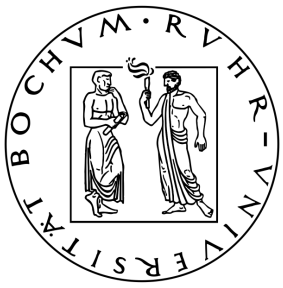


Ruhr Universität Bochum

Homework 2: Generalized-Alpha method

Abdelrahman Fathy Abdelhaleem Aly Abdelrahman
Matr.-Nr.: 108023251500

Course: Applied FEM Homework (Dynamic Part)
Supervisors: Prof. Dr. Roger Sauer, Dr-Ing. Sahir Nawaz Butt
Date: June 14, 2024

Part I:
Julia Code and Explanation

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 M & K matrices, as follows:

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

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

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

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

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

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

with,

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

Note

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

1.1. Data inputs:

$$\Delta t = 0.01\ s$$
$$L = 1.0\ m$$
$$\rho A = 1.0\ kg/m$$
$$EA = 500.0\ N$$
$$g = 9.8\ m/s^2$$

```
9.8
```

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

```
(
    M = 2×2 Matrix{Float64}:
         0.333333  0.0
         0.0       0.333333
    K = 2×2 Matrix{Float64}:
         4.9    0.0
         0.0  500.0
    R =  [0, 0]
)
```

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

```
(uₒ = [0.0, -0.2], vₒ = [1.27802, 0.0])
```

```
begin
    # Initial Conditions
    uₒ = [0 ; -L/5] # initial displacement
    vₒ = [sqrt(g/6L) ; 0] # initial velocity
    (;uₒ,vₒ)
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
```

1.2. Semi-discrete equation of motion:

$$\boldsymbol{M} = \begin{bmatrix} 0.3333 & 0 \\ 0 & 0.3333 \end{bmatrix}, \boldsymbol{K} = \begin{bmatrix} 4.9 & 0 \\ 0 & 500.0 \end{bmatrix}$$

when plugging in *Rayleigh* damping (i.e. $\boldsymbol{C}$) by $\alpha_1 = 1.0, \alpha_2 = 0.0$ will yield:

$$\boldsymbol{C} = \begin{bmatrix} 0.3333 & 0 \\ 0 & 0.3333 \end{bmatrix}$$

finally, we can write the final form of the semi-discrete equation of motion as follows:

$$\underbrace{\begin{bmatrix} 0.3333 & 0 \\ 0 & 0.3333 \end{bmatrix}}_{\boldsymbol{M}} \cdot \underbrace{\begin{bmatrix} \ddot{\theta} \\ \ddot{u} \end{bmatrix}}_{\ddot{\boldsymbol{u}}} + \underbrace{\begin{bmatrix} 0.3333 & 0 \\ 0 & 0.3333 \end{bmatrix}}_{\boldsymbol{C}} \cdot \underbrace{\begin{bmatrix} \dot{\theta} \\ \dot{u} \end{bmatrix}}_{\dot{\boldsymbol{u}}} + \underbrace{\begin{bmatrix} 4.9 & 0 \\ 0 & 500.0 \end{bmatrix}}_{\boldsymbol{K}} \cdot \underbrace{\begin{bmatrix} \theta \\ u \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\boldsymbol{R}}$$

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:

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

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:

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

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:

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

2.3.2. solve for u_{n+1} as follows:

$$u_{n+1} = K_{eff}^{-1} \cdot r_{eff}$$

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

$$\dot{u}_{n+1} = \frac{\gamma}{\beta \Delta t}(u_{n+1} - u_n) - \frac{\gamma - \beta}{\beta} \dot{u}_n - \frac{\gamma - 2\beta}{2\beta} \Delta t \ddot{u}_n \rightarrow \text{(slides pg. 60)}$$

$$\ddot{u}_{n+1} = \frac{1}{\beta \Delta t^2}(u_{n+1} - u_n) - \frac{1}{\beta \Delta t} \dot{u}_n - \frac{1 - 2\beta}{2\beta} \ddot{u}_n \rightarrow \text{(slides pg. 60)}$$

> **Note**
>
> The following structs are used to define the **input parameters** for the *Generalized-Alpha* method.

```
struct LinearMomentum
    M::Matrix{Float64} # mass matrix
    K::Matrix{Float64} # stiffness matrix
    R::Vector{Float64} # load vector (i.e. right hand side)
end
```

```
struct InitialConditions
    u₀::Vector{Float64} # initial displacement
    v₀::Vector{Float64} # initial velocity
end
```

```
# define abstract supertype for damping that will be inherited by
# physcial (i.e. α₁, α₂) and numerical damping (i.e. ρ_∞)
abstract type DampingParameters end
```

```
struct PhysicalDamping <: DampingParameters
    α₁::Float64
    α₂::Float64
end
```

```julia
struct NumericalDamping <: DampingParameters
    ρ_∞::Float64
end
```

```julia
struct RunTime
    tf::Float64 # amount of time we solve our system for
    Δt::Float64 # fixed time step or initial time step for the adaptive method.
end
```

> **Note**
>
> The following structs are used to define the **output parameters** for the *Generalized-Alpha* method.

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

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

```julia
struct TimeEvolution
    times::Vector{Float64} # current time vector.
    steps::Vector{Float64} # Δt vector
end
```

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

```
function generalized_alpha(moment::LinearMomentum,ic::InitialConditions,
time::RunTime, pd::PhysicalDamping,nd::NumericalDamping)

    M = moment.M # mass matrix
    K = moment.K # stiffness matrix

    α₁ = pd.α₁
    α₂ = pd.α₂

    # calculate damping
    C = α₁*M + α₂ * K

    # Initial conditions
    u_o = ic.u₀
    ud_o = ic.v₀
    udd_o = inv(M) * (R - C * ud_o - K * u_o)

    # Chung and Hulbert (1993)
    ρ_∞ = nd.ρ_∞
    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
    tf = time.tf
    Δt = time.Δt
    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

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

end
```

Task: Solve the system:

```
(
    momentum =   LinearMomentum(
                    M = 2×2 Matrix{Float64}:
                         0.333333  0.0
                         0.0       0.333333
                    K = 2×2 Matrix{Float64}:
                         4.9    0.0
                         0.0  500.0
                    R =   [0.0, 0.0]
                )
    initial_coditions =   InitialConditions(
                            u₀ =   [0.0, -0.2]
                            v₀ =   [1.27802, 0.0]
                        )
    runtime =   RunTime(
                    tf = 5.0
                    Δt = 0.01
                )
)
```

```
begin
    # common input parameters for both task 3a & 3b
    momentum = LinearMomentum(M,K,R) # linear momentum parameters
    initial_coditions = InitialConditions(u₀,v₀) # initial conditions
    runtime = RunTime(5.0,0.01)  # 0.01
    (;momentum,initial_coditions,runtime)
end
```

Task 3a:

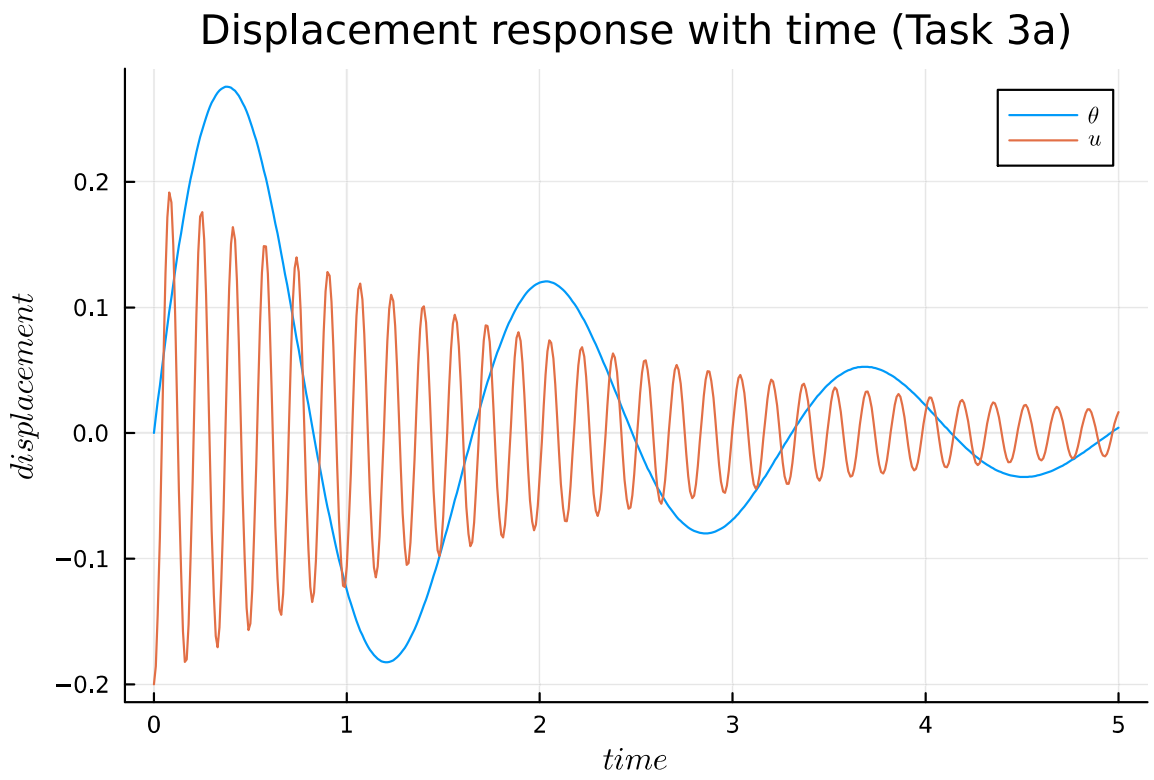
Given: $t_{final} = 5\ s, \alpha_1 = 1, \alpha_2 = 0, \rho_\infty = 1.0$

```
(
    physical_damping_a =   PhysicalDamping(
                               α₁ = 1.0
                               α₂ = 0.0
                           )
    numerical_damping_a =   NumericalDamping(
                                ρ_∞ = 1.0
                            )
)
```

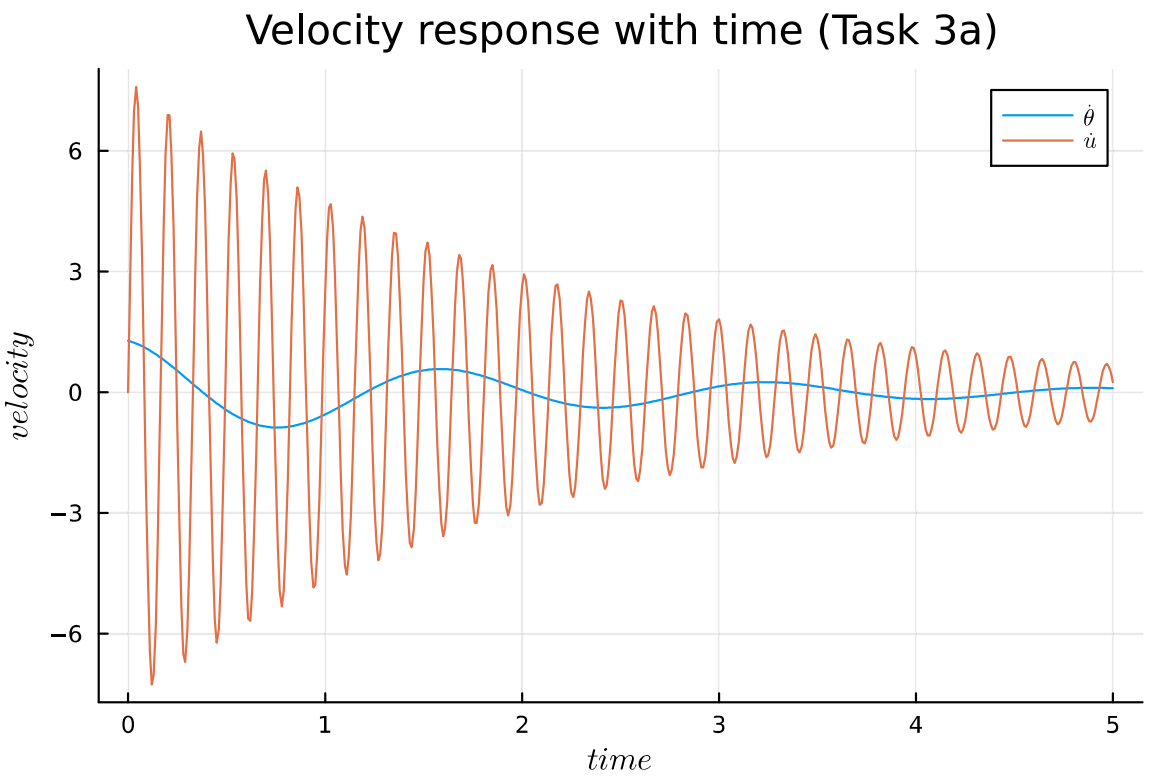
```
begin # parameters specific for task 3a.
    physical_damping_a = PhysicalDamping(1.0,0.0) # (α₁ = 1.0, α₂=0.0)
    numerical_damping_a = NumericalDamping(1.0) # (ρ_∞ = 1.0)
    (;physical_damping_a,numerical_damping_a)
end
```

```
(
    response =   DynamicResponse(2×501 Matrix{Float64}:
                                  0.0   0.012712   0.0252789   0.0376839  …  0.00219718
    errors =   Error([0.0, 0.000203842, 0.000532267, 0.000780959, 0.000914898,    more ,4.3

    time =   TimeEvolution([0.0, 0.01, 0.02, 0.03, 0.04,    more ,5.0], [0.01, 0.01, 0.01, (
)
```

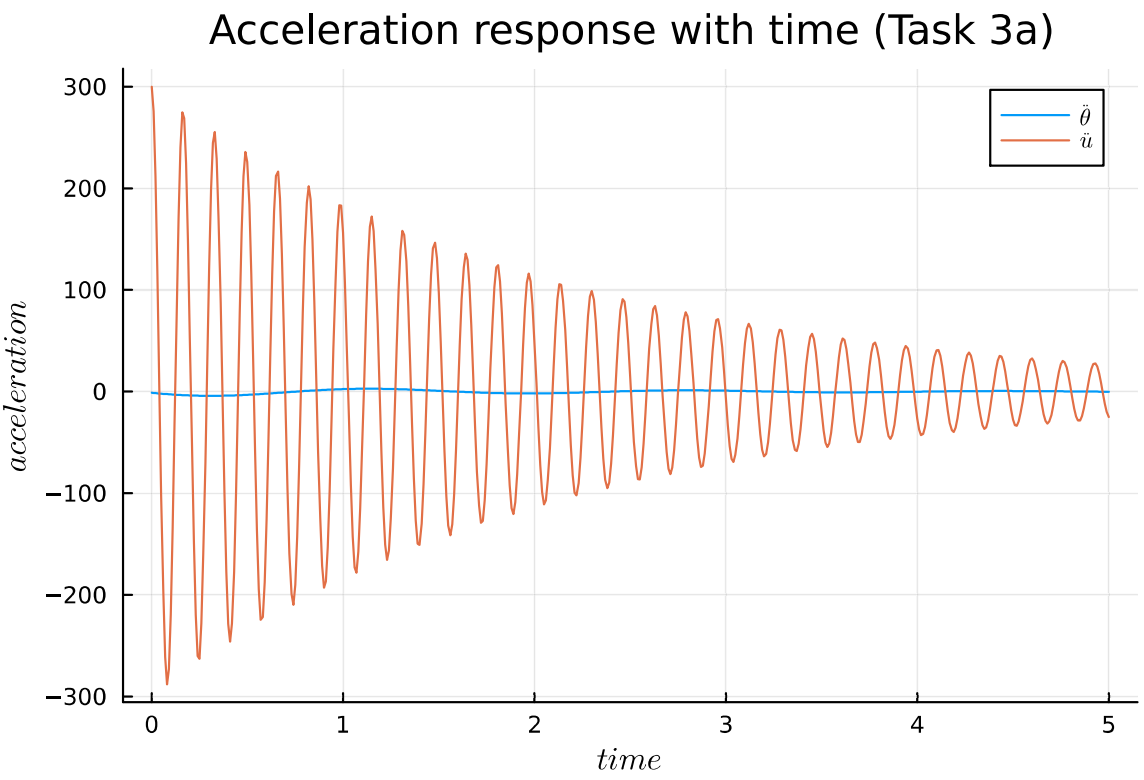
```
resp_a , err_a ,time_a=
generalized_alpha(momentum,initial_coditions,runtime,physical_damping_a,numerical_damp
ing_a)
```




```
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
```




```
begin
    plot(time_a.times,resp_a.ud[1,:],title="Velocity response with time (Task
    3a)",xlabel=L"time",ylabel=L"velocity",label=L"$\dot{\theta}$")
    plot!(time_a.times,resp_a.ud[2,:],label=L"\dot{u}")
end
```




```
begin
    plot(time_a.times,resp_a.udd[1,:],title="Acceleration response with time (Task
    3a)",xlabel=L"time",ylabel=L"acceleration",label=L"$\ddot{\theta}$")
    plot!(time_a.times,resp_a.udd[2,:],label=L"\ddot{u}")
end
```

Task 3b:

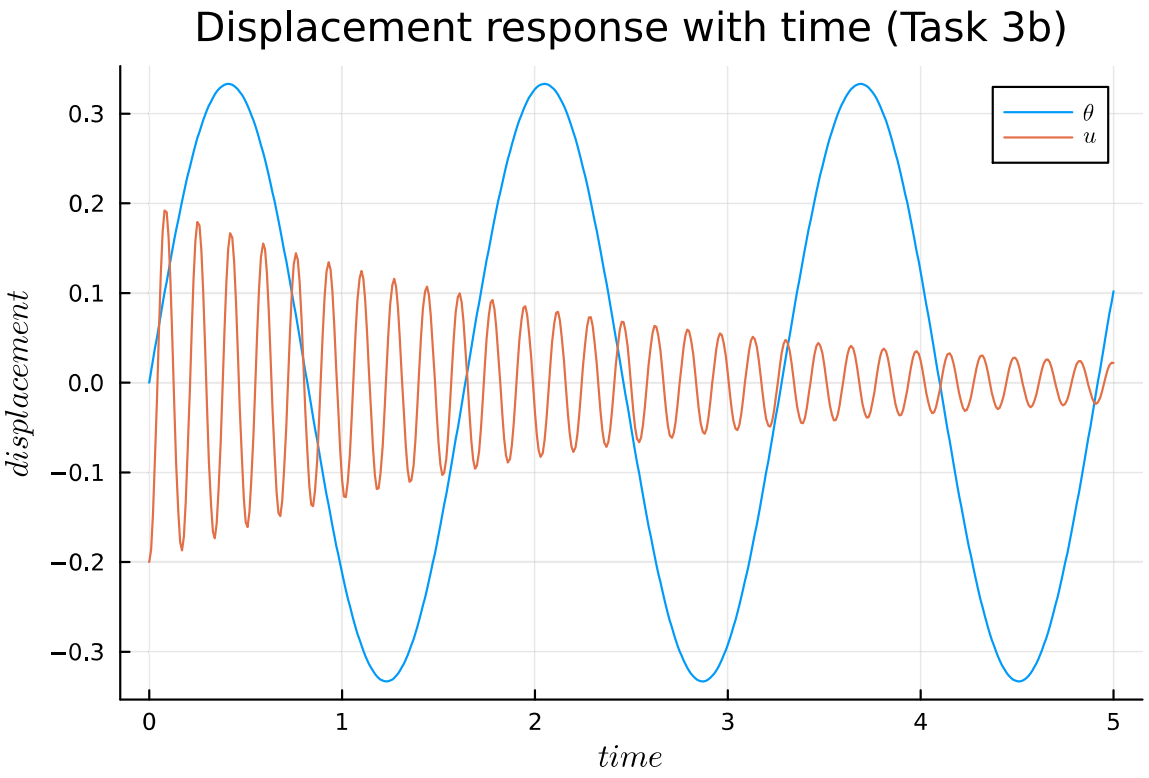
Given: $t_{final} = 5\ s, \alpha_1 = 0, \alpha_2 = 0, \rho_\infty = 0.1$

```
(
    physical_damping_b =   PhysicalDamping(
                               α₁ = 0.0
                               α₂ = 0.0
                           )
    numerical_damping_b =   NumericalDamping(
                                ρ_∞ = 0.1
                            )
)
```

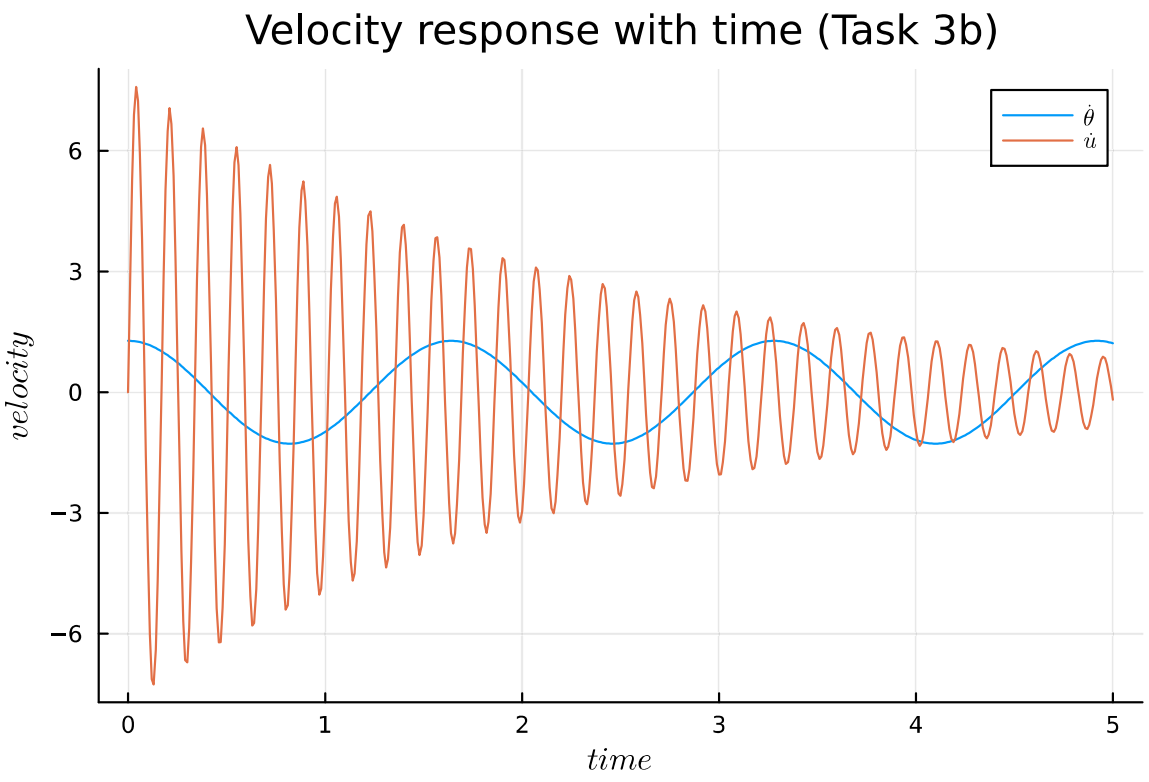
```
begin # parameters specific for task 3b.
    physical_damping_b = PhysicalDamping(0.0,0.0) # (α₁ = 0.0, α₂ = 0.0)
    numerical_damping_b = NumericalDamping(0.1) # (ρ_∞ = 0.1)
    (;physical_damping_b,numerical_damping_b)
end
```

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

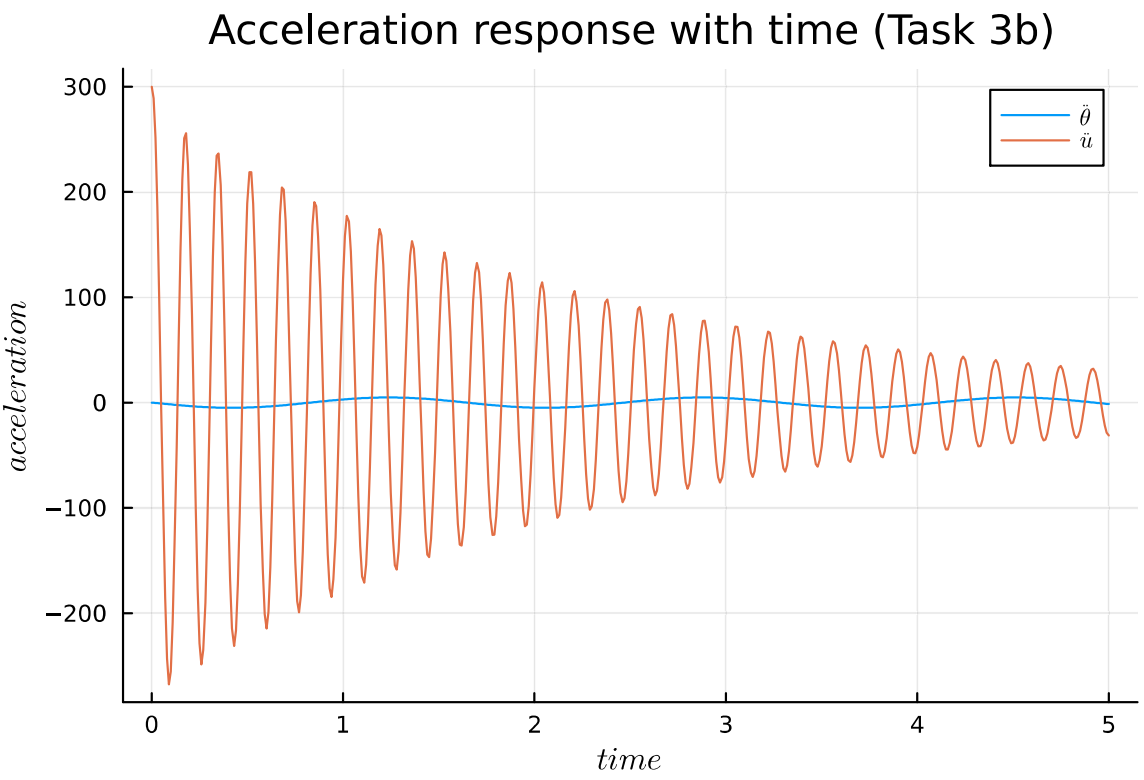
```
resp_b , err_b, time_b=
generalized_alpha(momentum,initial_coditions,runtime,physical_damping_b,numerical_damping_b)
```




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




```
begin
    plot(time_b.times,resp_b.ud[1,:],title="Velocity response with time (Task
    3b)",xlabel=L"time",ylabel=L"velocity",label=L"$\dot{\theta}$")
    plot!(time_b.times,resp_b.ud[2,:],label=L"\dot{u}")
end
```




```
begin
    plot(time_b.times,resp_b.udd[1,:],title="Acceleration response with time (Task
    3b)",xlabel=L"time",ylabel=L"acceleration",label=L"$\ddot{\theta}$")
    plot!(time_b.times,resp_b.udd[2,:],label=L"\ddot{u}")
end
```

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 $\boldsymbol{\theta}$ remains constant through time.

Explanation:

The reason behind such observation is; $\boldsymbol{\alpha_1}$ & $\boldsymbol{\alpha_2}$ 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, ρ_∞ 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, $\gamma = \frac{1}{2} \rightarrow \rho_\infty = 1$ (slides pg. 59) and numerical damping only occurs if $\gamma > \frac{1}{2} \rightarrow \rho_\infty < 1$

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 regarding the implementation and the error graphs as well.

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

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

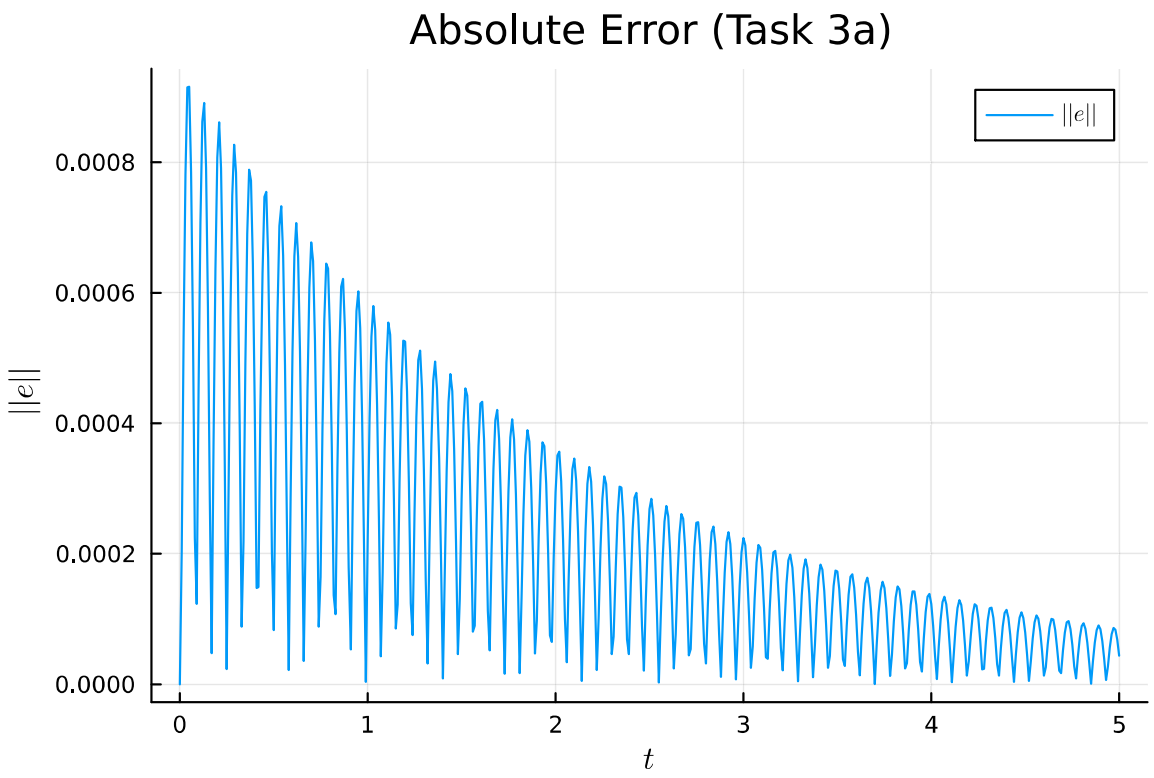
4.2. Relative error:

$$\eta = \frac{||\boldsymbol{e}||}{||\boldsymbol{u}_{n+1} - \boldsymbol{u}_n||} \rightarrow \text{(slides pg. 91)}$$

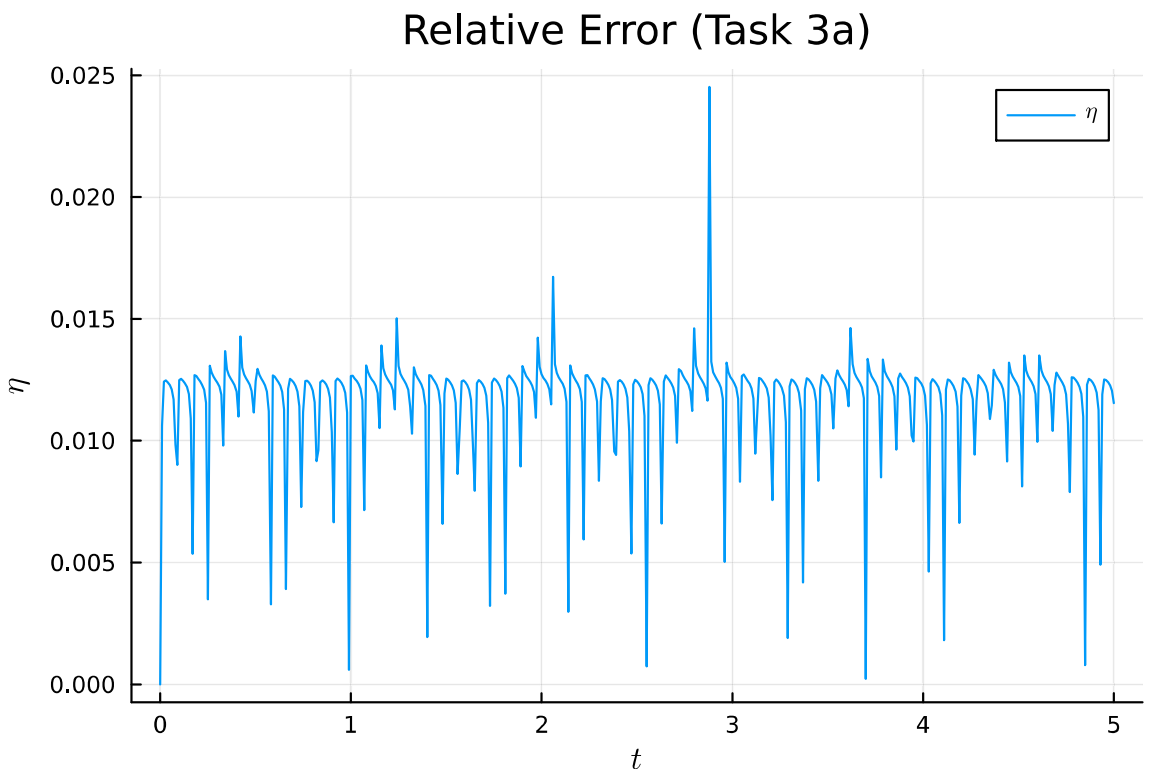
4.3. Cummulative error:

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

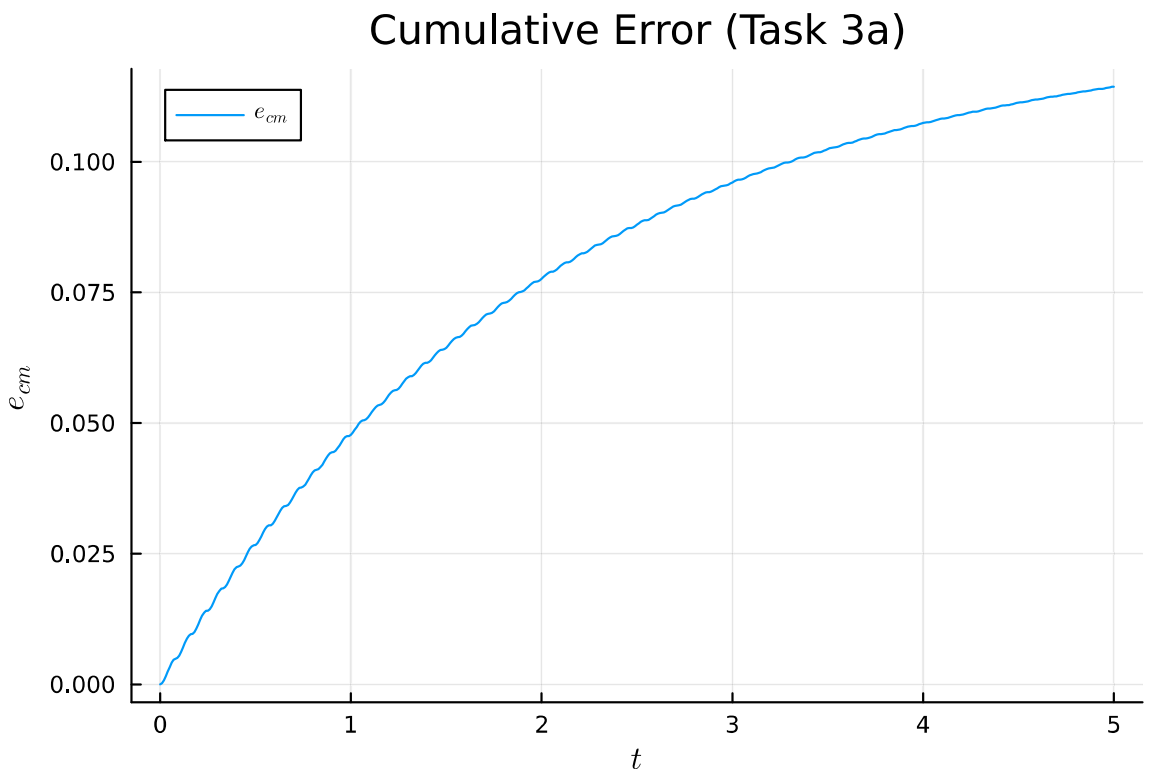
where $\boldsymbol{n}$ is the index of the current time step.



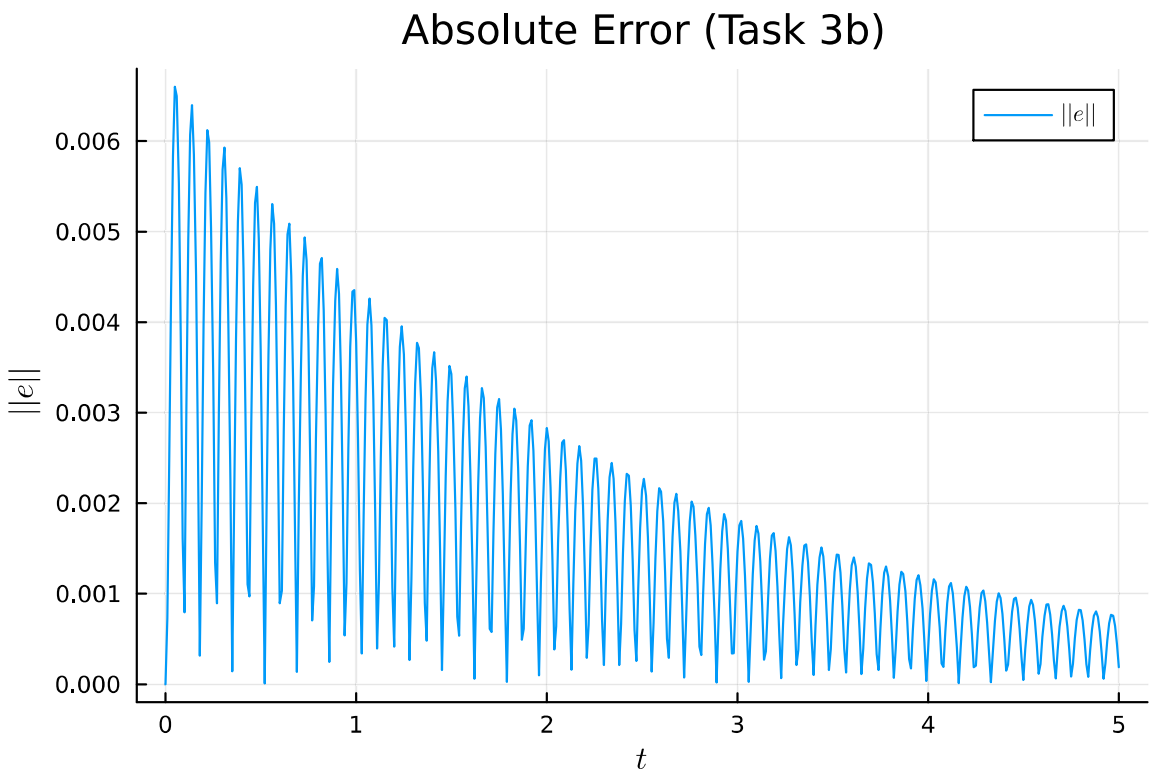

```
begin
    plot(time_a.times,err_a.e_abs,title="Absolute Error (Task
    3a)",xlabel=L"t",ylabel=L"||e||",label=L"$||e||$")
end
```



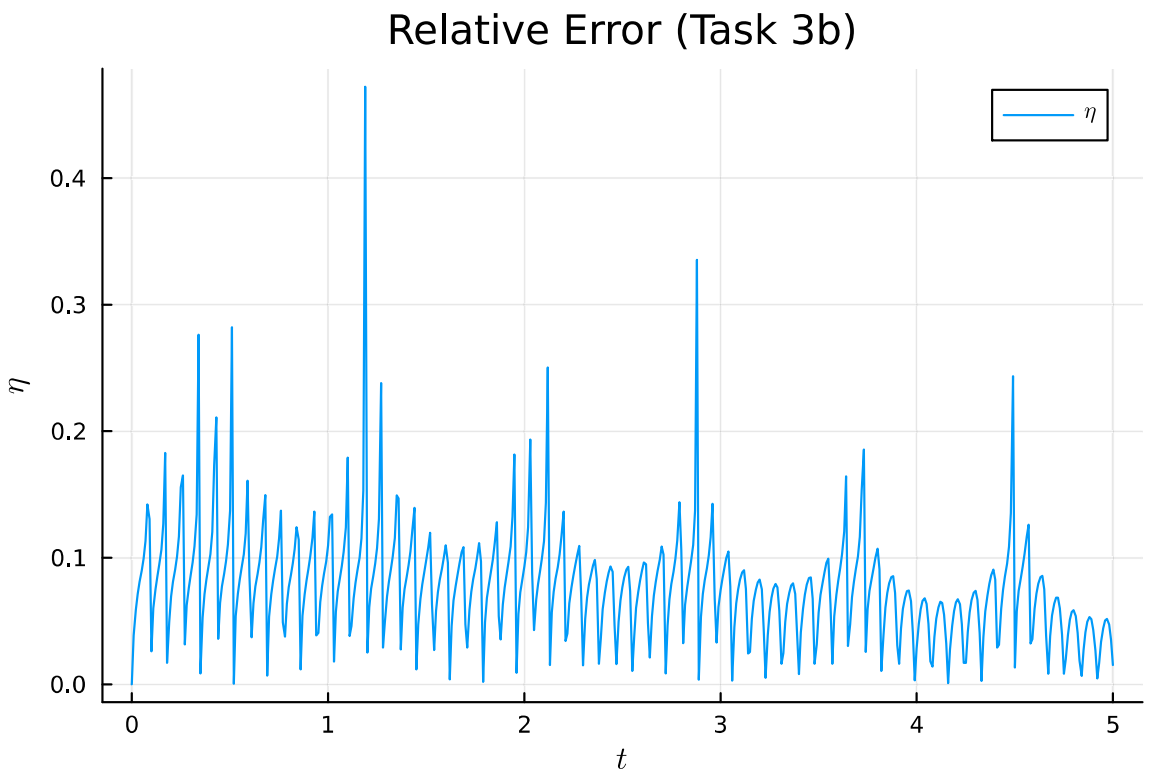

```
begin
    plot(time_a.times,err_a.η,title="Relative Error (Task
    3a)",xlabel=L"t",ylabel=L"η",label=L"$\eta$")
end
```



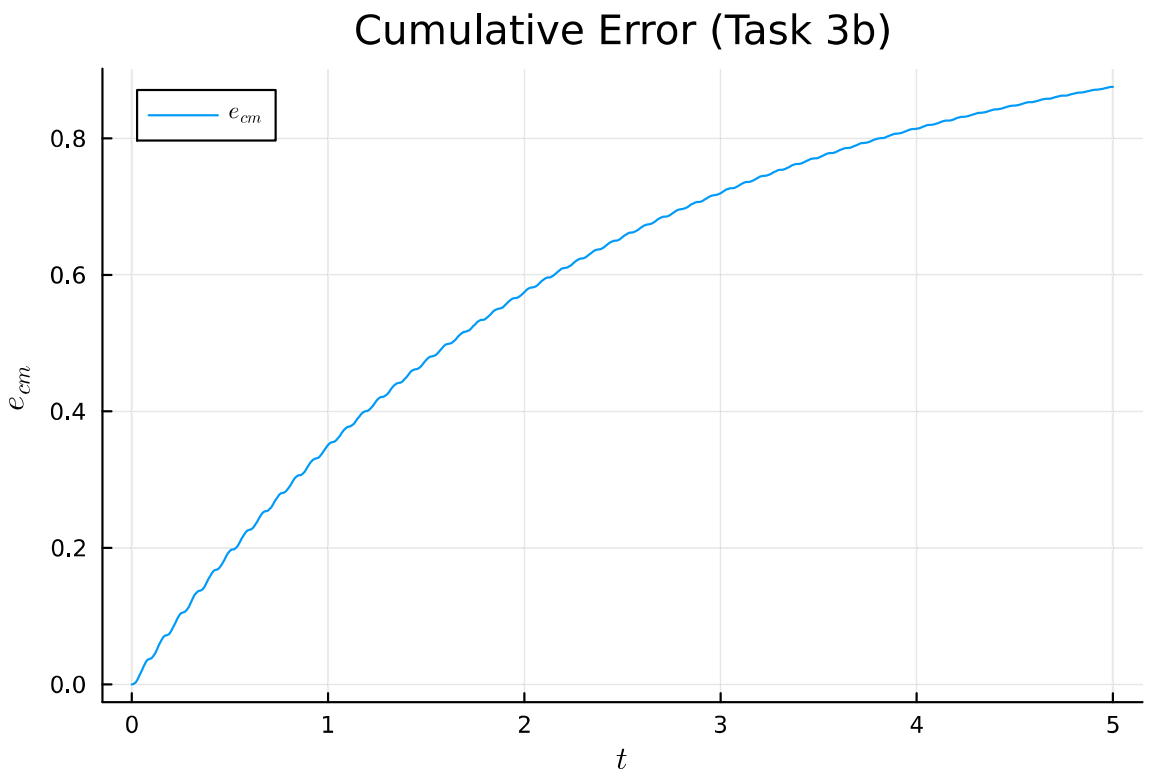

```
begin
    plot(time_a.times,err_a.e_cum,title="Cumulative Error (Task
    3a)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm}$")
end
```




```
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):

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

```julia
# this struct represents out time boundary that we need to stay in.
struct AdaptiveTimeBoundary
    ν₁::Float64
    ν₂::Float64
    ηₑ::Float64
end
```

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

```
function generalized_alpha_adaptive(moment::LinearMomentum,ic::InitialConditions,
time::RunTime,pd::PhysicalDamping,nd::NumericalDamping,boundary::AdaptiveTimeBoundary
)

    M = moment.M # mass matrix
    K = moment.K # stiffness matrix

    α₁ = pd.α₁
    α₂ = pd.α₂

    # calculate damping
    C = α₁*M + α₂ * K

    # Initial conditions
    u_o = ic.uₒ
    ud_o = ic.vₒ
    udd_o = inv(M) * (R - C * ud_o - K * u_o)

    # Chung and Hulbert (1993)
    ρ_∞ = nd.ρ_∞
    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



    # define the boundary for the adaptive newmark
    ν₁ = boundary.ν₁
    ν₂ = boundary.ν₂
    ηₑ = boundary.ηₑ
    lb = ν₁ * ηₑ # lower boundary
    ub = ν₂ * ηₑ # upper boundary


    # define our respone containers
    u = zeros(2,0)
    ud = zeros(2,0)
    udd = zeros(2,0)

    # time data
    tₒ = 0.0 # initial time
    t_current = tₒ # current time
    tf = time.tf # final time we want to solve for
    Δtₒ = time.Δt # initial time step
    t = [t_current] # container for current time values
    tstep = [Δtₒ] # container for the evolution of time steps
    Δt = Δtₒ # 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)
```

```
52
53      # define the containers for errors
54      e_abs = [0.0] # absolute error
55      η = [0.0] # relative error
56      e_cum = [0.0] # cummulative error
57
58      # counter for the number of iterations.
59      i = 1
60
61      while t_current <= tf
62
63          K_eff = M * ((1-α_m)/(β*Δt^2)) + C * (γ * (1 - α_f))/(β*Δt) + K * (1 - α_f)
64
65
66          r_eff = -K * α_f * u[:,i] +
67                  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]) +
68                  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])
69
70          # solve for u.
71          u_{n+1} = K_eff \ r_eff # u_{n+1} next displcement
72          u = hcat(u, u_{n+1})
73
74          # update velocity and acceleration
75          ud_{n+1} = ((γ)/(β*Δt)) * (u[:,i+1] - u[:,i]) - ((γ - β)/(β))*ud[:,i] - ((γ-
            2*β)/(2*β)) * Δt * udd[:,i]
76          ud = hcat(ud,ud_{n+1})
77
78          udd_{n+1} = (1/(β*Δt^2))*(u[:,i+1] - u[:,i]) - (1/(β*Δt)) * ud[:,i] - ((1-
            2*β)/(2*β)) * udd[:,i]
79          udd = hcat(udd,udd_{n+1})
80
81          # calculate errors
82          e_abs_{n+1} = norm(((6*β - 1)/(6))*(udd[:,i+1] - udd[:,i]) * Δt^2)
83          push!(e_abs,e_abs_{n+1})
84          η_{n+1} = e_abs[i+1] / norm(u[:,i+1] - u[:,i])
85          push!(η,η_{n+1})
86          push!(e_cum, sum(e_abs))
87
88          # adapting the time step
89          if η_{n+1}  > ub || η_{n+1} < lb
90              # here means Δt is either to small or to big
91              Δt = Δt * sqrt(η_e/η_{n+1})
92          end
93          t_current += Δt
94          push!(t,t_current)
95          push!(tstep,Δt)
96          i+=1
97
98      end
99
100     (response =  DynamicResponse(u,ud,udd), error =  Error(e_abs,η,e_cum),time =
        TimeEvolution(t,tstep))
101 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)
```

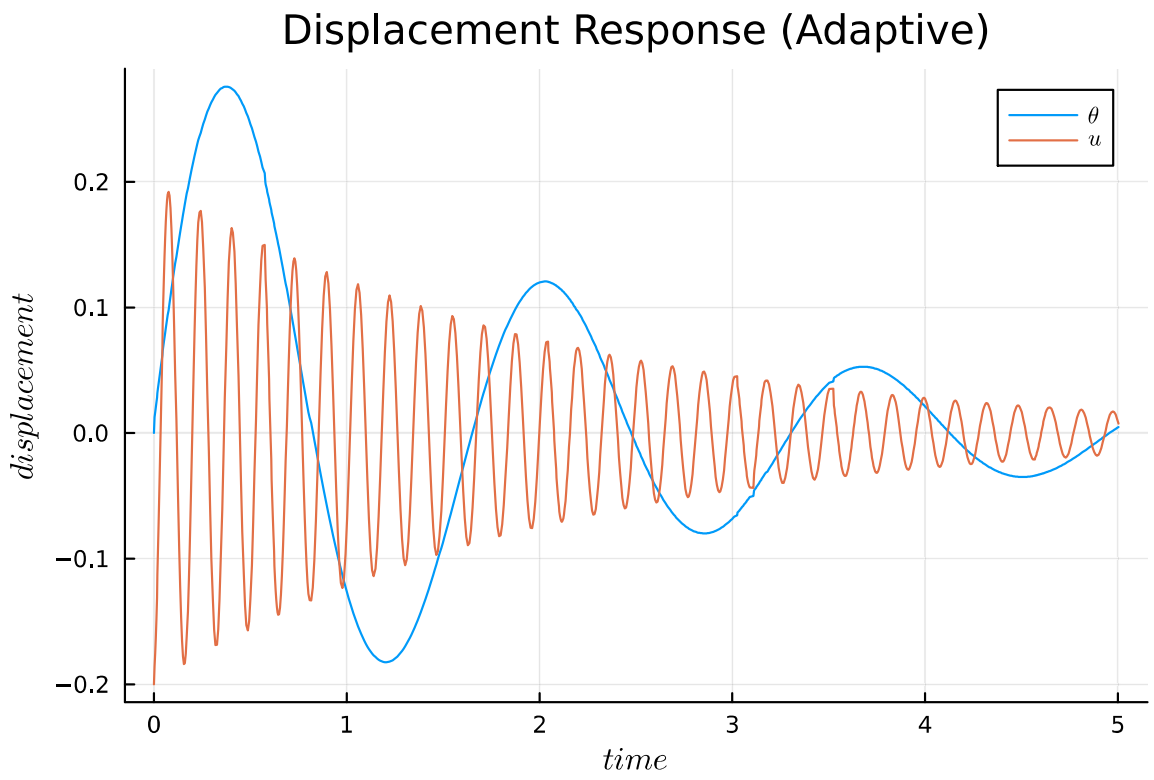
```
1 time_bound = AdaptiveTimeBoundary(1.0,10.0,1e-3)
```

```
(
    response =   DynamicResponse(2×871 Matrix{Float64}:
                                 0.0   0.012712   0.0165854   0.0204446      0.00336907
    error =   Error([0.0, 0.000203842, 1.25261e-5, 1.53519e-5, 1.79533e-5,    more ,2.81044
    time =   TimeEvolution([0.0, 0.00306901, 0.00613802, 0.00920704, 0.012276,    more ,5.0
)
```

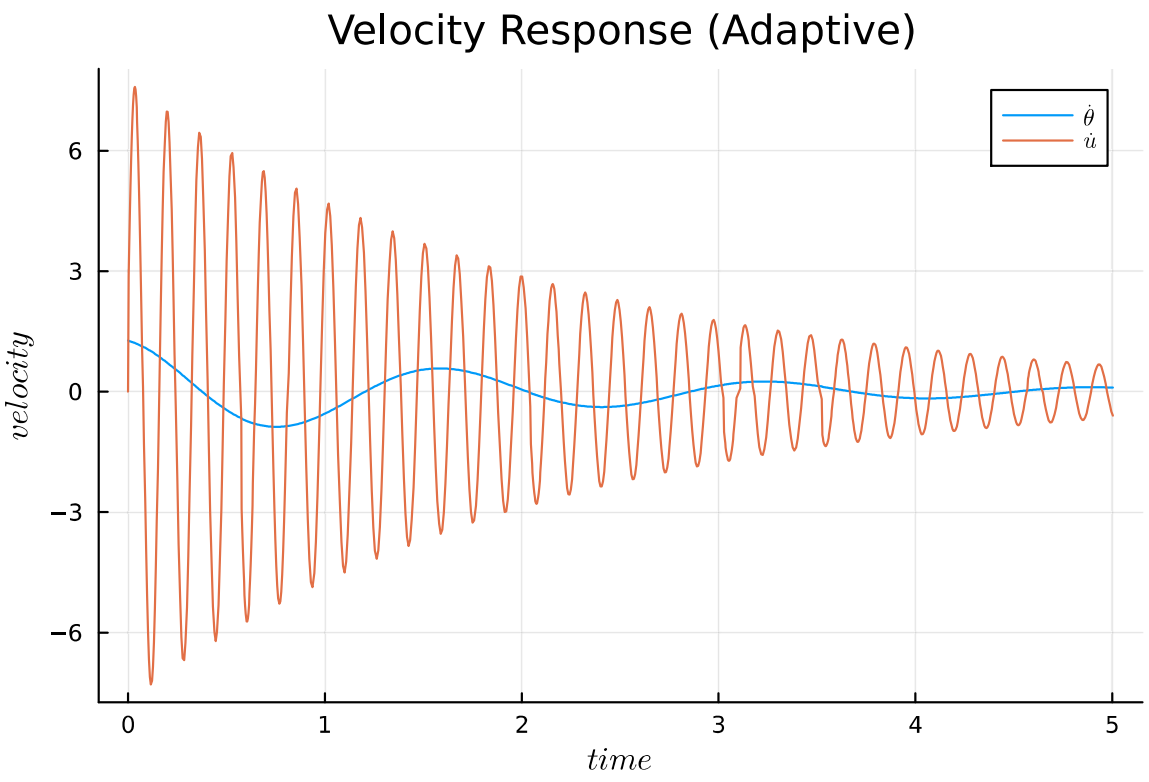
```
1 ad_resp , ad_err,ad_time =
  generalized_alpha_adaptive(momentum,initial_coditions,runtime,physical_damping_a,numer
  ical_damping_a,time_bound)
```

```
2×871 Matrix{Float64}:
  0.0   0.012712   0.0165854   0.0204446  …  0.00336907  0.00412685  0.00487573
 -0.2  -0.185612  -0.17552    -0.162992       0.0147341   0.0116075   0.00757211
```

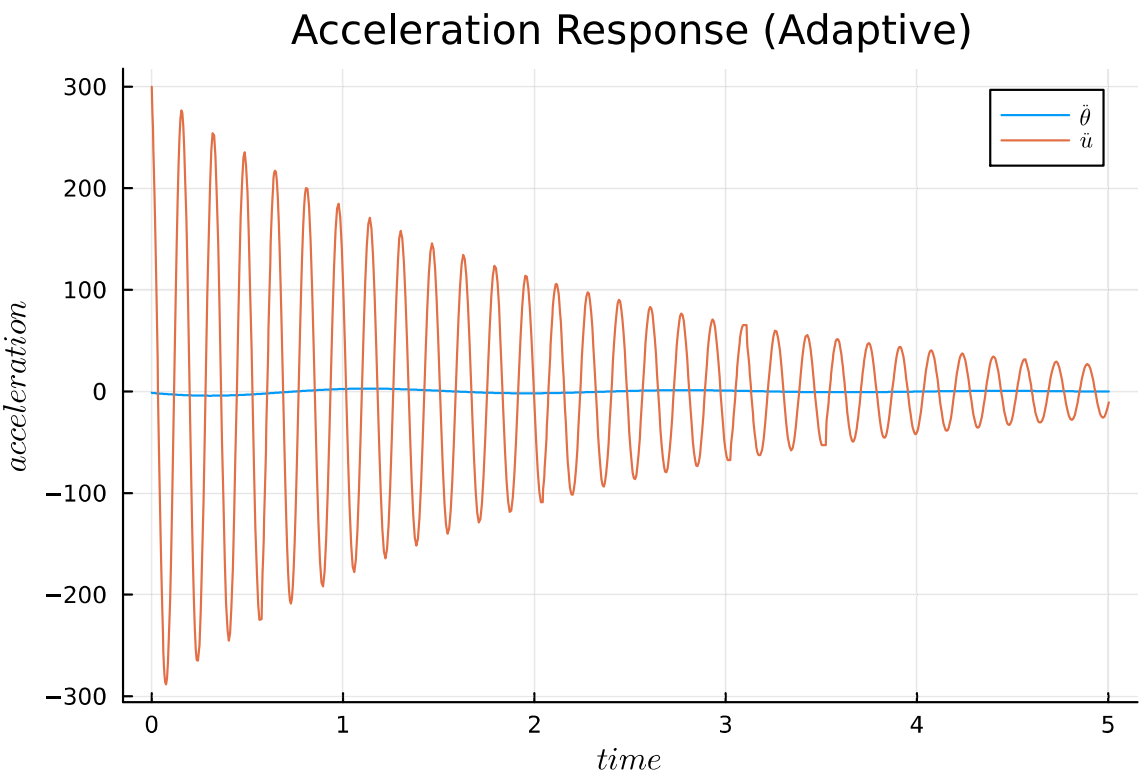
```
1 ad_resp.u
```



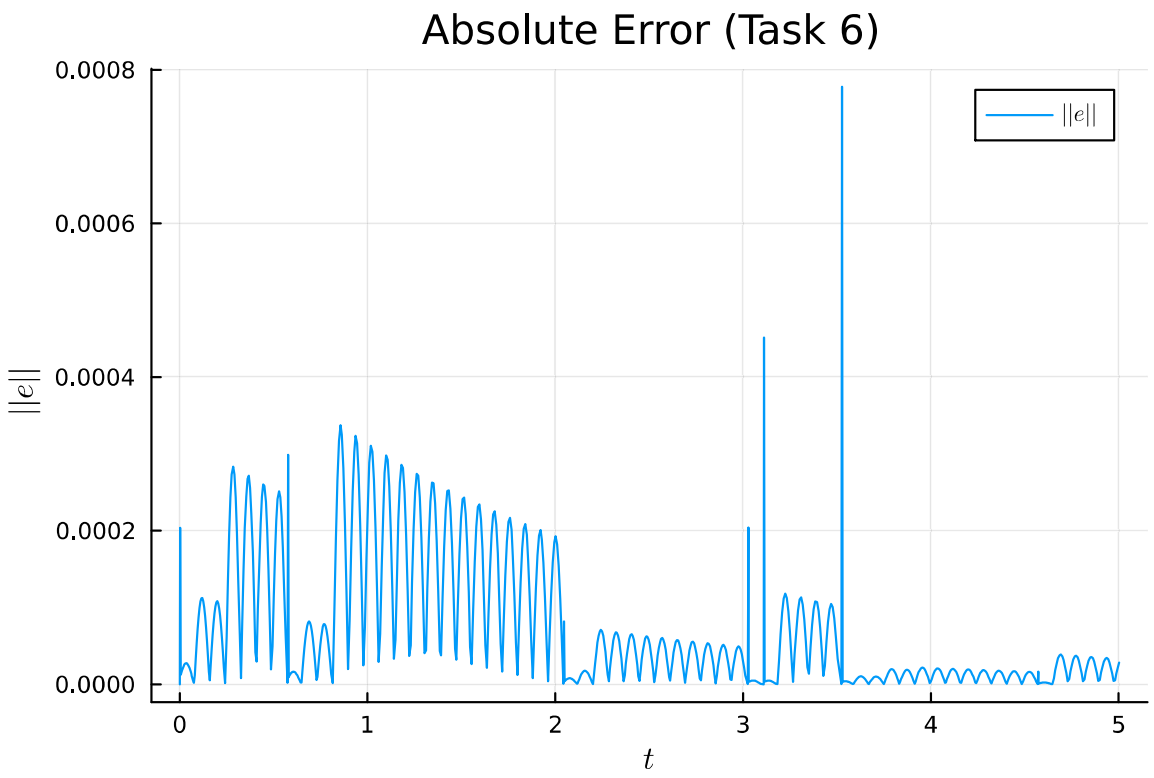

```
begin
    plot(ad_time.times,ad_resp.u[1,:],title="Displacement Response
    (Adaptive)",xlabel=L"time",ylabel=L"displacement",label=L"$\theta$")
    plot!(ad_time.times,ad_resp.u[2,:],label=L"u")

end
```



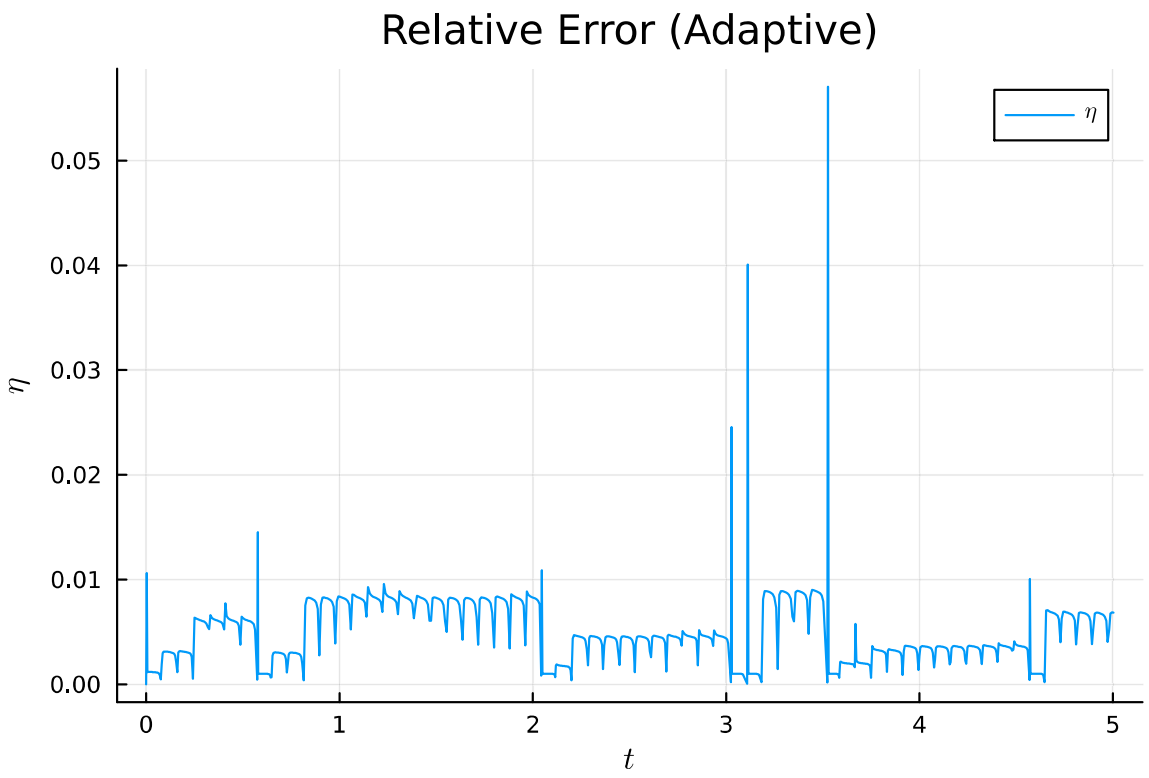

```
begin
    plot(ad_time.times,ad_resp.ud[1,:],title="Velocity Response
    (Adaptive)",xlabel=L"time",ylabel=L"velocity",label=L"$\dot{\theta}$")
    plot!(ad_time.times,ad_resp.ud[2,:],label=L"\dot{u}")
end
```



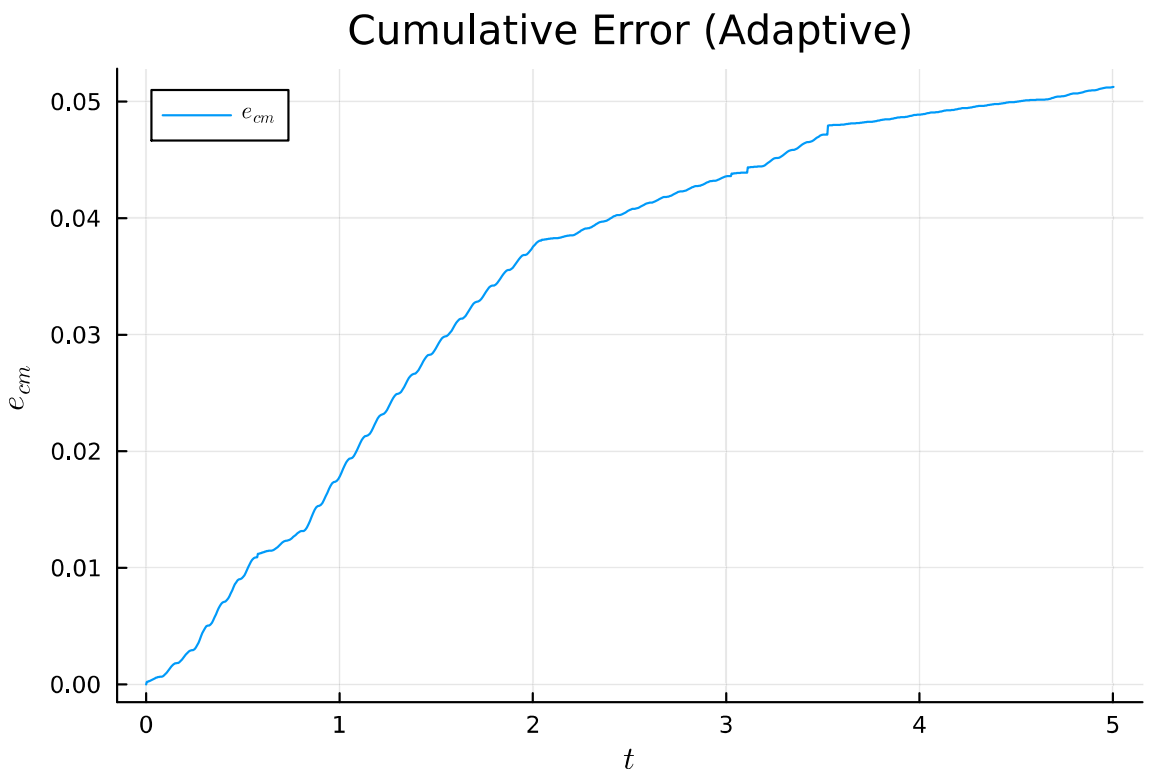

```
begin
    plot(ad_time.times,ad_resp.udd[1,:],title="Acceleration Response
    (Adaptive)",xlabel=L"time",ylabel=L"acceleration",label=L"$\ddot{\theta}$")
    plot!(ad_time.times,ad_resp.udd[2,:],label=L"\ddot{u}")
end
```




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

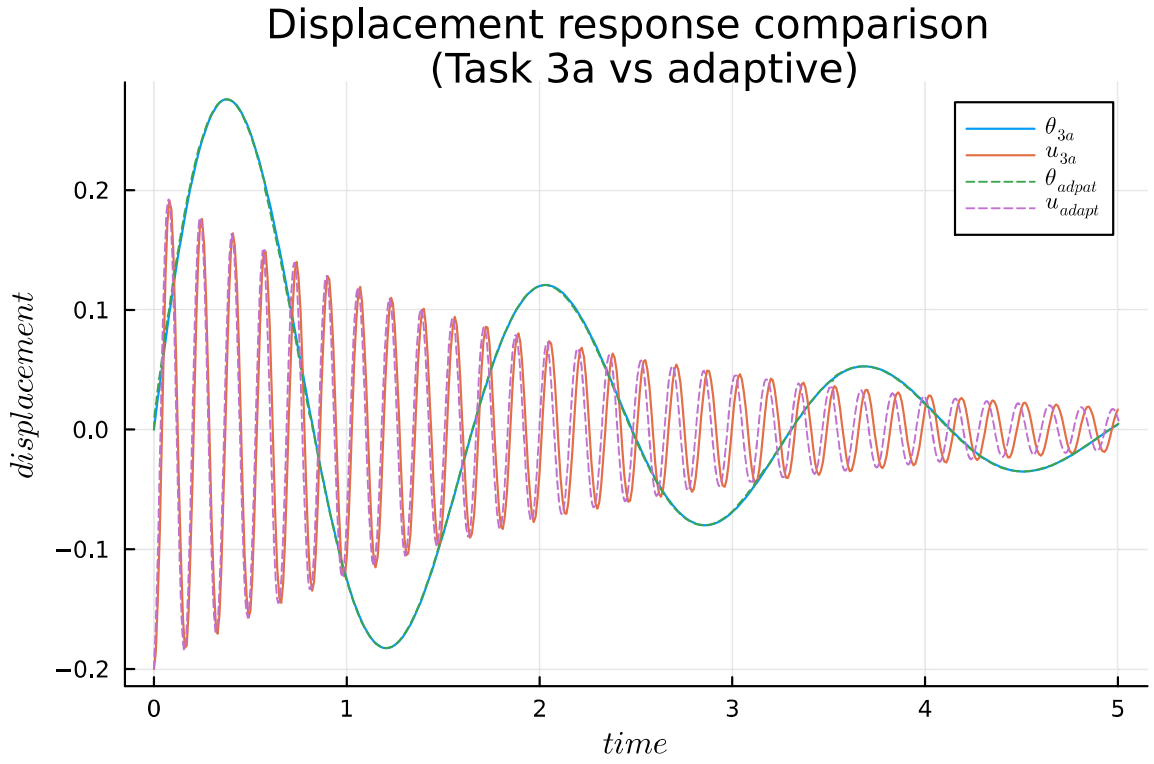



```
begin
    plot(ad_time.times,ad_err.η,title="Relative Error
    (Adaptive)",xlabel=L"t",ylabel=L"η",label=L"$\eta$")
end
```

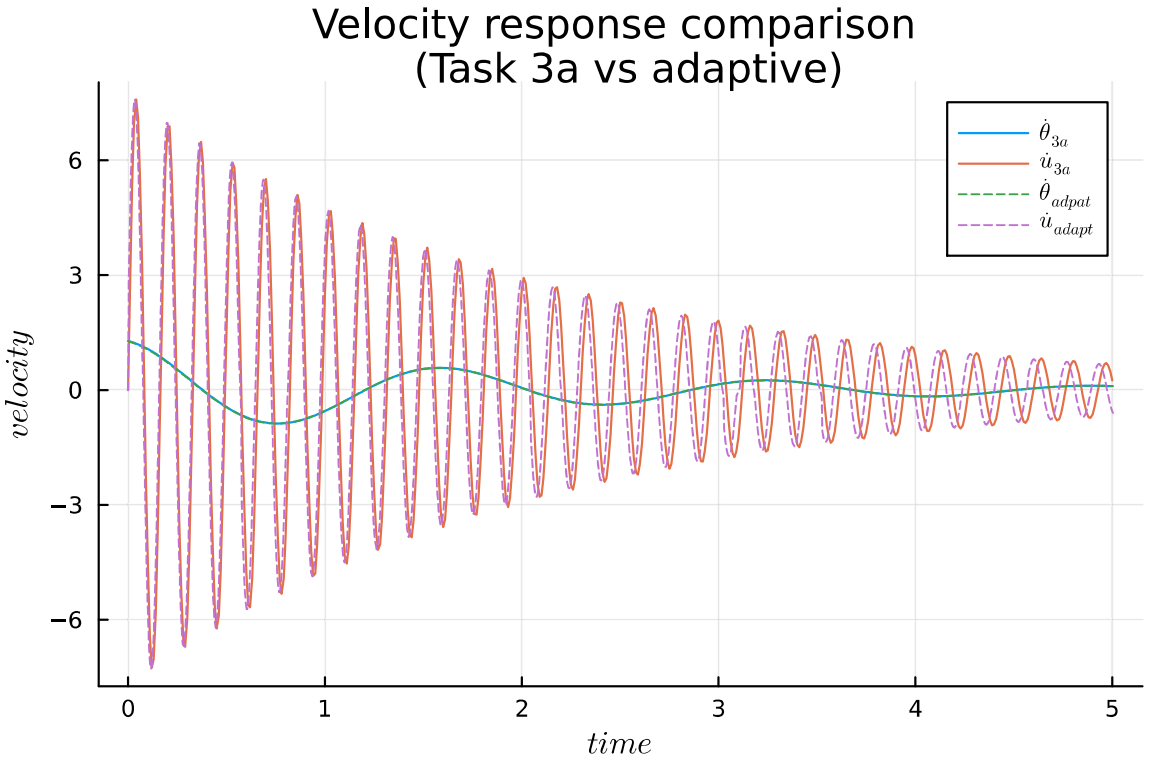



```
begin
    plot(ad_time.times,ad_err.e_cum,title="Cumulative Error
    (Adaptive)",xlabel=L"t",ylabel=L"e_{cm}",label=L"$e_{cm}$")
end
```

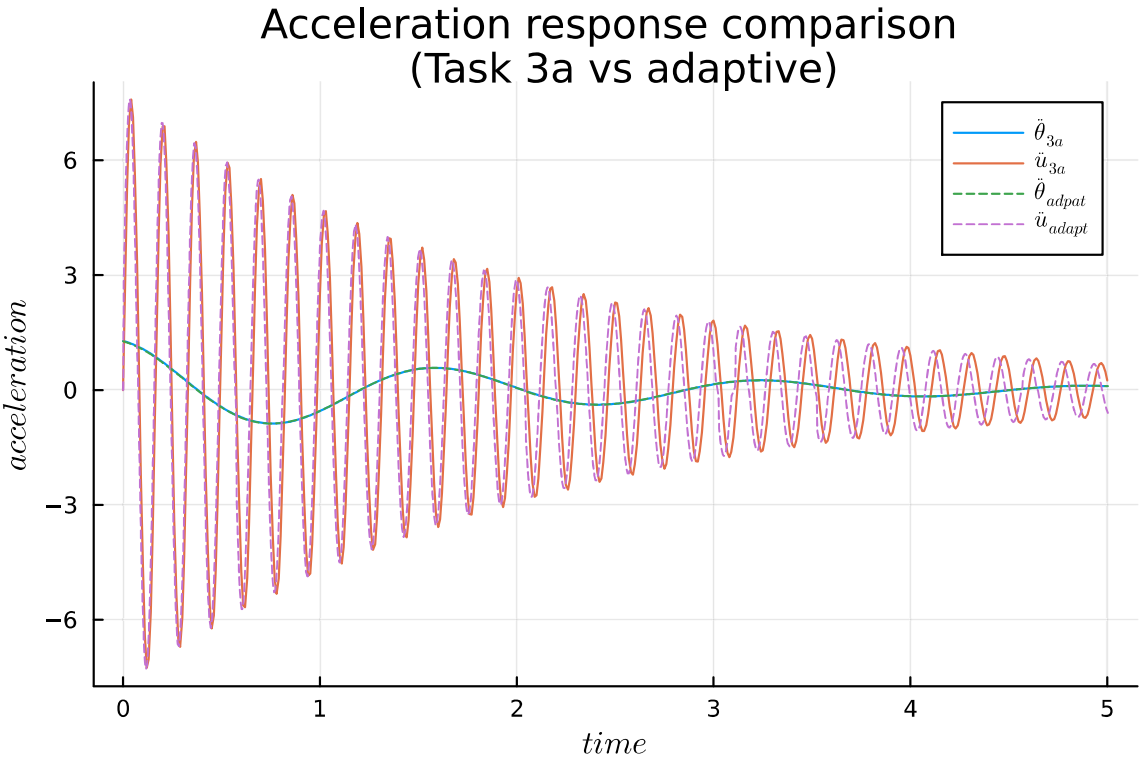
Comparing results:



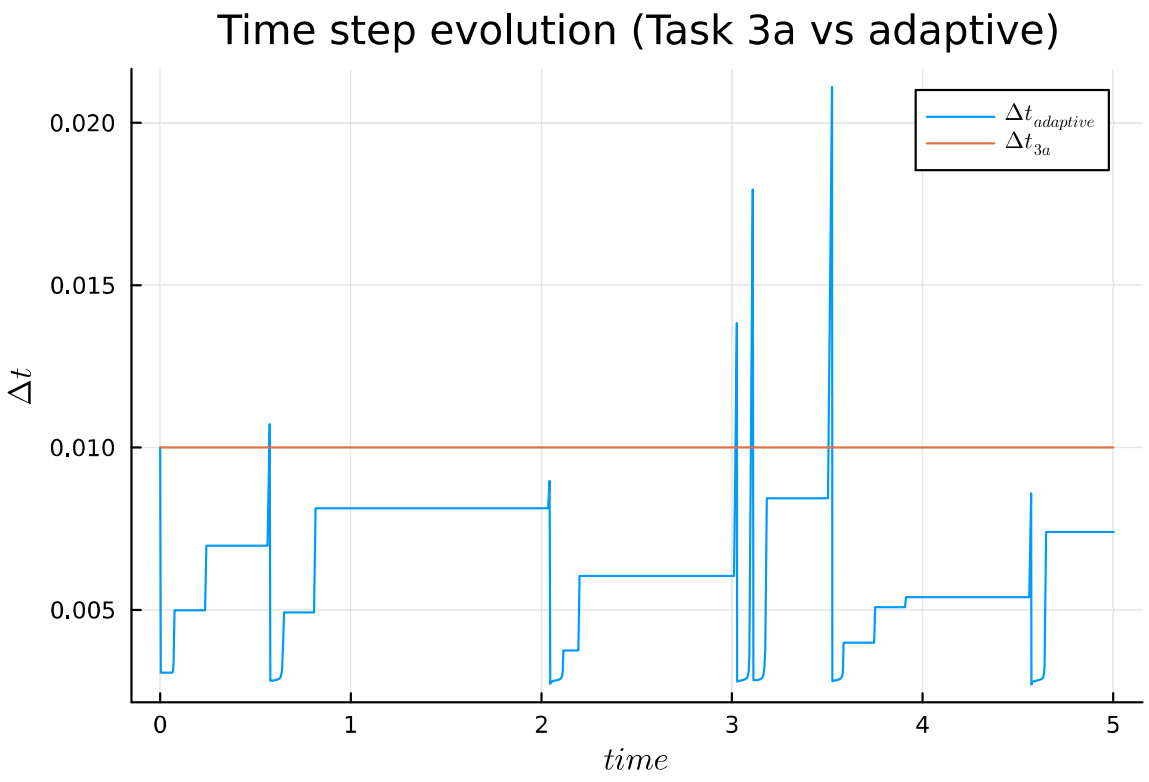

```
begin
    plot(time_a.times,resp_a.u[1,:],title="Displacement response comparison \n (Task
    3a vs adaptive)",xlabel=L"time",ylabel=L"displacement",label=L"\theta_{3a}")
    plot!(time_a.times,resp_a.u[2,:],label=L"u_{3a}")
    plot!(ad_time.times,ad_resp.u[1,:],label=L"$\theta_{adpat}$",linestyle=:dash)
    plot!(ad_time.times,ad_resp.u[2,:],label=L"u_{adapt}",linestyle=:dash)
end
```



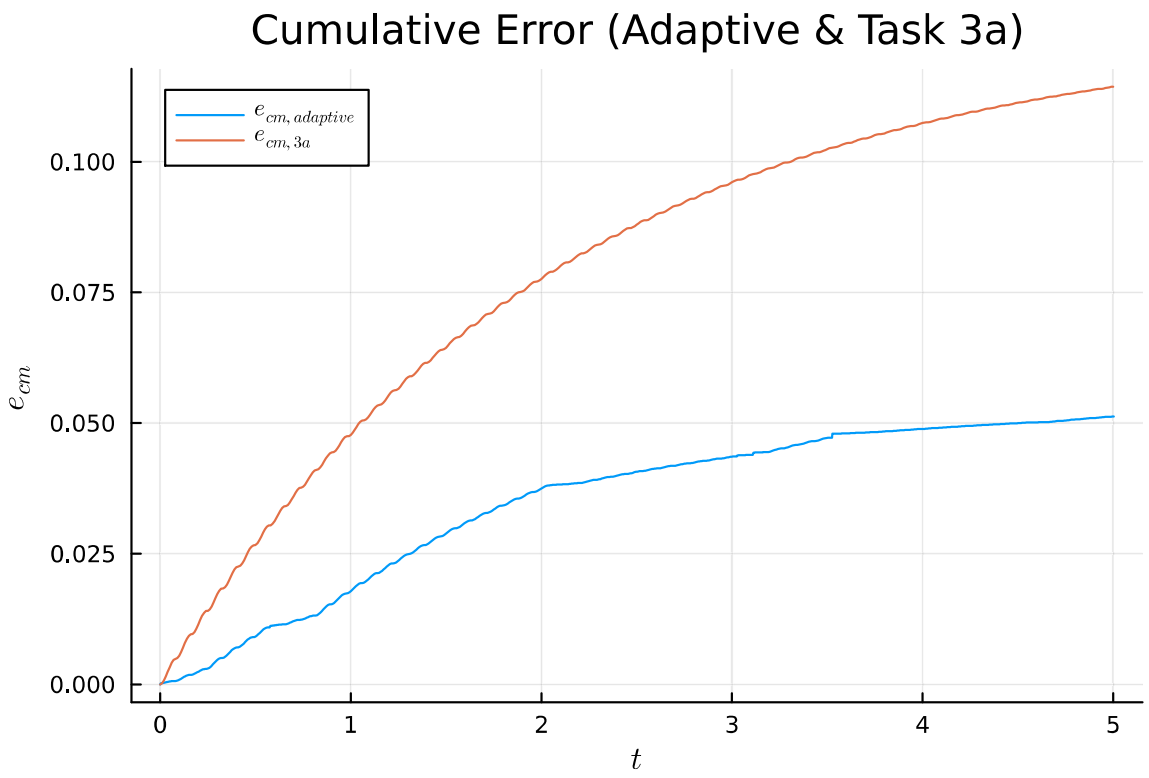

```
begin
    plot(time_a.times,resp_a.ud[1,:],title="Velocity response comparison \n (Task 3a
    vs adaptive)",xlabel=L"time",ylabel=L"velocity",label=L"\dot{\theta}_{3a}")
    plot!(time_a.times,resp_a.ud[2,:],label=L"\dot{u}_{3a}")
    plot!(ad_time.times,ad_resp.ud[1,:],label=L"\dot{\theta}_{adpat}",linestyle=:dash)
    plot!(ad_time.times,ad_resp.ud[2,:],label=L"\dot{u}_{adapt}",linestyle=:dash)
end
```




```
begin
    plot(time_a.times,resp_a.ud[1,:],title="Acceleration response comparison \n (Task 3a vs
    adaptive)",xlabel=L"time",ylabel=L"acceleration",label=L"\ddot{\theta}_{3a}")
    plot!(time_a.times,resp_a.ud[2,:],label=L"\ddot{u}_{3a}")
    plot!
    (ad_time.times,ad_resp.ud[1,:],label=L"\ddot{\theta}_{adpat}",linestyle=:dash)
    plot!(ad_time.times,ad_resp.ud[2,:],label=L"\ddot{u}_{adapt}",linestyle=:dash)
end
```




```
begin
    plot(ad_time.times,ad_time.steps,title="Time step evolution (Task 3a vs
    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_cum_err = 0.0512708, t3a_cum_error = 0.114384)
```

```
(adapt_cum_err = ad_err.e_cum[end],t3a_cum_error =err_a.e_cum[end] )
```

> **Note**
>
> From the previous result, we can observe that the cumulative error (i.e. ~0.051) in the adaptive method is less than the fixed one (~0.114) and this was anticipated.

Number of steps:

```
(adapt_steps = 871, t3a_steps = 501)
```

```
(adapt_steps = length(ad_time.steps),t3a_steps = length(time_a.steps) )
```

Part II:
Matlab Code

```
function [u, v , a  , e_abs, eta, e_cum, t, t_steps] = newmark(M,K,alpha_1,alpha_2,p_inf,u0,v0,t_f,dt)

% Rayleigh damping
C = alpha_1 * M + alpha_2 * K;

% initial acceleration
a0 = M \ (- C*v0 - K*u0);

% Chung and Hulbert (1993)
alpha_m = (2 * p_inf - 1)/(p_inf + 1);
alpha_f = p_inf / (p_inf + 1);
beta = 0.25 * (1 - alpha_m + alpha_f)^2;
gamma = 0.5 - alpha_m + alpha_f;


% time stepping
% t_f : final time
% dt : time step
% t : time vector
% t_steps : vector of time steps (for plotting)
t = 0:dt:t_f;
N = length(t);
t_steps = dt * ones(1,N);

% arrays for the response
u = zeros(2,N);
v = zeros(2,N);
a = zeros(2,N);

% initial conditions
u(:,1) = u0;
v(:,1) = v0;
a(:,1) = a0;

% define the container for the errors
e_abs = zeros(1,N); %absolute error
eta = zeros(1,N); %relative error
e_cum = zeros(1,N); %cumulative error


% effective stiffness matrix (slides pg. 63)
K_eff = M * ((1-alpha_m)/(beta*dt^2)) + C * (gamma *(1-alpha_f)/(beta*dt)) + K *(1 - alpha_f);

for i = 1:length(t)-1
    r_eff = -K * alpha_f * u(:,i) ...
          + C * ((gamma*(1-alpha_f)/(beta * dt))* u(:,i) + ((gamma - gamma * alpha_f - beta)/(beta))*v(:,i) + (((gamma-2*beta)*(1 - alpha_f))/(2*beta))*dt*a(:,i)) ...
          + M * (((1-alpha_m)/(beta * dt^2)) * u(:,i) + ((1-alpha_m)/(beta*dt)) * v(:,i) + ((1-alpha_m-2*beta)/(2*beta)) * a(:,i)) ;
```

```
    % solve for u at the next time step
    u(:,i+1) = K_eff\r_eff;

    % update v and a -> slides pg. 60
    v(:,i+1) = (gamma/(beta * dt))*(u(:,i+1) - u(:,i)) - ((gamma - beta)/beta)*v(:,i) ↙
((gamma-2*beta)/(2*beta))*dt*a(:,i);
    a(:,i+1) = (1/(beta*dt^2))*(u(:,i+1) - u(:,i)) - (1/(beta*dt)) * v(:,i) - ((1-↙
2*beta)/(2*beta))*a(:,i);

    %calculate our errors
    e_abs(i+1) = norm(((6 * beta - 1)/6) * (a(:,i+1) - a(:,i))*dt^2);
    eta(i+1) = (e_abs(i+1))/norm(u(:,i+1) - u(:,i));
    e_cum(i+1) = sum(e_abs);
end


end
```

```
clc;
clear;

fprintf('Homework 2: Generalized-Alpha method\n');
fprintf('Name: Abdelrahman Fathy Abdelhaleem Aly Abdelrahman \n');
fprintf('Matr.-Nr.: 108023251500 \n \n');
fprintf('Start Task 3a (and Task 4): \n');

% Data Inputs
L = 1.0 ;
pA = 1.0 ;
EA = 500;
g = 9.8;
dt = 0.01;
t_f = 5;

% Linear Momentum
M = [(pA*L^3)/3 0; 0 (pA*L)/3];
fprintf('Mass Matrix: \n');
disp(M);
K = [(pA*g*L^2)/2 0; 0 EA/L];
fprintf('Stiffness Matrix: \n');
disp(K);

% Initial Conditions
u0 = [0; -L/5];
fprintf('Initial Displacement: [%f, %f] \n',u0(1),u0(2));
v0 = [sqrt(g/6*L); 0];
fprintf('Initial Velocity: [%f, %f] \n',v0(1),v0(2));

% Task 3a:
alpha_1 = 1;
alpha_2 = 0;
p_inf = 1;

[u, v , a  , e_abs, eta, e_cum, t, t_steps] = newmark(M,K,alpha_1,alpha_2,p_inf,u0,v0,↙
t_f,dt);



% Create a figure for each plot and save it
% Define a directory to save the plots
outputDir = 'task3a_plots';
if ~exist(outputDir, 'dir')
    mkdir(outputDir);
end

% Plot u
figure;
plot(t, u(1, :), 'r', t, u(2, :), 'b');
xlabel('Time');
```

```
ylabel('u');
title('u vs Time (3a)');
legend('\theta', 'u');
saveas(gcf, fullfile(outputDir, 'u_vs_time.png'));

% Plot v
figure;
plot(t, v(1, :), 'r', t, v(2, :), 'b');
xlabel('Time');
ylabel('v');
title('v vs Time (3a)');
legend('$\dot{\theta}$', '$\dot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'v_vs_time.png'));

% Plot a
figure;
plot(t, a(1, :), 'r', t, a(2, :), 'b');
xlabel('Time');
ylabel('a');
title('a vs Time (3a)');
legend('$\ddot{\theta}$', '$\ddot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'a_vs_time.png'));

% Plot e_abs
figure;
plot(t, e_abs, 'r');
xlabel('Time');
ylabel('e\_abs','Interpreter','latex');
title('e\_abs vs Time (3a)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_abs_vs_time.png'));

% Plot eta
figure;
plot(t, eta, 'r');
xlabel('Time');
ylabel('$\eta$','Interpreter','latex');
title('$\eta$ vs Time (3a)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'eta_vs_time.png'));

% Plot e_cum
figure;
plot(t, e_cum, 'r');
xlabel('Time');
ylabel('e\_cum','Interpreter','latex');
title('e\_cum vs Time (3a)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_cum_vs_time.png'));

% Plot t_steps
figure;
plot(t, t_steps, 'r');
xlabel('Time');
```

```
ylabel('t\_steps','Interpreter','latex');
title('t\_steps vs Time (3a)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 't_steps_vs_time.png'));

% Close all figures
close all;

fprintf('Task 3a (and Task 4): plots is saved in [%s] folder \n', outputDir);
```

```
clc;
clear;

fprintf('Homework 2: Generalized-Alpha method\n');
fprintf('Name: Abdelrahman Fathy Abdelhaleem Aly Abdelrahman \n');
fprintf('Matr.-Nr.: 108023251500 \n \n');
fprintf('Start Task 3b (and Task 4): \n');

% Data Inputs
L = 1.0 ;
pA = 1.0 ;
EA = 500;
g = 9.8;
dt = 0.01;
t_f = 5;

% Linear Momentum
M = [(pA*L^3)/3 0; 0 (pA*L)/3];
fprintf('Mass Matrix: \n');
disp(M);
K = [(pA*g*L^2)/2 0; 0 EA/L];
fprintf('Stiffness Matrix: \n');
disp(K);

% Initial Conditions
u0 = [0; -L/5];
fprintf('Initial Displacement: [%f, %f] \n',u0(1),u0(2));
v0 = [sqrt(g/6*L); 0];
fprintf('Initial Velocity: [%f, %f] \n',v0(1),v0(2));

% Task 3a:
alpha_1 = 0;
alpha_2 = 0;
p_inf = 0.1;

fprintf('Task 3b (and Task 4): \n');
[u, v , a  , e_abs, eta, e_cum, t, t_steps] = newmark(M,K,alpha_1,alpha_2,p_inf,u0,v0,↙
t_f,dt);



% Create a figure for each plot and save it
% Define a directory to save the plots
outputDir = 'task3b_plots';
if ~exist(outputDir, 'dir')
    mkdir(outputDir);
end

% Plot u
figure;
plot(t, u(1, :), 'r', t, u(2, :), 'b');
```

```
xlabel('Time');
ylabel('u');
title('u vs Time (3b)');
legend('\theta', 'u');
saveas(gcf, fullfile(outputDir, 'u_vs_time.png'));

% Plot v
figure;
plot(t, v(1, :), 'r', t, v(2, :), 'b');
xlabel('Time');
ylabel('v');
title('v vs Time (3b)');
legend('$\dot{\theta}$', '$\dot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'v_vs_time.png'));

% Plot a
figure;
plot(t, a(1, :), 'r', t, a(2, :), 'b');
xlabel('Time');
ylabel('a');
title('a vs Time (3b)');
legend('$\ddot{\theta}$', '$\ddot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'a_vs_time.png'));

% Plot e_abs
figure;
plot(t, e_abs, 'r');
xlabel('Time');
ylabel('e\_abs','Interpreter','latex');
title('e\_abs vs Time (3b)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_abs_vs_time.png'));

% Plot eta
figure;
plot(t, eta, 'r');
xlabel('Time');
ylabel('$\eta$','Interpreter','latex');
title('$\eta$ vs Time (3b)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'eta_vs_time.png'));

% Plot e_cum
figure;
plot(t, e_cum, 'r');
xlabel('Time');
ylabel('e\_cum','Interpreter','latex');
title('e\_cum vs Time (3b)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_cum_vs_time.png'));

% Plot t_steps
figure;
plot(t, t_steps, 'r');
```

```
xlabel('Time');
ylabel('t\_steps','Interpreter','latex');
title('t\_steps vs Time (3b)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 't_steps_vs_time.png'));

% Close all figures
close all;

fprintf('Task 3b (and Task 4): plots is saved in [%s] folder \n', outputDir);
```

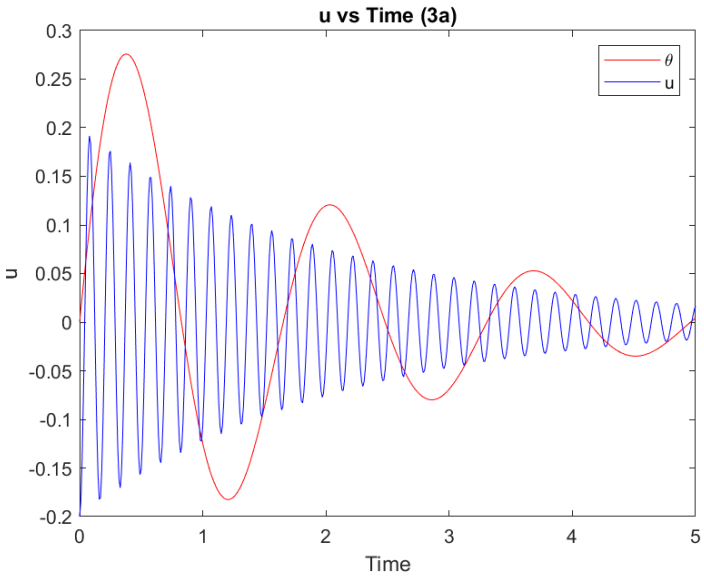
```
function [u, v , a  , e_abs, eta, e_cum, t, t_steps] = adapt_newmark(M,K,alpha_1,alpha_2,p_inf,u0,v0,t_f,dt_0,nu_1,nu_2,eta_e)

    % Rayleigh damping
    C = alpha_1 * M + alpha_2 * K;

    % initial acceleration
    a0 = M \ (- C*v0 - K*u0);

    % Chung and Hulbert (1993)
    alpha_m = (2 * p_inf - 1)/(p_inf + 1);
    alpha_f = p_inf / (p_inf + 1);
    beta = 0.25 * (1 - alpha_m + alpha_f)^2;
    gamma = 0.5 - alpha_m + alpha_f;


    % create the boundary for the adpative newmark method
    lb = nu_1 * eta_e; % lower bound
    ub = nu_2 * eta_e; % upper bound


    % time stepping
    % t_f : final time
    % dt : initial time step
    % t : time vector
    % t_steps : vector of time steps (for plotting)
    t_0 = 0;
    t_current = t_0;
    dt = dt_0; % current time step
    t = t_current;
    t_steps = dt;

    % arrays for the response with the initial conditions
    u = u0;
    v = v0;
    a = a0;


    % define the container for the errors
    e_abs = 0; %absolute error
    eta = 0;  % relative error
    e_cum = 0; %cumulative error

    i = 1; % counter for the number of iterations

    while t_current <= t_f

        % effective stiffness matrix (slides pg. 63)
        K_eff = M * ((1-alpha_m)/(beta*dt^2)) + C * (gamma *(1-alpha_f)/(beta*dt)) + K *(1 - alpha_f);
```

```
        r_eff = -K * alpha_f * u(:,i) ...
              + C * ((gamma*(1-alpha_f)/(beta * dt))* u(:,i) + ((gamma - gamma * alpha_f
- beta)/(beta))*v(:,i) + (((gamma-2*beta)*(1 - alpha_f))/(2*beta))*dt*a(:,i)) ...
              + M * (((1-alpha_m)/(beta * dt^2)) * u(:,i) + ((1-alpha_m)/(beta*dt)) * v
(:,i) + ((1-alpha_m-2*beta)/(2*beta)) * a(:,i)) ;

        % solve for u at the next time step
        %u(:,i+1) = K_eff\r_eff;
        u(:,i+1) = inv(K_eff)*r_eff;

        % update v and a -> slides pg. 60
        v(:,i+1) = (gamma/(beta * dt))*(u(:,i+1) - u(:,i)) - ((gamma - beta)/beta)*v(:,
i) - ((gamma-2*beta)/(2*beta))*dt*a(:,i);
        a(:,i+1) = (1/(beta*dt^2))*(u(:,i+1) - u(:,i)) - (1/(beta*dt)) * v(:,i) - ((1-
2*beta)/(2*beta))*a(:,i);

        %calculate our errors
        e_abs(i+1) = norm(((6 * beta - 1)/6) * (a(:,i+1) - a(:,i))*dt^2);
        eta(i+1) = (e_abs(i+1))/norm(u(:,i+1) - u(:,i));
        e_cum(i+1) = sum(e_abs);

        % check if the error is within the bounds
        if (eta(i+1) > ub || eta(i+1) < lb)
            dt = dt * sqrt(eta_e/eta(i+1));
        end
        t_current = t_current + dt;
        t(i+1) = t_current;
        t_steps(i+1) = dt;
        i = i + 1;
    end


end
```

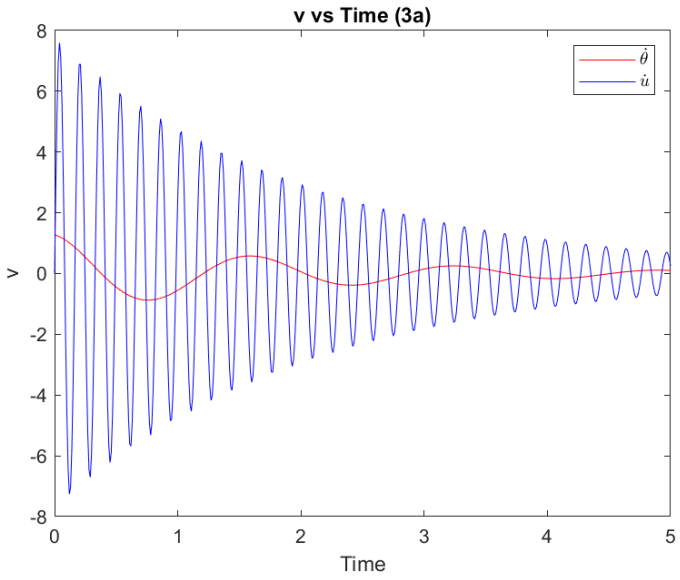
```
clc;
clear;

fprintf('Homework 2: Generalized-Alpha method\n');
fprintf('Name: Abdelrahman Fathy Abdelhaleem Aly Abdelrahman \n');
fprintf('Matr.-Nr.: 108023251500 \n \n');
fprintf('Start Task Adaptive: \n');

% Data Inputs
L = 1.0 ;
pA = 1.0 ;
EA = 500;
g = 9.8;
dt = 0.01;
t_f = 5;

% Linear Momentum
M = [(pA*L^3)/3 0; 0 (pA*L)/3];
fprintf('Mass Matrix: \n');
disp(M);
K = [(pA*g*L^2)/2 0; 0 EA/L];
fprintf('Stiffness Matrix: \n');
disp(K);

% Initial Conditions
u0 = [0; -L/5];
fprintf('Initial Displacement: [%f, %f] \n',u0(1),u0(2));
v0 = [sqrt(g/6*L); 0];
fprintf('Initial Velocity: [%f, %f] \n',v0(1),v0(2));

% Task 3a:
alpha_1 = 1;
alpha_2 = 0;
p_inf = 1;

[u, v , a  , e_abs, eta, e_cum, t, t_steps] = adapt_newmark(M,K,alpha_1,alpha_2,p_inf,↙
u0,v0,t_f,dt,1,10,0.001);



% Create a figure for each plot and save it
% Define a directory to save the plots
outputDir = 'adaptive_plots';
if ~exist(outputDir, 'dir')
    mkdir(outputDir);
end

% Plot u
figure;
plot(t, u(1, :), 'r', t, u(2, :), 'b');
xlabel('Time');
```

```
ylabel('u');
title('u vs Time (adaptive)');
legend('\theta', 'u');
saveas(gcf, fullfile(outputDir, 'u_vs_time.png'));

% Plot v
figure;
plot(t, v(1, :), 'r', t, v(2, :), 'b');
xlabel('Time');
ylabel('v');
title('v vs Time (adaptive)');
legend('$\dot{\theta}$', '$\dot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'v_vs_time.png'));

% Plot a
figure;
plot(t, a(1, :), 'r', t, a(2, :), 'b');
xlabel('Time');
ylabel('a');
title('a vs Time (adaptive)');
legend('$\ddot{\theta}$', '$\ddot{u}$','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'a_vs_time.png'));

% Plot e_abs
figure;
plot(t, e_abs, 'r');
xlabel('Time');
ylabel('e\_abs','Interpreter','latex');
title('e\_abs vs Time (adaptive)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_abs_vs_time.png'));

% Plot eta
figure;
plot(t, eta, 'r');
xlabel('Time');
ylabel('$\eta$','Interpreter','latex');
title('$\eta$ vs Time (adaptive)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'eta_vs_time.png'));

% Plot e_cum
figure;
plot(t, e_cum, 'r');
xlabel('Time');
ylabel('e\_cum','Interpreter','latex');
title('e\_cum vs Time (adaptive)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 'e_cum_vs_time.png'));

% Plot t_steps
figure;
plot(t, t_steps, 'r');
xlabel('Time');
```

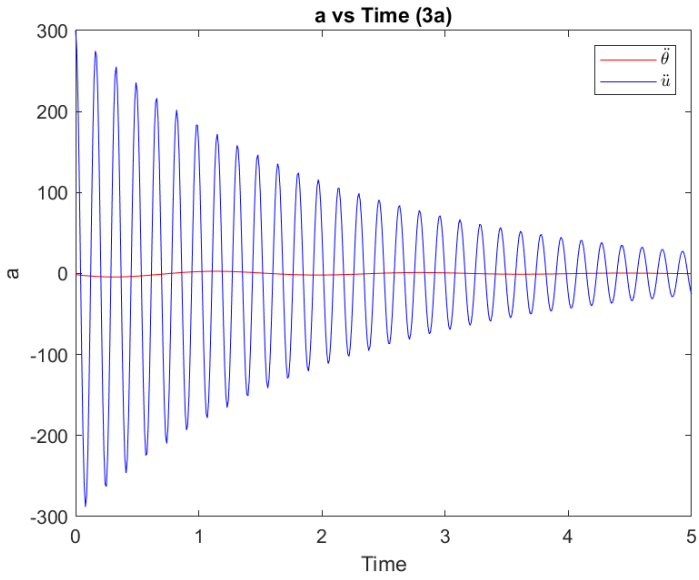
```
ylabel('t\_steps','Interpreter','latex');
title('t\_steps vs Time (adaptive)','Interpreter','latex');
saveas(gcf, fullfile(outputDir, 't_steps_vs_time.png'));

% Close all figures
close all;

fprintf('Task Adaptive: plots is saved in [%s] folder \n', outputDir);
```



(a) displacement through time

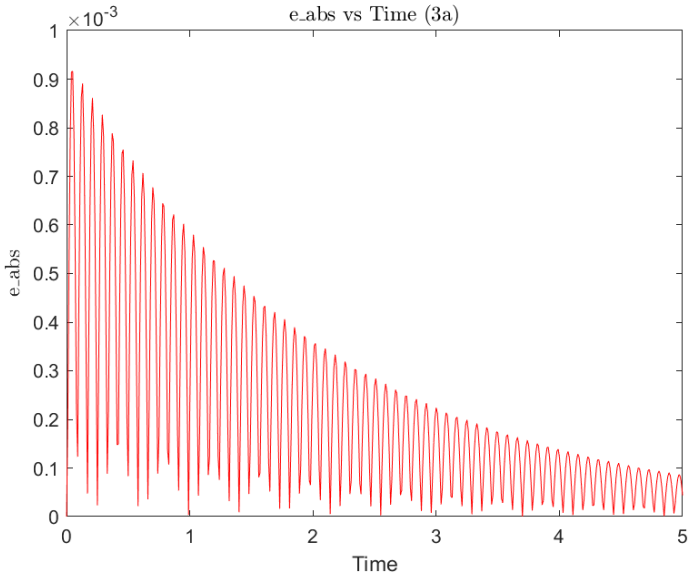


(b) velocity through time

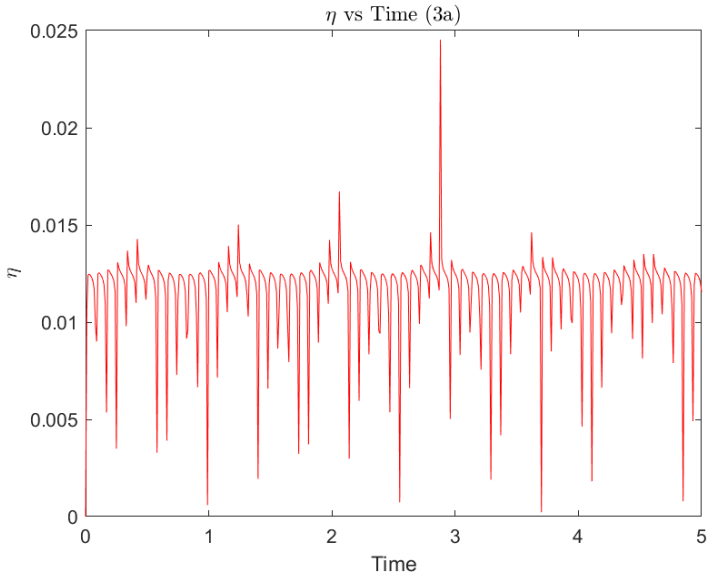


(c) acceleration through time

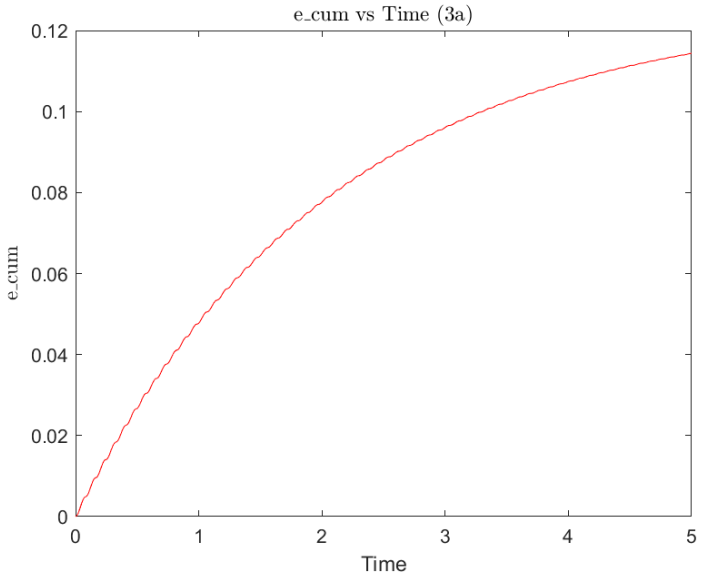
Figure 1: Rsponse of the system for Task 3a



(a) absolute error through time

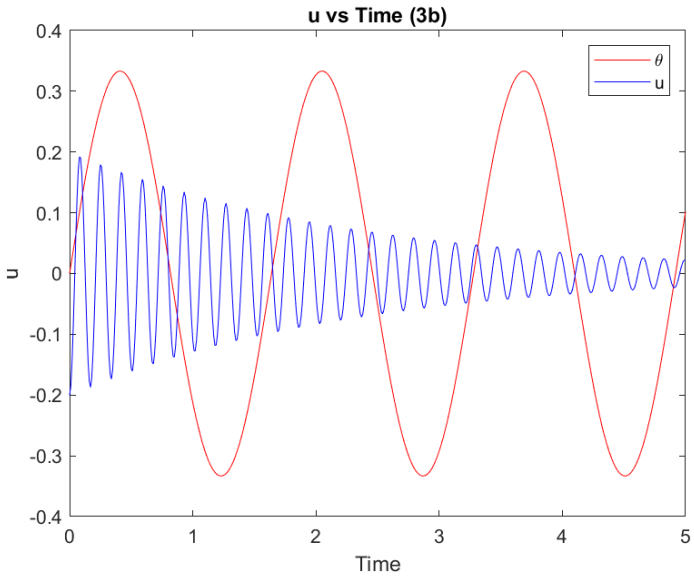


(b) relative error through time

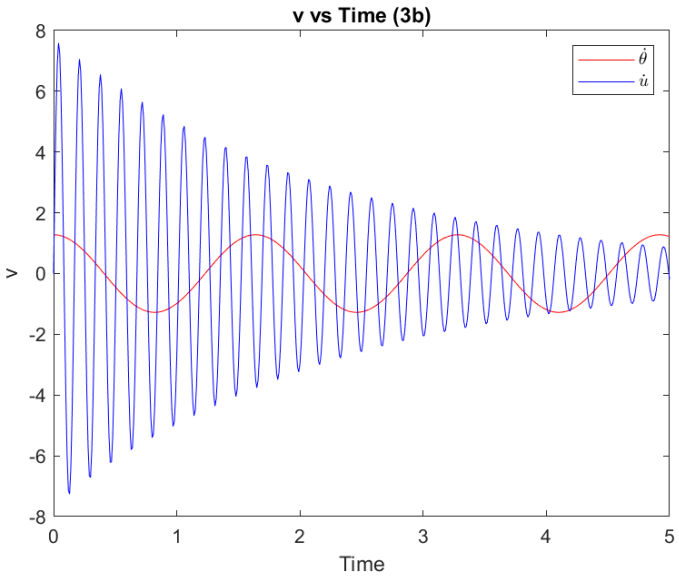


(c) cumulative error through time

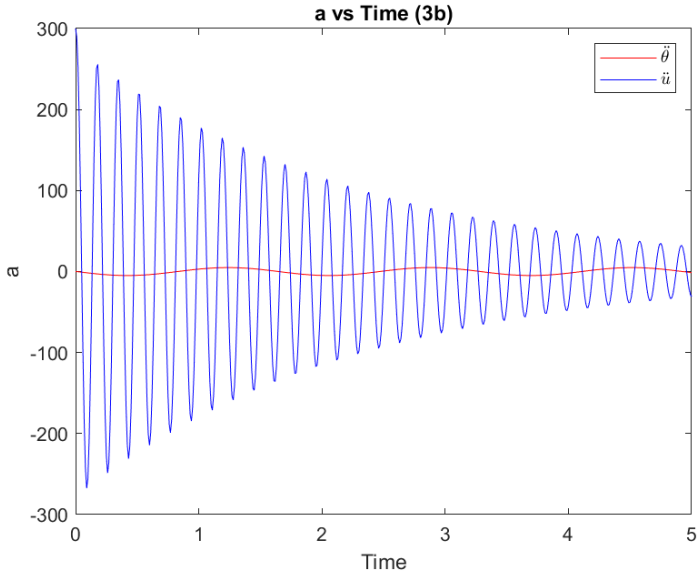
Figure 2: errors in the system for Task 3a



(a) displacement through time

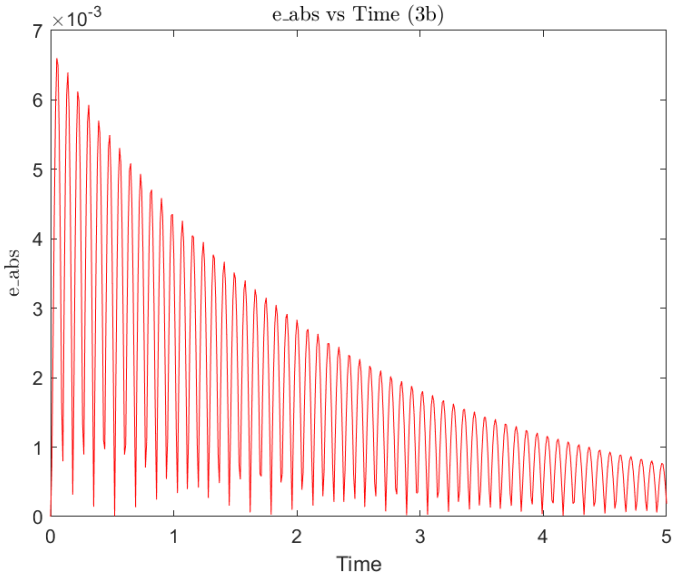


(b) velocity through time

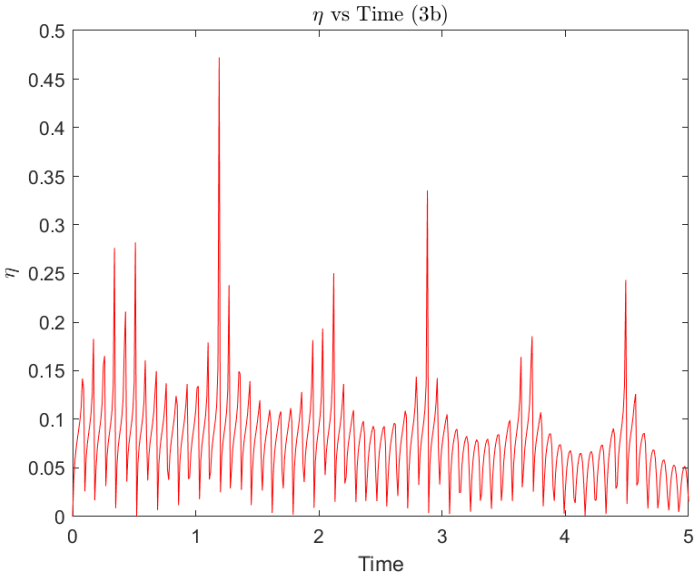


(c) acceleration through time

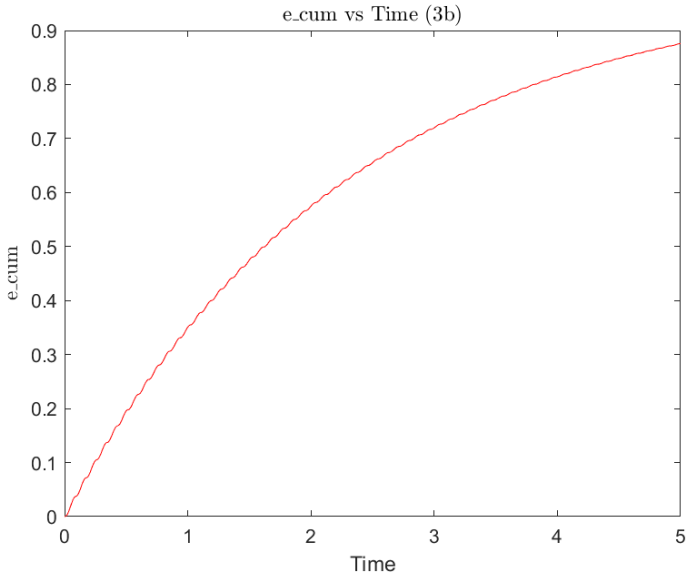
Figure 3: Rsponse of the system for Task 3b



(a) absolute error through time

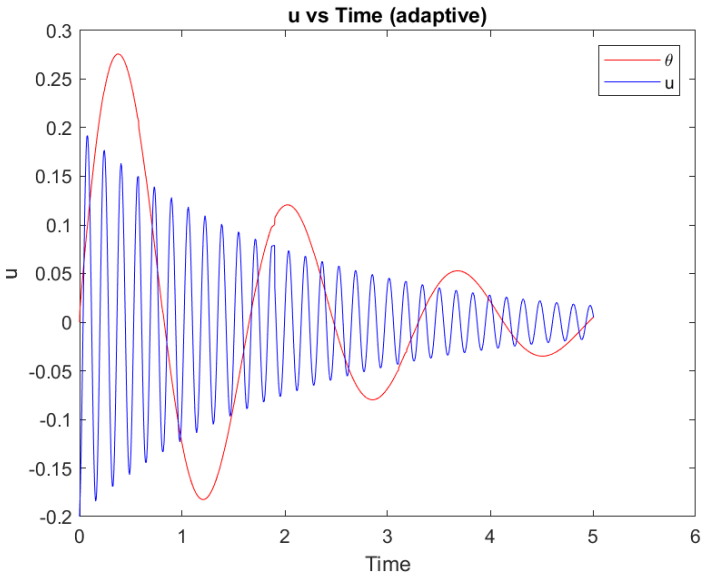


(b) relative error through time

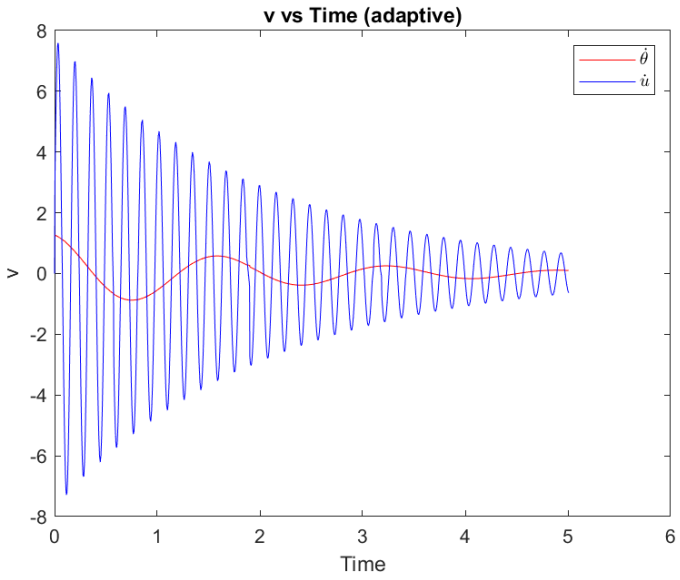


(c) cumulative error through time

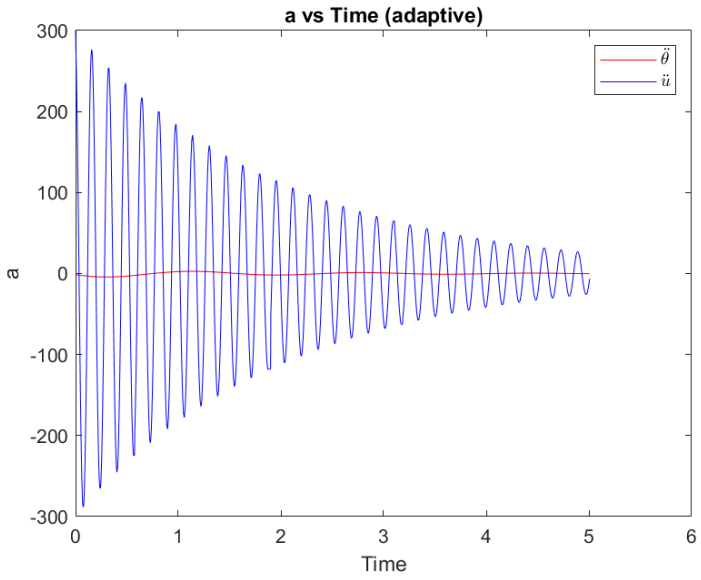
Figure 4: errors in the system for Task 3b



(a) displacement through time

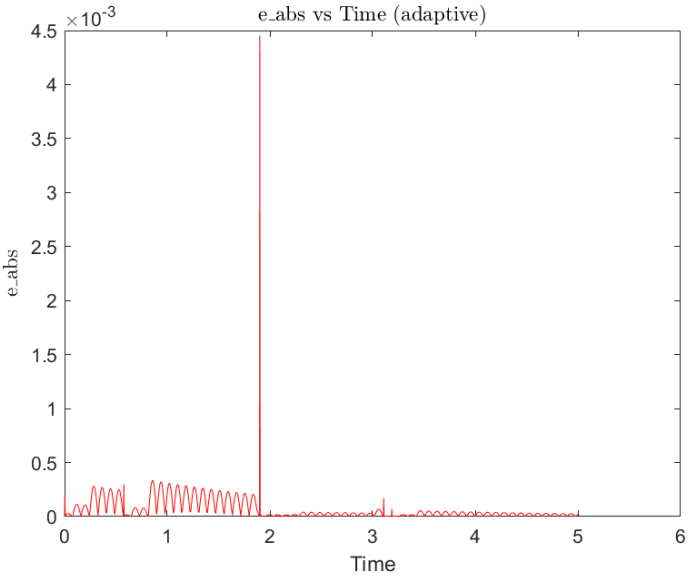


(b) velocity through time

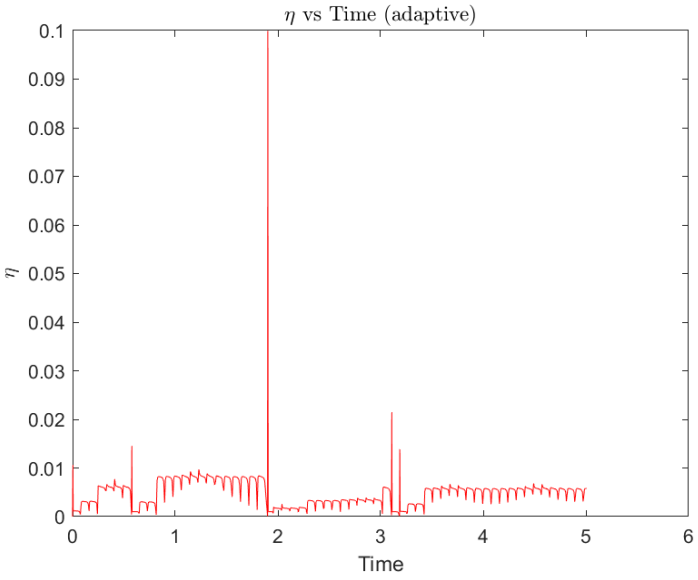


(c) acceleration through time

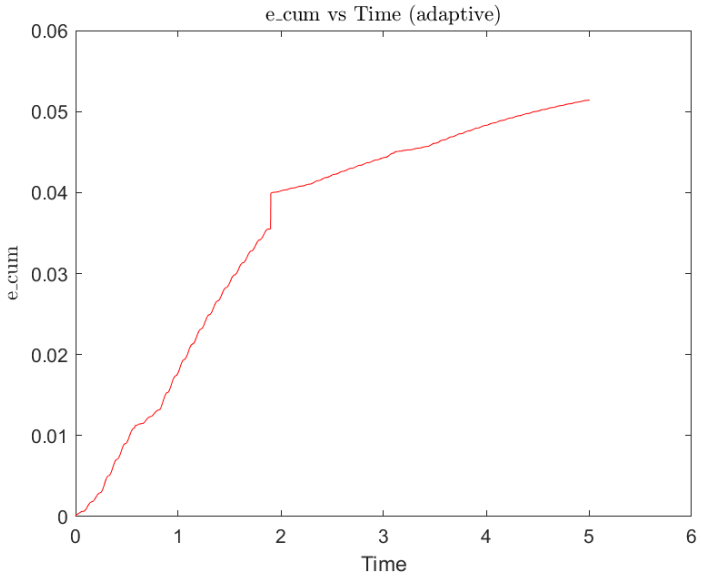
Figure 5: Rsponse of the system for Adaptive Newmark



(a) absolute error through time



(b) relative error through time



(c) cumulative error through time

Figure 6: errors in the system for Adaptive Newmark