Literature Cited.

Avella, M., E. Martuscelli, B. Pascucci, M. Raimo, B. Focher and A. Marzetti. 1993. A New Class of Biodegradable Materials: Poly-3-hydroxy-butyrate/ Steam Exploded Straw Fiber Composites. I. Thermal and Impact Behavior. *Journal of Applied Polymer Science,* Vol. 49, pp. 2091-2103.

Bafna, S.S. and D.G. Baird. 1991. Impregnation in Thermoplastic Prepregs: Model and Experiments, from *36th International SAMPE Symposium,* Society for the Advancement of Material and Process Engineering.

Beshay, A. and S.V. Hoa. 1990. Improved Interface Bonding Between Cellulosic Fibers and Thermoplastics. from *35th International SAMPE Symposium and Exhibition, Vol. 35, Book 2-Advanced Materials: The Challenge for the Next Decade,* Janicki, G., Bailey, V. and H. Schjelderup, eds. Society for the Advancement of Material and Process Engineering, California, U.S.A.

Bourban, C., Karamuk, E., deFondaumiere, M.J., Ruffieux, K., Mayer, J. and E. Wintermantel. 1997. Processing and Characterization of a New Biodegradable Composite Made of a PHB/V Matrix and Regenerated Cellulosic Fibers. *Journal of Environmental Polymer Degradation,* Vol. 5, No. 3, pp. 159-166.

Broyles, N.S., R. Chan, R.M. Davis, J.J. Lesko and J.S. Riffle. 1998. Sizing of Carbon Fibers with Aqueous Solutions of Poly(vinyl pyrollidone). *Polymer,* Vol. 39, No. 12, pp. 2607-2613.

Bucher, R.A. and J.A. Hinkley. 1992. Properties of Powder-Impregnated Graphite/PEKK. from *24th International SAMPE Technical Conference, Vol. 24: Advanced Materials: Meeting the Economic Challenge,* Reinhart, T.S., Rosenow, M., Cull, R.A. and E. Struckholt, eds. Society for the Advancement of Material and Process Engineering, Canada, U.S.A.

Canche-Escamilla, G., Cauich-Cupul, J.I., Mendizabal, E., Puig, J.E., Vazquez-Torres, H. and P.J. Herrera-Franco. 1999. Mechanical Properties of Acrylate-Grafted Henequen Cellulose Fibers and Their Application in Composites. *Composites Part A,* Vol. 30, pp. 349-359.

Cano, R.J. and J.M. Marchello. 1995. Solution Prepreg Quality Control. from *40th International SAMPE Symposium, Vol. 40, Book 1-Materials Challenge; Diversification and the Future,* Harmston, D., Carson, R., Bailey, G.D. and F.J. Riel, eds. Society for the Advancement of Material and Process Engineering, California, U.S.A.

Chand, N., Verma, S. and A.C. Khazanchi. 1989. SEM and Strength Characteristics of

Acetylated Sisal Fibre. *Journal of Materials Science Letters,* Vol. 8, No. 11, pp. 1307-1309.

Chang, Y.S., Lesko, J.J., Case, S.W., Dillard, D.A. and K.L. Reifsnider. 1994. The Effect of Fiber-Matrix Interphase Properties on the Quasi-Static Performance of Thermoplastic Composites. *Journal of Thermoplastic Composite Materials,* Vol. 7, p. 311-324.

Chen, M.L., Ueta, S. and M. Takayanagi. Improvement of Adhesion of Kevlar Fiber to Poly(methyl methacrylate). *Polymer Journal,* Vol. 20, No. 8, pp. 673-677.

Chen, X., Q. Guo and Y. Mi. 1998. Bamboo Fiber-Reinforced Polypropylene Composites: A Study of the Mechanical Properties. *Journal of Applied Polymer Science,* Vol. 69, pp. 1891-1899.

Dalvag, H., C. Klason and H.E. Stromvall. 1985. The Efficiency of Cellulosic Fillers in Common Thermoplastics. Part II. Filling with Processing Aids and Coupling Agents. *International Journal of Polymeric Materials,* Vol. 11, pp. 9-38.

Devi, L.U., S.S. Bhagawan, S. Thomas. 1997. Mechanical Properties of Pineapple Leaf Fiber Reinforced Polyester Composites. *Journal of Applied Polymer Science,* Vol. 64, pp. 1739.

Drzal, L.T. and M. Madhukar. 1993. Fibre-Matrix Adhesion and its Relationship to Composite Mechanical Properties. *Journal of Materials Science,* Vol. 28, pp. 569-610.

Felix, J.M. and P. Gatenholm. 1990. Evaluation of Surface Treatments for Cellulose Fibers and Interfacial Interactions in Composites. from *Proceedings of the Third International Conference on Composite Interfaces-Controlled Interphases in Composite Materials,* H. Ishida, ed. Elsevier Science Publishers, New York, U.S.A.

Felix, J.M. and P. Gatenholm. 1991. The Nature of Adhesion in Composites of Modified Cellulose Fibers and Polypropylene. *Journal of Applied Polymer Science,* Vol. 42, pp. 609-620.

Felix, J.M. and P. Gatenholm. 1993. Controlled Interactions in Cellulose-Polymer Composites. I: Effect on Mechanical Properties. *Polymer Composites,* Vol. 14, No. 6, pp. 449-457.

Geethamma, V.G., K.T. Mathew, R. Lakshminarayanan and S. Thomas. 1997. Composites of Short Coir Fibers and Natural Rubber: Effect of Chemical Modification, Loading and Orientation of Fiber. *Polymer,* Vol. 39, No. 6-7, pp. 1483-1491.

Gent, A.N. and G.R. Hamed. 1983. Fundamentals of Adhesion. from *Adhesive Bonding of Wood and Other Structural Materials Vol. III*, Blomquist, R.F., Christiansen, A.W., Gillespie, R.H. and G.E. Myers, eds. Educational Modules for Materials Science and Engineering Project, The Pennsylvania State University, Pa., U.S.A.

Glasser, W.G. and R.M. Davis. 1999. Advanced Thermoplastic, Cellulosic Composites. *Technical Progress Report submitted to Eastman Chemical Company and Courtaulds Corporate Technology*, Virginia Tech, Blacksburg, VA.

Glasser, W.G., R. Taib, R.K. Jain and R. Kander. 1999. Fiber-Reinforced Cellulosic Thermoplastic Composites. *Journal of Applied Polymer Science*, Vol. 73, pp. 1329-1340.

Good, R.J. 1979. Contact Angles and the Surface Free Energy of Solids. From *Surface and Colloid Science Vol. II – Experimental Methods.* Good, R.J. and R.R. Stromberg, eds. Plenum Press, New York, USA.

Hancox, N.L. 1989. High Performance Composites with Thermoplastic Matrices. in *Concise Encyclopedia of Composite Materials,* A. Kelly, ed. Pergamon Press, Oxford, England.

Herrera-Franco, P.J. and M.D.J. Aguilar-Vega. 1997. Effect of Fiber Treatment on the Mechanical Properties of LDPE-Henequen Cellulosic Fiber Composites. *Journal of Applied Polymer Science,* pp. 197-207.

Hill, C.A.S., Jones, D., Strickland, G. and N.S. Cetin. 1998. Kinetic and Mechanistic Aspects of the Acetylation of Wood with Acetic Anhydride. *Holzforschung,* Vol. 52, No. 6, pp. 623-629.

Hou, M., Ye, L., Lee, H.J. and Y.W. Mai. 1998. Manufacture of a Carbon Fabric Reinforced Polyetherimide (CF/PEI) Composite Material. *Composites Science and Technology*, Vol. 58, pp. 181-190.

Huang, M.R. and X.G. Li. 1998. Thermal Degradation of Cellulose and Cellulose Esters. *Journal of Applied Polymer Science,* Vol. 68, pp. 293-304.

Itoi, M., Yamada, Y. and R.B. Pipes. 1992. Effect of Surface Treatment of Pitch Based Carbon Fiber on Mechanical Properties of Polyethernitrile Composites. *Polymer Composites*, Vol. 13, No. 1, pp. 15-29.

Jain, R.K., Lal, K. and H.L. Bhatnagar. 1987. Thermal, Morphological and Spectroscopic Studies on Cellulose Modified with Phosphorus, Nitrogen, Sulphur and Halogens. *Journal of Applied Polymer Science,* Vol. 33, pp. 247-282.

Joly, C., R. Gauthier and B. Chabert. 1996. Physical Chemistry of the Interface in

Polypropylene/ Cellulosic Fiber Composites. *Composites Science and Technology,* Vol. 56, pp. 761-765.

Kim, T.J., Y.M. Lee and S.S. Im. 1997. The Preparation and Characteristics of Low Density Polyethylene Composites Containing Cellulose Treated with Cellulase. *Polymer Composites,* Vol. 18, No. 3, pp. 273-282.

Klason, C., J. Kubat and H.E. Stromvall. 1984. The Efficiency of Cellulosic Fillers in Common Thermoplastics. Part I. Filling without Processing Aids or Coupling Agents. *International Journal of Polymeric Materials,* Vol. 10, pp. 159-187.

Klason, C., J. Kubat and P. Gatenholm. 1989. New Wood-based Composites with Thermoplastics. from *Cellulosics Utilization: Research and Rewards in Cellulosics,* H. Inagaki and G.O. Phillips, ed. Elsevier Science Publishers, Essex, England.

Lacroix, F.v., Werwer, M. and K. Schulte. 1998. Solution Impregnation of Polyethylene Fiber/Polyethylene Matrix Composites. *Composites Part A,* Vol. 29A, pp. 371-376.

Maldas, D., B.V. Kokta and C. Daneault. 1989. Influence of Coupling Agents and Treatments on the Mechanical Properties of Cellulose Fiber-Polystyrene Composites. *Journal of Applied Polymer Science,* Vol. 37, pp. 751-775.

Maldas, D., B.V. Kokta and C. Daneault. 1989. The Mechanical Properties of Wood Fiber Reinforced Polymethylmethacrylate. *International Journal of Polymeric Materials,* Vol. 12, pp. 297-323.

Masson, J.F. and R. St. John Manley. 1992. Solid-State NMR of Some Cellulose/ Synthetic Polymer Blends. *Macromolecules,* Vol. 25, pp. 589-592.

Mayer, C., Wang, X. and M. Neitzel. 1998. Macro- and Micro- Impregnation Phenomena in Continuous Manufacturing of Fabric Reinforced Thermoplastic Composites. *Composites Part A,* Vol. 29A, pp. 783-793.

Michell, A.J., Vaughan, J.E. and D. Willis. 1978. Wood Fiber-Synthetic Polymer Composites. II. Laminates of Treated Fibers and Polyolefins. *Journal of Applied Polymer Science,* Vol. 22, pp. 2047-2061.

Moon, C.K., Um, Y.S., Lee, J.O., Cho, H.H. and C.H. Park. 1993. Development of Thermoplastic Prepreg by the Solution-Bond Method. *Journal of Applied Polymer Science,* Vol. 47, pp. 195-188.

Mortimer, S.A., A.A. Peguy and R.C. Ball. 1996. Influence of the Physical Process Parameters on the Structure Formation of Lyocell Fibers. *Cellulose Chemistry and Technology,* Vol. 30, pp. 251-266.

Oksman, K. and C. Clemons. 1998. Mechanical Properties and Morphology of Impact Modified Polypropylene-Wood Flour Composites. *Journal of Applied Polymer Science,* Vol. 67, pp. 1503-1513.

Patel, N. and L.J. Lee. 1996. Modeling of Void Formation and Removal in Liquid Composite Molding. Part I: Wettability Analysis. *Polymer Composites,* Vol. 17, No. 1, pp. 96-103.

Raj, R.G. and B.V. Kokta. 1989. Compounding of Cellulose Fibers with Polypropylene: Effect of Fiber Treatment on Dispersion in the Polymer Matrix. *Journal of Applied Polymer Science,* Vol. 38, pp. 1987-1996.

Raj, R.G., B.V. Kokta, D. Maldas and C. Daneault. 1989. Use of Wood Fibers in Thermoplastics. VII. The Effect of Coupling Agents in Polyethylene-Wood Fiber Composites. *Journal of Applied Polymer Science,* Vol. 37, pp. 1089-1103.

Ramsden, M.J. and F.S.R. Blake. 1997. A Kinetic Study of the Acetylation of Cellulose, Hemicellulose and Lignin Components in Wood. *Wood Science and Technology,* Vol. 31, pp. 45-50.

Rana, A.K., R.K. Basak, B.C. Mitra, M. Lawther and A.N. Banerjee. 1997. Studies of Acetylation of Jute using Simplified Procedure and Its Characterization. *Journal of Applied Polymer Science,* Vol. 64, pp. 1517-1523.

Rowell, R.M. 1990. Chemical Modification of Lignocellulosic Fibers To Produce High Performance Composites. in *Agricultural and Synthetic Polymers: Biodegradability and Utilization*, Glass, J.E. and G. Swift, eds. American Chemical Society, USA.

Rowell, R.M., Cleary, B.A., J.S. Rowell, C. Clemons and R.A. Young. 1993. Results of Chemical Modification of Lignocellulosic Fibers For Use in Composites. in *Wood Fiber/ Polymer Composites: Fundamental Concepts, Processes and Material Options,* Wolcott, M.P., ed. Forest Products Society, USA.

Rowell, R.M., Kawai, S. and M. Inoue. 1995. Dimensionally Stabilized, Very Low Density Fiberboard. *Wood and Fiber Science,* Vol. 27, No. 4, pp. 428-436.

Rowell, R.M., Simonson, R., Hess, S., Plackett, D.V., Cronshaw, D. and E. Dunningham. 1994. Acetyl Distribution in Acetylated Whole Wood and Reactivity of Isolated Wood Cell-Wall Components to Acetic Anhydride. *Wood and Fiber Science,* Vol. 26, No. 1, pp. 11-18.

Rowell, R.M., Tillman, A.M. and R. Simonson. 1986. A Simplified Procedure for the Acetylation of Hardwood and Softwood Flakes for Flakeboard Production. *Journal of Wood Chemistry and Technology,* Vol. 6, No. 3, pp. 427-448.

Rozman, H.D., Kon, B.K., Abusamah, A., Kumar, R.N. and Z.A. Mohd. Ishak. 1998. Rubberwood-High Density Polyethylene Composites: Effect of Filler Size and Coupling Agents on Mechanical Properties. *Journal of Applied Polymer Science*, Vol. 69, pp. 1993-2004.

Samaranayake, G. and W.G. Glasser. 1993. Cellulose Derivatives with Low DS. II. Analysis of Alkanoates. *Carbohydrate Polymers*, Vol. 22, pp. 79-86.

Saphieha, S., Allard, P. and Y.H. Zang. 1990. Dicumyl Peroxide-Modified Cellulose/ LLDPE Composites. *Journal of Applied Polymer Science,* Vol. 41, pp. 2039-2048.

Schmidt, S.R. and R.G. Launsby. 1993. Understanding Industrial Designed Experiments. Air Academy Press, Colorado, U.S.A.

Schonhorn, H. 1981. Adhesion and Adhesives: Interactions at Interfaces. from *Adhesion in Cellulosic and Wood Based Composites,* J.F. Oliver, ed. Plenum Press, London, England.

Shaw, D.J. 1992. Introduction to Colloid and Surface Chemistry. Butterworth-Heinemann Publishing, Oxford, England.

Simmons, T.G. 1993. Effect of Mold Parameters on Consolidation of Thermoplastic Composite Laminates. from *38th International SAMPE Symposium.*

Smith, F.C., L.D. Moloney and F.L. Matthews. 1996. Fabrication of Woven Carbon Fibre/ Polycarbonate Repair Patches. *Composites Part A,* Vol. 27A, pp. 1089-1095.

Takase, S. and N. Shiraishi. 1989. Studies on Composites from Wood and Polypropylenes. II. *Journal of Applied Polymer Science,* Vol. 37, pp. 645-659.

Vovor, T., Robitaille, F. and R. Gauvin. 1994. Optimum Consolidation of Prepreg Thermoplastic Composites using Statistical Experiment-Planning Methods. from *39th International SAMPE Symposium.*

Westin, M. and R. Simonson. 1992. High-Performance Composites from Acetylated Wood Fibres. in *Pacific Rim Bio-Based Composites Symposium,* Plackett, D.V. and E.A. Dunningham, eds. New Zealand Forest Research Institute, New Zealand.

Wu, G.M., J.M. Schultze, D.J. Hodge and F.N. Cogswell. 1990. Solution Impregnation of Carbon Fiber Reinforced Poly(ethersulphone) Composites. from *ANTEC 90 Conference Proceedings - Plastics in the Environment: Yesterday, Today and Tomorrow,* Texas, U.S.A.

Yu, T.H. and R.M. Davis. 1993. The Effect of Processing Conditions on the Properties of Carbon Fiber-LaRC TPI Composites Made by Suspension Prepregging. *Journal of Thermoplastic Composite Materials,* Vol. 6, pp. 62-90.

Zaborskaya, L.V., Dovgualo, V.A. and O.R. Yurkevich. 1995. On the Interaction of Molten Thermoplastics and Fiber Surfaces. *Journal of Adhesion Science and Technology,* Vol. 9, No. 1, pp. 61-71.

APPENDIX A.

Finite Element Code for a Diffusing Reaction within a Finite Cylinder

Program DifRxnCy

```
!! Version 1
!! This program is the finite element code for a diffusing reaction within a finite cylinder

!! This is where the key variables are defined
        implicit real*8(a-h,o-z)
        real (kind=8), dimension (:), allocatable :: Arrayvertexx, Arrayvertexy
        !! Arrayvertexx and y are the vertex coordinates, arranged according to vertex
number label
        integer, dimension (:,:), allocatable :: iarrayelemvert
        !!iarrayelemvert stores the element numbers consecutively, followed by the vertex
numbers
        !!belonging to that element
        real (kind=8), dimension (:), allocatable :: Arrayintercept
        !! arrayintercept stores the intercepts for the r linear function of ks-eye
        real (kind=8), dimension (:,:), allocatable :: ArraytimeAC
        !! arraytimeAC is the assembled array for the time derivative constants for A and
C
        !!(diagonal one's assembled)
        !! and also goes with the mixed constants term for species A
        real (kind=8), dimension (:,:), allocatable :: ArraynegtimeAC
        !! this matrix goes with the mixed constants term for species C
        real (kind=8), dimension (:,:), allocatable :: ArraytimeB
        !! arraytimeB is the assembled array for the time derivative constants for B
        real (kind=8) pos13, ineg13, convpos, convneg
        !! These are the numbers that are used in the Gaussian quadrature
        real (kind=8), dimension (:,:), allocatable :: ArrayB
        !! arrayB is the assembled array for the B constants
        real (kind=8) slope, deltatime, detje
        !! slope is the slope for the linear transformation of ks-eye into r
        real (kind=8), dimension (:,:), allocatable :: ArrayAB
        !! arrayAB is the assembled array for nonlinear term
        integer, dimension (:), allocatable :: iArrayboundary
        !! iarrayboundary stores the boundary vertex numbers
        real (kind=8), dimension (:), allocatable :: Arrayinitialvalues
        !! Arrayinitialvalues is the array that contains the initial values of c's (A-C)
        real (kind=8), dimension (:), allocatable :: Arraynewguess, Arraynewguess1
        !! Arraynewguess stores the iterative new guess values for the constants A-C
        real (kind=8), dimension (:), allocatable :: Arrayiterativefunction
```

```
        !! Arrayiterativefunction is the array that holds the 'F' vector for the Newton
Solver
        integer, dimension (:), allocatable :: ipvt
        !! ipvt is the integer array used in the lin alg solver
        real (kind=8), dimension (:,:), allocatable :: ArrayJacobian
        !! arrayJacobian is the array for the solved-for Jacobian
        real (kind=8), dimension (:,:), allocatable :: ArrayPlotter

!! Introductory Remarks
        print *,'This program computes solutions for a first order diffusing reaction'
        print *,' within a finite cylinder'
        print
*,'*****************************************************************'

!! This is where the neccessary variables are read into the program
        print *,'What is the radius of the cylinder?'
        read *,radius
      !! radius is the radius of the cylinder
        print *,'What is half of the height of the cylinder?'
        read *,halfht
        !! halfht is half of the height of the cylinder
        print *,'What is the number of elements desired for the r (radial) direction?'
        read *,Nr
        !! Nr is the number of elements in the r direction
        print *,'What is the number of elements desired for the z (height) direction?'
        read *,Nz
        !! Nz is the number of elements in the z direction
        print *,'What is the initial concentration of hydroxyl groups?'
        read *,Ai
        !! Ai is the initial concentration of hydroxyl groups
        print *,'What is the initial concentration of boundary anhydride groups?'
        read *,Bi
        !! Bi is the initial concentration of hydroxyl groups
        print *,'What is the diffusion coefficient for the diffusing reactant?'
        read *,D
        ! D is the diffusion coefficient for the diffusing reactant
        print *,'What is the rate constant for the reaction?'
        read *,rtcnst
        !! rtcnst is the rate constant for the reaction
        print *,'For how long do you want this diffusing reaction to take place?'
        read *,time
        !! time is the amount of time that this diffusing reaction is taking place
        print *,'How many time steps do you want to split one unit time into?'
        read *,itimes
        itimestep = itimes * time
        print *,'There are going to be ',itimestep,' timesteps in this calculation'
```

```
        !! itimestep is the amount of time steps
        deltatime = time / itimestep
        !! deltatime is the timestep length in time

!! This is where the arrays are allocated
        ivertextotal = (Nr * Nz) + Nr + Nz + 1.0
        !! ivertextotal is the total number of vertices for the problem
        itotelmnumb = Nr * Nz
        !! itotelmnumb is the total number of elements
        detje = (radius * halfht)/(Nr * Nz)
        !! detje is the determinant of the Jacobian of the transformation
        pos13 = (3.0**(-0.5))
        aneg13 = pos13*(-1.0)
        convpos = (pos13*(0.5))+(0.5)
        convneg = (aneg13*(0.5))+(0.5)
        slope = radius / Nr

        Allocate (arrayvertexx(ivertextotal), &
                stat = iallocation_status)

        if (iallocation_status > 0) then
                print *,'Allocation Error'
                stop
        end if

        Allocate (arrayvertexy(ivertextotal), &
                stat = iallocation_status)

        if (iallocation_status > 0) then
                print *,'Allocation Error'
                stop
        end if

        Allocate (iarrayelemvert(itotelmnumb,5), &
                stat = iallocation_status)

        if (iallocation_status > 0) then
                print *,'Allocation Error'
                stop
        end if

        Allocate (arrayintercept(itotelmnumb), &
                stat = iallocation_status)

        if (iallocation_status > 0) then
                print *,'Allocation Error'
```

```
        stop
end if

Allocate (arraytimeac(ivertextotal,ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arraynegtimeac(ivertextotal,ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arraytimeb(ivertextotal,ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arrayb(ivertextotal,ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arrayab(ivertextotal,ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (iarrayboundary(Nz+Nr+1), &
        stat = iallocation_status)
```

```
if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arrayinitialvalues(3*ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arraynewguess(3*ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arraynewguess1(3*ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arrayiterativefunction(3*ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (ipvt(3*ivertextotal), &
        stat = iallocation_status)

if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
end if

Allocate (arrayJacobian(3*ivertextotal,3*ivertextotal), &
```

```
        stat = iallocation_status)

    if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
    end if

    Allocate (arrayplotter(Nz+1,Nr+1), &
        stat = iallocation_status)

    if (iallocation_status > 0) then
        print *,'Allocation Error'
        stop
    end if

!! This next part of the program generates the mesh and stores the vertex coordinates
!! by vertex number.  The vertices will be numbered by starting in the lower left hand
corner
!! of the object (center of the cylinder) and then numbering consecutively moving right.
!! At the right hand boundary,
!! the numbering restarts at the lefthandside, next row up.  There will be two matrices,
!! one that stores the x values and the other stores the y values.
    do i = 1, Nz+1, 1
        do j = 1, Nr+1, 1
            xvertex = radius * (j-1.0) / Nr
            !! xvertex is the vertex x coordinate
            yvertex = halfht * (i-1.0) / Nz
            !! yvertex is the vertex y coordinate
            ivertexnumber = j + ((i - 1) * (Nr + 1))
            !! ivertexnumber is the vertex number label
            Arrayvertexx(ivertexnumber) = xvertex
            Arrayvertexy(ivertexnumber) = yvertex
        end do
    end do

!! This next part of the program generates a matrix that stores in the first column: element
number label,
!! the second column: lower left hand vertex label, the third column: lower right hand
vertex label,
!! the fourth column: upper left hand vertex label, and the fifth column: upper right hand
vertex label.
!! The elements are numbered in the same manner as the vertices were
    do i = 1, Nz, 1
        do j = 1, Nr, 1
            ielemnumber = j + ((i - 1)*(Nr))
            iarrayelemvert(ielemnumber,1) = ielemnumber
```

```
                        iarrayelemvert(ielemnumber,2) = j + ((i - 1)*(Nr)) + (i - 1)
                        iarrayelemvert(ielemnumber,3) =
(iarrayelemvert(ielemnumber,2))+1
                        iarrayelemvert(ielemnumber,4) =
(iarrayelemvert(ielemnumber,2))+(Nr+1)
                        iarrayelemvert(ielemnumber,5) =
(iarrayelemvert(ielemnumber,4))+1
                end do
        end do

        open (1,file='iarrayelemvert.dat')
                do i = 1, itotelmnumb, 1
                        write (1,*) (iarrayelemvert(i,j),j = 1, 5)
                end do

!! This next part stores the intercepts of the linear transformation of the real vertices to
the
!!canonical element (0,0-1,1)
!! Each element has its own intercept (note that the slopes of the linear transformation are
not
!!element dependent)
!! This is needed for the r direction only
        do i = 1, itotelmnumb, 1
                arrayintercept(i) = arrayvertexx(iarrayelemvert(i,2))
        end do

        open (1,file='intercept.dat')
                do i = 1, itotelmnumb, 1
                        write (1,*) arrayintercept(i)
                end do

!! Now for global matrix assembly
!! The first matrix to be assembled is the one multiplied by the species A (and C) solution
time
!!derivative term
!! Its the one with just a diagonal row of 1's
!! This assembled matrix also goes with the mixed constant term for species A and C
        do i = 1, itotelmnumb, 1
                i1 = iarrayelemvert(i,2)
                i2 = iarrayelemvert(i,3)
                i3 = iarrayelemvert(i,4)
                i4 = iarrayelemvert(i,5)
                ArraytimeAC(i1,i1) = ArraytimeAC(i1,i1) + 1.0
                ArraytimeAC(i2,i2) = ArraytimeAC(i2,i2) + 1.0
                ArraytimeAC(i3,i3) = ArraytimeAC(i3,i3) + 1.0
                ArraytimeAC(i4,i4) = ArraytimeAC(i4,i4) + 1.0
```

```
        end do

        open (1, file='timeAC.dat')
                do i = 1, ivertextotal, 1
                        write (1,*) (arraytimeac(i,j),j = 1,ivertextotal)
                end do

        ArraynegtimeAC = ArraytimeAC * (-1.0)

!! The second one is the time derivative part for species B
        do i = 1, itotelmnumb, 1
                i1 = iarrayelemvert(i,2)
                i2 = iarrayelemvert(i,3)
                i3 = iarrayelemvert(i,4)
                i4 = iarrayelemvert(i,5)
                ArraytimeB(i1,i1) = ArraytimeB(i1,i1) + (1.0/9.0)
                ArraytimeB(i1,i2) = ArraytimeB(i1,i2) + (1.0/18.0)
                ArraytimeB(i1,i3) = ArraytimeB(i1,i3) + (1.0/18.0)
                ArraytimeB(i1,i4) = ArraytimeB(i1,i4) + (1.0/36.0)
                ArraytimeB(i2,i1) = ArraytimeB(i2,i1) + (1.0/18.0)
                ArraytimeB(i2,i2) = ArraytimeB(i2,i2) + (1.0/9.0)
                ArraytimeB(i2,i3) = ArraytimeB(i2,i3) + (1.0/36.0)
                ArraytimeB(i2,i4) = ArraytimeB(i2,i4) + (1.0/18.0)
                ArraytimeB(i3,i1) = ArraytimeB(i3,i1) + (1.0/18.0)
                ArraytimeB(i3,i2) = ArraytimeB(i3,i2) + (1.0/36.0)
                ArraytimeB(i3,i3) = ArraytimeB(i3,i3) + (1.0/9.0)
                ArraytimeB(i3,i4) = ArraytimeB(i3,i4) + (1.0/18.0)
                ArraytimeB(i4,i1) = ArraytimeB(i4,i1) + (1.0/36.0)
                ArraytimeB(i4,i2) = ArraytimeB(i4,i2) + (1.0/18.0)
                ArraytimeB(i4,i3) = ArraytimeB(i4,i3) + (1.0/18.0)
                ArraytimeB(i4,i4) = ArraytimeB(i4,i4) + (1.0/9.0)
        end do

        ArraytimeB = detje * ArraytimeB

        open (1, file='timeB.dat')
                do i = 1, ivertextotal, 1
                        write (1,*) (arraytimeB(i,j),j = 1,ivertextotal)
                end do

!! The third one is the constant part for species B
        do i = 1, itotelmnumb, 1
                i1 = iarrayelemvert(i,2)
                i2 = iarrayelemvert(i,3)
                i3 = iarrayelemvert(i,4)
                i4 = iarrayelemvert(i,5)
```

```
                ArrayB(i1,i1) = ArrayB(i1,i1)&
                        & + (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i1,i2) = ArrayB(i1,i2)&
                        & - (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i1,i3) = ArrayB(i1,i3)&
                        & + (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i1,i4) = ArrayB(i1,i4)&
                        & - (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i2,i1) = ArrayB(i2,i1)&
                        & - (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i2,i2) = ArrayB(i2,i2)&
                        & + (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i2,i3) = ArrayB(i2,i3)&
                        & - (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i2,i4) = ArrayB(i2,i4)&
                        & + (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i3,i1) = ArrayB(i3,i1)&
                        & + (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i3,i2) = ArrayB(i3,i2)&
                        & - (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i3,i3) = ArrayB(i3,i3)&
                        & + (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i3,i4) = ArrayB(i3,i4)&
                        & - (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i4,i1) = ArrayB(i4,i1)&
                        & - (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i4,i2) = ArrayB(i4,i2)&
                        & + (((Nr**2.0)/(6.0*(radius**2.0))) -
((Nz**2.0)/(3.0*(halfht**2.0))))
                ArrayB(i4,i3) = ArrayB(i4,i3)&
                        & - (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(6.0*(halfht**2.0))))
                ArrayB(i4,i4) = ArrayB(i4,i4)&
```

```
                    & + (((Nr**2.0)/(3.0*(radius**2.0))) +
((Nz**2.0)/(3.0*(halfht**2.0))))

                aintegrateG = AF(convpos,convpos)/(slope*convpos + arrayintercept(i))
                aintegrateG = aintegrateG + (AF(convpos,convneg)/(slope * convpos +
arrayintercept(i)))
                aintegrateG = aintegrateG + (AF(convneg,convpos)/(slope * convneg +
arrayintercept(i)))
                aintegrateG = aintegrateG + (AF(convneg,convneg)/(slope * convneg +
arrayintercept(i)))
                aintegrateG = (0.25) * (aintegrateG)
                aintegrateG = Nr * aintegrateG / radius
                ArrayB(i1,i1) = ArrayB(i1,i1) + aintegrateG*(-1.0)


                aintegrateH = AF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateH = aintegrateH +
AF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateH = aintegrateH +
AF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateH = aintegrateH +
AF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateH = (0.25)*(aintegrateH)
                aintegrateH = -1.0 * Nr * aintegrateH / radius
                ArrayB(i1,i2) = ArrayB(i1,i2) + aintegrateH*(-1.0)


                aintegrateJ = BF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateJ = aintegrateJ +
BF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateJ = aintegrateJ +
BF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateJ = aintegrateJ +
BF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateJ = (0.25) * (aintegrateJ)
                aintegrateJ = -1.0 * Nr * aintegrateJ/radius
                ArrayB(i1,i3) = ArrayB(i1,i3) + aintegrateJ*(-1.0)


                aintegrateI = BF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateI = aintegrateI +
BF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateI = aintegrateI +
BF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateI = aintegrateI +
BF(convneg,convneg)/(slope*convneg+arrayintercept(i))
```

```
                aintegrateI = (0.25)*(aintegrateI)
                aintegrateI = Nr*aintegrateI/radius
                ArrayB(i1,i4) = ArrayB(i1,i4) + aintegrateI*(-1.0)


                aintegrateK = CF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateK = aintegrateK +
CF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateK = aintegrateK +
CF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateK = aintegrateK +
CF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateK = (0.25)*(aintegrateK)
                aintegrateK = Nr*aintegrateK/radius
                ArrayB(i2,i1) = ArrayB(i2,i1) + aintegrateK*(-1.0)


                aintegrateL = CF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateL = aintegrateL +
CF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateL = aintegrateL +
CF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateL = aintegrateL +
CF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateL = (0.25)*(aintegrateL)
                aintegrateL = -1.0 * Nr * aintegrateL/radius
                ArrayB(i2,i2) = ArrayB(i2,i2) + aintegrateL*(-1.0)

                aintegrateN = DF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateN = aintegrateN +
DF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateN = aintegrateN +
DF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateN = aintegrateN +
DF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateN = (0.25)*(aintegrateN)
                aintegrateN = -1.0*Nr*aintegrateN/radius
                ArrayB(i2,i3) = ArrayB(i2,i3) + aintegrateN*(-1.0)

                aintegrateM = DF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateM = aintegrateM +
DF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateM = aintegrateM +
DF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateM = aintegrateM +
DF(convneg,convneg)/(slope*convneg+arrayintercept(i))
```

```
                aintegrateM = (0.25)*(aintegrateM)
                aintegrateM = Nr*aintegrateM/radius
                ArrayB(i2,i4) = ArrayB(i2,i4) + aintegrateM*(-1.0)


                aintegrateS = BF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateS = aintegrateS +
BF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateS = aintegrateS +
BF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateS = aintegrateS +
BF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateS = (0.25)*(aintegrateS)
                aintegrateS = -1.0*Nr*aintegrateS/radius
                ArrayB(i3,i1) = ArrayB(i3,i1) + aintegrateS*(-1.0)

                aintegrateT = BF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateT = aintegrateT +
BF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateT = aintegrateT +
BF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateT = aintegrateT +
BF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateT = (0.25)*(aintegrateT)
                aintegrateT = Nr*aintegrateT/radius
                ArrayB(i3,i2) = ArrayB(i3,i2) + aintegrateT*(-1.0)

                aintegrateV = FF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateV = aintegrateV +
FF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateV = aintegrateV +
FF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateV = aintegrateV +
FF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateV = (0.25)*(aintegrateV)
                aintegrateV = -1.0*Nr*aintegrateV/radius
                ArrayB(i3,i3) = ArrayB(i3,i3) + aintegrateV*(-1.0)

                aintegrateU = FF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateU = aintegrateU +
FF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateU = aintegrateU +
FF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateU = aintegrateU +
FF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateU = (0.25)*(aintegrateU)
```

```
                aintegrateU = Nr*aintegrateU/radius
                ArrayB(i3,i4) = ArrayB(i3,i4) + aintegrateU*(-1.0)

                aintegrateO = DF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateO = aintegrateO +
DF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateO = aintegrateO +
DF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateO = aintegrateO +
DF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateO = (0.25)*(aintegrateO)
                aintegrateO = -1.0*Nr*aintegrateO/radius
                ArrayB(i4,i1) = ArrayB(i4,i1) + aintegrateO*(-1.0)

                aintegrateP = DF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateP = aintegrateP +
DF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateP = aintegrateP +
DF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateP = aintegrateP +
DF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateP = (0.25)*(aintegrateP)
                aintegrateP = Nr*aintegrateP/radius
                ArrayB(i4,i2) = ArrayB(i4,i2) + aintegrateP*(-1.0)

                aintegrateR = EF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateR = aintegrateR +
EF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateR = aintegrateR +
EF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateR = aintegrateR +
EF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateR = (0.25)*(aintegrateR)
                aintegrateR = -1.0*Nr*aintegrateR/radius
                ArrayB(i4,i3) = ArrayB(i4,i3) + aintegrateR*(-1.0)

                aintegrateQ = EF(convpos,convpos)/(slope*convpos+arrayintercept(i))
                aintegrateQ = aintegrateQ +
EF(convpos,convneg)/(slope*convpos+arrayintercept(i))
                aintegrateQ = aintegrateQ +
EF(convneg,convpos)/(slope*convneg+arrayintercept(i))
                aintegrateQ = aintegrateQ +
EF(convneg,convneg)/(slope*convneg+arrayintercept(i))
                aintegrateQ = (0.25)*(aintegrateQ)
                aintegrateQ = Nr*aintegrateQ/radius
                ArrayB(i4,i4) = ArrayB(i4,i4) + aintegrateQ*(-1.0)
```

```
        end do

        ArrayB = D * detje * ArrayB

        open (1, file='B.dat')
                do i = 1, ivertextotal, 1
                        write (1,*) (arrayB(i,j),j = 1,ivertextotal)
                end do

 !! The fourth one is the nonlinear part that goes with the constants a times b (for the b
species part)
        do i = 1, itotelmnumb, 1
                i1 = iarrayelemvert(i,2)
                i2 = iarrayelemvert(i,3)
                i3 = iarrayelemvert(i,4)
                i4 = iarrayelemvert(i,5)
                Arrayab(i1,i1) = Arrayab(i1,i1) + (1.0/16.0)
                Arrayab(i1,i2) = Arrayab(i1,i2) + (1.0/48.0)
                Arrayab(i1,i3) = Arrayab(i1,i3) + (1.0/48.0)
                Arrayab(i1,i4) = Arrayab(i1,i4) + (1.0/144.0)
                Arrayab(i2,i1) = Arrayab(i2,i1) + (1.0/48.0)
                Arrayab(i2,i2) = Arrayab(i2,i2) + (1.0/16.0)
                Arrayab(i2,i3) = Arrayab(i2,i3) + (1.0/144.0)
                Arrayab(i2,i4) = Arrayab(i2,i4) + (1.0/48.0)
                Arrayab(i3,i1) = Arrayab(i3,i1) + (1.0/48.0)
                Arrayab(i3,i2) = Arrayab(i3,i2) + (1.0/144.0)
                Arrayab(i3,i3) = Arrayab(i3,i3) + (1.0/16.0)
                Arrayab(i3,i4) = Arrayab(i3,i4) + (1.0/48.0)
                Arrayab(i4,i1) = Arrayab(i4,i1) + (1.0/144.0)
                Arrayab(i4,i2) = Arrayab(i4,i2) + (1.0/48.0)
                Arrayab(i4,i3) = Arrayab(i4,i3) + (1.0/48.0)
                Arrayab(i4,i4) = Arrayab(i4,i4) + (1.0/16.0)
        end do

        Arrayab = detje * Arrayab

        arrayab = arrayab * rtcnst

        open (1, file='AB.dat')
                do i = 1, ivertextotal, 1
                        write (1,*) (arrayAB(i,j),j = 1,ivertextotal)
                end do

!! Now we are going to produce a vector that contains the top and right side edges
!!(boundary vertices numbers)
```

```
!! The first Nz number of points are the numbers of the right side vertices (w/o top right
vertex)
!!- counting upwards!!



The second Nr number of points are the numbers of the top side vertices (w/o top right
vertex)
!!- counting leftwards
!! The last number is the number of the vertex that is in the top right of the area
        do i = 1, Nz, 1
                iarrayboundary(i) = iarrayelemvert(Nr*i,3)
        end do

        icounter = Nz + 1
        do i = Nr*Nz, (Nr*Nz)-Nr+1, -1
                iarrayboundary(icounter) = iarrayelemvert(i,4)
                icounter = icounter + 1
        end do

        iarrayboundary(Nr+Nz+1) = iarrayelemvert(Nr*Nz,5)

 !! This part of the program contains the initial values of all of the constants (A-C)
        do i = 1, ivertextotal, 1
                arrayinitialvalues(i) = Ai
        end do

        do i = 1, Nr+Nz+1, 1
                idummy = iarrayboundary(i)
                idummy = idummy + ivertextotal
                arrayinitialvalues(idummy) = Bi
        end do

        open (1, file='initval.dat')
                do i = 1, 3*ivertextotal, 1
                        write (1,*) arrayinitialvalues(i)
                end do

 !! This is the main body of the program, which contains the Newton Method solver
nested within the time loop
        do i = 1, itimestep, 1
                !! This is where the new guesses are initialized
                arraynewguess = arrayinitialvalues
```

```
!! This is where the iterations loop starts (1000 is the maximum number of
iterations)
do j = 1, 1000, 1

!!This is where the f function and the jacobian are initialized

do k = 1, 3*ivertextotal, 1
    arrayiterativefunction(k) = 0.0
end do

do k = 1, 3*ivertextotal, 1
    do l = 1, 3*ivertextotal, 1
        arrayjacobian(k,l) = 0.0
    end do
end do



!! This is where the 'F' function is calculated to be used in the
Newton Method
!! This first section is the first third of it - the 'A' part

do k = 1, ivertextotal, 1
    do l = 1, ivertextotal, 1
        arrayiterativefunction(k) =
arrayiterativefunction(k)&
        & + arraynewguess(l)*arraytimeAC(k,l)/deltatime
    end do
end do

do k = 1, ivertextotal, 1
    do l = 1, ivertextotal, 1
        arrayiterativefunction(k) =
arrayiterativefunction(k)&
        & + arrayinitialvalues(l)*arraytimeAC(k,l)/(-
1.0*deltatime)
    end do
end do

do k = 1, ivertextotal, 1
    do l = 1, ivertextotal, 1
        arrayiterativefunction(k) =
arrayiterativefunction(k)&
        & +
arraynewguess(l)*arraynewguess(l+ivertextotal)&
        &*arraytimeAC(k,l)/4.0*rtcnst
```

```
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        arrayiterativefunction(k) =
arrayiterativefunction(k)&
                        & +
arrayinitialvalues(l)*arraynewguess(l+ivertextotal)&
                        &*arraytimeAC(k,l)/4.0*rtcnst
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        arrayiterativefunction(k) =
arrayiterativefunction(k)&
                        & +
arraynewguess(l)*arrayinitialvalues(l+ivertextotal)&
                        &*arraytimeAC(k,l)/4.0*rtcnst
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        lnew = l + ivertextotal
                        arrayiterativefunction(k) =
arrayiterativefunction(k)&
                        & + arrayinitialvalues(l)*arrayinitialvalues(lnew)&
                        &*arraytimeAC(k,l)/4.0*rtcnst
                    end do
                end do

                !! This second section is the second third of it - the 'B' part
                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        knew = ivertextotal + k
                        lnew = ivertextotal + l
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arraynewguess(l +
ivertextotal)*arraytimeB(k,l)/deltatime
                    end do
                end do

                do k = 1, ivertextotal, 1
```

```
                do l = 1, ivertextotal, 1
                        knew = ivertextotal + k
                        lnew = ivertextotal + l
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arrayinitialvalues(l + ivertextotal)&
                        &*arraytimeB(k,l)/deltatime*(-1.0)
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arrayB(k,l)/(2.0)*(arraynewguess(l +
ivertextotal)&
                        & + arrayinitialvalues(l + ivertextotal))
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arraynewguess(l)*arraynewguess(l +
ivertextotal)&
                        &*arrayab(k,l)/4.0
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arrayinitialvalues(l)*arraynewguess(l +
ivertextotal)&
                        &*arrayab(k,l)/4.0
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arraynewguess(l)*arrayinitialvalues(l +
ivertextotal)&
```

```
                        &*arrayab(k,l)/4.0
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        arrayiterativefunction(k + ivertextotal)&
                        & = arrayiterativefunction(k + ivertextotal)&
                        & + arrayinitialvalues(l)*arrayinitialvalues(l +
ivertextotal)&
                        &*arrayab(k,l)/4.0
                    end do
                end do

                !! This third section is the third third of it - the 'C' part
                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & +
arraynewguess(newl2)*arraytimeAC(k,l)/deltatime
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & + arrayinitialvalues(newl2)&
                        &*arraytimeAC(k,l)/deltatime*(-1.0)
                    end do
                end do

                do k = 1, ivertextotal, 1
                    do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
```

```
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & + arraynewguess(l)*arraynewguess(newl1)&
                        &*arraynegtimeAC(k,l)/4.0*rtcnst
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & + arrayinitialvalues(l)*arraynewguess(newl1)&
                        &*arraynegtimeac(k,l)/4.0*rtcnst
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & + arraynewguess(l)*arrayinitialvalues(newl1)&
                        &*arraynegtimeac(k,l)/4.0*rtcnst
                end do
        end do

        do k = 1, ivertextotal, 1
                do l = 1, ivertextotal, 1
                        newk1 = ivertextotal + k
                        newl1 = ivertextotal + l
                        newk2 = (2 * ivertextotal) + k
                        newl2 = (2 * ivertextotal) + l
                        arrayiterativefunction(newk2)&
                        & = arrayiterativefunction(newk2)&
                        & +
arrayinitialvalues(l)*arrayinitialvalues(newl1)&
                        &*arraynegtimeac(k,l)/4.0*rtcnst
                end do
```

```
                    end do

 !! This is where the boundary conditions are inserted into the B part of the
Arrayinterativefunction vector
                    do k = 1, Nr+Nz+1, 1
                        idumbs = iarrayboundary(k) + ivertextotal
                        arrayiterativefunction(idumbs) = arraynewguess(idumbs) -
Bi
                    end do


                    Arrayiterativefunction = -1.0 * arrayiterativefunction

                    !! Now we have to compute the Jacobian of the transformation of
C to F
                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            ArrayJacobian(k,l) =
(ArrayTimeAC(k,l)/deltatime)&
                            & +
((arraytimeac(k,l))/4.0*rtcnst*(arraynewguess(newl)))&
                            & +
((arraytimeac(k,l))/4.0*rtcnst*(arrayinitialvalues(newl)))
                        end do
                    end do

                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            ArrayJacobian(k,newl) =
(ArrayTimeAC(k,l)/4.0*rtcnst*arraynewguess(l))&
                            & +
((arraytimeac(k,l))/4.0*rtcnst*(arrayinitialvalues(l)))
                        end do
                    end do

                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            newk = k + ivertextotal
                            ArrayJacobian(newk,l) =
(Arrayab(k,l)/4.0*rtcnst*arraynewguess(newl))&
                            & + (arrayab(k,l)/4.0*arrayinitialvalues(newl))
                        end do
                    end do
```

```
                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            newk = k + ivertextotal
                            ArrayJacobian(newk,newl) =
(ArrayTimeB(k,l)/deltatime)&
                            & + (arrayb(k,l)/2.0) +
((arrayab(k,l))/4.0*(arraynewguess(l)))&
                            & + ((arrayab(k,l))/4.0*(arrayinitialvalues(l)))
                        end do
                    end do

                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            newk = k + ivertextotal
                            newl2 = l + (2*ivertextotal)
                            newk2 = k + (2*ivertextotal)
                            ArrayJacobian(newk2,l) =
(ArraynegTimeAC(k,l)/4.0*rtcnst*arraynewguess(newl))&
                            & +
(arraynegtimeac(k,l)/4.0*rtcnst*arrayinitialvalues(newl))
                        end do
                    end do

                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            newk = k + ivertextotal
                            newl2 = l + (2*ivertextotal)
                            newk2 = k + (2*ivertextotal)
                            ArrayJacobian(newk2,newl) =
(ArraynegTimeAC(k,l)/4.0*rtcnst*arraynewguess(l))&
                            & +
(arraynegtimeac(k,l)/4.0*rtcnst*arrayinitialvalues(l))
                        end do
                    end do

                    do k = 1, ivertextotal, 1
                        do l = 1, ivertextotal, 1
                            newl = l + ivertextotal
                            newk = k + ivertextotal
                            newl2 = l + (2*ivertextotal)
                            newk2 = k + (2*ivertextotal)
```

```
                                        ArrayJacobian(newk2,newl2) =
(ArrayTimeAC(k,l)/deltatime)
                                end do
                        end do

 !! This is where the boundary conditions are imposed on the Jacobian
                        do k = 1, Nr+Nz+1, 1
                                do l = 1, ivertextotal, 1
                                        if (l/=iarrayboundary(k)) then

        arrayjacobian(ivertextotal+iarrayboundary(k),l) = 0.0

        arrayjacobian(ivertextotal+iarrayboundary(k),l+ivertextotal) = 0.0
                                        else

        arrayjacobian(ivertextotal+iarrayboundary(k),l) = 0.0

        arrayjacobian(ivertextotal+iarrayboundary(k),l+ivertextotal) = 1.0
                                        end if
                                end do
                        end do


                        i3vertextotal = 3 * ivertextotal

                        open (1, file='F3.dat')
                                do ide = 1, i3vertextotal, 1
                                        write (1,*) arrayiterativefunction(ide)
                                end do

                        open (1, file='jacob.dat')
                                do ide = 1, i3vertextotal, 1
                                        write (1,*) (arrayjacobian(ide,jfe),jfe =
1,i3vertextotal)
                                end do

                        call
DSOLVE(ArrayJacobian,i3vertextotal,i3vertextotal,Arrayiterativefunction,ipvt)

                        do k = 1, i3vertextotal, 1
                                Arraynewguess1(k) = Arrayiterativefunction(k) +
arraynewguess(k)
                        end do
```

```
                    !! This is where the tolerance step is carried out (error has to be
less than 0.0001)

                    ik = 0
                    do k = 1, i3vertextotal, 1
                        if (arraynewguess1(k)-arraynewguess(k) < 0.0001) then
                            ik = ik + 1
                        end if
                    end do

                    if (ik==i3vertextotal) then
                        go to 10
                    end if

                    arraynewguess = arraynewguess1

                end do

            10      arrayinitialvalues = arraynewguess1

            print *,'The ',i,' timestep is finished'

            end do



            do i = 1, Nz+1, 1
                do j = 1, Nr+1, 1
                    arrayplotter(i,j) = arrayinitialvalues(j+((Nr+1)*(i-1)))
                end do
            end do

            open (1, file='plotterA.dat')
            do i = 1, Nz+1, 1
                write (1,*) (arrayplotter(i,j),j=1, Nr+1)
            end do



            do i = 1, Nz+1, 1
                do j = 1, Nr+1, 1
                    arrayplotter(i,j) = arrayinitialvalues(ivertextotal+j+((Nr+1)*(i-1)))
                end do
            end do

            open (1, file='plotterB.dat')
```

```
        do i = 1, Nz+1, 1
                write (1,*) (arrayplotter(i,j),j=1, Nr+1)
        end do
!!!  This is the numerical integration routine for species B
        totalint = 0.0
        do i = 1, Nr, 1
                vol = 3.141592654*halfht*(((i*radius/Nr)**2)-(((i-1)*radius/Nr)**2))/Nz

                do j = 1, Nz, 1
                        arrayplotter(j,i) = (arrayplotter(j,i) + arrayplotter(j+1,i)&
                        & + arrayplotter(j,i+1) + arrayplotter(j+1,i+1))/4*vol
                        totalint = totalint + arrayplotter(j,i)
                end do
        end do

        totalint = totalint/(3.141592654*(radius**2)*halfht)

        print*,'the integral of the b species is ',totalint
!!!  This ends the numerical integration routine for species B


        do i = 1, Nz+1, 1
                do j = 1, Nr+1, 1
                        arrayplotter(i,j) = arrayinitialvalues((2*ivertextotal)+j+((Nr+1)*(i-
1)))
                end do
        end do

        open (1, file='plotterC.dat')
        do i = 1, Nz+1, 1
                write (1,*) (arrayplotter(i,j),j=1, Nr+1)
        end do

contains
        function AF(x,y) result (A_result)
        real (kind=8) :: A_result, x, y
        A_result = -1.0+(x)+(2.0*y)+(-2.0*x*y)+(-1.0*(y**2.0))+(x*(y**2.0))
        end function AF

        function BF(x,y) result (B_result)
        real (kind=8) :: B_result, x, y
        B_result = (y)-(x*y)-(y**2.0)+(x*(y**2.0))
        end function BF

        function CF(x,y) result (C_result)
        real (kind=8) :: C_result, x, y
```

```
        C_result = (-1.0*x)+(2.0*x*y)-(x*(y**2.0))
        end function CF

        function DF(x,y) result (D_result)
        real (kind=8) :: D_result, x, y
        D_result = (x*y)-(x*(y**2.0))
        end function DF

        function EF(x,y) result (E_result)
        real (kind=8) :: E_result, x, y
        E_result = (x*(y**2.0))
        end function EF

        function FF(x,y) result (F_result)
        real (kind=8) :: F_result, x, y
        F_result = (y**2.0)-(x*(y**2.0))
        end function FF

end program difrxncy


    SUBROUTINE DSOLVE(A,LDA,N,F,IPVT)
!**********************************************************************
!     Solve full linear system using LINPACK routines
!     with compressed row sparse storage
!**********************************************************************
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION A(lda,*),ipvt(*),F(*)
    DATA JOB/0/
!
!*****perform factorization
!
    call DGEFA(A,LDA,N,IPVT,INF)
!
!*****Perform the solve step
!
    call DGESL(A,lda,N,IPVT,F,JOB)
    RETURN
    END
!
    SUBROUTINE DGEFA(A,LDA,N,IPVT,INFO)
    INTEGER LDA,N,IPVT(1),INFO
    DOUBLE PRECISION A(LDA,1)
!
!    DGEFA FACTORS A DOUBLE PRECISION MATRIX BY GAUSSIAN
ELIMINATION.
```

```
!
!    DGEFA IS USUALLY CALLED BY DGECO, BUT IT CAN BE CALLED
!    DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
!    (TIME FOR DGECO) = (1 + 9/N)*(TIME FOR DGEFA) .
!
!    ON ENTRY
!
!       A      DOUBLE PRECISION(LDA, N)
!              THE MATRIX TO BE FACTORED.
!
!       LDA    INTEGER
!              THE LEADING DIMENSION OF THE ARRAY  A .
!
!       N      INTEGER
!              THE ORDER OF THE MATRIX  A .
!
!    ON RETURN
!
!       A      AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
!              WHICH WERE USED TO OBTAIN IT.
!              THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
!              L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
!              TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
!
!       IPVT   INTEGER(N)
!              AN INTEGER VECTOR OF PIVOT INDICES.
!
!       INFO   INTEGER
!              = 0  NORMAL VALUE.
!              = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
!                   CONDITION FOR THIS SUBROUTINE, BUT IT DOES
!                   INDICATE THAT DGESL OR DGEDI WILL DIVIDE BY ZERO
!                   IF CALLED.  USE  RCOND  IN DGECO FOR A RELIABLE
!                   INDICATION OF SINGULARITY.
!
!    LINPACK. THIS VERSION DATED 08/14/78 .
!    CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL
LAB.
!
!    SUBROUTINES AND FUNCTIONS
!
!    BLAS DAXPY,DSCAL,IDAMAX
!
!    INTERNAL VARIABLES
!
     DOUBLE PRECISION T
```

```
      INTEGER IDAMAX,J,K,KP1,L,NM1
!
!
!     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
!
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
        KP1 = K + 1
!
!       FIND L = PIVOT INDEX
!
        L = IDAMAX(N-K+1,A(K,K),1) + K - 1
        IPVT(K) = L
!
!       ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
!
        IF (A(L,K) .EQ. 0.0D0) GO TO 40
!
!         INTERCHANGE IF NECESSARY
!
          IF (L .EQ. K) GO TO 10
            T = A(L,K)
            A(L,K) = A(K,K)
            A(K,K) = T
   10     CONTINUE
!
!         COMPUTE MULTIPLIERS
!
          T = -1.0D0/A(K,K)
          CALL DSCAL(N-K,T,A(K+1,K),1)
!
!         ROW ELIMINATION WITH COLUMN INDEXING
!
          DO 30 J = KP1, N
            T = A(L,J)
            IF (L .EQ. K) GO TO 20
              A(L,J) = A(K,J)
              A(K,J) = T
   20       CONTINUE
            CALL DAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30     CONTINUE
        GO TO 50
   40   CONTINUE
          INFO = K
```

```
  50    CONTINUE
  60 CONTINUE
  70 CONTINUE
     IPVT(N) = N
     IF (A(N,N) .EQ. 0.0D0) INFO = N
     RETURN
     END
     SUBROUTINE DGESL(A,LDA,N,IPVT,B,JOB)
     INTEGER LDA,N,IPVT(1),JOB
     DOUBLE PRECISION A(LDA,1),B(1)
!
!    DGESL SOLVES THE DOUBLE PRECISION SYSTEM
!    A * X = B  OR  TRANS(A) * X = B
!    USING THE FACTORS COMPUTED BY DGECO OR DGEFA.
!
!    ON ENTRY
!
!       A       DOUBLE PRECISION(LDA, N)
!               THE OUTPUT FROM DGECO OR DGEFA.
!
!       LDA     INTEGER
!               THE LEADING DIMENSION OF THE ARRAY  A .
!
!       N       INTEGER
!               THE ORDER OF THE MATRIX  A .
!
!       IPVT    INTEGER(N)
!               THE PIVOT VECTOR FROM DGECO OR DGEFA.
!
!       B       DOUBLE PRECISION(N)
!               THE RIGHT HAND SIDE VECTOR.
!
!       JOB     INTEGER
!               = 0         TO SOLVE  A*X = B ,
!               = NONZERO   TO SOLVE  TRANS(A)*X = B  WHERE
!                           TRANS(A)  IS THE TRANSPOSE.
!
!    ON RETURN
!
!       B       THE SOLUTION VECTOR  X .
!
!    ERROR CONDITION
!
!       A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
!       ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES
SINGULARITY
```

```
!       BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
!       SETTING OF LDA .  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
!       CALLED CORRECTLY AND IF DGECO HAS SET RCOND .GT. 0.0
!       OR DGEFA HAS SET INFO .EQ. 0 .
!
!     TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A MATRIX
!     WITH  P  COLUMNS
!          CALL DGECO(A,LDA,N,IPVT,RCOND,Z)
!          IF (RCOND IS TOO SMALL) GO TO ...
!          DO 10 J = 1, P
!            CALL DGESL(A,LDA,N,IPVT,C(1,J),0)
!       10 CONTINUE
!
!     LINPACK. THIS VERSION DATED 08/14/78 .
!     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL
LAB.
!
!     SUBROUTINES AND FUNCTIONS
!
!     BLAS DAXPY,DDOT
!
!     INTERNAL VARIABLES
!
      DOUBLE PRECISION DDOT,T
      INTEGER K,KB,L,NM1
!
      NM1 = N - 1
      IF (JOB .NE. 0) GO TO 50
!
!       JOB = 0 , SOLVE  A * X = B
!       FIRST SOLVE  L*Y = B
!
        IF (NM1 .LT. 1) GO TO 30
        DO 20 K = 1, NM1
          L = IPVT(K)
          T = B(L)
          IF (L .EQ. K) GO TO 10
            B(L) = B(K)
            B(K) = T
   10     CONTINUE
          CALL DAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20   CONTINUE
   30   CONTINUE
!
!       NOW SOLVE  U*X = Y
!
```

```
      DO 40 KB = 1, N
        K = N + 1 - KB
        B(K) = B(K)/A(K,K)
        T = -B(K)
        CALL DAXPY(K-1,T,A(1,K),1,B(1),1)
 40   CONTINUE
      GO TO 100
 50 CONTINUE
!
!     JOB = NONZERO, SOLVE  TRANS(A) * X = B
!     FIRST SOLVE  TRANS(U)*Y = B
!
      DO 60 K = 1, N
        T = DDOT(K-1,A(1,K),1,B(1),1)
        B(K) = (B(K) - T)/A(K,K)
 60   CONTINUE
!
!     NOW SOLVE TRANS(L)*X = Y
!
      IF (NM1 .LT. 1) GO TO 90
      DO 80 KB = 1, NM1
        K = N - KB
        B(K) = B(K) + DDOT(N-K,A(K+1,K),1,B(K+1),1)
        L = IPVT(K)
        IF (L .EQ. K) GO TO 70
          T = B(L)
          B(L) = B(K)
          B(K) = T
 70     CONTINUE
 80   CONTINUE
 90   CONTINUE
100 CONTINUE
    RETURN
    END
    SUBROUTINE DGBFA(ABD,LDA,N,ML,MU,IPVT,INFO)
    INTEGER LDA,N,ML,MU,IPVT(1),INFO
    DOUBLE PRECISION ABD(LDA,1)
!
!    DGBFA FACTORS A DOUBLE PRECISION BAND MATRIX BY
ELIMINATION.
!
!    DGBFA IS USUALLY CALLED BY DGBCO, BUT IT CAN BE CALLED
!    DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
!
!    ON ENTRY
!
```

```
!     ABD     DOUBLE PRECISION(LDA, N)
!             CONTAINS THE MATRIX IN BAND STORAGE.  THE COLUMNS
!             OF THE MATRIX ARE STORED IN THE COLUMNS OF  ABD  AND
!             THE DIAGONALS OF THE MATRIX ARE STORED IN ROWS
!             ML+1 THROUGH 2*ML+MU+1 OF  ABD .
!             SEE THE COMMENTS BELOW FOR DETAILS.
!
!     LDA     INTEGER
!             THE LEADING DIMENSION OF THE ARRAY  ABD .
!             LDA MUST BE .GE. 2*ML + MU + 1 .
!
!     N       INTEGER
!             THE ORDER OF THE ORIGINAL MATRIX.
!
!     ML      INTEGER
!             NUMBER OF DIAGONALS BELOW THE MAIN DIAGONAL.
!             0 .LE. ML .LT. N .
!
!     MU      INTEGER
!             NUMBER OF DIAGONALS ABOVE THE MAIN DIAGONAL.
!             0 .LE. MU .LT. N .
!             MORE EFFICIENT IF  ML .LE. MU .
!  ON RETURN
!
!     ABD     AN UPPER TRIANGULAR MATRIX IN BAND STORAGE AND
!             THE MULTIPLIERS WHICH WERE USED TO OBTAIN IT.
!             THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
!             L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
!             TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
!
!     IPVT    INTEGER(N)
!             AN INTEGER VECTOR OF PIVOT INDICES.
!
!     INFO    INTEGER
!             = 0  NORMAL VALUE.
!             = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
!                  CONDITION FOR THIS SUBROUTINE, BUT IT DOES
!                  INDICATE THAT DGBSL WILL DIVIDE BY ZERO IF
!                  CALLED.  USE  RCOND  IN DGBCO FOR A RELIABLE
!                  INDICATION OF SINGULARITY.
!
!  BAND STORAGE
!
!       IF  A  IS A BAND MATRIX, THE FOLLOWING PROGRAM SEGMENT
!       WILL SET UP THE INPUT.
!
```

```
!               ML = (BAND WIDTH BELOW THE DIAGONAL)
!               MU = (BAND WIDTH ABOVE THE DIAGONAL)
!               M = ML + MU + 1
!               DO 20 J = 1, N
!                 I1 = MAX0(1, J-MU)
!                 I2 = MIN0(N, J+ML)
!                 DO 10 I = I1, I2
!                   K = I - J + M
!                   ABD(K,J) = A(I,J)
!            10   CONTINUE
!            20 CONTINUE
!
!         THIS USES ROWS  ML+1  THROUGH  2*ML+MU+1  OF  ABD .
!         IN ADDITION, THE FIRST  ML  ROWS IN  ABD  ARE USED FOR
!         ELEMENTS GENERATED DURING THE TRIANGULARIZATION.
!         THE TOTAL NUMBER OF ROWS NEEDED IN  ABD  IS  2*ML+MU+1 .
!         THE  ML+MU BY ML+MU  UPPER LEFT TRIANGLE AND THE
!         ML BY ML  LOWER RIGHT TRIANGLE ARE NOT REFERENCED.
!
!    LINPACK. THIS VERSION DATED 08/14/78 .
!    CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL
LAB.
!
!    SUBROUTINES AND FUNCTIONS
!
!    BLAS DAXPY,DSCAL,IDAMAX
!    FORTRAN MAX0,MIN0
!
!    INTERNAL VARIABLES
!
     DOUBLE PRECISION T
     INTEGER I,IDAMAX,I0,J,JU,JZ,J0,J1,K,KP1,L,LM,M,MM,NM1
!
!
     M = ML + MU + 1
     INFO = 0
!
!    ZERO INITIAL FILL-IN COLUMNS
!
     J0 = MU + 2
     J1 = MIN0(N,M) - 1
     IF (J1 .LT. J0) GO TO 30
     DO 20 JZ = J0, J1
       I0 = M + 1 - JZ
       DO 10 I = I0, ML
         ABD(I,JZ) = 0.0D0
```

```
  10    CONTINUE
  20 CONTINUE
  30 CONTINUE
     JZ = J1
     JU = 0
!
!    GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
!
     NM1 = N - 1
     IF (NM1 .LT. 1) GO TO 130
     DO 120 K = 1, NM1
       KP1 = K + 1
!
!       ZERO NEXT FILL-IN COLUMN
!
       JZ = JZ + 1
       IF (JZ .GT. N) GO TO 50
       IF (ML .LT. 1) GO TO 50
         DO 40 I = 1, ML
           ABD(I,JZ) = 0.0D0
  40     CONTINUE
  50   CONTINUE
!
!      FIND L = PIVOT INDEX
!
       LM = MIN0(ML,N-K)
       L = IDAMAX(LM+1,ABD(M,K),1) + M - 1
       IPVT(K) = L + K - M
!
!      ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
!
       IF (ABD(L,K) .EQ. 0.0D0) GO TO 100
!
!        INTERCHANGE IF NECESSARY
!
         IF (L .EQ. M) GO TO 60
           T = ABD(L,K)
           ABD(L,K) = ABD(M,K)
           ABD(M,K) = T
  60     CONTINUE
!
!        COMPUTE MULTIPLIERS
!
         T = -1.0D0/ABD(M,K)
         CALL DSCAL(LM,T,ABD(M+1,K),1)
!
```

```
!         ROW ELIMINATION WITH COLUMN INDEXING
!
          JU = MIN0(MAX0(JU,MU+IPVT(K)),N)
          MM = M
          IF (JU .LT. KP1) GO TO 90
          DO 80 J = KP1, JU
            L = L - 1
            MM = MM - 1
            T = ABD(L,J)
            IF (L .EQ. MM) GO TO 70
              ABD(L,J) = ABD(MM,J)
              ABD(MM,J) = T
   70       CONTINUE
            CALL DAXPY(LM,T,ABD(M+1,K),1,ABD(MM+1,J),1)
   80     CONTINUE
   90     CONTINUE
        GO TO 110
  100   CONTINUE
          INFO = K
  110   CONTINUE
  120 CONTINUE
  130 CONTINUE
      IPVT(N) = N
      IF (ABD(M,N) .EQ. 0.0D0) INFO = N
      RETURN
      END
      SUBROUTINE DGBSL(ABD,LDA,N,ML,MU,IPVT,B,JOB)
      INTEGER LDA,N,ML,MU,IPVT(1),JOB
      DOUBLE PRECISION ABD(LDA,1),B(1)
!
!     DGBSL SOLVES THE DOUBLE PRECISION BAND SYSTEM
!     A * X = B  OR  TRANS(A) * X = B
!     USING THE FACTORS COMPUTED BY DGBCO OR DGBFA.
!
!     ON ENTRY
!
!        ABD     DOUBLE PRECISION(LDA, N)
!                THE OUTPUT FROM DGBCO OR DGBFA.
!
!        LDA     INTEGER
!                THE LEADING DIMENSION OF THE ARRAY  ABD .
!
!        N       INTEGER
!                THE ORDER OF THE ORIGINAL MATRIX.
!
!        ML      INTEGER
```

```
!            NUMBER OF DIAGONALS BELOW THE MAIN DIAGONAL.
!
!     MU     INTEGER
!            NUMBER OF DIAGONALS ABOVE THE MAIN DIAGONAL.
!
!     IPVT    INTEGER(N)
!            THE PIVOT VECTOR FROM DGBCO OR DGBFA.
!
!     B      DOUBLE PRECISION(N)
!            THE RIGHT HAND SIDE VECTOR.
!
!     JOB    INTEGER
!            = 0        TO SOLVE  A*X = B ,
!            = NONZERO   TO SOLVE  TRANS(A)*X = B , WHERE
!                    TRANS(A)  IS THE TRANSPOSE.
!
!   ON RETURN
!
!     B      THE SOLUTION VECTOR  X .
!
!   ERROR CONDITION
!
!     A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
!     ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES
SINGULARITY
!     BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
!     SETTING OF LDA .  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
!     CALLED CORRECTLY AND IF DGBCO HAS SET RCOND .GT. 0.0
!     OR DGBFA HAS SET INFO .EQ. 0 .
!
!   TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A MATRIX
!   WITH  P  COLUMNS
!        CALL DGBCO(ABD,LDA,N,ML,MU,IPVT,RCOND,Z)
!        IF (RCOND IS TOO SMALL) GO TO ...
!        DO 10 J = 1, P
!          CALL DGBSL(ABD,LDA,N,ML,MU,IPVT,C(1,J),0)
!     10 CONTINUE
!
!   LINPACK. THIS VERSION DATED 08/14/78 .
!   CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL
LAB.
!
!   SUBROUTINES AND FUNCTIONS
!
!   BLAS DAXPY,DDOT
!   FORTRAN MIN0
```

```
!
!    INTERNAL VARIABLES
!
     DOUBLE PRECISION DDOT,T
     INTEGER K,KB,L,LA,LB,LM,M,NM1
!
     M = MU + ML + 1
     NM1 = N - 1
     IF (JOB .NE. 0) GO TO 50
!
!       JOB = 0 , SOLVE  A * X = B
!       FIRST SOLVE L*Y = B
!
        IF (ML .EQ. 0) GO TO 30
        IF (NM1 .LT. 1) GO TO 30
          DO 20 K = 1, NM1
            LM = MIN0(ML,N-K)
            L = IPVT(K)
            T = B(L)
            IF (L .EQ. K) GO TO 10
              B(L) = B(K)
              B(K) = T
   10       CONTINUE
            CALL DAXPY(LM,T,ABD(M+1,K),1,B(K+1),1)
   20     CONTINUE
   30   CONTINUE
!
!       NOW SOLVE  U*X = Y
!
        DO 40 KB = 1, N
          K = N + 1 - KB
          B(K) = B(K)/ABD(M,K)
          LM = MIN0(K,M) - 1
          LA = M - LM
          LB = K - LM
          T = -B(K)
          CALL DAXPY(LM,T,ABD(LA,K),1,B(LB),1)
   40   CONTINUE
      GO TO 100
   50 CONTINUE
!
!       JOB = NONZERO, SOLVE  TRANS(A) * X = B
!       FIRST SOLVE  TRANS(U)*Y = B
!
        DO 60 K = 1, N
          LM = MIN0(K,M) - 1
```

```
         LA = M - LM
         LB = K - LM
         T = DDOT(LM,ABD(LA,K),1,B(LB),1)
         B(K) = (B(K) - T)/ABD(M,K)
  60   CONTINUE
!
!      NOW SOLVE TRANS(L)*X = Y
!
       IF (ML .EQ. 0) GO TO 90
       IF (NM1 .LT. 1) GO TO 90
         DO 80 KB = 1, NM1
           K = N - KB
           LM = MIN0(ML,N-K)
           B(K) = B(K) + DDOT(LM,ABD(M+1,K),1,B(K+1),1)
           L = IPVT(K)
           IF (L .EQ. K) GO TO 70
             T = B(L)
             B(L) = B(K)
             B(K) = T
  70       CONTINUE
  80     CONTINUE
  90   CONTINUE
 100 CONTINUE
     RETURN
     END
     SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
!
!    CONSTANT TIMES A VECTOR PLUS A VECTOR.
!    USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
!    JACK DONGARRA, LINPACK, 3/11/78.
!
     DOUBLE PRECISION DX(1),DY(1),DA
     INTEGER I,INCX,INCY,IX,IY,M,MP1,N
!
     IF(N.LE.0)RETURN
     IF (DA .EQ. 0.0D0) RETURN
     IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
!
!      CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
!        NOT EQUAL TO 1
!
     IX = 1
     IY = 1
     IF(INCX.LT.0)IX = (-N+1)*INCX + 1
     IF(INCY.LT.0)IY = (-N+1)*INCY + 1
     DO 10 I = 1,N
```

```
      DY(IY) = DY(IY) + DA*DX(IX)
      IX = IX + INCX
      IY = IY + INCY
   10 CONTINUE
      RETURN
!
!       CODE FOR BOTH INCREMENTS EQUAL TO 1
!
!
!       CLEAN-UP LOOP
!
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        DY(I) = DY(I) + DA*DX(I)
        DY(I + 1) = DY(I + 1) + DA*DX(I + 1)
        DY(I + 2) = DY(I + 2) + DA*DX(I + 2)
        DY(I + 3) = DY(I + 3) + DA*DX(I + 3)
   50 CONTINUE
      RETURN
      END
      SUBROUTINE DSCAL(N,DA,DX,INCX)
!
!     SCALES A VECTOR BY A CONSTANT.
!     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
!     JACK DONGARRA, LINPACK, 3/11/78.
!
      DOUBLE PRECISION DA,DX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
!
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
!
!       CODE FOR INCREMENT NOT EQUAL TO 1
!
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DX(I) = DA*DX(I)
   10 CONTINUE
      RETURN
!
```

```
!       CODE FOR INCREMENT EQUAL TO 1
!
!
!       CLEAN-UP LOOP
!
  20 M = MOD(N,5)
     IF( M .EQ. 0 ) GO TO 40
     DO 30 I = 1,M
       DX(I) = DA*DX(I)
  30 CONTINUE
     IF( N .LT. 5 ) RETURN
  40 MP1 = M + 1
     DO 50 I = MP1,N,5
       DX(I) = DA*DX(I)
       DX(I + 1) = DA*DX(I + 1)
       DX(I + 2) = DA*DX(I + 2)
       DX(I + 3) = DA*DX(I + 3)
       DX(I + 4) = DA*DX(I + 4)
  50 CONTINUE
     RETURN
     END
     DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
!
!    FORMS THE DOT PRODUCT OF TWO VECTORS.
!    USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
!    JACK DONGARRA, LINPACK, 3/11/78.
!
     DOUBLE PRECISION DX(1),DY(1),DTEMP
     INTEGER I,INCX,INCY,IX,IY,M,MP1,N
!
     DDOT = 0.0D0
     DTEMP = 0.0D0
     IF(N.LE.0)RETURN
     IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
!
!       CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
!         NOT EQUAL TO 1
!
     IX = 1
     IY = 1
     IF(INCX.LT.0)IX = (-N+1)*INCX + 1
     IF(INCY.LT.0)IY = (-N+1)*INCY + 1
     DO 10 I = 1,N
       DTEMP = DTEMP + DX(IX)*DY(IY)
       IX = IX + INCX
       IY = IY + INCY
```

```
  10 CONTINUE
     DDOT = DTEMP
     RETURN
!
!       CODE FOR BOTH INCREMENTS EQUAL TO 1
!
!
!       CLEAN-UP LOOP
!
  20 M = MOD(N,5)
     IF( M .EQ. 0 ) GO TO 40
     DO 30 I = 1,M
       DTEMP = DTEMP + DX(I)*DY(I)
  30 CONTINUE
     IF( N .LT. 5 ) GO TO 60
  40 MP1 = M + 1
     DO 50 I = MP1,N,5
       DTEMP = DTEMP + DX(I)*DY(I) + DX(I + 1)*DY(I + 1) +  DX(I + 2)*DY(I + 2)
+ DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
!    *   DX(I + 2)*DY(I + 2) + DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
  50 CONTINUE
  60 DDOT = DTEMP
     RETURN
     END
     INTEGER FUNCTION IDAMAX(N,DX,INCX)
!
!    FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
!    JACK DONGARRA, LINPACK, 3/11/78.
!
     DOUBLE PRECISION DX(1),DMAX
     INTEGER I,INCX,IX,N
!
     IDAMAX = 0
     IF( N .LT. 1 ) RETURN
     IDAMAX = 1
     IF(N.EQ.1)RETURN
     IF(INCX.EQ.1)GO TO 20
!
!       CODE FOR INCREMENT NOT EQUAL TO 1
!
     IX = 1
     DMAX = DABS(DX(1))
     IX = IX + INCX
     DO 10 I = 2,N
       IF(DABS(DX(IX)).LE.DMAX) GO TO 5
       IDAMAX = I
```

```
        DMAX = DABS(DX(IX))
    5   IX = IX + INCX
   10 CONTINUE
      RETURN
!
!       CODE FOR INCREMENT EQUAL TO 1
!
   20 DMAX = DABS(DX(1))
      DO 30 I = 2,N
        IF(DABS(DX(I)).LE.DMAX) GO TO 30
        IDAMAX = I
        DMAX = DABS(DX(I))
   30 CONTINUE
      RETURN
      END




      DOUBLE PRECISION FUNCTION D1MACH (IDUM)
      INTEGER IDUM
!-----------------------------------------------------------------------
! THIS ROUTINE COMPUTES THE UNIT ROUNDOFF OF THE MACHINE IN
DOUBLE
! PRECISION.  THIS IS DEFINED AS THE SMALLEST POSITIVE MACHINE
NUMBER
! U SUCH THAT  1.0E0 + U .NE. 1.0E0 (IN SINGLE PRECISION).
!-----------------------------------------------------------------------
      DOUBLE PRECISION U, COMP
      U = 1.0E0
 10   U = U*0.5E0
      COMP = 1.0E0 + U
      IF (COMP .NE. 1.0E0) GO TO 10
      D1MACH = U*2.0E0
      RETURN
!----------------------- END OF FUNCTION D1MACH ----------------------
      END
      SUBROUTINE XERRWV (MSG, NMES, NERR, IERT, NI, I1, I2, NR, R1, R2)
      INTEGER MSG, NMES, NERR, IERT, NI, I1, I2, NR, I, LUN, LUNIT, MESFLG,
NWDS
!    1   I, LUN, LUNIT, MESFLG, NWDS
      DOUBLE PRECISION R1, R2
```

```
      DIMENSION MSG(NMES)
!-----------------------------------------------------------------------
! SUBROUTINE XERRWV, AS GIVEN HERE, CONSTITUTES
! A SIMPLIFIED VERSION OF THE SLATEC ERROR HANDLING PACKAGE.
! WRITTEN BY A. C. HINDMARSH AT LLL.  VERSION OF JANUARY 23, 1980.
! MODIFIED BY L. R. PETZOLD, APRIL 1982.
! THIS VERSION IS IN SINGLE PRECISION.
!
! ALL ARGUMENTS ARE INPUT ARGUMENTS.
!
! MSG    = THE MESSAGE (HOLLERITH LITTERAL OR INTEGER ARRAY).
! NMES   = THE LENGTH OF MSG (NUMBER OF CHARACTERS).
! NERR   = THE ERROR NUMBER (NOT USED).
! IERT   = THE ERROR TYPE..
!         1 MEANS RECOVERABLE (CONTROL RETURNS TO CALLER).
!         2 MEANS FATAL (RUN IS ABORTED--SEE NOTE BELOW).
! NI     = NUMBER OF INTEGERS (0, 1, OR 2) TO BE PRINTED WITH MESSAGE.
! I1,I2  = INTEGERS TO BE PRINTED, DEPENDING ON NI.
! NR     = NUMBER OF REALS (0, 1, OR 2) TO BE PRINTED WITH MESSAGE.
! R1,R2  = REALS TO BE PRINTED, DEPENDING ON NI.
!
! NOTE..  THIS ROUTINE IS MACHINE-DEPENDENT AND SPECIALIZED FOR
USE
! IN LIMITED CONTEXT, IN THE FOLLOWING WAYS..
! 1. THE NUMBER OF HOLLERITH CHARACTERS STORED PER WORD,
DENOTED
!    BY NCPW BELOW, IS SET IN A DATA STATEMENT BELOW.
! 2. THE VALUE OF NMES IS ASSUMED TO BE AT MOST 60.
!    (MULTI-LINE MESSAGES ARE GENERATED BY REPEATED CALLS.)
! 3. IF IERT = 2, CONTROL PASSES TO THE STATEMENT   STOP
!    TO ABORT THE RUN.  THIS STATEMENT MAY BE MACHINE-DEPENDENT.
! 4. R1 AND R2 ARE ASSUMED TO BE IN REAL AND ARE PRINTED
!    IN D21.13 FORMAT.
! 5. THE DATA STATEMENT BELOW CONTAINS DEFAULT VALUES OF
!      MESFLG = PRINT CONTROL FLAG..
!               1 MEANS PRINT ALL MESSAGES (THE DEFAULT).
!               0 MEANS NO PRINTING.
!      LUNIT  = LOGICAL UNIT NUMBER FOR MESSAGES.
!               THE DEFAULT IS 3 (MACHINE-DEPENDENT).
!               TO CHANGE LUNIT, CHANGE THE DATA STATEMENT
!               BELOW.
!-----------------------------------------------------------------------
! THE FOLLOWING ARE INSTRUCTIONS FOR INSTALLING THIS ROUTINE
! IN DIFFERENT MACHINE ENVIRONMENTS.
!
```

```
! TO CHANGE THE DEFAULT OUTPUT UNIT, CHANGE THE DATA
STATEMENT
! BELOW.
!
! FOR A DIFFERENT NUMBER OF CHARACTERS PER WORD, CHANGE THE
! DATA STATEMENT SETTING NCPW BELOW.
! ALTERNATIVES FOR VARIOUS COMPUTERS ARE SHOWN IN COMMENT
! CARDS.
!
! FOR A DIFFERENT RUN-ABORT COMMAND, CHANGE THE STATEMENT
FOLLOWING
! STATEMENT 100 AT THE END.
!---------------------------------------------------------------------
! THE FOLLOWING VALUE OF NCPW IS VALID FOR THE CDC-6600 AND
! CDC-7600 COMPUTERS.
!     DATA NCPW/10/
! THE FOLLOWING IS VALID FOR THE CRAY-1 COMPUTER.
!     DATA NCPW/8/
! THE FOLLOWING IS VALID FOR THE BURROUGHS 6700 AND 7800
COMPUTERS.
!     DATA NCPW/6/
! THE FOLLOWING IS VALID FOR THE PDP-10 COMPUTER.
!     DATA NCPW/5/
! THE FOLLOWING IS VALID FOR THE VAX COMPUTER WITH 4 BYTES PER
INTEGER,
! AND FOR THE IBM-360, IBM-303X, AND IBM-43XX COMPUTERS.
!     DATA NCPW/4/
! THE FOLLOWING IS VALID FOR THE PDP-11, OR VAX WITH 2-BYTE
INTEGERS.
!     DATA NCPW/2/
!---------------------------------------------------------------------
      DIMENSION NFORM(13)
      DATA NFORM(1)/1H(/,NFORM(2)/1H1/,NFORM(3)/1HX/,NFORM(4)/1H,/,
NFORM(7)/1HA/,NFORM(10)/1H,/,NFORM(11)/1HA/,NFORM(13)/1H)/
 !    1  NFORM(7)/1HA/,NFORM(10)/1H,/,NFORM(11)/1HA/,NFORM(13)/1H)/
      DATA NCPW/4/
      DATA MESFLG/1/,LUNIT/3/
!
      IF (MESFLG .EQ. 0) GO TO 100
! GET LOGICAL UNIT NUMBER. ------------------------------------------
      LUN = LUNIT
! GET NUMBER OF WORDS IN MESSAGE. -----------------------------------
      NCH = MIN0(NMES,60)
      NWDS = NCH/NCPW
      CALL S88FMT(2,NWDS,NFORM(5))
      CALL S88FMT(2,NCPW,NFORM(8))
```

```
      NREM = NCH - NWDS*NCPW
      IF (NREM .GT. 0) NWDS = NWDS + 1
      IF (NREM .LT. 1) NREM = 1
      CALL S88FMT(1,NREM,NFORM(12))
      WRITE(LUN,NFORM) (MSG(I),I=1,NWDS)
      IF (NI .EQ. 1) WRITE (LUN, 20) I1
 20   FORMAT(6X,23HIN ABOVE MESSAGE,  I1 =,I10)
      IF (NI .EQ. 2) WRITE (LUN, 30) I1,I2
 30   FORMAT(6X,23HIN ABOVE MESSAGE,  I1 =,I10,3X,4HI2 =,I10)
      IF (NR .EQ. 1) WRITE (LUN, 40) R1
 40   FORMAT(6X,23HIN ABOVE MESSAGE,  R1 =,D21.13)
      IF (NR .EQ. 2) WRITE (LUN, 50) R1,R2
 50   FORMAT(6X,15HIN ABOVE,  R1 =,D21.13,3X,4HR2 =,D21.13)
! ABORT THE RUN IF IERT = 2. ------------------------------------------
 100  IF (IERT .NE. 2) RETURN
      STOP
!---------------------- END OF SUBROUTINE XERRWV ----------------------
      END
      SUBROUTINE S88FMT(N,IVALUE,IFMT)
!***BEGIN PROLOGUE  S88FMT
!***REFER TO  XERROR
!     ABSTRACT
!        S88FMT REPLACES IFMT(1), ... ,IFMT(N) WITH THE
!        CHARACTERS CORRESPONDING TO THE N LEAST SIGNIFICANT
!        DIGITS OF IVALUE.
!
!     TAKEN FROM THE BELL LABORATORIES PORT LIBRARY ERROR
HANDLER
!     LATEST REVISION ---  7 JUNE 1978
!
!***REFERENCES
!   JONES R.E., *SLATEC COMMON MATHEMATICAL LIBRARY ERROR
HANDLING
!    PACKAGE*, SAND78-1189, SANDIA LABORATORIES, 1978.
!***ROUTINES CALLED  (NONE)
!***END PROLOGUE  S88FMT
!
      DIMENSION IFMT(N),IDIGIT(10)
 !     DATA
IDIGIT(1),IDIGIT(2),IDIGIT(3),IDIGIT(4),IDIGIT(5),IDIGIT(6),IDIGIT(7),IDIGIT(8),
IDIGIT(9),IDIGIT(10)  /1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
 !    1     IDIGIT(6),IDIGIT(7),IDIGIT(8),IDIGIT(9),IDIGIT(10)
 !    2     /1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
!***FIRST EXECUTABLE STATEMENT  S88FMT
      NT = N
      IT = IVALUE
```

```
10    IF (NT .EQ. 0) RETURN
      INDEX = MOD(IT,10)
      IFMT(NT) = IDIGIT(INDEX+1)
      IT = IT/10
      NT = NT - 1
      GO TO 10
    END
```

APPENDIX B.

Aminolysis/Gas Chromatograph (GC) Sample Calculation for Fiber Content Determination of Composites.

The purpose of this appendix is to provide a sample calculation regarding the fiber content determinations using the aminolysis/GC procedure. It will start after the preparation of the reaction mixture. This mixture contains a known amount of hexanoyl pyrrolidine (internal standard) and an unknown amount of acetyl pyrrolidine (from the matrix and acetylated fiber, if present) and butyryl pyrrolidine (exclusively from the matrix). We want to determine the amount of butyryl pyrrolidine is in the reaction mixture, after which we will be able to determine the amount of CAB originally in the analyzed sample.

1. Assume that we have run a sample of 34.5 mg pure CAB and 92.1 mg of an unknown fiber content composite (unmodified fiber). We obtained peak areas (pA*s) for butyryl-pyrrolidine and hexanoyl-pyrrolidine for CAB (3906 and 3602) and the composite sample (3373 and 2658). The first step is to convert the peak areas into the mass of each component within the sample volume injected into the GC. This is done by using the GC response factor of each component (obtained prior to the experiment).

$$
\begin{aligned}
m_{B,CAB} &= p_{area,CAB}\frac{mg}{10^6\ pA*s} = 0.003906mg \\
m_{B,composite} &= 0.003373mg \\
m_{H,CAB} &= p_{area,CAB}\frac{mg}{10^6\ pA*s} = 0.003602mg \\
m_{H,composite} &= 0.002658mg
\end{aligned}
\qquad \text{(B.0)}
$$

2. Once we have the mass of hexanoyl pyrrolidine in the injected sample, we can linearly scale this amount to the total amount of internal standard within the reaction mixture; once we have the scaling factor, we can then also scale up the butyryl pyrrolidine mass to reflect the total content of the reaction mixture.

$$
\begin{aligned}
x_{scale,CAB} &= \frac{m_{H,reaction,CAB}}{m_{H,CAB}} = \frac{20.58mg}{0.003602mg} = 5713 \\
x_{scale,composite} &= 7743 \\
m_{B,reaction,CAB} &= x_{scale,CAB}m_{B,CAB} = (5713)(0.003906mg) = 22.3mg \\
m_{B,reaction,composite} &= 26.1mg
\end{aligned}
\qquad \text{(B.1)}
$$

3. The butyryl pyrrolidine mass in the reaction mixture for CAB should be converted to the mass of butyryl ester groups originally on the cellulose backbone. This is done using the ratio of molecular weights.

$$m_{B,ester,CAB} = m_{B,reaction,CAB} \frac{\left(87\frac{g}{mol}\right)}{\left(87\frac{g}{mol} + 70.12\frac{g}{mol}\right)} = 12.4mg \tag{B.2}$$

4. With the amount of butyryl ester known, we should double-check the weight fraction of butyryl ester with the manufacturer's stated average value.

$$w_{butytyl,ester,CAB} = \frac{m_{B,ester,CAB}}{m_{sample,CAB}} = 0.36 \tag{B.3}$$

$$0.36 \approx 0.38_{manufacturer}$$

5. Now we can create a conversion factor that relates the mass of butyryl pyrrolidine in the reaction mixture to the mass of CAB in the reaction mixture using the numbers obtained from the neat CAB sample.

$$y_{conversion} = \frac{m_{sample,CAB}}{m_{B,reaction,CAB}} = 1.55 \tag{B.4}$$

6. This conversion factor can be multiplied by the mass of butyryl pyrrolidine in the unknown composite sample to directly obtain the mass of CAB within the unknown composite sample.

$$m_{CAB,composite} = m_{B,reaction,composite} y_{conversion} = 40.4mg \tag{B.5}$$

7. Using the densities of neat CAB and fiber, we can then calculate the volume fraction of fiber in the composite.

$$v_{fiber} = \frac{\left[\left(m_{sample,composite} - m_{CAB,composite}\right) \Big/ \rho_{fiber}\right]}{\left[\left(m_{sample,composite} - m_{CAB,composite}\right) \Big/ \rho_{fiber}\right] + \left[m_{CAB,composite} \Big/ \rho_{CAB}\right]} = 0.51 \tag{B.6}$$

APPENDIX C.

Raw Data for Chapter 3.
Experimental: Biobased Composites from Modified Fibers - Cellulose Acetate Butyrate with Acetylated Lyocell.

Table C.0 Density Values for the Composites and Neat Constituents.

Sample	Density (g/ml)
Lyocell	1.32[1]
CAB 381-20	1.20[1]
Unmodified Fiber Composite 1	1.09
Acetylated Fiber Composite 1	1.15

Table C.1 Tensile Properties of Unmodified and Modified Fiber Composites.

Sample	Modulus (GPa)	Max. Strength (MPa)	Max. Strain (%)
Unmodified Fiber Composite 1	12.1	175	2.70
Unmodified Fiber Composite 2	13.8	217	2.67
Unmodified Fiber Composite 3	14.1	206	2.10
Acetylated Fiber Composite 1	13.6	207	2.34
Acetylated Fiber Composite 2	17.9	227	1.72
Acetylated Fiber Composite 3	16.1	262	3.11
Acetylated Fiber Composite 4	13.1	199	2.20

[1] Manufacturer's stated value

Table C.2 Fiber Content of Composites (Voidless Basis).

Sample	Fiber Volume % (voidless basis)
Unmodified Fiber Composite 1	40.4
Unmodified Fiber Composite 2	43.5
Modified Fiber Composite 1	37.9
Modified Fiber Composite 2	38.6

Table C.3 Coefficient of Variations for Composite Tensile Properties.

Sample	Modulus COV	Max. Strength COV	Max. Strain COV
Unmodified Fiber Composite	8.09	10.9	13.6
Modified Fiber Composite	14.8	12.6	24.6

APPENDIX D.

Raw Data for Chapter 4.
Experimental: Optimizing the Consolidation Process - Cellulose Acetate Butyrate with Lyocell.

Table D.0 Density Values for the Composites.

Composite Panel Number	Density (g/ml)
1	1.16
2	1.19
3	0.95
4	1.13
5	0.99
6	1.23
7	1.13
8	1.19
9	1.21
10	1.21
11	1.23
12	1.15
13	1.13
14	1.12
15	1.17

Table D.1 Tensile Properties of Composites.

Composite Panel Number	Modulus (GPa)	Max. Stress (MPa)	Max. Strain (%)
2a	17.4	208	1.621
b	19.63	253	2.218
c	17.5	219	1.777
d	18.39	224	1.834
e	17.5	240	1.903
3a	11.14	124	1.857
b	10.1	106	1.308
c	8.44	119	3.747
d	10.1	115	1.45
e	14.4	137	1.213
9a	23.5	259	1.748

b	18.9	230	1.46
c	20.7	218	1.376
d	17.94	188	1.25
e	15.3	178	1.579
12a	25.5	192	1.013
b	14.3	138	0.8158
c	16.5	206	1.649
d	17.2	182	0.8766
e	16.2	166	1.241
f	21.1	218	1.323
13a	14.65	171	1.508
b	18.6	169	1.177
c	15.3	179	1.407
d	19.5	205	1.29
e	15.4	182	1.517
f	20.5	214	1.487
14a	18.9	153	0.9117
b	13.2	145	1.583
c	15.5	175	1.594
d	17.5	187	1.727
e	16.7	223	1.773
15a	16.3	188	1.42
b	18.8	163	1.033
c	20	183	1.177
d	19.9	138	0.7965
e	19.4	178	1.159
1a	19.3	196	1.44
b	12.38	212	2.2
c	20.7	229	2.41
d	17.5	197	2.38
e	21.2	214	2.08
4a	14.5	175	2.36
b	19.1	215	1.75
c	23.4	183	1.14
d	17.7	193	1.62
e	18.1	205	1.88

5a	17.5	234	2.33
b	17.2	218	2.11
c	19.2	229	1.95
d	16.6	211	1.85
e	16.5	211	1.6
6a	20.1	230	1.51
b	22.3	257	1.82
c	20.4	223	1.51
d	22.1	231	1.45
e	23	288	2.29
7a	26.3	237	1.68
b	14.3	175	1.94
c	19	225	1.68
d	16.9	190	1.77
e	15.5	189	1.98
8a	17.5	205	1.78
b	17.6	186	1.4
c	16.6	177	1.63
d	19.3	208	1.64
e	19.4	245	1.94
10a	19.8	185	1.29
b	21.6	230	1.69
c	19.5	202	1.45
d	20.2	206	1.64
e	21.1	234	1.52
11a	21.9	230	1.53
b	21.7	216	2.07
c	14	188	2.02
d	17.9	193	1.44
e	16.5	176	1.58

Table D.2 Coefficient of Variation for Tensile Properties.

Composite Panel Number	Modulus COV	Ultimate Stress COV	Ultimate Strain COV
1	20	6.5	19
2	5.3	7.8	12
3	20	9.5	55
4	17	8.3	25

5	6.3	4.8	14
6	5.9	11	21
7	26	13	7.9
8	6.8	13	12
9	16	15	13
10	4.3	9.7	10
11	19	11	17
12	22	16	27
13	14	9.9	9.9
14	13	17	23
15	8.0	12	20

Table D.3 ILSS for Composites.

Specimen ID	Thickness (in)	Width (in)	Max Load (kN)
1a	0.145	0.248	0.364
b	0.142	0.212	0.326
c	0.15	0.244	0.461
d	0.142	0.255	0.389
e	0.158	0.24	0.449
f	0.16	0.23	0.403
g	0.14	0.232	0.34
h	0.145	0.226	0.43
I	0.146	0.22	0.317
j	0.16	0.232	0.458
2a	0.152	0.236	0.489
b	0.152	0.241	0.428
c	0.151	0.236	0.427
d	0.156	0.236	0.255
e	0.152	0.225	0.418
f	0.16	0.205	0.213
g	0.154	0.224	0.37
h	0.171	0.255	0.441
I	0.16	0.22	0.292
j	0.149	0.231	0.25
3a	0.21	0.249	0.482
b	0.2	0.189	0.287
c	0.21	0.216	0.496
d	0.195	0.236	0.381
e	0.202	0.242	0.442
f	0.21	0.17	0.198
g	0.21	0.234	0.438

h	0.212	0.26	0.537
I	0.229	0.24	0.61
j	0.216	0.23	0.452
4a	0.152	0.268	0.41
b	0.17	0.252	0.243
c	0.161	0.222	0.439
d	0.162	0.241	0.362
e	0.154	0.23	0.505
f	0.166	0.21	0.322
g	0.158	0.23	0.452
h	0.16	0.26	0.271
I	0.159	0.182	0.331
j	0.162	0.238	0.442
5a	0.145	0.231	0.272
b	0.145	0.238	0.375
c	0.141	0.222	0.331
d	0.16	0.21	0.347
e	0.15	0.24	0.433
f	0.138	0.228	0.35
g	0.14	0.245	0.292
h	0.148	0.222	0.384
I	0.156	0.22	0.384
j	0.154	0.248	0.419
6a	0.139	0.236	0.439
b	0.158	0.226	0.44
c	0.14	0.242	0.455
d	0.14	0.246	0.478
e	0.139	0.241	0.501
f	0.149	0.245	0.452
g	0.152	0.219	0.438
h	0.142	0.244	0.452
I	0.146	0.24	0.486
j	0.149	0.204	0.392
7a	0.166	0.22	0.347
b	0.182	0.26	0.433
c	0.162	0.215	0.376
d	0.168	0.248	0.394
e	0.171	0.22	0.401
f	0.168	0.222	0.401
g	0.18	0.249	0.437

h	0.16	0.221	0.384
I	0.182	0.261	0.431
j	0.17	0.231	0.375
8a	0.15	0.238	0.438
b	0.15	0.23	0.391
c	0.16	0.23	0.366
d	0.14	0.252	0.436
e	0.141	0.24	0.47
f	0.158	0.255	0.441
g	0.152	0.23	0.434
h	0.14	0.231	0.419
I	0.146	0.242	0.447
j	0.135	0.245	0.42
9a	0.14	0.224	0.398
b	0.146	0.248	0.421
c	0.149	0.236	0.433
d	0.145	0.232	0.412
e	0.144	0.246	0.467
f	0.155	0.244	0.451
g	0.149	0.238	0.408
h	0.139	0.232	0.432
I	0.146	0.232	0.392
j	0.148	0.236	0.436
10a	0.15	0.251	0.504
b	0.15	0.245	0.471
c	0.14	0.239	0.472
d	0.151	0.239	0.475
e	0.134	0.235	0.468
f	0.151	0.245	0.437
g	0.152	0.226	0.439
h	0.142	0.245	0.444
I	0.148	0.235	0.458
j	0.148	0.239	0.431
11a	0.16	0.232	0.435
b	0.15	0.249	0.497
c	0.16	0.23	0.444
d	0.149	0.236	0.51
e	0.145	0.24	0.445
f	0.156	0.241	0.474
g	0.14	0.245	0.462

h	0.159	0.235	0.448
I	0.14	0.23	0.527
j	0.139	0.242	0.483
12a	0.139	0.234	0.355
b	0.14	0.246	0.454
c	0.14	0.245	0.484
d	0.13	0.242	0.456
e	0.132	0.246	0.427
f	0.14	0.24	0.361
g	0.128	0.24	0.414
h	0.145	0.239	0.392
I	0.144	0.238	0.428
j	0.135	0.231	0.401
13a	0.149	0.249	0.389
b	0.15	0.235	0.408
c	0.14	0.24	0.355
d	0.15	0.24	0.448
e	0.144	0.245	0.486
f	0.145	0.242	0.382
g	0.146	0.245	0.367
h	0.148	0.236	0.468
I	0.132	0.249	0.418
j	0.148	0.25	0.372
14a	0.15	0.231	0.44
b	0.16	0.249	0.371
c	0.151	0.241	0.458
d	0.152	0.211	0.419
e	0.155	0.23	0.412
f	0.151	0.231	0.453
g	0.151	0.245	0.412
h	0.15	0.219	0.352
I	0.16	0.24	0.365
j	0.152	0.25	0.429
15a	0.13	0.251	0.399
b	0.125	0.25	0.387
c	0.121	0.24	0.349
d	0.132	0.23	0.387
e	0.13	0.239	0.368
f	0.128	0.24	0.38
g	0.13	0.245	0.391

h	0.13	0.241	0.403
I	0.13	0.252	0.377
j	0.125	0.231	0.32

Table D.4 ILSS Coefficient of Variation Values for Composites.

Composite Panel Number	ILSS COV
1	9.9
2	25
3	18
4	26
5	12
6	5.8
7	7.3
8	9.2
9	5.4
10	6.7
11	11
12	10
13	12
14	12
15	4.5

APPENDIX E.

Comparison of the use of Water vs. Isopropyl Alcohol for Density Analysis of Cellulose and Cellulose Acetate Butyrate.

E.0 Density of Pure Components.

Density determination using buoyancy can be difficult for highly polar materials such as cellulose. Measuring the density of cellulose rather than the density of cellulose acetate butyrate proves to be impossible using water as the test liquid. The density of CAB is easily obtained through the use of water as the test liquid and the manufacturer's stated density can be matched. However, using water, the density of this particular regenerated cellulose is found to be approximately 1.5 g/cc. This is considerably higher than the manufacturer's stated value of 1.32 g/cc, which can be accurately obtained using IPA as the test liquid.

E.1 Density of Composites using IPA vs. Water.

Table E.0 illustrates the difference in density values obtained using IPA vs. water.

Table E.0 Density Values of Chapter 4 Specimens obtained using Water and IPA.

Composite Panel Number	Density (g/ml) from Water	Density (g/ml) from IPA
1	1.13	1.16
2	1.14	1.19
3	0.94	0.95
4	1.10	1.13
5	1.12	0.99
6	1.20	1.23
7	1.07	1.13
8	1.18	1.19
9	1.19	1.21
10	1.17	1.21
11	1.17	1.23
12	1.12	1.15
13	1.10	1.13
14	1.10	1.12
15	1.13	1.17

As can be seen, the density values of the composites do not vary greatly from the use of IPA and water with the exception of slightly higher density values generally obtained using IPA. This is simply a reflection of the fact that since not much cellulose

surface area is exposed during density determination (only on the specimen edges), little error is obtained due to sorption of the test liquid.

APPENDIX F.

A Simple Analysis of Unsteady State Heat Transfer in the Prepreg Warm-up Stage During Consolidation.

F.0 Necessary Constants for Cellulose Acetate Butyrate.

The necessary heat transfer related constants were obtained from <www.matweb.com>.

k = thermal conductivity = 0.25 kg m / s^3 K
ρ = density = 1200 kg / m^3
C_P = heat capacity = 1400 m^2 / s^2 K

The glass transition temperature (T_g) of this particular CAB used in this thesis work is ca. 120 °C[2].

F.1 System Particular Constant Values.

The half thickness of the prepreg will be estimated as half the thickness of the consolidated composite.

b = half thickness = 0.002 m

The temperature parameters include the temperature of the platens (T_1 = 200 °C) and the initial temperature of the prepreg (T_0 = 25 °C).

The thermal diffusivity (α) is calculated as follows:

α = thermal diffusivity = k / ρ C_P = 1.5E-7 m^2 / s (E.0)

F.2 Calculating the Time Needed for the Center to Reach a Particular Temperature.

Following the method in Bird *et al.*[3] in calculating a solution to unsteady state heat conduction in a semi-inifite slab having a finite thickness and a boundary condition of temperature T_0 applied to both surfaces at time (t) = 0, a dimensionless temperature (θ) is first calculated:

[2] Determined using Dynamic Mechanical Thermal Analysis (DMTA).
[3] Bird, R.B., Stewart, W.E. and E.N. Lightfoot. 1960. **Transport Phenomena**. John Wiley and Sons, N.Y., U.S.A.

$$\theta = \frac{T - T_0}{T_1 - T_0} \qquad \text{(E.1)}$$

Using a table illustrated in the text, a value for ($\alpha\, t / b^2$) can be obtained in which t is the time that it takes for the center of the semi-infinite slab to reach temperature T.

Following this method, the approximate time that it would take for the center of the composite prepreg to reach it's glass transition temperature would be 11 seconds. The time that it would take to reach thermal equilibrium with the platens is about 30 seconds. This indicates that a warm up time of two minutes is more than sufficient to prevent mechanical damage to the prepregs when applying pressure. A polymer prepreg system that is not in the melt state however would be expected to undergo damage under the application of pressure.

VITA

Kevin Christopher Seavey was born second of identical twins in Downey, California on March 4, 1975 and shortly thereafter moved to Maryland. During his stay at Old Mill Senior High School, Millersville, MD, he found that he was competitive in wrestling, co-captaining his team to a fifth straight state championship and was the state runner-up in the 130 lbs. weight class. Furthermore, during high school, he attained the rank of Eagle Scout in the Boy Scouts of America. He graduated from high school in the spring of 1993 and afterward accepted an offer of admission to attend The Johns Hopkins University, Baltimore, MD. During his time as an undergraduate, he discovered mountaineering, and currently enjoys free, aid as well as ice climbing. He graduated with a Baccalaureate Degree in Chemical Engineering in the spring of 1997. In the spring of 1998, he started his graduate study under the guidance of Dr. Wolfgang G. Glasser. After completing his coursework in three semesters as well as a summer internship at Eastman Chemical Company, Kingsport, TN, he finished his Masters of Science requirements in late September, 1999.