The Fast Fourier Transform (FFT) and MATLAB Examples

Learning Objectives

Discrete Fourier transforms (DFTs) and their relationship to the Fourier transforms

Implementation issues with the DFT via the FFT

- sampling issues (Nyquist criterion)
- resolution in the frequency domain (zero padding)
- neglect of negative frequency components

$$V(f) = \int_{-\infty}^{+\infty} v(t) \exp(2\pi i f t) dt$$

$$v(t) = \int_{-\infty}^{+\infty} V(f) \exp(-2\pi i f t) df$$

These Fourier integral pairs normally are performed numerically by sampling the time and frequency domain functions at discrete values of time and frequency and then using the discrete Fourier transforms to relate the sampled values

Fast Fourier Transform

Discrete Fourier Transform

$$V_p(f_n) = \frac{T}{N} \sum_{j=0}^{N-1} v_p(t_j) \exp(2\pi i\, jn / N)$$

$$v_p(t_k) = \frac{1}{T} \sum_{n=0}^{N-1} V_p(f_n) \exp(-2\pi i kn / N)$$

$T = N\Delta t$ … time window sampled with N points

Δt … sampling time interval

$t_j = j\Delta t \qquad f_n = n\Delta f = n / N\Delta t$

Fast Fourier Transform

As with the Fourier transforms, there are different choices made for the Discrete Fourier transform pairs. In general, we could write:

$$V_p(f_n) = n_1 \sum_{j=0}^{N-1} v_p(t_j) \exp(\pm 2\pi i\, jn/N)$$

$$v_p(t_k) = n_2 \sum_{n=0}^{N-1} V_p(f_n) \exp(\mp 2\pi i kn/N)$$

as long as $n_1 n_2 = \frac{1}{N}$

The indexing could also go from 1 to N instead of 0 to N-*1*

Fast Fourier Transform

$$V_p(f_n) = \frac{T}{N}\sum_{j=0}^{N-1} v_p(t_j)\exp(2\pi i\, jn / N)$$

$$v_p(t_k) = \frac{1}{T}\sum_{n=0}^{N-1} V_p(f_n)\exp(-2\pi ikn / N)$$

These discrete Fourier Transforms can be implemented rapidly with the Fast Fourier Transform (FFT) algorithm

N	$(N-1)^2$	$(N/2)\log_2 N$
256	65,025	1,024
1,024	1,046,529	5,120
4,096	16,769,025	24,576

FFTs are most efficient if the number of samples, N, is a power of 2. Some FFT software implementations <u>require</u> this.

Fast Fourier Transform

Mathematica

Fourier[{a_1,a_2,…,a_N}]

↕ lists

InverseFourier[{b_1,b_2,…,b_N}]

$$\frac{1}{\sqrt{N}}\sum_{r=1}^{N} a_r \exp\left[2\pi i(r-1)(s-1)/N\right]$$

$$\frac{1}{\sqrt{N}}\sum_{s=1}^{N} b_s \exp\left[-2\pi i(r-1)(s-1)/N\right]$$

Maple

FFT(N, x_{re}, x_{im})

↕ arrays

iFFT(N,X_{re},X_{im})

$$\sum_{j=0}^{N-1} x(j)\exp\left[-2\pi i\, jk/N\right]$$

$$\frac{1}{N}\sum_{k=0}^{N-1} X(k)\exp\left[2\pi i\, jk/N\right]$$

$$N = 2^n$$

MATLAB

fft(x)

↕ arrays

ifft(X)

$$\sum_{j=1}^{N} x(j)\exp\left[-2\pi i(j-1)(k-1)/N\right]$$

$$\frac{1}{N}\sum_{j=1}^{N} X(j)\exp\left[2\pi i(j-1)(k-1)/N\right]$$

Fast Fourier Transform

FFT function

```
function y = FourierT(x, dt)
% FourierT(x,dt) computes forward FFT of x with sampling time interval dt
% FourierT approximates the Fourier transform  where the integrand of the
% transform is x*exp(2*pi*i*f*t)
% For NDE applications the frequency components are normally in MHz,
% dt in microseconds
[nr, nc] = size(x);
if nr == 1
   N = nc;
else
   N = nr;
end
y = N*dt*ifft(x);
```

Fast Fourier Transform

inverse FFT function

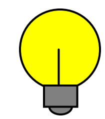
```
function y = IFourierT(x, dt)
% IFourierT(x,dt) computes the inverse FFT of x, for a sampling time interval dt
% IFourierT assumes the integrand of the inverse transform is given by
% x*exp(-2*pi*i*f*t)
% The first half of the sampled values of x are the spectral components for
% positive frequencies ranging from 0 to the Nyquist frequency 1/(2*dt)
% The second half of the sampled values are the spectral components for
% the corresponding negative frequencies. If these negative frequency
% values are set equal to zero then to recover the inverse FFT of x we must
% replace x(1) by x(1)/2 and then compute 2*real(IFourierT(x,dt))

[nr,nc] = size(x);
if nr == 1
        N = nc;
else
        N = nr;
end
y =(1/(N*dt))*fft(x);
```

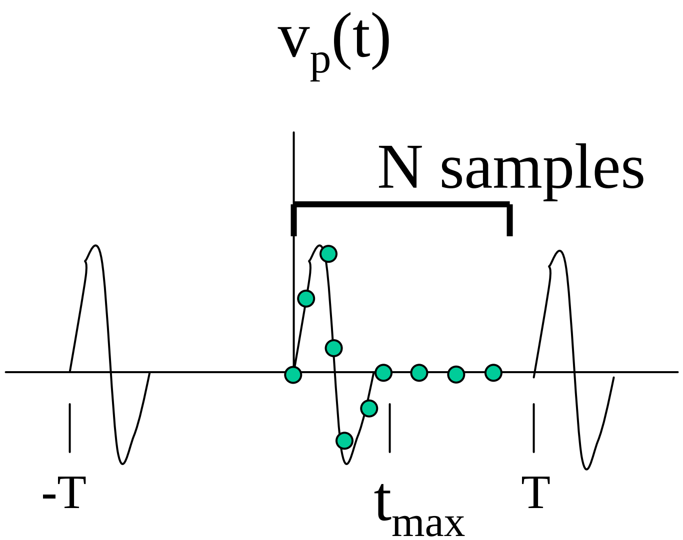
Fast Fourier Transform

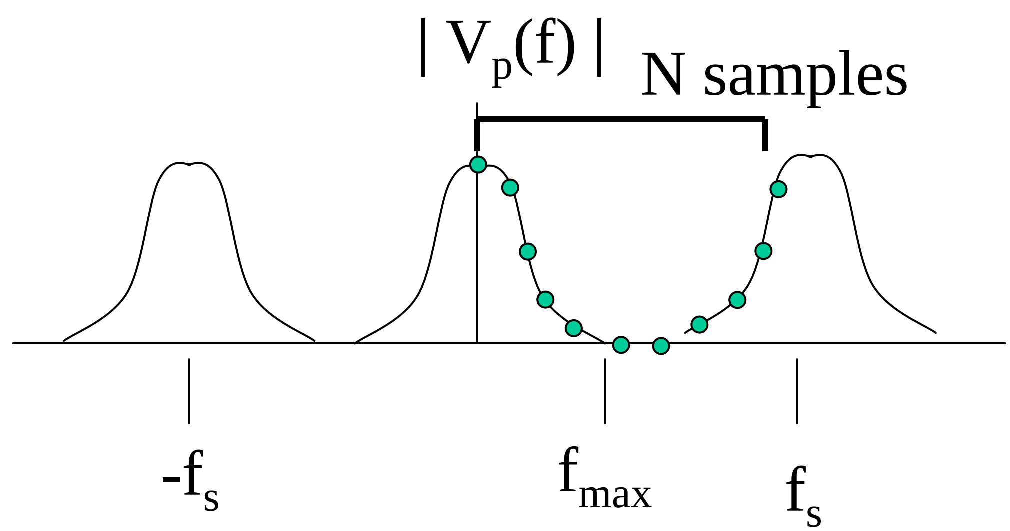
v_p and V_p in the discrete Fourier transforms are periodic functions. How do we guarantee these represent non-periodic pulses?

$$v_p(t) \cong v(t) \qquad \text{if} \qquad t_{\max} \le T = N\Delta t$$

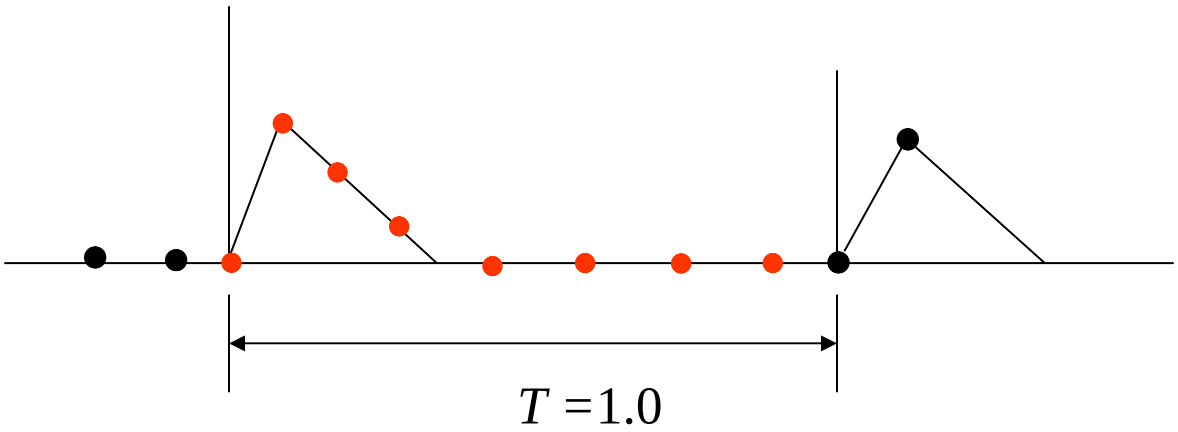
$$V_p(f) \cong V(f) \qquad f_s \equiv \frac{1}{\Delta t} \ge 2 f_{\max} \quad \text{Nyquist criterion}$$







The use of linspace and s_space functions for FFTs and inverse FFTs



$$\Delta t = T / N$$

In the example above $N = 8$, $T = 1.0$ so $\Delta t = 1/8 = 0.125$

```
>> t = linspace(0,1, 8);

>> dt = t(2) -t(1)

dt =    0.1429
```

To get the proper sampled values and the correct Δt we can use the alternate function s_space (it's on the ftp site)

```
>> t = s_space(0,1, 8);
>> dt = t(2)-t(1)
dt = 0.1250
```

```
>> t = linspace(0,1, 512);
>> dt = t(2) - t(1)
dt = 0.0020
```

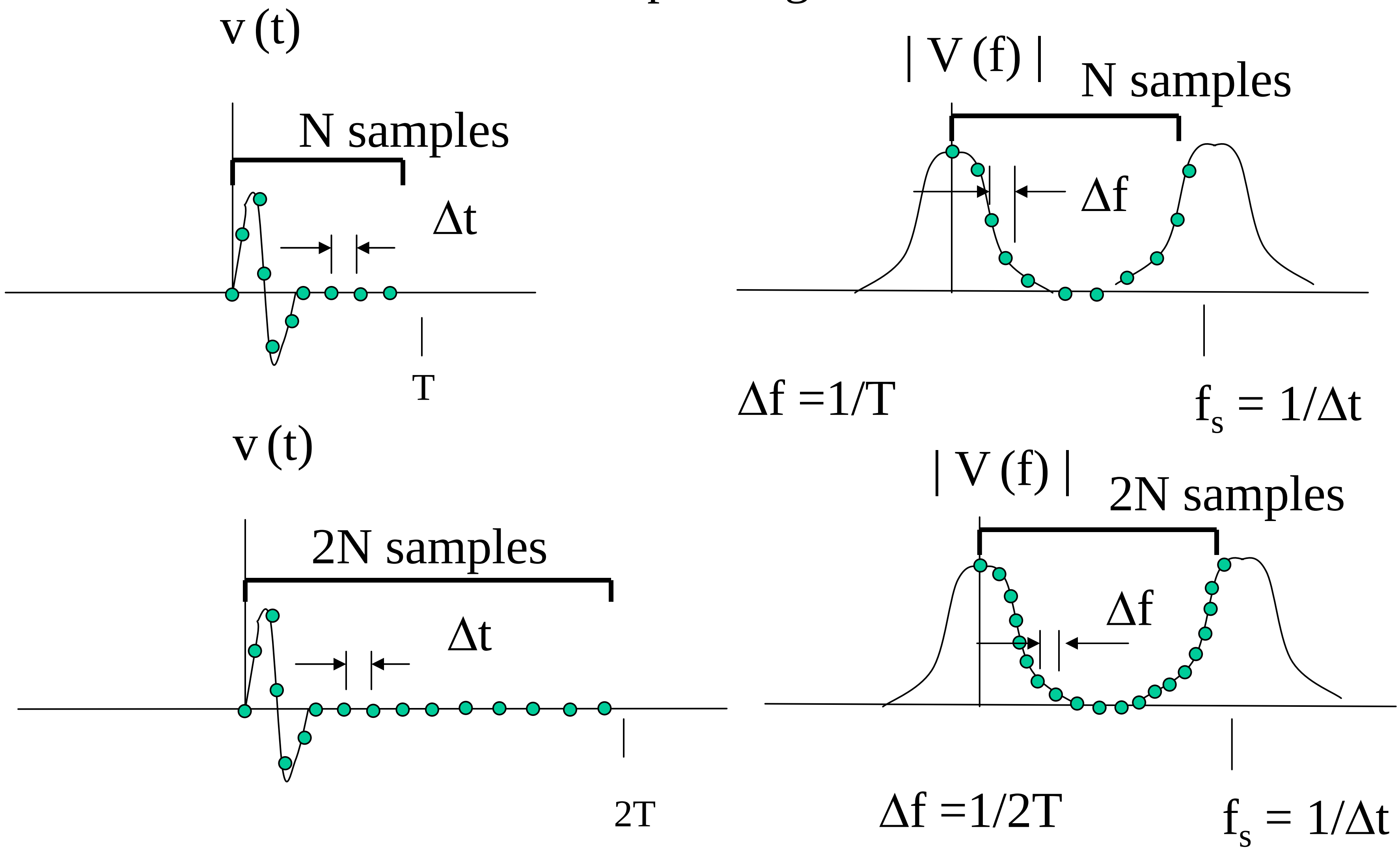
```
>> t = s_space(0,1, 512);
>> dt = t(2) - t(1)
dt = 0.0020
```

When you are using many points the difference is not large if we use either linspace or s_space but the difference is still there

```
>> format   long
>> t = linspace(0, 1, 500);
>> dt = t(2) -t(1)

dt = 0.00200400801603

>> t = s_space(0, 1, 500);
>> dt = t(2) -t(1)

dt = 0.00200000000000
```

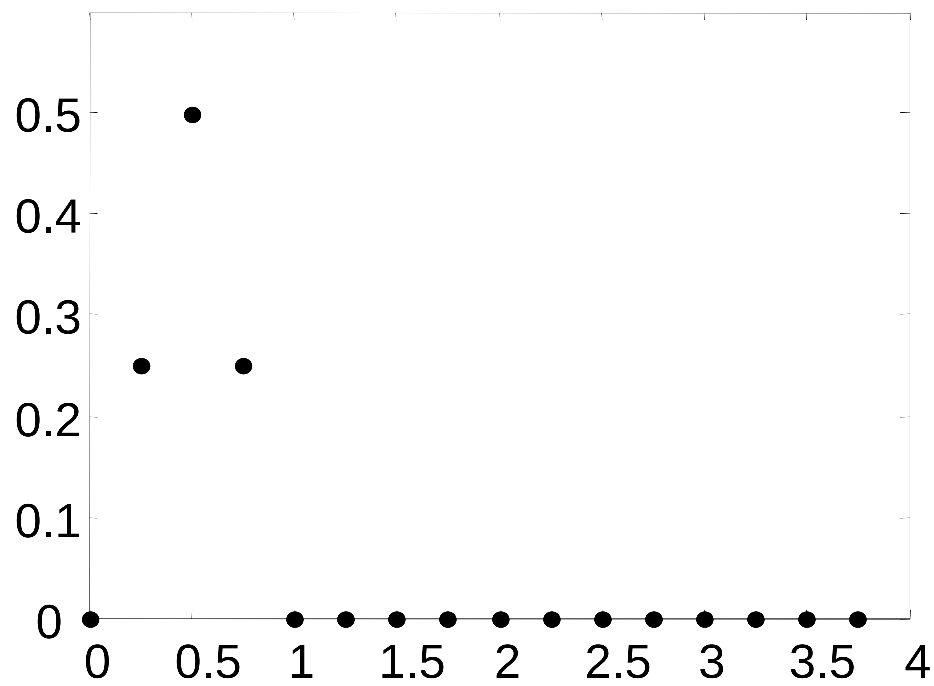
Fast Fourier Transform

zero "padding"

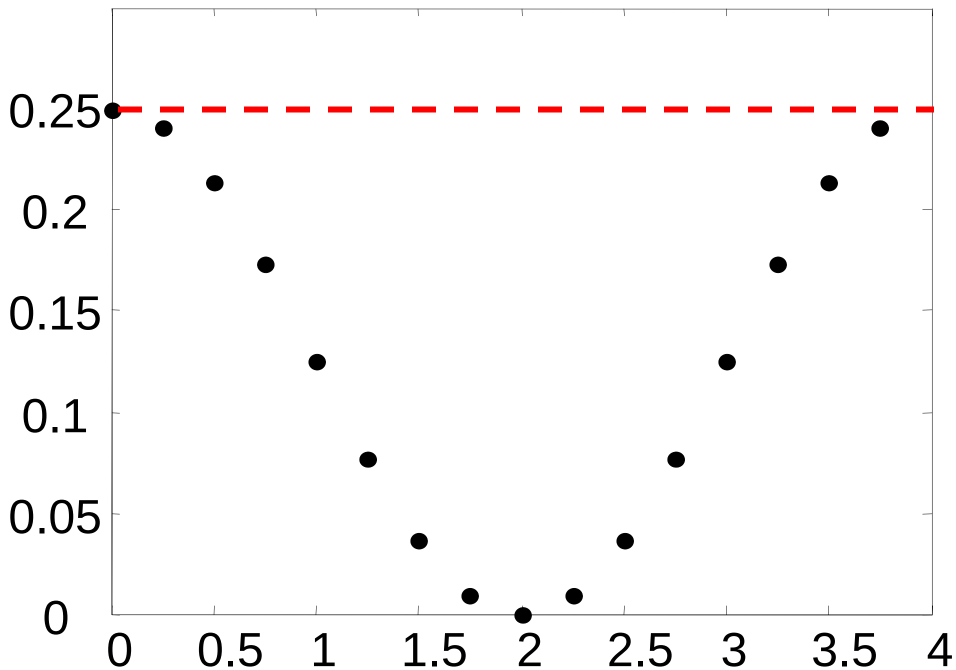


Zero padding example

```
>> t =s_space(0, 4, 16);
>> v = t.*(t < 0.5) + (1-t).*(t >= 0.5 & t<=1.0);
>> h = plot(t, v, 'ko');
>> set(h,  'MarkerFaceColor', 'k')
>> axis([  0  4  0   0.6])

>> dt = t(2) - t(1)

dt = 0.2500
```



```
>> f =s_space(0, 1/dt, 16);

>> vf = FourierT(v, dt);

>> h = plot(f, abs(vf), 'ko');

>>axis([ 0 4 0 0.3])


>> set(h,  'MarkerFaceColor', 'k')

>> df = f(2) - f(1)


df =  0.2500
```



Now, pad the original signal with zeros

```
>> vt2 =[v, zeros(1,16)];
```

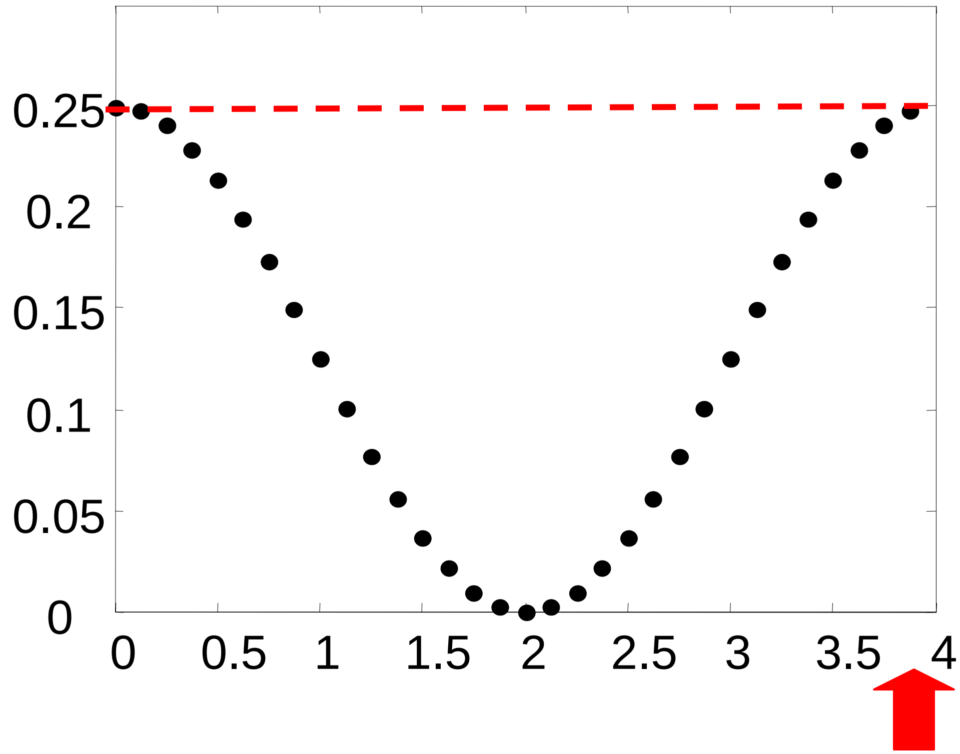
could also use

```
vt2 = padarray(v,[0, 16], 0, ‘post’);
```

```
>> vf2 = FourierT(vt2, dt);

>> h = plot(f, abs(vf2), 'ko');

??? Error using ==> plot

Vectors must be the same lengths.
```

```
>> f =s_space(0, 1/dt, 32);

>> h = plot(f, abs(vf2), 'ko');

>> set(h,  'MarkerFaceColor', 'k')

>>axis([ 0 4 0 0.3])

>> df=f(2) - f(1)

df = 0.1250
```

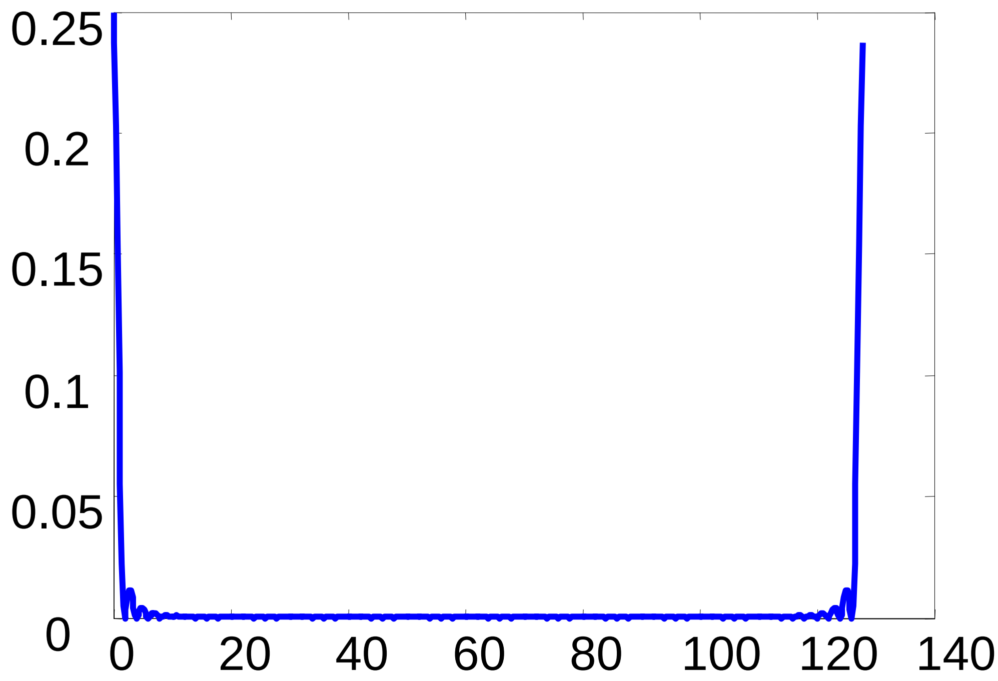
half the former df



same sampling frequency

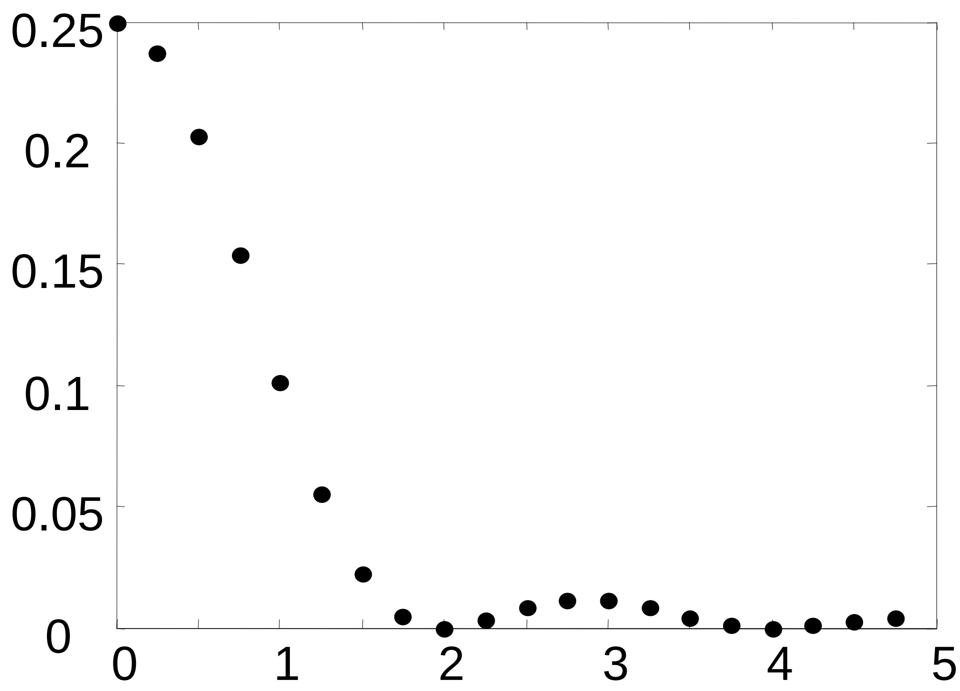
Now, use a much higher sampling frequency

```
>> t =s_space(0, 4, 512);
>> v = t.*(t < 0.5) + (1-t).*(t >= 0.5 & t<=1.0);
>> dt = t(2) -t(1)
dt = 0.0078
>> fs =1/dt
fs = 128

>> f =s_space(0, fs, 512);
>> vf =FourierT(v, dt);
>> plot(f, abs(vf))
```



To see the frequency spectrum on a finer scale

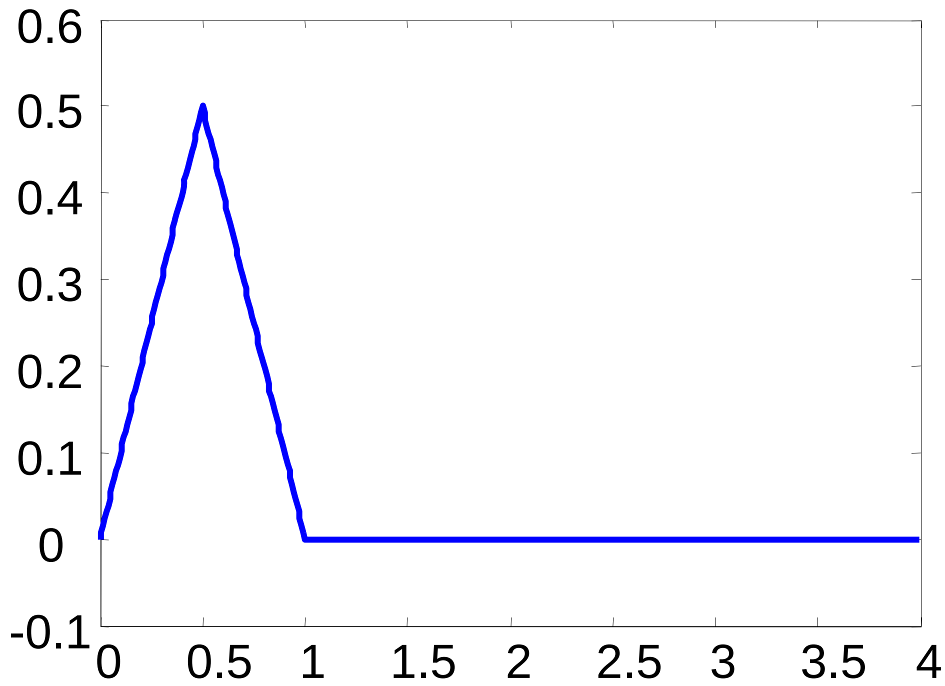
```
>> h = plot(f(1:20), abs(vf(1:20)), 'ko');
>> set(h,  'MarkerFaceColor', 'k')
```



Note: we had some aliasing before

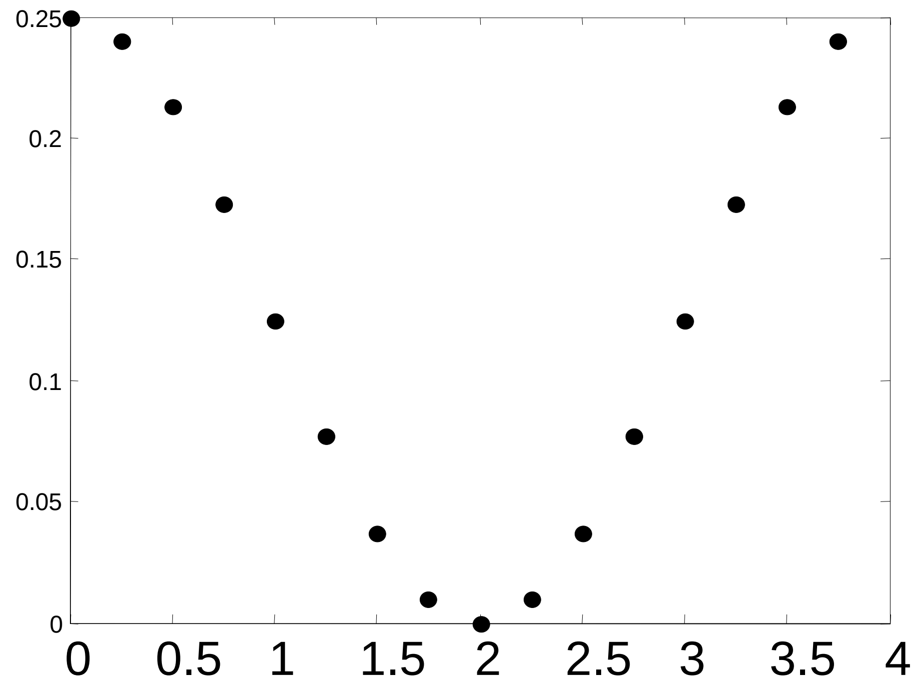
If we do the inverse FFT we do again recover the time function

```
>> vt =IFourierT(vf, dt);
>> plot(t, real(vt))
```



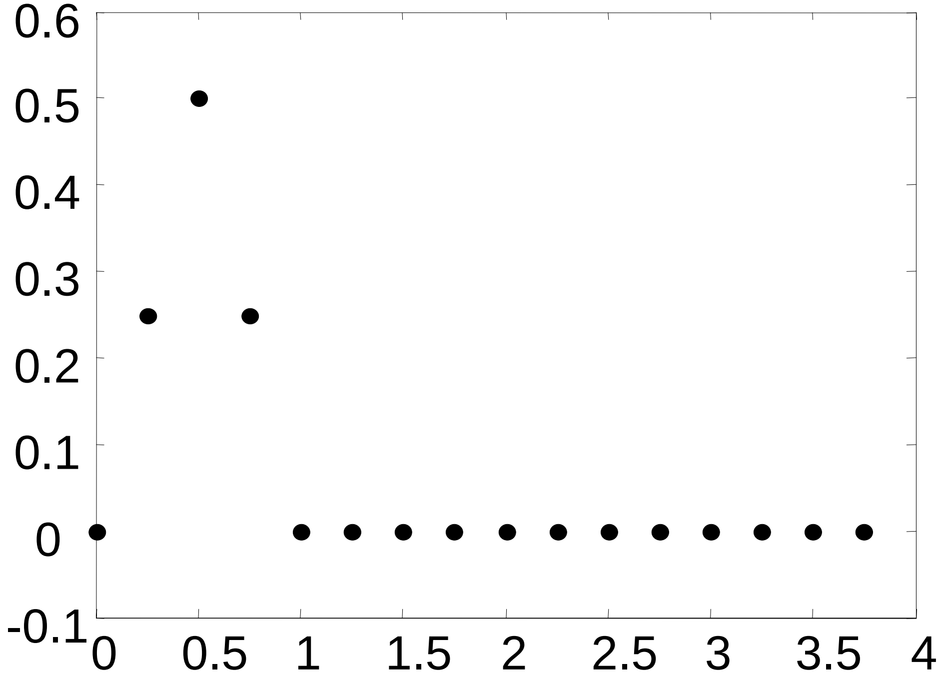
We can do multiple FFTs or IFFTs
all at once if we place the data in columns

```
>> t =s_space(0, 4, 16);
>> v = t.*(t < 0.5) + (1-t).*(t >= 0.5 & t<=1.0);
>> dt = t(2) - t(1);
>> mv= [ v'  v'  v'];
>> mvf = FourierT(mv, dt);
>> vf1 = mvf(:,1);
>> f = s_space(0, 1/dt, 16);
>> h = plot(f, abs(vf1)', 'ko')
>> set(h, 'MarkerFaceColor', 'k')
```



Note that even though there is aliasing here if we do the inverse FFT of these samples we do recover the original time samples:

```
>> v1 =IFourierT(vf1,dt);
>> h=plot(t, real(v1), 'ko');
>> set(h, 'MarkerFaceColor', 'k')
```



Fast Fourier Transform

FFT Examples using the function:

$$y(t) = \begin{cases} A\left[1 - \cos\left(2\pi Ft / N\right)\right]\cos\left(2\pi Ft\right) & \left(0 < t < N / F\right) \\ 0 \quad \mathit{otherwise} \end{cases}$$

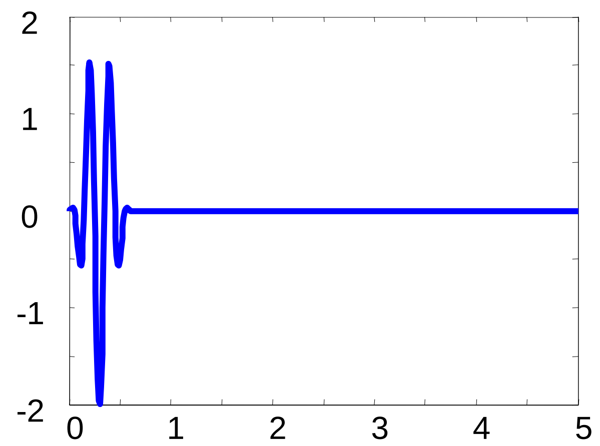
A … controls the amplitude
F … controls the dominant frequency in the pulse
N … controls the number of cycles (amount of "ringing")
in the pulse and hence the bandwidth

MATLAB function:

```
function y = pulse_ref(A,F,N, t)

y = A*(1 -cos(2*pi*F*t./N)).*cos(2*pi*F*t).*(t >= 0 & t <= N/F);
```

Frequency spectrum for relatively wideband pulse

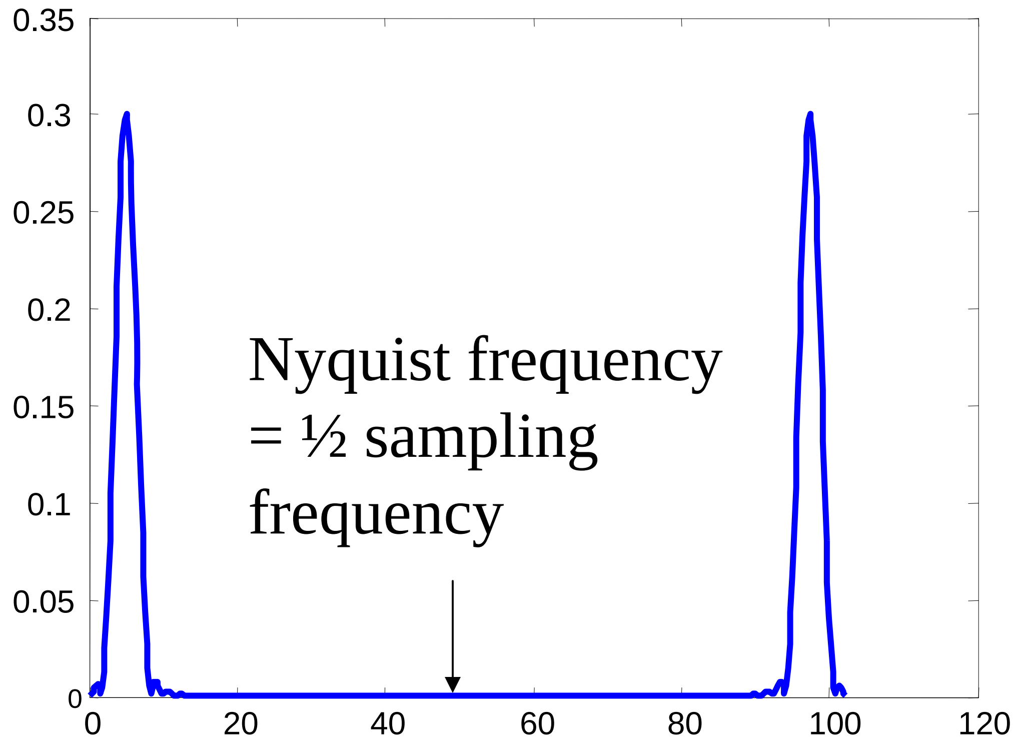
```
>> t = s_space(0,5,512);
>> dt = t(2)- t(1)
dt =  0.0098

>> y=pulse_ref(1,5,3,t);
>> plot(t, y)
>> yf1=FourierT(y,dt);
>> f=s_space(0, 1/dt, 512);
>> f(end)
ans = 102.2000
```



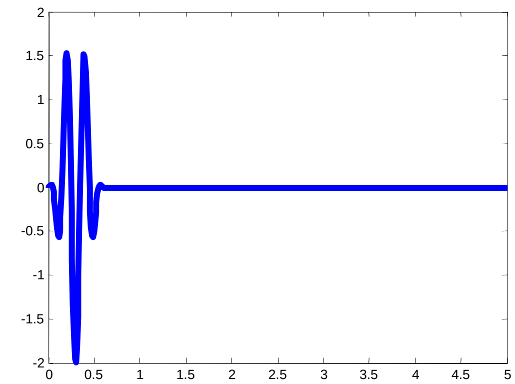
sampling frequency

```
>> plot(f, abs(yf1))
```

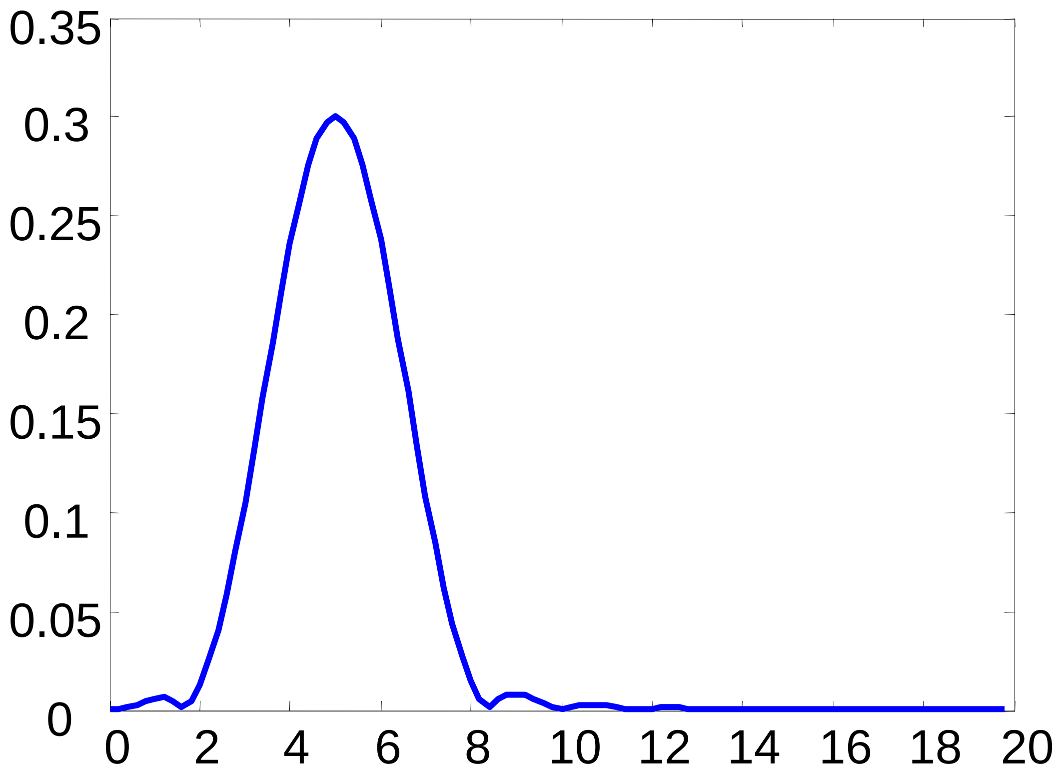
if t is in μsec
f is in MHz



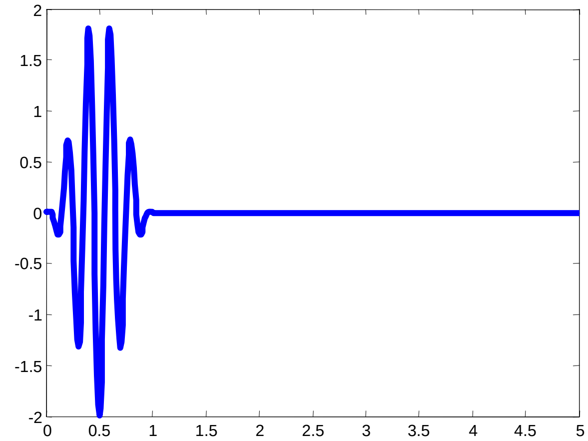
Expanded view
0 – 20 MHz



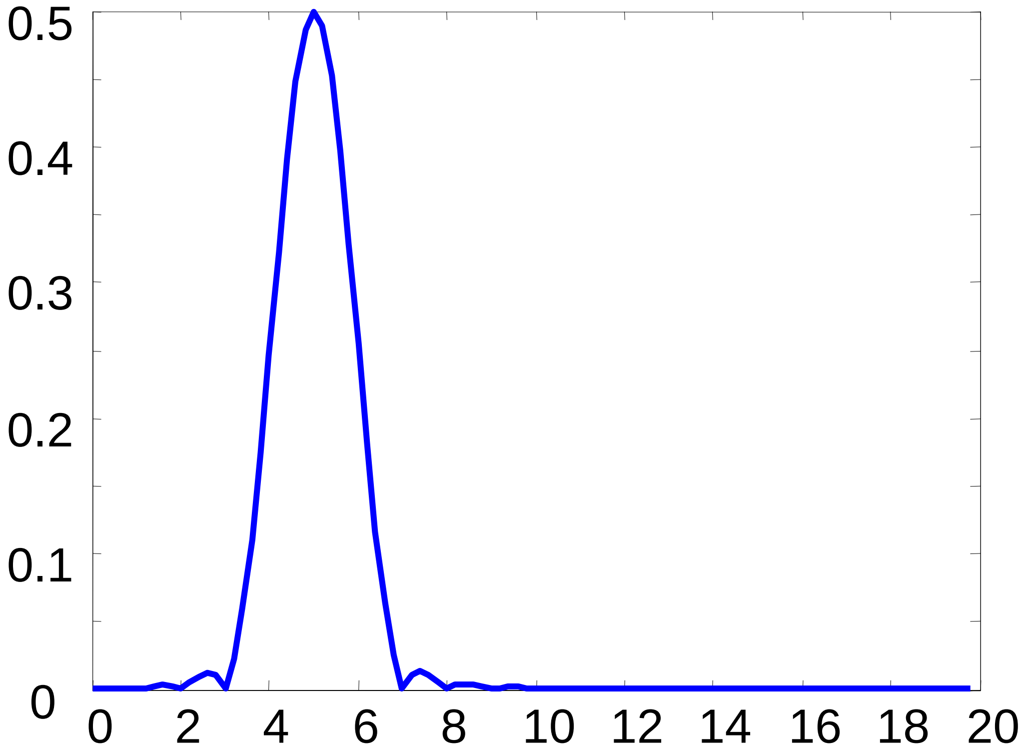
>> plot(f(1:100), abs(yf1(1:100)))



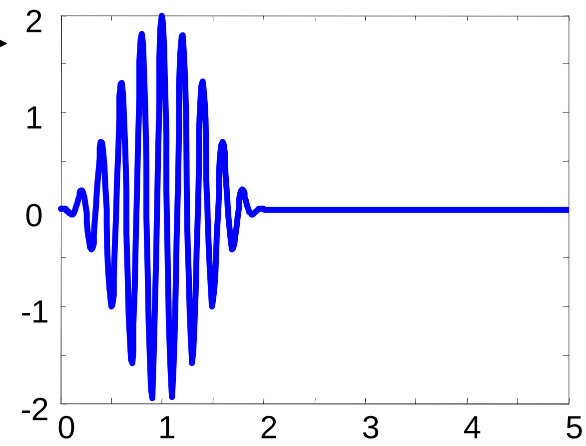
```
>> y = pulse_ref(1,5,5,t);
>> yf2 = FourierT(y, dt);
>> plot(f(1:100), abs(yf2(1:100)))
```



Somewhat more narrow band pulse (only 0 - 20 MHz shown)

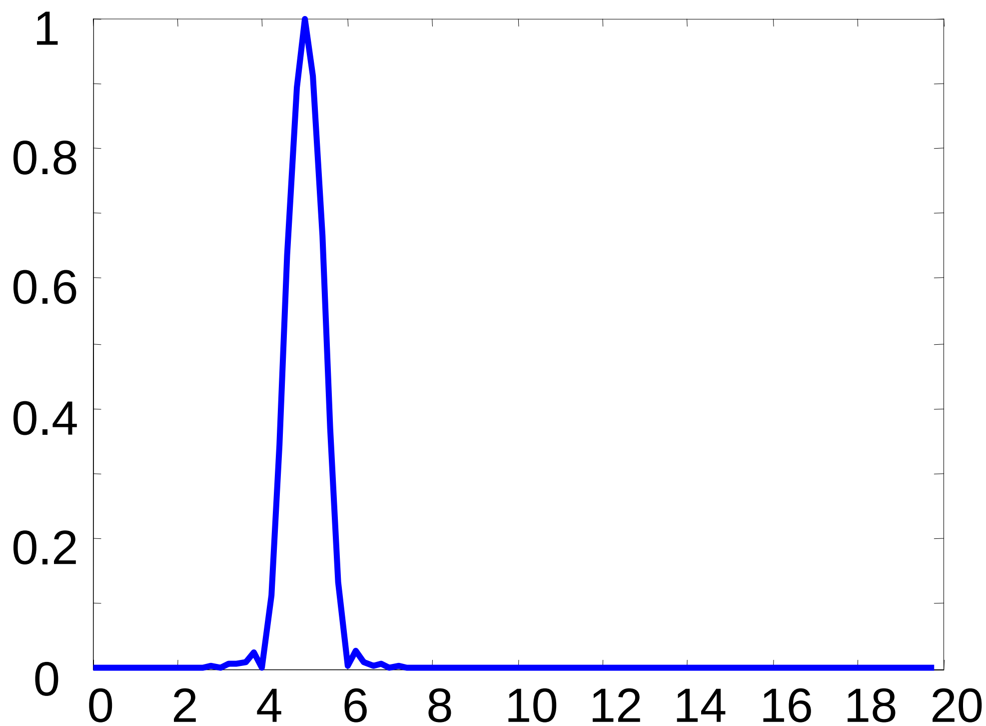


```
>> y = pulse_ref(1,5,10,t);
>> yf3 = FourierT(y, dt);
>> plot(f(1:100), abs(yf3(1:100)))
```



Decrease bandwidth even more

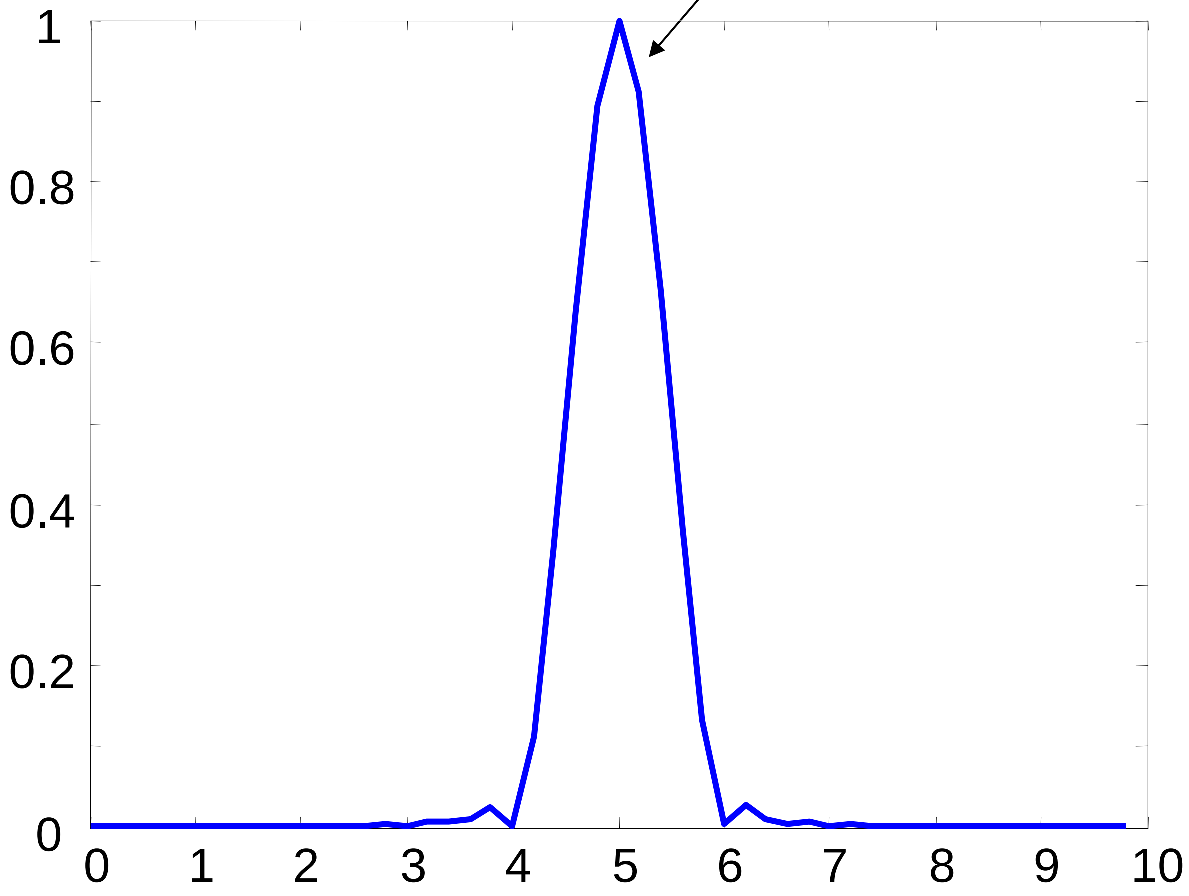
(only 0 - 20 MHz shown)



Show plot in more detail

>> plot(f(1:50), abs(yf3(1:50)))

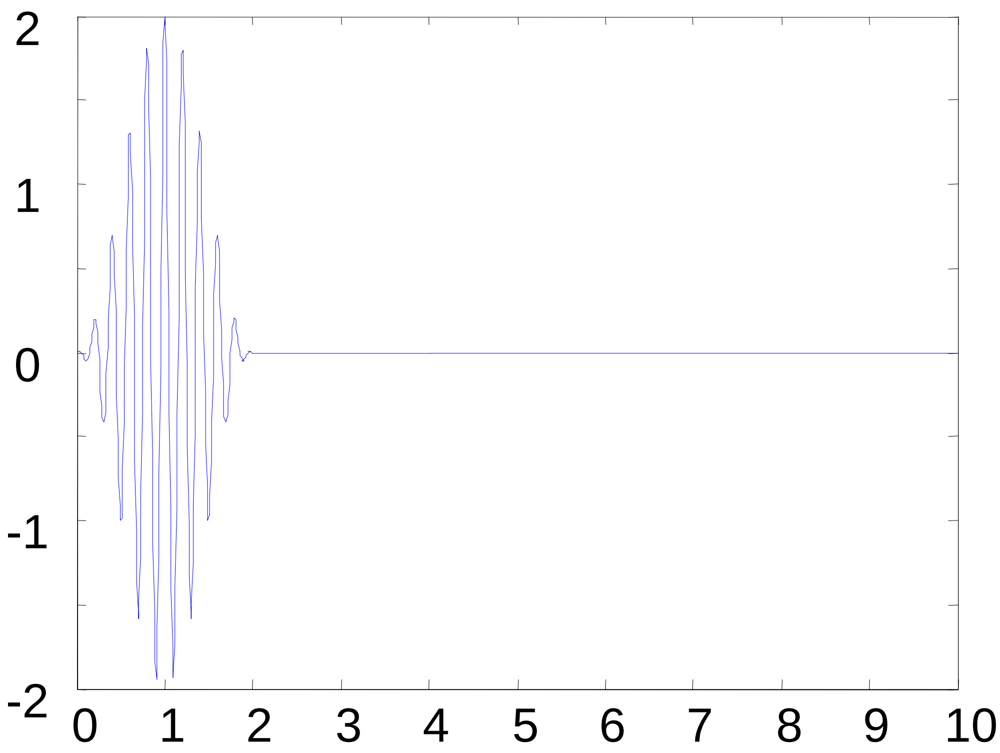
Δf is not quite adequate here
Δf =1/T = 1/5 MHz



we can improve these results with zero padding of our signal

zero padding

```
>> t=s_space(0, 10, 1024);
>> y=pulse_ref(1, 5,10, t);
>> plot(t,y)
>> dt = t(2) -t(1)

dt = 0.0098
```



recall, originally, we had

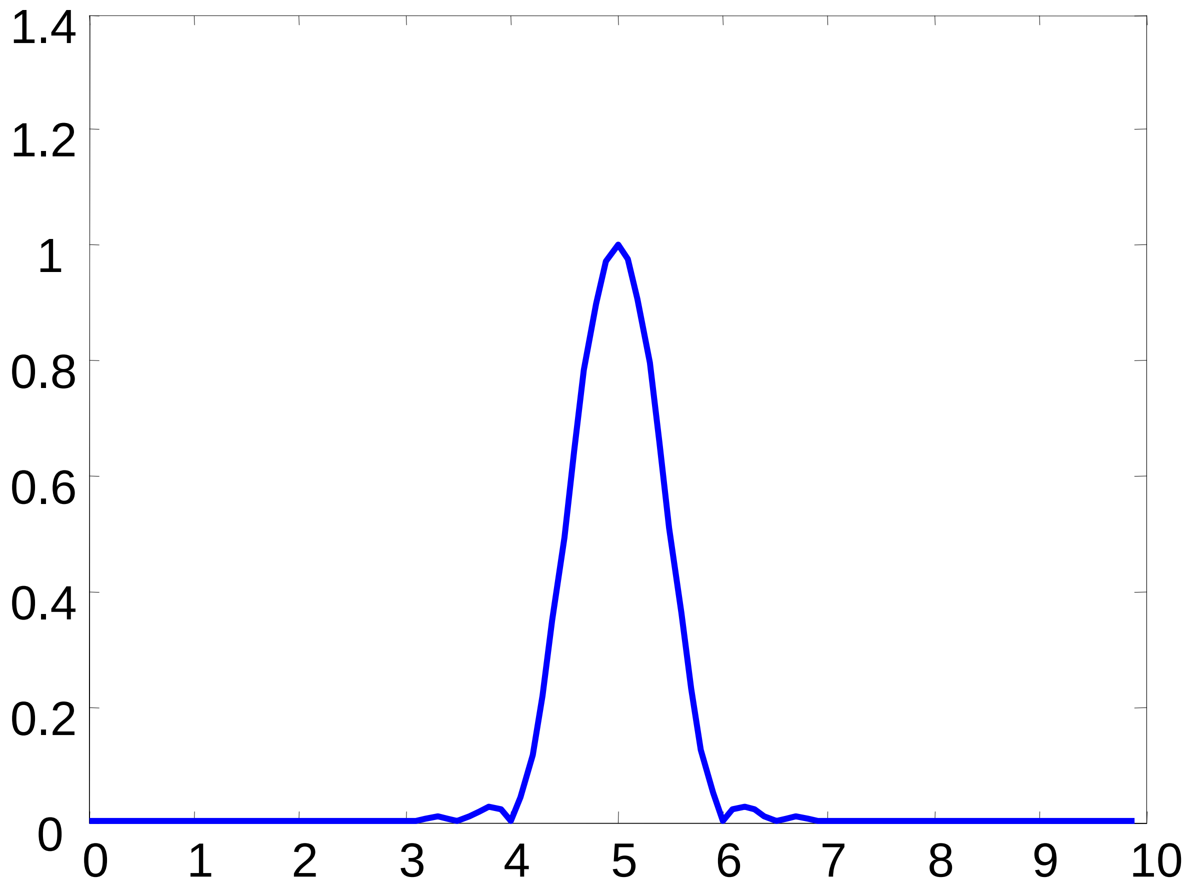
t = s_space(0,5, 512);

we could also zero pad via
t = [t, zeros(1, 512)];

now, have 1024 points over 10 microseconds, so dt is same but T = Ndt is twice as large

```
>> yf4 = FourierT(y, dt);
>> f = s_space(0, 1/dt, 1024);
>> plot(f(1:100), abs(yf4(1:100)))
```

Improved representation of the spectrum $\Delta f = 1/T = 1/10$ MHz

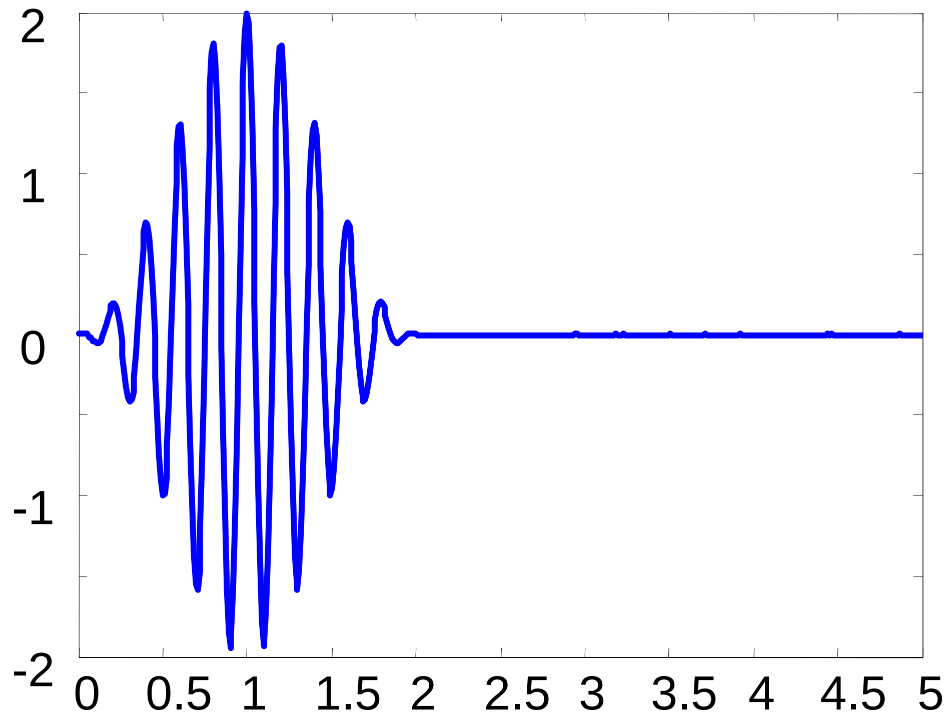


\>> y = IFourierT(yf3, dt);

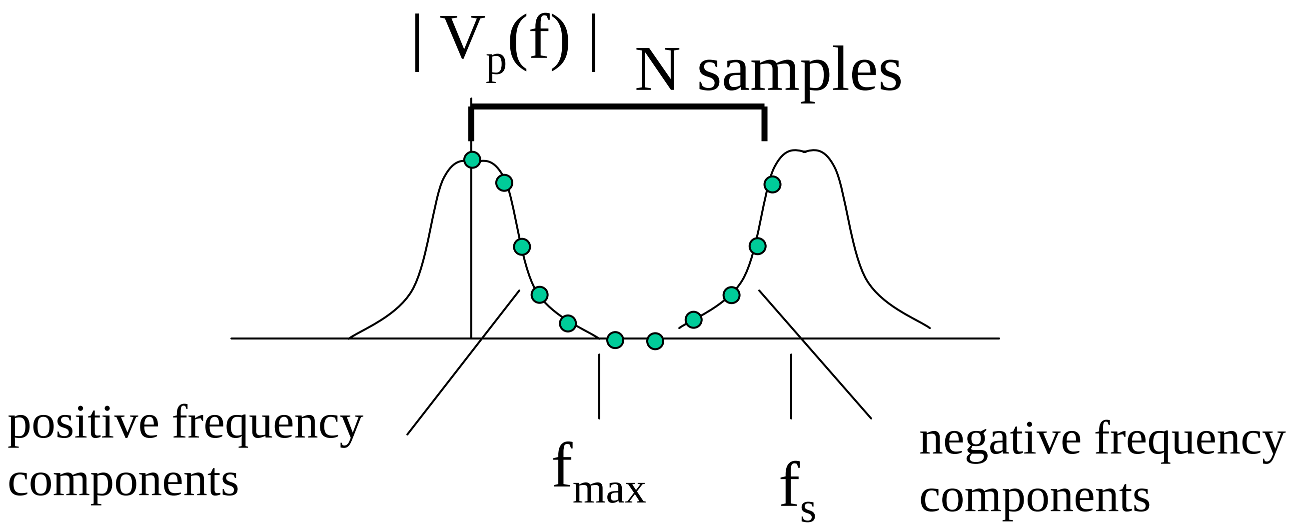
\>> plot(t , real(y));

Example inverse FFT (of yf3)

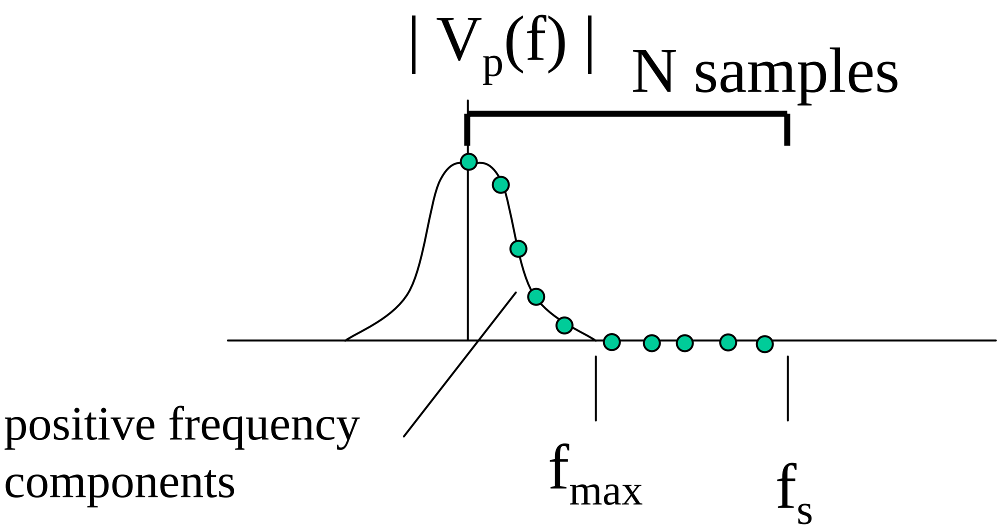
real part is taken here to eliminate any small imaginary components due to numerical round off errors



Fast Fourier Transform



$V(-f) = V^*(f)$ if $v(t)$ is real



$\Longrightarrow$?

in time domain

Fast Fourier Transform

$$v(t) = \int_{-\infty}^{+\infty} V(f)\exp(-2\pi i f t)\,df$$

$$\frac{v(t)}{2} - \frac{i}{2} H[v(t)] = \int_{0}^{+\infty} V(f)\exp(-2\pi i f t)\,df$$

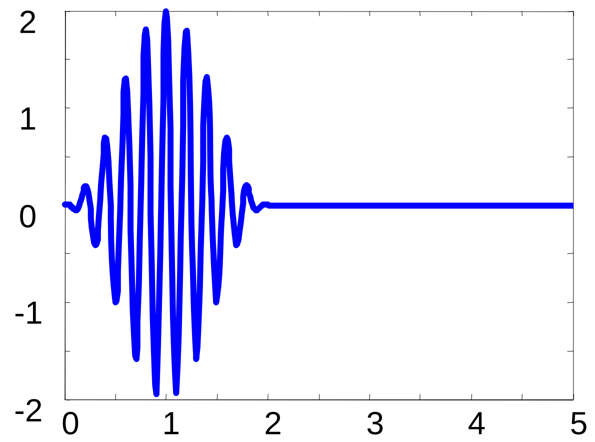
Hilbert transform of v(t)

so

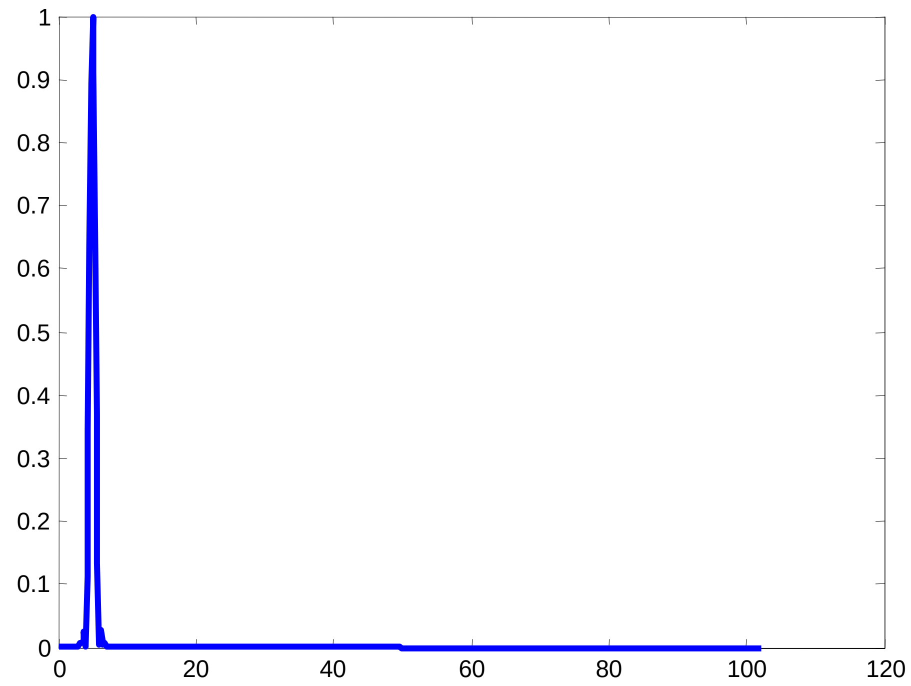
$$v(t) = 2\,\mathrm{Re}\left\{ \int_{0}^{+\infty} V(f)\exp(-2\pi i f t)\,df \right\}$$

Throwing out negative frequency components

```
>> t = s_space(0,5, 512);
>> y = pulse_ref(1,5,10,t);
>> plot(t, y)
>> yf= FourierT(y, dt);
>> yf5 = yf .*(f < 50);
>> plot(f, abs(yf5))
```



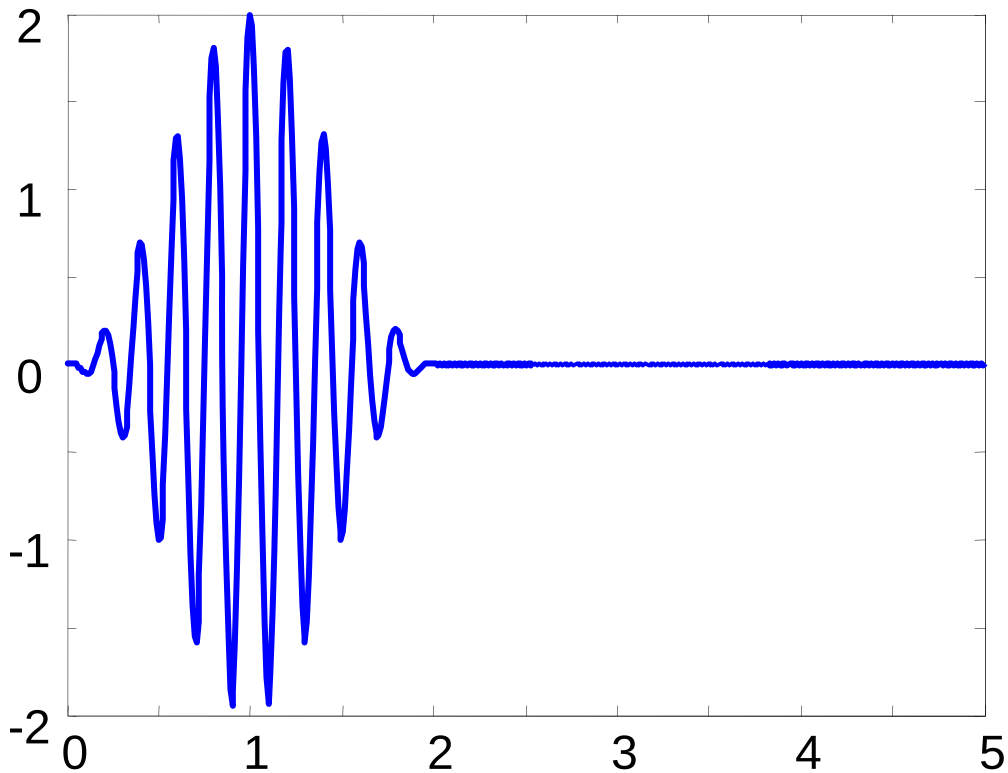
zero negative
frequency components



now, do the inverse FFT
on just these positive
frequency components

```
>> yf5(1) = yf5(1)/2;

>> y = 2*real(IFourierT(yf5, dt));

>> plot(t, y)
```

when throwing out negative frequency components, need to divide dc value by half also (not needed if dc value is zero or already very small)



References

Walker, J.S., **Fast Fourier Transforms**, CRC Press, 1996

Burrus, C.S. and T.W. Parks, **DFT/FFT and Convolution Algorithms**, John Wiley and Sons, New York, 1985.