Rheinisch-Westfälische Technische Hochschule Aachen
Institut für Eisenhüttenkunde

- Werkstoff- und Bauteilintegrität -

Studienarbeit

des
B.Eng. Pakaporn Tangnuntachai
Matr.-Nr. 383307

Thema: Entwicklung eines 3D Modells zur Oberflächenrauigkeitscharakterisierung

Topic: Development of 3D Surface Roughness Characterization and Modeling

Durchgeführt in der Abteilung Schädigungstoleranz
vom 20.12.2018 bis 20.03.2019

Betreuer: Univ. Prof. Dr.-Ing. Sebastian Münstermann
M.Sc. Peerapon Wechsuwanmanee

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, sowie Zitate kenntlich gemacht habe.

_____

B. Eng. Pakaporn Tangnuntachai

Hiermit erlaube ich, dass meine Arbeit nach der Abgabe durch weitere Personen als meine Prüfer eingesehen werden darf.

_____

B. Eng. Pakaporn Tangnuntachai

Table of Contents

Abstract

Due to the effort to study material roughness behavior under forming operation and compute damage initiation curve by Finite-Element Method (FEM), good representation of roughness surface as simulation input is the key to achieve an accurate result. This study proposes new model of surface characterization by signal processing and Fourier transform to extract principal information from measured raw surface to roughness. Subsequently, simplifying and importing the surface as the model output. To assess functionalities of the model, data sets representing different surface conditions, which are no process, drilling, milling, water jet and wire cut, are appointed as testing data. Result from the assessment shows that, the developed model can well characterize waviness and roughness. In addition, it is able to keep hypsography distribution which is the identical information of surface influencing damage initiation behavior. To ensure representativeness of the simplified surface to the reference roughness, the model sets up acceptable R-squared value at 0.5. By automatically assigned algorithm, the resulting R-squared values are in range above 0.5 as designated. This new development of 3D surface roughness characterization and modeling gives a promising method to simulate rough surface, bringing about the better understanding in material behavior with specific surface condition under forming operation in micro-scale.

1 Introduction

Damage initiation curve is an important tool for metallurgist to understand material behavior under forming operation before failure. This curve identifies starting point of damage in micro-scale, in explanatory, voids and micro-cracks, which could be later accumulated and results in macro-failure. In conventional way, the curve is computed based on smooth material surface. However, the effort to study influence of material roughness on damage initiation has been raised recently, since one might suspect that with rough surface, material tends to encounter earlier micro-defect formation due to stress concentration at rough surface tips.

To compute the curve, testing of forming process on real material has been conducted. In addition, the curve prediction by Finite-Element Method (FEM) is done in parallel. The difficulty which one might face by simulation is to find an effective way to represent roughness surface as an initial input to the Finite-Element analysis. This representation influences directly on accuracy of the result, since the failure considered in damage initiation curve is discussed in micro-scale.

The surface representation methodology employed currently is a Random height distribution. In detail, set of points are assigned throughout the surface space, by which in distance axis the points are equally spaced, and in height axis the hypsography is assigned randomly correlated to statistical normal distribution of the measured height. These allocated points are connected via polynomial fitting, hence representing as rough surface in FEM. Weak points to emphasize on this methodology is that, first: the hypsography is assigned randomly, second: the polynomial might leave sharp edge between 2 adjacent points due to poor fitting, and last: only the rough surface of 2D curve can be represented, application in 3D topography is limited.

Due to the mentioned deficiencies, this study aims to develop new surface representation model, by which its rough surface characteristic is fully retained, no surface distortion, applicable for 3D topography, and simple for reconstruction. To achieve the goals, tools of signal processing, mathematical theory of Fourier transform and programming are utilized. This study foresees the exceptional advantages of Fourier transform, that the surface result is interchangeably computed based on the input, having potential to well characterize the embedded information. In addition, the programming allows this model to handle with complex mathematics and be able to reproducible used for surface data variety.

With the newly developed surface characterization model, highly intention is to provide reliable method to represent material roughness in Finite-Element simulation. This would lead to the accurate computation of damage initiation curve and enable better understanding in relation between material behavior and surface topography. Lastly, this application can be additionally extended to other relevant surface conditions and forming processes.

2 Theoretical Background

In this chapter, theoretical background utilized to support the model development is described. The chapter is divided into 2 parts. First is surface filtration techniques and second is tools for surface filtration.

2.1 Filtration techniques for surface texture

In this section, the filtration techniques for surface texture is discussed. The content is outlined from definition of roughness and waviness, the surface filtration procedure, and lastly the list of standards relevant to surface characterization.

According to ISO 4287 and ISO 16610-21 [1], definition of surface profiles holding their own characteristics are given in **Table 2-1**.

Table 2-1: Definition of surface profiles

Surface profile	Definition
Actual profile	The real profile of workpiece resulting from the intersection of workpiece surface and surface plane normal
Measured profile	The profile resulting from scanning of the actual profile with measurement probe.
Primary profile (P-profile)	The profile resulting from filtration of the measured profile with low-pass filter at cut-off wavelength λ_s. The irrelevant noise on the surface is cut off.
Roughness profile (R-profile)	The profile resulting from filtration of the primary profile with high-pass filter at cut-off wavelength λ_c. This process removes long wavelength components and left the profile with roughness (fine height fluctuation).
Waviness profile (W-profile)	The profile resulting from filtration of the primary profile with low-pass filter at cut-off wavelength λ_c, followed by the filtration with high-pass filter at cut-off wavelength λ_f. This process removes short wavelength components and left the profile with waviness (wavy height fluctuation).

In more general term, such as in manufacturing aspect, roughness is usually the process marks produced by cutting tool. In some cases, roughness can be the prior condition of material structure. Meanwhile, waviness is usually the effect of machining process such as imbalance of machining equipment [2]. The illustration pictures of primary profile, roughness and waviness are given in **Figure 2-1**.

Another parameter to be introduced is Ra - Arithmetical mean roughness value. Referring to EN ISO 4287, Ra is defined by arithmetical mean of the absolute values of the profile deviations (z_i) from mean line of roughness surface [1]. The value implies that how high of roughness profile is in average. As displayed in **Figure 2-2**, Ra gives no information about surface height distribution. Therefore, by stating this parameter only, it is not sufficient to represent roughness profile characteristic aimed by this study for Finite-Element simulation.

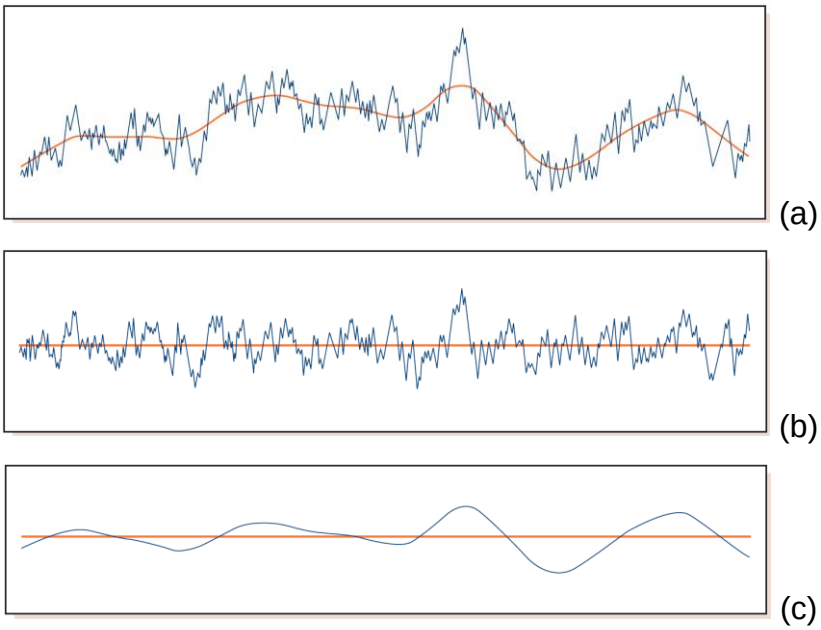


Figure 2-1: Surface profiles **[3]**

(a) Primary profile (P-profile)

(b) Roughness profile (R-profile)

(c) Waviness profile (W-profile)

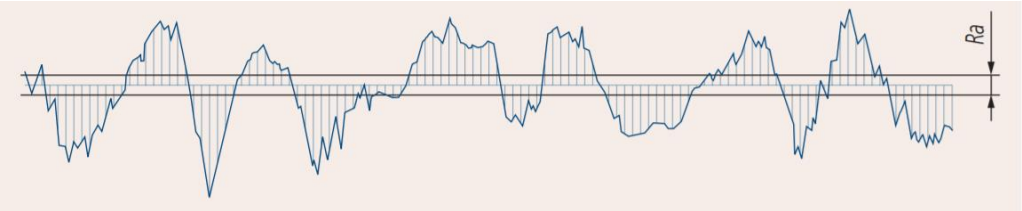


Figure 2-2: Arithmetical mean roughness value (Ra) **[3]**

To obtain the surface profiles mentioned above, filtration is executed by several steps for particular purposes. Flowchart of overall procedure derived from ISO 3274 is given in **Figure 2-3**. After surface measurement, a smoothing filter (cut-off wavelength λ_s) is required in some cases to clean up noise by filtering out the irrelevant short wavelength components. Subsequently, the separation filter (cut-off wavelength λ_c) is employed to separate long-scale components from short-scale components, in other words, to separate waviness from roughness. In the last step, applying particularly on waviness, F-filter (cut-off wavelength λ_f) is executed to standardize mean line of waviness to be horizontally flat by filtering out the long wavelength components. In **Figure 2-4**, 3 mentioned filters are displayed in form of transmission Gaussian parameters according to ISO 11562.

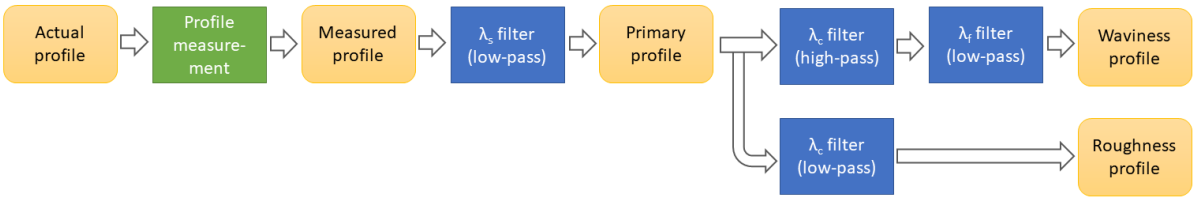


Figure 2-3: Surface filtration procedure **[4], [5]**

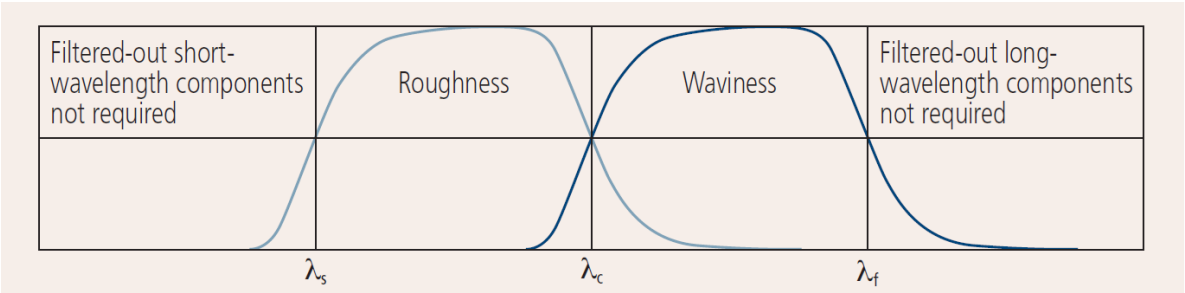


Figure 2-4: Transmission parameters of the filters in surface technology **[3]**
Roughness filter (light blue) and waviness filter (dark blue) cross each other at cut-off wavelength. The transmission coefficient at the intersection point is 50%.

In the history of surface technology, filtration techniques have been long developed. Each technique has its own advantages and disadvantages [5]. First technique to be introduced is RC filter, used formerly for almost 30 years. Name RC is an abbreviation of resistors and capacitors soldered in an analysis device of surface measuring instrument. The method principle is to filter the analog signal arriving at digital board. However, this technique comes with drawback of phase shifting and results in surface distortion. Next technique to be discussed is Gaussian filter.

Gaussian filter has been introduced for surface filtration since 1996 in ISO 11562 (replaced version - ISO 16610-21 in 2011). The standard defines a transfer functions of low-pass filter for waviness. For roughness, it is a subsequent subtraction of the primary profile to the mentioned waviness profile. Choice of cut-off value is critical, since it influences directly on the transmission curve and surface textures of roughness and waviness. One could see its effect in **Figure 2-5**, by which the cut-off wavelengths are set to 0.2 mm, 0.4 mm and 0.8 mm from top to bottom respectively. Gaussian filter has been successfully used for applications in research and industries for more than 20 years. Although, the technique has a disadvantage on outlier filtration.

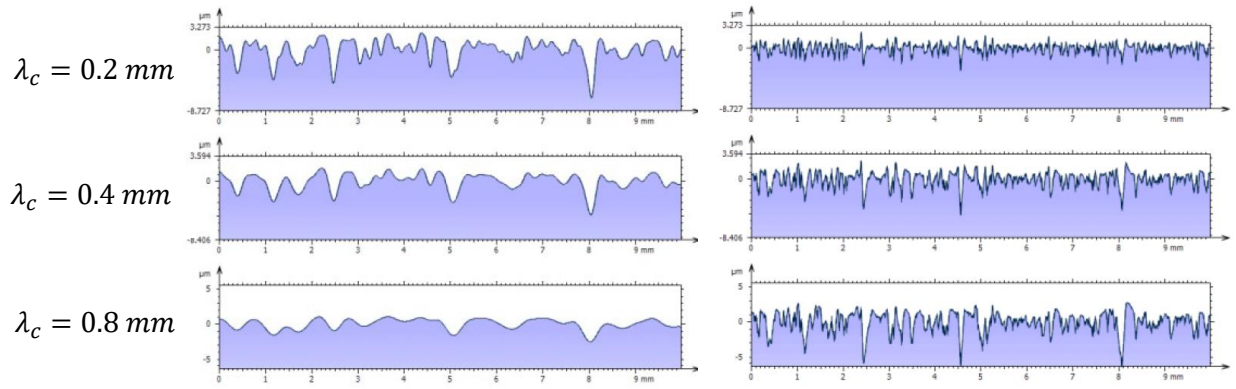


Figure 2-5: Surface profiles after Gaussian filters with different cut-off wavelengths at 0.2 mm, 0.4 mm and 0.8 mm, from top to bottom respectively **[5]**

(a) Waviness filtered surface
(b) Roughness filtered surface

The last filter technique to be introduced is Cubic spline filter. This technique employs the principle of cubic spline curve created by smooth connection between series of data points. The curve fitting is generally done by polynomial function. It is necessary that the curve passes through all data points. Contrarily, in surface filtration application, the curve does not pass through the points but connected with them by spring suspension. The resulting suspended curve is roughness and waviness surface. This filtration technique is described by ISO 16610-22. In practice, Cubic spline filter does not have significant advantage over Gaussian filter.

In addition to surface filtration techniques mentioned above, several new filters are developed to overcome the existing limitation. Still, Gaussian filter is generally accepted as a robust method for surface application. It gives simpler way of

implementation and provides reliable result [6]. In this study, Gaussian filtration technique is chosen because of the expressed advantages.

Lastly, the ISO standards mentioned in this section are listed in **Table 2-2** for reference.

Table 2-2: ISO standards related to surface characterization technology

Standard	Content
ISO 4287	Surface texture: Profile method - Terms, definitions and surface texture parameters
ISO 11562	Surface texture: Profile method - Metrological characteristics of phase correct filters
ISO 3274	Geometrical Product Specifications (GPS) - Surface texture: Profile method - Nominal characteristics of contact (stylus) instruments
ISO 16610-21	ISO 16610-21: Linear profile filters: Gaussian filters
ISO 16610-22	ISO 16610-22: Linear profile filters: Spline filters

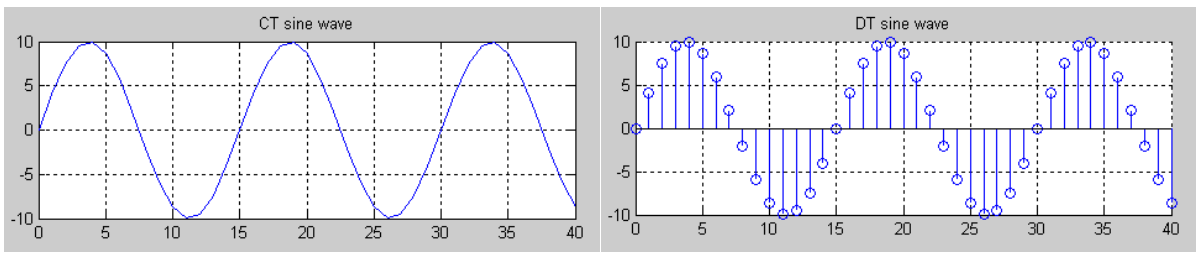
2.2 Tools for surface filtration

To develop the surface characterization model, various tools are utilized in this study. The tools of Signal processing, Mathematical theories and Programing are extensively discussed in this chapter.

2.2.1 Signal processing

Signal processing is a field concerning analysis, synthesis and modification of signals, aiming to reveal an obscure information of the signal behavior. The functionalities are for example, to detect data pattern, to amplify or filter the embedded signal components, to modify signal for specific application, or to compensate the poor signal because of sensor deficiency [7].

For comprehension, the definition of signal is given. A signal is a physical quantity that is varying by time, or in general speaking, varying by other independent parameter which is equally spaced [8]. Signal examples in real-world are such as temperature, pressure, sound and electricity. The signal can be represented in 2 possible forms, analog and digital. The analog is a signal whose value is continuous, the resolution is infinite. In contrast, the digital is a capture of signal which are measured and stored at regular time interval. Thus, the signal is not continuous, or it can be called as discrete. For better understanding, sine wave in analog and digital forms are represented in **Figure 2-6**.



(a) (b)

Figure 2-6: Sine wave signal in analog and digital format **[9]**

(a) Analog (continuous) signal

(b) Digital (discrete) signal

Discussing about procedure of signal processing [10], an input signal is firstly taken into a system. Because most of real-world signals are in analog format, it is necessary to convert analog signal to digital format first by Analog-to-Digital Converter (ADC). Afterwards, the transformed signal passes to the processor in order to apply numerical operations achieving a designated modification. Lastly, the outcome signal

is transformed back to analog format by Digital-to-Analog Converter (DAC), for applications. Flowchart of signal processing steps is shown in **Figure 2-7**.

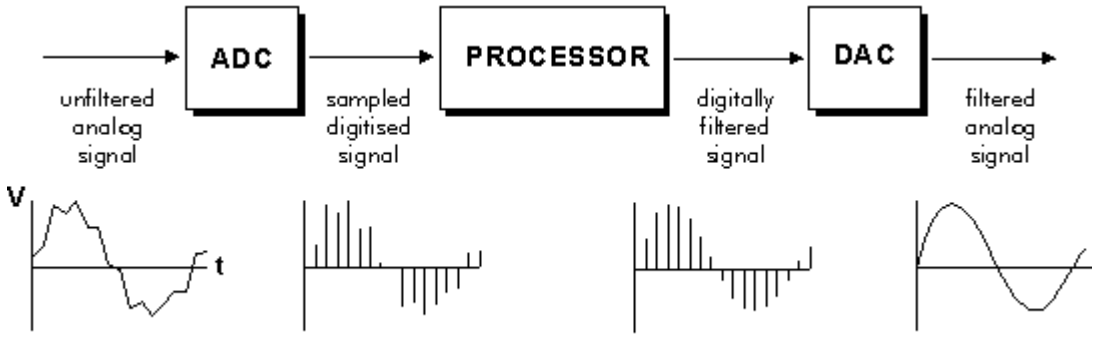


Figure 2-7: Procedure of signal processing **[10]**

This approach to modify signal in digital format is called Digital Signal Processing (DSP). The method provides 2 main advantages [11]. First, the flexibility, which means various signal operations can be conducted on the same set of hardware. Second, the repeatability, which means the operation can be repeated providing an identical result.

With its broad range of application, Digital Signal Processing is recognized as one of the most powerful technologies in the twenty-first century, exclusively in field of science and engineering [12]. The extensive application fields of DSP is shown in **Figure 2-8**.

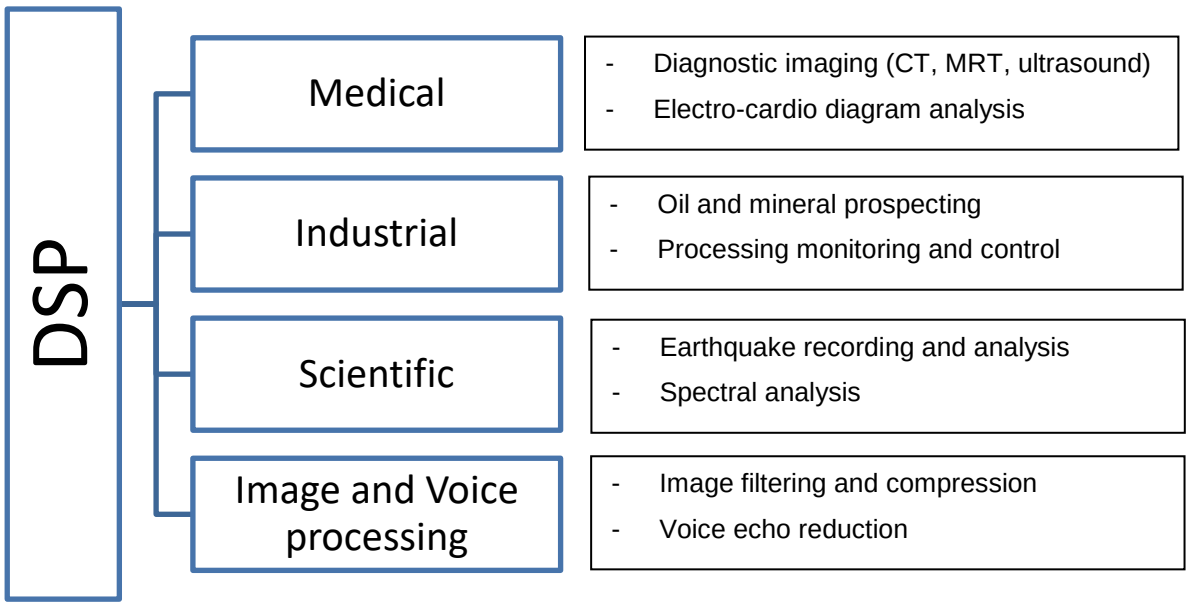


Figure 2-8: Applications of digital signal processing

In Metallurgical field, digital signal processing gives a contribution as well, such as the application of surface characterization. For the model developed by this study, the signal being processed is surface height varying by distance already in digital format. In processor, signal modification is conducted to extract embedded information of roughness and waviness from primary surface. The applied numerical operations are Fourier transform and convolution. Theory of those mentioned mathematics will be discussed in next topic.

2.2.2 Mathematical tools

In this section, Mathematical theories which are extensively utilized to process surface data are explained and discussed.

2.2.2.1 Superposition of sinusoids

In digital signal processing, most of data managing techniques are based on superposition. The principle is to break a complex input signal into an amount of simple components, each component is processed individually, then is reunited for the output result. Advantage of this technique is that, the numerical processes are done more conveniently with many simple sub-components instead of one large complicated component [13].

With the great discovery of Fourier - the mathematician in 1807, sinusoidal function falls perfectly to the aforementioned principles. The finding stated that any periodic and non-periodic functions can be represented by an integral of weighted sinusoids, or in discrete aspect, by superposition of weighted sinusoids [14]. The illustration showing superposition is given in **Figure 2-9,** by which the right-hand-side signal is the superposed result of the 2 left-hand-sides signals.

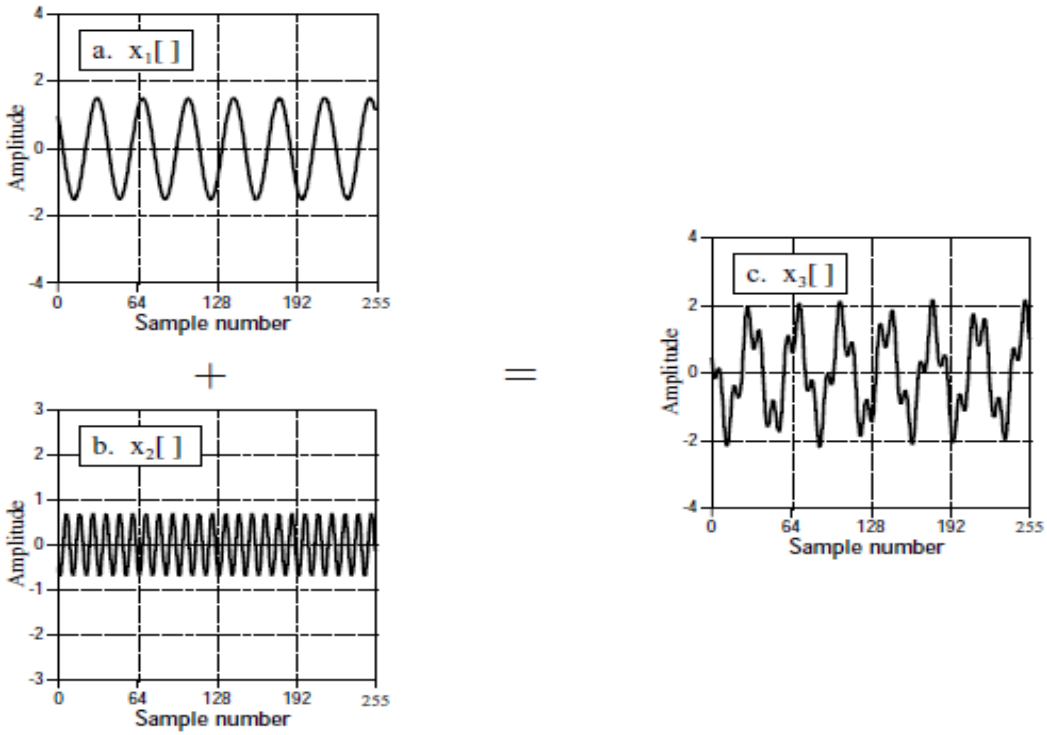


Figure 2-9: Signal derived by superposition of sinusoidal waves **[15]**

The Fourier superposition is important, mainly for 3 reasons [13]. First, most of the signals being processed in real-world are superimposed sinusoids, for example audio signal. Second, the sinusoid has a linear characteristic, which means its frequency remains after modification. Only the amplitude and phase are changed. This property allows the identity information of signal to be kept, no matter how many modifications the signal has passed. And third, broad mathematical tools and programming algorithms are available widely to support transformation operation. This allows user to play with signals conveniently.

2.2.2.2 Fourier transform

Fourier transform is a mathematical technique, aiming to decompose signal to set of sinusoidal frequencies [16]. At one frequency component, it refers to a constant called Fourier coefficient, representing as a frequency's weight function. This constant implies that how much one certain frequency is contained in the original signal. While the signal is represented in time or spatial domain, the frequency domain is a representation of the Fourier coefficient sequences.

There are 2 operation directions involving in the data transformation between the 2 mentioned domains. First is Forward Fourier transform (FT). Its duty is to

decompose signal in spatial domain to frequency components in frequency domain. Conversely, the second operation, called Inverse Fourier transform (IFT), has a duty to superimpose frequency components in frequency domain back to the original signal in spatial domain. The flow diagram of Fourier operations is represented in **Figure 2-10**. In conventional notation, the variable in spatial domain is represented by lower case, for example $x[n]$ and variable in frequency domain is represented by uppercase, for example $X[k]$. To show the sign of discrete index, square bracket [] is used.

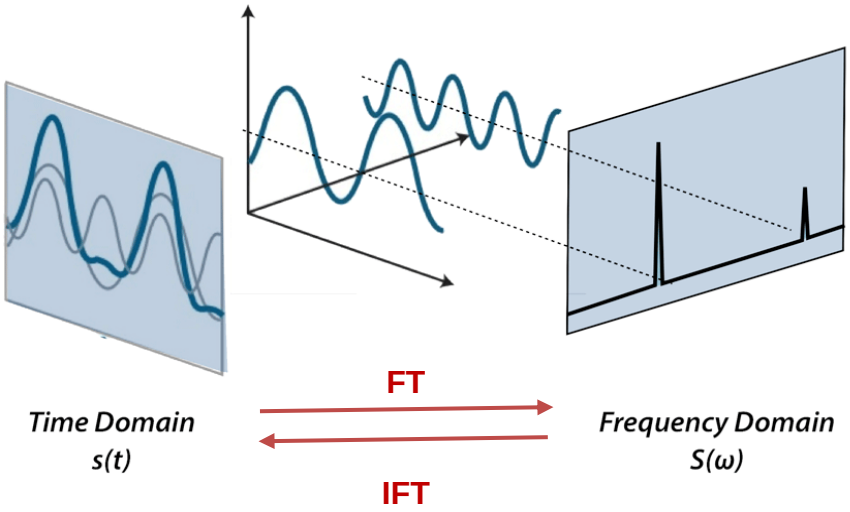


Figure 2-10: Operation directions of Forward Fourier transform (FT) and Inverse Fourier transform (IFT) **[17]**

Furthermore, Fourier transform can be categorized into 2 types, continuous and discrete, depending on type of the signal input. In this study, discrete Fourier transform is the main focus, because the model is dealing with the digitized signal from surface measurement. The equations of discrete forward Fourier transform and discrete inverse Fourier transform are given in **Equation 2-1** to **Equation 2-4** [18]. One should pay attention to the subdividing operation by the number of discrete points in inverse Fourier transform equations. This shall be done to normalize Fourier coefficient as an average after summation over N or $M \cdot N$ points in forward Fourier transform.

For 1D

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{i2\pi kn}{N}} \quad \text{for } k = 0, \dots, N-1 \quad \textbf{(Equation 2-1)}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{\frac{i2\pi kn}{N}} \quad \text{for } n = 0, \dots, N-1 \quad \textbf{(Equation 2-2)}$$

For 2D

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i\left(\frac{ux}{N}+\frac{vy}{M}\right)} \quad \text{for } u = 0, \dots, N-1 \quad \textbf{(Equation 2-3)}$$

$$v = 0, \dots, M-1$$

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i\left(\frac{ux}{N}+\frac{vy}{M}\right)} \quad \text{for } x = 0, \dots, N-1 \quad \textbf{(Equation 2-4)}$$

$$y = 0, \dots, M-1$$

Next topic is about Fourier coefficient. This constant is the key of signal processing, because it is a characteristic value, showing how much contribution of a particular frequency is, on the overall signal. By this reason, the numerical operations aiming to modify signal are operated on this constant. The Fourier coefficient is a complex number, containing the information of amplitude and phase. The formulas to derive these two parameters are given in **Equation 2-5** and **Equation 2-6**. In the next few topics, modification operation computed on Fourier coefficient, such as convolution, will be explained.

For Fourier coefficient: $a + bi$

Amplitude $A = \sqrt{a^2 + b^2}$ **(Equation 2-5)**

Phase $\theta = \tan\left(\frac{b}{a}\right)$ **(Equation 2-6)**

2.2.2.3 Frequency domain

After forward Fourier transform, the only result yielded is Fourier coefficient, giving the information about amplitude and phase. To match this coefficient with frequency axis, user has to assign on his or her own. Theory about the frequency axis assignation and the property of axis symmetry are discussed as following.

Deriving from data in time domain with N points sampling at every T second spacing, the frequency domain will arbitrarily consist of N points with the spacing interval $\frac{1}{T \cdot N}$ reciprocally [19]. The formula for frequency axis assignation is given in **Equation 2-7** and **Equation 2-8** [20]. The unit of frequency is an inverse of spatial unit. In conventional way, 0 Hz is represented at the middle.

$$f = \frac{1}{TN}\left[0, \pm 1, \pm 2, \dots, \pm\frac{N}{2} - 1, -\frac{N}{2}\right] \qquad \text{for } N = even \qquad \textbf{(Equation 2-7)}$$

$$f = \frac{1}{TN}\left[0, \pm 1, \pm 2, \dots, \pm\frac{N-1}{2}\right] \qquad \text{for } N = odd \qquad \textbf{(Equation 2-8)}$$

One important feature of Fourier transform is the symmetric property. In case if the data in time domain is real value, the Fourier coefficients with equal distance from 0 frequency are conjugate of each other X[-ω] = X*[ω]. It means that the real part of 2 conjugates are identical, while the imaginary part is inverted. This results the conjugates to hold same amplitude but phase correlation is 90 degree out. In graphical aspect, the display of frequency domain will show mirror-liked appearance, by which x-axis is a reflection line of the positive and negative sides of the mirror [21]. The illustration of this effect is shown in **Figure 2-11**.

Because of conjugate symmetric property, it is practical to display a result of Fourier coefficients only by Right-hand side (RHS). However, the amplitude has to be corrected by double multiplication. This is done to compensate the discarded value of Left-hand side (LHS) frequency.

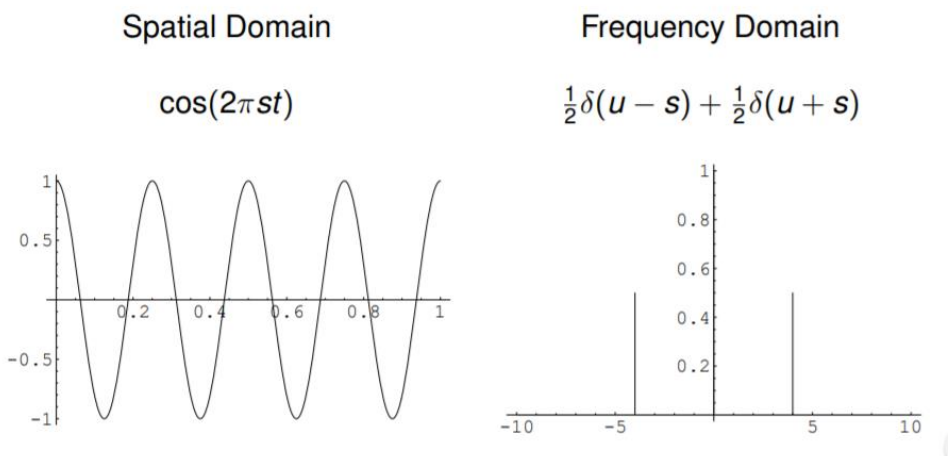


Figure 2-11: Symmetry property of Fourier coefficient **[22]**

2.2.2.4 Convolution

Convolution is a mathematical operation used to combine two signals in order to produce the third signal. This output signal implies the relationship between two primary inputs, that how the shape of one modifies the other [23]. This technique is significantly important in digital signal processing for the application of signal modification. The operation starts by feeding raw data into the system as the first signal, then convolve it with the second signal designed in certain shape for filter effect. This second signal is widely called as filter kernel.

There are 2 alternative ways to perform convolution. First is convolution operation (*) in spatial domain, as written by **Equation 2-9** in discrete format. To demonstrate the operation in physical meaning, **Figure 2-12** is given. By operation, the second signal is flipped and translating shifted by n, then mathematically compute with the first signal located in the original position. The area under these 2 signals is a convolution result at certain time n.

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \qquad \textbf{(Equation 2-9)}$$

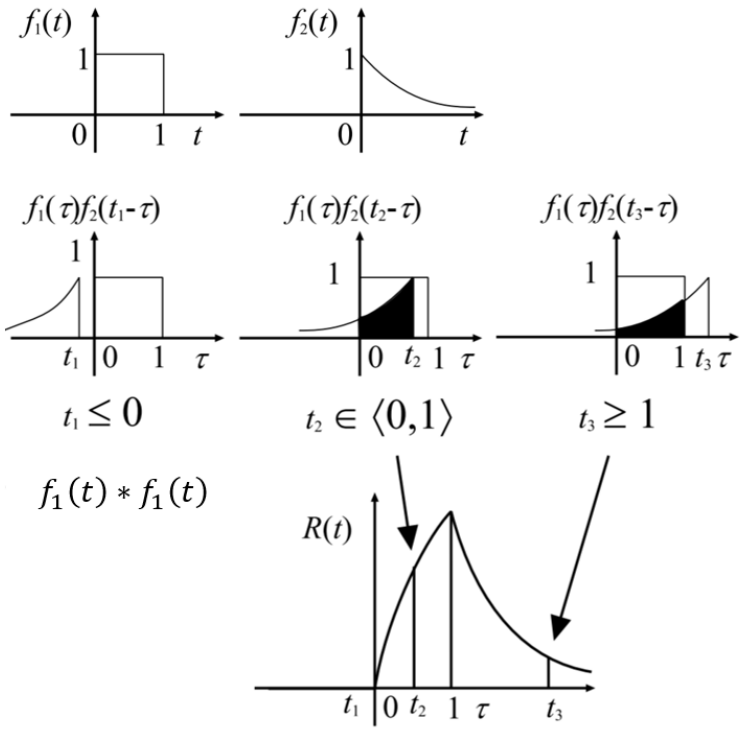


Figure 2-12: Convolution of signals in physical meaning **[24]**

The second alternative for convolution is an element-wise multiplication in frequency domain. Firstly, forward Fourier transform converts the first and the second signals into Fourier coefficients. The lists of these two coefficients are multiplied element-wisely, meaning that the multiplication is done index by index. The formula is given in **Equation 2-10**.

$$F\{f * g\} = F\{f\} \cdot F\{g\} \qquad \textbf{(Equation 2-10)}$$

In this study, the chosen technique is element-wise multiplication in frequency domain due to calculation efficiency, especially for a signal with large sample number [25]. However, drawback is complexity of the programming algorithm to cope with the operations.

2.2.2.5 Filters

Filter, or filter kernel, is the second signal in convolution operation as mentioned previously. The filter is used generally for 2 purposes [26]. First is signal separation, to characterize and extract information of interest embedded in the signal. And second is signal restoration, to repair signal which is originally distorted by poor data collection, such as noisy audio or blurry image.

The filters can be categorized to 4 basic types. Each of which are designed for particular purposes. Their application and visualization of these 4 filters are given in **Table 2-3** and in **Figure 2-13** respectively.

Table 2-3: Filter types and definition

Digital Filters	Applications
Low-pass filter	Allow low frequency to pass, and block high frequency not to pass
High-pass filter	Allow high frequency to pass, and block low frequency not to pass
Band-pass filter	Allow frequency band in the middle to pass
Band-reject filter	Block frequency band in the middle not to pass

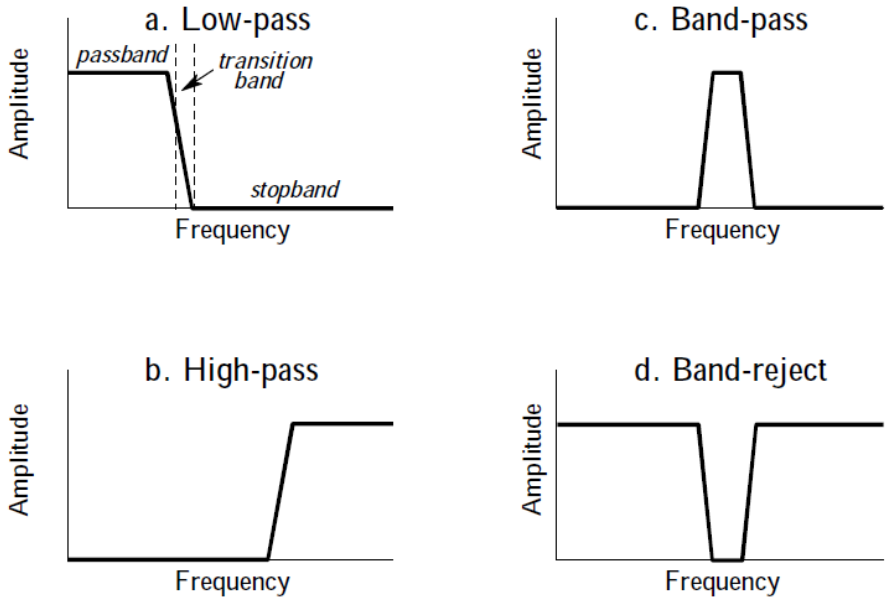


Figure 2-13: Digital filters in frequency domain **[26]**

To construct these filters, the general approach is to start by computation of low-pass filter from the given equation. Afterwards, convert it to other filters. For example, high-pass filter is computed by subtracting the maximum value of low-pass response by the low-pass response itself. Or another example, band-pass filter is constructed by convolution of low-pass and high-pass response which are preliminary constructed at designated cut-off frequency.

In this study, low-pass filter is deployed to extract waviness which contains characteristic of high wavelength / low frequency. And high-pass filter is deployed to extract roughness which contains characteristic of low wavelength / high frequency.

2.2.2.6 Surface reconstruction

Generally, surface reconstruction is computed by inverse Fourier transform of the modified frequencies. However, with the objective of this study to reconstruct the roughness surface in FEM simulation, in which the extensive mathematical operation is limited, the simple alternative technique of surface reconstruction is required. As mentioned in section 2.2.2.1, the superposition of sinusoids is utilized.

For surface curve in 1D, the reconstruction equation is given by **Equation 2-11**. Three variables, which are amplitude (A), frequency (f) and phase (θ), are essential to establish one certain sinusoidal component. After computation of sinusoids at all frequencies, the final surface is the summation of those all.

$$z = Acos(\omega x + \theta) = Acos(2\pi f x + \theta) \quad \textbf{(Equation 2-11)}$$

where x = list of spatial domain [distance unit]

ω = frequency [rad/s]

f = frequency [inverse unit of distance]

Analogue to surface in 2D, the reconstruction equation is given by **Equation 2-12** [27]. But in this case, 4 variables are involved, which are Amplitude (A), frequency in x-direction (f_x), frequency in y-direction (f_y) and phase (θ). The surface is computed by summation of sinusoidal surfaces at all characteristic frequencies. For illustration, the sinusoidal characteristic surface of a x and y frequency pair is shown in **Figure 2-14**.

$$z = A\cos\big((\omega_x x + \omega_y y) + \theta\big) = A\cos\big(2\pi\big(f_x x + f_y y\big) + \theta\big) \quad \textbf{(Equation 2-12)}$$

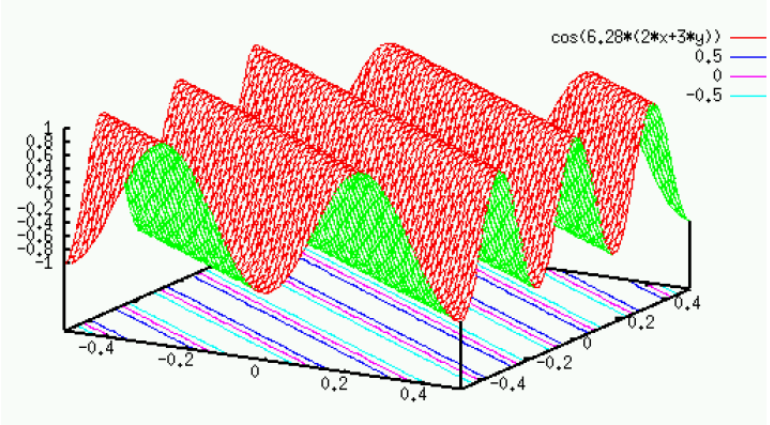


Figure 2-14: Sinusoidal curve at a pair of characteristic x and y frequencies

2.2.3 Python and scipy.fftpack module

According to the objective of this study, to characterize material topography of the measured data using surface filtration techniques, one can see that large amount of data and variety of complex mathematics is necessary to be involved. The high proficient tool sufficiently to meet the problem demands is programming.

In this study, the chosen programming language is Python, since it is built up with high level of data structure - conveniently to deal with huge amount of data and operate with simple syntax - easily for coding and interpretation. Besides it allows user

to freely access and distribute the written code. Furthermore, the most outstanding benefit of Python is, it widely supports modules and libraries which are conveniently called for complex or reproducible operations [28]. The module which is extensively used in this study's model is scipy.fftpack.

Scipy.fftpack is a Python module which is designed to support Discrete Fourier Transform problem specifically [29]. Instead of the complicated mathematical equations mentioned in previous sections, the operations can be successively done by few lines of code. The command examples are as follows:

- scipy.fftpack.fft for fast Fourier transform
- scipy.fftpack.ifft for inverse fast Fourier transform
- scipy.fftpack.fftfreq for frequency axis assignation
- and etc.

It's indeed helpful for user to learn the application of Python and its relevant library in signal processing, in order to apply the aforementioned techniques for advance applications.

3 Materials and Measurement

In this chapter, information about surface datasets to test functionality of the developed model are given. In addition, specification of the instrument using for surface measurement is described.

3.1 Materials

To examine the model's functionalities, 6 datasets of the measured surface topography are selected. Datasets detail is given in **Table 3-1**. It was intended to select variety of surface conditions to test coverage validity of the model.

Table 3-1: List of surface datasets

Dataset no.	Material	Surface condition	No. of measuring area	Resolution (µm * µm)
1	XAR 450 Martensitic steel	Normal surface with no process (1)	10 (2 and 10 are selected for one-file and multi-file model validation)	1.5625*1.5625
2	XAR 450 Martensitic steel	Normal surface with no process (2)	10 (2 are selected for model validation)	1.5625*1.5625
3	DP 1000 Dual phase steel	Drill surface	1 (1 is selected for model validation)	1.5625*1.5625
4	DP 1000 Dual phase steel	Mill surface	4 (2 are selected for model validation)	1.5625*1.5625
5	DP 1000 Dual phase steel	Water jet surface	4 (2 are selected for model validation)	1.5625*1.5625
6	DP 1000 Dual phase steel	Wire cut surface	5 (2 are selected for model validation)	1.5625*1.5625

3.2 Measurement

The instrument which was used to measure surface topography of the materials mentioned in previous section is White-light confocal microscope (NanoFocus). With the instrument, high resolution analysis is conducted to reveal surface parameters, such as roughness, tribology or surface properties in micro and nanometer scale [30]. The external appearance of the instrument is shown in **Figure 3-1**.

The measurement principle is based on CMP technology (Confocal-Multi-Pinhole), by which the LED source is focused through pinhole disc and lens onto the sample surface. This action generates reflection of light falling on the camera and consequently transforms to surface information.

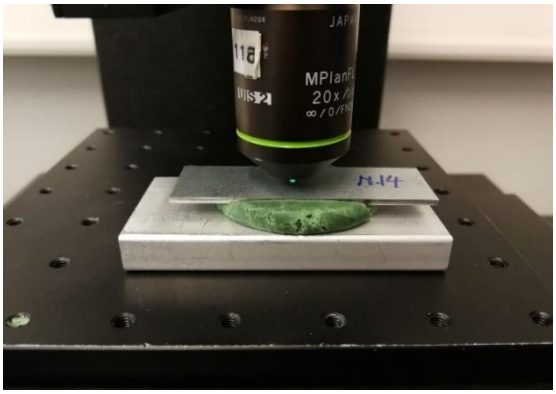


Figure 3-1: White-light confocal microscope

On one material sample, several small regions are selected to represent overall surface condition. Area of one measuring region is 1.475 x 1.475 mm. For the first dataset, 10 points are selected dispersedly over the surface. An illustration of the measuring locations is displayed in **Figure 3-2**.

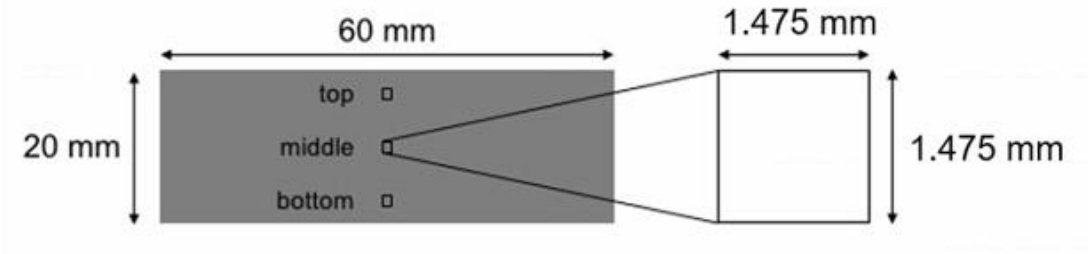


Figure 3-2: Surface measuring location

4 Methodology

Aim of this study is to develop programming model for extracting roughness from original measured surface. To reach the goal, several steps are employed. The main process to handle one-file dataset measured from one region of material sample is given by flowchart in **Figure 4-1**. The procedure consists of 3 main steps, from data pre-processing, surface transformation, then surface reconstruction. Objective of the first step, data pre-processing, is to primarily clean up data input in manageable form and eliminate error from measurement. While objective of the second step, surface transformation, is to distinguish surface topography and separate its characteristics into 2 parts, which are waviness, roughness. Dominant surface is a subsequent simplified version of roughness for practical application in FEM. Lastly, purpose of the final step, surface reconstruction, is to verify correctness of the extracted variables and to ensure validity of the surface reconstruction formulas.

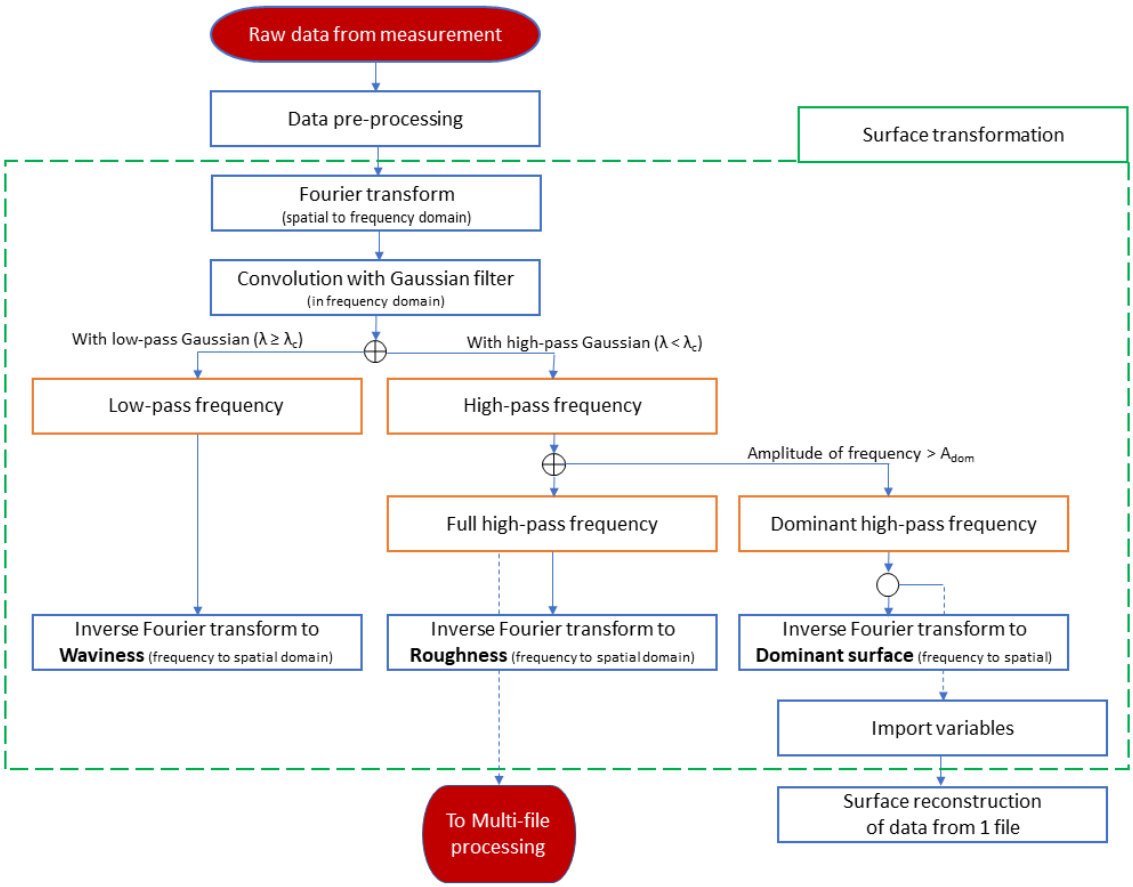


Figure 4-1: Flowchart procedure of one-file surface dataset processing

However, in one material sample, more than one region are measured to ensure coverage and representativeness of overall surface condition. This multi-area measurement gives benefit of bias reduction. To solve the problem, this study develops the model for multi-file processing additionally. The variables from surface transformation step of one-file processing is passed to the model, encouraged the data from different regions to be included in surface characterization operation. The detail explanation of both one-file processing and multi-file processing will be discussed in the next chapter.

5 Modeling

In this chapter, detail procedure of the surface characterization model is explained. The chapter outlines into 3 parts, data pre-processing, surface transformation and surface reconstruction. For demonstration, surface dataset 1 – martensitic steel XAR 450, unprocessed surface condition is selected as input to the developed model.

5.1 Data pre-processing

In real world, raw data usually contains inconsistency or error according to measuring instrument limitation. In most cases, raw data is in unarranged form, therefore data pre-processing is necessary.

In addition, this section includes parameter assignation step. Since these parameters influence directly to the result, one should pay attention when determining them. The process of data pre-processing is shown by flowchart in **Figure 5-1**.

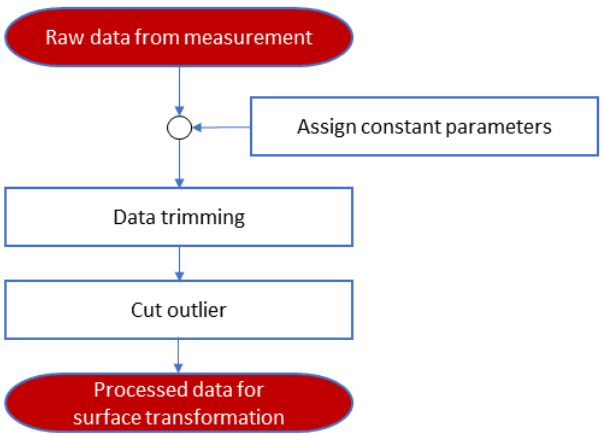


Figure 5-1: Flowchart procedure of pre-processing step

5.1.1 Parameter assignment

The constant parameters required to assign in advance are listed in **Table 5-1**.

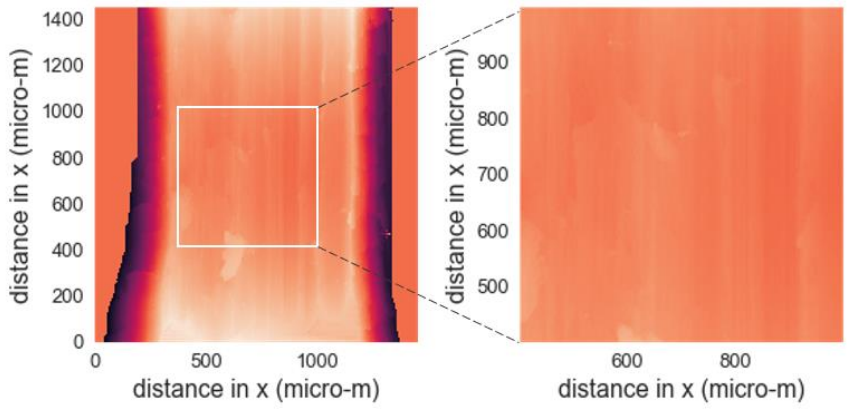
Table 5-1: Definition of the constant parameters

Parameters	Definition
outlier_angle (angle of outlier)	Angle between different height of the adjacent measuring points to horizontal distance, by which if the next height exceeds this angle, it will be rounded down to make the angle be at outlier_angle maximum.
lambda_c (cut-off wavelength)	Cut off wave length of Gaussian filters. This parameter is used as the cut-off threshold to differentiate frequency range of roughness and waviness.
ratio_amp_dom2max_1D (ratio of dominant amplitude to max amplitude in 1D)	Ratio percent of cut-off dominant amplitude to the maximum of frequency coefficient amplitude. This means frequency comprising amplitude higher than this value is counted as dominant frequency. The parameter is used in 1D dominant frequency selection.
accept_dom_rSquare_2D (acceptable R-squared of dominant surface to roughness in 2D)	Acceptable R-squared is a statistical value comparing similarity of dominant surface to roughness. Analogue to 1D, but this parameter automatically assigns cut-off dominant amplitude. The parameter is used in 2D dominant frequency selection.

5.1.2 Data trimming

In this step, data undergoes trimming operation to make measured point number to be equal in both spatial x and y direction. This makes further operations convenient, since the data is symmetric and requires only one set of Gaussian filter, and one set of spatial and frequency axis. In addition, this helps the addition of frequency coefficient in multi-files processing possible, because all the data are in the same dimension.

Additionally, this model provides possibility to trim off the irrelevant area because of extensive measurement over the area of required data. This happens especially in sample whose shape is difficult to be measured, for example drilling surface. The trimmed area of drilling surface raw data is displayed in **Figure 5-2**.



(a) (b)

Figure 5-2: Trimming of raw surface irrelevant area

(a) Raw surface
(b) Trimmed surface

5.1.3 Outlier cutting

According to limitation of surface measuring instrument, error of the measured data can occur in form that some measured heights represent outbound value compared to the neighbors. Regarding to the constant parameter outlier_angle, the program calls row from rows of 2D dataset and compared the measured height element-wisely to the previous. If the calculated angle exceeds the threshold, the height at that certain point is counted as measurement error and is rounded down to the maximum value at maximum outlier_angle. This process is done row-wisely first. Afterwards, it is performed again column-wisely. It is ensured that the order of processing in row- and column-wise does a negligible difference.

Equation to calculate the relevant angle between 2 adjacent points is given by **Equation 5-1**. The formulas to correct outbound data are given in **Equation 5-2**. It provides 2 possibilities that the current height is higher or lower in value than the previous.

$$\theta = \tan^{-1}\left(\frac{h}{w}\right) \qquad \textbf{(Equation 5-1)}$$

$$h = \text{height}[i+1] - \text{height}[i]$$

$$w = \text{spatial\_location}[i+1] - \text{spatial\_location}[i]$$

$$\text{new\_height} = \begin{cases} \text{height}[i] + w \cdot \tan(\text{outlier\_angle}) & \text{if } \theta > +\text{outlier\_angle} \\ \text{height}[i] - w \cdot \tan(\text{outlier\_angle}) & \text{if } \theta < -\text{outlier\_angle} \end{cases}$$

(Equation 5-2)

5.2 Surface transformation

As mentioned earlier, the real surface always contains overmuch information. Some is in interest for FEM simulation, while some is not. Thus, the surface transformation operation is needed to extract the interested characteristics out. In this case, it is the simplified roughness, or dominant surface. The detail procedure of this surface transformation is illustrated by flowchart in **Figure 5-3**.

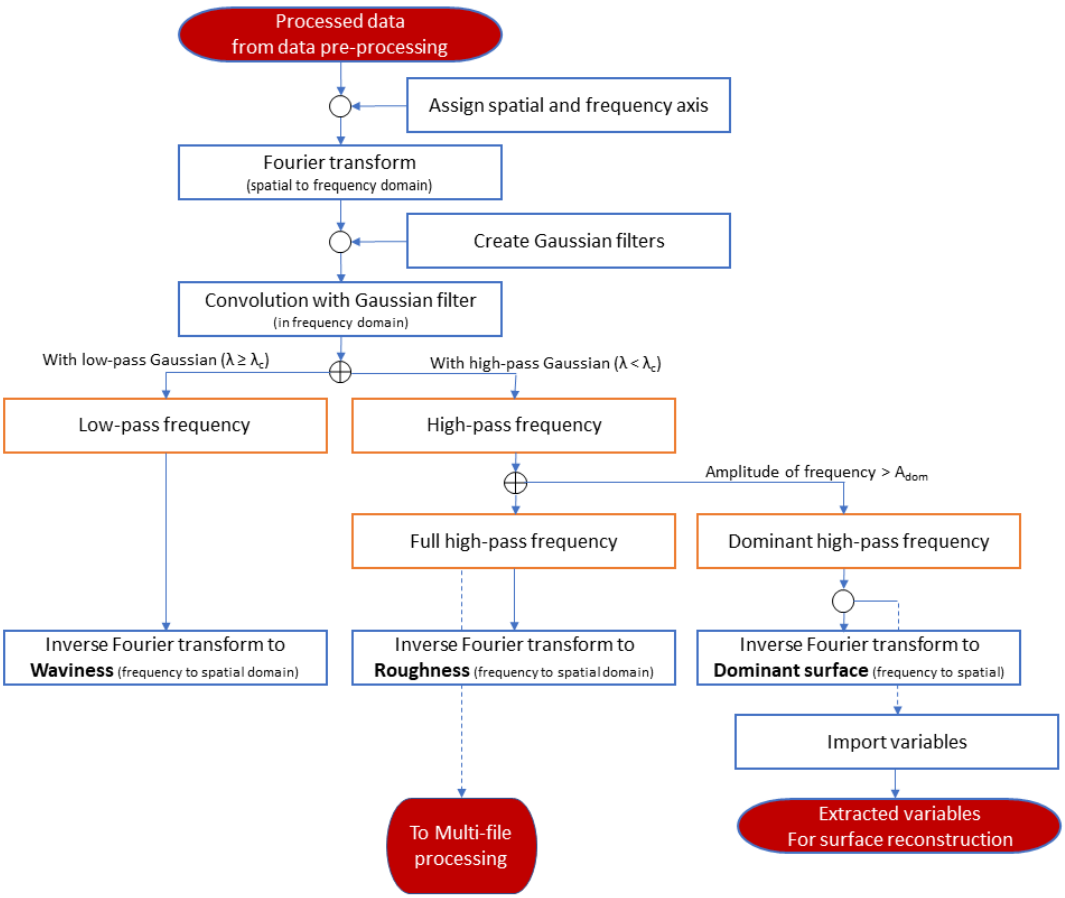


Figure 5-3: Flowchart procedure of surface transformation step

The process starts from spatial axis and frequency axis assignment, then Gaussian filter creation. The spatial axis is used for result illustration in spatial domain, while the frequency axis is used for result illustration in frequency domain. All of which are created at the beginning of the program.

Furthermore, the extraction operation is conducted. The data is transformed firstly to frequency domain by forward Fourier transform. From this point, the transformed frequency is handled by convolution. The resulting signal after low-pass Gaussian filter is waviness. In contrast, the resulting signal after high-pass Gaussian

filter is roughness. In order to display the outcomes in spatial domain, inverse Fourier transform is applied to convert the signal back.

Though the surface characteristic of this study is roughness, the data is still overmuch to process in FEM simulation. It is necessary that only the significant frequencies holding high amplitude are selected. The surface constructed by these dominant frequencies is called dominant surface. Subsequently, the left frequency coefficients are mathematically handled to be in variable format, then imported to .csv file for application of surface reconstruction.

The detail explanation with images and diagrams for demonstrating are given in next section. The topic starts firstly from 1D and 2D of one-file processing, afterwards the process in multi-file processing. This outline structure is intended for building up the understanding progressively.

5.2.1 1D-data of one-file processing

In this section, steps to process dataset measured from one region of material are explained. There are 6 steps as follows.

Step 1: Assign spatial and frequency axis

Spatial axis is as simple as a distance list at which the height data are located. It starts from 0, then increase its value with equal step (samp_space), till it ends at total length of the sample. Spatial axis example of the demonstrated data is shown in **Figure 5-4**.

```
print(samp_space)
print(data_list[0:5], "...", data_list[-5:])
```

```
1.5625
[0.0, 1.5625, 3.125, 4.6875, 6.25] ... [3420.31, 3421.87, 3423.44, 3425.0, 3426.56]
```

Figure 5-4: Spatial axis of the demonstration data

Referring to section 2.2.2.3 Frequency domain, the frequency axis can be simply computed using command scipy.fftpack.fftfreq in python. However, there are different arrangements of frequencies in list. The explanatory about the list arrangement and its application are given in **Table 5-2**.

Table 5-2: Definition and application of frequency axis in different arrangements

List name	List arrangement	Application
freq_full	Start from 0 to the frequency at half-range followed by minus of the frequency at half-range increasing to the last number before 0	Programming initial axis assignation
freq_full_shift	freq_full but sorting values from smallest to highest, 0 is at the middle	For matching with amplitude, so that the low frequency is at the middle when display
freq_half	Positive half of freq_full, negative half of freq_full is discarded	For display only in positive Right-hand side of frequency

Step 2: Create Gaussian filter

Gaussian filter plays important role in the separation operation of surface characteristics. The equation of 1D Gaussian filter for surface metrology application are proposed by Y. B. Yuan [31]. Formula of low-pass Gaussian is shown in **Equation 5-3**, while the formula of high-pass Gaussian is shown in **Equation 5-4**.

$$L(\lambda_c/\lambda) = e^{-\pi(\alpha\lambda_c/\lambda)^2} \quad \textbf{(Equation 5-3)}$$

$$H(\lambda_c/\lambda) = 1 - L(\lambda_c/\lambda) \quad \textbf{(Equation 5-4)}$$

$$f_c = \frac{1}{\lambda_c} \quad \textbf{(Equation 5-5)}$$

where α is a constant equal to 0.4697

λ is a list of wavelength in frequency domain [distance unit]

λ_c is a cut-off wavelength [distance unit]

f_c is a cut-off frequency [inverse of distance unit]

By input frequency axis to the formulas and set cut-off wavelength to 200 µm, the resulting high- and low-pass Gaussian filter are displayed in **Figure 5-5**. One can observe that 2 lines of filters cut each other at cut-off frequency at value 0.005 μm^{-1}, calculated by **Equation 5-5**.

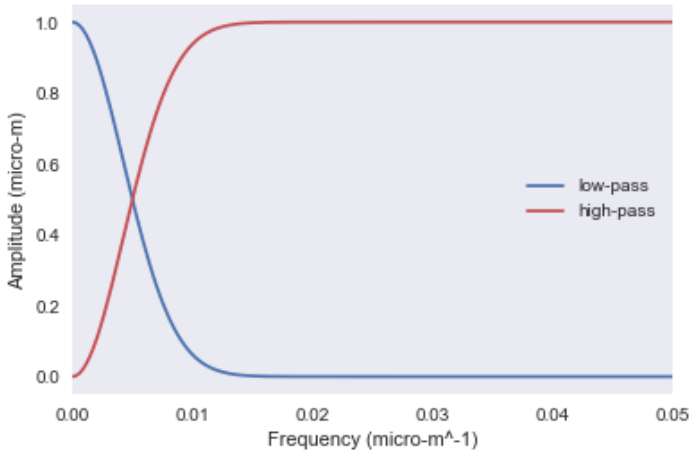


Figure 5-5: High-pass and low-pass Gaussian filter in 1D

Step 3: Forward Fourier transformation

In this step, fast forward Fourier transform (FFT) is conducted to transform surface information from spatial domain to frequency domain. In practical, Python has provided useful tool of scipy.fftpack.fft module. For 1D forward Fourier transform, the command is scipy.fftpack.fft.

Result of frequency coefficient in form of amplitude, is displayed in **Figure 5-6**. In specific, **Figure 5-6** (a) displays the full range of positive and negative frequency with zero at the middle. **Figure 5-6** (b) illustrates only positive-half side, by which the amplitude has been multiplied by 2 to compensate negative-half. **Figure 5-6** (c) is **Figure 5-6** (b) which covers the smaller frequency range between 0.000 to 0.025 μm^{-1} in order to show discrete characteristic of frequencies.

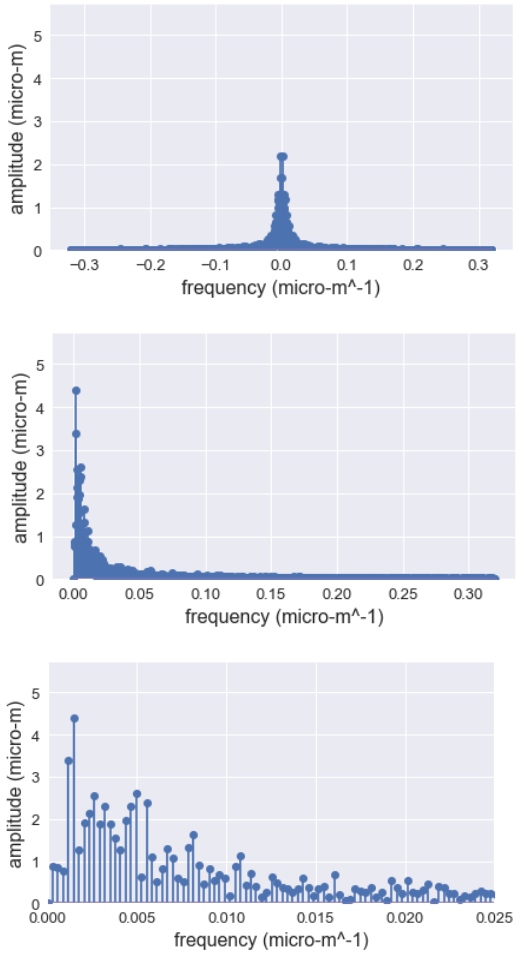


Figure 5-6: Fourier coefficient of 1D surface data

(a) Full Fourier coefficient

(b) Right-hand side (RHS) of Fourier coefficient with correction

(c) RHS of Fourier coefficient with correction covering smaller range

Step 4: Convolution and Inverse Fourier transform

In this step, convolution is applied between Fourier coefficient obtaining from step 3 and the Gaussian filter created in step 2. The convolution is an element-wise multiplication between 2 lists in frequency domain, by which they are prior assigned in the same length. Goal of the convolution is to separate surface characteristics into waviness and roughness.

From the unprocessed Fourier coefficient shown in **Figure 5-7** (a), low-pass Fourier coefficient after convolution of (a) with 1D low-pass filter is shown in **Figure 5-7** (b), and high-pass Fourier coefficient after convolution of (a) with 1D high-pass filter is shown in **Figure 5-7** (c).

Afterwards, inverse fast Fourier transform (IFFT) is conducted using the command scipy.fftpack.ifft. This operation transforms information from frequency domain back to spatial domain. **Figure 5-7** (d) is a spatial domain resulting from IFFT operation on unprocessed Fourier coefficient. One can see that every surface detail remains unchanged comparing to the surface before FFT. In **Figure 5-7** (e), this waviness is a result of IFFT on low-pass Fourier coefficient. The detail fluctuation is discarded, remaining only the major change in height. In contrast, **Figure 5-7** (f) shows the roughness as result of IFFT of high-pass Fourier coefficient. All detail fluctuation of surface is kept, while the major change in height is discarded.

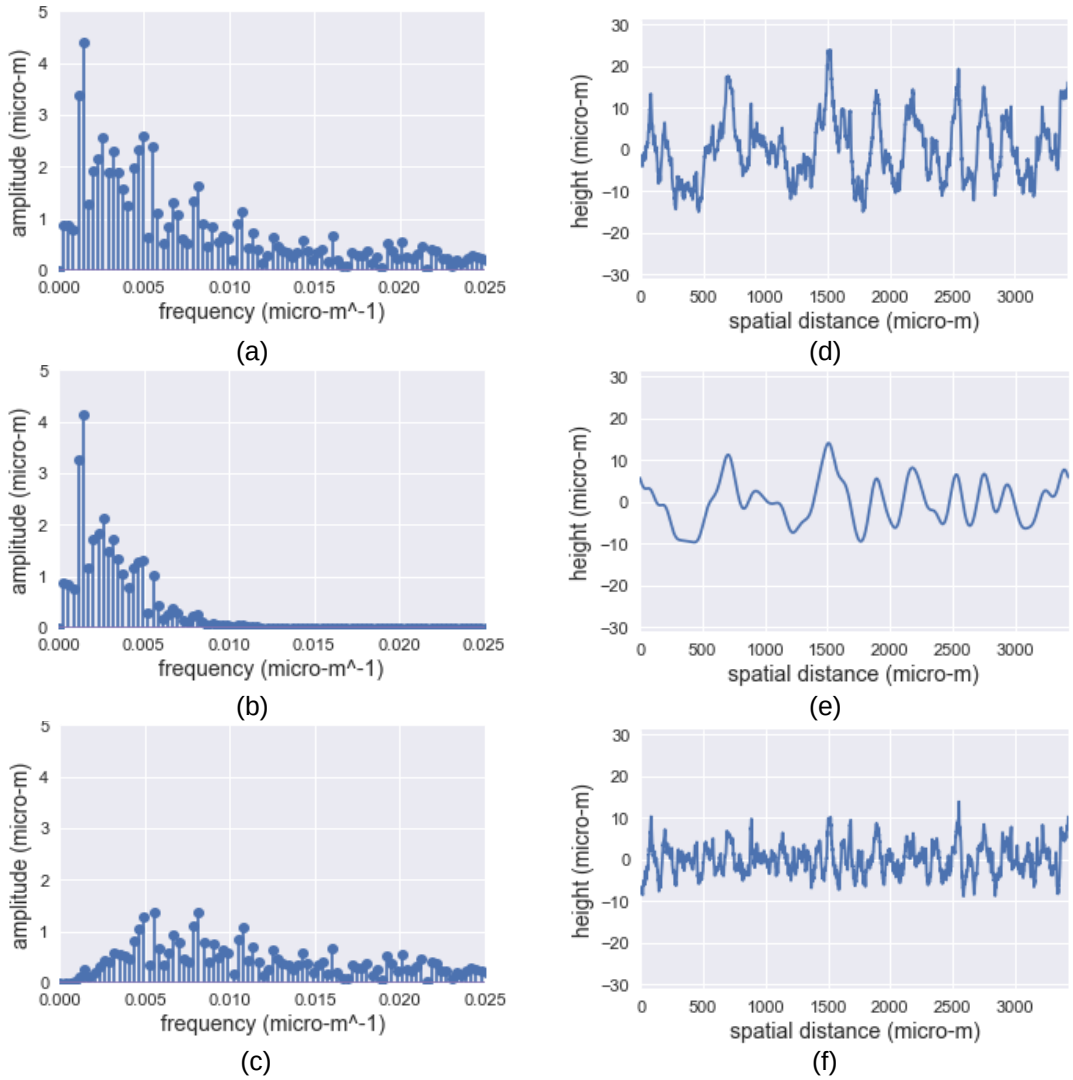


Figure 5-7: Inverse surface of convolved frequency coefficient

(a) Unprocessed Fourier coefficient
(b) Low-pass Fourier coefficient
(c) High-pass Fourier coefficient
(d) Original surface by IFFT of unprocessed Fourier coefficient
(e) Waviness surface by IFFT of low-pass Fourier coefficient
(f) Roughness surface by IFFT of high-pass Fourier coefficient

Step 5: Dominant frequency selection

Since the amount of frequency composed in low-pass Fourier coefficient is overmuch to create the surface in FEM software, the dominant frequency selection is necessary to extract only the frequencies comprising high amplitude.

For 1D dominant frequency selection, cut-off amplitude is determined by ratio to the maximum amplitude of frequency coefficient in whole range. In this demonstration example, the cut-off ratio is assigned to 50%. From high-pass Fourier coefficient in **Figure 5-8** (a), frequencies which hold amplitude higher than 0.69 μm are selected as dominant frequency shown in **Figure 5-8** (b). The simplified surface by IFFT operation on the selected dominant frequencies is illustrated in **Figure 5-8** (d).

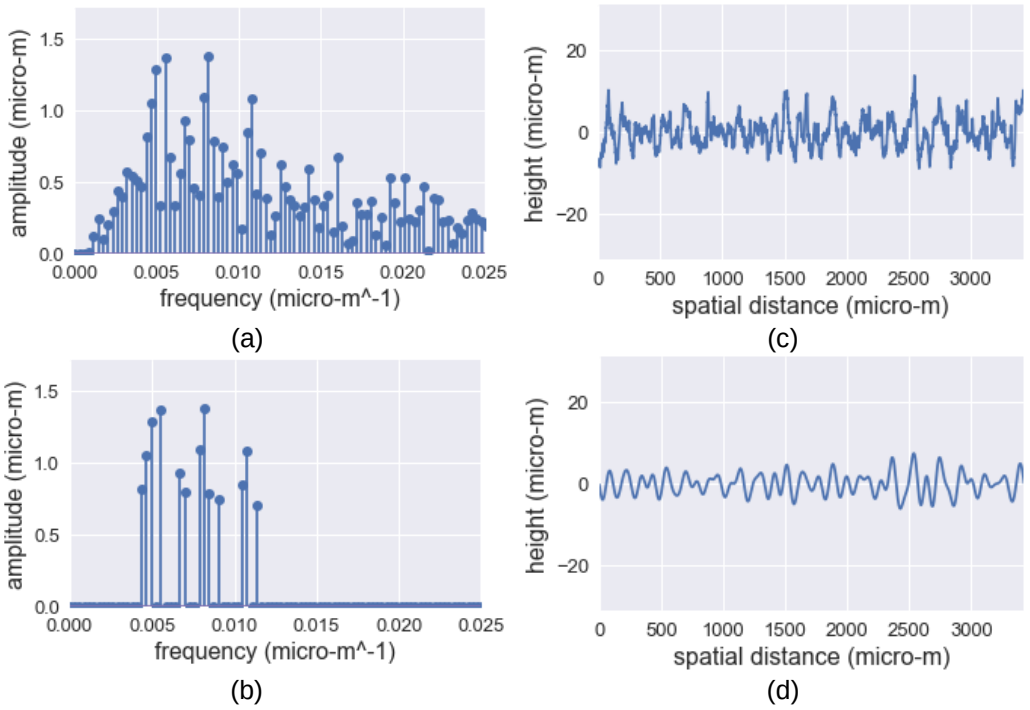


Figure 5-8: Simplified surface by dominant frequency selection

(a) Low-pass Fourier coefficient
(b) Dominant Fourier coefficient selected from low-pass Fourier coefficient
(c) Roughness surface by IFFT of low-pass Fourier coefficient
(d) Simplified surface by IFFT of dominant Fourier coefficient

Step 6: Import variables

To make use of the dominant Fourier coefficient in FEM software, the essential variables are exported. Three variables, which are amplitude (A), frequency (f) and phase (θ), are essential for surface in 1D. The frequency value is acquired by matching position of Fourier coefficient to frequency axis, while the amplitude and phase are acquired by calculation of imaginary Fourier coefficient. The formulas have been stated by **Equation 2-5** and **Equation 2-6** in chapter 2.

Since these three variables are the identity of each dominant frequency, thereby the amount of three-variable-dataset is equal to the number of the dominance. In this example, there are 13 dominant frequencies in positive-half range, hence 13 sets of extracted variables are collected as shown in **Table 5-3**. This table is imported to .csv file for further surface reconstruction in FEM.

Table 5-3: Variables of dominant frequency in 1D extracted to .csv file

Frequency (μm^{-1})	Frequency coefficient	Amplitude (μm)	Phase (rad)
0.004376	0.670 - 0.461j	0.815	-0.603
0.004667	0.793 + 0.686j	1.049	0.713
0.004959	-1.133 + 0.607j	1.286	2.650
0.005542	0.918 - 1.006j	1.363	-0.831
0.006709	-0.353 + 0.860j	0.930	1.961
0.007001	-0.433 + 0.664j	0.793	2.149
0.007876	0.263 + 1.059j	1.092	1.327
0.008168	-1.360 + 0.202j	1.376	2.994
0.008459	0.252 + 0.743j	0.786	1.243
0.009043	0.499 + 0.550j	0.743	0.835
0.010501	0.047 + 0.845j	0.847	1.515
0.010793	-1.064 - 0.159j	1.077	-2.993
0.011376	0.692 -0.106j	0.701	-0.153

5.2.2 2D-data of one-file processing

In this section, detail steps to process one-file data for 2D surface topography are explained. There are 6 steps in total.

Step 1: Assign spatial and frequency axis

The axes in 2D surface modeling are identically same as the one used in 1D procedure. Because 2D raw data has been primary trimmed its shape to have same dimension on both sides, only one set of spatial axis and frequency axis is required.

Step 2: Create Gaussian filter

Beside the formula of 1D Gaussian filter, Y. B. Yuan, et al. has proposed the formula of 2D Gaussian filter for surface roughness metrology application [32]. The equation of low-pass 2D Gaussian is shown in **Equation 5-6** and the high-pass 2D Gaussian is shown in **Equation 5-7**.

$$L(\lambda_x, \lambda_y) = exp\left\{-\pi\beta\left[\left(\frac{\lambda_{xc}}{\lambda_x}\right)^2 + \left(\frac{\lambda_{yc}}{\lambda_y}\right)^2\right]\right\} \quad \textbf{(Equation 5-6)}$$

$$H(\lambda_x, \lambda_y) = 1 - L(\lambda_x, \lambda_y) \quad \textbf{(Equation 5-7)}$$

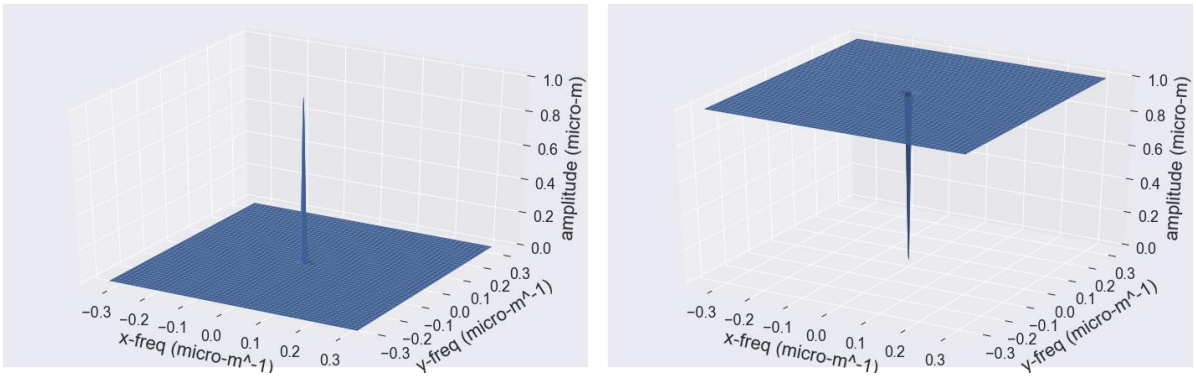
where α is a constant equal to 0.4697

β is a constant equal to 0.2206

λ_{xc} are cut-off wavelength in x direction

λ_{yc} are cut-off wavelength in y direction

The plot of 2D Gaussian filters setting cut-off wavelength at 200 μm are displayed in **Figure 5-9**.



(a) (b)

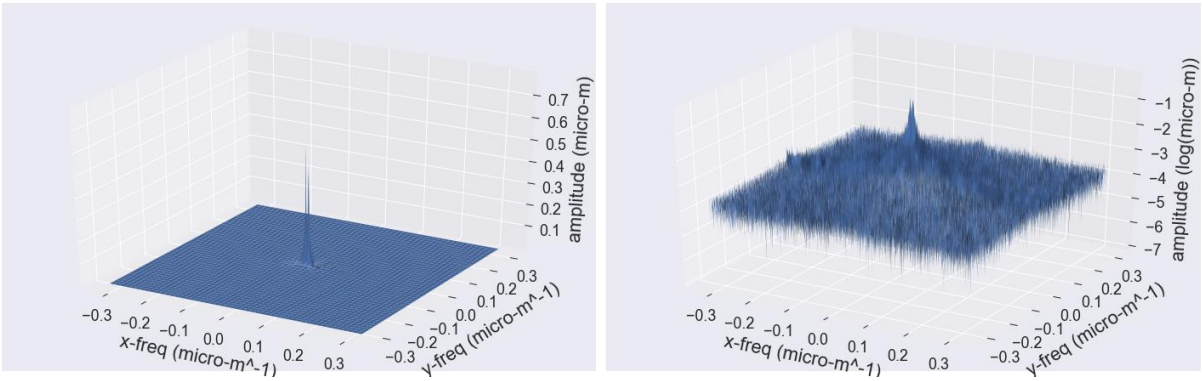
Figure 5-9: High-pass and low-pass Gaussian in 2D

(a) Low-pass Gaussian in 2D
(b) High-pass Gaussian in 2D

Step 3: Forward Fourier transformation

In this step, fast forward Fourier transform (FFT) is conducted to transform surface information in spatial domain to the characteristic frequencies in frequency domain. Similar to the operation in 1D, scipy.fftpack module is called and the utilized command is scipy.fftpack.fft2.

The result of 2D frequency coefficient is illustrated in **Figure 5-10**. **Figure 5-10** (a) displays the original amplitude of frequency coefficient, whose fine difference of amplitude is hardly observed. In **Figure 5-10** (b), Fourier coefficient is enhanced by log10 for the purpose to amplify amplitude height difference. However, it is intended only for illustration, not for practical function after.



(a) (b)

Figure 5-10: Frequency coefficient of 2D surface data

(a) Frequency coefficient
(b) Enhanced frequency coefficient in log-scale

Step 4: Convolution and Inverse Fourier transform

In this step, convolution is conducted by element-wise multiplication of the Fourier coefficient obtaining from step 3 and the Gaussian filter computing in step 2. The display plot starts firstly from amplitude of unprocessed Fourier coefficient in **Figure 5-11** (a). The low-pass Fourier coefficient after multiplying the unprocessed with low-pass Gaussian filter is displayed in **Figure 5-11** (b). Lastly, the high-pass Fourier coefficient after multiplying the unprocessed with high-pass Gaussian filter is displayed in **Figure 5-11** (c).

Subsequently, inverse fast Fourier transform (IFFT) is conducted with the command scipy.fftpack.ifft2. The spatial surfaces transformed back from frequency domains are the original, waviness and roughness surface shown in **Figure 5-11** (d), (e) and (f). They are retrieved from the unprocessed, low-pass and high-pass Fourier coefficient respectively.

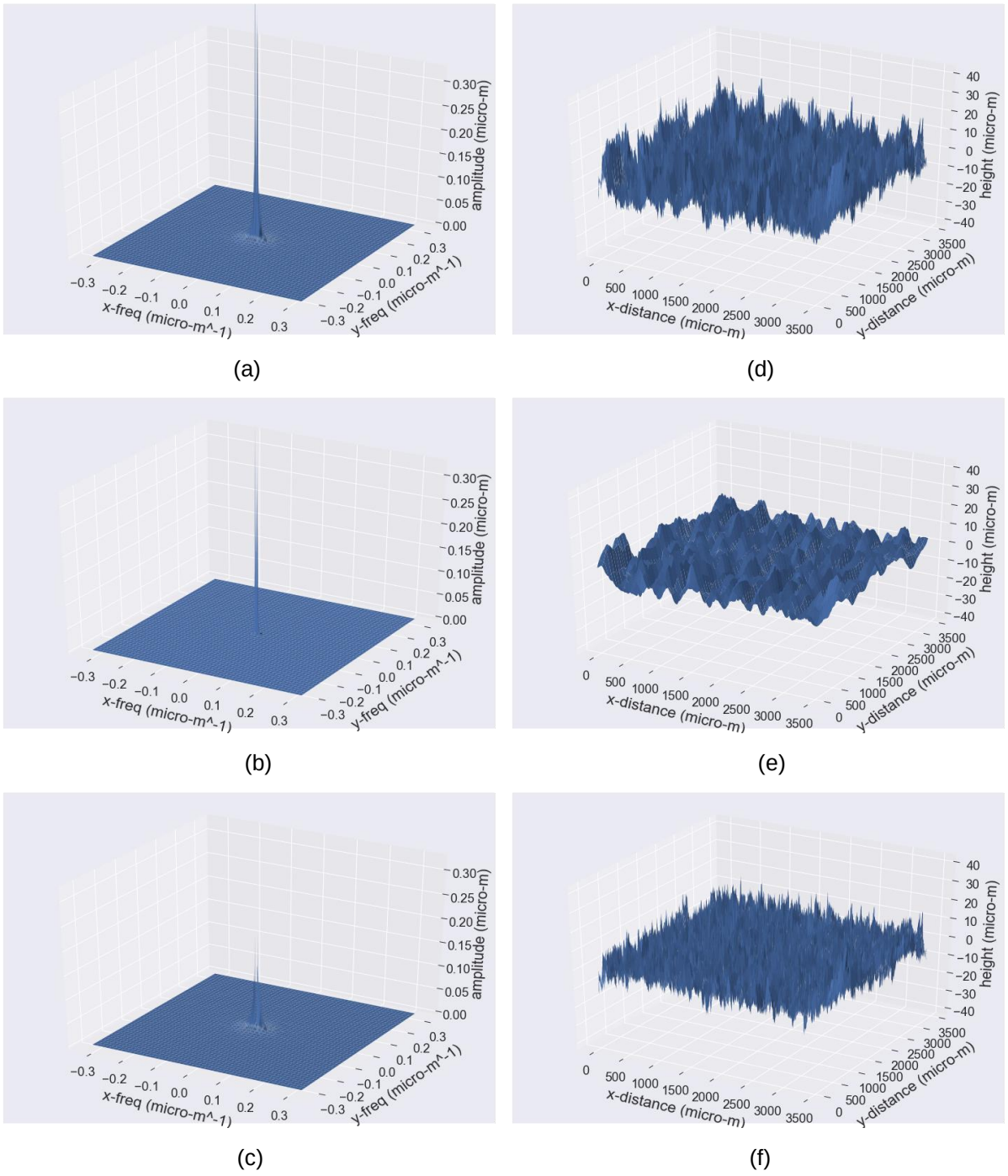


Figure 5-11: Convolved Fourier coefficient and the inverse surfaces (3D display)

(a) Unprocessed Fourier coefficient
(b) Low-pass Fourier coefficient
(c) High-pass Fourier coefficient
(d) Original surface by IFFT of unprocessed Fourier coefficient
(e) Waviness surface by IFFT of low-pass Fourier coefficient
(f) Roughness surface by IFFT of high-pass Fourier coefficient

Step 5: Dominant frequency selection

In contrast to the dominant frequency selection in 1D, the operation in 2D is designed so that the selection is done automatically. The algorithm is conducted based on R-squared statistics, showing how the extracted dominant surface correlated to the roughness. Procedure detail is described step by step in this section.

As one might suspect, the number of selected dominant frequencies plays important role on the simplified extracted surface. In the meaning that, the less number of selected frequencies, the more deviation between the simplified surface and the roughness surface will appear. Therefore, this study appoints number of dominant frequencies as independent variable and the R-squared surface deviation as a dependent variable. Number of dominant frequencies in percentage are assigned at 0.0025, 0.005, 0.01, 0.02, 0.04, 0.06, 0.08, 0.10, 0.20, 0.40, 0.60, 0.80, 1.00, 1.50 to 2.00% for 15-loop experiments.

Speaking about surface extraction procedure, the operation starts by selecting dominant frequencies from high-pass Fourier coefficient in the same amount as determined. Next step, the mentioned dominances are carried out for inverse fast Fourier transform and converted back to the simplified surface. Lastly, the resulting simplified surface is compared to the roughness using R-squared value. The statistical theory will be explained later in chapter 6, but for this model operation, the acceptable R-squared value is set at 0.5 minimum. After 15-loop experiments, the result was collected in **Table 5-4.** In addition, 1D section-cut of 2D surface are plotted in **Figure 5-12** to display influence of the dominant frequency amount to deviation of the simplified surface.

According to the results in **Table 5-4**, the least number of dominant frequencies contributing to R-squared value over 0.5 is 0.04% loop. Thereby, this loop is automatically selected for further surface operation. Dominant frequency amount is counted at 1925 in number with R-squared value at 0.5650.

Subsequently, these 1925 selected dominant frequencies are converted back to spatial domain by inverse fast Fourier transform. The resulting simplified surface and the roughness surface are shown in **Figure 5-13**. Comparing these two, one can observe that the dominant surface in **Figure 5-13** (b) comprises similar height distribution to the roughness in **Figure 5-13** (a). Only difference is that the small peak tips are attenuated.

Table 5-4: Simplified surface resulting by number of selected dominant frequency

Number of selected dominant frequency (%)	Number of selected dominant frequency (amount)	Similarity of roughness to simplified surface (R-squared)
0.0025	121	0.1403
0.0050	241	0.2298
0.0100	481	0.3344
0.0200	963	0.4508
0.0400	1925	0.5650
0.0600	2888	0.6288
0.0800	3851	0.6706
0.1000	4814	0.7017
0.2000	9627	0.7884
0.4000	19255	0.8549
0.6000	28882	0.8853
0.8000	38509	0.9037
1.0000	48136	0.9163
1.5000	72205	0.9358
2.0000	96273	0.9469

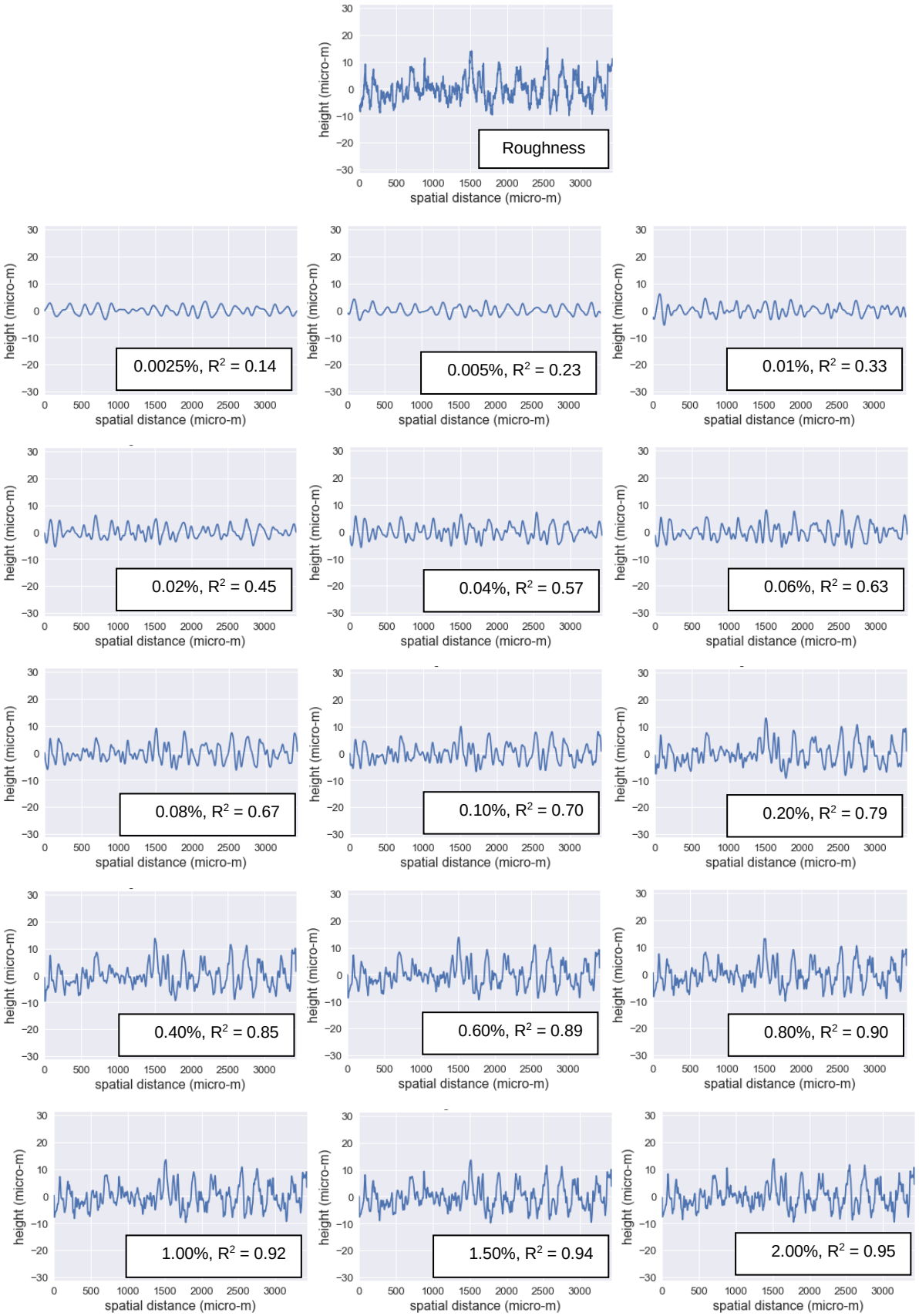
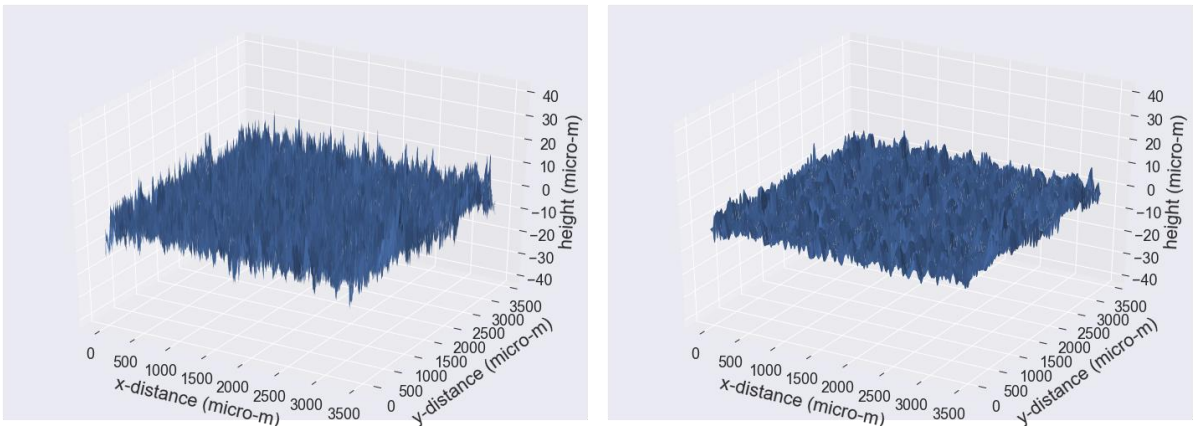


Figure 5-12: Comparison of roughness to the simplified surface varied by amount of selected dominant frequency at 0.0025, 0.005, 0.01, 0.02, 0.04, 0.06, 0.08, 0.10, 0.20, 0.40, 0.60, 0.80, 1.00, 1.50 and 2.00% respectively



(a) (b)

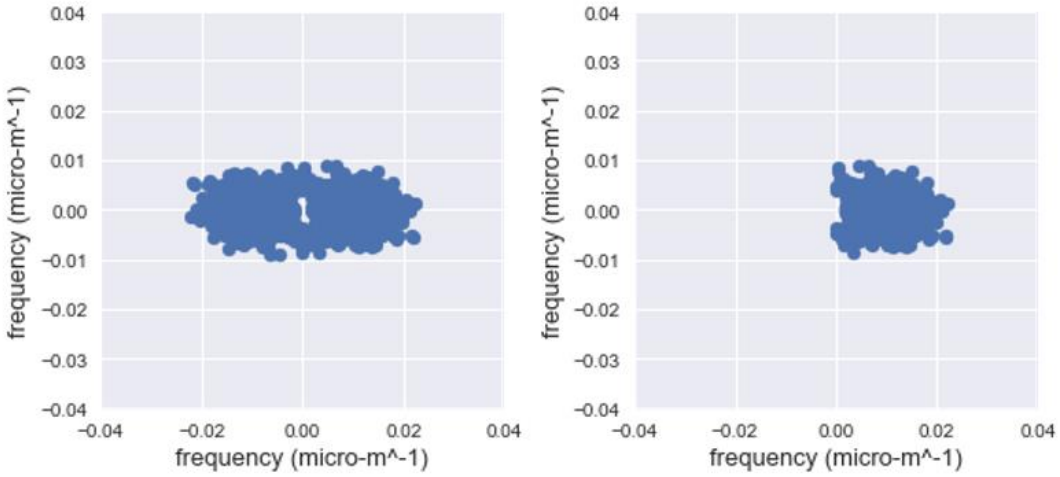
Figure 5-13: Roughness and the simplified surface

(a) Roughness surface by IFFT of high-pass Fourier coefficient
(b) Simplified surface by IFFT of dominant Fourier coefficient

Step 6: Import variables

As discussed in chapter 2, there are 4 essential variables for surface reconstruction in 2D. The variables consist of frequency in x-axis (f_x), frequency in y-axis (f_y), amplitude (A) and phase (θ). The equations to calculate amplitude and phase are also stated in the previous section. The amount of these 4-variables-sets is equal to the number of dominant frequencies.

According to the demonstration, 1925 frequencies are selected as the dominance. However, it shows symmetric behavior observed by scatter plot in **Figure 5-14** (a). Therefore, it is practical to reduce amount of the dominant frequencies down to the half at 965, as displayed in **Figure 5-14** (b). By this, amplitudes which are not located at the symmetry line have to be multiplied by 2 for correction. This significantly helps reducing surface computational time in FEM. Lastly, the half-symmetry variable sets are exported to .csv file. First 10 sets are listed in **Table 5-5** as an example.



(a) (b)

Figure 5-14: Scatter plot of dominant frequency in 2D

(a) Scatter plot of full dominant frequency
(b) Scatter plot of dominant frequency in symmetry-half

Table 5-5: Variables of dominant frequency in 2D extracted to .csv file

f_x (μm^{-1})	f_y (μm^{-1})	Frequency coefficient	Amplitude (μm)	Phase (rad)
0.000000	0.003792	-0.0087 + 0.0353j	0.0363	1.813
0.000000	0.004667	0.0215 + 0.0269j	0.0345	0.897
0.000000	-0.004670	0.0215 - 0.0269j	0.0345	-0.897
0.000000	-0.003790	-0.0087 - 0.0353j	0.0363	-1.813
0.000292	0.007876	0.0579 + 0.0411j	0.0710	0.618
0.000292	0.008459	0.0088 + 0.0692j	0.0698	1.444
0.000583	0.006126	-0.0452 - 0.0497j	0.0672	-2.310
0.001167	-0.005830	-0.0701 - 0.0038j	0.0702	-3.087
0.001167	-0.005250	-0.0633 - 0.0200j	0.0664	-2.836
0.001459	0.006126	-0.0419 - 0.0517j	0.0665	-2.252

5.2.3 Multi-file processing

As mentioned previously about the effort to combine information measured from different regions of material surface to ensure coverage representation of overall surface condition, this study has considered different alternatives to manage multiple files of raw data. In conclusion, the chosen method is the summation of high-pass frequency. The process in overview is shown by flowchart in **Figure 5-15**.

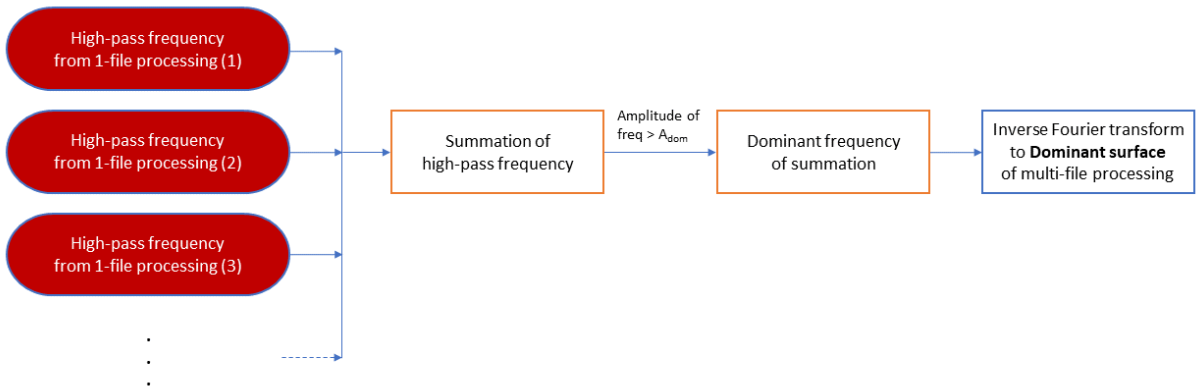
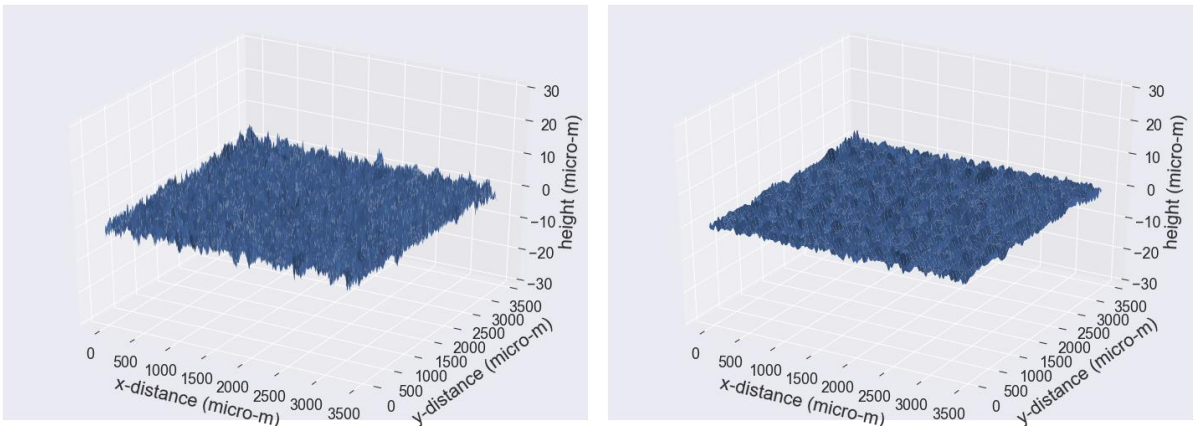


Figure 5-15: Flowchart of multi-file surface dataset processing

High-pass frequency is selected as an operational parameter to be passed to the multi-file processing model. The reason is that this parameter comprises of roughness frequency which is the characteristic surface in this study's focus. Firstly, all surface files measured from different regions are processed individually by one-file process modeling, resulting in high-pass Fourier coefficient matrix. Afterwards, the matrixes of all files are combined by direct addition of complex Fourier coefficient. This is done in the sense to specify how dominance of each x-y pair of frequency is, by its size of coefficient amplitude. Next step, selection operation of dominant frequency is conducted on this summed matrix. Analogue to the process in one-file processing, this multi-file processing is designed to assign cut-off amplitude automatically based on acceptable R-squared value. Last step, the dominant frequencies are converted back to spatial domain by inverse fast Fourier transform, giving the result which represents surface conditions from all different regions.

For demonstration, surface dataset 1 – martensitic steel XAR 450, unprocessed surface condition, is selected as the model input. Surface data are measured from 10 different regions and submitted to the process as described in previous paragraphs. The roughness surface computed by IFFT of direct high-pass frequency summation are displayed in **Figure 5-16** (a), while the simplified surface computed by IFFT of the selected dominant frequency are displayed in **Figure 5-16** (b).



(a) (b)

Figure 5-16: Roughness and dominant surface by multi-file processing

(a) Roughness surface by IFFT of high-pass frequency summation
(b) Dominant surface by IFFT of dominant frequency based on high-pass frequency summation

5.3 Surface reconstruction

Beside correctly characterized surface data, one important key of this study is to ensure that, the imported variables from section 5.2 are able to recreate the surface in FEM simulation accurately. To fulfill the objective, surface reconstruction process is primarily trialed in Python. This is done to learn and assess the procedure and additionally examine correctness of the imported variables. The process of this sub-section is shown by flowchart in **Figure 5-17**.

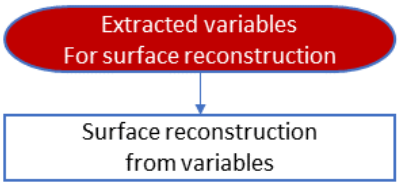


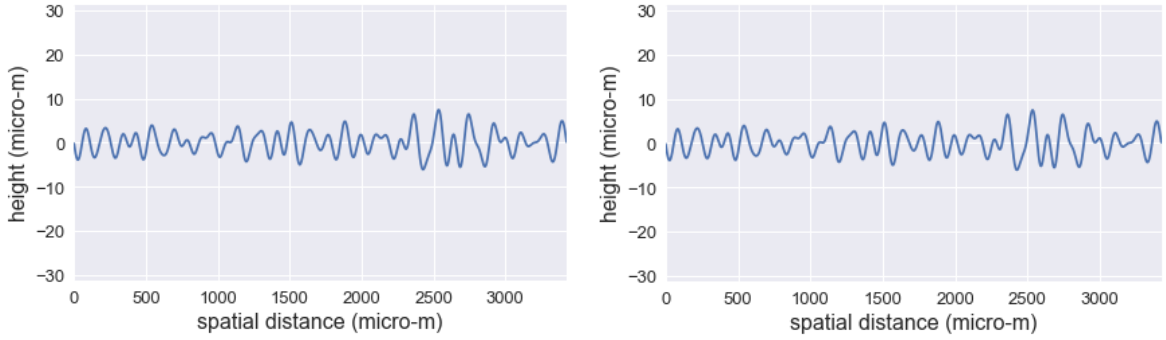
Figure 5-17: Flowchart of surface reconstruction step

5.3.1 1D-data of one-file processing

As stated in the Theoretical background, the surface of 1D data can be reconstructed by superposition of sinusoidal waves, one of which corresponding to the characteristic frequencies. The formula of sinusoidal wave is restated below.

$$z = Acos(2\pi ft + \theta) \qquad \textbf{(Equation 2-1)}$$

From the exported file of 1D data, 3 essential variables of each dominant frequency are given. The reconstruction process is programmed by calling amplitude variable as A, frequency as f, and phase as θ. Afterwards, computing the characteristic sinusoidal waves for each dominant frequency, then sum them up. The sinusoidal superposition results in the reconstructed curve in the final stage. Compared to the primary dominant curve in **Figure 5-18** (a), no deviation is observed in the reconstructed curve shown in **Figure 5-18** (b).



(a) (b)

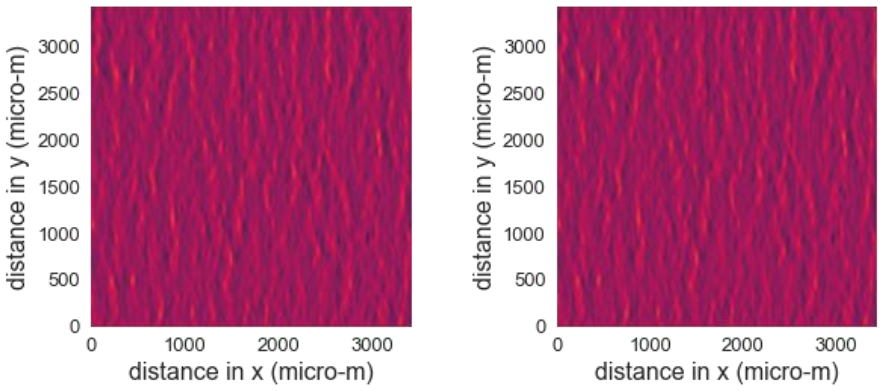
Figure 5-18: Reconstructed surface compared to the dominant surface in 1D
(a) Dominant surface by inverse Fourier transform
(b) Reconstructed surface by superposition of sinusoids

5.3.2 2D-data of one-file processing

Analogue to the process of 1D data, the surface in 2D can be reconstructed by superposition of sinusoidal surfaces corresponding to certain frequency x and y pair. The formula of sinusoid is restated below.

$$z = A\cos\left(2\pi\left(f_x x + f_y y\right) + \theta\right) \qquad \textbf{(Equation 2-2)}$$

From the exported variable file of 2D data, 4 essential variables at each dominant frequency are given (A, f_x, f_y and θ). The characteristic sinusoidal surface at a certain dominant frequency is constructed, hence loop over all dominant frequency pairs and superpose. The superposition results the reconstructed surface in final. Compared to the primary dominant surface in **Figure 5-19** (a), almost no deviation is observed in the reconstructed surface shown in **Figure 5-19** (b).



(a) (b)

Figure 5-19: Reconstructed surface compared to the dominant surface in 2D
(a) Dominant surface by inverse Fourier transform
(b) Reconstructed surface by superposition of sinusoids

6 Results and Discussion

In this chapter, assessment of the developed model's functionalities is discussed. Firstly, statistical tool used to evaluate the model is explained. Secondly, the assessment result of one-file processing and multi-file processing are examined. Lastly, the prominent and deficient features of the model are summarized, along with the recommendation for potential improvements.

6.1 Statistical method

R-squared (R^2), or in other name - coefficient of determination, is utilized as an evaluation tool in this model. R^2 score represents relationship between 2 variables that how well one dataset fits with the other [33]. The coefficient is apprehended by ratio of the predicted data variances by the total variances of the reference data. Formula is given in **Equation 6-1**.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - \bar{y}^2)^2} \qquad \textbf{(Equation 6-1)}$$

where $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ is the predicted value at $\boldsymbol{i}$-th sample

$\boldsymbol{y}_{\boldsymbol{i}}$ is the corresponding true value at $\boldsymbol{i}$-th sample

$\bar{\boldsymbol{y}}$ is an average value of $\boldsymbol{y}_{\boldsymbol{i}}$

$\boldsymbol{N}$ is number of the sample.

The value of coefficient of determination are ranged between -1 and 1. Best score at 1.0 indicates the well correlation between the model and the reference variability, while score at 0 indicates no correlation. In addition, the negative R^2 means inversed correlation, in the meaning that the model is arbitrarily worse in prediction. R^2 value larger than 0.5 is usually considered as a significant relationship [34].

In the developed characterization model of this study, minimum R^2 coefficient is set at 0.5. This assignment is done to ensure correlation of the outcome surface to the roughness.

6.2 Model assessment of one-file processing

To access the model, testing data with different surface conditions are appointed as input. Detail of datasets has been given in chapter 3. The evaluation outcome can be raised in 3 main aspects described as follow.

First aspect is about deviation in estimation of the dominant surface to roughness. For each surface condition, sets of data were selected (4 for no process, 1 for drilling, 2 for milling, 2 for water jet and 2 for wire cut). The resulting R^2 were collected and averaged separately by each. The result shows values in range of 0.50 to 0.60 displayed in **Figure 6-1**. This confirms ability of the model to automatically extract the dominant surface at the minimum acceptable R^2 at 0.5 as designated. Besides, number of dominant frequencies were collected by the same means. At around 0.002 to 0.03 percent of total frequencies were selected as dominance. If this percentage is converted to numbers, they are in range of hundreds to thousands. It is still inconvenient for surface reconstruction in FEM simulation with this large number of variables. Chart of the percentage amount of the dominant frequency is shown in **Figure 6-2**.

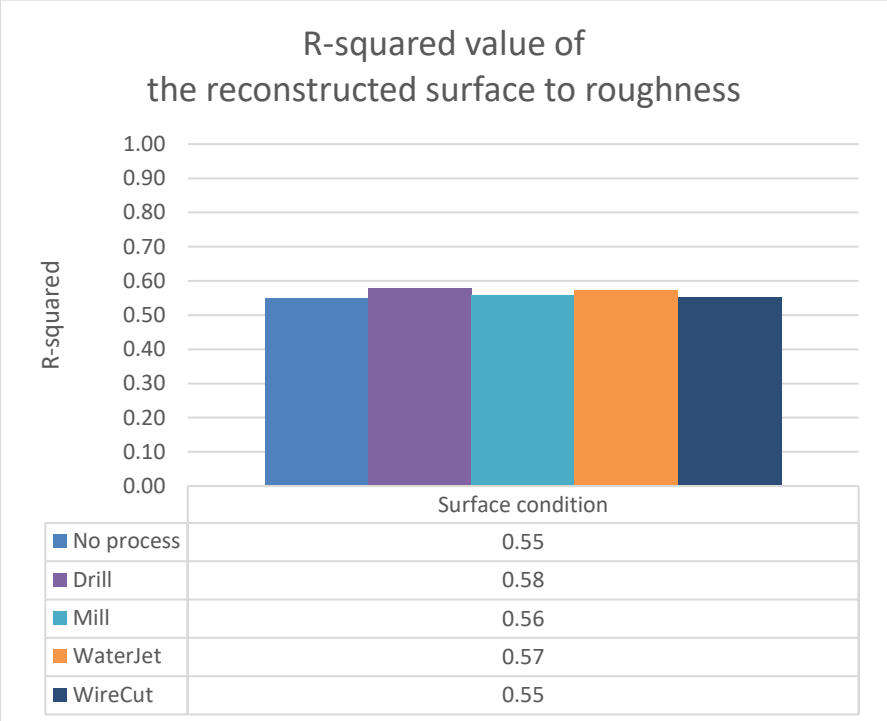


Figure 6-1: R-squared comparing reconstructed surface to roughness

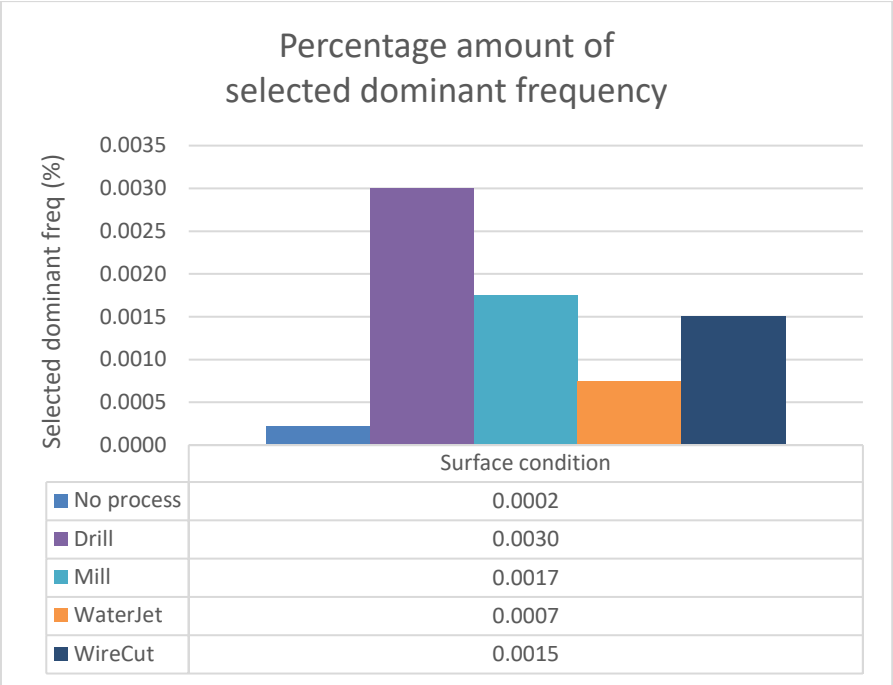


Figure 6-2: Amount of the selected dominant frequency

Second aspect to discuss is the model functionality to differentiate surface characteristics obtained by different surface processing. The computed surfaces after the model are displayed in **Figure 6-3**. Original, roughness and simplified surface by dominant frequency are displayed from left to right. First observation is that, from original surface to roughness, waviness is clearly discarded. Second observation is, comparing the simplified surfaces in **Figure 6-3** (a) to **Figure 6-3** (e), hypsography characteristics still remain and can be differentiated from process to process.

Last aspect to discuss is ability of the model to provide different surface pattern regarding to direction of section view aligned on the surface. For illustration, surface in **Figure 6-4** is cut in x- and y-spatial section, showing difference in height allocation.

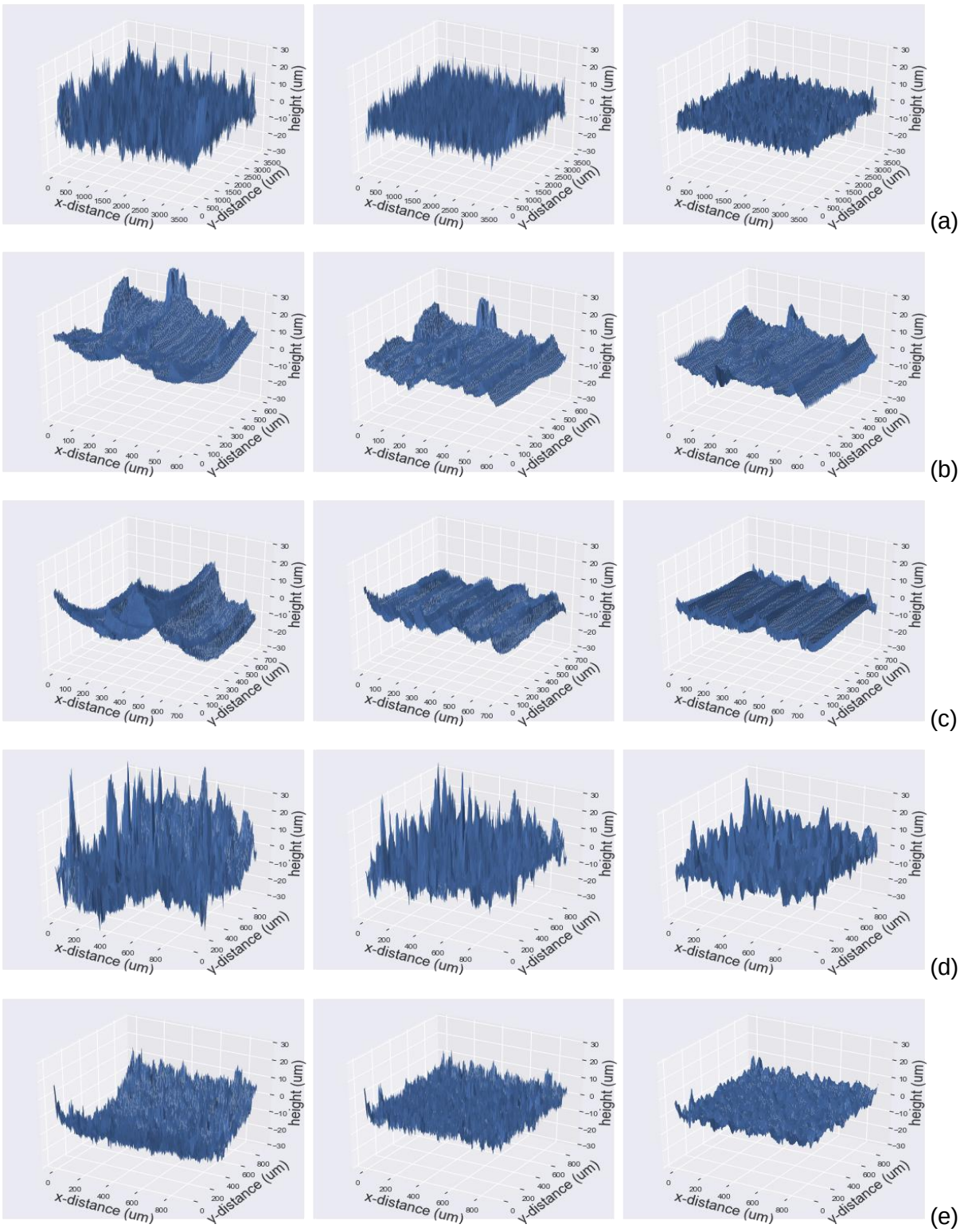


Figure 6-3: The computational surfaces
(from left to right: original surface, roughness and dominant surface)

(a) Surface with no process
(b) Surface with drilling
(c) Surface with milling
(d) Surface with water jet
(e) Surface with wire cut

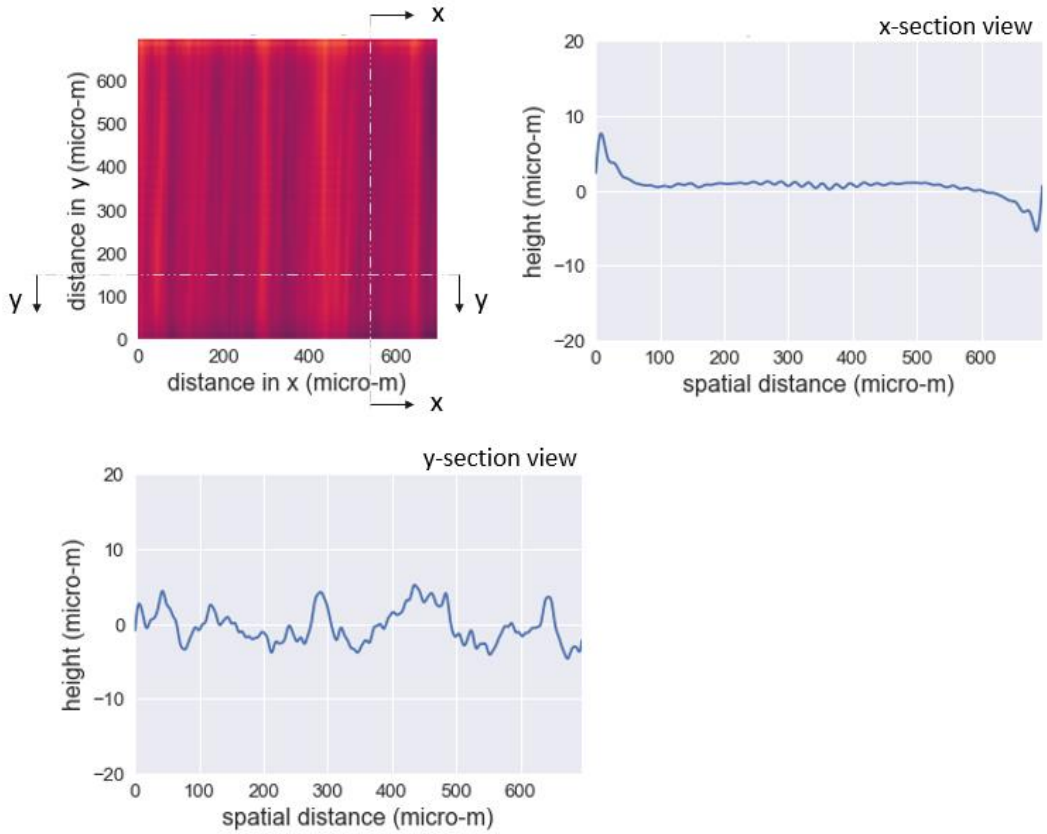
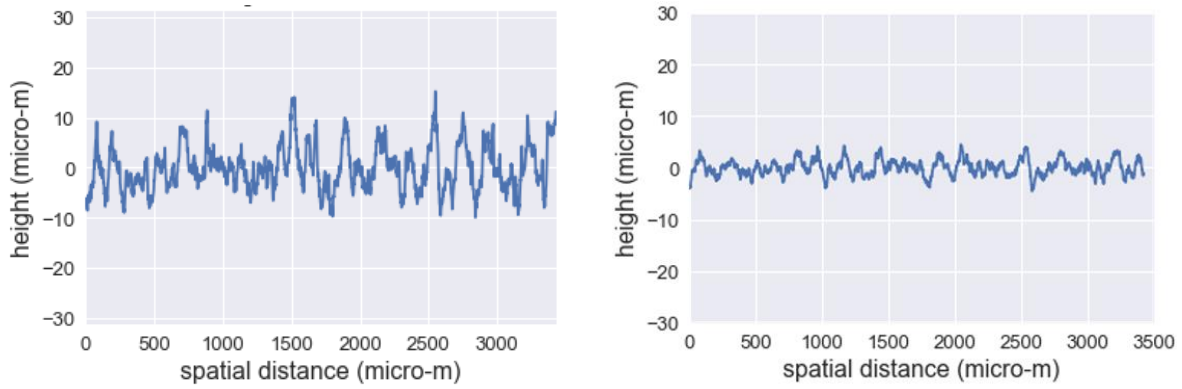


Figure 6-4: Different spatial height distribution regarding direction of section view

6.3 Model assessment of multi-file processing

As mentioned about the effort to combine surface data from several measured regions and compute them together as surface output, the applied technique is addition of high-pass Fourier coefficient. To evaluate the model, 10 testing datasets measured from same piece of XAR 450 no process surface condition were passed to the process. Unfortunately, it shows dissatisfaction result. Reduction in height amplitude can be obviously observed, despite the roughness characteristic remains. Display plots are given in **Figure 6-5**. This problem is on the basis of sinusoidal interference, which Fourier coefficients at the same frequency cancel out each other. This flaw of the model requires particularly further study and development.



(a) (b)

Figure 6-5: Computed roughness from the same material workpiece
(a) by One-file processing of one surface data
(b) by Multi-file processing of 10-combined surface data

6.4 Summary of model assessment

In this section, model assessment is summarized. The discussion is outlined from prominent features, deficient features and lastly the proposed methods for improvement.

Prominent features of the developed model compared to the previous model of Random height distribution are given in **Table 6-1**, while the deficient features and the improvement recommendations are given in **Table 6-2**. For user to utilize this developed model, points to be remarked is that, the input data has to be prior trimmed and the constants should be cautiously assigned.

Table 6-1: Prominent features of the developed model

The developed model of "Surface characterization"	**The previous model of "Random height distribution"**
• Separate off waviness which is irrelevant for Finite-Element simulation	• Done by separation tool in surface measuring instrument
• Remain characteristic of roughness which differs by surface condition. This enables computation of damage initiation curve for the surface from specific surface processing.	• Unable to keep rough surface characteristic since the height allocation is assigned randomly
• Able to provide different surface 1D pattern regarding to section view direction aligned on the surface	• Unable to differentiate
• Applicable for 3D surface topography	• Applicable in 2D surface curve only
• No discontinuity in the computed surface by Fourier transformation technique	• Possible for sharp edge discontinuity in the computed surface by poor polynomial fitting
• The simplified surface by dominant frequencies is suitable for mesh size assignment in FEM simulation. It is not necessary to give fine mesh size at rough tips. This decreases computational time of simulation.	• Feature of simplified surface is not available

Table 6-2: Deficient features of the developed model

The developed model of "Surface characterization"	**Prospect of improvement**
• Poor outlier cut. Outlier remains, especially in highly-light-reflected material by which the measuring instrument cannot detect surface height values.	• Utilize smoothing filter (λ_s) in addition by convolution technique with the raw surface
• Not suitable for application in smooth surface because of over-fitting in small height fluctuation	• Ensure outlier cut as mentioned above
• Large amount of dominant frequencies, resulting in long computational time for surface reconstruction	• Import x-y height value of simplified surface instead of dominant frequency variables. This allows user to put height data to FEM simulation directly. • Downsize area of preprocessed surface. Dominant frequency number is a proportion to the primary amount of frequency coefficient. • Combine adjacent dominant frequencies by clustering
• Frequency interference in multi-file processing model, causing amplitude height reduction in the combined surface output.	• New surface combination technique which alleviates interference effect. • Regard one measured region as representation of overall surface. Process this one dataset and pass the output to FEM. Representation of this region to others is ensured by "Dynamic time warping". Application of this technique is to measure similarity of two temporal sequences, which vary in speed pattern [35].

7 Conclusions and Prospects

According to the objective of this study seeking for an effective way to represent material rough surface into Finite-Element simulation, this study proposes the new developed model to characterize surface hypsography by Signal processing and Fourier transform. Outcome of the model provides good simplified surface proper for the application in Finite-Element analysis.

Highlight features of this model compared to the previous is an ability to keep height distribution which is an unique character of the surface. Besides, it is possible to construct surface in 3D topography and eliminate problem of surface discontinuity by poor fitting. This encourages user to simulate material surface properly, leading to the accurate computation of damage initiation curve. Moreover, the curve can be computed based on variety of material surface conditions.

For further development, deficiencies of the model are still about outlier cut and large number of dominant frequencies. The recommendations are to apply smoothing filter to eliminate outlier, find new method to import data output, or to perform frequency clustering. Above all, the most critical deficiency is frequency interference when passing multiple surface data into one processing model. New data combining technique or a technique to prove similarity of surfaces measured from different regions are challenge prospects for the study in the future.

List of References

[1] International Organization for Standardization, "ISO 4287:1997 Geometrical Product Specifications (GPS) -- Surface texture: Profile method -- Terms, definitions and surface texture parameters," 1997.

[2] A. Boryczko, "Distribution of Roughness and Waviness Components of Turned Surface Profiles," *Metrology and Measurement Systems,* pp. 611-620, 2010.

[3] Mitutoyo America Corporation, Quick Guide to Surface Roughness Measurement, USA, 2016.

[4] International Organization for Standardization, "ISO 3274:1996 Geometrical Product Specifications (GPS) -- Surface texture: Profile method -- Nominal characteristics of contact (stylus) instruments," 1996.

[5] Digital Surf, "Filtration Techniques for Surface Texture," [Online]. Available: https://guide.digitalsurf.com/en/guide-filtration-techniques.html. [Accessed Nov 2018].

[6] M. B. Raja J, Fu S and Liu X, "Recent Advances in Separation of Roughness, Waviness and Form," Center for Precision Metrology, The University of North Carolina at Charlotte, Charlotte, NC.

[7] M. Kuhn, "Digital Signal Processing," University of Cambridge, 2009.

[8] S. P., "Signal Processing Fundamental," in *Speech Processing in Embeded Systems*, Springer, 2010, pp. 9-36.

[9] Speaking Technology, "Plot continuous and discrete time wave sequences in matlab," 10 May 2012. [Online]. Available: https://www.123mylist.com/2012/05/plot-continuous-and-discrete-time-wave.html. [Accessed Feb 2019].

[10] R. Port, "Digital Signal Processing Overview," 21 Feb 2007. [Online]. Available: https://www.cs.indiana.edu/~port/teach/541/sig.proc.html. [Accessed Feb 2019].

[11] Nptel, "What is a Signal," [Online]. Available: https://nptel.ac.in/courses/Webcourse-contents/IIT-KANPUR/Digi_Sign_Pro/pdf/ch1.pdf. [Accessed Feb 2019].

[12] S. W. Smith, "Chapter 1: The Breadth and Depth of DSP," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 1-10.

[13] S. W. Smith, "Chapter 5: Linear Systems," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, Californai Technical Publishing, 1999, pp. 87-196.

[14] Sciences, Tampere University of Applied, "Chapter 4: Image Enhancement in the Frequency Domain," [Online]. Available: http://www.cs.tut.fi/~moncef/SGN-3016-DIP/Chap04.pdf. [Accessed Feb 2009].

[15] S. W. Smith, "Chapter 10 - Fourier Transform Properties," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 185-208.

[16] S. W. Smith, "Chapter 8: The Discrete Fourier Transform," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 141-165.

[17] "Fourier Transform: What is a Fourier transform?," [Online]. Available: http://mriquestions.com/fourier-transform-ft.html. [Accessed Feb 2019].

[18] V. Hlaváč, "Fourier transform, in 1D and in 2D," Czech Institute of Informatics, Robotics and Cybernetics, Prague.

[19] P. B. Osgood, "Fourier Series," in *Lecture Notes for EE261: The Fourier Transform and its Applications*, Stanford, Electrical Engineering Department, Standford University, p. 13.

[20] SciPy.org, "numpy.fft.fftfreq," 31 Jan 2019. [Online]. Available: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html. [Accessed Feb 2019].

[21] P. A. V. Oppenheim, "Chapter 9: Fourier Transform Properties," in *Signals and Systems*, MIT University, pp. 9-1 to 9-11.

[22] B. Morse, "The Fourier Transform: Examples, Properties, Common Pairs," BYU Computer Science.

[23] S. W. Smith, "Chapter 6: Convolution," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 107-122.

[24] "Convolution theorem," Comenius University, Bratislava.

[25] S. W. Smith, "Chapter 18: FFT Convolution," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 311-318.

[26] S. W. Smith, "Chapter 14: Introduction to Digital Filters," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999, pp. 261-276.

[27] R. Wang, "Physical Meaning of 2-D FT," 12 Nov 2015. [Online]. Available: http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html. [Accessed Feb 2019].

[28] "What is Python? Executive Summary," python, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed Feb 2019].

[29] SciPy.org, "Fourier Transforms (scipy.fftpack)," [Online]. Available: https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html. [Accessed Nov 2019].

[30] NanoFocus, "Operation Manual NanoFocus mSurf," Oberhausen, Germany, 2003.

[31] Y. B. Yuan and et al., "A Simplified Realization for the Gaussian Filter," *International Colloquium on Surfaces, Chemnitz (Germany),* pp. 133-144, 2000.

[32] Y. B. Yuan and et al., "A fast Gaussian filtering algorithm for threedimensional," *Journal of Physics: Conference Series 48,* pp. 1401-1406, 2006.

[33] scikit-learn developers, "Model evaluation: Quantifying the quality of predictions," [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score-the-coefficient-of-determination. [Accessed Mar 2019].

[34] M. K. Chung, "R2," [Online]. Available: http://www.stat.wisc.edu/~mchung/papers/R2.pdf. [Accessed Mar 2019].

[35] M. M., "Chapter 4: Dynamic Time Warping," in *Information Retrieval for Music and Motion*, Springer, 2007, p. 69.

Appendix

Script of One-file processing

Contents

1 Import data and variables

In [1]:
```
# import python module

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(color_codes=True)

from tqdm import tqdm
from scipy.fftpack import fft, ifft, fft2, ifft2, fftfreq, fftshift, ifftshift
from sklearn.metrics import r2_score
```

In [2]:
```
# import data file

input_file = '149-1A2.csv'
x_or_y = 'y'                            # data direction for 1D extraction (type 'x' or 'y')
row_no = 17                             # row no. for 1D data extraction
```

In [3]:
```
# assign constant

## outlier
outlier_angle = 65                      # degree

## Gaussian filter
lambda_c = 200                          # micro-m
freq_c = 1/lambda_c                     # (micro-m)^-1

## dominant freq
ratio_amp_dom2max_1D = 50               # %
accept_dom_rSqua_2D = 0.5               # value from -1 to 1
                                          ## -1 to 0 = no relation
                                          ## 0 = estimated data 'poorly' fit the reference data
                                          ## 1 = estimated data 'well' fit the reference data

#-----------------------------------------------------------------------------------------------#

## constant for gaussian filter (do not touch)
alfa = 0.4697
beta = 0.2206
```

In [4]:
```
# to trim non-related area of data
# ex. processed surface (drill, mill), the interested height is only one part of the measured surface

trim_command = 'n'                      # 'y' if trim, 'n' if not-trim
trim_min = 500                          # mm
trim_max = 2000                         # mm
```

In [5]:
```
# to assign length of dominant surface output (.csv)
# cut one corner of the area

assign_csv_len = 'y'                    # 'y' if assign, 'n' if not-assign
csv_len_1D = 500
csv_len_2D = 500
```

2 Data pre-processing

2.1 Data trimming

In [6]:
```
# import data
df = pd.read_csv(input_file, usecols=['# X','Y','Z'], skiprows=6)
df = df.rename(index=str, columns={"# X": "X"})

df.head()
```

Out[6]:

	X	Y	Z
0	0.0000	0.0	-8.68804
1	1.5625	0.0	-10.83410
2	3.1250	0.0	-10.39850
3	4.6875	0.0	-8.38015
4	6.2500	0.0	-9.10774

In [7]:
```
# trim input data to be square

x_list = sorted(list(set(df['X'])))
y_list = sorted(list(set(df['Y'])))

x_pnts = len(x_list)
y_pnts = len(y_list)
data_pnts = min(x_pnts, y_pnts)


# no of data point after cut = even

if data_pnts % 2 != 0:
    data_pnts = data_pnts - 1

data_list = x_list[0:data_pnts]
data_len = data_list[-1]


print(x_pnts)
print(y_pnts)
print(data_pnts)
```

```
2195
2202
2194
```

In [8]:
```
# pivot table

df_raw = df.pivot_table(values='Z', index='Y', columns='X', aggfunc='first')
df_raw = df_raw.iloc[0:data_pnts, 0:data_pnts]

df_raw.iloc[0:5,0:5]
```

Out[8]:

X	0.0	1.5625	3.125	4.6875	6.25
Y					
0.0000	-8.68804	-10.83410	-10.39850	-8.38015	-9.10774
1.5625	-7.89955	-10.23920	-10.67220	-9.13778	-9.65260
3.1250	-7.66675	-8.47945	-9.53161	-9.85035	-10.34100
4.6875	-6.04887	-7.41644	-9.23373	-8.36597	-8.99760
6.2500	-6.44270	-7.09102	-6.04804	-6.95919	-5.67674

In [9]:
```
# find index of trim_min, trim_max

if trim_command == 'y':

    trim_min_loc = 0
    trim_max_loc = 0

    for i in range(len(data_list)):
        if data_list[i] > trim_min:
            trim_min_loc = i
            break

    for i in range(len(data_list)):
        if data_list[len(data_list)-1-i] < trim_max:
            trim_max_loc = len(data_list)-1-i
            break

    if (trim_max_loc - trim_min_loc) % 2 != 0:              # no of data point after cut = even
        trim_min_loc = trim_min_loc + 1
```

In [10]:
```
# pivot table

if trim_command == 'y':
    df_2D = df.pivot_table(values='Z', index='Y', columns='X', aggfunc='first')
    df_2D = df_2D.iloc[trim_min_loc:trim_max_loc, trim_min_loc:trim_max_loc]

    df_2D.iloc[0:5,0:5]
```

In [11]:
```
# plot trimmed surface

if trim_command == 'y':

    # set axis
    axis_dist_raw = (0, data_len, 0, data_len)
    axis_dist_2D = (data_list[trim_min_loc], data_list[trim_max_loc], data_list[trim_min_loc], data_list[trim_max_loc])
    v_min_raw = np.amin(df_raw.values)
    v_max_raw = np.amax(df_raw.values)


    plot_size = (8, 4)
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
    f.suptitle("Trimmed surface", fontsize=16)

    ax1.imshow(df_raw, extent=axis_dist_raw, vmin=v_min_raw, vmax=v_max_raw)
    ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
    ax1.title.set_text('Raw surface')
    ax1.grid(False)

    ax2.imshow(df_2D, extent=axis_dist_2D, vmin=v_min_raw, vmax=v_max_raw)
    ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
    ax2.title.set_text('Trimmed surface')
    ax2.grid(False)

    plt.tight_layout()
    f.subplots_adjust(top=0.88)
```

In [12]:
```
# assign spatial axis

if trim_command == 'y':

    data_list = x_list[0:trim_max_loc-trim_min_loc]
    data_len = data_list[-1]
    data_pnts = len(data_list)

    print(trim_min_loc)
    print(trim_max_loc)
    print(data_pnts)
```

In [13]:
```
if trim_command == 'n':
    df_2D = df_raw.copy()
```

2.2 Axis assigns

In [14]:
```
# show spatial axis

print(data_list[0:5], "...", data_list[-5:])
print(data_len)
print(data_pnts)
```

```
[0.0, 1.5625, 3.125, 4.6875, 6.25] ... [3420.31, 3421.87, 3423.44, 3425.0, 3426.56]
3426.56
2194
```

In [15]:
```
# sampling info

samp_space = data_list[1] - data_list[0]            # micro-m
samp_rate = 1/samp_space                            # (micro-m)^(-1)

print(samp_space)
print(samp_rate)
```

```
1.5625
0.64
```

In [16]:
```
# assign freq axis

freq_full = fftfreq(data_pnts, samp_space)
freq_full_shift = fftshift(freq_full)
freq_half = freq_full_shift[freq_full_shift >= 0]

print(freq_full[0:5], "...", freq_full[-5:])
print(freq_full_shift[0:5], "...", freq_full_shift[-5:])
print(freq_half[0:5], "...", freq_half[-5:])
```

```
[0.         0.0002917  0.00058341 0.00087511 0.00116682] ... [-0.00145852 -0.00116682 -0.00087511 -0.00058341 -0.0002917 ]
[-0.32       -0.3197083  -0.31941659 -0.31912489 -0.31883318] ... [0.31854148 0.31883318 0.31912489 0.31941659 0.3197083 ]
[0.         0.0002917  0.00058341 0.00087511 0.00116682] ... [0.31854148 0.31883318 0.31912489 0.31941659 0.3197083 ]
```

2.3 Outlier cut

In [17]:
```
df_2D_corr = df_2D.copy()
```

In [18]:
```
# Detect in row direction

row_corr_count = 0

for row_i in tqdm(range(data_pnts)):
    row = df_2D_corr.iloc[row_i,:].values

    for col_i in range(data_pnts-1):                # data_pnts-1 : cut last index which has no next value to compare
        height = row[col_i + 1] - row[col_i]
        width = samp_space
        zeta = np.degrees(np.arctan(height/width))

        if zeta > outlier_angle:
            new_Z = row[col_i] + width * np.tan(np.radians(outlier_angle))
            df_2D_corr.iloc[row_i, col_i + 1] = new_Z
            row_corr_count = row_corr_count + 1

        if zeta < -outlier_angle:
            new_Z = row[col_i] - width * np.tan(np.radians(outlier_angle))
            df_2D_corr.iloc[row_i, col_i + 1] = new_Z
            row_corr_count = row_corr_count + 1
```

```
100%|██████████| 2194/2194 [00:19<00:00, 113.59it/s]
```

In [19]:
```
# Detect in column direction

col_corr_count = 0

for col_i in tqdm(range(data_pnts)):
    col = df_2D_corr.iloc[:, col_i].values

    for row_i in range(data_pnts-1):                # data_pnts-1 : cut last index which has no next value to compare
        height = col[row_i + 1] - col[row_i]
        width = samp_space
        zeta = np.degrees(np.arctan(height/width))

        if zeta > outlier_angle:
            new_Z = col[row_i] + width * np.tan(np.radians(outlier_angle))
            df_2D_corr.iloc[row_i + 1, col_i] = new_Z
            col_corr_count = col_corr_count + 1

        if zeta < -outlier_angle:
            new_Z = col[row_i] - width * np.tan(np.radians(outlier_angle))
            df_2D_corr.iloc[row_i + 1, col_i] = new_Z
            col_corr_count = col_corr_count + 1
```

```
100%|██████████| 2194/2194 [00:17<00:00, 127.56it/s]
```

In [20]:
```
print('Data points in matrix: ', data_pnts**2)
print('Corrected points_row:  ', row_corr_count)
print('Corrected points_col:  ', col_corr_count)
print('Corrected points_total:', row_corr_count + col_corr_count)
print('Percent correction:    ', (row_corr_count + col_corr_count)/(data_pnts**2) * 100, '%')
```

```
Data points in matrix:  4813636
Corrected points_row:   11389
Corrected points_col:   5987
Corrected points_total: 17376
Percent correction:     0.3609745315183782 %
```

In [21]:
```
# compare result

rSq_ori_corr = r2_score(df_2D, df_2D_corr)

print('Original to Corrected surface')
print('R-square: ', rSq_ori_corr)
```

```
Original to Corrected surface
R-square:  0.9994941888487899
```

2.4 Extract row of data (1D)

In [22]:
```
# check selection in x or y direction
# extract out row of data

if x_or_y == 'x':
    z_data = df_2D_corr.iloc[:, row_no].values

if x_or_y == 'y':
    z_data = df_2D_corr.iloc[row_no, :].values
```

3 Gaussian filter

3.1 1D-Gaussian

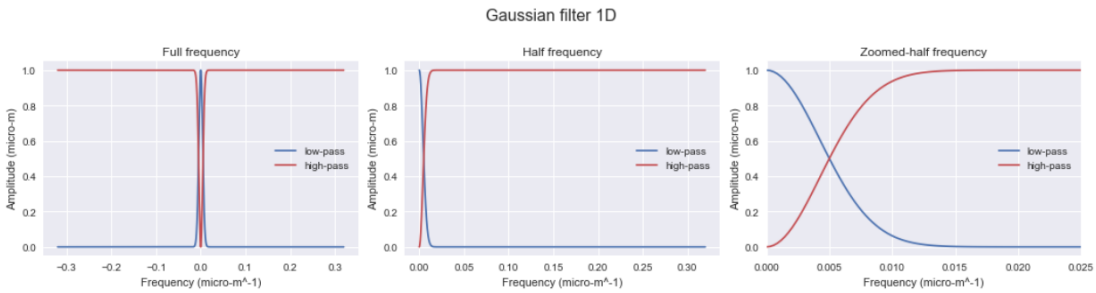
In [23]:
```
# create gaussian 1D

df_gauss_1D = pd.DataFrame(data = freq_full_shift, columns = ['freq'])
df_gauss_1D['f_to_fc'] = df_gauss_1D['freq'] / freq_c
df_gauss_1D['low_gauss'] = np.exp(-np.pi * (alfa * df_gauss_1D['f_to_fc'])**2)
df_gauss_1D['high_gauss'] = 1 - df_gauss_1D['low_gauss']

df_gauss_1D.head()
```

Out[23]:

	freq	f_to_fc	low_gauss	high_gauss
0	-0.320000	-64.000000	0.0	1.0
1	-0.319708	-63.941659	0.0	1.0
2	-0.319417	-63.883318	0.0	1.0
3	-0.319125	-63.824977	0.0	1.0
4	-0.318833	-63.766636	0.0	1.0

In [24]:
```
low_gauss_1D = df_gauss_1D['low_gauss'].values
high_gauss_1D = df_gauss_1D['high_gauss'].values
```

In [25]:
```
plot_size = (15, 4)
f, ((ax1, ax2, ax3)) = plt.subplots(1, 3, figsize=plot_size)
f.suptitle("Gaussian filter 1D", fontsize=16)

ax1.plot(freq_full_shift, low_gauss_1D, 'b')
ax1.plot(freq_full_shift, high_gauss_1D, 'r')
ax1.set(xlabel='Frequency (micro-m^-1)', ylabel='Amplitude (micro-m)')
ax1.title.set_text('Full frequency')
ax1.legend(('low-pass','high-pass'), loc='center right')

ax2.plot(freq_half, low_gauss_1D[data_pnts//2:], 'b')
ax2.plot(freq_half, high_gauss_1D[data_pnts//2:], 'r')
ax2.set(xlabel='Frequency (micro-m^-1)', ylabel='Amplitude (micro-m)')
ax2.title.set_text('Half frequency')
ax2.legend(('low-pass','high-pass'), loc='center right')

ax3.plot(freq_half, low_gauss_1D[data_pnts//2:], 'b')
ax3.plot(freq_half, high_gauss_1D[data_pnts//2:], 'r')
ax3.set_xlim(0, freq_c*5)
ax3.set(xlabel='Frequency (micro-m^-1)', ylabel='Amplitude (micro-m)')
ax3.title.set_text('Zoomed-half frequency')
ax3.legend(('low-pass','high-pass'), loc='center right')

plt.tight_layout()
f.subplots_adjust(top=0.8)
```

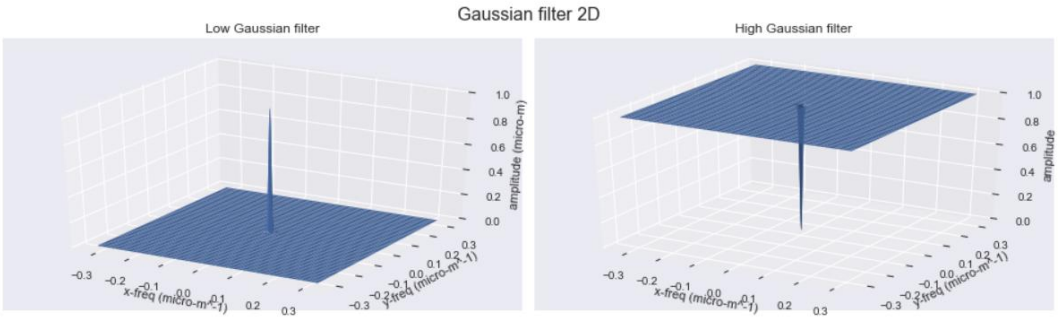


3.2 2D-Gaussian

In [26]:
```
# prepare fx/fc and fy/fc to put in equation

df_fx = pd.DataFrame(np.zeros((data_pnts, data_pnts)))

for i in range(data_pnts):
    df_fx.iloc[i,:] = freq_full_shift                          # row by rows


df_fy = pd.DataFrame(np.zeros((data_pnts, data_pnts)))

for i in range(data_pnts):
    df_fy.iloc[:,i] = freq_full_shift                          # column by columns


df_fx2fc = df_fx.copy() / freq_c
df_fy2fc = df_fy.copy() / freq_c
```

In [27]:
```
# input fx/fc and fy/fc to compute Gaussian

df_low_gauss = np.exp(-np.pi * beta * (df_fx2fc.pow(2) + df_fy2fc.pow(2)))
df_high_gauss = 1 - df_low_gauss
```

In [28]:
```
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=plt.figaspect(0.3))
fig.suptitle('Gaussian filter 2D', fontsize=16)


# First subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = df_low_gauss

surf = ax.plot_surface(X, Y, Z)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Low Gaussian filter')


# Second subplot
ax = fig.add_subplot(1, 2, 2, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = df_high_gauss

surf = ax.plot_surface(X, Y, Z)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude')
ax.title.set_text('High Gaussian filter')

plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
```



4 Surface transformation

4.1 1D-data

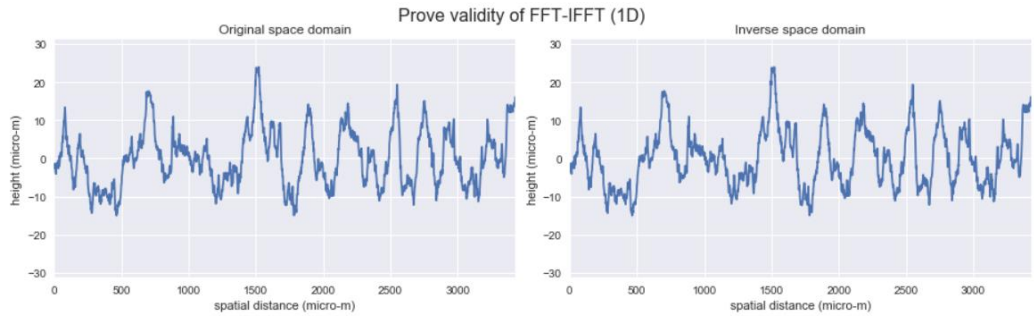
4.1.1 Fourier transform

In [29]:
```
z_fou_1D = fft(z_data)
z_inv_1D = ifft(z_fou_1D)
```

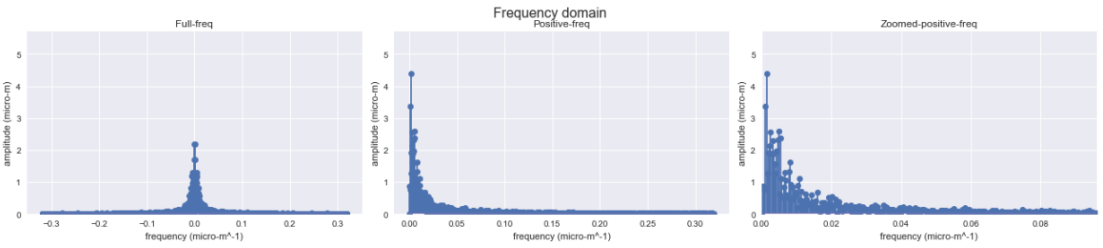
In [30]:
```
# shift freq domain to multiply with Gaussian

z_fou_1D_shift = fftshift(z_fou_1D)


# amplitude of z-fourier

z_fou_1D_full = 1/data_pnts * abs(z_fou_1D_shift)
z_fou_1D_half = 2/data_pnts * abs(z_fou_1D[0:data_pnts//2])    # fourier amplitude correction
                                                               # [0:data_pnts//2] : to cut only positive half of freq
                                                               # abs() : to get magnitude
                                                               # /data_pnts : to normalized operation of FFT summation
                                                               # *2 : to supplement value of f negative to f positive
```

In [31]:
```
# set limit for graph plot (1D)

plt_dist_xlim_1D = data_len
plt_dist_zlim_1D = max(np.amax(z_data), abs(np.amin(z_data))) * 1.3
plt_freq_xlim_1D = samp_rate
plt_freq_zlim_1D = np.amax(z_fou_1D_half) * 1.3
```

In [32]:
```
# prove validity of Fourier transform after cut outlier

plot_size = (13, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Prove validity of FFT-IFFT (1D)", fontsize=16)

ax1.plot(data_list, z_data)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Original space domain')

ax2.plot(data_list, z_inv_1D.real)
ax2.set_xlim(0, plt_dist_xlim_1D)
ax2.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax2.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax2.title.set_text('Inverse space domain')

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [33]:
```
# plot freq domain

plot_size = (18, 4)
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=plot_size)
f.suptitle("Frequency domain", fontsize=16)

ax1.stem(freq_full_shift, z_fou_1D_full)
ax1.set_ylim(0, plt_freq_zlim_1D)
ax1.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax1.title.set_text('Full-freq')

ax2.stem(freq_half, z_fou_1D_half)
ax2.set_ylim(0, plt_freq_zlim_1D)
ax2.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax2.title.set_text('Positive-freq')

ax3.stem(freq_half, z_fou_1D_half)
ax3.set_xlim(0, plt_freq_xlim_1D * 0.15)
ax3.set_ylim(0, plt_freq_zlim_1D)
ax3.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax3.title.set_text('Zoomed-positive-freq')

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [34]:
```
# compare result

rSq_fft_1D = r2_score(z_data, z_inv_1D.real)

print('Prove validity of Fourier transform 1D')
print('R-square: ', rSq_fft_1D)
```

```
Prove validity of Fourier transform 1D
R-square:  1.0
```

4.1.2 Convolution

In [35]:
```
# assign dataframe for operation

df_conv_1D = pd.DataFrame(data=freq_full_shift, columns=['freq'])
df_conv_1D['z_fou'] = z_fou_1D_shift
df_conv_1D['low_gauss'] = low_gauss_1D
df_conv_1D['high_gauss'] = high_gauss_1D
```

In [36]:
```
# convolution by mulplication in freq domain

df_conv_1D['low_conv'] = df_conv_1D['z_fou'] * df_conv_1D['low_gauss']
df_conv_1D['high_conv'] = df_conv_1D['z_fou'] * df_conv_1D['high_gauss']

df_conv_1D.head()
```

Out[36]:

	freq	z_fou	low_gauss	high_gauss	low_conv	high_conv
0	-0.320000	(-2.996872949888065+0j)	0.0	1.0	(-0+0j)	(-2.996872949888065+0j)
1	-0.319708	(3.7436894671459413+12.306405359315221j)	0.0	1.0	0j	(3.7436894671459413+12.306405359315221j)
2	-0.319417	(-18.351867043889378+9.553940774993578j)	0.0	1.0	(-0+0j)	(-18.351867043889378+9.553940774993578j)
3	-0.319125	(8.8788289519321-14.390786452804036j)	0.0	1.0	0j	(8.8788289519321-14.390786452804036j)
4	-0.318833	(2.814455285822987+14.41000864333023j)	0.0	1.0	0j	(2.814455285822987+14.41000864333023j)

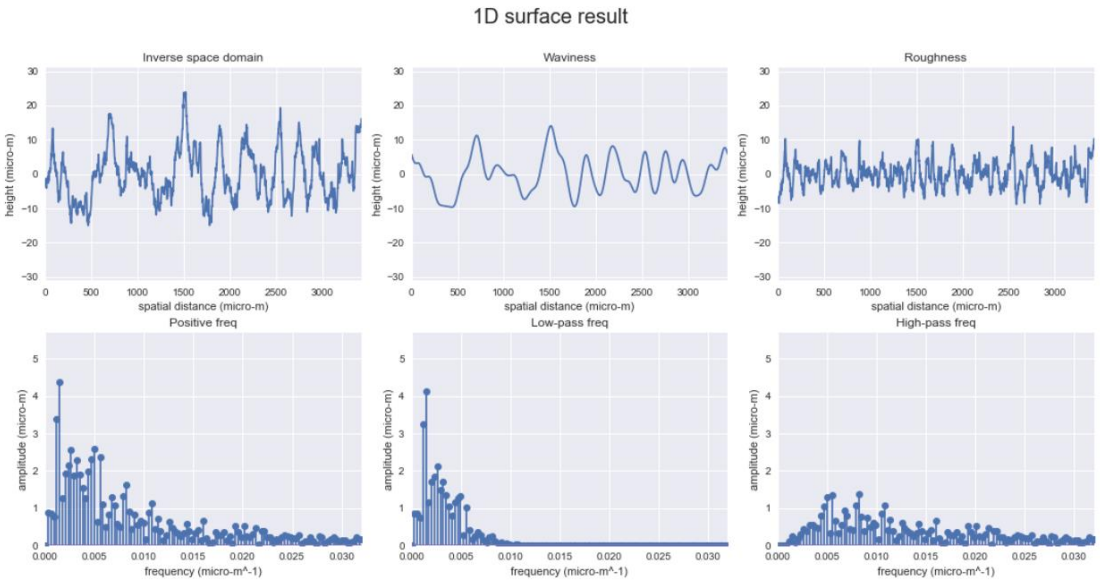
4.1.3 Inverse Fourier transform

In [37]:
```
# shift freq domain back for IFFT operation

z_fou_1D_ishift = ifftshift(df_conv_1D['z_fou'].values)
z_fou_low_1D_ishift = ifftshift(df_conv_1D['low_conv'].values)
z_fou_high_1D_ishift = ifftshift(df_conv_1D['high_conv'].values)
```

In [38]:
```
# correct freq amplitude for graph plot

z_fou_low_1D_full = 1/data_pnts * abs(z_fou_low_1D_ishift)
z_fou_low_1D_half = 2/data_pnts * abs(z_fou_low_1D_ishift[0:data_pnts//2])

z_fou_high_1D_full = 1/data_pnts * abs(z_fou_high_1D_ishift)
z_fou_high_1D_half = 2/data_pnts * abs(z_fou_high_1D_ishift[0:data_pnts//2])
```

In [39]:
```
# inverse Fourier transform

z_inv_1D = ifft(z_fou_1D_ishift)
z_inv_low_1D = ifft(z_fou_low_1D_ishift)
z_inv_high_1D = ifft(z_fou_high_1D_ishift)
```

In [40]:
```
plot_size = (15, 8)
f, ((ax1, ax2, ax3),(ax4, ax5, ax6)) = plt.subplots(2, 3, figsize=plot_size)
f.suptitle("1D surface result", fontsize=20)

ax1.plot(data_list, z_inv_1D.real)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Inverse space domain')

ax2.plot(data_list, z_inv_low_1D.real)
ax2.set_xlim(0, plt_dist_xlim_1D)
ax2.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax2.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax2.title.set_text('Waviness')

ax3.plot(data_list, z_inv_high_1D.real)
ax3.set_xlim(0, plt_dist_xlim_1D)
ax3.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax3.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax3.title.set_text('Roughness')

ax4.stem(freq_half, z_fou_1D_half)
ax4.set_xlim(0, plt_freq_xlim_1D * 0.05)
ax4.set_ylim(0, plt_freq_zlim_1D)
ax4.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax4.title.set_text('Positive freq')

ax5.stem(freq_half, z_fou_low_1D_half)
ax5.set_xlim(0, plt_freq_xlim_1D * 0.05)
ax5.set_ylim(0, plt_freq_zlim_1D)
ax5.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax5.title.set_text('Low-pass freq')

ax6.stem(freq_half, z_fou_high_1D_half)
ax6.set_xlim(0, plt_freq_xlim_1D * 0.05)
ax6.set_ylim(0, plt_freq_zlim_1D)
ax6.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax6.title.set_text('High-pass freq')

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



4.2 2D-data

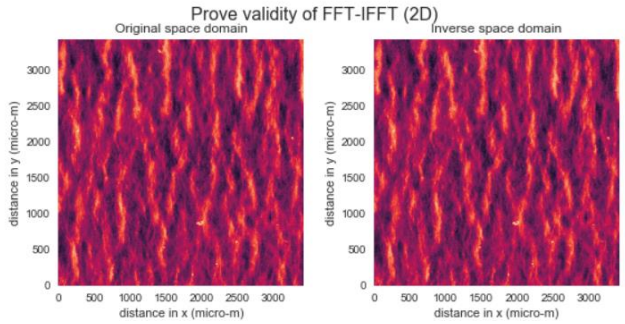
4.2.1 Fourier transform

In [41]:
```
z_fou_2D = fft2(df_2D_corr)
z_inv_2D = ifft2(z_fou_2D)
```

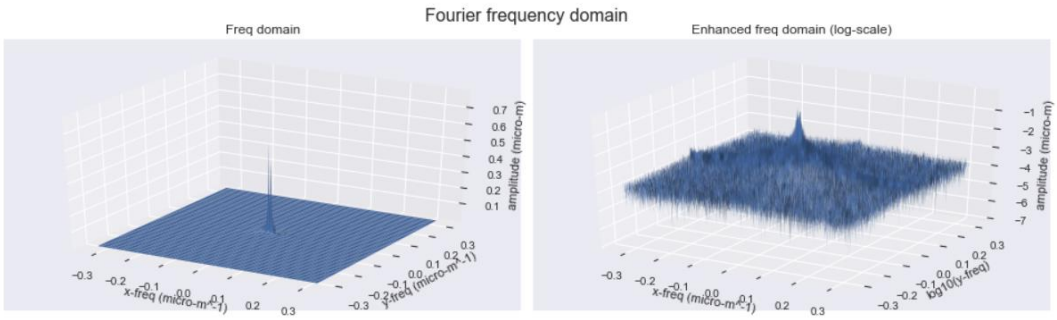
In [42]:
```
# shift freq domain to plot graph

z_fou_2D_shift = fftshift(z_fou_2D)
```

In [43]:
```
# set limit for graph plot (2D)

axis_dist = (0, data_len, 0, data_len)
axis_freq = (0, samp_rate, 0, samp_rate)
v_min_dist = np.amin(z_inv_2D.real)
v_max_dist = np.amax(z_inv_2D.real)
v_min_fAmp = np.amin(np.log10(1/data_pnts**2 * abs(z_fou_2D_shift)))
v_max_fAmp = np.amax(np.log10(1/data_pnts**2 * abs(z_fou_2D_shift)))
v_min_fPhs = np.amin(np.angle(z_fou_2D_shift))
v_max_fPhs = np.amax(np.angle(z_fou_2D_shift))

plt_dist_xlim_2D = data_len
plt_dist_zlim_2D = max(np.amax(df_2D_corr.values), abs(np.amin(df_2D_corr.values))) * 1.3
plt_freq_xlim_2D = samp_rate
plt_freq_zlim_2D = np.amax(z_fou_2D)/data_pnts**2 * 1.3
```

In [44]:
```
# prove validity of Fourier transform after cut outlier

plot_size = (8, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Prove validity of FFT-IFFT (2D)", fontsize=16)

ax1.imshow(df_2D_corr, extent=axis_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Original space domain')
ax1.grid(False)

ax2.imshow(z_inv_2D.real, extent=axis_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Inverse space domain')
ax2.grid(False)

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [45]:
```
# plot freq domain
# enhance freq domain by log-scale to show difference of freq hight


fig = plt.figure(figsize=plt.figaspect(0.3))
fig.suptitle('Fourier frequency domain', fontsize=16)


# First subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_2D_shift)

surf = ax.plot_surface(X, Y, Z)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Freq domain')


# Second subplot
ax = fig.add_subplot(1, 2, 2, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = np.log10(1/data_pnts**2 * abs(z_fou_2D_shift))                    # log10 to enhance value difference

surf = ax.plot_surface(X, Y, Z)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='log10(y-freq)', zlabel='amplitude (micro-m)')
ax.title.set_text('Enhanced freq domain (log-scale)')


plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
```



In [46]:
```
# compare result

rSq_fft_2D = r2_score(df_2D_corr, z_inv_2D.real)

print('Prove validity of Fourier transform 2D')
print('R-square: ', rSq_fft_2D)
```

```
Prove validity of Fourier transform 2D
R-square:  1.0
```

4.2.2 Convolution

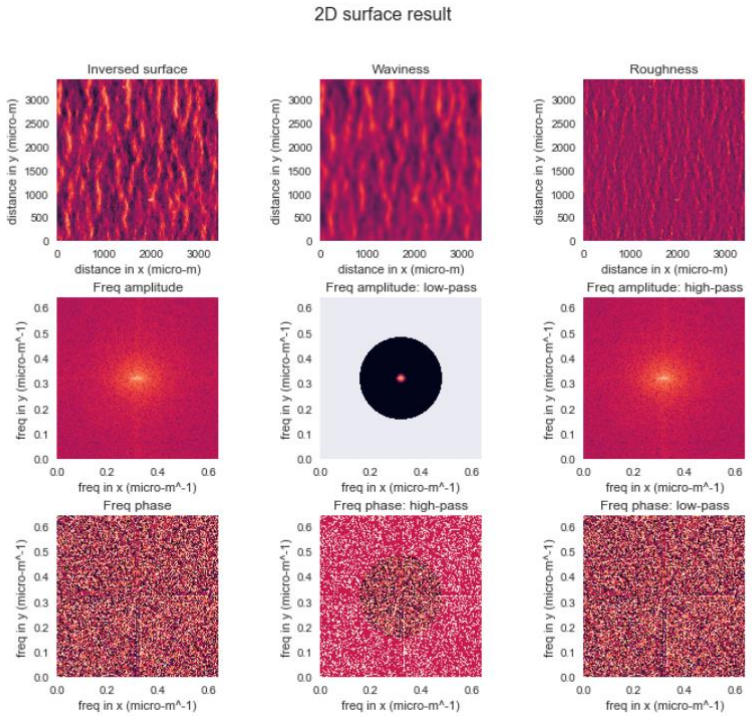
In [47]:
```
# convolution by element-wise mulplication in freq domain

## low-pass convolution
z_fou_low_2D = np.multiply(z_fou_2D_shift, df_low_gauss)


## high-pass convolution
z_fou_high_2D = np.multiply(z_fou_2D_shift, df_high_gauss)
```

4.2.3 Inverse Fourier transform

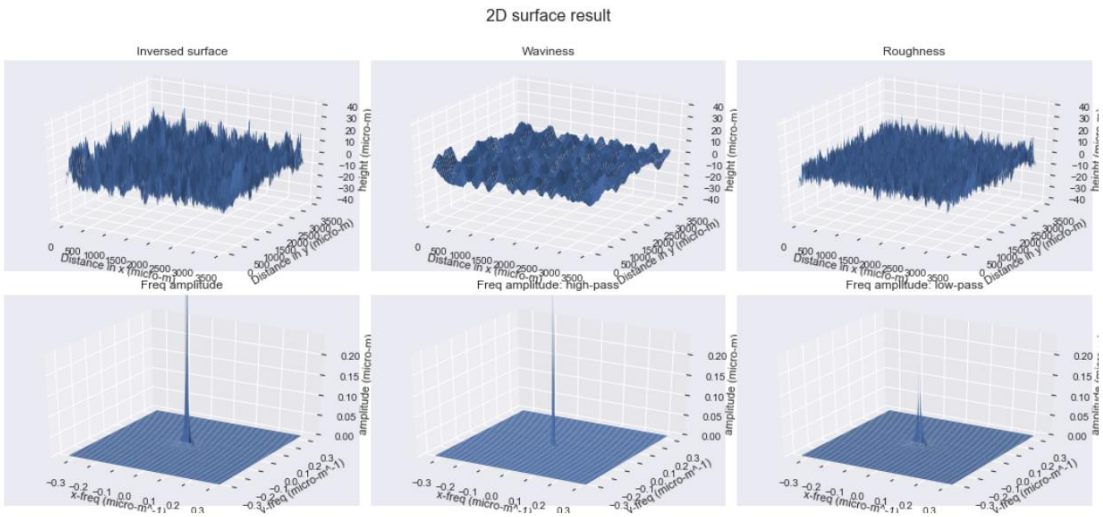
In [48]:
```
# shift freq domain back for IFFT operation

z_fou_2D_ishift = ifftshift(z_fou_2D_shift)
z_fou_low_2D_ishift = ifftshift(z_fou_low_2D)
z_fou_high_2D_ishift = ifftshift(z_fou_high_2D)
```

In [49]:
```
# inverse Fourier transform

z_inv_2D = ifft2(z_fou_2D_ishift)
z_inv_low_2D = ifft2(z_fou_low_2D_ishift)
z_inv_high_2D = ifft2(z_fou_high_2D_ishift)
```

In [50]:
```
# image plot

plot_size = (10, 9)
f, ((ax1, ax2, ax3), (ax4, ax5, ax6), (ax7, ax8, ax9)) = plt.subplots(3, 3, figsize=plot_size)
f.suptitle("2D surface result", fontsize=16)

ax1.imshow(z_inv_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Inversed surface')
ax1.grid(False)

ax2.imshow(z_inv_low_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Waviness')
ax2.grid(False)

ax3.imshow(z_inv_high_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax3.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax3.title.set_text('Roughness')
ax3.grid(False)

ax4.imshow(np.log10(1/data_pnts**2 * abs(z_fou_2D_shift)), extent=axis_freq, vmin=v_min_fAmp, vmax=v_max_fAmp)    # log10 to enha
ax4.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax4.title.set_text('Freq amplitude')
ax4.grid(False)

ax5.imshow(np.log10(1/data_pnts**2 * abs(z_fou_low_2D)), extent=axis_freq, vmin=v_min_fAmp, vmax=v_max_fAmp)      # log10 to enha
ax5.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax5.title.set_text('Freq amplitude: low-pass')
ax5.grid(False)

ax6.imshow(np.log10(1/data_pnts**2 * abs(z_fou_high_2D)), extent=axis_freq, vmin=v_min_fAmp, vmax=v_max_fAmp)      # log10 to enh
ax6.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax6.title.set_text('Freq amplitude: high-pass')
ax6.grid(False)

ax7.imshow(np.angle(z_fou_2D_shift), extent=axis_freq, vmin=v_min_fPhs, vmax=v_max_fPhs)
ax7.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax7.title.set_text('Freq phase')
ax7.grid(False)

ax8.imshow(np.angle(z_fou_low_2D), extent=axis_freq, vmin=v_min_fPhs, vmax=v_max_fPhs)
ax8.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax8.title.set_text('Freq phase: high-pass')
ax8.grid(False)

ax9.imshow(np.angle(z_fou_high_2D), extent=axis_freq, vmin=v_min_fPhs, vmax=v_max_fPhs)
ax9.set(xlabel='freq in x (micro-m^-1)', ylabel='freq in y (micro-m^-1)')
ax9.title.set_text('Freq phase: low-pass')
ax9.grid(False)

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



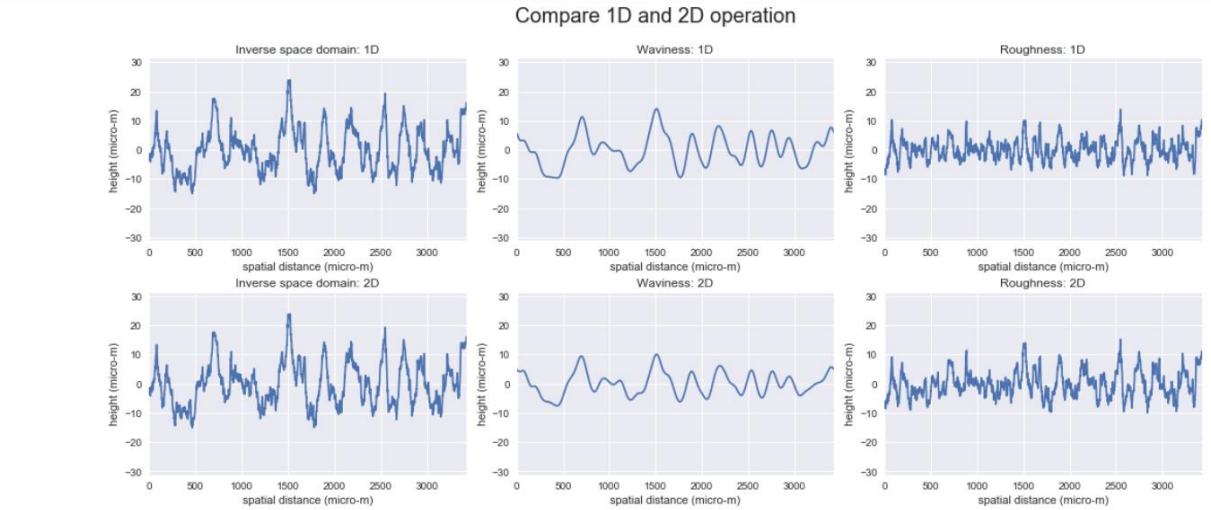
In [51]:
```
# surface plot

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

plot_size = (15, 7)
fig = plt.figure(figsize=plot_size)                                    # set up figure size
fig.suptitle('2D surface result', fontsize=16)


# First subplot
ax = fig.add_subplot(2, 3, 1, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_2D.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Inversed surface')


# Second subplot
ax = fig.add_subplot(2, 3, 2, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_low_2D.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Waviness')


# third subplot
ax = fig.add_subplot(2, 3, 3, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_high_2D.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Roughness')


# fourth subplot
ax = fig.add_subplot(2, 3, 4, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_2D_shift)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Freq amplitude')


# fifth subplot
ax = fig.add_subplot(2, 3, 5, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_low_2D)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Freq amplitude: high-pass')


# sixth subplot
ax = fig.add_subplot(2, 3, 6, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_high_2D)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Freq amplitude: low-pass')


plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
```



4.3 1D and 2D surface comparison

In [52]:
```
# extract row of 2D surface transformation to compare with 1D surface transformation

if x_or_y == 'x':
    z_inv_select = z_inv_2D[:, row_no]
    z_inv_select_low = z_inv_low_2D[:, row_no]
    z_inv_select_high = z_inv_high_2D[:, row_no]

if x_or_y == 'y':
    z_inv_select = z_inv_2D[row_no, :]
    z_inv_select_low = z_inv_low_2D[row_no, :]
    z_inv_select_high = z_inv_high_2D[row_no, :]
```

In [53]:
```
# plot compare 1D and 2D result in 1D-display

plot_size = (15, 7)
f, ((ax1, ax2, ax3),(ax4, ax5, ax6)) = plt.subplots(2, 3, figsize=plot_size)
f.suptitle("Compare 1D and 2D operation", fontsize=20)

ax1.plot(data_list, z_inv_1D.real)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Inverse space domain: 1D')

ax2.plot(data_list, z_inv_low_1D.real)
ax2.set_xlim(0, plt_dist_xlim_1D)
ax2.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax2.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax2.title.set_text('Waviness: 1D')

ax3.plot(data_list, z_inv_high_1D.real)
ax3.set_xlim(0, plt_dist_xlim_1D)
ax3.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax3.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax3.title.set_text('Roughness: 1D')

ax4.plot(data_list, z_inv_select)
ax4.set_xlim(0, plt_dist_xlim_1D)
ax4.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax4.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax4.title.set_text('Inverse space domain: 2D')

ax5.plot(data_list, z_inv_select_low)
ax5.set_xlim(0, plt_dist_xlim_1D)
ax5.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax5.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax5.title.set_text('Waviness: 2D')

ax6.plot(data_list, z_inv_select_high)
ax6.set_xlim(0, plt_dist_xlim_1D)
ax6.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax6.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax6.title.set_text('Roughness: 2D')


plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [54]:
```
# compare result

rSq_inv_1D2D = r2_score(z_inv_select, z_inv_1D.real)
rSq_wav_1D2D = r2_score(z_inv_select_low, z_inv_low_1D.real)
rSq_rou_1D2D = r2_score(z_inv_select_high,z_inv_high_1D.real)

print('Inverse space domain - 1D to 2D')
print('R-square: ', rSq_inv_1D2D)
print('\nWaviness - 1D to 2D')
print('R-square: ', rSq_wav_1D2D)
print('\nRoughness - 1D to 2D')
print('R-square: ', rSq_rou_1D2D)
```

```
Inverse space domain - 1D to 2D
R-square:  1.0

Waviness - 1D to 2D
R-square:  0.8096538769189294

Roughness - 1D to 2D
R-square:  0.851244133690483
```

5 Dominant frequency

5.1 1D dominant freq

In [55]:
```
# automatically assign cut-off amplitude by ratio to max value of freq

freq_dom_amp_c_1D = ratio_amp_dom2max_1D * z_fou_high_1D_half.max() / 100    # value as shown in graph (half-range)
freq_dom_amp_c_1D_full = freq_dom_amp_c_1D/2                                 # value for extraction operation (full-range)

print(freq_dom_amp_c_1D)
print(freq_dom_amp_c_1D_full)
```

```
0.68783918375017
0.343919591875085
```

In [56]:
```
# create dataframe for operation

df_freq_dom_1D = pd.DataFrame(data=freq_full, columns=['freq'])
df_freq_dom_1D['high_conv'] = z_fou_high_1D_ishift
df_freq_dom_1D['amplitude'] = z_fou_high_1D_full
df_freq_dom_1D['dominant?'] = np.zeros(len(df_freq_dom_1D))
df_freq_dom_1D['f_dom'] = np.zeros(len(df_freq_dom_1D))
```

In [57]:
```
# identify dominant freq by boolean True, False

df_freq_dom_1D['dominant?'] = np.where(
    df_freq_dom_1D['amplitude'] >= freq_dom_amp_c_1D_full,
    True,
    False
)


# remain only the value of dominant freq

df_freq_dom_1D['f_dom'] = np.where(
    df_freq_dom_1D['dominant?'] == True,
    df_freq_dom_1D['high_conv'],
    0.0
)

df_freq_dom_1D.head()
```

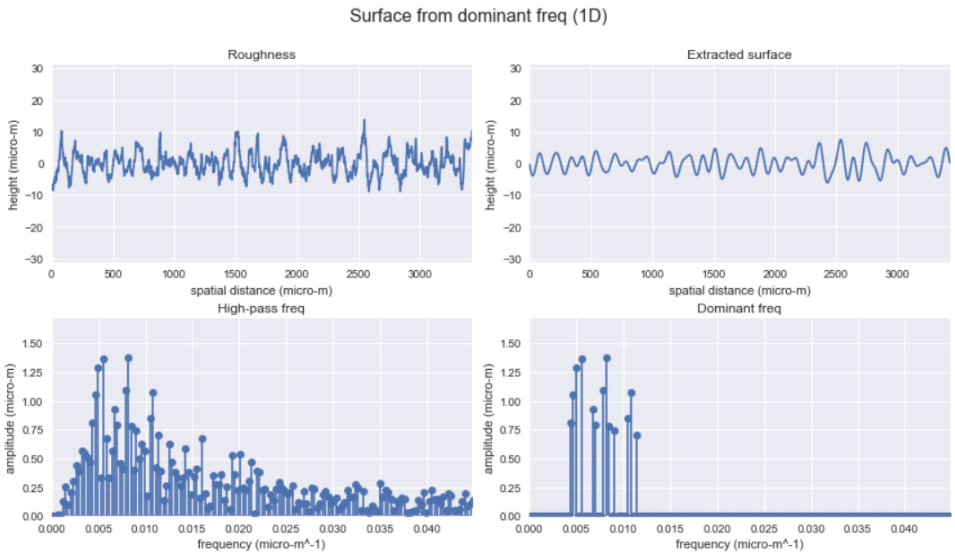
Out[57]:

	freq	high_conv	amplitude	dominant?	f_dom
0	0.000000	0j	0.000000	False	0j
1	0.000292	(-2.192910464412582+0.5501132822468441j)	0.001030	False	0j
2	0.000583	(-0.6686547222853475+8.797442026095766j)	0.004021	False	0j
3	0.000875	(2.045058489329037+17.627950144703743j)	0.008089	False	0j
4	0.001167	(101.00749173253176+92.97330091980548j)	0.062572	False	0j

In [58]:
```
# normalized amplitude for graph plot

z_fou_dom_1D_full = 1/data_pnts * abs(df_freq_dom_1D['f_dom'].values)
z_fou_dom_1D_half = 2/data_pnts * abs(df_freq_dom_1D['f_dom'].values[0:data_pnts//2])
```

In [59]:
```
# inverse Fourier transform

z_inv_dom_1D = ifft(df_freq_dom_1D['f_dom'].values)
```

In [60]:
```
plot_size = (12, 7)
f, ((ax1, ax2),(ax3, ax4)) = plt.subplots(2, 2, figsize=plot_size)
f.suptitle("Surface from dominant freq (1D)", fontsize=16)

ax1.plot(data_list, z_inv_high_1D.real)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Roughness')

ax2.plot(data_list, z_inv_dom_1D.real)
ax2.set_xlim(0, plt_dist_xlim_1D)
ax2.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax2.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax2.title.set_text('Extracted surface')

ax3.stem(freq_half, z_fou_high_1D_half)
ax3.set_xlim(0, plt_freq_xlim_1D * 0.07)
ax3.set_ylim(0, plt_freq_zlim_1D * 0.3)
ax3.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax3.title.set_text('High-pass freq')

ax4.stem(freq_half, z_fou_dom_1D_half)
ax4.set_xlim(0, plt_freq_xlim_1D * 0.07)
ax4.set_ylim(0, plt_freq_zlim_1D * 0.3)
ax4.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax4.title.set_text('Dominant freq')

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [61]:
```
# count amount of dominant freq 1D
f_dom_no_1D = (df_freq_dom_1D['dominant?'] == True).astype(int).values.sum()

print('Amount of dominant freq: ', f_dom_no_1D)
```

```
Amount of dominant freq:  26
```

In [62]:
```
# compare result

rSq_rou2dom_1D = r2_score(z_inv_high_1D.real, z_inv_dom_1D.real)

print('Roughness to Dominant surface 1D')
print('R-square: ', rSq_rou2dom_1D)
```

```
Roughness to Dominant surface 1D
R-square:  0.5188615061498657
```

5.2 2D dominant freq

5.2.1 Autimatically assign cut-off amplitude

In [63]:
```
# prepare dataframe for loop operation

df_freq = pd.DataFrame(data = z_fou_high_2D_ishift,
                       index = freq_full,
                       columns = freq_full)

df_freq_amp = abs(df_freq)
```

In [64]:
```
# prepare list of freq -> to find cut-off amplitude of dominant freq

list_freq_amp = df_freq_amp.values.flatten()
list_freq_amp = sorted(list_freq_amp)
```

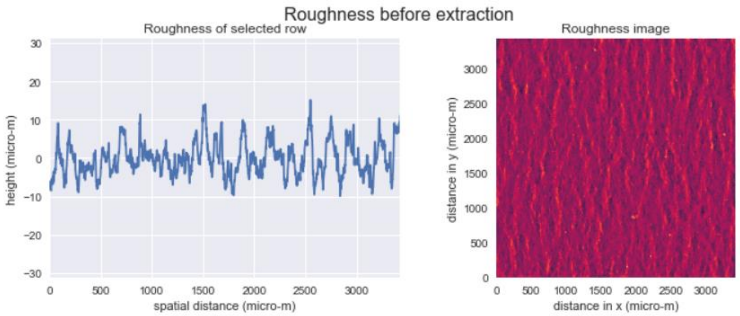
In [65]:
```
# create table to compare result based on amount of dominant freq

no_dom_freq_per = np.array([0.0025,0.005,0.01,
                            0.02,0.04,0.06,0.08,0.1,
                            0.2,0.4,0.6,0.8,1.0,
                            1.5,2.0])

df_dom_samp = pd.DataFrame(data = no_dom_freq_per, columns = ['no_f_per'])
df_dom_samp['no_f'] = df_dom_samp['no_f_per'] * len(list_freq_amp) / 100
df_dom_samp['no_f_int'] = df_dom_samp['no_f'].astype(int) + 1
df_dom_samp['f_dom_amp_c'] = np.zeros(len(no_dom_freq_per))
df_dom_samp['result_error'] = np.zeros(len(no_dom_freq_per))
```

In [66]:
```
# find cut-off amplitude for dominant freq

for i in range(len(no_dom_freq_per)):
    df_dom_samp.loc[i,'f_dom_amp_c'] = list_freq_amp[-df_dom_samp.loc[i,'no_f_int']]

df_dom_samp.head()
```

Out[66]:

	no_f_per	no_f	no_f_int	f_dom_amp_c	result_error
0	0.0025	120.3409	121	551182.982392	0.0
1	0.0050	240.6818	241	422678.474000	0.0
2	0.0100	481.3636	482	320325.803571	0.0
3	0.0200	962.7272	963	232189.056210	0.0
4	0.0400	1925.4544	1926	159046.522992	0.0

In [67]:
```
# roughness plot

plot_size = (10, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Roughness before extraction", fontsize=16)

ax1.plot(data_list, z_inv_select_high)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Roughness of selected row')

ax2.imshow(z_inv_high_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Roughness image')
ax2.grid(False)


plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [68]:
```
def find_error_dom_f(no_f_per, freq_dom_amp_c):

    df_freq_dom = df_freq.copy()

    for x in df_freq.columns:
        df_freq_dom[x] = np.where(
            df_freq_amp[x] > freq_dom_amp_c,
            df_freq_dom[x],
            0.0
        )

    z_fou_dom = df_freq_dom.values
    z_inv_dom_2D = ifft2(z_fou_dom)

#-----------------------------------------------------------------------------------------------------------------------#

    # find error

    rSq_rou2dom_2D = r2_score(z_inv_high_2D.real, z_inv_dom_2D.real)

#-----------------------------------------------------------------------------------------------------------------------#

    # extract row for 1D plot

    if x_or_y == 'x':
        z_inv_select_dom = z_inv_dom_2D[:, row_no]

    if x_or_y == 'y':
        z_inv_select_dom = z_inv_dom_2D[row_no, :]

#-----------------------------------------------------------------------------------------------------------------------#

    # plot graph to compare


    title = 'No. of selected dom freq: ' + str(no_f_per) + ', R-Square error: ' + str("{0:.2f}".format(rSq_rou2dom_2D))

    plot_size = (10, 4)
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
    f.suptitle(title, fontsize=16)
    axis_dist = (0, data_len, 0, data_len)                                          # assign axis for spatial domain
    axis_freq = (0, samp_rate, 0, samp_rate)                                        # assign axis for freq domain

    ax1.plot(data_list, z_inv_select_dom)
    ax1.set_xlim(0, plt_dist_xlim_1D)
    ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
    ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
    ax1.title.set_text('Roughness of selected row')

    ax2.imshow(z_inv_dom_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
    ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
    ax2.title.set_text('Roughness image')
    ax2.grid(False)

    plt.tight_layout()
    f.subplots_adjust(top=0.88)


    return rSq_rou2dom_2D
```

In [69]:
```
# find result error between roughness and extracted surface

for i in tqdm(range(len(no_dom_freq_per))):
    df_dom_samp.loc[i,'result_error'] = find_error_dom_f(df_dom_samp.loc[i,'no_f_per'], df_dom_samp.loc[i,'f_dom_amp_c'])

df_dom_samp
```

Out[69]:

	no_f_per	no_f	no_f_int	f_dom_amp_c	result_error
0	0.0025	120.3409	121	551182.982392	0.140319
1	0.0050	240.6818	241	422678.474000	0.229845
2	0.0100	481.3636	482	320325.803571	0.334355
3	0.0200	962.7272	963	232189.056210	0.450823
4	0.0400	1925.4544	1926	159046.522992	0.565032
5	0.0600	2888.1816	2889	124884.942185	0.628756
6	0.0800	3850.9088	3851	104738.690121	0.670638
7	0.1000	4813.6360	4814	92242.632326	0.701669
8	0.2000	9627.2720	9628	58374.091785	0.788416
9	0.4000	19254.5440	19255	35459.042552	0.854930
10	0.6000	28881.8160	28882	26347.012706	0.885273

In [70]:
```
# automatically assign cut-off dominant amplitude by acceptable percentage of error

for i in range(len(no_dom_freq_per)):
    if df_dom_samp.loc[i,'result_error'] > accept_dom_rSqua_2D:
        freq_dom_amp_c_2D = df_dom_samp.loc[i,'f_dom_amp_c']
        break

freq_dom_amp_c_2D
```

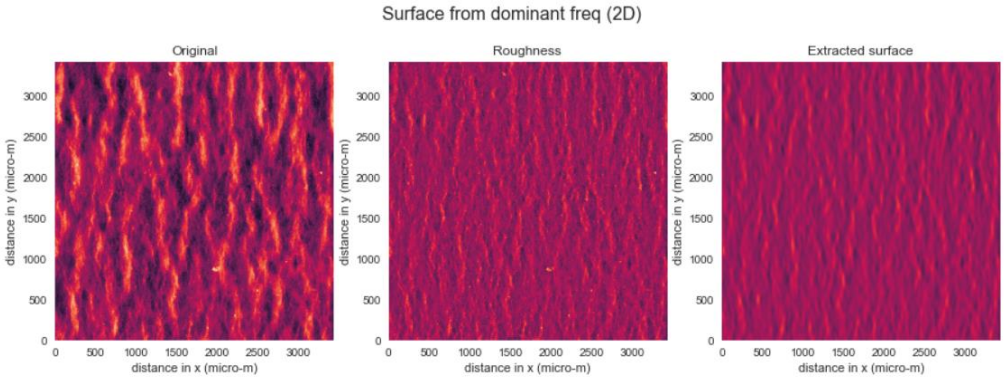
Out[70]: 159046.52299209594

5.2.2 Extraction operation

In [71]:
```
# remain only dominant freq in matrix

df_freq_dom = df_freq.copy()

for x in df_freq.columns:
    df_freq_dom[x] = np.where(
        df_freq_amp[x] > freq_dom_amp_c_2D,
        df_freq_dom[x],
        0.0
    )
```

In [72]:
```
z_fou_dom = df_freq_dom.values
z_inv_dom_2D = ifft2(z_fou_dom)

z_fou_dom_shift = fftshift(z_fou_dom)
z_fou_dom_ishift = ifftshift(z_fou_dom_shift)
```

In [73]:
```
# plot surface image

plot_size = (15, 5)
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=plot_size)
f.suptitle("Surface from dominant freq (2D)", fontsize=16)

ax1.imshow(df_2D_corr, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Original')
ax1.grid(False)

ax2.imshow(z_inv_high_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Roughness')
ax2.grid(False)

ax3.imshow(z_inv_dom_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax3.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax3.title.set_text('Extracted surface')
ax3.grid(False)
```

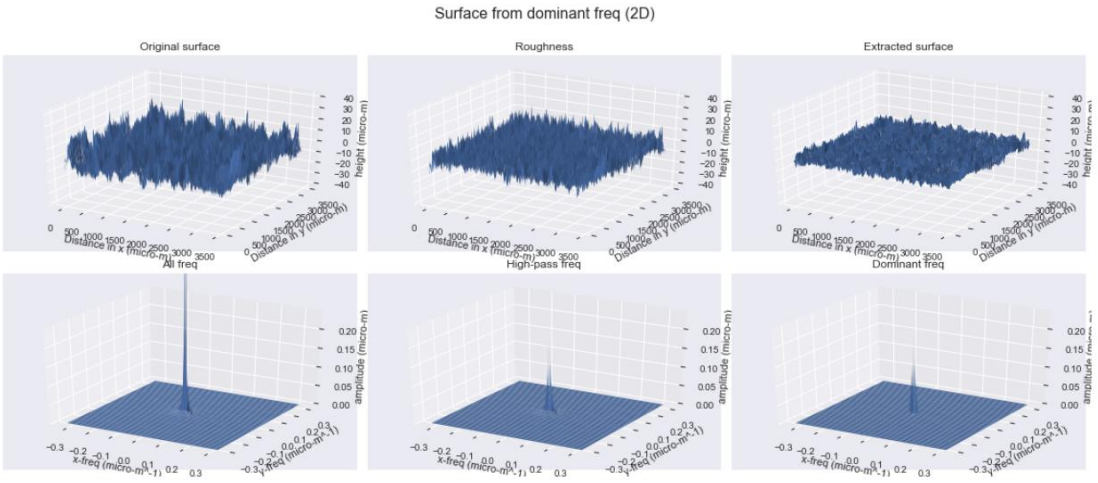


In [74]:
```
# plot surface 3D

plot_size = (16, 7)
fig = plt.figure(figsize=plot_size)
fig.suptitle('Surface from dominant freq (2D)', fontsize=16)


# First subplot
ax = fig.add_subplot(2, 3, 1, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = df_2D_corr

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Original surface')
```

```
# Second subplot
ax = fig.add_subplot(2, 3, 2, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_high_2D.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Roughness')
```

```
# third subplot
ax = fig.add_subplot(2, 3, 3, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_dom_2D.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-plt_dist_zlim_2D, plt_dist_zlim_2D)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Extracted surface')


# fourth subplot
ax = fig.add_subplot(2, 3, 4, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_2D_shift)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('All freq')
```

```
# fifth subplot
ax = fig.add_subplot(2, 3, 5, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_high_2D)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('High-pass freq')


# sixth subplot
ax = fig.add_subplot(2, 3, 6, projection='3d')

X = freq_full_shift
Y = freq_full_shift
X, Y = np.meshgrid(X, Y)
Z = 1/data_pnts**2 * abs(z_fou_dom_shift)

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(0, plt_freq_zlim_2D * 0.3)
ax.set(xlabel='x-freq (micro-m^-1)', ylabel='y-freq (micro-m^-1)', zlabel='amplitude (micro-m)')
ax.title.set_text('Dominant freq')


plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
```



In [75]:
```
# count amount of dominant freq
f_dom_no = (df_freq_dom != 0).astype(int).values.sum()

print('Amount of dominant freq: ', f_dom_no)
```

```
Amount of dominant freq:  1925
```

In [76]:
```
# compare result

rSq_rou2dom_2D = r2_score(z_inv_high_2D.real, z_inv_dom_2D.real)

print('Roughness to Dominant surface 2D')
print('R-square: ', rSq_rou2dom_2D)
```

```
Roughness to Dominant surface 2D
R-square:  0.565032253021999
```

6 Import data to csv file

6.1 1D import file

6.1.1 Variables of full-frequency

In [77]:
```
df_excel_1D = df_freq_dom_1D[['freq', 'high_conv', 'amplitude', 'dominant?']].copy()

# keep only rows with dominant freq
df_excel_1D = df_excel_1D[(df_excel_1D[['dominant?']] == True).any(axis=1)]
df_excel_1D = df_excel_1D.reset_index(drop=True)
```

In [78]:
```
df_excel_1D = df_excel_1D.drop(['dominant?'], axis=1)
df_excel_1D = df_excel_1D.rename(index=str, columns={'freq':'f(um)^-1',
                                                     'high_conv':'FourierCoef',
                                                     'amplitude':'A(um)'
                                                    })

df_excel_1D['FourierCoef'] = df_excel_1D['FourierCoef'] / data_pnts
df_excel_1D['Phs(rad)'] = np.angle(df_excel_1D['FourierCoef'])
```

6.1.2 Variables of half-frequency

In [79]:
```
# keep only positive freq

df_excel_1D_pos = df_excel_1D.copy()
df_excel_1D_pos = df_excel_1D_pos[(df_excel_1D_pos[['f(um)^-1']] >= 0).any(axis=1)]
df_excel_1D_pos = df_excel_1D_pos.reset_index(drop=True)
```

In [80]:
```
# correct the amplitude by *2

f_dom_no_1D_pos = int(f_dom_no_1D/2)

df_excel_1D_pos['FourierCoef'] = np.where(
    df_excel_1D_pos['f(um)^-1'] == 0,
    df_excel_1D_pos['FourierCoef'],
    2 * df_excel_1D_pos['FourierCoef']
)

df_excel_1D_pos['A(um)'] = np.where(
    df_excel_1D_pos['f(um)^-1'] == 0,
    df_excel_1D_pos['A(um)'],
    2 * df_excel_1D_pos['A(um)']
)

df_excel_1D_pos.head()
```

Out[80]:

	f(um)^-1	FourierCoef	A(um)	Phs(rad)
0	0.004376	(0.670996450330969-0.4619772105146483 5j)	0.814653	-0.602962
1	0.004667	(0.793785255673576+0.6860294776505427j)	1.049158	0.712709
2	0.004959	(-1.1333006923161002+0.6071917330858795j)	1.285711	2.649738
3	0.005542	(0.9187946650655916-1.0069793722892624j)	1.363155	-0.831158
4	0.006709	(-0.3538828799420414 3+0.8604519403188523j)	0.930382	1.960985

In [81]:
```
print('Amount of dominant freq (full): ', f_dom_no_1D)
print('Amount of dominant freq (half): ', f_dom_no_1D_pos)
```

```
Amount of dominant freq (full):  26
Amount of dominant freq (half):  13
```

In [82]:
```
# import dataframe to csv

df_excel_1D_pos.to_csv('onefile_variables_1D.csv', index=False)
```

6.1.3 Height of dominant surface

In [83]:
```
df_excel_1D_z_dom = pd.DataFrame(data = data_list, columns = ['x(um)'])
df_excel_1D_z_dom['z(um)'] = z_inv_dom_1D.real

df_excel_1D_z_dom.head()
```

Out[83]:

	x(um)	z(um)
0	0.0000	-0.207024
1	1.5625	-0.559982
2	3.1250	-0.905619
3	4.6875	-1.241616
4	6.2500	-1.565721

In [84]:
```
# cut length of imported height as assign

if assign_csv_len == 'y':
    df_excel_1D_z_dom = df_excel_1D_z_dom.iloc[0:csv_len_1D]

print('Processed length: ', data_pnts)
print('Imported length:  ', len(df_excel_1D_z_dom))
```

```
Processed length:  2194
Imported length:   500
```

In [85]:
```
# import dataframe to csv

df_excel_1D_z_dom.to_csv('onefile_height_1D.csv', index=False)
```

6.2 2D import file

6.2.1 Variables of full-frequency

In [86]:
```
df_freq_dom = pd.DataFrame(data = z_fou_dom_ishift)

# assign index to dataframe
df_freq_dom.index = freq_full
df_freq_dom.columns = freq_full
```

In [87]:
```
# unpivot table of df_freq_dom

df_excel_2D = df_freq_dom.unstack().reset_index(name='FourierCoef')
df_excel_2D.rename(columns={'level_0': 'fx(um)^-1', 'level_1': 'fy(um)^-1'}, inplace=True)
```

In [88]:
```
# assign amplitude and phase

df_excel_2D['FourierCoef'] = df_excel_2D['FourierCoef'] / data_pnts**2
df_excel_2D['A(um)'] = np.abs(df_excel_2D['FourierCoef'])
df_excel_2D['Phs(rad)'] = np.angle(df_excel_2D['FourierCoef'])


# remain only dominant freq

df_excel_2D = df_excel_2D[(df_excel_2D[['FourierCoef']] != 0).any(axis=1)]
df_excel_2D = df_excel_2D.reset_index(drop=True)
```

6.2.2 Variables of half-frequency

In [89]:
```
# remain only freq in half positive quarters

df_excel_2D_half = df_excel_2D.copy()
df_excel_2D_half = df_excel_2D_half[(df_excel_2D_half[['fx(um)^-1']] >= 0).any(axis=1)]
df_excel_2D_half = df_excel_2D_half.reset_index(drop=True)
```

In [90]:
```
# correct the amplitude by *2

f_dom_no_half = len(df_excel_2D_half)

df_excel_2D_half['FourierCoef'] = np.where(
    df_excel_2D_half['fx(um)^-1'] == 0,
    df_excel_2D_half['FourierCoef'],
    2 * df_excel_2D_half['FourierCoef']
)

df_excel_2D_half['A(um)'] = np.where(
    df_excel_2D_half['fx(um)^-1'] == 0,
    df_excel_2D_half['A(um)'],
    2 * df_excel_2D_half['A(um)']
)

df_excel_2D_half.head()
```

Out[90]:

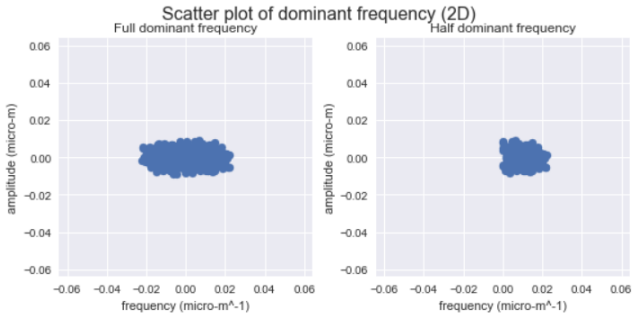
	fx(um)^-1	fy(um)^-1	FourierCoef	A(um)	Phs(rad)
0	0.000000	0.003792	(-0.008709647339385342+0.035264122900895443j)	0.036324	1.812934
1	0.000000	0.004667	(0.021494696489470 35+0.026929330473130 71j)	0.034456	0.897161
2	0.000000	-0.004667	(0.021494696489470 35-0.026929330473130 71j)	0.034456	-0.897161
3	0.000000	-0.003792	(-0.008709647339385347-0.035264122900895485j)	0.036324	-1.812934
4	0.000292	0.007876	(0.057873662000864 23+0.041137610208376266j)	0.071005	0.617949

In [91]:
```
print('Amount of dominant freq (full): ', f_dom_no)
print('Amount of dominant freq (half): ', f_dom_no_half)
```

```
Amount of dominant freq (full):  1925
Amount of dominant freq (half):  965
```

In [92]:
```
# import dataframe to csv

df_excel_2D_half.to_csv('onefile_variables_2D.csv', index=False)
```

In [93]:
```
# scatter plot to compare full & half dominant freq

axis_freq = (0, samp_rate, 0, samp_rate)

plot_size = (8, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Scatter plot of dominant frequency (2D)", fontsize=16)


## plot1
data = df_excel_2D[['fx(um)^-1', 'fy(um)^-1']].values
x1, y1 = data.T

ax1.scatter(x1,y1)
ax1.set_xlim((-samp_rate * 0.1, samp_rate * 0.1))
ax1.set_ylim((-samp_rate * 0.1, samp_rate * 0.1))
ax1.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax1.title.set_text('Full dominant frequency')


## plot2
data = df_excel_2D_half[['fx(um)^-1', 'fy(um)^-1']].values
x2, y2 = data.T

ax2.scatter(x2,y2)
ax2.set_xlim((-samp_rate * 0.1, samp_rate * 0.1))
ax2.set_ylim((-samp_rate * 0.1, samp_rate * 0.1))
ax2.set(xlabel='frequency (micro-m^-1)', ylabel='amplitude (micro-m)')
ax2.title.set_text('Half dominant frequency')


plt.tight_layout()
f.subplots_adjust(top=0.88)
```



6.2.3 Height of dominant surface

In [94]:
```
df_excel_2D_z_dom = pd.DataFrame(data = z_inv_dom_2D.real, columns = data_list)
df_excel_2D_z_dom.index = data_list

df_excel_2D_z_dom.iloc[0:5,0:5]
```

Out[94]:

	0.0	1.5625	3.125	4.6875	6.25
0.0000	-1.716525	-1.657371	-1.603700	-1.557744	-1.521363
1.5625	-1.709801	-1.673049	-1.641364	-1.616738	-1.600804
3.1250	-1.703856	-1.689785	-1.680346	-1.677281	-1.681988
4.6875	-1.698641	-1.707503	-1.720539	-1.739237	-1.764755
6.2500	-1.694102	-1.726119	-1.761832	-1.802468	-1.848937

In [95]:
```
# cut length of imported height as assign

if assign_csv_len == 'y':
    df_excel_2D_z_dom = df_excel_2D_z_dom.iloc[0:csv_len_2D, 0:csv_len_2D]

print('Processed length: ', data_pnts)
print('Imported length:  ', len(df_excel_2D_z_dom))
```

```
Processed length:  2194
Imported length:   500
```

In [96]:
```
# unpivot table

temp = df_excel_2D_z_dom.stack()
df_excel_2D_z_dom_melt = pd.DataFrame(temp)
df_excel_2D_z_dom_melt.columns = ['z(um)']
df_excel_2D_z_dom_melt = df_excel_2D_z_dom_melt.reset_index()
df_excel_2D_z_dom_melt = df_excel_2D_z_dom_melt.rename(index=str, columns={'level_0': 'x(um)',
                                                                          'level_1': 'y(um)'})

df_excel_2D_z_dom_melt.head()
```

Out[96]:

	x(um)	y(um)	z(um)
0	0.0	0.0000	-1.716525
1	0.0	1.5625	-1.657371
2	0.0	3.1250	-1.603700
3	0.0	4.6875	-1.557744
4	0.0	6.2500	-1.521363

In [97]:
```
# import dataframe to csv

df_excel_2D_z_dom_melt.to_csv('onefile_height_2D.csv', index=False)
```

7 Surface reconstruction

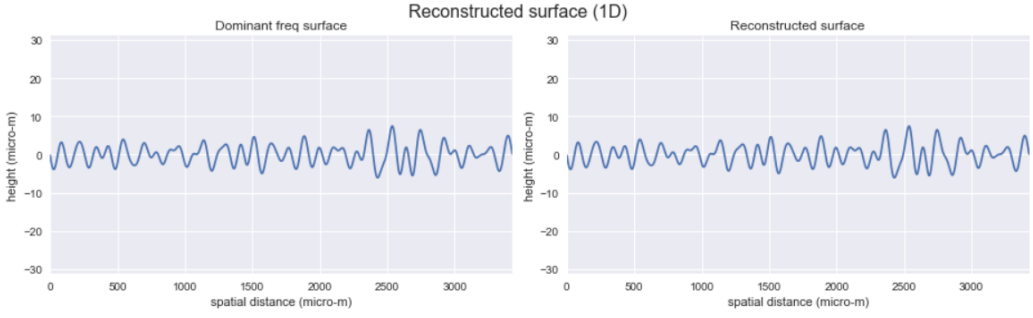
7.1 1D surface reconstruction

In [98]:
```
# reconstruct surface by superposition of sinusoidal wave

z_recon_1D_pos = np.zeros(data_pnts)
x = np.array(data_list)

for i in tqdm(range(f_dom_no_1D_pos)):

    A = df_excel_1D_pos['A(um)'].iloc[i]
    f = df_excel_1D_pos['f(um)^-1'].iloc[i]
    ph = df_excel_1D_pos['Phs(rad)'].iloc[i]

    zi = A * np.cos(2*np.pi*f*x + ph)
    z_recon_1D_pos = z_recon_1D_pos + zi

z_recon_1D_pos[0:5]
```

```
100%|██████████| 13/13 [00:00<00:00, 3258.39it/s]
```

Out[98]:
```
array([-0.20702446, -0.55998187, -0.90561902, -1.24161598, -1.56572061])
```

In [99]:
```
plot_size = (13, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Reconstructed surface (1D)", fontsize=16)

ax1.plot(data_list, z_inv_dom_1D.real)
ax1.set_xlim(0, plt_dist_xlim_1D)
ax1.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Dominant freq surface')

ax2.plot(data_list, z_recon_1D_pos)
ax2.set_xlim(0, plt_dist_xlim_1D)
ax2.set_ylim(-plt_dist_zlim_1D, plt_dist_zlim_1D)
ax2.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax2.title.set_text('Reconstructed surface')

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [100]:
```
# compare result

rSq_dom2recon_1D = r2_score(z_inv_dom_1D.real, z_recon_1D_pos)
rSq_rou2recon_1D = r2_score(z_inv_high_1D.real, z_recon_1D_pos)

print('Dominant to reconstructed surface 1D')
print('R-square: ', rSq_dom2recon_1D)
print('\nRoughness to reconstructed surface 1D')
print('R-square: ', rSq_rou2recon_1D)
```

```
Dominant to reconstructed surface 1D
R-square:  0.9999999829576853

Roughness to reconstructed surface 1D
R-square:  0.5188625270923166
```

7.2 2D surface reconstruction

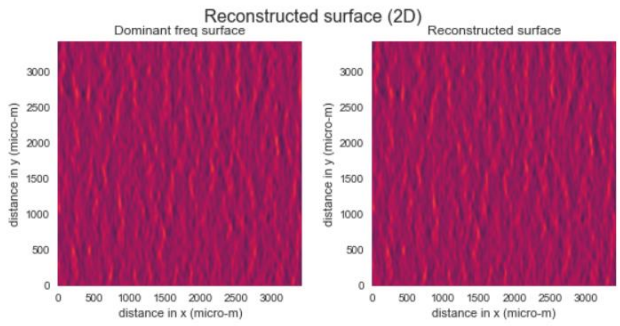
In [101]:
```
# reconstruct surface by superposition of sinusoidal wave

z_recon_2D_half = pd.DataFrame(np.zeros((data_pnts, data_pnts)))
x, y = np.meshgrid(data_list, data_list)

for i in tqdm(range(f_dom_no_half)):

    A = df_excel_2D_half['A(um)'].iloc[i]
    fx = df_excel_2D_half['fx(um)^-1'].iloc[i]
    fy = df_excel_2D_half['fy(um)^-1'].iloc[i]
    ph = df_excel_2D_half['Phs(rad)'].iloc[i]

    zi = A * np.cos(2*np.pi*(fx*x + fy*y) + ph)
    z_recon_2D_half = z_recon_2D_half + zi

z_recon_2D_half.iloc[0:5,0:5]
```

```
100%|██████████| 965/965 [02:56<00:00,  5.55it/s]
```

Out[101]:

	0	1	2	3	4
0	-1.749231	-1.690409	-1.636522	-1.589806	-1.552134
1	-1.742493	-1.706085	-1.674197	-1.648823	-1.631609
2	-1.736534	-1.722820	-1.713189	-1.709389	-1.712828
3	-1.731304	-1.740535	-1.753393	-1.771367	-1.795628
4	-1.726752	-1.759149	-1.794696	-1.834620	-1.879844

In [102]:
```
plot_size = (8, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Reconstructed surface (2D)", fontsize=16)

ax1.imshow(z_inv_dom_2D.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Dominant freq surface')
ax1.grid(False)

ax2.imshow(z_recon_2D_half.values.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Reconstructed surface')
ax2.grid(False)

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [103]:
```
# compare result

rSq_dom2recon_2D = r2_score(z_inv_dom_2D.real, z_recon_2D_half.values.real)
rSq_rou2recon_2D = r2_score(z_inv_high_2D.real, z_recon_2D_half.values.real)

print('Dominant to reconstructed surface 2D')
print('R-square: ', rSq_dom2recon_2D)
print('\nRoughness to reconstructed surface 2D')
print('R-square: ', rSq_rou2recon_2D)
```

```
Dominant to reconstructed surface 2D
R-square:  0.9999216785765779

Roughness to reconstructed surface 2D
R-square:  0.5650679433690099
```

Script of Multi-file processing

Contents

1 Input data and variables

In [1]:
```
# import python modole

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(color_codes=True)

from tqdm import tqdm
from scipy.fftpack import fft, ifft, fft2, ifft2, fftfreq, fftshift, ifftshift
from sklearn.metrics import r2_score
```

In [2]:
```
# import data files

input_files = ['149-1A1.csv',
               '149-1A2.csv',
               '149-1A3.csv',
               '149-1A4.csv',
               '149-1A5.csv',
               '149-1A6.csv',
               '149-1A7.csv',
               '149-1A8.csv',
               '149-1A9.csv',
               '149-1A10.csv',
              ]

no_input_files = len(input_files)
```

In [3]:
```
# select for 1D illustration

x_or_y = 'y'                            # data direction for 1D extraction (type 'x' or 'y')
row_no = 17                             # row no. for 1D data extraction
```

In [4]:
```
# assign constant

## outlier
outlier_angle = 65                      # degree

## Gaussian filter
lambda_c = 200                          # micro-m
freq_c = 1/lambda_c                     # (micro-m)^-1

## dominant freq
accept_dom_rSqua = 0.5                  # value from -1 to 1
                                          ## -1 to 0 = no relation
                                          ## 0 = estimated data 'poorly' fit the reference data
                                          ## 1 = estimated data 'well' fit the reference data

#-----------------------------------------------------------------------------------------------#

## constant for gaussian filter (do not touch)
alfa = 0.4697
beta = 0.2206
```

In [5]:
```
# to assign length of dominant surface output (.csv)
# cut one corner of the area

assign_csv_len = 'y'                    # 'y' if assign, 'n' if not-assign
csv_len_1D = 500
csv_len_2D = 500
```

2 Find common data length

In [6]:
```
def find_len(file):

# import data
    df = pd.read_csv(file, usecols=['# X','Y','Z'], skiprows=6)
    df = df.rename(index=str, columns={"# X": "X"})

# find length
    x_list = sorted(list(set(df['X'])))
    y_list = sorted(list(set(df['Y'])))

    x_pnts = len(x_list)
    y_pnts = len(y_list)

    len_data = min(x_pnts, y_pnts)

    if len_data % 2 != 0:
        len_data = len_data - 1

# find sample space
    samp_space = x_list[1] - x_list[0]

    return(len_data, samp_space)
```

In [7]:
```
list_data_len = []
list_samp_space = []

for file in tqdm(input_files):
    a, b = find_len(file)
    list_data_len.append(a)
    list_samp_space.append(b)

data_pnts = int(min(list_data_len))
samp_space = list_samp_space[0]

print(list_data_len)
print(data_pnts)
print(list_samp_space)
print(samp_space)
```

```
100%|██████████| 10/10 [00:45<00:00,  4.52s/it]
```

```
[2196, 2194, 2194, 2196, 2198, 2196, 2198, 2194, 2196, 2198]
2194
[1.5625, 1.5625, 1.5625, 1.5625, 1.5625, 1.5625, 1.5625, 1.5625, 1.5625, 1.5625]
1.5625
```

3 Surface transformation

3.1 Common part

In [8]:
```
# assign spatial axis

data_list = np.arange(0, data_pnts*samp_space, samp_space)
data_len = data_list[-1]

print(data_list[0:5], '...', data_list[-5:])
print(data_len)
```

```
[0.     1.5625 3.125  4.6875 6.25  ] ... [3420.3125 3421.875  3423.4375 3425.     3426.5625]
3426.5625
```

In [9]:
```
# assign frequency axis

samp_space = list_samp_space[0]          # micro-m
samp_rate = 1/samp_space                 # (micro-m)^(-1)

freq_full = fftfreq(data_pnts, samp_space)
freq_full_shift = fftshift(freq_full)

print(samp_space)
print(samp_rate)
```

```
1.5625
0.64
```

In [10]:
```
# Create Gaussian filter

## prepare fx/fc and fy/fc to put in the equation

df_fx = pd.DataFrame(np.zeros((data_pnts, data_pnts)))
for i in range(data_pnts):
    df_fx.iloc[i,:] = freq_full_shift                              # row by rows

df_fy = pd.DataFrame(np.zeros((data_pnts, data_pnts)))
for i in range(data_pnts):
    df_fy.iloc[:,i] = freq_full_shift                              # column by columns

df_fx2fc = df_fx.copy() / freq_c
df_fy2fc = df_fy.copy() / freq_c


## input variables to compute Gaussian filter 2D

df_low_gauss = np.exp(-np.pi*beta * (df_fx2fc.pow(2) + df_fy2fc.pow(2)))
df_high_gauss = 1 - df_low_gauss
```

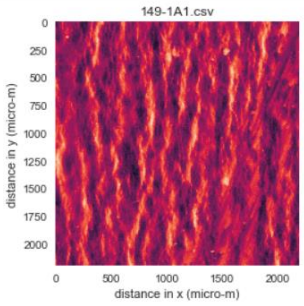
3.2 Loop operation

In [11]:
```
def extract_sur(file):


# import & pivot table

    ## import data
    df = pd.read_csv(file, usecols=['# X','Y','Z'], skiprows=6)
    df = df.rename(index=str, columns={"# X": "X"})

    ## pivot table
    df_2D = df.pivot_table(values='Z', index='Y', columns='X', aggfunc='first')
    df_2D = df_2D.iloc[0:data_pnts, 0:data_pnts]

#-----------------------------------------------------------------------------------------------------------------------#

# cut outlier

    df_2D_corr = df_2D.copy()


    ## detect in row direction

    for row_i in range(data_pnts):
        row = df_2D_corr.iloc[row_i,:].values

        for col_i in range(data_pnts-1):                # data_pnts-1 : cut last index - no next value to compare
            height = row[col_i + 1] - row[col_i]
            width = samp_space
            zeta = np.degrees(np.arctan(height/width))

            if zeta > outlier_angle:
                new_Z = row[col_i] + width * np.tan(np.radians(outlier_angle))
                df_2D_corr.iloc[row_i, col_i + 1] = new_Z

            if zeta < -outlier_angle:
                new_Z = row[col_i] - width * np.tan(np.radians(outlier_angle))
                df_2D_corr.iloc[row_i, col_i + 1] = new_Z


    ## detect in column direction

    for col_i in range(data_pnts):
        col = df_2D_corr.iloc[:, col_i].values

        for row_i in range(data_pnts-1):                # data_pnts-1 : cut last index - no next value to compare
            height = col[row_i + 1] - col[row_i]
            width = samp_space
            zeta = np.degrees(np.arctan(height/width))

            if zeta > outlier_angle:
                new_Z = col[row_i] + width * np.tan(np.radians(outlier_angle))
                df_2D_corr.iloc[row_i + 1, col_i] = new_Z

            if zeta < -outlier_angle:
                new_Z = col[row_i] - width * np.tan(np.radians(outlier_angle))
                df_2D_corr.iloc[row_i + 1, col_i] = new_Z

#-----------------------------------------------------------------------------------------------------------------------#

# FFT

    ## fast Fourier
    z_fou = fft2(df_2D_corr)
    z_inv = ifft2(z_fou)

    ## shift z_fou to prepare for convolution
    z_fou_fftshift = fftshift(z_fou)

#-----------------------------------------------------------------------------------------------------------------------#
```

```
# convolution

    ## low-pass convolution
    z_fou_low_conv = z_fou_fftshift.copy()
    z_fou_low_conv = np.multiply(z_fou_low_conv, df_low_gauss)

    ## high-pass convolution
    z_fou_high_conv = z_fou_fftshift.copy()
    z_fou_high_conv = np.multiply(z_fou_high_conv, df_high_gauss)

#-------------------------------------------------------------------------------------------------------------------#

# IFFT

    ## shift z_inv back prepared to compute image
    z_fou_low_conv_ishift = ifftshift(z_fou_low_conv)
    z_fou_high_conv_ishift = ifftshift(z_fou_high_conv)

    ## inverse Fourier
    z_inv_low = ifft2(z_fou_low_conv_ishift)
    z_inv_high = ifft2(z_fou_high_conv_ishift)

#-------------------------------------------------------------------------------------------------------------------#
```

```
# print image of surface

    plt.title(str(file))
    plt.imshow(df_2D_corr)
    plt.xlabel('distance in x (micro-m)')
    plt.ylabel('distance in y (micro-m)')
    plt.grid(False)
    plt.show()


    return(z_fou_high_conv_ishift)
```

In [12]:
```
# get summation of freq domain

df_freq_sum = pd.DataFrame(np.zeros((data_pnts, data_pnts)))

for file in tqdm(input_files):
    freq_result = extract_sur(file)
    df_freq_sum = df_freq_sum + freq_result
```



In [13]:
```
df_freq_sum.iloc[0:5,0:5]
```

Out[13]:

	0	1	2	
0	0j	(1575.7337838330277+1165.211620426719j)	(-20251.233431147815-11171.411692608042j)	(-30687.375177861915+50786.998
1	(-7799.786854088717+2579.949317759684j)	(18497.925981565742+27350.976357052j)	(58560.23306525167+27816.449386036336j)	(-17861.30268175453-56764.1093
2	(-14047.913675026419-10774.18909026809j)	(20870.098384083056-10393.466630387222j)	(48290.079187488765+13957.34870574787j)	(4131.8211901157765-7288.35333(
3	(34362.99539276372-2918.0371428906174j)	(40920.24857770439-9707.751004632133j)	(-44825.11050005212-395.6223656063903j)	(9292.310902208803+19652.7429(
4	(55590.307445250895-25963.93792783965j)	(47366.37653148155+119727.78499764355j)	(27003.680063404157+131246.49894198112j)	(109983.57957811319+64269.7681

4 Dominant frequency

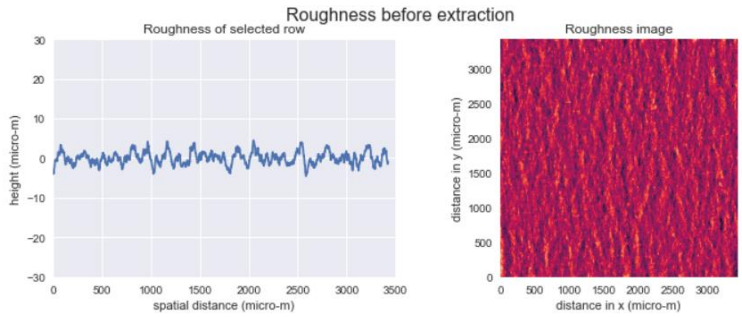
4.1 Automatically assign cut-off amplitude

In [14]:
```
# get average values of roughness from all files

df_freq_nom = df_freq_sum.copy() / no_input_files               # to normallize the sum of amplitude
df_freq_nom.index = freq_full                                   # set index name
df_freq_nom.columns = freq_full                                 # set column name


# prepare list of all freq -> to find cut-off amplitude of dominant freq

df_freq_amp = abs(df_freq_nom)
list_freq_amp = df_freq_amp.values.flatten()
list_freq_amp = sorted(list_freq_amp)
```

In [15]:
```
# IFFT for roughness

z_fou_rou = df_freq_nom.values
z_inv_rou = ifft2(z_fou_rou)
```

In [16]:
```
# prepare x-axis for 1D plot
t_data = data_list


# prepare y-axis for 1D plot
if x_or_y == 'x':
    z_inv_select_rou = z_inv_rou[:, row_no]

if x_or_y == 'y':
    z_inv_select_rou = z_inv_rou[row_no, :]
```

In [17]:
```
# roughness plot

plot_size = (10, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Roughness before extraction", fontsize=16)
axis_dist = (0, data_len, 0, data_len)                                    # assign axis for spatial domain
axis_freq = (0, samp_rate, 0, samp_rate)                                  # assign axis for freq domain
v_min_dist = np.amin(z_inv_rou.real)
v_max_dist = np.amax(z_inv_rou.real)

ax1.plot(t_data, z_inv_select_rou)
ax1.set_xlim(0, 3500)
ax1.set_ylim(-30, 30)
ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
ax1.title.set_text('Roughness of selected row')

ax2.imshow(z_inv_rou.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Roughness image')
ax2.grid(False)


plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [18]:
```
# create table to compare result based on amount of dominant freq

no_dom_freq_per = np.array([0.0025,0.005,0.01,
                            0.02,0.04,0.06,0.08,0.1,
                            0.2,0.4,0.6,0.8,1.0,
                            1.5,2.0])

df_dom_samp = pd.DataFrame(data = no_dom_freq_per, columns = ['no_f_per'])
df_dom_samp['no_f'] = df_dom_samp['no_f_per'] * len(list_freq_amp) / 100
df_dom_samp['no_f_int'] = df_dom_samp['no_f'].astype(int) + 1
df_dom_samp['f_dom_amp_c'] = np.zeros(len(no_dom_freq_per))
df_dom_samp['result_error'] = np.zeros(len(no_dom_freq_per))


# find cut-off amplitude for dominant freq

for i in range(len(no_dom_freq_per)):
    df_dom_samp.loc[i,'f_dom_amp_c'] = list_freq_amp[-df_dom_samp.loc[i,'no_f_int']]
```

In [19]:
```
def find_error_dom_f(freq_dom_amp_c):

    df_freq_dom = df_freq_nom.copy()

    for x in df_freq_dom.columns:
        df_freq_dom[x] = np.where(
            df_freq_amp[x] > freq_dom_amp_c,
            df_freq_dom[x],
            0.0
        )

    z_fou_dom = df_freq_dom.values
    z_inv_dom = ifft2(z_fou_dom)

#-----------------------------------------------------------------------------------------------------------------------#
```

```
    # extract row for 1D plot

    if x_or_y == 'x':
        z_inv_select_dom = z_inv_dom[:, row_no]

    if x_or_y == 'y':
        z_inv_select_dom = z_inv_dom[row_no, :]

#-----------------------------------------------------------------------------------------------------------------------#

    # find error

    rSq_rou2dom = r2_score(z_inv_rou.real, z_inv_dom.real)

#-----------------------------------------------------------------------------------------------------------------------#

    # plot graph to compare

    title = 'R-Square error: ' + str("{0:.2f}".format(rSq_rou2dom))

    plot_size = (10, 4)
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
    f.suptitle(title, fontsize=16)
    axis_dist = (0, data_len, 0, data_len)                                    # assign axis for spatial domain
    axis_freq = (0, samp_rate, 0, samp_rate)                                  # assign axis for freq domain

    ax1.plot(t_data, z_inv_select_dom)
    ax1.set_xlim(0, 3500)
    ax1.set_ylim(-30, 30)
    ax1.set(xlabel='spatial distance (micro-m)', ylabel='height (micro-m)')
    ax1.title.set_text('Roughness of selected row')

    ax2.imshow(z_inv_dom.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
    ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
    ax2.title.set_text('Roughness image')
    ax2.grid(False)

    plt.tight_layout()
    f.subplots_adjust(top=0.88)


    return rSq_rou2dom
```

In [20]:
```
# find result error between roughness and extracted surface

for i in tqdm(range(len(no_dom_freq_per))):
    df_dom_samp.loc[i,'result_error'] = find_error_dom_f(df_dom_samp.loc[i,'f_dom_amp_c'])

df_dom_samp
```

Out[20]:

	no_f_per	no_f	no_f_int	f_dom_amp_c	result_error
0	0.0025	120.3409	121	171811.292597	0.140790
1	0.0050	240.6818	241	134402.221617	0.224556
2	0.0100	481.3636	482	102504.700507	0.326053
3	0.0200	962.7272	963	73840.864047	0.441083
4	0.0400	1925.4544	1926	49675.006142	0.551398
5	0.0600	2888.1816	2889	40625.135937	0.613033
6	0.0800	3850.9088	3851	34382.845693	0.655891
7	0.1000	4813.6360	4814	30203.741596	0.688403
8	0.2000	9627.2720	9628	19307.844581	0.777598
9	0.4000	19254.5440	19255	11862.420554	0.847890
10	0.6000	28881.8160	28882	8815.129471	0.880396
11	0.8000	38509.0880	38510	7136.341349	0.900000

In [21]:
```
# automatically assign cut-off dominant amplitude by acceptable percentage of error

for i in range(len(no_dom_freq_per)):
    if df_dom_samp.loc[i,'result_error'] > accept_dom_rSqua:
        freq_dom_amp_c = df_dom_samp.loc[i,'f_dom_amp_c']
        break

freq_dom_amp_c
```

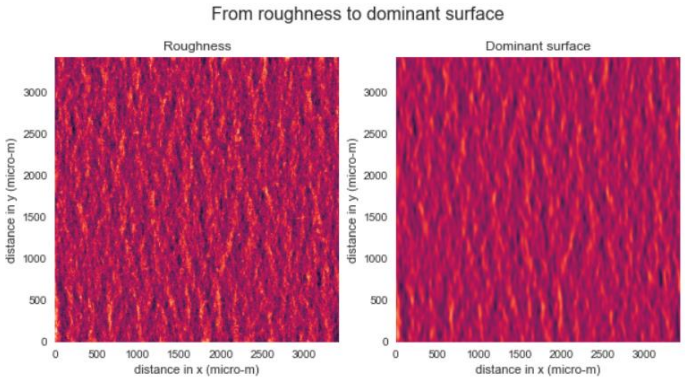
Out[21]: 49675.00614180178

4.2 Extraction operation

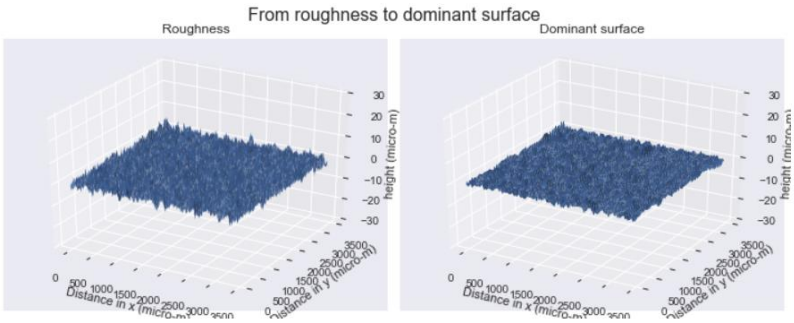
In [22]:
```
# remain only dominant freq in matrix

df_freq_dom = df_freq_nom.copy()

for x in df_freq_dom.columns:
    df_freq_dom[x] = np.where(
        df_freq_amp[x] > freq_dom_amp_c,
        df_freq_dom[x],
        0.0
    )
```

In [23]:
```
z_fou_dom = df_freq_dom.values
z_inv_dom = ifft2(z_fou_dom)
```

In [24]:
```
# plot surface image

plot_size = (10, 5)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("From roughness to dominant surface", fontsize=16)

axis_dist = (0, data_len, 0, data_len)
axis_freq = (0, samp_rate, 0, samp_rate)
v_min_dist = np.amin(z_inv_rou.real)
v_max_dist = np.amax(z_inv_rou.real)

ax1.imshow(z_inv_rou.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Roughness')
ax1.grid(False)

ax2.imshow(z_inv_dom.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Dominant surface')
ax2.grid(False)
```



In [25]:
```
# plot surface 3D

from matplotlib import cm
from mpl_toolkits.mplot3d.axes3d import get_test_data
from mpl_toolkits.mplot3d import Axes3D

plot_size = (10, 4)
fig = plt.figure(figsize=plot_size)                                    # set up figure size
fig.suptitle('From roughness to dominant surface', fontsize=16)


# first subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_rou.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-30, 30)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Roughness')


# second subplot
ax = fig.add_subplot(1, 2, 2, projection='3d')

X = data_list
Y = data_list
X, Y = np.meshgrid(X, Y)
Z = z_inv_dom.real

surf = ax.plot_surface(X, Y, Z)
ax.set_zlim(-30, 30)
ax.set(xlabel='Distance in x (micro-m)', ylabel='Distance in y (micro-m)', zlabel='height (micro-m)')
ax.title.set_text('Dominant surface')


plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
```



In [26]:
```
# count amount of dominant freq
f_dom_no = (df_freq_dom != 0).astype(int).values.sum()


# find error
rSq_rou2dom = r2_score(z_inv_rou.real, z_inv_dom.real)


print('Amount of dominant freq: ', f_dom_no)
print('\nRoughness to Dominant surface')
print('R-square: ', rSq_rou2dom)
```

```
Amount of dominant freq:  1924

Roughness to Dominant surface
R-square:  0.5513978803255862
```

5 Import data to csv file

5.1 Variables of frequency

In [27]:
```
# variables of full frequency

## assign index to dataframe
df_freq_dom.index = freq_full
df_freq_dom.columns = freq_full

## unpivot table of df_freq_dom
df_excel = df_freq_dom.unstack().reset_index(name='FourierCoef')
df_excel.rename(columns={'level_0': 'fx(um)^-1', 'level_1': 'fy(um)^-1'}, inplace=True)

## assign amplitude and phase
df_excel['FourierCoef'] = df_excel['FourierCoef'] / data_pnts**2
df_excel['A(um)'] = np.abs(df_excel['FourierCoef'])
df_excel['Phs(rad)'] = np.angle(df_excel['FourierCoef'])

## remain only dominant freq
df_excel = df_excel[(df_excel[['FourierCoef']] != 0).any(axis=1)]
df_excel = df_excel.reset_index(drop=True)
```

In [28]:
```
# variables of half positive frequency

## remain only freq in half positive quarters
df_excel_half = df_excel.copy()
df_excel_half = df_excel_half[(df_excel_half[['fx(um)^-1']] >= 0).any(axis=1)]
df_excel_half = df_excel_half.reset_index(drop=True)

## count number of frequency
f_dom_no_half = len(df_excel_half)

## correct amplitude by *2
df_excel_half['FourierCoef'] = np.where(
    df_excel_half['fx(um)^-1'] == 0,
    df_excel_half['FourierCoef'],
    2 * df_excel_half['FourierCoef']
)
df_excel_half['A(um)'] = np.where(
    df_excel_half['fx(um)^-1'] == 0,
    df_excel_half['A(um)'],
    2 * df_excel_half['A(um)']
)

df_excel_half.head()
```

Out[28]:

	fx(um)^-1	fy(um)^-1	FourierCoef	A(um)	Phs(rad)
0	0.000292	0.008168	(0.019848979934659602+0.01453219553557038j)	0.024600	0.631971
1	0.000292	0.008459	(0.011432768237928903+0.024350738896880452j)	0.026901	1.131842
2	0.000292	-0.007001	(-0.019339594653113724+0.017011778036649803j)	0.025757	2.420144
3	0.000583	0.005542	(-0.01965557637332621+0.006636025949564915j)	0.020746	2.815993
4	0.000583	-0.008751	(-0.01947184776190497+0.010480409161901209j)	0.022113	2.647828

In [29]:
```
print('Amount of dominant freq (full): ', f_dom_no)
print('Amount of dominant freq (half): ', f_dom_no_half)
```

```
Amount of dominant freq (full):  1924
Amount of dominant freq (half):  962
```

In [30]:
```
# import dataframe to csv
df_excel_half.to_csv('multifile_variables_2D.csv', index=False)
```

5.2 Height of dominant surface

In [31]:
```
# create dataframe
df_excel_z_dom = pd.DataFrame(data = z_inv_dom.real, columns = data_list)
df_excel_z_dom.index = data_list
```

In [32]:
```
# cut length of imported height as assign
if assign_csv_len == 'y':
    df_excel_z_dom = df_excel_z_dom.iloc[0:csv_len_2D, 0:csv_len_2D]

print('Processed length: ', data_pnts)
print('Imported length:  ', len(df_excel_z_dom))
```

```
Processed length:  2194
Imported length:   500
```

In [33]:
```
# unpivot table
temp = df_excel_z_dom.stack()
df_excel_z_dom_melt = pd.DataFrame(temp)
df_excel_z_dom_melt.columns = ['z(um)']
df_excel_z_dom_melt = df_excel_z_dom_melt.reset_index()
df_excel_z_dom_melt = df_excel_z_dom_melt.rename(index=str, columns={'level_0': 'x(um)',
                                                                     'level_1': 'y(um)'})

df_excel_z_dom_melt.head()
```

Out[33]:

	x(um)	y(um)	z(um)
0	0.0	0.0000	-0.486254
1	0.0	1.5625	-0.308512
2	0.0	3.1250	-0.126821
3	0.0	4.6875	0.054399
4	0.0	6.2500	0.230772

In [34]:
```
# import dataframe to csv
df_excel_z_dom_melt.to_csv('multifile_height_2D.csv', index=False)
```

6 Surface reconstruction

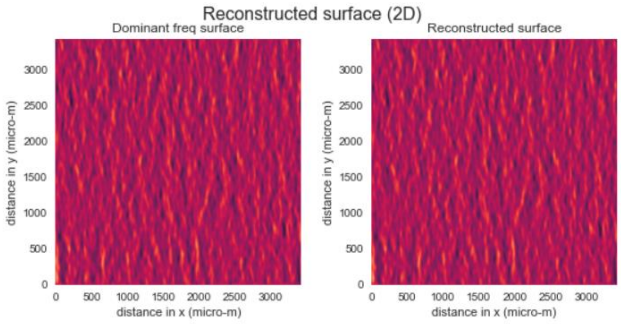
In [35]:
```
# reconstruct surface by superposition of sinusoidal wave

z_recon_half = pd.DataFrame(np.zeros((data_pnts, data_pnts)))
x, y = np.meshgrid(data_list, data_list)

for i in tqdm(range(f_dom_no_half)):

    A = df_excel_half['A(um)'].iloc[i]
    fx = df_excel_half['fx(um)^-1'].iloc[i]
    fy = df_excel_half['fy(um)^-1'].iloc[i]
    ph = df_excel_half['Phs(rad)'].iloc[i]

    zi = A * np.cos(2*np.pi*(fx*x + fy*y) + ph)
    z_recon_half = z_recon_half + zi

z_recon_half.iloc[0:5,0:5]
```

```
100%|██████████| 962/962 [02:53<00:00,  5.85it/s]
```

Out[35]:

	0	1	2	3	4
0	-0.486254	-0.308512	-0.126821	0.054399	0.230772
1	-0.535573	-0.365373	-0.190628	-0.015683	0.155145
2	-0.584798	-0.422215	-0.254488	-0.085883	0.079345
3	-0.633845	-0.478949	-0.318304	-0.156096	0.003485
4	-0.682631	-0.535480	-0.381973	-0.226212	-0.072321

In [36]:
```
plot_size = (8, 4)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=plot_size)
f.suptitle("Reconstructed surface (2D)", fontsize=16)

ax1.imshow(z_inv_dom.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax1.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax1.title.set_text('Dominant freq surface')
ax1.grid(False)

ax2.imshow(z_recon_half.values.real, extent=axis_dist, vmin=v_min_dist, vmax=v_max_dist)
ax2.set(xlabel='distance in x (micro-m)', ylabel='distance in y (micro-m)')
ax2.title.set_text('Reconstructed surface')
ax2.grid(False)

plt.tight_layout()
f.subplots_adjust(top=0.88)
```



In [37]:
```
# compare result

rSq_dom2recon = r2_score(z_inv_dom.real, z_recon_half.values.real)
rSq_rou2recon = r2_score(z_inv_rou.real, z_recon_half.values.real)

print('Dominant to reconstructed surface 2D')
print('R-square: ', rSq_dom2recon)
print('\nRoughness to reconstructed surface 2D')
print('R-square: ', rSq_rou2recon)
```

```
Dominant to reconstructed surface 2D
R-square:  1.0

Roughness to reconstructed surface 2D
R-square:  0.5513978803255862
```