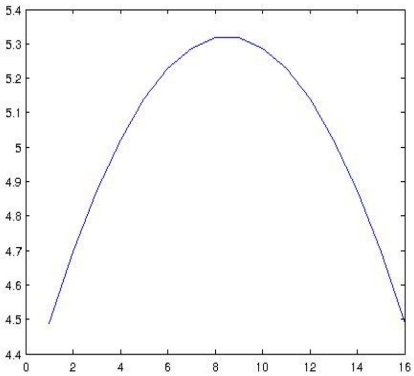
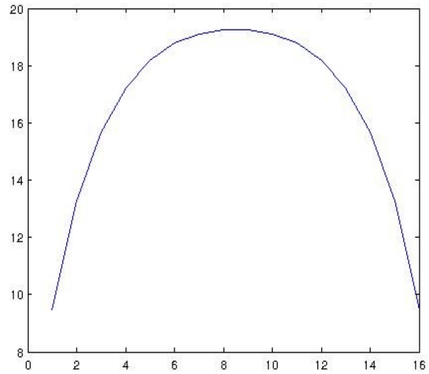
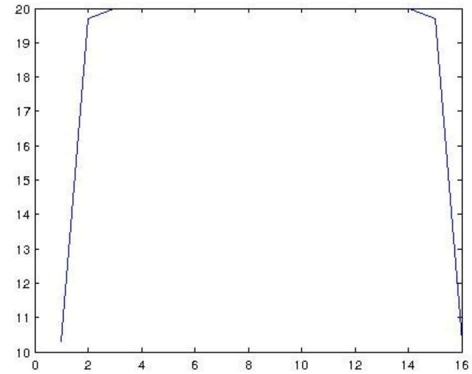
Testing the CFEM 1D diffusion equation code

1) **Trend test**: The length of the slab will be increased (with vacuum boundaries on both ends). We must see the solution approaching infinite medium solution (flat line with magnitude of S/sigma_a for a uniform source). Choose a problem with D(i) = 2/3, sigma_a(i) = 0.3 and q(i) = 6. Infinite medium solution is uniform 20. L = 1, 10, 50 (16 nodes for each length).







Clearly, we observe the right trend form the above figures.

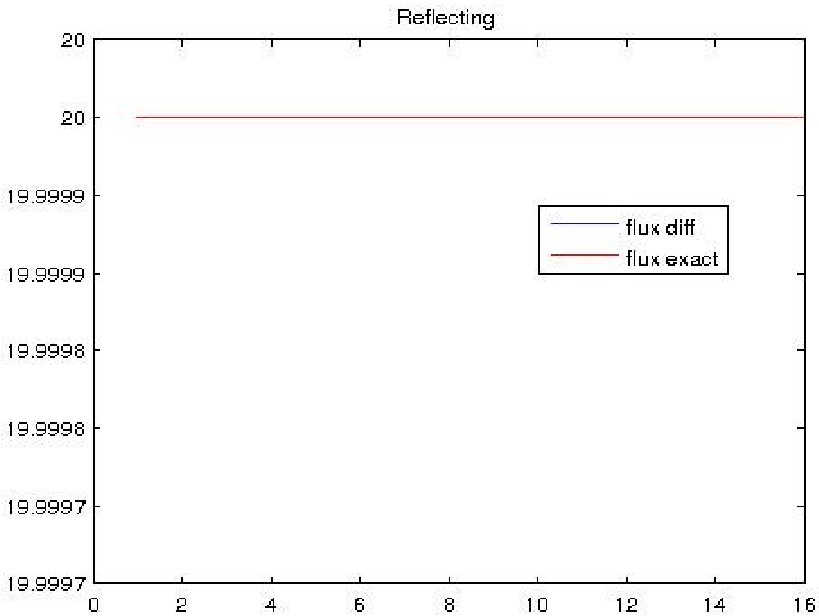
2) **Symmerty test**: The flux should be symmetric about the center with symmetric source and boundaries. In all of the figures above, we observe symmetry so the code passes the symmetry test. The flux profile for L = 1 (16 nodes) is as follows:

flux

4.48744392
4.69637299
4.87468529
5.02273750
5.14082479
5.22918415
5.28799248
5.31736755
5.31736755
5.28799343
5.22918558
5.14082623
5.02273893
4.87468719
4.69637489
4.48744535

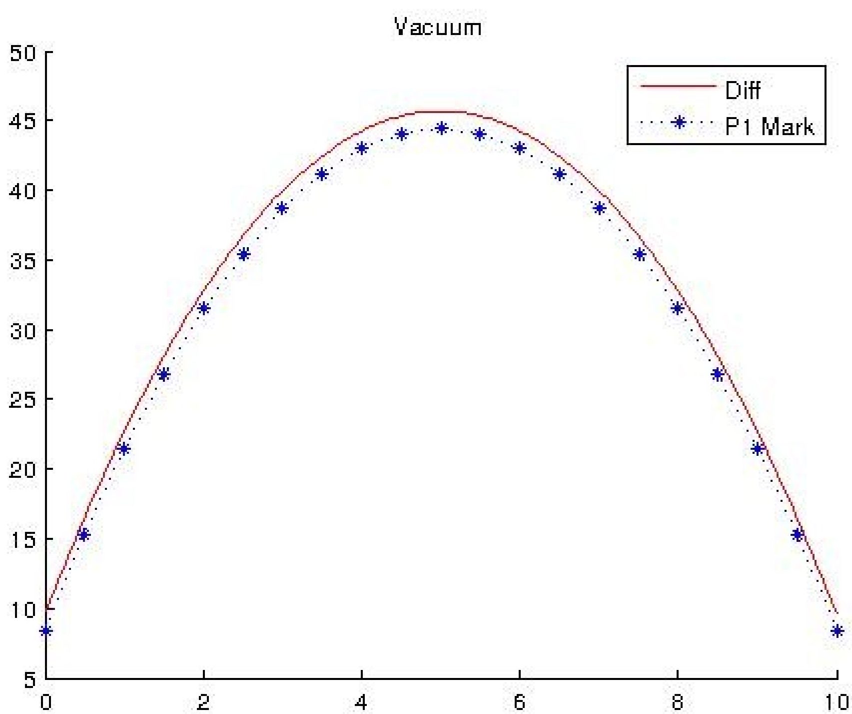
3) **Method of exact solutions**: For vacuum boundaries we compare our solution to the P1 Marshak boundary problem and for the reflecting boundaries we compare solution to the infinite medium solution (S/sigma_a). For vacuum boundaries we use the example homework problem and for reflecting boundaries we use the problem described in (1).

Reflecting boundaries:



Vacuum Boundaries:

I did something to my Pn Marshak code and it isn't giving me the right result (I don't remember what I did but it seems to be broken). I have compared it to P1 Mark. The diffusion code should give solution that is slightly higher than P1 Mark as seen in previous homework for P1 Marshak.



We observe all the expected behavior so it can be said with fair certainty that the diffusion code gives correct flux profiles.

TDMA (solver from wikipedia)

```
! This code was downloaded from wikipedia (Solver tested on a 3 by 3 matrix against matlab solve)

subroutine tdma(n,a,b,c,d,x)
         implicit none
    integer, intent(in) :: n
    real, intent(in) :: a(n), c(n)
    real, intent(inout), dimension(n) :: b, d
         real, intent(out) :: x(n)
         !  --- Local variables ---
         integer :: i
         real :: q
    !  --- Elimination ---
    do i = 2,n
      q = a(i)/b(i - 1)
      b(i) = b(i) - c(i - 1)*q
      d(i) = d(i) - d(i - 1)*q
    end do
    ! --- Backsubstitution ---
    q = d(n)/b(n)
    x(n) = q
    do i = n - 1,1,-1
      q = (d(i) - c(i)*q)/b(i)
      x(i) = q
    end do
    return
    end
end subroutine tdma
```

Diffusion code:

```
program DiffFEM

   ! Finite element diffusion equation matrix setup
   ! This code will generate the global stiffness matrix and load vector for steady state
   ! diffusion problem 1D 1G with finite element formulation with linear trial and test functions

   ! Initialize the problem first

   implicit none

   integer :: i
   integer :: n_elem = 155;           ! number of elements
   integer :: n_node = 156;      ! number of nodes
   real, dimension(155) :: h, d, sigma_a;   ! element length, diffusion coefficient (1/3sigma_t),
```

```
absorption xn, and source
    integer, dimension(2) :: BC;                ! boundary conditions
    real, dimension(156) :: low_lhs1, diag_lhs1, up_lhs1, low_lhs2, diag_lhs2, up_lhs2;
    real, dimension(156) :: low, diag, up, source, q, x;


    do i = 1, n_elem

        h(i) = 10.0/n_elem;
        d(i) = 1/3.0;
        sigma_a(i) = 0.001;
        q(i) = 1.0;

!        print*, h(i), d(i), sigma_a(i);

    end do

    q(n_node) = q(1);

    BC(1) = 1;                          ! Reflecting boundary - 1 for vacuum boundary
    BC(2) = 1;                          ! Vacuum Boundary - 0 for reflecting boundary

    ! First we generate the stiffness matrix - unlike finite volume method, we apply the
    ! boundary conditions directly to the stiffness matrix after generating the matrix

    ! We generate generate the stiffness matrix in 4 steps - 1) Generate LHS1 - leakage
    ! 2) LHS2 - absorption term, 3) add LHS1 and LHS 2, 4) Apply boundary conditions to the matrix
    ! For this Fortran program (unlike MatLab where sparse storage was used) we store the matrix as
    ! three vectors corresponding to each matrix - lower_diag, diag, upper_diag

    ! 1) Now generate  the LHS1 Matrix - Leakage term

    do i = 2, n_elem

        low_lhs1(i) = -d(i-1)/h(i-1);
        diag_lhs1(i) = d(i-1)/h(i-1) + d(i)/h(i);
        up_lhs1(i) = -d(i)/h(i);

    end do

    low_lhs1(1) = 0;
    diag_lhs1(1) = d(1)/h(1);
    up_lhs1(1) = -d(1)/h(1);

    low_lhs1(n_node) = -d(n_elem)/h(n_elem);
    diag_lhs1(n_node) = d(n_elem)/h(n_elem);
    up_lhs1(n_node) = 0;

    ! 2) Generate LHS2 Matrix - Absorption term
```

```
do i = 2, n_elem

    low_lhs2(i) = sigma_a(i-1)*h(i-1)/6;
    diag_lhs2(i) = (sigma_a(i-1)*h(i-1)/3) + (sigma_a(i)*h(i)/3);
    up_lhs2(i) = sigma_a(i)*h(i)/6;

end do

low_lhs2(1) = 0;
diag_lhs2(1) = sigma_a(1)*h(1)/3;
up_lhs2(1) = sigma_a(1)*h(1)/6;

low_lhs2(n_node) = sigma_a(n_elem)*h(n_elem)/6;
diag_lhs2(n_node) = sigma_a(n_elem)*h(n_elem)/3;
up_lhs2(n_node) = 0;

! 3) Add LHS1 and LHS 2 to form the LHS matrix

do i = 1, n_node

    low(i) = low_lhs1(i) + low_lhs2(i);
    diag(i) = diag_lhs1(i) + diag_lhs2(i);
    up(i) = up_lhs1(i) + up_lhs2(i);

end do

! 4) Now apply boundary conditions

! Left boundary

if (BC(1) == 1) then

    diag(1) = diag(1) + 0.5; ! Vacuum boundary

else

    diag(1) = diag(1) ! Reflecting boundary

end if

! Right boundary

if (BC(2) == 1) then

    diag(n_node) = diag(n_node) + 0.5; ! Vacuum boundary

else
```

```
    diag(n_node) = diag(n_node); ! Reflecting boundary

end if


! Now we print the stiffness matrix for testing

print*, 'matrix'

do i = 1, n_node

    print*, low(i), diag(i), up(i);

end do

! Now generate the load vector

do i = 2, n_elem

    source(i) = 0.5*(q(i-1)*h(i-1) + q(i)*h(i));

end do

source(1) = 0.5*q(1)*h(1);
source(n_node) = 0.5*q(n_node)*h(n_elem);

print*, 'source';

do i = 1, n_node

    print*, source(i);

end do



! Now we solve the system of equations using the tdma.f90 subroutine

call tdma(n_node, low, diag, up, source, x)

print*,'flux';

do i= 1, n_node

    print*, x(i);

end do
end program DiffFEM
```