AESM1511

Assignment 3.1: Frequency-Wavenumber Filtering

Introduction

In the previous assignment, you tried to filter surface waves in the frequency domain, i.e., you applied a one-dimensional filter. You also performed two-dimensional Fourier transformation from the time-space domain to the frequency-wavenumber (f-k) domain. Now, you are going to apply a two-dimensional filter in the f-k domain, i.e., an f-k filter.

Tasks and questions

1. We are going to use the model from Assignment 2. The reflection common-source gather is also the same, except that now the latest times are tapered.
   Write a function file to read the reflection data and use the function in the main file.
   Visualize the reflection data (the common-source gather) in a figure using imagesc. Use the parameters from Assignment 2, Task 1. Further to that, scale the colorbar to 0.01 of the maximum and minimum value.
2. Now, we are going to transform the common-source gather from the time-space domain to the frequency-wavenumber domain.
   A plane wave propagating along the x-axis can be described as $Ae^{-i(\omega t-kx)}$. (A seismic record can be represented as a summation of plane waves.) This means that if we are going to transform the data to the f-k domain, we would need to perform a transformation to the frequency domain by multiplying the Fourier integrand by $e^{-i\omega t}$ and a transformation to the wavenumber domain by multiplying the Fourier integrand by e^{ikx}. That is, the two factors have opposite signs. Read in the help file of *fft()* how the fft is implemented in Matlab and decide how to perform the f-k transformation using *fft()* and *ifft()*. Explain for which transform you use *fft()* and for which *ifft()*.
3. We are now going to perform f-k filtering to filter out the surface-wave energy. For this, write a code that uses as an input an estimate of the propagation velocity of the surface waves. This velocity defines a cone in the f-k plane when taking the positive and negative wavenumbers into account. Visualize the data in the f-k domain using the parameters from Assignment 2, Task 5. Decide what velocity you want to use. Mute the energy outside the velocity cone, i.e., to the left and right of the cone. Based on your experiments with the three types of low-cut filter in Assignment 2, you know that you should use a linear or a sin filter. Use a linear filter.
4. Transform the data back to the time-space domain. What do you observe about the quality of the surface-wave events when comparing them to the events in the original gather?

Assignment 3.2: Interpolation and Integration of Diffusive Fields

Introduction

In exploration geophysics, wave and diffusive-field signals are computed in difficult geometries. That means that the computational costs might be very high, and we

would like to minimize our computational efforts. Time-domain signals are transformed to the frequency domain and time-domain results are obtained again. Controlled-source electromagnetic fields are diffusive fields and behave as smooth functions in the space-time and in the space-frequency domains. For modelling, usually logarithmic frequency spacing is sufficient to capture the field. To transform the frequency-domain data back to the time domain, though, linear frequency spacing is necessary.
Suppose the function we need is known in a transformed domain and the space-frequency domain function must be computed from the following integral:

$$\hat{F} = i\omega \int_{\kappa=0}^{\infty} \frac{exp(-\Gamma h)}{\Gamma} J_0(\rho\kappa) d\kappa, \quad (1)$$

where $\Gamma = \sqrt{\kappa^2 + \gamma^2}, \gamma^2 = i\omega\sigma\mu$, with $\omega = 2\pi f$ denoting angular frequency, f – frequency, σ – conductivity, and μ – magnetic permeability; h is the vertical distance between transmitter and receiver, and κ is the integration variable. The function $J_0(\rho\kappa)$, with ρ being the horizontal distance, is the Bessel function of order zero. You can compute this function using the command *besselj*.

Tasks and questions

1. Create a function file, which gets the variables κ, γ, ρ, h as input arguments and the function value as an output argument. In this function file, write the code for the integrand in equation (1).
2. Create an m-file in which you take $\sigma = 1, \mu = 4\pi \times 10^{-7}, \rho = 1000, h = 500$. You make your own choices for the values of the frequency, e.g., $f = 0.1$, or $f = 1$, or $f = 10$. Use the integration routine *quadgk* and compute the integral of equation (1), where the integration routine uses the function created in task 1.
   The above result should theoretically be equal to
   $$\hat{F} = i\omega I_0\left(\frac{1}{2}\gamma\left(\sqrt{\rho^2 + h^2} - h\right)\right) K_0\left(\frac{1}{2}\gamma\left(\sqrt{\rho^2 + h^2} + h\right)\right), \quad (2)$$
   where $I_0(\zeta)$ and $K_0(\zeta)$ are modified Bessel functions of order zero of the first and second kinds, respectively. You can compute these zeroth-order Bessel functions using the commands *besseli* and *besselk*, see the help-function in Matlab for further details. Display the error between the two computational methods, i.e., using equation (1) and using equation (2), to screen.
3. Create a discrete frequency axis with a logarithmic spacing from $f = 10^{-2.5}$ Hz to $f = 10^{2.5}$ Hz spanning 5 decades (meaning 5 powers of 10), using 2 points per decade. Note that this does not force you to use the Matlab command *logspace*. Compute the integral for all frequencies by using a *for* loop over all frequencies, and store all results in an array.

Now, we will use an interpolation algorithm to minimize the number of numerical integrations we have to do to find the function for all frequencies between the values $f = 10^{-2.5}$ Hz and $f = 10^{2.5}$ Hz. We use Piecewise Cubic Hermite Interpolating Polynomials, the function *pchip* in Matlab, to find function values at frequencies for which we did not compute them. Suppose we have the function $\hat{F}$ in an array for 11 points (what is asked in Task 3), we take away the even points,

and try to estimate the function values at these missing points by interpolation using *pchip*. We need as input the 6 frequency values, stored in *freq*(1:6), where we know the function, stored in *fld*(1:6), and the new frequency values, stored in *freqint*(1:11), for which we will create function values, stored in *fldint*(1:11), using *pchip*. This could look like: *fldint=pchip(freq,fld,freqint)*. Because we have also computed the values at all even frequency points, we can test the accuracy of the interpolated result.

4. Remove all of the function values in the even indices and keep the other 6 values. Estimate the removed values by interpolation using *pchip*, and compute the normalized error in the amplitudes of the interpolated result. Compute the normalization with respect to the function's maximum amplitude for all frequencies of the original values.

If at any frequency the amplitude of the computed field value is higher than 10^{-2} times the function's maximum amplitude over all frequencies, and the amplitude of the interpolation result has an error larger than 1% (with respect to the function's maximum amplitude for all frequencies), then we use the already computed result (at the point we were doing the interpolation) and choose new frequency points to the left and right of the previous interpolated result by bisecting the frequency interval. If at least one of the conditions is not met, we just keep the already computed value. We repeat this for all odd frequency values. Now we are done with the first level of bisecting and will look for possible need to further bisect some intervals.

We start again the loop over all even frequency points. If at any frequency the amplitude of the field value is higher than 10^{-2} times the function maximum amplitude over all frequencies, and the amplitude of the interpolation result has an error larger than 1%, we bisect again and we continue until for all points, where the amplitude of the field value is higher than 10^{-2} times the function maximum amplitude over all frequencies, the interpolated values have an error less than 1% compared to computed values. This occurs when no more frequency points are introduced. Then, we are ready and can interpolate to any frequency value in the frequency range between $f = 10^{-2.5}$ Hz and $f = 10^{2.5}$ Hz. The resulting frequency values are not at equidistant spacing on a logarithmic scale, because we have adapted the grid to the needs for accurate interpolation.

5. Write the code for carrying out this interpolation of function values computed using the integration of the function in equation (1) and testing of the accuracy of the interpolated results. For how many frequencies did you need to compute the integral? Plot the real and imaginary parts, using different markers (no line plots), as a function of the frequencies used, so that it can be clearly seen how irregular the frequency axis is discretized. Add a legend to the plot to identify real and imaginary parts and annotate the axes.
6. Now, the lowest frequency you have used is $f = 10^{-2.5}$ Hz, which is the value to be used for the constant step on the linear frequency axis to be used for the fft. The maximum frequency is $f = 10^{2.5}$ Hz, hence you need *nf*=100000 points on the linear frequency axis with step $f = 10^{-2.5}$ Hz. At zero frequency, the function is zero. Compute the field for all frequencies by

interpolating the field computed at the limited number of frequencies obtained in Task 5.

7. Because discrete signals are cyclic over their lengths determined by the step-sizes, we can compute the time-step that corresponds to the discrete range of frequencies we have chosen in Task 3. The time-step is obtained as $dt = 1/(df * 2 * nf)$. Turn the result from Task 6 to the time domain using *ifft()*. Suppose the field obtained in Task 7 is called *awint* and it has a length *nf*=100000, the time-domain result is obtained by using something like *awt=2*real(ifft(awint,2*nf))/dt*. As you have seen from the previous assignment, the output of *ifft()* is complex while your time-domain result is real and is contained in the real part of the output from *ifft()*. Plot this result on a log-log scale between 0.01 < t < 200. Make sure the sign of the function can still be seen in the plot by giving the negative function values a different color than the positive function values. You must write the code such that no warning messages appear on the screen. Give proper annotations and labels to the axes and make a legend showing the results.
8. Make a flowchart of the final code.

Submitting your results

Submit your complete solution before 18.00 on October 24, 2022 in Brightspace under Assignments at the top of the course's webpage. Submit the results in the form of a zip file containing an executable .m file (or files if applicable) and a file (or files if applicable) with the flowchart. The name of the zip file should be

AESM1511_2022_matlab_a3_surname_studentnumber.zip

or

AESM1511_2022_matlab_a3_surname1_surname2_surname3.zip if you are working in a group. Working in groups is encouraged. Still, working in groups of more than 3-4 people is not effective and not encouraged.