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, N	=	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 one-dimensional wave equation can provide some insight on the way that stress waves propagate in a material. The speed of sound in a material is proportional to $\sqrt{\frac{E}{\rho}}$, making wave speed largely a function of the material properties. A typical machine may involve multiple materials butted together. Stress pulses traveling across material boundaries may be partially reflected, partially transmitted.

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

This problem consists of three distinct regions. In the first region, a stress pulse is beginning to propagate from the left. At the right is a material which may have a different speed constant, resulting in a reflected wave and a transmitted wave into Region 2, which is initially of zero amplitude. The transmitted wave from Region 1 travels through Region 2 with a change in velocity and amplitude, then into Region 3, which may again result in reflection and transmission.

This problem could be modeled using Heaviside step functions at the material boundaries, making $c = \sqrt{\frac{E}{\rho}} =$

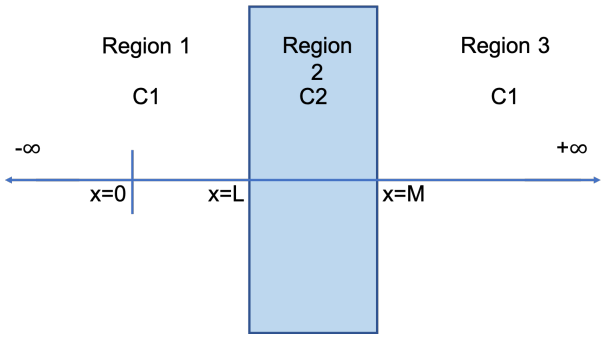

Figure 1: Regions caption

$f(x)$. The addition of a variable coefficient poses challenges to solving the full wave equation. An attempt is made to model using three instances of the wave equation. If we are interested in how a stress wave is attenuated over a material "gap," we can focus on how the stress wave hits the material boundary, which can kick off a wave into the next material.

Thus, Region 1 may be modeled with a Dirichlet boundary condition on the left-hand side, where the pulse is beginning to move from. On the right hand side is a Robin boundary condition. The solution to the wave equation with these boundary conditions may be obtained using integral transform methods.One result is an explicit equation for $u(L_1, t)$, the equation for wave amplitude at the RHS of Region 1.

We may use this equation as the input to Region 2; in other words, we set the LHS of Region 2 as a Dirichlet boundary condition with amplitude specified by $u_1(L_1, t)$

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} \tag{1}$$

Where $u = f(x, t)$, i.e., the solution to the wave equation. In this paper, $u(x, t)$ and u will be used interchangeably.

For Region 1, the wave equation is solved subject to the following initial and boundary conditions:

$$u(x, 0) = \sin(A\pi x) - sin(A\pi x)H(A - x) \tag{2}$$

Where H is the Heaviside function, implying that the initial condition is a pulse of width A, with the value of $u(x, 0)$ equal to zero everywhere else.

$$\frac{\partial u}{\partial t} = 0 \tag{3}$$

$$u(0, t) = 0 \tag{4}$$

$$k_1\frac{\partial u(L_1, t}{\partial x} + h_1 u(x, t) = 0 \tag{5}$$

The solution corresponds with that found in VIII.3.5.1 by Soloviev. The general solution is:

$$\sum_{n=1}^{\infty} \frac{\sin\lambda_n x}{\frac{L}{2} - \frac{\sin 2\lambda_n L_1}{4\lambda_n}} \left(\int_0^{L_1} u_0(x)\sin(\lambda_n x)dx \cos(\lambda_n ct) + \frac{\int_0^{L_1} u_1(x)\sin(\lambda_n x)dx}{\lambda_n c}\sin(\lambda_n ct) \right) \tag{6}$$

The Sturm-Liouville problem associated with the multi-material wave equation is difficult to solve,

0.1 Wave pulse equation

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

$$\Big(H(x_n - c_n t) - H(x_n + \lambda_n - c_n t)\Big) u_n\Big(\frac{2\pi}{\lambda}(x - (x_n + P) - c_n t)\Big) \tag{7}$$

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.

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 L boundary, the material properties change immediately to new wave speed c_2.

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

$$u_i(L, t) + u_r(L, t) = u_t(L, t) \tag{8}$$

$$\frac{\partial}{\partial x}u_i(L,t) + \frac{\partial}{\partial x}u_r(L,t) = \frac{\partial}{\partial x}u_t(L,t) \tag{9}$$

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:

$$B = A\frac{c_2 - c_1}{c_2 + c_1} \tag{10}$$

$$C = A\frac{2c_2}{c_2 + c_1} \tag{11}$$

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

$$\lambda_C = \lambda_A\frac{c2}{c_1} \tag{12}$$

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.

A pulse function may be initialized with the following parameters:

- Amplitude A
- 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 in implemented in a `while` loop.

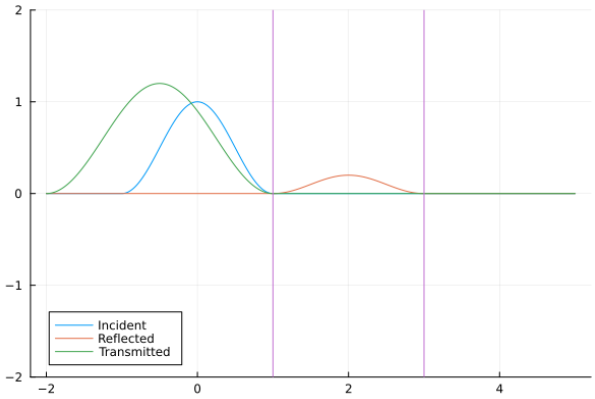

Figure 2: The incident pulse meets the wall at $x = 2$ simultaneously with the reflected pulse and transmitted pulse. All waves are summed only in the regions in which they apply, so at this time frame the reflected and transmitted waves are "virtual," and not included in the wave intensity calculation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

Algorithm 1 Generation of Pulse Functions

Initialize N Regions R_1: R_n with speed of sound c_1 to c_n
Initialize pulse contained in Region 1 with speed c_1 moving toward Region 2 →Incident
Calculate reflected pulse back into Region 1 →L1
Calculate transmitted pulse into Region 2 →R2
while $A > threshold$ **do**
    L2 = `Reflect`(R2)
    R3 = `Transmit`(22)
    R2 = `Reflect`(L2)
    L1 = `Transmit`(L2)
end while
for pulse in pulseList **do**
    `GenerateFunction`(pulse)
end for
    return List of Pulse Functions per region

Results

The resulting program is able to efficiently

Conclusions

A method for analyzing wave pulse behavior across material boundaries has been developed. The method developed is generalizable to any number of material layers, provided materials at the left and right extremes go towards negative and positive infinity, respectively. 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.

Appendix

Young's Modulus per density: 10^6 m^2/s^2

- Aluminum: 26
- Silicon: 79
- Steel: 25
- Copper: 13
- Zinc: 15
- Tungsten: 21
- Sapphire: 101
- Beryllium: 155
- Diamond: 347

```
----------------------Wave Function-------------------------------
using Plots

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

function filter(A::Pulse, x1,x2,x,c,t,f)
    if x < x1+c*t #nonzero only on [x1, x2]
        return 0
    elseif x > x2 + c*t
        return 0
    else
        return f(x,t)
# Any functional definition
# between the endpoints
    end
end

function waveFunction(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 and x2
```



```
    end
    return wave
end

function generatePulse(A::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 = waveFunction(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)
    c2 = abs(cL)
    ampT = A.amplitude * 2*c2/(c2+c1)
    lamT = c2 *A.wavelength/c1
    posT = -c2/c1*(-A.position+L+A.wavelength) +L +lamT
    # (x1+ L + wavelength)/c1 = (L-x2+wavelength)/c2

    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
        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))
    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 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
    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 = 0.0:0.001:N
```



```
    for i in range(1,length(tvec);step=1)
        t = tvec[i]
        plot(x,A.(x,t),label="I")
        plot!(x,B.(x ,t),label="R1")
        plot!(x,C.(x,t),label="T1")
        plot!(x,D.(x,t),label="R2")
        plot!(x,E.(x,t),label="T2")
        vline!([L,M])
        ylims!((-2,2))
        plot!(legend=:bottomleft)

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

    end


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

