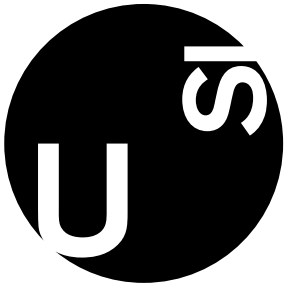


Università
della
Svizzera
italiana

Faculty
of
Informatics

Bachelor Thesis

June 24, 2023

Rotation of multi-dimensional signals with spectrally accurate schemes

Dylan Reid Ramelli

Abstract

Advisor
Prof Rolf Krause
Co-Advisors
Dr Diego Rossinelli, Dr Patrick Zulian

Advisor's approval (Prof Rolf Krause): Date:

Contents

1 Introduction, Motivation

1.1 Light transport

In scientific image visualization there is a need to work with great magnitudes of data. We want to perform operations at this enormous scale because there might be some natural properties that might otherwise escape our ability to sense them.To be able to perform calculations on this much information we make use of high performing hardware with well defined structures. To visualize 3D models we calculate the amount of light that gets to one place from another. While light-rays traverse a model they suffer from attenuation, which is the property of any material that dictates how easily it can be penetrated by light. This property carries over to other forms of energy such as sound or particles.
Since we deal with high amounts of data we must also use massive amounts of ray casting to light models correctly. To carry out these calculations as efficiently as possible we can rotate the model to facilitate the arrangement of data through which the light has to pass through. This facilitates many tasks such as Multi-Modal imaging of computerized tomography (CT) scans and Positron Emission Tomography (PET) scans. While the former provides detailed structural images, the latter measures molecular activity and we can use image rotation to combine the two to capture other aspects of the structure. Another task that needs accurate data transformations is Data Augmentation for Convoluted Neural Networks (CNN).
While 3D rotations make use of Euler angles to dictate the orientation of rotation, we can also express three dimensional rotations as a series of 2D rotations. This can be imagined as taking multiple slices of a model and rotating them by some degree. We can go even further by saying that a 2D image rotation can be expressed as three 1D signal translations in a cardinal direction[2]. This notion is very powerful since it enables the creation of 1D schemes that can be used on almost any kind of hardware structure.
The raw imaging data that is acquired provides us with multi-scale frequency components of the image. Our task is to manipulate the frequencies to obtain a higher fidelity rotation.

1.2 Manipulating frequency components of an image

Given that we can interpret 3D or 2D image rotation as a series of 1D translations we can define the scope of this project: we want to take a large scale signal and translate it by any real value, while maintaining a high level of accuracy and performance.
To achieve this objective, we make use of filter convolution and frequency component analysis in the Fourier domain. Our code will be efficient and minimalist in order to ensure high performance on any kind of hardware.

Figure 1. Large-scale in-silico investigation of homeostasis in the subarachnoid space of the human optic nerve [Rossinelli et al. 2023, in-preparation]. Homeostasis is directly related with the interaction between the cerebrospinal fluid flow (not shown here) and the meningeal surface. Blue denotes poor levels of homeostasis , whereas red denotes sufficient material exchange between fluid and structure.

[2]

2 Methodology

As stated in the introduction we will take advantage of the fact that a 2D rotation can be expressed as a 1D translation which we reference from this article [2]. In the case of the article the authors propose the use of different convolution-based interpolation schemes such as polynomial splines of degree n, cubic spline, etc. to translate a 1D signal. These operations are inherently global, which is not particularly effective since translation is a fundamentally local operation. In our case however we will perform local direct convolutions on the original signal with a filter, whose frequency components we manipulate in the frequency domain. To create our own convolution filter in the frequency domain we will start with the Discrete Time Fourier Transform of a cosine function in an interval $[-M, M]$. Defining the function in the Fourier space allows us to fine tune the filter to have a better control over the smoothing effect that will occur when we shift by fractional amounts. The filter is small compared to the original signal. Finally we will perform direct convolution of the signal with the computed filter.

2.1 2D rotation as 1D translation

As stated above, in the article that this project takes inspiration from, it is shown that the traditional rotation matrix

$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

can be factorized as three matrices each of which represents a shearing of the image in a cardinal direction.

$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} = \begin{pmatrix} 1 & -tan\theta/2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ sin\theta & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -tan\theta/2 \\ 0 & 1 \end{pmatrix} \tag{1}$$

The first matrix shears the image in the x direction by $\Delta_x = -y \cdot tan(\theta/2)$, the second matrix shears the image in the y direction by $\Delta_y = x \cdot sin(\theta)$ and the last matrix shears the image again in the x direction by Δ_x.

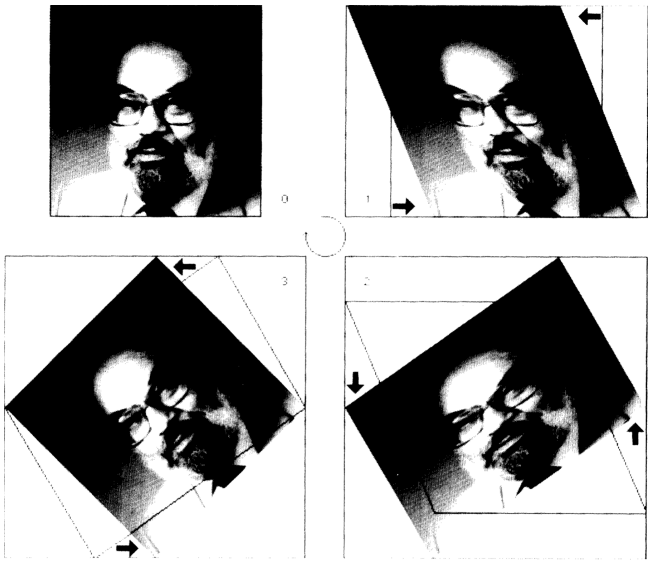


Figure 2. Rotation of an image done with 3 1D translations.
[2]

2.2 Fourier Transform

In signal processing working in the frequency domain has a great advantage since any function that is expressed in this way can be reconstructed completely via an inverse process, with no loss of information.[3] For a function $g(x)$ to be represented in the frequency domain we must then use of the Discrete Fourier Transform (DFT), which is defined as follows:

$$G_m = \sum_{k=0}^{N-1} g_k e^{-i\frac{2\pi}{N}km} \tag{2}$$

In this project the above equation will not be used since the filter that is going to be defined in one of the next chapters, will be created directly in the Fourier domain. Nevertheless it is useful to show the definition since it is the

Inverse Discrete Fourier Transform that will be used to reconstruct the filter in the time domain:

$$g_n = \frac{1}{N}\sum_{k=0}^{N-1} G_k e^{i\frac{2\pi}{N}kn} \tag{3}$$

2.3 Derivation of the shift interpolant

Here we derive the shift filter for a discrete signal starting from the normal Fourier Transform with f being our 1D input array, N the number of samples and m the frequency:

$$F_m = \sum_{k=0}^{N-1} f[k] e^{-i\frac{2\pi}{N}km}$$

Now instead of $f[k]$, we want $f[k-\delta]$ where δ is the amount we want to shift. So the above equation can be rewritten as:

$$Z_m = \sum_{k=0}^{N-1} f[k-\delta] e^{-i\frac{2\pi}{N}km}$$

$$Z_m = \sum_{r=0-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}km}, r = k-\delta$$

Since $r = k-\delta$ then $k = r+\delta$, and as such:

$$Z_m = \sum_{r=-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}(r+\delta)m}$$

We can then separate the exponential:

$$Z_m = \sum_{r=-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}rm} e^{-i\frac{2\pi}{N}\delta m}$$

And factor it out of the sum:

$$Z_m = e^{-i\frac{2\pi}{N}\delta m} \sum_{r=-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}rm}$$

$$Z_m = e^{-i\frac{2\pi}{N}\delta m} \sum_{k=0}^{N-1} f[k] e^{-i\frac{2\pi}{N}km}$$

$$Z_m = e^{-i\frac{2\pi}{N}\delta m} F_m = H_m \cdot F_m \tag{4}$$

2.4 Interpretation of frequencies for fractional shifts

The final equation from the previous section 4 works well for any integer shifts that we want to perform on our signal but for any fractional amount we encounter some problems with the use of the correct frequency indexes.

As stated in equation 4 we multiply each sample by its corresponding phase, which is based on the frequency number m. We need to take into consideration the negative frequencies for fractional shifts and as such we can define a function called wavenum that returns the correct frequency index to use:

$$wave\_n(m) = (m + \lfloor N/2 \rfloor) \bmod N - \lfloor N/2 \rfloor$$

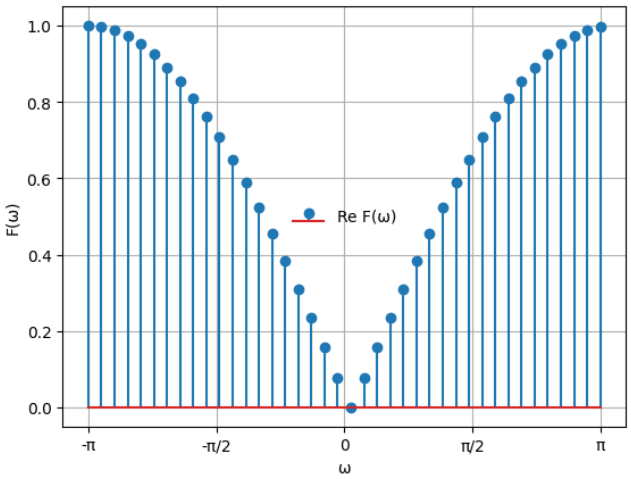


Figure 3. Shift filter of size $n = 50$, with a fractional shift of $\delta = 0.5$

Modify original equation to use wavenum.

$$H(\omega) = e^{-i\frac{2\pi}{N}\delta wave\_n(m)}$$

$$Z_m = e^{-i\frac{2\pi}{N}\delta m} F_m \tag{5}$$

2.5 Smoothing Filter (Low-Pass)

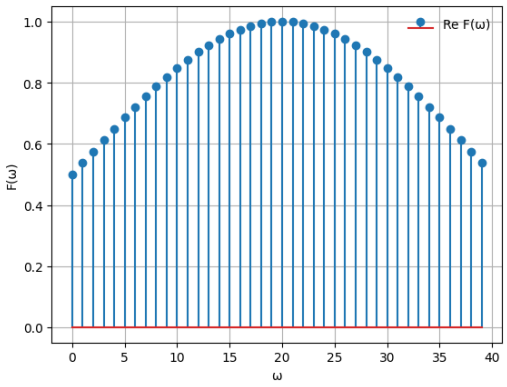
The smoothing filter is what we will use to alleviate the oscillations that occur with fractional shifts. To start off we define the filter in the Frequency domain with the use of the Continuous Fourier transform (CFT)on a cosine function.

$$F(\omega) = \int_{-M}^{M} \cos(\frac{\pi}{M}t)\mathrm{e}^{-\mathrm{i}\omega t}\mathrm{d}t \tag{6}$$

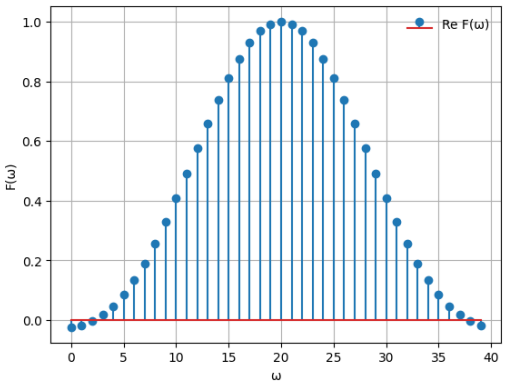
This equation allows us to define our CFT in an interval M, that we choose based on the fractional amount we want to shift. The interval M is useful to us to fine tune the filter and choose how big of a smoothing effect we want.

$$F(\omega) = \frac{2M^2\omega\sin(M\omega)}{M^2\omega^2 - \pi^2} \tag{7}$$

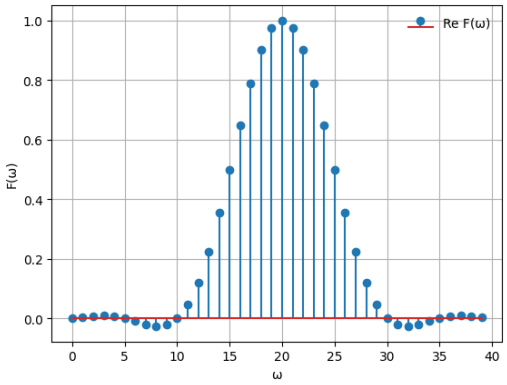
Using this function we can build the following filters for each M:



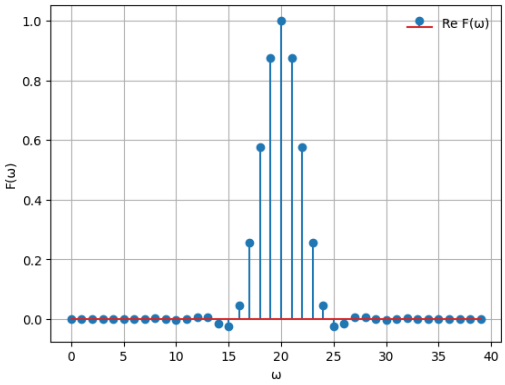
Filter with M = 1.



Filter with M = 1.5.



Filter with M = 2.



Filter with M = 3.

Figure 4. Compact schemes of different M for local computation (DTFT).

For better accuracy we generally choose our interval M to be small when we have a small fractional shift and big when we have a shift that gets closer to 0.5. We can write a simple expression that defines M in respect to the fractional shift x to be:

$$M = \begin{cases} 1.5 & if\, 0.1 < x < 0.3 \\ 2 & if\, 0.3 < x < 0.5 \end{cases}$$

With some experimenting we found that the filter with $M = 3/2$ is a good compromise between smoothing the values and keeping them as accurate as possible.

2.6 Creating a Filter in the Frequency domain

In this section we propose an example of the smoothing filter. We will create one of size 40 with $M = 2$. We then follow equation 7, while also paying attention to evaluating the limit when $\omega = \frac{\pi}{M}$ or $\omega = -\frac{\pi}{M}$. This gives us:

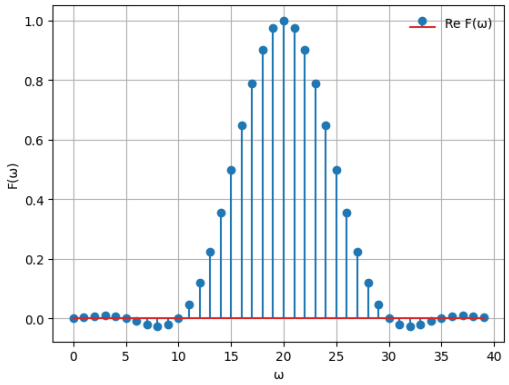


Figure 5. Filter with $M = 2$

Now let us say that the fractional amount to shift is 0.25. We then multiply our smoothing filter $F(\omega)$ by our shift filter $H(\omega)$.

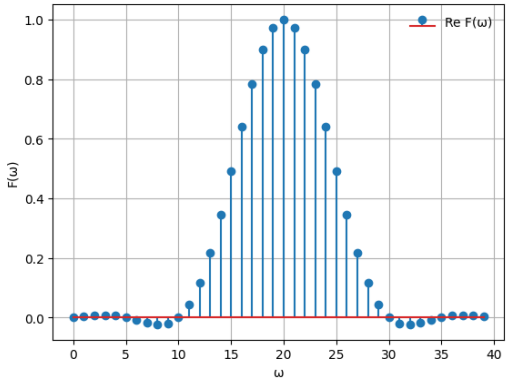


Figure 6. Filter with $M = 2$ after shift of 0.25

Finally we do an Inverse Fourier Transform to acquire our filter in the time domain.

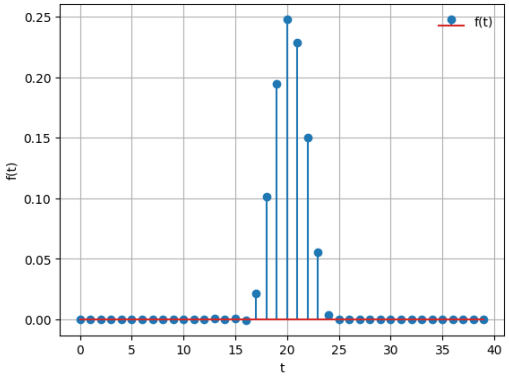


Figure 7. Filter with $M = 2$ in time domain.

3 Implementation and Project Design

We chose a bottom-up approach for this project since we saw that the implementation steps were well suited for it. We started with the fundamentals of signal processing by exploring the Fourier Transform and its properties to build our shift filter. We used Jupyter Notebook extensively to test our method and quickly plot the results, while also documenting everything so that it would be easier to complete this report. Since we used the bottom-up approach we were able to re-use parts of the code to test some other aspect of the shift. We started with simple shifts of a 1D signal by an integer values then we tried to shift the same signal by a fractional amount. This is where some problems arose with the usage of the correct frequency index but in code we easily implemented it with a basic method. The source code of this project is available at: `https://github.com/DylanReidRamelli/Bachelor-Project-2023`. The tools that were used: MakeFile, CMake, Jupyter Notebook and some built-in libraries of C and Python like numpy and matplotlib.

3.1 Libraries

Numpy and Matplotlib were a huge help in visualizing the results of the Fourier Transform and of the shifts. These libraries were also useful for when we started writing code in C. Where we would output the binary result to a file and then read it and plot the result in python. There are a few plotting libraries for C like the gnuplot library but we felt that this was just the faster/easier solution. The whole project started out as a CMake and this was chosen mainly because I was already familiar with this tool and was able to quickly setup an environment that would build the executable. However towards the end we decided to just use a plain Makefile to keep everything nice and simple. The added advantage of the makefile was to be able to easily run a python script at compile time to create other files needed by the main target.

3.2 Efficient implementation of direct convolution on x86 microarchitectures

Once the filter creation is done we must retrieve the signal in the time domain and to do this we implemented the "slow" version of the Inverse Fourier Transform since the size of the operation will always be much smaller than the original data. To keep our one dimensional direct convolution as simple and as efficient as possible we analysed it using a Web tool called "Compiler Explorer"[1]. This tool allows us to see which assembly code is being used to perform the convolution operation and choose which implementation should run faster on certain compilers and CPU's.

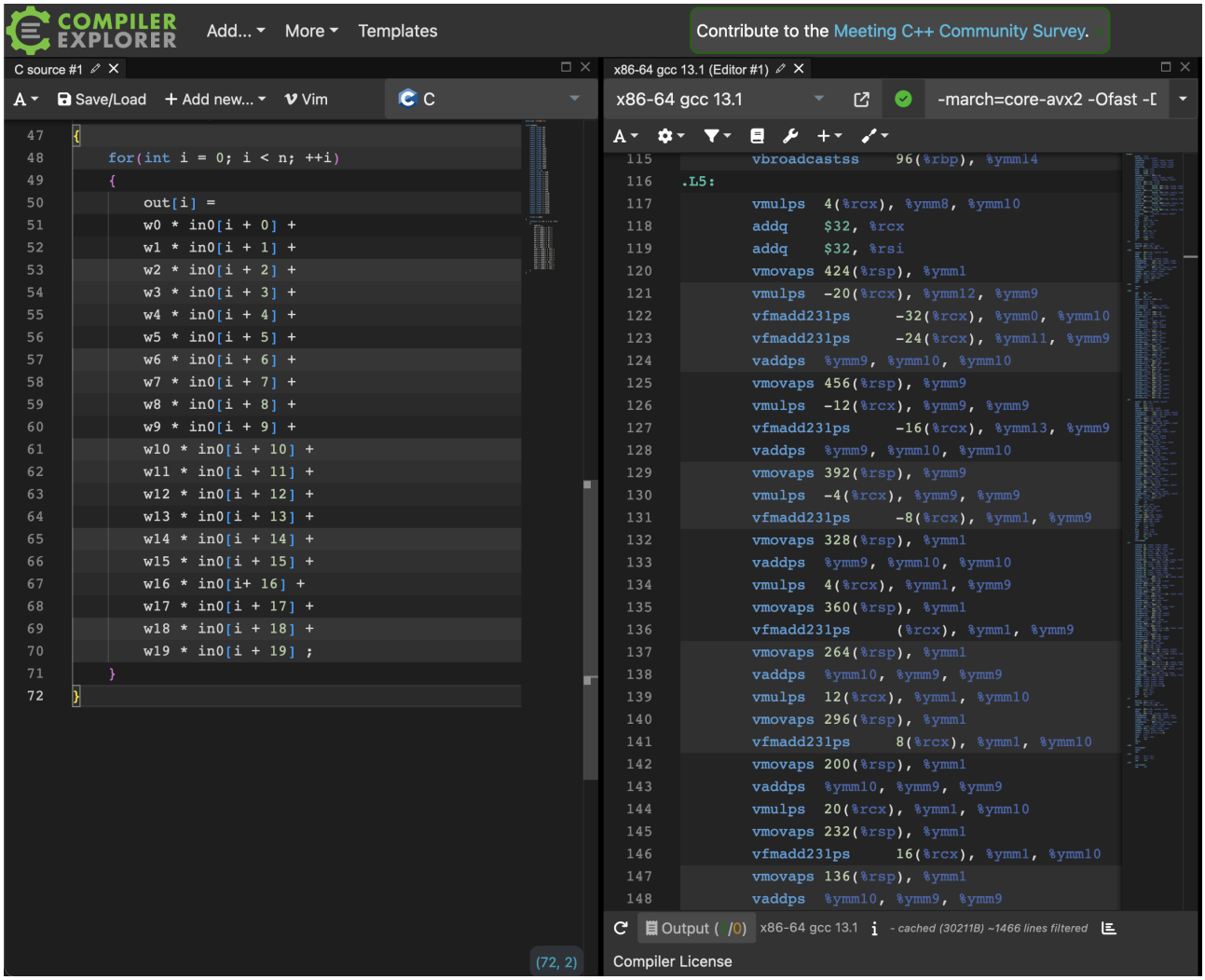


Figure 8. Convolution function as seen in Compiler Explorer

The code that we came up with is easily written by a python script that is called at compile time by the make file, the only parameter we define is the size M that we want to use and the script will generate a header and source file in C that will be used for the convolution.

3.3 Integer Shift

Now that the fractional part of the shift has been taken care of we need to shift the signal by the integer value. First of all it should be clarified that if the shift does not have a fractional part then we skip the convolution with the filter entirely. To shift a signal by an integer amount we used the C function *memcpy* to copy the values of the original array to a resulting array but at a different position, which is defined by the integer shift. We chose *memcpy* instead of just moving the values inside the same array with *memmove* since *memcpy* is designed to be the fastest library routine to copy data from memory to memory.

4 Results

4.1 Results with 1D signals.

One dimensional signals are the starting point of our tests and as such we chose a pretty aggressive signal to test out. In this example we are using a starting signal of binary value of length $n = 100$ samples and it is defined as:

$$f(x) = \begin{cases} 1 & \text{if } x \leq \frac{n}{2} \\ 0 & \text{otherwise} \end{cases}$$

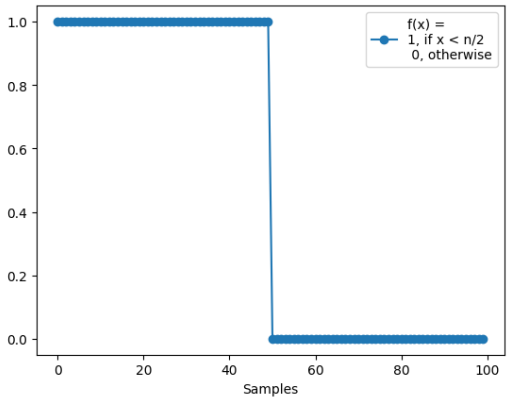
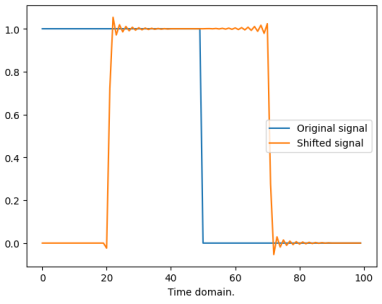
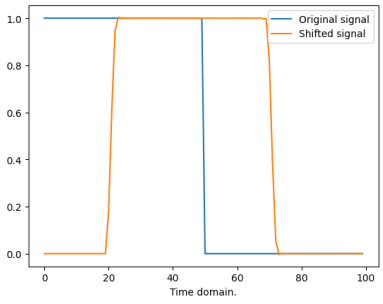


Figure 9. Original signal of size $n = 100$ samples.

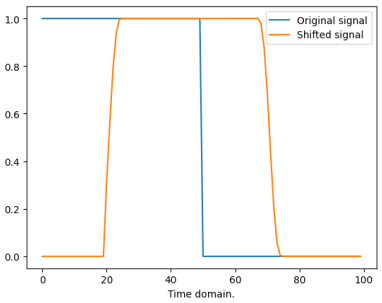
By taking this signal and shifting it by 20.25 we get a different result depending on the M chosen when creating the filter.



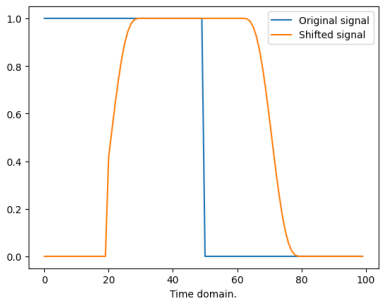
Filter with M = 1.



Filter with M = 3/2



Filter with M = 2.



Filter with M = 3.

Figure 10. Shift by 20.25

At first glance we can see that with $M = 1$ we have too many oscillations present in the resulting signal and with $M = 3$ the signal is smoothed out too much. We can see though in figure 11 that if we set $M = 1$ and the fractional value is small, for example 0.01 we have less oscillations and the signal is more similar to the original.

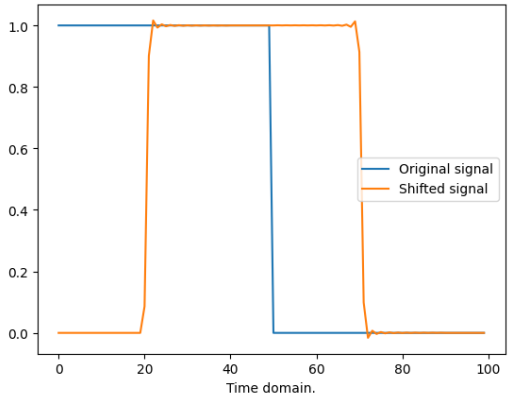
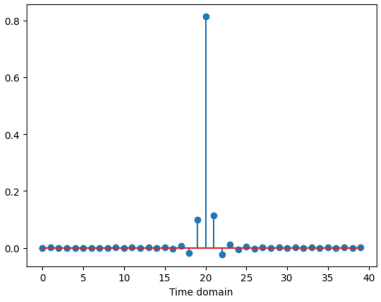
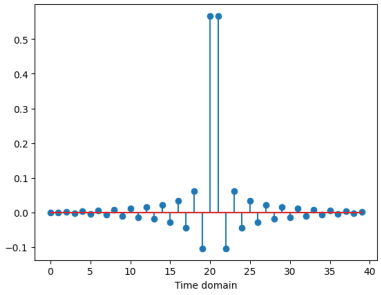


Figure 11. Shift by 20.01 with $M = 1$

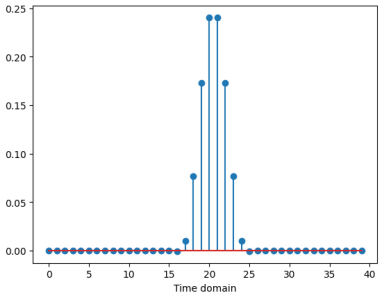
This result is attributed to the fact that the filter contains fewer oscillations as well. In fact if we take a look at the filter defined for $M = 1$ we can see that we have maximum oscillations exactly at 0.5. This is why for a fractional shift that gets closer to 0.5 it is necessary to use a bigger value for M such as $\frac{3}{2}$ or 2 to smooth out the oscillations.



Filter shifted by 0.01 with $M = 1$



Filter shifted by 0.5 with $M = 1$



Filter shifted by 0.5 with $M = 2$

Figure 12. Difference in oscillations with varying fractional shifts.

5 Conclusion

One way to improve upon this work would be to implement the code with CUDA and a function that chooses which M to use based on the fractional shift and the error.

References

[1] Matt Godbolt.

[2] IEEE Philippe Thévenaz Michel Unser, Senior Member and Leonid Yaroslavsky. Convolution-based interpolation for fast, high-quality rotation of images.

[3] Richard E. Woods Rafael C. Gonzalez. *Digital Image Processing*.