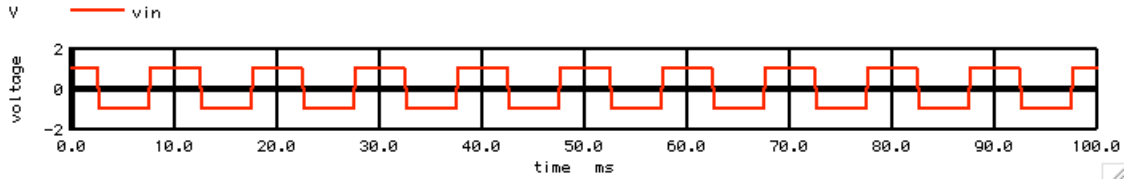
*=========TIMING_Needs===============

Using the new FFT/IFFT functions correctly requires some attention to details. Start off with a sample time of 1 second and a 100Hz square wave input. Note that the linearize function is applied to make time periods consistent.

==========

```
*V_PULSE# NODE_P NODE_N DC     VALUE   PULSE( VINIT  VPULSE TDELAY TRISE   TFALL   PWIDTH PERIOD )
V_SQR     VIN    0      DC     0       PULSE( -1     1      -2.50m  1u      1u      5m     10m    )

.control
*TRAN     TSTEP  TSTOP  TSTART TMAX    ?UIC?
tran      .1m    1      0      .1m
set       pensize = 2

linearize
plot      vin  xlimit 0 100m
```



*=========Having_an_odd_number_of_Points=====================

It might be easier thinking in terms of frequency bins. The output of the FFT function will be an array in that format. The transient analysis is set for one whole second at .1msec time periods. That divides up to give 10,000 time periods. But 10,001 time points are needed to express those points.

==========

```
let       numb2 = length(vin)
print     numb2
```

==========

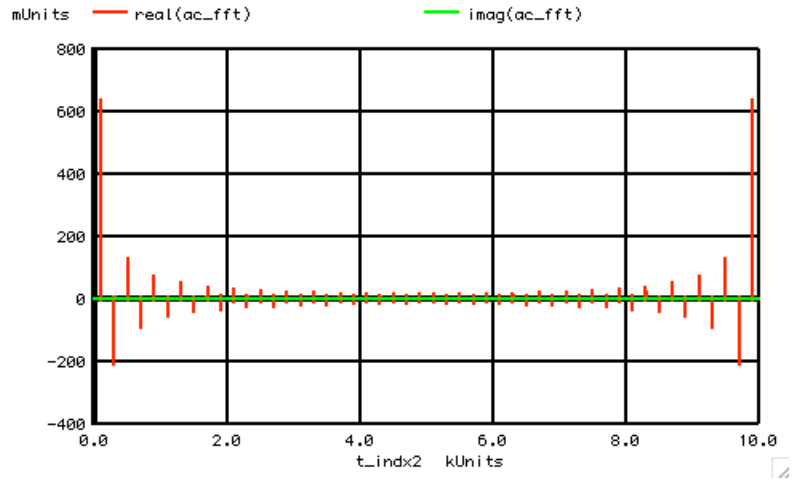
```
numb2 =   1.00010e+04
```

==========

The vector function provides an easy way to construct an index array which can be used to plot the output in terms of its index number. In this case the input signal is made to be complex by adding the +j(0) term. Stick 10,001 complex elements into the fft function, and get out 10,001 complex elements.

==========

```
let       t_indx2 = vector($&numb2)
let       ac      = vin +j(0)
let       ac_fft  = fft(ac)
plot      real(ac_fft) imag(ac_fft) vs t_indx2
```



***=========Covert_to_Frequency_format====================**

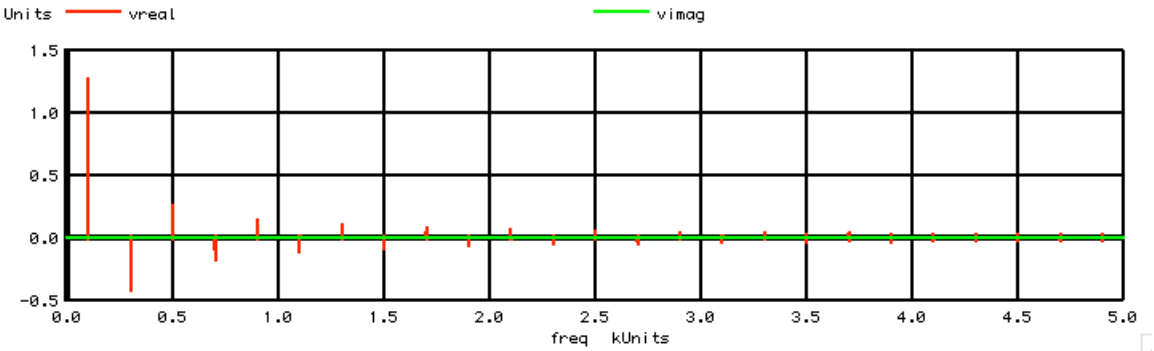
The are really only 5,000 unique points in the fft output.
Values from 0 to 5,000 represent the jw terms and from
10,000 to 5,0001 the -jw terms.

cos(wt) = exp(jwt)/2 + exp(-jwt)/2

The jw terms are a video camera image of the -jw terms.
So only the jw terms from 0 to 5,000 are needed.

Several 5,000 point arrays can be constructed and filled.
Since the total test period is 1sec, the frequency array
does not need a scale factor.

```
================================================================
let       numb_f = (numb2-1)/2
compose   freq   start = 1 stop = $&numb_f  step =1
compose   vreal  start = 1 stop = $&numb_f  step =1
compose   vimag  start = 1 stop = $&numb_f  step =1

let       i = 0
repeat    $&numb_f
let       freq[i] =  freq[i]
let       vreal[i] = 2*real(ac_fft[i])
let       vimag[i] = 2*imag(ac_fft[i])
let       i = i +1
end

plot      vreal vimag vs freq
```



Because the test time is one second, one would expect a value at

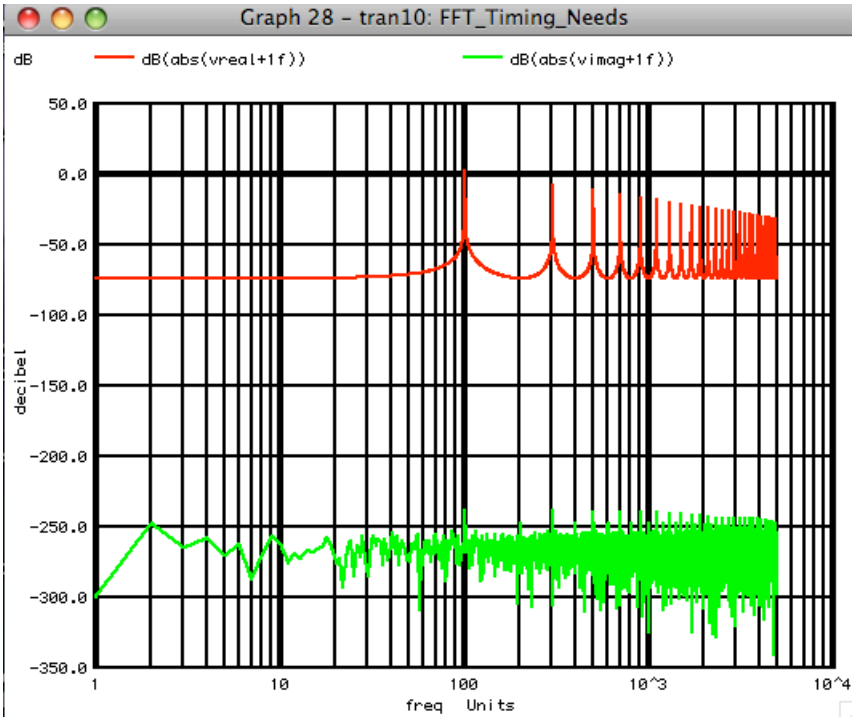
the index of 1 to represent 1Hz, 10Hz at index 10, etc. So in this case the index and frequency in Hz are the same thing.

Note the 100Hz square wave produces signals at frequency bin numbers 100, 300, 500, etc. The bin number for a signal equals how many whole cycles of the signal are contained in the total sample period. Five whole cycles therefore appear at frequency bin number 5.

***=========Need_to_see_in_dB_Fomat====================**

But it is a good idea to check out the spectrum on a dB/freqlog scale.

```
=================================================================
plot      dB(abs(vreal+1f)) dB(abs(vimag+1f)) vs freq xlog
```



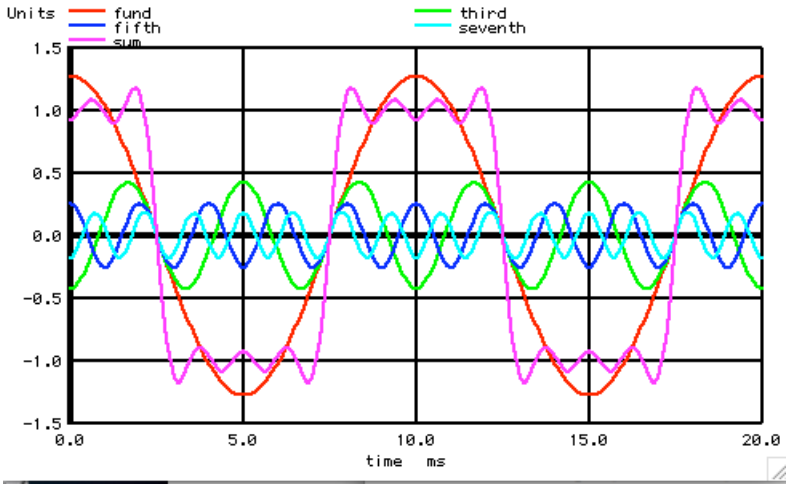
Notice the spectrum leakage at -75dB. Thats because the input is using 10,001 points.

***=========Dissect_the_Spectrum====================**

The spectrum can dissected into harmonic components, and reconstructed using the IFFT function.

```
=================================================================

let       unvect                  = unitvec($&numb2)

let       fundspec                = unvect*0 +j(0)
let       fundspec[100]           = real(ac_fft[100])        +j(imag(ac_fft[100] ))
let       fundspec[numb2-100]     = real(ac_fft[numb2-100])  +j(imag(ac_fft[numb2-100] ))
let       fund =  ifft(fundspec)

let       thirdspec               = unvect*0 +j(0)
let       thirdspec[300]          = real(ac_fft[300])        +j(imag(ac_fft[300] ))
let       thirdspec[numb2-300]    = real(ac_fft[numb2-300])  +j(imag(ac_fft[numb2-300] ))
let       third                   = ifft(thirdspec)
```

```
let       fifthspec               = unvect*0 +j(0)
let       fifthspec[500]          = real(ac_fft[500])         +j(imag(ac_fft[500] ))
let       fifthspec[numb2-500]    = real(ac_fft[numb2-500])   +j(imag(ac_fft[numb2-500] ))
let       fifth                   = ifft(fifthspec)

let       seventhspec             = unvect*0 +j(0)
let       seventhspec[700]        = real(ac_fft[700])         +j(imag(ac_fft[700] ))
let       seventhspec[numb2-700]  = real(ac_fft[numb2-700])   +j(imag(ac_fft[numb2-700] ))
let       seventh                 = ifft(seventhspec)

let       sum  = fund + third + fifth + seventh

set       scale  time
plot      fund  third fifth seventh  sum xlimit 0 20m
```



This is a useful way to checkout group delays in lowpass filters.

==========Full_Netlist_For_Copy_Paste==========

```
FFT_Timing_Needs_SQ0

*V_PULSE# NODE_P NODE_N DC     VALUE  PULSE( VINIT  VPULSE TDELAY TRISE  TFALL  PWIDTH PERIOD )
V_SQR     VIN    0      DC     0      PULSE( -1     1      -2.50m  1u     1u     5m     10m    )

.control
*TRAN     TSTEP  TSTOP  TSTART TMAX   ?UIC?
tran     .1m     1      0      .1m
set       pensize = 2

linearize
plot      vin  xlimit 0 100m

let       numb2 = length(vin)
print     numb2
let       t_indx2 = vector($&numb2)

let       ac = vin +j(0)
let       ac_fft=fft(ac)
plot      real(ac_fft) imag(ac_fft) vs t_indx2

let       numb_f = (numb2-1)/2
compose   freq   start = 1 stop = $&numb_f  step =1
compose   vreal  start = 1 stop = $&numb_f  step =1
compose   vimag  start = 1 stop = $&numb_f  step =1

let       i = 0
repeat    $&numb_f
let       freq[i] =  freq[i]
let       vreal[i] = 2*real(ac_fft[i])
let       vimag[i] = 2*imag(ac_fft[i])
let       i = i +1
end

plot      vreal vimag vs freq

plot      mag(vreal+1n) mag(vimag+1n) vs freq loglog

plot      dB(abs(vreal+1f)) dB(abs(vimag+1f)) vs freq xlog
```

```
let       harm=sortorder(1/(mag(real(ac_fft))+1u))
*plot     harm

print     ac_fft[100]
print     ac_fft[numb2-100]

let       unvect               = unitvec($&numb2)

let       fundspec                = unvect*0 +j(0)
let       fundspec[100]           = real(ac_fft[100])        +j(imag(ac_fft[100] ))
let       fundspec[numb2-100]     = real(ac_fft[numb2-100])  +j(imag(ac_fft[numb2-100] ))
let       fund =  ifft(fundspec)

let       thirdspec               = unvect*0 +j(0)
let       thirdspec[300]          = real(ac_fft[300])        +j(imag(ac_fft[300] ))
let       thirdspec[numb2-300]    = real(ac_fft[numb2-300])  +j(imag(ac_fft[numb2-300] ))
let       third                   = ifft(thirdspec)

let       fifthspec               = unvect*0 +j(0)
let       fifthspec[500]          = real(ac_fft[500])        +j(imag(ac_fft[500] ))
let       fifthspec[numb2-500]    = real(ac_fft[numb2-500])  +j(imag(ac_fft[numb2-500] ))
let       fifth                   = ifft(fifthspec)

let       seventhspec             = unvect*0 +j(0)
let       seventhspec[700]        = real(ac_fft[700])        +j(imag(ac_fft[700] ))
let       seventhspec[numb2-700]  = real(ac_fft[numb2-700])  +j(imag(ac_fft[numb2-700] ))
let       seventh                 = ifft(seventhspec)

let       sum  = fund + third + fifth + seventh

set       scale  time
plot      fund  third fifth seventh  sum xlimit 0 20m

.endc
.end
```