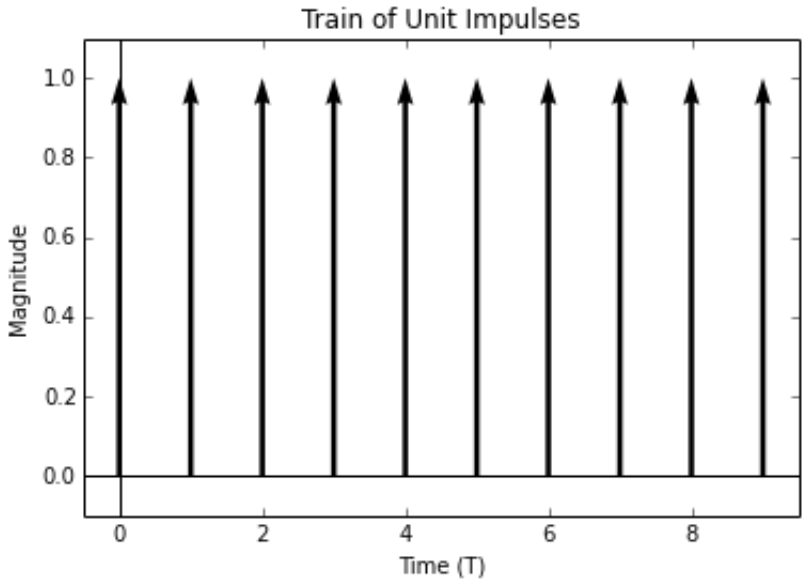
Problem Set 7 - Mitchell Kwock

In [1]:
```
# Setup and import
import numpy as np
import matplotlib.pyplot as plt
plot = plt.plot

%matplotlib inline

def arrow_plot(arrows):
    # Draws an array of vectors in the form x_start, y_start, x_mag, y_mag
    # Obtained from: http://stackoverflow.com/questions/12265234/plotting-2d-ve
ctors-in-python-matplotlib
    X, Y, U, V = zip(*arrows)
    plt.figure()
    ax = plt.gca()
    ax.quiver(X, Y, U, V, angles='xy', scale_units='xy', scale=1)

def plot_axes(xlims=[-1, 1], ylims=[-1, 1]):
    # I use this too often, very nice to turn on axes to see what's going on
    plt.plot(xlims, [0, 0], 'k')
    plt.plot([0, 0], ylims, 'k')
    plt.xlim(xlims)
    plt.ylim(ylims)
```

Problem 1) Consider a Train of unit impulses separated by T time units given by $p(t) = \sum\limits_{k=-\infty}^{\infty} \delta(t-kt)$

a) Sketch a representation of p(t)

In [2]:
```
# Make Vectors to plot
arrows = np.array([[x, 0, 0, 1] for x in range(10)])

arrow_plot(arrows)

plt.xlabel('Time (T)')
plt.ylabel('Magnitude')
plt.title('Train of Unit Impulses')
plot_axes([-0.5, 9.5], [-0.1, 1.1])
plt.draw()
plt.show()
```



b) Find the Fourier Series representation of p(t) with an infinite number of terms

$b_n = \frac{1}{T}\int_{-T/2}^{T/2}p(t) * cos(\frac{2\pi nt}{T})dt$

Of course, $p(t) = \delta(t)$, which is zero at all points except at t=0, where the integral over that point is equal to 1, therefore

$b_n = \frac{1}{T} (1 * cos(0)) = \frac{1}{T}$

Which translates to a Fourier Series of: $f(x) = \frac{1}{T}\sum\limits_{n=1}^{\infty}cos(\frac{n\pi x}{T})$

c) Let a function x(t) be represented as a Fourier Series with an infinite number of terms as follows: $x(t) = \sum\limits_{k=-\infty}^{\infty}C_ke^{\frac{2\pi jkt}{T}}$ Find $X(\omega)$ in terms of C_k.

Because x(t) is already expressed in a Fourier Series sum, $X(\omega)$ can be expressed as a sum of the complex coefficients C_k multiplied by the respective $\delta(t)$ functions.

$X(\omega) = \sum\limits_{k = -\infty}^{\infty}C_k * \delta(\omega - \frac{k}{T})$

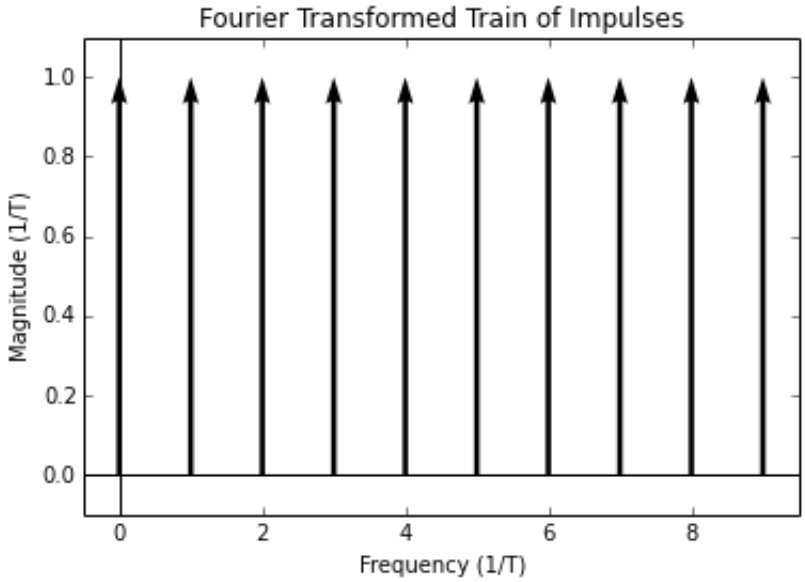
d) Using our answer to the previous two parts, find $P(\omega)$

$C_k = 1$

$P(\omega) = \frac{1}{T}\sum\limits_{k=-\infty}^{\infty}\delta(\omega - \frac{k}{T})$

e) Sketch $P(\omega)$. How does changing T affect p(t) and $P(\omega)$? Is this what you would expect?

In [3]:
```
arrows = np.array([[x, 0, 0, 1] for x in range(10)])

arrow_plot(arrows)

plt.xlabel('Frequency (1/T)')
plt.ylabel('Magnitude (1/T)')
plt.title('Fourier Transformed Train of Impulses')
plot_axes([-0.5, 9.5], [-0.1, 1.1])
plt.draw()
plt.show()
```

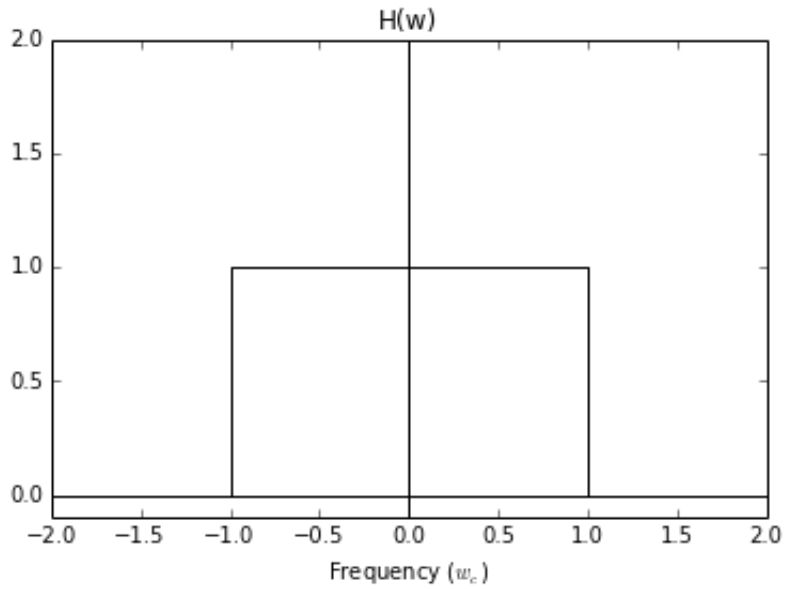


Increasing T (The period of the impulses) decreases the Fundamental Frequency of the impulses in $P(\omega)$ and spaces the impulses more closely. This is exactly what we would expect since increasing the period of the signal in the time domain decreases the frequency because Frequency and Period are inversely proportional.

Problem 2) Consider an LTI system with an impulse response $h(t)$, input signal $x(t)$ and output $y(t)$. It is known that $H(\omega)$ is the following.

In [4]:
```
plot([-2, -1, -1, 1, 1, 2], [0, 0, 1, 1, 0 , 0], 'k')
plot_axes([-2, 2], [-0.1, 2])
plt.title('H(w)')
plt.xlabel('Frequency ($w_c$)')
```

Out[4]: <matplotlib.text.Text at 0xa6fb940>



a) Find $h(t)$

$h(t) = F^{-1}(H(\omega)) = \int_{-\infty}^{\infty}H(\omega)e^{2\pi jt\omega}d\omega$

$= \int_{-\omega_c}^{\omega_c}e^{2\pi jt\omega}d\omega$

$= \frac{1}{2\pi jt} (e^{2\pi it\omega})$ from $\omega = $ -1 to 1

$= \frac{1}{2\pi jt} (e^{2\pi it} - e^{-2\pi it})$

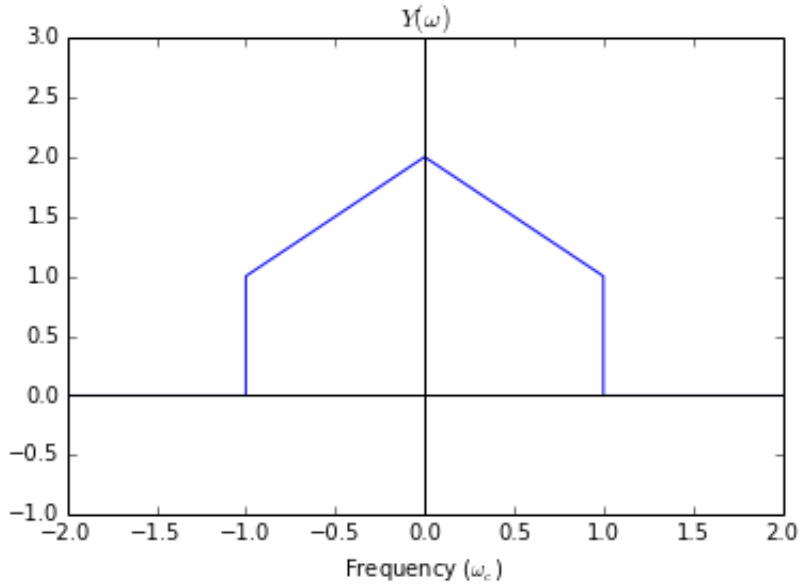
$= \frac{1}{2\pi jt} * 2cos(2\pi t)$

$= \frac{1}{\pi jt} * cos(2\pi t) = \frac{cos(2\pi t)}{\pi jt}$

b) Suppose that $X(\omega)$ is a four part piecewise function described in the actual documentation. Please sketch $Y(\omega)$

In [5]:
```
plot([-2, -1, -1, 0, 1, 1, 2], [0, 0, 1, 2, 1, 0, 0])
plot_axes([-2, 2], [-1, 3])
plt.xlabel('Frequency ($\omega_c$)')
plt.title('$Y(\omega)$')
```

Out[5]: `<matplotlib.text.Text at 0xa9a0e80>`



c) Explain why this LTI system is known as an ideal low-pass filter with cut-off frequency ω_c

The multiplication in the frequency domain causes any frequencies above the cut-off frequency to be completely removed. All frequencies below the cut-off are unchanged. Therefore, the lower frequencies pass through unchanged, and the higher frequencies are removed

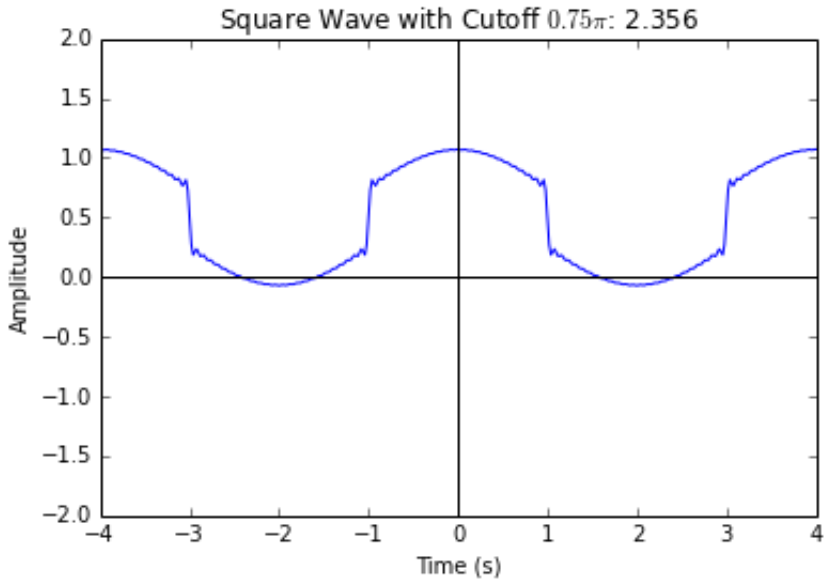
d) [Text] Modify the code in the third cell of the ipython notebook to implement a low-pass filter with cutoff frequency of $\omega_c = 0.75\pi$. Run the code to see the filtered version of the square wave. Repeate this for $\omega_c=1.75\pi$. You should turn in the plots that are generated with both cut-off frequencies.

In [6]:
```
def fs_square_lpass(ts, K=3, T=4, C=np.pi):
    # computes a fourier series representation of a square wave
    # with K terms in the Fourier series approximation
    # if K is odd, terms -(K-1)/2 -> (K-1)/2 are used
    # if K is even terms -K/2 -> K/2-1 are used

    # Stops adding terms once the frequency is higher than the Cutoff

    # create an array to store the signal
    x = np.zeros(len(ts), dtype=np.complex128)


    # if K is even
    if np.mod(K,2) == 0:
        for n in range(-int(K/2), int(K/2)):
            if(2 * np.pi / T * n <= C):
                Coeff= 0.5*np.sinc(float(n)/2)
                x = x + Coeff*np.exp(1j*2*np.pi/T*n*ts)


    # if K is odd
    if np.mod(K,2) == 1:
        for n in range(-int((K-1)/2), int((K-1)/2)+1):
            if(2 * np.pi / T * n <= C):
                Coeff = 0.5*np.sinc(float(n)/2)
                x = x + Coeff*np.exp(1j*2*np.pi/T*n*ts)

    return x
```

In [7]:
```
exes = np.linspace(-4, 4, 401)

cutoff = 0.75 * np.pi

wise = fs_square_lpass(exes, 100, 4, cutoff)
plot(exes, wise)

plot_axes([-4, 4], [-2, 2])
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Square Wave with Cutoff $0.75\pi$: ' + str(cutoff)[:5])
```

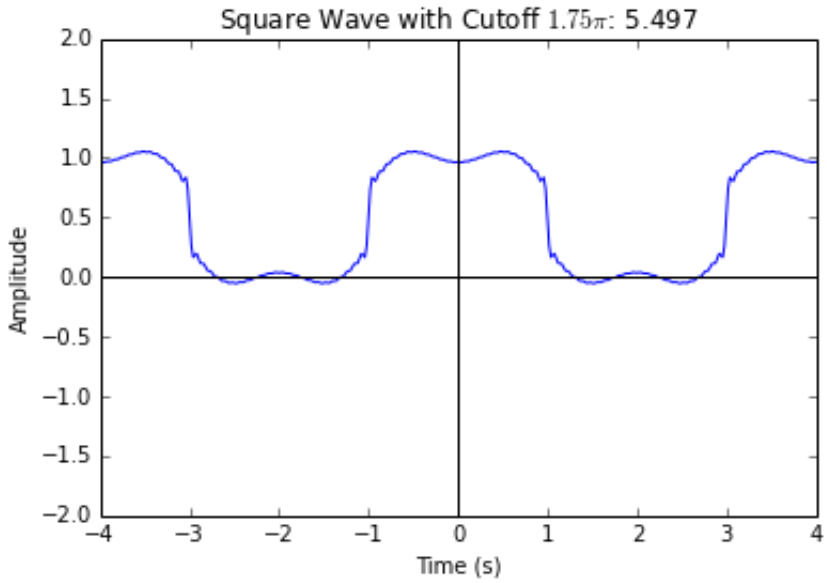
```
C:\Users\mkwock\AppData\Local\Continuum\Anaconda\lib\site-packages\numpy\core\n
umeric.py:460: ComplexWarning: Casting complex values to real discards the imag
inary part
  return array(a, dtype, copy=False, order=order)
```

Out[7]:
```
<matplotlib.text.Text at 0xad24cf8>
```



In [8]:
```
exes = np.linspace(-4, 4, 401)

cutoff = 1.75 * np.pi

wise = fs_square_lpass(exes, 100, 4, cutoff)
plot(exes, wise)

plot_axes([-4, 4], [-2, 2])
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Square Wave with Cutoff $1.75\pi$: ' + str(cutoff)[:5])
```

Out[8]:
```
<matplotlib.text.Text at 0xb61d400>
```



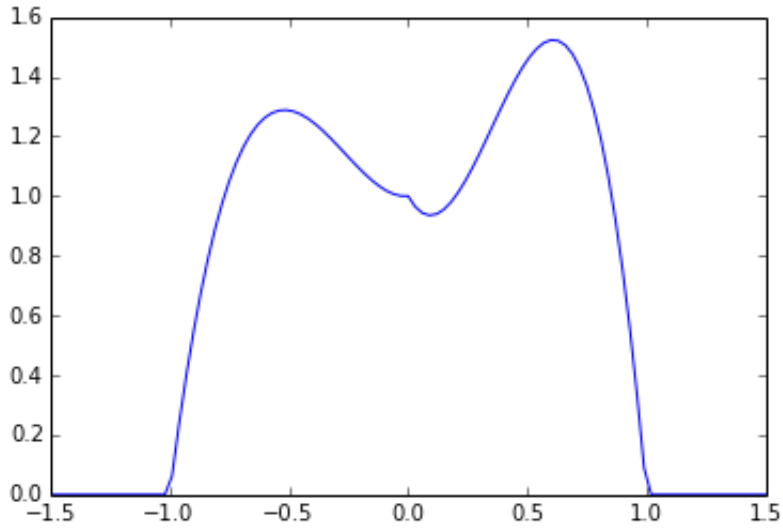
In [8]:
```

```

Problem 3) Consider a signal $x(t)$ which is band-limited to the f < ω_M. Suppose $X(\omega)$ is a relatively complex curve seen in the homework. Sketch $y(t) = x(t)cos(\omega_ct)$

Because multiplication in the time domain is convolution in the frequency domain, multiplying $cos(\omega t)$ is the same as convolving $X(\omega)$ with $\frac{1}{2}[\delta (t - \omega_c) + \delta (t + \omega_c)]$

In [9]:
```
def appx_curve(t):
    if(t < -1 or t > 1):
        return 0
    if(t < 0):
    # Cubic function through (-1, 0), (-2/3, 1.2), (-1/3, 1.,2,), and (0,
1),,,5,7,8
        return 4.485 * t ** 3 + 3.578 * t **2 + 0.094 * t  + 1
    # Cubic function through (0, 1), (0.333, 0.9), (0.666, 1.3), (1, 0)
    return -8.56603 * t**3 +\
    9.02923 * t**2 + \
    -1.46321 * t + \
    1
exes = np.linspace(-1.5, 1.5, 101)
wise = [appx_curve(x) for x in exes]


plot(exes, wise)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0xb8947b8>]
```



In [10]:
```
def appx_cos_multiply(t):
    t_new = 0
    omega_c = 2.5
    if(t < 0):
        t_new = t + omega_c
    else:
        t_new = t - omega_c

    return 0.5 * appx_curve(t_new)

exes = np.linspace(-5, 5, 201)
wise = [appx_cos_multiply(x) for x in exes]

arrows = [[-2.5, 0, 0, 0.5], [2.5, 0, 0, 0.5]]
arrow_plot(arrows)

plot_axes([-5, 5], [-0.1, 2])



plot(exes, wise)
plt.xlabel('Frequency (Centers at +/- $\omega_c$) and widths are $2\omega_M$')
plt.ylabel('Intensity')
plt.title('$Y(\omega) = X(\omega) * COS(\omega)$')
```

Out[10]: <matplotlib.text.Text at 0xb9e7828>