Ridgecrest loading

This notebook processes the time series corresponding to Ridgecrest July 6th mainshock, obtained from ground measurements (see readme_load.txt), to generate the DRM forces necessary to add the seismic wave excitation in the FEM model. The FEM model is not generated here.

Fix some parameters

Image style:

```
In[∘]:= texStyle = {FontFamily → "Latin Modern Roman", FontSize → 20, FontColor → Black};
```

The material properties (all units in International Systems, since distances are in meters):

```
In[∘]:= ρ = 1700.; (*Density*)
        Y = 1.7 * 10^9; (*Young's modulus*)
        ν = 0.15; (*Poisson's ratio*)
        Vs = Sqrt[(Y / ρ)/(2 (1 + ν))];
        Vp = Vs * Sqrt[(2 (1 - ν))/(1 - 2 ν)];
```

Load data Event

Set directory where the file “CICCC.V2” is:

```
folderName = "";
SetDirectory[folderName];
```

Load reference location:

Set event name:

```
In[∘]:= eventName = "Ridgecrest";
        eventDescriptor = "Ridgecrest records. 7/6/19, Christmas Canyon";
        fileName = "CICCC.V2";
        rawDataFile = Import[fileName, "Table"];
```

Parameters necessary to read the file:

```
In[•]:= lineI = 47; lineF = 1928;
        Nlines = lineF - lineI + 1;
        blockLines = -5695 + 5740;
```

Read the data in the adequate channels:

```
In[•]:= channels = Table[{{}, {}, {}}, {3}];
        ch = 1;
        vari = 1;
        lineCounter = lineI;
        While[lineCounter < 17085,
         Which[rawDataFile[[lineCounter]][[1]] == 15050,
           (*If true, we are changing variable*)
           (*Print["Ready to add new data to the channel... "<>
              ToString@rawDataFile[[lineCounter]]];*)
           lineCounter = lineCounter + 1;(*Skip this line*)
           ,
           rawDataFile[[lineCounter]][[1]] == "/&",
           (*If trie, we are changing channel*)
           ch = ch + 1;
           vari = 1;
           (*Print["Before changing channel"<>ToString@rawDataFile[[lineCounter]]];*)
           lineCounter = lineCounter + blockLines + 1;(*Skip this block of text*)
           (*Print["After changing channel"<>ToString@rawDataFile[[lineCounter]]];*)
           ,
           True, (*if nothing else, add info to channels and move to next line*)
           channels[[ch]][[vari]] = rawDataFile[[lineCounter ;; lineCounter + Nlines - 1]];
           vari = vari + 1;
           lineCounter = lineCounter + Nlines;
           If[lineCounter > 17085, Break];
           (*Print["After adding new data to the channel"<>
              ToString[{ch,vari}]<>ToString@rawDataFile[[lineCounter]]];*)
          ];
        ]
```

Timestep:

```
In[•]:= deltaT = 0.02;(*seconds*)
```

Reshape results (and convert from cm to m) & Fix issues with Heads

```
In[•]:= accels2 = Table[{}, {3}];
vels2 = Table[{}, {3}];
disps2 = Table[{}, {3}];
Do[
  accels2[[ch]] = 0.01 *
    Flatten@ArrayReshape[channels[[ch]][[1]][[1 ;; Nlines - 1]], {8 * (Nlines - 1), 1}];
  vels2[[ch]] = 0.01 * Flatten@ArrayReshape[
      channels[[ch]][[2]][[1 ;; Nlines - 1]], {8 * (Nlines - 1), 1}];
  disps2[[ch]] = 0.01 * Flatten@ArrayReshape[channels[[ch]][[3]][[1 ;; Nlines - 1]],
      {8 * (Nlines - 1), 1}];
  , {ch, 1, 3}];
(*--------------------------*)
```

The data measured in this first passing requires a fix (some of the data is not read properly in the native format).
The next loop takes care of this issue. The correct information that we need to generate the loading is gathered in the tables “**accels**”, “**vels**” and “**disps**”.

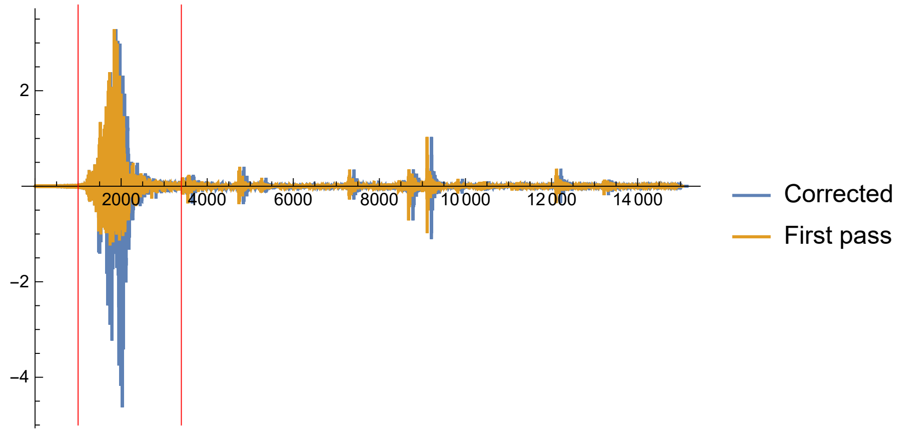
```
In[•]:= accels = Table[{}, {3}];
vels = Table[{}, {3}];
disps = Table[{}, {3}];
Do[
  (*fix displacements*)
  headers = Head[#] & /@ Flatten[disps2[[ch]]];
  poss = Flatten[Position[headers, Times]];
  Do[
   If[MemberQ[poss, jj], (*if it is a special element*)
    auxCh1 = ToExpression@StringSplit[disps2[[ch]][[jj]][[2]], "-" → -1];
    auxCh2 = {};
    kk = 1;
    While[kk ≤ Length[auxCh1],
     If[auxCh1[[kk]] == -1,
      (*add negative sign*)
      auxCh2 = AppendTo[auxCh2, auxCh1[[kk]] * auxCh1[[kk + 1]]];
      kk = kk + 2, (*skip one*)
      (*else, positive number*)
      auxCh2 = AppendTo[auxCh2, auxCh1[[kk]]];
      kk = kk + 1 (*move to next*)]
    ];
    Do[
     disps[[ch]] = AppendTo[disps[[ch]], 0.01 * auxCh2[[uu]]]
     , {uu, 1, Length@auxCh2}]
    , (*if it is just any element*)
    disps[[ch]] = AppendTo[disps[[ch]], disps2[[ch]][[jj]]]]
   , {jj, 1, Length@headers}];
  (* fix velocities *)
  headers = Head[#] & /@ Flatten[vels2[[ch]]];
  poss = Flatten[Position[headers, Times]];
Do[
 If[MemberQ[poss, jj], (*if it is a special element*)
  auxCh1 = ToExpression@StringSplit[vels2[[ch]][[jj]][[2]], "-" → -1];
  auxCh2 = {};
  kk = 1;
  While[kk ≤ Length[auxCh1],
   If[auxCh1[[kk]] == -1,
    (*add negative sign*)
    auxCh2 = AppendTo[auxCh2, auxCh1[[kk]] * auxCh1[[kk + 1]]];
    kk = kk + 2, (*skip one*)
    (*positive number*)
    auxCh2 = AppendTo[auxCh2, auxCh1[[kk]]];
    kk = kk + 1 (*move to next*)]
  ];
  Do[
   vels[[ch]] = AppendTo[vels[[ch]], 0.01 * auxCh2[[uu]]]
   , {uu, 1, Length@auxCh2}]
  , (*if it is just any element*)
  vels[[ch]] = AppendTo[vels[[ch]], vels2[[ch]][[jj]]]]
 , {jj, 1, Length@headers}];
(*fix accelerations*)
headers = Head[#] & /@ Flatten[accels2[[ch]]];
poss = Flatten[Position[headers, Times]];
Do[
 If[MemberQ[poss, jj], (*if it is a special element*)
  auxCh1 = ToExpression@StringSplit[accels2[[ch]][[jj]][[2]], "-" → -1];
  auxCh2 = {};
  kk = 1;
  While[kk ≤ Length[auxCh1],
   If[auxCh1[[kk]] == -1,
    (*add negative sign*)
    auxCh2 = AppendTo[auxCh2, auxCh1[[kk]] * auxCh1[[kk + 1]]];
    kk = kk + 2, (*skip one*)
    (*positive number*)
    auxCh2 = AppendTo[auxCh2, auxCh1[[kk]]];
    kk = kk + 1 (*move to next*)]
  ];
  Do[
   accels[[ch]] = AppendTo[accels[[ch]], 0.01 * auxCh2[[uu]]]
   , {uu, 1, Length@auxCh2}]
  , (*if it is just any element*)
  accels[[ch]] = AppendTo[accels[[ch]], accels2[[ch]][[jj]]]]
 , {jj, 1, Length@headers}];
, {ch, 1, 3}];
```

Visualize load

At this point we decide what time interval we are gonna take
(the portion of the record that contains the main shock, delineated by red lines in the plot).

```
In[∘]:= indexOn = 1000;
        indexOff = indexOn + 2400 - 1;
        ListPlot[{accels[[2, All]],
          accels2[[2, All]]},
         Joined → True,
         PlotLegends → {"Corrected", "First pass"},
         PlotRange → All,
         Epilog →
          {Red, Line[{{indexOn, -5}, {indexOn, 5}}], Line[{{indexOff, -5}, {indexOff, 5}}]}]
```

Out[∘]=



Save relevant portion:

```
In[∘]:= ACC = accels[[All, indexOn ;; indexOff]];
        VEL = vels[[All, indexOn ;; indexOff]];
        DIS = disps[[All, indexOn ;; indexOff]];
```

Prepare the DRM files

Set work directory

(wherever the file “finalmesh_v2.mx” is):

```
folderName = "";
SetDirectory[folderName];
```

Load mesh:

```
In[∘]:= meshLin = Import["finalmesh_v2.mx"];
        elements = meshLin["MeshElements"][[1]][[1]];
        points = meshLin["Coordinates"];
```

Characteristic mesh size used in this mesh and position of supporting plane (**hsp**, this parameter must remain consistent across notebooks):

```
In[∘]:= L = 0.1;
        hsp = 0.;
```

We also need some information regarding the DRM contour at this point, namely, the depth at which the deepest elements are (beware, all this parameters must be consistent over notebooks in order to the final simulation to be consistent itself):

```
In[]:= xDRM = 2.0; yDRM = 2.0; zDRM = -2.0; hDRM = hsp - zDRM
Out[]= 2.
```

Since we have already created the DRM for the pedestal model, we can recycle most of the code:

```
In[]:= testPoints = Flatten@Quiet[Position[points, _? (#[[3]] ≥ zDRM &)]];
outerNodesDRM = {};
Do[
 If[(xDRM - L/2 ≤ Abs[points[[testPoints[[ii]]]][[1]]] ≤ xDRM + L/2 &&
      Abs[points[[testPoints[[ii]]]][[2]]] ≤ yDRM + L/4) ||
   (yDRM - L/4 ≤ Abs[points[[testPoints[[ii]]]][[2]]] ≤ yDRM + L/4 &&
      Abs[points[[testPoints[[ii]]]][[1]]] ≤ xDRM + L/4) ||
   (points[[testPoints[[ii]]]][[3]] == zDRM && Abs[points[[testPoints[[ii]]]][[2]]] ≤
      yDRM && Abs[points[[testPoints[[ii]]]][[1]]] ≤ xDRM),
  outerNodesDRM = AppendTo[outerNodesDRM, testPoints[[ii]]]]
 , {ii, 1, Length@testPoints}]
```

Next, choose the actual elements (the 8 in the loop obeys to using linear elements; should we use quadratic ones, a 20 would appear instead):

```
In[]:= auxELemDRM = Table[
   Count[MemberQ[outerNodesDRM, #] & /@ elements[[ii]], True], {ii, 1, Length@elements}];
posAuxELemDRM = Flatten[Position[auxELemDRM, _?Positive]];
auxCuboid = Cuboid[
   {Min[points[[#, {1}]] & /@ outerNodesDRM],
    Min[points[[#, {2}]] & /@ outerNodesDRM],
    Min[points[[#, {3}]] & /@ outerNodesDRM]}
   ,
   {Max[points[[#, {1}]] & /@ outerNodesDRM],
    Max[points[[#, {2}]] & /@ outerNodesDRM],
    Max[points[[#, {3}]] & /@ outerNodesDRM]}];
elemsDRM = {};
Do[
 If[
  Count[RegionMember[auxCuboid, points[[#]] & /@ elements[[posAuxELemDRM[[jj]]]]], True] ==
   8, (*the element is out*)
  elemsDRM = AppendTo[elemsDRM, posAuxELemDRM[[jj]]]]
 , {jj, 1, Length@posAuxELemDRM}]
```

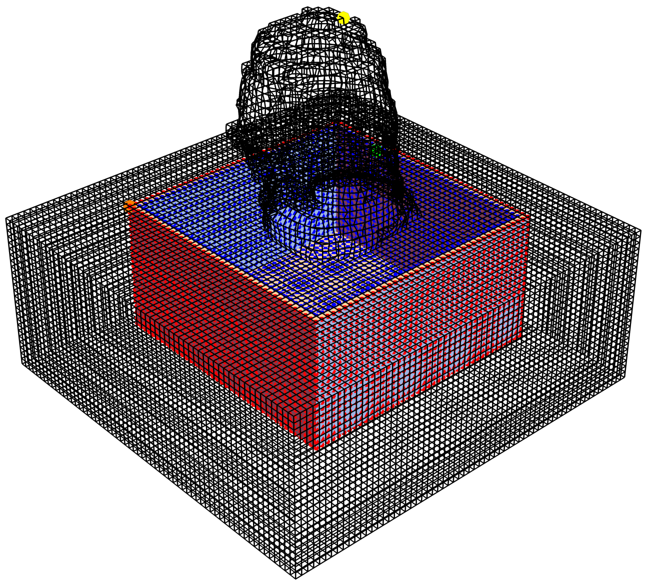
elemsDRM is a list of element tags that correspond to those where the DRM forces are to be applied. Next, we must distinguish interior from exterior nodes.
Remove those nodes in the elements that are not exterior ones to find the internal nodes in the DRM:

```
In[∘]:= innerNodesDRM = DeleteDuplicates@
          DeleteCases[Flatten[elements[[elemsDRM[[#]]]] & /@ Range[Length@elemsDRM]],
           Alternatives @@ outerNodesDRM];
```

Visualize the DRM

```
In[∘]:= Show[meshLin["Wireframe"],
         Graphics3D[{
            {Yellow, PointSize[Large], Point[points[[7274]]]},
            {Green, PointSize[Large], Point[points[[94609]]]},
            {Orange, PointSize[Large], Point[points[[32414]]]},
            {Blue,
             Point[points[[#]] & /@ DeleteDuplicates[innerNodesDRM]]},
            {Red,
             Point[points[[#]] & /@ outerNodesDRM]},
            {EdgeForm[None],
             Hexahedron[points[[#]] & /@ elements[[elemsDRM[[#]]]] & /@ Range[Length@elemsDRM]]}
           }, ImageSize → Large, Boxed → False]
        ]
```

Out[∘]=



Generate full wavefield

Change the timestep from seismometers to simulation:

```
In[∘]:= tlist = Table[deltaT * (ii - 1), {ii, 1, Dimensions[ACC][[2]]}];

In[∘]:= dt = 0.01;
```

```
In[ ]:= Max@tlist
Out[ ]= 47.98

In[ ]:= oldtlist = tlist;

        tlist = Range[0., Max@oldtlist, dt];

In[ ]:= Nsim = Length@tlist;

In[ ]:= vX = VEL[[1]];
        vY = VEL[[2]];
        vZ = VEL[[3]];
```

Turn data into interpolant:

```
In[ ]:= interpVx = Interpolation[Transpose@{oldtlist, vX}];
        interpVy = Interpolation[Transpose@{oldtlist, vY}];
        interpVz = Interpolation[Transpose@{oldtlist, vZ}];
```

Check that the timestep has been halved:

```
In[ ]:= {Length@tlist, Length@oldtlist}
Out[ ]= {4799, 2400}
```

Add waveform depending on height:

```
In[ ]:= velXfun[z_, t_] := 0.5 * If[t * Vs ≥ (z - zDRM), interpVx[t] , 0.] +
           0.5 * If[t * Vs ≥ (hDRM + hsp - z), interpVx[t] , 0.];
        velYfun[z_, t_] := 0.5 * If[t * Vs ≥ (z - zDRM), interpVy[t] , 0.] +
           0.5 * If[t * Vs ≥ (hDRM + hsp - z), interpVy[t] , 0.];
        velZfun[z_, t_] := 0.5 * If[t * Vp ≥ (z - zDRM), interpVz[t] , 0.] +
           0.5 * If[t * Vp ≥ (hDRM + hsp - z), interpVz[t] , 0.];
```

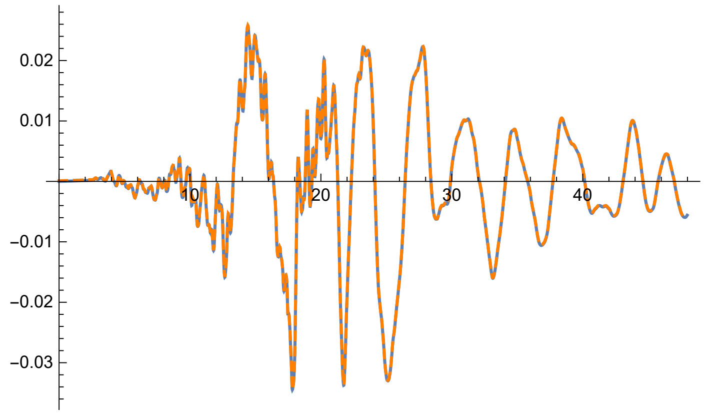
Test the discretization

```
In[•]:= zNode = 0.;
        (*Create input
         data -----------------------------------------------------------------*)
        velX = velXfun[zNode, #] & /@ tlist;
        velY = velYfun[zNode, #] & /@ tlist;
        velZ = velZfun[zNode, #] & /@ tlist;
        (*----------------------------*)
        (*X-direction*)
        dispX = Table[(1/2 (velX[[ii + 1]] + velX[[ii]])) * dt, {ii, 1, Length[velX] - 1}];
        auxX = Join[{0}, dispX];
        dispX = Table[Total[auxX[[1 ;; ii]]], {ii, 1, Length@auxX}];
        accelX = Table[1/dt (velX[[ii + 1]] - velX[[ii]]), {ii, 1, Length[velX] - 1}];
        accelX = Join[{0}, accelX];
        (*Y-direction*)
        dispY = Table[(1/2 (velY[[ii + 1]] + velY[[ii]])) * dt, {ii, 1, Length[velY] - 1}];
        auxY = Join[{0}, dispY];
        dispY = Table[Total[auxY[[1 ;; ii]]], {ii, 1, Length@auxY}];
        accelY = Table[1/dt (velY[[ii + 1]] - velY[[ii]]), {ii, 1, Length[velY] - 1}];
        accelY = Join[{0}, accelY];
        (*Z-direction*)
        dispZ = Table[(1/2 (velZ[[ii + 1]] + velZ[[ii]])) * dt, {ii, 1, Length[velZ] - 1}];
        auxZ = Join[{0}, dispZ];
        dispZ = Table[Total[auxZ[[1 ;; ii]]], {ii, 1, Length@auxZ}];
        accelZ = Table[1/dt (velZ[[ii + 1]] - velZ[[ii]]), {ii, 1, Length[velZ] - 1}];
        accelZ = Join[{0}, accelZ];
```

Visualize vertical displacement to verify that the interpolation has been achieved properly:

```
In[•]:= ListPlot[{
          Transpose@{tlist, dispZ},
          {oldtlist[[#]], disps[[3, indexOn + # - 1]]} & /@ Range[Length@oldtlist]
         },
         PlotStyle → {Automatic, {Orange, Dashed}},
         Joined → True,
         PlotRange → All]
```

Out[•]=



Loop to create DRMForce files:

Set directory to wherever the DRM files ar to be stored.

```
folderName = "";
SetDirectory[folderName];
```

Writing these files is a tedius process. I use 14 kernels to speed the task up.
Start by writing files for the external nodes:

```
In[•]:= CloseKernels[];
        LaunchKernels[14];
```

```
In[•]:= ParallelDo[(*for each node*)
         SIGNAL = {};
         auxLine = ToString[Nsim] <> " 9 1";
         SIGNAL = AppendTo[SIGNAL, auxLine];
         (*Position*)
         zNode = points[[outerNodesDRM[[kk]]]][[3]];
         (*Create input
          data ------------------------------------------------------------*)
         velX = velXfun[zNode, #] & /@ tlist;
         velY = velYfun[zNode, #] & /@ tlist;
         velZ = velZfun[zNode, #] & /@ tlist;
         (*----------------------------*)
         (*Create input
          data ------------------------------------------------------------*)
         velX = velXfun[zNode, #] & /@ tlist;
         velY = velYfun[zNode, #] & /@ tlist;
         velZ = velZfun[zNode, #] & /@ tlist;
         (*----------------------------*)
(*X-direction*)
dispX = Table[(1/2 (velX[[ii + 1]] + velX[[ii]])) * dt, {ii, 1, Length[velX] - 1}];
auxX = Join[{0}, dispX];
dispX = Table[Total[auxX[[1 ;; ii]]], {ii, 1, Length@auxX}];
accelX = Table[1/dt (velX[[ii + 1]] - velX[[ii]]), {ii, 1, Length[velX] - 1}];
accelX = Join[{0}, accelX];
(*Y-direction*)
dispY = Table[(1/2 (velY[[ii + 1]] + velY[[ii]])) * dt, {ii, 1, Length[velY] - 1}];
auxY = Join[{0}, dispY];
dispY = Table[Total[auxY[[1 ;; ii]]], {ii, 1, Length@auxY}];
accelY = Table[1/dt (velY[[ii + 1]] - velY[[ii]]), {ii, 1, Length[velY] - 1}];
accelY = Join[{0}, accelY];
(*Z-direction*)
dispZ = Table[(1/2 (velZ[[ii + 1]] + velZ[[ii]])) * dt, {ii, 1, Length[velZ] - 1}];
auxZ = Join[{0}, dispZ];
dispZ = Table[Total[auxZ[[1 ;; ii]]], {ii, 1, Length@auxZ}];
accelZ = Table[1/dt (velZ[[ii + 1]] - velZ[[ii]]), {ii, 1, Length[velZ] - 1}];
accelZ = Join[{0}, accelZ];
(*--------------------------------------------------------------------------
                                                    ----*)
Do[
 auxLine = ToString[DecimalForm[dispX[[jj]], 5]] <>
   " " <> ToString[DecimalForm[dispY[[jj]], 5]] <> " "
   <> ToString[DecimalForm[dispZ[[jj]], 5]] <> " " <>
   ToString[DecimalForm[velX[[jj]], 5]] <> " "
   <> ToString[DecimalForm[velY[[jj]], 5]] <> " " <> ToString[
    DecimalForm[velZ[[jj]], 5]] <> " " <> ToString[DecimalForm[accelX[[jj]], 5]] <> " "
   <> ToString[DecimalForm[accelY[[jj]], 5]] <> " " <>
   ToString[DecimalForm[accelZ[[jj]], 5]];
 SIGNAL = AppendTo[SIGNAL, auxLine];
 , {jj, 1, Nsim}];
fileName = "DRMForces." <> ToString[outerNodesDRM[[kk]]] <> ".txt";
SetDirectory[newDir];
Export[fileName, ExportString[SIGNAL, "Table"]]
, {kk, 1, Length@outerNodesDRM}]
```

Now write files for the internal nodes:

```
In[ ]:= ParallelDo[(*for each node*)
 SIGNAL = {};
 auxLine = ToString[Nsim] <> " 9 0";
 SIGNAL = AppendTo[SIGNAL, auxLine];
 (*Position*)
 zNode = points[[innerNodesDRM[[kk]]]][[3]];
 (*Create input
 data ------------------------------------------------------------*)
velX = velXfun[zNode, #] & /@ tlist;
velY = velYfun[zNode, #] & /@ tlist;
velZ = velZfun[zNode, #] & /@ tlist;
(*----------------------------*)
(*Create input
 data ------------------------------------------------------------*)
velX = velXfun[zNode, #] & /@ tlist;
velY = velYfun[zNode, #] & /@ tlist;
velZ = velZfun[zNode, #] & /@ tlist;
(*----------------------------*)
(*X-direction*)
dispX = Table[(1/2 (velX[[ii + 1]] + velX[[ii]])) * dt, {ii, 1, Length[velX] - 1}];
auxX = Join[{0}, dispX];
dispX = Table[Total[auxX[[1 ;; ii]]], {ii, 1, Length@auxX}];
accelX = Table[1/dt (velX[[ii + 1]] - velX[[ii]]), {ii, 1, Length[velX] - 1}];
accelX = Join[{0}, accelX];
(*Y-direction*)
dispY = Table[(1/2 (velY[[ii + 1]] + velY[[ii]])) * dt, {ii, 1, Length[velY] - 1}];
auxY = Join[{0}, dispY];
dispY = Table[Total[auxY[[1 ;; ii]]], {ii, 1, Length@auxY}];
accelY = Table[1/dt (velY[[ii + 1]] - velY[[ii]]), {ii, 1, Length[velY] - 1}];
accelY = Join[{0}, accelY];
(*Z-direction*)
dispZ = Table[(1/2 (velZ[[ii + 1]] + velZ[[ii]])) * dt, {ii, 1, Length[velZ] - 1}];
auxZ = Join[{0}, dispZ];
dispZ = Table[Total[auxZ[[1 ;; ii]]], {ii, 1, Length@auxZ}];
accelZ = Table[1/dt (velZ[[ii + 1]] - velZ[[ii]]), {ii, 1, Length[velZ] - 1}];
accelZ = Join[{0}, accelZ];
(*-----------------------------------------------------------------------------
                                            ----*)
Do[
 auxLine = ToString[DecimalForm[dispX[[jj]], 5]] <>
   " " <> ToString[DecimalForm[dispY[[jj]], 5]] <> " "
   <> ToString[DecimalForm[dispZ[[jj]], 5]] <> " " <>
   ToString[DecimalForm[velX[[jj]], 5]] <> " "
   <> ToString[DecimalForm[velY[[jj]], 5]] <> " " <> ToString[
    DecimalForm[velZ[[jj]], 5]] <> " " <> ToString[DecimalForm[accelX[[jj]], 5]] <> " "
   <> ToString[DecimalForm[accelY[[jj]], 5]] <> " " <>
   ToString[DecimalForm[accelZ[[jj]], 5]];
 SIGNAL = AppendTo[SIGNAL, auxLine];
 , {jj, 1, Nsim}];
fileName = "DRMForces." <> ToString[innerNodesDRM[[kk]]] <> ".txt";
SetDirectory[newDir];
```

```
 Export[fileName, ExportString[SIGNAL, "Table"]]
 , {kk, 1, Length@innerNodesDRM}]
```

This concludes the load preprocessing, now the FEM model is ready to run with an impinging wave equivalent to the one that caused the original records.