A and B are determined by:

$$\begin{aligned} T_2 - T_1 &= A.(x_2 - x_1) + B.(y_2 - y_1) \\ T_3 - T_1 &= A.(x_3 - x_1) + B.(y_3 - y_1) \end{aligned}$$

Two equations with two unknowns. The solution is found by Cramer's rule:

$$\begin{aligned} A &= [(y_3 - y_1).(T_2 - T_1) - (y_2 - y_1).(T_3 - T_1)]/\Delta \\ B &= [(x_2 - x_1).(T_3 - T_1) - (x_3 - x_1).(T_2 - T_1)]/\Delta \end{aligned}$$

There are several forms of the determinant Δ, which should be memorized when it is appropriate:

$$\begin{aligned}
\Delta &= (x_2 - x_1).(y_3 - y_1) - (x_3 - x_1).(y_2 - y_1) \\
\Delta &= 2 \times \text{ triangle area} \\
\Delta &= x_1.y_2 + x_2.y_3 + x_3.y_1 - y_1.x_2 - y_2.x_3 - y_3.x_1 \\
\Delta &= x_1.(y_2 - y_3) + x_2.(y_3 - y_1) + x_3.(y_1 - y_2) \\
\Delta &= y_1.(x_3 - x_2) + y_2.(x_1 - x_3) + y_3.(x_2 - x_1) \\
\Delta &= \left\| \begin{array}{ccc} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{array} \right\|
\end{aligned}$$

Anyway, it is concluded that:

$$T - T_1 = \xi.(T_2 - T_1) + \eta.(T_3 - T_1)$$

Where:

$$\begin{aligned}
\xi &= [(y_3 - y_1).(x - x_1) - (x_3 - x_1).(y - y_1)]/\Delta \\
\eta &= [(x_2 - x_1).(y - y_1) - (y_2 - y_1).(x - x_1)]/\Delta
\end{aligned}$$

Or:

$$\begin{bmatrix} \xi \\ \eta \end{bmatrix} = \begin{bmatrix} +(y_3 - y_1) & -(x_3 - x_1) \\ -(y_2 - y_1) & +(x_2 - x_1) \end{bmatrix} /\Delta \begin{bmatrix} x - x_1 \\ y - y_1 \end{bmatrix}$$

The inverse of the following problem is recognized herein:

$$\begin{bmatrix} x - x_1 \\ y - y_1 \end{bmatrix} = \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix}$$

Or:

$$\begin{aligned}
x - x_1 &= \xi.(x_2 - x_1) + \eta.(x_3 - x_1) \\
y - y_1 &= \xi.(y_2 - y_1) + \eta.(y_3 - y_1)
\end{aligned}$$

But also:

$$T - T_1 = \xi.(T_2 - T_1) + \eta.(T_3 - T_1)$$

Therefore the *same* expression holds for the function T as well as for the coordinates x and y . This is precisely what people mean by an *isoparametric* ("same parameters") transformation.

Now recall the formulas which express ξ and η into x and y :

$$\begin{aligned}
\xi &= [(y_3 - y_1).(x - x_1) - (x_3 - x_1).(y - y_1)]/\Delta \\
\eta &= [(x_2 - x_1).(y - y_1) - (y_2 - y_1).(x - x_1)]/\Delta
\end{aligned}$$

Thus ξ can be interpreted as: area of the sub-triangle spanned by the vectors $(x-x_1, y-y_1)$ and (x_3-x_1, y_3-y_1) divided by the whole triangle area. And η can be interpreted as: area of the sub-triangle spanned by the vectors $(x-x_1, y-y_1)$

and $(x_2 - x_1, y_2 - y_1)$ divided by the whole triangle area. This is the reason why ξ and η are sometimes called *area-coordinates*; see the figure above, where (two times) the area of the triangle as a whole is denoted as Δ. There are even *three* of these coordinates in literature; see [**?**]. The third area-coordinate is, of course, dependent on the other two, being equal to $(1 - \xi - \eta)$. Instead of area-coordinates, we prefer to talk about *local coordinates* ξ and η of an element, in contrast to the *global coordinates* x and y. It is possible that local coordinates coincide with the global coordinates. A triangle for which this is the case is called a *parent element*. The portrait of the parent triangle is also depicted in the above figure: it is rectangular, and two sides of it are equal.

Let's reconsider for a moment the expression:

$$T - T_1 = \xi.(T_2 - T_1) + \eta.(T_3 - T_1)$$

Partial differentiation to ξ and η gives:

$$\partial T/\partial\xi = T_2 - T_1 \quad ; \quad \partial T/\partial\eta = T_3 - T_1$$

Therefore, with node (1) as the origin, hence $T(0) = T_1$:

$$T = T(0) + \xi\frac{\partial T}{\partial\xi} + \eta\frac{\partial T}{\partial\eta}$$

This is part of a Taylor series expansion around node (1). Such Taylor series expansions are very common in Finite Difference analysis. Now rewrite as follows:

$$T = (1 - \xi - \eta).T_1 + \xi.T_2 + \eta.T_3$$

Here the functions $(1 - \xi - \eta)$, ξ, η are called the *shape functions* of the Finite Element. Shape functions N_k have the property that they are unity in one of the nodes (k), and zero in all other nodes. In our case:

$$N_1 = 1 - \xi - \eta \quad ; \quad N_2 = \xi \quad ; \quad N_3 = \eta$$

So we have two representations, which are allmost trivially equivalent:

$$\begin{array}{ll} T = T_1 + \xi.(T_2 - T_1) + \eta.(T_3 - T_1) & : \text{Finite Difference like} \\ T = (1 - \xi - \eta).T_1 + \xi.T_2 + \eta.T_3 & : \text{Finite Element like} \end{array}$$

What kind of terms can be discretized at the domain of a linear triangle? In the first place, the function $T(x, y)$ itself, of course. But one may also try on the first order partial derivatives $\partial T/\partial x$, $\partial T/\partial y$. We find:

$$\begin{array}{l} \partial T/\partial x = A = [(y_3 - y_1).(T_2 - T_1) - (y_2 - y_1).(T_3 - T_1)]/\Delta \\ \partial T/\partial y = B = [(x_2 - x_1).(T_3 - T_1) - (x_3 - x_1).(T_2 - T_1)]/\Delta \end{array}$$

By collecting terms belonging to the same T_k, this can also be written as:

$$\Delta \begin{bmatrix} \partial T/\partial x \\ \partial T/\partial y \end{bmatrix} = \begin{bmatrix} +(y_2 - y_3) & +(y_3 - y_1) & +(y_1 - y_2) \\ -(x_2 - x_3) & -(x_3 - x_1) & -(x_1 - x_2) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Or, in operator form:

$$\begin{bmatrix} \partial/\partial x \\ \partial/\partial y \end{bmatrix} = \begin{bmatrix} +(y_2 - y_3) & +(y_3 - y_1) & +(y_1 - y_2) \\ -(x_2 - x_3) & -(x_3 - x_1) & -(x_1 - x_2) \end{bmatrix} / \Delta$$

The right hand side will be called a *Differentiation Matrix* in subsequent work. Thus the gradient operator is represented at a linear triangle by a 2×3 Differentiation Matrix.

Bibliography

[1] Horst Fichtner, Hans Jörg Fahr, S. Ranga Sreenivasan, "The Influence of a Non-Spherical Solar Wind Termination Shock on the Pressure Distribution of the Anomalous Component of Cosmic Rays in the Heliosphere", Proceedings of the 23rd International Cosmic ray Conference (1993).

[2] Han de Bruijn, ftp.rc.tudelft.nl:pub/pc/msdos/misc/suna.zip

[3] O.C. Zienkiewicz, "The Finite Element Method", 3th edition, Mc.Graw-Hill U.K. 1977.

[4] S.V. Patankar, "Numerical Heat Transfer and Fluid Flow", Hemisphere Publishing Company U.S.A. 1980.