

# Homework 2: Generalized-Alpha method

Name: Abdelrahman Fathy Abdelhaleem Aly Abdelrahman

Matr.-Nr.: 108023251500

```
1 begin
2     using LinearAlgebra
3     using Plots
4     using LaTeXStrings
5 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  $\mathbf{M}$  &  $\mathbf{K}$  matrices, as follows:

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

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

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

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

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

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

with,

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

Note

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

## 1.1. Data inputs:

$$\begin{aligned}\Delta t &= 0.01 \text{ s} \\ L &= 1.0 \text{ m} \\ \rho A &= 1.0 \text{ kg/m} \\ EA &= 500.0 \text{ N} \\ g &= 9.8 \text{ m/s}^2\end{aligned}$$

9.8

```
1 begin
2     # Data inputs
3     L = 1.0
4     ρA = 1.0
5     EA = 500
6     g = 9.8
7 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]
)
```

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

$(u_0 = [0.0, -0.2], v_0 = [1.27802, 0.0])$

```
1 begin
2     # Initial Conditions
3     u₀ = [0 ; -L/5] # initial displacement
4     v₀ = [sqrt(g/6L) ; 0] # initial velocity
5     (;u₀,v₀)
6 end
```

2×2 Matrix{Float64}:

```
0.333333 0.0
0.0      0.333333
```

```
1 begin
2   # Calculate Rayleigh damping
3   α_1 = 1.0
4   α_2 = 0.0
5   C = α_1*M + α_2 * K
6 end
```

## 1.2. Semi-discrete equation of motion:

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

when plugging in *Rayleigh damping* (i.e.  $\mathbf{C}$ ) by  $α_1 = 1.0, α_2 = 0.0$  will yield:

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

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

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

## Task 2:

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

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

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

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

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

$$α_m = \frac{2ρ_∞ - 1}{ρ_∞ + 1}, \quad α_f = \frac{ρ_∞}{ρ_∞ + 1}, \quad β = \frac{1}{4}[1 - α_m + α_f]^2, \quad γ = \frac{1}{2} - α_m + α_f$$

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

$$\mathbf{K}_{eff} = \mathbf{M} \frac{1 - α_m}{βΔt^2} + \mathbf{C} \frac{γ(1 - α_f)}{βΔt} + \mathbf{K}(1 - α_f) \rightarrow (\text{slides pg. 63})$$

2.3. Loop over time from  $t_0 = 0$  upto  $t_{final}$  with time step =  $\Delta t$  and for each step  $n$  do:

2.3.1 calculate the effective right hand side from the following equation:

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

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

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

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

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

### Note

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

```
1 struct LinearMomentum
2     M::Matrix<Float64> # mass matrix
3     K::Matrix<Float64> # stiffness matrix
4     R::Vector<Float64> # load vector (i.e. right hand side)
5 end
```

```
1 struct InitialConditions
2     u_0::Vector<Float64> # initial displacement
3     v_0::Vector<Float64> # initial velocity
4 end
```

```
1 # define abstract supertype for damping that will be inherited by
2 # physical (i.e.  $\alpha_1, \alpha_2$ ) and numerical damping (i.e.  $\rho_{-\infty}$ )
3 abstract type DampingParameters end
```

```
1 struct PhysicalDamping <: DampingParameters
2     alpha_1::Float64
3     alpha_2::Float64
4 end
```

```
1 struct NumericalDamping <: DampingParameters
2     ρ_∞::Float64
3 end
```

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

### Note

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

```
1 struct DynamicResponse
2     u::Matrix{Float64} # displacement
3     ud::Matrix{Float64} # velocity
4     udd::Matrix{Float64} # acceleration
5 end
```

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

```
1 struct TimeEvolution
2     times::Vector{Float64} # current time vector.
3     steps::Vector{Float64} # Δt vector
4 end
```

```

generalized_alpha (generic function with 1 method)
1 function generalized_alpha(moment::LinearMomentum,ic::InitialConditions,
2   time::RunTime, pd::PhysicalDamping,nd::NumericalDamping)
3
4   M = moment.M # mass matrix
5   K = moment.K # stiffness matrix
6
7   α₁ = pd.α₁
8   α₂ = pd.α₂
9
10  # calculate damping
11  C = α₁*M + α₂ * K
12
13  # Initial conditions
14  u_o = ic.u_o
15  ud_o = ic.v_o
16  udd_o = inv(M) * (R - C * ud_o - K * u_o)
17
18  # Chung and Hulbert (1993)
19  ρ_∞ = nd.ρ_∞
20  α_m = (2*ρ_∞ - 1)/(ρ_∞ + 1)
21  α_f = (ρ_∞)/(ρ_∞+1)
22  β = 0.25 * (1 - α_m + α_f)^2
23  γ = 0.5 - α_m + α_f
24
25  # create time interval range to loop over later
26  tf = time.tf
27  Δt = time.Δt
28  t = 0:Δt:tf
29  n = length(t)
30  steps = fill(Δt,n)
31
32  # define our response containers
33  n = length(t)
34  u = zeros(2,n)
35  ud = zeros(2,n)
36  udd = zeros(2,n)
37
38  # set initial conditions in our response containers
39  u[:,1] = u_o
40  ud[:,1] = ud_o
41  udd[:,1] = udd_o
42
43  # define the containers for errors
44  e_abs = zeros(n) # absolute error
45  η = zeros(n) # relative error
46  e_cum = zeros(n) # cumulative error
47
48  K_eff = M * ((1-α_m)/(β*Δt^2)) + C * (γ * (1 - α_f))/(β*Δt) + K * (1 - α_f)
49  for i = 1:n-1
50
51    r_eff = -K * α_f * u[:,i] +
52

```

```

53   C * (((γ * (1-α_f))/(β*Δt)) * u[:,i] + ((γ - γ*α_f - β)/(β)) *
54     ud[:,i] + (((γ-2*β)*(1-α_f))/(2*β)) * Δt*udd[:,i]) +
55   M * (((1-α_m)/(β*Δt^2))*u[:,i] + ((1-α_m)/(β*Δt))* ud[:,i] + ((1-α_m
56   - 2 *β)/(2*β)) * udd[:,i])
57
58   # solve for u.
59   u[:,i+1] = K_eff \ r_eff
60
61   # update velocity and acceleration
62   ud[:,i+1] = ((γ)/(β*Δt)) * (u[:,i+1] - u[:,i]) - ((γ - β)/(β))*ud[:,i] -
63   ((γ- 2*β)/(2*β)) * Δt * udd[:,i]
64
65   # calculate errors
66   e_abs[i+1] = norm(((6*β - 1)/(6))*(udd[:,i+1] - udd[:,i]) * Δt^2);
67   η[i+1] = e_abs[i+1] / norm(u[:,i+1] - u[:,i]);
68   e_cum[i+1] = sum(e_abs);
69 end
70
71 (response = DynamicResponse(u,ud,udd), errors = Error(e_abs,η,e_cum),time =
72 TimeEvolution(t,steps))
73
74 end

```

## Task: Solve the system:

```

(
    momentum = LinearMomentum(
        M = 2x2 Matrix{Float64}:
            0.333333 0.0
            0.0 0.333333
        K = 2x2 Matrix{Float64}:
            4.9 0.0
            0.0 500.0
        R = [0.0, 0.0]
    )
    initial_conditions = InitialConditions(
        u₀ = [0.0, -0.2]
        v₀ = [1.27802, 0.0]
    )
    runtime = RunTime(
        tf = 5.0
        Δt = 0.01
    )
)
1 begin
2   # common input parameters for both task 3a & 3b
3   momentum = LinearMomentum(M,K,R) # linear momentum parameters
4   initial_conditions = InitialConditions(u₀,v₀) # initial conditions
5   runtime = RunTime(5.0,0.01) # 0.01
6   (;momentum,initial_conditions, runtime)
7 end

```

## Task 3a:

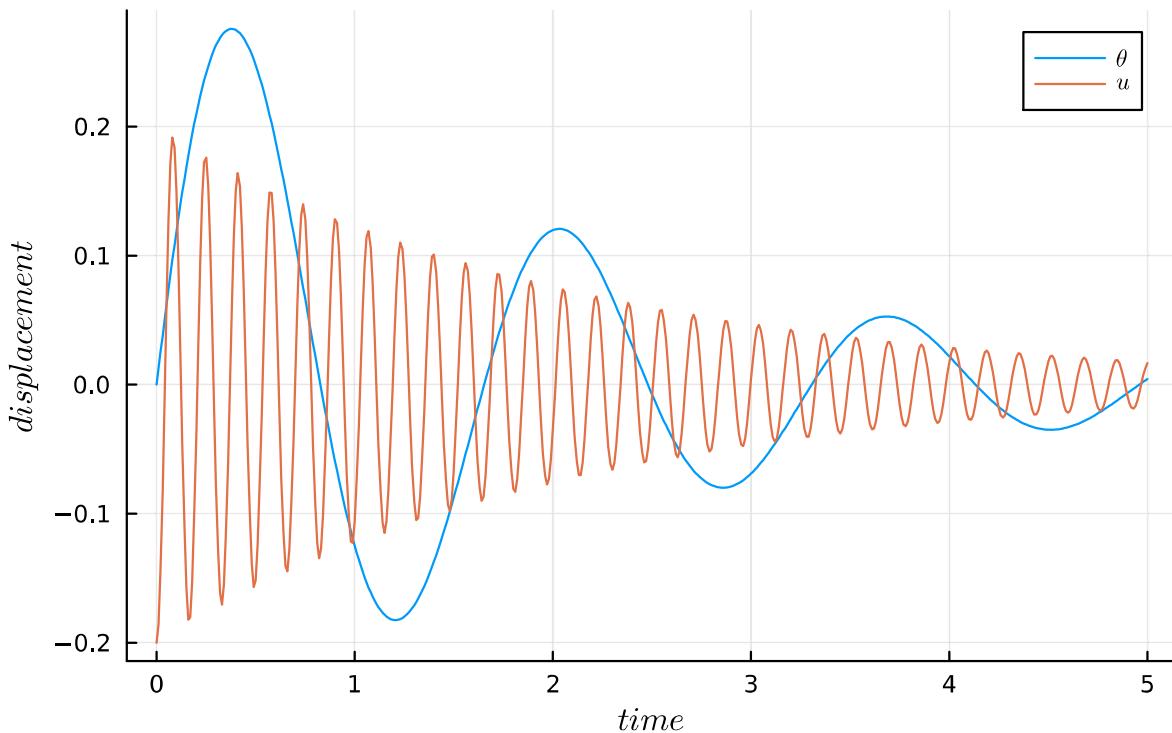
Given:  $t_{final} = 5 \text{ s}$ ,  $\alpha_1 = 1$ ,  $\alpha_2 = 0$ ,  $\rho_\infty = 1.0$

```
(  
    physical_damping_a = PhysicalDamping(  
        α₁ = 1.0  
        α₂ = 0.0  
    )  
    numerical_damping_a = NumericalDamping(  
        ρ_∞ = 1.0  
    )  
)  
  
1 begin # parameters specific for task 3a.  
2     physical_damping_a = PhysicalDamping(1.0,0.0) # (α₁ = 1.0, α₂=0.0)  
3     numerical_damping_a = NumericalDamping(1.0) # (ρ_∞ = 1.0)  
4     (;physical_damping_a,numerical_damping_a)  
5 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, (
```

◀ ▶

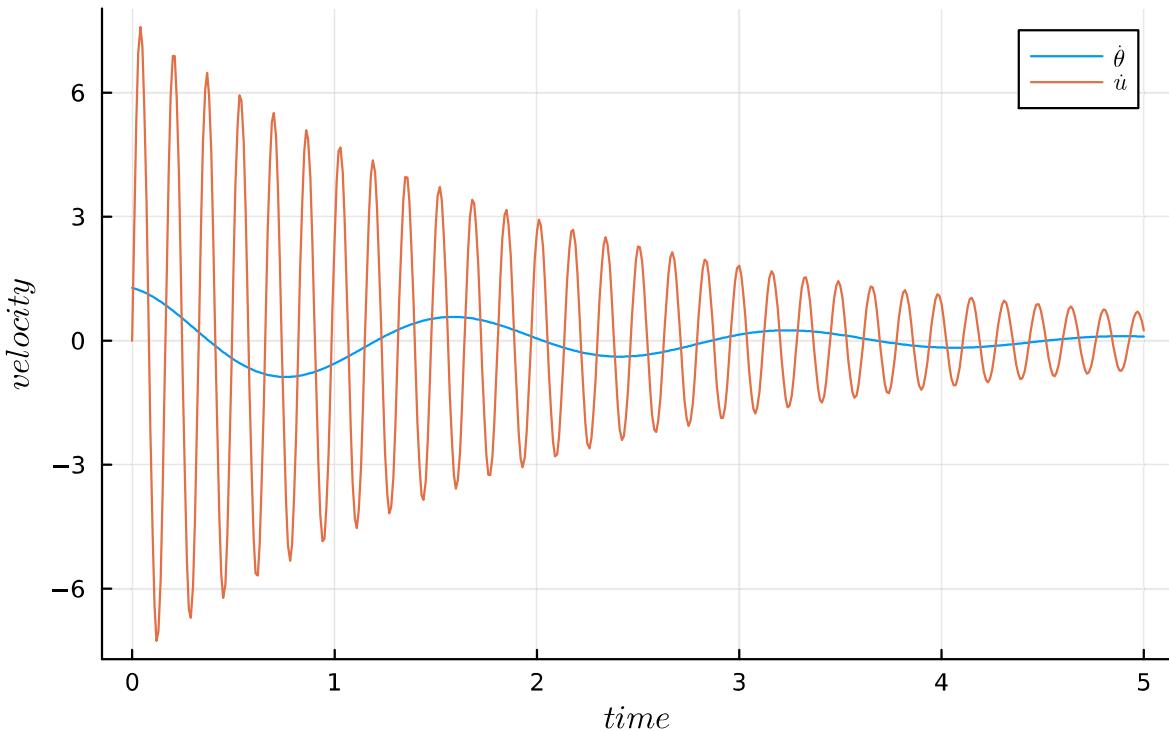
```
1 resp_a , err_a ,time_a=  
generalized_alpha(momentum,initial_coditions,runtime,physical_damping_a,numerical_damp  
ing_a)
```

## Displacement response with time (Task 3a)



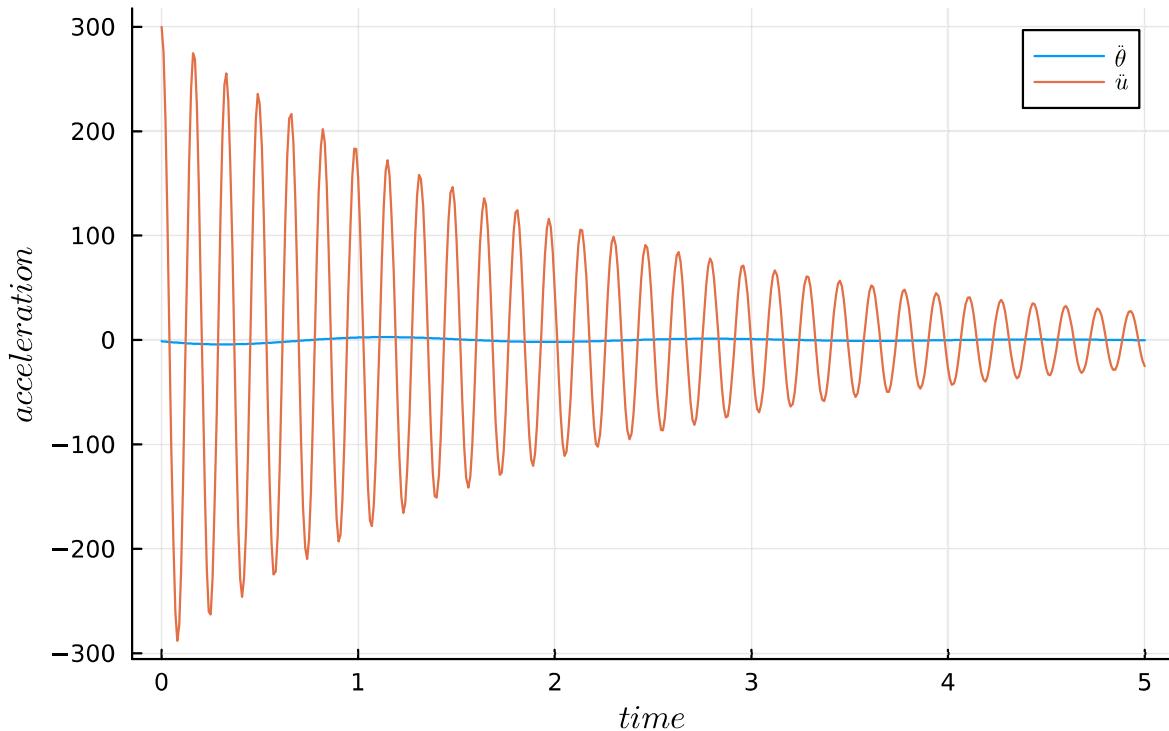
```
1 begin
2     plot(time_a.times,resp_a.u[1,:],title="Displacement response with time (Task
3     3a)", xlabel=L"time", ylabel=L"displacement", label=L"\theta")
4     plot!(time_a.times,resp_a.u[2,:],label=L"u")
5 end
```

## Velocity response with time (Task 3a)



```
1 begin
2     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}")
3     plot!(time_a.times,resp_a.ud[2,:],label=L"\dot{u}")
4 end
```

## Acceleration response with time (Task 3a)



```

1 begin
2   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}")
3   plot!(time_a.times,resp_a.udd[2,:],label=L"\ddot{u}")
4 end

```

## Task 3b:

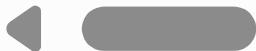
Given:  $t_{final} = 5 \text{ s}$ ,  $\alpha_1 = 0$ ,  $\alpha_2 = 0$ ,  $\rho_\infty = 0.1$

```

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

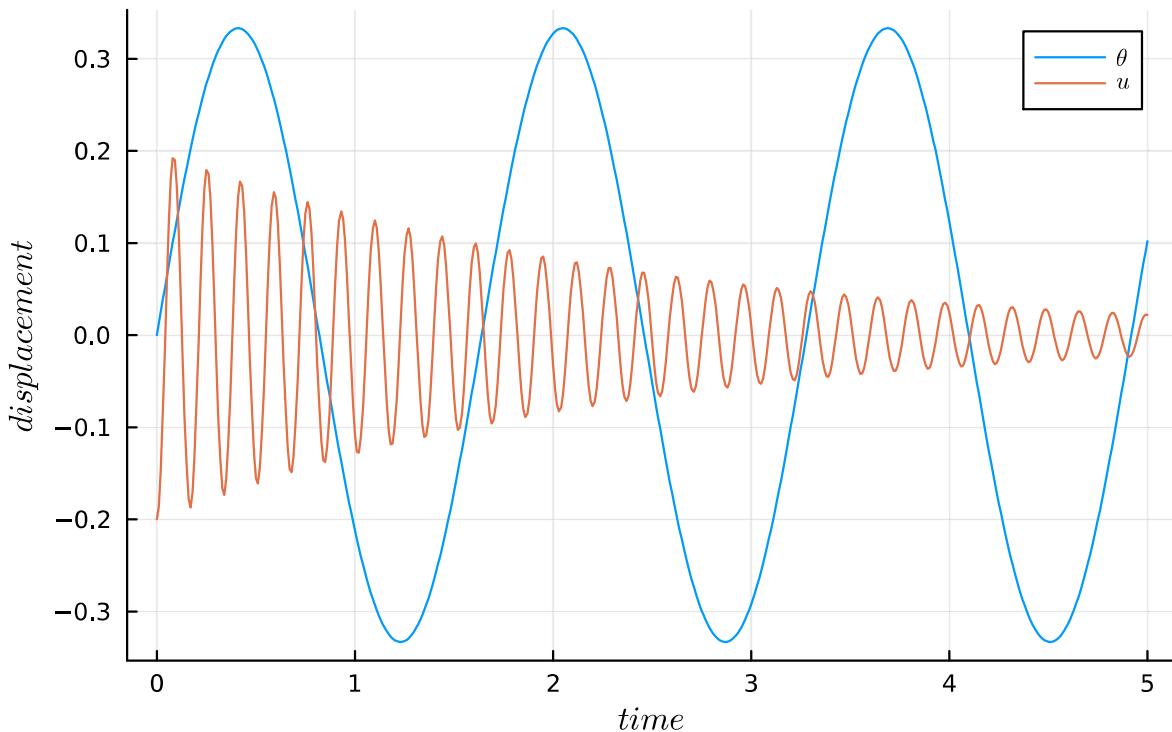
```

```
(response = DynamicResponse(2x501 Matrix{Float64}:
    0.0   0.012772   0.0255218   0.0382327   ...   0.0774721   0.085
```



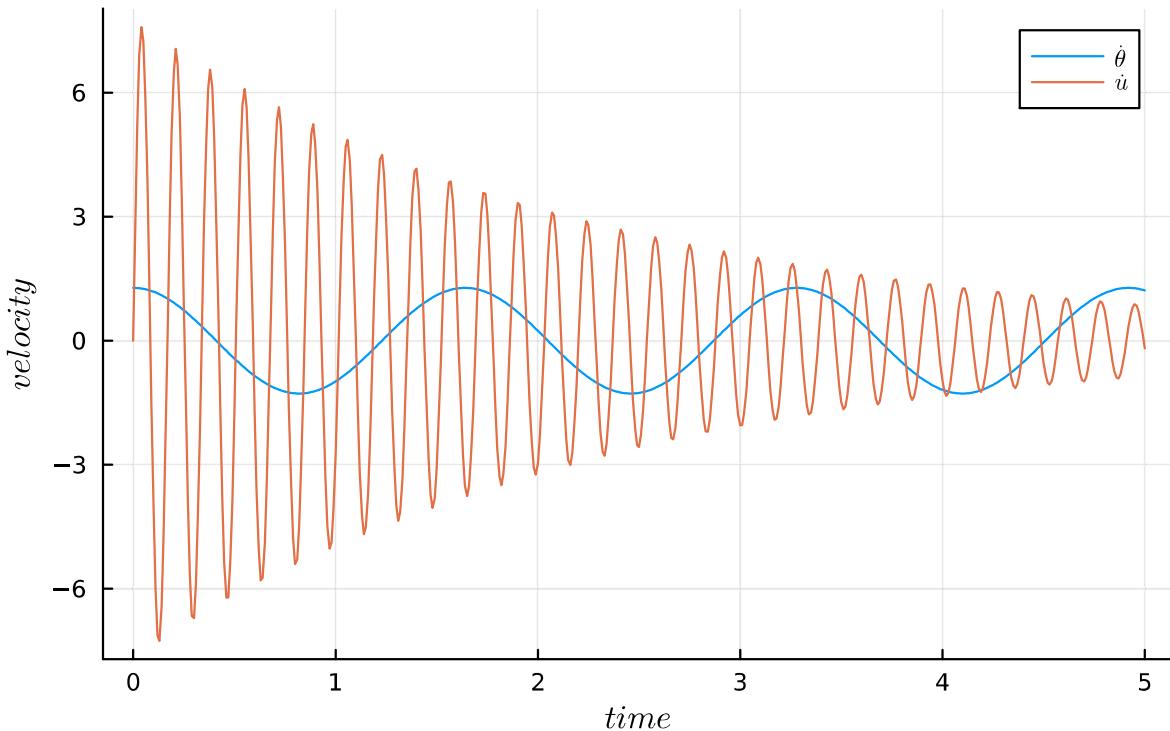
```
1 resp_b , err_b, time_b=
generalized_alpha(momentum,initial_coditions,runTime,physical_damping_b,numerical_damp
ing_b)
```

### Displacement response with time (Task 3b)



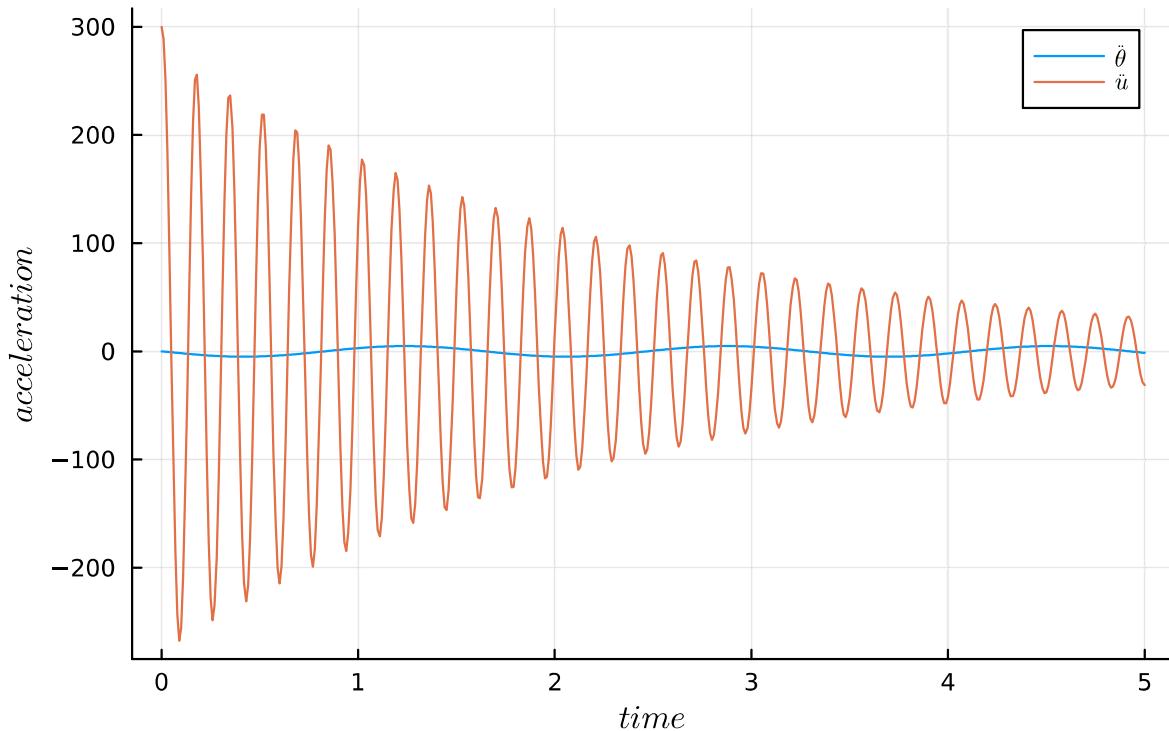
```
1 begin
2     plot(time_b.times,resp_b.u[1,:],title="Displacement response with time (Task
3b)", xlabel=L"time", ylabel=L"displacement", label=L"\theta")
3     plot!(time_b.times,resp_b.u[2,:],label=L"u")
4 end
```

## Velocity response with time (Task 3b)



```
1 begin
2   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}")
3   plot!(time_b.times,resp_b.ud[2,:],label=L"\dot{u}")
4 end
```

## Acceleration response with time (Task 3b)



```

1 begin
2   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}")
3   plot!(time_b.times,resp_b.udd[2,:],label=L"\ddot{u}")
4 end

```

### Task 3 (Explanation):

#### Observation:

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

#### Explanation:

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

However,  $\rho_\infty$  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. Finally, 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):

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

### 4.2. Relative error:

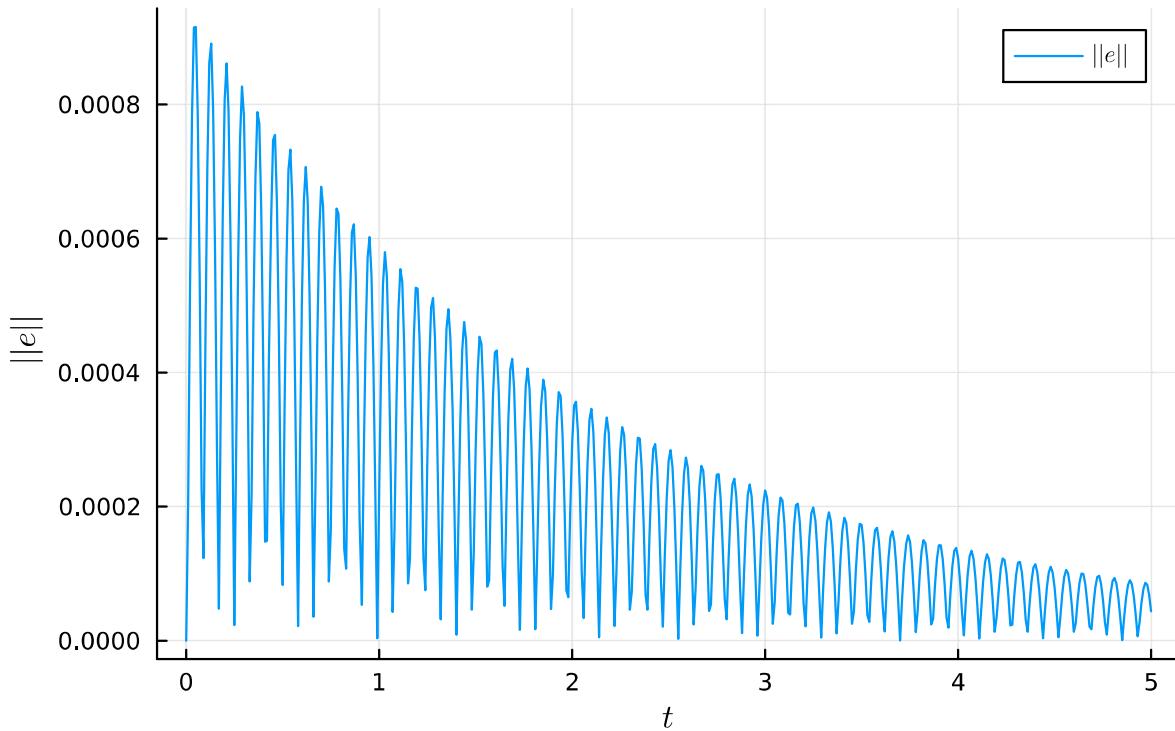
$$\eta = \frac{\|\mathbf{e}\|}{\|\mathbf{u}_{n+1} - \mathbf{u}_n\|} \rightarrow (\text{slides pg. 91})$$

### 4.3. Cummulative error:

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

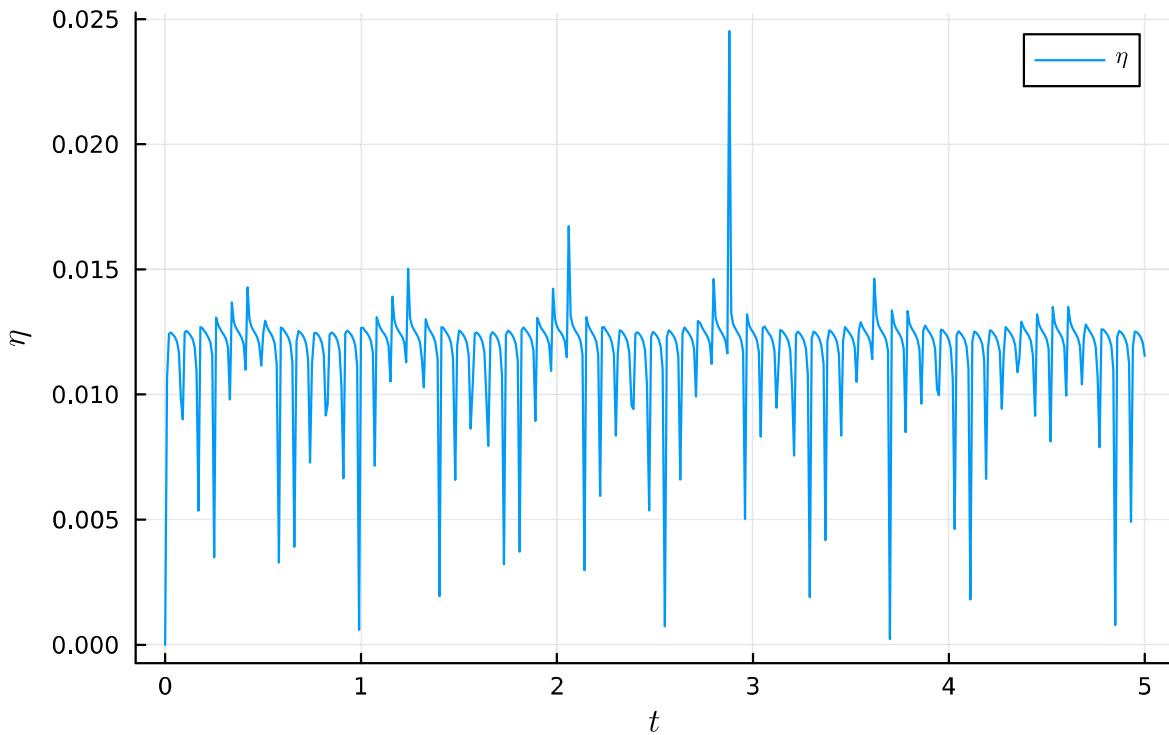
where  $n$  is the index of the current time step.

## Absolute Error (Task 3a)



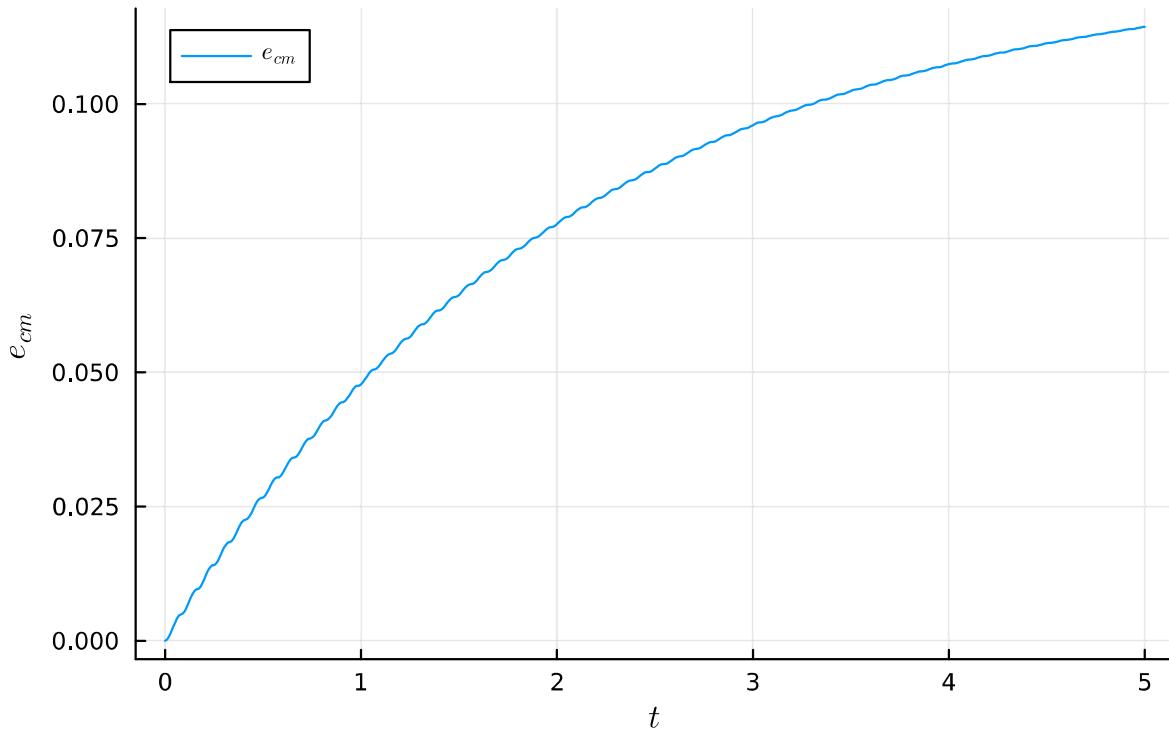
```
1 begin
2     plot(time_a.times,err_a.e_abs,title="Absolute Error (Task
3a)", xlabel=L"t", ylabel=L"\|e\|", label=L"\$\|e\|\$")
3 end
```

## Relative Error (Task 3a)



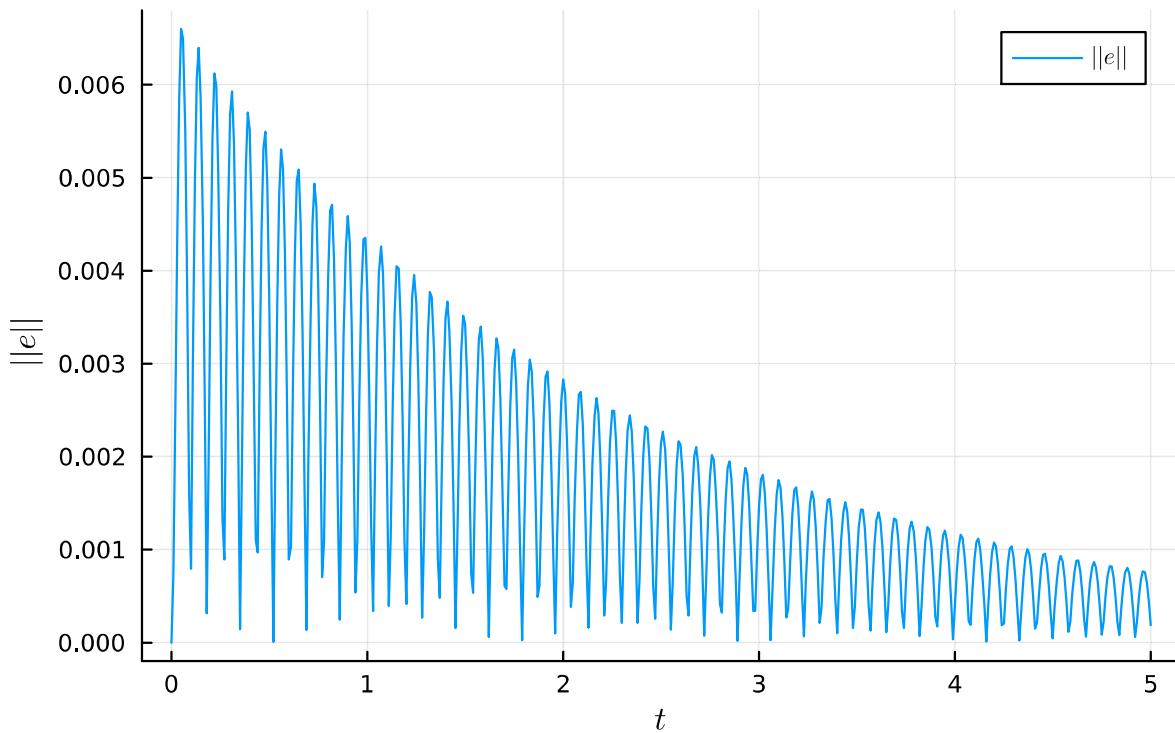
```
1 begin
2     plot(time_a.times,err_a.η,title="Relative Error (Task
3a)", xlabel=L"t", ylabel=L"η", label=L"$\eta$")
3 end
```

### Cumulative Error (Task 3a)



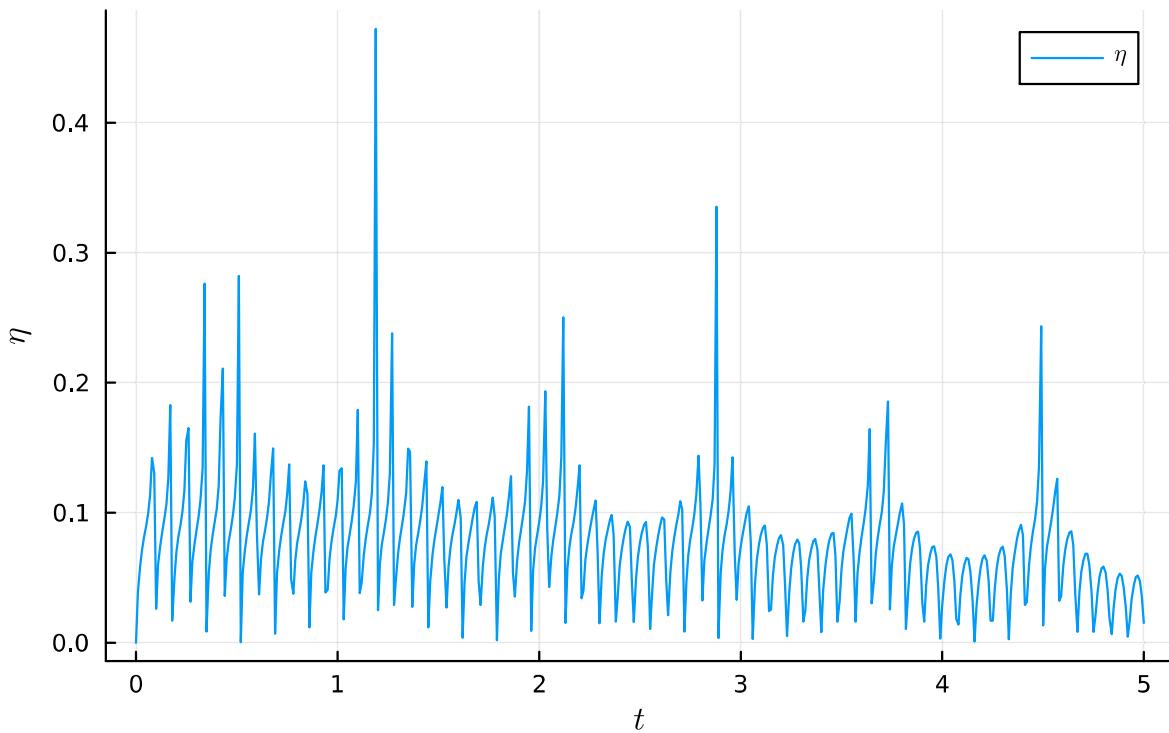
```
1 begin
2     plot(time_a.times,err_a.e_cum,title="Cumulative Error (Task
3a)", xlabel=L"t", ylabel=L"e_{cm}", label=L"$e_{cm}$")
3 end
```

## Absolute Error (Task 3b)



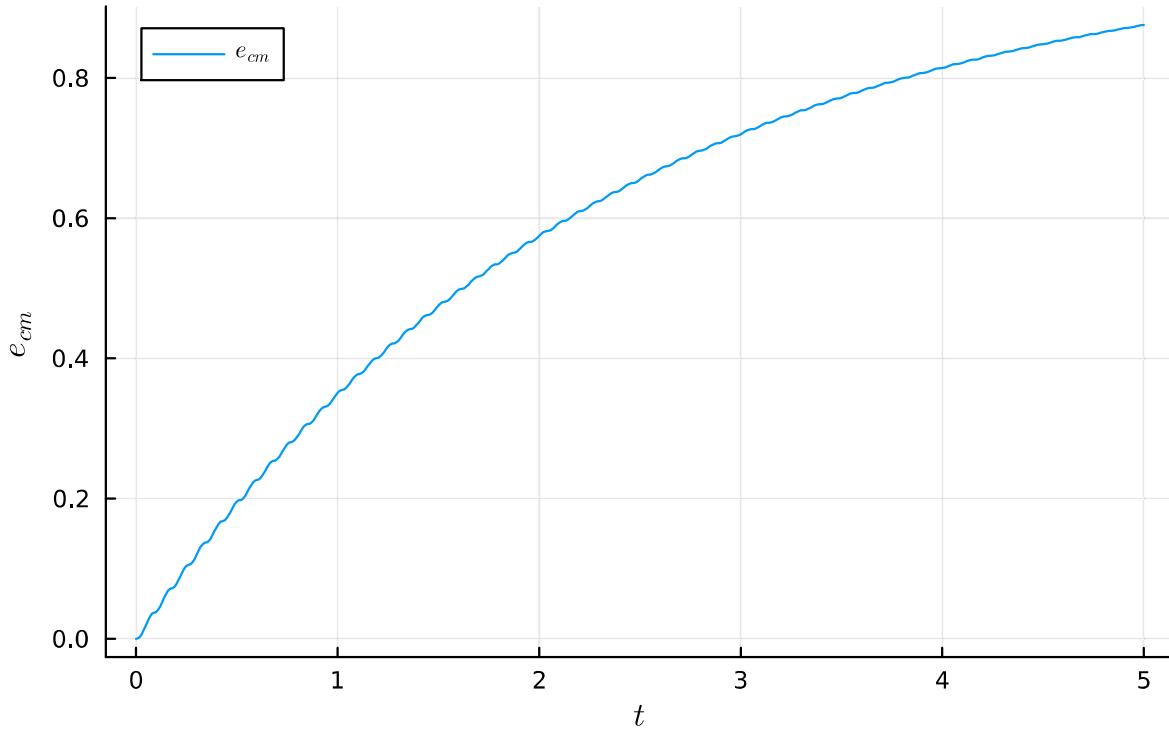
```
1 begin
2     plot(time_b.times,err_b.e_abs,title="Absolute Error (Task
3b)", xlabel=L"t", ylabel=L"\|e\|", label=L"\$\|e\|\$")
3 end
```

## Relative Error (Task 3b)



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

## Cumulative Error (Task 3b)



```

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

```

### Task 5 (Adaptive time stepping algorithm):

- 5.1. We start by setting the current time to the initial time  $t \leftarrow t_0$ , and the current time step with the initial time step  $\Delta t \leftarrow \Delta t_0$ . 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  $\Delta t$  is now changing.

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

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

- 5.2.3. Update current time ( $t$ ):

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

```
1 # this struct represents out time boundary that we need to stay in.  
2 struct AdaptiveTimeBoundary  
3     v1::Float64  
4     v2::Float64  
5     ne::Float64  
6 end
```

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

```
1 function generalized_alpha_adaptive(moment::LinearMomentum,ic::InitialConditions,
2                                     time::RunTime,pd::PhysicalDamping,nd::NumericalDamping,boundary::AdaptiveTimeBoundary
3                                     )
4
5
6     M = moment.M # mass matrix
7     K = moment.K # stiffness matrix
8
9
10    α₁ = pd.α₁
11    α₂ = pd.α₂
12
13    # calculate damping
14    C = α₁*M + α₂ * K
15
16    # Initial conditions
17    u_o = ic.u_o
18    ud_o = ic.v_o
19    udd_o = inv(M) * (R - C * ud_o - K * u_o)
20
21    # Chung and Hulbert (1993)
22    ρ_∞ = nd.ρ_∞
23    α_m = (2*ρ_∞ - 1)/(ρ_∞ + 1)
24    α_f = (ρ_∞)/(ρ_∞+1)
25    β = 0.25 * (1 - α_m + α_f)^2
26    γ = 0.5 - α_m + α_f
27
28
29
30    # define the boundary for the adaptive newmark
31    v₁ = boundary.v₁
32    v₂ = boundary.v₂
33    η_e = boundary.η_e
34    lb = v₁ * η_e # lower boundary
35    ub = v₂ * η_e # upper boundary
36
37
38    # define our response containers
39    u = zeros(2,0)
40    ud = zeros(2,0)
41    udd = zeros(2,0)
42
43    # time data
44    t₀ = 0.0 # initial time
45    t_current = t₀ # current time
46    tf = time.tf # final time we want to solve for
47    Δt₀ = time.Δt # initial time step
48    t = [t_current] # container for current time values
49    tstep = [Δt₀] # container for the evolution of time steps
50    Δt = Δt₀ # current time step
51
52
53    # set initial conditions in our response containers
54    u = hcat(u,u_o)
55    ud = hcat( ud,ud_o)
56    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)) *
68             ud[:,i] + (((γ-2*β)*(1-α_f))/(2*β)) * Δt*udd[:,i]) +
69             M * (((1-α_m)/(β*Δt^2))*u[:,i] + ((1-α_m)/(β*Δt))* ud[:,i] + ((1-α_m
70             - 2 *β)/(2*β)) * udd[:,i])
71
72     # solve for u.
73     u_n1 = K_eff \ r_eff # u_{n+1} next displacement
74     u = hcat(u, u_n1)
75
76     # update velocity and acceleration
77     ud_n1 = ((γ)/(β*Δt)) * (u[:,i+1] - u[:,i]) - ((γ - β)/(β))*ud[:,i] - ((γ-
78     2*β)/(2*β)) * Δt * udd[:,i]
79     ud = hcat(ud,ud_n1)
80
81     udd_n1 = (1/(β*Δt^2))*(u[:,i+1] - u[:,i]) - (1/(β*Δt)) * ud[:,i] - ((1-
82     2*β)/(2*β)) * udd[:,i]
83     udd = hcat(udd,udd_n1)
84
85     # calculate errors
86     e_abs_n1 = norm(((6*β - 1)/(6))*(udd[:,i+1] - udd[:,i]) * Δt^2)
87     push!(e_abs,e_abs_n1)
88     η_n1 = e_abs[i+1] / norm(u[:,i+1] - u[:,i])
89     push!(η,η_n1)
90     push!(e_cum, sum(e_abs))
91
92     # adapting the time step
93     if η_n1 > ub || η_n1 < lb
94         # here means Δt is either to small or to big
95         Δt = Δt * sqrt(η_e/η_n1)
96     end
97     t_current += Δt
98     push!(t,t_current)
99     push!(tstep,Δt)
100    i+=1
101
102    end
103
104    (response = DynamicResponse(u,ud,udd), error = Error(e_abs,η,e_cum),time =
105     TimeEvolution(t,tstep))
106

```

## 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.

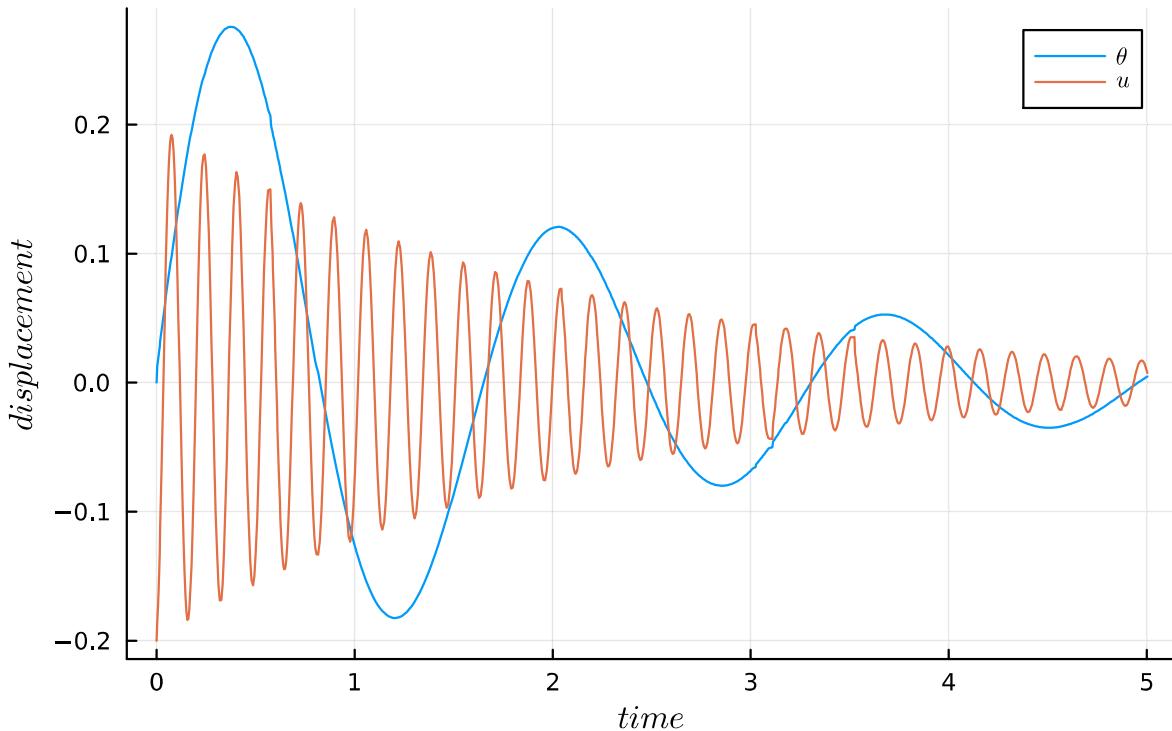
```
time_bound = AdaptiveTimeBoundary(1.0, 10.0, 0.001)
1 time_bound = AdaptiveTimeBoundary(1.0,10.0,1e-3)

(
    response = DynamicResponse(2x871 Matrix{Float64}:
        0.0  0.012712  0.0165854  0.0204446  ...
        0.0  -0.185612 -0.17552   -0.162992 ...
    error = Error([0.0, 0.000203842, 1.25261e-5, 1.53519e-5, 1.79533e-5, ...
    time = TimeEvolution([0.0, 0.00306901, 0.00613802, 0.00920704, 0.012276, ...
)
    ↪
    ↵
    ↷
    ↸

1 ad_resp , ad_err,ad_time =
generalized_alpha_adaptive(momentum,initial_coditions,runtime,physical_damping_a,numerical_damping_a,time_bound)

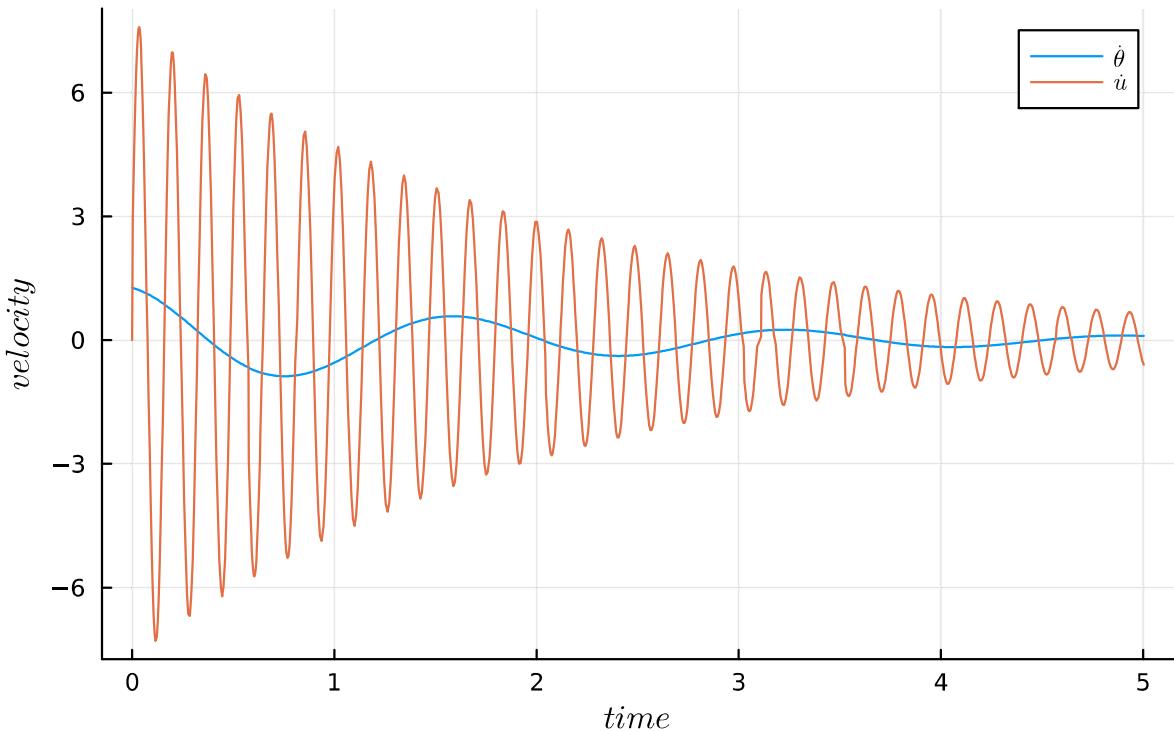
2x871 Matrix{Float64}:
 0.0  0.012712  0.0165854  0.0204446  ...
 -0.2  -0.185612 -0.17552   -0.162992 ...
 1 ad_resp.u
```

## Displacement Response (Adaptive)



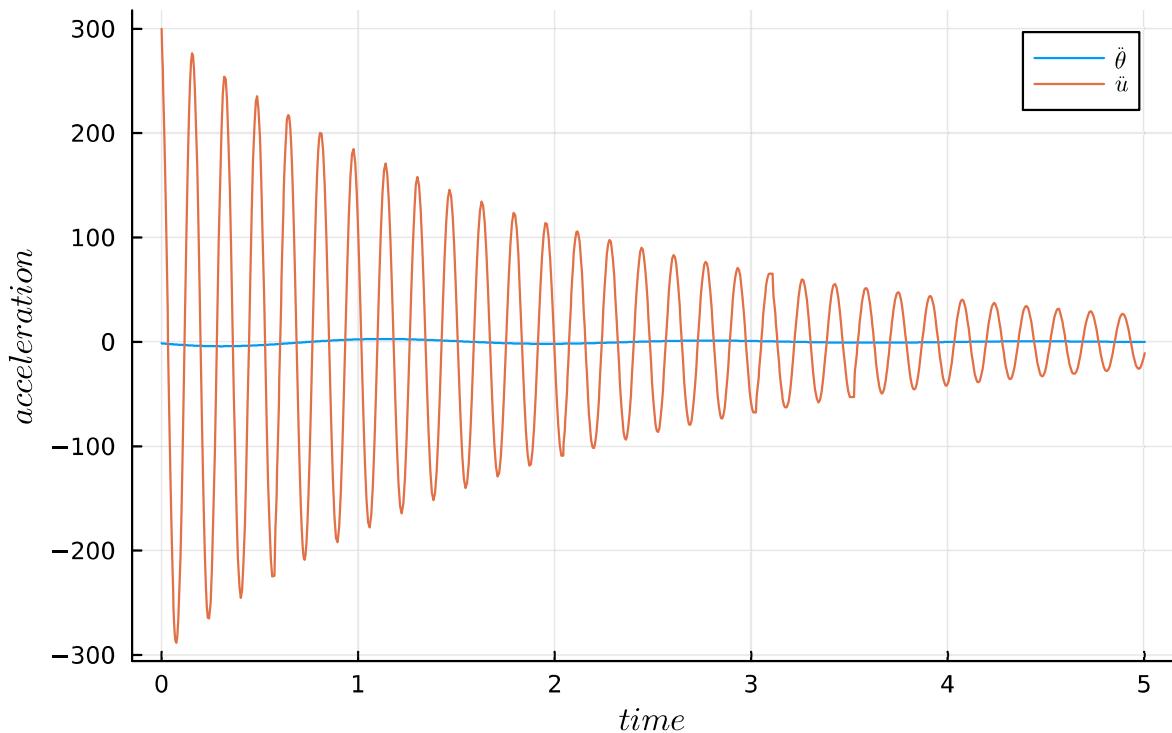
```
1 begin
2   plot(ad_time.times,ad_resp.u[1,:],title="Displacement Response
3   (Adaptive)", xlabel=L"time", ylabel=L"displacement", label=L"\theta")
4   plot!(ad_time.times,ad_resp.u[2,:],label=L"u")
5 end
```

## Velocity Response (Adaptive)



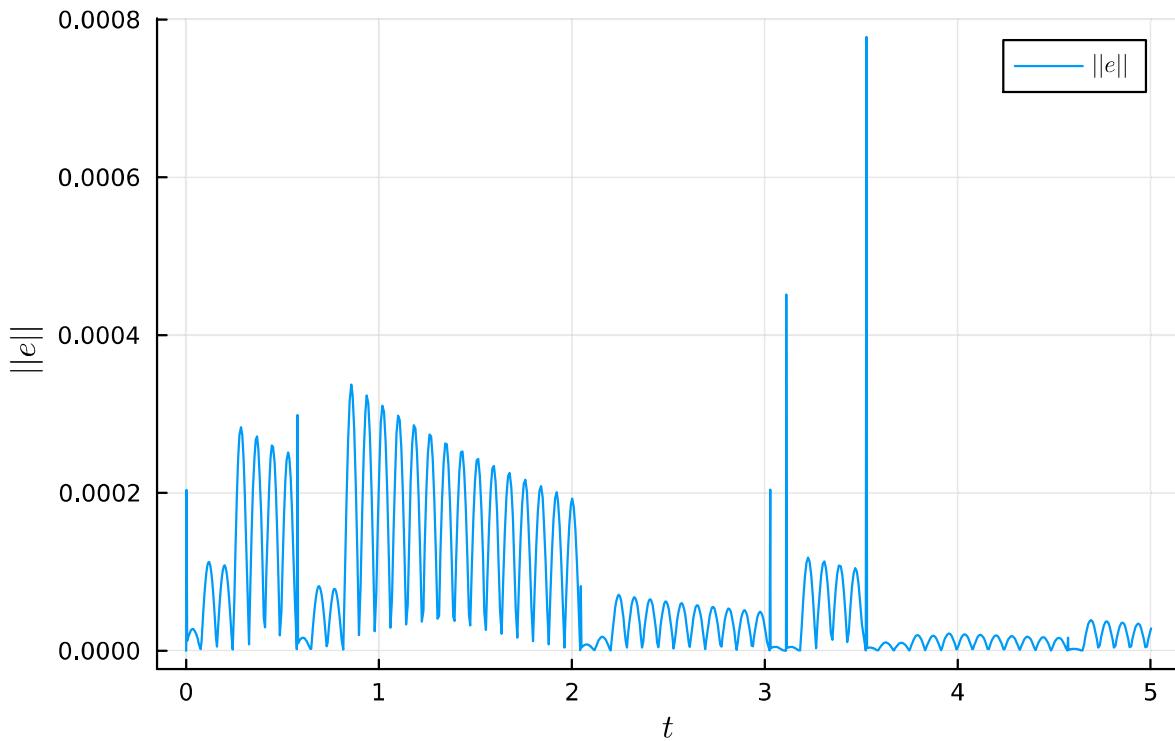
```
1 begin
2     plot(ad_time.times,ad_resp.ud[1,:],title="Velocity Response
3         (Adaptive)", xlabel=L"time", ylabel=L"velocity", label=L"\dot{\theta}")
4     plot!(ad_time.times,ad_resp.ud[2,:],label=L"\dot{u}")
5 end
```

## Acceleration Response (Adaptive)



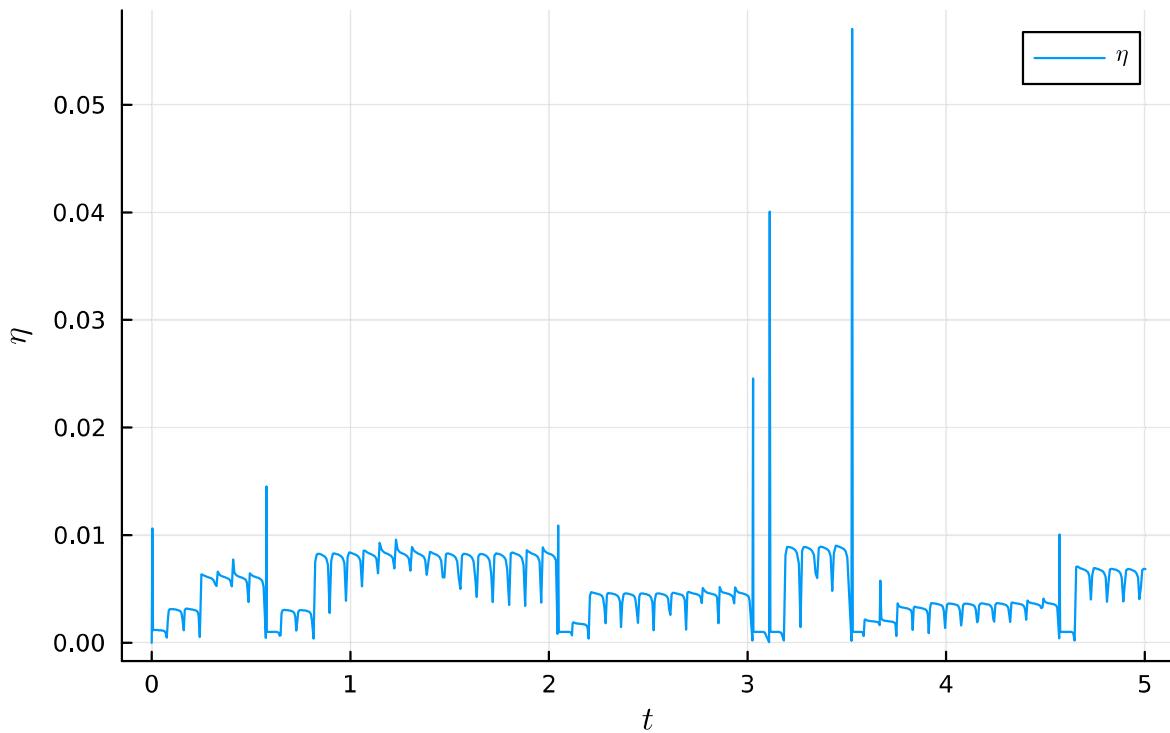
```
1 begin
2   plot(ad_time.times,ad_resp.udd[1,:],title="Acceleration Response
  (Adaptive)", xlabel=L"time", ylabel=L"acceleration", label=L"\ddot{\theta}")
3   plot!(ad_time.times,ad_resp.udd[2,:],label=L"\ddot{u}")
4 end
```

## Absolute Error (Task 6)



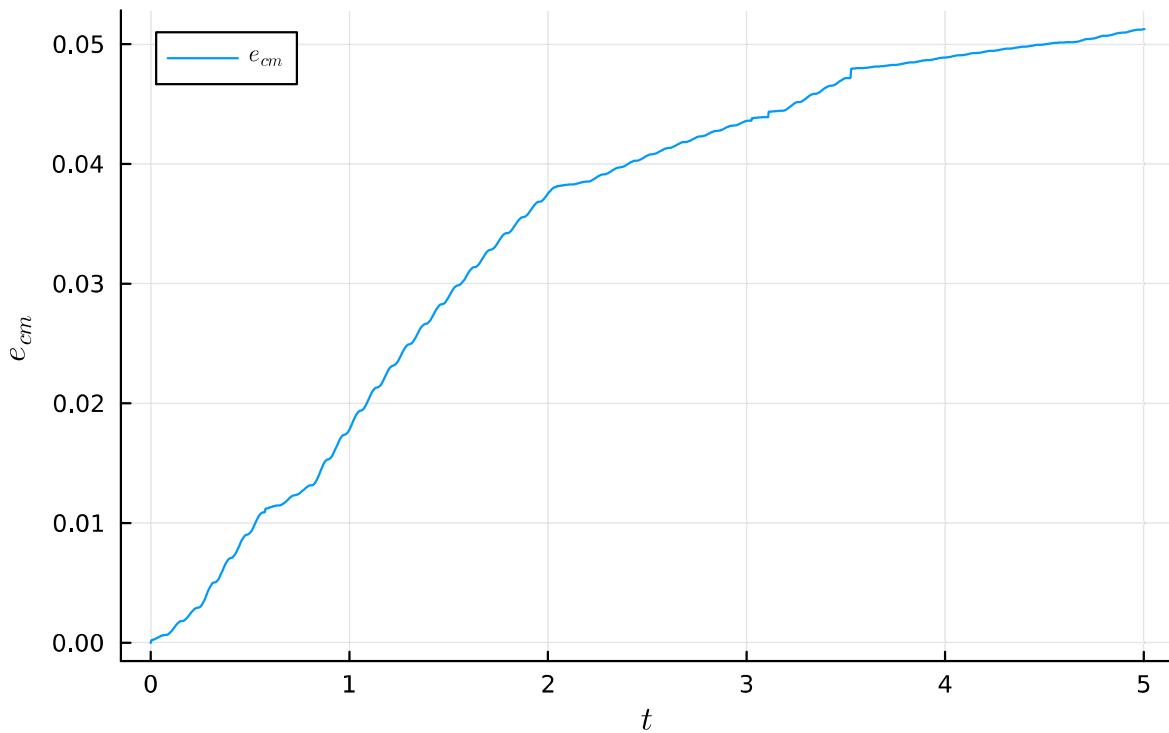
```
1 begin
2     plot(ad_time.times,ad_err.e_abs,title="Absolute Error (Task
6)", xlabel=L "t", ylabel=L "\|e\|", label=L "$\|e\|$")
3 end
```

## Relative Error (Adaptive)



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

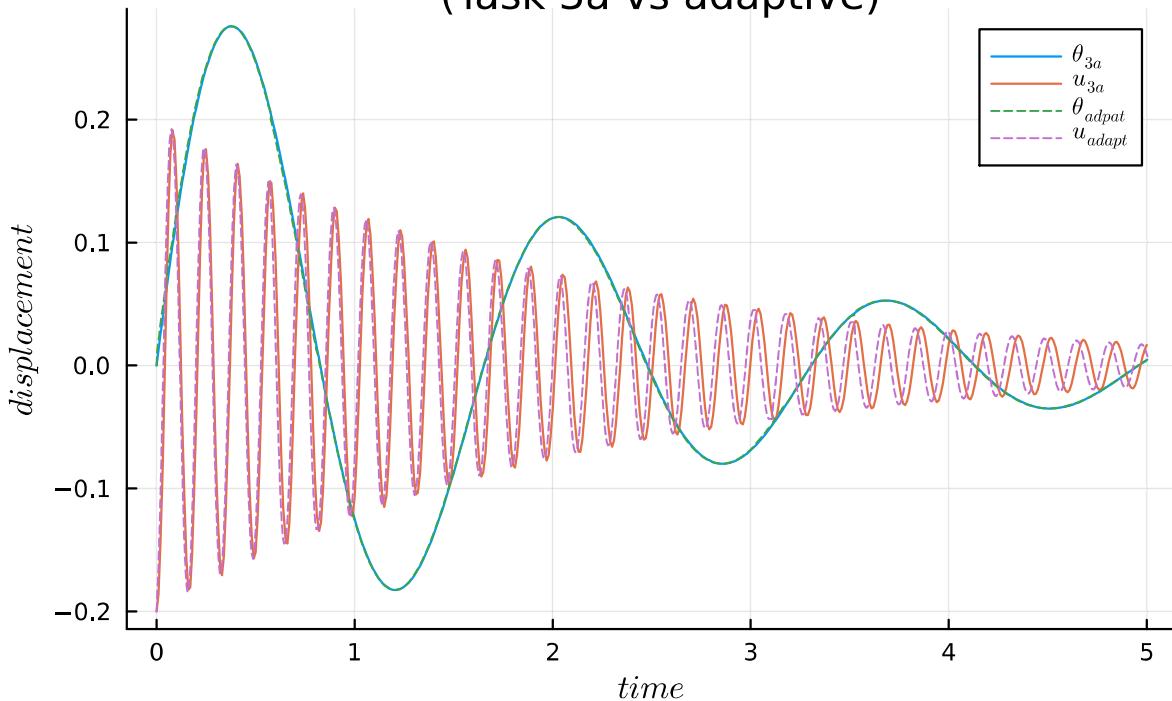
## Cumulative Error (Adaptive)



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

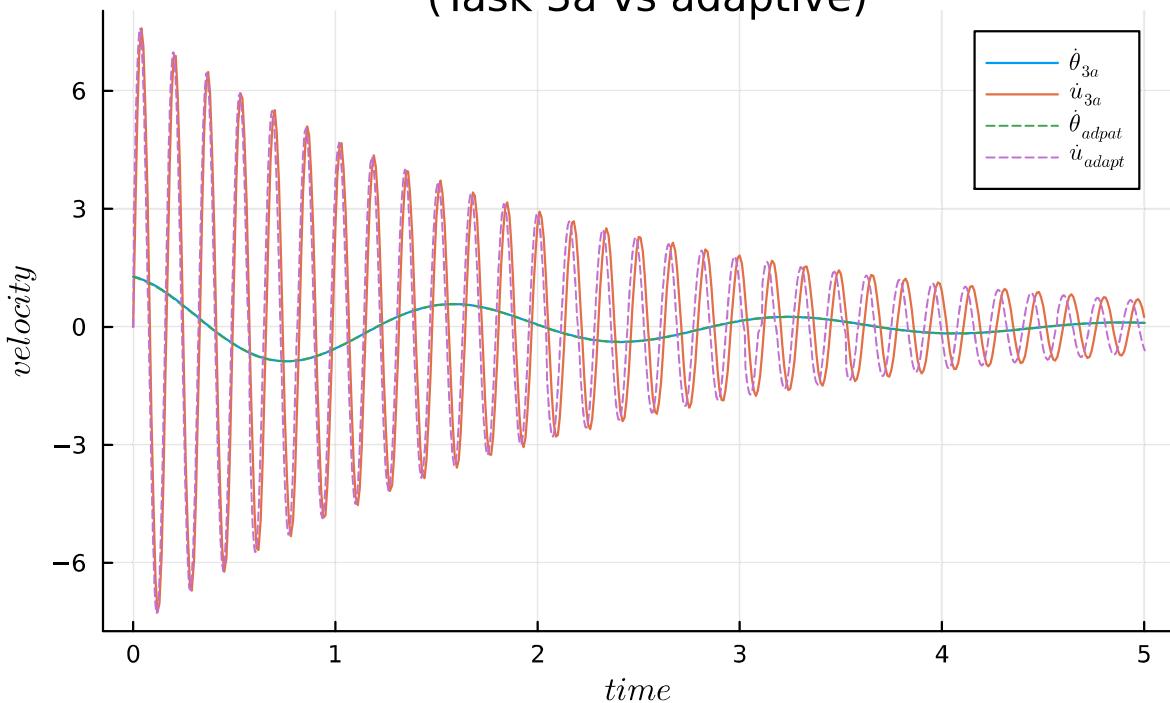
Comparing results:

## Displacement response comparison (Task 3a vs adaptive)



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

## Velocity response comparison (Task 3a vs adaptive)

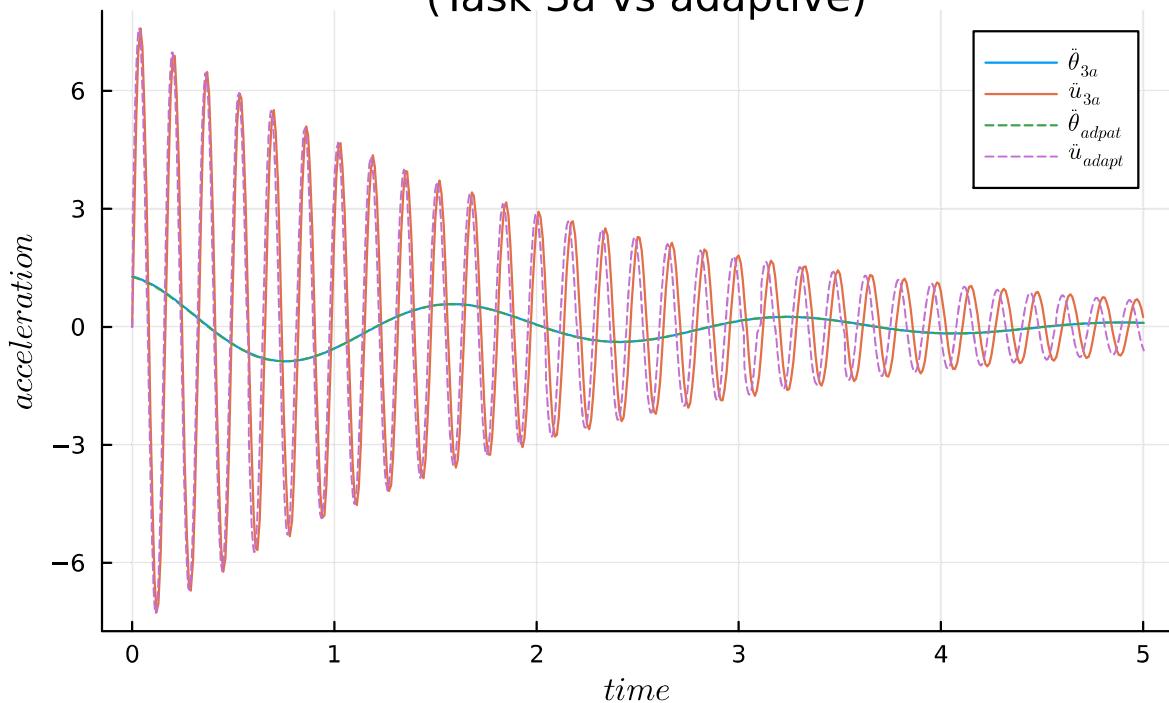


```

1 begin
2   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}")
3   plot!(time_a.times,resp_a.ud[2,:],label=L"\dot{u}_{3a}")
4   plot!(ad_time.times,ad_resp.ud[1,:],label=L"\dot{\theta}_{adapt}",linestyle=:dash)
5   plot!(ad_time.times,ad_resp.ud[2,:],label=L"\dot{u}_{adapt}",linestyle=:dash)
6 end

```

## Acceleration response comparison (Task 3a vs adaptive)

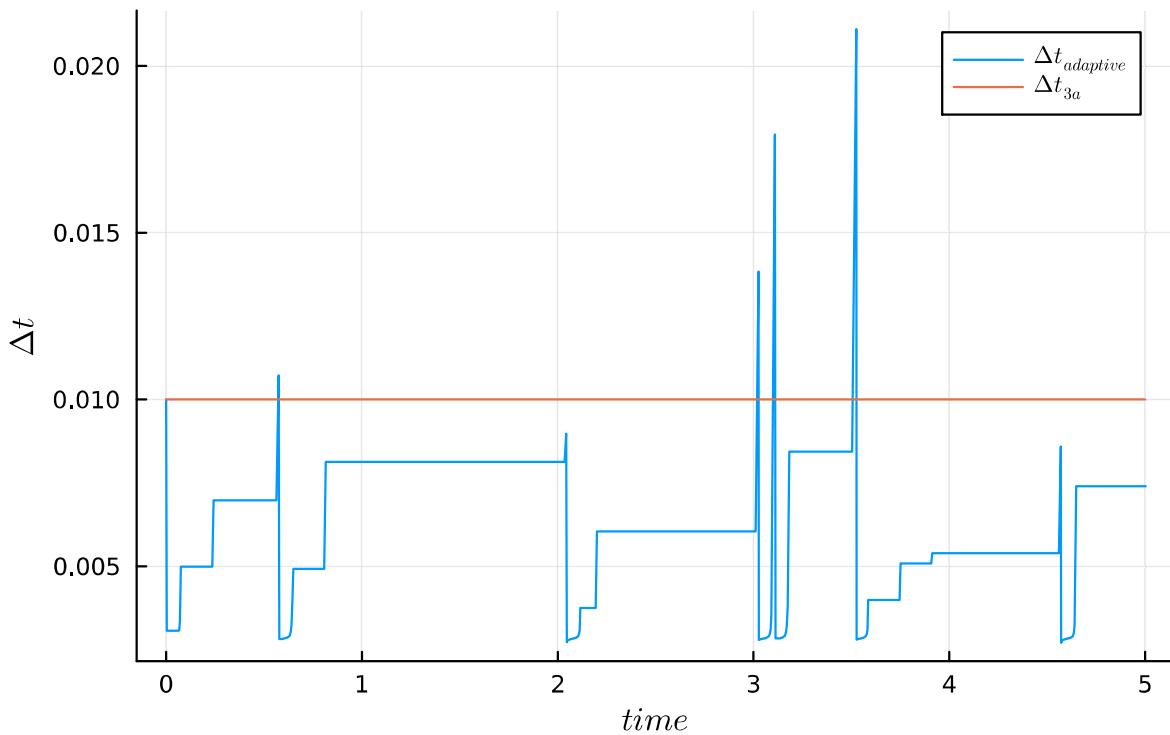


```

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

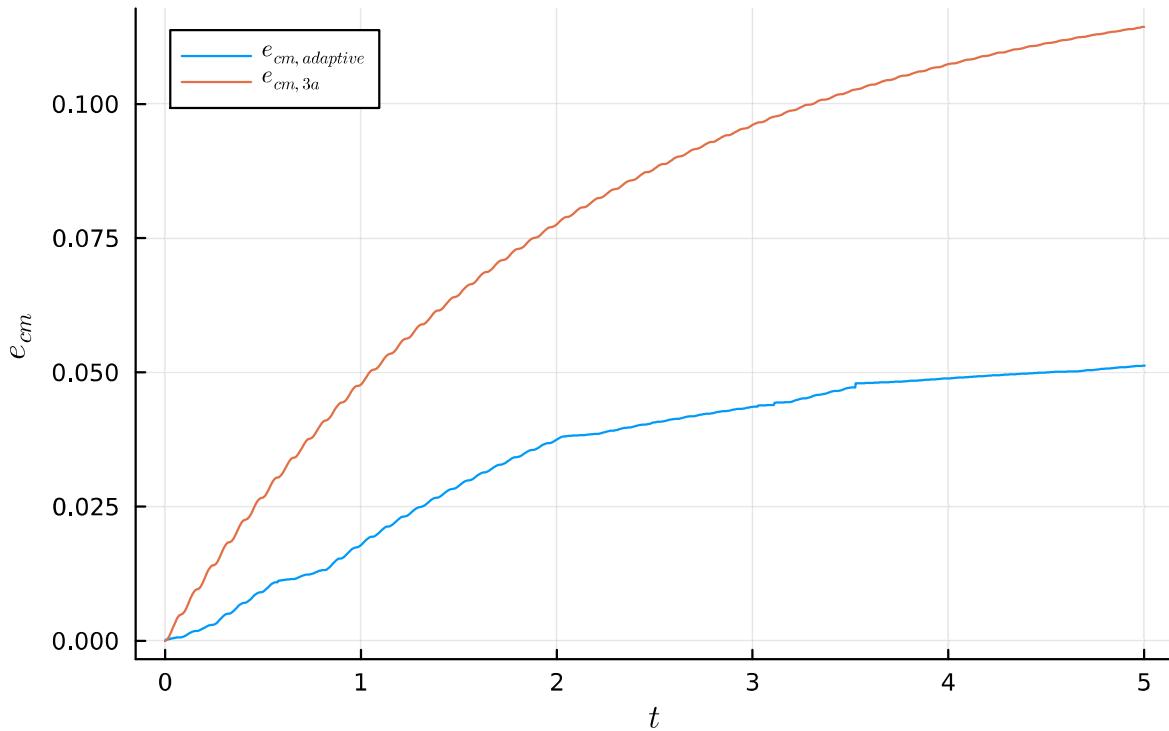
```

## Time step evolution (Task 3a vs adaptive)



```
1 begin
2     plot(ad_time.times,ad_time.steps,title="Time step evolution (Task 3a vs
3         adaptive)", xlabel=L"time", ylabel=L"\Delta t", label=L"\$\\Delta_t_{adaptive}\\$")
4     plot!(time_a.times,time_a.steps,label=L"\$\\Delta_t_{3a}\\$")
5 end
```

## Cumulative Error (Adaptive & Task 3a)



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

**Cumulative error after 5 s:**

```
(adapt_cum_err = 0.0512708, t3a_cum_error = 0.114384)
```

```
1 (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)
```

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

