Ryerson University
Faculty of Engineering & Architectural Science
Department of Mechanical and Industrial Engineering

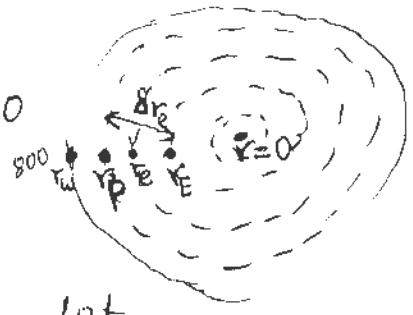
Course Number	AE8112
Course Title	Computational Fluid Dynamics and Heat Transfer
Semester/Year	Summer/Spring 2021
Instructor	Dr. Seth Dworkin

Problem Set 2

Submission Date	June 10, 2021
Programing Language Used	Fortran90

Student Name	Student Number
Ezeorah Godswill	501012886

Q1.a. Given $\nabla.(\lambda \nabla T) - \frac{\varepsilon \sigma A (T^4 - T_\infty^4)}{V} = 0$



we are also given ε, σ & T_∞ values,

To solve the above in term of r, let

the Area, $A = 2\pi r^2$ (we multiply by two, because radiation happens on the top and bottom surfaces)

the volume, $V = \pi r^2 (t)$

where thickness, $t = 0.001\,m$

since $T_\infty = 0$, The eqn will reduce to

$$\frac{\partial}{\partial r}\cdot\left(1000\frac{\partial T}{\partial r}\right) - \frac{2(5.67\times10^{-8})\pi r^2 (T^4)}{\pi r^2 (0.001} = 0$$

Integrating

$$\int_w^e \frac{\partial}{\partial r}\cdot\left(1000\frac{\partial T}{\partial r}\right)dr - \int_w^e 11.34\times10^{-5} T^4 dr = 0$$

we will get

$$1000\left.\frac{\partial T}{\partial r}\right|_e - 1000\left.\frac{\partial T}{\partial r}\right|_w - 11.34\times10^{-5}\left(T_e^4 r_e - T_w^4 r_w\right) = 0$$

using energy flux conservation, we will have that

$$T_e = \frac{T_E + T_p}{2} \quad \& \quad T_w = \frac{T_p + T_w}{2}$$

So that the equation becomes

And note that since r reduces from the outer edge

$\delta r = -\delta r_e = r_E - r_p$, similarly $\delta r = -\delta r_w$

Q1.a| continued|

$$\frac{1000(T_P - T_W)}{\delta r_w} - \frac{1000(T_E - T_P)}{\delta r_e} - \frac{11.34\times10^{-5}}{16}\left[\frac{(T_E+T_P)^4(r_e)}{i}\right]$$

$$+ \frac{11.34\times10^{-5}}{16}(T_P+T_W)^4(r_w) = 0$$

for ease of computation, we can re-write this as

$$\gg \quad a(T_P - T_W) - a(T_E - T_P) - b(T_E+T_P)^4 r_e + b(T_P+T_W)^4 r_w = 0$$

where $a = 1000/\delta r$ and $b = 11.34\times10^{-5}/16$

For the first C.V., lets apply the B.C. for $r=0.1$, as $T_w = 800$, our eqn will become

$$\frac{2000(T_P - 800)}{\delta r_w} - \frac{1000(T_E - T_P)}{\delta r_e} - \frac{11.34\times10^{-5}}{16}(T_E+T_P)^4 r_e$$

$$+ \frac{11.34\times10^{-5}}{16}(800)^4 r_w$$

This can be re-written as

$$\gg \quad 2a(T_P - 800) - a(T_E - T_P) - b(T_E+T_P)^4 r_e + 16b(800)^4 r_w$$

Also for the last C.V., we apply the Neumann, B.C of $\left.\frac{dT}{dr}\right|_{r=0} = 0$, so that our eqn becomes

$$\frac{1000(T_P - T_W)}{\delta r_w} - \frac{2000\cancel{(T_P - T_P)}^{0}}{\delta r_e} - 11.34\times10^{-5} T_P^4 r_e$$

$$+ \frac{11.34\times10^{-5}}{16}(T_P+T_W)^4 r_w$$

This can be re-written as,

Q1.a] continued]

» $a - \frac{1}{6} b T_p^4 r_e + b(T_p + T_w)^4 r_w$

* Power

Next the Power into the disc can be calculated using heat conduction formular

$$q_p^{in} = \frac{V_c (T_w - T_p) 4\lambda}{r_w^2 - r_e^2}$$

where the control volume, $V_c = \pi (r_w^2 - r_e^2)(0.001)$

So that the eqn becomes

$$q_i^{in} = 0.001 \pi (T_w - T_p) 4\lambda \quad , \quad i = 1 \text{ to } N$$

This means that the total power-in,

$$P_{in} = \sum_{i=1}^{n} q_i^{in}$$

→ Also we can verify this using another method, if we consider the entire disc volume as a whole, and extract the $T_{r=0}$ (we computed) and the ~~te~~ $T_{r=0.1} = 800K$

So that $P_{in} = 0.001 \pi (800 - T_n) 4\lambda$

Both P_{in} equations will yield same result.

Next for the power radiated out of the two surfaces of the disc, we have the heat radiation formular as,

Q1.a] continued! $q_p^{out} = \varepsilon \sigma A_c (T_p^4)$

where the areas of the top & bottom part of the c.v. is, $A_c = 2\pi(r_\omega^2 - r_e^2)$

so that $q_i^{out} = (5.67 \times 10^{-8}) 2\pi (r_\omega^2 - r_e^2) T_p^4$

hence the total power-out

$$P_{out} = \sum_{i=1}^{n} q_i^{out}$$

The above discretized will be program in fortran-90

Q1.b] My code was able to handle upto 2,499 control volumes (n), before exceeding ten minutes, for convergance.

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on June 10, 2021.
!This program solves a linear/non-linear equation using finite volume method
!and was written as a solution to AE8112 PS2 q1a
!**************************End Header*****************************************
program newt_multiv
    implicit none
    DOUBLE PRECISION, dimension(:), ALLOCATABLE :: T_bar, f_T, Ti, ri
    DOUBLE PRECISION, dimension(:,:), ALLOCATABLE :: Jac
    DOUBLE PRECISION, PARAMETER :: Pi = 4*atan(1.0), del=8.4*10.0**(-8), r=0.10
    DOUBLE PRECISION :: tol, norm, Pin, Pout, Pint, HL_error, Teg, start, finish
    INTEGER :: n, i1
    call cpu_time(start) !gets the start time, for timing purpose
    !Initialization of variables
    tol=10.0**(-9) !The given tolerance
    Teg=800        !Temperature at the edge of the disc
    open(1, file = 'PS2_Q1a.csv', status = 'unknown')
    !Loop through for n=100, 200 and 300 control volumes
    do n = 100, 300, 100
        ALLOCATE(T_bar(n),f_T(n),jac(n,n),Ti(n),ri(n))
        norm=1
        i1=1
        !Initial geuss
        T_bar=Teg
        do while (norm>tol)
            call solve_fT(T_bar,F_T, ri)
            call solve_jx(jac, T_bar)
            call solve_gauss(jac,f_T,Ti)
            T_bar=T_bar+Ti
            norm=norm2(Ti)/n
            i1=i1+1
        end do
        Pint=pi*0.001*(Teg-T_bar(n))*4*1000 !Another approach to calculating Power-in
        call solve_P(T_bar,Pin,Pout)
        HL_error=100*abs(Pin-Pout)/Pin  !calculates % heat loss error
        !Printing our results
        print *, "no. of Control Volume", n
        print *, "no. Iteration before Convergance", i1
        write(*,2) "T(r) = ", T_bar
        write(*,3) "Power in = ", Pin
        write(*,3) "Pin another_method = ", Pint
        write(*,3) "Power out = ", Pout
        write(*,3) "% Heat Loss Error = ", HL_error
        if (n<300) then
            DEALLOCATE(T_bar,f_T,jac,Ti,ri)
        end if
    end do
    do i1 = 1, 300
        write(1,*) ri(i1), T_bar(i1) !Writing the results to a .txt file
    end do
    close(1)
    call cpu_time(finish) !gets the end time
    write(*,1) "Program Execution Time = ", (finish-start)/60,"min"
    1 format(a40,f10.3,a3)
    2 format(a7,300f10.3)
    3 format(a40,f10.3)

    contains
```

```
!*****************************************************************************
    subroutine solve_fT(Tr, fT, rs)
        DOUBLE PRECISION, dimension(n), INTENT(OUT) ::  fT, rs
        DOUBLE PRECISION, dimension(n) :: Tr
        DOUBLE PRECISION :: aa, ab, TW, TP, TE, rr, dr
        INTEGER :: i
    !!This subroutine creates and solve the discretized function

        !Initialization of variables
        dr=r/n
        rr=r
        aa=1000/dr    !for equispaced grid
        ab=(11.34*10.0**(-5))/16
        do i = 1,n !Loops through our control volumes to solve the function vector
            TP=Tr(i)
            if ( i==1 ) then
                TE=Tr(i+1)
                fT(1)=-aa*(TE-TP) + 2*aa*(TP-Teg) - ab*(rr-dr)*(TE+TP)**4 + ab*16*rr*Teg**4
            else if ( i<n ) then
                TW=Tr(i-1)
                TE=Tr(i+1)
                fT(i)=-aa*(TE-TP) + aa*(TP-TW) - ab*(rr-dr)*(TE+TP)**4 + ab*rr*(TP+TW)**4
            else
                TW=Tr(i-1)
                fT(n)= aa*(TP-TW) - ab*16*(rr-dr)*(TP)**4 + ab*rr*(TP+TW)**4
            end if
            rs(i)=rr-dr/2
            rr=rr-dr
        end do
        fT=-fT !Using Newton's methos, this has to be negative
    end subroutine solve_fT
!*****************************************************************************
!*****************************************************************************
    subroutine solve_Jx(Jx, T_b)
        DOUBLE PRECISION, dimension(n,n) ::  Jx
        DOUBLE PRECISION, dimension(n) :: f_per, T_b, rs
        DOUBLE PRECISION :: per
        integer  :: j, k
    !!This subroutine creates and solve the Jacobian of our function
    !!Using numerically evaluated Jacobian

        do j=1,n
            per=del*T_b(j)+del
            T_b(j)=T_b(j)+per
            call solve_fT(T_b, f_per, rs)
            T_b(j)=T_b(j)-per
            do k=1,n
                Jx(k,j)=(-f_per(k)+f_T(k))/per
            end do
        end do
    end subroutine solve_Jx
!*****************************************************************************
!*****************************************************************************
    subroutine solve_P(Tr, P1, P2)
        DOUBLE PRECISION, INTENT(OUT) :: P1, P2
        DOUBLE PRECISION, dimension(n) ::  qin, qout
        DOUBLE PRECISION, dimension(n) :: Tr
        DOUBLE PRECISION :: aa, ac, ab, TW, TP, rr, dr
```

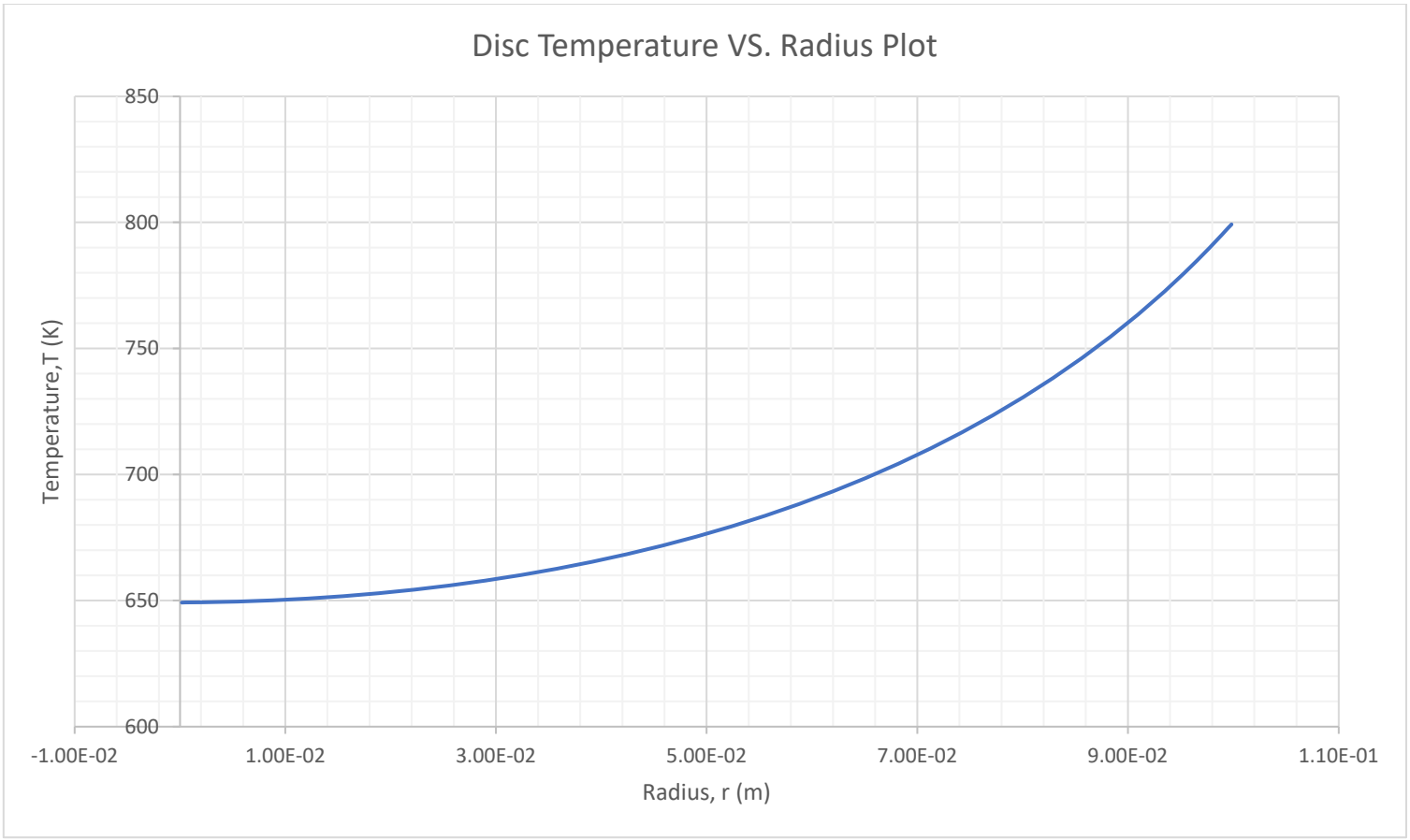
```
        INTEGER :: i
    !!This subroutine solves the Power-in and Power-out, using our formulation

        !Variable initialization
        dr=r/n
        rr=r
        aa=0.001*pi
        ac=pi*11.34*10.0**(-8)
        ab=aa*4*1000
        TW=Teg   !Temperature maintained at the edge
        do i = 1,n
            TP=Tr(i)
            if ( i==1 ) then
                qin(1)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(1)=ac*(rr**2-(rr-dr)**2)*TP**4
            else if ( i<n ) then
                TW=Tr(i-1)
                qin(i)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(i)=ac*(rr**2-(rr-dr)**2)*TP**4
            else
                TW=Tr(i-1)
                qin(n)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(n)=ac*(rr**2-(rr-dr)**2)*TP**4
            end if
            rr=rr-dr
        end do
        P1=sum(qin)
        P2=2*sum(qout) !Multiplied by 2, because radiation occurs from the top and bottom.
    end subroutine solve_P
!*********************************************************************************
!*********************************************************************************
    subroutine solve_gauss(Ag1, b, X)
        implicit none
        DOUBLE PRECISION, dimension(n), INTENT(OUT)   :: X
        DOUBLE PRECISION, dimension(n,n)              :: Ag1
        DOUBLE PRECISION, dimension(n,n+1)            :: A
        DOUBLE PRECISION, dimension(n+1)              :: prod, new, switch1, switch2
        DOUBLE PRECISION, dimension(n)                :: b
        INTEGER, dimension(1)                      :: ros
        DOUBLE PRECISION                              :: ratio
        INTEGER                                    :: steps, k1, m1, t1, i, j
        !! This subroutine solve a linear problem using Guassian Elimination method (with partial pivoting)

        !Initialization of variables
        X=0
        t1=1
        !rewriting the given matrices in the augumented matrix form
        A(1:n,1:n)=Ag1
        A(1:n,n+1)=b
        !!Foreward elimination
        steps=n-1                   !Total number of forward elimination steps
        do i=1,steps             !for new pivot point (diagonally, starting from 1,1)
            ros=maxloc(abs(A(i:n,i)))

            if ( ros(1)==1 ) then    !if the 1st element of the 1st row is the largest (i.e. at pivot)
                do j=i,steps           !for consecutive rows below the pivot
                    ratio=A(j+1,i)/A(i,i)
                    prod=A(i,:)*ratio
```

```
                    new=A(j+1,:)-prod
                    A(j+1,:)=new
                end do
                else                !else switch the 1st row with the with the row with the largest column
element
                switch1=A(i,:)
                switch2=A(ros(1),:)
                A(i,:)=switch2
                A(ros(1),:)=switch1
                do j=i,steps              !Steps
                    ratio=A(j+1,i)/A(i,i)
                    prod=A(i,:)*ratio
                    new=A(j+1,:)-prod
                    A(j+1,:)=new
                end do
                end if
        end do

        !!Since the upper diagonal is obtained from the above, the X varibles can be solved as:
        ! x8=A(n,n+1)/A(n,n)
        ! x7=A(n,n+1)/A(n-1,n-1)-x8*A(n-1,n)/A(n-1,n-1)
        ! x6=A(n,n+1)/A(n-2,n-2)-x8*A(n-2,n)/A(n-2,n-2)-x7*A(n-2,n-1)/A(n-2,n-2)
        ! x5=A(n,n+1)/A(n-3,n-3)-x8*A(n-3,n)/A(n-3,n-3)-x7*A(n-3,n-1)/A(n-3,n-3)-x6*A(n-3,n-2)/A(n-3,n-3)
        !...
        !using the above pattern we can create the backward substition, using a loop and a recuresive proced
ures
        !as shown below

      !! this function account for the vertical change in the above pattern
        do k1 = steps, 0, -1
            m1=n-k1
            if ( k1==0 ) then
                 X(n)=A(n,n+1)/A(n,n)
             else
                 X(n)=A(n,n+1)/A(n,n)
                 X(k1)=A(k1,n+1)/A(k1,k1)-r_fun(A,X,k1,m1,t1)
            end if
        end do
        end subroutine solve_gauss
      !! this function accounts for the horinzontal change in the pattern
              recursive function r_fun(A1,Xr,k,m,t) result(fr)
                  DOUBLE PRECISION, dimension(n,n+1) :: A1
                  DOUBLE PRECISION, dimension(n) :: xr
                  DOUBLE PRECISION :: fr
                  INTEGER :: m, k, t
                  if ( m==0 ) then
                      fr=Xr(n-(t-1))*(A1(k,n-(t-1))/A1(k,k))
                  else
                      fr=Xr(n-(t-1))*(A1(k,n-(t-1))/A1(k,k))+r_fun(A1,Xr,k,m-1,t+1)
                  end if
              end function r_fun
!*******************************************************************************
end program newt_multiv
```

no. of Control Volume 100
no. Iteration before Convergence 6
T(r) = 797.678 793.183 788.833 784.619 780.536 776.577 772.739 769.015 765.400 761.891 758.482 755.171 751.953 748.825 745.784 7
42.825 739.947 737.147 734.422 731.769 729.186 726.670 724.221 721.834 719.510 717.245 715.037 712.886 710.790 708.747 706.755
704.813 702.921 701.075 699.277 697.523 695.813 694.147 692.522 690.938 689.395 687.890 686.423 684.994 683.602 682.245 680.92
4 679.637 678.383 677.163 675.975 674.818 673.693 672.599 671.535 670.500 669.495 668.518 667.569 666.649 665.755 664.888 664.0
48 663.235 662.447 661.684 660.947 660.234 659.546 658.882 658.242 657.626 657.033 656.463 655.917 655.393 654.891 654.412 653.
956 653.521 653.107 652.716 652.346 651.997 651.669 651.363 651.077 650.813 650.569 650.346 650.143 649.961 649.799 649.657 64
9.536 649.435 649.355 649.294 649.254 649.234
Power in = 1894.586 W
Pin another_method = 1894.586 W
Power out = 1894.284 W
% Heat Loss Error = 0.016

no. of Control Volume 200
no. Iteration before Convergence 6
T(r) = 798.839 796.554 794.307 792.097 789.921 787.780 785.673 783.600 781.558 779.548 777.569 775.620 773.701 771.810 769.948 7
68.114 766.306 764.526 762.771 761.042 759.338 757.658 756.002 754.370 752.761 751.175 749.611 748.069 746.548 745.048 743.569
742.110 740.671 739.252 737.851 736.470 735.107 733.763 732.436 731.127 729.836 728.561 727.304 726.063 724.838 723.629 722.43
5 721.258 720.095 718.948 717.815 716.697 715.594 714.504 713.429 712.367 711.319 710.284 709.262 708.254 707.258 706.274 705.3
04 704.345 703.399 702.464 701.542 700.631 699.731 698.843 697.966 697.101 696.246 695.402 694.568 693.746 692.933 692.131 691.
339 690.557 689.786 689.024 688.271 687.528 686.795 686.071 685.357 684.651 683.955 683.268 682.590 681.920 681.259 680.607 67
9.963 679.328 678.702 678.083 677.473 676.871 676.277 675.691 675.113 674.542 673.980 673.425 672.878 672.338 671.806 671.281 6
70.764 670.254 669.751 669.256 668.767 668.286 667.812 667.344 666.884 666.430 665.984 665.544 665.110 664.684 664.264 663.850
663.443 663.043 662.649 662.261 661.880 661.505 661.136 660.774 660.417 660.067 659.723 659.385 659.053 658.727 658.407 658.09
3 657.785 657.483 657.186 656.896 656.611 656.332 656.059 655.791 655.529 655.273 655.022 654.777 654.537 654.303 654.075 653.8
52 653.634 653.422 653.216 653.015 652.819 652.628 652.443 652.264 652.089 651.920 651.756 651.598 651.445 651.297 651.154 651.
016 650.884 650.757 650.635 650.518 650.406 650.300 650.199 650.102 650.011 649.925 649.844 649.768 649.698 649.632 649.571 64
9.516 649.466 649.420 649.380 649.345 649.314 649.289 649.269 649.254 649.244 649.239
Power in = 1894.522 W
Power out = 1894.447 W
% Heat Loss Error = 0.004

no. of Control Volume 300
no. Iteration before Convergence 6
T(r) = 799.226 797.695 796.180 794.682 793.200 791.734 790.284 788.849 787.429 786.025 784.635 783.260 781.899 780.552 779.218 7
77.899 776.593 775.300 774.021 772.754 771.500 770.259 769.030 767.813 766.608 765.415 764.234 763.064 761.905 760.758 759.622
758.497 757.383 756.279 755.186 754.103 753.030 751.967 750.915 749.872 748.839 747.816 746.802 745.797 744.802 743.816 742.83
9 741.871 740.911 739.961 739.019 738.085 737.160 736.244 735.335 734.435 733.543 732.658 731.782 730.913 730.052 729.199 728.3
53 727.514 726.683 725.859 725.043 724.233 723.431 722.635 721.847 721.065 720.290 719.522 718.760 718.005 717.257 716.514 715.
779 715.049 714.326 713.609 712.898 712.193 711.494 710.802 710.115 709.434 708.758 708.089 707.425 706.766 706.114 705.466 70
4.825 704.188 703.557 702.932 702.312 701.696 701.087 700.482 699.882 699.288 698.698 698.114 697.534 696.959 696.389 695.824 6
95.264 694.708 694.157 693.611 693.070 692.533 692.000 691.472 690.949 690.430 689.915 689.405 688.899 688.397 687.900 687.407
686.918 686.434 685.953 685.477 685.005 684.536 684.072 683.612 683.156 682.704 682.255 681.811 681.370 680.934 680.501 680.07
2 679.646 679.225 678.807 678.393 677.982 677.576 677.172 676.773 676.377 675.984 675.595 675.210 674.828 674.449 674.074 673.7
03 673.335 672.970 672.608 672.250 671.896 671.544 671.196 670.851 670.510 670.171 669.836 669.504 669.175 668.850 668.527 668.
208 667.892 667.579 667.269 666.962 666.658 666.357 666.059 665.764 665.472 665.183 664.898 664.615 664.335 664.057 663.783 66
3.512 663.244 662.978 662.715 662.455 662.198 661.944 661.693 661.444 661.198 660.955 660.715 660.478 660.243 660.011 659.781 6
59.555 659.331 659.109 658.891 658.675 658.461 658.251 658.043 657.837 657.634 657.434 657.237 657.042 656.849 656.659 656.472
656.287 656.105 655.925 655.748 655.574 655.401 655.232 655.065 654.900 654.738 654.578 654.421 654.266 654.114 653.964 653.81
7 653.672 653.529 653.389 653.251 653.116 652.983 652.852 652.724 652.599 652.475 652.354 652.236 652.119 652.005 651.894 651.7
85 651.678 651.573 651.471 651.371 651.274 651.179 651.086 650.995 650.907 650.821 650.737 650.656 650.577 650.500 650.426 650.
354 650.284 650.216 650.151 650.088 650.027 649.969 649.913 649.859 649.807 649.758 649.710 649.666 649.623 649.583 649.544 64
9.509 649.475 649.443 649.414 649.387 649.363 649.340 649.320 649.302 649.287 649.273 649.262 649.253 649.246 649.242 649.240
Power in = 1894.511 W
Power out = 1894.477 W
% Heat Loss Error = 0.002
Program Execution Time = 0.005min


Disc Temperature VS. Radius Plot
850
800
750
700
650
600
Temperature,T (K)
-1.00E-02
1.00E-02
3.00E-02
5.00E-02
7.00E-02
9.00E-02
1.10E-01
Radius, r (m)

Q1.c] In order to show that the Jacobian forms a tri-diagonal matrix, lets consider the following samples of our c.v. discretized f^n s.

$$f_1(T_1, T_2) = 2a(T_1 - 800) - a(T_2 - T_1) - b(T_2 - T_1)^4 r_e + 16b(800^4) r_w$$

$$f_2(T_1, T_2, T_3) = a(T_2 - T_1) - a(T_3 - T_2) - b(T_3 + T_2)^4 r_e + b(T_2 + T_1)^4 r_w$$

$$f_3(T_2, T_3, T_4) = a(T_3 - T_2) - a(T_4 - T_3) - b(T_4 + T_3)^4 r_e + b(T_3 + T_2)^4 r_w$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$f_i(T_{i-1}, T_i, T_{i+1}) = a(T_i - T_{i-1}) - a(T_{i+1} - T_i) - b(T_{i+1} + T_i)^4 r_e + b(T_i + T_{i-1})^4 r_w$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$f_n(T_{n-1}, T_n) = a(T_n - T_{n-1}) - 16\, b T_n^4 r_e + b(T_n + T_{n-1})^4 r_w$$

where $a = \frac{1000}{\delta r}$ & $b = \frac{11.34 \times 10^{-5}}{16}$ } Same as before

Thus if we consider the Jacobian of these f^n s. at ith iteration, we will get a tri-diagonal, that looks like;

$$\begin{pmatrix}
\frac{\partial f_1}{\partial T_1} & \frac{\partial f_1}{\partial T_2} & 0 \cdots 0 & \cdots & 0 \cdots 0 \\
\frac{\partial f_2}{\partial T_1} & \frac{\partial f_2}{\partial T_2} & \frac{\partial f_2}{\partial T_3} & 0 \cdots 0 & \cdots 0 \\
0 & \frac{\partial f_3}{\partial T_2} & \frac{\partial f_3}{\partial T_3} & \frac{\partial f_3}{\partial T_4} \cdots 0 & \cdots 0 \\
\vdots & & \ddots & \ddots & \vdots \\
0 \cdots 0 & \cdots & \frac{\partial f_i}{\partial T_{i-1}} & \frac{\partial f_i}{\partial T_i} & \frac{\partial f_i}{\partial T_{i+1}} \cdots 0 \\
\vdots & & \cdots 0 \cdots 0 & \cdots & \vdots \\
0 & \cdots 0 & \cdots 0 & \cdots 0 & \frac{\partial f_{n-1}}{\partial T_{n-1}} \quad \frac{\partial f_n}{\partial T_n}
\end{pmatrix}$$

Q1.c] Continued

For computation reason, lets consider the following differentiation;

$$\frac{\partial f_1}{\partial T_1} = 3a - 4b(T_2+T_1)^3 r_e \;,\quad \frac{\partial f_1}{\partial T_2} = -a - 4b(T_2+T_1)^3 r_e$$

$$\frac{\partial f_2}{\partial T_1} = -a + 4b(T_2+T_1)^3 r_w \;,\quad \frac{\partial f_2}{\partial T_2} = 2a - 4b(T_3+T_2)^3 r_e + 4b(T_2+T_1)^3 r_w$$

$$\frac{\partial f_2}{\partial T_3} = -a - 4b(T_3+T_2)^3 r_e$$

$$\frac{\partial f_3}{\partial T_2} = -a + 4b(T_3+T_2)^3 r_w \;,\quad \frac{\partial f_3}{\partial T_3} = 2a - 4b(T_4+T_3)^3 r_e + 4b(T_3+T_2)^3 r_w$$

$$\frac{\partial f_3}{\partial T_4} = -a - 4b(T_4+T_3)^3 r_e$$

$$\vdots$$

$$\frac{\partial f_i}{\partial T_{i-1}} = -a + 4b(T_i+T_{i-1})^3 r_w \;,\quad \frac{\partial f_i}{\partial T_i} = 2a - 4b(T_{i+1}+T_i)^3 r_e + 4b(T_i+T_{i-1})^3 r_w$$

$$\frac{\partial f_i}{\partial T_{i+1}} = -a - 4b(T_{i+1}+T_i)^3 r_e$$

$$\vdots$$

$$\frac{\partial f_n}{\partial T_{n-1}} = -a + 4b(T_n+T_{n-1})^3 r_w \;,\quad \frac{\partial f_n}{\partial T_n} = a - 64bT_n^3 r_e + 4b(T_n+T_{n-1})^3 r_w$$

The above will be used for simulation in Fortran-90

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on June 10, 2021.
!This program solves a linear/non-linear equation using finite volume method
!and was written as a solution to AE8112 PS2 q1c
!**************************End Header*************************************
program newt_Jac_TDMA
    implicit none
    DOUBLE PRECISION, dimension(:), ALLOCATABLE :: T_bar, f_T, Ti, ej, fj, gj
    DOUBLE PRECISION, dimension(:,:), ALLOCATABLE :: Jac
    DOUBLE PRECISION, PARAMETER :: Pi = 4*atan(1.0), r=0.10
    DOUBLE PRECISION :: tol, norm, Pin, Pout, Pint, HL_error, Teg, start, finish
    INTEGER, dimension(5):: N_cvs
    INTEGER :: n, i1, tim, count
    N_cvs=(/500,1000,3000,8000,10000/) !Array of control Volumes to be computed
    !Initialization of variables
    tol=10.0**(-8) !The given Newton Tolerance
    Teg=800        !Temperature at the edge of the disc

    do i1 = 1, size(N_cvs) !Loop through for different control volumes in the N_cvs array
        call cpu_time(start)
        n=N_cvs(i1)
        ALLOCATE(T_bar(n),f_T(n),jac(n,n),Ti(n),ej(n),fj(n),gj(n))
        do tim = 1, 10000 !for timing purpose
            count=1
            norm=1
            !Initial geuss
            T_bar=Teg
            do while (norm>tol)
                call solve_fT(T_bar,F_T)
                call solve_jx(ej,fj,gj,T_bar) !creates the Tri-diagonal vectors
                call tdma(ej,fj,gj,f_T,Ti)
                T_bar=T_bar+Ti
                norm=norm2(Ti)/n
                count=count+1
            end do
            Pint=pi*0.001*(Teg-T_bar(n))*4*1000 !Another approach to calculating Power-in
            call solve_P(T_bar,Pin,Pout)
            HL_error=100*abs(Pin-Pout)/Pin  !% heat loss error
        end do !for timing purpose

        print *, "no. of Control Volume", n
        print *, "no. Iteration before Convergance", count
        write(*,3) "% Heat Loss Error = ", HL_error
        call cpu_time(finish)
        !To get the time taken for one tim_loop run, we divide the time diff. with tim.
        write(*,1) "Program Execution Time = ", ((finish-start)/tim)*1D6,"micro_sec"
        if (n<10000) then
            DEALLOCATE(T_bar,f_T,jac,Ti,ej,fj,gj)
        end if
    end do

    1 format(a40,f8.3,a10)
    3 format(a40,f9.7)

    contains
!*************************************************************************************
    subroutine solve_fT(Tr, fT)
        DOUBLE PRECISION, dimension(n), INTENT(OUT) ::  fT
```

```
    DOUBLE PRECISION, dimension(n) :: Tr
    DOUBLE PRECISION :: aa, ab, TW, TP, TE, rr, dr
    INTEGER :: i
!!This subroutine creates and solve the discretized function

    !Initialization of variables
    dr=r/n
    rr=r
    aa=1000/dr    !for equispaced grid
    ab=(11.34*10.0**(-5))/16
    do i = 1,n
        TP=Tr(i)
        if ( i==1 ) then
            TE=Tr(i+1)
            fT(1)=-aa*(TE-TP) + 2*aa*(TP-Teg) - ab*(rr-dr)*(TE+TP)**4 + ab*16*rr*Teg**4
        else if ( i<n ) then
            TW=Tr(i-1)
            TE=Tr(i+1)
            fT(i)=-aa*(TE-TP) + aa*(TP-TW) - ab*(rr-dr)*(TE+TP)**4 + ab*rr*(TP+TW)**4
        else
            TW=Tr(i-1)
            fT(n)= aa*(TP-TW) - ab*16*(rr-dr)*(TP)**4 + ab*rr*(TP+TW)**4
        end if
        rr=rr-dr
    end do
    fT=-fT
end subroutine solve_fT

subroutine solve_Jx(e, f, g, Tr)
    DOUBLE PRECISION, dimension(n) :: e, f, g, Tr
    DOUBLE PRECISION :: aa, ab, dr, rr, TW, TP, TE
    integer  :: i
!!This subroutine creates and solve the Jacobian of our function
!!Using Thomas TDMA algorithm

    !Variable initialization
    dr=r/n
    rr=r
    aa=1000/dr
    ab=(11.34*10.0**(-5))/16
    do i = 1,n
        TP=Tr(i)
        if ( i==1 ) then
            TE=Tr(i+1)
            g(1)=-aa - 4*ab*(rr-dr)*(TE+TP)**3
            f(1)=3*aa - 4*ab*(rr-dr)*(TE+TP)**3
        else if ( i<n ) then
            TW=Tr(i-1)
            TE=Tr(i+1)
            g(i)=-aa - 4*ab*(rr-dr)*(TE+TP)**3
            f(i)=2*aa - 4*ab*(rr-dr)*(TE+TP)**3 + 4*ab*rr*(TP+TW)**3
            e(i)=-aa + 4*ab*rr*(TP+TW)**3
        else
            TW=Tr(i-1)
            f(n)=aa - 64*ab*(rr-dr)*TP**3 + 4*ab*rr*(TP+TW)**3
            e(n)=-aa + 4*ab*rr*(TP+TW)**3
        end if
        rr=rr-dr
```

```
        end do
    end subroutine solve_Jx
!*******************************************************************************
!*******************************************************************************
    subroutine solve_P(Tr, P1, P2)
        DOUBLE PRECISION, INTENT(OUT) :: P1, P2
        DOUBLE PRECISION, dimension(n) ::  qin, qout
        DOUBLE PRECISION, dimension(n) :: Tr
        DOUBLE PRECISION :: aa, ac, ab, TW, TP, rr, dr
        INTEGER :: i
    !!This subroutine solves the Power-in and Power-out, using our formulation

        !Variable initialization
        dr=r/n
        rr=r
        aa=0.001*pi
        ac=pi*11.34*10.0**(-8)
        ab=aa*4*1000
        TW=Teg   !Temperature maintained at the edge
        do i = 1,n
            TP=Tr(i)
            if ( i==1 ) then
                qin(1)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(1)=ac*(rr**2-(rr-dr)**2)*TP**4
            else if ( i<n ) then
                TW=Tr(i-1)
                qin(i)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(i)=ac*(rr**2-(rr-dr)**2)*TP**4
            else
                TW=Tr(i-1)
                qin(n)=ab*(rr**2-(rr-dr)**2)*(TW-Tp)/(rr**2-(rr-dr)**2)
                qout(n)=ac*(rr**2-(rr-dr)**2)*TP**4
            end if
            rr=rr-dr
        end do
        P1=sum(qin)
        P2=2*sum(qout)!Multiplied by 2, because radiation occurs from the top and bottom.
    end subroutine solve_P

!*********************************************************************************************
    subroutine tdma(e, f, g, b1, x)
        DOUBLE PRECISION, dimension(n), INTENT(IN):: b1
        DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: x
        DOUBLE PRECISION, DIMENSION(n):: b, e, f, g
        INTEGER :: k
    !!This subroutine solves a tri-diagonal linear system using the Thomas Algorithm

        b=b1
       !Decomposition
        do k = 2,n
            e(k) = e(k)/f(k-1)
            f(k) = f(k) - e(k)*g(k-1)
        end do
       !Forward Substitution
        do k = 2,n
            b(k) = b(k) - e(k)*b(k-1)
        end do
       !Backward Substitution
```

```
        x(n)=b(n)/f(n)
        do k = n-1,1,-1  !(step size of -1)
            x(k) = (b(k) - g(k)*x(k+1))/f(k)
        end do
    end subroutine tdma
    !*****************************************************************************************
end program newt_Jac_TDMA
```

```
no. of Control Volume 500
no. Iteration before Convergence 5
% Heat Loss Error = 0.0006389
Program Execution Time = 139.049 micro_sec

no. of Control Volume 1000
no. Iteration before Convergence 5
% Heat Loss Error = 0.0001612
Program Execution Time = 249.975 micro_sec

no. of Control Volume 3000
no. Iteration before Convergence 5
% Heat Loss Error = 0.0000196
Program Execution Time = 629.625 micro_sec

no. of Control Volume 8000
no. Iteration before Convergence 5
% Heat Loss Error = 0.0000044
Program Execution Time = 1806.069 micro_sec


no. of Control Volume 10000
no. Iteration before Convergence 5
% Heat Loss Error = 0.0000035
Program Execution Time = 4244.888 micro_sec
```

Q1.d] The over all execution time of my Program was 41.062 sec

>> And I observed, from the above simulation that the % heat loss error reduces as the number of control volumes (N) increases. And that the execution time for the program increases (in micro-seconds) as the number of control volume increases.

Q2.a] To perform the fourier stability analysis for

$$\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

we know that for Crank-Nicholson method

Let $\theta = \frac{1}{2}$ in

$$\left.\frac{\partial T}{\partial t}\right|_{n \to n+1} = \theta \left.\frac{\partial^2 T}{\partial x^2}\right|_{n+1} + (1-\theta) \left.\frac{\partial^2 T}{\partial x^2}\right|_{n}$$

(n - represents the nth time-step

and with the T.s expansion, we have that,

$$\left.\frac{\partial T}{\partial t}\right|_{n \to n+1} = \frac{T_i^{n+1} - T^n}{\Delta t}$$

(standard forward/diff. Euler)

and that for the second order differentiation,

Q2 a) continued

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$$

so that the eqn becomes,

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{1}{2}\left[\frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{\Delta x^2} + \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2}\right]$$

let $r = \frac{\Delta t}{\Delta x^2}$, now re-arranging, we get,

$$(1+r)T_i^{n+1} = \frac{r}{2}\left[T_{i+1}^{n+1} + T_{i-1}^{n+1} + T_{i+1}^n + T_{i-1}^n\right] - (r-1)T_i^n$$

If we seperate our time and space variables we can assume that for a B.Cs of

$$T(x_0, t) = 0 \text{ and } T(x_N, t) = 0$$

($N \rightarrow$ is the last grid point on x-Dimension)

we know that the seperated variables can be represented as,

$$T^n = Ae^{\alpha t^n} \text{ and } x_i = Be^{\sqrt{-1}\lambda x_i}$$

numerically we have that $x_i = \Delta x$ & $t^n = n\Delta t$ ($\lambda \rightarrow$ is a fourier solution parameter)

so that $T_i^n = Ae^{\sqrt{-1}\lambda i\Delta x} e^{\alpha n\Delta t} = Ae^{\sqrt{-1}\lambda i\Delta x + \alpha n\Delta t}$

with this assumption our eqn can now be re-written as,

$$(1+r)Ae^{\sqrt{-1}\lambda i\Delta x + \alpha(n+1)\Delta t} = \frac{r}{2}\left[Ae^{\sqrt{-1}(i+1)\Delta\cdot}\right.$$

Q 2/a] Continued

$$(1+r)Ae^{\sqrt{-1}\lambda i\Delta x+\alpha(n+1)\Delta t} = \frac{r}{2}\Big[Ae^{\sqrt{-1}(i+1)\Delta x+\alpha(n+1)\Delta t} + Ae^{\sqrt{-1}(i-1)\Delta x+\alpha(n+1)\Delta t} + Ae^{\sqrt{-1}\lambda(i+1)\Delta x+\alpha n\Delta t} + Ae^{\sqrt{-1}\lambda(i-1)\Delta x+\alpha n\Delta t}\Big] - (1-r)Ae^{\sqrt{-1}\lambda i\Delta x+\alpha n\Delta t}$$

Cancelling out like terms between the LHS & RHS, we will get,

$$(1+r)e^{\alpha\Delta t} = \frac{1}{2}\Big[e^{\sqrt{-1}\lambda\Delta x+\alpha\Delta t} + e^{-\sqrt{-1}\lambda\Delta x+\alpha\Delta t} + e^{\sqrt{-1}\lambda\Delta x} + e^{-\sqrt{-1}\lambda\Delta x}\Big] - (r-1)e^{0}$$

$e^{0} = 1$

Applying the trignometric Identity of $2\cos\theta = e^{i\theta}+e^{-i\theta}$ we get,

$$e^{\alpha\Delta t} = \frac{r}{2}\Big[2\cos\lambda\Delta x e^{\alpha\Delta t} + 2\cos\lambda\Delta x\Big] - (r-1)$$

collecting like terms,

$$(1+r-r\cos\lambda\Delta x)e^{\alpha\Delta t} = r\cos(\lambda\Delta x) - r + 1$$

Lets apply another trignometric Identity of

$$\cos\theta = 1 - 2\sin^2\left(\frac{\theta}{2}\right)$$

our eqn becomes,

$$\Big[1+r-r+2r\sin^2\left(\frac{\lambda\Delta x}{2}\right)\Big]e^{\alpha\Delta t} = r - 2r\sin^2\left(\frac{\lambda\Delta x}{2}\right) - r + 1$$

$$e^{\alpha\Delta t} = \frac{-2r\sin^2\left(\frac{\lambda\Delta x}{2}\right)+1}{1+2r\sin^2\left(\frac{\lambda\Delta x}{2}\right)}$$

we know the Stability Criteria requires

$$\left|e^{\alpha\Delta t}\right| \leq 1$$

Q2 (a) continued

$$\text{i.e.} \quad \left| \frac{-2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)+1}{2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)+1} \right| \leq 1$$

$$\left|-2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)+1\right| \leq \left|2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)+1\right|$$

since we know $0 < \sin^2\gamma \leq 1$

Our stability criteria will have two posible scenario;

$-2r\sin^2\left(\lambda\frac{\Delta x}{2}\right) \leq 2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)$ and $2r\sin^2\left(\lambda\frac{\Delta x}{2}\right) - 2 \leq -2r\sin^2\left(\lambda\frac{\Delta x}{2}\right)$

This will translate to,

$$r \geq 0 \qquad \text{and} \qquad r \leq 1$$

so that the stability criteria to be satisfied, we have

$$\boxed{0 \leq r \leq 1}$$

≫ Next to check for consistency, lets look at the T.S of our governing eqn

$$2\frac{\partial T}{\partial t} = 2\frac{(T_i^{n+1} - T_i^n)}{\Delta t} + \Delta t\frac{\partial^2 T}{\partial t^2}\bigg|_{t^n} + \frac{\Delta t^2}{3}\frac{\partial^3 T}{\partial t^2}\bigg|_{t^n} + \cdots$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} + 0 + 0 + \cdots$$

This clearly shows that our discretization

$$\sum \varepsilon_t \to 0 \quad \text{as} \quad \Delta t, \Delta x \to 0$$

Therefore our formulation is consistent.

Q 2.a) Continued

>> Hence this means that if $0 \leq r \leq 1$ our formulation will be convergent, but if $0 \geq r \geq 1$, then our formulation will no longer be convergent, but rather it will be oscillatory (which in practical sense, is still unconditional stable).

Q 3.a) Comparing the error norms calculated using Gauss-Seidel method to that of the chaprat Canale TOMA method, I observed that at the number of control volume, $N = 64$, the three error norms higher with the Gauss-Seidel method. I also observed that the difference in error between these two methods increases as N increases.

The programming is done in fortran-90 as shown below;

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on June 10, 2021.
!This program solves a linear/non-linear equation using finite volume method
!and was written as a solution to AE8112 PS1 q3a
!**************************End Header*****************************************
program Vfinite_Gauss_seid
    !Variable declaration
    implicit none
    DOUBLE PRECISION, dimension(:,:), ALLOCATABLE:: Ag
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE:: bg, Ti, Te
    DOUBLE PRECISION, PARAMETER :: Pi = 4*atan(1.0) !pi parameter definition
    DOUBLE PRECISION  :: L1, L2, Linf, R, dx
    integer :: N, ii, m
    N=8 !First number of control volumes
    open(1, file = 'PS2_Q3a.txt', status = 'unknown')
do while(n<=64)  !This will run for N=8,16,32 and 64
    ALLOCATE(Ag(n,n),bg(n),Ti(n),Te(n)) !array allocation
   call creat_eTDMA(Ag,bg,Te,dx) !creates Tri-diagonal matric for equispaced grid
   ! for the initial guess
    do ii = 1, n
        Ti(ii)=bg(ii)/Ag(ii,ii)
    end do
   !For Gauss Siedel iteration
   do m = 1, 3000
        call gauss_siedel(Ag,bg,Ti)
        R=sqrt(sum((bg-matmul(Ag,Ti))**2))/N
        write(1,*) m, R
   end do

   close(1)
   !Calculating the error terms
    L1=sum(dabs(Ti(1:n)-Te(1:n)))
    L2=sum((Ti(1:n)-Te(1:n))**2)
    Linf=maxval(dabs(Ti(1:n)-Te(1:n)))
    !Calculating the linear system residual term

    print 3, 'For number of grid points, n =',n
    write(*,1)  "T(x) = ",Ti
    write(*,2) "L1 = ", L1
    write(*,2) "L2 = ", L2
    write(*,2) "L∞ = ", Linf
    write(*,2) "R = ", R
    1 format(a6,64f8.3)
    2 format(a7,E10.3)
    3 format(a30,i3)
    !writing one of the errors (L2) as a function of grid spacing (dx) to a .txt file
    write(1,*) dx, L2
    if(n<=64) deallocate(Ag,bg,Ti,Te) !deallocates the arrays for every new N
    N=n+n
end do

contains
!*****************************************************************************
subroutine creat_eTDMA(Ag1,bg1,Te1,dx1)
    implicit none
    DOUBLE PRECISION, dimension(n,n), INTENT(OUT):: Ag1
    DOUBLE PRECISION, DIMENSION(n), INTENT(OUT):: bg1, Te1
    DOUBLE PRECISION, INTENT(OUT) :: dx1
```

```
    DOUBLE PRECISION  :: aa, ap, xi
    integer :: i
    !!This subroutine generates the tri-diagonal matric using our derived equations

    !Initializing Variables
    Ag1=0
    xi=0
        dx1=2*pi/n
        aa=1/(dx1)
        ap=aa+aa
    do i = 1, n  !This loop Matrix composition of the given problem
        xi=xi+dx1
        !for the the first gridpoint, applying B.Cs T(0)=1
        if ( i==1 ) then
            Ag1(i,i)=-aa-2*aa
            Ag1(i,i+1)=aa
            bg1(i)=-2*aa-cos(dx1/2)*dx1
            Te1(i)=cos(dx1/2) !This is T_exact from our analytical solution
        !for the the intermediate gridpoint
        else if ( i<n ) then
            Ag1(i,i-1)=aa
            Ag1(i,i)=-ap
            Ag1(i,i+1)=aa
            bg1(i)=-cos(xi-dx1/2)*dx1
            Te1(i)=cos(xi-dx1/2)
        !for the the last gridpoint, applying B.Cs T(2pi)=1
        else
            Ag1(n,n-1)=aa
            Ag1(n,n)=-2*aa-aa
            bg1(n)=-2*aa-cos(xi-dx1/2)*dx1
            Te1(n)=cos(xi-dx1/2)
        end if
    end do

end subroutine creat_eTDMA
!*******************************************************************************
!*******************************************************************************
subroutine gauss_siedel(A,b,x)
    implicit none
    DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: x
    DOUBLE PRECISION, dimension(n,n), INTENT(IN):: A
    DOUBLE PRECISION, dimension(n), INTENT(IN):: b
    DOUBLE PRECISION, DIMENSION(n) :: x_m
    INTEGER :: i
!!This subroutine solves a tri-diagonal linear system using the Gauss Siedel Method
    do i = 1,N
        x_m(i)=(1/a(i,i))*(b(i)-sum(a(i,1:(i-1))*x_m(1:(i-1)))-sum(a(i,(i+1):N)*x((i+1):N)))
    end do
    x=x_m
end subroutine gauss_siedel
!*******************************************************************************

end program Vfinite_Gauss_seid
```

For number of grid points, n = 8

T(x) = 1.000 0.430 -0.376 -0.946 -0.946 -0.376 0.430 1.000
L1 = 0.304E+00
L2 = 0.171E-01
L∞ = 0.761E-01
R = 0.628E-16

For number of grid points, n = 16

T(x) = 1.000 0.849 0.569 0.204 -0.191 -0.556 -0.836 -0.987 -0.987 -0.836 -0.556 -0.191 0.204 0.569 0.849 1.00
0
L1 = 0.149E+00
L2 = 0.202E-02
L∞ = 0.192E-01
R = 0.118E-15

For number of grid points, n = 32

T(x) = 1.000 0.962 0.886 0.777 0.638 0.475 0.293 0.100 -0.097 -0.290 -0.471 -0.635 -0.774 -0.883 -0.958 -0.99
7 -0.997 -0.958 -0.883 -0.774 -0.635 -0.471 -0.290 -0.097 0.100 0.293 0.475 0.638 0.777 0.886 0.962 1.000
L1 = 0.740E-01
L2 = 0.249E-03
L∞ = 0.482E-02
R = 0.656E-15

For number of grid points, n = 64

T(x) = 1.000 0.990 0.971 0.943 0.905 0.859 0.804 0.742 0.673 0.597 0.515 0.428 0.338 0.244 0.147 0.050 -0.0
48 -0.146 -0.243 -0.337 -0.427 -0.514 -0.596 -0.671 -0.741 -0.803 -0.858 -0.904 -0.942 -0.970 -0.989 -0.999 -0.
999 -0.989 -0.970 -0.942 -0.904 -0.858 -0.803 -0.741 -0.671 -0.596 -0.514 -0.427 -0.337 -0.243 -0.146 -0.048 0.
050 0.147 0.244 0.338 0.428 0.515 0.597 0.673 0.742 0.804 0.859 0.905 0.943 0.971 0.990 1.000
L1 = 0.395E-01
L2 = 0.373E-04
L∞ = 0.122E-02
R = 0.663E-06

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on June 10, 2021.
!This program solves a linear/non-linear equation using finite volume method
!and was written as a solution to AE8112 PS1 q3b
!**************************End Header*****************************************
program Vfinit_pBi_CGSTAB
    !Variable declaration
    implicit none
    DOUBLE PRECISION, dimension(:,:), ALLOCATABLE:: Ag
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE:: bg, Ti, Te
    DOUBLE PRECISION, PARAMETER :: Pi = 4*atan(1.0) !pi parameter definition
    DOUBLE PRECISION  :: L1, L2, Linf, Rs, dx
    integer :: N, ii
    N=64 !Number of control volumes
    ALLOCATE(Ag(n,n),bg(n),Ti(n),Te(n)) !array allocation
   call creat_eTDMA(Ag,bg,Te,dx)
   ! for the initial guess
    do ii = 1, n
        Ti(ii)=bg(ii)/Ag(ii,ii)
    end do
    call Bi_CGSTAB_P(Ag,bg,Ti,ii)
   !Calculating the error terms
    L1=sum(dabs(Ti(1:n)-Te(1:n)))
    L2=sum((Ti(1:n)-Te(1:n))**2)
    Linf=maxval(dabs(Ti(1:n)-Te(1:n)))

    Rs=sqrt(sum((bg-matmul(Ag,Ti))**2))/N !Calculates the linear system residual term
    !Printing our results
    print 3, 'For number Iteration i =',ii
    write(*,1)  "T(x) = ",Ti
    write(*,2) "L1 = ", L1
    write(*,2) "L2 = ", L2
    write(*,2) "L∞ = ", Linf
    write(*,2) "R = ", Rs
    1 format(a6,64f8.3)
    2 format(a7,E10.3)
    3 format(a30,i3)

contains
!*****************************************************************************
subroutine creat_eTDMA(Ag1,bg1,Te1,dx1)
    implicit none
    DOUBLE PRECISION, dimension(n,n), INTENT(OUT):: Ag1
    DOUBLE PRECISION, DIMENSION(n), INTENT(OUT):: bg1, Te1
    DOUBLE PRECISION, INTENT(OUT) :: dx1
    DOUBLE PRECISION  :: aa, ap, xi
    integer :: i
!!This subroutine generates the tri-diagonal matric using our derived equations

    !Initializing Variables
    Ag1=0
    xi=0
        dx1=2*pi/n
        aa=1/(dx1)
        ap=aa+aa
    do i = 1, n  !This loop Matrix composition of the given problem
        xi=xi+dx1
        !for the the first gridpoint, applying B.Cs T(0)=1
```

```
        if ( i==1 ) then
            Ag1(i,i)=-aa-2*aa
            Ag1(i,i+1)=aa
            bg1(i)=-2*aa-cos(dx1/2)*dx1
            Te1(i)=cos(dx1/2) !This is T_exact from our analytical solution
        !for the the intermediate gridpoint
        else if ( i<n ) then
            Ag1(i,i-1)=aa
            Ag1(i,i)=-ap
            Ag1(i,i+1)=aa
            bg1(i)=-cos(xi-dx1/2)*dx1
            Te1(i)=cos(xi-dx1/2)
        !for the the last gridpoint, applying B.Cs T(2pi)=1
        else
            Ag1(n,n-1)=aa
            Ag1(n,n)=-2*aa-aa
            bg1(n)=-2*aa-cos(xi-dx1/2)*dx1
            Te1(n)=cos(xi-dx1/2)
        end if
    end do

end subroutine creat_eTDMA
!*****************************************************************************************
!*****************************************************************************************
subroutine Bi_CGSTAB_P(A,b,x,i)
    implicit none
    DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: x
    DOUBLE PRECISION, dimension(n,n) :: A, K
    DOUBLE PRECISION, dimension(n), INTENT(IN):: b
    DOUBLE PRECISION, DIMENSION(n) ::   p, r, r0, y, z, v, s, t
    DOUBLE PRECISION :: rho0, rho, w, alpha, beta, r_check
    INTEGER :: i
!!This subroutine solves a tri-diagonal linear system using The Preconditioned Bi_CGSTAB Algorithm
!!by Van Der Vorst

    !Variable initialization
    r0=b-matmul(A,x); r=r0
    w=1; alpha=1; rho0=1; r_check=1
    v=0; p=0;  k=0
    !For the inverse of K
    do i = 1, n
        k(i,i)=1/A(i,i)
    end do
    i=0
    open(1, file = 'PS2_Q3b.txt', status = 'unknown')
    !main algorithm loop
    do while (r_check>=0.663E-6)
        i=i+1
        rho=dot_product(r0,r)
        beta=(rho/rho0)*(alpha/w)
        rho0=dot_product(r0,r)
        p=r+beta*(p-w*v)
        y=matmul(K,p)
        v=matmul(A,y)
        alpha=rho/dot_product(r0,v)
        s=r-alpha*v
        z=matmul(K,s)
        t=matmul(A,z)
```

```
        w=dot_product(matmul(K,t),matmul(K,s))/dot_product(matmul(K,t),matmul(K,t))
        x=x+alpha*y+w*z
        r=s-w*t
        r_check=norm2(r)
        write(1,*) i, r_check !writes the results to .txt file
    end do
    close(1)
end subroutine Bi_CGSTAB_P
!*****************************************************************************************

end program Vfinit_pBi_CGSTAB
```

For number Iteration i = 34

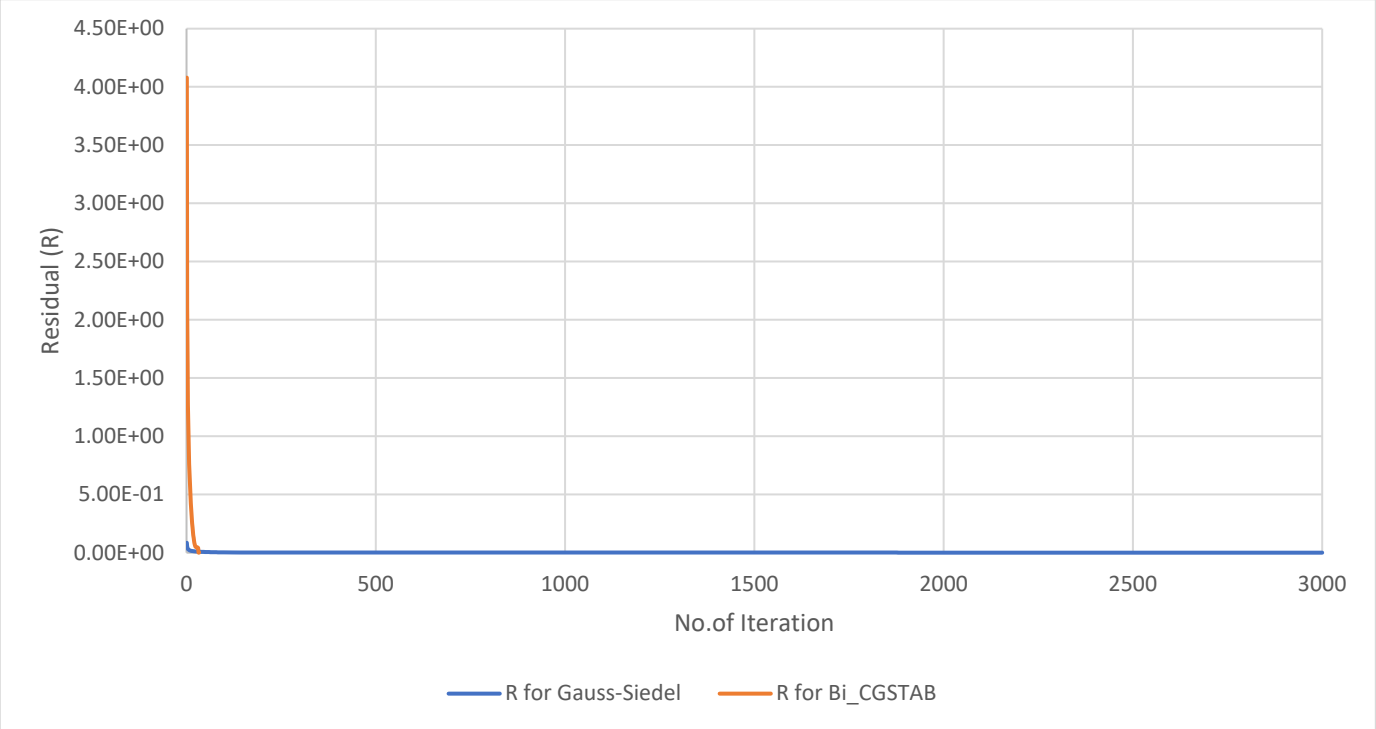
T(x) = 1.000 0.990 0.971 0.943 0.905 0.859 0.804 0.742 0.673 0.597 0.515 0.428 0.338 0.244 0.147 0.050 -0.049 -0.146 -0.243 -0.337 -0.427 -0.514 -0.596 -0.672 -0.741 -0.803 -0.858 -0.904 -0.942 -0.970 -0.990 -0.999 -0.999 -0.990 -0.970 -0.942 -0.904 -0.858 -0.803 -0.741 -0.672 -0.596 -0.514 -0.427 -0.337 -0.243 -0.146 -0.049 0.050 0.147 0.244 0.338 0.428 0.515 0.597 0.673 0.742 0.804 0.859 0.905 0.943 0.971 0.990 1.000

L1 = 0.369E-01

L2 = 0.310E-04

L∞ = 0.120E-02

R = 0.204E-09



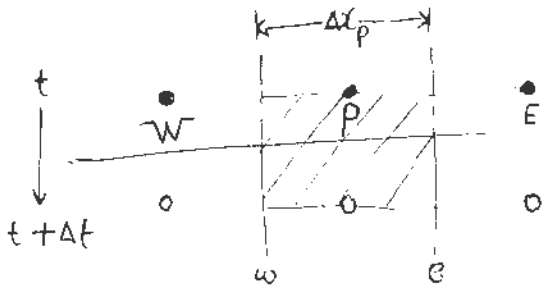
Q3.b| using the Preconditioned Bi-CGSTAB method, I was able to get better linear system residual term(R) at just 34 iterations.

>> And from the plot shown above, I observed that the preconditioned ~~Bi-STAB~~ Bi-CGSTAB yields much lower residual term(R), rapidly at a given short range of ~~any given~~ iteration, Hence it minimizes the number of iteration required for convergence, shown as almost a vertical straight line.

Q4.a| Given an unsteady thermal diffusion eqn

$$\alpha \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

using finite volume method with O'Brien's method, ie., $\theta = 1$



We have that $$\left(T_p^{n+1} - T_p^{n}\right)\Delta x_p = \alpha \Delta t \left[\left.\frac{dT}{dx}\right|_e - \left.\frac{dT}{dx}\right|_w\right]^{t+\Delta t} \quad \text{---(4.1)}$$

If we assume linear profiles for $\frac{dT}{dx}$, we can re-arrange the above, to get,

Q4.a) continued)

$$\left(\frac{\Delta x_p}{\Delta t}+\frac{\alpha}{\Delta x_e}+\frac{\alpha}{\Delta x_w}\right)T_p^{n+1}-\frac{\alpha T_E^{n+1}}{\Delta x_e}-\frac{\alpha T_W^{n+1}}{\Delta x_w}=\frac{\Delta x_p T_p^{n}}{\Delta t}$$

Since we are considering an equispaced grid

let, $r=\frac{\Delta t}{\Delta x^2}$, the above will become,

$$\boxed{(1+\alpha 2r)T_p^{n+1}-\alpha r T_E^{n+1}-\alpha r T_W^{n+1}=T_p^{n}}$$

where, thermal Diffusivity $\alpha=\frac{\lambda}{\rho c_p}$

$$\alpha=\frac{12}{(8933)(385)}=3.48918\times10^{-6}\ m^2/s$$

>> For the first C.V, we can apply the B.C of

$$T(x,0)=T_i^{o}=0^{\circ}C=273.15K$$

and

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} \text{ or } \left.\frac{\partial T}{\partial x}\right|_{w}=\frac{-h}{\lambda}\left(T_\infty-T(0,t)\right) \quad \text{——(4.2)}$$

where, $T_\infty=25^{\circ}C+273.15=298.15K$

we also know that

$$\left.\frac{\partial T}{\partial x}\right|_{w}=\frac{T_p^{n+1}-T_w^{n+1}}{\Delta x_{w/2}}$$

This means that $\frac{2\left(T_p^{n+1}-T_w^{n+1}\right)}{\Delta x}=\frac{-h}{\lambda}\left(T_\infty-T_w^{n}\right)$

solving for T_w^{n+1}, we get,

$$\boxed{T_w^{n+1}=\frac{\cancel{2}}{\cancel{\Delta x}}T_p^{n+1}+\frac{h\Delta x}{2\lambda}\left(T_\infty-T_w^{n}\right)}$$

Q4.a| continued|

Now substituting eqn (4.2) into eqn(4.1) will yield,

$$\left(T_P^{n+1} - T_P^{n}\right)\Delta x_P = \alpha \Delta t\left[\frac{T_E^{n+1} - T_P^{n+1}}{\Delta x_e} + \frac{h}{\lambda}\left(298.15 - T_w^{n}\right)\right]$$

So that at our first time step t=0,

$$T_w = T(0,0) = 273.15\,K$$

we can also re-write the above as,

$$\boxed{(1+\alpha r)T_P^{n+1} - \alpha r T_E^{n+1} = \frac{\alpha \Delta t h}{\lambda \Delta x}\left(298.15 - T_w^{n}\right) + T_P^{n}}$$

» Next, for the last C.V. we can apply the B.C of

$$\left.\frac{\partial T}{\partial x}\right|_{x=25cm} \text{ or } \left.\frac{\partial T}{\partial x}\right|_{e} = 0$$

So that our eqn (4.1) becomes,

$$\left(T_P^{n+1} - T_P^{n}\right)\Delta x_P = \alpha \Delta t\left[-\left(\frac{T_P^{n+1} - T_W^{n+1}}{\Delta x_w}\right)\right]$$

we can also write this as,

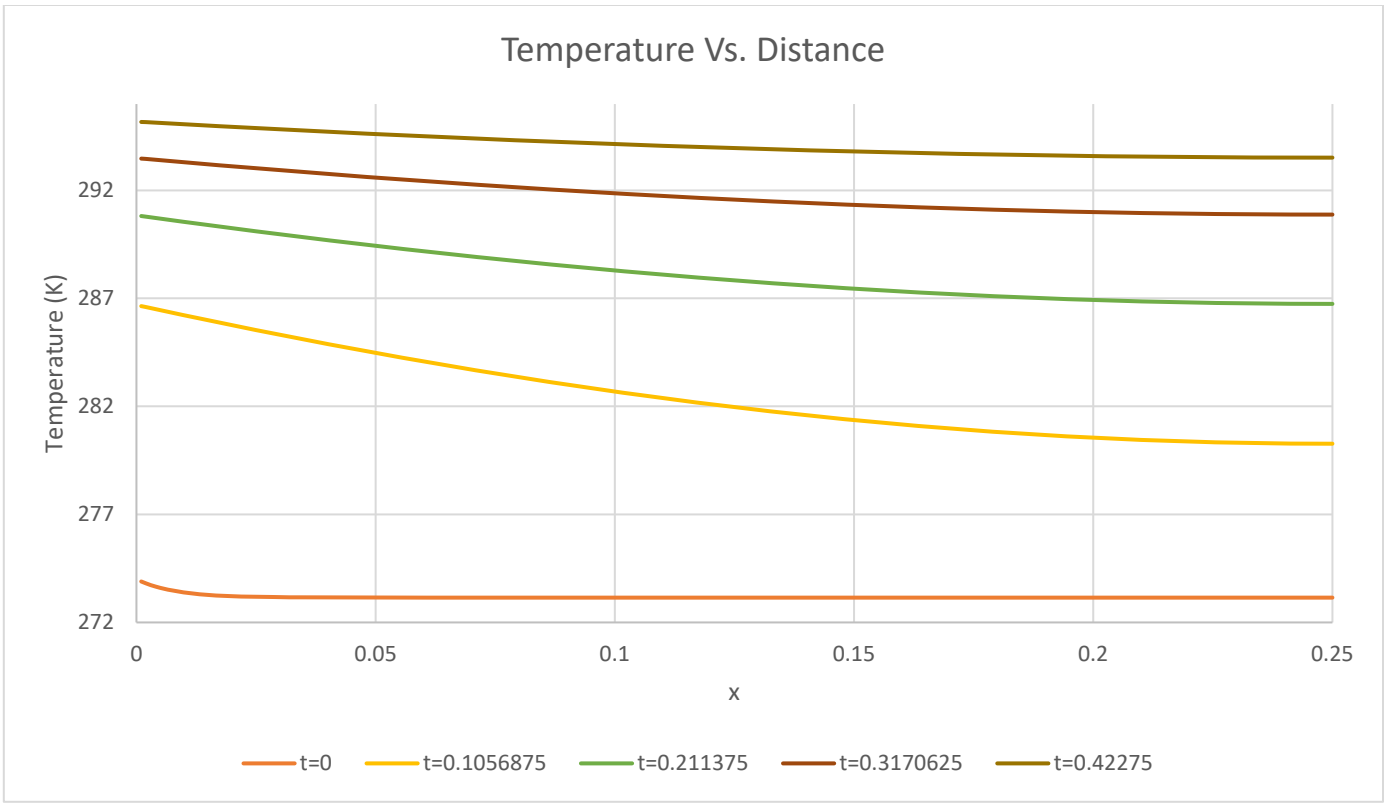
$$\boxed{(1+\alpha r)T_P^{n+1} - \alpha r T_W^{n+1} = T_P^{n}}$$

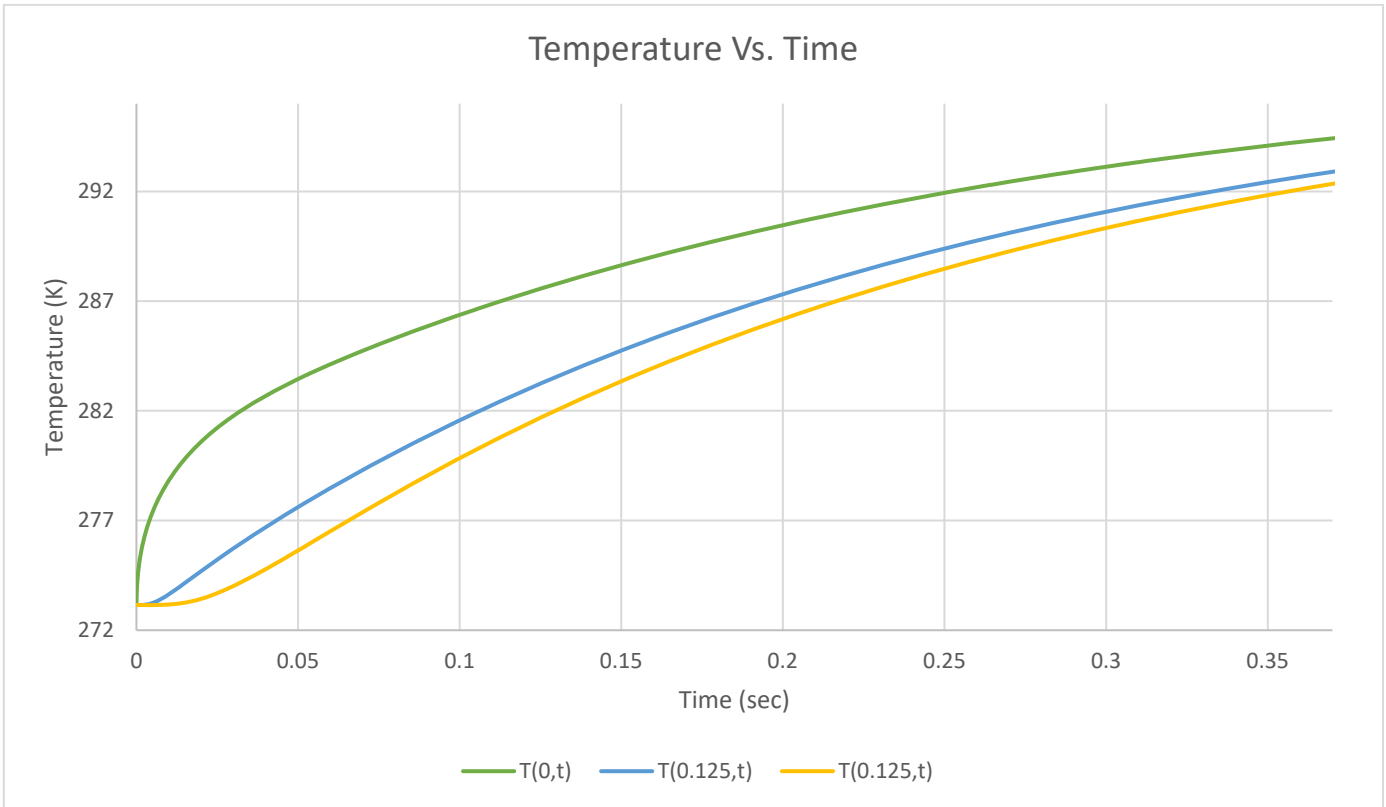
These equations in boxes will be used in our Fortran-90 code, as shown below;

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on June 10, 2021.
!This program solves an unsteady linear/non-linear equation using finite volume method
!and was written as a solution to AE8112 PS1 q4a
!**************************End Header*****************************************
program unsteady_Vfinite
    implicit none
    !Variable declaration
    DOUBLE PRECISION, dimension(:), ALLOCATABLE :: Ti, b_nth
    DOUBLE PRECISION, PARAMETER :: lamda=12, rho=8933, Cp=385, h=50, L=0.25
    DOUBLE PRECISION :: T_wall, dt, tt
    INTEGER :: N
    N=10*25 !Number of Control Volume
    dt=16.91     !Our choosen timestep (del_t)
    ALLOCATE(Ti(n),b_nth(n))
    Ti=273.15        !Initial Temperature at t=0
    T_wall=273.15     !Initial Temperature at x=0 and t=0
    call unsteady(b_nth,Ti,T_wall, tt) !solves the unsteady system
    Print *, "t_total = ", (tt-dt)*1D-5,"sec"

    contains
!*********************************************************************************
    subroutine unsteady(b, Tp, Tww, t)
        DOUBLE PRECISION, dimension(n) :: b, e, f, g, Tp, xi
        DOUBLE PRECISION :: aa, dx, r, Tww, alpha, t
        integer  :: i, j
    !!This subroutine solves unsteady thermal problem using O'Brien's method

        open(1, file = 'PS2_Q4a.txt', status = 'unknown')
        !Variable initialization
        t=0; e=0; g=0
        dx=L/n
        r=dt/dx**2
        alpha=lamda/(rho*Cp) !Thermal diffusivity
        aa=h*dx/(2*lamda)
        do while (Tww<0.99*298.15) !Loop for time step
            Tww=Tp(1)+aa*(298.15-Tww) !Temperature at the surface from our formulation
            do i = 1,n !Loop for control volume
                if ( i==1 ) then
                    g(1)=-alpha*r
                    f(1)=1+alpha*r
                    b(1)=alpha*dt*h*(298.15-Tww)/(lamda*dx)+Tp(1) !The RHS of our formulation
                    xi(1)=dx/2
                else if ( i<n ) then
                    g(i)=-alpha*r
                    f(i)=1+2*alpha*r
                    e(i)=-alpha*r
                    b(i)=Tp(i)
                    xi(i)=xi(i-1)+dx
                else
                    f(n)=1+alpha*r
                    e(n)=-alpha*r
                    b(n)=Tp(n)
                    xi(i)=xi(i-1)+dx
                end if
            end do
            !For computational efficiency, the A tri-diagonal matrix is split to 3 vectors
            call tdma(e,f,g,b,Tp) !Solves the linear system at each time-step
```

```
            if ( t==0 ) then
                write(1,3) "Time(sec)", "T(0,t)", (xi(j), j=1,n) !writes the result's title to .txt file
            end if
            write(1,4) t*1D-5, Tww, (Tp(j), j=1,n) !writes the results to .txt file
            t=t+dt
        end do
        3 format(a9,3x,a6,250(2x,f6.3))
        4 format(f9.7,3x,f7.3,250(1x,f7.3))
        close(1)
    end subroutine unsteady
!*****************************************************************************
!*****************************************************************************
    subroutine tdma(e, f, g, b1, x)
        DOUBLE PRECISION, dimension(n), INTENT(IN):: b1
        DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: x
        DOUBLE PRECISION, DIMENSION(n):: b, e, f, g
        INTEGER :: k
    !!This subroutine solves a tri-diagonal linear system using the Thomas Algorithm

        b=b1
       !Decomposition
        do k = 2,n
            e(k) = e(k)/f(k-1)
            f(k) = f(k) - e(k)*g(k-1)
        end do
       !Forward Substitution
        do k = 2,n
            b(k) = b(k) - e(k)*b(k-1)
        end do
       !Backward Substitution
        x(n)=b(n)/f(n)
        do k = n-1,1,-1  !(step size of -1)
            x(k) = (b(k) - g(k)*x(k+1))/f(k)
        end do
    end subroutine tdma
!*****************************************************************************
end program unsteady_Vfinite
```

```
t_total = 0.42274999618530279 sec
```


Temperature Vs. Distance
Temperature (K)
292
287
282
277
272
0
0.05
0.1
0.15
0.2
0.25
x
t=0
t=0.1056875
t=0.211375
t=0.3170625
t=0.42275


Temperature Vs. Time
Temperature (K)
292
287
282
277
272
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time (sec)
T(0,t)
T(0.125,t)
T(0.125,t)

Q4.a) ≫ From the first plot, if we assume a rod ($L = 0.25m$), then we can see that at $t = 0$, the temperature diffusion along the rod from the inserted surface, is almost linear. But the temperature at this surface increase with time, as well as the temperature along the rod (with unique temperature gradients, at different time step).

≫ Also Looking at the second plot, we can observe that at the surface ($T(0,t)$), due to our assumed convective formulation, which contains temperature difference ($T_\infty - T_w^n$).

This means that at the surface the temperature will initially rapidly increase, and with time it will gradually tend to a straight line (i.e. There's no more convective heat transfer, so that $T_w^{n+1} = T_p^{n+1}$). But this is different at other locations along the rod, which depends on the heat diffusion from the surface, and will also tend to a straight line with time (if there's no heat dissipation).