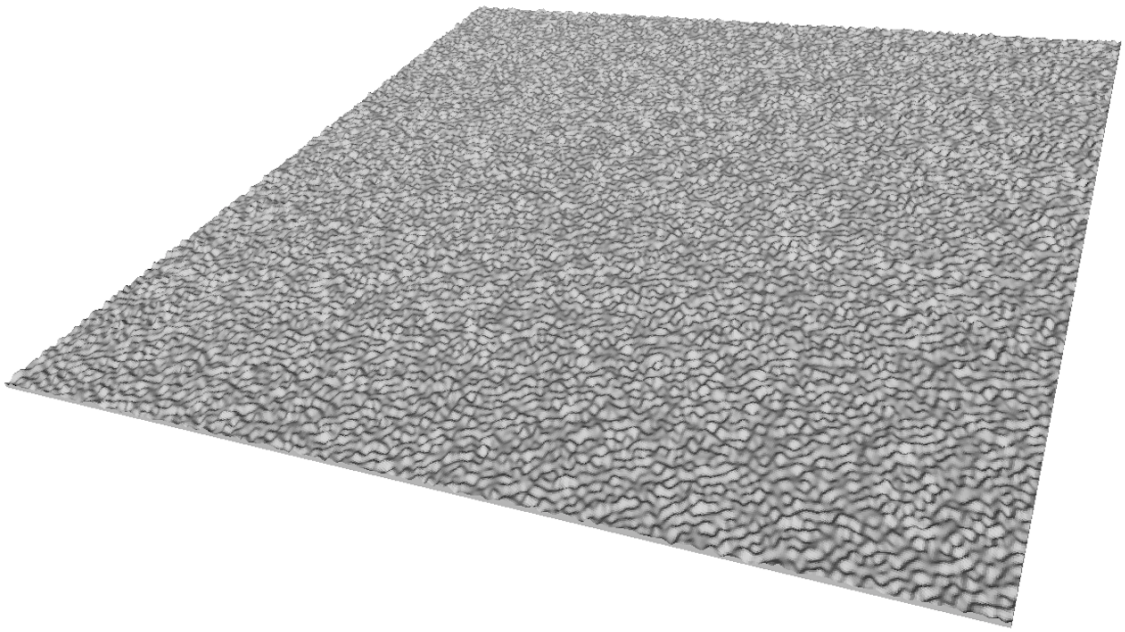
Multi-scale **A**nisotropic **R**ough **S**urface (**MARS**) Algorithm
Version 0.0

Thomas O. Jelly
Department of Mechanical Engineering
University of Melbourne
Victoria 3010 Australia
✉ tom.jelly@unimelb.edu.au

Angela Busse
School of Engineering
University of Glasgow
G12 8QQ Scotland
✉ angela.busse@glasgow.ac.uk



Technical Documentation & User Guide.

1 Background

This document explains how to use the **M**ulti-scale **A**nisotropic **R**ough **S**urface (**MARS**) Algorithm — a computer program capable of generating "realistic" rough surfaces with user-specified statistical properties. Details regarding the numerical implementation, a minimal working example and some excerpts from the MATLAB source code are included.

The **MARS** algorithm was developed at the University of Glasgow in 2017/8 as part of an Engineering and Physical Sciences Research Council (EPSRC) grant entitled "*Fluid dynamic properties of irregular, multi-scale rough surfaces, EP/P004687/1*". The **MARS** algorithm is an extended version of the numerical framework developed first by Patir (1978). Heightmaps generated using the **MARS** algorithm have already led to publications in the literature, e.g. see work by Jelly and Busse (2018b), Jelly and Busse (2018a), Busse and Jelly (2019) and Jelly and Busse (2019).

2 Numerical aspects of the MARS algorithm

The **MARS** algorithm generates irregular heightmaps by taking linear combinations of Gaussian random number matrices using a moving average (MA) process. A doubly-periodic Gaussian heightmap, h_{ij}, of size $N_x \times N_y$, can be generated by evaluating the linear transformation

$$h_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{m} \alpha_{kl}\eta_{rs} \qquad \begin{array}{l} i = 1, 2, \ldots, N_x \\ j = 1, 2, \ldots, N_y \\ r = [i + k - 1 \pmod{N_x}] + 1 \\ s = [j + l - 1 \pmod{N_y}] + 1 \end{array} \tag{1}$$

where η_{ij} is a matrix of uncorrelated Gaussian random numbers, α_{kl} are a set of coefficients that give a specified correlation function, mod denotes the modulo operator and where $n \times m$ is the dimension of the MA window.

Before the the linear transformation (equation 1) can be evaluated, three steps must be taken:

1. Specify the dimensions and size of the heightmap, h_{ij}.
2. Specify the dimensions and size of the correlation function, R_{pq}.
3. Solve a set of non-linear simultaneous equations to obtain a set of weights, α_{kl}.

Each step is discussed in further detail below.

2.1 Step 1: Tile set up

The **MARS** algorithm generates heightmaps on a two-dimensional tile with doubly-periodic boundaries. The planform area of each tile is $A = L_x \times L_y$, where L_x and L_y are the specified streamwise and spanwise lengths, respectively. Each heightmap is resolved using an equi-spaced grid with $N_x \times N_y$ points.

The size and dimensions of the heightmap are specified in the *mars_input.m* file:

```
hmap.Lx = 6.d0; % Streamwise length of heightmap
hmap.Ly = 3.d0; % Spanwise width of heightmap
hmap.Nx = 512;  % Number of streamwise points
hmap.Ny = 256;  % Number of spanwise points
```

The total number of heightmaps to be generated is also specified in the *mars_input.m* file:

```
hmap.N  = 2;    % Number of heightmaps to generate
```

2.2 Step 2: Correlation function

The default correlation function in the **MARS** algorithm is a two-dimensional exponentially decaying function of the form

$$R_{pq} = \exp\left[-2.3\sqrt{\left(\frac{p}{n-1}\right)^2 + \left(\frac{q}{m-1}\right)^2}\right] \qquad \begin{array}{c} p = 0, 1, \ldots, n-1 \\ q = 0, 1, \ldots, m-1 \\ R_{pq} = 0 \ \text{ if } \ p \geq n \ \text{ or } \ q \geq m \end{array} \tag{2}$$

where n and m are the number of points in the streamwise and spanwise directions, respectively. The default values of n and m are based on a 0.1 cutoff criterion, i.e. the spatial separations at which the streamwise and spanwise correlation profiles reduce to 10% of their value at the origin.

The default dimensions of the discrete correlation function are specified in the *mars_input.m* file:

```
corr.n = 42; % Number of streamwise points in the correlation function
corr.m = 3;  % Number of spanwise   points in the correlation function
```

Sometimes it will be necessary to reduce the value of the default cutoff criteria, which, in turn, will increase the size of the correlation function. Note that the **MARS** algorithm will do this automatically. For a given cutoff criteria, say γ, the corresponding values of n and m can be determined using the formula

$$n = \text{floor}\left(n_{0.1}\frac{\log\gamma}{-2.3}\right), \qquad m = \text{floor}\left(m_{0.1}\frac{\log\gamma}{-2.3}\right) \tag{3}$$

where $n_{0.1} \times m_{0.1}$ is the default size of R_{pq} based on the $\gamma = 0.1$ cutoff criteria and "floor" is a function that rounds down to the nearest integer.

2.3 Step 3: Numerical solution of non-linear system

The coefficients, α_{kl}, are determined by solving the system of non-linear equations

$$R_{pq} = \sum_{k=1}^{n-p}\sum_{l=1}^{m-q} \alpha_{kl}\alpha_{k+p,l+q}, \qquad \begin{array}{rcl} p & = & 0, 1, \ldots, n-1 \\ q & = & 0, 1, \ldots, m-1 \end{array} \tag{4}$$

The solution of equation 4 is obtained by assembling a system of the form

$$f_{pq} \quad = \quad \sum_{k=1}^{n-p}\sum_{l=1}^{m-q} a_{kl}a_{k+p,l+q} - R_{pq} \tag{5}$$

and then solving it using MATLAB's inbuilt `fsolve` function. The initial guess used to solve equation 5 can be written as

$$\alpha_{kl}^{0} = sc_{kl} \tag{6}$$

where

$$c_{kl} = \frac{1}{(n-k+1)\,(m-l+1)}, \qquad s = \frac{1}{\sqrt{\sum_{k=1}^{n}\sum_{l=1}^{m} c_{kl}^{2}}} \tag{7}$$

Further details are given in the past work of Patir (1978).

Two stopping criteria are associated with the numerical solution of equation 5. First, a user-specified tolerance which, if crossed, will terminate `fsolve`. Second, a maximum number of iterations which, if exceeded, will also terminate `fsolve`.

The tolerance and maximum number of iterations are specified in the *mars_input.m* file:

```
nls.epsilon  = 1e-5; % Tolerance (Recommended value .le. 1e-5);
nls.max_iter = 8;    % Max number of iterations (Recommended value .eq. 8)
```

If the numerical solution fails to converge, then the value of the cutoff parameter, γ, will be reduced and the **MARS** algorithm will try to solve equation 5 again.

If the numerical of equation 5 converges successfully, then a copy of the coefficients will be saved into the folder called `archive`. For example, if the default dimensions of the correlation function were specified to be $n = 42$ and $m = 3$, then the coefficients will be archived into a file called

archive_042_003.bin

Therefore, for a given $n \times m$ correlation function, equation 5 only ever needs to be solved once. Before trying to solve equation 5, the **MARS** algorithm will automatically check if the coefficients exist.

After the coefficients have been determined, two further steps need to be taken:

4. Evaluate the linear transformation (equation 1).
5. Apply an (optional) low-pass circular Fourier filter to the heightmap.

Each step is discussed in further detail below.

2.4 Step 4: Evaluate linear transformation

Once the coefficients have been determined, the **MARS** algorithm will generate heightmaps by taking linear combinations of Gaussian random number matrices by evaluating equation 1. For each heightmap, an $N_x \times N_y$ Gaussian number matrix is generated using MATLAB's inbuilt `randn` function.

Note: All heightmaps generated using the **MARS** algorithm have a default variance equal to one and mean height equal to zero, as well as approximately zero skewness and kurtosis approximately equal to three.

2.5 Step 5: Low-pass circular Fourier filter

Following Busse et al. (2015), a circular low-pass Fourier filter can be applied to each heightmap to obtain a smoothly varying surface with periodic boundaries, $h_{\text{filt}}(x, y)$. First, the discrete Fourier transform of the unfiltered heightmap is computed to obtain $\hat{h}(k_x, k_y)$. Here, $k_x = \frac{p}{\Delta s M}$ and $k_y = \frac{q}{\Delta s N}$ are the streamwise and spanwise components of the two-dimensinoal wave-vector, where $p = -\frac{N_x}{2}, -\frac{N_x}{2}+1, \ldots, \frac{N_x}{2}-1$ and $q = -\frac{N_y}{2}, -\frac{N_y}{2}+1, \ldots, \frac{N_y}{2}-1$ and where Δs is the spatial resolution of the heightmap. The discrete Fourier transform of the filtered heightmap is then given by

$$h_{\text{filt}}(k_x, k_y) \quad = \quad h_{\text{raw}}(k_x, k_y)\, f_c(k_x, k_y) \tag{8}$$

where

$$f_c(k_x, k_y) = \begin{cases} 1 & \text{for} \quad k_x^2 + k_y^2 \leq k_c^2 \\ 0 & \text{for} \quad k_x^2 + k_y^2 > k_c^2 \end{cases} \tag{9}$$

Here, $f_c(k_x, k_y)$ is the circular filter function and (k_x, k_y) are the cut-off streamwise and spanwise wave numbers, respectively. The filtered surface is then represented by an exact analytical function $h_{\text{filt}}(x, y)$, i.e. a sum of Fourier modes, and can be used to describe the heightmap at any level of resolution.

Applying the low-pass circular Fourier filter is optional. The filter flag and cutoff wavenumber are specified in the *mars_input.m* file:

```
filt.option = 1;     % [1] Apply filter [0] Do not apply filter.
filt.cutoff = 32;    % Cutoff wavenumber in streamwise (x) direction
```

A flowchart representation of the **MARS** algorithm outlining Steps 1-5 is provided below in figure 1. The directory and file structure of the **MARS** algorithm source code is as follows:

```
src
├── mars_algorithm.m              % Top-level script
├── mars_input.m                  % Input script
├── lib
│   ├── mars_stepone.m            % Step 1 script
│   ├── mars_steptwo.m            % Step 2 script
│   ├── mars_stepthree.m          % Step 3 script
│   ├── mars_stepfour.m           % Step 4 script
│   ├── mars_stepfive.m           % Step 5 script
│   ├── nls_init.m                % Initialisation of non-linear system
│   ├── nls_solve.m               % Solution of non-linear system
│   ├── nls_rhs.m                 % Right-hand-side of non-linear system
│   ├── mars_viz.m                % Visualisation script
│   ├── mars_stl.m                % Convert heightmap to .stl script
│   ├── stlwrite.m                % Function to create .stl file
│   └── surf2solid.m              % Function to create 3D surface
└── archive
    └── archive_042_03.dat        % Pre-computed coefficients
```

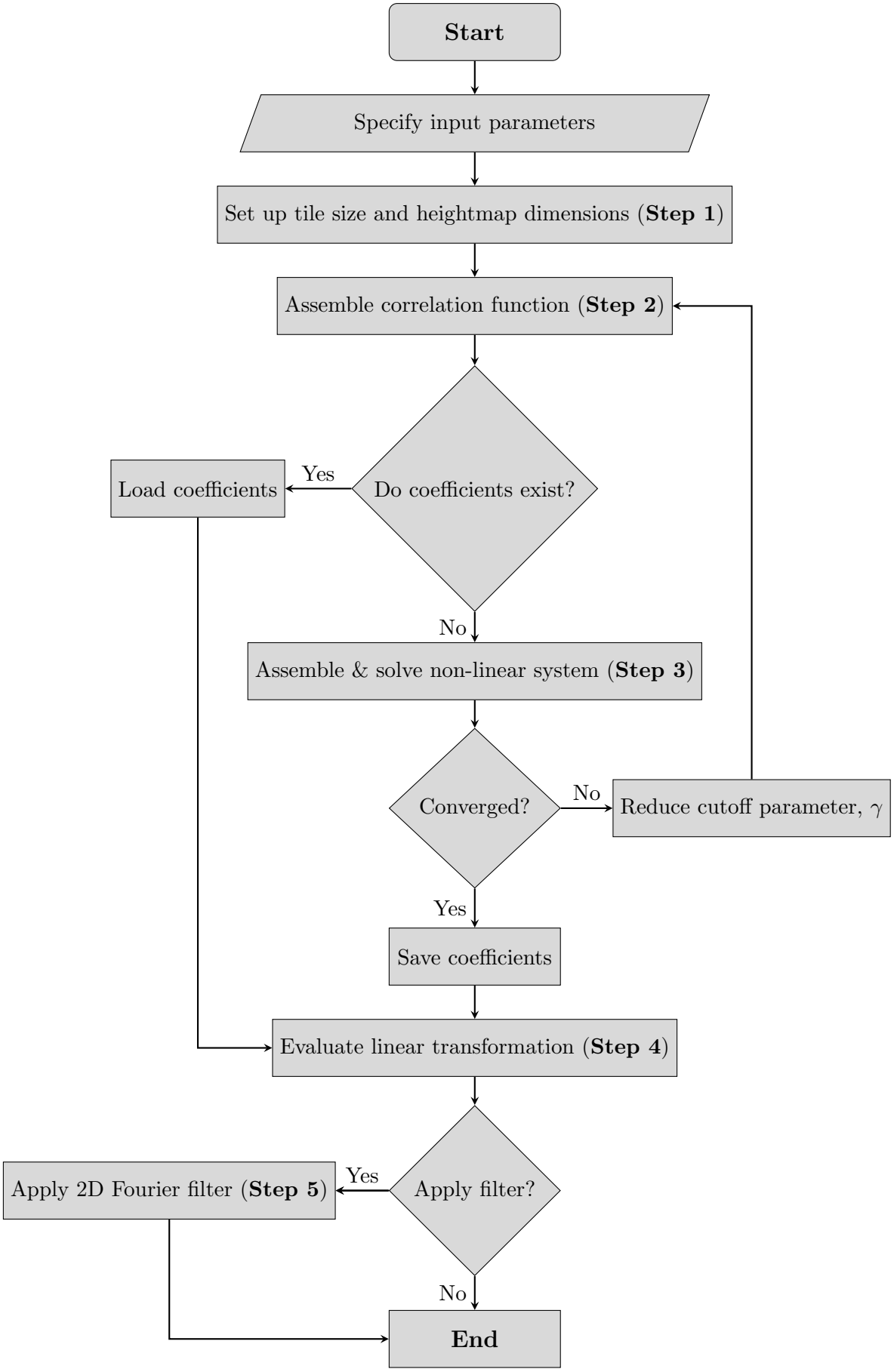


Figure 1: Flowchart representation of the **MARS** algorithm.

3 Minimal working example of the MARS algorithm

3.1 Checking your MATLAB version

The **MARS** algorithm has been implemented using MATLAB Version 9.4.0.813654 (R2018a). To check what version you are using, open MATLAB and then issue the following command:

```
>>  version

ans =

    '9.4.0.813654 (R2018a)'
```

To the best of the authors' knowledge, the **MARS** algorithm is backwards compatible up to MATLAB Version 8.4.0.150421 (R2014b).

3.2 Step-by-step minimal working example

Open MATLAB and then issue the following the command:

```
>> edit mars_input.m
```

and specify the following parameters in the **MARS** algorithm input file:

```
hmap.Lx = 6.d0;      % Streamwise length of heightmap
hmap.Ly = 3.d0;      % Spanwise width of heightmap
hmap.Nx = 512;       % Number of streamwise points
hmap.Ny = 256;       % Number of spanwise points
hmap.N = 2;          % Number of heightmaps to generate
corr.n = 44;         % Number of streamwise points in the correlation function
corr.m = 3;          % Number of spanwise   points in the correlation function
nls.epsilon = 1e-5;  % Tolerance (Recommended value .le. 1e-5);
nls.max_iter = 8;    % Max number of iterations (Recommended value .eq. 8)
filt.option = 1;     % [1] Apply filter [0] Do not apply filter.
filt.cutoff= 64;     % Cutoff wavenumber for circular Fourier filter
```

Save the changes, close the *mars_input.m* file and then issue the following command:

```
>> mars_algorithm
```

After some time, the following output from `fsolve` will appear in the MATLAB command window

```
                                        Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0        985         7.03084                        1.99              1
     1       1970        0.334407       0.467571         0.682             1
     2       2955      0.00153518        0.14884        0.0331          1.17
     3       3940     1.47554e-05      0.0459887       0.00146          1.17
     4       4925     4.24335e-08       0.010632      6.89e-05          1.17
     5       5910     1.02353e-12    0.000744286      3.18e-07          1.17

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the selected value of the function tolerance, and
the problem appears regular as measured by the gradient.
```

After the coefficients have been determined a copy will saved in the `archive` folder.

Next, the **MARS** algorithm will create the heightmaps and the following output will appear in the MATLAB command window:

```
  - Generated 1/2 heightmaps

  - Generated 2/2 heightmaps
```

Once the **MARS** algorithm has finished running the following message will appear:

```
>> MARS algorithm has exited sucessfully.
```

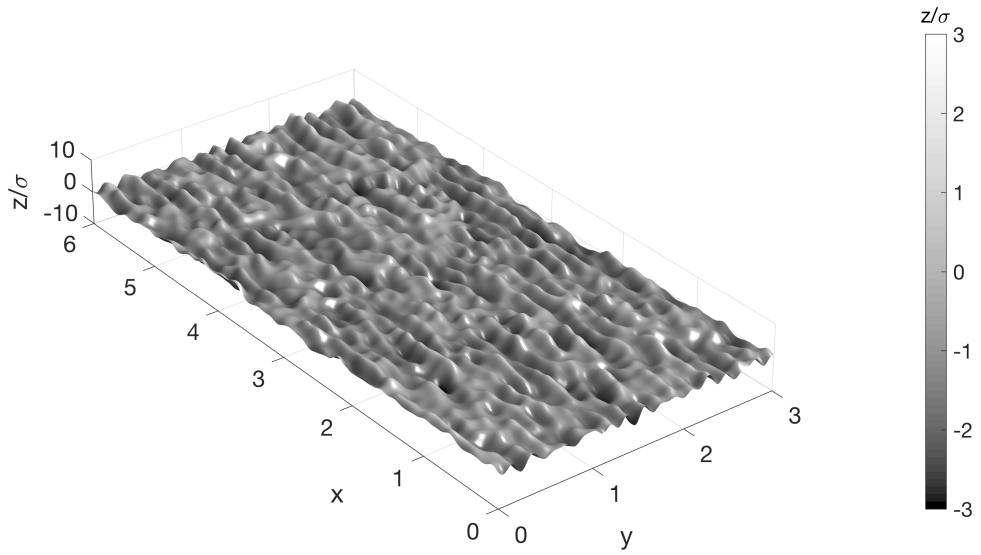
At this point you might want to save your heightmaps by issuing the command:

```
>> save('my_first_heightmaps.mat','hmap');
```

for further post-processing and analysis. Visualise the heightmap by issuing the commands:

```
>> hpick = 1;
>> mars_viz(hmap,hpick);
```

which will produce a figure that looks similar to this:



Issue the following commands to save a heightmap in .stl format:

```
>> hpick = 1;
>> filename = 'my_first_heightmap.stl';
>> mars_stl(filename,hmap,hpick);
```

The .stl file can be opened in your preferred software, e.g. Paraview or Meshlab. Recall that every heightmap generated using is scaled to have zero mean and unit variance. You might want to rescale the height distribution for the purposes of your own problem.

Acknowledgements

This work was funded by Engineering and Physical Sciences Research Council (EPSRC) grants EP/P004687/1. T. O. J. acknowledges his Honorary Research Fellowship at the University of Glasgow and the University of Melbourne Early Career Researcher (ECR) Award.

Disclaimer

The **MARS** algorithm is distributed 'as is' and comes with no warranties or guarantees of any kind.

References

Busse, A. and Jelly, T. O. (2019). Influence of surface anisotropy on turbulent flow over irregular roughness (Accepted). *Flow, Turb. Combust.*

Busse, A., Lützner, M., and Sandham, N. D. (2015). Direct numerical simulation of turbulent flow over a rough surface based on a surface scan. *Comput. Fluids*, 116:129–147.

Jelly, T. O. and Busse, A. (2018a). Reynolds and dispersive shear stress contributions above highly skewed roughness. *J. Fluid Mech.*, 852:710–724.

Jelly, T. O. and Busse, A. (2018b). Impact of Irregular Anisotropic Surface Roughness on the Near-wall Region of Turbulent Channel Flow. In *12th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements*.

Jelly, T. O. and Busse, A. (2019). Reynolds number dependence of Reynolds and dispersive stresses in turbulent channel flow past irregular near-Gaussian roughness (Accepted). *Int. J. Heat Fl. Flow*.

Patir, N. (1978). A numerical procedure for random generation of rough surfaces. *Wear*, 47(2):263–277.