STUDIA UNIVERSITATIS BABEŞ-BOLYAI
Engineering 69(1) 2024 DOI: 10.24193/subbeng.2024.1.9

FocusedDFT: A DFT implementation for vibration-based structural damage detection

Petar Prvulović

Abstract. *This paper introduces focusedDFT, a novel Discrete Fourier Transform (DFT) implementation optimized for short signals with low frequency components, particularly where a single frequency component is of interest. The results suggest focusedDFT as a viable and efficient solution for vibration-based damage detection methods, offering improved execution times compared to traditional DFT approaches.*

Keywords: *focusedDFT, DFT, frequency bins, vibration-based damage detection*

1. Introduction

Detecting structural defects and cracks is crucial in mechanical engineering, impacting safety, reliability, and the longevity of structures. Damage from stress, fatigue, environmental factors, material flaws, or vibrations can lead to serious failures, making early detection essential. Vibration-based damage detection has emerged as a valuable non-destructive method, leveraging a structure's dynamic response to identify potential defects. Vibration-based damage detection involves analyzing changes in natural frequencies, typically by recording vibrations with sensors and extracting frequency components using Fourier transforms. The Discrete Fourier Transform (DFT) converts signals from the time domain to the frequency domain, which allows engineers to pinpoint frequency components that may indicate structural damage. However, precise frequency detection is challenging due to the discrete nature of digital computation.

One way to improve precision is to adjust the signal length by cropping or padding, which changes the frequency bin resolution and moves the bins closer to the actual monitored frequency. This approach can reduce spectral leakage and increase spectral amplitudes, as signal energy is less dispersed to neighboring bins. Methods based on this idea iteratively recalculate the DFT for various signal lengths. A drawback of standard Fast Fourier Transform (FFT) implementations in this case is that they often split the signal into chunks of specific lengths, typically powers of two, in order to improve the efficiency, which can introduce unwanted effects like spectral leakage and interfere with intended signal length manipulation.

In vibration-based damage detection, high precision is required, and it often involves short signals with low frequency components and non-integer number of periods. A “clean” DFT algorithm is preferred in this case as it doesn’t interfere with signal length manipulation. The challenge with DFT is its order of complexity which can be slow when used in iterative signal length manipulation.

This paper introduces *focusedDFT*, a DFT implementation suited for processing short signals with low frequency components, targeting a single frequency component. This implementation processes only a specified range of frequency bins around the targeted frequency, offering execution times practical for the intended application.

The paper is organized as follows: Section 2 discusses background information and related work. Section 3 introduces the algorithm implementation and helper functions in Python. Section 4 presents the results of testing the helper functions and implemented DFT algorithm output, comparing execution times to NumPy's FFT. Section 5 discusses the results and the potential for *focusedDFT* use in methods for improving DFT precision.

2. Background and Related Work

Vibration-based structural damage detection requires precise analysis of acquired signals, as damage is detected by noticing changes in the natural frequencies of the structure. In these cases, signals are often short and contain low-frequency components, typically below 100 Hz [1]. The discrete nature of the Discrete Fourier Transform (DFT) results in a set of discrete frequency bins, where the exact frequency may not be captured, causing the signal's energy to be spread to neighboring bins, leading to spectral leakage. Various interpolation methods can mitigate this issue to some extent. However, these methods may produce poor results when low-frequency components are close and neighboring bins overlap. One possible approach involves gradually changing the signal length by cropping or padding to adjust the frequency bin resolution, moving frequency bins closer to the actual frequency of the targeted component [1-4]. Standard Fast Fourier Transform (FFT) implementations enhance efficiency by chunking the signal into lengths that are powers of two, but this can interfere with the aforementioned approach [5], The DFT's complexity is generally acceptable for processing short signals, but it becomes problematic with iterative signal length manipulation as execution time increases.

Focusing on the frequency range of interest can reduce the number of iterations and provide a faster DFT implementation without compromising precision. This paper introduces focusedDFT, a targeted approach to processing short signals with low-frequency components efficiently and accurately.

3. Discrete Fourier Transform

Discrete Fourier transform formula is:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{k}{N}n} \tag{1}$$

where *N* represents number of samples in the signal and X_k represents the value of k-th frequency bin in signal spectrum. This sum can be implemented as a for loop which *n* over the range [0, N-1] for each *k* frequency bin. The part $\frac{2\pi}{N}$ can be extracted into variable *D*, which is calculated before the loop, to save on total execution time, as defined in formula 2:

$$D = \frac{2\pi}{N} \tag{2}$$

By applying the Euler's formula (3) e^{ix} can be replaced with sum of sines and cosines. This improves the DFT implementation in two ways: sinus and cosines in Python are using native C functions which use precalculated tables, which should be faster than calculating the power of e, and imaginary and real part are separate so there is no need to introduce additional libraries or helper functions to deal with that part.

$$e^{ix} = \cos(x) + i\sin(x) \tag{3}$$

After applying formulas 2 and 3 into formula 1 we get the following expression:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \cos(-knD) + ix_n \cdot \sin(-knD) \tag{4}$$

which can be further simplified by identities:

$$\begin{gathered} \cos(-\alpha) = cos(\alpha) \\ \sin(-\alpha) = -sin(\alpha) \end{gathered} \tag{5}$$

which brings the following formula:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \cos(knD) - ix_n \cdot \sin(knD) \tag{6}$$

X_k values are complex numbers and need to be converted to real numbers to form the signal spectrum. As each X_k contains real and imaginary part, which are on their respective axes, Euclidean distance of the (Re,Im) coordinate from x axis is calculated and used for signal spectra. If formula 6 is represented as sum of Real and Imaginary parts, as shown in formula 7, k-th frequency bin value is calculated per formula 8.

$$X_k = \sum_{n=0}^{N-1}(Re - Im) = \sum_{n=0}^{N-1} Re - \sum_{n=0}^{N-1} Im = SRe - SIm \qquad (7)$$

$$X'_k = \sqrt{SRe^2 + SIm^2} \qquad (8)$$

This can be implemented as a double for loop which iterates *k* over the range [0, length(signal)) and for each *k* iterates *n* over the range [0, N-1]. As we are concerned about only a specific range of frequency bins around the targeted frequency, outer loop can iterate over that range only, which is a basis for the FocusedDFT.

4. Methodology and materials

The FocusedDFT was implemented as a function *focused_dft* in Python. To test the function, we created a test script, and a helper function called generateSignal to generate sinusoidal signals with desired frequency components, amplitudes, and phase shifts for a given sampling rate. This function generates an array of double-type values.

FocusedDFT was designed to calculate the DFT in a straightforward manner, looping through the signal with two nested loops to compute the spectrum. This function returns an array of double-type values representing the spectrum. To save on computation time, the function supports limiting the frequency range for calculation. For iterative signal length manipulation, the function also supports signal padding and cropping for the desired number of samples from the start or end of the signal array.

We used the standard FFT function from the numpy package, *numpy.fft.fft,* to test the correctness of the implemented function. An experiment was designed to generate simple and complex sinusoids, calculate the DFT and FFT of these generated signals, and compare the resulting spectra. Execution time of *focused_dft* and *numpy.fft.fft* was compared for short and long signals, both for signals whose lengths are powers of 2, which are typically suitable for FFT, and for slightly different lengths. Below is the description of each segment of the experiment. Section 4 provides the results.

4.1. Signal generation function

The signal generation function creates a sinusoid with specified frequency components, phase offsets, amplitudes, sample rate, and duration in seconds. The function signature is:

```
generateSignal(frequencyComponents, phaseOffsets, amplitudes,
                      sample_rate, duration)
```

The return value is an array of double values with *sample_rate* * *duration* elements. The function returns an array of double values with sample_rate * duration elements. It starts with an array of zeros and iterates through each frequency component, adding the appropriate value to the corresponding index based on the given parameters. Below is the Python implementation of the function:

Code Listing 1: Function for Signal Generation

```
def generateSignal(frequencyComponents, phaseOffsets, amplitudes,
                   sample_rate, duration):
  numSamples = sample_rate * duration
  signal = [0] * numSamples
  sampleX = [i/sample_rate for i in range(sample_rate * duration)]
  for i in range(0,len(frequencyComponents)):
    for j in range(0,numSamples):
      signal[j] += amplitudes[i]
           * math.sin(2*math.pi*frequencyComponents[i]*sampleX[j]
                    + phaseOffsets[i] )
  return signal
```

The function was tested by generating sinusoidal signals with integer and non-integer numbers of periods, featuring 1, 2, and 3 frequency components, with and without phase shifts. Table 1 lists all the parameters, and Figure 1 shows the generated signals. The sample rate is 10,000 samples/sec in all cases.

Table 1. Results for DFT and FFT

Signal	Duration (sec)	No. of periods	Frequency components	Amplitudes	Phases
I1	1	Integer	10	1	0
I2	1	Integer	10	1	π
I3	1	Integer	10, 50	5, 1	0, 0
I4	1	Integer	10, 50	5, 1	π, π /2
I5	1	Integer	5, 10, 50	1, 5, 2	0, π, π /2
N1	1.03	Non-integer	10	1	0
N2	1.03	Non-integer	10	1	π
N3	1.03	Non-integer	10, 50	5, 1	0, 0
N4	1.03	Non-integer	10, 50	5, 1	π, π /2
N5	1.03	Non-integer	5, 10, 50	1, 5, 2	0, π, π /2

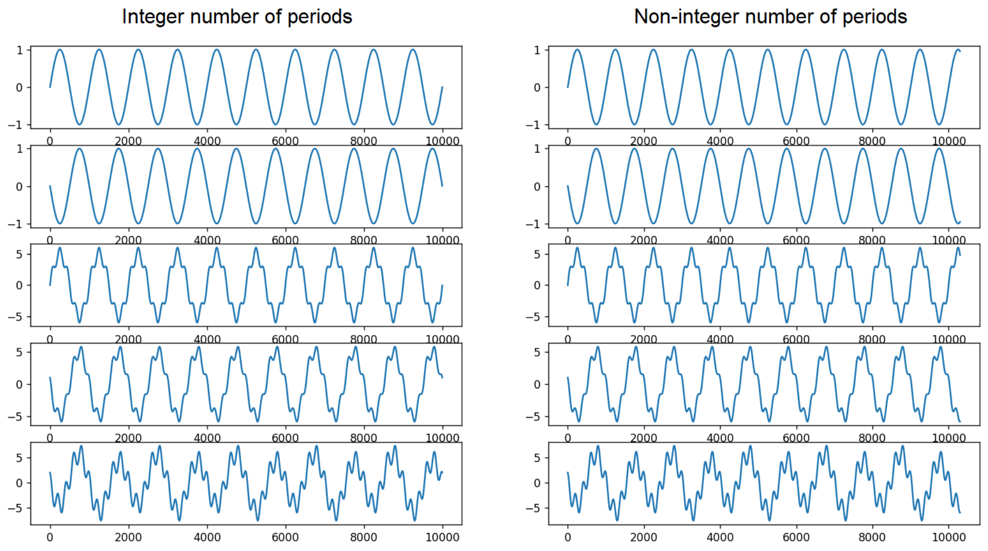

Figure 1. Output of generateSignal() function

4.2. DFT function

The experiment used two functions for computing Fourier Transforms: a custom DFT implementation and the standard FFT from the numpy package. The DFT function has the following signature:

```
focused_dft(signal, freq_from=None, freq_to=None, sample_rate=None,
            left_padding=0, right_padding=0)
```

The parameters include: the signal (an array of double values), an optional frequency range to calculate the DFT (to accelerate computation when interested in a specific spectral component), the sample rate (needed when frequency range is provided), and optional sample counts to add to the beginning or end of the signal (left and right padding). The return value is an array of double values representing the left half of the spectrum, as DFT generates a symmetric spectrum.

Figure 2 shows the spectra obtained from the FocusedDFT and FFT functions for generated signals noted in Table 1, zoomed in on the parts with frequency components. FocusedDFT was used without focusing on a specific frequency to obtain a full spectrum. It is evident that both FocusedDFT and FFT produce the same spectra, detecting the same frequency components as expected from the generated signals.

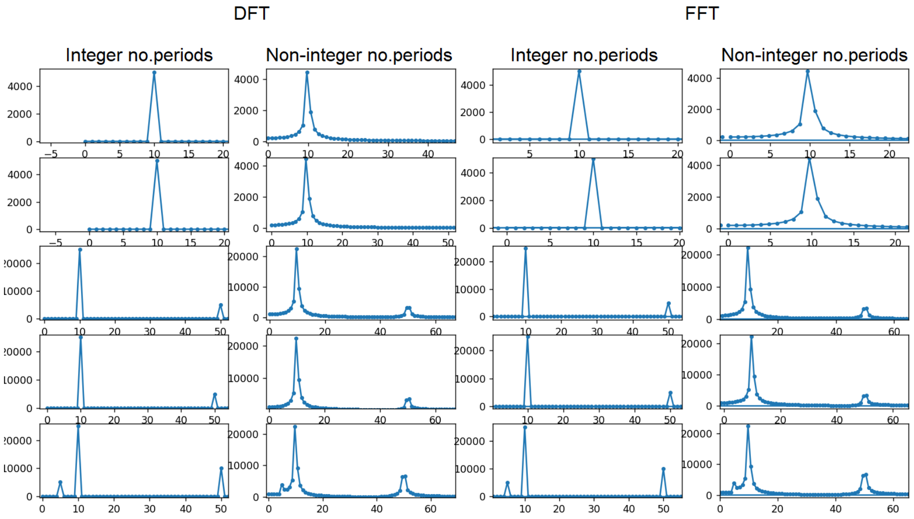

Figure 2. DFT and FFT spectra of generated signals

To test the FocusedDFT with focus on a specific range of frequencies, spectra of DFT and FocusedDFT for range [8.5, 11.5] Hz was calculated over the generated signals noted in Table 1. The resulting spectra, zoomed in on the parts where frequency components are present, are shown in Figure 3. It can be seen that FocusedDFT produces accurate spectra with only 10Hz component, ignoring the rest of the frequency bins, as intended.

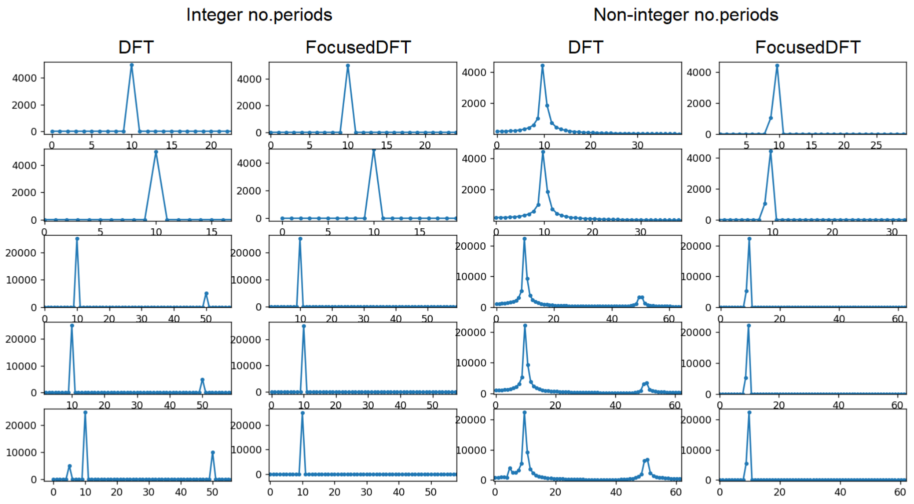

Figure 3. FocusedDFT and DFT spectra of generated signals

The DFT function implementation includes logic for signal padding and cropping based on the provided parameters. It then uses two nested loops to calculate the real and imaginary components, which are combined using Euclidean distance to produce the resulting spectrum, as derived in formula 8. The core of the code for the DFT function is shown in Code Listing 2.

Code listing 2. FocusedDFT function

```
def focused_dft(signal_original, freq_from=None, freq_to=None,
                sample_rate=None, left_padding=0, right_padding=0):
  # left and right signal padding and cropping ...
  # dft
  Xreal = numpy.zeros(N)
  Ximag = numpy.zeros(N)
  X = numpy.zeros(N)
  for k in range(k_range_from, k_range_to):
    for n in range(0, N):
      Xreal[k] = Xreal[k] + signal[n]*math.cos(k*n*2*math.pi/N)
      Ximag[k] = Ximag[k] - signal[n]*math.sin(k*n*2*math.pi/N)
    X[k] = math.sqrt( Xreal[k]**2 + Ximag[k]**2 )
  return X
```

To generate the frequency array (x-axis) for the frequency bins, a function called *frequencyBins* was implemented, which calculates these values based on the length of the spectrum array and the sample rate. The function is provided in Code Listing 3.

Code Listing 3: Frequency Bins Function

```
def frequencyBins(dft_spectrum, sample_rate):
  N = len(dft_spectrum)
  frequencyResolution = sample_rate / N
  frequencyBins = [ i * frequencyResolution for i in range(N) ]
  return frequencyBins
```

4.3. Execution time

To measure and compare the execution time, FocusedDFT and FFT were executed over 4 generated signals: a short signal, simulating the expected case in vibration-based damage detection; a long signal, for comprehensive measurement; a signal whose length is a power of 2, which is supposedly suitable for FFT; and a signal whose length is a slightly different from the previous one, which should be unsuitable for FFT but indifferent for FocusedDFT. The generated signals are listed in Table 2.

Table 2. Test signals

Signal	Sample rate	Length (sec)	Frequency components	Amplitudes	Phases
S1 - Short	1000	1	10, 50	5, 1	1, 2
S2 - Long	22000	1	10, 50	5, 1	1, 2
S3 - FFT suitable	1024	1	10, 50	5, 1	1, 2
S4 - FFT unsuitable	1027	1	10, 50	5, 1	1, 2

5. Results

In the experiment, 4 signals were generated, followed by applying the DFT over entire spectrum, FocusedDFT around the target component of 10Hz, and FFT. Experiment was run on Intel i5-1035G1 CPU, on Windows 10. To avoid possible increase in execution time due to randomly activated background processes, each test was executed 4 times. Measured execution times and calculated average times are listed in Table 3.

Table 3. Execution times in ms

DFT type	Signal	Test 1	Test 2	Test 3	Test 4	Rounded average
DFT	S1	380419.7	376956.7	370383.1	366535.3	373573
	S2	169263715.6	172438375.3	171855072.3	172453596.9	171502690
	S3	378081.9	380420.5	378003.3	377855.1	378590
	S4	379929.9	377849.8	387159.4	385697.9	382659
Focused DFT	S1	2950.9	3111.1	2905.2	2986.2	2988
	S2	56586.3	54741.4	59146.4	58865.2	57335
	S3	2561.9	3020.4	2321.4	2268	2543
	S4	3034.4	2442.3	3016.0	2762.7	2814
FFT	S1	188.2	142.4	149.7	143.9	156
	S2	2768.6	2229	2379.5	3081.8	2615
	S3	104.4	75	90.1	107.1	94
	S4	135.5	92.9	141.5	150.9	130

As expected, execution time for DFT is significantly longer then Focused DFT and FFT. In case of S3-signal with 1024 samples, which should be suitable for FFT, and S4-signal with 1027 samples, which should be unsuitable for FFT, it is visible that DFT execution time proportionally increases, which is not the case with FFT.

6. Discussion

The implemented DFT produces accurate results, aligning closely with the Fast Fourier Transform (FFT) in identifying frequency components, as depicted in figures 2 and 3. Table 3 illustrates that the Focused DFT is approximately 20 times slower than FFT for signals containing 1000 samples.

The primary application for the Focused DFT lies in vibration-based damage detection methods, which prioritize precision in identifying specific frequency components. These methods manipulate signal length to refine frequency binning, a process where FFT implementations often fall short due to inherent signal chunking. By adjusting the signal length to accommodate an integer number of periods of the targeted frequency, precision is optimized. For instance, for a 10Hz signal, adjustments up to 1/10th of the original length may be necessary, requiring several hundred iterations and potential DFT recalculations. However, experiments demonstrate that executing Focused DFT for a 1000-sample signal takes approximately 3ms, allowing for 300 iterations per second. With a total execution time of several seconds, the Focused DFT emerges as a viable solution in this context

7. Conclusion

Vibration-based damage detection is a popular non-intrusive approach which heavily relies on precise signal processing. DFT is an important part of this process. Small frequency changes are important and need to be captured. The discrete nature of computer systems and data acquisition poses challenges to precision, particularly regarding frequency bin resolution. While spectral interpolation may seem like a solution, in some specific cases it falls short in practice. Manipulating signal length to align with frequency bin resolution proves to be a successful strategy, albeit one that demands multiple DFT calculations across varying signal lengths. FFT implementations in this case fail to provide good results due to inherent signal chunking. Traditional DFT implementation is slow and total execution time can be problematic.

In the case of vibration-based damage detection FFT can be used for an initial processing of the acquired signal to get the quick insight of the spectra and spectral components present. The engineer can then focus on a specific frequency component and direct the algorithm to provide the precise frequency within that focused frequency range, by manipulating the signal length and frequency bin resolution. FocusedDFT execution time is acceptable, it doesn't suffer from long execution time like DFT does and doesn't interfere with signal length as FFT does. FocusedDFT can be used as a viable alternative in such methods.

Future research efforts could be directed to employ FocusedDFT in PyFEST and similar methods in order to improve their execution time without compromising the precision.

Acknowledgment. The author would like to acknowledge the significant contribution of prof. dr. Đorđe Babić, School of Computing, Union University, in providing guidance and support during the writing process. His expertise and encouragement have been truly appreciated.

References

1. X. Sun, S. Ilanko, Y. Mochida, R.C Tighe., A Review on Vibration-Based Damage Detection Methods for Civil Structures, *Vibration*, 6(4), Art. no. 4, Dec. 2023, doi: 10.3390/vibration6040051
2. D. Nedelcu, G.-R. Gillich, A structural health monitoring Python code to detect small changes in frequencies, *Mech. Syst. Signal Process.*, vol. 147, p. 107087, Jan. 2021, doi: 10.1016/j.ymssp.2020.107087.
3. G.-R. Gillich, N.N.N. Maia, I.C. Mituletu, *Problem of Detecting Damage Through Natural Frequency Changes*, in Computational and Experimental Methods in Structures, vol. 10, WORLD SCIENTIFIC (EUROPE), 2018, pp. 105–139. doi: 10.1142/9781786344977_0004.
4. C. Chioncel, N. Gillich, O. Tirian, J.L Ntakpe, Limits of the Discrete Fourier Transform in Exact Identifying of the Vibrations Frequency, *Romanian J. Acoust. Vib.*, 12(1), pp. 16–19, Jan. 2015.
5. L. Jin, L. Liang, A power-of-two FFT algorithm and structure for DRM receiver, *IEEE Trans. Consumer Electron.*, 56(4), pp. 2061–2066, Nov. 2010, doi: 10.1109/TCE.2010.5681072.

Address:

- Petar Prvulović, School of Computing, Union University, Kneza Mihaila 6/6, Belgrade, Serbia
  pprvulovic@raf.rs