LEAF – A NEW LAYERED ELASTIC COMPUTATIONAL PROGRAM FOR FAA PAVEMENT DESIGN AND EVALUATION PROCEDURES

By:
Gordon F. Hayhoe
FAA William J. Hughes Technical Center
Airport Technology Research and Development Branch, AAR-410
Atlantic City International Airport, NJ 08405, U.S.A.
Phone: (609) 485- 8555; FAX: (609) 485-4845
gordon.hayhoe@tc.faa.gov

PRESENTED FOR THE 2002 FEDERAL AVIATION ADMINISTRATION AIRPORT TECHNOLOGY TRANSFER CONFERENCE

05/02

ABSTRACT

A new computer program implementation of the layered elastic response equations has been written for use in Federal Aviation Administration (FAA) airport pavement design and evaluation computer programs. The program is written in Visual Basic (VB) 6.0 for Microsoft Windows 95, or higher, as a Dynamic Link Library with a defined interface. It can therefore be executed from programs written in other languages compiled for execution under Windows. Major objectives in writing the new program were to improve the efficiency of the computation of linear elastic pavement responses in the LEDFAA thickness design program and to provide a well-documented methodology and implementation suitable for further development when necessary. Wheel loads are modeled as circular loads with constant vertical pressure. Efficiency has been improved by structuring the program loops so that redundant computations are eliminated for multiple aircraft on a multiple-layered structure. This makes the computation time only very weakly dependent on the number of layers and the number of aircraft when all of the structure and aircraft information is passed to the program before execution. A code fragment is presented to illustrate the structure of the program loops. The use of Gauss-Laguerre integration, with offset of the layer origins, and part inversion in the solution of the matrix equations also improve efficiency. The development environment for LEAF was a computer program for backcalculating the layer modulus values of pavement structures represented by linear elastic layers of infinite horizontal extent. The requirements for calling the DLL from an application are illustrated by code excerpts from the backcalculation program.

INTRODUCTION

LEAF is a layered elastic analysis computer program developed for use as a component in Federal Aviation Administration (FAA) airport pavement design and analysis application computer programs. Intended originally to replace JULEA as used in LEDFAA 1.2, LEAF has the same general structure as JULEA, with Gauss-Laguerre integration, but introduces a more generalized transformation of the origin in the integral equations and organizes the inner loops for more efficient computation of pavement responses for multiple aircraft mixes.

The fundamental computational element of any layered elastic program is based on the idealized multiple-layered half-space with circular load applied at the surface, as shown in figure 1. The bottom layer in LEAF is of infinite depth and the load on the surface has uniform pressure distribution. The axis system is axisymmetric with radius *r*, depth *z*, and angle in the horizontal plane *q* defining any point in an infinite disk or an infinite half-space.

The basic set of structural responses which are calculated from the layered elastic equations are vertical stress, s_z, radial stress, s_r, tangential stress, s_t, vertical-radial shear stress, t_{zr}, vertical deflection, *w*, and radial deflection, *u*.

As illustration of the type of computation required, equation 1 shows the layered elastic equation for vertical stress in a given layer (the evaluation layer). See, for example, references 1 and 2.

$$s_z = -qa\int_0^\infty J_0(ar)J_1(aa)\begin{pmatrix}\{A(a)-C(a)(1-2u-az)\}e^{az}\\-\{B(a)+D(a)(1-2u+az)\}e^{-az}\end{pmatrix}da \qquad (1)$$

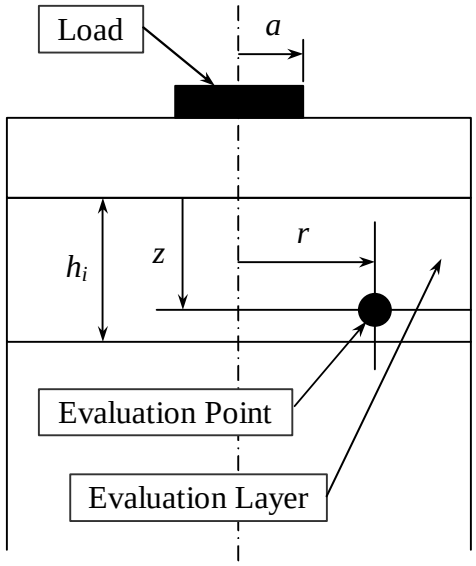

Figure 1. Nomenclature and coordinates.

a = tire contact radius
r = evaluation point radius
z = depth from the top of the evaluation layer
q = tire contact pressure
α = integration variable (Hankel domain variable)
J_0, J_1 = Bessel functions

The coefficients $A(\alpha)$, $B(\alpha)$, $C(\alpha)$, and $D(\alpha)$ are determined according to boundary conditions at the top and bottom surfaces of each of the layers. They are functions of α. In the following, the α and the parentheses are dropped, but it should be remembered that the boundary condition coefficients are functions of α and must be recomputed whenever α changes. Values of the boundary condition coefficients A, B, C, and D must be computed for each of the layers, but the complete integral equation only needs to be solved for the evaluation point in the evaluation layer. Also, because the coefficients are functions of α, the terms in braces in equation 1 can be multiplied by any function of α (taking a common factor in α outside the braces), provided only that the multiplier is consistently applied to all integrands used to set up the boundary conditions.

SOLUTION OF THE LAYERED ELASTIC EQUATIONS

The basic procedure used to compute the structure responses is as follows:

1. Set up two equilibrium and two compatibility equations for each interface between layers. These are the boundary condition equations.
2. Assemble a matrix equation containing the boundary condition equations for all of the interfaces (including the surface and the interface at infinity).
3. Solve the matrix equation for the boundary condition coefficients.
4. Integrate the response equations (represented here by equation 1).

To illustrate the general structure of the boundary condition matrix equation, the assembled equation for a three-layer system is as follows.

$$\begin{pmatrix}
1 & -1 & -(1-2u_1) & -(1-2u_1) & & & & & & & \\
1 & 1 & 2u_1 & -2u_1 & & & & & & & \\
e^{ah_1} & -e^{-ah_1} & a_{3,3}e^{ah_1} & a_{3,4}e^{-ah_1} & -1 & 1 & (1-2u_2) & (1-2u_2) & & & \\
e^{ah_1} & e^{-ah_1} & a_{4,3}e^{ah_1} & a_{4,4}e^{-ah_1} & -1 & -1 & -2u_2 & 2u_2 & & & \\
e^{ah_1} & e^{-ah_1} & a_{5,3}e^{ah_1} & a_{5,4}e^{-ah_1} & -R_1 & -R_1 & (2-4u_2)R_1 & -(2-4u_2)R_1 & & & \\
e^{ah_1} & -e^{-ah_1} & a_{6,3}e^{ah_1} & a_{6,4}e^{-ah_1} & -R_1 & R_1 & -R_1 & -R_1 & & & \\
 & & & & e^{ah_2} & -e^{-ah_2} & a_{7,7}e^{ah_2} & a_{7,8}e^{-ah_2} & 1 & (1-2u_3) & \\
 & & & & e^{ah_2} & e^{-ah_2} & a_{8,7}e^{ah_2} & a_{8,8}e^{-ah_2} & -1 & 2u_3 & \\
 & & & & e^{ah_2} & e^{-ah_2} & a_{9,7}e^{ah_2} & a_{9,8}e^{-ah_2} & -R_2 & (2-4u_2)R_2 & \\
 & & & & e^{ah_2} & -e^{-ah_2} & a_{10,7}e^{ah_2} & a_{10,8}e^{-ah_2} & -R_2 & R_2 &
\end{pmatrix}
\begin{pmatrix} A_1 \\ B_1 \\ C_1 \\ D_1 \\ A_2 \\ B_2 \\ C_2 \\ D_2 \\ B_3 \\ D_3 \end{pmatrix}
=
\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The first two rows are the boundary condition equations for vertical and shear stress at the surface. The two other boundary condition equations at the surface, those for vertical and radial deflection, are eliminated because there is no deflection constraint applied in this application. The next four rows are the boundary condition equations for the first interface below the surface (equating vertical and shear stresses and vertical and radial deflections across the interface). The last four rows are the equations for the interface at the top of the subgrade (coefficients A_3 and C_3 are set to zero to satisfy the conditions for the infinite depth of the subgrade).

The presence of the exponentials in the matrix equation can lead to underflow and overflow during solution of the equations. Making a change in the origin of the z coordinate in each layer changes the values of the exponents and allows control over the values of the exponentials. The procedure used in LEAF is to multiply the first and third columns of each equilibrium equation by $e^{-ah_{Oi,1}}$ and multiply the second and fourth columns of each equation by $e^{-ah_{Oi,2}}$. The fifth and seventh and the sixth and eighth columns are multiplied by $e^{-ah_{Oi+1,1}}$ and $e^{-ah_{Oi+1,2}}$, respectively.

The change in variable must also be duplicated in the integrals as shown in equation 2. The primes in equation 2 indicate that the numerical values of the coefficients are changed after application of the change of variable.

$$\begin{aligned} s_z = &-qa\int_0^\infty J_0(ar)J_1(aa)\{A_i' - C_i'(1-2u_i - az)\}e^{-a(h_{Oi,1}-z)}da \\ &+ qa\int_0^\infty J_0(ar)J_1(aa)\{B_i' + D_i'(1-2u_i + az)\}e^{-a(h_{Oi,2}+z)}da \end{aligned} \tag{2}$$

Numerical integration by the Gauss-Laguerre method requires a further change of variable:

$$a_1 = a(h_{Oi,1} - z) = az_1 \qquad a_2 = a(h_{Oi,2} + z) = az_2$$
$$da = \frac{da_1}{z_1} \quad \text{, and} \quad da = \frac{da_2}{z_2} \tag{3}$$

giving

$$s_z = -qa\int_0^\infty J_0\left(\frac{ra_1}{z_1}\right)J_1\left(\frac{aa_1}{z_1}\right)\left\{A_i' - C_i'\left(1-2u_i - \frac{za_1}{z_1}\right)\right\}e^{-a_1}\frac{da_1}{z_1} + qa\int_0^\infty J_0\left(\frac{ra_2}{z_2}\right)J_1\left(\frac{aa_2}{z_2}\right)\left\{B_i' + D_i'\left(1-2u_i + \frac{za_2}{z_2}\right)\right\}e^{-a_2}\frac{da_2}{z_2} \quad (4)$$

Finally, the infinite continuous integral is transformed to a finite sum (see, for example, reference 3).

$$s_z = -qa\sum_{j=1}^{N} J_0\left(\frac{ra_{1j}}{z_1}\right)J_1\left(\frac{aa_{1j}}{z_1}\right)\left\{A_i' - C_i'\left(1-2u_i - \frac{za_{1j}}{z_1}\right)\right\}\frac{w_j}{z_1} + qa\sum_{j=1}^{N} J_0\left(\frac{ra_{2j}}{z_2}\right)J_1\left(\frac{aa_{2j}}{z_2}\right)\left\{B_i' + D_i'\left(1-2u_i + \frac{za_{2j}}{z_2}\right)\right\}\frac{w_j}{z_2} \quad (5)$$

Where a_{1j} and a_{2j} are the abscissa values for the summation of N points and w_j are weights corresponding to the a_{1j} and a_{2j} values.

Computing the sum requires that the Bessel functions $J_0\left(\frac{r}{z}a\right)$ and $J_1\left(\frac{a}{z}a\right)$ be computed at each step. The independent variable is a, and increasing r or a increases the frequency of the corresponding function. The number of integration points required to maintain accuracy (and avoid aliasing) therefore increases as the radius to the evaluation point increases or the radius of the load increases. But with the change of variable introduced to enable Gauss-Laguerre integration, increasing the value of z by an additional shift in the origin of the z coordinate decreases the frequency of the oscillation. This avoids, within reason, changing the number of integration points and recomputing the values of the abscissa and weight at each summation point.

The procedure for selection of the $h_{Oi,1}$ and $h_{Oi,2}$ values in the origin shift for each layer is as follows.

1. Set the offsets for the first layer to twice the thickness of the first layer, $h_{Oi,1} = h_{Oi,2} = 2h_1$.
2. Set the offsets for all of the other layers according to the relationship $h_{Oi,1} = h_{Oi,2} = (h_{Oi-1,2} + h_{i-1})$, for i from 2 to the number of layers.
3. Compute new offsets for the evaluation layer as (R_{max} / 4 + z), where R_{max} is the maximum of the load radius and the radius to the evaluation point, and z is the distance to the evaluation point from the top of the evaluation layer.
4. If the new values for the evaluation layer are greater than the first values, then reset the offset values of all of the layers according to the relationship in step 2 but increased to match the values computed in step 3. Otherwise, leave the offsets unchanged.

The effects of this procedure are to minimize the possibility of overflow or underflow in the solution of the boundary condition equations (step 2) and to eliminate the effects of aliasing in the solution of the response integrals (step 3). It was originally thought that best convergence would probably be achieved by assigning different values to the two offsets, but further work indicated that keeping both at the same value minimizes the tendency to overflow and underflow. The conditions for convergence are, however, not very well understood and separate offset variables were retained in the derivation and in the computer program in case better conditions are found in the future. Further details on the derivation of the equations and the numerical methods used in their solution are given in reference 4. The offset (origin shift) for positive exponents is illustrated in figure 2.

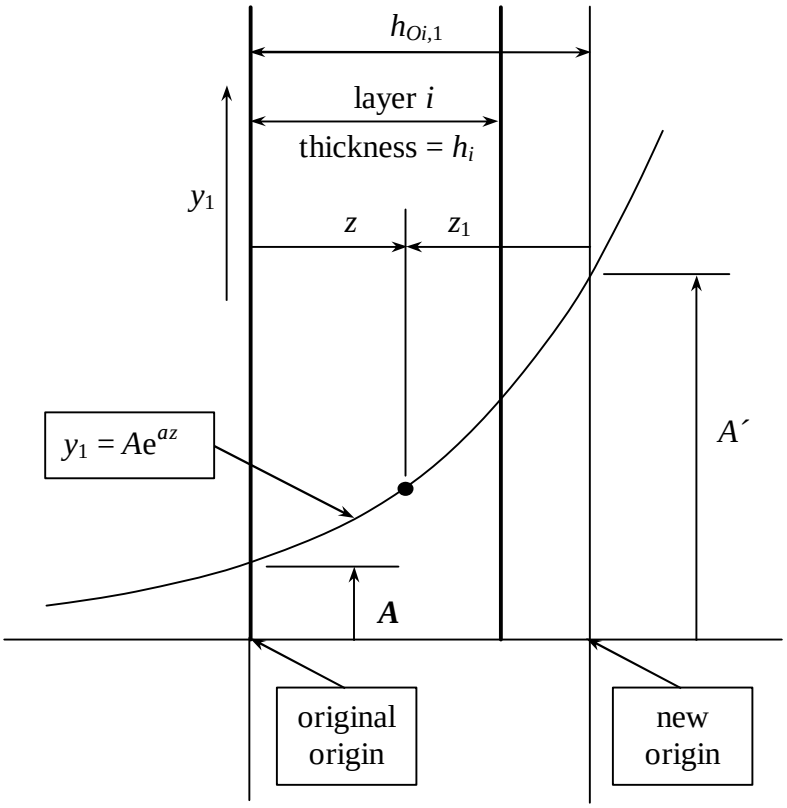

Figure 2. Origin shift for positive exponents, $y_1 = Ae^{az} = A'e^{-az_1}$.

VARIABLE INTERFACE BONDING

In order to allow relative horizontal movement between two layers at an interface, a uniformly distributed shear spring is assumed to connect the layers. This is the same model as used in other layered elastic computer programs having the same feature, including JULEA. The spring acts in the radial direction and has the following law:

$$\begin{aligned} t_i &= k_i(u_i - u_{i+1}), \text{ or} \\ t_{i+1} &= k_i(u_i - u_{i+1}) \text{ since } t_i = t_{i+1} \end{aligned} \tag{6}$$

where

t_i = radial shear stress at the interface between layers i and $i+1$.

$(u_i - u_{i+1})$ = relative radial displacement across the interface.

k_i = interface spring stiffness, with units of lb/inch relative radial displacement/inch along the radius/inch along the circumference at radius r (lb/in^3). That is, a radial spring connects elemental areas either side of the interface. The spring resists relative radial displacement across the interface.

To reduce numerical complications, the computer implementation can use the relationship

$$k_i = \frac{l_i}{1-l_i}, \text{ giving}$$
$$(1-l_i)t_i = l_i(u_i - u_{i+1}) \tag{7}$$

This change of variable is used in JULEA, with a further logarithmic transformation in the input data parameter. LEAF uses the same variable, l, both within the computational routines and as the input data parameter.

For fully bonded layers, $k_i = \infty,\ l_i = 1, \text{and}\ u_i = u_{i+1}$.

For fully unbonded layers, $k_i = 0,\ l_i = 0, \text{and}\ t_i = 0$.

For reference, the transformation in JULEA is as follows:

m_i = input parameter for interface i.

E_2 = modulus of layer 2 (directly below the surface layer).

$$\textbf{If} \quad m_i \geq 100{,}000 \quad \textbf{Then} \quad l_i = 0 \quad \textbf{Else} \quad l_i = 10^{-\frac{m_i}{E_2}}$$

Note that l is a parameter. It does not have units and it has no physical significance other than the manner in which it varies the spring stiffness. It has a value from 0 to 1 (fully unbonded to fully bonded).

COMPUTER PROGRAM IMPLEMENTATION OF THE INTEGRATIONS

Computer program implementation of Gauss-Laguerre integration is basically the same for stresses, strains, and displacements. The only difference is that the equations evaluated in the program loops are different. A primary objective was to reduce the run-time for the pavement thickness design procedures implemented in the computer program LEDFAA. These implementations require the computation of one, or at most, two responses for a single pavement structure of rigid or flexible type with or without an overlay and with many landing gear configurations in the design aircraft mix. The strategy adopted was based on the following.

1. Allow for independent computation of those responses required for a particular type of pavement design. For example, new flexible design requires computation of only vertical strain at the top of the subgrade in order to iterate to the design thickness.
2. Arrange the program loops so that individual elements of the integrand are computed the minimum number of times.
3. Assume that wheel loads and tire pressures are the same for each wheel in an individual landing gear wheel group.

Subroutines are provided for independently computing vertical deflection, horizontal deflection, vertical strain, and horizontal stress. Visual Basic program code from the subroutine for computing vertical strain is given below to illustrate the implementation of the integration subroutines. The integral equation for vertical strain is derived from the basic integral equations for vertical, radial, and transverse stress through the relation $e_z = \frac{1}{E}(s_z - u(s_r + s_t))$. The Gauss-Laguerre summation of the resulting layered elastic integral is:

$$e_z = \frac{qa(1+u_i)}{E_i}\left[\begin{array}{l} -\sum_{j=1}^{N} J_0\left(\frac{ra_{1j}}{z_1}\right) J_1\left(\frac{aa_{1j}}{z_1}\right)\left\{A_i' - C_i'\left(1-4u_i - \frac{za_{1j}}{z_1}\right)\right\}\frac{w_j}{z_1} \\ +\sum_{j=1}^{N} J_0\left(\frac{ra_{2j}}{z_2}\right) J_1\left(\frac{aa_{2j}}{z_2}\right)\left\{B_i' + D_i'\left(1-4u_i + \frac{za_{2j}}{z_2}\right)\right\}\frac{w_j}{z_2} \end{array}\right] \quad (8)$$

Data is passed to the subroutine for a single structure and a complete traffic mix. The traffic mix has `NAC` number of aircraft. An "aircraft" is a group of `NTires` wheels, usually representing a landing gear, but not necessarily. For example, all 16 main landing gear wheels on a B-747 could be defined as a wheel group. Each wheel in an aircraft wheel group has the same wheel load and tire pressure. Each aircraft wheel group has `NevalPoints` evaluation points at a single depth measured from the top of the pavement structure. Vertical strain is to be calculated at each evaluation point for each aircraft.

The expression to be summed can be split into four levels. The levels are, in decreasing order in the hierarchy, as follows.

1. Under the assumption that all wheels on a landing gear have the same load and the same tire pressure, the term $\frac{qa(1+u_i)}{E_i}$ only needs to be computed once for each aircraft.
2. Terms like $\left\{A_i - C_i\left(1-4u_i - \frac{za_{1j}}{z_1}\right)\right\}\frac{w_j}{z_1}$ only need to be computed once for each integration point. That is, they are common to all evaluation points on all wheels on all aircraft. These terms are also the most expensive to compute because the coefficients *A*, *B*, *C*, and *D*, must be computed from the boundary condition equations each time. Special cases can also apply to these terms. For example, in the computation of vertical strain at the top of the subgrade, only the term containing *B* and *D* needs to be computed because *A* and *C* are zero in the subgrade layer.

3. Terms like $J_1\left(\frac{aa_{1j}}{z_1}\right)$, containing aa, where a is the tire radius, only need to be computed once for each wheel at each integration point. That is, they are common to all evaluation points relative to each wheel taken separately. However, if both load and pressure are the same for all of the wheels on an aircraft, then these terms only need to be computed once for each aircraft at each integration point.
4. Terms like $J_0\left(\frac{ra_{1j}}{z_1}\right)$ and $J_1\left(\frac{ra_{1j}}{z_1}\right)$must be calculated for all evaluation points on all wheels on every aircraft at each integration point.

Computation loops in the program are therefore ordered as shown in the Visual Basic code fragment below.

```
‘ Depth from top of evaluation layer to evaluation point.
  ZLayer = ZEval - ZInterface(I - 1)
‘ RMax = maximum of evaluation point radius or load radius.
  Call SetOShifts(OShift(), I, ZLayer, RMax)

  Z1 = OShift(I, 1) - ZLayer
  Z2 = OShift(I, 2) + ZLayer

  ZLayer1 = ZLayer / Z1
  ZLayer2 = ZLayer / Z2
  Poisx2 = Poissons(EvalLayer) * 2

  IConstants = (EvalLayer - 1) * 4

‘ GLAlpha()  = Gauss-Laguerre abscissa values.
‘ GLWeight() = Gauss-Laguerre weight values.
  For IG = 1 To GLNGauss  ‘ Gauss-Laguerre integration points.
    StrainWIGforConverge = 0  ‘ Initialize convergence criterion.
    If EvalLayer < NLayers Then   ‘ A and C are zero for subgrade.
‘     Put constants for all layers in B().
      Call FindConstants(GLAlpha(IG) / Z1, B(), OShift())
      AK = B(IConstants + 1)      ‘ A for evaluation layer.
      AlphaZ = GLAlpha(IG) * ZLayer1
      CK = B(IConstants + 3) * (1 - Poisx2 * 2 - AlphaZ)
      StrainWIG1 = -(AK - CK) * GLWeight(IG) / Z1
```

$$= -\left\{A_i - C_i\left(1 - 4u_i - \frac{za_{1j}}{z_1}\right)\right\}\frac{w_j}{z_1}$$

```
      StrainWIGforConverge = (Abs(AK) + Abs(CK)) * GLWeight(IG) / Z1
    End If
‘   Do the same for constants B and D.
    Call FindConstants(GLAlpha(IG) / Z2, B(), OShift())
    BK = B(IConstants + 2)
    AlphaZ = GLAlpha(IG) * ZLayer2
    DK = B(IConstants + 4) * (1 - Poisx2 * 2 + AlphaZ)
    StrainWIG2 = (BK + DK) * GLWeight(IG) / Z2
```

$$= \left\{ B_i + D_i \left(1 - 4u_i + \frac{za_{2j}}{z_2} \right) \right\} \frac{w_j}{z_2}$$

```
    StrainWIGforConverge = StrainWIGforConverge _
                         + (Abs(BK) + Abs(DK)) * GLWeight(IG) / Z2
'   Loop over all aircraft.
    For IAC = 1 To NAC
      If Not ACConverged(IAC) Then
'       Skip if all evaluation points for aircraft IAC have
'       converged.
        A2 = TireRadius(IAC, 1) ' Move down 1 loop for varying
'                                  pressures.
        A1 = A2 / Z1
        A2 = A2 / Z2
        If EvalLayer < NLayers Then
'         The first line below is the load function for uniform
'         pressure. The second line is for selectable load
'         functions. Parabolic is currently available as an
'         alternative to uniform pressure.
'          J1AlphaA1 = bessj1(GLAlpha(IG) * A1) * StrainWIG1
          J1AlphaA1 = LoadFunction(GLAlpha(IG) * A1) * StrainWIG1
```

$$= -J_1 \left(\frac{aa_{1j}}{z_1} \right) \left\{ A_i - C_i \left(1 - 4u_i - \frac{za_{1j}}{z_1} \right) \right\} \frac{w_j}{z_1}$$

```
        End If
        J1AlphaA2 = LoadFunction(GLAlpha(IG) * A2) * StrainWIG2
```

$$= J_1 \left(\frac{aa_{2j}}{z_2} \right) \left\{ B_i + D_i \left(1 - 4u_i + \frac{za_{2j}}{z_2} \right) \right\} \frac{w_j}{z_2}$$

```
'       Loop over all tires on each gear (aircraft).
        For ITire = 1 To NTires(IAC)
'         Loop over all evaluation points on each gear (aircraft).
          For IEval = 1 To NEvalPoints(IAC)
'           R2 is the horizontal distance between the evaluation
'           point and center of the tire. Previously computed.
            R2 = Radius(IAC, ITire, IEval)
            R1 = R2 / Z1
            R2 = R2 / Z2
            If EvalLayer < NLayers Then
              J0AlphaR1 = bessj0(GLAlpha(IG) * R1)
```

$$= J_0 \left(\frac{ra_{1j}}{z_1} \right)$$

```
            End If
            J0AlphaR2 = bessj0(GLAlpha(IG) * R2)
```

$$= J_0 \left(\frac{ra_{2j}}{z_2} \right)$$

```
'           StrainWI = elemental strain for the current integration
'                      point.
            StrainWI = J0AlphaR1 * J1AlphaA1 + J0AlphaR2 * J1AlphaA2
'           Accumulate sum at each evaluation point for each gear.
            StrainW(IAC, IEval) = StrainW(IAC, IEval) + StrainWI
'           Convergence criterion always positive and conservative.
```

```
            If Abs(StrainWIGforConverge / StrainW(IAC, IEval)) _
                 < ConvergenceLimit Then  ' Default = 0.00001.
              If Not Converged(IAC, ITire, IEval) Then
'               Keep track of the number of evaluation points
'               which have converged.
                NEvalsConverged(IAC, ITire) _
                = NEvalsConverged(IAC, ITire) + 1
                If NEvalsConverged(IAC, ITire) = NEvalPoints(IAC) _
                  Then
'                 All evaluation points for tire Itire on aircraft
'                 IAC have converged.
                  NtiresConverged(IAC) = NtiresConverged(IAC) + 1
                  IterationstoConverge = IG
                End If
                Converged(IAC, Itire, IEval) = True
                If NtiresConverged(IAC) = Ntires(IAC) Then
                  ACConverged(IAC) = True
                  NACConverged = NACConverged + 1
                End If
              End If
            End If
          Next IEval
        Next ITire
      End If
    Next IAC
    If NACConverged = NAC Then
'     All evaluation points on all aircraft have converged.
      NConverge = IG
      Exit For  ' Quit integration.
    End If
  Next IG

  For IAC = 1 To NAC  ' Apply the common load/materials factor.
    Factor = TirePress(IAC, 1) * TireRadius(IAC, 1)
    Factor = Factor * (1 + Poissons(EvalLayer)) / Youngs(EvalLayer)
           = qa(1+u_i) / E_i
    For IEval = 1 To NEvalPoints(IAC)
'     Apply final factor to compute strains at all eval. points.
      StrainW(IAC, IEval) = StrainW(IAC, IEval) * Factor
      If StrainW(IAC, IEval) > StrainWmax(IAC) Then
'       Maximum strain for each gear (aircraft).
        StrainWmax(IAC) = StrainW(IAC, IEval)
      End If
    Next Ieval
  Next IAC
```

The second IAC loop is more complicated when computing horizontal responses because the responses at a single evaluation point from each tire must be transformed into rectangular coordinates before they are summed. Vertical responses can be summed directly in the first loop as shown above.

IMPLEMENTATION AS A DYNAMIC LINK LIBRARY

LEAF is written entirely in Visual Basic 6.0 and is compiled as an ActiveX dynamic link library (DLL). The DLL exports LEAF as `LEAFD.clsLEAF`. The interface consists of one callable subroutine, five data structures (one array, three user-defined types (UDT), and one enumeration type), three properties, and one event. The data structures are defined in LEAF. Two of the UDTs must be dimensioned in the calling program.

The subroutine declaration statement is:

```
Public Sub ComputeResponse(ResponseType As LEAFoptions, NACarg As Long,
LEAAircraft() As LEAFACParms, LEAStructure As LEAFStrParms, Response() As
Double, AllResps() As LEAFAllResponses)
```

`ResponseType` is an enumeration type which defines the response variable for which values will be computed (vertical strain, horizontal stress, etc.). The value of `ResponseType` is passed as one of the constant names defined by `LEAFoptions` (see below).

`NACarg` is a variable defining the number of aircraft for which data is being passed to the subroutine.

`LEAAircraft()` is a UDT defining the parameters for each aircraft. A new instance of the UDT is declared and dimensioned in the calling program. All variable values are set in the calling program.

`LEAStructure` is a UDT defining the parameters for the pavement structure, including the evaluation depth and the index for the evaluation layer. A new instance of the UDT is declared and dimensioned in the calling program. All variable values are set in the calling program.

`Response()` is a two-dimensional double-precision array returning the values of the computed responses for all evaluation points for all aircraft. It is declared and dimensioned in LEAF. The dimension indexes are (aircraft number, evaluation point number) as passed in `LEAAircraft()`.

`ALLResps()` is a UDT returning all responses when `ResponseType` has the value `AllReponses` (see the definition of `LEAFOptions` below). `AllResps()` has a lengthy definition not given here (see reference 4). The responses returned are all in rectangular coordinates and include deflections, strains and stresses, principal strains and stresses, and maximum and octahedral shear stresses.

The event is `LEAFStopped()`. It is defined and raised in LEAF.

The definitions of the UDTs for `LEAAircraft` and `LEAStructure` are, respectively:

```
Public Type LEAFACParms  ' 1 To NAircraft
' Type for passing aircraft data to LEA routine.
' Dimensioned and set in the client program.
' Must be dimensioned as an array for compatibility
' with Sub ComputeResponse calling list.
  ACname As String
  GearLoad As Double
  NTires As Long
  TirePress() As Double  ' 1 To NTires
  TireX() As Double      ' 1 To NTires
  TireY() As Double      ' 1 To NTires
  NEvalPoints As Long
  EvalX() As Double      ' 1 To NEvalPoints
  EvalY() As Double      ' 1 To NEvalPoints
```

```
End Type

Public Type LEAFStrParms
' Type for passing pavement structure data to LEAF.
' Dimensioned and set in the client program.
' Only one structure is allowed so the type is not an array.
  NLayers As Long
  Thick() As Double           ' 1 To NLayers
  Modulus() As Double         ' 1 To NLayers
  Poisson() As Double         ' 1 To NLayers
  InterfaceParm() As Double   ' 1 To NLayers
  EvalDepth As Double
  EvalLayer As Double
End Type

The definition of the enumeration type is:

Public Enum LEAFoptions
  LEAFVerticalStrain = 1    ' Select the response to be computed and
  LEAFVerticalDeflection = 2 ' returned. Selecting a single response
  LEAFHorizontalStress = 3  ' reduces run time considerably compared
  LEAFAllResponses = 4      ' with requesting all responses.
End Enum
```

`ConvergenceLimit` is a property which returns or sets the limit for convergence of the integrals to a desired relative accuracy. The default value is 0.00001.

`GLNumberofPoints` is a property which returns or sets the number of points used in the Gauss-Laguerre numerical integration. The default value is 500. Decreasing the value reduces the run time and reduces the accuracy. Increasing the value has the opposite effect. The relationship between `GLNumberofPoints` and run time or accuracy is not linear.

`LEAFError` is a read only property which returns an error message if LEAF detects an input or computational error. An empty string denotes that no errors have been detected by LEAF. At present, very little input parameter checking is done and the user is expected to ensure that the input parameters are within reasonable bounds and that there is correct dimensioning of the data types. Additional error checking will be added as time permits.

DEVELOPMENT APPLICATION

The development environment for LEAF consists of a Visual Basic application which runs a routine for backcalculating layer modulus values from falling weight deflectometer (FWD) test data and a routine for computing pavement responses for a user-defined landing gear wheel group. The backcalculation routine demonstrates calling LEAF for computation of a single pavement response (vertical deflection). The response routine demonstrates the computation and printout of all pavement responses for multiple aircraft. Declaring an instance of `clsLEAF`, setting paramenter values, and calling `ComputeResponse` is straightforward if the event `LEAFStopped` is not required. However, if `LEAFStopped` is required then an indirect reference has to be made to an instance of `clsLEAF` created from a declaration made “with events” in the declarations section of a class module. For the LEAF development application, the declaration is made in the main form:

```
Dim WithEvents RunLEAFEvent As clsLEAF.
```

Another declaration is made in the LEAF setup module:

```
Public RunLEAF As clsLEAF.
```

The instance and the indirect reference are set in the form load routine:

```
Set RunLEAFEvent = New clsLEAF   ' A new instance.
Set RunLEAF = RunLEAFEvent   ' A Public reference to RunLEAFEvent.
```

All of the variables, properties, and methods are then available. For example, the response subroutine is run with:

```
Call RunLEAF.ComputeResponse(LEAFAllResponses, NAC, CallAC(), _
                             LEAStructure, Response(), AllResp()),
```

where the argument names are as used in the application. A subroutine structure called `RunLEAFEvent_LEAFStopped()` is also automatically created in the main form. Code responding to the `LEAFStopped` event being raised by LEAF is placed in this subroutine by the user.

Using the development application, LEAF has been tested against Boussinesq solutions for a half-space and against other layered elastic computer programs for multiple-layered structures. Accuracy is good compared to the other solution methods. Details are given in reference 4.

The application is available on the FAA Airport Technology R&D Branch web site www.airporttech.tc.faa.gov as source code (including LEAF) and as an installable Windows application.

SUMMARY

A new computer program has been written for solving the layered elastic equations representing simplified pavement structures. The program implementation is arranged for efficient solution of the design procedures in the LEDFAA computer program for airport pavement thickness design. The program is written in Visual Basic 6.0 and is compiled as a dynamic link library. A sample application incorporating the dynamic link library is available as source code and as an installable executable on the FAA Airport Technology web site www.airporttech.tc.faa.gov. Documentation is also available on the web site.

ACKNOWLEDGEMENTS/DISCLAIMER

The work described in this paper was supported by the FAA Airport Technology Research and Development Branch, Dr. Satish K. Agrawal, Manager. The contents of the paper reflect the views of the author, who is responsible for the facts and accuracy of the data presented within. The contents do not necessarily reflect the official views and policies of the FAA. The paper does not constitute a standard, specification, or regulation.

REFERENCES

1. Burmister, D.M., “The Theory of Stresses and Displacements in Layered Systems and Application to the Design of Airport Runways,” Proceedings, Highway Research Board, Vol. 23, 1943.
2. Huang, Yang H., “Pavement Analysis and Design,” Prentice-Hall, New Jersey, 1993.
3. Press, William H., Teukolsky, Saul A., Vetterling, William T., and Flannery, Brian P., “Numerical Recipes in FORTRAN,” 2nd ed., Cambridge University Press, 1992.

4. Hayhoe, Gordon F., McQueen, Roy D., Guo, Edward H., and Brill, David R., “LEDFAA, The FAA’s New Software for Airport Pavement Design,” U.S. Department of Transportation Report, Office of Aviation Research, Washington, D.C., in preparatiuon.