1 BACKGROUND – THE EXPLICIT FINITE DIFFERENCE METHOD

1.1 An Explanation of Terms and Concepts

Since FLAC is described as an “explicit, finite difference program” that performs a “Lagrangian analysis,” we examine these terms first and describe their relevance to the process of numerical modeling.*

1.1.1 Finite Differences

The finite difference method is perhaps the oldest numerical technique used for the solution of sets of differential equations, given initial values and/or boundary values (see, for example, Desai and Christian 1977). In the finite difference method, every derivative in the set of governing equations is replaced directly by an algebraic expression written in terms of the field variables (e.g., stress or displacement) at discrete points in space; these variables are undefined within elements.

In contrast, the finite element method has a central requirement that the field quantities (stress, displacement) vary throughout each element in a prescribed fashion, using specific functions controlled by parameters. The formulation involves the adjustment of these parameters to minimize error terms or energy terms.

Both methods produce a set of algebraic equations to solve. Even though these equations are derived in quite different ways, it is easy to show (in specific cases) that the resulting equations are identical for the two methods. It is pointless, then, to argue about the relative merits of finite elements or finite differences: the resulting equations are the same.

However, over the years, certain “traditional” ways of doing things have taken root: for example, finite element programs often combine the element matrices into a large global stiffness matrix, whereas this is not normally done with finite differences because it is relatively efficient to regenerate the finite difference equations at each step. As explained below, FLAC uses an “explicit,” time-marching method to solve the algebraic equations, but implicit, matrix-oriented solution schemes are more common in finite elements. Other differences are also common, but it should be stressed that features may be associated with one method rather than another because of habit more than anything else.

Finally, we must dispose of one persistent myth. Many people (including some who write textbooks) believe that finite differences are restricted to rectangular grids. This is not true! Wilkins (1964)...
presented a method of deriving difference equations for elements of any shape. This method, also described as the “finite volume method,” is used in FLAC. The erroneous belief that finite differences and rectangular grids are inseparable is responsible for many statements concerning boundary shapes and distribution of material properties. Using Wilkins’ method, boundaries can be any shape, and any element can have any property value – just like finite elements.

1.1.2 Explicit, Time-Marching Scheme

Even though we want FLAC to find a static solution to a problem, the dynamic equations of motion are included in the formulation. One reason for doing this is to ensure that the numerical scheme is stable when the physical system being modeled is unstable. With nonlinear materials, there is always the possibility of physical instability (e.g., the sudden collapse of a pillar). In real life, some of the strain energy in the system is converted into kinetic energy, which then radiates away from the source and dissipates. FLAC models this process directly, because inertial terms are included – kinetic energy is generated and dissipated. In contrast, schemes that do not include inertial terms must use some numerical procedure to treat physical instabilities. Even if the procedure is successful at preventing numerical instability, the path taken may not be a realistic one. One penalty for including the full law of motion is that the user must have some physical feel for what is going on. FLAC is not a black box that will give “the solution”; the behavior of the numerical system must be interpreted. Some guidelines are provided in Section 3.9 in the User’s Guide to assist in doing this.

The general calculation sequence embodied in FLAC is illustrated in Figure 1.1. This procedure first invokes the equations of motion to derive new velocities and displacements from stresses and forces. Then, strain rates are derived from velocities, and new stresses from strain rates. We take one timestep for every cycle around the loop. The important thing to realize is that each box in Figure 1.1 updates all of its grid variables from known values that remain fixed while control is within the box. For example, the lower box takes the set of velocities already calculated and, for each element, computes new stresses. The velocities are assumed to be frozen for the operation of the box (i.e., the newly calculated stresses do not affect the velocities). This may seem unreasonable because we know that if a stress changes somewhere, it will influence its neighbors and change their velocities. However, we choose a timestep so small that information cannot physically pass from one element to another in that interval. (All materials have some maximum speed at which information can propagate.) Since one loop of the cycle occupies one timestep, our assumption of “frozen” velocities is justified – neighboring elements really cannot affect one another during the period of calculation. Of course, after several cycles of the loop, disturbances can propagate across several elements, just as they would propagate physically.
The previous paragraph contains a descriptive statement of the explicit method; later on, a mathematical version will be provided. The central concept is that the calculational “wave speed” always keeps ahead of the physical wave speed, so that the equations always operate on known values that are fixed for the duration of the calculation. There are several distinct advantages to this (and at least one big disadvantage!): most importantly, no iteration process is necessary when computing stresses from strains in an element, even if the constitutive law is wildly nonlinear. In an implicit method (which is commonly used in finite element programs), every element communicates with every other element during one solution step: several cycles of iteration are necessary before compatibility and equilibrium are obtained. Table 1.1 compares the explicit and implicit methods. The disadvantage of the explicit method is seen to be the small timestep, which means that large numbers of steps must be taken. Overall, explicit methods are best for ill-behaved systems (e.g., nonlinear, large-strain, physical instability); they are not efficient for modeling linear, small-strain problems.
### Table 1.1 Comparison of explicit and implicit solution methods

<table>
<thead>
<tr>
<th>Explicit</th>
<th>Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestep must be smaller than a critical value for stability.</td>
<td>Timestep can be arbitrarily large, with unconditionally stable schemes.</td>
</tr>
<tr>
<td>Small amount of computational effort per timestep.</td>
<td>Large amount of computational effort per timestep.</td>
</tr>
<tr>
<td>No significant numerical damping introduced for dynamic solution.</td>
<td>Numerical damping dependent on timestep present with unconditionally stable schemes.</td>
</tr>
<tr>
<td>No iterations necessary to follow nonlinear constitutive law.</td>
<td>Iterative procedure necessary to follow nonlinear constitutive law.</td>
</tr>
<tr>
<td>Provided that the timestep criterion is always satisfied, nonlinear laws are always followed in a valid physical way.</td>
<td>Always necessary to demonstrate that the above-mentioned procedure is: (a) stable; and (b) follows the physically correct path (for path-sensitive problems).</td>
</tr>
<tr>
<td>Matrices are never formed. Memory requirements are always at a minimum.</td>
<td>Stiffness matrices must be stored. Ways must be found to overcome associated problems such as bandwidth. Memory requirements tend to be large.</td>
</tr>
<tr>
<td>Since matrices are never formed, large displacements and strains are accommodated without additional computing effort.</td>
<td>Additional computing effort needed to follow large displacements and strains.</td>
</tr>
</tbody>
</table>

#### 1.1.3 Lagrangian Analysis

Since we do not need to form a global stiffness matrix, it is a trivial matter to update coordinates at each timestep in large-strain mode. The incremental displacements are added to the coordinates so that the grid moves and deforms with the material it represents. This is termed a “Lagrangian” formulation, in contrast to an “Eulerian” formulation, in which the material moves and deforms relative to a fixed grid. The constitutive formulation at each step is a small-strain one, but is equivalent to a large-strain formulation over many steps.
1.1.4 Plasticity Analysis

A common question is whether FLAC is better-suited than a finite element method (FEM) program for plasticity analysis. There are many thousands of FEM programs and hundreds of different solution schemes. Therefore, it is impossible to make general statements that apply to “The Finite Element Method.” In fact, there may be so-called finite element codes that embody the same solution scheme as FLAC (as described above in Section 1.1.2). Such codes should give results identical to FLAC.

FEM codes usually represent steady plastic flow by a series of static equilibrium solutions. The quality of the solution for increasing applied displacements depends on the nature of the algorithm used to return stresses to the yield surface, following an initial estimate using linear stiffness matrices. The best FEM codes will give a limit load (for a perfectly plastic material) that remains constant with increasing applied displacement. The solution provided by these codes will be similar to that provided by FLAC. However, FLAC’s formulation is simpler because no algorithm is necessary to bring the stress of each element to the yield surface: the plasticity equations are solved exactly in one step. (For details, see Section 2.4.) Therefore, FLAC may be more robust and more efficient than some FEM codes for modeling steady plastic flow.

FLAC is also robust in the sense that it can handle any constitutive model with no adjustment to the solution algorithm; many FEM codes need different solution techniques for different constitutive models.

For further information, we recommend the publication by Frydman and Burd (1997), which compares FLAC to one FEM code and concludes that FLAC is superior in some respects for footing problems (e.g., efficiency and smoothness of the pressure distribution).
1.2 Field Equations

The solution of solid-body, heat-transfer or fluid-flow problems in FLAC invokes the equations of motion and constitutive relations, Fourier’s Law for conductive heat transfer, and Darcy’s Law for fluid flow in a porous solid, as well as boundary conditions. This section reviews the basic governing equations for the solid body; corresponding equations for groundwater and thermal problems are provided in Section 1 in Fluid-Mechanical Interaction and Section 1 in Thermal Analysis, respectively. The same method of generating finite difference equations applies to all sets of differential equations.

1.2.1 Motion and Equilibrium

In its simplest form, the equation of motion relates the acceleration, \( \dot{\ddot{u}}/dt \), of a mass, \( m \), to the applied force, \( F \), which may vary with time. Figure 1.2 illustrates a force acting on a mass, causing motion described in terms of acceleration, velocity and displacement.

Newton’s law of motion for the mass-spring system is

\[
m \frac{d\ddot{u}}{dt} = F
\]  

(1.1)

When several forces act on the mass, Eq. (1.1) also expresses the static equilibrium condition when the acceleration tends to zero (i.e., \( \sum F = 0 \), where the summation is over all acting forces). This property of the law of motion is exploited in FLAC when solving “static” problems. Note that the conservation laws (of momentum and energy) are implied by Eq. (1.1), since they may be derived from it (and Newton’s other two laws).
In a continuous solid body, Eq. (1.1) is generalized as follows:

\[ \rho \frac{\partial \dot{u}_i}{\partial t} = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \]  

(1.2)

where: \( \rho \) = mass density;
\( t \) = time;
\( x_i \) = components of coordinate vector;
\( g_i \) = components of gravitational acceleration (body forces); and
\( \sigma_{ij} \) = components of stress tensor.

In this equation, and those that follow, indices \( i \) denote components in a Cartesian coordinate frame, and summation is implied for repeated indices in an expression.

### 1.2.2 Constitutive Relation

The other set of equations that apply to a solid, deformable body is known as the constitutive relation, or stress/strain law. First, strain rate is derived from velocity gradient as follows:

\[ \dot{\varepsilon}_{ij} = \frac{1}{2} \left[ \frac{\partial \dot{u}_i}{\partial x_j} + \frac{\partial \dot{u}_j}{\partial x_i} \right] \]  

(1.3)

where: \( \dot{\varepsilon}_{ij} \) = strain-rate components; and
\( \dot{u}_i \) = velocity components.

Mechanical constitutive laws are of the form

\[ \sigma_{ij} := M(\sigma_{ij}, \dot{\varepsilon}_{ij}, \kappa) \]  

(1.4)

where: \( M(\ ) \) is the functional form of the constitutive law;
\( \kappa \) is a history parameter(s) which may or may not be present, depending on the particular law; and
\( := \) means “replaced by.”

In general, nonlinear constitutive laws are written in incremental form because there is no unique relation between stress and strain. Eq. (1.4) provides a new estimate for the stress tensor, given the
old stress tensor and the strain rate (or strain increment). The simplest example of a constitutive law is that of isotropic elasticity:

$$\sigma_{ij} := \sigma_{ij} + \left\{ \delta_{ij} \left( K - \frac{2}{3} G \right) \dot{e}_{kk} + 2 G \dot{e}_{ij} \right\} \Delta t$$

(1.5)

where:

- $\delta_{ij}$ is the Kronecker delta;
- $\Delta t$ = timestep; and
- $G$, $K$ = shear and bulk modulus, respectively.

The particular formulation for each constitutive law in FLAC is provided in Section 2.

1.2.3 Frame Indifference

There is another contribution to the stress tensor, due to the finite rotation of a zone during one timestep: the stress components referred to the fixed frame of reference change as follows:

$$\sigma_{ij} := \sigma_{ij} + (\omega_{ik} \sigma_{kj} - \sigma_{ik} \omega_{kj}) \Delta t$$

(1.6)

where

$$\omega_{ij} = \frac{1}{2} \left\{ \frac{\partial \dot{u}_i}{\partial x_j} - \frac{\partial \dot{u}_j}{\partial x_i} \right\}$$

(1.7)

The adjustment of Eq. (1.6) is only done in large-strain mode and is, in fact, applied before Eq. (1.5). Stress adjustments due to other finite strain components are not made.

1.2.4 Boundary Conditions

Either stress or displacement may be applied at the boundary of a solid body in FLAC. Displacements are specified in terms of prescribed velocities at given gridpoints; Eq. (1.2) is not invoked at those gridpoints. At a stress boundary, forces are derived as follows:

$$F_i = \sigma^b_{ij} n_j \Delta s$$

(1.8)

where $n_j$ is the unit outward normal vector of the boundary segment, and $\Delta s$ is the length of the boundary segment over which the stress $\sigma^b_{ij}$ acts. The force $F_i$ is added into the force sum for the appropriate gridpoint, described in Section 1.3.5.
1.3 Numerical Formulation

1.3.1 Introduction

This section presents the finite difference form of the field equations provided in the previous section. FLAC’s formulation is conceptually similar to that of dynamic relaxation (proposed by Otter et al. 1966), with adaptations for arbitrary grid shapes, large-strains and different damping. The finite difference scheme follows the approach of Wilkins (1964).

1.3.2 The Grid

The solid body is divided by the user into a finite difference mesh composed of quadrilateral elements. Internally, FLAC subdivides each element into two overlaid sets of constant-strain triangular elements, as shown in Figure 1.3.

The four triangular sub-elements are termed a, b, c and d. As explained in Section 1.3.3.2, the deviatoric stress components of each triangle are maintained independently, requiring sixteen stress components to be stored for each quadrilateral ($4 \times \sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}$). The force vector exerted on each node is taken to be the mean of the two force vectors exerted by the two overlaid quadrilaterals. In this way, the response of the composite element is symmetric, for symmetric loading. If one pair of triangles becomes badly distorted (e.g., if the area of one triangle becomes much smaller than the area of its companion), then the corresponding quadrilateral is not used; only nodal forces from the other (more reasonably shaped) quadrilateral are used. If both overlaid sets of triangles are badly distorted, FLAC complains with an error message.

Figure 1.3 (a) Overlaid quadrilateral elements used in FLAC (b) Typical triangular element with velocity vectors (c) Nodal force vector
1.3.3 Finite Difference Equations

The difference equations for a triangle are derived from the generalized form of Gauss’ divergence theorem (e.g., Malvern 1969):

\[
\int_{s} n_i f \, ds = \int_{A} \frac{\partial f}{\partial x_i} \, dA \tag{1.9}
\]

where:
- \(s\) is the integral around the boundary of a closed surface;
- \(n_i\) is the unit normal to the surface, \(s\);
- \(f\) is a scalar, vector or tensor;
- \(x_i\) are position vectors;
- \(ds\) is an incremental arc length; and
- \(A\) is the integral over the surface area.

Defining the average value of the gradient of \(f\) over the area \(A\) as

\[
< \frac{\partial f}{\partial x_i} > = \frac{1}{A} \int_{A} \frac{\partial f}{\partial x_i} \, dA \tag{1.10}
\]

one obtains, by substitution into Eq. (1.9),

\[
< \frac{\partial f}{\partial x_i} > = \frac{1}{A} \int_{s} n_i f \, ds \tag{1.11}
\]

For a triangular sub-element, the finite difference form of Eq. (1.11) becomes

\[
< \frac{\partial f}{\partial x_i} > = \frac{1}{A} \sum_{s} < f > n_i \Delta s \tag{1.12}
\]

where \(\Delta s\) is the length of a side of the triangle, and the summation occurs over the three sides of the triangle. The value of \(< f >\) is taken to be the average over the side.
1.3.3.1 Strain Rates and Strains

Eq. (1.12) enables strain rates, \( \dot{e}_{ij} \), to be written in terms of nodal velocities for a triangular subzone by substituting the average velocity vector of each side for \( f \). (The strain rate for the zone is the average for the values of the triangular subzones.) The equations are:

\[
\frac{\partial \dot{u}_i}{\partial x_j} \approx \frac{1}{2A} \sum_s \left( \dot{u}_i^{(a)} + \dot{u}_i^{(b)} \right) n_j \Delta s \tag{1.13}
\]

\[
\dot{e}_{ij} = \frac{1}{2} \left[ \frac{\partial \dot{u}_i}{\partial x_j} + \frac{\partial \dot{u}_j}{\partial x_i} \right] \tag{1.14}
\]

where the summation is over the sides of the triangular subzone, and \((a)\) and \((b)\) are two consecutive nodes on a side. Note that the expression Eq. (1.13) is identical to that derived by exact integration if there is a linear variation in velocity between nodes.

Eqs. (1.13) and (1.14) can be used to derive all the components of the strain rate tensor based on nodal velocities. (The exception is for the plane stress calculation: the out-of-plane strain rate is not directly calculated in FLAC.) Similarly, the strain tensor is derived by substituting nodal displacements for velocities in Eqs. (1.13) and (1.14).

For the purposes of printing and plotting, the term “maximum shear strain” means the radius of the Mohr’s circle in the \( xy \)-plane, as illustrated in Figure 1.4:

![Figure 1.4 Mohr’s circle of strain](image)

Thus, for conditions of two-dimensional plane-strain analysis, the maximum shear strain, \( \gamma \), is defined as
\[ \gamma = \frac{1}{2} \left( (e_{xx} - e_{yy})^2 + 4e_{xy}^2 \right)^{1/2} \] (1.15)

This is the equation used for calculating the maximum shear strain values, \( \text{ssi} \) (strains derived from displacements; average value of subzones) and \( \text{ssr} \) (strains based on velocities; average value of subzones), accessed by the \text{PRINT}, \text{PLOT} and \text{HISTORY} commands, or via \text{FISH}, when running in plane-strain mode.

In three dimensions, a measure for maximum shear strain, \( \gamma \), is given by the square root of the second invariant of the strain deviator tensor, \( J'_2 \) – i.e.,

\[ \gamma = \sqrt{J'_2} = \sqrt{\frac{1}{6} \left[ (e_{xx} - e_{yy})^2 + (e_{yy} - e_{zz})^2 + (e_{zz} - e_{xx})^2 \right] + e_{xy}^2 + e_{yz}^2 + e_{zx}^2} \] (1.16)

Eq. (1.16) is used in the calculation for shear strain values, \( \text{ssi} \) and \( \text{ssr} \), when running in axisymmetry mode. The three-dimensional values can also be obtained when running in plane-strain mode by using the keywords \( \text{ssi3d} \) and \( \text{ssr3d} \) in place of \( \text{ssi} \) and \( \text{ssr} \). Note that the three-dimensional measure of shear strain, Eq. (1.16), does not degenerate to the two-dimensional form, Eq. (1.15), when the out-of-plane components of strain are zero (i.e., when \( e_{zz} = e_{yz} = e_{xz} = 0 \)).*

Additional FLAC zone variables are available to access strain rates and strains (see \text{Strain Calculations} in Section 2.5.3 in the \text{FISH} volume). Volumetric strain rate, \( \text{vsr} (= \dot{e}_{xx} + \dot{e}_{yy} + \dot{e}_{zz}) \), and volumetric strain, \( \text{vsi} (= e_{xx} + e_{yy} + e_{zz}) \), are provided. \text{FISH} functions \( \text{fsr} \) and \( \text{fsi} \) calculate all the tensor components for the full strain rate and strain increment tensors.

The following simple example (Example 1.1) demonstrates the application of these variables, and functions to monitor strains in an unconfined elastic material subjected to gravity loading. The shear strain rates and shear strains, and volumetric strain rates and volumetric strains, calculated from the tensor components, are compared to \( \text{ssr} \), \( \text{ssi} \), \( \text{ssr3d} \), \( \text{ssi3d} \), \( \text{vsr} \) and \( \text{vsi} \) in Example 1.1:

\textit{Example 1.1 Test of FISH strain measures}

| ;--- Test of FISH strain measures --- |
| conf ext 6 |
| grid 5 5 |
| m e |
| pro d 1000 s 1e8 b 2e8 |

* For plane-stress mode, the maximum shear strain values are not conventional: \( \text{ssi} \) and \( \text{ssr} \) values are produced using Eq. (1.15), and do not include the out-of-plane strain, \( e_{zz} \). \( \text{ssi3d} \) and \( \text{ssr3d} \) values are produced using Eq. (1.16), assuming the out-of-plane strains are zero. However, the out-of-plane strains are not zero for plane-stress analysis; they are dependent upon the constitutive model, and only available internally within each model.
set grav 10
fix x y j=1
cyc 100
def qqq
array ar(4) ai(4)
loop i (1,izones)
  loop j (1,jzones)
    dum = fsr(i,j,ar)
    dum = fsi(i,j,ai)
    ex_1(i,j) = sqrt((ar(1)-ar(2))^2 + 4.0 * ar(4)^2) / 2.0
    ex_2(i,j) = sqrt((ai(1)-ai(2))^2 + 4.0 * ai(4)^2) / 2.0
    ex_3(i,j) = ar(1) + ar(2) + ar(3)
    ex_4(i,j) = ai(1) + ai(2) + ai(3)
    ; ssr in 3D formulation
    _arav = ex_3(i,j)/3.
    _rar11 = ar(1) - _arav
    _rar22 = ar(2) - _arav
    _rar33 = ar(3) - _arav
    _arj2 = (_rar11*_rar11+_rar22*_rar22+_rar33*_rar33)/2.+ar(4)*ar(4)
    ex_5(i,j) = sqrt(_arj2)
    ; ssi in 3D formulation
    _aiav = ex_4(i,j)/3.
    _rai11 = ai(1) - _aiav
    _rai22 = ai(2) - _aiav
    _rai33 = ai(3) - _aiav
    _aij2 = (_rai11*_rai11+_rai22*_rai22+_rai33*_rai33)/2.+ai(4)*ai(4)
    ex_6(i,j) = sqrt(_aij2)
  endLoop
endLoop
end
qqq
;--- to test, give the following commands, line by line, & compare
; print ssr ex_1 zon
; print ssi ex_2 zon
; print vsr ex_3 zon
; print vsi ex_4 zon
; print ssr3d ex_5 zon
; print ssi3d ex_6 zon
1.3.3.2 Mixed Discretization

The use of triangular elements eliminates the problem of hourglass deformations which may occur with constant-strain finite difference quadrilaterals. The term “hourglassing” comes from the shape of the deformation pattern of elements within a mesh. For polygons with more than three nodes, there are combinations of nodal displacements which produce no strain and result in no opposing forces. The resulting effect is unopposed deformations of alternating direction.

A common problem which occurs in modeling of materials undergoing yielding is the incompressibility condition of plastic flow. The use of plane-strain or axisymmetric geometries introduces a kinematic restraint in the out-of-plane direction, often giving rise to overprediction of collapse load. This condition is sometimes referred to as “mesh-locking” or “excessively stiff” elements, and is discussed in detail by Nagtegaal et al. (1974). The problem arises as a condition of local mesh incompressibility which must be satisfied during flow, resulting in over-constrained elements. To overcome this problem, the isotropic stress and strain components are taken to be constant over the whole quadrilateral element, while the deviatoric components are treated separately for each triangular sub-element. This procedure, referred to as mixed discretization, is described by Marti and Cundall (1982). The term mixed discretization arises from the different discretizations for the isotropic and deviatoric parts of the stress and strain tensors.

The volumetric strain is averaged over each pair of triangles, while the deviatoric strains remain unchanged. The strain rates in triangles a and b of Figure 1.3(a) are adjusted in the following way, where subscript \( m \) denotes “mean” and subscript \( d \) denotes “deviatoric”:

\[
\dot{e}_m = \frac{\dot{e}_{11}^a + \dot{e}_{22}^a + \dot{e}_{11}^b + \dot{e}_{22}^b}{2}
\]

\[ (1.17) \]

\[
\dot{e}_d^a = \dot{e}_{11}^a - \dot{e}_{22}^a
\]

\[ (1.18) \]

\[
\dot{e}_d^b = \dot{e}_{11}^b - \dot{e}_{22}^b
\]

FLAC Version 6.0
\[ \dot{e}_{11}^a = \frac{\dot{e}_m + \dot{e}_{d}^a}{2} \]  
\[ \dot{e}_{11}^b = \frac{\dot{e}_m + \dot{e}_{d}^b}{2} \]  
\[ \dot{e}_{22}^a = \frac{\dot{e}_m - \dot{e}_{d}^a}{2} \]  
\[ \dot{e}_{22}^b = \frac{\dot{e}_m - \dot{e}_{d}^b}{2} \]

Similar adjustments are made for triangles c and d. The component \( \dot{e}_{12} \) is unchanged. The above formulation is for plane-strain conditions only. In axisymmetry, all three direct strains are used to derive the mean stress, \( \dot{e}_m \).

### 1.3.3.3 Stresses from Strain Rates

The constitutive law (Eq. (1.4)) and rotation adjustment (Eq. (1.6)) are then used to derive a new stress tensor from the strain-rate tensor. Mixed discretization is invoked again, but on the stresses, in order to equalize isotropic stress between the two triangles in a pair, using area weighting:

\[ \sigma_o^{(a)} = \sigma_o^{(b)} := \left[ \frac{\sigma_o^{(a)} A^{(a)} + \sigma_o^{(b)} A^{(b)}}{A^{(a)} + A^{(b)}} \right] \]  

(1.20)

where \( \sigma_o^{(a)} \) is the isotropic stress in triangle (a); and

\( A^{(a)} \) is the area of triangle (a).

Eq. (1.20) only has an effect for dilatant constitutive laws that produce changes in isotropic stress when shearing occurs; for other laws, the isotropic stresses in the two triangles are already equal.

For the explicit scheme used in FLAC, the constitutive law is only consulted once per zone per timestep. No iterations are necessary because the timestep is small enough that information cannot physically propagate from one zone to the next within one timestep. The estimation of critical timestep is considered in Section 1.3.5.
1.3.3.4 Nodal Forces

Once the stresses have been calculated, the equivalent forces applied to each nodal point are determined. The stresses in each triangular subzone act as tractions on the sides of the triangle. Each traction is taken to be equivalent to two equal forces acting at the ends of the corresponding side. Each triangle corner receives two force contributions, one from each adjoining side (see Figure 1.3 (c)). Hence,

\[ F_i = \frac{1}{2} \sigma_{ij} (n_j^{(1)} S^{(1)} + n_j^{(2)} S^{(2)}) \]  

(1.21)

Recall that each quadrilateral element contains two sets of two triangles. Within each set, the forces from triangles meeting at each node are summed. The forces from both sets are then averaged, to give the nodal force contribution of the quadrilateral.

1.3.3.5 Equations of Motion

At each node, the forces from all surrounding quadrilaterals are summed to give the net nodal force vector, \( \sum F_i \). This vector includes contributions from applied loads, as discussed in Section 1.2.4, and from body forces due to gravity. Gravity forces \( F_i^{(g)} \) are computed from

\[ F_i^{(g)} = g_i m_g \]  

(1.22)

where \( m_g \) is the lumped gravitational mass at the node, defined as the sum of one-third of the masses of triangles connected to the node. If a quadrilateral zone does not exist (e.g., it is null), its stress contribution to \( \sum F_i \) is omitted. If the body is at equilibrium, or in steady-state flow (e.g., plastic flow), \( \sum F_i \) on the node will be zero. Otherwise, the node will be accelerated according to the finite difference form of Newton’s second law of motion:

\[ \dot{u}^{(t+\Delta t/2)}_i = \dot{u}^{(t-\Delta t/2)}_i + \sum F_i^{(t)} \frac{\Delta t}{m} \]  

(1.23)

where the superscripts denote the time at which the corresponding variable is evaluated. For large-strain problems, Eq. (1.23) is integrated again to determine the new coordinate of the gridpoint:

\[ x^{(t+\Delta t)}_i = x^{(t)}_i + \dot{u}^{(t+\Delta t/2)}_i \Delta t \]  

(1.24)

Note that Eqs. (1.23) and (1.24) are both centered in time: it can be shown that first-order error terms vanish for central difference equations. Velocities exist at points in time that are shifted by half a timestep from the displacements and forces.
1.3.4 Mechanical Damping

To solve static problems, the equations of motion must be damped to provide static or quasi-static (non-inertial) solutions. The objective in FLAC is to achieve the steady state (either equilibrium or steady-flow) in a numerically stable way with minimal computational effort. The damping used in standard dynamic relaxation methods is velocity-proportional (i.e., the magnitude of the damping force is proportional to the velocity of the nodes). This is conceptually equivalent to a dashpot fixed to the ground at each nodal point.

The use of velocity-proportional damping in standard dynamic relaxation involves three main difficulties:

1. The damping introduces body forces, which are erroneous in “flowing” regions, and may influence the mode of failure in some cases.

2. The optimum proportionality constant depends on the eigenvalues of the matrix, which are unknown unless a complete modal analysis is done. In a linear problem, this analysis needs almost as much computer effort as the dynamic relaxation calculation itself. In a nonlinear problem, eigenvalues may be undefined.

3. In its standard form, velocity-proportional damping is applied equally to all nodes (i.e., a single damping constant is chosen for the whole grid). In many cases, a variety of behavior may be observed in different parts of the grid. For example, one region may be failing while another is stable. For these problems, different amounts of damping are appropriate for different regions.

In an effort to overcome one or more of these difficulties, alternative forms of damping may be proposed. In soil and rock, natural damping is mainly hysteretic; if the slope of the unloading curve is higher than that of the loading curve, energy may be lost. The type of damping can be reproduced numerically, but there are at least two difficulties. First, the precise nature of the hysteresis curve is often unknown for complex loading-unloading paths. This is particularly true for soils, which are typically tested with sinusoidal stress histories. Cundall (1976) reports that very different results are obtained when the same energy loss is accounted for by different types of hysteresis loops. Second, “ratcheting” can occur (i.e., each cycle in the oscillation of a body causes irreversible strain to be accumulated). This type of damping has been avoided, since it increases path-dependence and makes the results more difficult to interpret.

Adaptive global damping has been described briefly by Cundall (1982). Viscous damping forces are still used, but the viscosity constant is continuously adjusted in such a way that the power absorbed by damping is a constant proportion of the rate of change of kinetic energy in the system. The adjustment to the viscosity constant is made by a numerical servo-mechanism that seeks to keep the following ratio equal to a given ratio (e.g., 0.5):

\[ R = \frac{\sum P}{\sum E_k} \]  \hspace{1cm} (1.25)
where: $P$ is the damping power for a node;

$\dot{E}_k$ is the rate of change of nodal kinetic energy; and

$\sum$ represents the summation over all nodes.

This form of damping overcomes difficulty (2) above, and partially overcomes (1) since, as a system approaches steady state (equilibrium or steady-flow), the rate of change of kinetic energy approaches zero and, consequently, the damping power tends to zero.

**Local Damping** – In order to overcome all three difficulties, a form of damping, called *local nonviscous damping*, in which the damping force on a node is proportional to the magnitude of the unbalanced force, is used in FLAC. The direction of the damping force is such that energy is always dissipated. Eq. (1.23) is replaced by the following equation, which incorporates the local damping scheme:

$$
\dot{u}_i^{(t+\Delta t/2)} = \dot{u}_i^{(t-\Delta t/2)} + \left\{ \sum F_i^{(t)} - (F_d)_i \right\} \frac{\Delta t}{m_n}
$$

(1.26)

where

$$(F_d)_i = \alpha \left| \sum F_i^{(t)} \right| \text{sgn} \left( \dot{u}_i^{(t-\Delta t/2)} \right)
$$

(1.27)

$F_d$ is the damping force, $\alpha$ is a constant (set to 0.8 in FLAC), and $m_n$ is a fictitious nodal mass, derived in Section 1.3.5.

This type of damping is equivalent to a local form of adaptive damping. In principle, the difficulties reported above are addressed: body forces vanish for steady-state conditions; the magnitude of damping constant is dimensionless and is independent of properties or boundary conditions, and the amount of damping varies from point to point (Cundall 1987, pp. 134-135).

Figures 1.5 and 1.6 illustrate typical FLAC results for a problem that involves a suddenly applied compression on the end of a column which is fixed at the opposite end. Figure 1.5 shows the maximum unbalanced force ($\sum F_i$) in the model plotted against number of steps; Figure 1.6 shows the $y$-displacement at the center of the column, just beneath the applied load. Examination of the unbalanced force history shows the progression toward equilibrium (zero unbalanced force). Small oscillations of the system occur as the solution evolves. The damping effects are less evident in the plot of displacement history, which displays a slightly overdamped response.

Note that local damping may also be used for dynamic simulations. See Section 1.4.3.7 in Dynamic Analysis.
**Figure 1.5** Maximum unbalanced force for the problem of sudden end-load application to a column

**Figure 1.6** y-displacement at the center of the column for the problem of sudden end-load application to a column
Combined Damping – A variation on local damping is also provided in FLAC for situations in which the steady-state solution includes a significant uniform motion. This may occur, for example, in a creep simulation or in the calculation of the ultimate capacity of an axially loaded pile. This damping is called combined damping. Combined damping is more efficient than local damping at removing kinetic energy, for this special case.

The damping formulation described by Eq. (1.26) is only activated when the velocity component changes sign. In situations where there is significant uniform motion (in comparison to the magnitude of oscillations that are to be damped), there may be no “zero-crossings,” and hence no energy dissipation.

In order to develop a damping formulation that is insensitive to rigid-body motion, consider periodic motion superimposed on steady motion:

\[ \dot{u} = V \sin(\omega t) + \dot{u}_o \]  \hspace{1cm} (1.28)

where \( V \) is the maximum periodic velocity, \( \omega \) is the angular frequency and \( \dot{u}_o \) is the superimposed steady velocity. Differentiating twice, and noting that \( m\ddot{u} = F \),

\[ \dot{F} = -mV \omega^2 \sin(\omega t) \]  \hspace{1cm} (1.29)

In Eq. (1.29), \( \dot{F} \) is proportional to the periodic part of \( \dot{u} \), without the constant \( \dot{u}_o \). We may substitute \(-\text{sgn}(\dot{F})\) for the damping force in Eq. (1.26) to obtain the same damping force, if the motion is periodic:

\[ F_d = \alpha |F| \text{sgn}(\dot{F}) \]  \hspace{1cm} (1.30)

This equation is insensitive to a constant offset in velocity, since \( \dot{F} \) does not involve \( \dot{u}_o \). In practice, Eq. (1.30) is not as efficient as the local damping force term, Eq. (1.27), if the motion is not strictly periodic. However, the combination of both formulas in equal proportions gives good results:

\[ F_d = \alpha |F| \left( \text{sgn}(\dot{F}) - \text{sgn}(\dot{u}) \right) / 2 \]  \hspace{1cm} (1.31)

This form of damping should be used if there is significant rigid-body motion of a system in addition to oscillatory motion to be dissipated. For this reason, combined damping is the default damping mode for creep analysis. See Section 1.5.10 in Creep Material Models for further discussion and an example application of combined damping. Combined damping is found to dissipate energy at a slower rate compared to local damping based on velocity, and therefore local damping is preferred in most cases.

Rayleigh Damping – For dynamic simulations, “Rayleigh” damping is available; this is described in Section 1.4.3 in Dynamic Analysis.
Hysteretic Damping – Hysteretic damping is also available for dynamic analysis. This form of damping allows strain-dependent modulus and damping functions to be incorporated into the simulation (see Section 1.4.3 in Dynamic Analysis).

1.3.5 Mechanical Timestep Determination: Solution Stability and Mass Scaling

As described previously, the explicit-solution procedure is not unconditionally stable: the speed of the “calculation front” must be greater than the maximum speed at which information propagates. A timestep that is smaller than some critical timestep must be chosen.

The stability condition for an elastic solid discretized into elements of size $\Delta x$ is

$$\Delta t < \frac{\Delta x}{C}$$  \hspace{1cm} (1.32)

where $C$ is the maximum speed at which information can propagate – typically, the $p$-wave speed, $C_p$, where

$$C_p = \sqrt{\frac{K + 4G/3}{\rho}}$$  \hspace{1cm} (1.33)

For a single mass-spring element, the stability condition is

$$\Delta t < 2 \sqrt{\frac{m}{k}}$$  \hspace{1cm} (1.34)

where $m$ is the mass, and $k$ is the stiffness. In a general system, consisting of solid material and arbitrary networks of interconnected masses and springs, the critical timestep is related to the smallest natural period of the system, $T_{\text{min}}$:

$$\Delta t < \frac{T_{\text{min}}}{\pi}$$  \hspace{1cm} (1.35)

It is impractical to determine the eigenperiods of the complete system, so estimates of the local critical timestep are made. This is described below.

Since FLAC is designed to supply the static solution to a problem, the nodal masses may be regarded as relaxation factors in the motion equation, (Eq. (1.26)): they can be adjusted for optimum speed of convergence. Note that gravitational forces are not affected by this scaling of inertial masses (see Eq. (1.22)). The optimum convergence is obtained when the local values of critical timestep are equal (i.e., when the natural response periods of all parts of the system are equal). For convenience,
we set the timestep to unity and adjust nodal masses to obtain this value, assuming a “safety factor” of 0.5 on critical timestep (since it can only be estimated).

Using Eq. (1.32) for a triangular zone of area $A$, and estimating the minimum propagation distance for the zone as $A/\Delta x_{\text{max}}$, we obtain

$$
\Delta t = \frac{A}{C_p \Delta x_{\text{max}}} \quad (1.36)
$$

Substituting $\Delta t = 1$ and $C_p^2 \rho = K + 4G/3$,

$$
\rho = \frac{(K + 4G/3) \Delta x_{\text{max}}^2}{A^2} \quad (1.37)
$$

Noting that the zone mass is $m_z = \rho A$,

$$
m_z = \frac{(K + 4G/3) \Delta x_{\text{max}}^2}{A} \quad (1.38)
$$

Taking the gridpoint mass ($m_{gp}$) of a triangle as one-third of the zone mass,

$$
m_{gp} = \frac{(K + 4G/3) \Delta x_{\text{max}}^2}{3A} \quad (1.39)
$$

Finally, the nodal “mass” of each FLAC gridpoint is the sum of all the connected triangle gridpoint masses:

$$
m_n = \sum \frac{(K + 4G/3) \Delta x_{\text{max}}^2}{6A} \quad (1.40)
$$

where the additional factor of two comes from the inclusion of two sets of overlaid zones in the summation.

The effect of objects such as structural elements and interfaces is included by adding to the summation of Eq. (1.40), equivalent masses computed according to Eq. (1.34), assuming that $\Delta t = 1$; each mechanical element connected to a grid node contributes an extra mass to the summation as follows:

$$
m_{\text{struct}} = 4k \quad (1.41)
$$
where \( k \) is the diagonal term corresponding to the structural node. The factor of 4 accounts for the fact that higher oscillation modes are possible for a system of connected springs and masses, in contrast to the single element, which has one period.

For computational reasons, the reciprocal of \( m_n \) is stored in the FLAC grid. Hence \( 1/m_n \) is printed out when the command `PRINT gpm` is given.
1.4 Tutorial on the Explicit Finite Difference Method

The equations embodied in FLAC are presented earlier in this section. Here, we provide a simple working program that demonstrates, in one dimension, several characteristics of the explicit finite difference method used to solve the equations. The interested user is encouraged to modify the program and its parameters in order to gain insight into the method; the best way to understand something is to experiment with it.

We use FLAC’s embedded language, FISH, to write the demonstration program. At first sight, it may seem strange and confusing to use FLAC to simulate its own inner workings. However, there are several advantages. First, not everybody has access to a compiler, or the knowledge to use it to write a program. Second, we can use FLAC’s graphics directly to plot the results. It should be emphasized that FLAC’s normal operation is being suppressed for this demonstration (the command SET mech=off prevents FLAC from doing any of its own calculations). We write a program in the FISH language that takes over FLAC’s grid variables (xvelocity, xdisplacement, sxx) and uses them in a way that we prescribe and control in our program. First, the equations are given in their basic form, without gravity and damping, so that the resulting program models one-dimensional wave propagation.

The differential equations for a solid, one-dimensional bar of density, \( \rho \), and Young’s modulus, \( E \), are given as follows. The constitutive law is

\[
\sigma_{xx} = E \frac{\partial u_x}{\partial x} \tag{1.42}
\]

The law of motion (or equilibrium) is

\[
\rho \frac{\partial^2 u_x}{\partial t^2} = \frac{\partial \sigma_{xx}}{\partial x} \tag{1.43}
\]

We assume the bar to be unconfined laterally. The bar is discretized into, say, 50 equal finite difference zones (or elements), and numbered as illustrated in Figure 1.7.

The central finite difference equation corresponding to Eq. (1.42) for a typical zone \( i \) is given by Eq. (1.44). Here the quantities in parentheses (e.g., \( (t) \)) denote the time at which quantities are evaluated; the superscripts, \( i \), denote the zone number, not that something is raised to a power.

\[
\sigma_{xx}^i(t) = E \frac{u_{x}^{i+1}(t) - u_{x}^{i}(t)}{\Delta x} \tag{1.44}
\]
The equation of motion is similarly discretized for gridpoint $i$:

$$\frac{\rho}{\Delta t} \left\{ \ddot{u}_x^i \left( t + \frac{\Delta t}{2} \right) - \ddot{u}_x^i \left( t - \frac{\Delta t}{2} \right) \right\} = \frac{1}{\Delta x} \left\{ \sigma_{xx}^i(t) - \sigma_{xx}^{i-1}(t) \right\}$$

(1.45)

or, rearranging:

$$\ddot{u}_x^i \left( t + \frac{\Delta t}{2} \right) = \ddot{u}_x^i \left( t - \frac{\Delta t}{2} \right) + \frac{\Delta t}{\rho \Delta x} \left\{ \sigma_{xx}^i(t) - \sigma_{xx}^{i-1}(t) \right\}$$

(1.46)

Integrating again to get displacements:

$$u_x^i(t + \Delta t) = u_x^i(t) + \dot{u}_x^i \left( t + \frac{\Delta t}{2} \right) \Delta t$$

(1.47)

In the explicit method, the quantities on the right-hand sides of all difference equations are “known”; therefore, we must evaluate Eq. (1.44) for all zones before moving on to Eqs. (1.46) and (1.47), which are evaluated for all gridpoints. Conceptually, this process is equivalent to a *simultaneous* update of variables (rather than a *successive* update in some other method, in which “old” and “new” values are mixed on the right-hand sides).

Eq. (1.44) is encoded into the function `constit`:

```python
def constit
    loop i (1,nel)
        sxx(i,1) = e * (xdisp(i+1,1) - xdisp(i,1)) / dx
    end_loop
end
```

---

**Figure 1.7 Numbering scheme for elements and gridpoints in a bar**
Note that we have to use double indices to identify grid variables because FLAC’s arrays are two-dimensional; however, we just set the second index to one.

Eq. (1.46) is encoded into the function `motion`:

```python
def motion
    loop i (2, nel)
        xvel(i,1) = xvel(i,1) + (sxx(i,1) - sxx(i-1,1)) * tdx
    end_loop
end
```

Note that the last (right-hand) gridpoint is implicitly fixed, because its velocity is not changed. Eq. (1.47) translates to `dis_calc`:

```python
def dis_calc
    loop i (1, nel)
        xdisp(i,1) = xdisp(i,1) + xvel(i,1) * dt
    end_loop
end
```

Time is implied in these functions according to Eqs. (1.44), (1.46) and (1.47). Note that if the program is halted at any stage, the variables correspond to different points in time (for example, velocities are shifted by half a timestep from displacements).

The above functions are invoked sequentially in the main function `scan_all`, which is executed every time FLAC does one step:

```python
def scan_all
    while stepping
        time = time + dt
        constit
        motion
        bc
        dis_calc
    end
```

During execution of this function, time is incremented by $\Delta t$. Function `bc` supplies one-half of a cycle of an inverted cosine wave to the left-hand end of the bar; at all later times, the applied velocity is zero. The pulse is cosine-shaped in order to limit its high frequency components. When modeling wave propagation in a numerical grid, a common rule-of-thumb is that there should be at least ten elements within the shortest wavelength to be propagated.

The function `start-up` supplies initial values for all variables, and calculates $\Delta t$ based on a given fraction of critical timestep. Variables defined in `start-up` are shown below.
Table 1.2  Variables defined in start-up

<table>
<thead>
<tr>
<th>FISH name</th>
<th>Name within equations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nel</td>
<td>number of elements</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>$E$</td>
<td>Young’s modulus</td>
</tr>
<tr>
<td>ro</td>
<td>$\rho$</td>
<td>density</td>
</tr>
<tr>
<td>dx</td>
<td>$\Delta x$</td>
<td>element size</td>
</tr>
<tr>
<td>p</td>
<td>number of wavelengths per element</td>
<td></td>
</tr>
<tr>
<td>vmax</td>
<td>amplitude of velocity pulse</td>
<td></td>
</tr>
<tr>
<td>frac</td>
<td>fraction of critical timestep</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>$c$</td>
<td>wave speed</td>
</tr>
<tr>
<td>dt</td>
<td>$\Delta t$</td>
<td>timestep</td>
</tr>
<tr>
<td>twave</td>
<td>duration of input pulse</td>
<td></td>
</tr>
<tr>
<td>freq</td>
<td>frequency of input pulse</td>
<td></td>
</tr>
<tr>
<td>tdx</td>
<td>$\Delta t/(\rho \Delta x)$</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>$\omega = 2\pi f$</td>
<td>number of timesteps for 50 “seconds”</td>
</tr>
</tbody>
</table>

The complete program is stored in the file “BAR.DAT” (Example 1.2); this may be called from FLAC in the normal way.

1.4.1  Experiment 1

We initialize variables by executing start-up, then take enough timesteps to accumulate 50 time units. Histories of velocity are requested at three points along the bar, spaced at distances of 10 units. After the run is finished, the histories may be plotted by the command

`plot his 1,2,3 vs 4`
The resulting picture is reproduced as Figure 1.8. The time delay between pulses should correspond to $T = L/c$, where $L$ is the distance between history points, and $c$ is the velocity of sound in the bar ($\sqrt{E/\rho}$). In our case, there should be a time delay of 10 units between pulses.

It is instructive to rerun the simulation with different parameters. For example, the timestep may be changed (by altering $\text{frac}$), to demonstrate that the solution is almost insensitive to timestep, provided that $\text{frac}$ is less than 1. (Caution! If you set $\text{frac}$ at a value greater than 1, then be prepared to limit the simulation to only a few steps, since numerical instability will cause the magnitude of the grid variables to exceed the computer’s limits and cause FLAC to crash.)

Some other suggestions for experiments:

1. different end conditions (e.g., free; the program can be run for longer times to observe reflections);
2. nonlinear constitutive model (Caution! $\Delta t$ may need to be revised); and
3. tension cutoff, with free end to simulate tensile spalling.
1.4.2 Experiment 2

We now modify the program so that it more closely resembles the solution embodied in FLAC. We add damping and solve a static problem with body forces. With body forces (e.g., gravitational acceleration \( g_x \) in the \( x \)-direction), Eq. (1.43) becomes

\[
\rho \frac{\partial^2 \sigma_{xx}}{\partial t^2} = \frac{\partial \sigma_{xx}}{\partial x} + \rho g_x
\]  

(1.48)

If we add the extra term into Eq. (1.46) and split it up so that acceleration \( \ddot{u}_x^i \) is defined separately, then:

\[
\ddot{u}_x^i = \frac{1}{\rho \Delta x} \left\{ \sigma_{xx}^i(t) - \sigma_{xx}^{i-1}(t) \right\} + g_x
\]  

(1.49)

\[
\dot{u}_x^i(t + \frac{\Delta t}{2}) = \dot{u}_x^i(t - \frac{\Delta t}{2}) + \ddot{u}_x^i \Delta t
\]  

(1.50)

The damping in FLAC is unusual, because it is designed to vanish for steady motion (e.g., so that body forces do not retard the motion of a region that is flowing plastically with constant velocity). We provide a force that always opposes motion: its sign is always opposite to the current velocity. The magnitude of this damping force is proportional to the acceleration of a gridpoint. Hence, it will vanish for steady-flow, or equilibrium. Thus revised, Eq. (1.50) becomes

\[
\dot{u}_x^i(t + \frac{\Delta t}{2}) = \dot{u}_x^i(t - \frac{\Delta t}{2}) + \{ \ddot{u}_x^i - \alpha |\ddot{u}_x^i| \text{sgn}(\ddot{u}_x^i) \} \Delta t
\]  

(1.51)

Here, \( \alpha \) is a damping coefficient. The revised function \texttt{motion} is listed:

```python
def motion
  loop i (1, nel)
    if i = 1 then
      dxl = dx / 2.0 ; half-element for free surface
      sleft = 0.0 ; zero stress to left of surface
    else
      dxl = dx
      sleft = sxx(i-1,1)
    end_if
    accel = (sxx(i,1) - sleft) / (ro * dxl) + grav
    dxv = (accel - dfac*abs(accel)*sgn(xvel(i,1)))*dt
    xvel(i,1) = xvel(i,1) + dxv
  end_loop
end
```

FLAC Version 6.0
Note that we build the left-hand boundary conditions into the function by setting the stress to zero at this end and using half the element size. New variables are `grav`, for $g_x$, and `dfac`, for the damping factor, $\alpha$. These are defined in start-up, and other unused variables are deleted. The number of elements is reduced to 10 in order to allow fast execution. The revised FISH program is available as data file “BARG.DAT” (Example 1.3).

When FLAC is run with this data file, the plot shown in Figure 1.9 may be made, giving displacement histories at the left-hand end and the middle of the bar. The system is seen to converge to equilibrium in a time that is about twice the natural period of the bar. An elastic system is usually underdamped with this type of damping. Figure 1.10 records the final displacement profile, which shows the parabolic distribution caused by gravity loading.

**Figure 1.9** Displacement histories at two points: gravity loading
Figure 1.10  Displacement profile at the final state of equilibrium

Example 1.2  Data file “BAR.DAT”

; Wave propagation simulator in FISH
g 51 l
m e
prop d l s l b l
set mech=off
def start_up
  nel = 50
  e = 1.0
  ro = 1.0
  dx = 1.0
  p = 15.0
  vmax = 1.0
  frac = 0.2
  c = sqrt(e / ro)
  dt = frac * dx / c
  twave = p * dx / c
  freq = 1.0 / twave
  tdx = dt / (ro * dx)
  w = 2 * pi * freq
  ncyc = int(50.0 / dt)
loop i (1,nel+1) ;initialize FLAC’s grid variables
  x(i,1) = (i-1) * dx
  xdisp(i,1) = 0.0
  sxx(i,1) = 0.0
  xvel(i,1) = 0.0
end_loop

end

time = -dt / 2.0

--- main loop ... time is incremented by dt ---
def scan_all
  while_stepping
    time = time + dt
    constit
    motion
    bc
    dis_calc
  end
end

--- constitutive law: stresses are derived from strains ---
def constit
  loop i (1,nel)
    sxx(i,1) = e * (xdisp(i+1,1) - xdisp(i,1)) / dx
  end_loop
end

--- law of motion: new velocities are derived from stresses ---
def motion
  loop i (2,nel)
    xvel(i,1) = xvel(i,1) + (sxx(i,1) - sxx(i-1,1)) * tdx
  end_loop
end

--- displacements are derived from velocities ---
def dis_calc
  loop i (1,nel)
    xdisp(i,1) = xdisp(i,1) + xvel(i,1) * dt
  end_loop
end

def bc ;boundary conditions --- cosine pulse applied to left end
  if time >= twave then
    xvel(1,1) = 0.0
  else
    xvel(1,1) = vmax * 0.5 * (1.0 - cos(w * time))
  end_if
end

his xvel i=1 j=1
his xvel i=10 j=1
his xvel i=20 j=1
his time
Example 1.3  Data file “BARG.DAT”

; Test of quasi-static compaction of bar by gravity
g 51 1
m e
prop d 1 s 1 b 1
set mech=off
def start_up
   nel  = 10
   e   = 1.0
   ro  = 1.0
   dx  = 1.0
   frac = 0.5
   dfac = 0.8
   grav = 10.0
   c    = sqrt(e / ro)
   dt   = frac * dx / c
   loop i (1,nel+1) ;initialize FLAC’s grid variables
      x(i,1) = (i-1) * dx
      xdisp(i,1) = 0.0
      sxx(i,1) = 0.0
      xvel(i,1) = 0.0
   end_loop
end
;--- main loop ... time is incremented by dt ---
def scan_all
   while_stepping
      constit
      motion
      dis_calc
   end
;--- constitutive law: stresses are derived from strains ---
def constit
   loop i (1,nel)
      sxx(i,1) = e * (xdisp(i+1,1) - xdisp(i,1)) / dx
   end_loop
end
;--- law of motion: new velocities are derived from stresses
def motion
  loop i (1, nel)
    if i = 1 then
      dxl = dx / 2.0 ; half element for free surface
      sleft = 0.0 ; zero stress to left of surface
    else
      dxl = dx
      sleft = sxx(i-1,1)
    end_if
    accel = (sxx(i,1) - sleft) / (ro * dxl) + grav
    dxv = (accel - dfac * abs(accel) * sgn(xvel(i,1))) * dt
    xvel(i,1) = xvel(i,1) + dxv
  end_loop
end

--- displacements are derived from velocities ---
def dis_calc
  loop i (1, nel)
    xdisp(i,1) = xdisp(i,1) + xvel(i,1) * dt
  end_loop
end

his xdis i=1 j=1
his xdis i=6 j=1
his nstep=2
start_up
step 200
; plot his 1,2
; plot xdis line 0,0 10,0 11
ret
1.5 References


