Homework 2: Generalized-Alpha method

Name: Abdelrahman Fathy Abdelhaleem Aly Abdelrahman

Matr.-Nr.: 108023251500


```
begin
    using LinearAlgebra
    using Plots
    using LaTeXStrings
end
```

Task 1:

1.1. From small amplitude vibrations assumption: $sin(\theta) \approx \theta$.

1.2. Then from the given two balance of linear momentum equations, one can formulate the $\boldsymbol{M}$ & $\boldsymbol{K}$ matrices, as follows:

$$\boldsymbol{M} = \begin{bmatrix} \frac{\rho A L^3}{3} & 0 \\ 0 & \frac{\rho A L}{3} \end{bmatrix}, \; \boldsymbol{K} = \begin{bmatrix} \frac{\rho A g L^2}{2} & 0 \\ 0 & \frac{EA}{L} \end{bmatrix}$$

with $\boldsymbol{u}$ & $\ddot{\boldsymbol{u}}$ are vectors, and there components are as follows:

$$\boldsymbol{u} = \begin{bmatrix} \theta \\ u \end{bmatrix}, \; \ddot{\boldsymbol{u}} = \begin{bmatrix} \ddot{\theta} \\ \ddot{u} \end{bmatrix}$$

1.3. For given α_1 & α_2, one can calculate the Rayleigh damping matrix $\boldsymbol{M}$ from the following equation: $\boldsymbol{C} = \alpha_1 \boldsymbol{M} + \alpha_2 \boldsymbol{K}$

1.4. Finally, the semi-discrete equation can be written as follows:

$$\boldsymbol{M} \cdot \ddot{\boldsymbol{u}} + \boldsymbol{C} \cdot \dot{\boldsymbol{u}} + \boldsymbol{K} \cdot \boldsymbol{u} = \boldsymbol{R}$$

with,

$$\boldsymbol{R} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \; \dot{\boldsymbol{u}} = \begin{bmatrix} \dot{\theta} \\ \dot{u} \end{bmatrix}$$

> Note

Semi-discrete means that the equation is discrete in space only, but continous in time.

9.8

```
begin
    # Data inputs
    L = 1.0
    ρA = 1.0
    EA = 500
    g = 9.8
end
```

```
(M = 2×2 Matrix{Float64}:, K = 2×2 Matrix{Float64}:)
     0.333333  0.0             4.9    0.0
```

```
begin
    # Global variables
    M = [(ρA*L^3)/3 0; 0  (ρA*L)/3]
    K = [(ρA*g*L^2)/2 0; 0 EA/L]
    R = [0 , 0]
    (M = M, K=K)
end
```

```
2×2 Matrix{Float64}:
 0.333333  0.0
 0.0       0.333333
```

```
begin
    # Calculate Rayleigh damping
    α_1 = 1.0
    α_2 = 0.0
    C = α_1*M + α_2 * K
end
```

Task 2:

2.1. Calculate the initial conditions $\boldsymbol{u_o}, \dot{\boldsymbol{u}}_o, \ddot{\boldsymbol{u}}_o$ as follows:

$$\dot{\boldsymbol{u}}_o = \begin{bmatrix} 0 \\ -\frac{L}{5} \end{bmatrix}, \; \dot{\boldsymbol{u}}_o = \begin{bmatrix} \sqrt{\frac{g}{6L}} \\ 0 \end{bmatrix} \rightarrow \textbf{both are given in the problem prompt}$$

whereas, $\ddot{\boldsymbol{u}}_o$ can be calculated from the semi-discrete equation of motion as follows:

$$\begin{aligned} \boldsymbol{M} \cdot \ddot{\boldsymbol{u}}_o + \boldsymbol{C} \cdot \dot{\boldsymbol{u}}_o + \boldsymbol{K} \cdot \boldsymbol{u}_o &= \boldsymbol{R} \\ \Rightarrow \ddot{\boldsymbol{u}}_o &= \boldsymbol{M}^{-1} \cdot (\boldsymbol{R} - \boldsymbol{C} \cdot \dot{\boldsymbol{u}}_o + \boldsymbol{K} \cdot \boldsymbol{u}_o) \end{aligned}$$

2.2. For given ρ_∞, calculate the Newmark parameters according to *Chung and Hulbert (1993)* method:

$$\alpha_m = \frac{2\rho_\infty - 1}{\rho_\infty + 1}, \; \alpha_f = \frac{\rho_\infty}{\rho_\infty + 1}, \; \beta = \frac{1}{4}[1 - \alpha_m + \alpha_f]^2, \; \gamma = \frac{1}{2} - \alpha_m + \alpha_f$$

2.2. Calculate $\boldsymbol{K}_{eff}$ from the following equation:

$$\boldsymbol{K}_{eff} = \boldsymbol{M}\frac{1-\alpha_m}{\beta\Delta t^2} + \boldsymbol{C}\frac{\gamma(1-\alpha_f)}{\beta\Delta t} + \boldsymbol{K}(1-\alpha_f) \rightarrow \text{(slides pg. 63)}$$

2.3. Loop over time from $t_o = 0$ upto t_{final} with time step = Δt and for each step n do:
2.3.1 calculate the effective right hand side from the following equation:

$$\begin{aligned}\boldsymbol{r}_{eff} &= -\boldsymbol{K}\cdot\alpha_f\boldsymbol{u}_n \\ &+ \boldsymbol{C}\cdot\left[\frac{\gamma(1-\alpha_f)}{\beta\Delta t}\boldsymbol{u}_n + \frac{\gamma-\gamma\alpha_f-\beta}{\beta}\dot{\boldsymbol{u}}_n + \frac{(\gamma-2\beta)(1-\alpha_f)}{2\beta}\Delta t\ddot{\boldsymbol{u}}_n\right] \\ &+ \boldsymbol{M}\cdot\left[\frac{1-\alpha_m}{\beta\Delta^2}\boldsymbol{u}_n\right] \rightarrow \text{(slides pg. 63)}\end{aligned}$$

2.3.2. solve for $\boldsymbol{u}_{n+1}$ as follows:

$$\boldsymbol{u}_{n+1} = \boldsymbol{K}_{eff}^{-1}\cdot\boldsymbol{r}_{eff}$$

2.3.3 update velocity and acceleration (i.e. $\dot{\boldsymbol{u}}_{n+1}, \ddot{\boldsymbol{u}}_{n+1}$) from the following equations:

$$\begin{aligned}\dot{\boldsymbol{u}}_{n+1} &= \frac{\gamma}{\beta\Delta t}(\boldsymbol{u}_{n+1}-\boldsymbol{u}_n) - \frac{\gamma-\beta}{\beta}\dot{\boldsymbol{u}}_n - \frac{\gamma-2\beta}{2\beta}\Delta t\ddot{\boldsymbol{u}}_n \rightarrow \text{(slides pg. 60)} \\ \ddot{\boldsymbol{u}}_{n+1} &= \frac{1}{\beta\Delta t^2}(\boldsymbol{u}_{n+1}-\boldsymbol{u}_n) - \frac{1}{\beta\Delta t}\dot{\boldsymbol{u}}_n - \frac{1-2\beta}{2\beta}\ddot{\boldsymbol{u}}_n \rightarrow \text{(slides pg. 60)}\end{aligned}$$

```
struct DynamicResponse
    u::Matrix{Float64} # displacement
    ud::Matrix{Float64} # velocity
    udd::Matrix{Float64} # acceleration
end
```

```
struct Error
    e_abs::Vector{Float64} # absolute error (Zienkiewicz and Xie)
    η::Vector{Float64} # relative error
    e_cum::Vector{Float64} # cummulative error
end
```

```
struct TimeData
    times::Vector{Float64}
    steps::Vector{Float64}
end
```

generalized_alpha (generic function with 1 method)

```
function generalized_alpha(tf,Δt, α₁, α₂, ρ_∞)
    # calculate damping
    C = α₁*M + α₂ * K

    # Initial conditions
    u_o = [0 , - L /5 ]
    ud_o = [sqrt(g/(6*L)),0]
    udd_o = inv(M) * (R - C * ud_o - K * u_o)

    # Chung and Hulbert (1993)
    α_m = (2*ρ_∞ - 1)/(ρ_∞ + 1)
    α_f = (ρ_∞)/(ρ_∞+1)
    β = 0.25 * (1 - α_m + α_f)^2
    γ = 0.5 - α_m + α_f

    K_eff = M * ((1-α_m)/(β*Δt^2)) + C * (γ * (1 - α_f))/(β*Δt) + K * (1 - α_f)

    # create time interval range to loop over later
    t = 0:Δt:tf
    n = length(t)
    steps = fill(Δt,n)

    # define our respone containers
    n = length(t)
    u = zeros(2,n)
    ud = zeros(2,n)
    udd = zeros(2,n)

    # set initial conditions in our response containers
    u[:,1] = u_o
    ud[:,1] = ud_o
    udd[:,1] = udd_o

    # define the containers for errors
    e_abs = zeros(n) # absolute error
    η = zeros(n) # relative error
    e_cum = zeros(n) # cummulative error
    for i = 1:n-1

        r_eff = -K * α_f * u[:,i] +
                C * (((γ * (1-α_f))/(β*Δt)) * u[:,i] + ((γ - γ*α_f - β)/(β)) *
                ud[:,i] + (((γ-2*β)*(1-α_f))/(2*β)) * Δt*udd[:,i]) +
                M * (((1-α_m)/(β*Δt^2))*u[:,i] + ((1-α_m)/(β*Δt))* ud[:,i] + ((1-α_m
                - 2 *β)/(2*β)) * udd[:,i])

        # solve for u.
        u[:,i+1] = K_eff \ r_eff

        # update velocity and acceleration
        ud[:,i+1] = ((γ)/(β*Δt)) * (u[:,i+1] - u[:,i]) - ((γ - β)/(β))*ud[:,i] - ((γ
        - 2*β)/(2*β)) * Δt * udd[:,i]


```

```
        udd[:,i+1] = (1/(β*Δt^2))*(u[:,i+1] - u[:,i]) - (1/(β*Δt)) * ud[:,i] - ((1-
        2*β)/(2*β)) * udd[:,i]

        # calculate errors
            e_abs[i+1] = norm(((6*β - 1)/(6))*(udd[:,i+1] - udd[:,i]) * Δt^2)
        η[i+1] = e_abs[i+1] / norm(u[i+1] - u[i])
        e_cum[i+1] = sum(e_abs)
    end

    (response = DynamicResponse(u,ud,udd), errors = Error(e_abs,η,e_cum),time =
    TimeData(t_steps))
```

```
(response = DynamicResponse(2×501 Matrix{Float64}:
                            0.0   0.012712   0.0252789   0.0376839  …  0.00219718  0.00
```

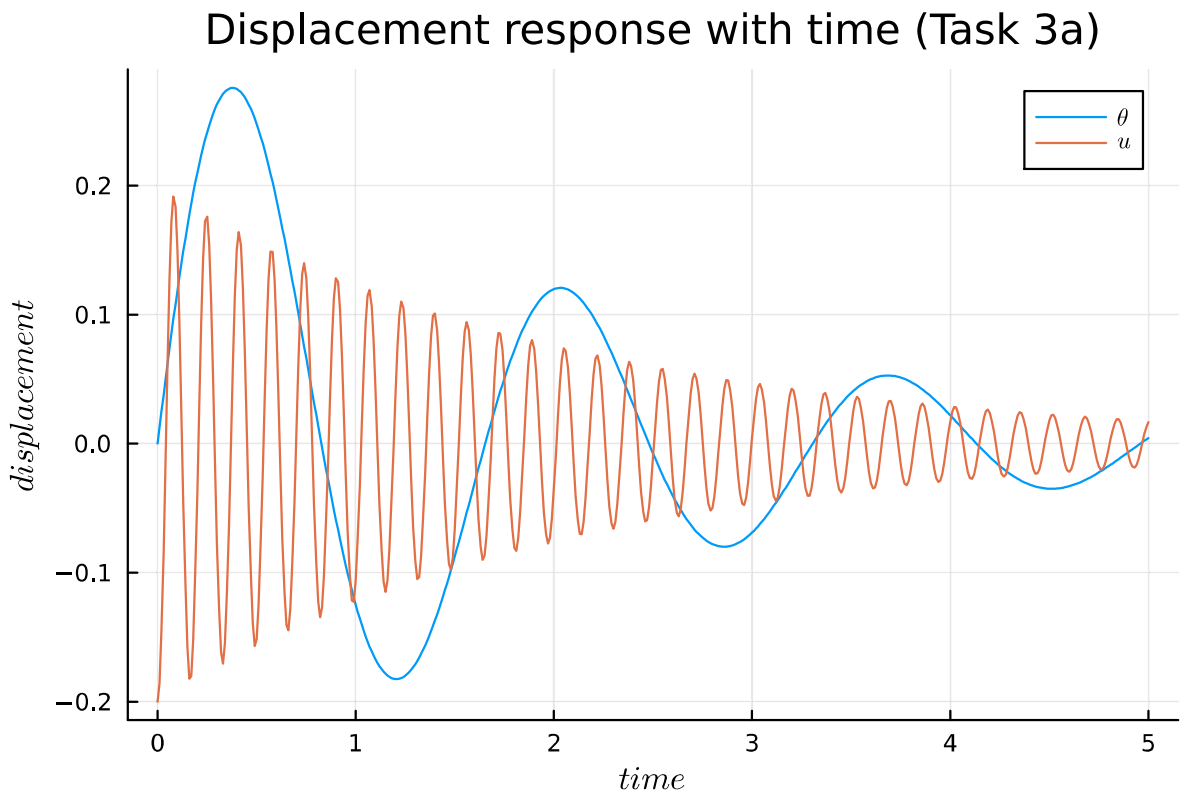
```
generalized_alpha(5.0,0.01, 1.0, 0.0, 1.0)
```

Task 3a:

Given: $t_{final} = 5\ s, \alpha_1 = 1, \alpha_2 = 0, \rho_\infty = 1.0$

```
(response = DynamicResponse(2×501 Matrix{Float64}:
                            0.0   0.012712   0.0252789   0.0376839  …  0.00219718  0.00
```

```
resp_a , err_a ,time_a= generalized_alpha(5.0,0.01, 1.0, 0.0, 1.0)
```



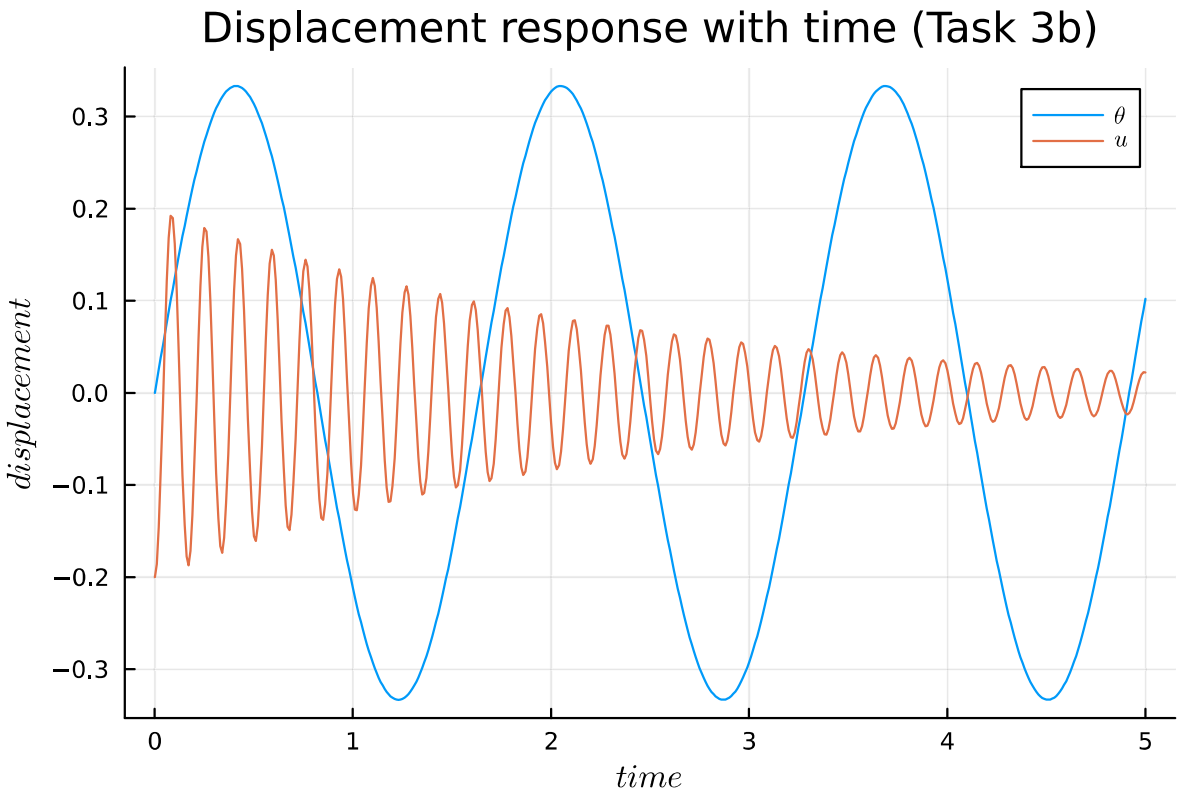

```
begin
    plot(time_a.times,resp_a.u[1,:],title="Displacement response with time (Task
    3a)",xlabel=L"time",ylabel=L"displacement",label=L"$\theta$")
    plot!(time_a.times,resp_a.u[2,:],label=L"u")

end
```

Task 3b:

Given: $t_{final} = 5\ s, \alpha_1 = 0, \alpha_2 = 0, \rho_\infty = 0.1$

```
(response = DynamicResponse(2×501 Matrix{Float64}:
                            0.0   0.012772   0.0255218   0.0382327  …  0.0774721  0.089
```

```
resp_b , err_b, time_b= generalized_alpha(5.0,0.01, 0.0, 0.0, 0.1)
```



Task 3 (Explanation):

Observtion:

In *Task 3a* **both** degree of freedoms (i.e. $\boldsymbol{u}$ & $\boldsymbol{\theta}$) are being damped with time, whereas in *Task 3b* **only** $\boldsymbol{u}$ is being damped with time and the amplitude of $\boldsymbol{\theta}$ is constant through time.

Explanation:

The reason behind such observation is that $\boldsymbol{\alpha_1}$ & $\boldsymbol{\alpha_2}$ are called **physical damping parameters** because they contribute to the formulation of the *Rayleigh damping* (i.e. $\boldsymbol{C = \alpha_1 M + \alpha_2 K}$) and as given in *Task 3a* $\boldsymbol{\alpha_1 \neq 0 \rightarrow C \neq 0}$, consequently, all degree of freedoms are being damped by $\boldsymbol{C}$ regardless their frequencies.

However, $\boldsymbol{\rho_\infty}$ is called **numerical damping parameter** due to the numerical error arised from this parameter that leads to damping. Additionally, it only damps degree of freedoms that has higher frequencies and this obvious in *task 3b* which has **no physical** damping but has **numerical** damping.

Finaly, in *task 3a* there is no numerical damping because, $\boldsymbol{\gamma = \frac{1}{2} \rightarrow \rho_\infty = 1}$ (slides pg. 59) and numerical damping only occurs if $\boldsymbol{\gamma > \frac{1}{2} \rightarrow \rho_\infty < 1}$

Task 4 (Error Calculation):

> **Note**
>
> Error calculation is already implemented in `generalized_alpha`. Accordingly, in this section, I will only show the equations that were used, some notes about regarding the implementation and the

error graphs as well.

4.1. Zienkiewicz and Xie (i.e. absolute error):

$$e^{ZX} = \frac{6\beta - 1}{6}(\ddot{u}_{n+1} - \ddot{u}_n)\Delta t^2 \rightarrow \text{(slides pg. 90)}$$

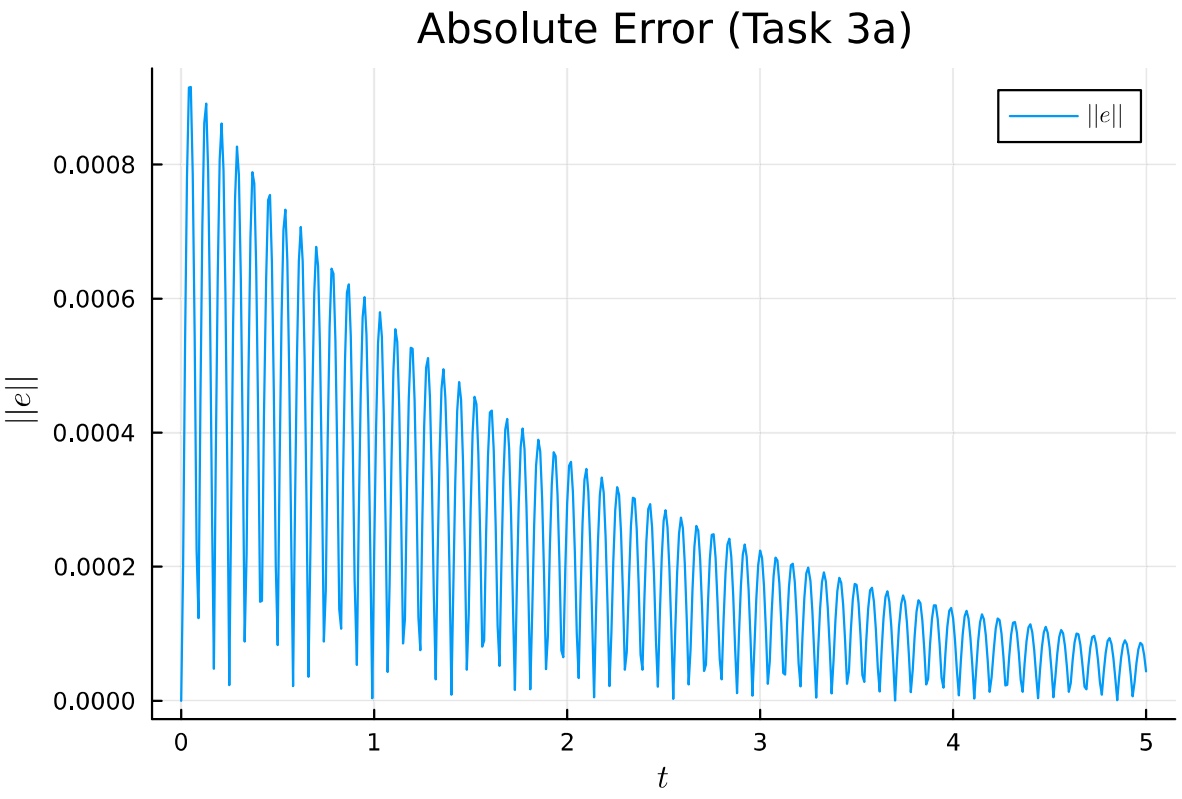
4.2. Relative error:

$$\eta = \frac{||e||}{||u_{n+1} - u_n||} \rightarrow \text{(slides pg. 91)}$$

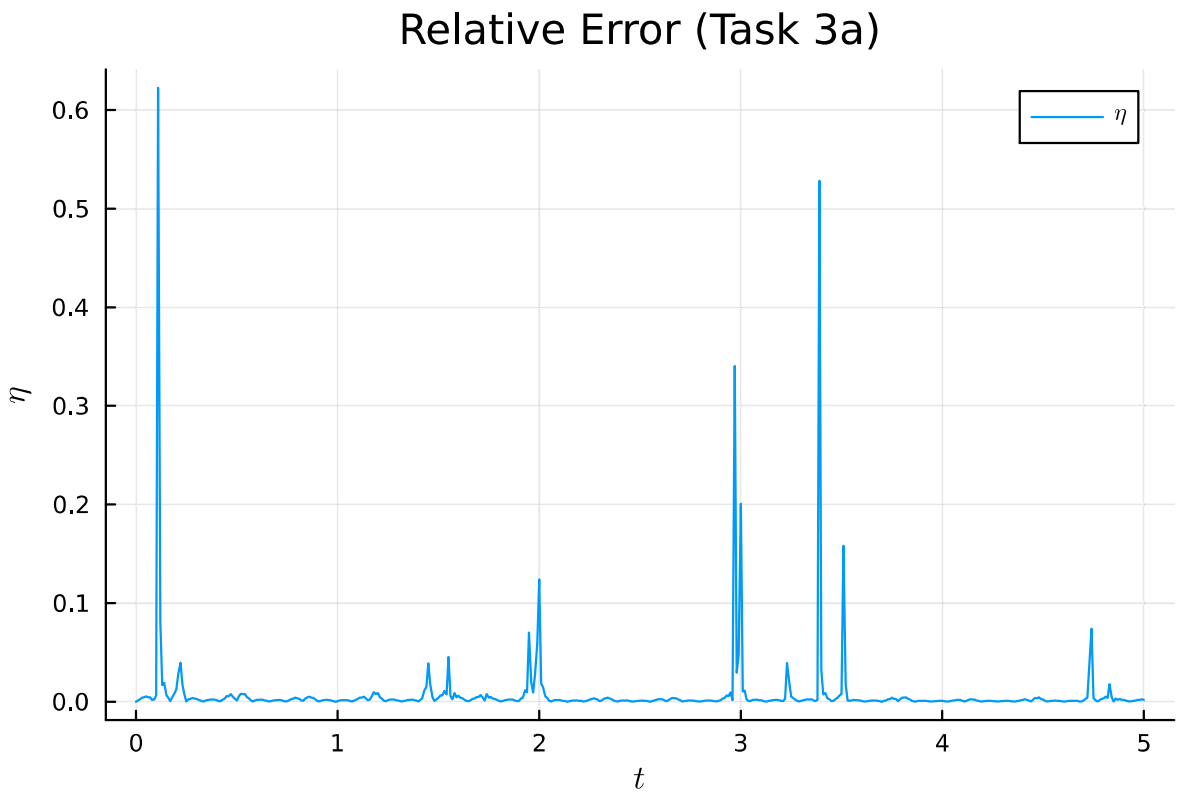
4.3. Cummulative error:

$$e_{cm} = \sum_{i=1}^{n} ||e||$$

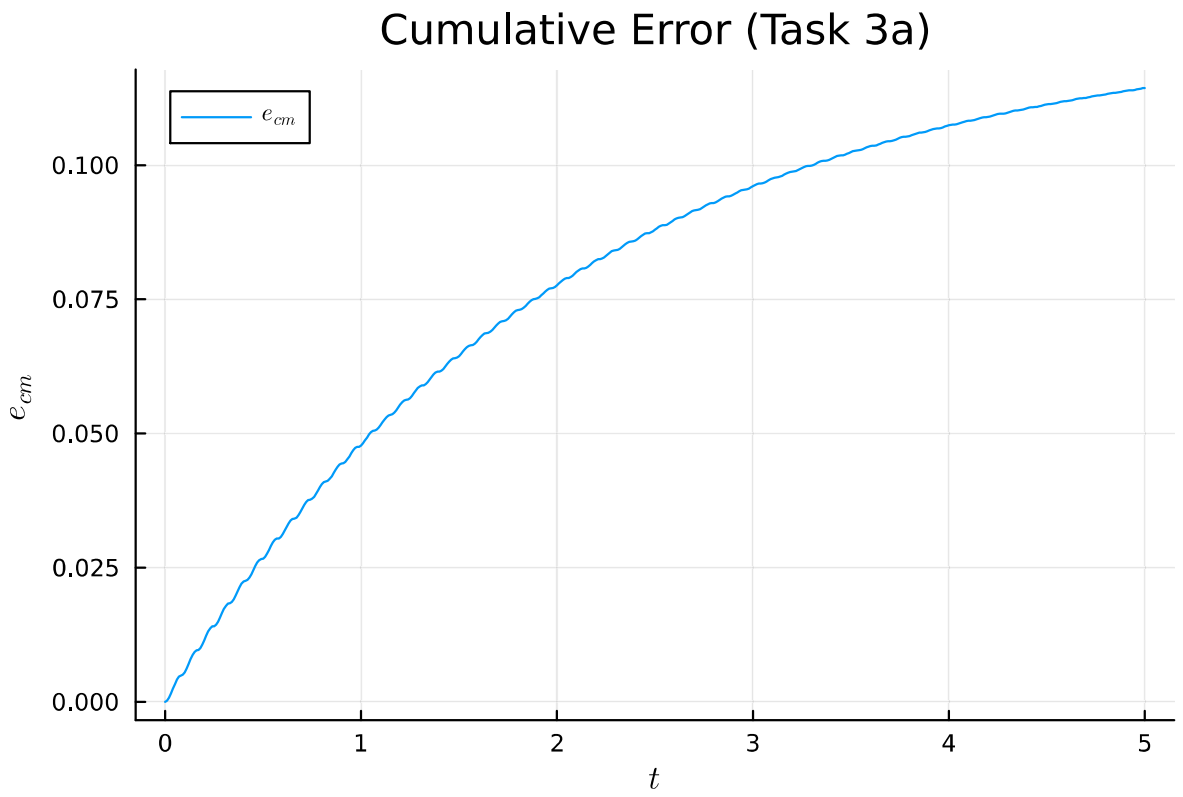
where n is the index of the current time step.



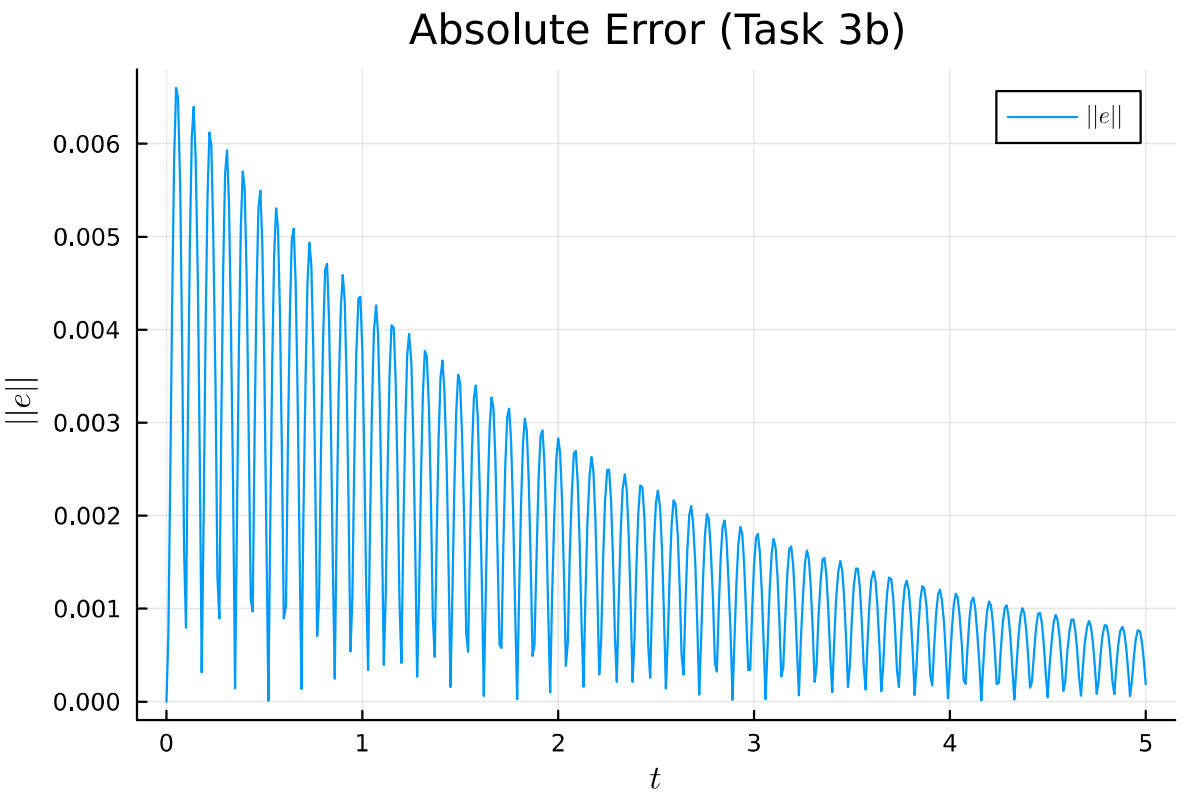

```
begin
    plot(time_a.times,err_a.e_abs,title="Absolute Error (Task
    3a)",xlabel=L"t",ylabel=L"||e||",label=L"$||e||$")
end
```



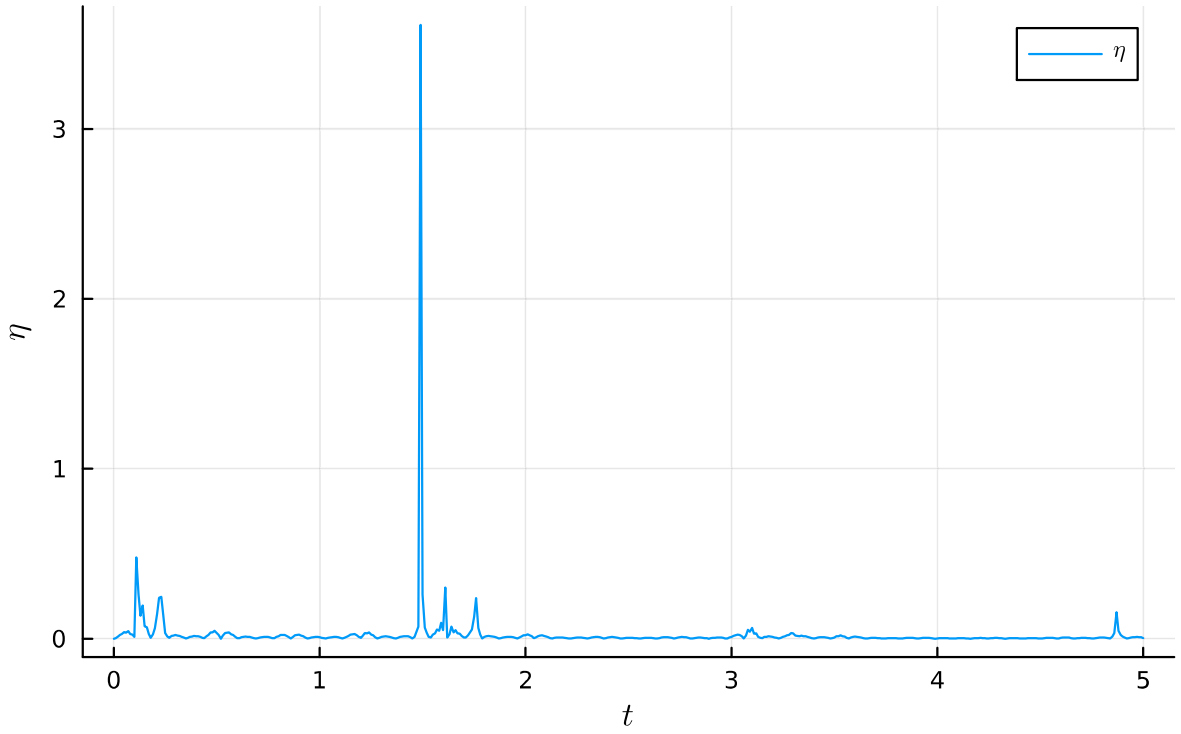

```
begin
    plot(time_a.times,err_a.η,title="Relative Error (Task
    3a)",xlabel=L"t",ylabel=L"η",label=L"$\eta$")
end
```



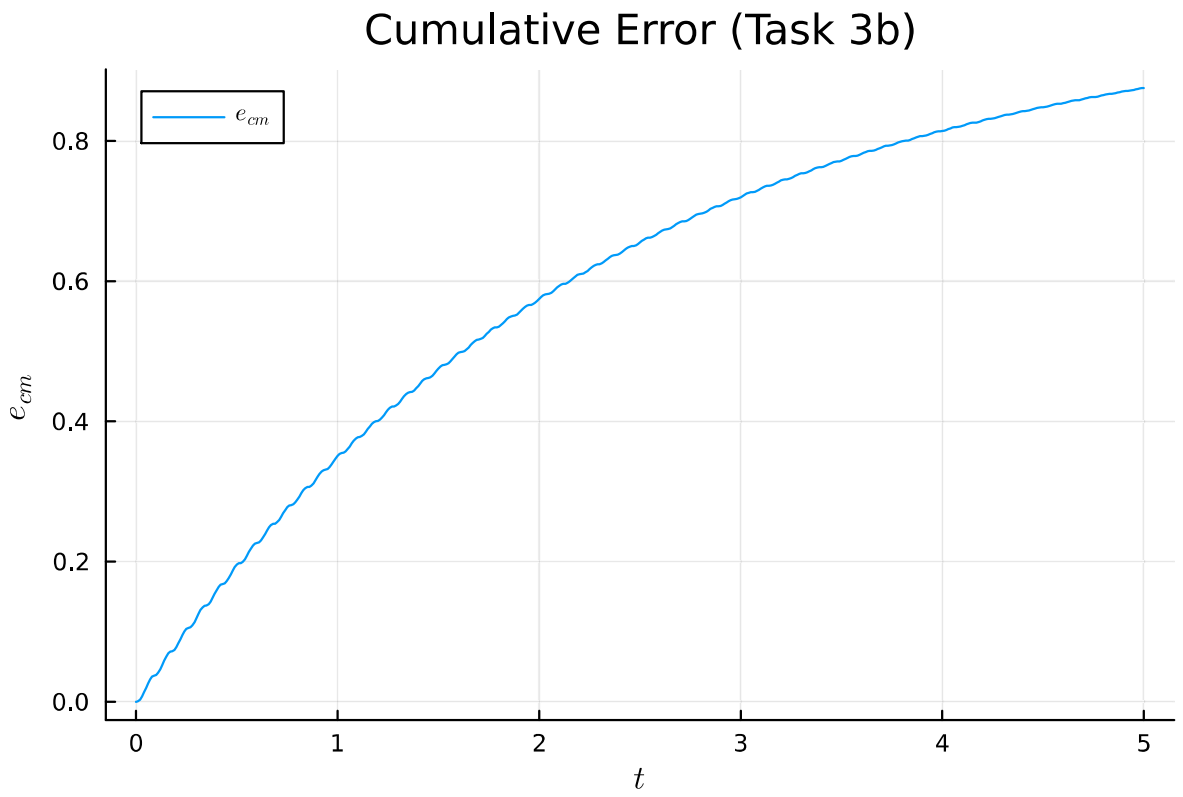

```
begin
    plot(time_a.times,err_a.e_cum,title="Cumulative Error (Task
    3a)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm}$")
end
```




```
begin
    plot(time_b.times,err_b.e_abs,title="Absolute Error (Task
    3b)",xlabel=L"t",ylabel=L"||e||",label=L"$||e||$")
end
```




```
begin
    plot(time_b.times,err_b.η,title="Relative Error (Task
    3b)",xlabel=L"t",ylabel=L"η",label=L"$\eta$")
end
```




```
begin
    plot(time_b.times,err_b.e_cum,title="Cumulative Error (Task
    3b)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm}$")
end
```

Task 5 (Adaptive time stepping algorithim):

5.1. We start by setting the current time to the initial time $t \leftarrow t_o$, and the current time step with the initil time step $\Delta t \leftarrow \Delta t_o$. 5.2. Loop as long as current time is less or equal final time $(t \leq t_f)$: 5.2.1. calculate the same objects as in the `generalized_alpha`.

> **Note**
>
> Unlike `generalized_alpha`, K_{eff} now has to be calculated inside the loop, because Δt is now changing.

5.2.2. At the end of the loop we check the relative error (η), whether it is inside the boundary $[\nu_1 \eta_e, \nu_2 \eta_e]$ or outside, and if it is outside, update Δt:

$$\Delta t \leftarrow \begin{cases} \Delta t \sqrt{\frac{\eta_e}{\eta}}, & \text{for } \eta \leq \nu_1 \eta_e \text{ or } \eta \geq \nu_2 \eta_e \\ \Delta t, & \text{otherwise} \end{cases}$$

5.2.3. Update current time (t):

$$t \leftarrow t + \Delta t$$

```
struct AdaptiveTimeBoundary
    ν₁::Float64
    ν₂::Float64
    ηₑ::Float64
end
```

```
generalized_alpha_adaptive (generic function with 1 method)
```

```
function generalized_alpha_adaptive(tf,Δt₀, α₁, α₂,
ρ_∞,boundary::AdaptiveTimeBoundary)
    # calculate damping
    C = α₁*M + α₂ * K

    # Initial conditions
    u_o = [0 , - L /5 ]
    ud_o = [sqrt(g/(6*L)),0]
    udd_o = inv(M) * (R - C * ud_o - K * u_o)

    # Chung and Hulbert (1993)
    α_m = (2*ρ_∞ - 1)/(ρ_∞ + 1)
    α_f = (ρ_∞)/(ρ_∞+1)
    β = 0.25 * (1 - α_m + α_f)^2
    γ = 0.5 - α_m + α_f



    # create time interval range to loop over later
    ν₁ = boundary.ν₁
    ν₂ = boundary.ν₂
    ηₑ = boundary.ηₑ
    lb = ν₁ * ηₑ
    ub = ν₂ * ηₑ
    t₀ = 0.0 # initial time
    t_current = t₀ # current time

    # define our respone containers
    u = zeros(2,0)
    ud = zeros(2,0)
    udd = zeros(2,0)
    t = [t₀]
    tstep = [Δt₀]
    Δt = Δt₀ # current time step

    # set initial conditions in our response containers
    u = hcat(u,u_o)
    ud =hcat( ud,ud_o)
    udd = hcat(udd,udd_o)

    # define the containers for errors
    e_abs = [0.0] # absolute error
    η = [0.0] # relative error
    e_cum = [0.0] # cummulative error
    while t_current <= tf

        K_eff = M * ((1-α_m)/(β*Δt^2)) + C * (γ * (1 - α_f))/(β*Δt) + K * (1 - α_f)

        uₙ = u[:,end] # current displacement
        udₙ = ud[:,end] # current velocity
        uddₙ = udd[:,end] # current acceleration

        r_eff = -K * α_f * uₙ +
```

```
                C * (((γ * (1-α_f))/(β*Δt)) * uₙ + ((γ - γ*α_f - β)/(β)) * udₙ +
                (((γ-2*β)*(1-α_f))/(2*β)) * Δt*uddₙ) +
                M * (((1-α_m)/(β*Δt^2))*uₙ + ((1-α_m)/(β*Δt))* udₙ + ((1-α_m - 2
                *β)/(2*β)) * uddₙ)

        # solve for u.
        uₙ₁ = K_eff \ r_eff # u_{n+1} next displcement
        u = hcat(u, uₙ₁)

        # update velocity and acceleration
        udₙ₁ = ((γ)/(β*Δt)) * (uₙ₁ - uₙ) - ((γ - β)/(β))*udₙ - ((γ - 2*β)/(2*β)) *
        Δt * uddₙ
        ud = hcat(ud,udₙ₁)

        uddₙ₁ = (1/(β*Δt^2))*(uₙ₁ - uₙ) - (1/(β*Δt)) * udₙ - ((1-2*β)/(2*β)) * uddₙ
        udd = hcat(udd,uddₙ₁)

        # calculate errors
            e_absₙ₁ = norm(((6*β - 1)/(6))*(uddₙ₁ - uddₙ) * Δt^2)
            push!(e_abs,e_absₙ₁)
        ηₙ₁ = e_absₙ₁ / norm(uₙ₁ - uₙ)
        push!(η,ηₙ₁)
        push!(e_cum, sum(e_abs))

        # adapting the time step
        if ηₙ₁  > ub || ηₙ₁ < lb
            # here means Δt is either to big ()
            Δt = Δt * sqrt(ηₑ/ηₙ₁)
        end
        t_current += Δt
        push!(t,t_current)
        push!(tstep,Δt)

    end

    (response =  DynamicResponse(u,ud,udd), error =  Error(e_abs,η,e_cum),time =
    TimeData(t,tstep))
end
```

Task 6:

Given $\alpha_1 = 1$, $\alpha_2 = 0$, and $\rho_\infty = 1$, It's required to:

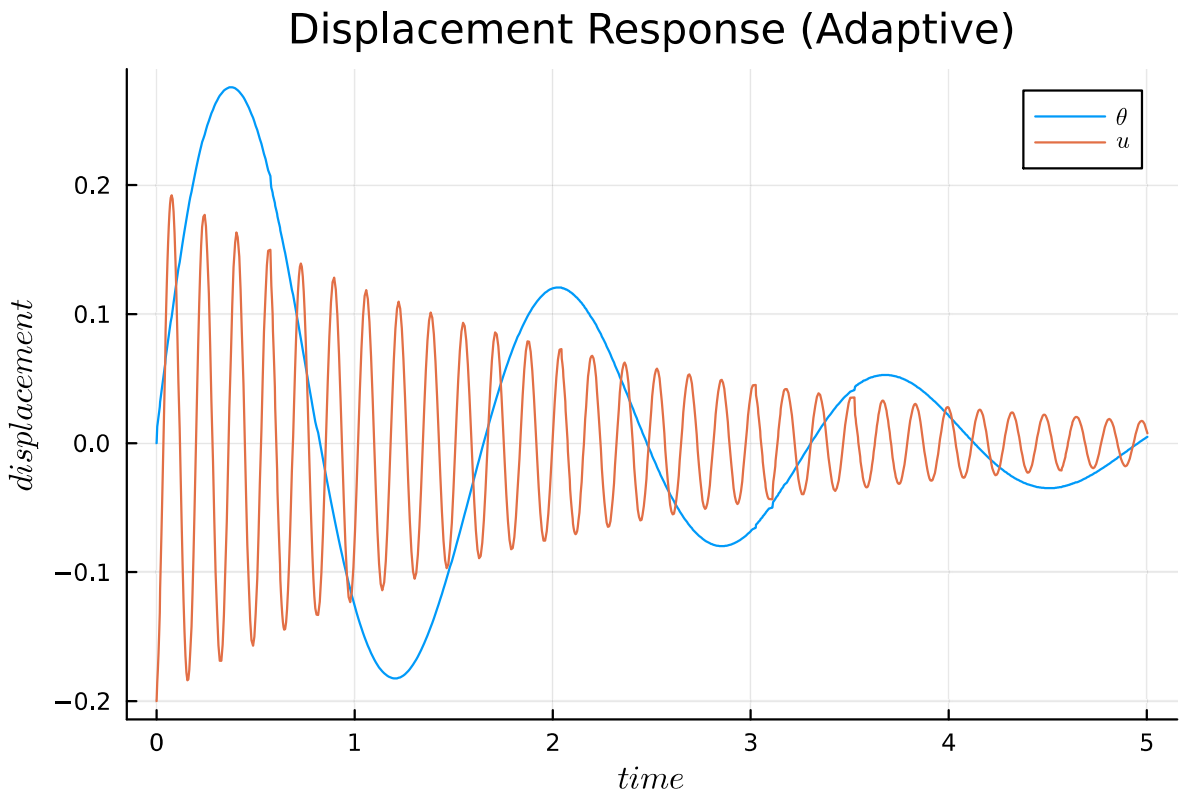
- Solve the system for 5.0 s.
- Plot the dynamic response, error, and the evolution of time step.
- Compare the results with those from task 3a.
- Cumulative error for both cases after 5.0 s and the number of time steps.

```
time_bound =   AdaptiveTimeBoundary(1.0, 10.0, 0.001)
```

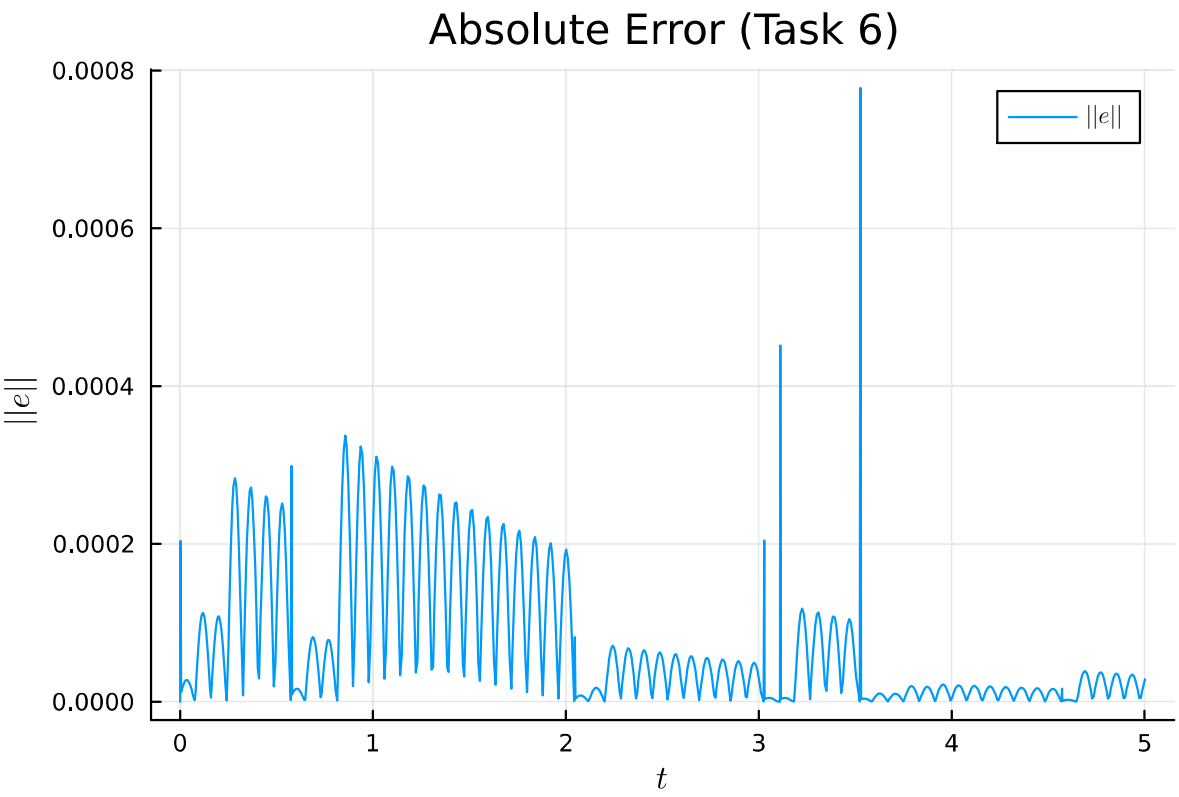
```
time_bound = AdaptiveTimeBoundary(1.0,10.0,1e-3)
```

```
(response = DynamicResponse(2×871 Matrix{Float64}:
                            0.0   0.012712   0.0165854   0.0204446  …  0.00336907  0.00
```

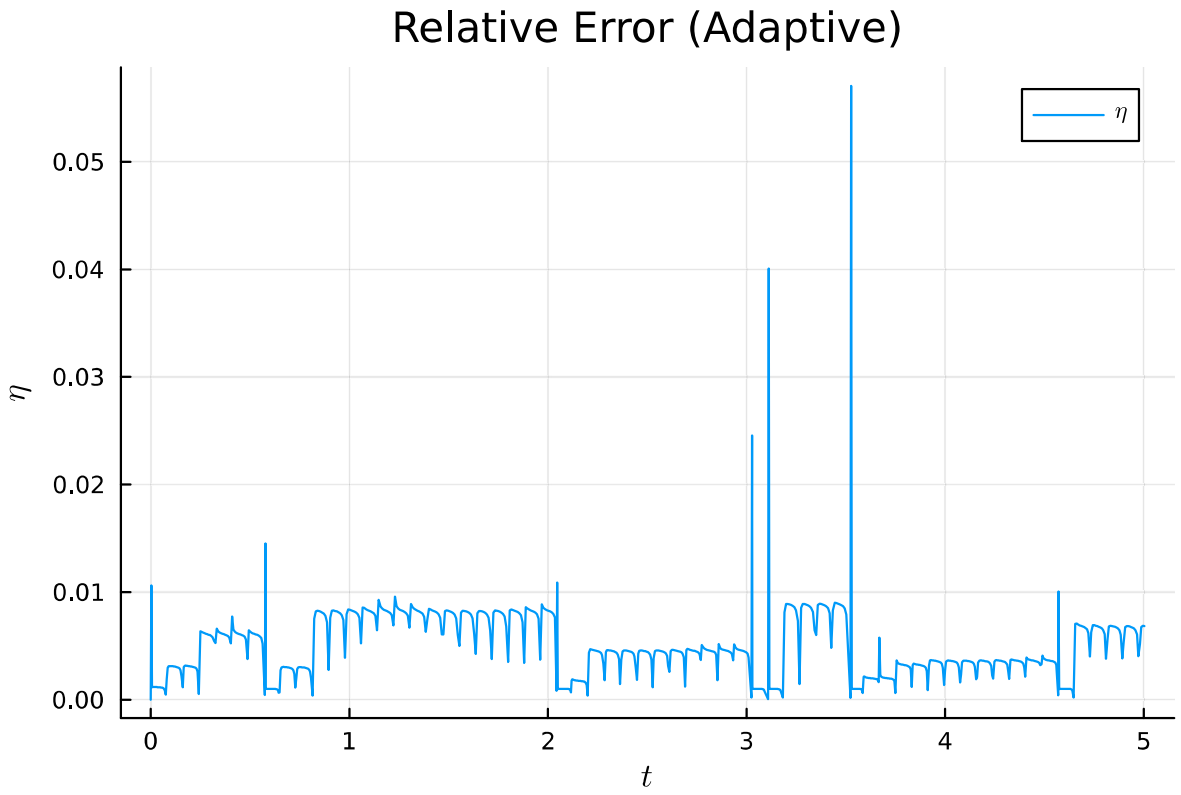
```
1 ad_resp , ad_err,ad_time = generalized_alpha_adaptive(5.0,0.01, 1.0, 0.0,
  1.0,time_bound)
```



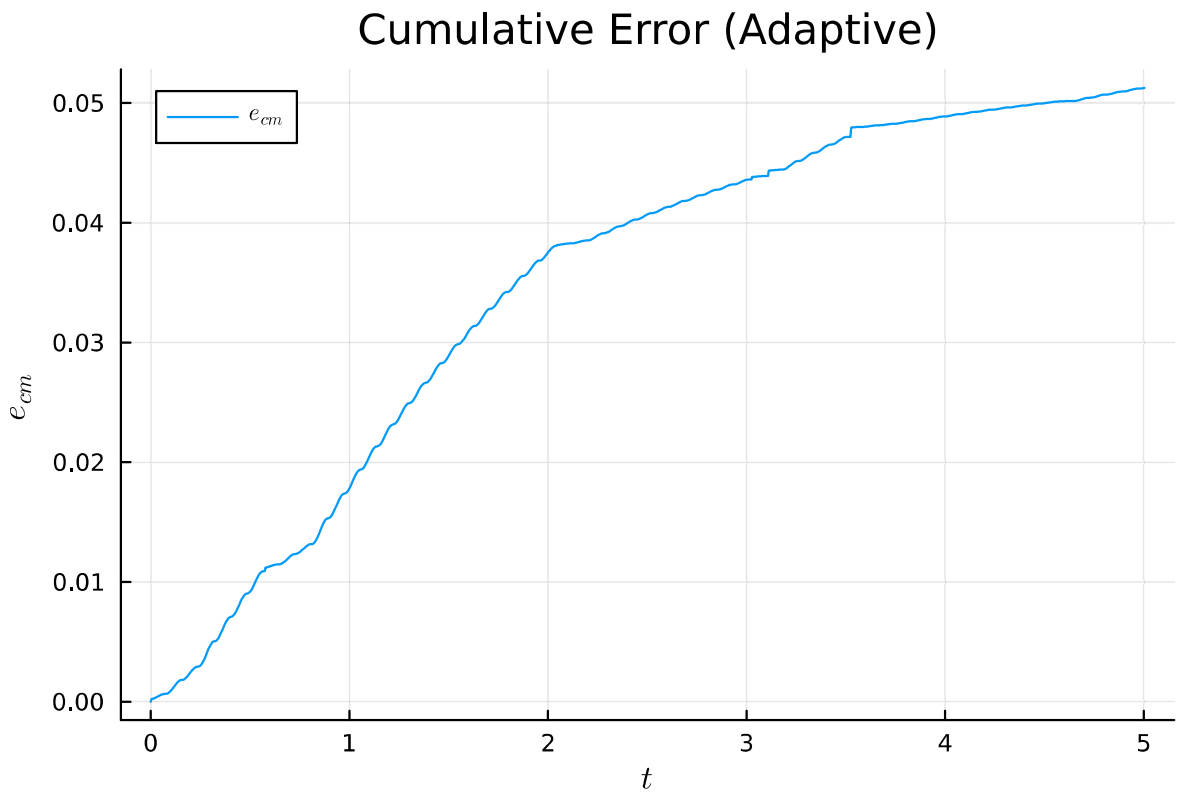

```
1 begin
2     plot(ad_time.times,ad_resp.u[1,:],title="Displacement Response
      (Adaptive)",xlabel=L"time",ylabel=L"displacement",label=L"$\theta$")
3     plot!(ad_time.times,ad_resp.u[2,:],label=L"u")
4
5 end
```



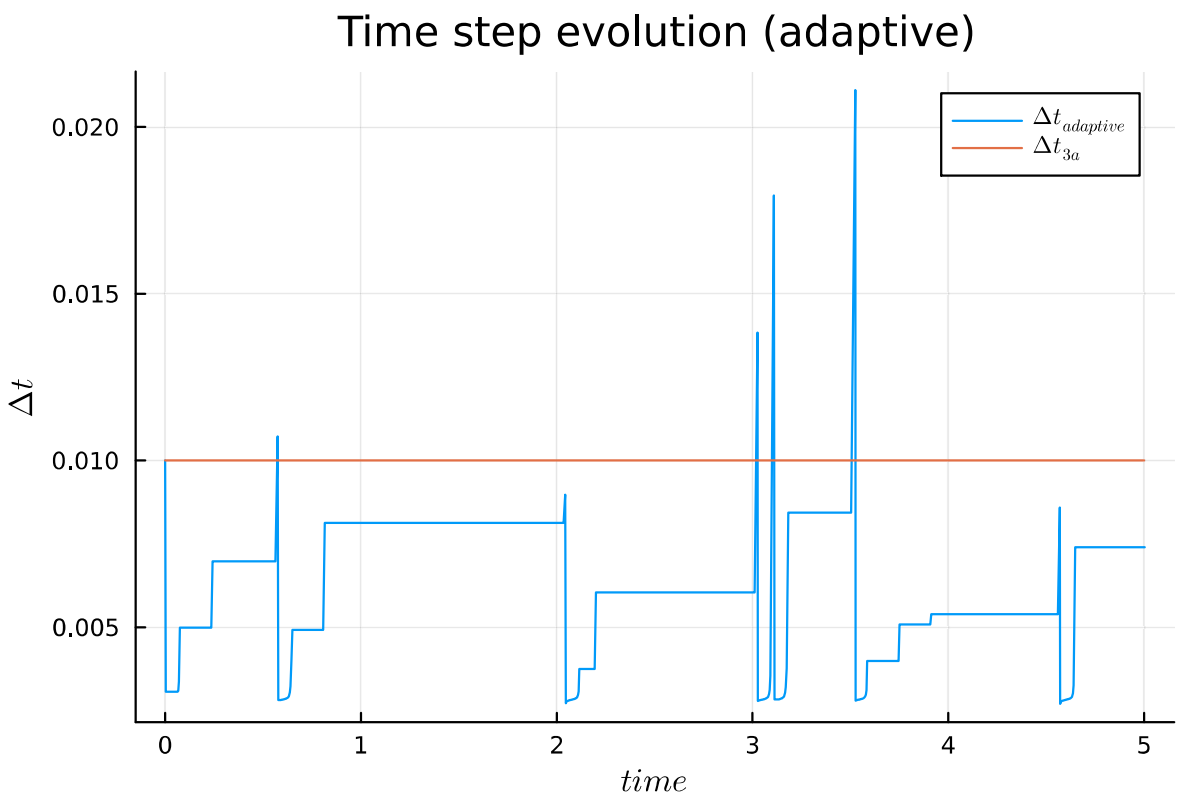

```
begin
    plot(ad_time.times,ad_err.e_abs,title="Absolute Error (Task
    6)",xlabel=L"t",ylabel=L"||e||",label=L"$||e||$")
end
```



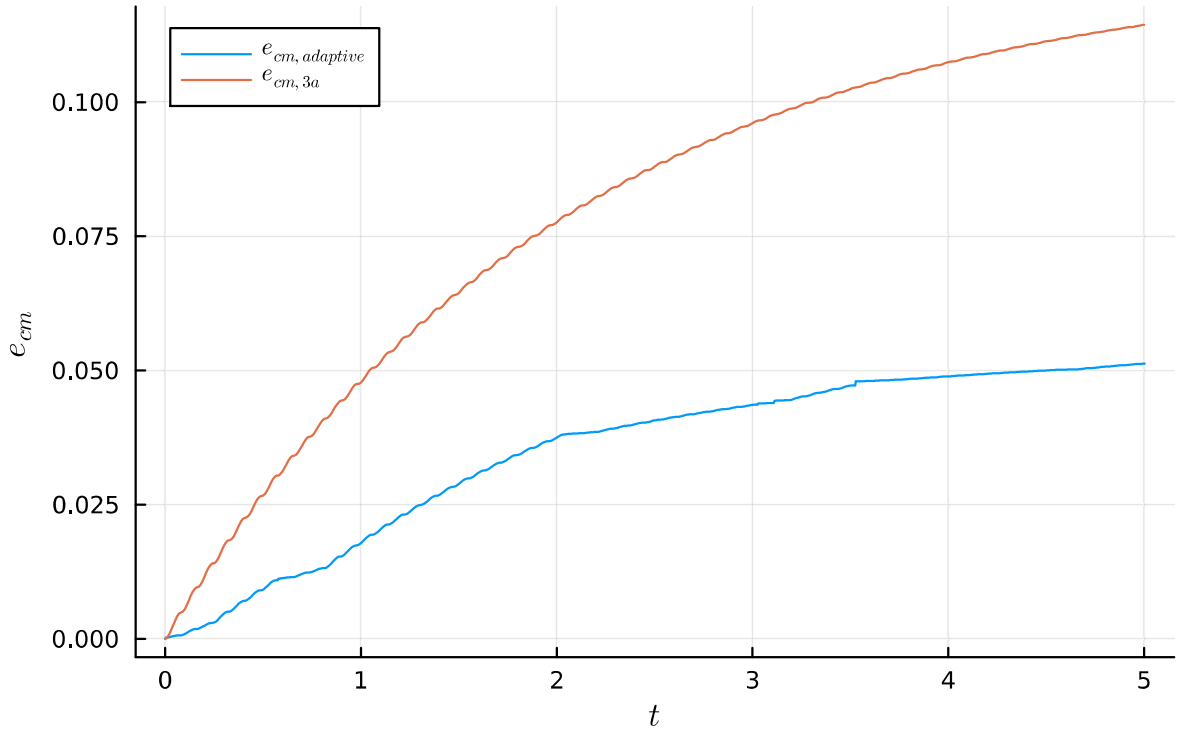

```
begin
    plot(ad_time.times,ad_err.η,title="Relative Error
    (Adaptive)",xlabel=L"t",ylabel=L"η",label=L"$\eta$")
end
```




```
begin
    plot(ad_time.times,ad_err.e_cum,title="Cumulative Error
    (Adaptive)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm}$")
end
```




```
begin
    plot(ad_time.times,ad_time.steps,title="Time step evolution
    (adaptive)",xlabel=L"time",ylabel=L"Δt",label=L"$Δt_{adaptive}$")
    plot!(time_a.times,time_a.steps,label=L"Δt_{3a}")
end
```




```
begin
    plot(ad_time.times,ad_err.e_cum,title="Cumulative Error (Adaptive & Task
    3a)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm, adaptive}$")
    plot!(time_a.times,err_a.e_cum,label=L"$e_{cm, 3a}$")
end
```

Cumulative error after 5 s:

```
(adapt = 0.0512708, t3a = 0.114384)
```

```
(adapt = ad_err.e_cum[end],t3a =err_a.e_cum[end] )
```

> **Note**
>
> From the previous result, we can observe that the cumulative error in the adaptive method is less than the fixed one and this was anticipated.

Number of steps:

```
(adapt = 871, t3a = 501)
```

```
(adapt = length(ad_time.steps),t3a = length(time_a.steps) )
```