```
!*******************************************************
!          无网格伽辽金法程序主要计算模块简单例子
!       Kernel modules for element-free Galerkin Method
!            By Lu Xinzheng of Tsinghua University
!*******************************************************

module TypeDef
    use Lxz_Tools
    implicit none;
    !integer,parameter :: MLS_m=6; !定义基为二次基
    integer,parameter :: MLS_m=3; !定义基为1次基
    integer,parameter :: NInt=4;  !定义高斯积分点数目
    integer,parameter :: NDOF=2;  !定义自由度数目
    integer,parameter :: MLS_n=6;

    type :: typ_Kcol !定义变带宽总体刚度矩阵
        real(rkind),pointer :: row(:)
    end type typ_Kcol

    type :: typ_GValue !定义总体控制变量
        integer(ikind) :: NNode; !总节点数
        integer(ikind) :: NCell; !总积分区域数
        integer(ikind) :: NIntPoint; !总高斯积分点数
        integer(ikind) :: NPress; !总压力数
        integer(ikind) :: NSupport; !总位移约束数
        real(rkind)  :: dm;    !节点影响半径
        real(rkind)  :: c;     !权函数相对权重参数
        real(rkind)  :: Plenty !罚函数大小
    end type typ_GValue;

    type :: typ_Node; !定义节点类型
        real(rkind) :: X,Y; !节点坐标
        real(rkind) :: dX, dY; !节点位移
    end type typ_Node;

    type :: typ_IntPoint; !定义高斯积分点类型

        !以下为高斯点几何数据
        real(rkind) :: CenterX, CenterY; ! 中心点坐标
        real(rkind) :: Len, Wide; ! 高斯积分点的积分面积
        integer(ikind) :: NodeNo(MLS_n); ! 包含节点的编号

        !以下为移动最小二乘所需数据
        real(rkind)  :: a,b    !x,y的最大距离
        real(rkind)  :: dm;    !节点影响半径
        real(rkind)  :: c;     !权函数相对权重参数
        real(rkind)  :: Pasi(MLS_n); !形函数矩阵，维数是 Pasi(MLS_n)
        real(rkind)  :: Pasidx(MLS_n); !形函数对x求导，维数是 Pasidx(MLS_n)
        real(rkind)  :: Pasidy(MLS_n); !形函数对y求导，维数是 Pasidy(MLS_n)

        !以下为高斯点力学数据
        real(rkind) :: Strain(3); !高斯点的应变 Strain=B*ui
        real(rkind) :: D(3,3);    !高斯点材料性质矩阵
        real(rkind) :: Stress(3); !高斯点的应力 Stress=D*Strain
        real(rkind) :: E,mu;
```

```
    end type typ_IntPoint;


    type :: typ_Load !定义节点荷载
        integer(ikind) :: NodeNo !高斯点编号
        real(rkind) :: Value(2) !荷载大小
    end type typ_Load

    type :: typ_Support !定义位移约束
        integer(ikind)  :: NodeNo !约束节点号
        real(rkind) :: Value(2) !位移大小
    end type typ_Support


!   contains


end module TypeDef

module MLS
    use Lxz_Tools;
    use TypeDef;
    implicit none;

    contains
    !SetIntPoint(IntPoint,Node) !设定积分区域内各个参数的数值

    subroutine SetIntPoint(IntPoint,Node) !设定积分区域内各个参数的数值
        type(typ_IntPoint) :: IntPoint;
        type(typ_Node) :: Node(:);
        integer(ikind) :: I,J; !循环变量
        real(rkind)  :: d,x1,y1,dx,dy; ! d节点和积分点之间的距离, x1,y1节点和积分点之间
x,y方向的距离
        real(rkind)  :: temp1; ! 中间变量
        real(rkind)  :: tempAdx(MLS_m,MLS_m),tempAdy(MLS_m,MLS_m) !对A求导时的中间变量
        real(rkind)  :: Ptemp1(1,MLS_m),Ptemp2(MLS_m,1); !为了向量相乘设定的二维数组中
间变量

        real(rkind)  :: P(MLS_m,MLS_n); !基函数数值, 维数应该是P(MLS_m,MLS_n)
        real(rkind)  :: Px(MLS_m);
        real(rkind)  :: Pdx(MLS_m); !基函数数值对x求导, 维数应该是Pdx(MLS_m)=(0,1,0)
        real(rkind)  :: Pdy(MLS_m); !基函数数值对y求导, 维数应该是Pdy(MLS_m)=(0,0,1)
        real(rkind)  :: A(MLS_m,MLS_m); !矩阵A
        real(rkind)  :: Adx(MLS_m,MLS_m); !矩阵A对x求导
        real(rkind)  :: Ady(MLS_m,MLS_m); !矩阵A对y求导
        real(rkind)  :: InvA(MLS_m, MLS_m); !A的逆矩阵
        real(rkind)  :: B(MLS_m,MLS_n); !矩阵B, 维数应该是B(MLS_m,MLS_n)
        real(rkind)  :: Bdx(MLS_m,MLS_n); !矩阵B对x求导, 维数应该是Bdx(MLS_m,MLS_n)
        real(rkind)  :: Bdy(MLS_m,MLS_n); !矩阵B对y求导, 维数应该是Bdy(MLS_m,MLS_n)
        real(rkind)  :: Weight(MLS_n); !权函数, 维数是Weight(MLS_n)
        real(rkind)  :: Weightdx(MLS_n); !权函数对x求导, 维数是Weightdx(MLS_n)
        real(rkind)  :: Weightdy(MLS_n); !权函数对y求导, 维数是Weightdy(MLS_n)

        P=0.0d0;
```

```
        B=0.0d0
        Bdx=0.0d0;
        Bdy=0.0d0;
        Weight=0.0d0;
        Weightdx=0.0d0;
        Weightdy=0.0d0;

        do I=1, MLS_n

            x1=Node(IntPoint%NodeNo(I))%X;
            y1=Node(IntPoint%NodeNo(I))%Y;
            ! 求节点和高斯积分点之间的距离
            d=sqrt( (x1-IntPoint%CenterX)**2 + (y1-IntPoint%CenterY)**2);
            !d=d/IntPoint%dm;

            ! 防止有点超出积分区
            if(d<IntPoint%dm) then

                ! 设定积分点P的值，二次基，参见李卧东论文公式(2.2b)
                Px=(/1.0d0,IntPoint%CenterX,IntPoint%CenterY/);
                !设定积分点关于x,y求导时的值，二次基，
                Pdx=(/0.0d0,1.0d0,0.0d0/);
                Pdy=(/0.0d0,0.0d0,1.0d0/);
                ! 设定节点二次基的值
                P(:,I)=(/1.0d0,x1,y1/);

                ! 计算各个节点的权重，参见李卧东论文公式(2.12)
                Weight(I)=(exp(-(d/IntPoint%c)**IntPoint%k)-exp(-(IntPoint%dm/IntPoint%
c)**IntPoint%k))&
                            /(1.0-exp(-(IntPoint%dm/IntPoint%c)**IntPoint%k));
                ! 计算各个节点权重的导数
                temp1=IntPoint%CenterX**2-2.0d0*IntPoint%CenterX*x1+x1**2+IntPoint%Cent
erY**2-2.0d0*y1*IntPoint%CenterY+y1**2;

                Weightdx(I)=-0.5d0*(temp1**0.5d0/IntPoint%c)**IntPoint%k*IntPoint%k*&
                    (2.0d0*IntPoint%CenterX-2*x1)*exp(-(temp1**0.5d0/IntPoint%c)**IntPo
int%k);
                Weightdx(I)=((Weightdx(I)/temp1))&
                    /(1.0d0-exp(-(IntPoint%dm/IntPoint%c)**IntPoint%k));

                Weightdy(I)=-0.5d0*(temp1**0.5d0/IntPoint%c)**IntPoint%k*IntPoint%k*&
                    (2.0d0*IntPoint%CenterY-2*y1)*exp(-(temp1**0.5d0/IntPoint%c)**IntPo
int%k);
                Weightdy(I)=((Weightdy(I)/temp1))&
                    /(1.0d0-exp(-(IntPoint%dm/IntPoint%c)**IntPoint%k));

            else
                Weight(I)=0;     Weightdx(I)=0; Weightdy(I)=0
            end if
        end do

        A=0;
        Adx=0; Ady=0;
        do I=1, MLS_n
            ! 得到P'
```

```
            do J=1,MLS_m;
                Ptemp1(1,J)=P(J,I);
            end do
            Ptemp2(:,1)=P(:,I);
            A=A+Weight(I)*matmul(Ptemp2,Ptemp1);
            B(:,I)=Weight(I)*P(:,I);
            Adx=Adx+Weightdx(I)*matmul(Ptemp2,Ptemp1);
            Ady=Ady+Weightdy(I)*matmul(Ptemp2,Ptemp1);
            Bdx(:,I)=Weightdx(I)*P(:,I);
            Bdy(:,I)=Weightdy(I)*P(:,I);
        end do;
        InvA=matinv(A);

        TempAdx=-matmul(InvA,matmul(Adx,InvA));
        TempAdy=-matmul(InvA,matmul(Ady,InvA));
        do I=1, MLS_n;
            IntPoint%Pasi(I)=0;
            IntPoint%Pasidx(I)=0;   IntPoint%Pasidy(I)=0;
            IntPoint%Pasi(I)=dot_product(Px,matmul(InvA,B(:,I)));

            IntPoint%Pasidx(I)=dot_product(Pdx,matmul(InvA,B(:,I)))+&
                dot_product(Px,matmul(TempAdx,B(:,I))+matmul(InvA,Bdx(:,I)));
            IntPoint%Pasidy(I)=dot_product(Pdy,matmul(InvA,B(:,I)))+&
                dot_product(Px,matmul(TempAdy,B(:,I))+matmul(InvA,Bdy(:,I)));
        end do;
        return
    end subroutine SetIntPoint

end module MLS

module M_Int !高斯积分区力学数据计算
    use Lxz_Tools;
    use TypeDef;
    use MLS;
    implicit none;

    contains
    !   GetD(IntPoint)  ! 计算材料性质矩阵D，注意：可以根据材料性质加以改变
    !   GetM_K(IntPoint)    ! 计算积分点的刚度矩阵

    subroutine GetD(IntPoint)  ! 计算材料性质矩阵D，注意：可以根据材料性质加以改变
        type(typ_IntPoint) :: IntPoint;
        IntPoint%D=0.0d0;
        !采用弹性平面应力
        IntPoint%D(1,:)=(/1.0d0,IntPoint%mu,0.0d0/);
        IntPoint%D(2,:)=(/IntPoint%mu,1.0d0,0.0d0/);
        IntPoint%D(3,:)=(/0.0d0,0.0d0,(0.5d0*(1.0d0-IntPoint%mu))/);
        IntPoint%D=IntPoint%E/(1.0d0-IntPoint%mu**2)*IntPoint%D
        return;
    end subroutine GetD;

    function GetM_B(IntPoint) result(M_B)
        type(typ_IntPoint) :: IntPoint;
        real(rkind) :: M_B(3,2*MLS_n);
        Integer(ikind) :: I
```

```
        M_B=0.0d0;
        do I=1,MLS_n
            M_B(1,2*I-1)=IntPoint%Pasidx(I);
            M_B(2,2*I)=IntPoint%Pasidy(I);
            M_B(3,2*I)=IntPoint%Pasidx(I);
            M_B(3,2*I-1)=IntPoint%Pasidy(I);
        end do
        return
    end function GetM_B

    function GetM_K(IntPoint) result(M_K)! 计算积分点的刚度矩阵
        type(typ_IntPoint) :: IntPoint
        real(rkind) :: M_B(3,2*MLS_n),M_K(2*MLS_n,2*MLS_n); !,M_B1(:,:)
        M_B=0.0d0;  M_K=0.0d0;
        M_B=GetM_B(IntPoint);
        M_K=matmul(transpose(M_B),matmul(IntPoint%D,M_B));
        M_K=M_K*IntPoint%Len*IntPoint%Wide;
        return
    end function GetM_K

    subroutine GetStress(IntPoint, Node)
        type(typ_IntPoint) :: IntPoint(:)
        type(typ_Node) :: Node(:)
        real(rkind) :: DispVecter(2*MLS_n)
        real(rkind) :: X
        integer(ikind) :: I,J;
        do I=1, size(IntPoint)
            do J=1,MLS_n
                X=Node(IntPoint(I)%NodeNo(J))%dX;
                DispVecter(J*2-1)=X;
                X=Node(IntPoint(I)%NodeNo(J))%dY;
                DispVecter(J*2)=X;
            end do
            IntPoint(I)%Strain=matmul(GetM_B(IntPoint(I)),DispVecter);
            IntPoint(I)%Stress=matmul(IntPoint(I)%D,matmul(GetM_B(IntPoint(I)),DispVect
er));
        end do
        return
    end subroutine GetStress

end module M_Int
```