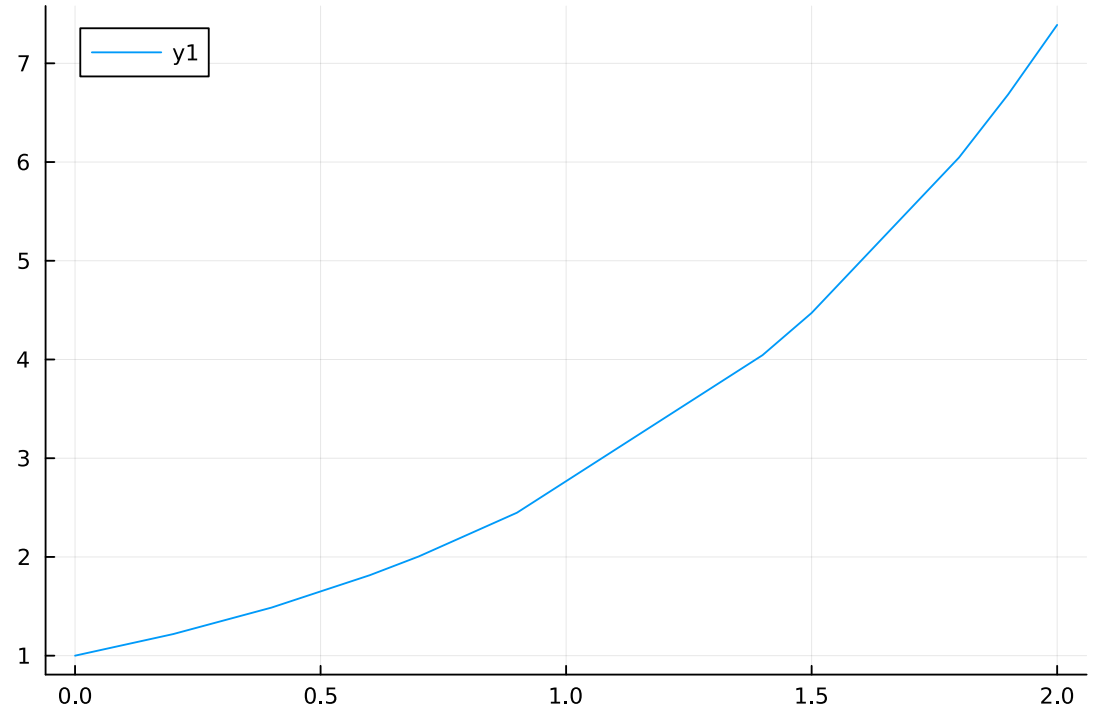
Exercise 1.2

In [1]:
```
using LinearAlgebra
using SparseArrays
using Plots
using Optim
using Polynomials
```

a)

In [2]:
```
function BVP1D(c::Float64, d::Float64, x::Vector{Float64}) :: Vector{Floa
    M = length(x)

    # Algorithm 1
    A = spzeros(M, M)
    b = zeros(M)

    for i in 1:M-1
        h = x[i+1] - x[i]
        k1 = 1/h + h/3
        k2 = -1/h + h/6

        A[i, i] += k1
        A[i, i+1] = k2
        A[i+1, i] = k2
        A[i+1, i+1] = k1
    end


    # Algorithm 2
    b[1] = c
    b[2] -= A[1,2] * c
    b[M-1] -= A[M-1,M] * d
    b[M] = d

    A[1,1] = 1
    A[1,2] = 0
    A[2,1] = 0
    A[M, M] = 1
    A[M-1, M] = 0
    A[M, M-1] = 0

    u = A \ b

    return u
end

c = 1.0
d = exp(2)
x = [0.0, 0.2, 0.4, 0.6, 0.7, 0.9, 1.4, 1.5, 1.8, 1.9, 2.0]
u = BVP1D(c, d, x)
plot(x, u)
```

Out[2]:

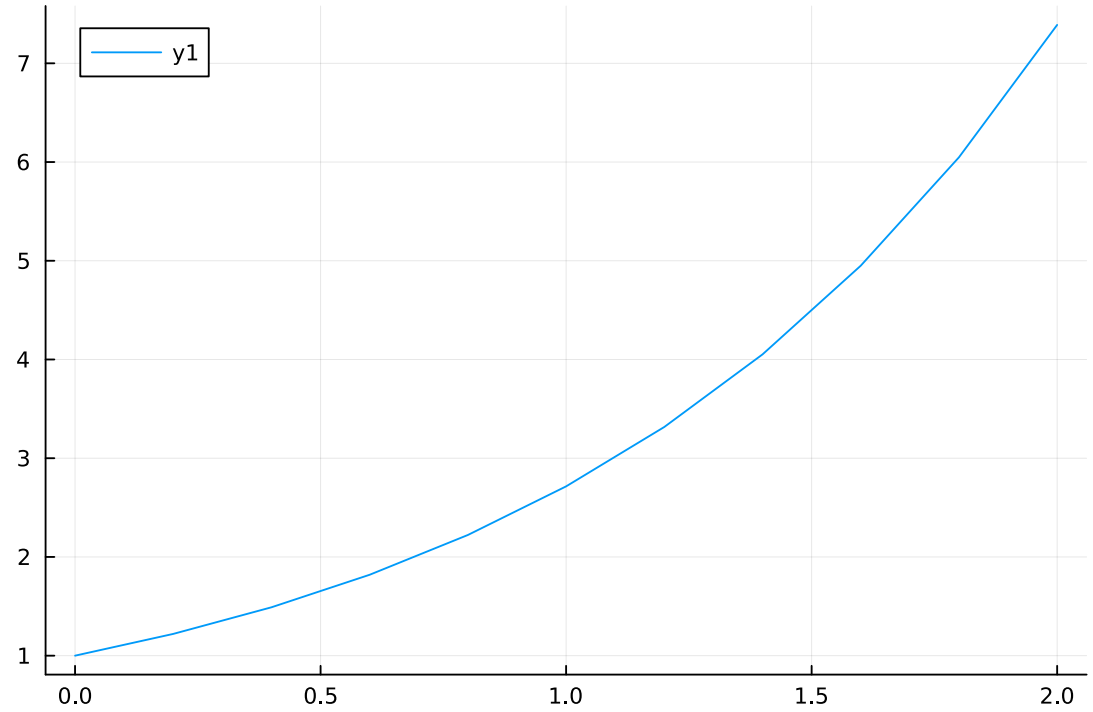


b)

In [3]:

```
function BVP1D(c::Float64, d::Float64, L::Float64, M::UInt64) :: Tuple{Ve
    x = collect(range(0, L, M))
    u = BVP1D(c, d, x)
    return u, x
end

c = 1.0
d = exp(2)
L = 2.0
M::UInt64 = 11

u, x = BVP1D(c, d, L, M)
plot(x, u)
```

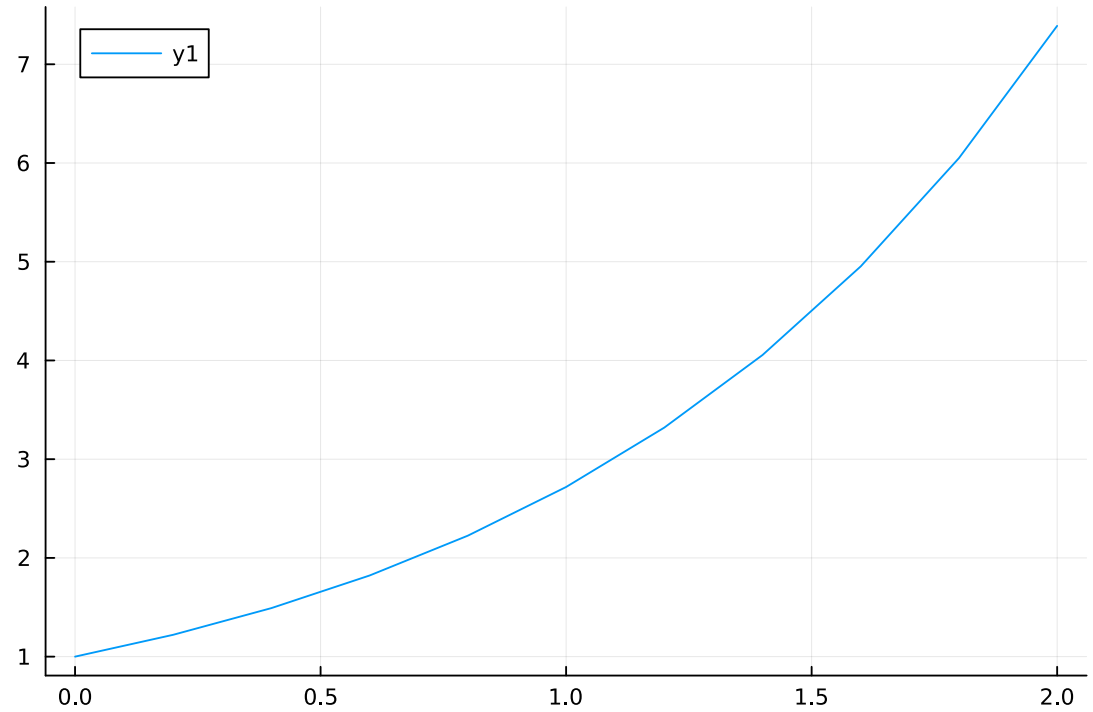
Out[3]:



c)

In [4]:
```
x = collect(range(0, L, M))
plot(x, exp.(x))
```

Out[4]:



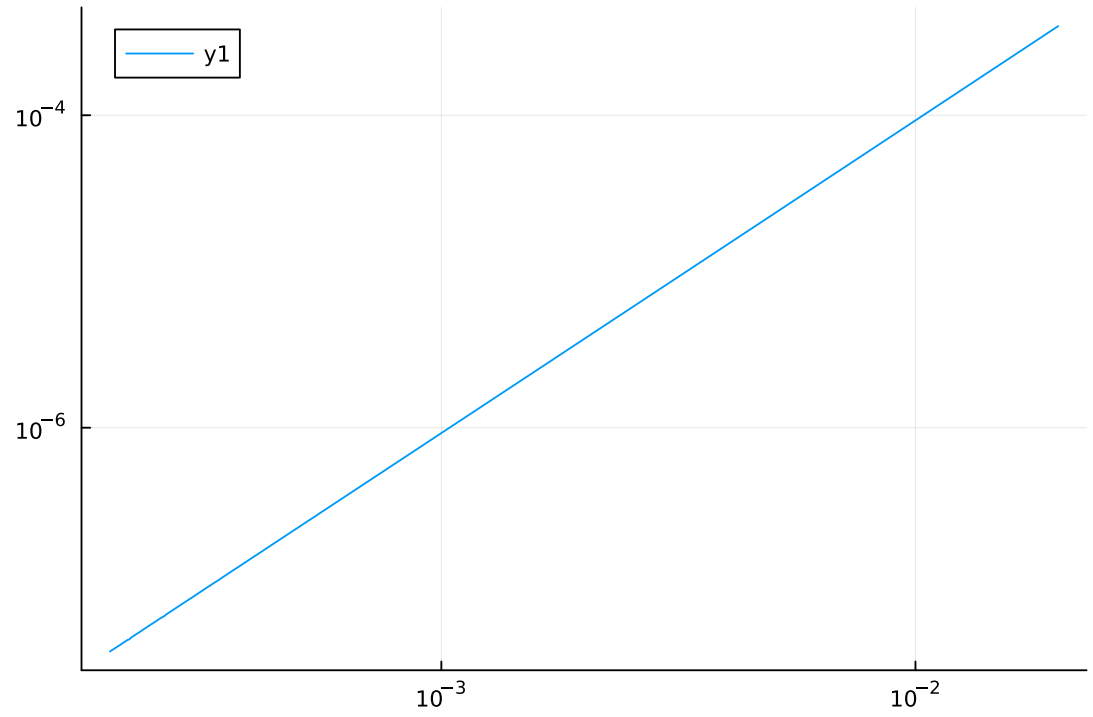
d)

In [5]:
```
function uhat(x::Float64, xs::Vector{Float64}, us::Vector{Float64}) :: Flo
    i = max(searchsortedfirst(xs, x) - 1, 1)
    h = xs[i+1] - xs[i]
```

```
    N1 = 1 - (x - xs[i]) / h
    N2 = (x - xs[i]) / h

    return us[i] * N1 + us[i+1] * N2
end

function uhat(x::Vector{Float64}, xs::Vector{Float64}, us::Vector{Float64}
    return uhat.(x, Ref(xs), Ref(us))
end

c = 1.0
d = exp(2)
L = 2.0

iterations = 100
step_size = 100
granularity = 100000

hs = zeros(iterations)
max_errors = zeros(iterations)

x_values = collect(range(0, L, granularity))

for i in 1:iterations
    M = i*step_size
    uhat_points, x_points = BVP1D(c, d, L, M)
    error(x) = abs(uhat(x, x_points, uhat_points) - exp(x))

    hs[i] = L / M
    max_errors[i] = maximum(error.(x_values))
end

display(fit(log.(hs), log.(max_errors), 1))
plot(hs, max_errors, xscale=:log10, yscale=:log10)
```

-0.07174784906369909 + 2.0010500256960615·x

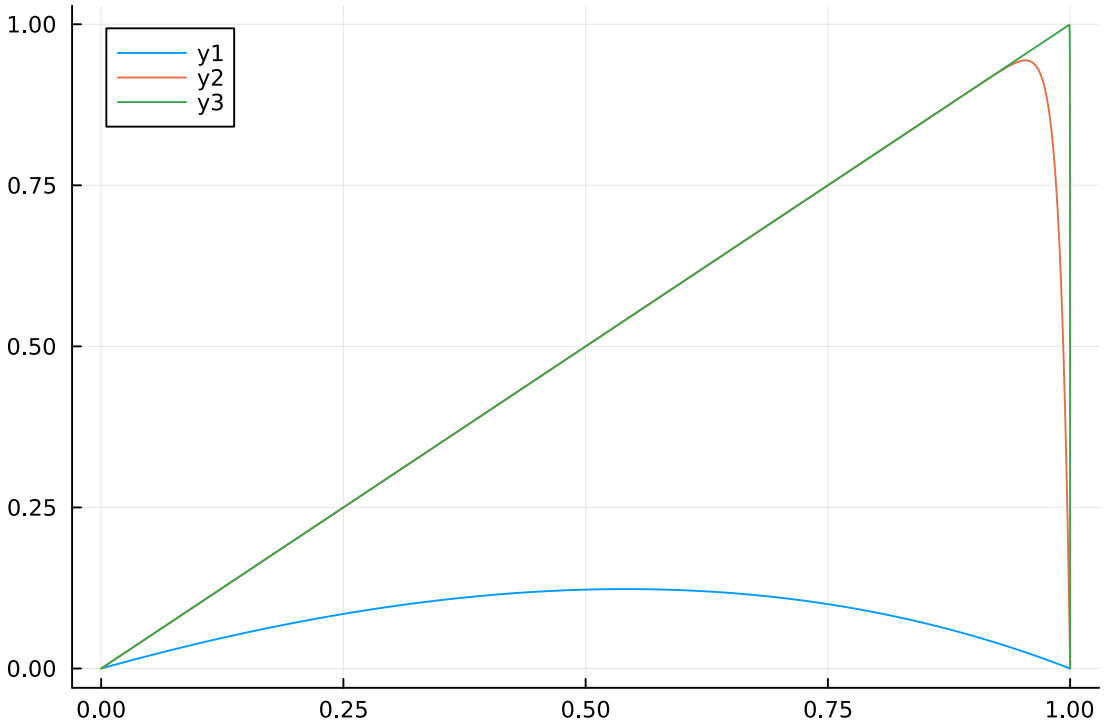
Out[5]:



Exercise 1.5

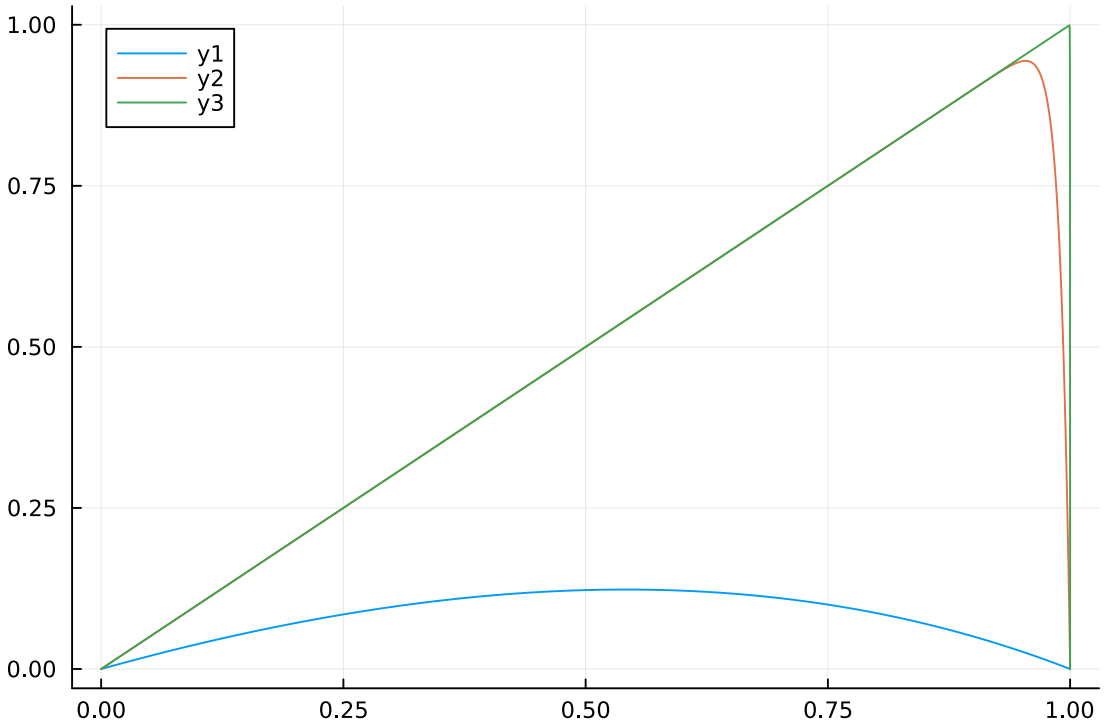
d)

In [6]:
```
psi = 1
x = collect(range(0,1,10000))

p = plot()
for epsilon in [1, 0.01, 0.0001]
    uanal(x) = 1/psi * (x - (exp((x-1)*psi/epsilon) - exp(-psi/epsilon))/(
    plot!(x, uanal.(x))
end
display(p)
```



In [7]:
```
function LADE1D(psi::Float64, epsilon::Float64, x::Vector{Float64}) :: Ve
    M = length(x)

    # Algorithm 1
    A = spzeros(M, M)
    b = zeros(M)

    for i in 1:M-1
        h = x[i+1] - x[i]
        psi2 = psi/2
        epsh = epsilon/h

        k11 = psi2 + epsh
        k12 = psi2 - epsh
        k21 = -psi2 - epsh
        k22 = -psi2 + epsh

        A[i, i] += k11
        A[i, i+1] = k12
```

```
        A[i+1, i] = k21
        A[i+1, i+1] = k22
    end

    for i in 2:M-1
        b[i] = (x[i+1] - x[i-1]) / 2
    end

    # Algorithm 2
    A[1,1] = 1
    A[1,2] = 0
    A[2,1] = 0
    A[M, M] = 1
    A[M-1, M] = 0
    A[M, M-1] = 0

    u = A \ b

    return u
end

psi = 1.0
x = collect(range(0,1,10000))

p = plot()
for epsilon in [1, 0.01, 0.0001]
    u = LADE1D(psi, epsilon, x)
    plot!(x, u)
end
display(p)
```



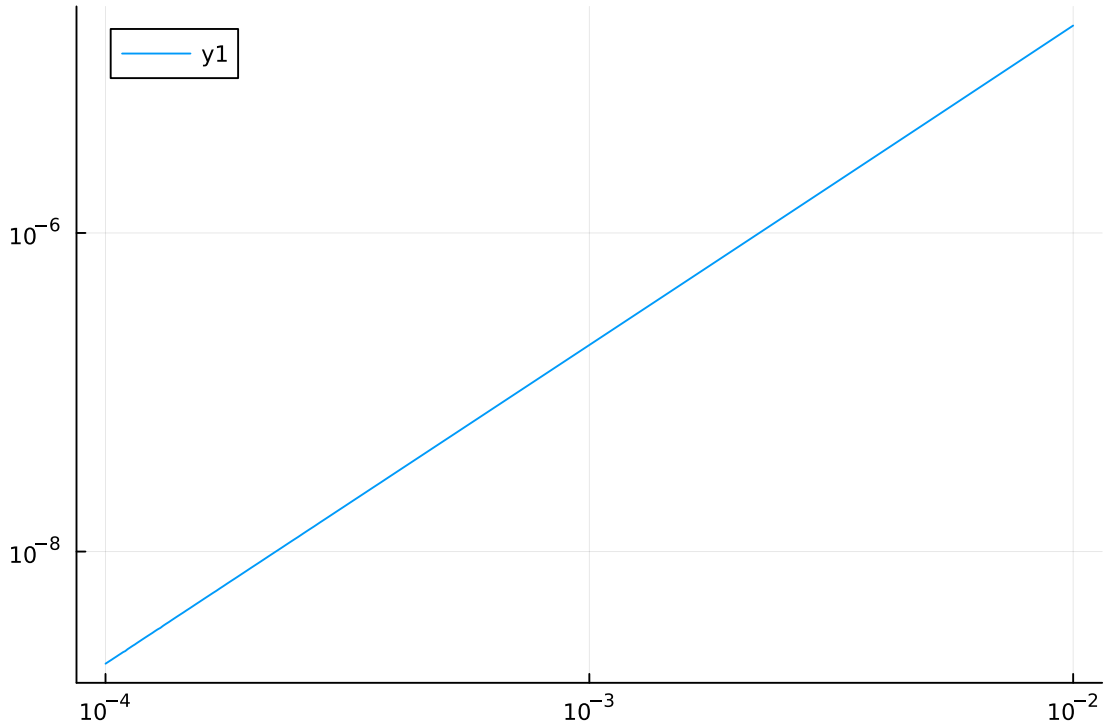
In [8]:

```
function uhat(x::Float64, xs::Vector{Float64}, us::Vector{Float64}) :: Flo
    i = max(searchsortedfirst(xs, x) - 1, 1)
    h = xs[i+1] - xs[i]

    N1 = 1 - (x - xs[i]) / h
```

```
    N2 = (x - xs[i]) / h

    return us[i] * N1 + us[i+1] * N2
end

function uhat(x::Vector{Float64}, xs::Vector{Float64}, us::Vector{Float64}
    return uhat.(x, Ref(xs), Ref(us))
end

psi = 1.0
iterations = 100
step_size = 100
granularity = 100000

for epsilon in [1, 0.01, 0.0001]
    uanal(x) = 1/psi * (x - (exp((x-1)*psi/epsilon) - exp(-psi/epsilon))/(

    hs = zeros(iterations)
    max_errors = zeros(iterations)

    x_values = collect(range(0, 1, granularity))

    for i in 1:iterations
        M = i*step_size
        x_points = collect(range(0, 1, M))
        uhat_points = LADE1D(psi, epsilon, x_points)
        error(x) = abs(uhat(x, x_points, uhat_points) - uanal(x))

        hs[i] = 1 / M
        max_errors[i] = maximum(error.(x_values))
    end

    display(epsilon)
    display(fit(log.(hs), log.(max_errors), 1))
    display(plot(hs, max_errors, xscale=:log10, yscale=:log10))
end
```

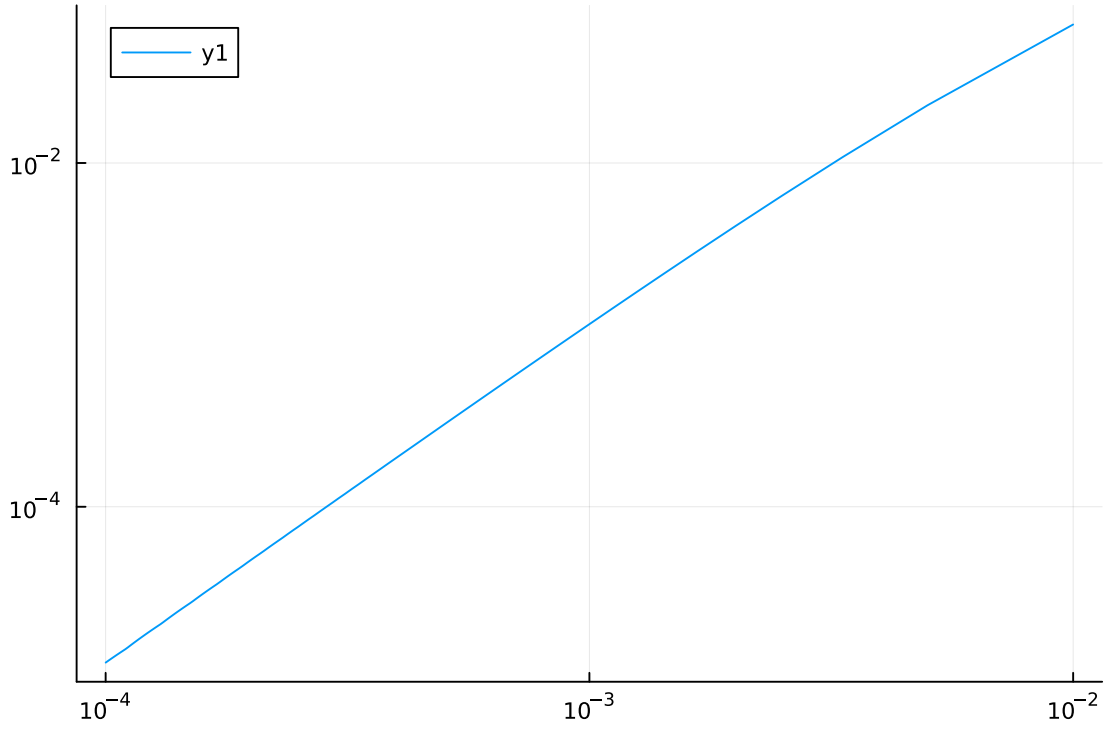
```
1.0
```

-1.6073154040275304 + 2.0016014785648872·x



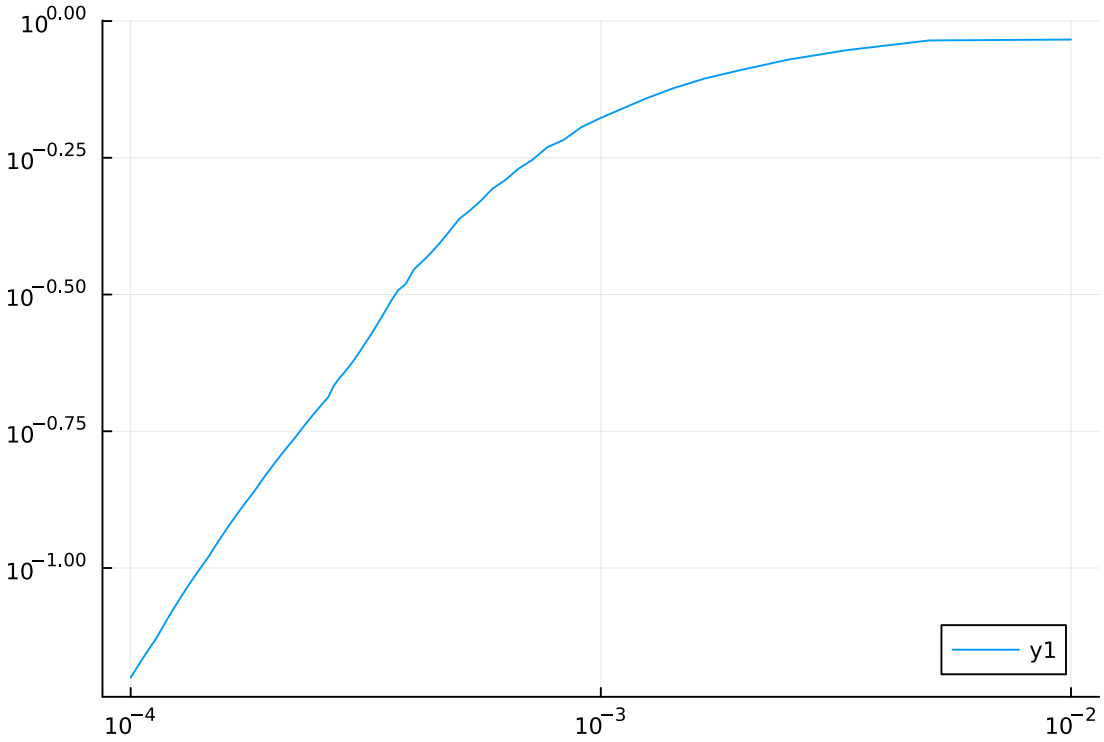

```
0.01
```

6.516245322851517 + 1.930376599777763·x




```
0.0001
```

5.030960208272509 + 0.8170226702495486·x

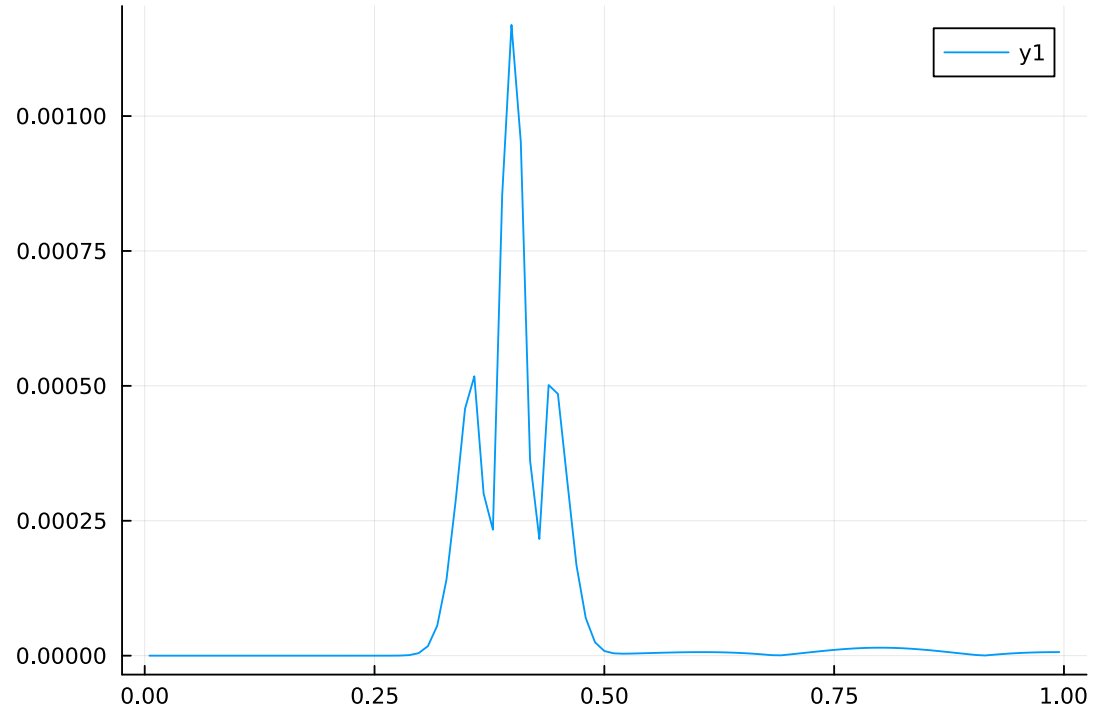


Exercise 1.6

a)

In [9]:
```
function compute_error_decrease(fun::Function, VX::Vector{Float64}, EToV:
    M, N = size(EToV)
    err = zeros(M)

    for i in 1:M
        for j in 1:N-1
            k = EToV[i, j]
            k2 = EToV[i, j+1]
            h = VX[k2] - VX[k]
            xi = VX[k]
            err[i] = sqrt(3)*sqrt(h*(fun(xi) + fun(xi + h) - 2*fun(xi + h/
        end
    end

    return err
end

uanal(x) = exp(-800*(x - 0.4)^2) + 0.25*exp(-40*(x - 0.8)^2)
M = 100
VX = collect(range(0, 1, M))
EToV = hcat(1:M-1, 2:M)

err_vec = compute_error_decrease(uanal, VX, EToV)
plot((VX[1:M-1] + VX[2:M])/2, err_vec)
```

Out[9]:



b)

In [10]:

```
function refine_marked(EToVcoarse::Matrix{UInt64}, xcoarse::Vector{Float64
    M, N = size(EToVcoarse)
    K = length(idxMarked)

    EToVfine = zeros((M + K, N))
    xfine = zeros(length(xcoarse) + K)

    i = 1
    for j in 1:M
        xfine[j+i-1] = xcoarse[j]
        EToVfine[j+i-1, :] = [j+i-1, j+i]

        if i <= K && idxMarked[i] == j
            xfine[j+i] = (xcoarse[j+1] + xcoarse[j])/2
            EToVfine[j+i, :] = [j+i, j+i+1]
            i += 1
        end
    end

    xfine[end] = xcoarse[end]

    return EToVfine, xfine
end
```

Out[10]: refine_marked (generic function with 1 method)

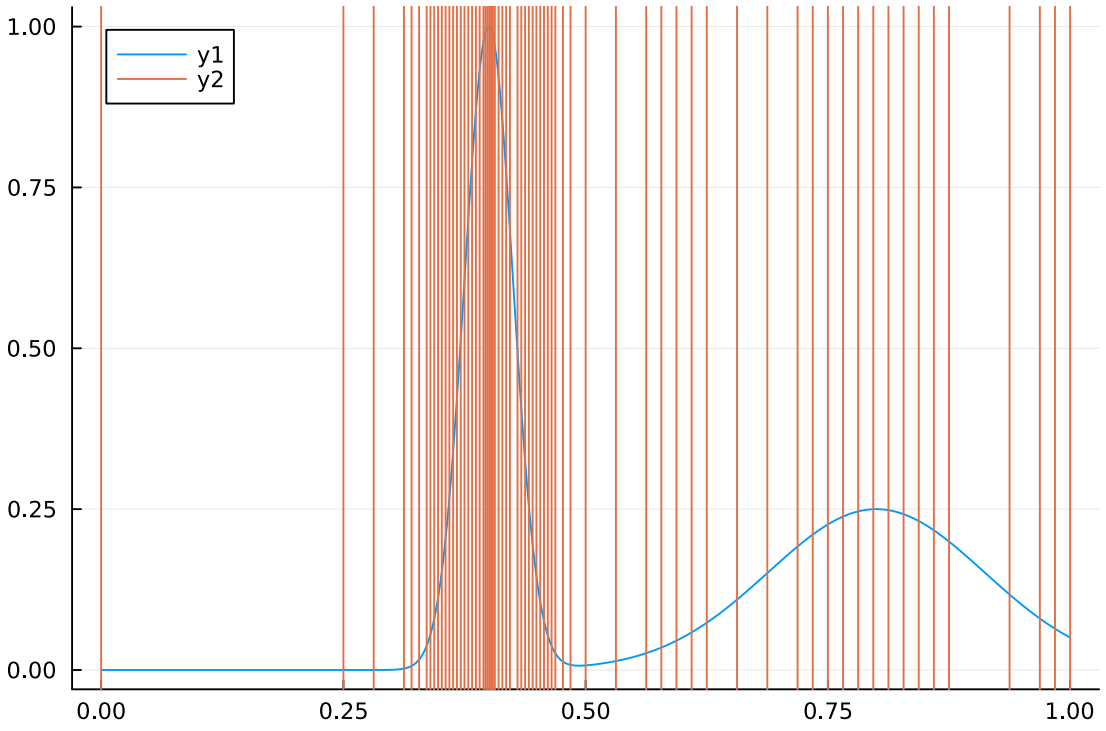
c) and d)

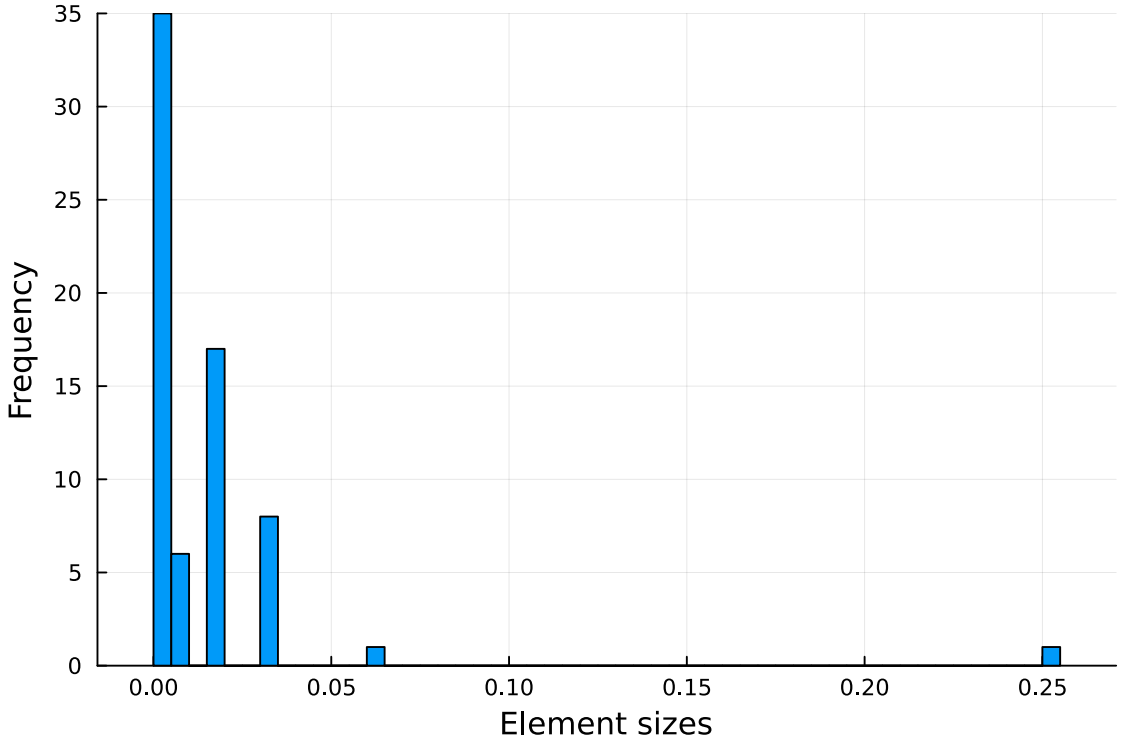
In [11]:

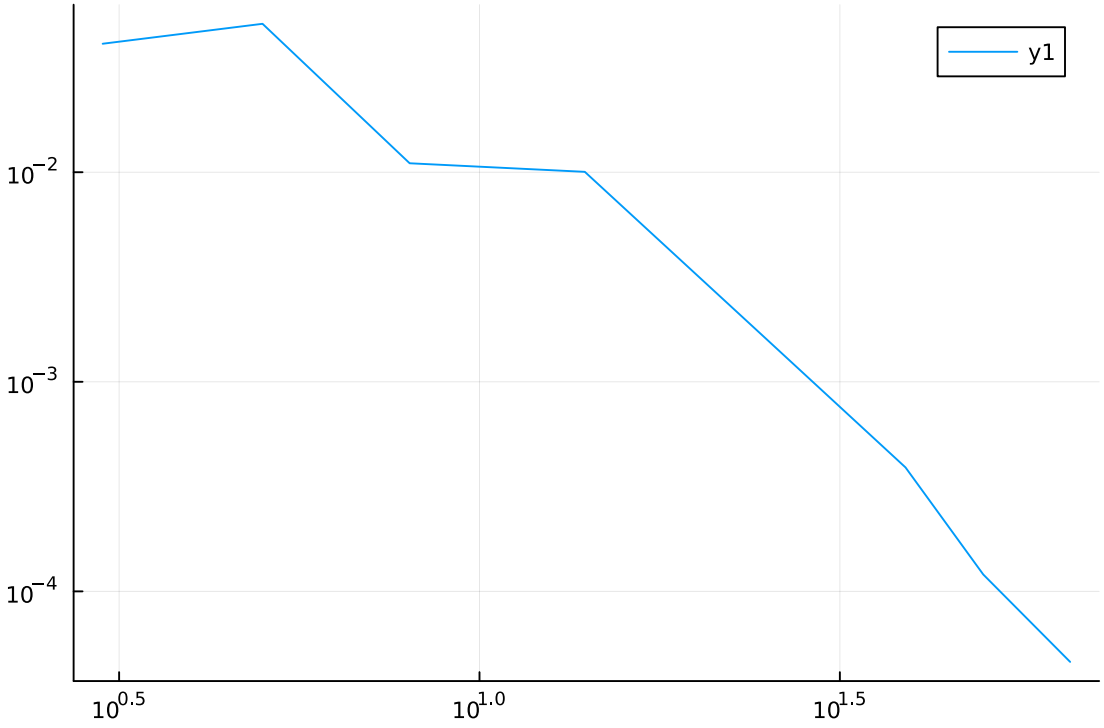
```
uanal(x) = exp(-800*(x - 0.4)^2) + 0.25*exp(-40*(x - 0.8)^2)
M = 3
VX = collect(range(0, 1, M))
```

```
EToV = hcat(1:M-1, 2:M)
tolerance = 10^(-4)

mean_errors = []
point_counts = []

while true
    err = compute_error_decrease(uanal, VX, EToV)
    idxMarked = findall(err .>= tolerance)

    if isempty(idxMarked)
        break
    end

    push!(mean_errors, sum(err) / length(err))
    push!(point_counts, length(VX))

    EToV, VX = refine_marked(EToV, VX, idxMarked)
end

display(length(VX))

x = collect(range(0, 1, 1000))
p = plot(x, uanal.(x))
vline!(VX)
display(p)

hs = VX[2:end] - VX[1:end-1]
display(histogram(hs, bins=100, xlabel="Element sizes", ylabel="Frequency"

display(plot(point_counts, mean_errors, xscale=:log10, yscale=:log10))
```

69







Exercise 1.7

b)

In [12]:
```
function errorestimate(
    xc::Vector{Float64},
    xf::Vector{Float64},
    uhc::Vector{Float64},
    uhf::Vector{Float64},
```

```
    EToVc::Matrix{UInt64},
    EToVf::Matrix{UInt64},
    idxMarked::Vector{Int64}
)::Vector{Float64}
    K = length(idxMarked)
    err = zeros(K)

    for k in 1:K
        i = idxMarked[k]
        j = i + k - 1
        h = xc[i+1] - xc[i]

        err[i] = sqrt(3)*sqrt(2)*sqrt(h*(2*uhf[j+1]^2 + (uhf[j] - 3*uhc[i]
    end

    return err
end
```

Out[12]: errorestimate (generic function with 1 method)

c)

In [13]:
```
function refine_marked(EToVcoarse::Matrix{UInt64}, xcoarse::Vector{Float64
    M, N = size(EToVcoarse)
    K = length(idxMarked)

    EToVfine = zeros((M + K, N))
    xfine = zeros(length(xcoarse) + K)

    i = 1
    for j in 1:M
        xfine[j+i-1] = xcoarse[j]
        EToVfine[j+i-1, :] = [j+i-1, j+i]

        if i <= K && idxMarked[i] == j
            xfine[j+i] = (xcoarse[j+1] + xcoarse[j])/2
            EToVfine[j+i, :] = [j+i, j+i+1]
            i += 1
        end
    end

    xfine[end] = xcoarse[end]

    return EToVfine, xfine
end
```

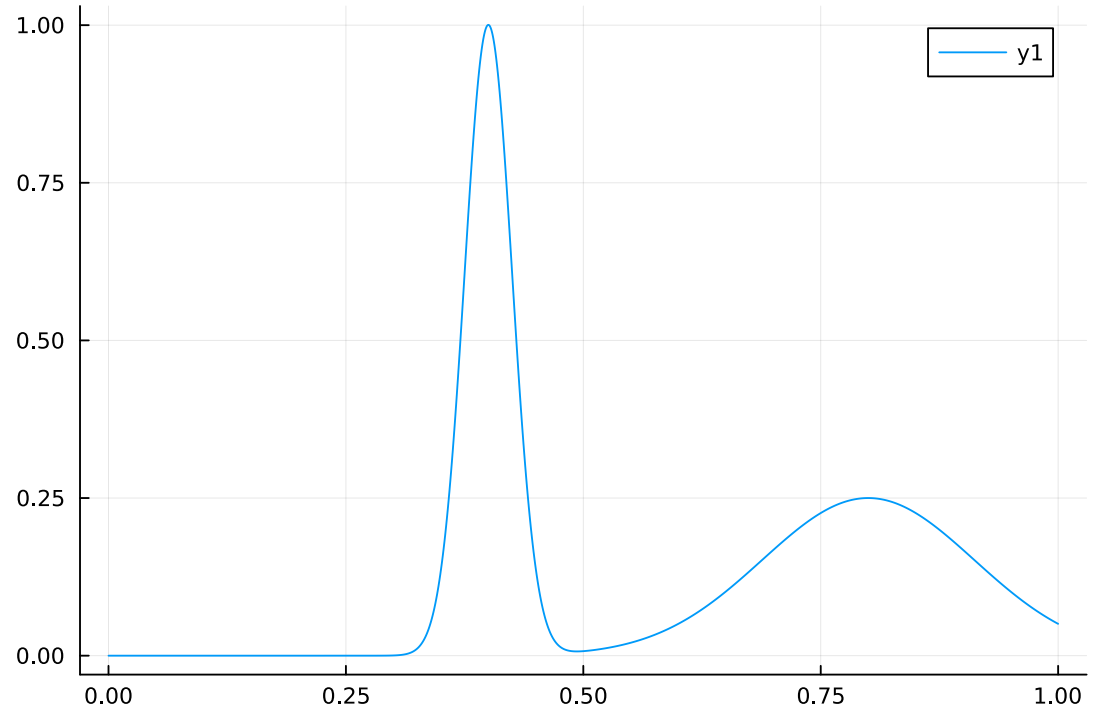
Out[13]: refine_marked (generic function with 1 method)

e)

In [15]:
```
function BVP1Drhs(c::Float64, d::Float64, x::Vector{Float64}, f::Function
    M = length(x)

    # Algorithm 1
    A = spzeros(M, M)
    b = zeros(M)
```

```
    for i in 1:M-1
        h = x[i+1] - x[i]
        k1 = 1/h + h/3
        k2 = -1/h + h/6

        A[i, i] += k1
        A[i, i+1] = k2
        A[i+1, i] = k2
        A[i+1, i+1] = k1
    end

    for i in 2:M-1
        h1 = x[i] - x[i-1]
        h2 = x[i+1] - x[i]
        b[i] = -f(x[i-1])*h1/6 - f(x[i])*(h1 + h2)/3 - f(x[i+1])*h2/6
    end

    # Algorithm 2
    b[1] = c
    b[2] -= A[1,2] * c
    b[M-1] -= A[M-1,M] * d
    b[M] = d

    A[1,1] = 1
    A[1,2] = 0
    A[2,1] = 0
    A[M, M] = 1
    A[M-1, M] = 0
    A[M, M-1] = 0

    u = A \ b

    return u
end

function BVP1Drhs(c::Float64, d::Float64, L::Float64, M::UInt64, f::Funct
    x = collect(range(0, L, M))
    u = BVP1Drhs(c, d, x, f)
    return u, x
end


c = 1.905466299*10^(-12)
d = 0.05047412950
f(x) = -1601*exp(-800*(x - 0.4)^2) + (-1600*x + 640.0)^2*exp(-800*(x - 0.4

M = 10000
L = 1.0
u, x = BVP1Drhs(c, d, L, M, f)
plot(x, u)
```

Out[15]:

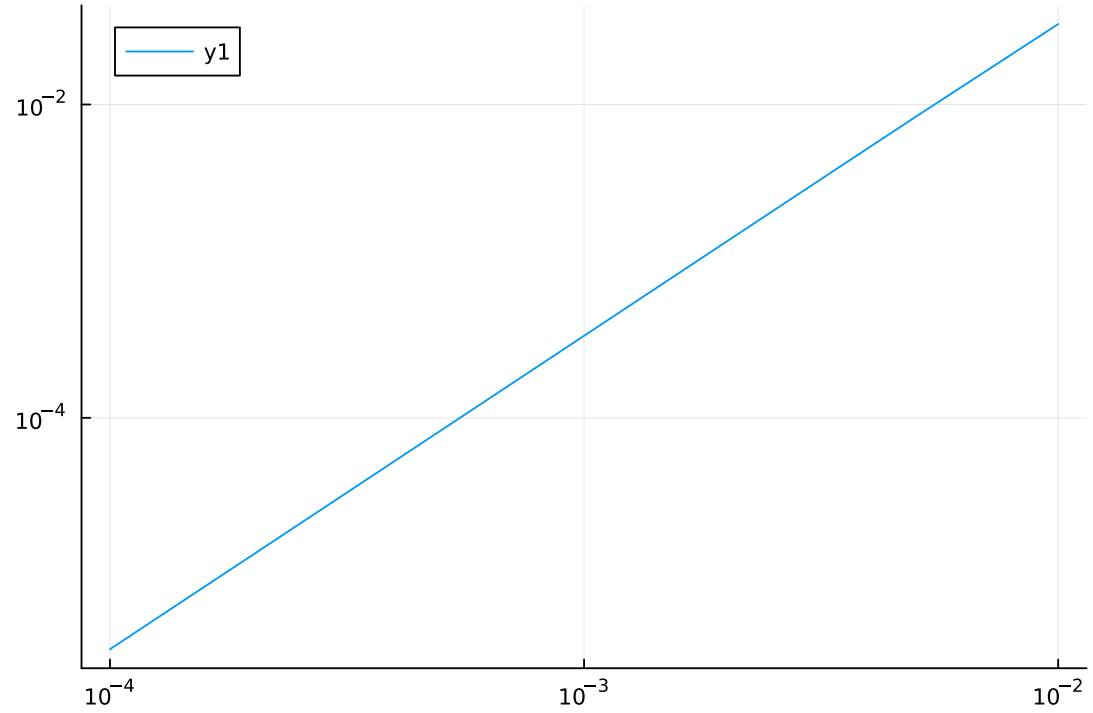


In [16]:

```
function uhat(x::Float64, xs::Vector{Float64}, us::Vector{Float64}) :: Flo
    i = max(searchsortedfirst(xs, x) - 1, 1)
    h = xs[i+1] - xs[i]

    N1 = 1 - (x - xs[i]) / h
    N2 = (x - xs[i]) / h

    return us[i] * N1 + us[i+1] * N2
end

function uhat(x::Vector{Float64}, xs::Vector{Float64}, us::Vector{Float64}
    return uhat.(x, Ref(xs), Ref(us))
end

iterations = 100
step_size = 100
granularity = 100000

hs = zeros(iterations)
max_errors = zeros(iterations)

x_values = collect(range(0, L, granularity))

uanal(x) = exp(-800*(x - 0.4)^2) + 0.25*exp(-40*(x - 0.8)^2)

for i in 1:iterations
    M = i*step_size
    uhat_points, x_points = BVP1Drhs(c, d, L, M, f)
    error(x) = abs(uhat(x, x_points, uhat_points) - uanal(x))

    hs[i] = L / M
    max_errors[i] = maximum(error.(x_values))
end

display(fit(log.(hs), log.(max_errors), 1))
plot(hs, max_errors, xscale=:log10, yscale=:log10)
```

5.80636675093669 + 1.9996163537707674·x

Out[16]:



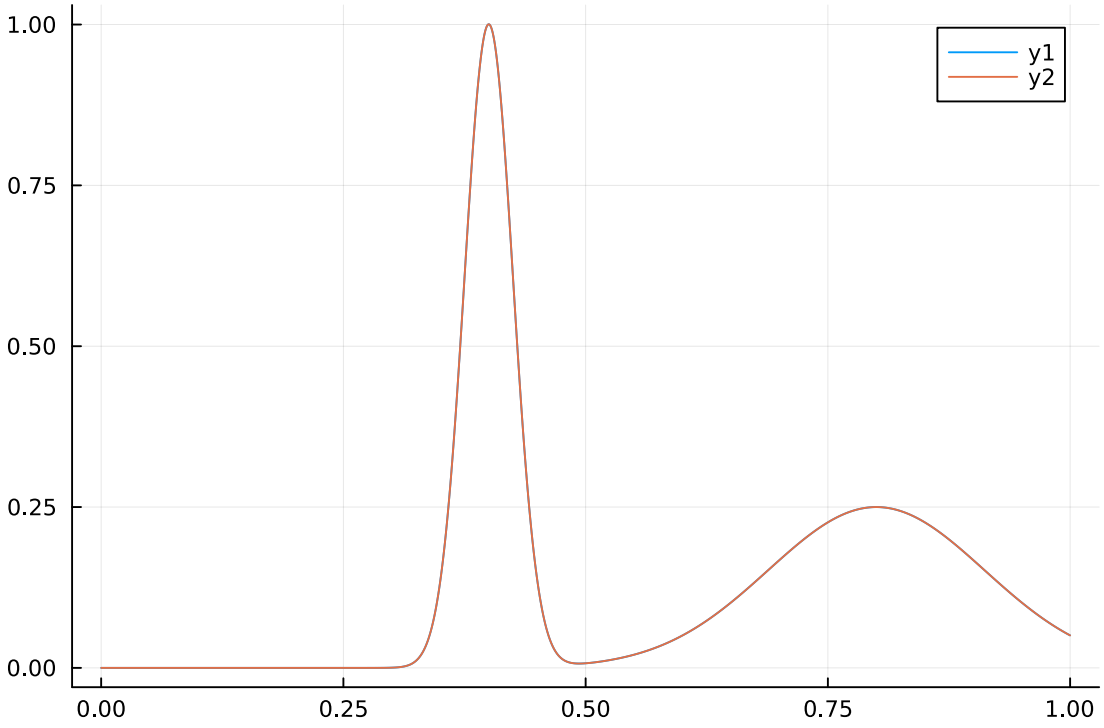
f)

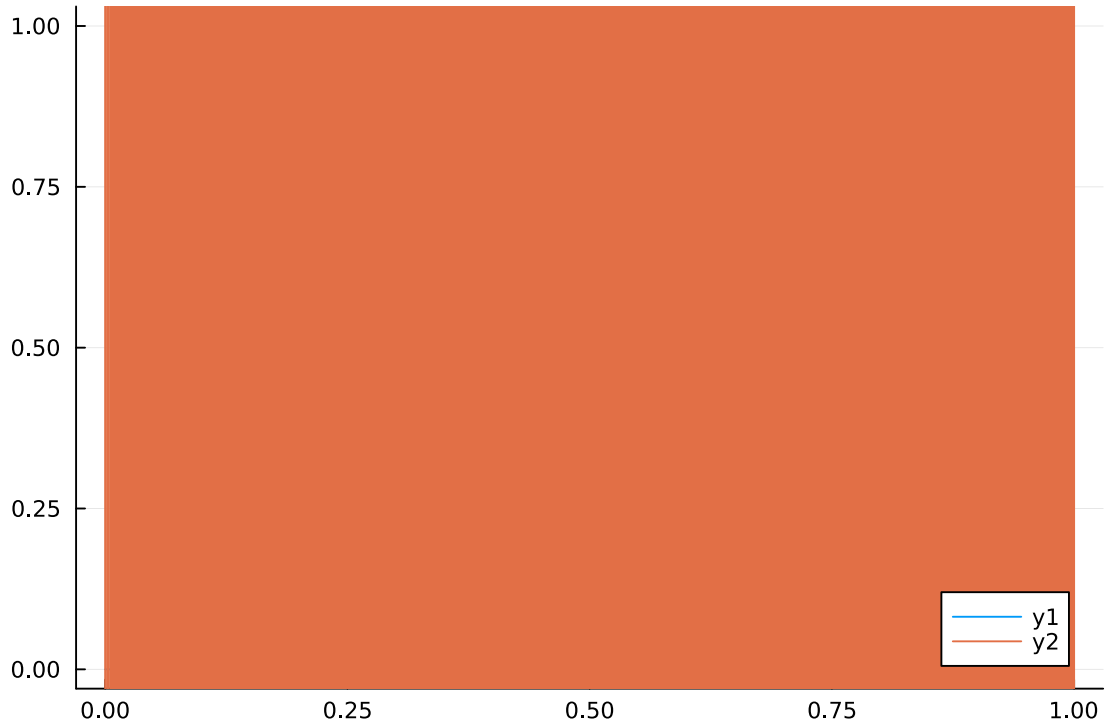
In [22]:

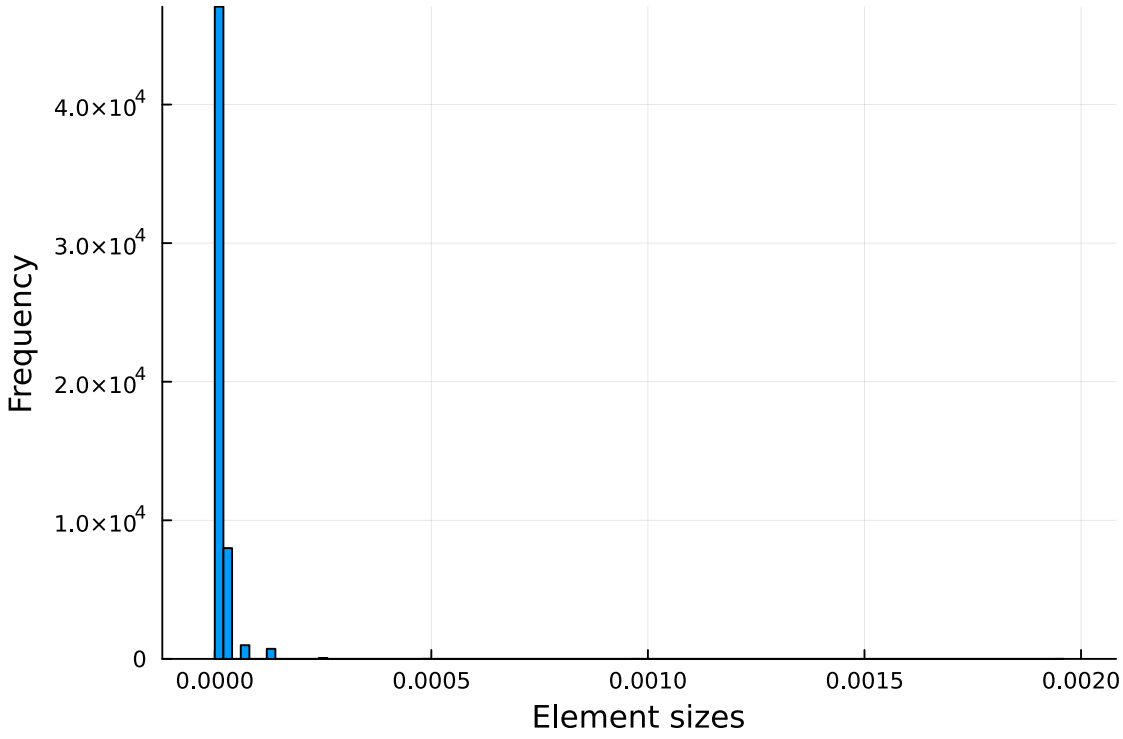
```
M = 3
L = 1
xc = collect(range(0, L, M))
EToVc = convert(Matrix{UInt64}, hcat(1:M-1, 2:M))
tolerance = 10^(-9)

mean_errors = []
point_counts = []

iterations = 0

while true
    idxMarked = collect(1:length(xc)-1)
    EToVf, xf = refine_marked(EToVc, xc, idxMarked)

    uc = BVP1Drhs(c, d, xc, f)
    uf = BVP1Drhs(c, d, xf, f)

    err = errorestimate(xc, xf, uc, uf, EToVc, EToVf, idxMarked)
    push!(mean_errors, maximum(err))
    push!(point_counts, length(xc))

    idxMarked = findall(err .>= tolerance)
    if isempty(idxMarked)
        break
    end

    EToVc, xc = refine_marked(EToVc, xc, idxMarked)
    iterations += 1
end

display(iterations)
```

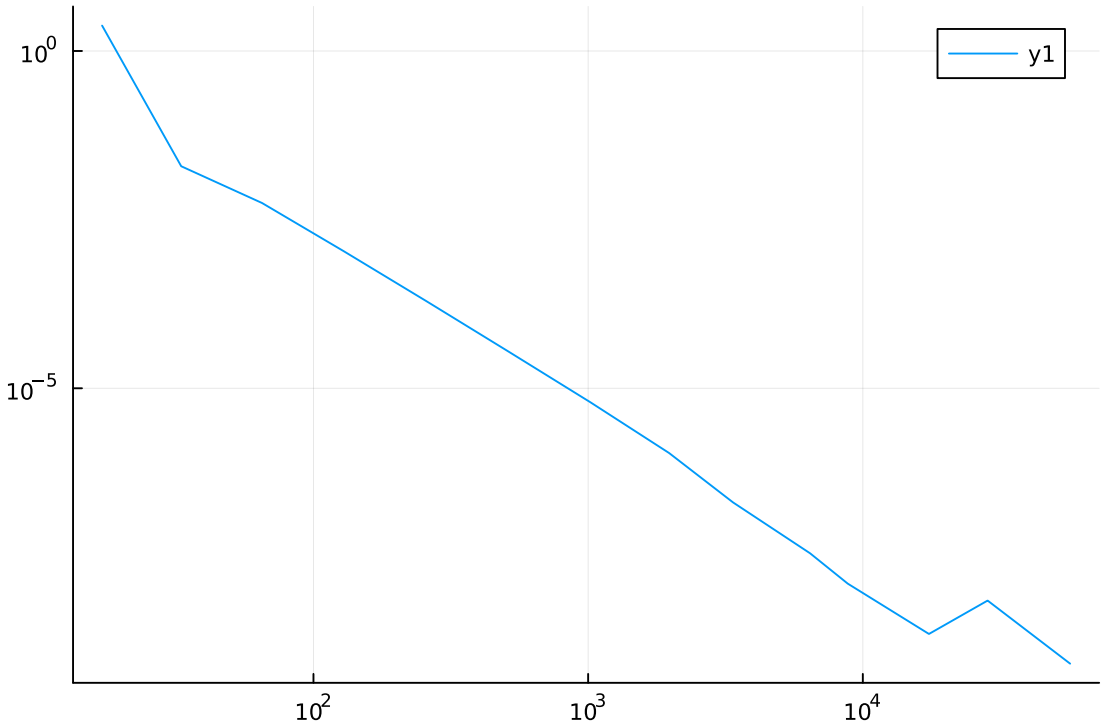
```
display(length(xc))

uc = BVP1Drhs(c, d, xc, f)
p = plot(x, uanal.(x))
plot!(x, uhat(x, xc, uc))
display(p)

uanal(x) = exp(-800*(x - 0.4)^2) + 0.25*exp(-40*(x - 0.8)^2)
x = collect(range(0, 1, 10000))
p = plot(x, uanal.(x))
vline!(xc)
display(p)

hs = xc[2:end] - xc[1:end-1]
display(histogram(hs, bins=100, xlabel="Element sizes", ylabel="Frequency"

display(plot(point_counts[4:end], mean_errors[4:end], xscale=:log10, ysca
fit(log.(point_counts[4:end]), log.(mean_errors[4:end]), 1)
```

```
16
56859
```




1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
y1
y2


4.0×10^4
3.0×10^4
2.0×10^4
1.0×10^4
0
Frequency
0.0000
0.0005
0.0010
0.0015
0.0020
Element sizes



Out[22]: 5.863976182409583 - 2.555534366585035·x

In [ ]: