buildModelProto (generic function with 1 method)

```julia
##########################build simulation model###############################
function buildModelProto(dimensionTmp::Integer, ρbTmp::Float64, cbTmp::Float64, ρpTmp::Float64, cpTmp::Float64, ωbTmp::Float64, pNoTmp::Integer, RsTmp::Float64, parPos::Matrix{Float64})
parPosTmp=deepcopy(parPos);
######################set incident waves#######################################
incAmpTmp = 3000.0; #amplitude of incident beam
incDirTmp1 = [0.0, 0.0, 1.0];   #set incident direction 1
incDirTmp2 = [0.0, 0.0, -1.0];  #set incident direction 2
incPosTmp1 = [0.0, 0.0, -1];    #original position of incident wave 1
incPosTmp2 = [0.0, 0.0, 1]; #original position of incident wave 2
bgMediumTmp = Acoustic(dimensionTmp; ρ = ρbTmp, c = cbTmp); #build background acoustic model
waveTmp = plane_source(bgMediumTmp; amplitude = incAmpTmp, direction = incDirTmp1, position = incPosTmp1)+plane_source(bgMediumTmp; amplitude = -incAmpTmp, direction = incDirTmp2, position = incPosTmp2); #build incident plane wave
#########################incident wave done########################################
#########################set particles#############################################
parMediumTmp = Acoustic(dimensionTmp; ρ = ρpTmp, c = cpTmp);    #build the acoustic model in particles
particlesTmp=Array{Particle{dimensionTmp, Acoustic{Float64, dimensionTmp}, Sphere{Float64, dimensionTmp}}}(undef, 0);  #define a null array to store particles model
#build particle set
for iTmp in 1:pNoTmp
    parShapeTmp=Sphere(parPosTmp[iTmp,:],RsTmp);
    particlesTmp=push!(particlesTmp,Particle(parMediumTmp,parShapeTmp));
end
##########################particles done############################################
simModelTmp=FrequencySimulation(particlesTmp,waveTmp);#build simulation model
return simModelTmp
end
```

getCoefProto (generic function with 1 method)

```julia
########################get expansion coefficients###########################
function getCoefProto(ωbTmp::Float64, modelTmp::FrequencySimulation, coefOrderTmp::Integer)
simModelTmp=modelTmp;
coefDataTmp=basis_coefficients(simModelTmp,ωbTmp,basis_order=coefOrderTmp);#store the expansion coefficients
return coefDataTmp
end
```

```
pProto (generic function with 1 method)
```

```julia
##########define function to calculate pressure in position [x,y,z]###########
function pProto(x::Float64, y::Float64, z::Float64, ωbTmp::Float64, modelTmp::FrequencySimulation, coefData::Matrix{ComplexF64})
coefDataTmp=deepcopy(coefData);
simModelTmp=modelTmp;
pNoTmp=length(simModelTmp.particles);
dimensionTmp=typeof(simModelTmp.source.medium).parameters[2];
cbTmp=Float64(simModelTmp.source.medium.c);     #make soundspeed a real number
coefOrderTmp=Int(sqrt(length(coefDataTmp[:,1])))-1;
parPosTmp=Matrix{Float64}(undef,pNoTmp,dimensionTmp);
k=ωbTmp/cbTmp;
for iTmp in 1:pNoTmp
    for jTmp in 1:dimensionTmp
        parPosTmp[iTmp,jTmp]=simModelTmp.particles[iTmp].shape.origin[jTmp];
    end
end
pField=0.0+0.0*im;
r=Array{Float64}(undef,pNoTmp);
θ=Array{Float64}(undef,pNoTmp);
ϕ=Array{Float64}(undef,pNoTmp);
for iTmp in 1:pNoTmp
    parPosTmp2=deepcopy(parPosTmp[iTmp,:]);
    xx=x-parPosTmp2[1];
    yy=y-parPosTmp2[2];
    zz=z-parPosTmp2[3];
    r[iTmp]=sqrt(xx*xx+yy*yy+zz*zz);
    θ[iTmp]=acos(zz/max(r[iTmp],0.000000001));
    if yy==0&&xx>0
        ϕ[iTmp]=0.0;
    elseif yy==0&&xx<0
        ϕ[iTmp]=π;
    else
        ϕ[iTmp]=(1-sign(yy))*π+sign(yy)*acos(xx/max(sqrt(xx*xx+yy*yy),0.00000001));
    end
    for nTmp in 0:coefOrderTmp
        for mTmp in -nTmp:nTmp
            pField+=coefDataTmp[nTmp*nTmp+nTmp+mTmp+1,iTmp]*hk(nTmp,k*r[iTmp])*ymn(nTmp,mTmp,θ[iTmp],ϕ[iTmp]);
        end
    end
end
return pField+simModelTmp.source.field([x,y,z],ωbTmp);
end
```

```
vProto (generic function with 1 method)
```

```julia
###########Calculate velocity field by using five-point stencil##############
function vProto(x::Float64, y::Float64, z::Float64, ωbTmp::Float64, modelTmp::FrequencySimulation, coefData::Matrix{ComplexF64})
coefDataTmp=deepcopy(coefData);
simModelTmp=modelTmp;
dimensionTmp=typeof(simModelTmp.source.medium).parameters[2];
δh=0.00005;
ρbTmp=simModelTmp.source.medium.ρ;
vField=Array{ComplexF64}(undef,dimensionTmp);
pX2=pProto(x+2*δh,y,z,ωbTmp,simModelTmp,coefDataTmp);
pX1=pProto(x+δh,y,z,ωbTmp,simModelTmp,coefDataTmp);
pXN2=pProto(x-2*δh,y,z,ωbTmp,simModelTmp,coefDataTmp);  #N denotes negative
pXN1=pProto(x-δh,y,z,ωbTmp,simModelTmp,coefDataTmp);
pY2=pProto(x,y+2*δh,z,ωbTmp,simModelTmp,coefDataTmp);
pY1=pProto(x,y+δh,z,ωbTmp,simModelTmp,coefDataTmp);
pYN2=pProto(x,y-2*δh,z,ωbTmp,simModelTmp,coefDataTmp);
pYN1=pProto(x,y-δh,z,ωbTmp,simModelTmp,coefDataTmp);
pZ2=pProto(x,y,z+2*δh,ωbTmp,simModelTmp,coefDataTmp);
pZ1=pProto(x,y,z+δh,ωbTmp,simModelTmp,coefDataTmp);
pZN2=pProto(x,y,z-2*δh,ωbTmp,simModelTmp,coefDataTmp);
pZN1=pProto(x,y,z-δh,ωbTmp,simModelTmp,coefDataTmp);
vField[1]=-im/ρbTmp/ωbTmp*(-pX2+8*pX1-8*pXN1+pXN2)/12/δh;
vField[2]=-im/ρbTmp/ωbTmp*(-pY2+8*pY1-8*pYN1+pYN2)/12/δh;
vField[3]=-im/ρbTmp/ωbTmp*(-pZ2+8*pZ1-8*pZN1+pZN2)/12/δh;
return vField
end
```