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

```julia
###########calculate time average of stress tensor################
function T(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];
TDataTmp=Matrix{Float64}(undef,dimensionTmp,dimensionTmp);  #matrix to store stress tensor in the form:
##############################
######### Txx Txy Txz ########
######### Tyx Tyy Tyz ########
######### Tzx Tzy Tzz ########
##############################
ρbTmp=simModelTmp.source.medium.ρ;
cbTmp=simModelTmp.source.medium.c;
vTmp=vProto(x,y,z,ωbTmp,simModelTmp,coefDataTmp)
vxTmp=vTmp[1];
vyTmp=vTmp[2];
vzTmp=vTmp[3];
cjvxTmp=conj(vxTmp);
cjvyTmp=conj(vyTmp);
cjvzTmp=conj(vzTmp);
pTmp=pProto(x,y,z,ωbTmp,simModelTmp,coefDataTmp);
cjpTmp=conj(pTmp);
vSqu=real(vxTmp*cjvxTmp+vyTmp*cjvyTmp+vzTmp*cjvzTmp);
pSquCoeff=real(pTmp*cjpTmp/2/ρbTmp/cbTmp/cbTmp);
TDataTmp[1,1]=0.5*(ρbTmp*(real(vxTmp*cjvxTmp)-0.5*vSqu)+pSquCoeff);
TDataTmp[1,2]=0.5*ρbTmp*real(vxTmp*cjvyTmp);
TDataTmp[1,3]=0.5*ρbTmp*real(vxTmp*cjvzTmp);
TDataTmp[2,1]=0.5*ρbTmp*real(vyTmp*cjvxTmp);
TDataTmp[2,2]=0.5*(ρbTmp*(real(vyTmp*cjvyTmp)-0.5*vSqu)+pSquCoeff);
TDataTmp[2,3]=0.5*ρbTmp*real(vyTmp*cjvzTmp);
TDataTmp[3,1]=0.5*ρbTmp*real(vzTmp*cjvxTmp);
TDataTmp[3,2]=0.5*ρbTmp*real(vzTmp*cjvyTmp);
TDataTmp[3,3]=0.5*(ρbTmp*(real(vzTmp*cjvzTmp)-0.5*vSqu)+pSquCoeff);
return TDataTmp
end
```

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

```julia
##################calculate force density########################
function fDens(θ::Float64,ϕ::Float64,parID::Integer,ωbTmp::Float64, modelTmp::FrequencySimulation, coefData::Matrix{ComplexF64})
coefDataTmp=deepcopy(coefData);
simModelTmp=modelTmp;
dimensionTmp=typeof(simModelTmp.source.medium).parameters[2];
fDensDataTmp=Array{Float64}(undef,dimensionTmp);    #[fDensx,fDensy,fDensz]
R=simModelTmp.particles[parID].shape.radius+0.00015;
x0=simModelTmp.particles[parID].shape.origin[1];#x0,y0,z0 denote particle's position
y0=simModelTmp.particles[parID].shape.origin[2];
z0=simModelTmp.particles[parID].shape.origin[3];
sθ=sin(θ);cθ=cos(θ);sϕ=sin(ϕ);cϕ=cos(ϕ);
xTmp=x0+R*sθ*cϕ;
yTmp=y0+R*sθ*sϕ;
zTmp=z0+R*cθ;
TData=T(xTmp,yTmp,zTmp,ωbTmp,simModelTmp,coefDataTmp);
fDensDataTmp[1]=(TData[1,1]*sθ*cϕ+TData[1,2]*sθ*sϕ+TData[1,3]*cθ)*R*R*sθ;
fDensDataTmp[2]=(TData[2,1]*sθ*cϕ+TData[2,2]*sθ*sϕ+TData[2,3]*cθ)*R*R*sθ;
fDensDataTmp[3]=(TData[3,1]*sθ*cϕ+TData[3,2]*sθ*sϕ+TData[3,3]*cθ)*R*R*sθ;
return fDensDataTmp
end
```