# Companion to “Roughness evolution induced by third body wear”

This notebook details the post-processing of data to generate the results presented in the aforementioned article. For any further queries, please e-mail the author at joaquin.garciasuarez@epfl.ch



## Preliminaries

We illustrate the post-processing procedure with a specific data batch, corresponding to the simulation of Si surfaces, in the configuration where the bulk features grain boundaries and the initial surface is rough.

As mentioned in the paper, the MD simulations featured a total

sliding of 1 micrometer; the surface atoms' position are extracted for post-processing after every 0.1micrometer of sliding. In this notebook, we will show the surface configuration corresponding to 0.6$\mu$m ("roll6").

## Set directory

Set the directory where the files "surface-bottom.dump" and "surface-top.dump" to be the same as the notebook

```
In[ ]:= SetDirectory[NotebookDirectory[]];
```

## Launch parallel kernels

Since the post-processing is data intensive, we will use some of the parallelization capabilities of Mathematica.

```
In[ ]:= nKernels = 5;(*number of parallel kernels to
         use -- adapt depeding on your hardware*)
        CloseKernels[];
        LaunchKernels[nKernels];
```

# Processing the bottom surface

Name of the file

```
In[ ]:= surf = "bottom";
        filename = "surface-" <> surf <> ".dump";
```

Load file

In[ ]:=
```
data = Import[filename, "Table"];(*import file*)
firstRow = 10;(*remove headers*)
pos = data[[firstRow ;; -1, -3 ;; -1]];
(*save atom positions*)
```

Get the data limit values

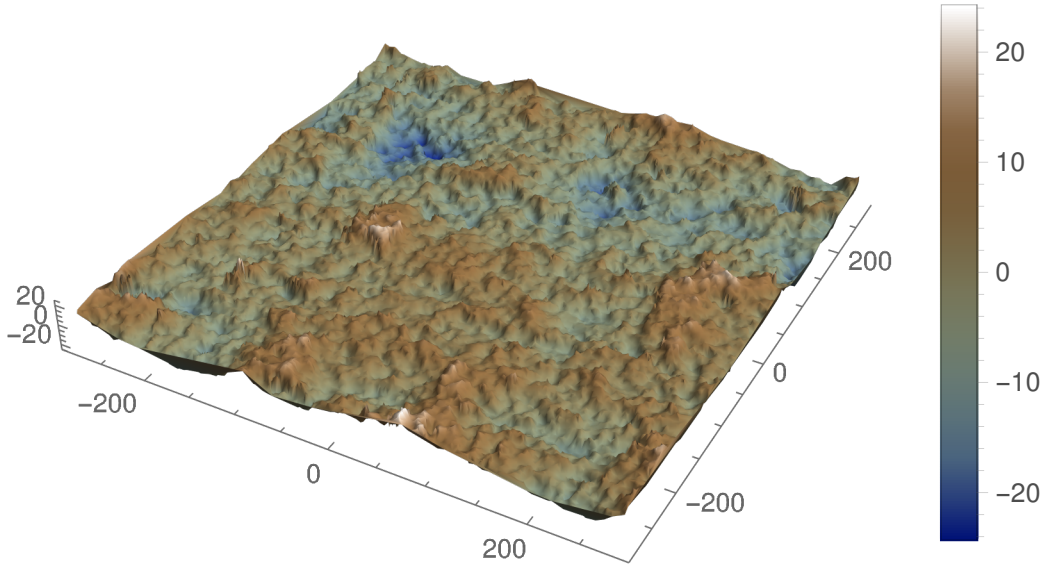
In[ ]:=
```
meanZ = Mean[Last /@ pos];
minZ = Min[Last /@ pos];
maxZ = Max[Last /@ pos];
minX = Min[First /@ pos];
maxX = Max[First /@ pos];
minY = Min[(#[[2]]) & /@ pos];
maxY = Max[(#[[2]]) & /@ pos];
plotRange = {{minX, maxX}, {minY, maxY}, {minZ, maxZ}};
Lx = maxX - minX;
Ly = maxY - minY;
heighttRange = maxZ - minZ;
```

Lower position (zero mean) and make sure that all points enter the grid

## Let's see

```
In[ ]:= posLowered = (# - {0, 0, meanZ}) & /@ pos;
        myLegends =
          BarLegend[{"DarkTerrain", {meanZ - maxZ, maxZ - meanZ}}];
        surfaceBottom = Legended[
          Show[
           ListPlot3D[posLowered,
            PlotRange → All,
            InterpolationOrder → 1,
            BoundaryStyle → None,
            Boxed → False,
            Mesh → None,
            (*Axes→True,
            AxesOrigin→{0,0,0},
            AxesLabel→{"X","Y","Z"},*)
            ColorFunction → Function[{x, y, z},
              ColorData[{"DarkTerrain",
                  {meanZ - maxZ, maxZ - meanZ}}][z]],
            ColorFunctionScaling → False,
            BoxRatios → {1, 1, 0.1},
            ImageSize → 500
           ], PlotRange → {{minX, maxX}, {minY, maxY},
             {meanZ - maxZ, maxZ - meanZ}}
          ], myLegends]
```

Out[ ]=




## Resample surface

The atoms do not fall into a regular grid, so we shall interpolate linearly among the actual positions and then resample using a regular grid before we can use Fourier analysis.

Interpolating function:

In[ ]:=
```
interp = Interpolation[
    {{#[[1]], #[[2]]}, #[[3]]} & /@ posLowered,
    InterpolationOrder → 1, PeriodicInterpolation → True,
    "ExtrapolationHandler" →
     {Indeterminate &, "WarningMessage" → False}];
```

Create the grid:

In[ ]:=
```
deltaX = 1.;(*angstrom*)
```

```
spanX = maxX - minX;
nPoints = Round[spanX / deltaX, 2];
discX = Subdivide[Ceiling@minX, Floor@maxX,
    nPoints - 1];
discY = discX;(*using same grid*)
```

Interpolate and re-sample in regular grid:

```
window[x_, y_] = ((3 π)/8 - 2/π)^(-1/2)*
    (1 + Cos[(2 π*Sqrt[x^2 + y^2])/spanX])*
    Boole[Sqrt[x^2 + y^2] < spanX/2];(*Hanh window*)
(*new regular grid data*)
data =
  Table[window[discX[[j]], discX[[i]]]*
    interp[discX[[j]], discX[[i]]], {i, 1, nPoints},
   {j, 1, nPoints}];
(*correct rounding errors*)
data = ReplaceAll[data, {Indeterminate → 0.}];
```

## Spectral analysis

Compute discrete Fourier transform

```
In[ ]:= fft = Fourier[data, FourierParameters → {1, -1}];
```

Compute amplitudes squared and wavenumbers:

In[ ]:=
```
(*Amplitudes squared*)
amps2 =
  Table[1/nPoints^2 Re[fft[[ii, jj]]*Conjugate[fft[[ii, jj]]]],
   {ii, 1, nPoints}, {jj, 1, nPoints}];
(*corresponding positive wavenumbers*)
qSubList = Table[2 π*k/spanX, {k, 0, nPoints/2 - 1}];
```

In[ ]:=
```
(*full wavenumber list*)
qListX =
  qListY = Join[1.*qSubList,
    -1.*Reverse[qSubList + qSubList[[2]]]];
```

Create the 2D spectrum, both with actual amplitudes and the log thereofL

In[ ]:=
```
spectrumLog =
  ParallelTable[{qListX[[ii]], qListY[[jj]],
    Log10@amps2[[ii, jj]]}, {jj, 1, nPoints},
   {ii, 1, nPoints}];
spectrum = ParallelTable[
   {qListX[[ii]], qListY[[jj]], amps2[[ii, jj]]},
   {jj, 1, nPoints}, {ii, 1, nPoints}];
```

Intermediate steps as the resampling and the windowing yield an average height different from zero.
Because of this we must put the intermediate zero-wavelength harmonic amplitude to zero manually
(this is equivalent to enforcing mean(heights) = 0)

```
In[ ]:= spectrum[[1, 1]][[3]] = 0.;
        spectrumLog[[1, 1]][[3]] = 0.;
```

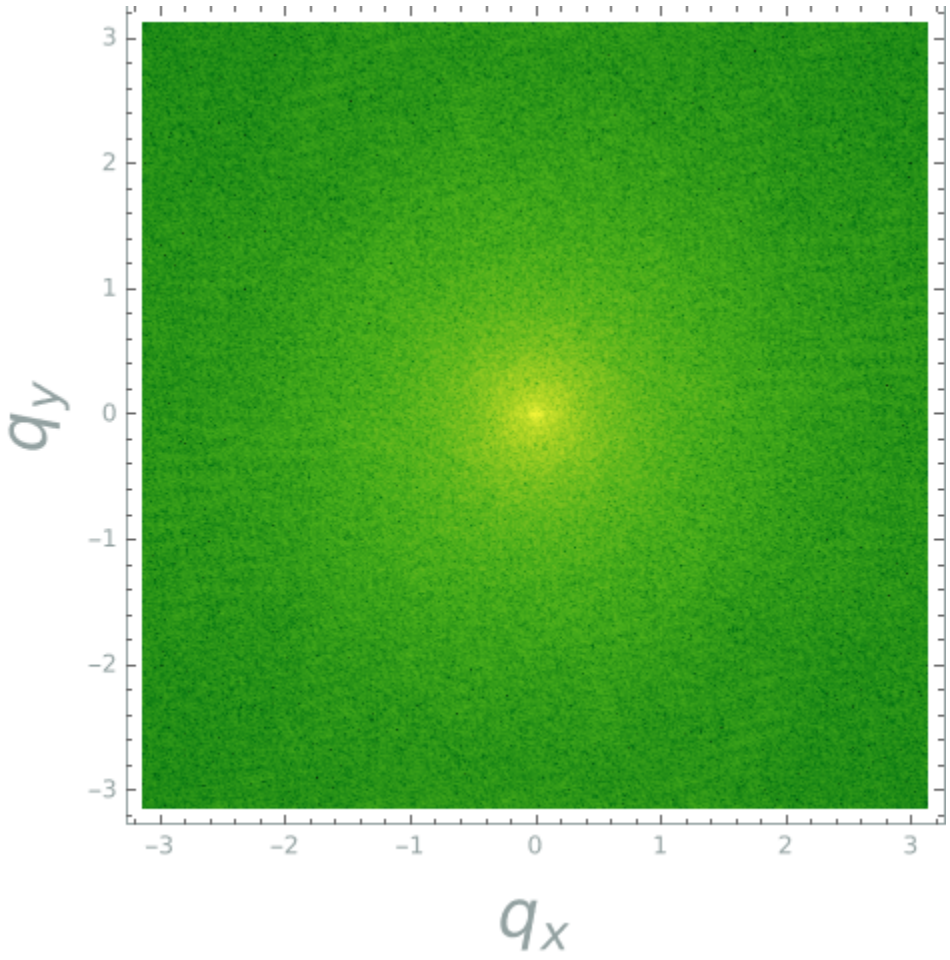
## Let's see the PSD

```
In[ ]:= interval = {Min@(Last /@ Flatten[spectrumLog, 1]),
          Max@(Last /@ Flatten[spectrumLog, 1])};
        legend = BarLegend[{"AvocadoColors", interval}];
        bottomPSD = ListDensityPlot[Flatten[spectrumLog, 1],
          PlotRange → All,
          ColorFunction → "AvocadoColors",
          FrameLabel → {Style["qx", 25], Style["qy", 25]}
         ];
```

See how this image is not perfectly radially symmetric yet (hence, the surface at this point is not isotropic)

In[ ]:=
```
Rasterize@bottomPSD
```

Out[ ]=




## Average radially to estimate the Hurst exponent:

Analyze next the Hurst exponent, what required approximating the PSD through a radially-averaged function.

In[ ]:=
```
auxArray = Table[0, {ii, nPoints^2}];
(*auxiliary array*)
```

The auxiliary array re-shapes the spectrum for later interpolation (to be followed by averaging over the circumferences centered in the origin)

```
In[ ]:= Do[
          Do[
           auxArray[[ii + (jj - 1) * nPoints]] = spectrum[[ii, jj]];
           , {ii, 1, nPoints}]
          , {jj, 1, nPoints}]
```

Linearly interpolating the PSD discrete data.

```
In[ ]:= interpQ = Interpolation[auxArray,
            InterpolationOrder → 1];

        nPointsAngle = 100;(*sample every 2π/100 rads*)
```

```
In[ ]:= amps2polar = {};
        Do[
         ρ = 1. * qSubList[[kk]];
         (*only possitive values*)
         auxValues = {};
         Do[
          qx = ρ * Cos[θ * 2 π / nPointsAngle];
          (*x wavelength at that angle*)
          qy = ρ * Sin[θ * 2 π / nPointsAngle];
          (*y wavelength at that angle*)
          AppendTo[auxValues, interpQ[qx, qy]];
          , {θ, 0, -1 + nPointsAngle}];
         AppendTo[amps2polar, {ρ, Mean@auxValues}]
         , {kk, 1, nPoints / 2 - 1}]
```

Post - process to find the Hurst exponent:

In[ ]:=
```
dat2 = Log10 /@ amps2polar[[nPoints / 4 ;; nPoints / 2 - 1]];
lm = LinearModelFit[dat2, x, x];
estimates = lm["BestFitParameters"];
(*Hurst*)
hurst = (-estimates[[2]] / 2 - 1)
```

Out[ ]=
```
0.856847
```

## Let's see (in a plot) the results and the Hurst exponent fitting

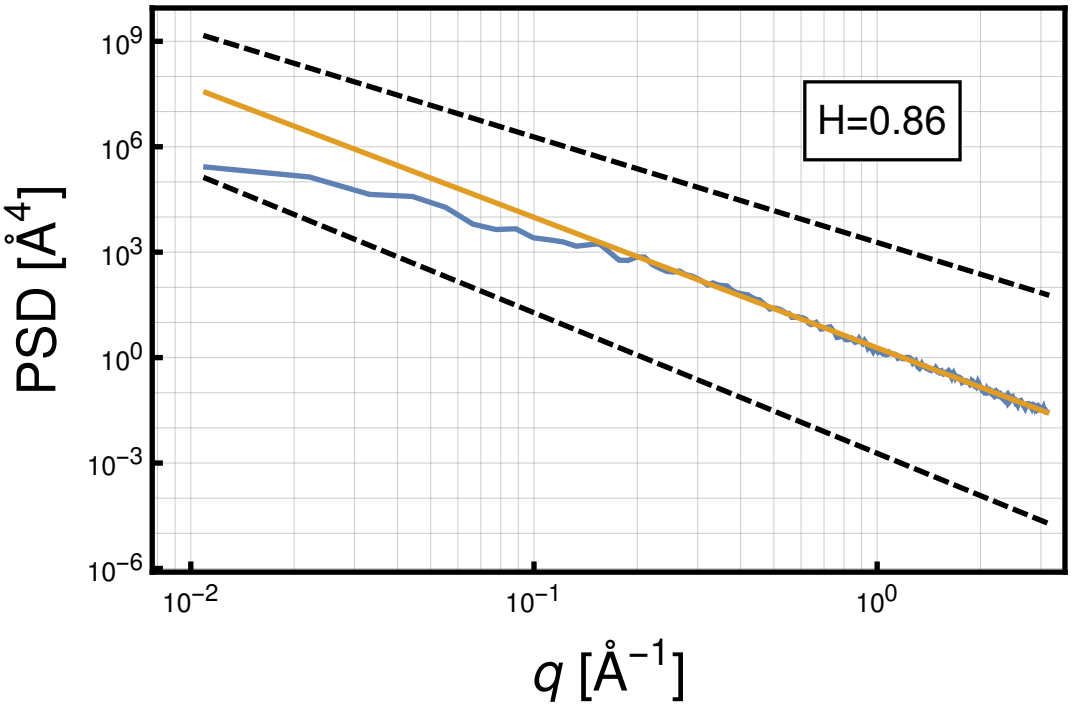
In[ ]:=
```
(*ticks for the plot*)
ticksX = Table[{10 ^ i, 10 ^ Defer[Evaluate[i]]}, {i, -9, 9}];
ticksY = Table[{10 ^ i, 10 ^ Defer[Evaluate[i]]},
   {i, -12, 12, 3}];
(*limit lines for the plot*)
hurstLineOne = Table[{amps2polar[[i + 1, 1]],
    10 ^
     (estimates[[1]] - 3 -
        2 (1 + 1) Log10[amps2polar[[i + 1, 1]]])},
   {i, 1, nPoints / 2 - 2}];
```

In[ ]:=
```
hurstLineOneHalf = Table[{amps2polar[[i + 1, 1]],
    10 ^
     (estimates[[1]] + 3 -
        2 (1 + 0.5) Log10[amps2polar[[i + 1, 1]]])},
   {i, 1, nPoints / 2 - 2}];

hurstLine = Table[{amps2polar[[i + 1, 1]],
    10 ^
```

```
      (estimates[[1]] -
         2 (1 + hurst) Log10[amps2polar[[i + 1, 1]]])},
    {i, 1, nPoints / 2 - 2}];
plotPSDbottom = ListLogLogPlot[{
    amps2polar,
    hurstLine,
    hurstLineOne,
    hurstLineOneHalf
   }
   , Joined → {True, True},
   GridLines → All,
   Axes → Off,
   Frame → True,
   PlotStyle → {Automatic, Automatic, {Black, Dashed},
     {Black, Dashed}},
   FrameStyle → Directive[Black, Thick],
   FrameTicks → {{ticksY, None}, {ticksX, None}},
   FrameLabel → {{Style["PSD [Å^4]", 20], None},
     {Style[(*Å^-1*)"q [Å^-1]", 20], None}},
   Background → White,
   ImageSize → 400,
   Epilog → {
     {Text[
       Style[
        Framed["H=" <> ToString[NumberForm[hurst, {2, 2}]],
         Background → White], 15], Scaled[{0.8, 0.8}]]}
    }
 ]
```

Out[ ]=




# Processing the top surface

Name of the file

```
surf = "top";
filename = "surface-" <> surf <> ".dump";
```

From here, repeat the process...