2259880C
Sarah Clark

DAMP 3 Programming Task

Using an FIR, create a low-pass filter with a cut-off at a frequency read from the command line. Normalise the frequency so that 0→ 0Hz and 1→ fs/2.
Plot the frequency response of your filter in gnuplot using a peak-following line. Compare the filter response with and without the use of a raised-cosine window function.

Code:

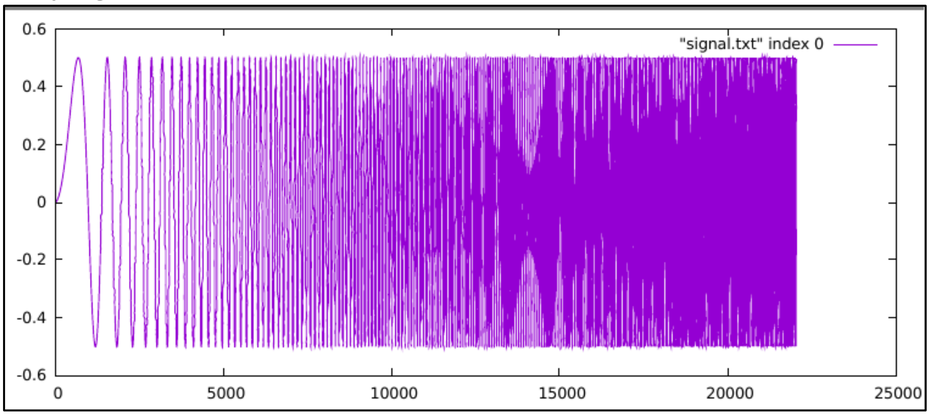
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


//There are two arguments: the first is the name of the program.
//The second can be inputted when the program is run. Here it is
//then assigned to be the cut-off frequency
int main(int argc, char *argv[])
{
              if (argc != 2) {
                     fprintf(stderr, "Usage: %s <Cut-off frequency> \n", argv[0]);
                     exit(1);
              }

              //Set up constants and variables
              const double cutf = atof(argv[1]);       //Cut-off frequency
              const double startf = 1;                 //Start frequency of chirp
              const double endf = 1000;                //End frequency of chirp
              const double duration = 0.5;             //Duration of chirp sweep
              const double amp = 0.5;                  //Amplitude
              const int srate = 22050;                 //Sample rate
              const double twopi = 2.0*M_PI;           //Two x pi
              const double calc = (endf-startf)/(2*duration); //Calculate here to save
                                                              //having to repeatedly
                                                              //calculate this
              const double dt = duration/(double)srate;       //Delta time
              double t=0;                                     //Time
              int klen = 1024;                                //Kernal length
              float sum =0;

              //Set up arrays to store signal data
              double signal [srate];      //Chirp signal array
              double kernal[1024];        //Convolution kernal
              double convolve[srate];     //Array for convolution result

              for (int i=0; i < srate; i++)
                     {
                     //Calculate chirp using this equation
                     signal[i] = amp*sin(twopi*((startf*t)+(calc*t*t)));
                     t=t+dt;
                 }


              //Calculate convolution kernal using these equations
              for (int i=0; i < 1024; i++)
                     {
                     if (i-(klen/2)==0){                 //Calculate sinc function
                            kernal[i] = twopi*cutf;      //Sinc of 0 is impossible so
                                                         //set equal to 2pi*cut-off
                                                         //frequency
```

```
				}
			else	{
				kernal[i] = sin(twopi*cutf*(i-(klen/2)))/(i-(klen/2));
				}
			//Construct window function
			kernal[i]= kernal[i]*(0.54- 0.46*cos(twopi*i/(float)klen));
			}


		//Normalise
		for (int i=0; i<1024; i++)
		{
			sum = sum + kernal[i];
		}

		for (int i=0; i<klen; i++)
		{
			kernal[i] = kernal[i]/sum;
		}


		//Convole signal array with kernal array
		for (int i=klen; i<srate; i++)
		{
			for (int j=0; j<klen; j++){
				//Increase array number of kernal by one and reduce
				//array number of signal by one using nested loop
				//First 1024 values of array set to 0
				convolve[i]=convolve[i]+signal[i-j]*kernal[j];
				}

		}

		//Code to plot Maxima points as given to us
		typedef struct Maximum {
			int t;
			struct Maximum *next;
		}	Maximum;
		Maximum *list = NULL, *list_back = NULL;

		for (int i=1; i < 15000-1; i++){
			if (convolve[i-1] < convolve[i] && convolve[i] > convolve[i+1]) {
				Maximum *newMax = (Maximum*)malloc(sizeof(Maximum));
				if (!list)
					list = newMax;
				if (list_back)
					list_back-> next = newMax;
				newMax->t = i;
				newMax->next = NULL;
				list_back = newMax;
				}
			}



		//Print results of convolution array
		//Whole result could not be printed as gnuplot crashed
		//Therefore loop was reduced from 22050 (sample rate) to 15000 samples

	for (int i=klen; i<15000; i++)
```

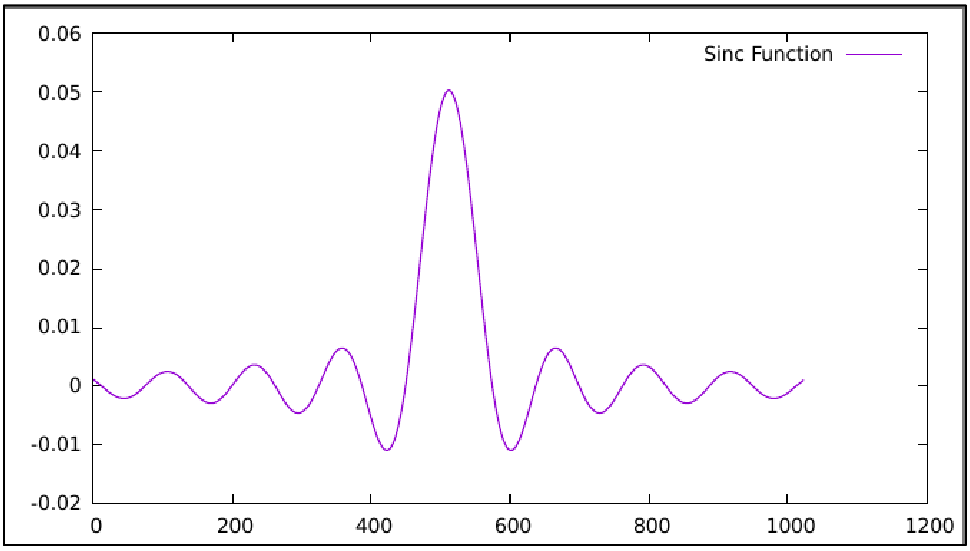
```
            {
                    printf("%i\t%f\n", i, convolve[i]);
            }

            printf("\n\n");

            while(list) {                //Write out the maxima as a seperate data set
                    Maximum* m = list;
                    printf("%i\t%f\n", m->t, convolve[m->t]);
                    list = m->next;
                    free(m);
                    }
}
```

Results: (using the 0.008 as the cut-off value)

Chirp Signal

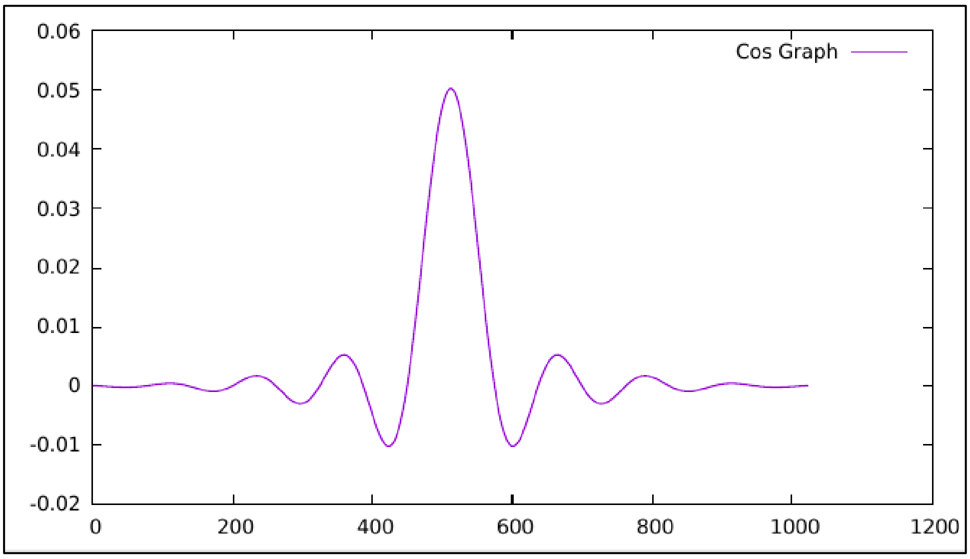


Sinc Function

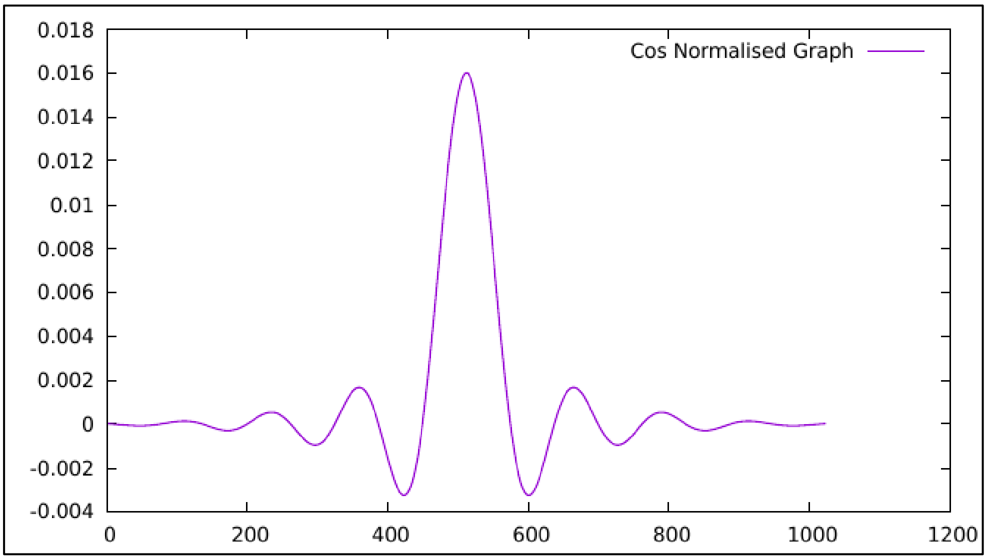


2259880C
Sarah Clark

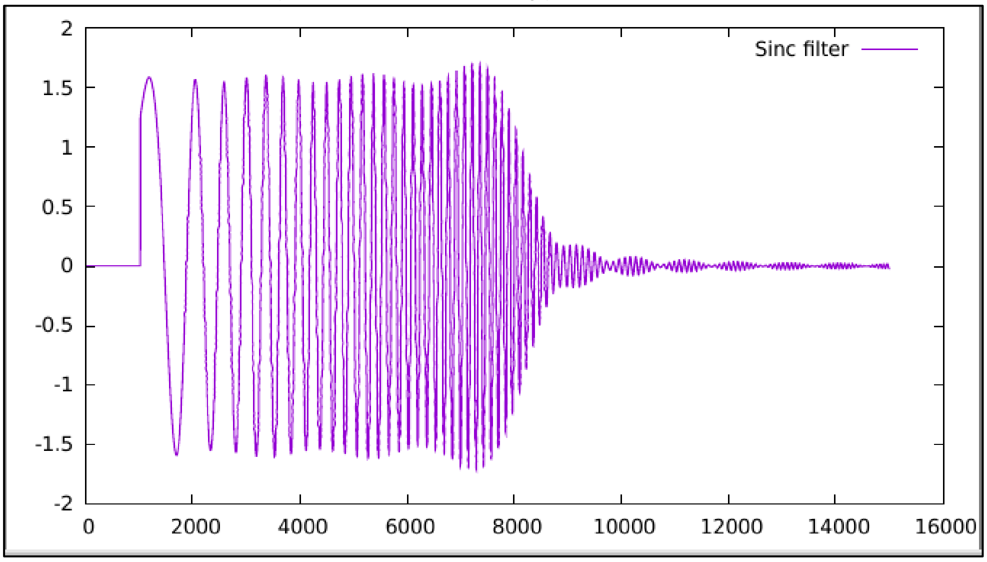
Sinc Function with Cos Window



Sinc Function with Cos Window Normalised

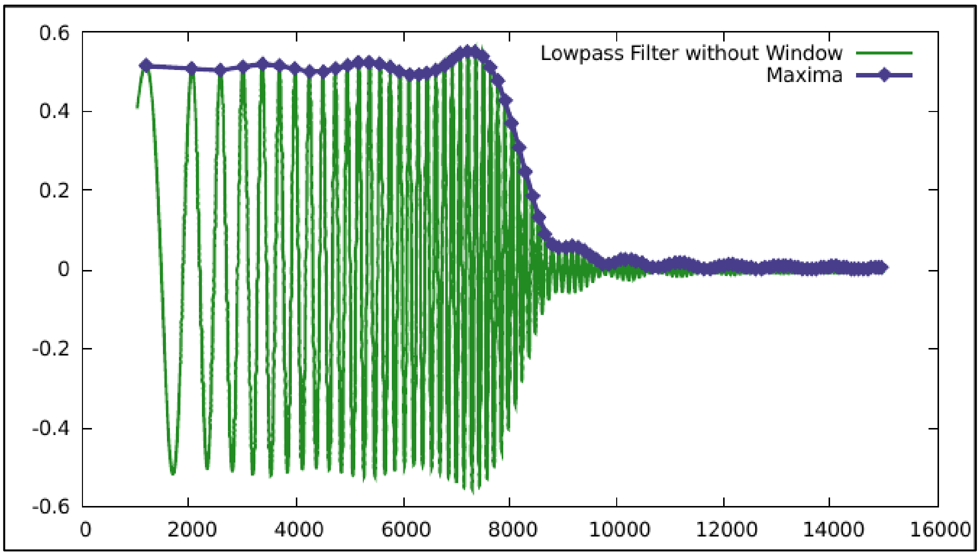


Normalised Sinc Filter (Convolved with Chirp)

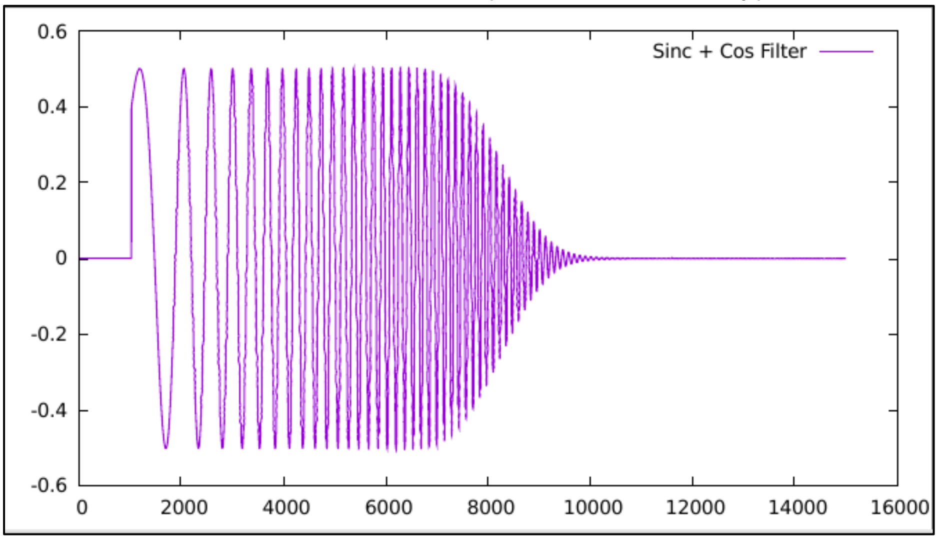


2259880C
Sarah Clark

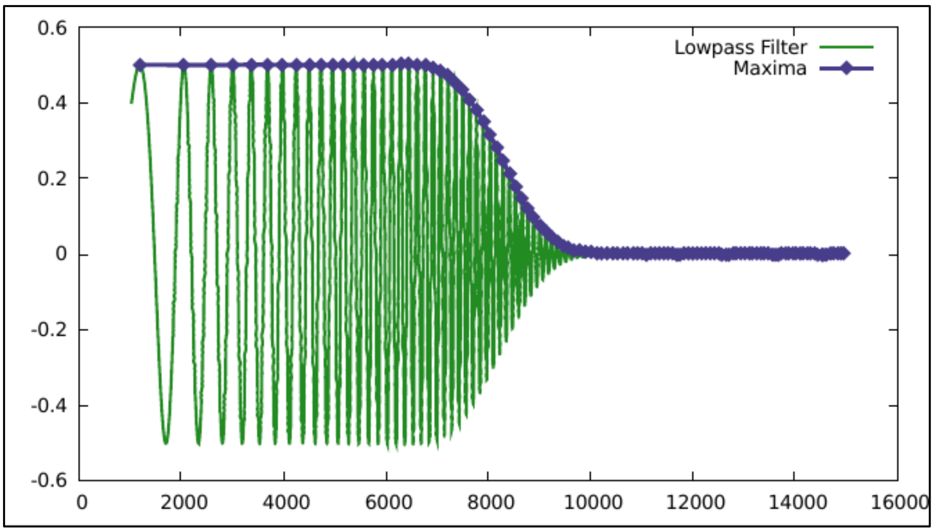
Normalised Sinc Filter with Maxima Points



Normalised Sinc and Cos Window Filter (Convolved with Chirp)



Normalised Sinc and Cos Window Filter with Maxima Points



2259880C
Sarah Clark

Normalised Sinc and Cos Window Filter (Convolved with Chirp)

Here 0.012 was used as the cut-off value instead of 0.008, to demonstrate that the filter works for different cut-off frequencies.