Exercise 1. (a) Write a program to solve the boundary value problem for the nonlinear pendulum as discussed in the text. See if you can find yet another solution for the boundary conditions illustrated in Figures 2.4 and 2.5.

(b) Find a numerical solution to this BVP with the same general behavior as seen in Figure 2.5 for the case of a longer time interval, say, $T = 20$, again with $\alpha = \beta = 0.7$. Try larger values of T. What does $\max_i \theta_i$ approach as T is increased? Note that for large T this solution exhibits "boundary layers".

Solution 1. (a) We solve the BVP for the non-linear pendulum. Trying a random sampling of 20 initial guess, we arrive at the three solutions pictured below.
(b) Tweaking the grid size and T, we arrive at several solutions which appear to increase in height. From this, I would guess that $\max_i \theta_i$ approaches infinity as T increases. Plots are embedded below.

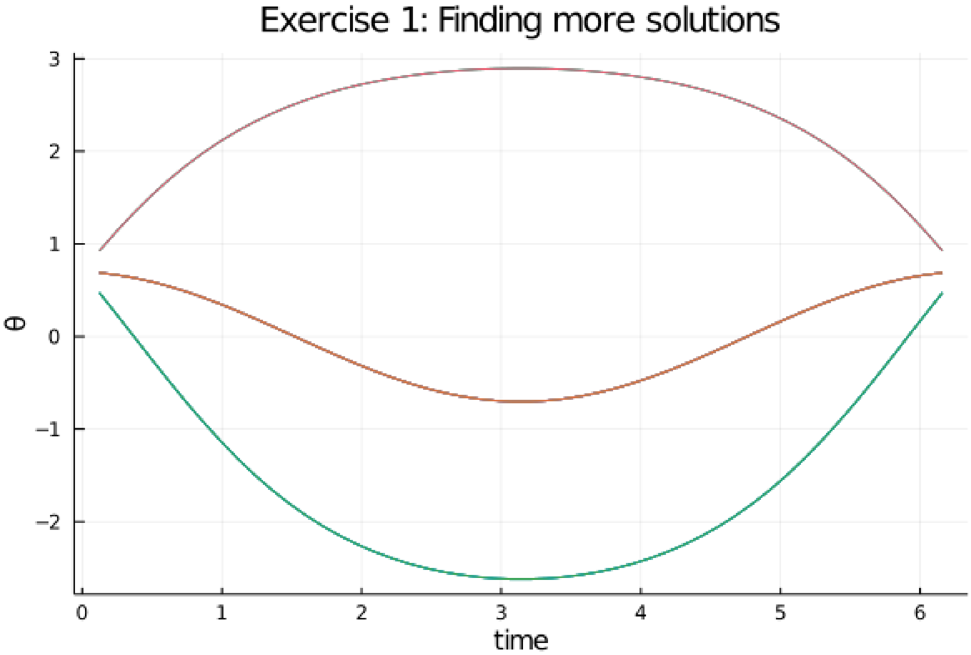


Figure 1: Finidng more solutions with random sampling of initial guesses.

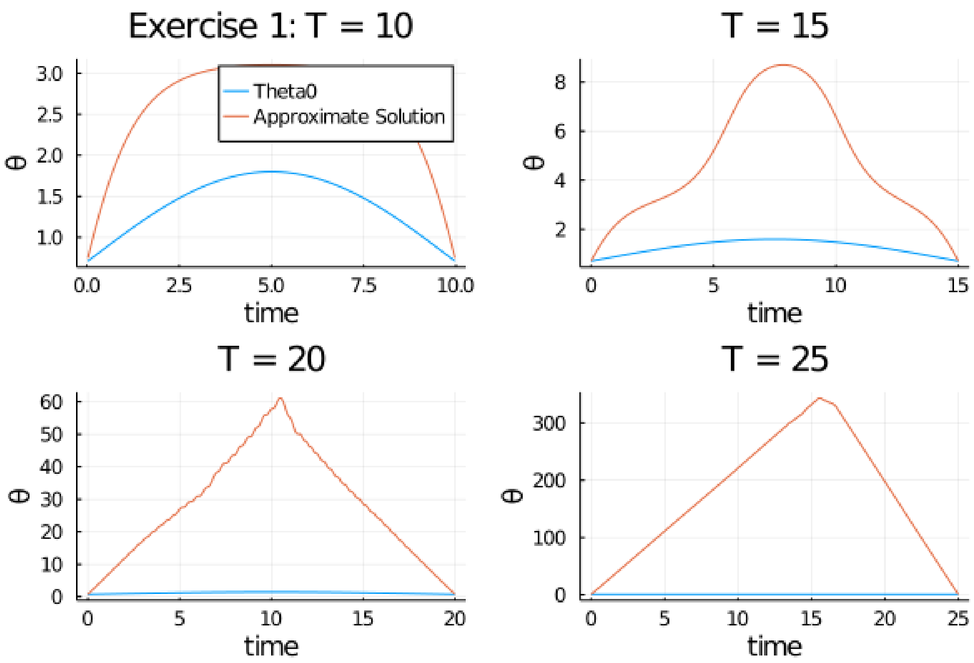


Figure 2: Solutions of the pendulum problem as T increases.

Exercise 2 (Gerschgorin's theorem and stability)**.** Consider the boundary value problem

$$\begin{aligned} -u_{xx} + (1+x^2)u = f, 0 \leq x \leq 1, \\ u(0) = 0, u(1) = 0. \end{aligned}$$

On a uniform grid with spacing $h = \frac{1}{m+1}$, the following set of difference equations has local truncation error $O(h^2)$:

$$\frac{2u_i - u_{i+1} - u_{i-1}}{h^2} + (1+x_i^2)u_i = f(x_i), i = 1, \ldots, m.$$

(a) Use Gerschgorin's theorem to determine upper and lower bounds on the eigenvalues of the coefficient matrix for this set of difference equations. (b) Show that the L_2-norm of the global error is of the same order as the local truncation error.

Solution 2. Using Gerschgorin's theorem, we bound the eigenvalues of the coefficient matrix A as follows. Each row has entries of the form

$$\begin{aligned} A_{i-1,i} &= -\frac{1}{h^2} \\ A_{i,i} &= \frac{2}{h^2} + x_i^2 + 1 \\ A_{i,i+1} &= -\frac{1}{h^2}, \end{aligned}$$

so that the eigenvalues live in disks of the form

$$D_i = \left\{ z \in \mathbb{C} \colon \left| z - \left(\frac{2}{h^2} + x_i^2 + 1 \right) \right| \leq \frac{2}{h^2} \right\}.$$

We can simplify this inequality for any i as follows

$$1 + x_i^2 \leq \lambda \leq \frac{4}{h^2} + 1 + x_i^2.$$

With the knowledge that $x_i \in [0,1]$, we have that any eigenvalue has

$$1 \leq \lambda \leq 2 + \frac{4}{h^2}.$$

Therefore, we have that

$$\left|\left|A^{-1}\right|\right|_2 = \left(\min_i |\lambda_i| \right)^{-1} \approx 1$$

where λ_i are the eigenvalues of A. As we can see this norm does not depend on h due to the minimum we've derived above. Therefore, we have that the L_2 norm of the global error is bounded by

$$||E||_2 \leq \left|\left|A^{-1}\right|\right|_2 ||\tau||_2 \approx ||\tau||_2$$

i.e. the L_2 norm of the global error is on the same order as the local truncation error.

Exercise 3 (Richardson extrapolation)**.** Use your code from problem 6 of assignment 1 to do the follow exercise. Run the code with $h = .1$ (10 subintervals) and with $h = 0.05$ (20 subintervals) and apply Richardson extrapolation to obtain more accurate solution values on the coarser grid. Record the L_2-norm or the ∞-norm of the error in the approximation obtained with each h value and in that obtained with extrapolation.
Suppose that you assume the coarse grid approximation is piecewise linear, so that the approximation at the midpoint of each subinterval is the average of the values at the two endpoints. Can one use Richardson extrapolation with the fine grid approximation and these interpolated values on the coarse grid to obtain a more accurate approximation at these points? Explain why or why not?

Solution 3. Using the L_2 norm, we see that the L_2 error for $h = .1$ is 2.0×10^{-3}, $h = .2$ is 5×10^{-5}, and the error is 9.5×10^{-6} for the Richardson extrapolation.
I'm a bit unclear on what this is asking, so I'll rephrase the question a bit and answer it as I understand it. If the question is asking whether it is possible to compute linearly interpolated values between elements on a coarse grid and combine these interpolated values along with the value of the approximation on a finer grid to get a better approximation at an element on the fine grid, I would say yes.
Suppose that we have a function f which is being linearly interpolated between points x_0 and $x_2 = x_0 + 2h$. Then we have that the error on this interval can be written as

$$E(x) = f(x) - \left(f(x_0) + \frac{f(x_2) - f(x_0)}{x_2 - x_0}(x - x_0) \right).$$

Expanding $f(x)$ around $x_1 = x_0 + h$, we see that this is on the order

$$\begin{aligned} E(x_1) &\approx f'(x_1)h - \frac{f(x_2) - f(x_0)}{x_2 - x_0}h + O(h^2) \\ &= \left(f'(x_1) - \frac{f(x_1 + h) - f(x_1 - h)}{2h} \right) h + O(h^2) \end{aligned}$$

Noting the centered difference approximation of the first derivative is on the order of $O(h^2)$, we see that the error of the approximation is on the order of $O(h^2)$. From this argument, I believe we could combine the above estimates of x_1 using h and $h/2$ in a Richardson approximation to approximate x_1 at a higher order. Though this is still distinct from what has been asked.
As we've seen the linear interpolation at the midpoint on the coarse grid should be on the order of $O(h^2)$ which is the same as the order of the value computed on finer grids method. Specifically using Richardson extrapolation, this doesn't fit because it relies on using the fact that we are using the same general method to generate our approximations I do think it may be possible to make another approximation though since the methods have the same order accuracy by taking something like

$$\frac{\lambda_1 A(h/2) - \lambda_2 B(h)}{\lambda_3},$$

where $A(h/2)$ denotes the approximation on the finer grid and $B(h)$ is the linear interpolation and the λ are picked to cancel out the coefficient of order $O(h^2)$.

Exercise 4. Write down the Jacobian matrix associated with Example 2.2 and the nonlinear difference equations (2.106) on p. 49. Write a code to solve these difference equations when $a = 0$,$b = 1$, $\alpha = -1$,$\beta = 1.5$,and $\epsilon = 0.01$. Use an initial guess of the sort suggested in the text. Try $h = 1/20$, $h = 1/40$, $h = 1/80$,and $h = 1/160$, and turn in a plot of your results.

Solution 4. Computing the Jacobian associated with example 2.2, we get

$$\begin{aligned}
\frac{\partial G_i(U)}{\partial U_{i-1}} &= \frac{\epsilon}{h^2} - \frac{U_i}{2h} \\
\frac{\partial G_i(U)}{\partial U_i} &= -\frac{2\epsilon}{h^2} + \left(\frac{U_{i+1} - U_{i-1}}{2h} - 1\right) \\
\frac{\partial G_i(U)}{\partial U_{i+1}} &= \frac{\epsilon}{h^2} + \frac{U_i}{2h}
\end{aligned}$$

Implementing this in julia for various step sizes, we get the below figure. Notice the oscillatory behavior near the interior layer. Steps sizes less than 0.05 appear to remedy this behavior which is consistent with the estimate described by (2.108).

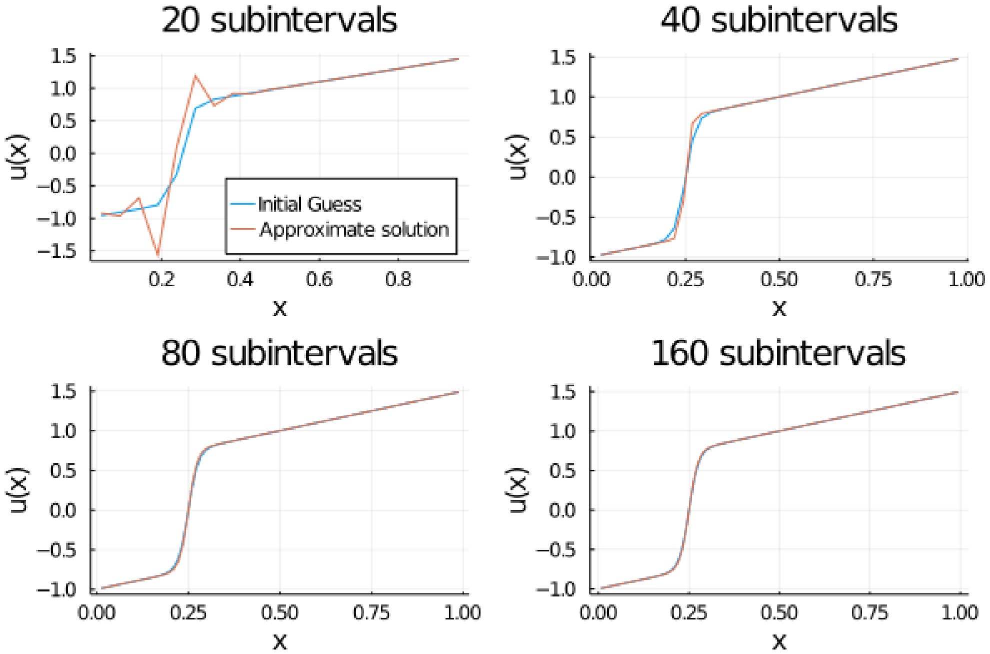


Figure 3: Solution of difference equations from exercise 4 using Newton's method.

HW-3-Code-Figgins

February 1, 2021

```
[1]: using Plots, LinearAlgebra
```

0.1 Exercise 1

Write a program to solve the boundary value problem for the nonlinear pendulum as discussed in the text. See if you can find yet another solution for the boundary conditions illustrated in Figures 2.4 and 2.5.

```
[13]: function nl_pendulum( ,  , T, m,  0; max_iter = 4)
          h = T / (m + 1)

            =  0
          for iter in 1:max_iter
               += J_nl_pendulum( ,h) \ (-1 * G_nl_pendulum( , h,  ,  ))
          end

          return
      end

      function J_nl_pendulum( , h)
          # Compute Jacobian for discretized non-linear pendulum
          m = length( )
          J = (1/h^2) * Tridiagonal( ones(m-1),
                          [-2 + h^2 * cos( _i) for  _i in  ],
                          ones(m-1))
          return J
      end

      function G_nl_pendulum( , h,  ,  )
          # Computes discretized pendulum RHS
          # u'' + sin = 0

          m = length( )
          G = zeros(m)

          # Using intitial condition theta_0 = alpha
          G[1] = (1/h^2)*(  - 2* [1] +  [2]) + sin( [1])
```

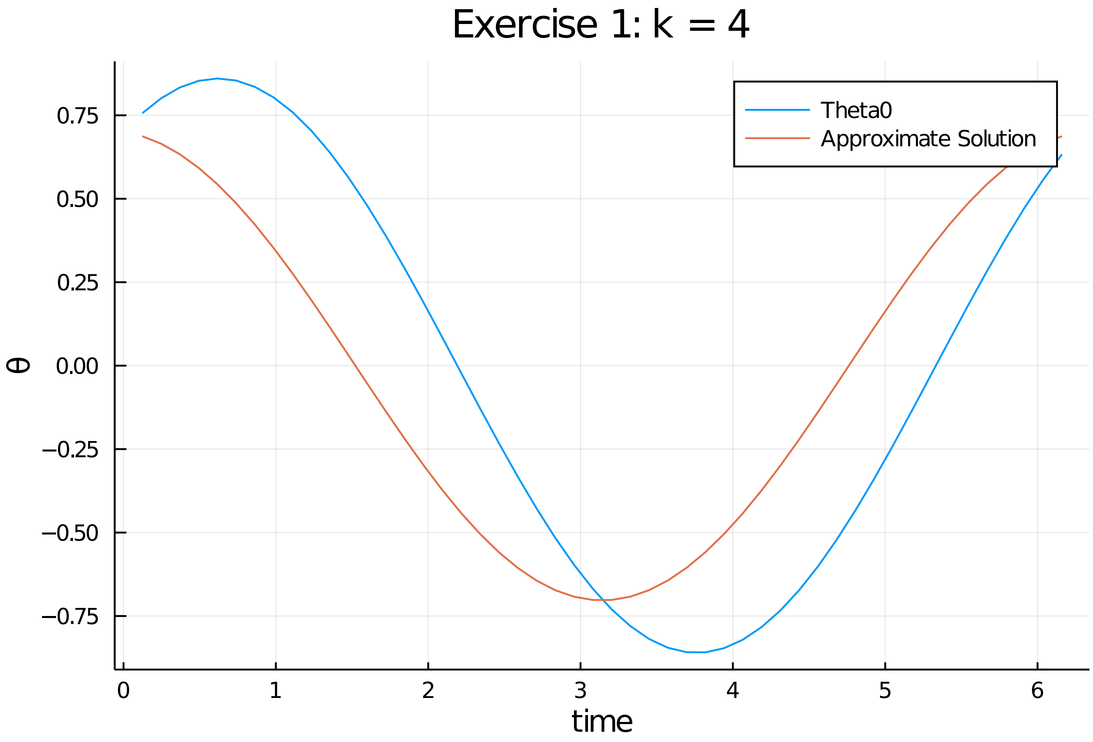
```
    # Discretized equations
    for i in 2:(m-1)
        G[i] = (1/h^2)*( [i-1] - 2* [i] +  [i+1]) + sin( [i])
    end

    # Using intitial condtion theta_(m+1) = beta
    G[m] = (1/h^2)*( [m-1] - 2* [m] +  ) + sin( [m])

    return G
end
```

[13]: G_nl_pendulum (generic function with 1 method)

[14]:

```
m = 50
T = 2
  = 0.7
  = 0.7
t = [T*i/(m + 1) for i in 1:m]
theta0 = [0.7 * cos(ti) + 0.5 * sin(ti) for ti in t]

test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 4)

plot(t, theta0,
    label = "Theta0",
    ylabel = " ",
    xlabel = "time")
plot!(t, test_sol,
    label = "Approximate Solution",
    title = "Exercise 1: k = 4")
```

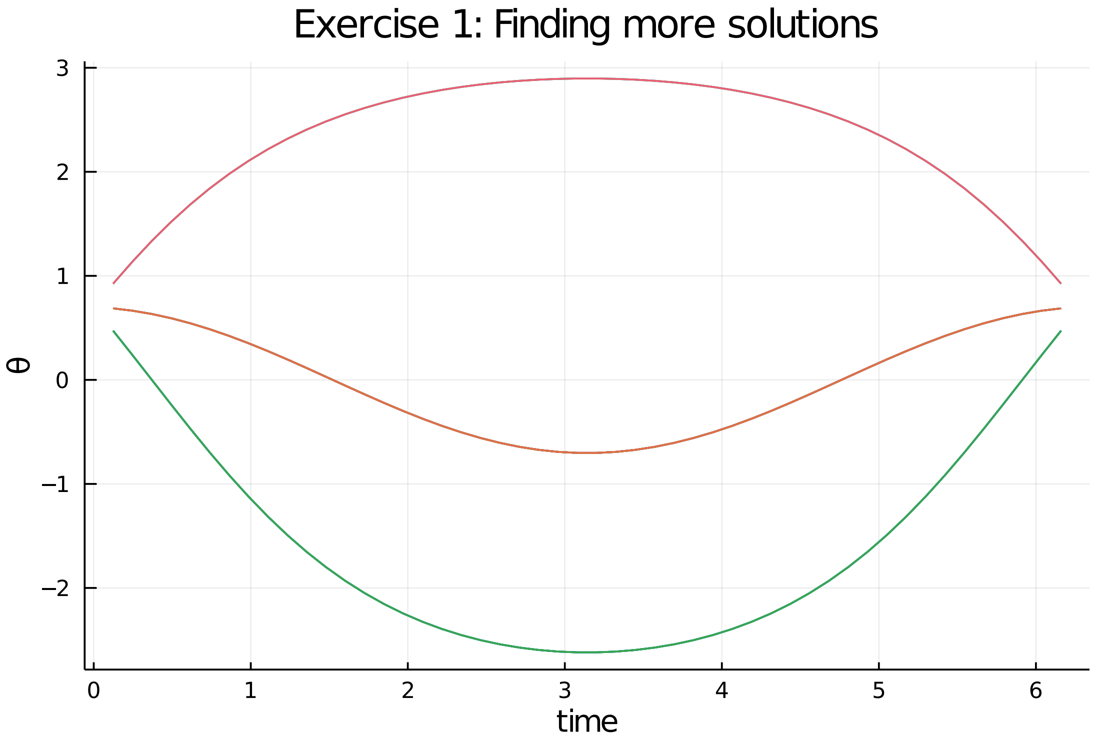
[14]:



Trying different initial values to search for more solutions. Random sampling of coefficients for sin and cosine,

```
[16]: theta0 = randn() .+ [sin(4*ti) for ti in t]
      test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 20)

      plot(t, test_sol,
          label = false,
          title = "Exercise 1: Finding more solutions",
          ylabel = " ",
          xlabel = "time")

      # ADDING A COUPLE MORE GUESSES
      for i in 2:19
          theta0 = randn() .+ [sin(4*ti) for ti in t]
          test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 30)
          plot!(t, test_sol,
              label = false)
      end

      plot!()
```

[16]:



[17]:

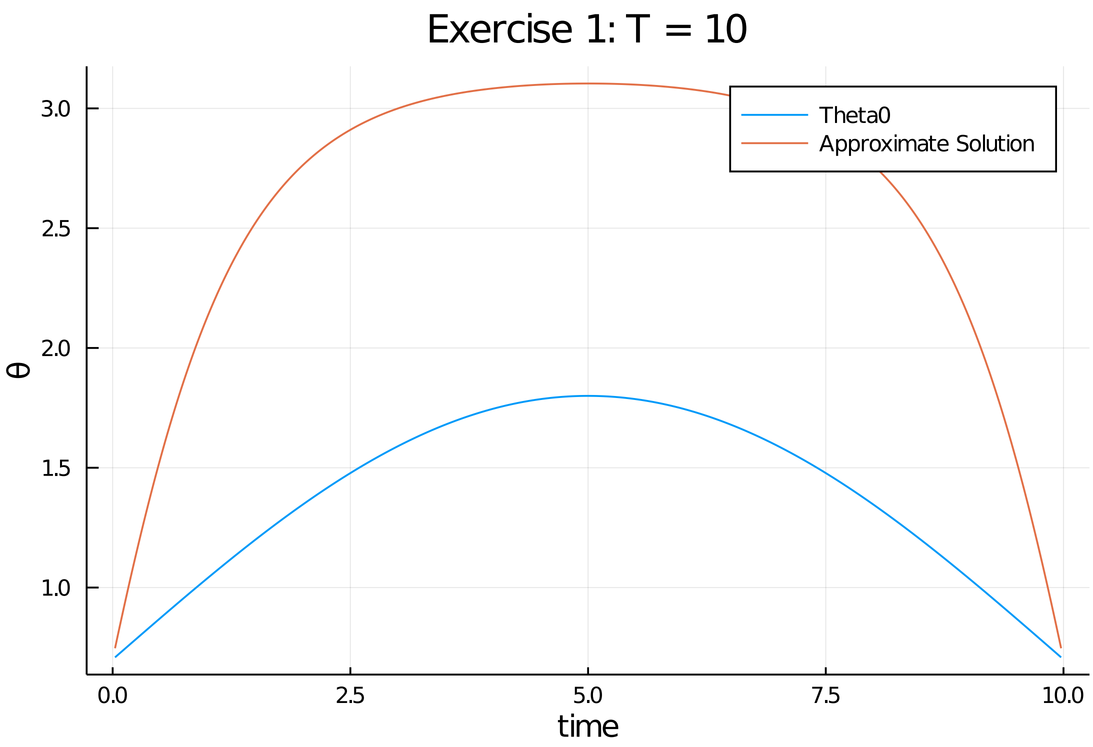
```
savefig("../hw/figs/hw-3-exer1-more-sol.png")
```

(b) Find a numerical solution to this BVP with the same general behavior as seen in Figure 2.5 for the case of a longer time interval, say, $T = 20$, again with $\alpha = \beta = 0.7$. Try larger values of T. What does $\max_i \theta_i$ approach as T is increased? Note that for large T this solution exhibits "boundary layers".

[18]:

```
m = 400
T = 10
t = [T*i/(m + 1) for i in 1:m]

theta0 = [0.7 + 1.1 * sin( *ti/T) for ti in t]

test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 300)

plot(t, theta0,
    label = "Theta0",
    ylabel = " ",
    xlabel = "time")
exerT10 = plot!(t, test_sol,
    label = "Approximate Solution",
    title = "Exercise 1: T = $T")
```

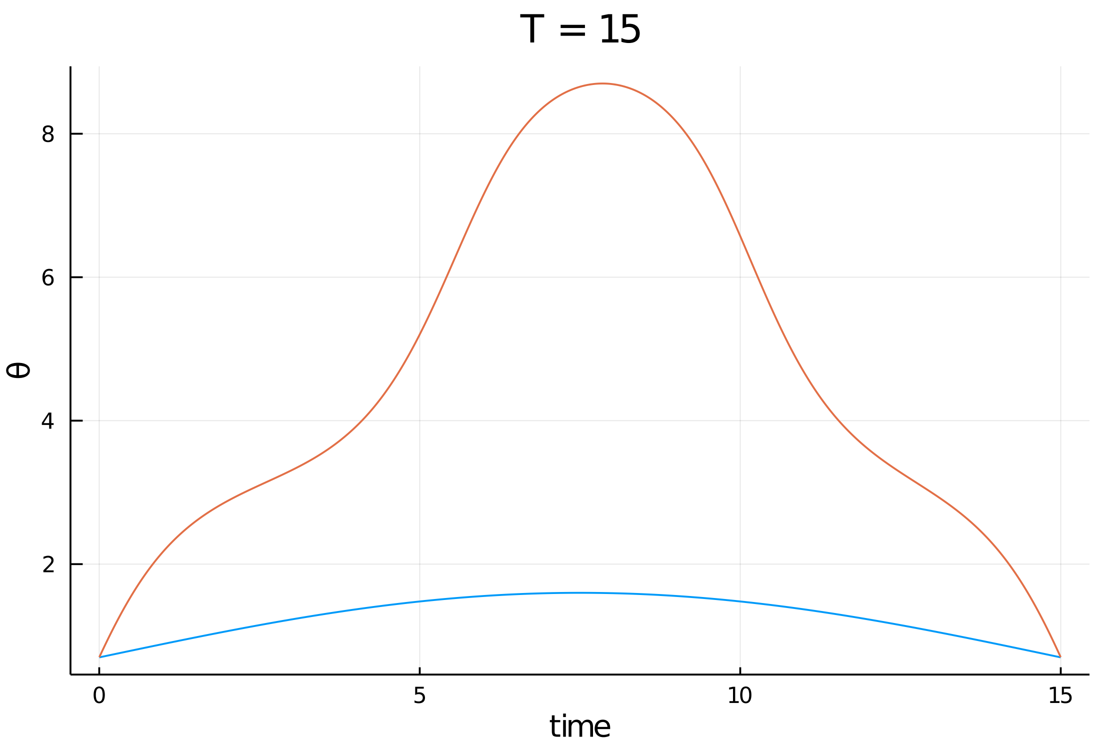
[18]:



[19]:

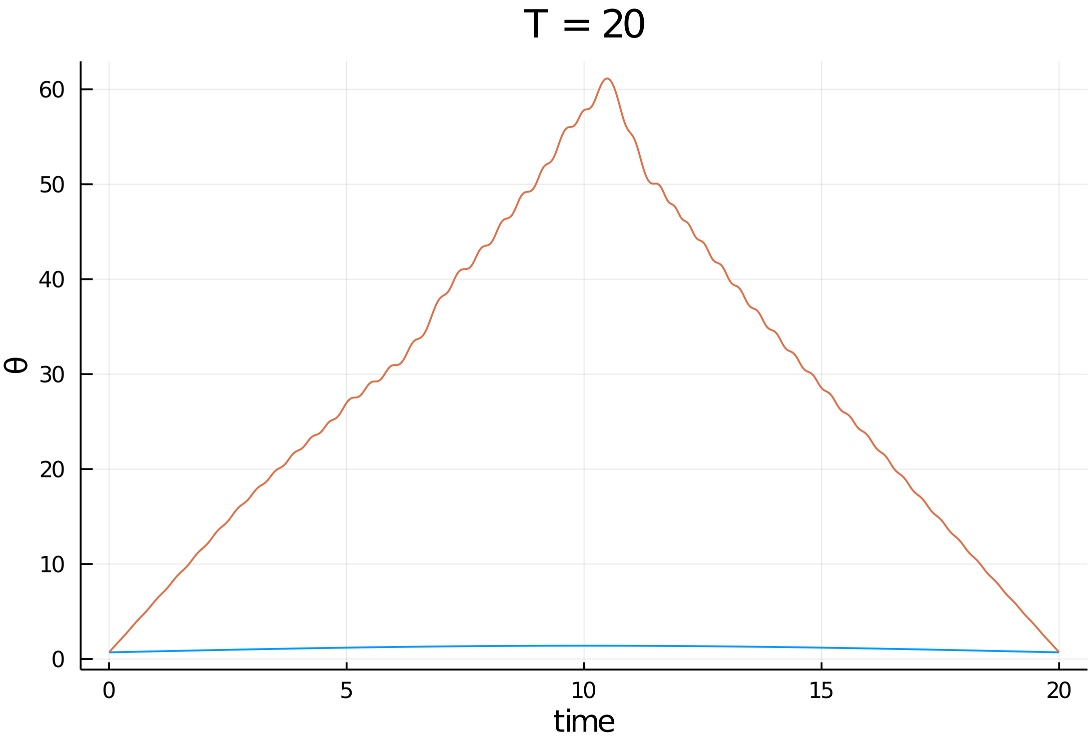
```
m = 10000
T = 15
t = [T*i/(m + 1) for i in 1:m]

theta0 = [0.7 + 0.9 * sin( *ti/T) for ti in t]

test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 300)

plot(t, theta0,
    label = false,
    ylabel = " ",
    xlabel = "time")
exerT15 = plot!(t, test_sol,
    label = false,
    title = "T = $T")
```

[19]:



```
[23]: m = 10000
      T = 20
      t = [T*i/(m + 1) for i in 1:m]

      theta0 = [0.7 + 0.7 * sin( *ti/T) for ti in t]

      test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 300)

      plot(t, theta0,
          label = false,
          ylabel = " ",
          xlabel = "time")
      exerT20 = plot!(t, test_sol,
          label = false,
          title = "T = $T")
```

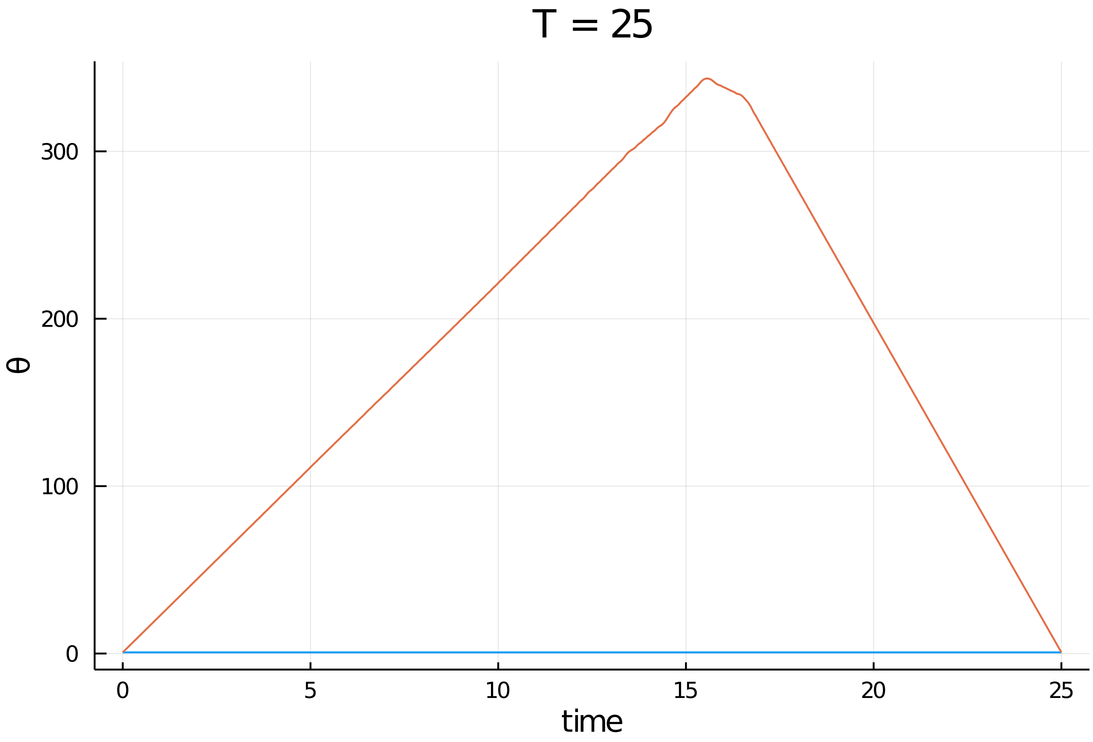
[23]:



[24]:

```
m = 30000
T = 25
t = [T*i/(m + 1) for i in 1:m]

theta0 = [0.7 + 0.0 * sin( *ti/T) for ti in t]

test_sol = nl_pendulum( ,  , T, m, theta0; max_iter = 300)

plot(t, theta0,
    label = false,
    ylabel = " ",
    xlabel = "time")
exerT25 = plot!(t, test_sol,
    label = false,
    title = "T = $T")
```

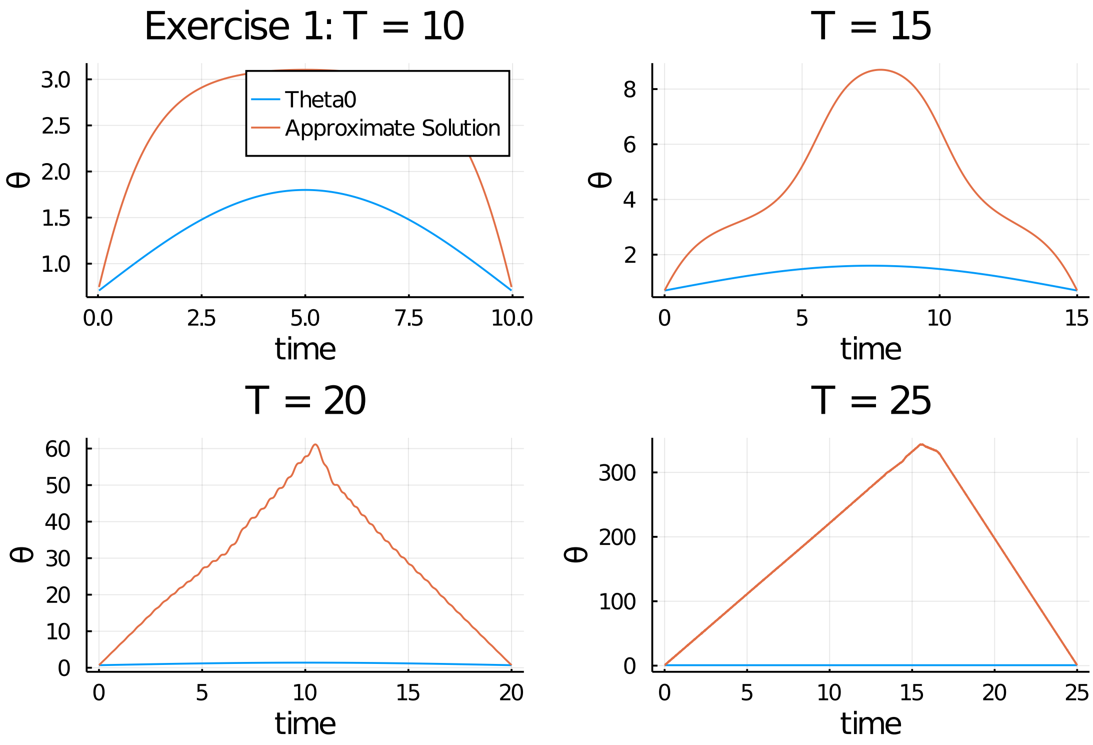
[24]:



[25]:

```
exer1_plot = plot(exerT10, exerT15, exerT20, exerT25)
```

[25]:



[26]:
```
savefig("../hw/figs/hw-3-max-theta.png")
```

From these, it appears that $\max \theta_i$ approaches infinity as T increases.

0.2 Exercise 3

[27]:
```
function make_matrix(M)
   # Step size
   h = 1/M

    # Xjs of interest
    xjs = [j*h for j in 1:M]

    # Defining diagonal
    diagA = [ -(1 + (xj + h/2)^2) - (1 + (xj - h/2 )^2)  for xj in xjs]
    diagB = [  (1 + (xj + h/2)^2)  for xj in xjs[1:M-1]]

    ##Building Matrix
    A = Array(Tridiagonal(diagB, diagA, diagB))
    A[end, [M, M-1, M-2]] .= 3*h/2, -2*h, h/2
    A = A ./ h^2

    return A
end

# Define test function
u(x) = (1 - x)^2

# Define true solution for test function
true_f(x) = 2*(3*x^2 - 2*x + 1)
```

[27]:
```
true_f (generic function with 1 method)
```

[28]:
```
function get_u_approx(f, M)
    h = 1/M

    F = [f(i*h) for i in 1:M]
    F[1] -= 1/h^2
    F[end] = 0

    U = make_matrix(M) \ F
end

M = 10
U = get_u_approx(true_f, M)
```

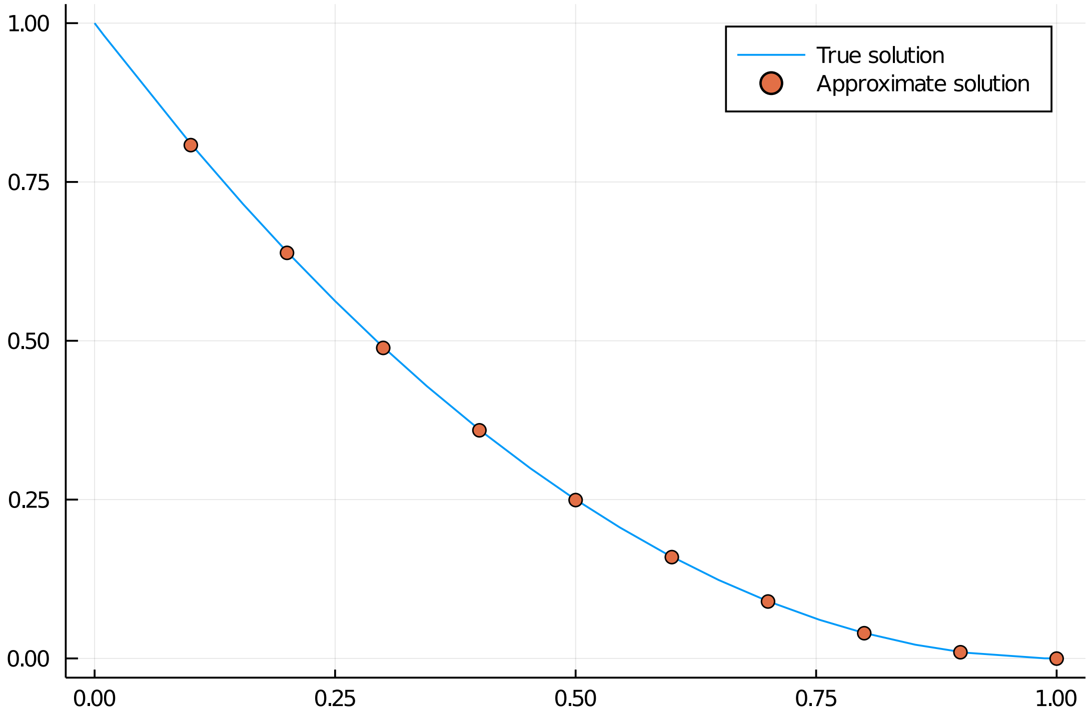
[28]:

```
10-element Array{Float64,1}:
  0.8079841743268176
  0.6384038660262153
  0.4887606987439885
  0.35905391456816127
  0.24928604326889642
  0.15946196247245417
  0.089587892006819
  0.03967053825490476
  0.009716480061949168
 -0.00026820600236936225
```

[29]:

```
plot(u, 0, 1,
    label = "True solution")
scatter!([i/M for i in 1:M], U,
    label = "Approximate solution")
```

[29]:



[30]:

```
# Applying Richardson Extrapolation
M = 10

U10 = get_u_approx(true_f, M)
U20 = get_u_approx(true_f, 2*M)
```

```
RE = similar(U10)
for i in 1:length(U10)
    # Same grid point is at 2i in finer grid/
   RE[i] = (4*U20[2*i] - U10[i])/(4 - 1)
end
```

[31]:
```
function get_infinity_error(uhat)
    M = length(uhat)

    # Discretize True Solution
    xjs = [j/M for j in 1:M]
    true_u_val = u.(xjs)

    # Infinity norm
    return maximum( abs.(true_u_val .- uhat) )
end

function get_L2_error(uhat)
    M = length(uhat)

    # Discretize True Solution
    xjs = [j/M for j in 1:M]
    true_u_val = u.(xjs)

    # Infinity norm
    return sqrt(sum((1/M)*(true_u_val .- uhat).^2))
end
```

[31]:
```
get_L2_error (generic function with 1 method)
```

[32]:
```
U10_error = get_infinity_error(U10)
```

[32]:
```
0.002015825673182481
```

[33]:
```
U20_error = get_infinity_error(U20)
```

[33]:
```
0.0005635871159848094
```

[34]:
```
RE_error = get_infinity_error(RE)
```

[34]:
```
9.544184155884283e-6
```

[390]:
```
function guess_f(u, M)
    h = 1 / M

    # Discretize u
    xjs = [j*h for j in 1:M]
    U = u.(xjs)

    # Approximate the F vector
    F_approx = make_matrix(M)*U

    # End points will be funky due to boundary conditions

    ## First end point has extra term
    F_approx[1] += (1+ (h/2)^2)/h^2

    ## Last is a derivative and isn't needed
    return F_approx[1:(M-1)]
end
```

[390]: guess_f (generic function with 1 method)

0.3 Exercise 4

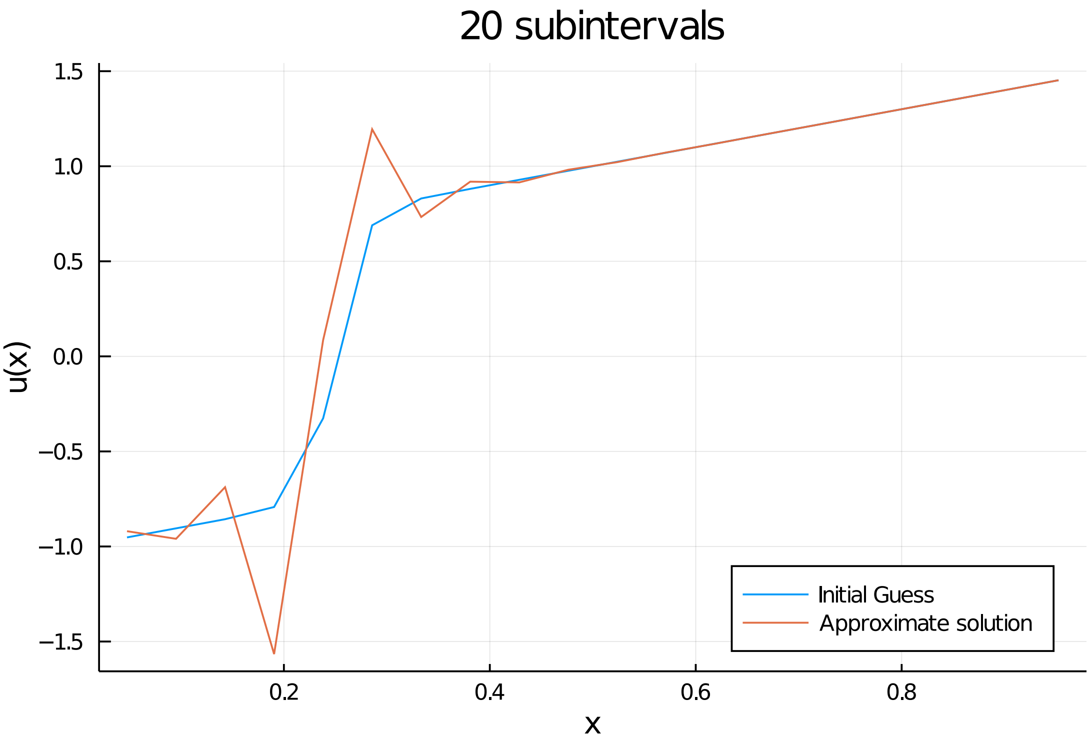
[35]:
```
function nl_exer_4_nm( ,  , a, b,  , m, U0; max_iter = 1)
    h = (b - a) / (m + 1)

    U = U0
    for iter in 1:max_iter
        U += J_exer_4(U, h,  ,  , ) \ (-1 * G_exer_4(U, h,  ,  , ))
    end

    return U
end

function J_exer_4(U, h,  ,  , )
    # Compute Jacobian for discretized
    m = length(U)

    # Upper Off-Diagonal
    JUpper = [ /h^2 + U[i]/(2*h) for i in 1:(m-1)]

    JDiag = zeros(m)

    # Diagonal
    JDiag[1] = -2* /h^2 + (U[2] -  )/(2*h) - 1
    JDiag[2:(m-1)] .= [-2* /h^2 + (U[i+1] - U[i-1])/(2*h) - 1 for i in 2:(m-1)]
    JDiag[m] = -2* /h^2 + (  - U[m-1])/(2*h) - 1

    # Lower Off-Diagonal
    JLower = [ /h^2 - U[i]/(2*h) for i in 2:m]

    return Tridiagonal(JLower, JDiag, JUpper)
```

```
end

function G_exer_4(U, h,  ,  , )
    # Computes discretized ...

    m = length(U)
    G = zeros(m)

    # Using intitial condition U_0 = alpha
    G[1] =  *( - 2*U[1] + U[2])/(h^2) + U[1]*((U[2] -  )/(2*h) - 1)

    # Discretized equations
    for i in 2:(m-1)
        G[i] =  *(U[i-1] - 2*U[i] + U[i+1])/(h^2) + U[i]*((U[i+1] - U[i-1])/
 ↪(2*h) - 1)
    end

    # Using intitial condtion U_(m+1) = beta
    G[m] =  *(U[m-1] - 2*U[m] +  )/(h^2) + U[m]*(( - U[m-1])/(2*h) - 1)

    return G
end
```

[35]: G_exer_4 (generic function with 1 method)

[36]:
```
a = 0
b = 1
  = -1
  = 1.5
  = 0.01
m = 20

x = [(b-a)*i/(m+1) for i in 1:m]
xbar = (1/2)*(a + b -   - )
w0 = (1/2)*(a - b +   - )

U0 = [xi - xbar + w0*tanh(w0*(xi - xbar)/(2* )) for xi in x]
test_sol = nl_exer_4_nm( ,  , a, b,  , m, U0; max_iter = 10)

plot(x, U0,
    label = "Initial Guess",
    xlabel = "x",
    ylabel = "u(x)",
    title = "$m subintervals",
    legend = :bottomright)
exer4_20 = plot!(x, test_sol,
    label = "Approximate solution")
```

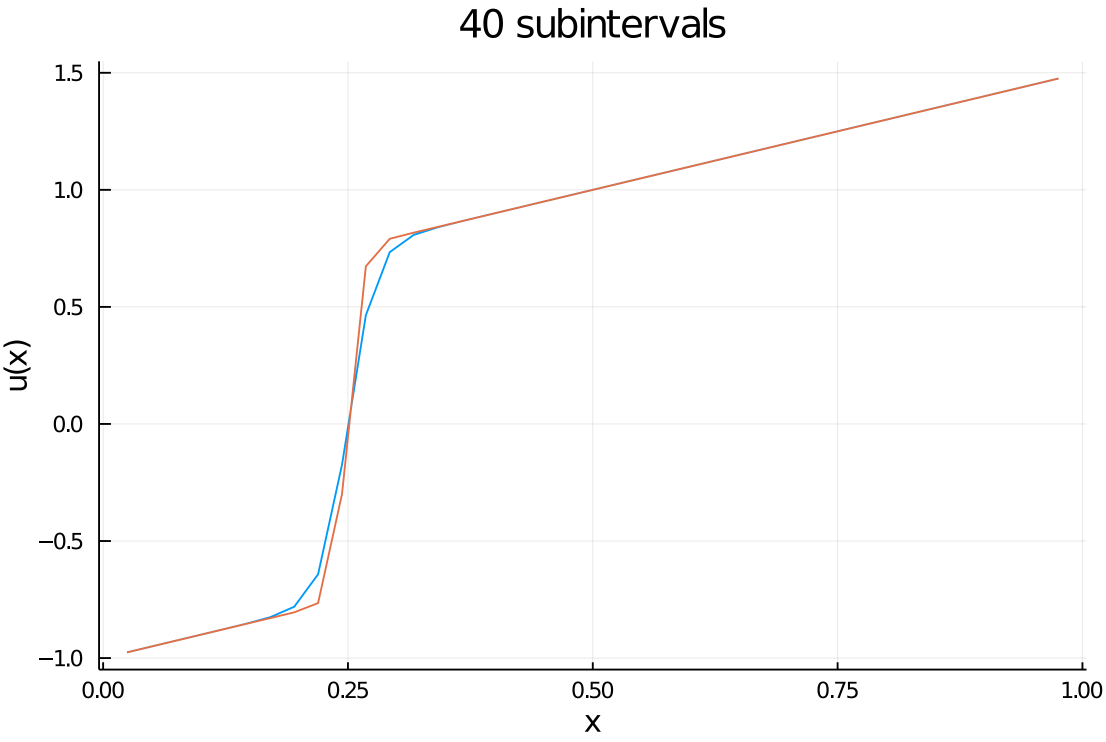
[36]:



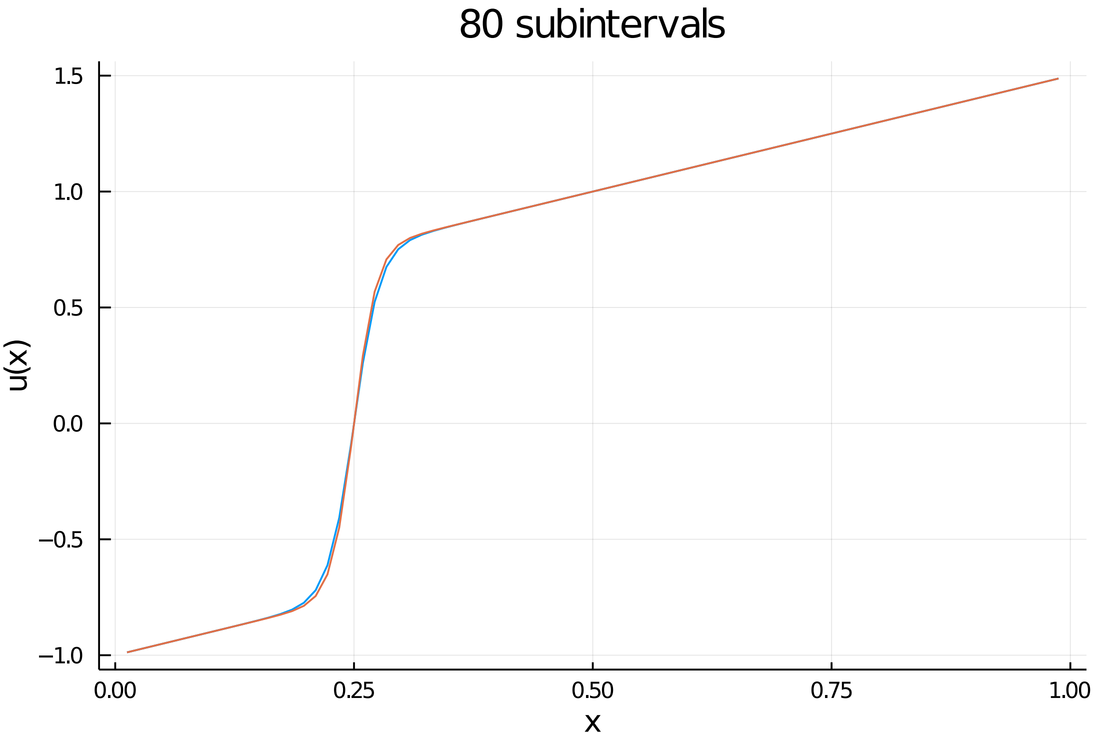
[37]:

```
m = 40

x = [(b-a)*i/(m+1) for i in 1:m]
xbar = (1/2)*(a + b -   -  )
w0 = (1/2)*(a - b +   -  )

U0 = [xi - xbar + w0*tanh(w0*(xi - xbar)/(2* )) for xi in x]
test_sol = nl_exer_4_nm( ,  , a, b,  , m, U0; max_iter = 10)

plot(x, U0,
    label = false,
    xlabel = "x",
    ylabel = "u(x)",
    title = "$m subintervals")
exer4_40 = plot!(x, test_sol,
    label = false)
```

[37]:



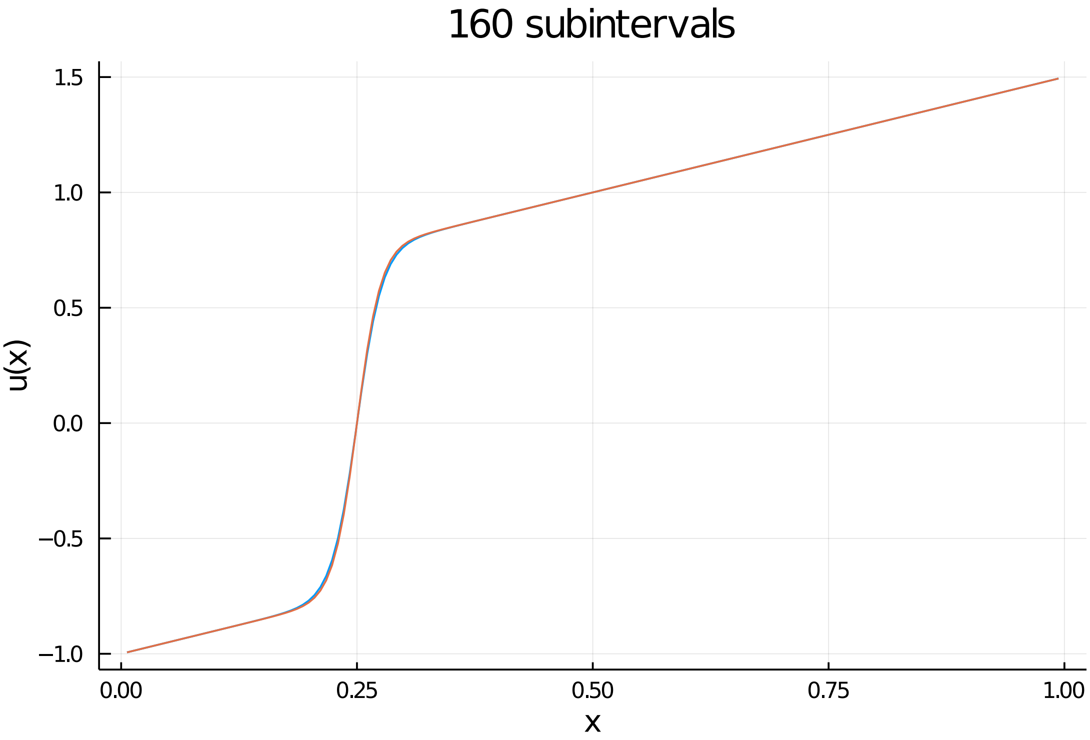
```
[38]: m = 80

      x = [(b-a)*i/(m+1) for i in 1:m]
      xbar = (1/2)*(a + b -   -  )
      w0 = (1/2)*(a - b +   -  )

      U0 = [xi - xbar + w0*tanh(w0*(xi - xbar)/(2* )) for xi in x]
      test_sol = nl_exer_4_nm( ,  , a, b,  , m, U0; max_iter = 10)

      plot(x, U0,
          label = false,
          xlabel = "x",
          ylabel = "u(x)",
          title = "$m subintervals")
      exer4_80 = plot!(x, test_sol,
          label = false)
[38]:
```



[39]:

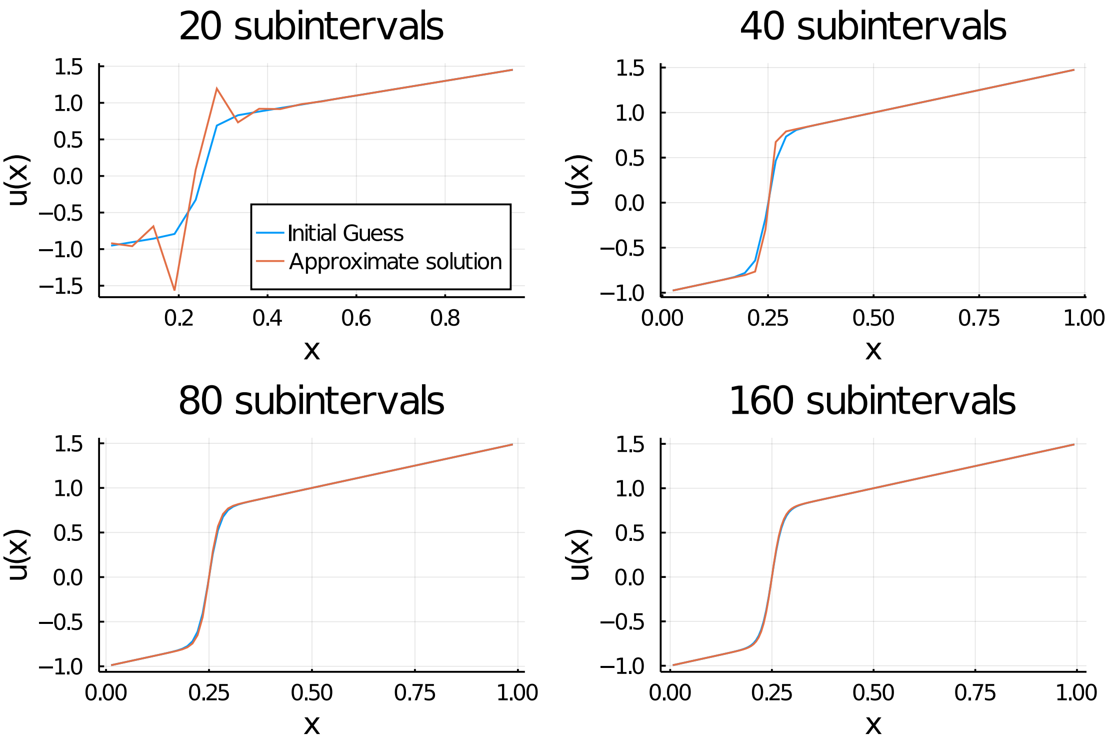
```
m = 160

x = [(b-a)*i/(m+1) for i in 1:m]
xbar = (1/2)*(a + b -   -  )
w0 = (1/2)*(a - b +   -  )

U0 = [xi - xbar + w0*tanh(w0*(xi - xbar)/(2* )) for xi in x]
test_sol = nl_exer_4_nm( ,  , a, b,  , m, U0; max_iter = 10)

plot(x, U0,
    label = false,
    xlabel = "x",
    ylabel = "u(x)",
    title = "$m subintervals")
exer4_160 = plot!(x, test_sol,
    label = false)
```

[39]:



[40]:
```
exer4_plot = plot( exer4_20, exer4_40, exer4_80, exer4_160)
```

[40]:



[41]:

```
savefig("../hw/figs/hw-3-exer-4.png")
```