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

```julia
################calculate force matrix#####################
function forceMat(parPosTmp::Matrix{Float64},Rs::Float64)
parPos=deepcopy(parPosTmp);
δh=0.0005;
dimension=3;
dimension2=2;
rndDigNo=9;
pNo=length(parPos[:,1]);
kMat=Matrix{Float64}(undef,pNo*dimension2,pNo*dimension2);
for iTmp in 1:pNo
    for jTmp in 1:dimension2
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]+2*δh;
        f1=round.(forcePackHigh(Rs,parPos); digits=rndDigNo);
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]-2*δh;
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]+δh;
        f2=round.(forcePackHigh(Rs,parPos); digits=rndDigNo);
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]-δh;
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]-2*δh;
        f3=round.(forcePackHigh(Rs,parPos); digits=rndDigNo);
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]+2*δh;
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]-δh;
        f4=round.(forcePackHigh(Rs,parPos); digits=rndDigNo);
        parPos[iTmp,jTmp]=parPos[iTmp,jTmp]+δh;
        kk1=(-f1.+8*f2.-8*f4.+f3)/12/δh;
        if dimension2==dimension
            kk2=copy(kk1);
        else
            kk2=Array{Float64}(undef,pNo*dimension2);
            countTmp=1;
            for kTmp in 1:pNo*dimension
                if kTmp%3==0
                    continue
                else
                    kk2[countTmp]=kk1[kTmp];
                    countTmp+=1;
                end
            end
        end
        kMat[:,(iTmp-1)*dimension2+jTmp]=kk2;
    end
    println("particle "*string(iTmp)*" done")
end
return kMat
end
```

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

```julia
########sweep particle radius and output force matrix to file#########
function sweepRs(parPosTmp::Matrix{Float64},Rs::Float64)
parPos=deepcopy(parPosTmp);
dRs=0.0001;
println("Input Data file name and press Enter:");
fileName=readline();
outputData=open(fileName*".txt","w");
totRndNo=10;
rndDigNo=9;
for iTmp in 1:totRndNo
    write(outputData,"round"*string(iTmp));
    println("round"*string(iTmp));
    write(outputData,"\n");
    write(outputData,"Rs: "*string(Rs));
    write(outputData,"\n");
    write(outputData,"initial positions: ");
    write(outputData,"\n");
    writedlm(outputData,parPos,'\t');
    write(outputData,"\n");
    parPos=md(parPos,Rs);
    write(outputData,"equilibrium positions: ");
    write(outputData,"\n");
    writedlm(outputData,parPos,'\t');
    write(outputData,"\n");
    kMat=forceMat(parPos,Rs);
    write(outputData,"force matrix: ");
    write(outputData,"\n");
    writedlm(outputData,kMat,'\t');
    println(kMat);
    write(outputData,"\n");
    egv=eigen(kMat);
    write(outputData,"eigenvalues: ");
    write(outputData,"\n");
    writedlm(outputData,round.(egv.values;digits=rndDigNo),'\t');
    println(round.(egv.values,digits=rndDigNo));
    write(outputData,"\n");
    write(outputData,"eigenvectors: ");
    write(outputData,"\n");
    writedlm(outputData,round.(egv.vectors;digits=rndDigNo),'\t');
    write(outputData,"\n");
    write(outputData,"\n");
    Rs=Rs+dRs;
    flush(outputData);
end
close(outputData);
return 0;
end
```