Appendix A

Grid independence and time step convergence:

Grid independence

The heat flux signal for a spatial step size of 0.25 mm can be seen below. See *figure A.1*. The flux does not change noticeably much with a change in the number of spatial steps.

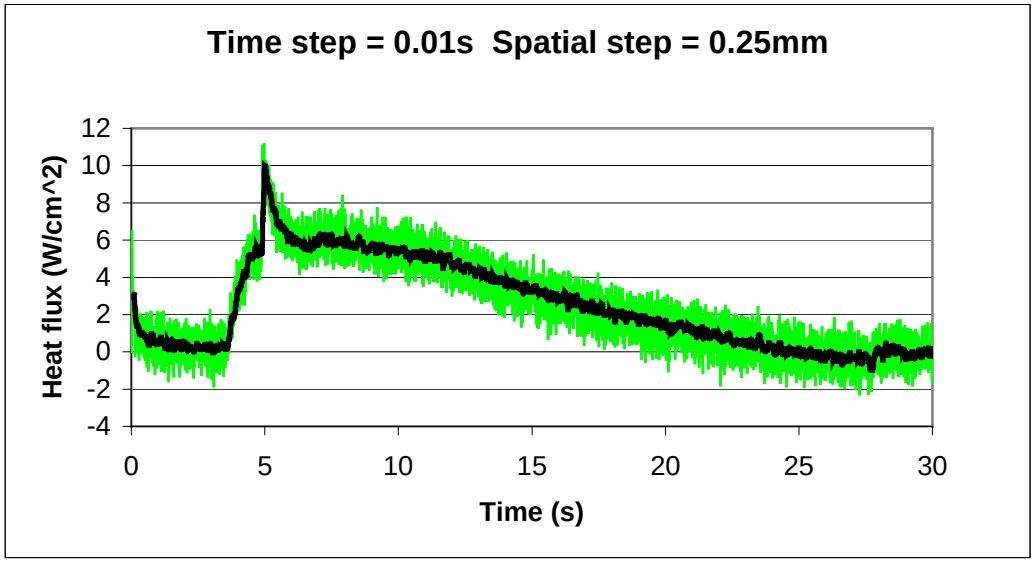

Figure A.1.

If the actual maximum of the flux is retrieved from the data and plotted against the number of spatial steps, the effect is obvious. See *figure A.2* for more detail.

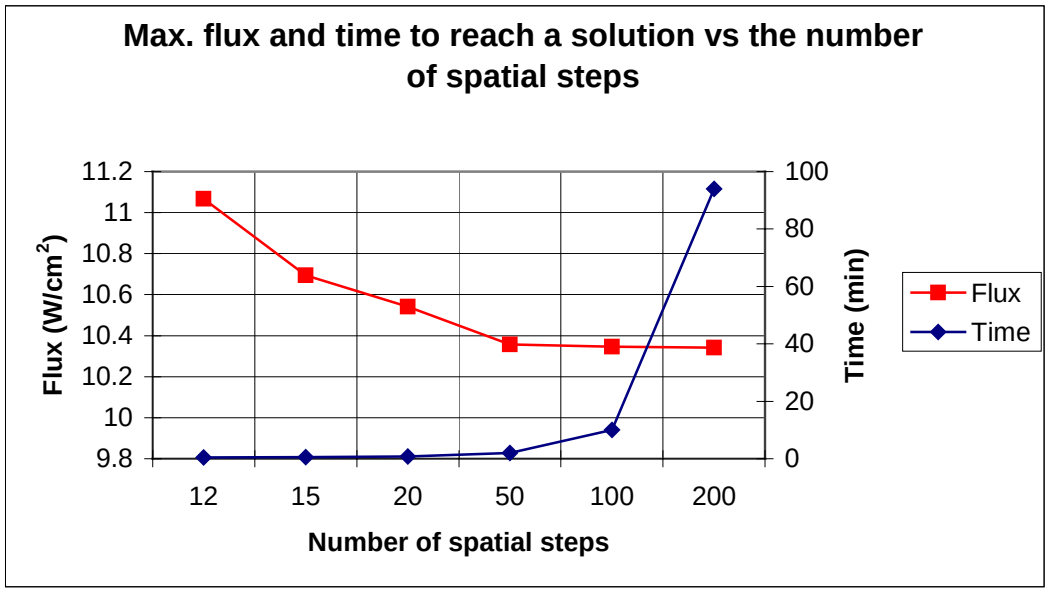

Figure A.2

In *figure A.2* the effect of the number of spatial steps on the computer time can also be seen. The time raise steeply to 94 min for this particular case when the number of spatial steps is increased from 100 to 200. Looking at the flux on the red line, it seems that the flux reached grid independence at about 50 spatial steps. That means 0.5 mm per step. To be a little more conservative a value of 100 steps were used. The value of 100 seemed realistic in the sense that convergence has been reached and the time for solving has not yet increased dramatically.

Time step convergence

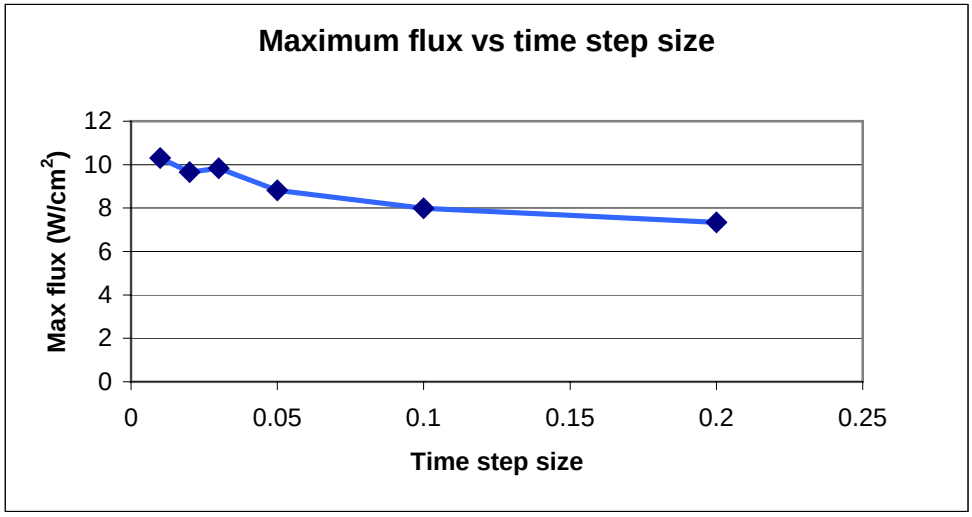

Figure A.3

Looking at the variance of the maximum flux with the change in the time step size the graph shown in *figure A.3* indicates that the maximum flux levels out in the region of 0.03s and lower. A realistic and conservative value of 0.01s was used in the test cases throughout the text. The flux will thus stay the same for smaller time steps and will not increase further as is the case when one moves from a time step size of 0.2s to 0.05s. The conclusion on this simple test is that a time step convergence was achieved with a step size of 0.01s.

Appendix B

Thermocouples:

Thomas Seebeck discovered in 1821 that if two wires composed of dissimilar metals are joined at both ends and one of the ends is heated, there is a continuous current that flows in the thermoelectric circuit.

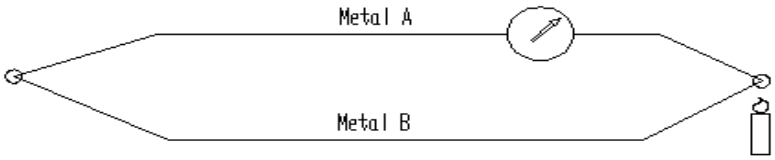

Figure B.1 Thermoelectric circuit

If the circuit is broken in the middle, then the open circuit voltage will be a function of the temperature at the junction and the properties of the two metals.

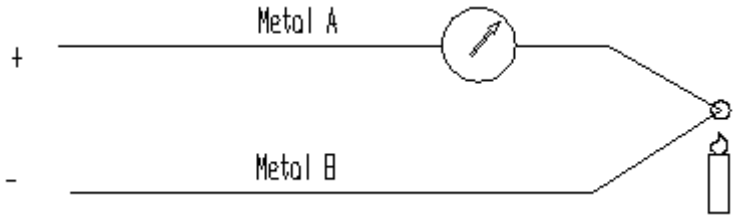

Figure B.2 Open ended thermoelectric circuit

If the voltage is to be measured, care should be taken. Connecting the voltmeter across the ends of the circuit creates two additional junctions which in turn are also thermocouples giving a current proportional to the temperature and the composition of the junctions.

If the two metals are Aluminum and Constantan (as is the case in this study) and the voltage read is done with a voltmeter with copper internal wires, the circuit will look like this:

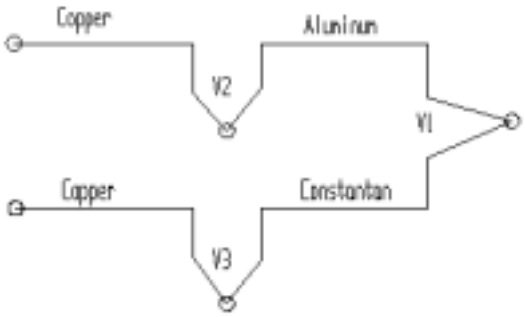

Figure B.3 Circuit with three different metals

There will thus be a voltage V1, V2 and V3. The voltage read by the voltmeter will be a combination of the three and not just that of the junction at V1. To overcome this problem a

reference point is used. In this case it is a 0^0C ice bath– a mixture of ice and water. See the figure.

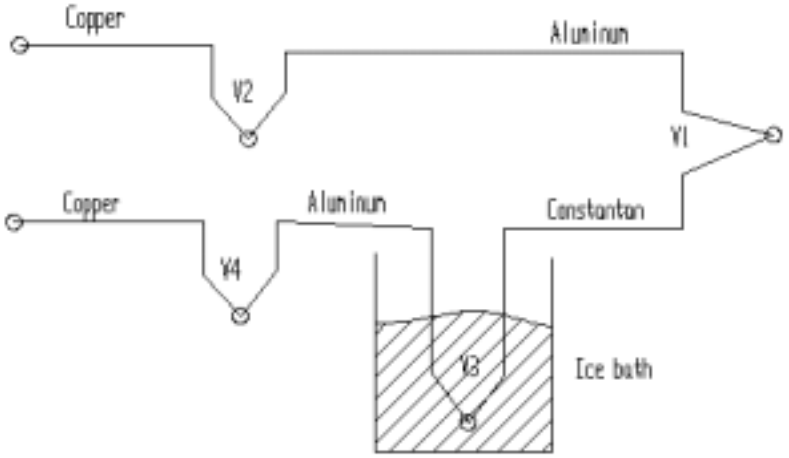

Figure B.4 Circuit with an ice-bath

If the temperatures at V2 and V4 are kept the same, the voltage created by these two junctions will cancel each other. The measured voltage by the voltmeter will be a function of the difference in the temperature between the junctions at V1 and V3. The fact that the reference temperature is 0^0C makes it simpler in the sense that the voltage at V1 (in degrees Celsius) is now proportional to the temperature at junction V1.

Appendix C

Resistor-capacitor low-pass filters:

A combination of a resistor-capacitor in series can be connected as a low-pass filter. This will allow only low frequencies to pass the filter, while high frequency noise is bypassed to ground. See the figure for a schematic layout of a resistor-capacitor low-pass filter.

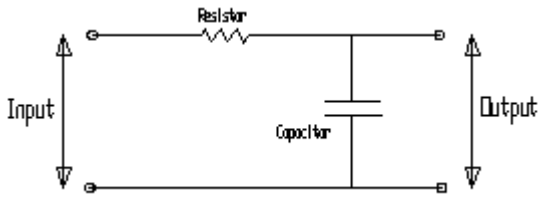

Figure C.1 A simple filter circuit

If the cutoff frequency is to be 50 Hz, which is a good choice, because most electrical noise is at 60 Hz, one needs to establish the sizes of the resistor and the capacitor needed for the circuit. Choosing a resistor of 30 kΩ the capacitance of the capacitor can be calculated with the following equation:

$$C = \frac{1}{2\pi f R}$$

The frequency f is called the high-frequency cutoff. Frequencies below the cutoff point are said to be in the passband of the filter, and frequencies above are in the rejection band of the filter. The characteristics of a resistor-capacitor filter are shown below. The vertical axis is the relative voltage or percent of the input voltage. The frequency f corresponds to 70.7 percent.

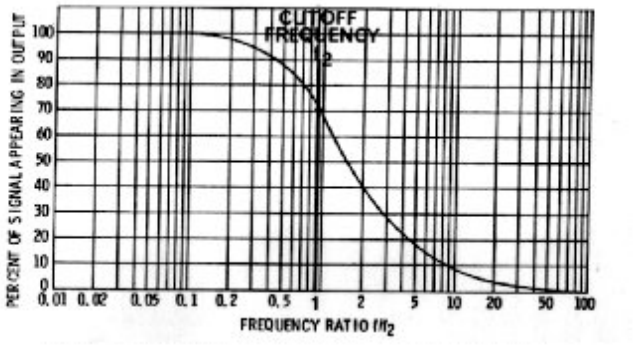

Figure C.2 Cut-off frequency chart

Using the above equation gives a capacitance of 106 pF. This is unfortunately not a standard capacitor capacitance. The closest will be a 100 pF capacitor. Using the same equation as above the cutoff frequency can be calculated to be 53.05 Hz. This is close to the desired value of 50 Hz.

Appendix D

The supersonic wind tunnel

The supersonic wind tunnel in which some of the experiments were conducted is shown for a little more clarity.

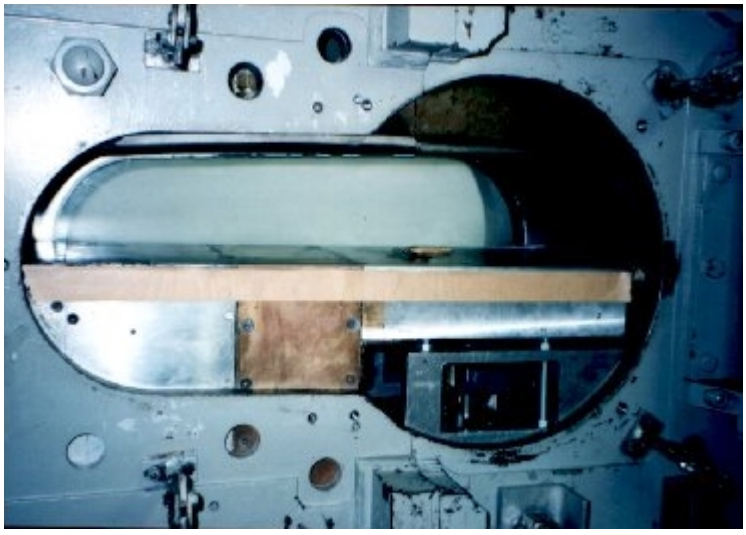

Figure D.1 Supersonic tunnel test section

In the first picture, the tunnel is shown with the side door off. The skin-friction gage was fixed in the door (not shown) at a position at a certain position not impinge by the shock.

Figure D.2 Supersonic tunnel at Virginia Tech

Next is an overview of the supersonic tunnel. Seen on the left top corner is the control valve. The air is fed into the settling-chamber and then through the nozzle and the exhaust to the outside of the laboratory.

The cascade tunnel

The most important experiments were done in the cascade tunnel. The next picture shows the tunnel and the "heater loop". Looking at figure 4.1 will give more clarity on the picture angle.

Figure D.3 Cascade tunnel

The next picture shows the blades in the cascade.

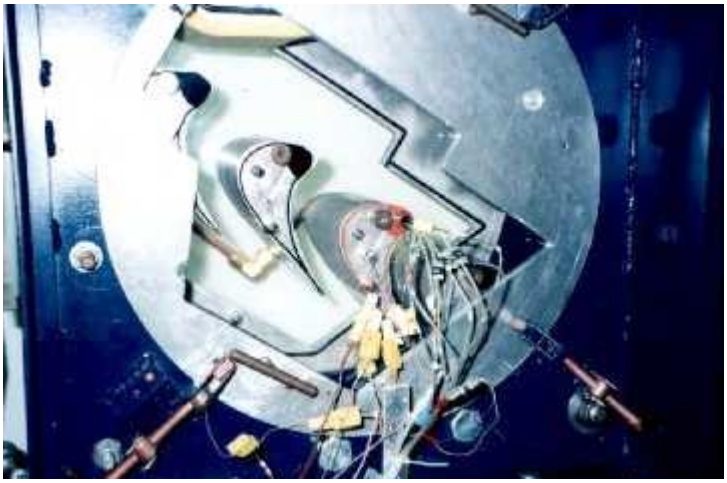

Figure D.4 Turbine blades in the cascade tunnel

Here, the instrumented blade can be seen with the thermocouples, pressure sensors and the HFS heat flux gages in. Only one of the blades is instrumented at a time, and the others are just there to complete the cascade. The additional blades help retain the true flow of gas around the instrumented blade.

The picture below shows the blade from the other side with the coolant flow coming in through the copper pipe. This cool air (± -90^{o}C) is then blown through the small holes on in

the blade surface. This cold air forms a thin film of cold air around the blade, protecting it from the hot air coming from the settling-chamber. From there the name film cooling.

Figure D.5 The pipe with cooling air blowing into the blade.

See web page: http://www.aoe.vt.edu/aoe/physical/cascade.html

Appendix E

```
      PROGRAM Implicitinverse
*     This program calculates the heat flux at the surface
*     of an object, given the surface temperature history.
*     It uses a implicit solver to solve the inverse of the
*     matrix of temperatures and incorporates this into the
*     inverse calculation of the surface flux.

      INTEGER x, t, r
      PARAMETER(x=100,t=2500,r=3)

*     x = the number of nodes in the region looked at.
*     t = the number of time steps for the run.
*     r = an index.

      REAL*8 dx(x), dt, xt, v(r)
      REAL*8 temp(x), temp1(x), tsurf(t)
      REAL*8 bft(t), bft0(x), bft1(x)
      REAL*8 nodetemp(r), nodetemp1(r), xx1(r,r), time(t), h(r,r)
      REAL*8 a(r,r), c(r), p(t)
      REAL*8 qguess(t), qsave, qb(t), qi(0:r)
      REAL*8 penetra
      REAL*8 h0(r,r), h1(r,r), h2(r,r)
      REAL*8 w, w0, w1, w2
      REAL*8 error1, error2
      REAL*8 relax, sensetivity, tollerance, one, alpha, d
      INTEGER indx(r)
      INTEGER nodes
      INTEGER count, tpos, dqct, i, j

*     dx(x) is the spacial grid spacing between nodes. In
*           this case it will be a constant, but it can be a
*           function of the possition relative to the surface.
*     dt is the timestep size.
*     bft stands for the back face temperature, measured inside
*           the object.
*     time is the time
*     qsave saves the heat flux of every iteration for the next.
*     qguess is the guessed flux used as input to the code. The
*           initial guess is usually zero.
*     penetra is the penetration into the object. Also the distance
*           from the surface to the second/interior thermocouple.
*     w and relax is the relaxation factor.
*     error1 and error2 is for interation purposes.
*     relax is the relaxation factor.
*     count is just a counter.
*     i is a index.
*     j is a index.

c   'Enter the input file name'
c   'Enter the output file name '
            OPEN(5, FILE='Aprildata.prn')
            OPEN(12,FILE='Testflux.dat')

c     Read the time, the guessed flux, the surface temperature and the
c     back face temperature.
```

```
          count = 1
1       READ(5,*,END=2) time(count), qguess(count), tsurf(count),
     +                  bft(count)
          count = count + 1
          GOTO 1

c         Nodes is the number of nodes.
2         nodes = count-1

          alpha             = 1.0d-4
          tolerance         = 1.0d-4
          one               = 1.0
          sensitivity       = 0.00001
          xt                = 0.0
          w0                = 0.0
          w1                = 1.0
          w2                = 0.0
          relax             = 1.0

c         Assume that the penetration depth is 1 inch (25 mm)

        penetra =  0.025

c         Calculate  the distance between the nodes. dfloat(x) is the
c         number of nodes

          DO count = 1,x
            dx(count) = penetra/DFLOAT(x)
          ENDDO

c         Save the measured temp in array "t"

          DO count = 1,x
           temp(count) = tsurf(1)
           bft0(count) = bft(1)
          ENDDO

          DO i = 0,r
           qi(i) = 0.0d0
          ENDDO

          DO count = 1,t
            p(count) = one
          ENDDO
            p(t) = 1.0d-2

C---- Inverse parameters

        INCLUDE 'Regularization.f'

          DO i = 1,r
           qi(i) = 1.0d-3
          ENDDO

C---- Begin inverse loop section

          WRITE(*,6) time(1),qguess(1),tsurf(1), bft(1), 0.0
          WRITE(12,6) time(1),qguess(1),tsurf(1), bft(1), 0.0

          DO 3 tpos = 2,nodes-r+1
```

```
c       dt is the time step. In this case it will stay the same throughout
c       the program.
        dt = time(tpos) - time(tpos-1)
        dqct = 0
        w = relax

C---- Iteration loop

4       CONTINUE
        dqct = dqct + 1

        DO count = 1,x
          temp1(count) = temp(count)
           bft1(count) = bft0(count)
        ENDDO

        CALL Implicit1(qb(tpos),bft(tpos),x,r,temp1,qi,dx,dt,nodetemp1,xt)

        DO j = 1,r
          DO count = 1,x
            temp1(count) = temp(count)
              bft1(count) = bft0(count)
          ENDDO
          qsave = qi(j)
          qi(j) = qsave*(one+sensitivity)+sensitivity

          CALL Implicit1(qb(tpos),bft(tpos),x,r,temp1,qi,dx,dt,nodetemp,xt)

          DO i = 1,r
            xx1(i,j) = (nodetemp(i) -nodetemp1(i))/(qi(j)-qsave)
          ENDDO

      qi(j) = qsave
        ENDDO

C---- Calculate left hand side (A matrix)

        DO i = 1,r
          DO j = 1,r
            a(i,j) = 0.0d0
            DO count = 1,r
              a(i,j) = a(i,j) + xx1(count,i)*p(tpos+count-1)
     +              *xx1(count,j)

            ENDDO
            a(i,j) = a(i,j) + h(i,j)
          ENDDO
        ENDDO

C---- Calculate temperature difference

        DO i = 1,r
          v(i) = tsurf(tpos+i-1) - nodetemp1(i)
        ENDDO

C---- Calculate right hand side (c vector)

        DO i = 1,r
          c(i) = 0.0d0
```

```
            DO count = 1,r
                        c(i) = c(i) + xx1(count,i)*p(tpos+count-1)*v(count)
     +          - h(i,count)*qi(count)

            ENDDO
           ENDDO


C---- Solve the matrix equations (A*dq = c)

           CALL ludcmp(a,r,r,indx,d)
           CALL lubksb(a,r,r,indx,c)


C---- Calculate err (dq)

           error1 = 0.0
           error2 = 0.0
           DO i = 1,r
            error1 = error1 + ABS(c(i)/qi(i))
            error2 = error2 + ABS(c(i))
           ENDDO

C---- Update count and flux "guess"
           IF (dqct.gt.5) w = relax/10.0d0
           IF (dqct.gt.15) w = relax/30.0d0

           DO i = 1,r
             qi(i) = qi(i) + w*c(i)
           ENDDO

           IF (dqct.lt.40) THEN
           IF ((error1.gt.tolerance).and.(error2.gt.tolerance)) GOTO 4
           ENDIF

C---- Direct solution for correct flux and write out results

           CALL Implicit2(qb(tpos),bft(tpos),x,temp,qi(0),qi(1),dx,dt)

           WRITE(*,6) time(tpos),qguess(tpos),tsurf(tpos), bft(tpos), qi(1)
           WRITE(12,6) time(tpos),qguess(tpos), tsurf(tpos),bft(tpos),qi(1)

C---- Bump up flux "guess" for next time step

        DO i = 0,r-1
          qi(i) = qi(i+1)
        ENDDO

3         CONTINUE

5         FORMAT(1X,F10.7,F10.7,F12.4,I3,F12.4)
6         FORMAT(5(3X,E13.6))

          END

C     Include impliter subroutine and property functions
      INCLUDE 'LUdecomp.f'
      INCLUDE 'Nodetemperatures.f'
      INCLUDE 'Alum.f'
```

File Regularization

```
      DO i = 1,r
        DO j = 1,r
          h0(i,j) = 0.0d0
          h1(i,j) = 0.0d0
          h2(i,j) = 0.0d0
        ENDDO
      ENDDO

      DO count = 1,r
        h0(count,count) = 1.0d0
      ENDDO
      DO count = 1,r-1
        h1(count,count) = -1.0d0
        h1(count,count+1) = 1.0d0
      ENDDO
      DO count = 1,r-2
        h2(count,count) = 1.0d0
        h2(count,count+1) = -2.0d0
        h2(count,count+2) = 1.0d0
      ENDDO

      DO i = 1,r
        DO j = 1,r
          h(i,j) = 0.0d0
          DO count = 1,r
            h(i,j) = h(i,j) + w0*h0(count,i)*h0(count,j)
            h(i,j) = h(i,j) + w1*h1(count,i)*h1(count,j)
            h(i,j) = h(i,j) + w2*h2(count,i)*h2(count,j)
          ENDDO
          h(i,j) = alpha*h(i,j)
        ENDDO
      ENDDO
```

File LUdecomp

```
SUBROUTINE ludcmp(a,n,np,indx,d)
    INTEGER n,np,indx(n),NMAX
    REAL*8 d,a(np,np),TINY
    PARAMETER (NMAX=500,TINY=1.0e-20)
    INTEGER i,imax,j,k
    REAL*8 aamax,dum,sum,vv(NMAX)
    d=1.0d0
    DO 12 i=1,n
      aamax=0.0d0
      DO 11 j=1,n
        IF (ABS(a(i,j)).gt.aamax) aamax=ABS(a(i,j))
11    CONTINUE
      IF (aamax.eq.0.0d0) PAUSE 'singular matrix in ludcmp'
      vv(i)=1.0d0/aamax
12    CONTINUE
    DO 19 j=1,n
      DO 14 i=1,j-1
        sum=a(i,j)
        DO 13 k=1,i-1
          sum=sum-a(i,k)*a(k,j)
13      CONTINUE
        a(i,j)=sum
```

```
14     CONTINUE
      aamax=0.0d0
      DO 16 i=j,n
       sum=a(i,j)
       DO 15 k=1,j-1
        sum=sum-a(i,k)*a(k,j)
15     CONTINUE
       a(i,j)=sum
       dum=vv(i)*ABS(sum)
       IF (dum.ge.aamax) THEN
        imax=i
        aamax=dum
       ENDIF
16     CONTINUE
      IF (j.ne.imax) THEN
       DO 17 k=1,n
        dum=a(imax,k)
        a(imax,k)=a(j,k)
        a(j,k)=dum
17     CONTINUE
       d=-d
       vv(imax)=vv(j)
      ENDIF
      indx(j)=imax
      IF(a(j,j).eq.0.0d0)a(j,j)=TINY
      IF(j.ne.n) THEN
       dum=1./a(j,j)
       DO 18 i=j+1,n
        a(i,j)=a(i,j)*dum
18     CONTINUE
      ENDIF
19    CONTINUE
     RETURN
     END


C  (C) Copr. 1986-92 Numerical Recipes Software J!V%03#1y.
     SUBROUTINE lubksb(a,n,np,indx,b)
     INTEGER n,np,indx(n)
     REAL*8 a(np,np),b(n)
     INTEGER i,ii,j,ll
     REAL*8 sum
     ii=0
     DO 12 i=1,n
      ll=indx(i)
      sum=b(ll)
      b(ll)=b(i)
      IF (ii.ne.0) THEN
       DO 11 j=ii,i-1
        sum=sum-a(i,j)*b(j)
11     CONTINUE
      ELSE IF (sum.ne.0.0d0) THEN
       ii=i
      ENDIF
      b(i)=sum
12    CONTINUE
     DO 14 i=n,1,-1
      sum=b(i)
      DO 13 j=i+1,n
       sum=sum-a(i,j)*b(j)
```

```
13     CONTINUE
       b(i)=sum/a(i,i)
14    CONTINUE
      RETURN
      END
C  (C) Copr. 1986-92 Numerical Recipes Software J!V%03#1y.
```

File Implicitinverse

```
C************************************************************
      SUBROUTINE Implicit1(qback,back,nx,ni,t,qv,dx,dti,ts,xt)
C************************************************************
C           Implicit routine to determine temperature history
C           Thomas Algorithm
C           Properties can vary with temperature


      INTEGER nx, ni
      REAL*8 t(nx), dx(nx), dti, qv(0:ni), ts(ni), xt,back
      INTEGER nmax
      REAL*8 tol,qback
      PARAMETER (nmax=500,tol=1.0d-4)

      INTEGER ct, ctr, ctil, tl, n
      REAL*8 ix, xc, q
      REAL*8 t1, dt
      REAL*8 to(nmax), a(nmax), b(nmax), c(nmax), r(nmax)

      REAL*8 dxr,dxl,dxp,tw,te
      REAL*8 rc,rcr,rcl,kr,kl,for,fol


      REAL*8 rho,cp,k,rtc

      tl = INT(8.0d0*dti*k(t(1))/dx(1)/dx(1)/rho(t(1))/cp(t(1)))
      dt = dti / DFLOAT(tl)

      ix = 0
      xc = 0.0d0
81    ix = ix + 1
      xc = xc + dx(ix)
      IF (xc.lt.xt) GOTO 81

      DO 10 ctr = 1,ni

            DO 22 ctil = 1,tl

              n = 0
        q = qv(ctr-1) + ctil*(qv(ctr)-qv(ctr-1))/DFLOAT(tl)

c     Save the previous temperature values
            DO ct = 1,nx
              to(ct) = t(ct)
            ENDDO

4           t1 = t(1)
```

```
C----Here the matrix to be solved is built.
      INCLUDE 'Implicitinverse.f'

          CALL tridag(a,b,c,r,t,nx,nx)


          IF (n.eq.200) STOP 'temp dist not converging – Implicit1'
          n = n + 1

          IF (ABS(t1-t(1))/t(1).gt.tol) GOTO 4

22        CONTINUE

          ts(ctr) = t(ix+1) - (xc-xt)*(t(ix+1)-t(ix))/dx(ix)

10    CONTINUE

      RETURN
      END

C*************************************************************
      SUBROUTINE Implicit2(qback,back,nx,t,q1,q2,dx,dti)
C*************************************************************
c           Implicit routine to determine temperature distribution
c           Thomas Algorithm
c           Specified Flux at surface
c           Properties can vary with temperature


      INTEGER nx, tpos
      REAL*8 t(nx), dx(nx), dti, q1, q2, back,qback

      INTEGER nmax
      REAL*8 tol
      PARAMETER (nmax=500,tol=1.0d-4)

      INTEGER ct, ctil, tl, n
      REAL*8 t1, dt
      REAL*8 to(nmax), a(nmax), b(nmax), c(nmax), r(nmax)

      REAL*8 dxr,dxl,dxp,tw,te
      REAL*8 rc,rcr,rcl,kr,kl,for,fol

      REAL*8 rho,cp,k,rtc

      tl = INT(8.0d0*dti*k(t(1))/dx(1)/dx(1)/rho(t(1))/cp(t(1)))
      dt = dti / DFLOAT(tl)

      DO 22 ctil = 1,tl

          n=0
     q =  q1 + ctil*(q2-q1)/DFLOAT(tl)

c         Save the previuos temperature values

          DO ct = 1,nx
            to(ct) = t(ct)
          ENDDO

4         t1 = t(1)
```

```
c       Here the matrix to be solved is built
        INCLUDE 'Implicitinverse.f'

        CALL tridag(a,b,c,r,t,nx,nx)

        IF (n.eq.200) STOP 'temp dist not converging - imp2'
        n = n + 1

        IF (ABS(t1-t(1))/t(1).gt.tol) GOTO 4

22   CONTINUE

    RETURN
    END

c*************************************************************************
        SUBROUTINE tridag(a,b,c,r,u,n,nx)
c*************************************************************************

        INTEGER n,NMAX
        REAL*8 a(n),b(n),c(n),r(n),u(n)
        PARAMETER (NMAX=500)
c         Solves
        INTEGER j
        REAL*8 bet,gam(NMAX)
        IF(b(1).eq.0.0d0) PAUSE 'tridag: rewrite equations'
        bet=b(1)
        u(1)=r(1)/bet
        DO 11 j = 2,nx
          gam(j) = c(j-1)/bet
          bet = b(j)-a(j)*gam(j)
          IF(bet.eq.0.0d0) PAUSE 'tridag failed'
          u(j) = (r(j)-a(j)*u(j-1))/bet
 11     CONTINUE
        DO 12 j = nx-1,1,-1
          u(j) = u(j)-gam(j+1)*u(j+1)
 12     CONTINUE
        RETURN
        END
```

File Alum

```
C*************************************************************
        REAl*8 FUNCTION k(t)
        REAL*8 t
*       k is the thermalconductivity
*       t is the timestep.

      k = 180

        RETURN
        END
```

```
C*****************************************************************
      REAL*8 FUNCTION rtc(t)
      REAL*8 t

      rtc = 2700*896

      RETURN
      END

C*****************************************************************
      REAL*8 FUNCTION cp(t)
      REAL*8 t
*     cp specific heat constant.

      cp = 896

      RETURN
      END

C*****************************************************************
      REAL*8 FUNCTION rho(t)
      REAL*8 t
*     rho is the density.

      rho = 2700

      RETURN
      END
```

File Implicitinverse

```
c     This section of program builds the conduction matrix.
c     It is then solved for every point in time. A semi-infinite
c     approach was used in the past, but this is improoved with
c     the use of the temperature at the back face of the object
c     as the boundary condition.

c  Variables declared in the main program.
c     t0(x),t1(x) = Temperatures
c     dx(x)= Distance between nodes.
c     rho,cp,rtc,k = Properties.
c     a(x),b(x),c(x),r(x) = Matrix coefficients
c     q,qf,dt

c     Local variables
c     dxr,dxl,dxp,tw,te
c     rc,rcr,rcl,kr,kl,for,fol


c     Node one is the flux boundary

      dxr = dx(1)
      dxp = dxr/2.0d0

      rc = rtc(t(1))

      te = (t(1) + t(2))/2.0d0
      kr = k(te)
```

```
      for = kr*dt/(rc*dxp*dxr)

      a(1) = 0.0d0
      b(1) = 1.0d0 + for
      c(1) = -for
      r(1) = to(1) + q*dt/(rc*dxp)

C     Interior boundary at a specific temperature

      a(nx) = -0.5*for
      b(nx) = 1 +for
      c(nx) = 0.0d0
      r(nx) = (0.5*for*back) + t(nx-1)

c     Interior boundary is insulated/adiabatic

*     a(nx) = -1.0d0
*     b(nx) = 1.0d0
*     c(nx) = 0.0d0
*     r(nx) = 0.0d0


      DO 1 ct = 2,nx-1

       dxr = dx(ct)
       dxl = dx(ct-1)
       dxp = (dxr+dxl)/2.0d0

       rcr = rtc(t(ct))
       rcl = rtc(t(ct))
       rc = (rcl*dxl + rcr*dxr)/(dxr+dxl)

       te = (t(ct)+t(ct+1))/2.0d0
       kr = k(te)
       tw = (t(ct)+t(ct-1))/2.0d0
       kl = k(tw)

       for = kr*dt/(rc*dxr*dxp)
       fol = kl*dt/(rc*dxl*dxp)

       a(ct) = -fol
       b(ct) = 1.0d0 + fol + for
       c(ct) = -for
       r(ct) = to(ct)

 1     CONTINUE
```