Journal of Applied Engineering Mathematics

Volume 9, December 2022

Transmission and reflection across material boundaries

Teagan KJ Nakamoto

Mechanical Engineering Department
Brigham Young University
Provo, Utah 84602
tekajuna@byu.edu

Abstract

Stress waves in solid materials may be modeled using the wave equation. When waves meet material boundaries, there is reflection and transmission that must be accounted for to ensure safe, optimal material performance.

Nomenclature

u_r	=	reflected wave function
u_t	=	transmitted wave function
u_i	=	incident wave function
c_n	=	wave speed in the nth material
L, M	=	location of material/domain boundaries
λ	=	wavelength
A, B, C	=	wave amplitudes
x_n	=	starting position of the nth wave function
P	=	pulse length

Introduction

The wave equation can be used to mode physical phenomena from the transverse motion of oceanic surface waves to the longitudinal pulsations of sound, pressure, and stress waves in fluids and solids. The two-dimensional wave equation for waves traveling in a single medium is, in a number of cases, solvable using integral transform methods [1, 2, 3, 4]

When a traveling wave encounters a change of medium, part of the incident wave is reflected back, and part of the wave is transmitted into the new medium. Interference between reflected waves can cause unexpected wave intensities in a thin "wall" of material; it is therefore desirable to have develop a method that allows for the analysis of this case. The introduction of variable material properties introduces a step change in the variable c in the wave equation, which represents the wave speed in the material (or speed of sound).

In this paper will be developed a method for analyzing wave propagation across material boundaries for the one-dimensional wave equation.

Method

This problem consists of three distinct regions (See Fig. 1). In the first region, a pulse is beginning to propagate from the left at a speed c_1. At the right is a material which may have a different speed constant c_2, resulting in a reflected wave back into Region 1 and a transmitted wave into Region 2, which initially contains no waves. The transmitted wave from Region 1 travels through Region 2 with a change in velocity and amplitude, then into Region 3, resulting in additional reflected and transmitted waves.

For the purposes of this paper, a wave pulse will be de-

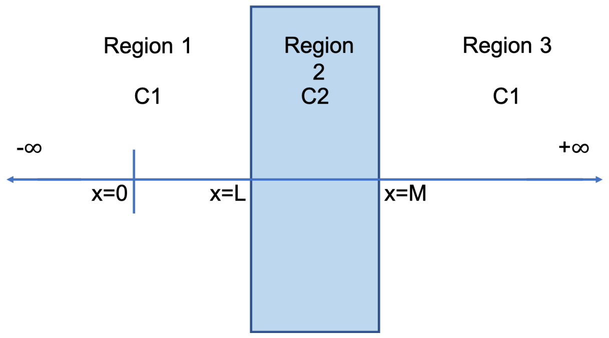

Figure 1: The simplest generalizable case for propagating a wave through material boundaries involves a single slab (Region 2) of material with a speed coefficient that differs from its surroundings (Region 1 and Region 3, which are semi-infinite).

fined with a finite duration and a simple offset-sine-wave form such that the wave intensity increases smoothly from zero up to twice the amplitude, then back down to zero.

0.1 Wave pulse equation

A wave pulse may be modeled by a modification of the d'Alembert wave solution as follows:

$$\Big(H(x_n - c_n t) - H(x_n + \lambda_n - c_n t)\Big) u_n(\lambda_n, c_n, x_n, P, x, t) \tag{1}$$

By applying a moving filter function, we are able to isolate a part of the wave form, bounded between x_n and $x_n + P$, where length of the pulse is denoted P, and with x_n as the location of the left-hand endpoint of the pulse function.

The function u_n can actually be any function, but for the implementation in this paper it will take the form:

$$u_n = A_n + A_n \cos\left(\frac{2\pi}{\lambda_n}(x - \frac{\lambda_n}{2} - x_n - c_n t)\right) \tag{2}$$

where A_n is the amplitude of the pulse, λ_n is the wavelength, c_n is the wave velocity, and x_n is the location of the left-hand-side endpoint of the wave pulse.

0.2 Material boundary conditions

Let an incident wave u_i traveling from left to right towards a material boundary at location $x = L$ has amplitude A, wavelength λ, and speed c_1. At the $x = L$ boundary, the material properties change immediately to new wave speed c_2.

For continuity of the wave form, the following conditions must apply:

$$u_i(L,t) + u_r(L,t) = u_t(L,t) \tag{3}$$

$$\frac{\partial}{\partial x}u_i(L,t) + \frac{\partial}{\partial x}u_r(L,t) = \frac{\partial}{\partial x}u_t(L,t) \tag{4}$$

From these conditions can be derived the necessary change in amplitude and effective wavelength that occurs between the incident, reflected, and transmitted waves. For a pulse of amplitude A and wavelength λ_A, moving from material with speed c_1 to material with speed c_2 we obtain the following relations:

$$B = A\frac{c_2 - c_1}{c_2 + c_1} \tag{5}$$

$$C = A\frac{2c_2}{c_2 + c_1} \tag{6}$$

B is the amplitude associated with the reflected wave, and C is the amplitude associated with the transmitted wave.

$$\lambda_C = \lambda_A \frac{c2}{c_1} \tag{7}$$

Note that the reflected pulse has the same speed (opposite-sign velocity) of the incident pulse, and the same wavelength as well.

The concept of "boundary conditions" can now be applied to solve for the starting positions of the reflected and transmitted waves. The calculated position is such that the incident, reflected, and transmitted pulses all meet the material boundary at the same time.

0.3 Implementation of wave equations

A pulse function may be initialized with the following parameters:

- Amplitude A

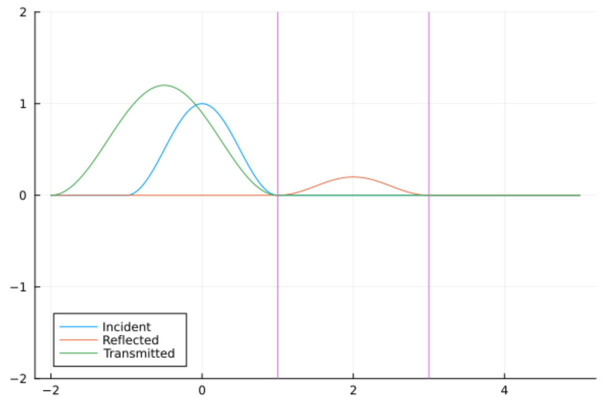

Figure 2: The incident pulse meets the wall at $x = 2$ simultaneously with the reflected pulse and transmitted pulse. All waves are summed only in the regions in which they apply, so at this time frame the reflected and transmitted waves are "virtual," and not included in the wave intensity calculation where presently located.

- Wavelength λ
- Starting Position $x1$, $x2$
- Velocity c

We therefore can define a `Pulse` object that makes these parameters conveniently available and extensible for many pulse functions. The following algorithm generates pulse functions such that each Region is assigned pulse functions representing every incident, reflected, and transmitted pulse. Mathematically, pulse amplitudes will decay below a threshold after a certain number of transmission/reflection events. This is easily handled with a `while` loop. See Algorithm 1.

Algorithm 1 Generation of Pulse Functions

```
1: Initialize N Regions $R_1$: $R_n$ with speed of sound $c_1$ to $c_n$
2: Initialize pulse contained in Region 1 with speed $c_1$ moving toward Region 2 $\rightarrow$Incident
3: Calculate reflected pulse back into Region 1 $\rightarrow$L1
4: Calculate transmitted pulse into Region 2 $\rightarrow$R2
5: while $A > threshold$ do
6:     L2 = Reflect(R2)
7:     R3 = Transmit(22)
8:     R2 = Reflect(L2)
9:     L1 = Transmit(L2)
10: end while
11: for pulse in pulseList do GenerateFunction(pulse)
12: end for
        return List of Pulse Functions per region
```

Algorithm 2 Reflect

```
1: Inputs: Pulse object with $A$, $\lambda$, $c$, $x_0$,
2: Boundary Location $L$ and $c_L$ on other side of boundary
3: $A_R$ = Pulse.Amplitude*(c2-c1)/(c2+c1)
4: $x_R$ = L + L -Pulse.position - Pulse.wavelength
5: ReflectedPulse = PulseObject($A_R$,Pulse.wavelength,$x_R$, -Pulse.speed)
        return ReflectedPulse
```

Algorithm 3 Transmit

```
1: Inputs: Pulse object with $A$, $\lambda$, $c$, $x_0$,
2: Boundary Location $L$ and $c_L$ on other side of boundary
3: $A_T$ = Pulse.Amplitude*(2*c2)/(c2+c1)
4: $\lambda_T$ = c2/c1*Pulse.wavelength
5: $x_T$ = -c2/c1 * (-Pulse.position+L+A.wavelength) + L + $\lambda_T$
6: TransmittedPulse = PulseObject($A_T$, $\lambda_T$, $x_T$, Pulse.speed *$c_L$)
        return TransmittedPulse
```

Results

Four initial cases are completely simulated and results are available at `https://github.com/tekajuna/pulse-thru-wall`.

For each case, $c_1 = 2.0$, $c_2 = 8.0$, the wall spans from $x = 2.0$ to $x = 3.0$, and one pulse is located with the left endpoint at the origin with an amplitude of 0.5. A second pulse is also initialized as follows:

1. Second pulse is located at $x = -2$ with amplitude 0.5 (Pulses separated by one wavelength)
2. Second pulse is located at $x = -1$ with amplitude 0.5 (Two pulses appear as a single sine wave with two full periods)
3. Second pulse is located at $x = -0.5$ with amplitude 0.5 (Sine wave with a flattened top due to partial interference)
4. Second pulse is located at $x = -1$ with amplitude -0.5 (Second pulse is reflected across x-axis, resulting in the appearance of a full-period sine wave with amplitude of 1 and no vertical offset)

Two parameter variations studies were performed. In the first, the starting position of one of two identical pulses is varied. The stationary pulse is located at the origin, as in the Cases 1 through 4. The varied-position pulse is located at positions between $x = -1$ and $x = 1$. At $x = 0$, the two pulses constructively interfere, resulting in a single pulse with double amplitude. For each starting position of the second pulse, the maximum wave intensity value in Region 2 is plotted.

The second parameter variation study is similar to the first, but the second pulse has negative amplitude, resulting in perfectly destructive interference when $x = 0$. Peak Region 2 wave intensity is found when $x = \pm 0.5$

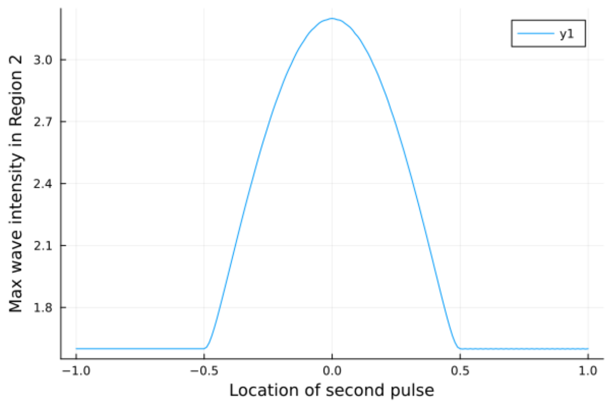

Figure 3: Starting location of one pulse was varied with respect to the starting location of the other, effectively varying the relative time-of-arrival of two pulses to the first material boundary. Both pulses have positive amplitude, resulting in varying degrees of constructive interference. The peak wave intensity within Region 2 is maximized when the pulses totally interfere constructively, resulting in a wave amplitude over three times that of the individual pulses

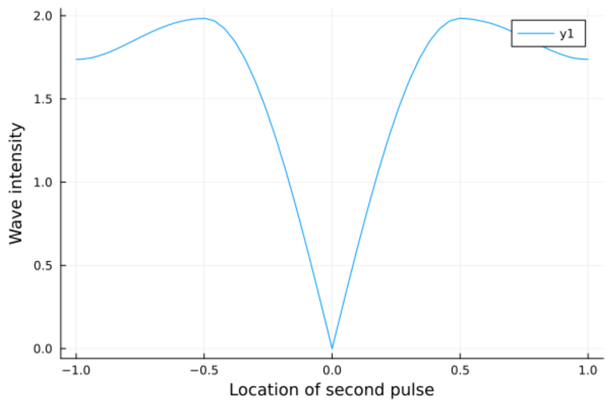

Figure 4: The second pulse in this case has a negative amplitude. Starting location of this pulse was varied with respect to the starting location of the other. Varying degrees of destructive interference is observed. The peak wave intensity within Region 2 is maximized when there is a certain amount of offset between the negative and positive pulses.

Conclusions

A method for analyzing wave pulse behavior across material boundaries has been developed. The method developed involves a single slab of material sandwiched between semi-infinite regions of the same material property c_1, but adding additional finite regions is easily accomplished. Present development includes one particular type of wave pulse, but the concepts are easily extensible to other pulse functions, such as semi-infinite, sawtooth, Gaussian, and Lorentzian pulse/wave variants.

Further development should aid in the interpretation of the pulse intensity.

References

[1] Dustin Badger and Paul McMullin. “2-D Waves: Combinations and Characteristic Frequencies”. In: *Journal of Applied Engineering Mathematics* 7 (2020).

[2] Tevin Dickerson and Spencer Truman. “Doppler Effect with Gravitational waves in a 2D space”. In: *Journal of Applied Engineering Mathematics* 8 (2021).

[3] Eric Lee and Joseph Furner. “Reverberations of Plucked Guitar String”. In: *Journal of Applied Engineering Mathematics* 8 (2021).

[4] Natasha Wilson and Benjamin Varela. “Interactions of Boat Wakes on a Lake Surface”. In: *Journal of Applied Engineering Mathematics* 8 (2021).

Appendix

Julia code for pulse analysis and plotting

```
using Plots

struct Pulse #Characteristics of a pulse
    amplitude
    wavelength
    position # Position of the LHS of the pulse
    speed
end

function filter(A::Pulse, x1,x2,x,c,t,f)
    if x < x1+c*t # pulse is defined only on interval x1 to x2
        return 0
    elseif x > x2 + c*t
        return 0
    else
        return f(x,t) # Pulse can have any functional definition between the endpoints
    end
end

function cosWaveFunction(A,lambda,c,x1)
    function wave(x,t)
        return A + A*cos(2*pi/lambda*(x - 0.5*lambda -x1 - c*t) ) #centers the pulse within [x1, x2]
    end
    return wave
end

function generatePulse(A::Pulse)
    # TODO: Specify location and direction of the pulse
    function pulse(x,t)
        x1 = A.position # Pulse is located left of the origin
        x2 = A.position + A.wavelength       # RHS of the pulse is at the origin
        f = cosWaveFunction(A.amplitude,A.wavelength,A.speed,A.position)
        return filter(A,x1,x2,x,A.speed,t,f)

    end
    return pulse
end

function Reflect(A::Pulse, L,cL)
    c1 = abs(A.speed)     # Characteristic Speed of the Incidient wave
    c2 = abs(cL) # Characteristic speed on the OTHER side of "L"
    ampR = A.amplitude*(c2-c1)/(c2+c1) # Calculate amplitude of the reflected wave (absolute value)
    lamR = A.wavelength # Reflected wave has same wavel. as incident
    posR = L + L -A.position - A.wavelength   #Reflected wave is positioned
    Rpulse = Pulse(ampR,lamR,posR,-A.speed)
    return Rpulse
end

function Transmit(A::Pulse,L,cL)
    c1 = abs(A.speed) # Velocity of the incident wave
    c2 = abs(cL)      # Speed of target material
    ampT = A.amplitude * 2*c2/(c2+c1)
    lamT = c2 *A.wavelength/c1
    posT = -c2/c1*(-A.position+L+A.wavelength) +L +lamT
    Tpulse = Pulse(ampT,lamT,posT,sign(A.speed)*cL)
    return Tpulse
end

function calculatePulses(inc,c1,c2, L, M)
    Reg1 =[]
    Reg2 =[]
    Reg3 =[]
    # R Right-moving
    # L Left-moving
    # Number indicates region
    R1 = inc            # Incident Wave
    push!(Reg1,R1)
    L1 = Reflect(inc,L,c2)   # Reflects toward negative infinity
    push!(Reg1,L1)
    R2 = Transmit(inc,L,c2)  # Transmits toward M
    push!(Reg2,R2)
    # L2 = Reflect(R2)    # Internal Reflection
    # R3 = Transmit(R2)   # Transmits toward infinity
    # L1 = Transmit(L2)   # Transmits to negative infinity
    # R2 = Reflect(L2)    # internal Reflection
    # L2 = Reflect(R2)
    # R3 = Transmit(L2)
    # generates:
    # L1, R2
    # L2, R3
    # L1. R2
    # L2, R3

    while abs(R2.amplitude) > 1e-3 #Calculate reflections until amplitude goes below threshold
        L2 = Reflect(R2,M,c1)    # Internal Reflection
        push!(Reg2,L2)
        R3 = Transmit(R2,M,c1)   # Transmits toward infinity
        push!(Reg3,R3)
        L1 = Transmit(L2,L,c1)   # Transmits to negative infinity
        push!(Reg1,L1)
        R2 = Reflect(L2,L,c1)
        push!(Reg2,R2)
    end
    # println(length(Reg1))
    # println(length(Reg2))
    # println(length(Reg3))
    #Transform pulse objects to functions
    for i = 1:length(Reg1)
        Reg1[i] = generatePulse(Reg1[i])
    end
    for i = 1:length(Reg2)
        Reg2[i] = generatePulse(Reg2[i])
    end
    for i = 1:length(Reg3)
        Reg3[i] = generatePulse(Reg3[i])
    end


    return Reg1, Reg2, Reg3
end

function testPulseGeneration()
    c1 = 8.0
    c2 = 2.0
    L  = 1.0
    M  = 3.0
    N  = 5.0
    initialPulse = Pulse(1.0,1,0,c1)
    R1,R2,R3 = calculatePulses(initialPulse,c1,c2, L, M)
    println("Neat")
end

function analyze()


end

function testRegions()
    c1 = 2.0
    c2 = 8.0
    L  = 2.0
    M  = 3.0
    N  = 5.0
    initialPulse = Pulse(0.5,1,0,c1)
    R1,R2,R3 = calculatePulses(initialPulse,c1,c2, L, M)
    additionalPulse=Pulse(-0.5,1,-0.5,c1)
    R1b,R2b,R3b = calculatePulses(additionalPulse,c1,c2, L, M)


    # A = generatePulse(initialPulse)
    # push!(R1,A)
    # B = generatePulse(Reflect(initialPulse,L,c2))
    # C = generatePulse(Transmit(initialPulse,L,c2))
    # push!(R1,B)
    # push!(R2,C)
    tvec = 0:0.01:5.0
    x1 = 0.0:0.01:L
    x2 = L:0.01:M
    x3 = M:0.01:N
    for i = 1:length(tvec)
        t = tvec[i]
        # plot(x,A.(x,t),label="I")
        # plot!(x,B.(x ,t),label="R1")
        # plot!(x,C.(x,t),label="T1")
        # u1 = R1[1].(x1,t) + R1[2].(x1,t)
        u1 = sum([R1[i].(x1,t) for i in 1:length(R1)])+sum([R1b[i].(x1,t) for i in 1:length(R1b)])
        u2 = sum([R2[i].(x2,t) for i in 1:length(R2)])+sum([R2b[i].(x2,t) for i in 1:length(R2b)])
        u3 = sum([R3[i].(x3,t) for i in 1:length(R3)])+sum([R3b[i].(x3,t) for i in 1:length(R3b)])
        # plot(x1,A.(x1,t)+B.(x1,t))
        plot(x1,u1)
        plot!(x2,u2)
        plot!(x3,u3)
        # plot!(x2,C.(x2,t))
        # plot!(x3,zeros(length(x3)))
        vline!([L,M])
        ylims!((-2,2))
        if i <10
            savefig("nice4/Thing00"*string(i)*".png")
        elseif i < 100
            savefig("nice4/Thing0"*string(i)*".png")
        else
            savefig("nice4/Thing"*string(i)*".png")
        end

    end


end

function varyPulseSpacing()
```

```
    c1 = 2.0
    c2 = 8.0
    L  = 2.0
    M  = 3.0
    N  = 5.0
    initialPulse = Pulse(-0.5,1,0,c1)
    R1,R2,R3 = calculatePulses(initialPulse,c1,c2, L, M)


    xAddl = -1:0.1:1 # Vary Second Pulse location; otherwise pulses are identical
    maxR2 = []

    for k = 1:length(xAddl)
        additionalPulse=Pulse(0.5,1,xAddl[k],c1)
        R1b,R2b,R3b = calculatePulses(additionalPulse,c1,c2, L, M)
        tvec = 0:0.01:5.0
        x1 = 0.0:0.01:L
        x2 = L:0.01:M
        x3 = M:0.01:N
        maxtemp=0 # Reset max pulse intensity for the current initial pulse spacing
        for i = 1:length(tvec)
            t = tvec[i]
            # plot(x,A.(x,t),label="I")
            # plot!(x,B.(x ,t),label="R1")
            # plot!(x,C.(x,t),label="T1")
            # u1 = R1[1].(x1,t) + R1[2].(x1,t)
            u1 = sum([R1[i].(x1,t) for i in 1:length(R1)])+sum([R1b[i].(x1,t) for i in 1:length(R1b)])
            u2 = sum([R2[i].(x2,t) for i in 1:length(R2)])+sum([R2b[i].(x2,t) for i in 1:length(R2b)])
            u3 = sum([R3[i].(x3,t) for i in 1:length(R3)])+sum([R3b[i].(x3,t) for i in 1:length(R3b)])
            if maximum(abs.(u2)) > maxtemp
                maxtemp = maximum(abs.(u2))
            end

        end
        push!(maxR2,maxtemp)
    end
    plot(xAddl,maxR2)
    xlabel("Pulse Spacing")
    ylabel("Max Wave Intensity")
    savefig("figs/PulseSpacingReverse.png")

end
```

```
function testdalambert()
    amp = 0.5
    wavel=2
    c1 = 2.0
    c2 = 3.0
    L = 1.0 # Position of the first wall, transitioning from c1 to c2
    M = 3.0
    N = 5.0
    initialPulse = Pulse(amp,wavel,-wavel,c1)
    A = generatePulse(initialPulse) # Generate Incident pulse # Amplitude, wavelength, LHS location
    Rpulse = Reflect(initialPulse,L,c2)
    B = generatePulse(Rpulse)
    Tpulse = Transmit(initialPulse,L,c2)
    C = generatePulse(Tpulse)
    D = generatePulse(Reflect(Tpulse,M,c1))
    E = generatePulse(Transmit(Tpulse,M,c1))
    # t = 0
    tvec = 0.0:0.1:8.0

    # R1 0 to L
    # R2 L to M
    # R3 M to N

    x = -2.0:0.01:N

    for i in range(1,length(tvec);step=1)
        t = tvec[i]
        plot(x,A.(x,t),label="Incident")
        plot!(x,B.(x ,t),label="Reflected")
        plot!(x,C.(x,t),label="Transmitted")
        # plot!(x,D.(x,t),label="R2")
        # plot!(x,E.(x,t),label="T2")
        vline!([L,M],label=nothing)
        ylims!((-2,2))
        plot!(legend=:bottomleft)

        # legend
        savefig("TP/Thing"*string(i)*".png")

    end


end
if abspath(PROGRAM_FILE) ==@__FILE__
    # testdalambert()
    # testRegions()
    # testPulseGeneration()
    varyPulseSpacing()
end
```