Homework Manual

A series of homework problems are listed throughout the course Modeling and Measuring Ultrasonic Nondestructive Evaluation Systems. Some are taken from the chapters and appendices of the book, *Ultrasonics Nondestructive Evaluation Systems – Models and Measurements*, and some are special problems associated with the course notes. Here are the problem statements as well as solutions. Where homework problems taken from the book refer to results or figures in the book text I have tried to include those items with the homework problem statement. There are other problems in the book whose statements and solutions are not given that you might like to examine.

HOMEWORK MANUAL TABLE OF CONTENTS

COURSE SECTION	PROBLEMS	HW PAGE
1 Introduction to Ultrasonic Nondestructive Evaluation Systems		
2 Introduction to Matlab	M.1	5
	M.2	7
3 Ultrasonic System Overview (Chapter 1)		
4 The Fourier Transform (Appendix A)		
5 The Fast Fourier Transform (FFT) and Matlab Examples (Appendix A)	A.1	9
	A.2	18
	S.0	25
	S.1	34
6 Impedance Concepts and Thévenin Equivalent Systems (Appendix B)	B.2	44
7 Pulser Characteristics – Models and Measurements (Chapter 2)	2.1	49
8 Linear Electrical and Acoustic Systems – Some Basic Concepts (Appendix C)	C.2	59
	C.3	77

9 Cabling – Models and Measurements (Chapter 3)	3.1	84
	3.2	87
10 Transducer Modeling (Chapter 4)		
11 Wave Propagation Fundamentals -1 (Appendix D) Propagation of Plane and Spherical Waves		
12 Wave Propagation Fundamentals -2 (Appendix D) Reflection and Transmission		
13 Wave Propagation Fundamentals -3 (Appendix D) Ultrasonic Attenuation	D.1	95
	S.2	108
14 Waves Used in NDE (Appendix E)	S.3	114
	S.4	116
	S.5	118
15 Acoustic/Elastic Transfer Function (Chapter 5)		
16 The Ultrasound Reception Process (Chapter 5)	5.1	121
17 Transducer Characterization (Chapter 6)	6.1	125

18 The System Function (Chapter 7)	7.1	134
19 Transducer Sound Radiation (Chapter 8)	8.1	141
	8.2	145
	8.4	151
	8.5	155
20 The Paraxial Approximation (Chapter 8)		
21 Gaussian Beams and Transducer Modeling (Appendix F, Chapter 9)	F.2	159
	F.3	169
22 Flaw Scattering Models (Chapter 10)		
23 Ultrasonic System Measurement Model – I (Chapter 11)		
24 Ultrasonic System Measurement Model – II (Chapter 11)		
25 Measurement Model Examples (Chapter 11)		
26 Special Topics –I Dispersion		
27 Statistical Foundations of Pattern Recognition		
28 Neural Networks		

M.1 When a wave is incident on an interface at oblique incidence, the angle of the refracted wave is controlled by Snell's law (see Fig 1). Write a MATLAB function snells_law that takes as its input the refracted angle (in degrees) and the wave speeds in the two media and returns the required incident angle (in degrees), i.e. the calling sequence should look like ang_in =snells_law(ang_out, c1, c2)

Take the two media to be water (c1 =1480 m/sec) and steel (c2 = 5900 m/sec). What are the incident angles for refracted angles of 0, 45, 70, and 90 degrees for this case? What happens if the incident angle was 15 degrees?

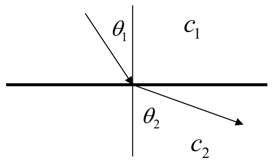
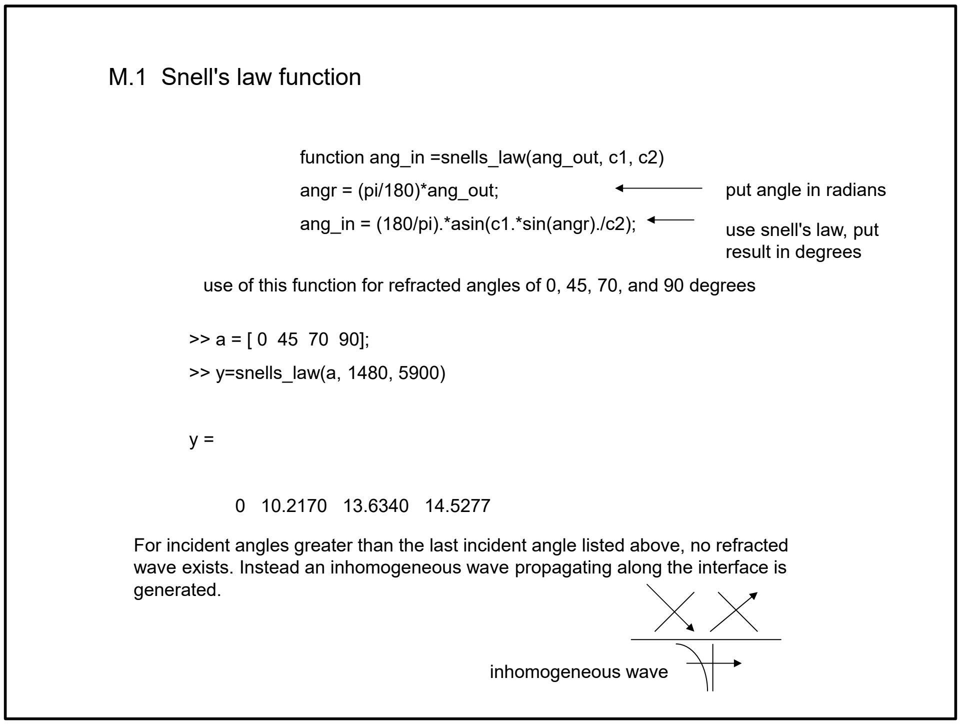


Fig. 1

Snell's law

$$\frac{\sin\theta_1}{c_1} = \frac{\sin\theta_2}{c_2}$$



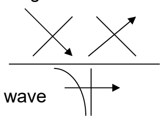
M.1 Snell's law function

```
function ang_in =snells_law(ang_out, c1, c2)
angr = (pi/180)*ang_out;                        ← put angle in radians
ang_in = (180/pi).*asin(c1.*sin(angr)./c2);     ← use snell's law, put
                                                  result in degrees
```

use of this function for refracted angles of 0, 45, 70, and 90 degrees

```
>> a = [ 0  45  70  90];
>> y=snells_law(a, 1480, 5900)


y =


      0   10.2170   13.6340   14.5277
```

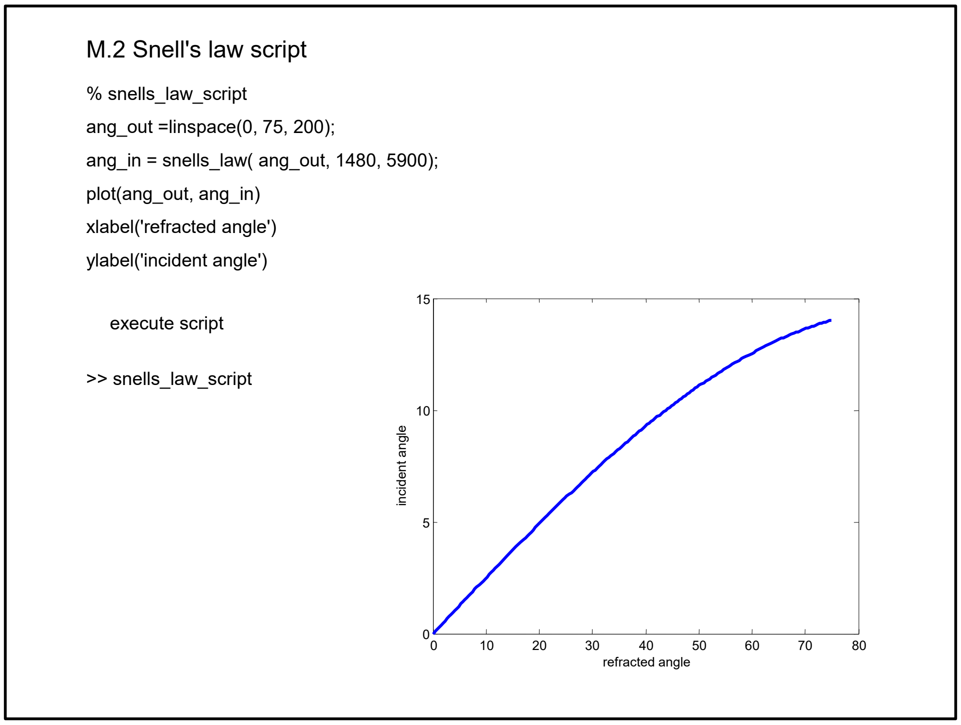
For incident angles greater than the last incident angle listed above, no refracted wave exists. Instead an inhomogeneous wave propagating along the interface is generated.



inhomogeneous wave

Here is a Snell’s law Matlab function that can take the refracted angle and wave speeds and produce the incident angle. There is another snells_law function (same name, unfortunately) given in the Matlab downloads (from book 1) that is much more general and can solve for either the incident angle or refracted angle, depending on the given data. You might want to rename this function so there is not a conflict. The wave speeds used here are the compressional (P) wave speeds for water and steel. There is also a shear wave speed for the steel and a corresponding Snell’s law for the conversion of compressional waves in the water to shear waves in the steel.

M.2 Write a MATLAB script that generates 200 refracted angles ranging from 0 to 75 degrees and uses the function from problem 1 to generate the corresponding incident angles. The script should then plot the resulting incident angles versus the refracted angles.

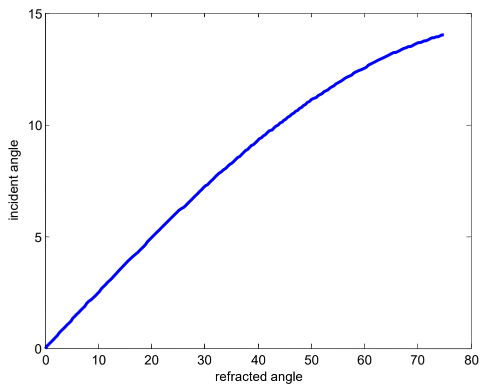


M.2 Snell's law script

```
% snells_law_script
ang_out =linspace(0, 75, 200);
ang_in = snells_law( ang_out, 1480, 5900);
plot(ang_out, ang_in)
xlabel('refracted angle')
ylabel('incident angle')
```

execute script

```
>> snells_law_script
```



Here is the script which uses the snells_law function of problem M.1 to solve and plot the incident angle versus refracted angle for a water-steel interface.

A.1 (a). Write a MATLAB function, spectrum1, which computes the positive frequency components of a signal given by

$$V(f) = A\sqrt{\pi}\exp\left[-4\pi^2 A^2 (f - f_c)^2\right],$$

where A is given in terms of the -6 *dB* bandwidth, *bw*, as $A = \sqrt{\ln 2}/(\pi bw)$. Spectrum1 should return sampled values of V for a set of sampled frequencies, f (in MHz), and a specified center frequency, fc , (in MHz), and bandwidth, bw, (in MHz), i.e. we should have for the MATLAB function call:
c

```
>> V = spectrum1(f, fc, bw)
```

Show that your function is working by evaluating the spectrum for 512 frequencies ranging from zero to 100 MHz with fc = 5 MHz, bw = 1 MHz. Generate the frequencies with the function s_space (see Appendix G), i.e. evaluate

```
>> f = s_space(0, 100, 512);
```

Plot V over a range of frequencies 0 – 10 MHz (approximately).

A.1 (b). Use the MATLAB function IFourierT to obtain a sampled time domain function, $v(t)$, from the sampled spectrum computed in problem A.1(a). Plot $v(t)$ by first generating a set of 512 time domain values with the function s_space, i.e.

```
>> t  = s_space(0,  512*dt, 512);
```

where dt is the time interval between samples (which in this case is dt = 1/100). Then use the t_shift and c_shift functions given in Appendix G so that the sampled values of $v(t)$ shown are not split between the first and last half of the window. Show that your results agree with the analytical result:

$$v(t) = \cos(2\pi f_c t)\exp\left[-t^2 / 4A^2\right]$$

if we take into account the fact that we only used the positive frequency components.

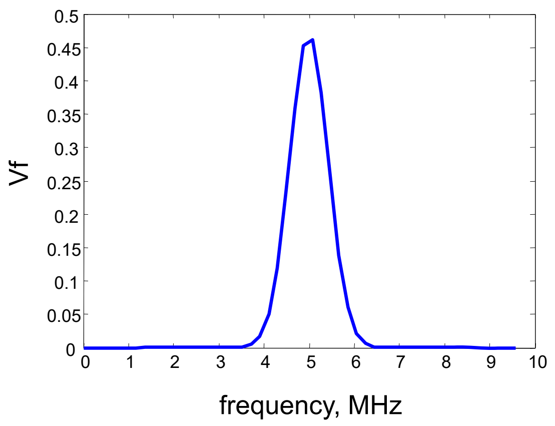
A.1 (c). Now apply the FourierT function to the sampled $v(t)$ obtained from A.1(b) and plot the magnitude of the resulting spectrum $V(f)$. How is this $V(f)$ different from the one you started with?

A.1 (d). If you look carefully at your plot of $V(f)$ in A.1 (a) you should see that the sampling interval, Δf, in the frequency domain is not quite small enough to give an accurate representation of the Gaussian function. Take the $v(t)$ signal obtained in A.1 (b), shift it so that all the values in the second half of the time domain window are zero, and then append 512 zeros to that signal. Apply FourierT to this longer signal to obtain the spectrum. Show that this process, which is called *zero padding*, improves our resolution in the frequency domain. Note that zero padding does not affect the sampling frequency.

Problem A.1 (a)

```
>> f = s_space(0, 100, 512);
>> dt = 1/100;
>> Vf = spectrum1(f, 5, 1);
>> plot(f(1:50), Vf(1:50))
>> xlabel( 'frequency, MHz')
>> ylabel( ‘ Vf ‘ )
```

Can also write plot(f(f<10), abs(Vf(f<10)))

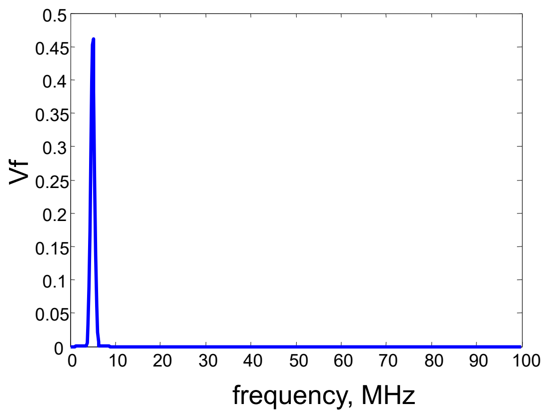


```
function V = spectrum1(f, fc, bw)
A = sqrt(log(2))/(pi*bw);
V = A*sqrt(pi)*exp(-4*(pi^2)*(A^2)*(f -fc).^2);
```

Here is the Matlab spectrum1 function. Note that the spectrum ranges from 0 to 100 MHz but we want to only plot 0 to 10 MHz. Two ways of doing the plot are shown, either by enumerating the indices explicitly or getting those indices through a logical statement.

over the full frequency range

```
>> plot(f, Vf)
>> xlabel('frequency, MHz')
>> ylabel('Vf')
```

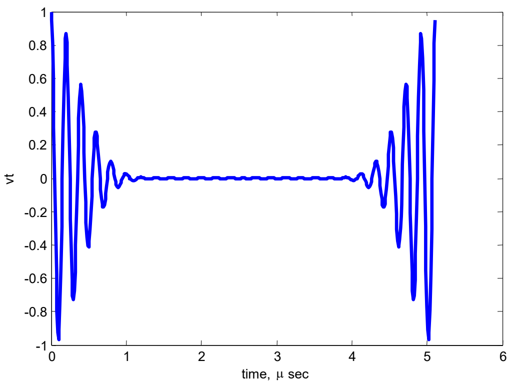


Here is the full spectrum. Note that there are no negative frequencies here, which would show up in the higher half of this full spectrum for the Fourier transform of a real signal.

Problem A.1 (b)

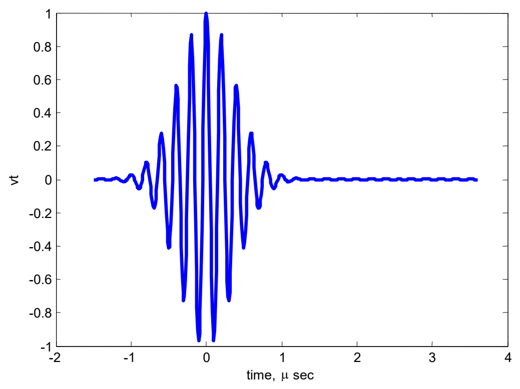
```
>> Vf(1)
ans =  3.7054e-031
>> Vf(1)=Vf(1)/2;
>> vt =2*real(IFourierT(Vf, dt));
>> t = s_space(0, 512*dt, 512);
>> plot(t, vt)
>> xlabel( 'time, \mu sec')
>> ylabel('vt')
```

Not really necessary here since dc value is so small



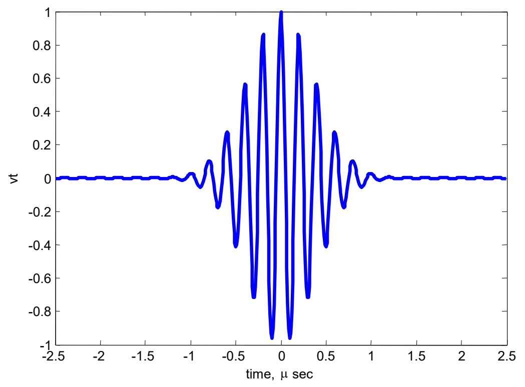
We do not need the negative frequencies to compute the real time domain function, as discussed in the notes, since we can just take twice the real part of the complex waveform generated. Note that the signal here is split in the time domain since it has values for negative times, which appear in the upper half of the time domain signal just as the negative frequencies will be in the upper half of the frequency spectrum.

```
>> plot(t_shift(t, 150), c_shift(vt,150))
>> xlabel( 'time, \mu sec')
>> ylabel('vt')
```



We can use the c_shift and t_shift functions to eliminate this splitting and retrieve the time domain function in a more recognizable form.

```
>> t2 = s_space(-2.5, 2.5, 512);
>> vt2 =time1(t2, 5, 1);
>> plot(t2, vt2)
>> xlabel( 'time, \mu sec')
>> ylabel('vt')
```



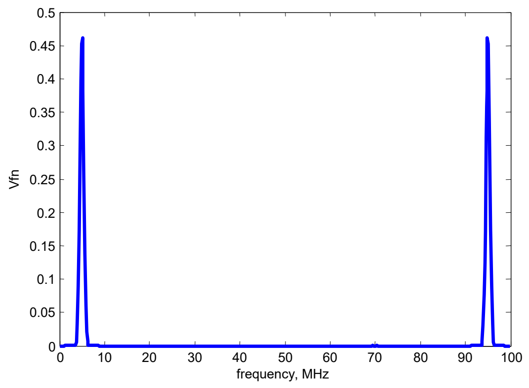
```
function v = time1(t, fc, bw)
A =sqrt(log(2))/(pi*bw);
v = cos(2*pi*fc*t).*exp(-t.^2./(4*A^2));
```

Here is the analytical form of the time domain function, time1, which corresponds to spectrum1. It agrees with our FFT results.

Problem A.1 (c)

original time signal coming from doing inverse FFT of spectrum1

```
>> Vfn = FourierT(vt, dt);
>> plot(f, abs(Vfn))
>> xlabel('frequency, MHz')
>> ylabel( 'Vfn' )
```

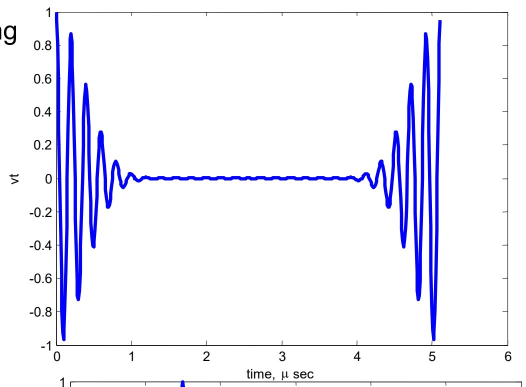


Now we have both positive and negative frequencies

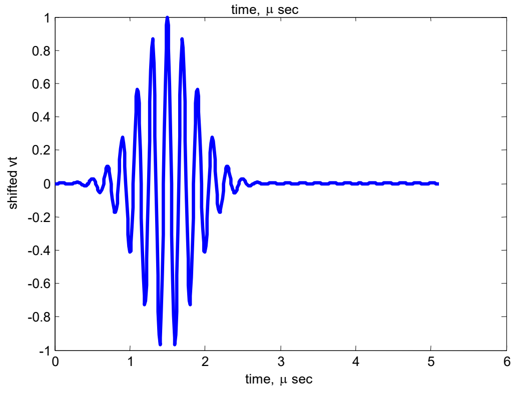
We see that if we compute the spectrum of the real time domain signal, we do get both positive and negative frequencies.

Problem A.1 (d) zero padding

```
>> plot(t, vt)
>> xlabel( 'time, \mu sec')
>> ylabel('vt')
```

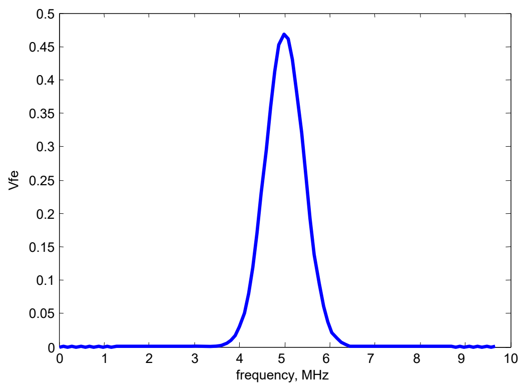


```
>> vts =c_shift(vt, 150);

>> plot(t, vts)

>> xlabel( 'time, \mu sec')

>> ylabel('shifted vt')
```



Here is the original time domain function and the one that was shifted with c_shift. If you look at spectrum1 you can see our original sampling gave a jagged function, which we can improve with zero padding.

```
>> vte =[vts, zeros(1,512)];
>> Vfe = FourierT(vte, dt);
>> fe =s_space(0, 100, 1024);
>> plot(fe(1:100), abs(Vfe(1:100)))
>> ylabel('Vfe')
>> xlabel('frequency, MHz')
```



some improvement here but we could add more zeros for a smother function

Here we pad the time signal with an additional 512 zeros and compute the spectrum on the finer frequency scale. The spectrum looks smoother.

2. The Hilbert transform, $\mathbb{H}\left[f(t)\right]$, of a function $f(t)$ is defined as

$$\mathbb{H}\left[f(t)\right]=\frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{f(\tau)d\tau}{\tau-t}.$$

When a plane traveling wave of the form $f(t-x/c)$ is reflected from a surface beyond a critical angle, as discussed in Appendix D, the Hilbert transform of the function $f(t)$ appears in the reflected wave causing the reflected waveform to be distorted from the incident wave. It can be shown [A.2] that if $F(\omega)$ is the Fourier transform of $f(t)$ then the Fourier transform of the Hilbert transform of $f(t)$, $\tilde{\mathbb{H}}(\omega)$, is given by $\tilde{\mathbb{H}}(\omega)=-i\,\mathrm{sgn}(\omega)F(\omega)$ where

$$\mathrm{sgn}(\omega)=\begin{cases}+1 & \omega>0\\ -1 & \omega<0\end{cases}.$$

We can use this fact in conjunction with the Fast Fourier transform as a convenient way to compute the Hilbert transform of a function. To see this consider the follow example:

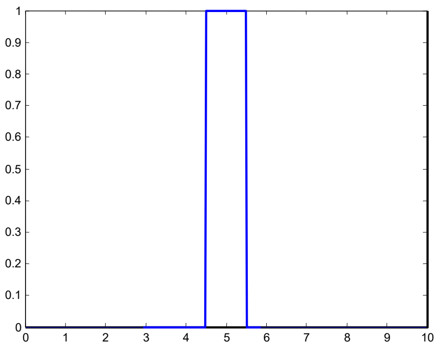
In MATLAB, define a sampled time axis consisting of 1024 points going from t = 0 to t = 10 μsec. Over this time interval define a sampled function $f(t)$ that has unit amplitude for $4.5<t<5.5$ μsec and is zero otherwise. Use FourierT to calculate the Fourier transform, $F(\omega)$, of $f(t)$. Then use the relationship $\tilde{\mathbb{H}}(\omega)=-i\,\mathrm{sgn}(\omega)F(\omega)$ to find the Fourier transform of the Hilbert transform of $f(t)$ and compute the Hilbert transform itself by performing an inverse Fourier transform on this result with IFourierT. [Note that if you use only the positive frequency values to compute the inverse, then we only need to multiply those values by $-i$]. Plot your results versus time. In this case the Hilbert transform of $f(t)$ can be obtained analytically [Fundamentals]. It is

$$\mathbb{H}\left[f(t)\right]=\frac{1}{\pi}\ln\left|\frac{t-5.5}{t-4.5}\right|.$$

Using a different plotting symbol, plot this function also on the same graph. How do your results compare?

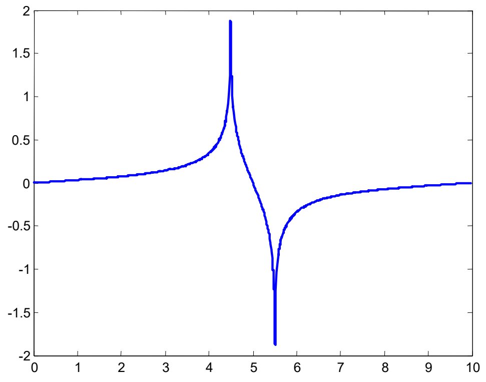
Problem A.2

```
>> t=s_space(0, 10, 1024);
>> dt=t(2) - t(1);
>> v = (t>4.5 & t < 5.5) + 0.5*( t ==4.5) + 0.5*(t == 5.5);
>> plot(t, v)
```



Consider this box-like function of time.

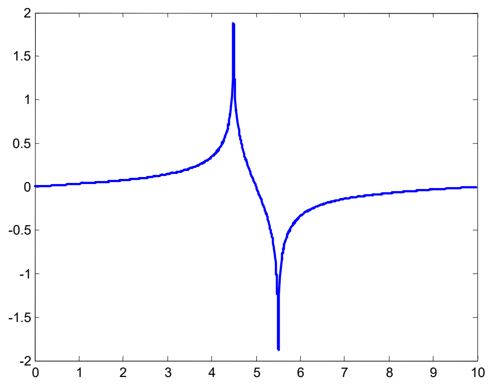
```
>> Vf = FourierT(v, dt);
>> f = s_space(0, 1/dt, 1024);
>> fs = 1/dt;
>> Vfp = Vf.*( f < fs/2);        ←———    zero out negative frequencies
>> VH = -i*Vfp;    ←———    multiply positive frequencies by -i
>> VH(1) = VH(1)/2;    ←——————    not really necessary
                                      since VH(1) is purely
>> vH = 2*real(IFourierT(VH, dt));    imaginary so taking
                                      real part in next step
>> plot(t, vH)                        will eliminate it
```



We can calculate the spectrum of this box function. If we zero out the negative frequencies and multiply the positive frequencies by – i, then the Hilbert transform of the box function will be twice the real part of the resulting complex time domain signal, as shown here.

alternatively, we could have calculated

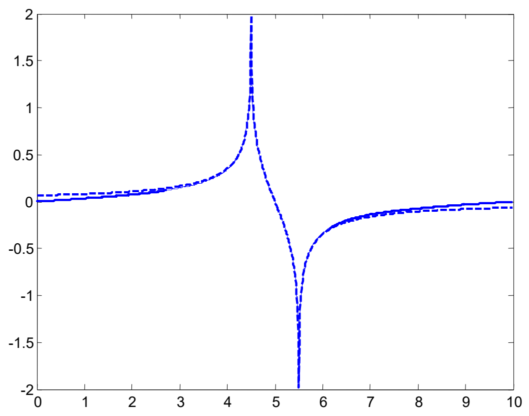
```
>> VfH = -i*Vf.*(f<fs/2) +i*Vf.*(f >= fs/2);
>> vHt= IFourierT(VfH, dt);
>> plot(t, real(vHt))
```



Alternatively, we could have kept both positive and negative frequencies but then we have to multiply the positive and negative frequency values differently, as shown here and as discussed in the problem statement.

compare with analytical result

```
>> hold on
>> vH2 = log(abs((t-5.5)./(t-4.5)))/pi;
>> plot(t, vH2, '--')
```



Here, vH2 is the analytical Hilbert transform of the boxlike function originally given. It compares favorably with our numerical calculations.

Another way to get the Hilbert transform is: (see page 124 in the notes but note there is a sign error on the Hilbert transform part on that page)

signal/2 + i* (Hilbert Trans)/2 $\leftarrow$ IFourierT(Positive frequency components)

$\uparrow$ in the vector Vfp

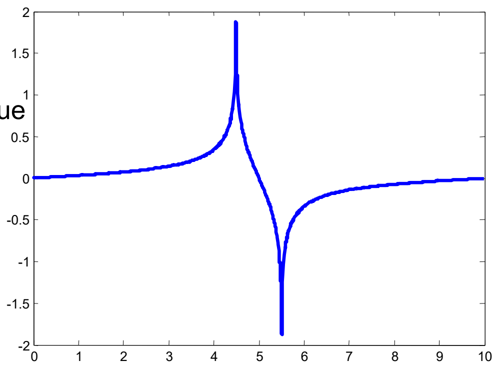
$\Uparrow$

take twice imaginary part

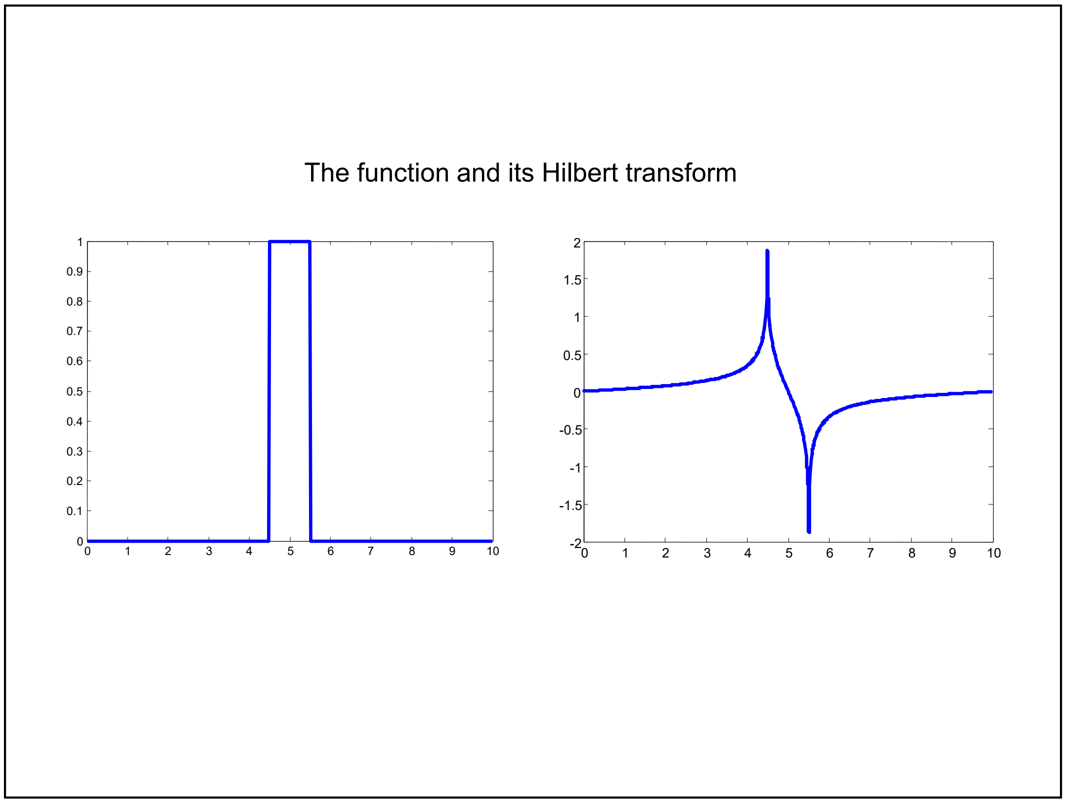
```
>> Vfp(1)=Vfp(1)/2;
```

$\leftarrow$ necessary since box function has large dc value

```
>> vH3 =2*imag(IFourierT(Vfp, dt));
>> plot(t, vH3)
```



Another way to get the Hilbert transform is to recognize it shows up as twice the imaginary part of the complex-valued time domain function generated when we do an inverse FFT of just the positive frequency components of the box function. The expression involving the Hilbert transform on page 124 in the notes has a sign error, as noted above. You should know that some authors define the Hilbert transform as the negative of the definition we are using here, so such sign errors can easily creep into an analysis.



Here is the original box function and its Hilbert transform.

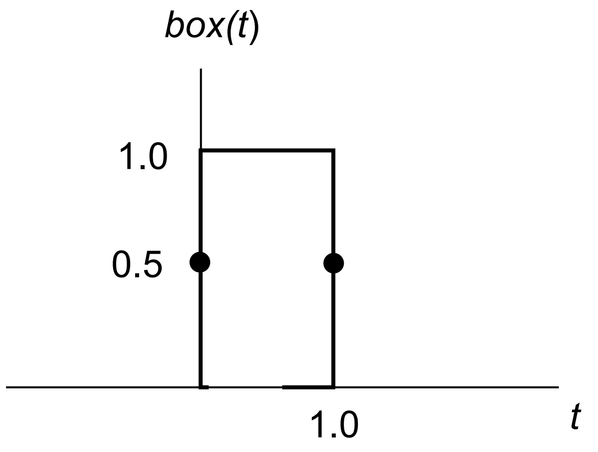
S.0. In MATLAB write a function y = box(t); that returns the the "box" function given below as a function of the time, t. [Hint: use logical vectors]

Use the function s_space to generate a vector of 512 time values over the interval [0, 10.24) and evaluate the output of the box function with these values and plot the result to verify that the box function is working properly.

Use the MATLAB fast Fourier transform function yf =FourierT(y, dt); to determine the complex frequency spectrum of this signal. Set up a set of sampled frequency values using f =s_space(0, 1/dt, 512); and plot the magnitude and phase of this spectrum. From the magnitude plot does it appear there is aliasing present here? Do these plots look like the plots shown in Fig. A.2 on the next slide (which are based on analytical values)? What is different?

Use the function yi = IFourierT(yf, dt); to invert the results back into the time domain. Plot the results versus time. Why do you get a warning message? Verify that you obtain the function box(t), including the values of 0.5 at t = 0 and t = 1.

$$box(t) = \begin{cases} 0.5 & t = 0 \\ 1.0 & 0 < t < 1 \\ 0.5 & t = 1 \\ 0 & otherwise \end{cases}$$



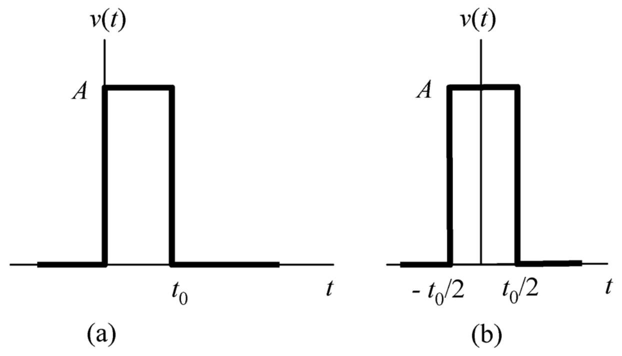


Fig. A.1. **(a)** An example of a simple “box” time domain function and **(b)** the same function shifted in time.

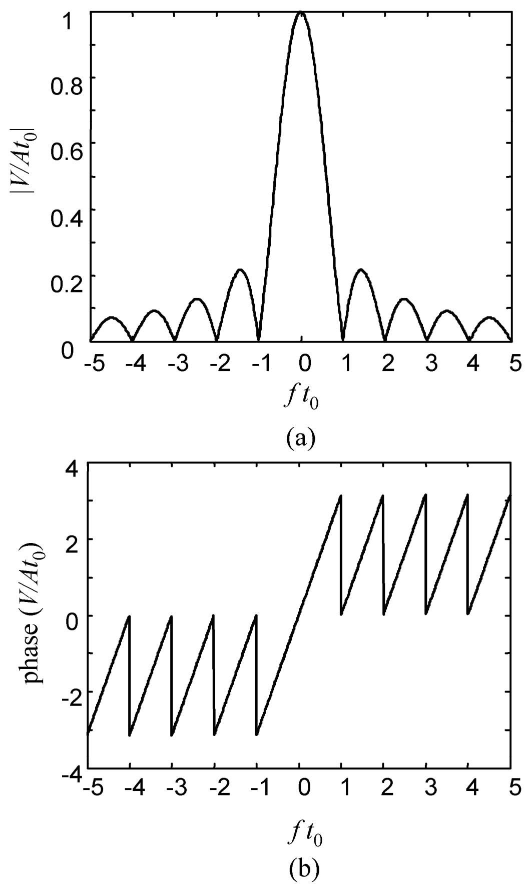


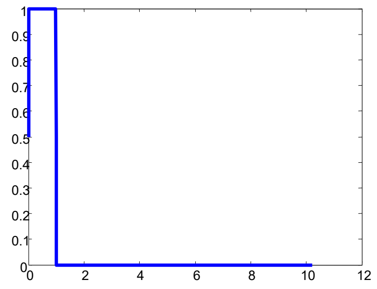
Fig. A.2. **(a)** The magnitude of the Fourier transform of the "box" function of Fig. A.1 (a) and **(b)** the phase of this transform.

Prob. S.0

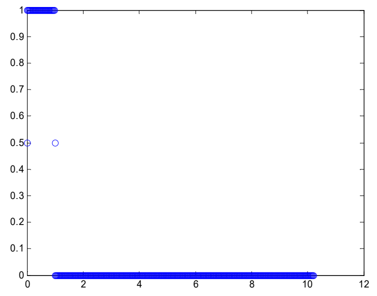
```
function y = box(t)
y = (t > 0 & t < 1.0) + 0.5*(t == 0)+ 0.5*(t ==1);
```

alternatively, we could use
`y=(t > 0 & t <1.0)+0.5*(t == 0 | t ==1);`

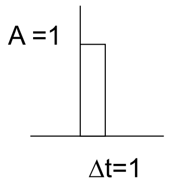
```
>> t=s_space(0, 10.24, 512);
>> y=box(t);
>> plot(t, y)
```



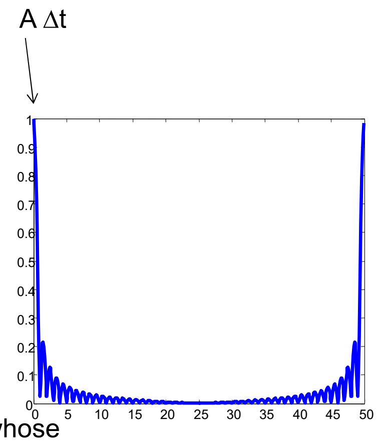
```
>> plot(t, y, 'o')
```



We can use logical vectors to easily generate the box function. If we plot with lines we cannot see the ½ values but if we plot points with open circles they are clearly visible.



```
>> dt = t(2) - t(1);
>> yf =FourierT(y, dt);
>> f=s_space(0, 1/dt, 512);
>> plot(f, abs(yf))
>> yf(1)
ans = 1     ←    A Δt
```



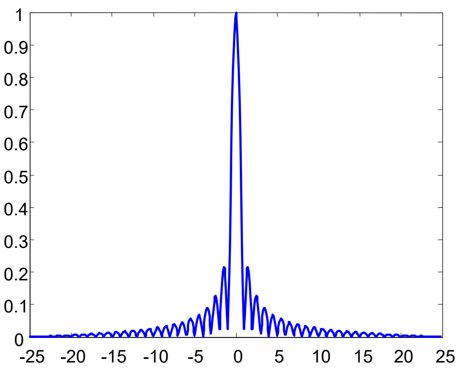
If we don't use 0.5 values, the answers contain errors as shown below:

define a box function whose values at 0 and 1 are both 1.0

```
>> y2 =(t>= 0 & t <= 1);
>> yf 2= FourierT(y2, dt);
>> yf2(1)

ans =

   1.0200
```

If we calculate the spectrum of the box function it appears the frequency components are small at the Nyquist frequency so aliasing should not be a problem. Note that the zero frequency value is 1.0, as it should be from the analytical values. If we make a box function whose values at t = 0 and t= 1 are both one, then when we compute the spectrum the zero frequency value is incorrect. This is because the discrete Fourier transform, like a Fourier series, converges to the mean value of the function (which in this case is ½) at any discontinuities (see any book which discusses this aspect of Fourier series).

```
plot(t_shift(f, 256), c_shift(abs(yf), 256))
```

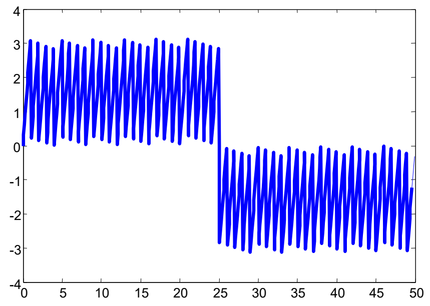


There is also a built-in
function fftshift that will do the same shifting:

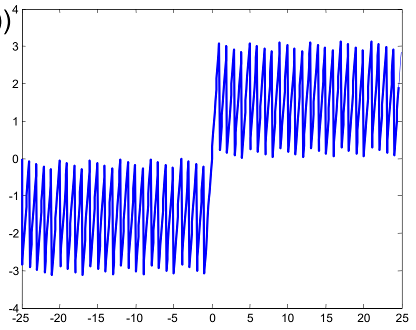
```
plot(t_shift(f, 256), fftshift(abs(yf)))
```

We can use the t_shift and c_shift functions here to put the positive and negative frequencies in their normal positions. Note: t_shift here is being used on frequency values, not time values, but it works the same in either case.

```
>> plot(f, angle(yf))
```



```
plot(t_shift(f, 256), c_shift(angle(yf), 256))
```

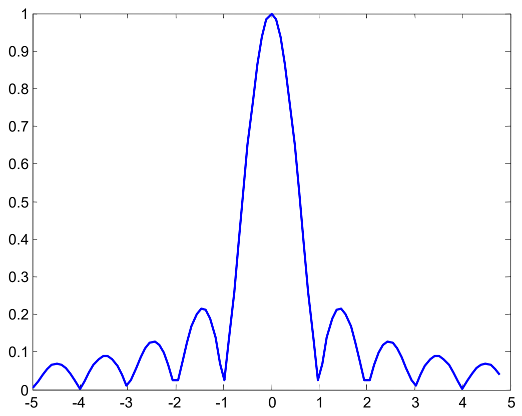


If we look at the phase, we can also use c_shift and t_shift to put the positive and negative values in their normal places.

On a finer scale this looks just like Fig. A.2 for the magnitude

```
>> fsh = t_shift(f, 256);
>> absh =c_shift(abs(yf), 256);
>> plot(fsh(256-50:256+50), absh(256-50:256+50))
```

Or use plot(fsh(fsh > -5 & fsh <5), absh(fsh > -5 & fsh < 5))

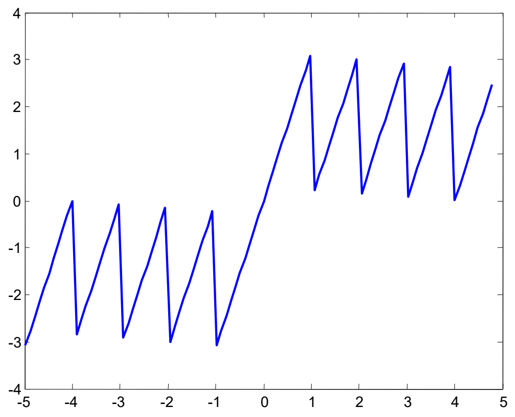


The spectrum agrees with the analytical values in Fig. A.2 if we plot on the same scale.

And the phase

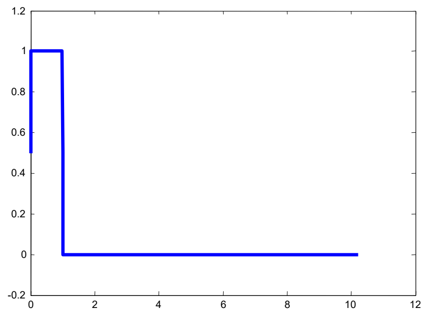
```
>> phsh =c_shift(angle(yf), 256);
>> plot(fsh(256-50:256+50), phsh(256-50:256+50))
```

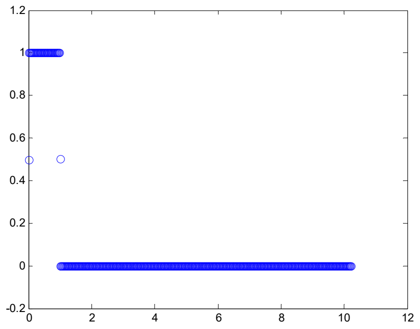
Or use plot(fsh(fsh>-5 & fsh < 5), phsh(fsh> -5 & fsh < 5))



Similarly, the phase looks the same as the analytical results on the same scale.

```
>> yi=IFourierT(yf, dt);
>> plot(t, yi)
Warning: Imaginary parts of complex X
and/or Y arguments ignored.
>> plot(t, real(yi))
>> yi(1)

ans = 0.5000 - 0.0000i
>> yi(51)
ans = 0.5000 + 0.0000i
```





```
plot(t, real(yi), 'o')
```

If we do an inverse FFT we do recover the correct box function values, including the ½ values at the discontinuities. The Matlab warning exists when we try to plot since there are very small imaginary values present due to numerical roundoff errors, which are eliminated by taking the real part. We can see these small imaginary values are present when we examine the values at the discontinuities.

S.1 Consider a time domain signal given by

$$y(t) = \begin{cases} \cos(3\pi t) & -0.5 < t < 0.5 \\ 0 & otherwise \end{cases}$$

where t is measured in μsec

1. Evaluate and plot this signal in MATLAB from t = -4 μsec to t = +4 μsec. Obtain the Fourier transform of this function analytically (Hint: the function is even in t so what does this imply about its Fourier transform?) and plot it in MATLAB from zero to 14MHz .

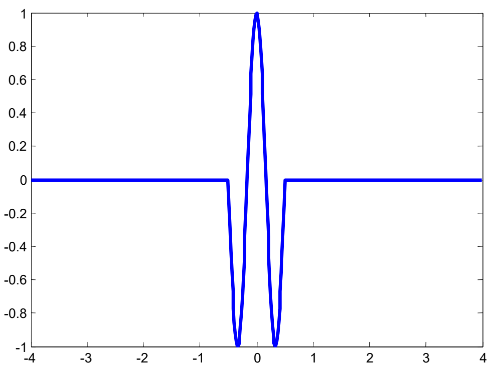
(a) What is the value of the Fourier transform at zero frequency ? How is this value related to the properties of y(t)?

(b) At what frequency is the Fourier transform a maximum? How is this frequency related to the properties of y(t)?

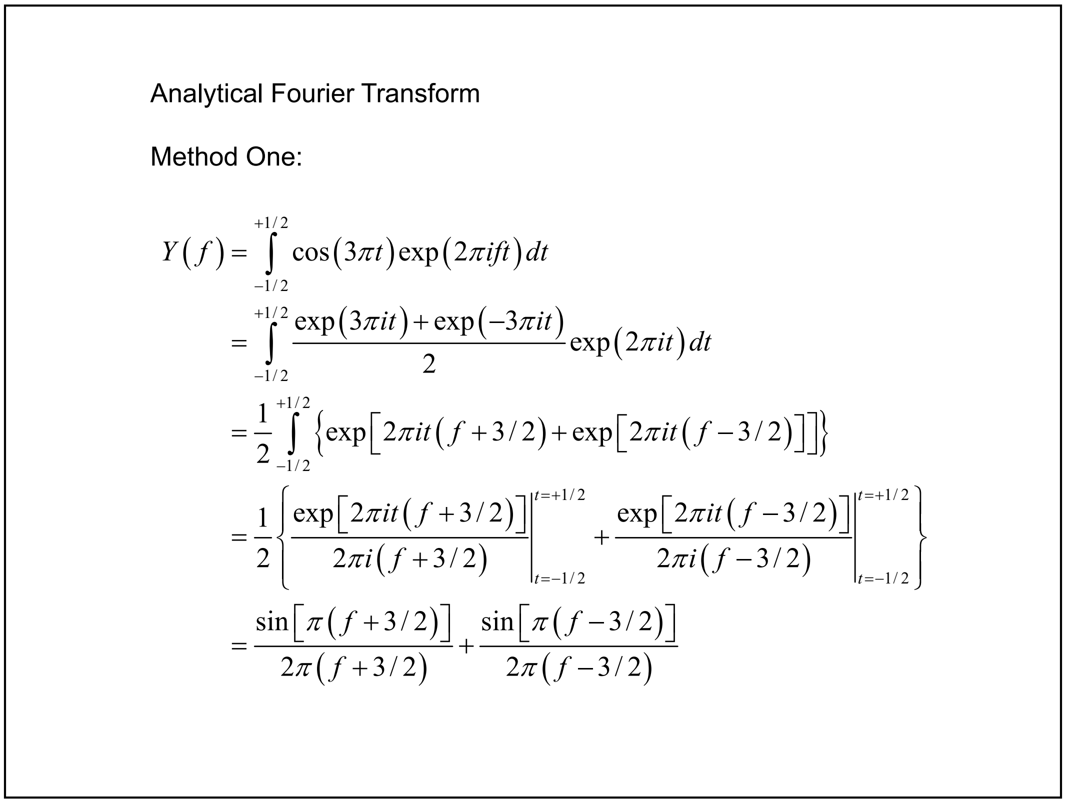
2. From the samples of y(t) over the time interval (-4, 4) obtain the Fourier transform numerically with the MATLAB FFT function FourierT. Plot the magnitude of this function and show that it agrees with the magnitude of your analytical result. Use IFourierT and show that you recover the y(t) function.

Prob. S.1

```
>> t=s_space(-4, 4, 512);
>> y= cos(3*pi*t).*(t>= -0.5 & t <=0.5);
>> plot(t, y)
```



Here is the Matlab generated time domain signal.

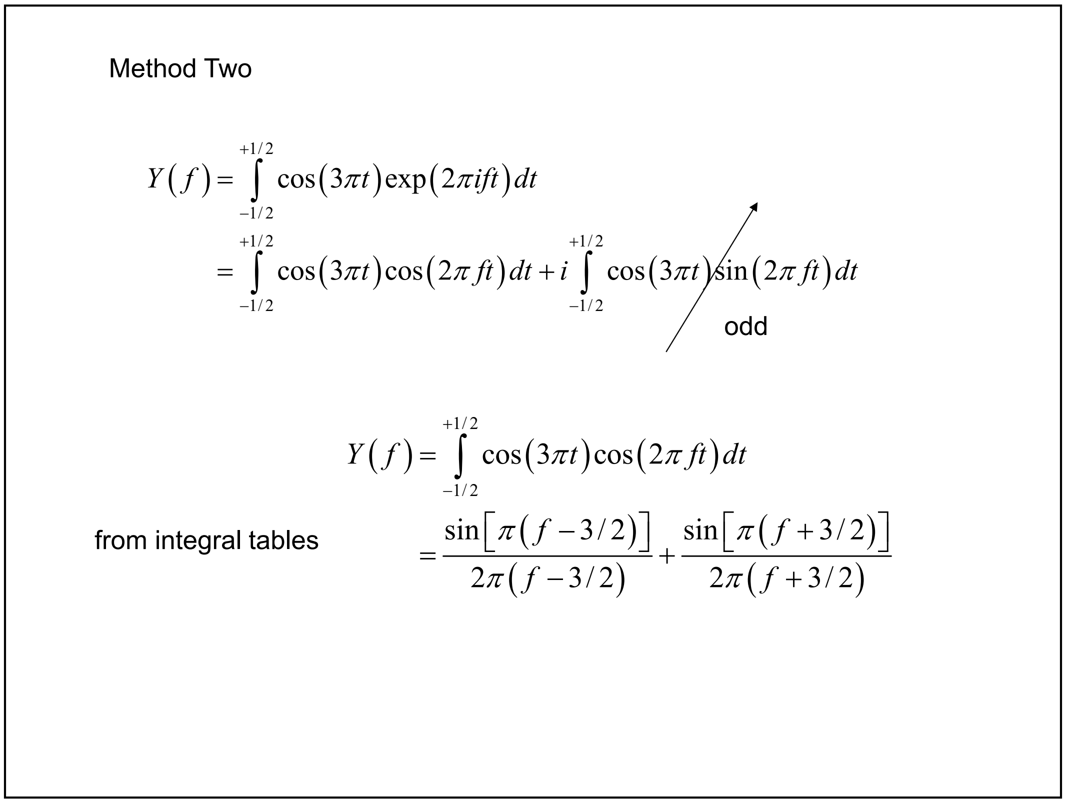


Analytical Fourier Transform

Method One:

$$
\begin{aligned}
Y(f) &= \int_{-1/2}^{+1/2} \cos(3\pi t)\exp(2\pi i f t)\,dt \\
&= \int_{-1/2}^{+1/2} \frac{\exp(3\pi i t)+\exp(-3\pi i t)}{2}\exp(2\pi i t)\,dt \\
&= \frac{1}{2}\int_{-1/2}^{+1/2} \left\{\exp\left[2\pi i t\left(f+3/2\right)+\exp\left[2\pi i t\left(f-3/2\right)\right]\right]\right\} \\
&= \frac{1}{2}\left\{ \left.\frac{\exp\left[2\pi i t\left(f+3/2\right)\right]}{2\pi i\left(f+3/2\right)}\right|_{t=-1/2}^{t=+1/2} + \left.\frac{\exp\left[2\pi i t\left(f-3/2\right)\right]}{2\pi i\left(f-3/2\right)}\right|_{t=-1/2}^{t=+1/2} \right\} \\
&= \frac{\sin\left[\pi\left(f+3/2\right)\right]}{2\pi\left(f+3/2\right)} + \frac{\sin\left[\pi\left(f-3/2\right)\right]}{2\pi\left(f-3/2\right)}
\end{aligned}
$$

In this case we can do the Fourier transform analytically since we can write the sine and cosine functions in terms of exponentials that can be easily integrated.



Method Two

$$Y(f) = \int_{-1/2}^{+1/2} \cos(3\pi t)\exp(2\pi i f t)\,dt$$

$$= \int_{-1/2}^{+1/2} \cos(3\pi t)\cos(2\pi f t)\,dt + i\cancel{\int_{-1/2}^{+1/2} \cos(3\pi t)\sin(2\pi f t)\,dt}$$

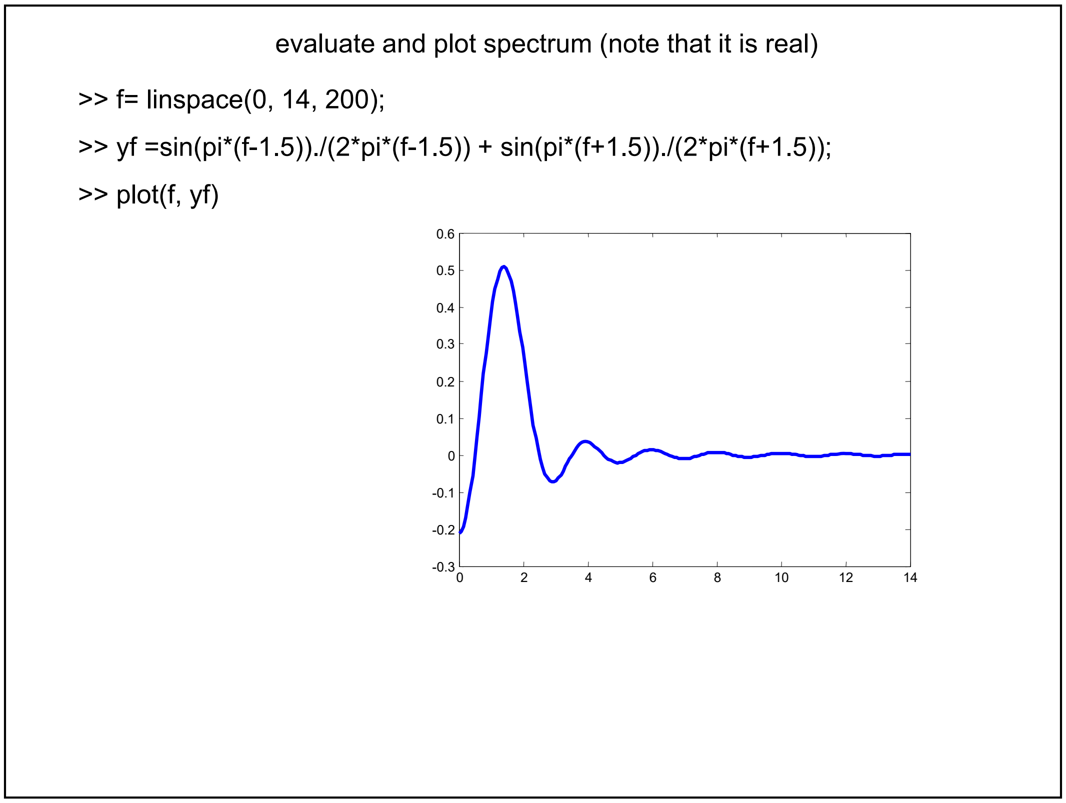
odd

$$Y(f) = \int_{-1/2}^{+1/2} \cos(3\pi t)\cos(2\pi f t)\,dt$$

from integral tables

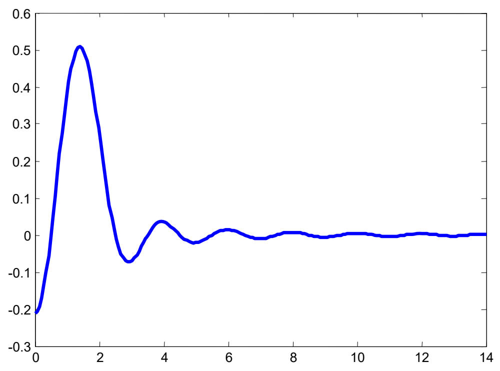
$$= \frac{\sin\left[\pi\left(f-3/2\right)\right]}{2\pi\left(f-3/2\right)} + \frac{\sin\left[\pi\left(f+3/2\right)\right]}{2\pi\left(f+3/2\right)}$$

Alternatively, we can find the Fourier transform integrals in this case in tables.



evaluate and plot spectrum (note that it is real)

```
>> f= linspace(0, 14, 200);
>> yf =sin(pi*(f-1.5))./(2*pi*(f-1.5)) + sin(pi*(f+1.5))./(2*pi*(f+1.5));
>> plot(f, yf)
```

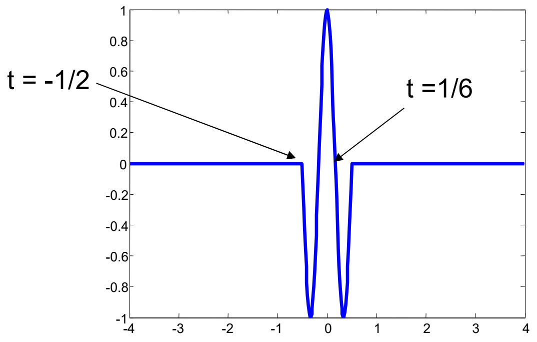


Here is the plot of the Fourier transform, which in this case is real (why?).

(a) $$Y(0) = -\frac{2}{3\pi} = \int_{-1/2}^{+1/2} \cos(3\pi t)\,dt$$ = area under the signal =-0.2122

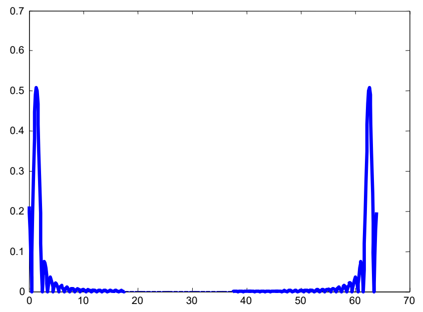
(b) Y(f) is max at approximately f= 3/2 MHz. Why?

The signal cos(3πt) is oscillating, and goes through a complete cycle from t = -1/2 to t= 1/6, so the period, T, is T= 4/6 = 2/3 μ sec
Since the frequency of this oscillation is f = 1/T the predominate frequency should be f = 3/2 MHz approximately



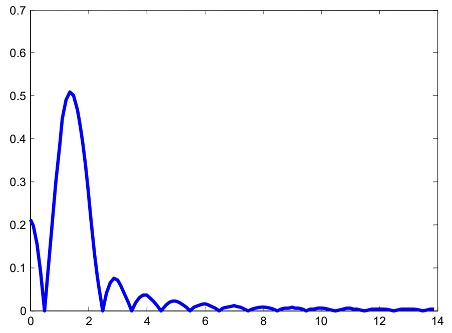
From the analytical transform expression we see the zero frequency value is just the area under the time domain signal. The maximum of the Fourier transform we see is just approximately the dominant frequency at which the time domain signal is oscillating.

```
>> dt = t(2) –t(1);
>> f = s_space(0, 1/dt, 512);
>>yf2 = FourierT(y ,dt);
>> plot(f, abs(yf2))

>> 1/dt
ans = 64      max frequency
```



To get a plot from 0 to 14 MHz

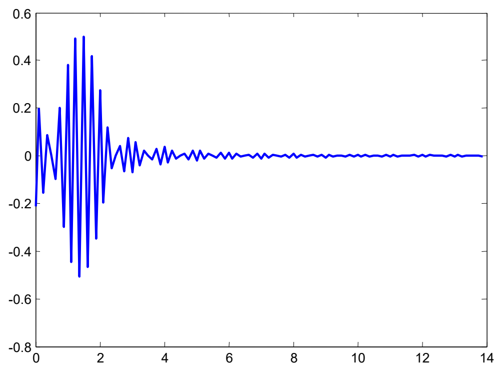
```
>> plot(f(f<14), abs(yf2(f<14)))
```



If we compute the frequency response with the FFT we see the same magnitude as found from the analytical function.

If we plot yf2 itself (not its magnitude), we see:

```
>> plot(f(f<14), yf2(f<14))
```



This does not look like our Y(f) at all. Why?

However, here the Fourier transform is not real so if we try to plot it we do not recover the same result as with the analytical result.

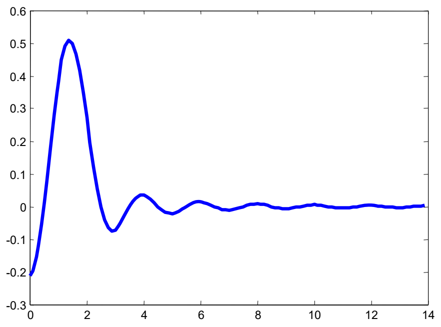
The reason is that we started sampling at t = -4 μsec, but in the FFT this initial time is taken to be t = 0, so as far as the transform is concerned our signal looks like it is centered at t = 4 μsec, not t = 0. This shifting introduces a phase term in the Fourier transform

$$y(t) \leftrightarrow Y(f)$$
$$y(t-t_0) \leftrightarrow \exp(2\pi i f t_0) Y(f)$$

so for our case $$Y(f)\Big|_{FFT} = \exp(8\pi i f) Y(f)\Big|_{analytical}$$

we can get rid of this factor and plot again:

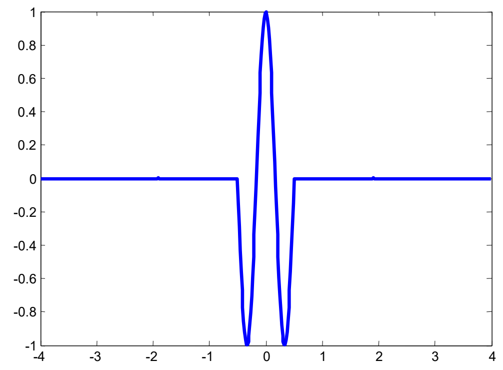
```
>> yf3=exp(-8*pi*i*f).*yf2;

>> plot(f(f<14), yf3(f<14))

Warning: Imaginary parts of complex X
and/or Y arguments ignored.
```



There are values for the time domain signal for negative times, so just as in the frequency domain, these negative time values should appear in the upper half of the time domain window over which we sample the signal, i.e. the signal should look "split" in order to properly represent a periodic function, as required by the discrete Fourier transform. We did not do that splitting so our signal appears centered at t = 4 microseconds, which introduces a phase term in the Fourier transform. If we eliminate that phase and then plot we do see the proper frequency response (with very small imaginary parts from numerical roundoff, which can be ignored).

Now, use the inverse FFT to recover the y(t):

```
>> yi =IFourierT(yf2, dt);
>> plot(t, yi)
Warning: Imaginary parts of complex X
and/or Y arguments ignored.
```



We do recover the original time domain signal from the FFT when plotted on the original time axis.

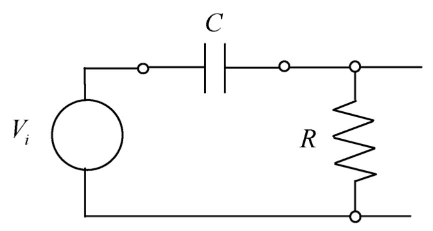
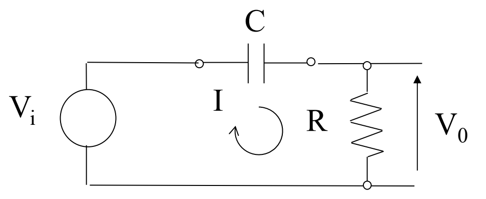


Fig. B.14. An example circuit.

B.2. For the circuit shown in Fig. B.14 obtain the Thévenin equivalent source and impedance in terms of the given circuit elements.

Problem B.2

find the Thevenin equivalent circuit
for this example



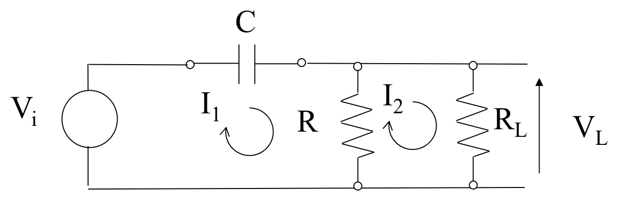
Open-circuit
voltage

$$V_i - V_0 = \frac{I}{-i\omega C}$$

$$V_0 = RI$$

Eliminating I gives $V_0 = \frac{-i\omega RC}{1 - i\omega RC} V_i$

Using the circuit relations for the voltage across the capacitor and the resistor, we can eliminate the current and express the open-circuit voltage in terms of the known circuit parameters. This open circuit voltage is just the Thevenin equivalent source.



(1) $V_i - V_L = \dfrac{I_1}{-i\omega C}$

(2) $V_L = 2R\left(I_1 - I_2\right)$

(3) $V_L = R_L I_2$

From the second and third equations $I_1 = V_L\left(\dfrac{1}{2R} + \dfrac{1}{R_L}\right)$

Using this current in the first equation gives

If we put a load resistor at the output we can examine the voltage drops in the two "loops" of the circuit to get an expression for the voltage across the load resistor, as shown on the next page.

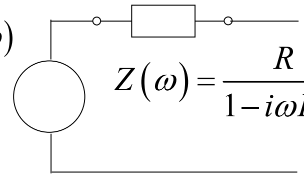
$$V_L = \frac{-i\omega RC}{\left(1 - i\omega RC + R/R_L\right)} V_i$$

The equivalent impedance is given by

$$Z = R_L\left(\frac{V_0}{V_L} - 1\right)$$ so we find

$$Z = R_L\left(\frac{R/R_L}{1 - i\omega RC}\right) = \frac{R}{1 - i\omega RC}$$

$$V_S(\omega) = \frac{-i\omega RC}{1 - i\omega RC} V_i(\omega)$$

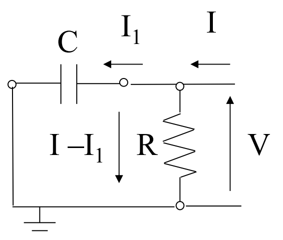


$$Z(\omega) = \frac{R}{1 - i\omega RC}$$

The equivalent impedance can be obtained from the voltage across the load and the open circuit voltage, thus yielding the Thevenin equivalent circuit.

alternate method

$$V = \frac{I_1}{-i\omega C} = R\left(I - I_1\right)$$



$$V = R\left(I + i\omega CV\right)$$

$$\Rightarrow \quad Z = \frac{V}{I} = \frac{R}{1 - i\omega RC}$$

Another method to get the equivalent impedance is to short out (i.e. remove) the voltage source and examine the ratio of the voltage at the output in this configuration to the current flowing into the circuit. This ratio is just the equivalent impedance.

2.1 The MATLAB function model_pulser takes as inputs an energy setting (energy = 1, 2, 3, 4), a damping setting (damping = 0, 5, 10), a resistance loading , RL, (in ohms) across the output terminals of the pulser and returns the sampled voltage, vt, across RL (in volts) and the sampled time values, t, (in μsec). The form of the calling sequence of this function is:

```
>> [t , vt] =model_pulser( energy, damping, RL);
```

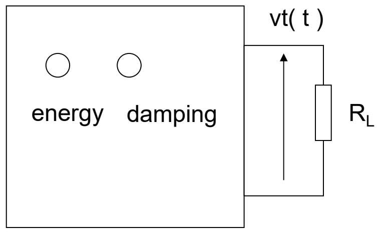
Use this model pulser at energy = 2, damping = 5 settings for both open circuit conditions (RL = inf) and a given load (RL = 250 ohms) to determine the Thévenin equivalent source voltage (in volts) and impedance (in ohms) of the pulser at these settings as functions of frequency over the range of frequencies from 0-20 MHz and plot the magnitude and phase of these functions over the same frequency range. Use the MATLAB unwrap function to eliminate any artificial jumps of in the phase plots. Example:

```
>> plot(f, unwrap(angle(Vf)))
```

Show and explain all the steps you used to obtain your answers.

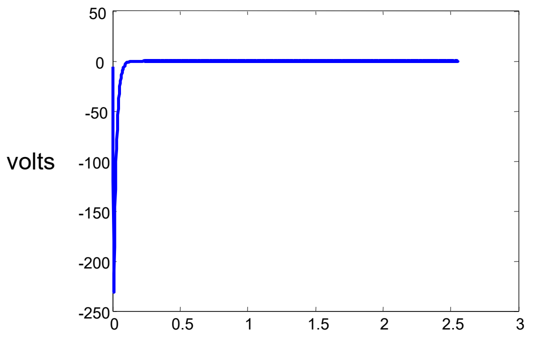
Problem 2.1

[t, vt] = model_pulser(energy, damping, RL)



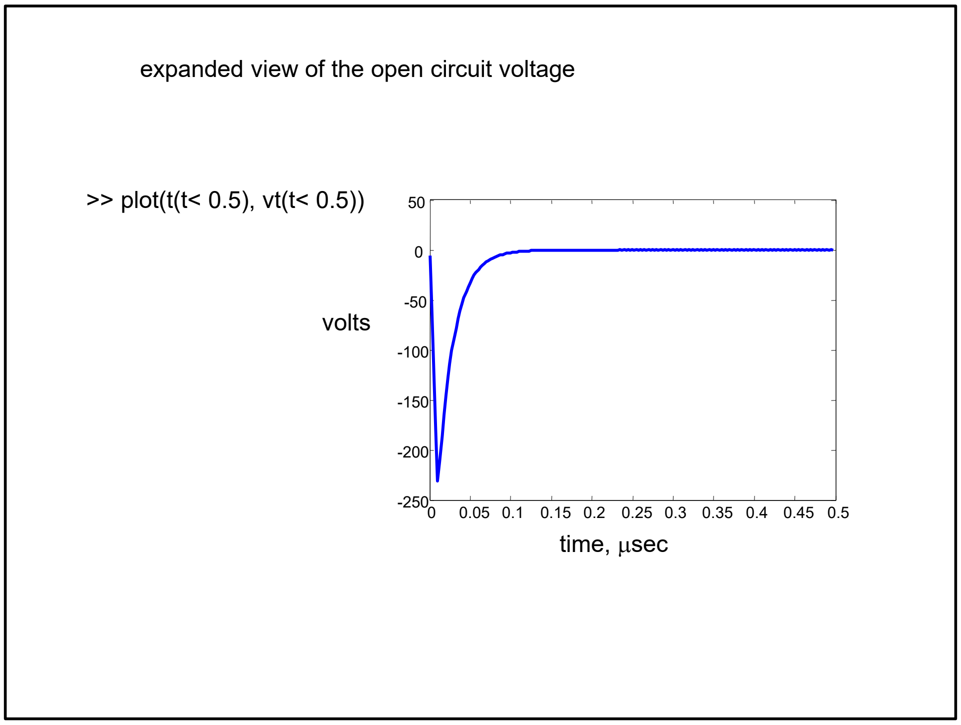
The Matlab function model_pulser simulates this simple pulser and load model.

```
>> [t, vt] = model_pulser(2,5,inf);
>> plot(t, vt)
>> size(t)

ans = 1        1024

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

dt =0.0025

>> fmax = 1/dt

fmax = 400.3914
```



open circuit voltage versus time in μsec

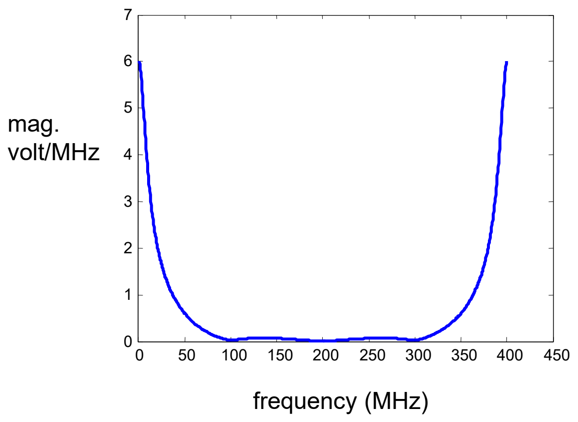
Here is the open circuit voltage of the model_pulser. We see there are 1024 time points with a time interval between sampling corresponding to a sampling frequency of about 400 MHz.


expanded view of the open circuit voltage

```
>> plot(t(t< 0.5), vt(t< 0.5))
```

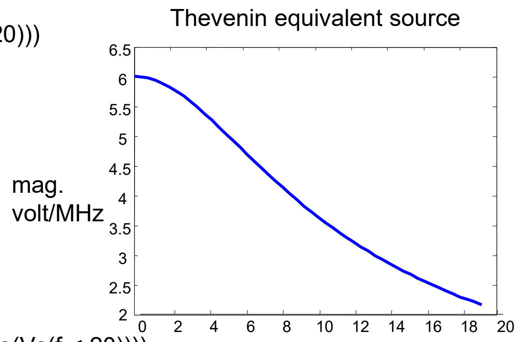
Here is an expanded view of the open circuit voltage so you can see the characteristic output of a “spike” pulser

```
>> Vs =FourierT(vt, dt);
>> f=s_space(0, fmax, 1024);
>> plot(f , abs(Vs))
```

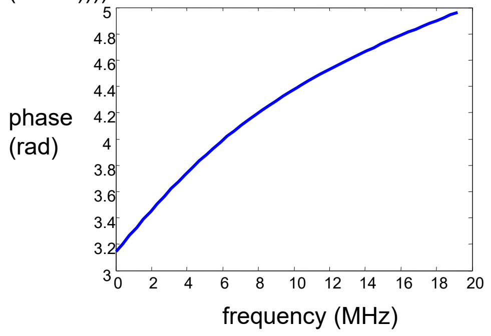


Here is the magnitude of the pulser open circuit response in the frequency domain. Since the magnitude is small at the Nyquist frequency our sampling frequency was likely sufficient.

```
>> plot(f(f< 20), abs(Vs(f<20)))
```

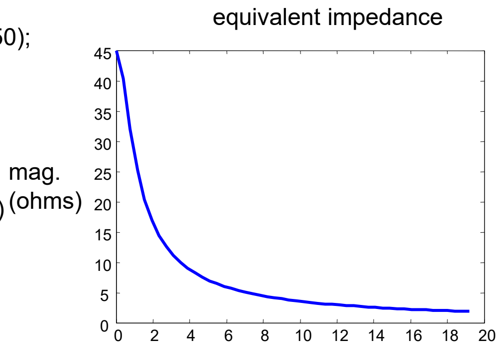


```
>> plot(f(f<20), unwrap(angle(Vs(f < 20))))
```

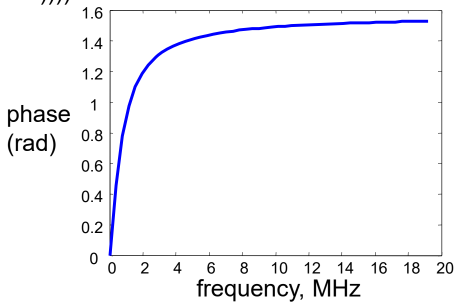


Here is the magnitude and phase of the open circuit voltage spectrum over a range of about 20 MHz.

```
>> [t, vl] = model_pulser(2,5,250);
>> Vlf =FourierT(vl, dt);
>> Z = 250*(Vs./Vlf -1);
Warning: Divide by zero.
>> plot(f(f < 20), abs(Z(f < 20)))
```



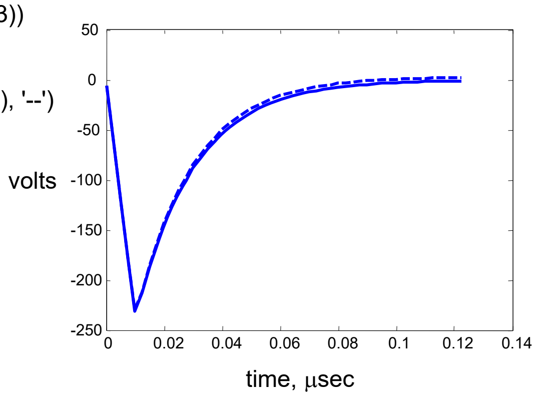
```
>> plot(f(f < 20), unwrap(angle(Z(f <20))))
```



Here is the output of model_pulser with a 250 ohm resistor at the output and the pulser impedance calculated according to the procedure outlined in the notes. One could avoid the Matlab warning by simply omitting the zero frequency calculation. Plotted are the magnitude and phase of the impedance.

Note that the waveform has changed very little from open-circuit to 250 ohm loading:

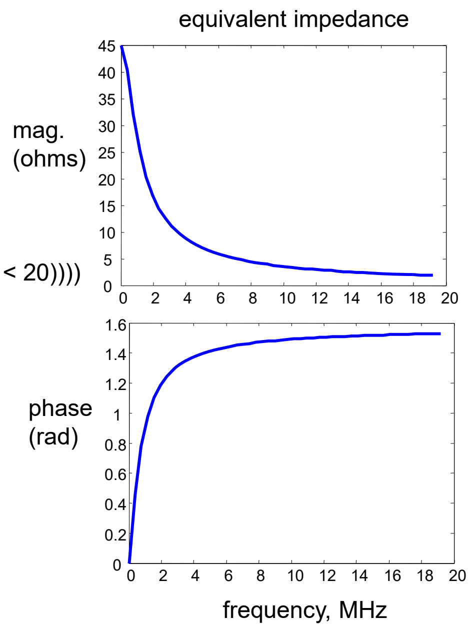
```
>> plot(t(t < .13),vt(t < .13))
>> hold on
>> plot(t(t< .13),vl(t < .13), '--')
>> hold off
```



Here is a detailed plot of the open circuit voltage and the output voltage under load, showing the small changes in the output waveform.

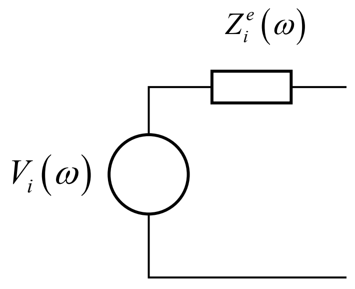
Check that results don’t depend on load resistance RL. Use 500 ohms instead

```
>> [t, vl2] = model_pulser(2,5,500);
>> Vlf2 =FourierT(vl2, dt);
>> Z2 = 500*(Vs./Vlf2 -1);
Warning: Divide by zero.
>> plot(f(f <20), abs(Z2(f <20)))
>> plot(f(f <20), unwrap(angle(Z2(f < 20))))
```



One can see that the calculated impedance does not depend on the load resistance.

The Thevenin equivalent pulser



Here is the Thevenin equivalent circuit to the model_pulser.

$$G(f) = \frac{O(f)}{I(f)}. \tag{C.23}$$

```
>> V = pulserVT(200, 0.05, 0.2, 12, t);
```

C.2 It is not physically possible to generate a delta function as the input of an LTI system to obtain its impulse response. However, it is possible to obtain the transfer function of an LTI system by deconvolution of a measured output with a known input as shown in Eq. (C.23). Consider a MATLAB function LTI_X that represents a "black box" LTI system. It can be evaluated in the form

```
>> O =LTI_X(I, dt)
```

Where I is a sampled time domain input (with sampling interval dt) and O is the time domain output. Use as an input for this LTI system the voltage output of the pulserVT function discussed in the notes, whose Matlab call is shown above, and where t is a vector containing 512 sampled times ranging from 0 to 5 μsec., and obtain the transfer function of this system as a function of frequency by deconvolution. Plot the magnitude and phase (in degrees) of this transfer function from zero to approximately 30 MHz. To obtain the impulse response function from this transfer function we would have to compute its inverse Fourier transform. Is that possible with this function?

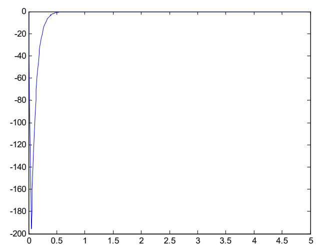
C.2

LTI System Transfer function by Deconvolution

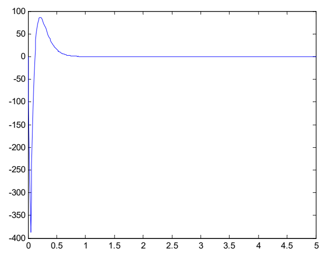
```
>> t = s_space(0,5, 512);
>> V = pulserVT(200, 0.05, 0.2, 12, t);
>> plot(t, V)
>> dt = t(2) - t(1);
>> O = LTI_X(V, dt);
>> plot(t, O)
>> 1/dt

ans =

  102.4000
```

← sampling frequency

V

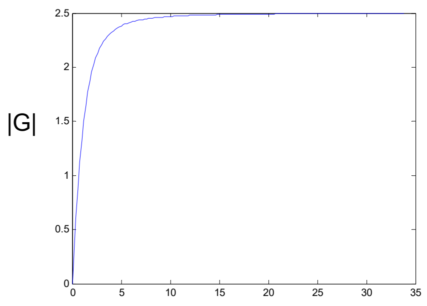


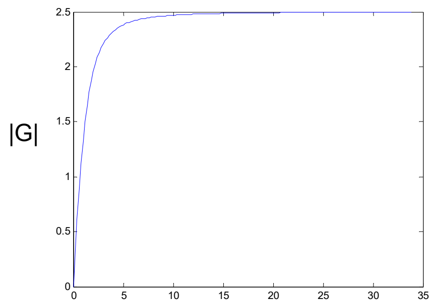
O



Here is the output of pulserVT, which is used as the input to our LTI_X system, and the corresponding LTI_X output. We see the sampling frequency here is about 100 MHz.

```
>> If =FourierT( V, dt);
>> Of = FourierT( O, dt);
>> G =Of./If;
>> f =s_space(0, 1/dt, 512);

>> plot(f(f<35), abs(G(f<35)))
```





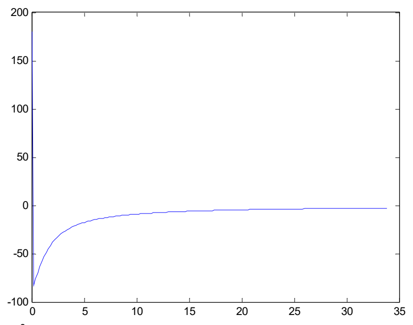
actual transfer function magnitude

$$G = \frac{-100\pi i f}{200 - 40\pi i f}$$

Since the only "noise" in this example is due to very small numerical errors, we can try to do the deconvolution directly by computing the input and output spectra and do a direct division, as shown. We see the magnitude of the resulting transfer function agrees with the exact transfer function, whose expression and plot are given above from 0 to 30 MHz.

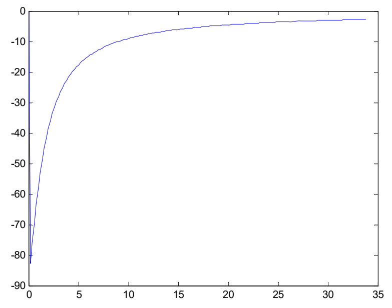
```
>> plot(f(f<35),(180/pi)*angle(G(f<35)))
```

phase of G
(degrees)



actual transfer
function is zero at f =0 so
phase is indeterminant

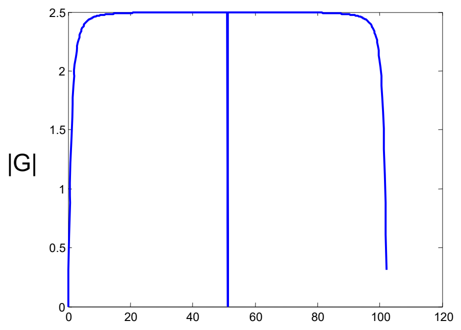
phase of G
(degrees)



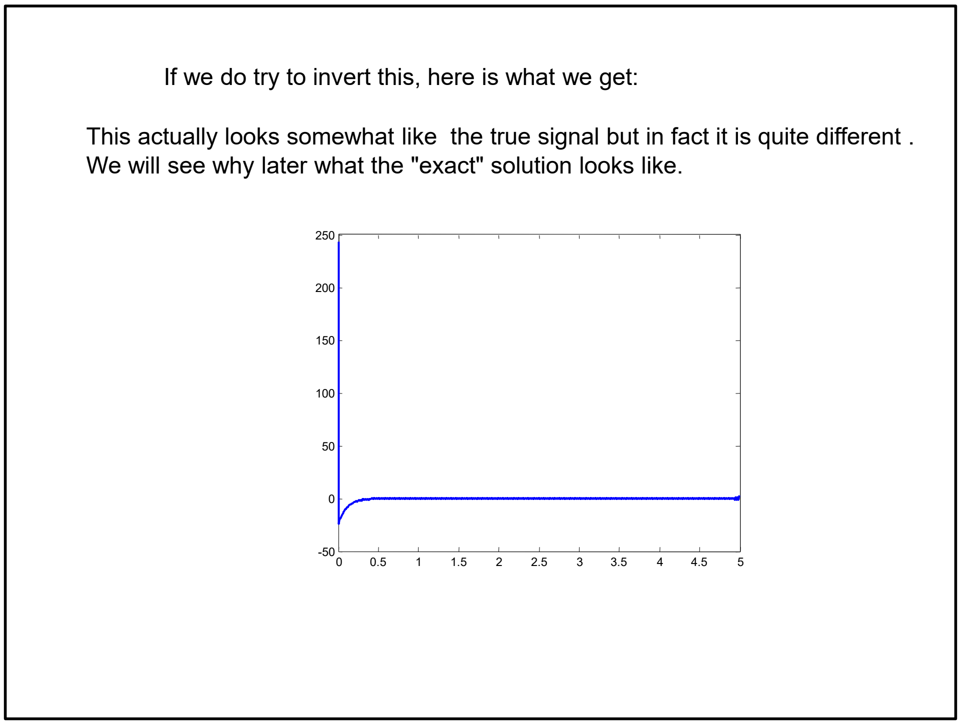
The phase also agrees except at f = 0 but since the transfer function is zero at f = 0 its phase is undefined and so arbitrary at that frequency.

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

G does not satisfy Nyquist criterion so we need to filter out frequencies above the Nyquist frequency which is about 51.2 MHz here before we can truly invert with the Fourier transform

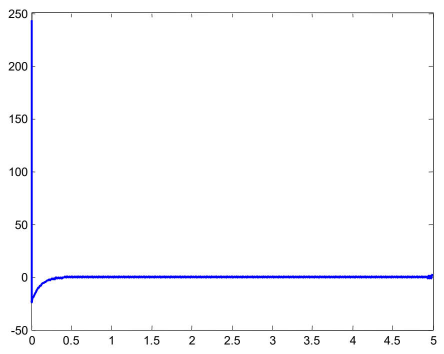


If we examine the transfer function from 0 to 100 MHz we see it does not satisfy the Nyquist criterion so we need to do some filtering if we hope to reliably invert this function to obtain the impulse response function in the time domain.

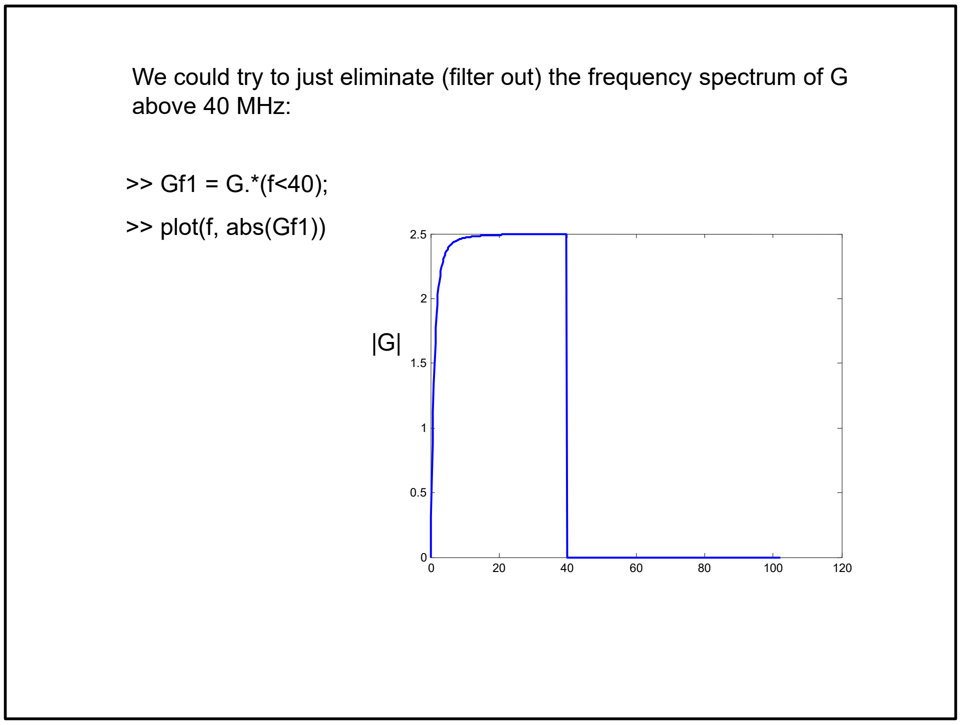


If we do try to invert this, here is what we get:

This actually looks somewhat like the true signal but in fact it is quite different .
We will see why later what the "exact" solution looks like.

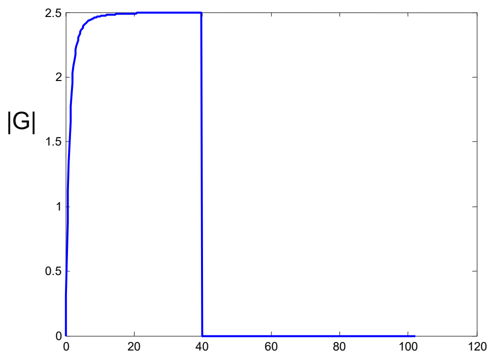


However, if we do try to do the inversion, here is what we get. We will now show how we can get the impulse response function in a correct manner.



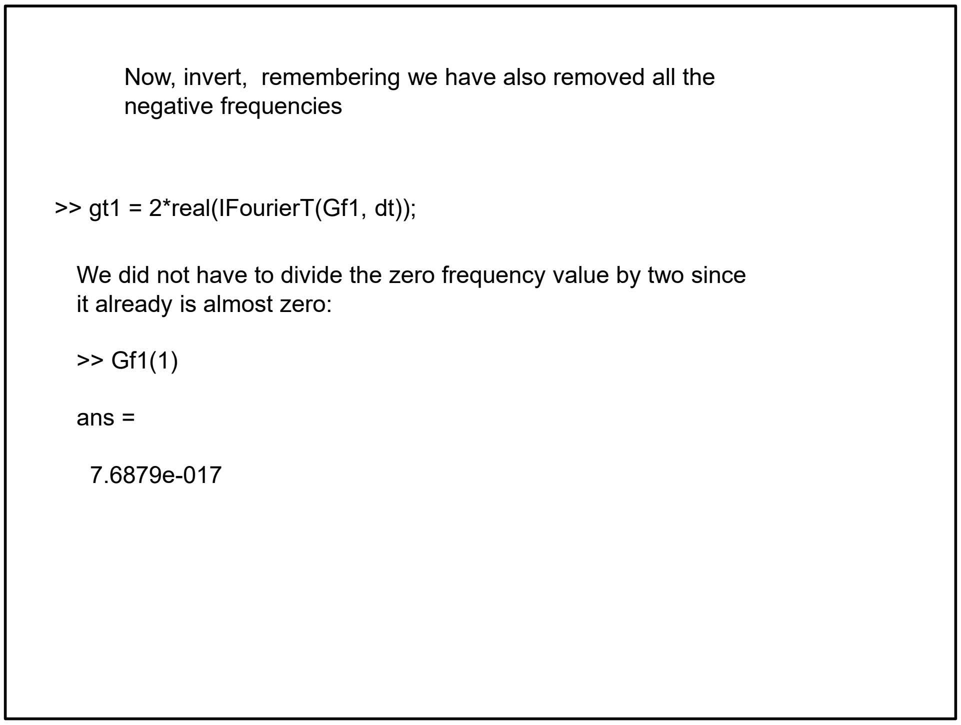
We could try to just eliminate (filter out) the frequency spectrum of G above 40 MHz:

```
>> Gf1 = G.*(f<40);
>> plot(f, abs(Gf1))
```



NOTE: This slide and the following slides continue the discussion beyond what was asked in the problem.

A very simple way to do a more proper inversion is to simply zero out all positive frequencies above 40 MHz and all the negative frequencies.



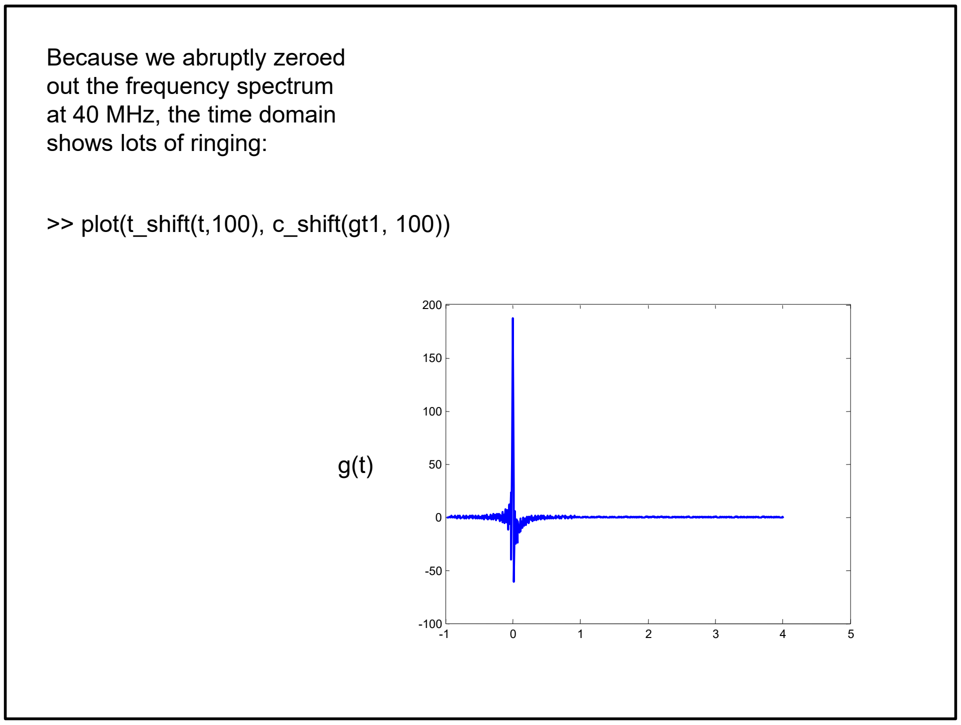
Now, invert, remembering we have also removed all the negative frequencies

```
>> gt1 = 2*real(IFourierT(Gf1, dt));
```

We did not have to divide the zero frequency value by two since it already is almost zero:

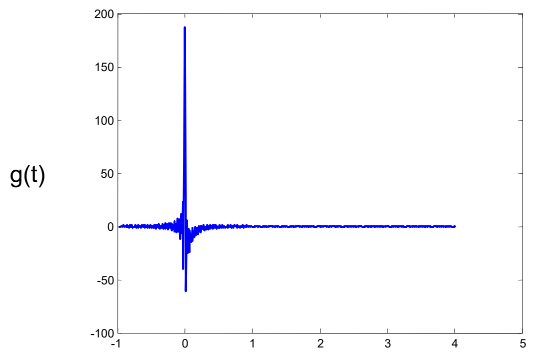
```
>> Gf1(1)

ans =

  7.6879e-017
```

Here is the inverse Fourier transform of our filtered function, using only the positive frequencies. The zero frequency value is almost zero so we do not have to divide it by two before doing the inversion.



Because we abruptly zeroed out the frequency spectrum at 40 MHz, the time domain shows lots of ringing:

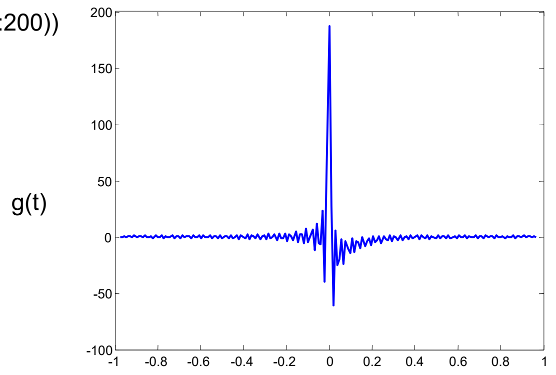
```
>> plot(t_shift(t,100), c_shift(gt1, 100))
```



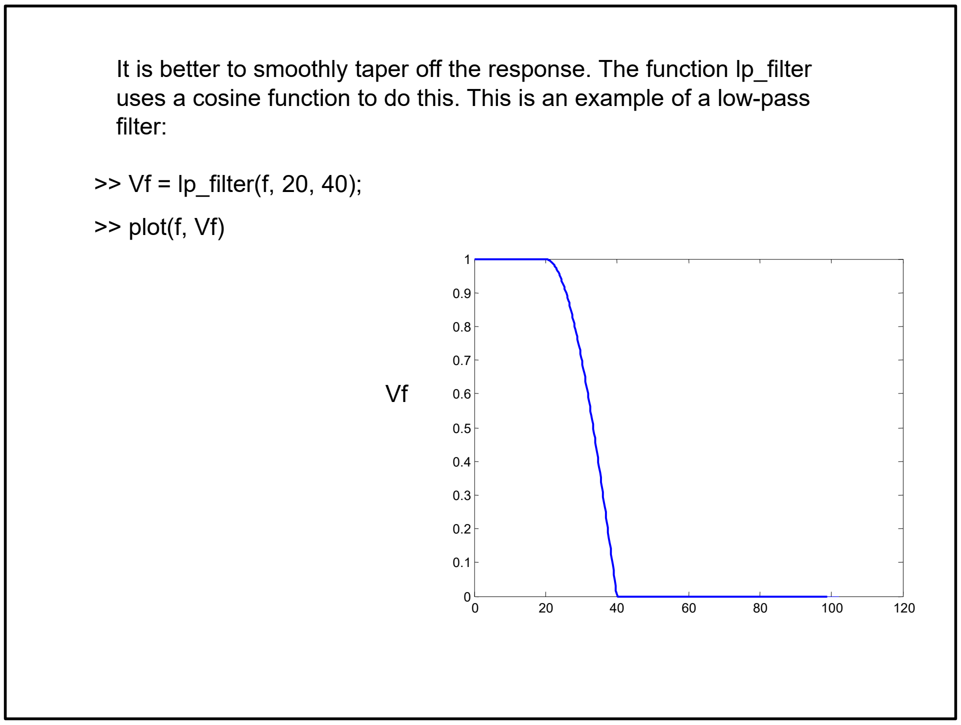
This is the impulse response function we get. Note that we have used c_shift and t_shift here since the impulse response function has values for negative times so without these shifts the waveform would appear to be “split” in the time window. Because we abruptly zeroed out the frequency components at 40 MHz, this filtered function will have a jump in the frequency domain, which causes the ringing we see above in the time domain. This ringing can hide the true behavior of the function so we would like to eliminate it.

Expanded view

```
>> ts = t_shift(t,100);
>> gt1s = c_shift(gt1,100);
>> plot(ts(1:200), gt1s(1:200))
```

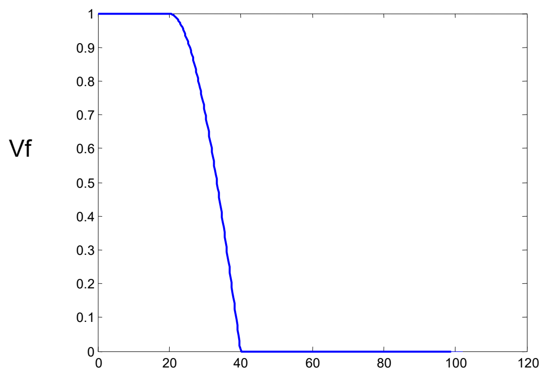


Here is an expanded view of the impulse response function that shows the ringing more clearly.

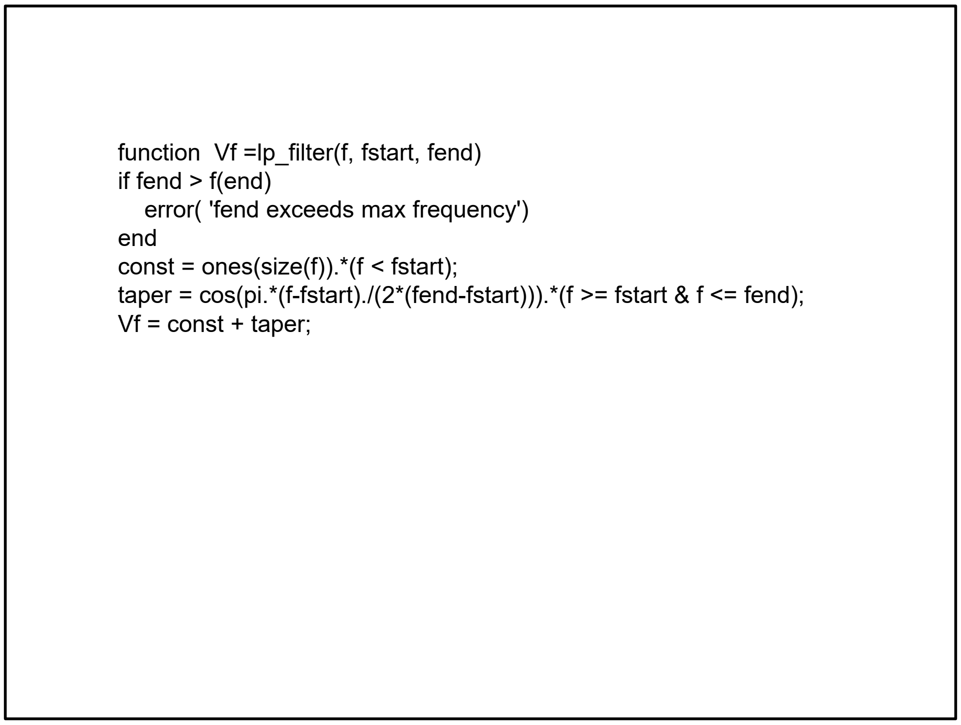


It is better to smoothly taper off the response. The function lp_filter uses a cosine function to do this. This is an example of a low-pass filter:

```
>> Vf = lp_filter(f, 20, 40);
>> plot(f, Vf)
```



Rather than abruptly cutting off the high frequencies is it better to use a smooth taper. The lp_filter function (it is a “low pass” filter, hence the name) uses a cosine function taper, which in this case starts at 20 MHz and ends at 40 MHz, and where all the frequency values below 20 MHz are left unchanged.



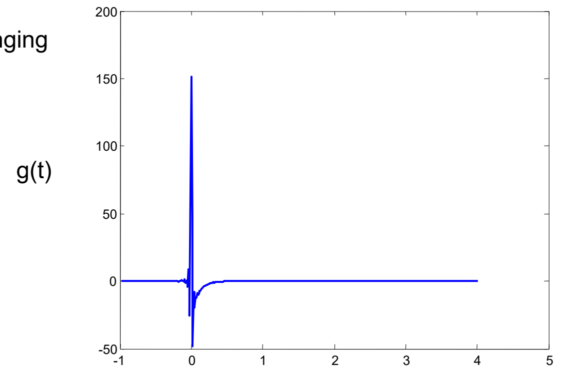
```
function  Vf =lp_filter(f, fstart, fend)
if fend > f(end)
   error( 'fend exceeds max frequency')
end
const = ones(size(f)).*(f < fstart);
taper = cos(pi.*(f-fstart)./(2*(fend-fstart))).*(f >= fstart & f <= fend);
Vf = const + taper;
```

Here is the Matlab lp_filter function.

Now , multiply our G function by this filter and then invert:

```
>> Gf2 = G.*Vf;
>> gt2  = 2*real(IFourierT(Gf2, dt));
>> plot(t_shift(t,100), c_shift(gt2, 100))
```

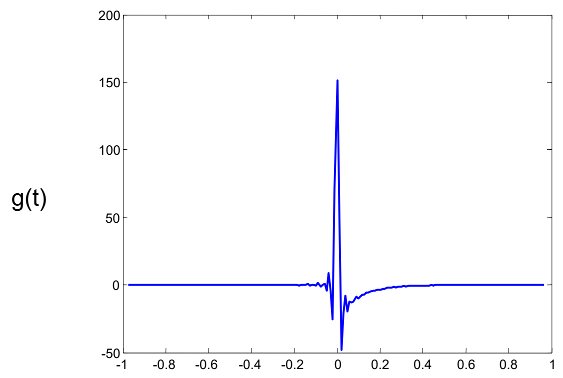
Now there is much less ringing



If we multiply the transfer function by this low pass filter and then do an inverse Fourier transform, we see the ringing is greatly reduced.

Expanded view

```
>> ts = t_shift(t,100);
>> gt2s = c_shift(gt2,100);
>> plot(ts(1:200), gt2s(1:200))
```



Here is an expanded view of the impulse function. There is, however, still some ringing. We can actually do even better with a different approach which we will now discuss.

Another way to get the impulse response

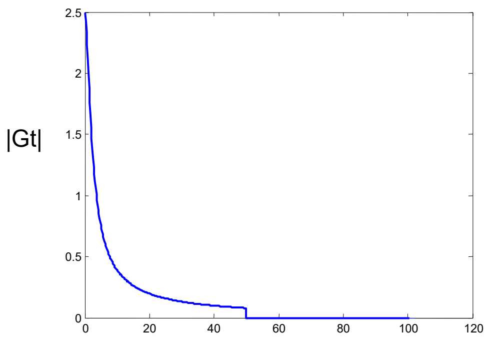
As the frequency goes to infinity G goes to 2.5 (a constant)

But, we know the Fourier Transform of a delta function is 1.0 so

the inverse Fourier Transform of 2.5 gives $2.5\,\delta(t)$

Since we know this inverse transform exactly, subtract the constant 2.5 part from G (throwing out the negative frequencies):

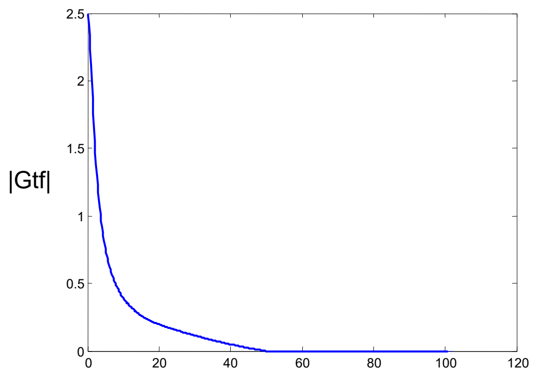
```
>> Gt = (G -2.5).*(f <50);
>> plot(f, abs(Gt))
```



The transfer function goes to a constant value of 2.5 at high frequencies. We know that the inverse Fourier transform of a constant at all frequencies is just that constant times a delta function. Thus, let us subtract off that constant value and eliminate all frequencies above 50 MHz. The plot of the resulting function is shown.

Now use a low pass filter to make the function go to zero smoothly

```
>> Vf = lp_filter(f, 20, 50);
>> Gtf = Gt.*Vf;
>> plot(f, abs(Gtf))
```

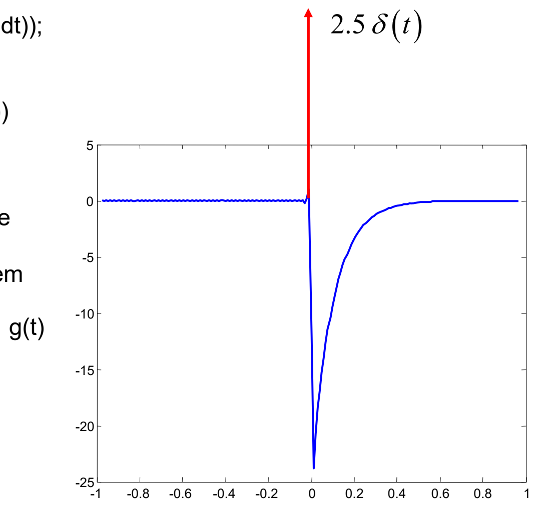


There is still a small discontinuity in our function, which will cause some ringing, so lets now use a low pass filter to make it smoothly go to from 20 to 50 MHz, as shown in the above plot.

Now, invert this part, remembering to take half the zero frequency value:

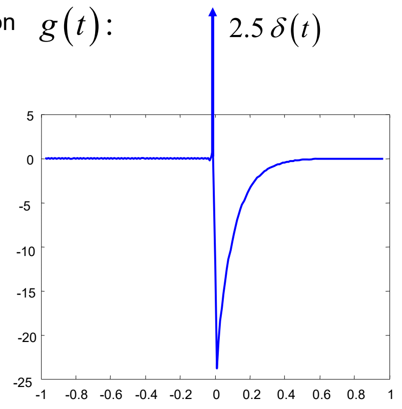
```
>> Gtf(1) = Gtf(1)/2;
>> gt3 =2*real(IFourierT(Gtf, dt));
>> gt3s =c_shift(gt3, 100);
>> plot(ts(1:200), gt3s(1:200))
```

If we add back in the delta function part (shown in red) we can see clearly what the total impulse response of this system looks like:



If we do an inverse Fourier transform of these positive frequency components and add in the delta function part we subtracted off, we get a response which is very close in fact to the exact impulse response.

impulse response function $g(t)$:



$$\begin{tikzcd} i(t) \arrow[r] & \boxed{\text{LTI\_X}} \arrow[r] & o(t) \end{tikzcd}$$

$$o(t) = \int_{-\infty}^{+\infty} i(\tau)\, g(t-\tau)\, d\tau$$

or $$o(t) = IFourierT\left[G(\omega)I(\omega)\right] \Leftarrow \text{easier}$$

With this impulse response we can then determine the response of our LTI_X system to *any* input, either through deconvolution (in the time domain) or through multiplication in the frequency domain followed by an inverse Fourier transform. The frequency domain approach is generally much easier.

C.3. Consider an LTI system which has as its transfer function

$$G(f) = \begin{cases} \cos(\pi f/40) & 0 < f < 20MHz \\ 0 & otherwise \end{cases}.$$

Also, consider an input spectrum to this system given by

$$I(f) = \begin{cases} 1 - f/20 & 0 < f < 20MHz \\ 0 & otherwise \end{cases}.$$

We expect the output of this system will then have the spectrum

$$O(f) = G(f)I(f).$$

However, if we add noise to these functions then given *O* and *I* it may not be possible to reliably obtain G by simple division and we must use some filter instead such as the Wiener filter. The MATLAB function noisy will generate noisy sampled versions of both the *O* and *I* given previously over a range 0-40 MHz. The function call is:

```
>> [O, I] = noisy( ) ;
```

Plot both *O* and *I* from 0 to 40 MHz to verify those functions are correct (the noise you will see is very small) and then attempt to obtain *G* by direct division, i.e. compute

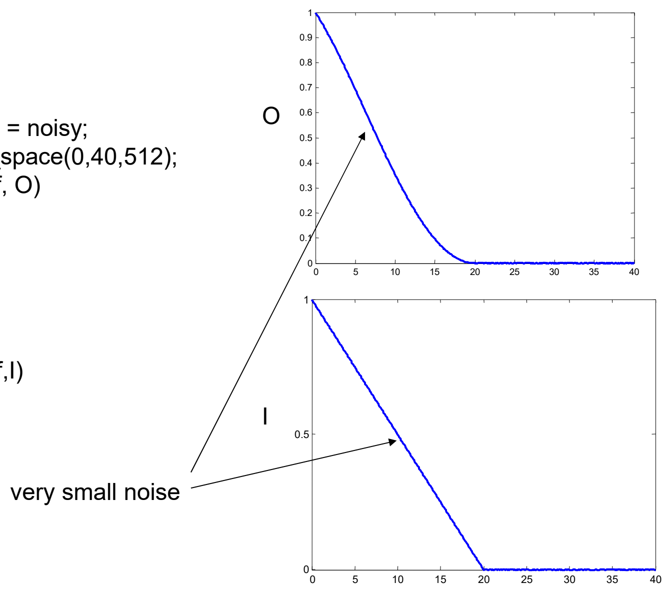
$$G(f) = \frac{O(f)}{I(f)}$$

and plot your results 0-40 MHz. Then use a Wiener filter instead to find *G* and plot your results. Take ε = 0.01. Are the results sensitive to ε? Is there any other way (besides using the Wiener filter) that you can get the “right” answer?

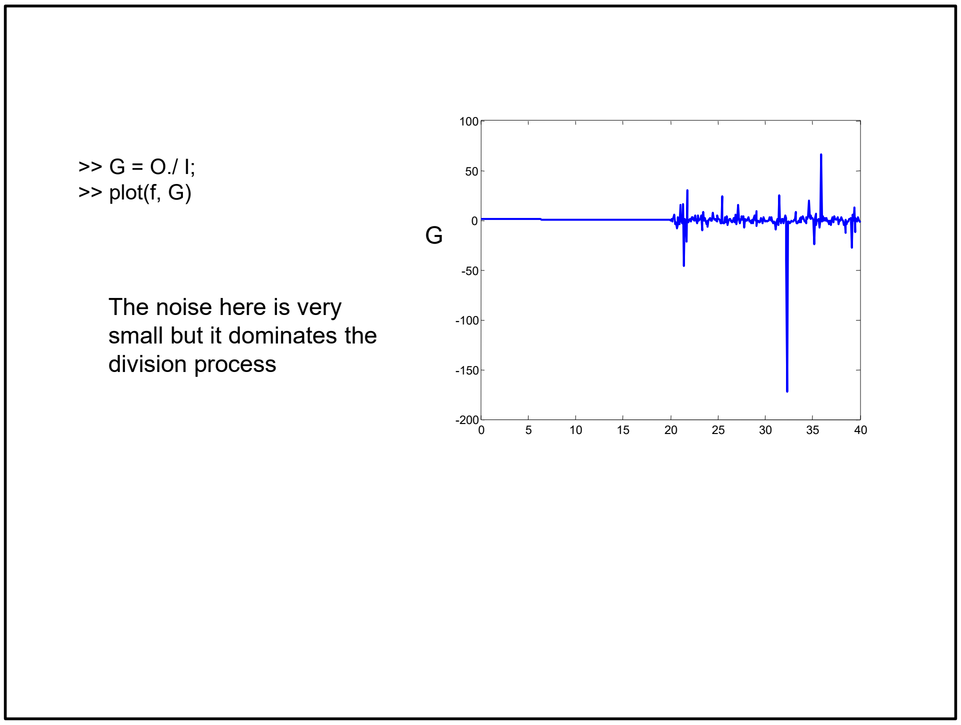
C.3 Noisy Deconvolution

```
>> [O, I ] = noisy;
>> f = s_space(0,40,512);
>> plot( f, O)
```

```
>> plot(f,I)
```

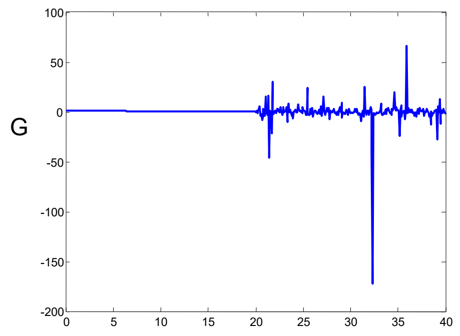


The function noisy reproduces the input and output spectra of a system with a very small amount of noise added to them. Plotted above are these noisy signals, which look much like their original noise-free versions.



```
>> G = O./ I;
>> plot(f, G)
```

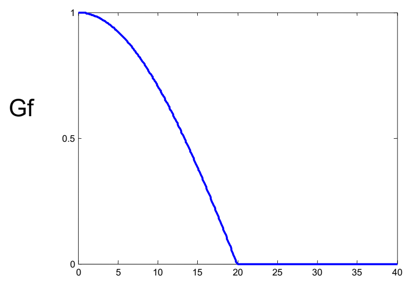
The noise here is very small but it dominates the division process



Here is a result of the deconvolution by division, which shows that the noise is much larger than any of the wanted signals (which are in the 0 to 20 MHz range).

```
function Y = Wiener_filter( O, I, e)
% WIENER_FILTER provides a 1-D filter for desensitizing
% division in the frequency domain (deconvolution) to noise.
% The filter takes a sampled output spectrum ,O, and an
% input spectrum, I, and computes Y = O*conj(I)/(|I|^2 + e^2*M^2)
% where M is the maximum value of |I| and conj( I ) is the
% complex conjugate of I. The constant e is generally taken as
% a constant to represent the noise level. Small values of e
% such as e = 0.01 often work well for ultrasonic systems.
%The calling sequence is:
%   Y = Wiener_filter( O,I, e);
%
M = max( abs(I) );
Y = O.*conj(I)./((abs(I)).^2 + e^2*M^2);
```

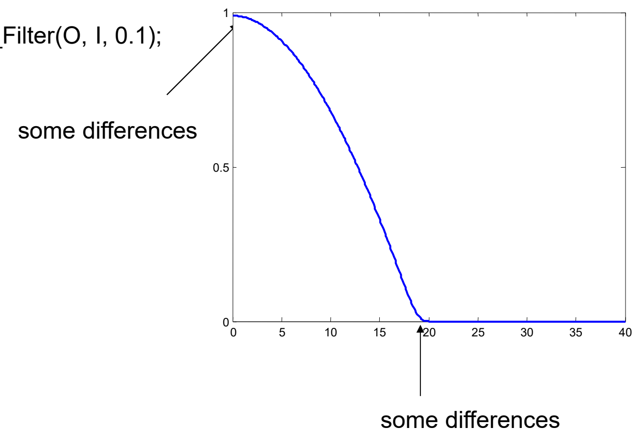
```
>> Gf = Wiener_Filter( O, I, .01);
>> plot( f, Gf)
```



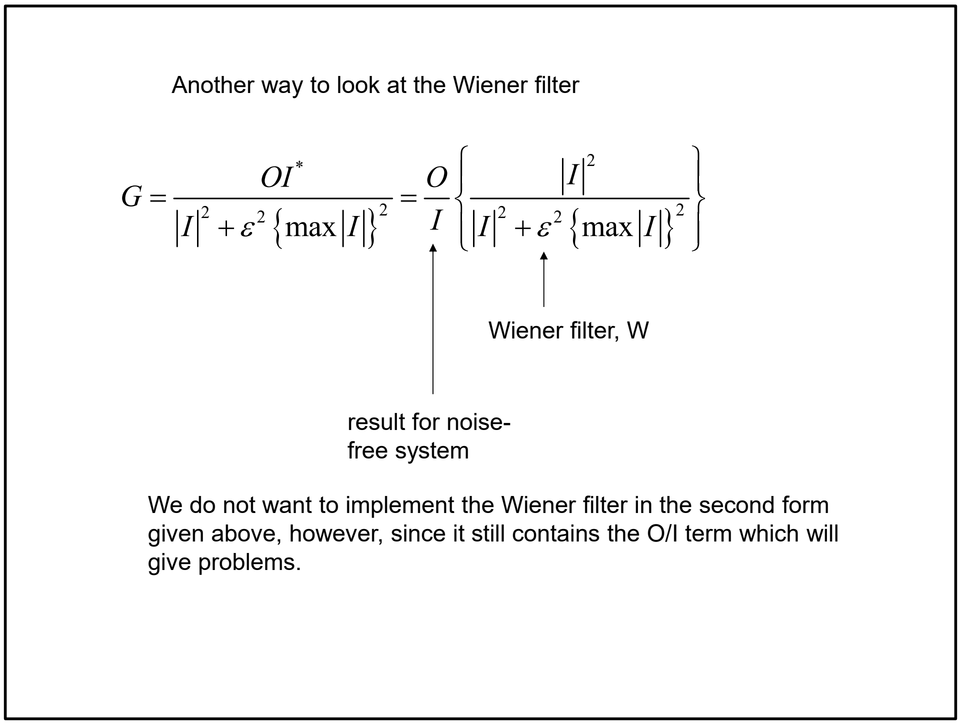
If, instead we use the Wiener_filter function to do the deconvolution we see the proper transfer function in the 0-20MHz bandwidth of the system and the noise at higher frequencies has been eliminated.

If we make the constant too large it will affect the division process where noise is not a problem:

```
>> Gf = Wiener_Filter(O, I, 0.1);
>> plot( f, Gf)
```



We are free to choose the noise constant but we must not make it too large or it will affect the values within the bandwidth of the system. Here a choice of e = 0.1 rather than e = 0.01 shows some small differences. Generally, we try to choose an e which represents the average noise we see in the system.



Another way to look at the Wiener filter

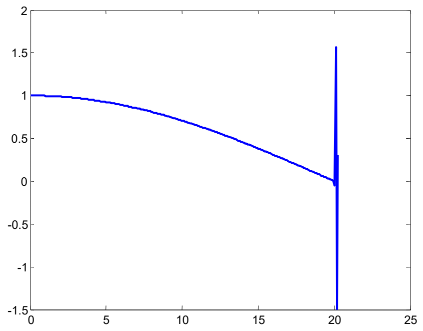
$$G = \frac{OI^*}{|I|^2 + \varepsilon^2 \{\max|I|\}^2} = \underbrace{\frac{O}{I}}_{\text{result for noise-free system}} \underbrace{\left\{ \frac{|I|^2}{|I|^2 + \varepsilon^2 \{\max|I|\}^2} \right\}}_{\text{Wiener filter, W}}$$

We do not want to implement the Wiener filter in the second form given above, however, since it still contains the O/I term which will give problems.

The actual Wiener filter modifies the original direct division in the manner shown by the second form given here. However, we do not want to implement the deconvolution in this second form as it still involves a direct division.

We could also recover the correct G here by computing G = O/I and just looking over the 0-20 MHz range where O and I are both bigger than the noise. Here we look over a slightly larger range so we can also see the noise

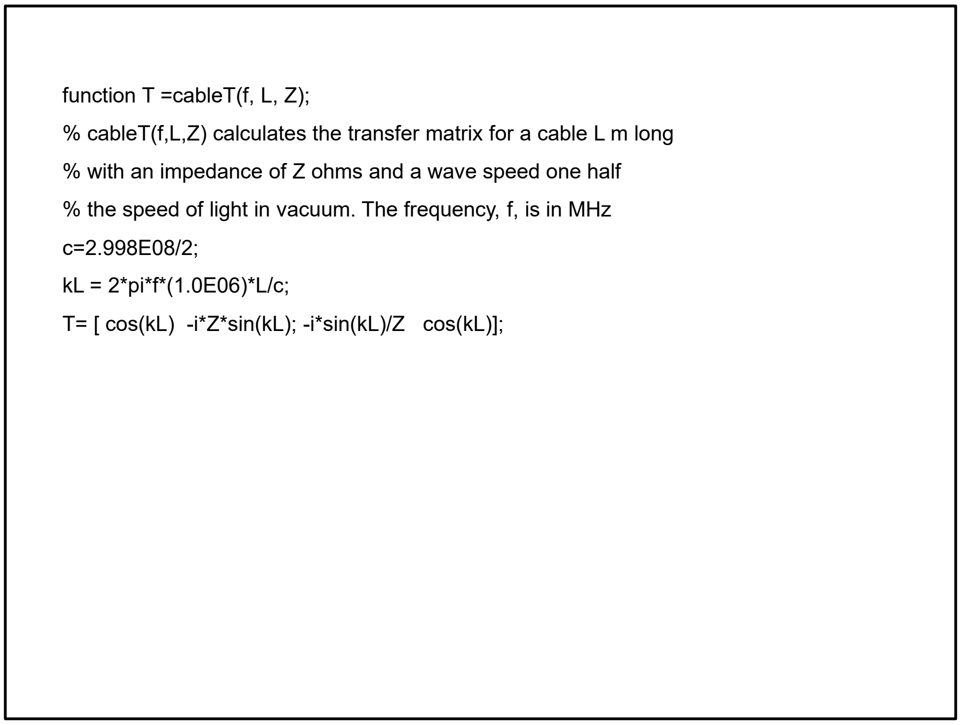
```
>> plot(f(1:260), G(1:260))
>> f(260)
ans = 20.2740
```



The problem here is how do we know beforehand where can we reliably trust the division? The Wiener filter takes care of that automatically

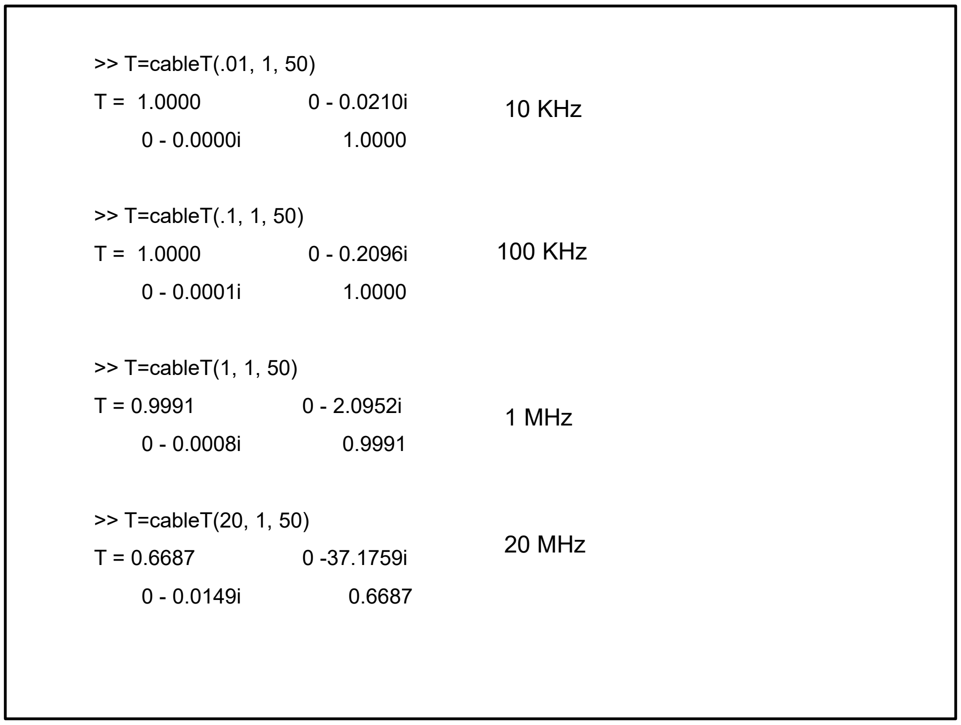
We could also get good deconvolution results by only doing the direct division where the signal-to-noise ratio is high, as shown here, but the Wiener filter essentially already does that automatically for us.

3.1. Consider a 1 meter long, 50 ohm cable, where the wave speed in the cable is one half the wave speed of light, c_0 ,in a vacuum (c_0 = 2.998 x 10^8 m/sec). Determine the transfer matrix components of the cable at 10 kHz, 100 kHz, 1 MHz, 20 MHz.



```
function T =cableT(f, L, Z);
% cableT(f,L,Z) calculates the transfer matrix for a cable L m long
% with an impedance of Z ohms and a wave speed one half
% the speed of light in vacuum. The frequency, f, is in MHz
c=2.998E08/2;
kL = 2*pi*f*(1.0E06)*L/c;
T= [ cos(kL)  -i*Z*sin(kL); -i*sin(kL)/Z   cos(kL)];
```

Here is a Matlab function that computes the transfer matrix for a cable.



```
>> T=cableT(.01, 1, 50)
T =  1.0000                0 - 0.0210i          10 KHz
        0 - 0.0000i              1.0000

>> T=cableT(.1, 1, 50)
T =  1.0000                0 - 0.2096i         100 KHz
        0 - 0.0001i              1.0000

>> T=cableT(1, 1, 50)
T = 0.9991                 0 - 2.0952i           1 MHz
        0 - 0.0008i              0.9991

>> T=cableT(20, 1, 50)
T = 0.6687                 0 -37.1759i          20 MHz
        0 - 0.0149i               0.6687
```

We see at 10 kHz the cable acts just as essentially a pass through device but as the frequency gets higher that is no longer true. At MHz frequencies the cable does modify the signals passing through it.

$$
\begin{aligned}
T_{11}(\omega) &= V_1^{(1)}(\omega) / V_\infty(\omega) \\
T_{21}(\omega) &= I_1^{(1)}(\omega) / V_\infty(\omega) \\
T_{12}(\omega) &= V_1^{(2)}(\omega) / I_s(\omega) \\
T_{22}(\omega) &= I_1^{(2)}(\omega) / I_s(\omega)
\end{aligned}
\tag{3.7}
$$

3.2. Consider a cable for which we wish to measure the transfer matrix components (as a function of frequency). We can do this in MATLAB for a function cable_X which has the calling sequence:

```
>> [ v1, i1, vt, it] = cable_X( V, dt, R,  L, 'term');
```

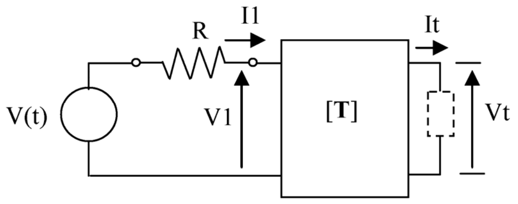


Fig. 3.12. A measurement setup for obtaining the transfer matrix components of a cable.

The input arguments of cable_X are as follows. V is a sampled voltage source versus time, where the sampling interval is dt. R is an external resistance (in ohms). This source and resistance are connected in series to one end of the cable, which is of length L (in m) as shown in Fig. 3.12. The other end of cable can be either open-circuited or short-circuited. The string 'term' specifies the termination conditions. It can be either 'oc' for open-circuit or 'sc' for short-circuit. Cable_X then returns the “measured” sampled voltages and currents versus time (v1, i1, vt, it) where (v1, i1) are on the input side of the cable and (vt, it) are at the terminated end (Note: for open-circuit conditions it = 0 and for short-circuit conditions vt = 0). As a voltage source to supply the V input to cable_X use the MATLAB function pulserVT. For a set of sampled times this function returns a sampled voltage output that is typical of a "spike" pulser. Make a vector, t, of 512 sampled times ranging from 0 to 5 μsec with the MATLAB call:

```
>> t = s_space(0, 5, 512);
```

(see the discussion of the s_space function in Appendix A; a code listing of the function is given in Appendix G and the course slides) and call the pulserVT function as follows:

```
>> V = pulserVT(200, 0.05, 0.2, 12, t);
```

For the resistance, take R = 200 ohms, and specify the length of the cable as L = 2 m.

Using Eq. (3.7), determine the four cable transfer matrix components and plot their magnitudes and phases from 0 to 30 MHz. Note that the outputs of cable_X are all time domain signals but the quantities in Eq. (3.7) are all in the frequency domain so you will need to define a set of 512 sampled frequency values, f, through:

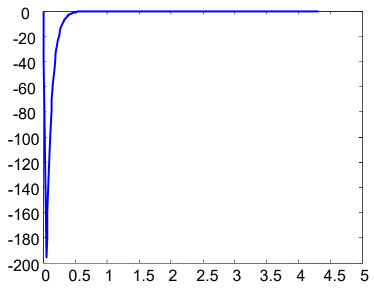
```
>> dt = t(2) –t(1);
>> f =s_space(0, 1/dt, 512);
```

What is the range of frequencies contained in f here?

3.2

Cable Transfer Matrix Components

```
>> t = s_space(0,5, 512);
>> V = pulserVT(200, 0.05, 0.2, 12, t);
>> plot(t, V)
>> dt = t(2) - t(1);
>> f = s_space(0, 1/dt, 512);
```



sampling frequency = 1/dt = 102 MHz

To determine the transfer matrix components we need a voltage source, which we get here by using the Matlab function pulserVT, a function which simulates a spike pulser. We see the sampling used here generates a maximum frequency of about 100 MHz.

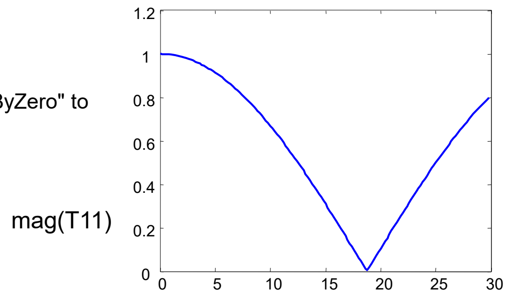
```
>> [V1, I1, Vt, It] =cable_X(V, dt, 200, 2, 'oc');
>> [V12, I12, Vt2, It2] =cable_X(V, dt, 200, 2, 'ss');
>> data =[V1' I1' Vt' It' V12' I12' Vt2' It2'];
>> fdata=FourierT(data, dt);
>> V1f = fdata(:,1);
>> I1f =fdata(:,2);
>> Vtf =fdata(:,3);
>> Itf =fdata(:,4);
>> V12f =fdata(:,5);
>> I12f =fdata(:,6);
>> Vt2f =fdata(:,7);
>> It2f =fdata(:,8);
```

$\longleftarrow$ put data in columns, do FFTS all at once

$\longleftarrow$ should be zeros

$\longleftarrow$ should be zeros

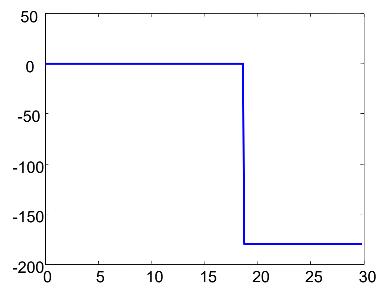
```
>> T11 = V1f./Vtf;
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to
suppress this warning.)
>> plot(f(f<30), abs(T11(f<30)))
```



Here we feed the output of pulserVT into one end of the unknown cable modeled by cable_X under both open circuit and short circuit conditions at the other cable end. The cable_X function gives us the “measured” voltage and current at both ends under both of these termination conditions. We can put all of these “measurements” into the columns of a data matrix and then do an FFT of all this data at once. Then using Eq. (3.7) we can find the transfer matrix components. Here is the magnitude of T11.

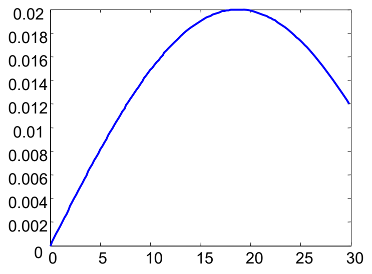
```
>> plot(f(f<30), (180/pi)*unwrap(angle(T11(f<30))))
```

phase of T11



```
>> T21 =I1f./Vtf;
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero"
to suppress this warning.)
>> plot(f(1:150), abs(T21(1:150)))
```

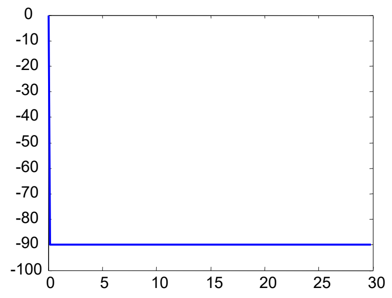
mag(T21)



Here is the phase of T11 and the magnitude of T21.

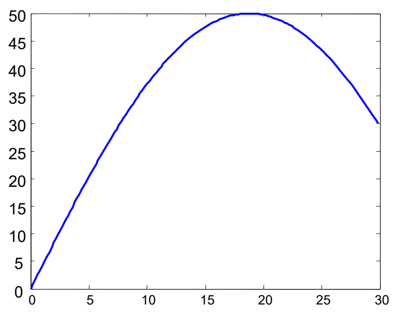
```
>> plot(f(f<30), (180/pi)*unwrap(angle(T21(f<30))))
```

phase of T21



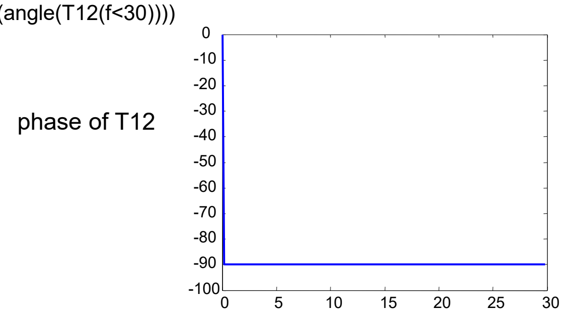
```
>> T12= V12f./It2f;
>> plot(f(f<30), abs(T12(f<30)))
```

mag(T12)

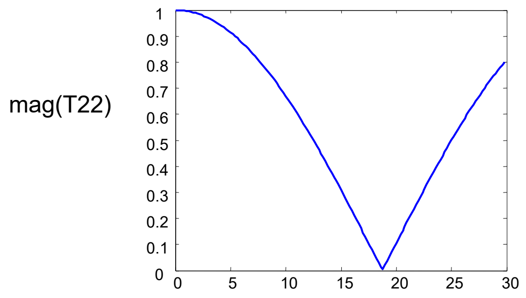


Here is the phase of T21 and the magnitude of T12. Note that T12 is not equal to T21.

```
>> plot(f(f<30), (180/pi)*unwrap(angle(T12(f<30))))
```

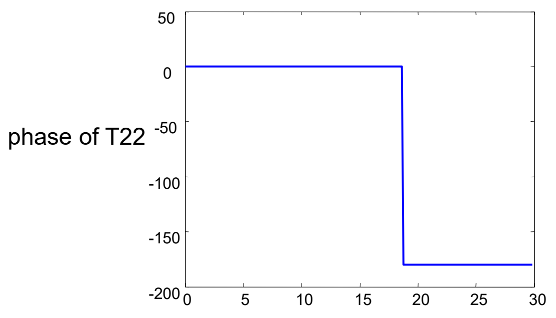


```
>> T22=I12f./It2f;
>> plot(f(f<30), abs(T22(f<30)))
```



Here is the phase of T12 and the magnitude of T22.

```
>> plot(f(f<30), (180/pi)*unwrap(angle(T22(f<30))))
```



Finally, here is the phase of T22. We see that the behavior of all of these elements follows what we expect from the cable model discussed in the notes.

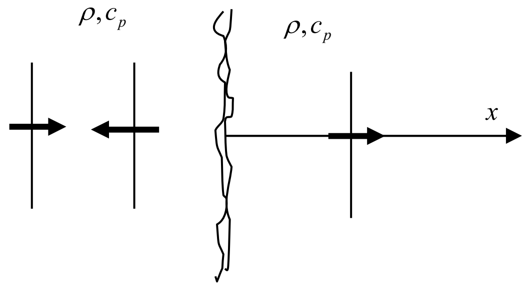


Fig. D.27. A plane P-wave incident on a rough interface.

D.1. Consider the case where a solid is split along a rough planar interface which lies in the plane $x = 0$ as shown in Fig D.27. The two portions of the solid are in contact over some places on the interface and are not in contact at other places. Where the two sides touch the stress τ_{xx} is continuous, and where the sides do not touch $\tau_{xx} = 0$ (on both sides) so again the stress is continuous. Where the sides touch the displacement u_x is continuous but where the sides do not touch there can be a displacement of one side

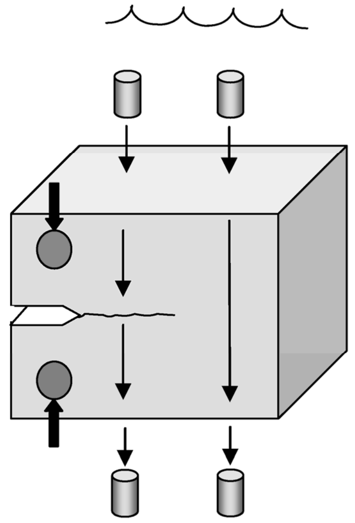


Fig. D.28. A measurement setup for ultrasonically examining a partially closed crack. Note that the crack extends across the entire width of the block.

relative to another. It is reasonable to expect that the amount of this relative displacement is proportional to the stress at the interface. Thus, under these conditions, we could expect that we might specify boundary conditions on the interface as:

continuity of stress:

$$\tau_{xx}\left(x = 0^-, t\right) = \tau_{xx}\left(x = 0^+, t\right)$$

stress proportional to the relative displacement:

$$\tau_{xx}\left(x=0,t\right)=\kappa_s\left[u_x\left(x=0^+,t\right)-u_x\left(x=0^-,t\right)\right].$$

The constant κ_s determines the relative "springiness" of the interface. The case $\kappa_s = 0$ corresponds to a stress free interface (no transmission) while $\kappa_s \to \infty$ means that the displacement is also continuous so that we have perfect contact and complete transmission (no reflection).

(a) Determine the reflection and transmission coefficients (based on stress ratios) for a P-wave at normal incidence to this rough interface and plot the magnitude and phase of these coefficients versus frequency from 0-20 MHz for steel with $\kappa_s = 150$ MPa/μm.

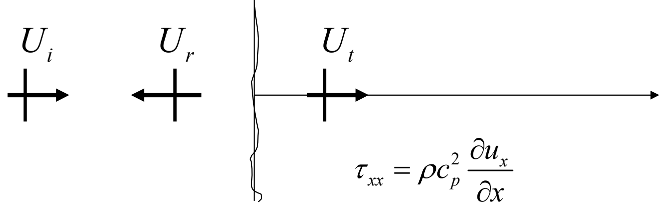
(b) The magnitude of transmission coefficient, *T*, you obtain in part (a) should be of the form , $\left|T\left(f\right)\right| = 1/\sqrt{1+C^2 f^2}$, where *f* is the frequency and the constant *C* is related to κ_s and ρc_p. We could use this transmission coefficient to try to estimate the effects of crack closure of a rough crack as follows. Figure D.28 shows a compact tension specimen which is used to grow a through-thickness crack from a starter notch. If the compact tension specimen is then loaded, the sides of this rough crack will touch at some points and not at others, so the crack surface will look like two rough surfaces in partial contact, the same problem as shown in Fig. D.27. Suppose we now examine this crack with a through-transmission immersion ultrasonic experiment, as shown in Fig. D.28, and also do a reference experiment where we move the transducers laterally so that they are not over the crack. Let $V_c\left(f\right)$ be the frequency components of the measured voltage for the case when we are over the crack, and let $V_r\left(f\right)$ be the frequency components of the measured voltage for the reference experiment, Since the only difference between two setups is the transmission coefficient at the crack, we expect that the voltages to satisfy $V_c\left(f\right)=T\left(f\right)V_r\left(f\right)$. The MATLAB function rough_crack gives the sampled received voltage versus time, vc, for the case when the transducers are placed over the crack, the sampled voltage versus time, vr, for the reference setup, and the sampled time values, t. The function call is:

```
>> [ vc, vr, t] =rough_crack
```

Using this function, obtain the measured transmission coefficient, $T\left(f\right)$, versus frequency [Note: if you want to use a Wiener filter here, the numerical round off "noise" is extremely small so choose a small constant value such as $\varepsilon = 0.001$ or smaller]. Using this transmission coefficient, determine a best fit value of *C*, and the corresponding value of κ_s (in MPa/μm), which is a measure of how closed the crack is. Assume the compact tension specimen is made of steel whose specific plane wave impedance is $z^a = 46\text{x}10^6$ kg/(m^2-sec).

D.1

Rough Crack, part (a)



$$\tau_{xx} = \rho c_p^2 \frac{\partial u_x}{\partial x}$$

$$u_x = U_i \exp\left(ik_p x - i\omega t\right) \qquad \tau_{xx} = T_i \exp\left(ik_p x - i\omega t\right) = i\rho\omega c_p U_i \exp\left(ik_p x - i\omega t\right)$$

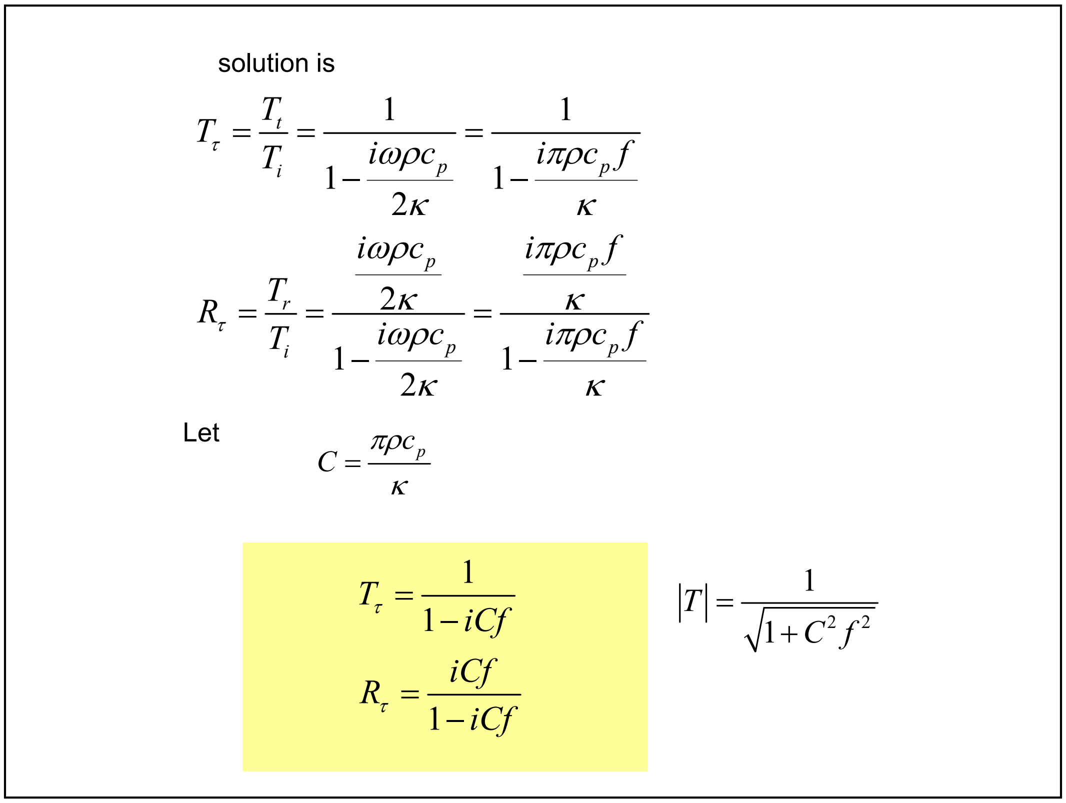
$$u_x = U_r \exp\left(-ik_p x - i\omega t\right) \qquad \tau_{xx} = T_r \exp\left(-ik_p x - i\omega t\right) = -i\rho\omega c_p U_r \exp\left(-ik_p x - i\omega t\right)$$

$$u_x = U_t \exp\left(ik_p x - i\omega t\right) \qquad \tau_{xx} = T_t \exp\left(ik_p x - i\omega t\right) = i\rho\omega c_p U_t \exp\left(ik_p x - i\omega t\right)$$

$$\tau_{xx}\left(0^+\right) = \tau_{xx}\left(0^-\right) \quad \Rightarrow \quad T_i + T_r = T_t$$

$$\tau_{xx}\left(0^+\right) = \kappa\left[u_x\left(0^+\right) - u_x\left(0^-\right)\right] \quad \Rightarrow \quad T_t = \kappa\left[\frac{T_t}{i\rho\omega c_p} - \left(\frac{T_i}{i\rho\omega c_p} + \frac{T_r}{-i\rho\omega c_p}\right)\right]$$

Here are the displacements and stresses for plane waves incident on and reflected or transmitted from our rough crack (modeled as a planar surface with boundary conditions that simulate the roughness.) For places where the faces of the crack touch the stress is continuous while the stress is zero on both faces where the crack is open (and so also is continuous). Thus, we can say the stress is continuous at the crack, which is our first boundary condition. If the crack were completely closed we would expect the displacement also to be continuous, but for those portions of the crack that are open, the two faces of the surface can have different displacements. We will assume that any displacement differences that are present at the interface are proportional to the stress at the interface. This is our second boundary condition. Physically, we are treating the interface here like a “springy” interface, where κ is like a spring constant. We can use these two conditions at the crack surface to obtain relations between the stress amplitudes of the waves present.



solution is

$$T_\tau = \frac{T_t}{T_i} = \frac{1}{1-\frac{i\omega\rho c_p}{2\kappa}} = \frac{1}{1-\frac{i\pi\rho c_p f}{\kappa}}$$

$$R_\tau = \frac{T_r}{T_i} = \frac{\frac{i\omega\rho c_p}{2\kappa}}{1-\frac{i\omega\rho c_p}{2\kappa}} = \frac{\frac{i\pi\rho c_p f}{\kappa}}{1-\frac{i\pi\rho c_p f}{\kappa}}$$

Let

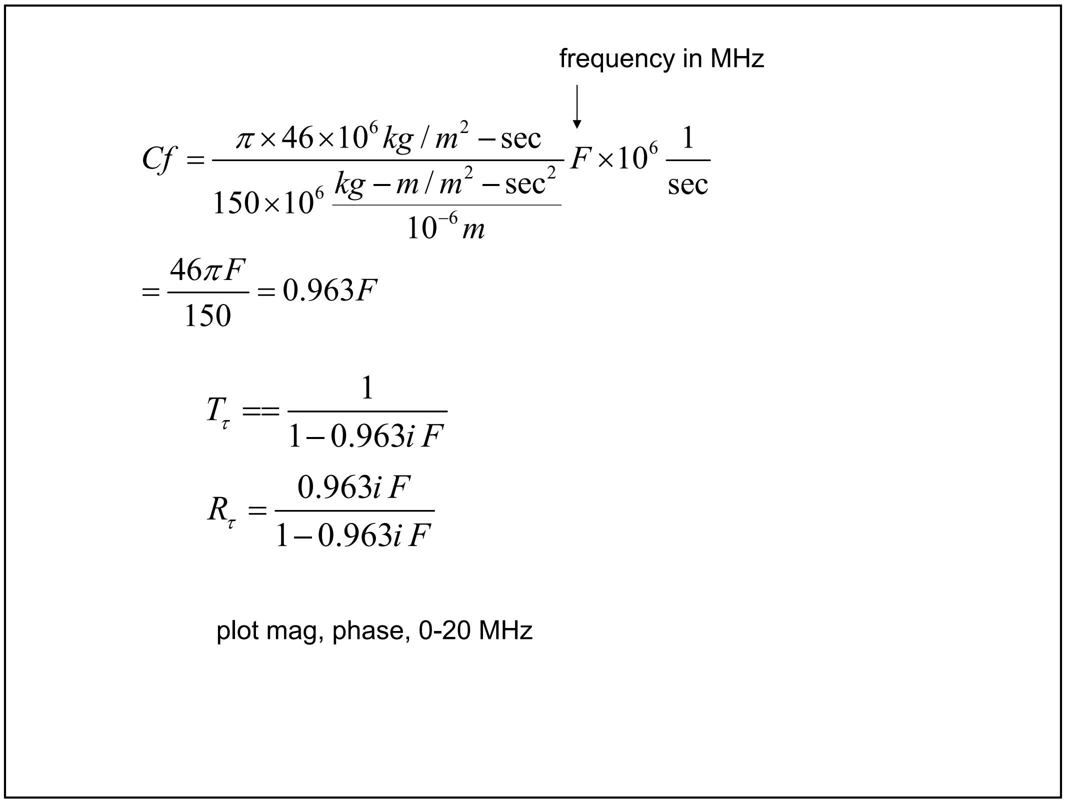
$$C = \frac{\pi\rho c_p}{\kappa}$$

$$T_\tau = \frac{1}{1-iCf}$$

$$R_\tau = \frac{iCf}{1-iCf}$$

$$|T| = \frac{1}{\sqrt{1+C^2 f^2}}$$

We can use those two relations to solve for the transmission and reflection coefficients at the rough crack, which we can put in rather simple forms in terms of the constant C. Also shown is the form of the magnitude of the transmission coefficient as a function of frequency.



frequency in MHz

$$Cf = \frac{\pi \times 46 \times 10^6\, kg/m^2 - \sec}{150 \times 10^6 \dfrac{kg - m/m^2 - \sec^2}{10^{-6}\, m}} \overset{\downarrow}{F} \times 10^6 \frac{1}{\sec}$$

$$= \frac{46\pi F}{150} = 0.963F$$

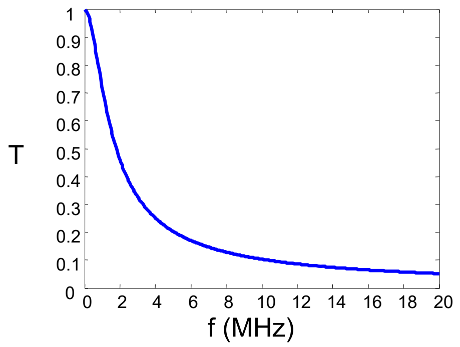
$$T_\tau == \frac{1}{1 - 0.963i\,F}$$

$$R_\tau = \frac{0.963i\,F}{1 - 0.963i\,F}$$

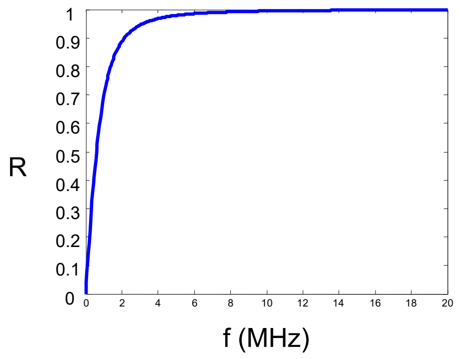
plot mag, phase, 0-20 MHz

For the specified value of the “springiness” constant given in the problem statement, here are the explicit values of the transmission and reflection coefficients as a function of frequency, as measured in MHz.

```
>> f = linspace(0, 20, 500);
>> T = 1./(1-0.963*i*f);
>> R = 0.963*i*f./(1-0.963*i*f);
>> plot(f, abs(T))
```

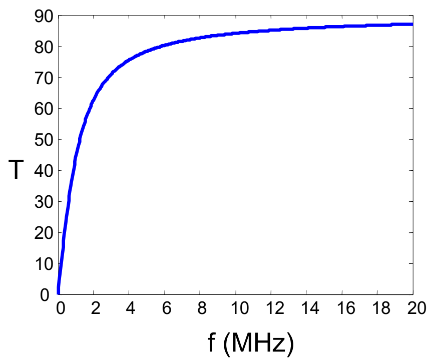


```
>> plot(f, abs(R))
```

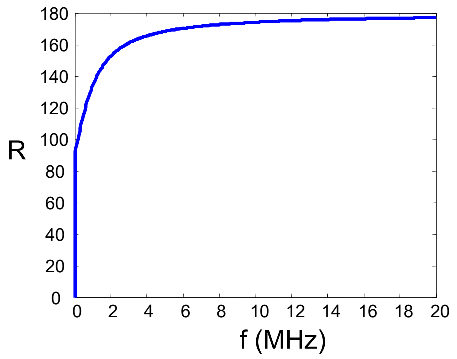


If we plot the magnitude of the transmission and reflection coefficients versus frequency, we see the behavior shown here.

```
>> plot(f, (180/pi)*angle(T))
```



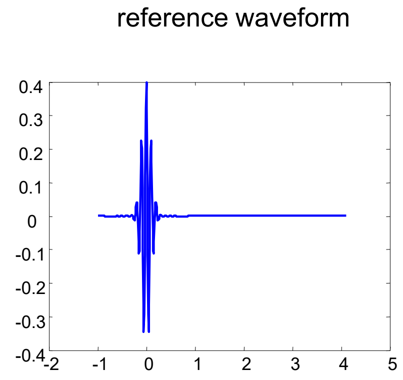
```
>> plot(f, (180/pi)*angle(R))
```



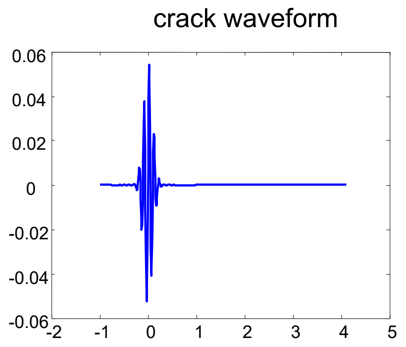
Here are the corresponding phases (in degrees).

Rough Crack, part (b)

```
>> [vc,vr,t] =rough_crack( );
>> plot(t_shift(t, 100),c_shift(vr,100))
```

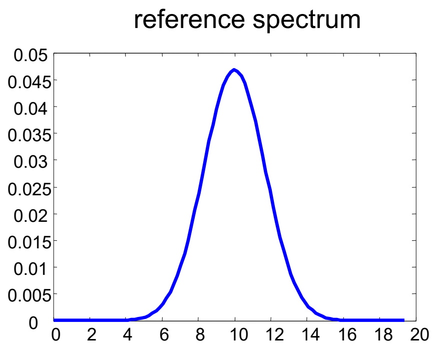


```
>> plot(t_shift(t,100), c_shift(vc,100))
```

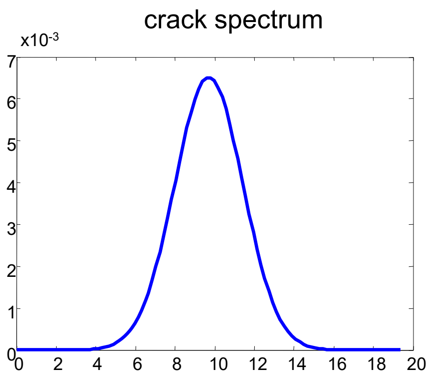


The Matlab function rough_crack gives a simulated waveform signal after it has passed through a rough crack, vc and a simulated reference signal with the crack absent, vr. Note that the amplitude of the wave that has passed through the crack is much smaller in amplitude.

```
>> dt = t(2) - t(1)
dt = 0.0100

>> Vrf = FourierT(vr, dt);
>> f = s_space(0, 1/dt, 512);
>> plot(f(f<20), abs(Vrf(f<20)))
```



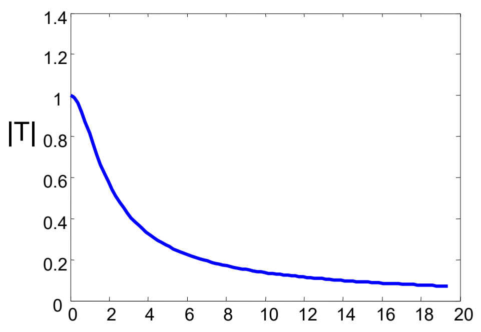
```
>> Vcf = FourierT(vc, dt);
>> plot(f(f<20), abs(Vcf(f<20)))
```



Here are reference and rough crack spectra.

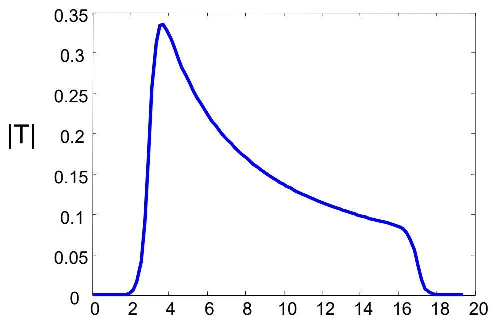
The transfer function (no filter)

```
>> T =Vcf./Vrf;
Warning: Divide by zero.
>> plot(f(f<20),abs(T(f<20)))
```



with Wiener filter

```
>> T = Wiener_filter(Vcf, Vrf, .001);
>> plot(f(f<20), abs(T(f<20)))
```



Since we expect vc(f) = T(f) * vr(f), in principle we can get the transmission coefficient by direct division (deconvolution). Since there is essentially no noise in these simulated signals we can do the direct division, but in practice we should use a Wiener filter. Both cases are shown here. We see the Wiener filter produces a result similar to the straight division case, but only over the frequencies where the values are not small.

Simple fit – match at one frequency (not a good idea in general)

$$|T| = \frac{1}{\sqrt{1+C^2 f^2}}$$

```
>> f(50)
ans = 9.5703
>> abs(T(50))
ans = 0.1428
```

$$1 + C^2 \left(9.5703\right)^2 = 1/\left(0.1428\right)^2$$

$C = 0.724$ actual value was C = 0.723 μsec

$$\kappa = \frac{\pi\left(\rho c_p\right)}{C} = \frac{\pi\left(46\times 10^6\right) kg/m^2 - \sec}{0.724 \mu\sec}$$

$$= 199.6\times 10^{12} \frac{kg}{m^2 - \sec^2}$$

$$= 199.6\times 10^{12} \frac{Pa}{m}$$

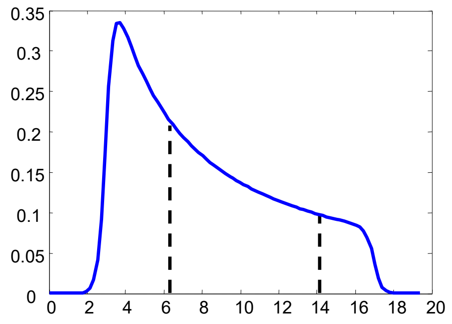
$$= 199.6 \quad MPa/\mu m$$

actual value was κ = 200

Since we known the form of the transmission coefficient, one way to estimate the springiness constant is to simply match the known form to the measured value somewhere within the bandwidth of the results where the Wiener filter is essentially equal to one. Here we chose a frequency of about 9.5 MHz. We see the predicted value of the springiness constant matches well the actual result used to generate the simulated signals. However, in practice we would like to have a more robust way to fit the measured results to the model-based transmission coefficient.

pick out frequency values and transmission coefficient values
from approximately 6 to 14 MHz:

```
>> f(35)
ans =6.6406
>> f(75)
ans =14.4531
>> ff=f(35:75);
>> Tff =T(35:75);
```

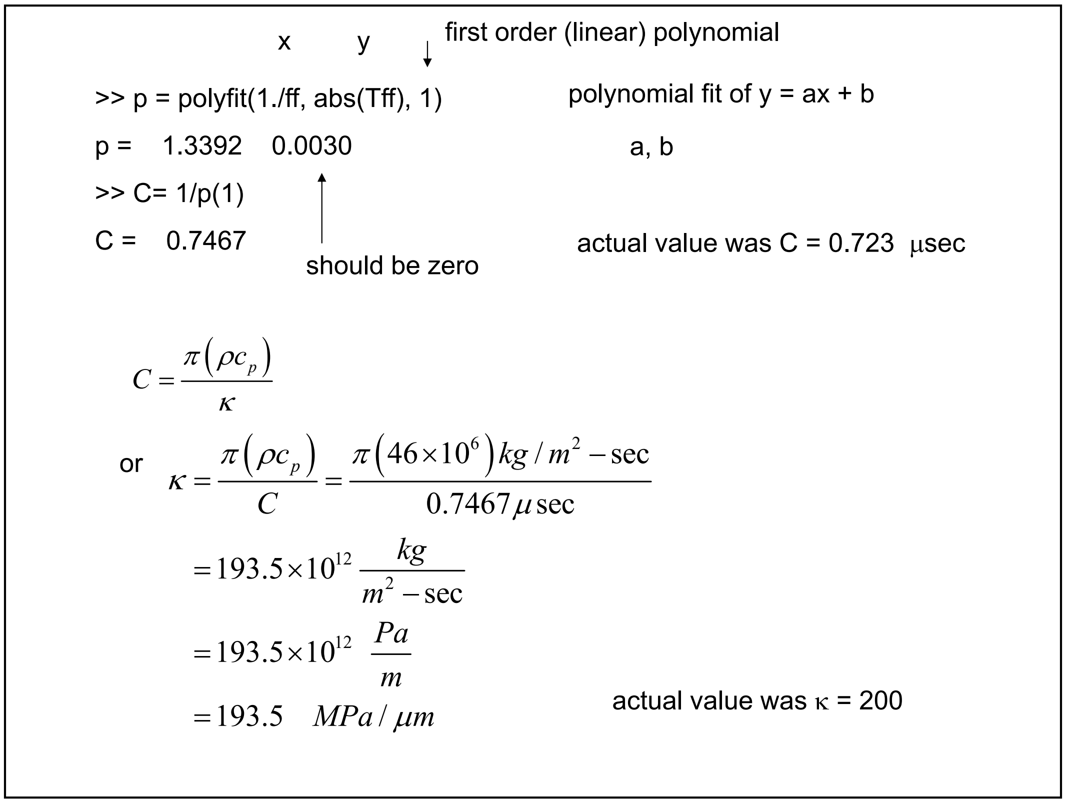


For large f the magnitude of the transmission coefficient looks like

$$|T| = \frac{1}{Cf}$$

Let |T| = y, 1/f = x, 1/C = a then this looks like a straight line
y = ax

A better way to proceed is to fit results over a range of values. Here, we choose the frequency values from about 6.6 MHz to 14.5 MHz, and note that at high frequencies the transmission coefficient has a much simpler behavior that we can use to express the transmission coefficient fitting as a problem of fitting the data to that of a straight line.



```
                 x      y    first order (linear) polynomial
>> p = polyfit(1./ff, abs(Tff), 1)        polynomial fit of y = ax + b
p =    1.3392    0.0030                          a, b
>> C= 1/p(1)
C =    0.7467                           actual value was C = 0.723 μsec
                    should be zero
```

$$C = \frac{\pi\left(\rho c_p\right)}{\kappa}$$

or $$\kappa = \frac{\pi\left(\rho c_p\right)}{C} = \frac{\pi\left(46\times10^6\right)kg/m^2-\sec}{0.7467\mu\sec}$$

$$= 193.5\times10^{12}\,\frac{kg}{m^2-\sec}$$

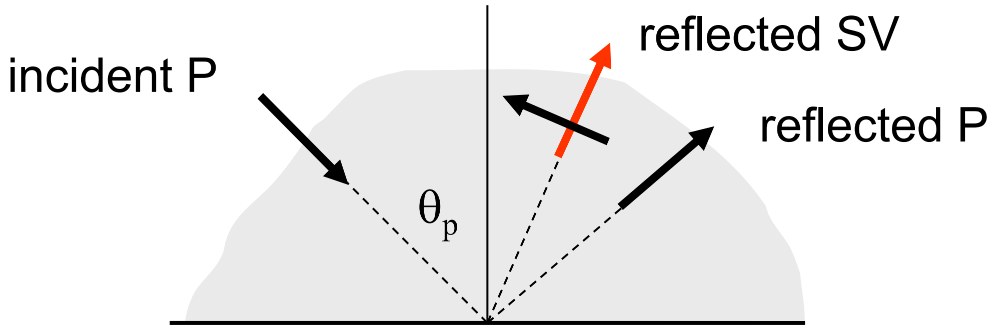
$$= 193.5\times10^{12}\,\frac{Pa}{m}$$

$$= 193.5\quad MPa/\mu m$$

actual value was κ = 200

Matlab has a built-in function polyfit that allows us to do the linear fitting we need over this range of frequencies. We see the results are still quite good in comparison the actual springiness value even though we only used the high frequency behavior of the transmission coefficient in the fitting procedure.

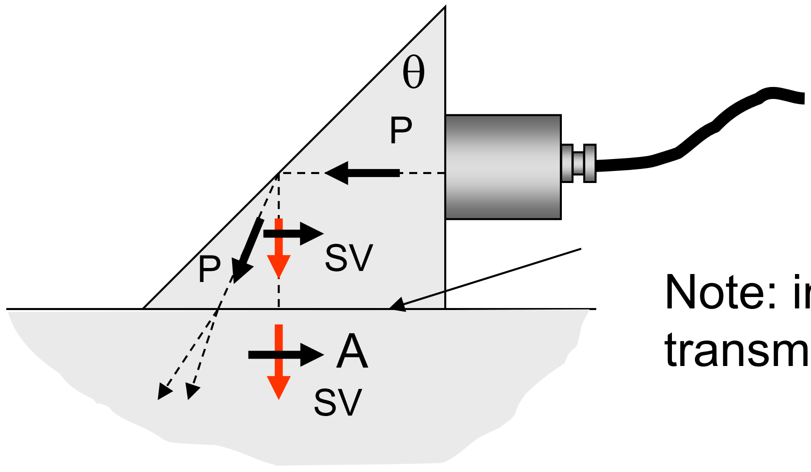
S.2 When a P-wave strikes a stress-free planar interface of an elastic solid, both reflected P-waves and SV-waves are generated:



The MATLAB function stress_freeP can be used to obtain the reflection coefficients (based on velocity ratios) for these reflected waves. The polarizations of these waves (i.e. the assumed directions of the velocity) are shown by the black arrows.

(a) plot these reflection coefficients versus the incident angle for angles from 0 to 90 degrees). Justify the values you obtain for these reflection coefficients when the incident angle is zero.

The mode conversion present in this free surface interaction can be used to generate an SV-wave traveling normal to the surface of a component that is being inspected by using a P-wave transducer and a wedge as shown:

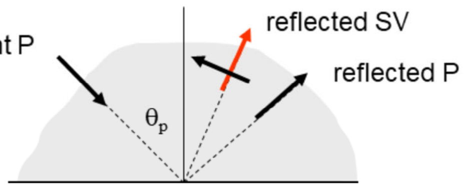


Note: interface needs shear-transmitting couplant

(b) Determine the wedge angle, θ, that is needed to generate a normally incident SV-wave if the wedge is made of aluminum.

(c) Determine the amplitude, A, of velocity of the SV-wave transmitted into the component being inspected if the component is made of steel. Assume the incident P-wave amplitude is unity and assume perfect contact between the aluminum wedge and the steel. Use plane wave relations to determine A even though the transducer does not generate purely plane waves.

Prob S.2



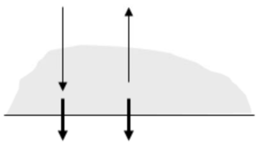
This problem considers the reflection of a plane P-wave off a free surface, where reflected plane P- and SV-waves are generated.

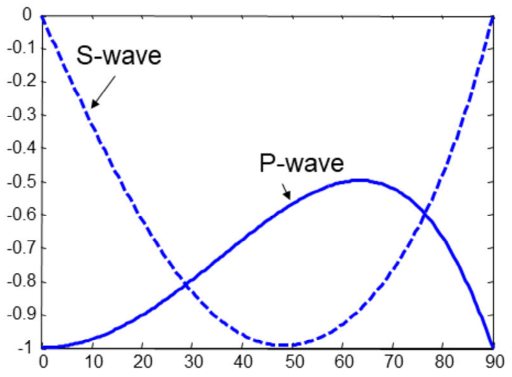
(a)

```
>> ang = linspace(0, 90, 200);
>> [rp , rs] =stress_freeP(ang, 6420, 3040);
>> plot(ang, rp)
>> hold on
>> plot(ang, rs, '--')
```

reflection coefficients

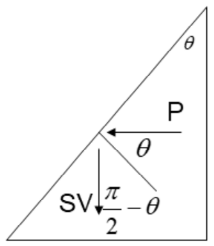
at normal incidence there is no mode conversion and the velocity of the reflected P-wave is equal to that of the incident P-wave at the free surface. The sign is negative since the velocity of the reflected wave is opposite to the direction of propagation.





If we generate angles from 0 to 90 degrees and use them in the stress_freeP function we get the reflected wave amplitudes shown plotted here. At zero degrees (normal incidence) there is no mode conversion so the reflected shear wave coefficient, rs, is zero. The reflected SV-wave amplitude is -1 since at a free surface the incident velocity (or displacement) amplitude is just doubled, but the direction of the reflected SV wave is opposite to that of the incident wave so its amplitude in its propagating direction must be -1.

(b)



$$\frac{\sin\theta}{c_p}=\frac{\sin(\pi/2-\theta)}{c_s} \Rightarrow \tan\theta=\frac{c_p}{c_s}$$

for aluminum

$\tan\theta = 6420/3040$

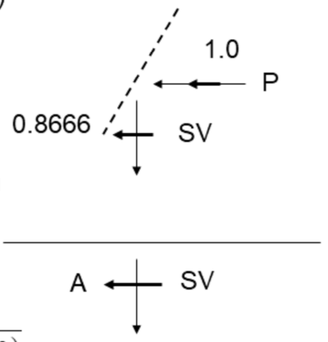
giving $\theta = 64.66^{o}$

For an incident P-wave reflecting off the Al wedge, the geometry shows that to get a reflected SV-wave traveling normal to the base of the wedge we must have the angle θ to be about 64.7 degrees. This is the angle of the wedge and also the angle the incident P-wave makes with the inclined wedge surface.

(c)

```
>> ang =64.66;
>> [rp,rs] = stress_freeP(ang, 6420,3040)
rp = -0.4991
rs = -0.8666
```

$\longrightarrow$ minus sign means polarization is as shown



$$T = \frac{2\rho_1 c_{s1}}{\rho_1 c_{s1} + \rho_2 c_{s2}} = \frac{(2)(2.7)(3040)}{(2.7)(3040)+(7.9)(3200)}$$
$$= 0.49$$

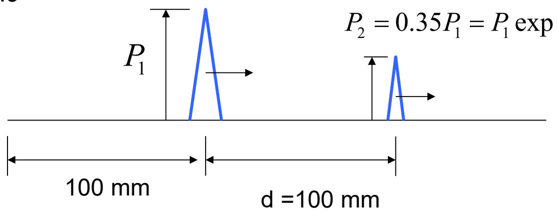
A = (0.49)(0.8666) =0.425

We can use stress_freeP to get the SV-wave amplitude, which is negative, since the polarization of the SV-wave is opposite to that assumed in deriving the reflection coefficient. If we calculate the transmission coefficient for the Al-steel interface at normal incidence, assuming perfect welded contact, we get an amplitude of the SV-wave traveling at normal incidence to the steel surface of about 43 percent of the original incident P-wave amplitude.

S.3. A plane wave travels 100 mm in a material to a point where its amplitude is P_1 . After the wave travels through an additional 100 mm of material its amplitude is reduced to $P_2 = 0.35\ P_1$. What is the average attenuation of this material in dB/m ?

S.3

$$P_2 = 0.35P_1 = P_1 \exp(-\alpha d)$$



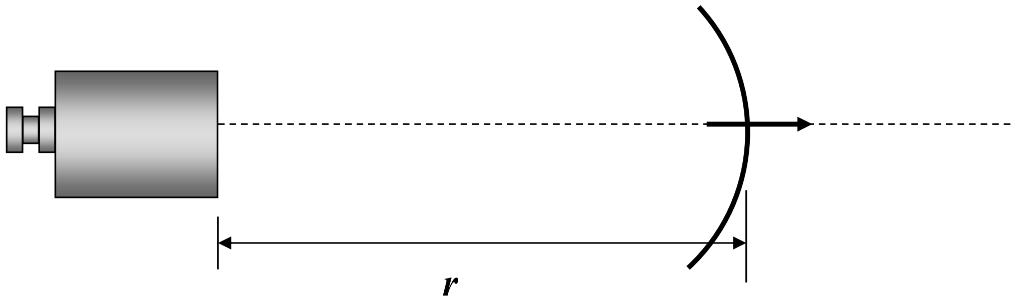
$$\Delta P_{dB} = 20\log_{10}\left(\frac{P_2}{P_1}\right)$$
$$= 20\log_{10}(0.35)$$
$$= -9.12\ dB$$

attenuation $\alpha_{dB/l} = \frac{|\Delta P_{dB}|}{d} = \frac{9.12}{0.1} = 91.2\ dB/m$

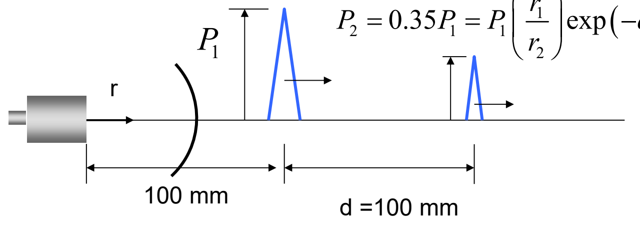
$$\alpha_{dB/m} = 8.686\,\alpha_{Np/m} \Rightarrow \alpha_{Np/m} = 10.5$$

Here, we are determining the average attenuation of a plane wave by examining the change in amplitude of the wave over a given distance. We can express the result in terms of decibels/unit length or in Nepers/unit length.

S.4. A transducer beam spreads as it propagates. In the far-field of the transducer this spreading is just like that of a spherical wave, i.e. it varies as 1/r, where r is the distance from the transducer. At a distance of 100 mm from the transducer the amplitude of the pressure is P_1. After the beam has propagated an additional 100 mm the amplitude is reduced to $P_2 = 0.35\ P_1$. What is the average attenuation of the material in dB/m ?



S.4



$$P_2 = 0.35 P_1 = P_1 \left(\frac{r_1}{r_2}\right) \exp(-\alpha d)$$

geometric spreading of a spherical wave $\quad \dfrac{P_2}{P_1} = \dfrac{r_1}{r_2}$

$$\left(\Delta P_{dB}\right)_{spreading} = 20 \log_{10}\left(\frac{r_1}{r_2}\right) = 20 \log_{10}\left(\frac{100}{200}\right) = -6.02\ dB$$

total amplitude change

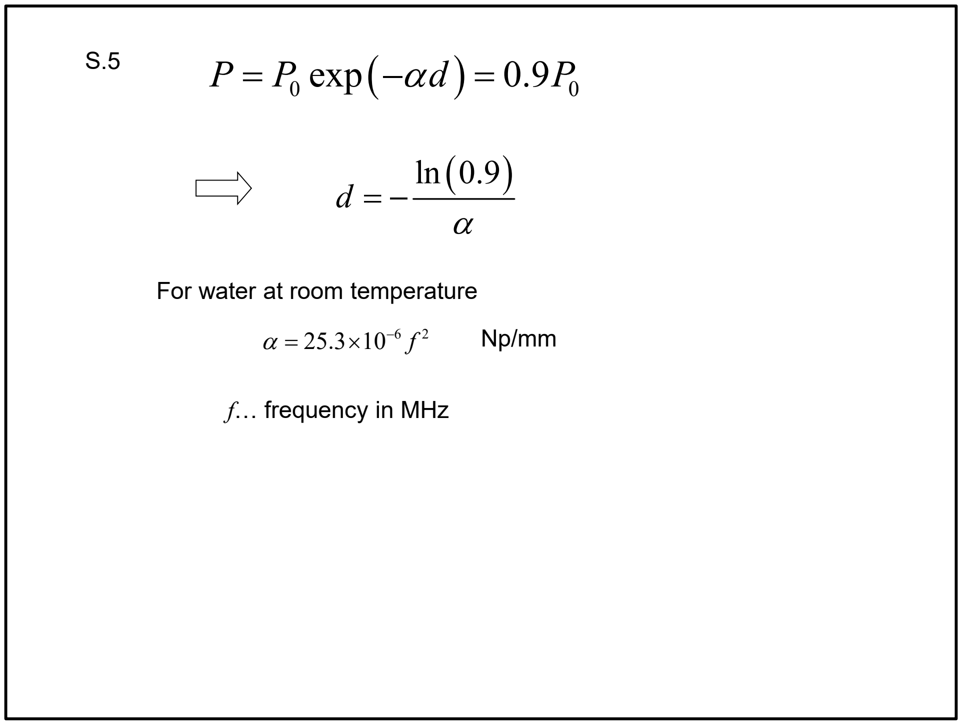
$$\left|\Delta P_{dB}\right| = 20 \log_{10}(0.35) = 9.12\ dB$$

attenuation $\quad \alpha_{dB/l} = \dfrac{\left|\Delta P_{dB}\right| - \left|\left(\Delta P_{dB}\right)_{spreading}\right|}{d} = \dfrac{9.12 - 6.02}{0.1} = 31\ dB/m$

$$\alpha_{dB/m} = 8.686\, \alpha_{Np/m} \quad \Rightarrow \quad \alpha_{Np/m} = 3.57$$

In problem S.3 we examined the attenuation of a plane wave, where amplitude changes were only due to attenuation . Here, for a spherical wave there are amplitude changes due to both wave spreading and attenuation, so we must account for and remove the wave spreading. In this case we see that the beam spreading produces a 6 dB drop in amplitude, so we must subtract off this 6B part from the total amplitude change.

S.5. Consider a harmonic plane P-wave traveling in water at room temperature. Determine an expression for the distance (as a function of frequency) that this wave must travel to reduce its amplitude by 10% due to attenuation. Plot this function for f = 1 MHz to f = 20 MHz.



S.5 $$P = P_0 \exp\left(-\alpha d\right) = 0.9 P_0$$

$$\Rightarrow \qquad d = -\frac{\ln\left(0.9\right)}{\alpha}$$

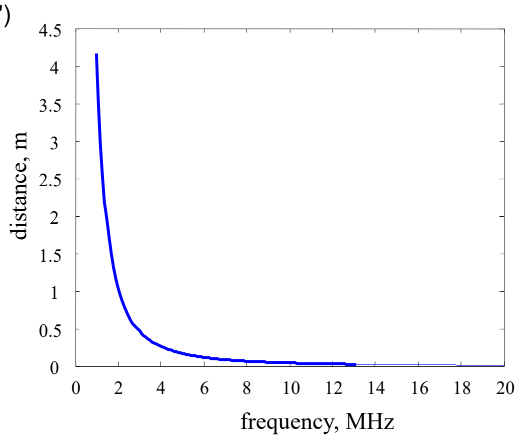
For water at room temperature

$\alpha = 25.3 \times 10^{-6} f^2$ Np/mm

f… frequency in MHz

From the amplitude expression for changes due to attenuation of a plane wave we can determine the distance that must be traveled to reduce the amplitude by 10 percent. But for water we know the attenuation as a function of frequency so we know the distance that must be traveled also as a function of frequency.

```
%attenuation_script
clear
f =linspace(1, 20, 200);
alpha = (25.3E-03)*f.^2; % f in MHz, alpha in Np/m
d =-log(0.9)./alpha ;  %distance in meters
plot(f, d)
xlabel('frequency, MHz')
ylabel('distance, m')
```



This Matlab script uses the relations of the previous page and plots the distance traveled in meters versus frequency. As expected, for higher frequencies, we do not need to travel as long a distance to have a 10 percent change in amplitude.

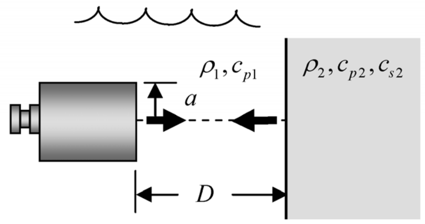


Fig. 5.5. An ultrasonic pulse-echo calibration setup where the waves generated by a circular, planar, piston transducer are reflected from a plane fluid-solid interface at normal incidence and the reflected waves are received by the same transducer.

attenuation of water:

$$\alpha_w(f) = 25.3 \times 10^{-6} f^2 \text{ Np/mm} \qquad (5.21)$$

where *f* is the frequency in MHz.

$$t_A(\omega) = \tilde{D}_p\left(k_p a^2 / 2D\right) R_{12} \exp\left(2ik_p D\right) \exp\left[-2\alpha_w(f) D\right] \qquad (5.22b)$$

where

$$\tilde{D}_p(u) = 2\left[1 - \exp(iu)\left\{J_0(u) - iJ_1(u)\right\}\right].$$

5.1 Using Eqs. (5.21) and (5.22b) write a MATLAB function t_a that computes the acoustic/elastic transfer function for the pulse-echo setup shown in Fig. 5.5, where the fluid is water at room temperature. The calling sequence for this function should be:

```
>> t =t_a(f, a, d, d1, d2,c1,c2);
```

where f is the frequency (in MHz), a is the radius of the transducer (in mm), d is the distance from the transducer to the plane surface (in mm), d1 is the density of the fluid (in gm/cm^3), c1 is the compressional wave speed of the fluid (in m/sec), d2 is the density of the solid (in gm/cm^3), and c2 is the compressional wave speed of the solid (in m/sec).

Using this function, obtain a plot of the magnitude of this transfer function versus frequency for a = 6.35 mm, d = 100 mm, d1 = 1.0 gm/cm^3, c1 = 1480 m/sec, d2 = 7.9 gm/cm^3, c2 = 5900 m/sec (steel). Let the frequencies range from 0 to 20 MHz. On the same plot, show the magnitude of this transfer function versus frequency when the attenuation of the fluid is neglected, so that the effects of attenuation on this function can be demonstrated.

Prob 5.1

```
function y = t_a(f, a, d, d1, d2, c1, c2)
%t_a (f, a, d1, d2, c1, c2) calculates the acoustic/elastic
%transfer function for the reflection of the wave field off a
% planar surface of a solid at a distance d (in mm) from a circular
% immersion piston transducer of radius a (in mm). The frequency,
% f is in MHz and the densities of the fluid and solid (d1, and d2),
% are in gm/cm^3. The wavespeeds c1, c2 of the fluid and solid,
% respectively, are in m/sec. The phase term exp(2*i*k*d) in
% this transfer function is omitted.

ka = 2000*pi*f*a./c1;
a2d = a/(2*d);
R12 = (d2*c2 -d1*c1)/(d1*c1 + d2*c2);
att =(25.3E-06)*f.^2; % edit out one term here for attenuation
% att =0.;
u = ka.*a2d;
D = 2.*(1-exp(i.*u).*(besselj(0,u) - i.*besselj(1, u)));
y= R12.*D.*exp(-(2*d)*att);
```

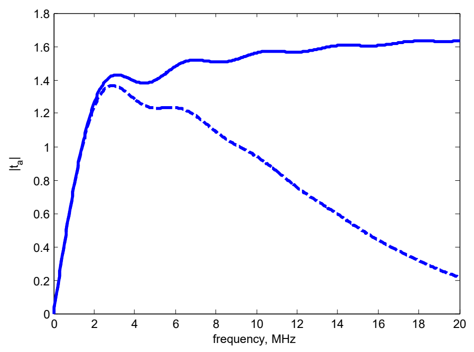
Here is the Matlab function for the calculation of the transfer function (without the phase term) when attenuation is present. To simplify the function the zero attenuation case is obtained by editing the function so that the attenuation, att, is zero, saving it, and then re-evaluating the function.

```
>> f =linspace(0,20, 500);
>> t =t_a(f, 6.35, 100, 1.0, 7.9, 1480, 5900);
>> plot(f, abs(t))
>> hold on
>> % edit function to make attenuation term zero, save and then evaluate
again:
>> t =t_a(f, 6.35, 100, 1.0, 7.9, 1480, 5900);
>> plot(f, abs(t), '--')
```

In this case:

R12 = 0.9384

purely plane wave result:
|t_a |= constant =(2)(R12) = 1.88



Here is the evaluation of the acoustic/elastic transfer function in the two cases. We also show the case where the transfer function is calculated for a purely plane wave (without attenuation) to show what happens when we both neglect attenuation and wave diffraction effects. The factor of two is present in the plane wave result because the transfer function is calculated with the blocked force on reception, which is twice that of the force in the reflected wave which is received. Obviously, both diffraction effects and attenuation are important here, as they are for most ultrasonic NDE problems.

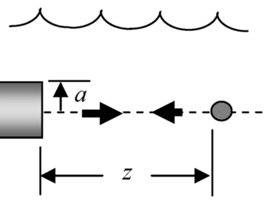


Fig. 6.15. A circular piston transducer of radius *a* receiving the waves reflected from the front surface of a spherical reflector located on the central axis of the transducer.

$$a_{eff} = \sqrt{2\lambda z_{\min}}\,. \tag{6.14}$$

6.1 The MATLAB function transducer_x(z) returns the time-domain sampled voltage received from a spherical reflector in water (c = 1480 m/sec) located at a distance z (in mm) along the axis of a planar transducer as shown in Fig. (6.15). There are 1024 samples in this waveform, each separated by a sampling time interval Δt =.01 μsec. First, let z be the vector of values:

```
>> z = linspace (25, 400, 100);
```

Use this set of values in the transducer_x function, i.e. evaluate

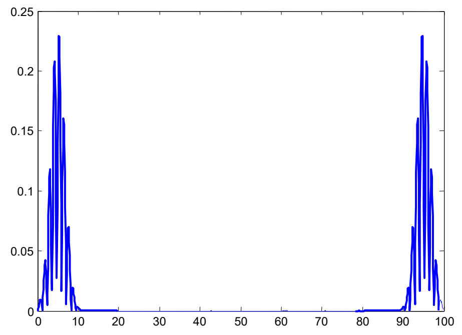
```
>> V = transducer_x(z);
```

The matrix V will contain 100 waveforms calculated at each of these z-values. Use FourierT to generate the frequency spectra of these waveforms. Note that FourierT can operate on all of these waveforms at once as long as they are in columns (which is the case) and will return a matrix of the corresponding spectra, also in columns. Examine the magnitude of some of these spectra versus frequency to determine the range of frequencies over which there is a significant response. Pick one frequency value near the center frequency in this range and plot the magnitude of the spectra at that value versus the distance z.

Locate the last on-axis minimum in this plot and use Eq. (6.14) to determine the effective radius of this transducer. Try using a different frequency value within the transducer bandwidth to determine the effective radius. Does your answer vary with the frequency chosen?

Prob 6.1

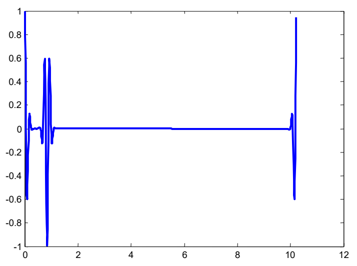
```
>> z = linspace(25, 400, 100);
>> V = transducer_X(z);
>> dt = 0.01;
>> Vf = FourierT(V, dt);
>> f = s_space(0, 1/dt, 1024);
>> plot(f, abs(Vf(:, 1)))
```



Here we generate a series of 100 z-values for a small sphere placed on the axis of an immersion transducer and evaluate the function transducer_X which models the 100 received sampled voltages versus time received by a planar piston transducer. The sampling interval is 0.01 microseconds. We can compute the Fourier transform of all 100 signals with FourierT and set up the appropriate sampled frequencies with s_space. We can plot the magnitude of any of spectra. Here, we arbitrarily choose to use the very first spectra (obtained from z = 24 mm). We see both the positive and negative frequencies and we see that most of the frequency content is centered around 5 MHz.

this spectrum corresponds to the following waveform:

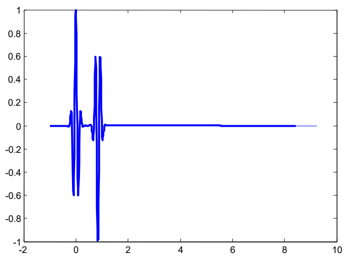
```
>> t= s_space(0, 10.24, 1024);
>> plot(t, V(:,1))
```



we need to shift this to see waves clearly

```
>> plot(t_shift(t, 100), c_shift(V(:,1)', 100))
```

turn into row vector

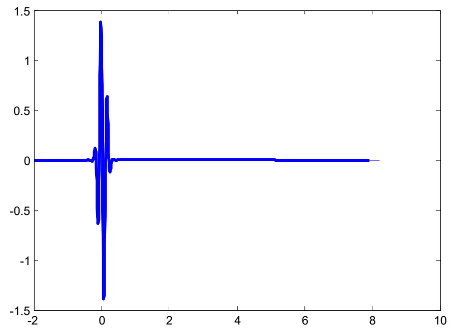


this is in the very near field so we can see the direct and edge waves

If we examine the waveform from this first spectra by generating an appropriate set of sampled times with s_space, we see the signal appears in the lower and upper half of the time domain plot since the signal has values for negative times which show up in the upper half of the time plot. To better see the signal we can use the functions c_shift and t_shift, which shows us the waveforms more clearly. Note that since the waveforms are in columns, before using c_shift we need to place the waveform in a row vector. We see there are two waves appearing here, which are the direct and edge waves coming from the piston transducer. We can see them separately here because we are very close to the transducer.

If we had chosen a point at a much larger distance, the direct and edge waves would "merge":

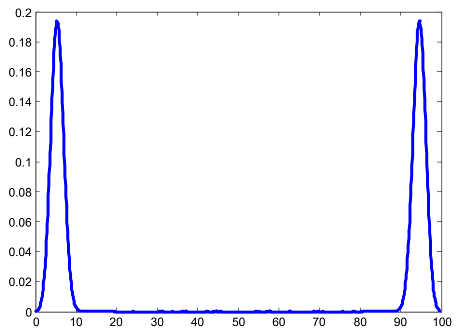
```
>> plot(t_shift(t,200)  , c_shift(V(:,90)', 200))
```



If we had examined one of the waveforms, as shown here, at a much larger distance from the transducer, the direct and edge waves merge into what looks like a single response.

and the spectrum would look like:

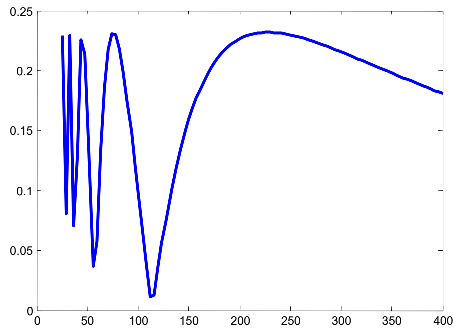
```
plot(f, abs(Vf(:,90)))
```



The magnitude of the spectrum of the signal at this larger distance is as shown. There is much less ringing in the frequency domain from the spectrum of the first choice since such ringing is caused here by the presence of two signals (the direct and edge waves) separated by a time Δt, and as Δt gets smaller the ringing gets less pronounced. Here, we can see more clearly the bandwidth of the signal.

Now, find a good frequency component value to plot versus z:

```
>> find( abs(Vf(:,1)) == max(abs(Vf(:,1))))

ans =  55   971
>> f(55)
ans = 5.2734

>> vp = Vf(55,:);
>> plot(z, abs(vp))
```

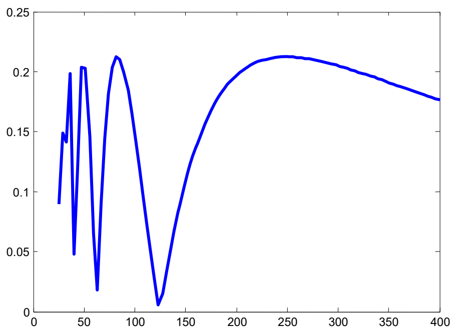


Now, let’s go back to the magnitude of the spectrum for out first choice of signal and compute the value of the frequency at the maximum frequency response. We see we get two values because of the presence of the negative frequencies, so we choose the smaller value which we see is at a positive frequency of about 5 MHz (which is consistent with what we expected). If we plot the magnitude of the frequency response versus z we see the on-axis response discussed in the notes, where there are rapid variations in the near field, followed by a monotonic decay as we go to the far field.

```
>> find( abs(vp) == min(abs(vp)))

ans =24

>> z(24)

ans =112.1212

>> aeff = sqrt(2*1.48*z(24)/f(55))

aeff = 7.9331  mm
```

We can try to find the last minimum of the on-axis response, which we see here occurs at about 112 mm, which is consistent with the plot seen on the previous slide. We can use this minimum location and the frequency to calculate an effective radius for the transducer.

```
>> f(60)

ans =5.7617

>> vp = Vf(60,:);
>> plot(z, abs(vp))
>> find( abs(vp) == min(abs(vp)))
ans =27

>> z(27)
ans =123.4848

>> aeff = sqrt(2*1.48*z(27)/f(60))
aeff = 7.9648  mm
```



If we had chosen a somewhat larger frequency of about 5.8 MHz (but still well within the bandwidth of the signal) and plotted the on-axis frequency response, we can locate the last on-axis minimum, as before, and calculate an effective radius, which is within a few percent of our previous value.

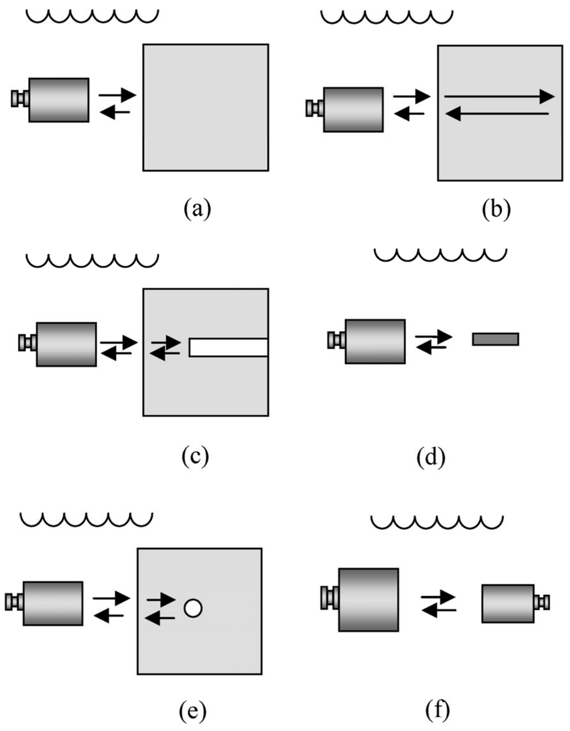


Fig. 7.2. Reference experiments that can be used to determine the system function or system efficiency factor where circular planar transducers are involved: **(a)** reflection from a plane front surface of a block at normal incidence, **(b)** reflection from the back surface of a block at normal incidence, **(c)** reflection from an on-axis flat-bottom hole at normal incidence, **(d)** reflection from an on-axis solid cylinder at normal incidence, **(e)** reflection from an on-axis side-drilled hole at normal incidence, and **(f)** two transducers (not necessarily the same) whose axes are aligned.

7.1 The beam of a planar immersion transducer is reflected off the front surface of a steel block (see Fig. 7.2 (a)) and this reference signal can be used to determine the system function. The file FBH_ref contains a sampled reference signal of this type and its corresponding sampled times. Place this file in your current MATLAB directory and then load it with the MATLAB command

```
>> load( ' FBH_ref ' )
```

This command will place in the MATLAB workspace 1000 sampled time values in the variable t_ref, and a 1000 point reference time domain waveform in the variable ref. Plot this waveform. Take the FFT of this reference waveform and keep only the first 200 values of the resulting 1000 point spectrum (from 0 to 20 MHz) in a variable, Vc. Plot the magnitude of Vc from 0 to 20 MHz. Use Vc and the data given below to determine the system function via deconvolution (using a Wiener filter) and plot the magnitude of this system function versus frequency from zero to 20 MHz. Compare this system function with Vc. Use the acoustic/elastic transfer function for this configuration as:

$$\begin{aligned} t_A &= 2R_{12}\exp\left(-2\alpha_{p1}D\right)\Big[1-\exp\left(ik_{p1}a^2/2D\right) \\ &\cdot\left\{J_0\left(k_{p1}a^2/2D\right)-iJ_1\left(k_{p1}a^2/2D\right)\right\}\Big] \end{aligned}$$

where we have dropped the phase term $\exp\left(2ik_pD\right)$ as it only produces a time delay and the plane wave reflection coefficient is:

$$R_{12}=\frac{\rho_2 c_{p2}-\rho_1 c_{p1}}{\rho_2 c_{p2}+\rho_1 c_{p1}}$$

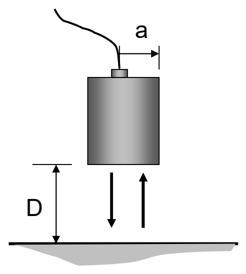
The parameters for this setup are:

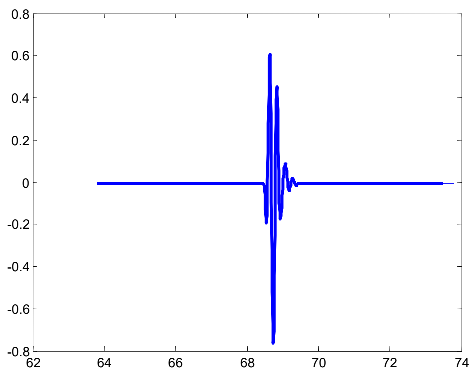
$\rho_1 = 1.0$, $\rho_2 = 7.86$: density of the water and steel, respectively (gm/cm^3)
$c_{p1} = 1484$, $c_{p2} = 5940$: P-wave speeds of the water and steel, respectively (m/sec)
$\alpha_{p1} = 24.79\times10^{-6} f^2$: water attenuation (Np/ mm) with *f* the frequency (in MHz)
$D = 50.8$: distance from the transducer to the block (mm)
$a = 6.35$: radius of the transducer (mm)
$e = 0.3$: noise coefficient for the Wiener filter

Note that the Bessel functions are available directly in MATLAB. The Bessel function of order zero, $J_0\left(x\right)$, is given by the MATLAB function BesselJ(0, x) and the Bessel function of order one, $J_1\left(x\right)$, is given by the MATLAB function BesselJ(1, x).

7.1 System Function

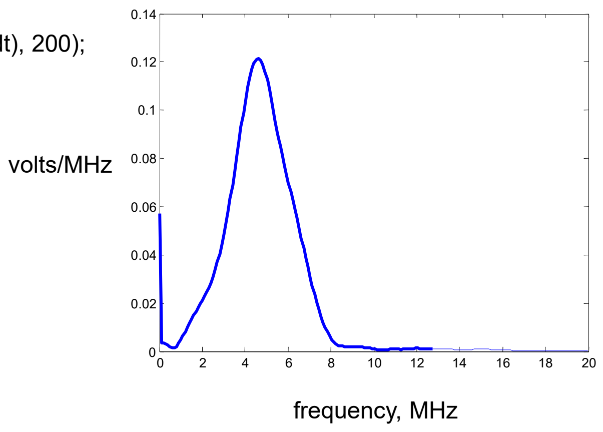
```
>> load('FBH_ref')
>> plot(t_ref, ref)
```





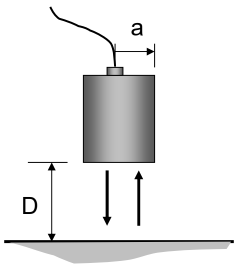
If we load the flaw FBH_ref we obtain a series of sampled times, t_ref, and the values of the measured time domain signal, ref, we receive when the transducer beam is reflected off the front planar surface of a block, as shown. This is a simple configuration where we can model the acoustic/elastic transfer function and hence obtain the system function by deconvolution.

```
>> dt = t_ref(2) -t_ref(1)

dt =
   0.0100

>> Vf =FourierT(ref, dt);
>> Vc=Vf(1:200);
>> f =s_space(0, 1/(5*dt), 200);
>> plot(f, abs(Vc))
```



To compute the Fourier transform we need the sampling time interval, which shows that a 100 MHz sampling frequency was used here. If we compute the first 200 of the 1000 corresponding sampled frequencies we will obtain the frequency response from 0 to 20 MHz, approximately, contained in the Vc function. We see that most of the frequency content of the signal is between, say, 1 and 8 MHz.

```
% sys_function_script
clear
d1 =1.0;
d2 = 7.86;
c1 = 1484;
c2 =5940;
D=50.8;
a =6.35;
e =0.03;
load('FBH_ref')
dt = t_ref(2) -t_ref(1);
Vf =FourierT(ref, dt);
Vc=Vf(1:200);
```



$$V_c(f) = s(f)\,t_A(f)$$

$$t_A = 2R_{12}\exp(-2\alpha_{p1}D)$$
$$\cdot\left[1-\exp\left(ik_{p1}a^2/D\right)\left\{J_0\left(k_{p1}a^2/D\right)-iJ_1\left(k_{p1}a^2/D\right)\right\}\right]$$

$$R_{12} = \frac{\rho_2 c_{p2} - \rho_1 c_{p1}}{\rho_2 c_{p2} + \rho_1 c_{p1}}$$

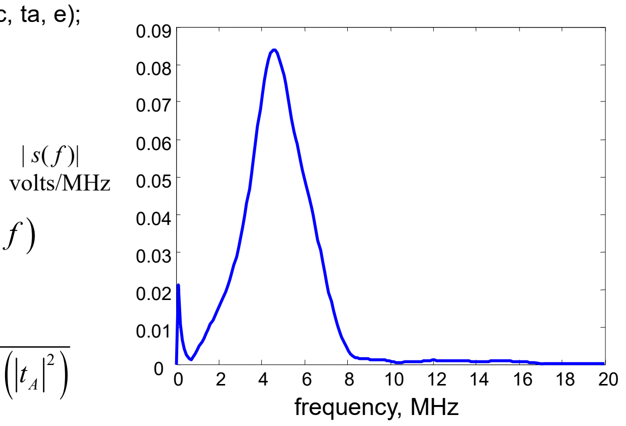
Here is a Matlab script, sys_function_script, that calculates the system function. On this page we see the script generates the given values for the parameters in this problem which are the densities of the two media, their P-wave speeds, the distance to the front surface, the radius of the transducer, and the noise constant e we will use in the Wiener filter. We load the measured signal, ref, and time axis, t_ref, and compute the frequency components, Vc, from 0 to 20 MHz, approximately.

```
f =s_space(0, 1/(5*dt), 200);
alpha = (24.79E-06)*f.^2;
atten =exp(-2*D*alpha);
R12 =(d2*c2 -d1*c1)/(d2*c2 +d1*c1);
kp1a =2000*pi*a*f./c1;
r=a/(2*D);
x=r*kp1a;
ta=2*R12*atten.*(1 -exp(i*x).*(BesselJ(0,x) -i*BesselJ(1,x)));
s=Wiener_filter(Vc, ta, e);
plot(f, abs(s))
```

$$ka = 2\pi fa/c$$
$$= \frac{2\pi\left(f\times 10^6\, MHz\right)\left(a\ mm\right)}{\left(c\ m/\sec\right)\left(10^3\, mm/m\right)}$$
$$= 2000\pi fa/c$$

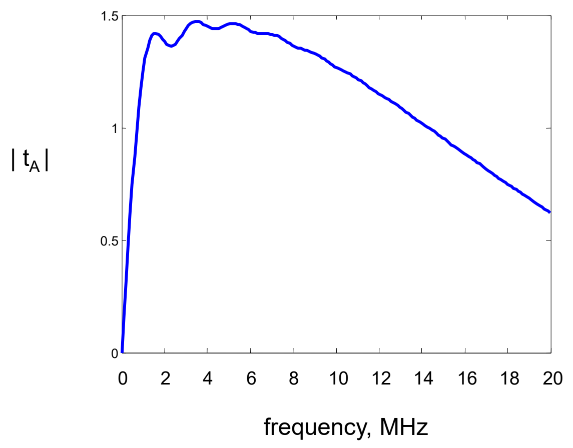
$$V_C\left(f\right) = s\left(f\right)t_A\left(f\right)$$

$$s = \frac{V_c\, t_A^*}{\left|t_A\right|^2 + e^2 \max\left(\left|t_A\right|^2\right)}$$



The script continues by generating the appropriate sampled frequencies and computes the attenuation factor for the water. The value of the reflection coefficient and the nondimensional wavenumber, kp1a, for the water are evaluated, as well as a non-dimensional radius, r, as these can be used to generate the expression for the acoustic/elastic transfer function (see the next slide for the behavior of the transfer function). The function Wiener_filter is then used to do the deconvolution of the spectrum of the measured signal with this transfer function and the magnitude of this system function is plotted versus frequency.

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

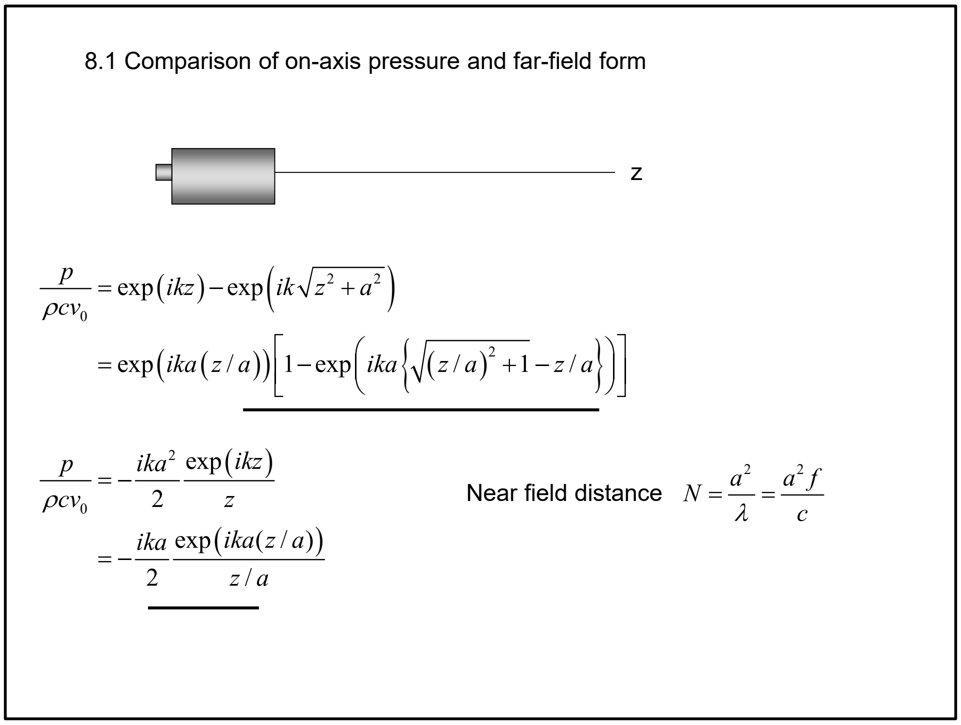


Here is the plot of the magnitude of the acoustic/elastic transfer function.

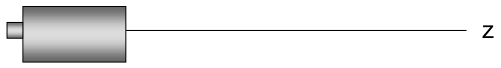
$$p\left(z,\omega\right)=\rho c v_0\left(\omega\right)\left[\exp\left(ikz\right)-\exp\left(ik\sqrt{z^2+a^2}\right)\right]. \tag{8.20}$$

$$p\left(z,\omega\right)=\frac{-i\omega\rho a^2 v_0\left(\omega\right)}{2}\frac{\exp\left(ikz\right)}{z}, \tag{8.21}$$

8.1 The exact on-axis pressure for a circular piston transducer was given by Eq. (8.20) and the far field approximation for this same pressure was given by Eq. (8.21). Using MATLAB, write a script that computes these two pressure expressions and plots the magnitude of the normalized pressure, $p/\rho c v_0$, versus the normalized distance, z/N, for both of these expressions on the same plot, where N is the near field distance. Let the transducer radius a = 6.35 mm, the frequency f = 5 MHz, and the wave speed of the fluid c = 1480 m/sec. Show both pressure plots over the range z/N = 0.2 to z/N = 4.0. What can you conclude about when Eq. (8.21) is valid?



8.1 Comparison of on-axis pressure and far-field form



$$\frac{p}{\rho c v_0} = \exp(ikz) - \exp\left(ik\sqrt{z^2+a^2}\right)$$

$$= \exp\left(ika(z/a)\right)\underline{\left[1-\exp\left(ika\left\{\sqrt{(z/a)^2+1}-z/a\right\}\right)\right]}$$

$$\frac{p}{\rho c v_0} = -\frac{ika^2}{2}\frac{\exp(ikz)}{z}$$

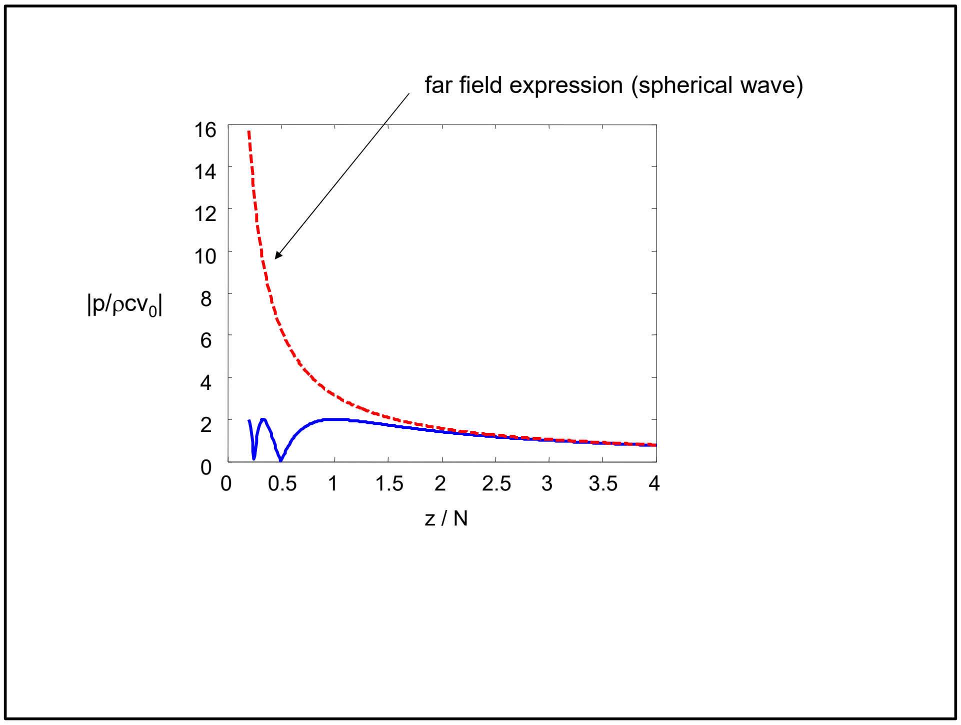
$$= \underline{-\frac{ika}{2}}\frac{\exp(ika(z/a))}{z/a}$$

Near field distance $N = \dfrac{a^2}{\lambda} = \dfrac{a^2 f}{c}$

Here are the exact and far field on-axis responses of a circular piston transducer. Note that the near field distance, N, used here is an approximate expression for the exact near field distance but one that is good for most NDE applications.

```
% onaxis_nf_ff       script
clear
f=5;                                  %frequency (MHz)
a =6.35                               % transducer radius (mm)
c =1480;                              % wave speed (m/sec)
ka = 2000*pi*f*a./c;                  % ka value
N = 1000*(a^2)*f/c;                   % near field distance (mm)
zN = linspace (0.2, 4, 500);          %normalized distance z/N
z= zN.*N;                             % actual on-axis distance (mm)
zn = z/a;                              % normalized on-axis distance, z/a

pe = 1 -exp(i*ka.*(sqrt(zn.^2 +1 )-zn));          %on-axis normalized pressure
                                                   % with exp(ikz) term removed

pff = -i*ka./(2*zn);              % far field pressure with exp(ikz) term removed

plot(zN, abs(pe),'b')
hold on
plot(zN, abs(pff), 'r--')
hold off
xlabel('z/N')
ylabel('|p/\rhocv_0|')
```

Here is the Matlab script to compute the exact on-axis pressure, pe, and the far-field on-axis pressure, pff. The factors of 2000 and 1000 in the ka and N expressions appear because of the mixture of units of the wave speed , radius, and frequency that appear in these expressions (examine them more carefully yourself).



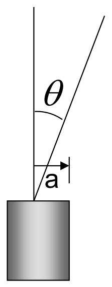
Here are the plots. The two expressions are close for distances greater than the **Rayleigh distance**, which is defined as $\pi a^2/\lambda$. Thus, the expressions agree for distances about three times the near field distance.

$$p(\mathbf{x},\omega) = -i\omega\rho a^2 v_0(\omega)\frac{J_1(ka\sin\theta)}{ka\sin\theta}\frac{\exp(ikR)}{R}, \tag{8.31}$$

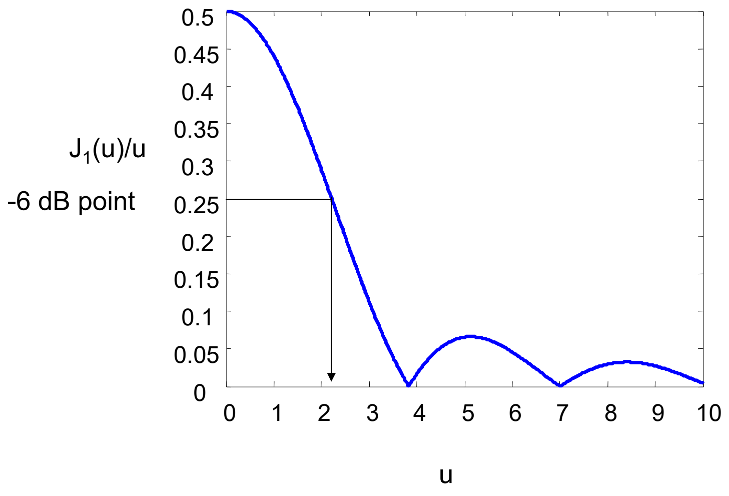
8.2 Equation (8.31) shows that the angular distribution of the far field radiation field of a circular planar piston transducer is controlled by the directivity function $J_1(ka\sin\theta)/(ka\sin\theta)$. Using MATLAB, write a function that calculates the angle where the amplitude of this directivity function drops by 6 dB from its maximum on-axis value. Use this function to determine the 6 dB angular spread of a 0.5 inch diameter piston transducer radiating into water at frequencies of 2.25, 5, and 10 MHz.

8.2 Far field Directivity function of a piston transducer of radius a

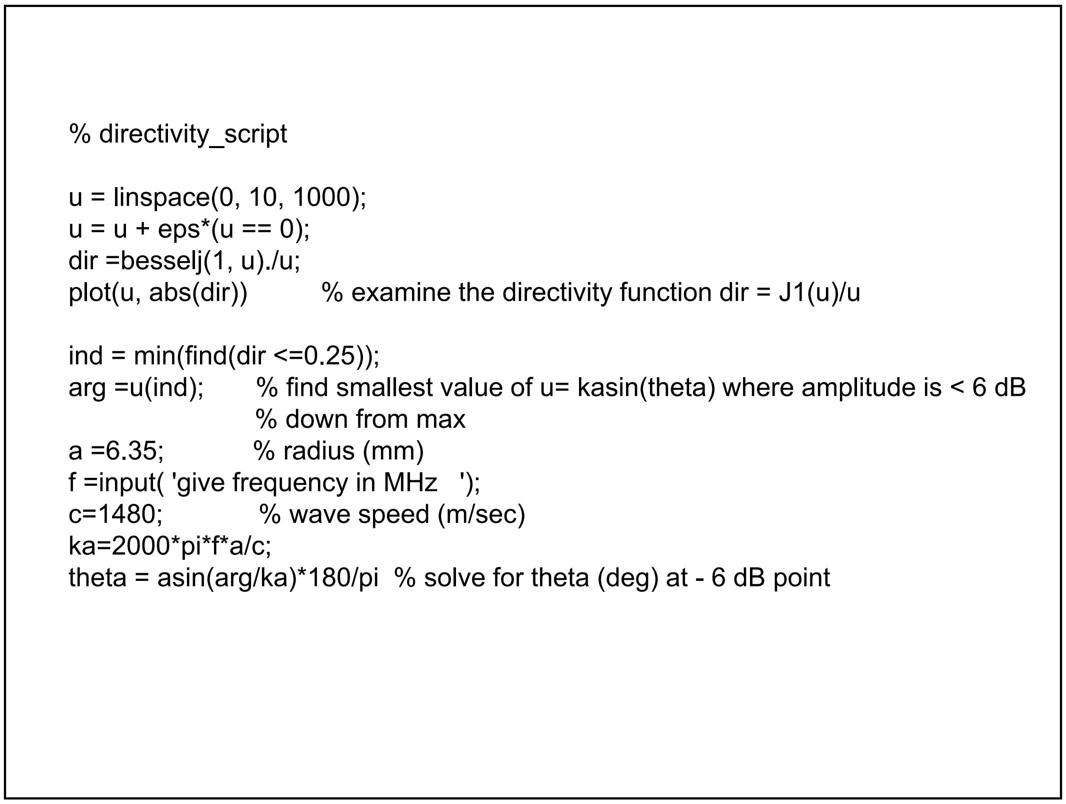
$$J_1\left(ka\sin\theta\right)/ka\sin\theta$$



Let u = ka sin(θ)



Shown is the directivity function that defines the far field angular behavior of a circular piston transducer. We can see when the amplitude is ½ of the maximum value of 0.5 we have the corresponding u value. From that u value we need to determine the actual corresponding angle.



```
% directivity_script

u = linspace(0, 10, 1000);
u = u + eps*(u == 0);
dir =besselj(1, u)./u;
plot(u, abs(dir))         % examine the directivity function dir = J1(u)/u

ind = min(find(dir <=0.25));
arg =u(ind);       % find smallest value of u= kasin(theta) where amplitude is < 6 dB
                   % down from max
a =6.35;           % radius (mm)
f =input( 'give frequency in MHz   ');
c=1480;            % wave speed (m/sec)
ka=2000*pi*f*a/c;
theta = asin(arg/ka)*180/pi  % solve for theta (deg) at - 6 dB point
```

Here is the directivity script that calculates the angle of spread at the -6dB point. First the directivity is evaluated in terms of the non-dimensional quantity u = kasinθ, which is then plotted (the plot seen on the previous page. Then we examine all values of the directivity less than or equal to 0.25. The smallest of these directivities will then be the nearest value where the 0.25 (-6 dB) value occurs and the value of u at the same index value will be the value of u we seek. If we give a value for the radius, frequency (which must be entered by the user), and wave speed (for water in this case), we can calculate the ka value. The factor of 2000 exists in this expression because of the mixed units. Since we know the u value and the ka value, we can then calculate the corresponding angle.

```
>> directivity_script
give frequency in MHz   2.25

theta =

    2.0996

>> directivity_script
give frequency in MHz   5

theta =

    0.9446

>> directivity_script
give frequency in MHz   10

theta =

    0.4723
```

Here is the directivity for different frequencies entered by the user. Note that all of these angles are very small so the beams of such NDE transducers are highly directional and we say the beam is well-collimated.

Now see what happens if we change the material to steel

```
% directivity_script

u = linspace(0, 10, 1000);
u = u + eps*(u ==0);
dir =besselj(1, u)./u;
plot(u, abs(dir))         % examine the directivity function J1(u)/u

ind = min(find(dir <=0.25));
arg =u(ind);       % find smallest value of u= kasin(theta) where amplitude is < 6 dB

a =6.35;            % radius (mm)
f =input( 'give frequency in MHz   ');
c=5900;             % wave speed (m/sec)
ka=2000*pi*f*a/c;
theta = asin(arg/ka)*180/pi  % solve for theta (deg) at - 6 dB point
```

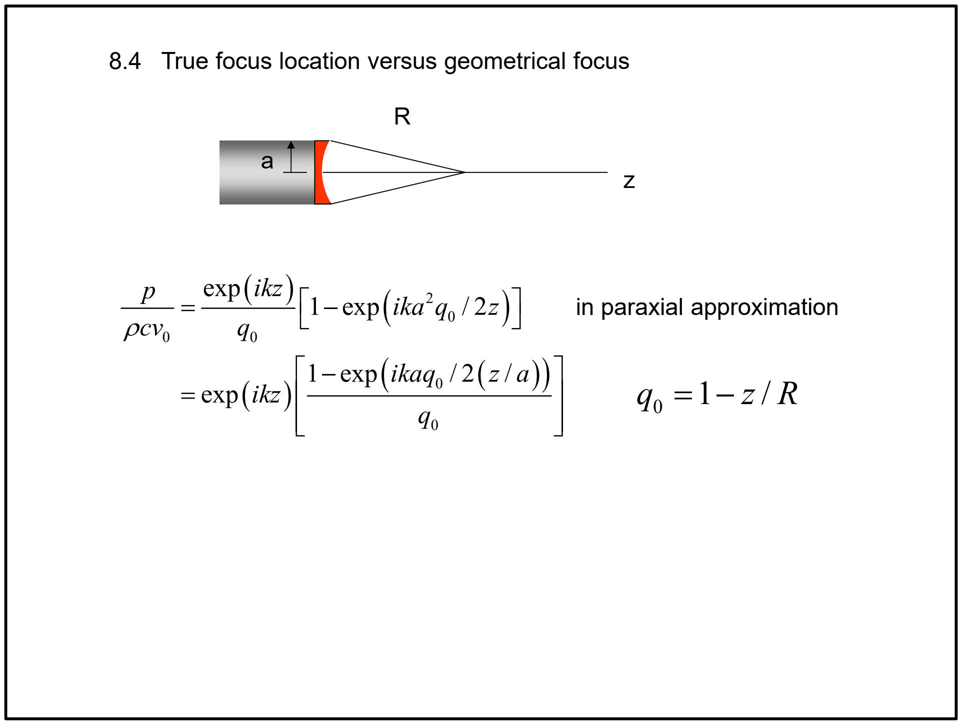
It was not asked for in the problem, but let us examine the same results when the transducer is radiating into steel. We simply change the wave speed in the script to be the P-wave speed in steel and rerun the script.

```
>> directivity_script
give frequency in MHz   2.25

theta =

   8.3981

>> directivity_script
give frequency in MHz   5

theta =

   3.7684

>> directivity_script
give frequency in MHz   10

theta =

   1.8832
```

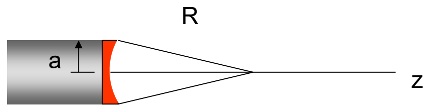
Here are the corresponding angles, which are larger than when the transducer is radiating into water, but still are small.

$$p(z,\omega) = \rho c v_0 \exp(ikz)\left\{\frac{1}{q_0}\left[1-\exp\left(ika^2 q_0 / 2z\right)\right]\right\}, \qquad (8.37)$$

8.4 Write a MATLAB function that returns the normalized on-axis pressure, $p/\rho c v_0$, versus distance for a spherically focused piston transducer (see Eq. (8.37)). The input arguments of the function should be the distance values (in mm), the frequency (in MHz), the radius (in mm), the geometrical focal length (in mm), and the wave speed (in m/sec). Use this function to find the location of the true focus (i.e. the distance to the maximum pressure) for a 12.7 mm (0.5 inch) diameter, 101.6 mm (4 in.) focal length transducer radiating into water at 5, 10, and 20 MHz. What can you conclude about the relationship between the location of the true focus versus the geometrical focal length?



8.4 True focus location versus geometrical focus



$$\frac{p}{\rho c v_0} = \frac{\exp(ikz)}{q_0}\left[1-\exp\left(ika^2 q_0/2z\right)\right] \qquad \text{in paraxial approximation}$$

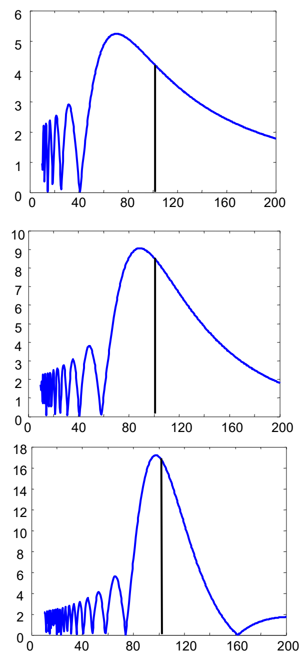
$$= \exp(ikz)\left[\frac{1-\exp\left(ikaq_0/2(z/a)\right)}{q_0}\right] \qquad q_0 = 1 - z/R$$

Here is the expression for the on-axis pressure of a spherical focused circular piston transducer.

```
          function p = onaxis_focused(z, f, a ,R, c)
          ka = 2000*pi*f*a./c;                    % ka value
          q =1 -z/R;                              % q0 value
          zn = z/a;                                % normalized on-axis distance, z/a

          p = (1./q).*(1 -exp(i*ka.*q./(2.*zn)));   %on-axis normalized pressure
                                                   % with exp(ikz) term removed


    % onaxis_focused_script
    clear
    z= linspace(10, 200, 500);  % set up z-values well beyond geometrical focus
    f= input(' frequency (MHz) =  ')
    a= 6.35;                  % radius (mm)
    R = 101.6;                 % geometrical focal length (mm)
    c=1480;                   % wave speed (m/sec)
    p= onaxis_focused(z, f, a, R, c);   % compute on-axis pressure
    pm =max(abs(p));                  % find value of max pressure
    ind =find(abs(p) == pm);
    zgeom = R
    ztrue =z(ind)                    %z(ind) is where max pressure is located
```

Here is the Matlab function requested in the problem and a script that uses that function to find where the maximum pressure occurs (the true focus) so we can compare it with the geometric focus location, which is just the R in our pressure expression.

```
>> onaxis_focused_script
 frequency (MHz) =  5

zgeom = 101.6000


ztrue = 70.5411

>> onaxis_focused_script
 frequency (MHz) =  10

zgeom = 101.6000


ztrue = 88.8176

>> onaxis_focused_script
 frequency (MHz) =  20

zgeom = 101.6000


ztrue = 97.5752
```



We see that at 5 MHz there is considerable difference between the true focus location and the location of the geometrical focus and it is only until about 20 MHz (a very high frequency in NDE testing) that the two locations are somewhat close. Here we have also plotted the on-axis magnitude of the pressure and shown the location of the geometrical focus relative to the true focus, which is at the maximum of these curves.

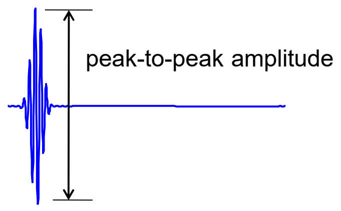
$$p(z,\omega) = \rho c v_0(\omega)\left[\exp(ikz) - \exp\left(ik\sqrt{z^2+a^2}\right)\right]. \qquad (8.20)$$

8.5 Equation (8.20) gives the exact on-axis pressure for a planar immersion transducer at a single frequency. Ultrasonic NDE transducers, however, do not normally operate at a single frequency but are driven by a voltage pulse and hence contain a spectrum of frequencies that generate a time domain pulse. The near field behavior of such a pulsed transducer does not show nearly the same strong near field structure as a single frequency model suggests.

Write a MATLAB function that computes the normalized pressure, $p/\rho c v_0$, at a given on-axis distance at many frequencies and multiplies this pressure at each frequency by the MATLAB function spectrum1 written for homework exercise A.1. The function should evaluate this product at 1024 positive frequencies ranging from 0 to 100 MHz and then use the Fourier transform IFourierT defined in Appendix A to obtain the time-domain pulse generated by the transducer at the given location. Finally, the function should compute the peak-to-peak magnitude of this pulse and return that value. The inputs to the MATLAB function should be the distance (in mm), the transducer radius (in mm), the wave speed of the fluid (in m/sec), the center frequency, fc (in MHz), and the bandwidth, bw (in MHz).

Use this function to evaluate the peak-to peak response of a transducer radiating into water for 200 points ranging from 10 to 400 mm and plot this peak-to-peak response versus distance. Take the radius of the transducer to be 6.35 mm (0.25 in.), the center frequency fc = 5 MHz and the bandwidth bw = 2 MHz.

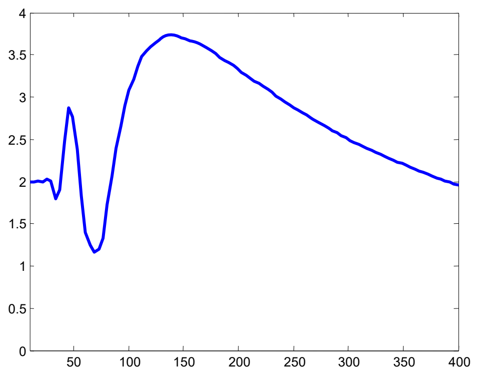
8.5 On-axis peak-to-peak response of a transducer



```
function pp =pp_onaxis( z, a, c, fc, bw)

f=s_space(0, 100, 1024);    % frequencies 0 -100 MHz
dt =.01;                     % sampling time interval
ka = 2000*pi*f*a./c;        % ka value
zn = z/a;                    % normalized on-axis distance

p = 1 -exp(i*ka.*(sqrt(zn.^2 +1 )-zn)); % on-axis normalized pressure

yf =spectrum1(f, fc, bw).*p;    % multiply by a Gaussian window

yf(1) = yf(1)/2;                % this is needed when only using positive frequencies
y =2*real(IFourierT(yf,dt));     % compute time domain pulse
pp=max(y) -min(y);               %find peak-to-peak value
```

When modeling the wave field of a transducer, we often use a plot of the on-axis pressure at a given frequency to show the values in both the near and far field. However, these do not reflect how the amplitude of the time domain signal of the transducer changes, which is composed of many frequencies. Here we will calculate the peak-to-peak amplitude of the time domain signal shown above and plot that quantity versus on-axis distance to see how it compares with the single frequency results. We write a Matlab function pp_onaxis which calculates The peak-to peak value at a given on-axis distance, z.

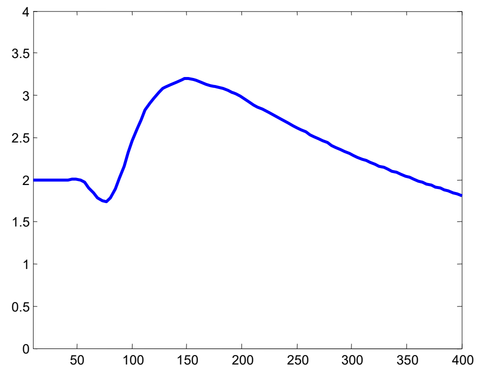
```
%pp_script
clear
z= linspace(10, 400, 100);
for n = 1:100
    pp(n)=pp_onaxis(z(n), 6.35, 1480, 5, 2);
end
plot(z, pp)
axis([10, 400, 0, 4])
```



We then write a script, pp_script, which evaluates the pp_onaxis function at multiple values of z and plots the peak-to-peak response for a given transducer radius, wave speed of the material (which is water here), center frequency, and bandwidth as defined in the spectrum1 function. Here the center frequency is 5 MHz and the bandwidth is 2 MHz. We do see a monotonic decay of the peak-to-peak response in the far field due to beam spreading, as found in the single frequency case, but the strong oscillations in the near field seen in the single frequency case have been greatly reduced. This makes sense since the locations of the on-axis peaks and nulls in the near field change with frequency so that when we superimpose many frequencies to generate a pulse, the peaks and nulls tend to get "washed out".

Consider a wider bandwidth response (bw = 4 MHz)

```
%pp_script
clear
z= linspace(10, 400, 100);
for n = 1:100
   pp(n)=pp_onaxis(z(n), 6.35, 1480, 5, 4);
end
plot(z, pp)
axis([10, 400, 0, 4])
```



If we keep all the parameters in the script the same but simply change the bandwidth to 4 MHz, save the script, and rerun it, we see even less oscillation in the near field when we have such a wide bandwidth.

F.2 Wen and Breazeale also defined 15 Gaussian beam coefficients (A_n, B_n) that improve on the modeling of a circular planar piston transducer in comparison to their original 10 coefficients [F.5]. Use the MATLAB function gauss_c15 that returns those 15 coefficients and write a MATLAB script that obtains the normalized pressure field, $p/\rho_1 c_{p1} v_0$, for a 6.35 mm radius piston transducer radiating through spherically curved water-steel interface ($R_0 = 76$ mm) at a frequency of 5 MHz (see Fig. F.11) and plots the magnitude of the on-axis normalized pressure versus the distance z_2 in the steel from $z_2 = 0$ to $z_2 = 50$ mm. Modify the script and consider the same case but where $R_0 = -76$ mm.

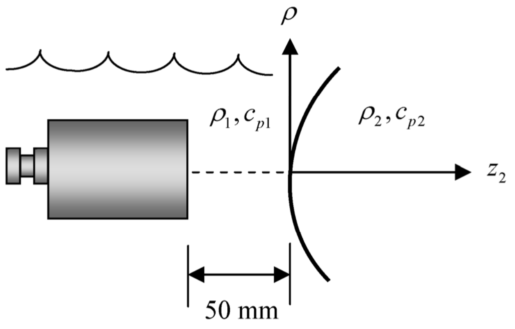
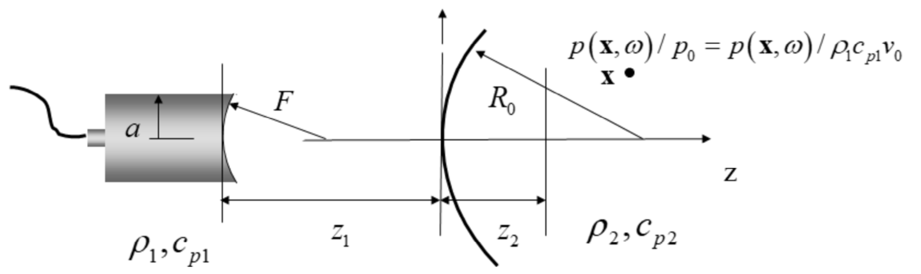


Fig. F.11. A circular planar piston transducer radiating a sound beam through a spherically curved fluid-solid interface.

F.2

MATLAB multi- Gaussian Beam Model

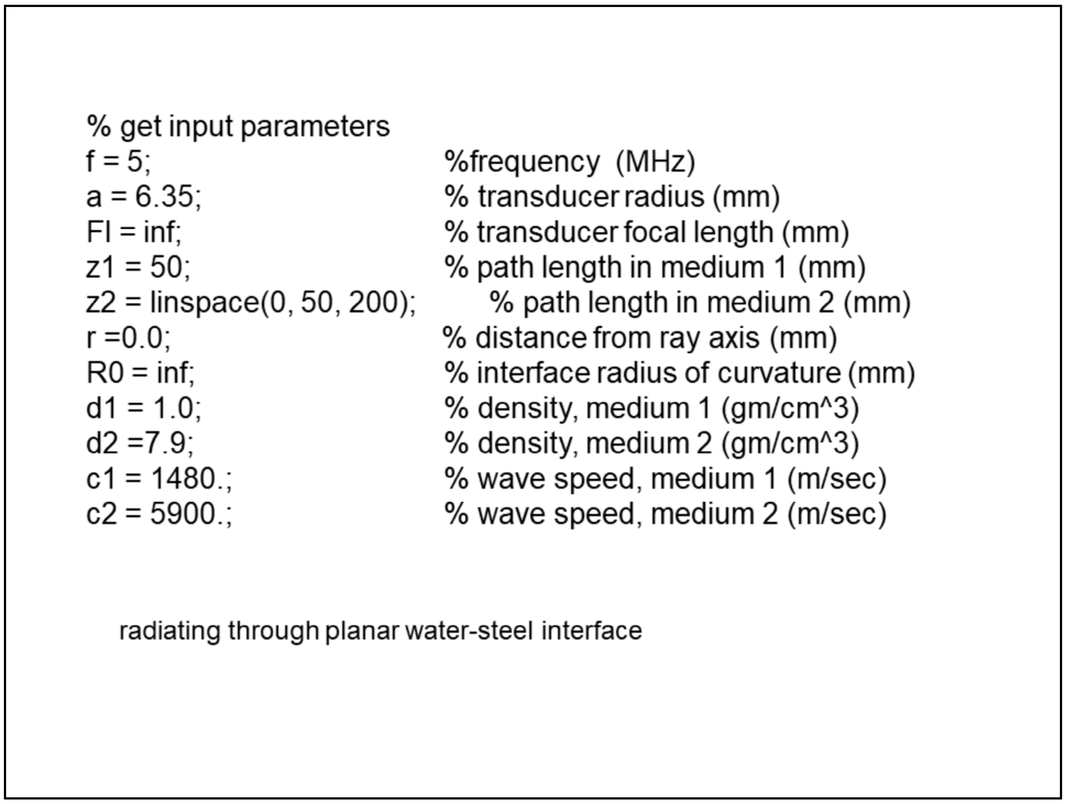


$p(\mathbf{x},\omega)/p_0 = p(\mathbf{x},\omega)/\rho_1 c_{p1} v_0$
$\mathbf{x}$
F
a
R_0
z
z_1
z_2
ρ_1, c_{p1}
ρ_2, c_{p2}

Note: the time delay factor $\exp(ik_{p1}z_1 + ik_{p2}z_2)$ is not included in this model

```
% prob_F2
% This script calculates the normalized
% pressure, p/p0, due to a planar piston transducer radiating at normal
% incidence through a spherically curved interface.
% This is a multi-Gaussian beam model that uses the 15 coefficients
%  of Wen and Brezeale to calculate the wave field.

clear
```

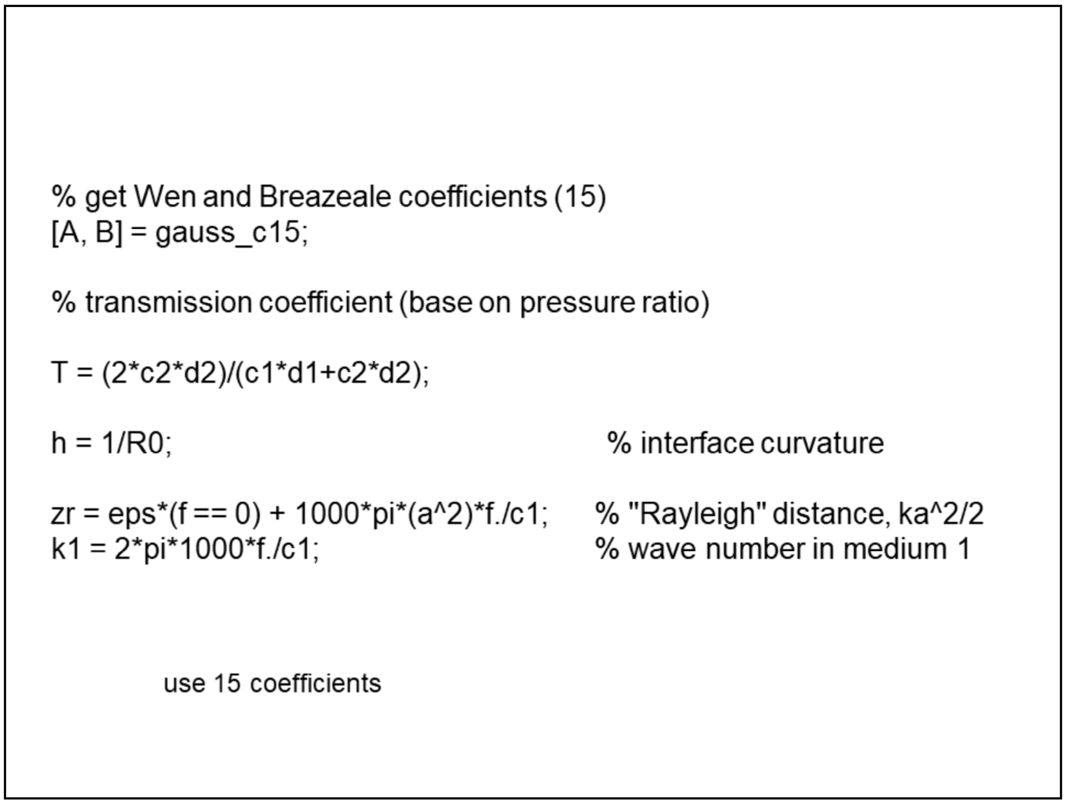
Here is a Matlab script named prob_F2 that uses a multi-Gaussian beam model to calculate the pressure for a transducer radiating through a water-steel spherically curved interface, a simple case but one that we can use to show the effects of the interface curvature. The script begins by clearing the workspace so we can define all the parameters needed in this script.



```
% get input parameters
f = 5;                         %frequency  (MHz)
a = 6.35;                      % transducer radius (mm)
Fl = inf;                      % transducer focal length (mm)
z1 = 50;                       % path length in medium 1 (mm)
z2 = linspace(0, 50, 200);       % path length in medium 2 (mm)
r =0.0;                        % distance from ray axis (mm)
R0 = inf;                      % interface radius of curvature (mm)
d1 = 1.0;                      % density, medium 1 (gm/cm^3)
d2 =7.9;                       % density, medium 2 (gm/cm^3)
c1 = 1480.;                    % wave speed, medium 1 (m/sec)
c2 = 5900.;                    % wave speed, medium 2 (m/sec)
```

radiating through planar water-steel interface

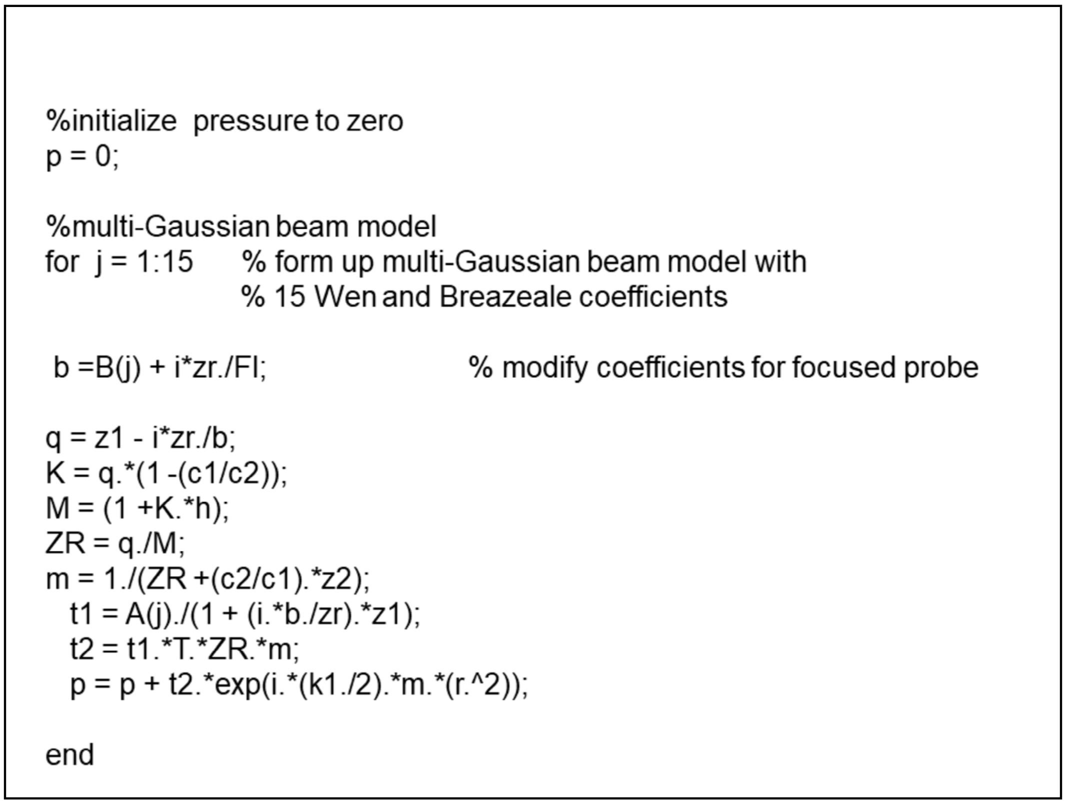
We first input all the parameters needed for this problem. We start with a script that considers the interface to be planar. Note that we be evaluating the response on the axis of the transducer at multiple distances in the steel contained in z2.



```
% get Wen and Breazeale coefficients (15)
[A, B] = gauss_c15;

% transmission coefficient (base on pressure ratio)

T = (2*c2*d2)/(c1*d1+c2*d2);

h = 1/R0;                                       % interface curvature

zr = eps*(f == 0) + 1000*pi*(a^2)*f./c1;      % "Rayleigh" distance, ka^2/2
k1 = 2*pi*1000*f./c1;                         % wave number in medium 1
```

use 15 coefficients

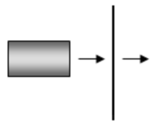
Here we use the 15 Wen and Breazeale coefficients and define the plane wave transmission coefficient, the interface curvature, Rayleigh distance, and wave number in the water.



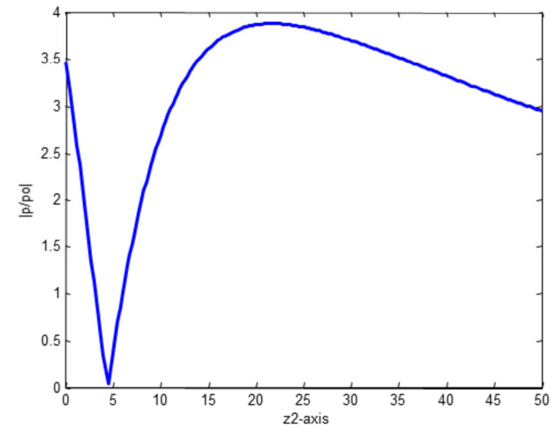
```
%initialize  pressure to zero
p = 0;

%multi-Gaussian beam model
for  j = 1:15      % form up multi-Gaussian beam model with
                   % 15 Wen and Breazeale coefficients

 b =B(j) + i*zr./Fl;                    % modify coefficients for focused probe

q = z1 - i*zr./b;
K = q.*(1 -(c1/c2));
M = (1 +K.*h);
ZR = q./M;
m = 1./(ZR +(c2/c1).*z2);
  t1 = A(j)./(1 + (i.*b./zr).*z1);
  t2 = t1.*T.*ZR.*m;
  p = p + t2.*exp(i.*(k1./2).*m.*(r.^2));

end
```

We initialize the pressure to be zero and then calculate the Gaussian beam response for all the z2 values at each of the 15 coefficient values and add them up. Note that we have a planar transducer here so that the transducer focal length, Fl = inf, is not needed for the planar transducer cases we will consider here. This multi-Gaussian beam model can be written very compactly. Here we have separately combined a number of terms (such as q, K, M, ZR, zr) to write the final Gaussian beam expression, which is in terms of only the t2, k1, and m expressions.

```
%plot magnitude of on-axis pressure
plot(z2, abs(p))
xlabel('z2-axis')
ylabel('|p|/|po|')
```

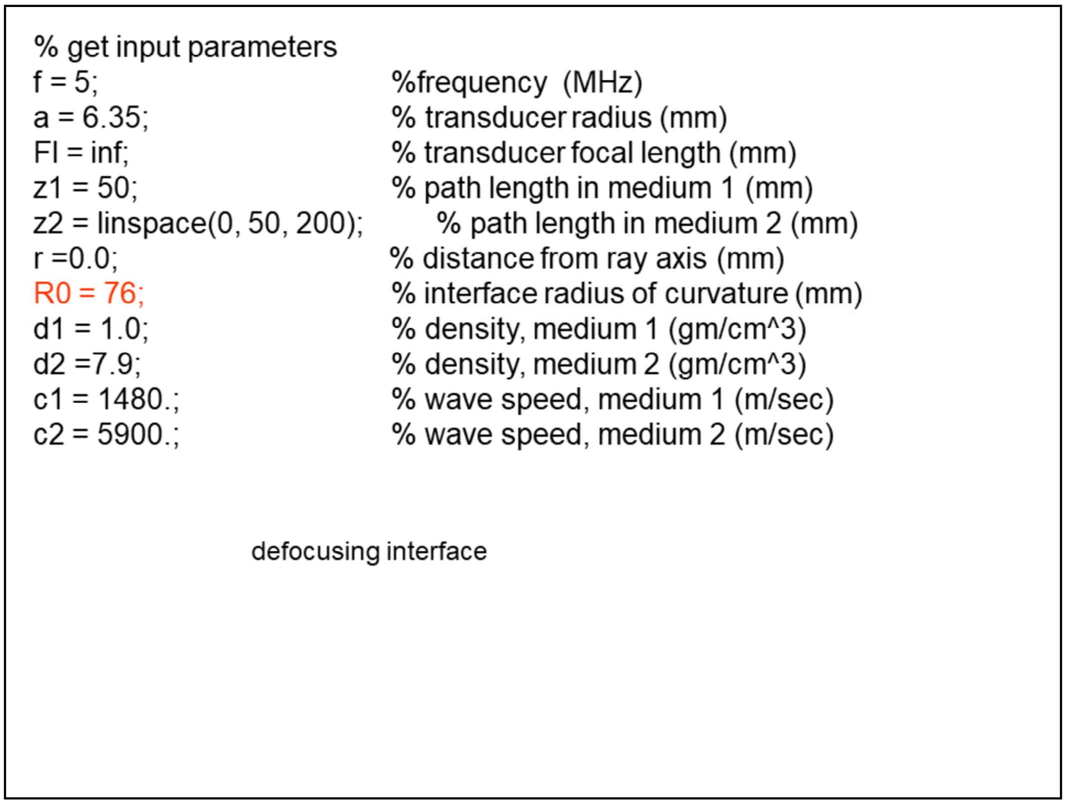


```
>> prob_F2
```



planar interface

Finally, the script plots the magnitude of the on-axis pressure in the steel. Running the script then produces the plot shown

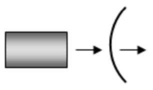


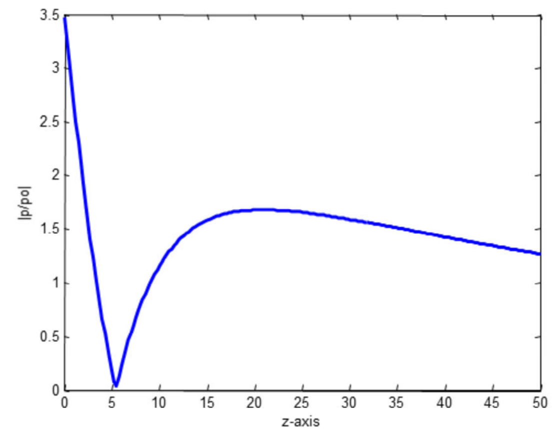
```
% get input parameters
f = 5;                      %frequency  (MHz)
a = 6.35;                   % transducer radius (mm)
Fl = inf;                   % transducer focal length (mm)
z1 = 50;                    % path length in medium 1 (mm)
z2 = linspace(0, 50, 200);      % path length in medium 2 (mm)
r =0.0;                     % distance from ray axis (mm)
R0 = 76;                    % interface radius of curvature (mm)
d1 = 1.0;                   % density, medium 1 (gm/cm^3)
d2 =7.9;                    % density, medium 2 (gm/cm^3)
c1 = 1480.;                 % wave speed, medium 1 (m/sec)
c2 = 5900.;                 % wave speed, medium 2 (m/sec)
```

defocusing interface

Now we change the radius of the interface to be 76 mm and save the script.

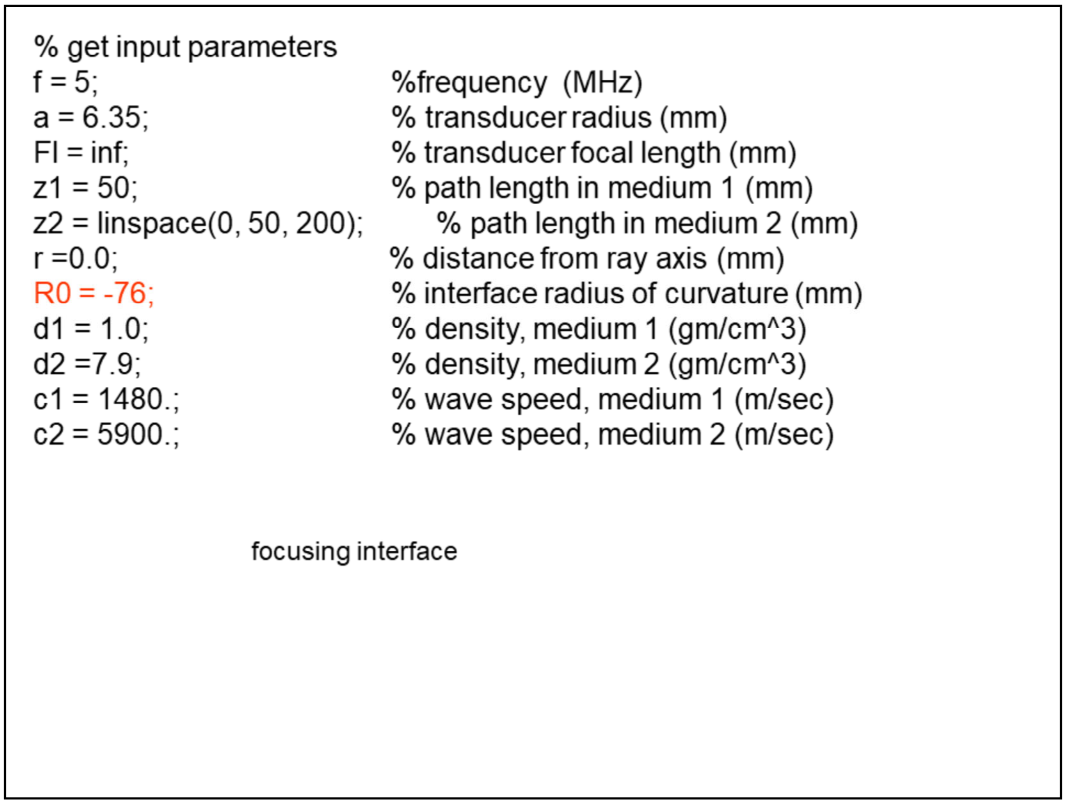
```
>> prob_F2
```





defocusing interface

When we rerun the script we see that the amplitude in the steel has been reduced. Because of this behavior we say that this is a **defocusing interface**.

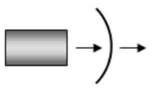


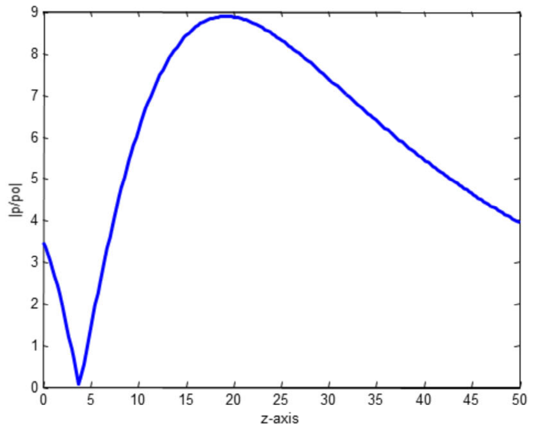
```
% get input parameters
f = 5;                    %frequency  (MHz)
a = 6.35;                 % transducer radius (mm)
Fl = inf;                 % transducer focal length (mm)
z1 = 50;                  % path length in medium 1 (mm)
z2 = linspace(0, 50, 200);    % path length in medium 2 (mm)
r =0.0;                   % distance from ray axis (mm)
R0 = -76;                 % interface radius of curvature (mm)
d1 = 1.0;                 % density, medium 1 (gm/cm^3)
d2 =7.9;                  % density, medium 2 (gm/cm^3)
c1 = 1480.;               % wave speed, medium 1 (m/sec)
c2 = 5900.;               % wave speed, medium 2 (m/sec)
```

focusing interface

Now, lets make the radius of the interface be minus 76 mm and save the script.

```
>> prob_F2
```

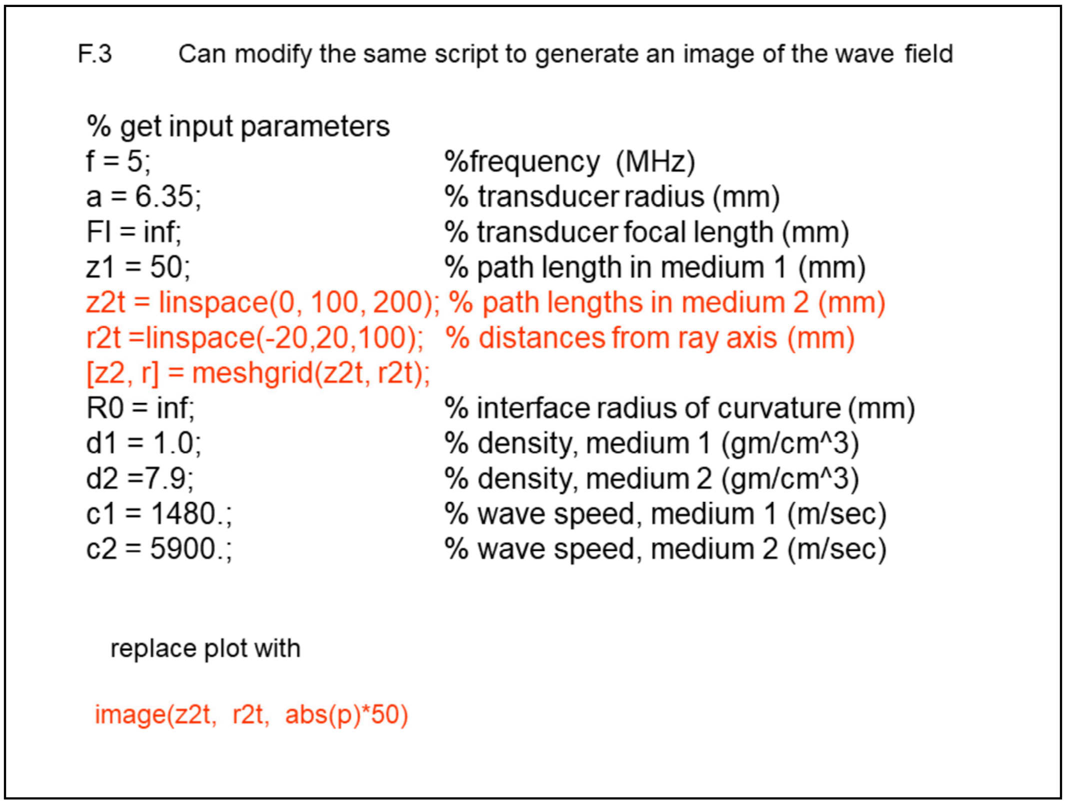




focusing interface

In this case we see the amplitude response has been increased from the planar interface case so we say that this is a **focusing interface**.

F.3 Rewrite the scripts of problem F.2 so that they display a 2-D image of the magnitude of the normalized pressure (i.e. normalized pressure versus (ρ, z_2)) in the steel for both the defocusing and focusing interfaces considered there.



F.3 Can modify the same script to generate an image of the wave field

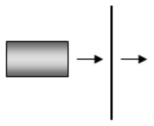
```
% get input parameters
f = 5;                          %frequency  (MHz)
a = 6.35;                       % transducer radius (mm)
Fl = inf;                       % transducer focal length (mm)
z1 = 50;                        % path length in medium 1 (mm)
z2t = linspace(0, 100, 200); % path lengths in medium 2 (mm)
r2t =linspace(-20,20,100);   % distances from ray axis (mm)
[z2, r] = meshgrid(z2t, r2t);
R0 = inf;                       % interface radius of curvature (mm)
d1 = 1.0;                       % density, medium 1 (gm/cm^3)
d2 =7.9;                        % density, medium 2 (gm/cm^3)
c1 = 1480.;                     % wave speed, medium 1 (m/sec)
c2 = 5900.;                     % wave speed, medium 2 (m/sec)
```

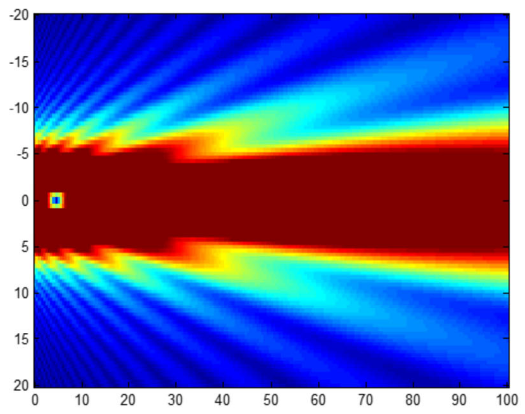
replace plot with

```
image(z2t,  r2t,  abs(p)*50)
```

We can reuse the script of problem F.2 if we simply change the inputs so we are generating the multiple (r, z) values in matrices which can be used to generate an image, and also replace the plot output by an image output (the pressure has been multiplied by 50 to get a good set of colors in the images. Alternately, you could use the Matlab function imagesc which does the scaling automatically so the factor of 50 is not needed).

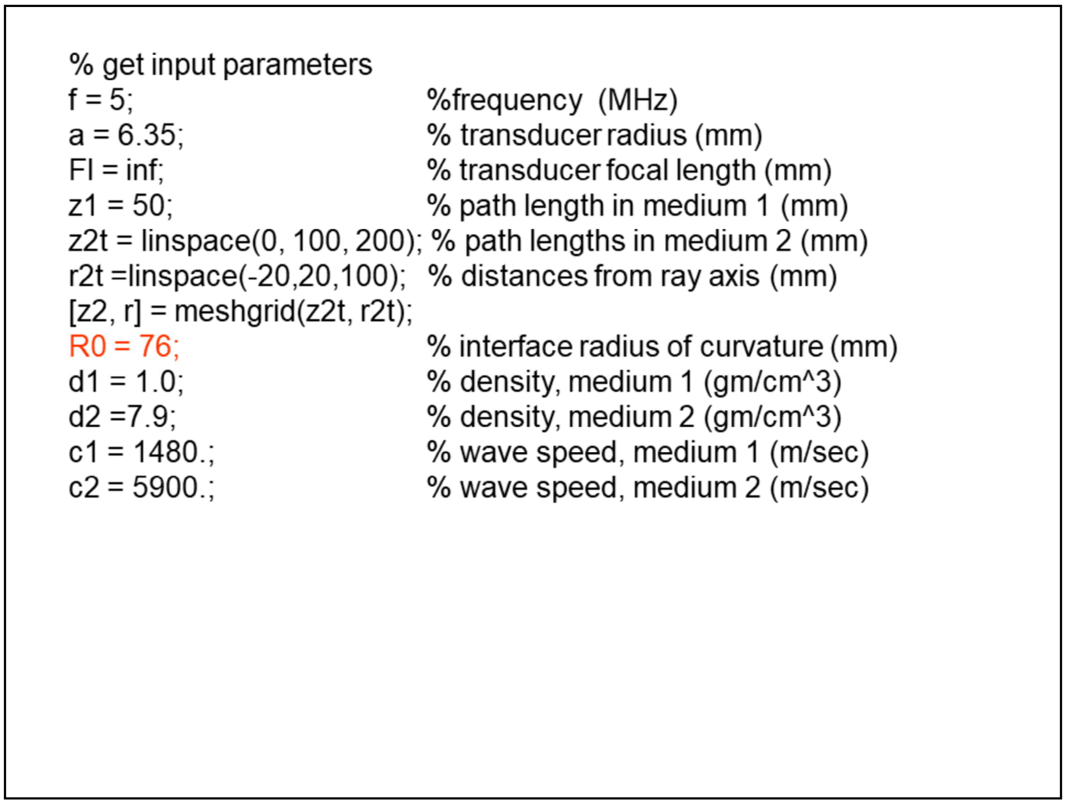
```
>> prob_F2
```





planar interface

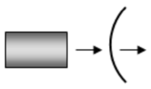
Here is the planar interface case. We see that there is one small near-field value in the steel along the central axis and a well collimated beam.

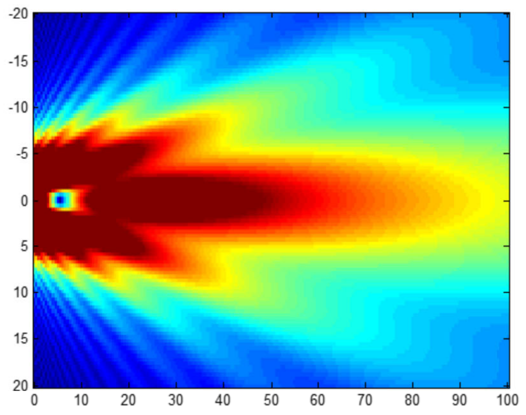


```
% get input parameters
f = 5;                          %frequency  (MHz)
a = 6.35;                       % transducer radius (mm)
Fl = inf;                       % transducer focal length (mm)
z1 = 50;                        % path length in medium 1 (mm)
z2t = linspace(0, 100, 200); % path lengths in medium 2 (mm)
r2t =linspace(-20,20,100);  % distances from ray axis (mm)
[z2, r] = meshgrid(z2t, r2t);
R0 = 76;                        % interface radius of curvature (mm)
d1 = 1.0;                       % density, medium 1 (gm/cm^3)
d2 =7.9;                        % density, medium 2 (gm/cm^3)
c1 = 1480.;                     % wave speed, medium 1 (m/sec)
c2 = 5900.;                     % wave speed, medium 2 (m/sec)
```

Now we consider the 76 mm defocusing interface.

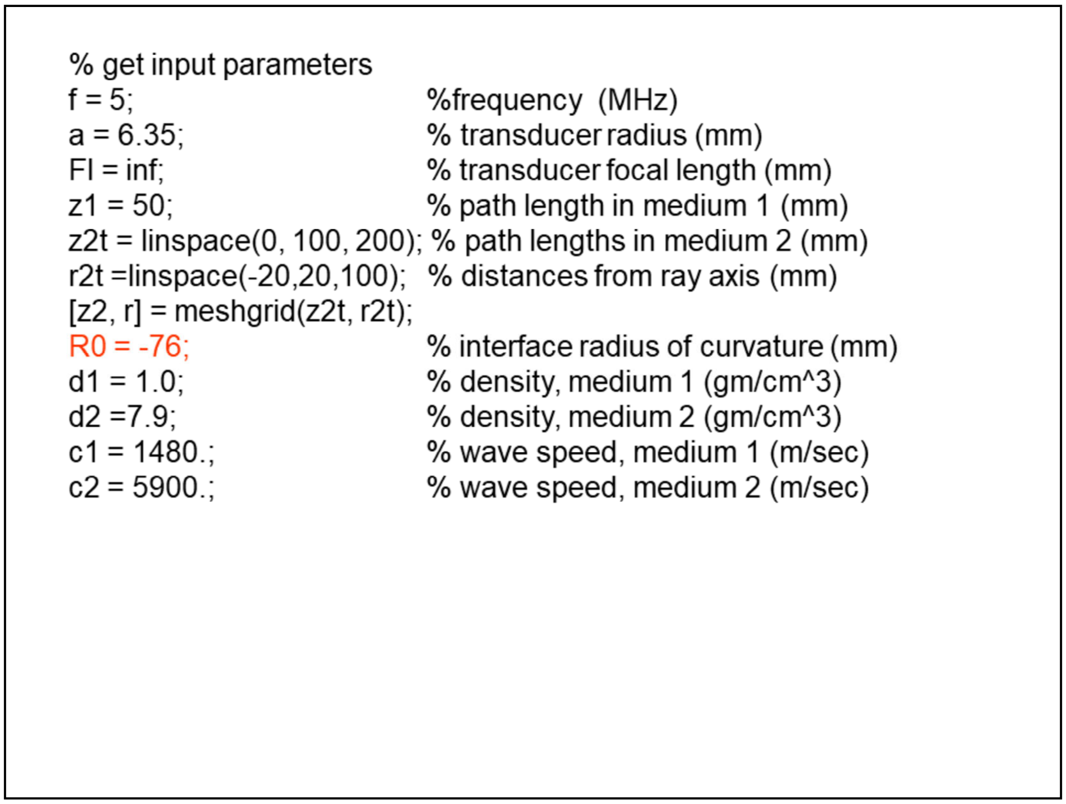
```
>> prob_F2
```





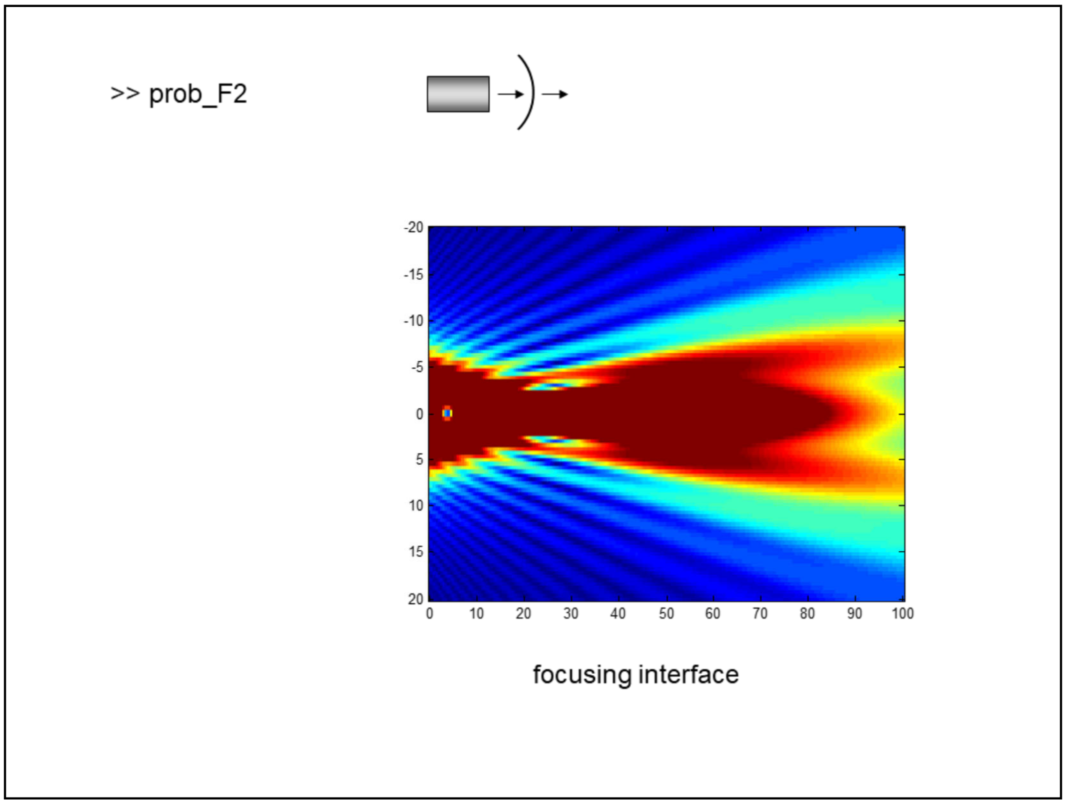
defocusing interface

When we rerun the script we see the wave field in the steel has spread considerably and decreased in amplitude.

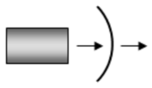


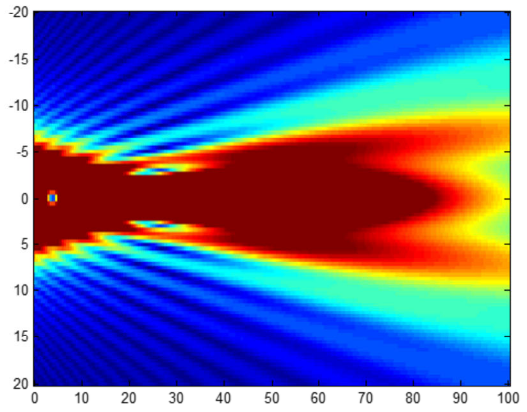
```
% get input parameters
f = 5;                          %frequency  (MHz)
a = 6.35;                       % transducer radius (mm)
Fl = inf;                       % transducer focal length (mm)
z1 = 50;                        % path length in medium 1 (mm)
z2t = linspace(0, 100, 200); % path lengths in medium 2 (mm)
r2t =linspace(-20,20,100);  % distances from ray axis (mm)
[z2, r] = meshgrid(z2t, r2t);
R0 = -76;                       % interface radius of curvature (mm)
d1 = 1.0;                       % density, medium 1 (gm/cm^3)
d2 =7.9;                        % density, medium 2 (gm/cm^3)
c1 = 1480.;                     % wave speed, medium 1 (m/sec)
c2 = 5900.;                     % wave speed, medium 2 (m/sec)
```

Now we consider the minus 76 mm focusing interface.



```
>> prob_F2
```





focusing interface

When we rerun the script we see that the beam has been concentrated in the steel, as we would expect with a focusing type of behavior.