Title: Computational Physics - Lecture 18

Date: Nov 09, 2018 01:00 PM

URL: http://pirsa.org/18110044

Abstract:

Finite Differencing

Setup

In [1]:
```
pwd()
```

Out[1]:
```
"/Users/eschnett/txt/Courses/CompPhys2018/jl/FiniteDifferencing"
```

In [2]:
```
]activate .
```

In [3]:
```
]add Plots
```

```
  Updating registry at `~/.julia/registries/General`
  Updating git-repo `https://github.com/JuliaRegistries/General.git`
 Resolving package versions...
  Updating
`~/txt/Courses/CompPhys2018/jl/FiniteDifferencing/Project.toml`
 [no changes]
  Updating
`~/txt/Courses/CompPhys2018/jl/FiniteDifferencing/Manifest.toml`
 [no changes]
```

Piecewise Linear Continuous Functions

We approximate functions by a piecewise linear continuous approximation, with a regular grid spacing.

In [6]:
```
# A piecewise linear continuous function mapping a type T to a type U
struct PLCFun{T,U}
    # Domain: [0; 1]
    points::Vector{U}
end
```

In [7]:
```
# Functions can be scaled and added
function Base. *(a::U, f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    PLCFun{T, U}(a .* f.points)
end
function Base. +(f::PLCFun{T, U}, g::PLCFun{T, U})::PLCFun{T, U} where {T,
    @assert length(f.points) == length(g.points)
    PLCFun{T, U}(f.points .+ g.points)
end
```

In [8]:
```
# Calculate the x coordinates of the endpoints of the lines
function xcoord(::Type{T}, nlines::Int, i::Int)::T where {T}
    @assert 1 <= i <= nlines + 1
    dx = 1 / nlines
    x = (i-1) * dx
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code

In [8]:
```
# Calculate the x coordinates of the endpoints of the lines
function xcoord(::Type{T}, nlines::Int, i::Int)::T where {T}
    @assert 1 <= i <= nlines + 1
    dx = 1 / nlines
    x = (i-1) * dx
    x
end
```

Out[8]: xcoord (generic function with 1 method)

In [9]:
```
# The inverse of "xcoord": Determine the line segment on which a particula
function lineidx(f::PLCFun{T, U}, x::T)::Int where {T, U}
    @assert 0 <= x <= 1
    nlines = length(f.points) - 1
    dx = 1 / nlines
    i = floor(Int, x / dx) + 1
    i = max(1, i)
    i = min(nlines, i)
    i
end
```

Out[9]: lineidx (generic function with 1 method)

In [10]:
```
# Convert a general Julia function into a PLCFun. We need to specify the t
# as well as the number of line segments to use.
function samplePLC(::Type{T}, ::Type{U}, nlines::Int, f::Function)::PLCFun
    ys = U[f(xcoord(T, nlines, i)) for i in 1:nlines+1]
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code

In [12]:
```
# Evaluate a PLCFun at point x
function evaluate(f::PLCFun{T,U}, x::T)::U where {T, U}
    @assert 0 <= x <= 1
    nlines = length(f.points) - 1
    # Find out on which line segment the point lies
    i = lineidx(f, x)
    # Interpolate between the two endpoints of the line segment
    x1 = xcoord(T, nlines, i)
    x2 = xcoord(T, nlines, i+1)
    y1 = f.points[i]
    y2 = f.points[i+1]
    y = linterp(x1, y1, x2, y2, x)
    y
end
```

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

In [13]:
```
# Calculate the derivative of a PLCFun, using a right-biased derivative
function derivRight(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [[(f.points[i+1] - f.points[i]) / dx for i in 1:nlines];
          (f.points[end] - f.points[end-1]) / dx]
    PLCFun{T, U}(ys)
end
```

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

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ | FiniteDifferencing

File Edit View Insert Cell Kernel Widgets Help | Trusted | Julia 1.0.1

Run | Code

In [13]:
```julia
# Calculate the derivative of a PLCFun, using a right-biased derivative
function derivRight(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [[(f.points[i+1] - f.points[i]) / dx for i in 1:nlines];
          (f.points[end] - f.points[end-1]) / dx]
    PLCFun{T, U}(ys)
end
```

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

Example use of PLCFun

In [14]:
```julia
fsinpi = samplePLC(Float64, Float64, 4, sinpi)
```

Out[14]: PLCFun{Float64,Float64}([0.0, 0.707107, 1.0, 0.707107, 0.0])

In [15]:
```julia
(evaluate(fsinpi, 0.3333), sinpi(0.3333))
```

Out[15]: (0.8046988016951899, 0.8659730391584589)

In [16]:
```julia
xs = collect(range(0, stop=1, length=100))
plot(sinpi.(xs))
plot!([evaluate(fsinpi, x) for x in xs])
```

Out[16]:

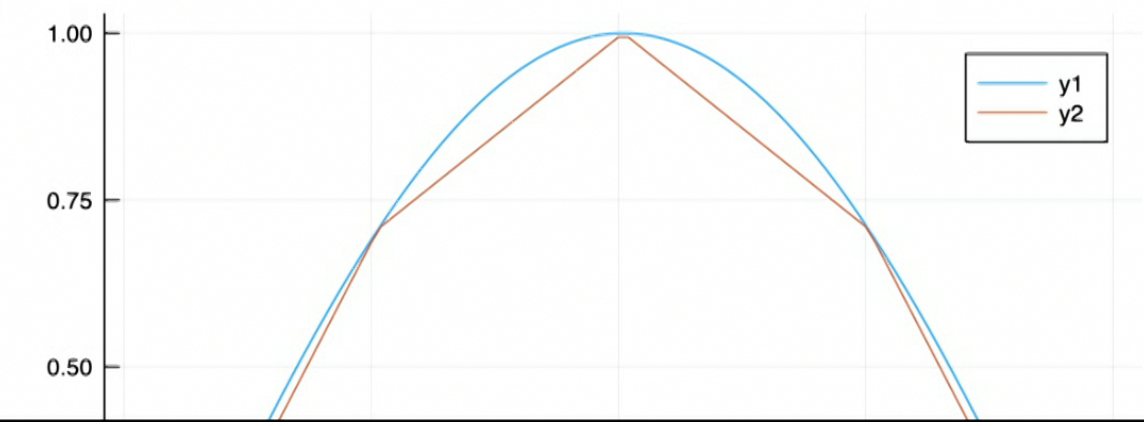
File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Example use of PLCFun

In [14]:
```
fsinpi = samplePLC(Float64, Float64, 4, sinpi)
```
Out[14]:
```
PLCFun{Float64,Float64}([0.0, 0.707107, 1.0, 0.707107, 0.0])
```

In [15]:
```
(evaluate(fsinpi, 0.3333), sinpi(0.3333))
```
Out[15]:
```
(0.8046988016951899, 0.8659730391584589)
```

In [16]:
```
xs = collect(range(0, stop=1, length=100))
plot(sinpi.(xs))
plot!([evaluate(fsinpi, x) for x in xs])
```
Out[16]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help | Trusted | Julia 1.0.1

Run | Code

0 25 50 75 100

Advection Equation

In [17]:
```
# The state vector describing how we evolve the advection equation in time
struct AdvectionState{T}
    time::T
    u::PLCFun{T, T}
end
```

In [18]:
```
# A Gaussian, centred at x=1/2, with a width of 1/10
function gaussian(x::T)::T where {T}
    exp(- ((x - 0.5) * 10)^2)
end
```

Out[18]: gaussian (generic function with 1 method)

In [19]:
```
# Define initial conditions: A Gaussian at t=0
function initialGaussian(nlines::Int)::AdvectionState{Float64}
    t = 0
    u = samplePLC(Float64, Float64, nlines, gaussian)
    AdvectionState{Float64}(t, u)
end
```

Out[19]: initialGaussian (generic function with 1 method)

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

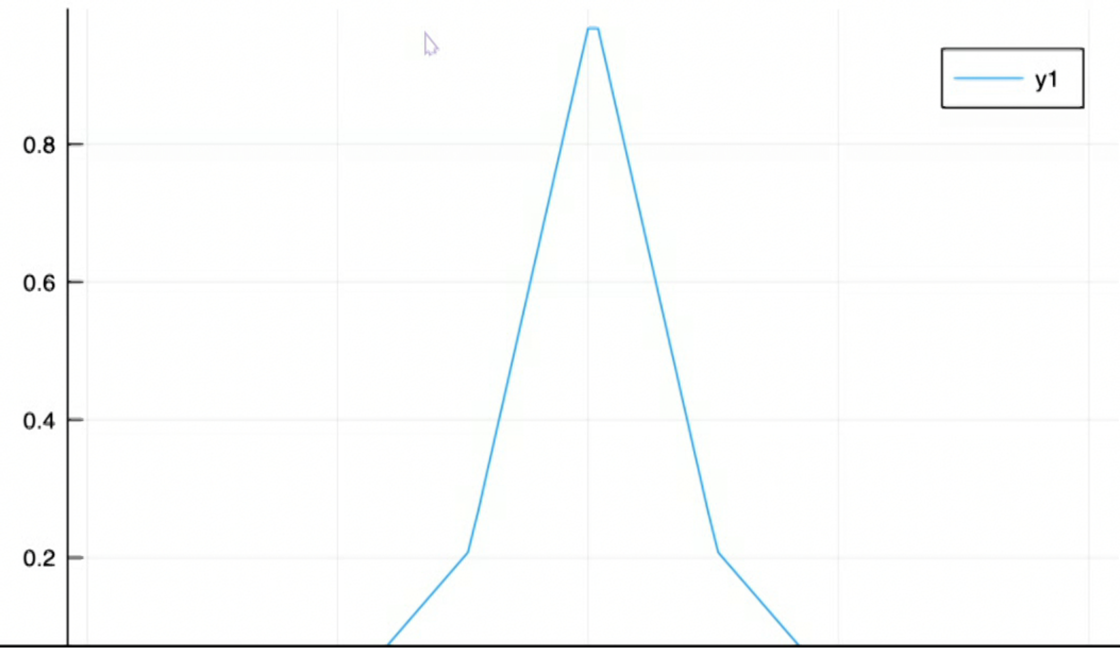
Run Code

In [20]:
```
s0 = initialGaussian(8)
```

Out[20]:
```
AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([1.38879e-11, 7.8114
9e-7, 0.00193045, 0.209611, 1.0, 0.209611, 0.00193045, 7.81149e-7, 1.3887
9e-11]))
```

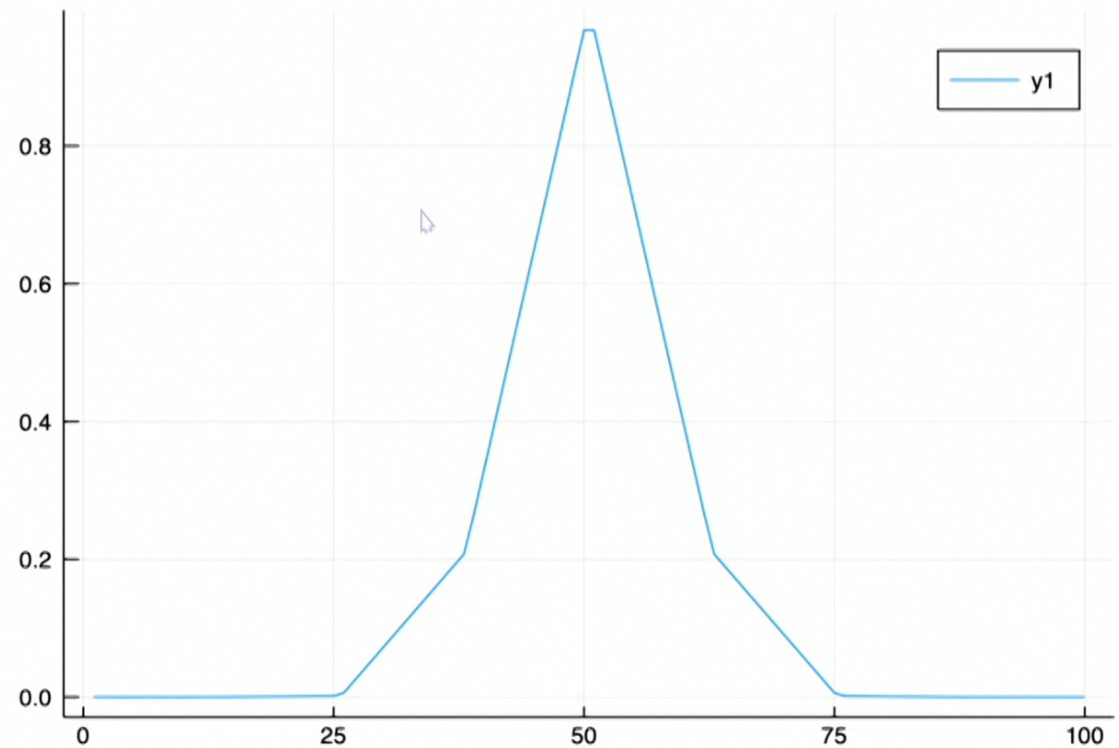
In [21]:
```
plot([evaluate(s0.u, x) for x in xs])
```

Out[21]:



In [21]:
```
plot([evaluate(s0.u, x) for x in xs])
```

Out[21]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code

In [22]:

```
# The RHS (Right Hand Side) of the advection equation, including boundary
function rhsAdvection(s::AdvectionState{T})::AdvectionState{T} where {T}
    t = s.time
    u = s.u
    nlines = length(u.points)
    # Calculate spatial derivative
    ux = derivRight(u)
    # Define time derivative as the spatial derivative almost everywhere,
    # except at the right boundary where we set the time derivative to zer
    ut = PLCFun{T, T}([[ux.points[i] for i in 1:nlines]; 0])
    AdvectionState{T}(t, ut)
end
```

Out[22]:

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

In [23]:

```
r0 = rhsAdvection(s0)
```

Out[23]:

```
AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([6.24908e-6, 0.01543
74, 1.66145, 6.32311, -6.32311, -1.66145, -0.0154374, -6.24908e-6, -6.249
08e-6, 0.0]))
```

In [24]:

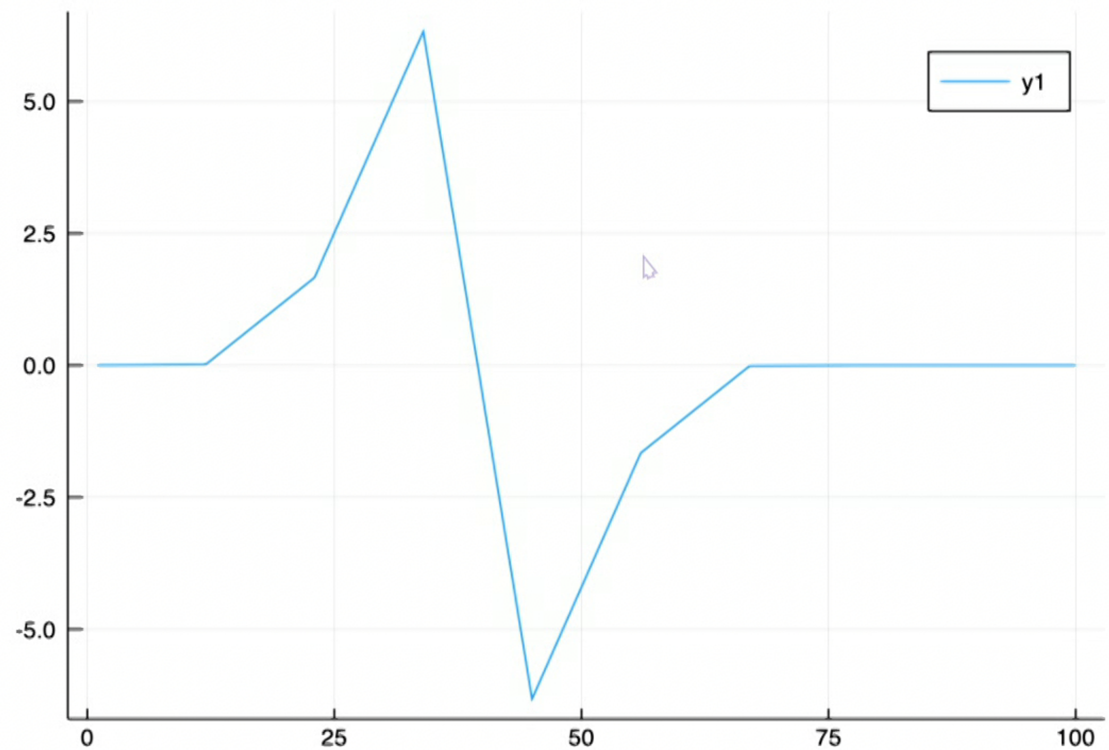
```
plot([evaluate(r0.u, x) for x in xs])
```

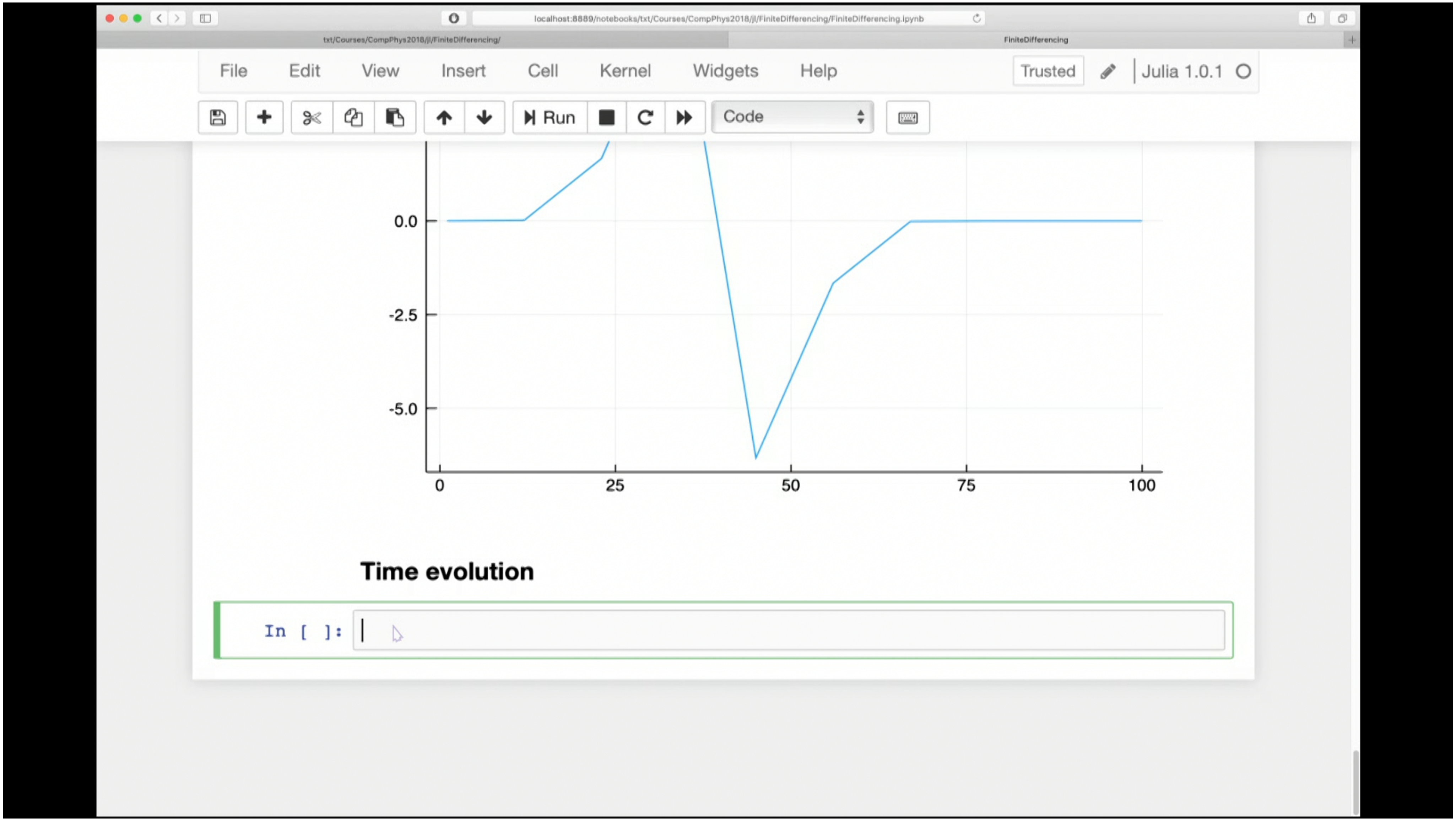
Out[24]:

74, 1.66145, 6.32311, -6.32311, -1.66145, -0.0154374, -6.24908e-6, 0.0]))

In [24]:
```
plot([evaluate(r0.u, x) for x in xs])
```

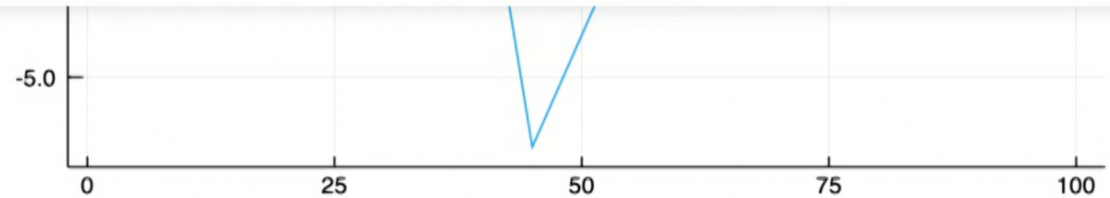
Out[24]:




localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
0.0
-2.5
-5.0
0
25
50
75
100
Time evolution
In [ ]:

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help | Trusted | Julia 1.0.1



Time evolution

In [27]:
```
function euler(rhs::Function, dt::T,
               s0::AdvectionState{T})::AdvectionState{T} where {T}
    r0 = rhs(s0)
    s1 = s0 + dt * r0
    s1
end
```

Out[27]: euler (generic function with 1 method)

In [ ]:

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

15 matches length Done

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code

```
end
```

In [31]:
```
# State vectors can be scaled and added
function Base. *(a::T, s::AdvectionState{T})::AdvectionState{T} where {T}
    AdvectionState{T}(s.time, a .* s.u)
end
function Base. +(s1::AdvectionState{T},
                 s2::AdvectionState{T})::AdvectionState{T} where {T}
    @assert abs(s1.time - s2.time) <= 100*eps(T)
    AdvectionState{T}(s1.time, s1.u + s2.u)
end
```

In [18]:
```
# A Gaussian, centred at x=1/2, with a width of 1/10
function gaussian(x::T)::T where {T}
    exp(- ((x - 0.5) * 10)^2)
end
```

Out[18]: gaussian (generic function with 1 method)

In [19]:
```
# Define initial conditions: A Gaussian at t=0
function initialGaussian(nlines::Int)::AdvectionState{Float64}
    t = 0
    u = samplePLC(Float64, Float64, nlines, gaussian)
    AdvectionState{Float64}(t, u)
end
```

Out[19]: initialGaussian (generic function with 1 method)

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

15 matches length Done

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code

Time evolution

In [27]:

```
function euler(rhs::Function, dt::T,
               s0::AdvectionState{T})::AdvectionState{T} where {T}
    r0 = rhs(s0)
    s1 = s0 + dt * r0
    s1
end
```

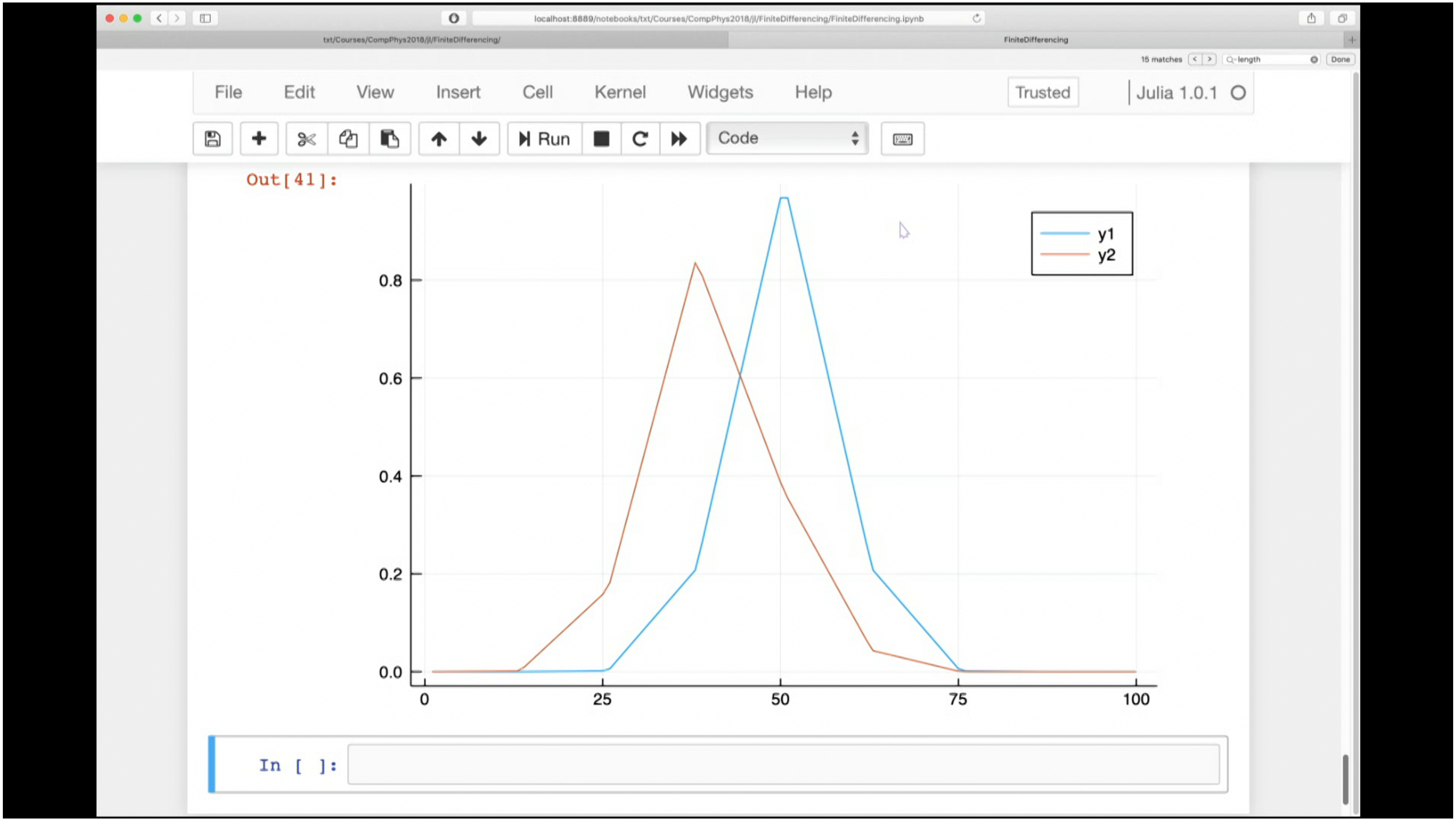
Out[27]:

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

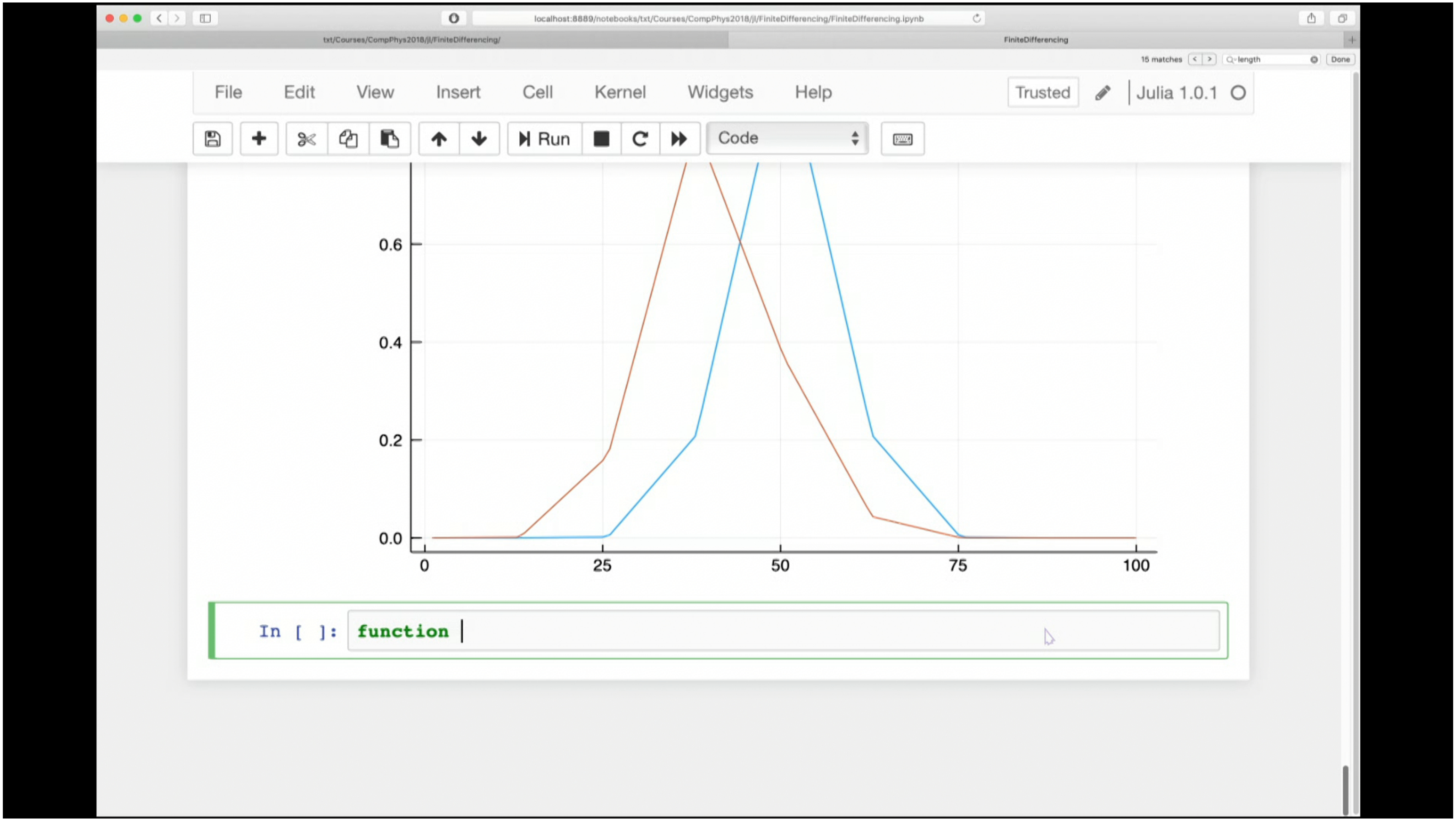
In [36]:

```
euler(rhsAdvection, 0.1, s0)
```

```
MethodError: no method matching iterate(::PLCFun{Float64,Float64})
Closest candidates are:
  iterate(!Matched::Core.SimpleVector) at essentials.jl:589
  iterate(!Matched::Core.SimpleVector, !Matched::Any) at essentials.jl:58
9
  iterate(!Matched::ExponentialBackOff) at error.jl:171
  ...

Stacktrace:
 [1] copyto!(::Array{Any,1}, ::PLCFun{Float64,Float64}) at ./abstractarra
y.jl:646
 [2] _collect(::UnitRange{Int64}, ::PLCFun{Float64,Float64}, ::Base.HasEl
type, ::Base.HasLength) at ./array.jl:563
 [3] collect(::PLCFun{Float64,Float64}) at ./array.jl:557
```



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.0.1
Run
Code
Out[41]:
y1
y2
0.0
0.2
0.4
0.6
0.8
0
25
50
75
100
In [ ]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
0.6
0.4
0.2
0.0
0
25
50
75
100
In [ ]: function

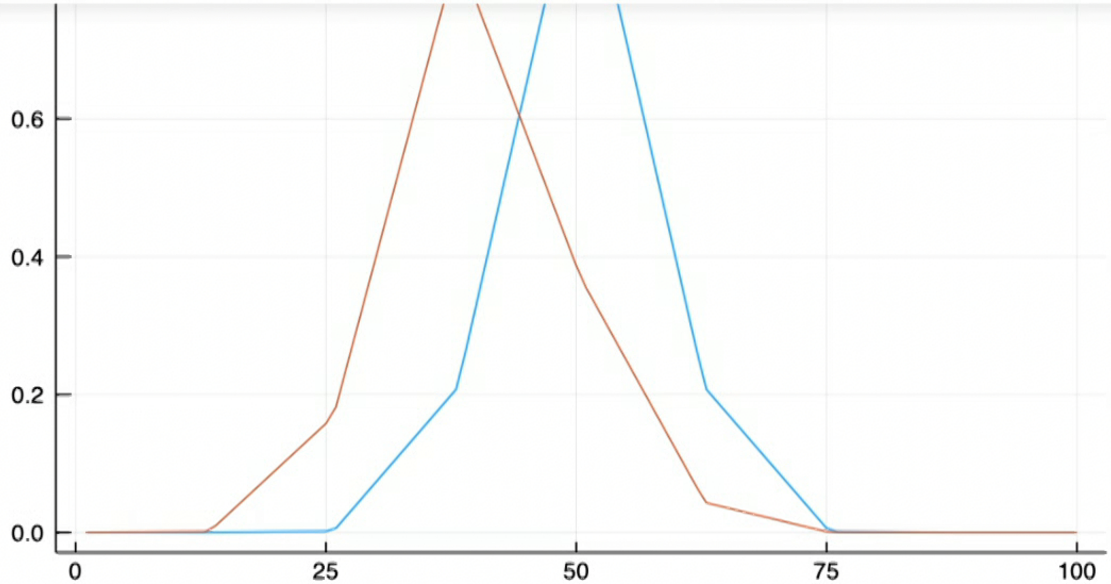
localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

15 matches length Done

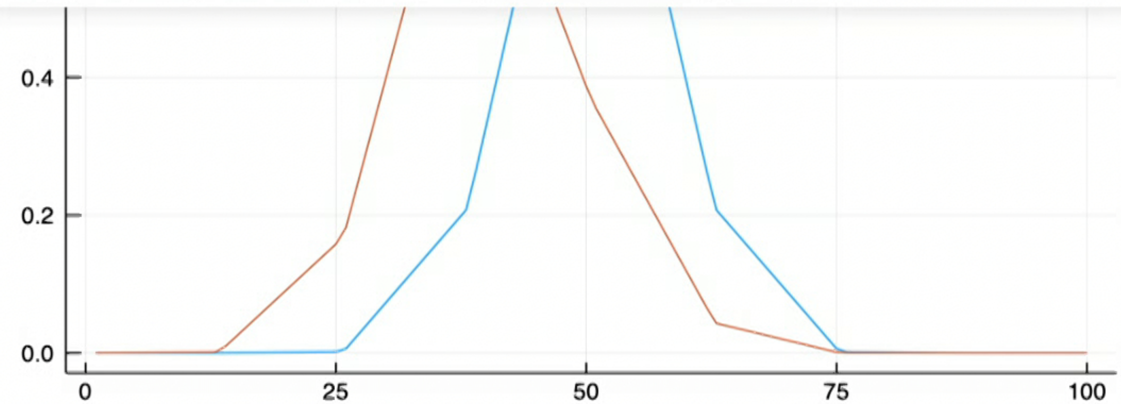
File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code



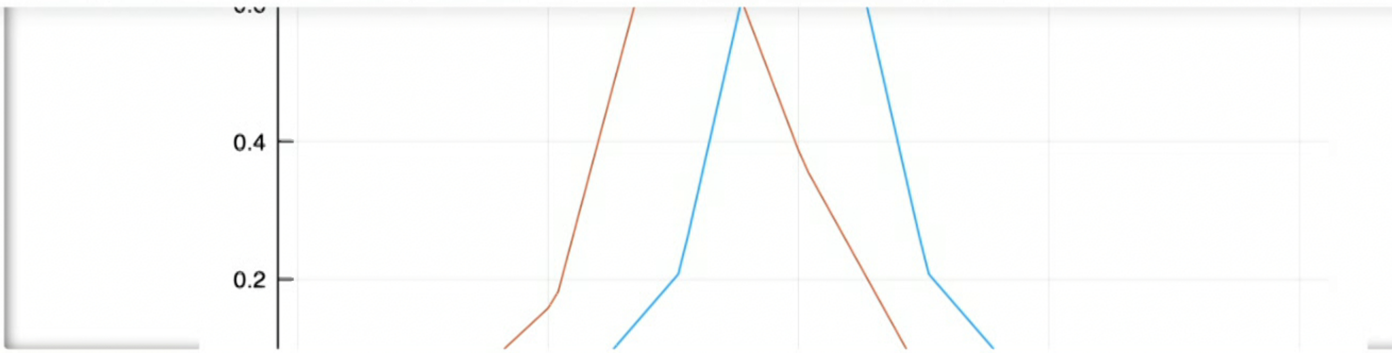
In [ ]:

```
function solveAdvection(tmax::T, nlines::Int, lambda::T) where {T}
    s = initialGaussian(nlines)
end
```



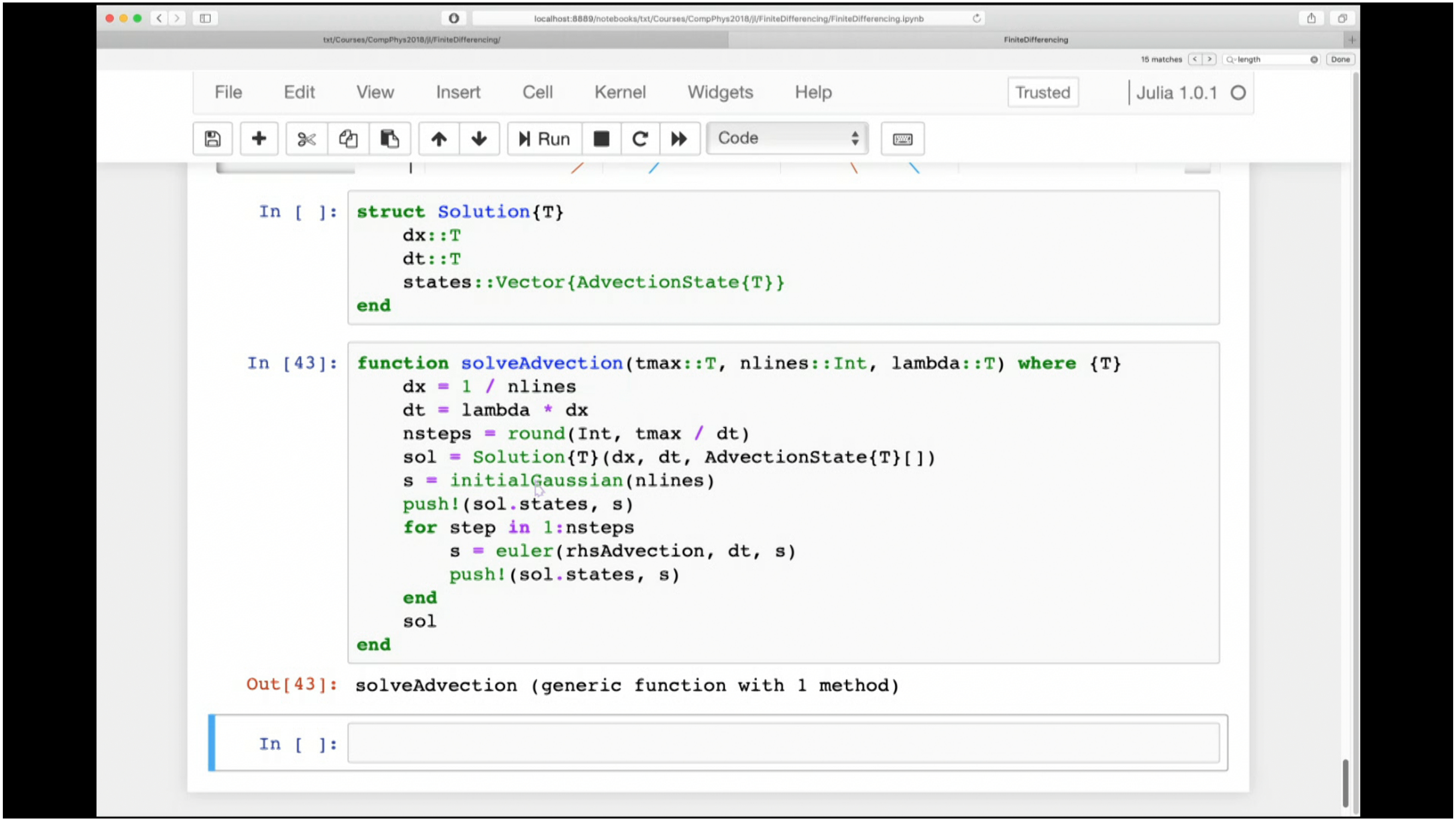
In [ ]:

```
function solveAdvection(tmax::T, nlines::Int, lambda::T) where {T}
    dx = 1 / nlines
    dt = lambda * dx
    nsteps = round(Int, tmax / dt)
    s = initialGaussian(nlines)
    for step in 1:nsteps
        s = euler(rhsAdvection, dt, s)
    end
end
```



In [ ]:
```julia
struct Solution{T}
    states::Vector{AdvectionState{T}}
end
```

In [ ]:
```julia
function solveAdvection(tmax::T, nlines::Int, lambda::T) where {T}
    dx = 1 / nlines
    dt = lambda * dx
    nsteps = round(Int, tmax / dt)
    s = initialGaussian(nlines)
    for step in 1:nsteps
        s = euler(rhsAdvection, dt, s)
    end
end
```


localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.0.1
Run
Code
In [ ]:
struct Solution{T}
    dx::T
    dt::T
    states::Vector{AdvectionState{T}}
end
In [43]:
function solveAdvection(tmax::T, nlines::Int, lambda::T) where {T}
    dx = 1 / nlines
    dt = lambda * dx
    nsteps = round(Int, tmax / dt)
    sol = Solution{T}(dx, dt, AdvectionState{T}[])
    s = initialGaussian(nlines)
    push!(sol.states, s)
    for step in 1:nsteps
        s = euler(rhsAdvection, dt, s)
        push!(sol.states, s)
    end
    sol
end
Out[43]: solveAdvection (generic function with 1 method)
In [ ]:

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

```
483004, 0.0533683, 0.355288, 0.604806, 0.355288, 0.0533683, 0.000483004,
1.95298e-7, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCFun{Float64,
Float64}([0.0269256, 0.204328, 0.480047, 0.480047, 0.204328, 0.0269256,
0.0002416, 9.76558e-8, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCF
un{Float64,Float64}([0.115627, 0.342188, 0.480047, 0.342188, 0.115627, 0
.0135836, 0.000120849, 4.88348e-8, 1.38879e-11])), AdvectionState{Float6
4}(0.0, PLCFun{Float64,Float64}([0.228907, 0.411117, 0.411117, 0.228907,
0.0646053, 0.00685223, 6.04488e-5, 2.44244e-8, 1.38879e-11])), Advection
State{Float64}(0.0, PLCFun{Float64,Float64}([0.320012, 0.411117, 0.32001
2, 0.146756, 0.0357288, 0.00345634, 3.02366e-5, 1.22191e-8, 1.38879e-11]
)), AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([0.365565, 0.36
5565, 0.233384, 0.0912425, 0.0195926, 0.00174329, 1.51244e-5, 6.11651e-9
, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([
0.365565, 0.299475, 0.162313, 0.0554175, 0.0106679, 0.000879207, 7.56526
e-6, 3.0652e-9, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCFun{Floa
t64,Float64}([0.33252, 0.230894, 0.108865, 0.0330427, 0.00577356, 0.0004
43386, 3.78416e-6, 1.53954e-9, 1.38879e-11])), AdvectionState{Float64}(0
.0, PLCFun{Float64,Float64}([0.281707, 0.16988, 0.0709541, 0.0194081, 0.
```

```
In [ ]:
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

```
Float64}([0.0269256, 0.204328, 0.480047, 0.480047, 0.204328, 0.0269256,
0.0002416, 9.76558e-8, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCF
un{Float64,Float64}([0.115627, 0.342188, 0.480047, 0.342188, 0.115627, 0
.0135836, 0.000120849, 4.88348e-8, 1.38879e-11])), AdvectionState{Float6
4}(0.0, PLCFun{Float64,Float64}([0.228907, 0.411117, 0.411117, 0.228907,
0.0646053, 0.00685223, 6.04488e-5, 2.44244e-8, 1.38879e-11])), Advection
State{Float64}(0.0, PLCFun{Float64,Float64}([0.320012, 0.411117, 0.32001
2, 0.146756, 0.0357288, 0.00345634, 3.02366e-5, 1.22191e-8, 1.38879e-11]
)), AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([0.365565, 0.36
5565, 0.233384, 0.0912425, 0.0195926, 0.00174329, 1.51244e-5, 6.11651e-9
, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([
0.365565, 0.299475, 0.162313, 0.0554175, 0.0106679, 0.000879207, 7.56526
e-6, 3.0652e-9, 1.38879e-11])), AdvectionState{Float64}(0.0, PLCFun{Floa
t64,Float64}([0.33252, 0.230894, 0.108865, 0.0330427, 0.00577356, 0.0004
43386, 3.78416e-6, 1.53954e-9, 1.38879e-11])), AdvectionState{Float64}(0
.0, PLCFun{Float64,Float64}([0.281707, 0.16988, 0.0709541, 0.0194081, 0.
```

In [48]:

```
plot(evaluate.(sol.states[1].u(xs)))
```

```
MethodError: objects of type PLCFun{Float64,Float64} are not callable

Stacktrace:
 [1] top-level scope at In[48]:1
```

In [ ]:

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

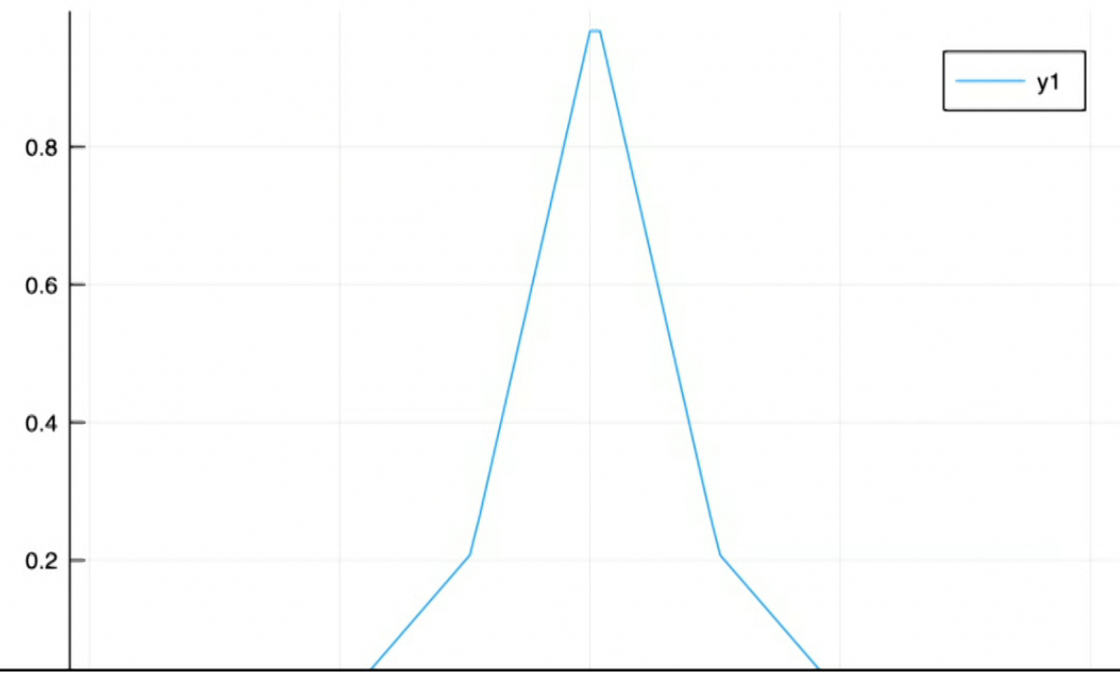
File Edit View Insert Cell Kernel Widgets Help — Trusted — Julia 1.0.1

Out[43]: solveAdvection (generic function with 1 method)

In [50]:
```
sol = solveAdvection(1.0, 8, 0.5);
```

In [49]:
```
plot([evaluate(sol.states[1].u, x) for x in xs])
plot!([evaluate(sol.states[1].u, x) for x in xs])
```

Out[49]:



File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

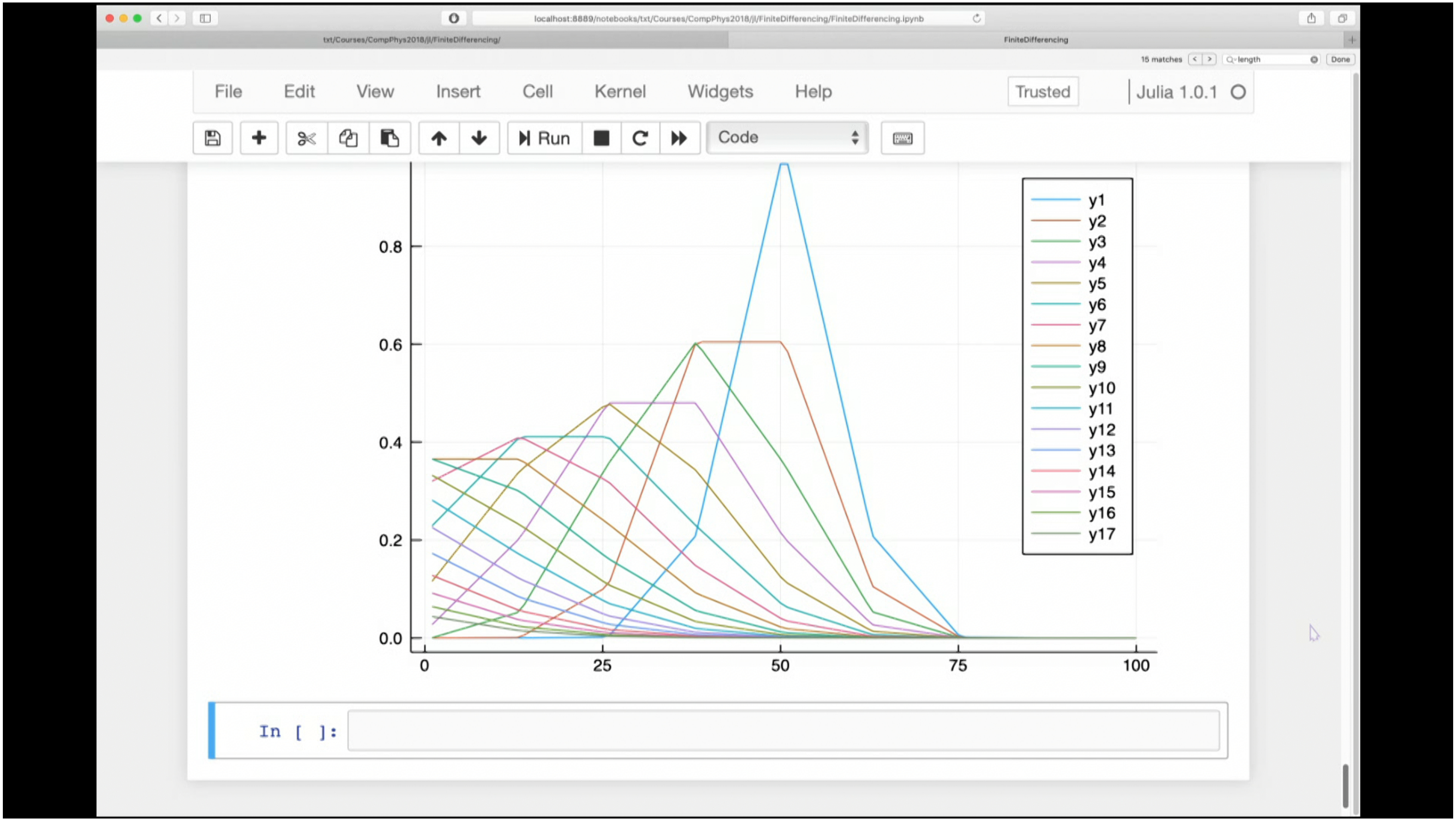
```julia
    s = initialGaussian(nlines)
    push!(sol.states, s)
    for step in 1:nsteps
        s = euler(rhsAdvection, dt, s)
        push!(sol.states, s)
    end
    sol
end
```

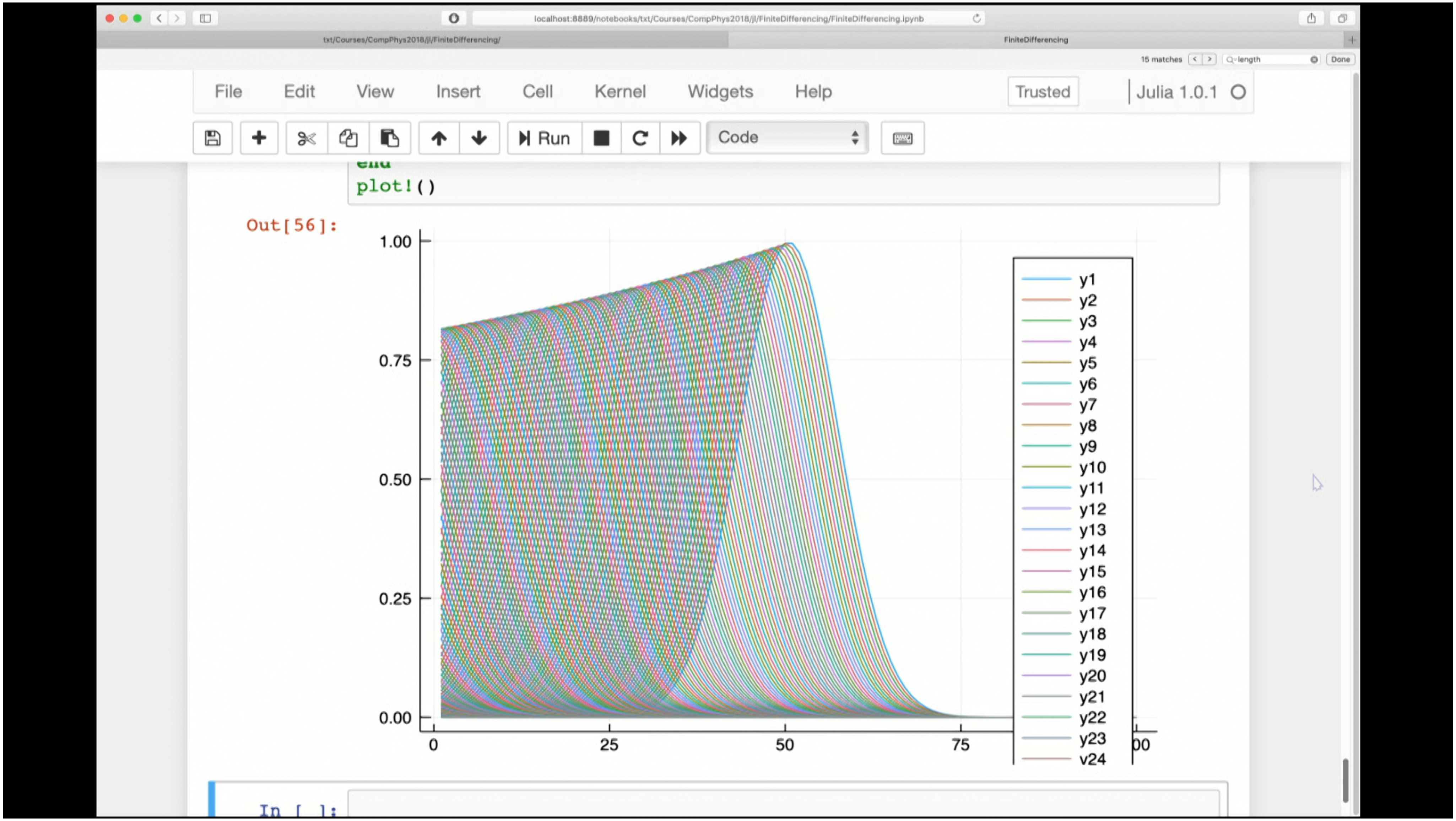
Out[43]: solveAdvection (generic function with 1 method)

In [50]:
```julia
sol = solveAdvection(1.0, 8, 0.5);
```

In [*]:
```julia
plot([evaluate(sol.states[1].u, x) for x in xs])
for state in sol.states[2:end]
    plot!([evaluate(state.u, x) for x in xs])
end
plot!()
```

In [ ]:

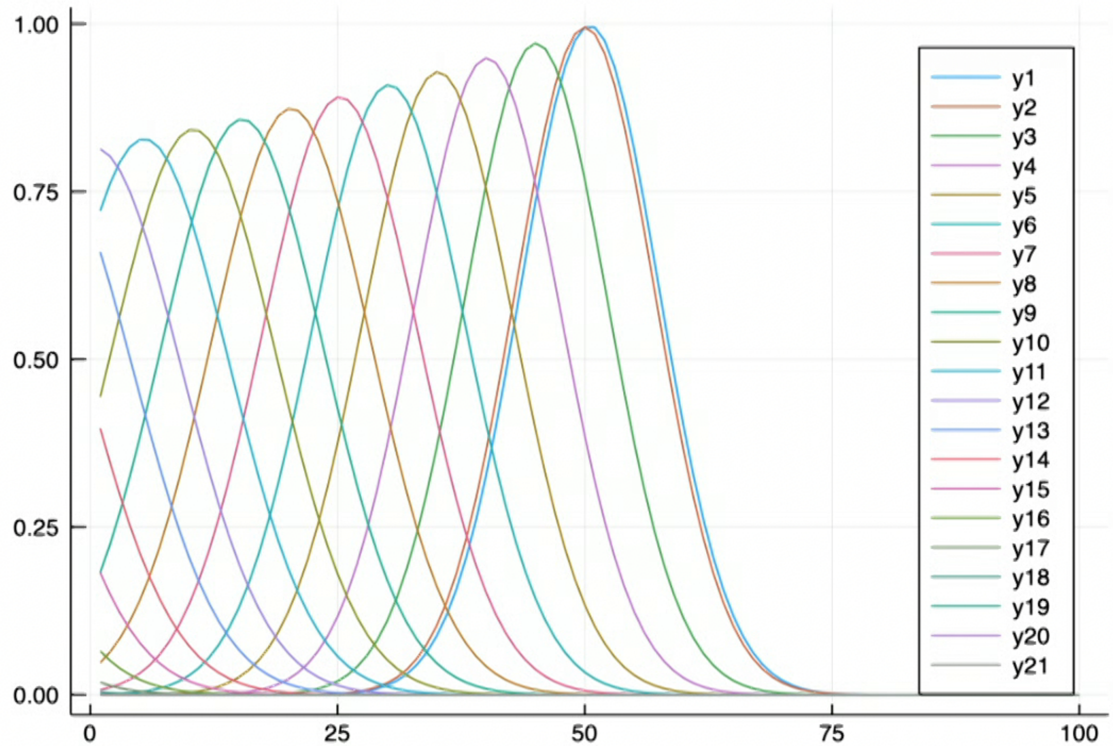

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
y14
y15
y16
y17
In [ ]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.0.1
Run
Code
plot!()
Out[56]:
1.00
0.75
0.50
0.25
0.00
0
25
50
75
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
y14
y15
y16
y17
y18
y19
y20
y21
y22
y23
y24
In [ ]:

```julia
for state in sol.states[2:10:end]
    plot!([evaluate(state.u, x) for x in xs])
end
plot!()
```

Out[58]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help

Trusted Julia 1.0.1

```
        push!(sol.states, s)
    end
    sol
end
```

Out[43]: solveAdvection (generic function with 1 method)

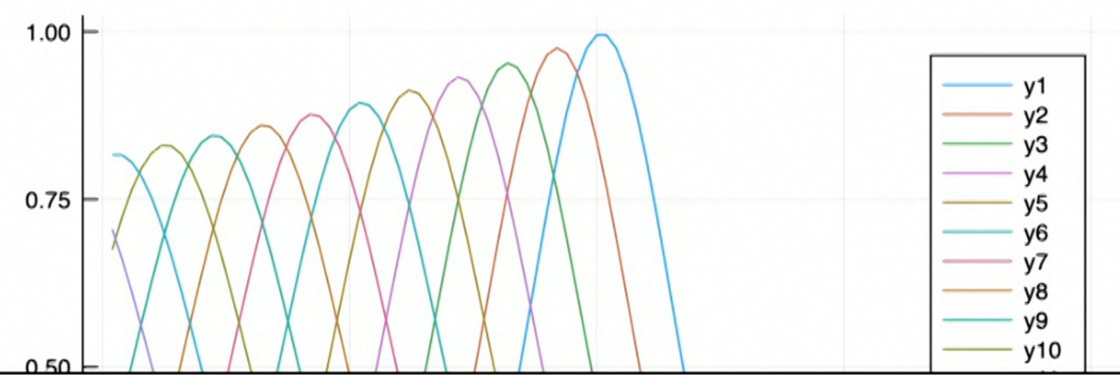
In [55]:

```
sol1 = solveAdvection(1.0, 100, 0.5);
sol2 = solveAdvection(1.0, 100, 0.5);
```

In [59]:

```
plot([evaluate(sol.states[1].u, x) for x in xs])
for state in sol.states[10:10:end]
    plot!([evaluate(state.u, x) for x in xs])
end
plot!()
```

Out[59]:



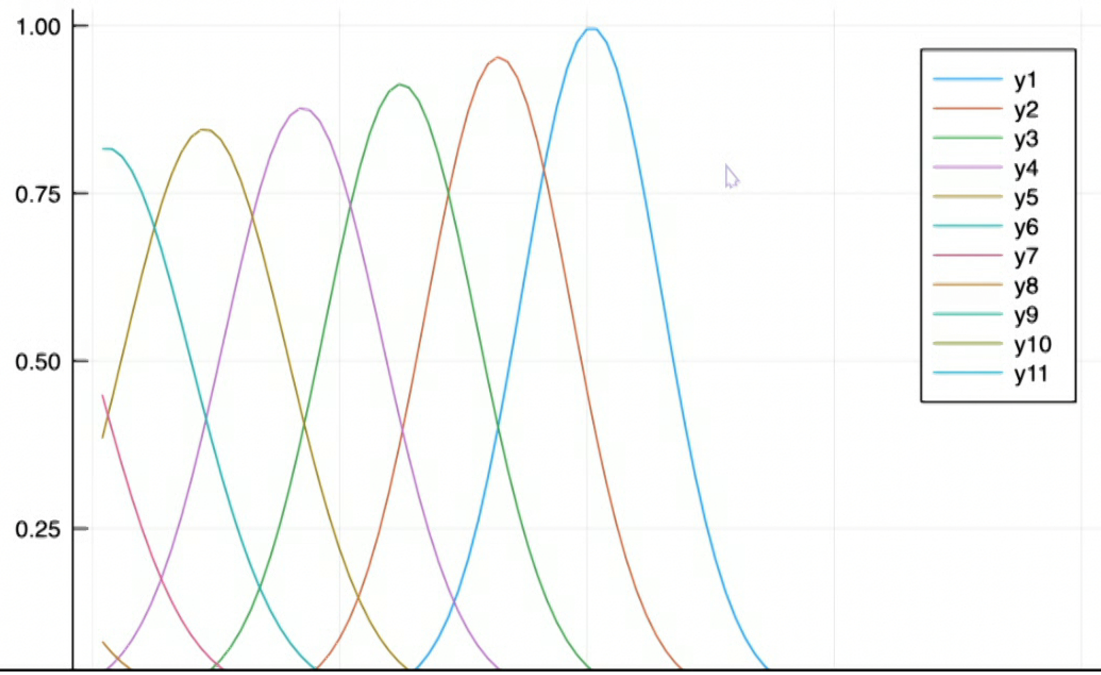
File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

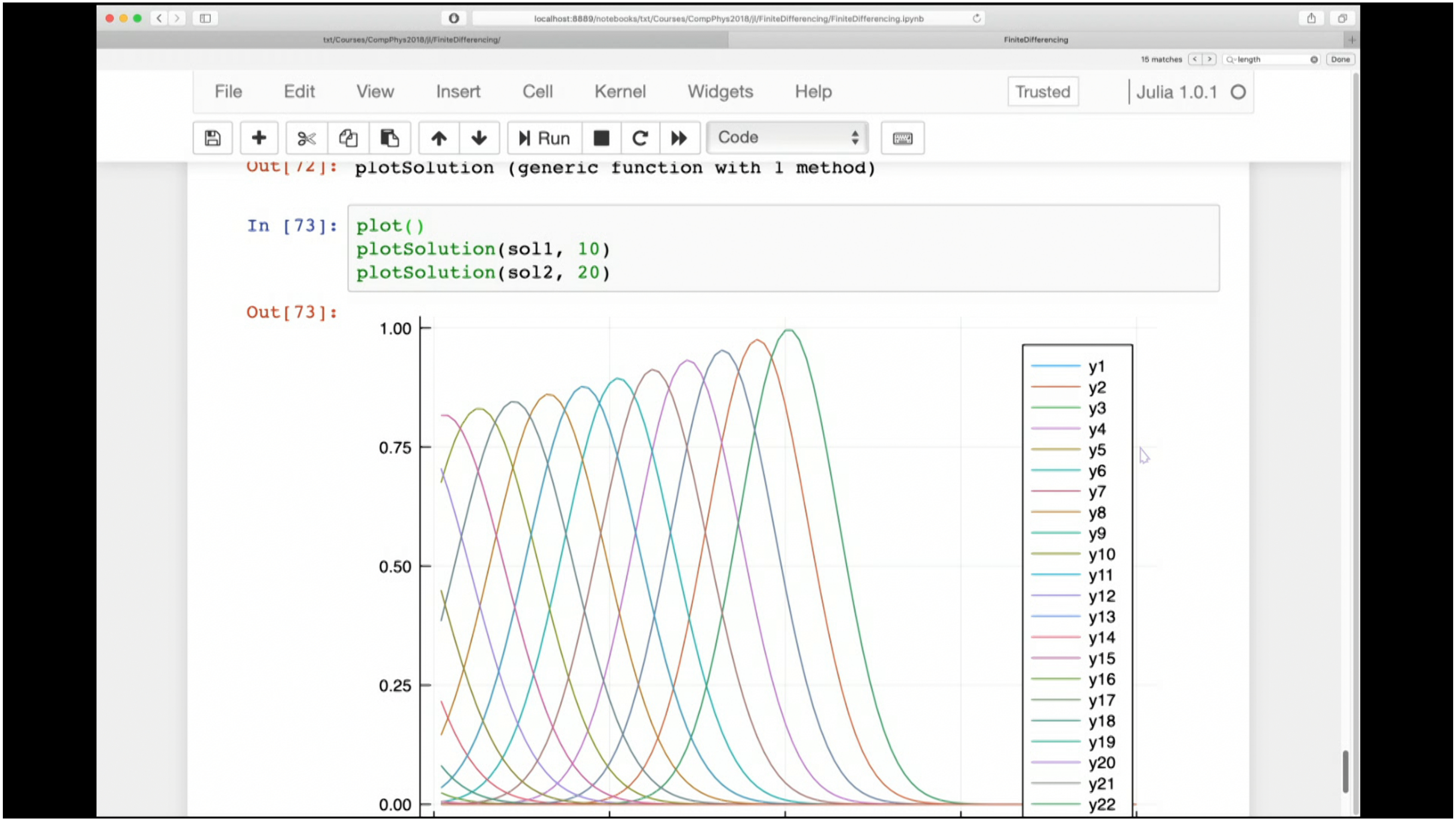
```
end
```

Out[61]: plotSolution (generic function with 1 method)

In [63]:
```
plotSolution(sol1, 10)
plotSolution(sol2, 20)
```

Out[63]:




localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
Out[72]: plotSolution (generic function with 1 method)
In [73]:
plot()
plotSolution(sol1, 10)
plotSolution(sol2, 20)
Out[73]:
1.00
0.75
0.50
0.25
0.00
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
y14
y15
y16
y17
y18
y19
y20
y21
y22

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

txt/Courses/CompPhys2018/jl/FiniteDifferencing/ FiniteDifferencing

15 matches length Done

File Edit View Insert Cell Widgets Help Trusted Julia 1.0.1

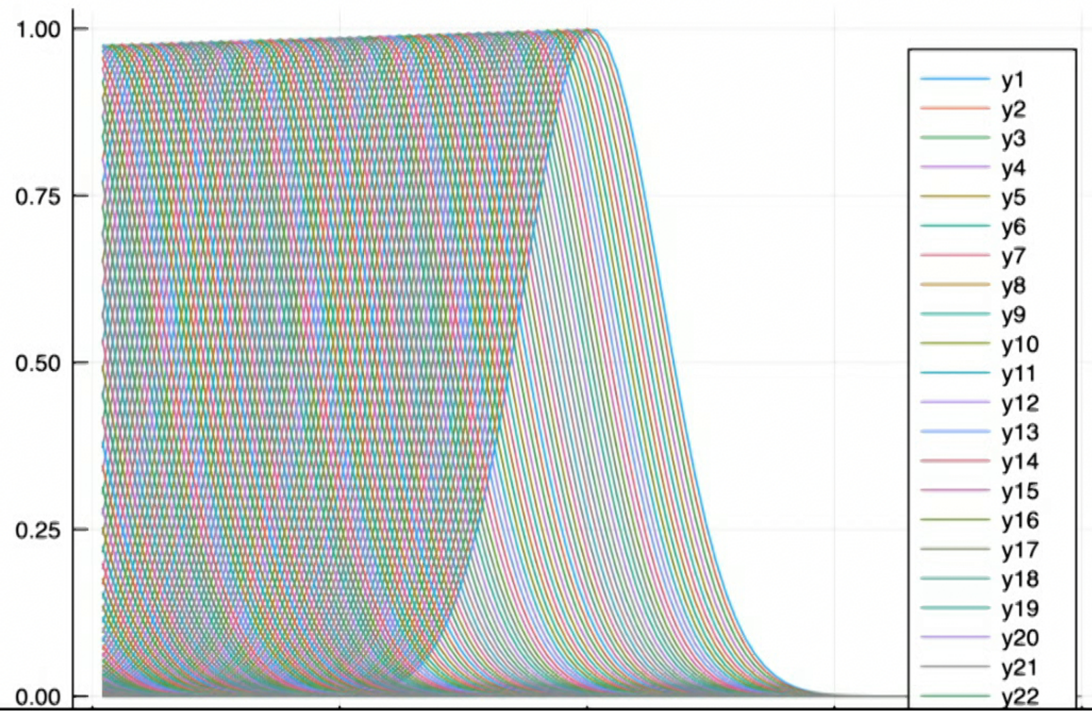
Run Code

Out[76]: plotSolution (generic function with 1 method)

In [77]:

```
plot()
plotSolution(sol1, 10)
# plotSolution(sol2, 20)
```

Out[77]:



File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

In [6]:
```julia
# A piecewise linear continuous function mapping a type T to a type U
struct PLCFun{T,U}
    # Domain: [0; 1]
    points::Vector{U}
end
```

In [35]:
```julia
# Functions can be scaled and added
# function Base.length(f::PLCFun{T, U})::Int where {T, U}
#     length(f.points)
# end
function Base. *(a::U, f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    PLCFun{T, U}(a .* f.points)
end
function Base. +(f::PLCFun{T, U}, g::PLCFun{T, U})::PLCFun{T, U} where {T,
    @assert length(f.points) == length(g.points)
    PLCFun{T, U}(f.points .+ g.points)
end
```

In [8]:
```julia
# Calculate the x coordinates of the endpoints of the lines
function xcoord(::Type{T}, nlines::Int, i::Int)::T where {T}
    @assert 1 <= i <= nlines + 1
    dx = 1 / nlines
    x = (i-1) * dx
    x
end
```

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

In [8]:
```julia
# Calculate the x coordinates of the endpoints of the lines
function xcoord(::Type{T}, nlines::Int, i::Int)::T where {T}
    @assert 1 <= i <= nlines + 1
    dx = 1 / nlines
    x = (i-1) * dx
    x
end
```

Out[8]: `xcoord (generic function with 1 method)`

In [9]:
```julia
# The inverse of "xcoord": Determine the line segment on which a particula
function lineidx(f::PLCFun{T, U}, x::T)::Int where {T, U}
    @assert 0 <= x <= 1
    nlines = length(f.points) - 1
    dx = 1 / nlines
    i = floor(Int, x / dx) + 1
    i = max(1, i)
    i = min(nlines, i)
    i
end
```

Out[9]: `lineidx (generic function with 1 method)`

In [10]:
```julia
# Convert a general Julia function into a PLCFun. We need to specify the t
# as well as the number of line segments to use.
```

File Edit View Insert Cell Kernel Widgets Help | Trusted | Julia 1.0.1

```
end
```

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

In [13]:
```
# Calculate the derivative of a PLCFun, using a right-biased derivative
function derivRight(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [[(f.points[i+1] - f.points[i]) / dx for i in 1:nlines];
          (f.points[end] - f.points[end-1]) / dx]
    PLCFun{T, U}(ys)
end
```

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

In [ ]:
```
function refine(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [xcoord(T, 2*nlines, i)      for i in 1:2*nlines+1]
end
```

Example use of PLCFun

In [14]:
```
fsinpi = samplePLC(Float64, Float64, 4, sinpi)
```

Out[14]: PLCFun{Float64,Float64}([0.0, 0.707107, 1.0, 0.707107, 0.0])

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

```
end
```

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

In [13]:
```julia
# Calculate the derivative of a PLCFun, using a right-biased derivative
function derivRight(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [[(f.points[i+1] - f.points[i]) / dx for i in 1:nlines];
          (f.points[end] - f.points[end-1]) / dx]
    PLCFun{T, U}(ys)
end
```

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

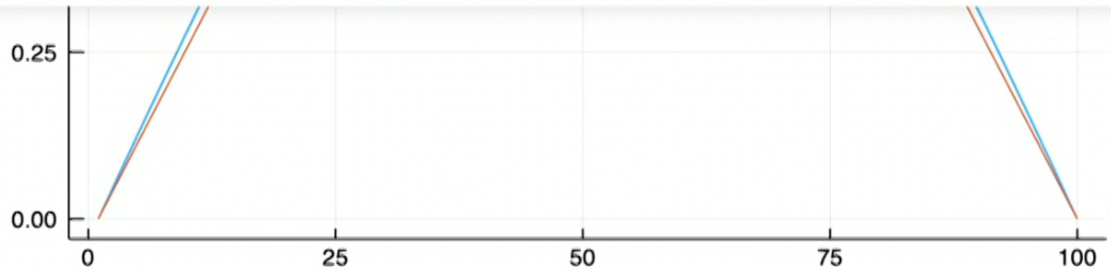
In [ ]:
```julia
function refine(f::PLCFun{T, U})::PLCFun{T, U} where {T, U}
    nlines = length(f.points) - 1
    dx = 1 / nlines
    ys = [evaluate(f, xcoord(T, 2*nlines, i)) for i in 1:2*nlines+1]

end
```

Example use of PLCFun

In [14]:
```julia
fsinpi = samplePLC(Float64, Float64, 4, sinpi)
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1



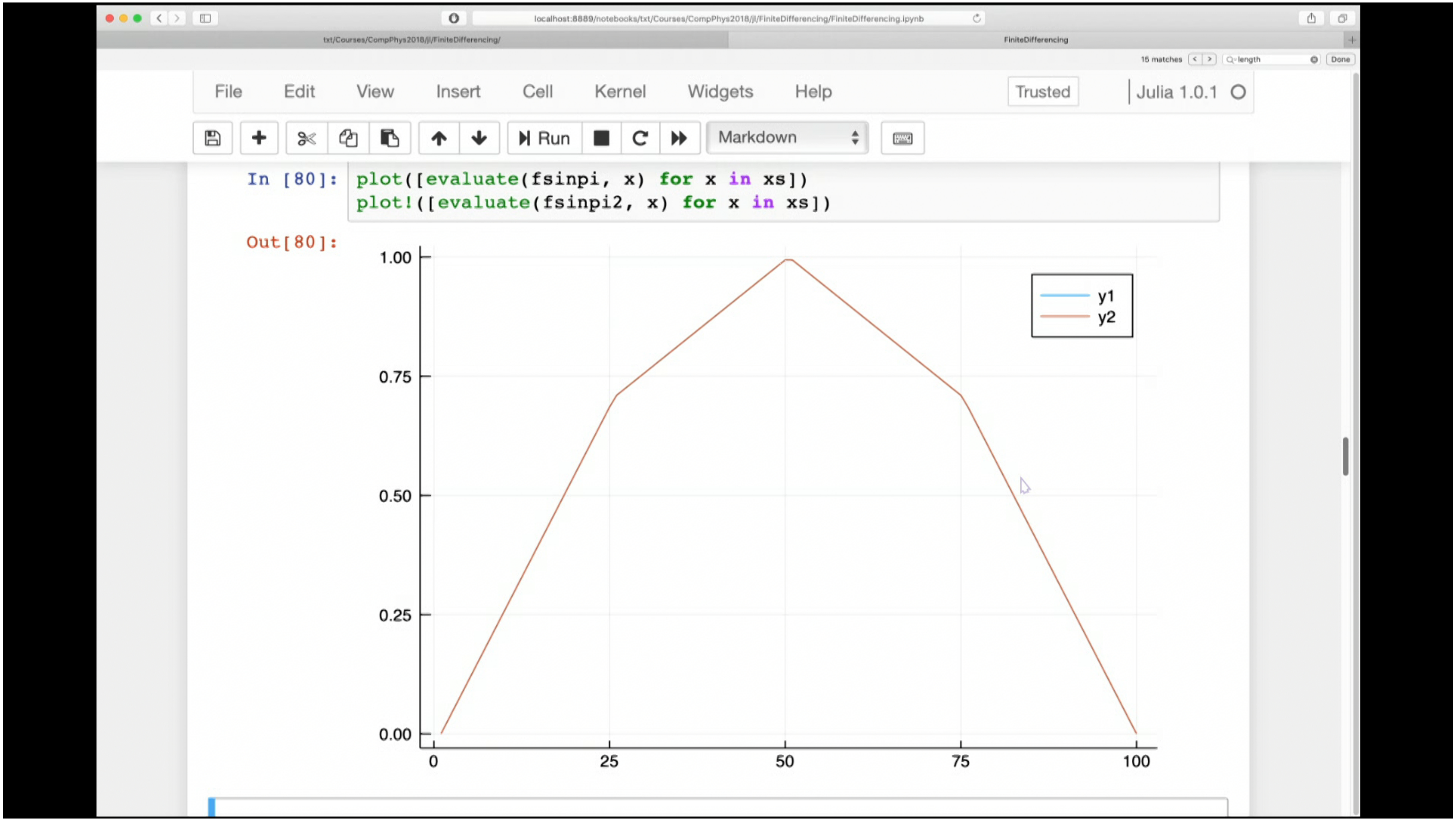
In [79]:
```
fsinpi2 = refine(fsinpi)
```

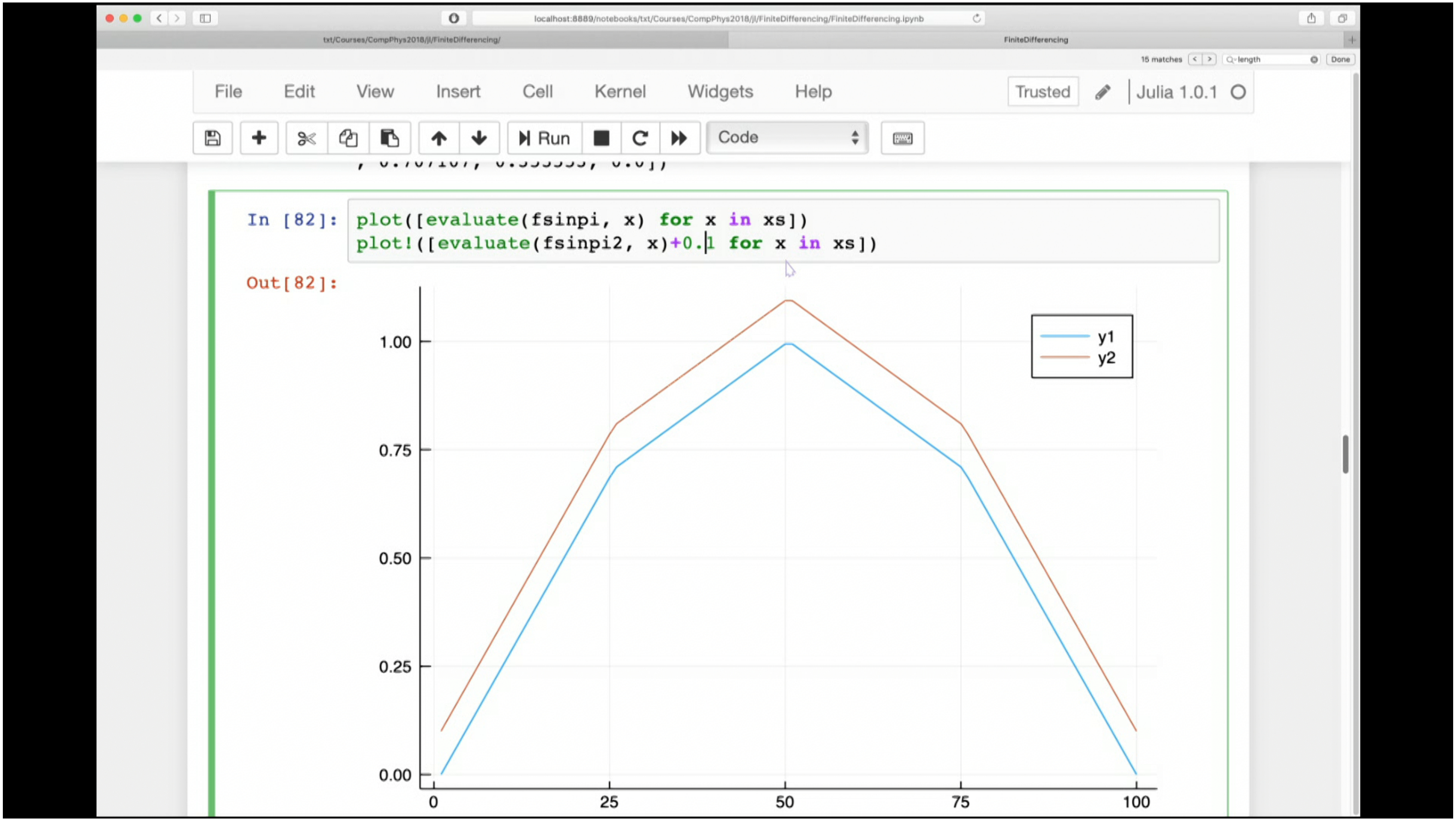
Out[79]:
```
PLCFun{Float64,Float64}([0.0, 0.353553, 0.707107, 0.853553, 1.0, 0.853553
, 0.707107, 0.353553, 0.0])
```

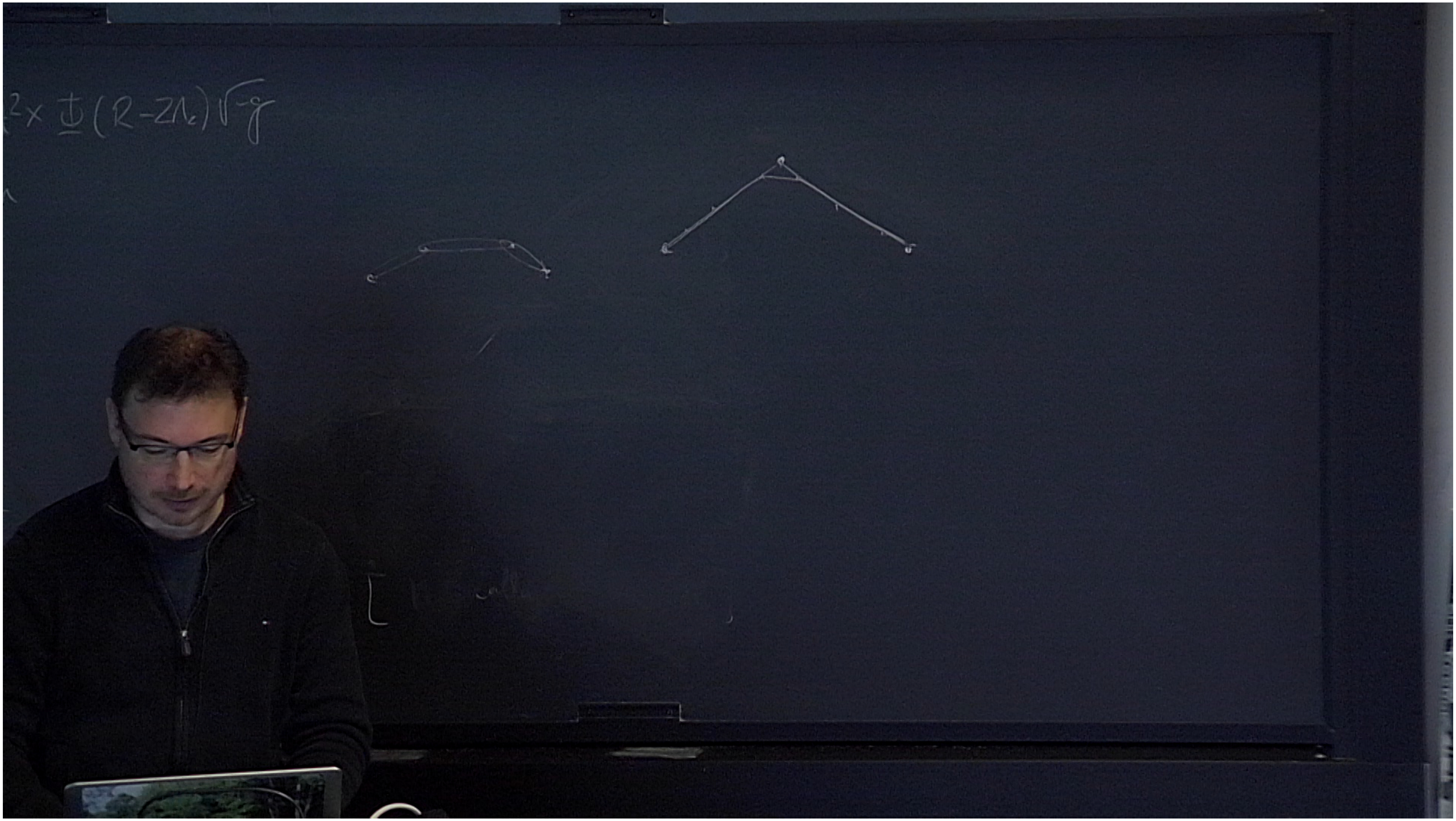
In [ ]:
```
plot([evaluate(fsinpi, x) for x in xs])
plot!([evaluate(fsinpi2, x) for x in xs])
```

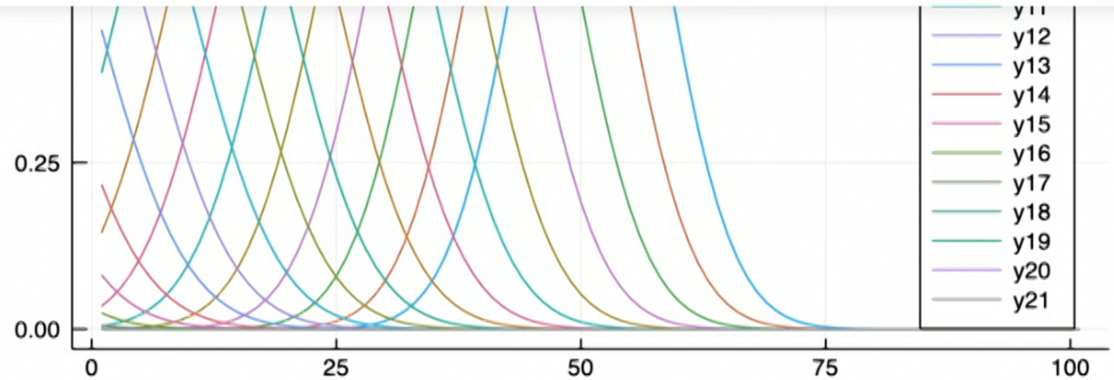
Advection Equation

In [17]:
```
# The state vector describing how we evolve the advection equation in time
struct AdvectionState{T}
    time::T
    u::PLCFun{T, T}
end
```


localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Markdown
In [80]:
plot([evaluate(fsinpi, x) for x in xs])
plot!([evaluate(fsinpi2, x) for x in xs])
Out[80]:
y1
y2
1.00
0.75
0.50
0.25
0.00
0
25
50
75
100


localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
15 matches
length
Done
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
In [82]:
plot([evaluate(fsinpi, x) for x in xs])
plot!([evaluate(fsinpi2, x)+0.1 for x in xs])
Out[82]:
y1
y2
1.00
0.75
0.50
0.25
0.00
0
25
50
75
100





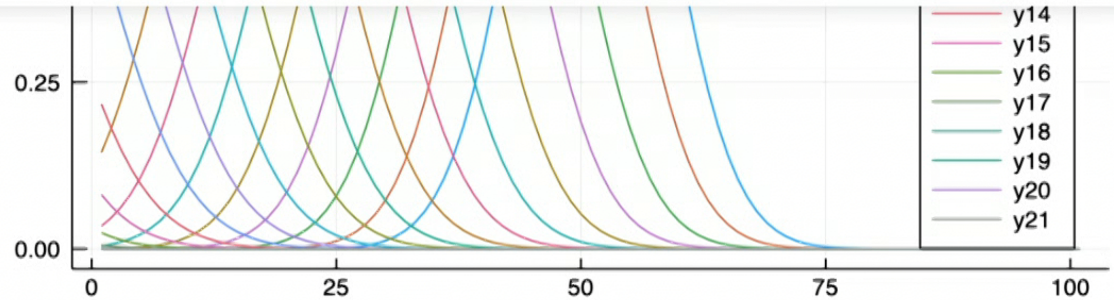
Discretization Error, Convergence

```
In [92]: sol1 = solveAdvection(1.0, 100, 0.5);
         sol2 = solveAdvection(1.0, 200, 0.5);
```

```
In [ ]:
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1



Discretization Error, Convergence

In [92]:
```
sol1 = solveAdvection(1.0, 100, 0.5);
sol2 = solveAdvection(1.0, 200, 0.5);
```

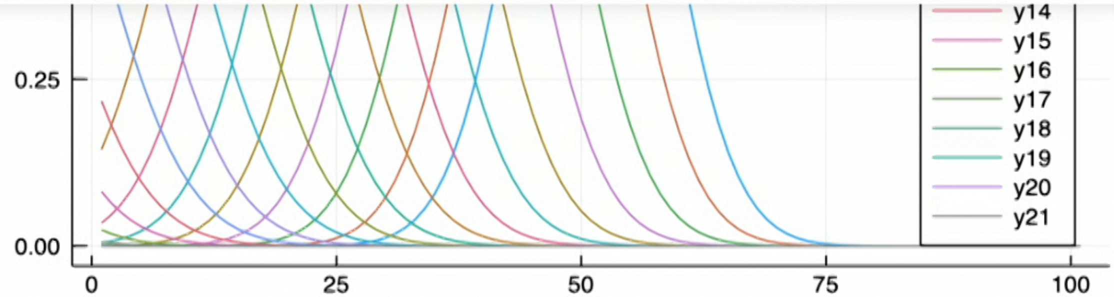
In [95]:
```
sol1 + (-1.0) * sol2
```

```
MethodError: no method matching *(::Float64, ::Solution{Float64})
Closest candidates are:
  *(::Any, ::Any, !Matched::Any, !Matched::Any...) at operators.jl:502
  *(::Float64, !Matched::Float64) at float.jl:399
  *(::AbstractFloat, !Matched::Bool) at bool.jl:120
  ...

Stacktrace:
 [1] top-level scope at In[95]:1
```

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code



Discretization Error, Convergence

In [92]:
```
sol1 = solveAdvection(1.0, 100, 0.5);
sol2 = solveAdvection(1.0, 200, 0.5);
```

In [98]:
```
refine(sol1.states[1]) + (-1.0) * sol2.states[1]
```

```
MethodError: no method matching refine(::AdvectionState{Float64})
Closest candidates are:
  refine(!Matched::PLCFun{T,U}) where {T, U} at In[78]:2

Stacktrace:
 [1] top-level scope at In[98]:1
```

In [ ]:

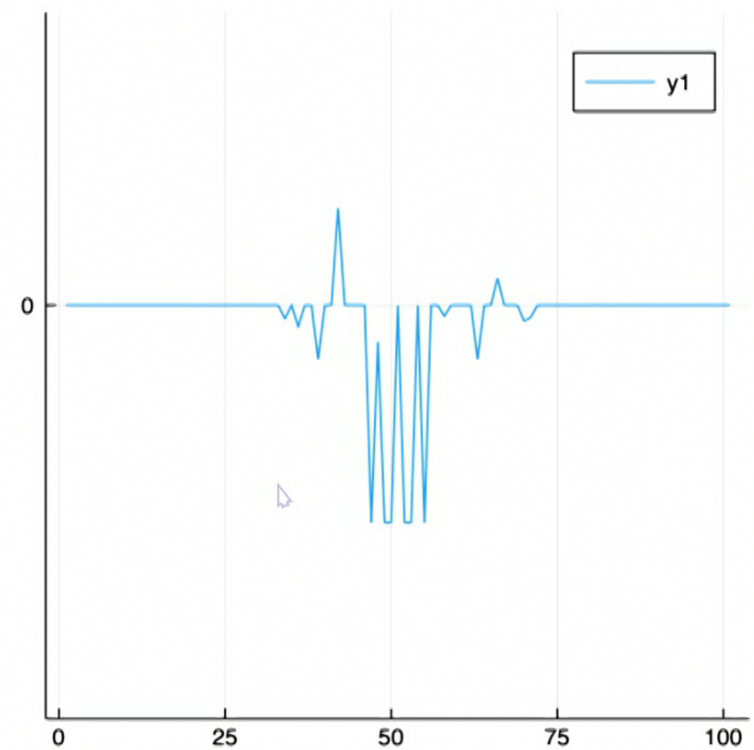
File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

```
6595e-11, 0.0, 1.78275e-11, 1.29247e-26, 7.12873e-12, 0.0, 2.79165e-12, 0
.0])
```

In [101]:

```
plot([evaluate(ue, x) for x in xs])
```

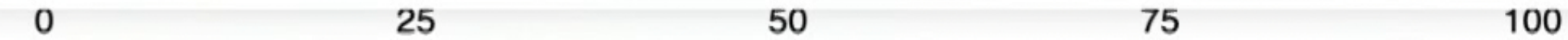
Out[101]:



File Edit View Insert Cell Kernel Widgets Help

Trusted Julia 1.0.1

Run Code



Discretization Error, Convergence

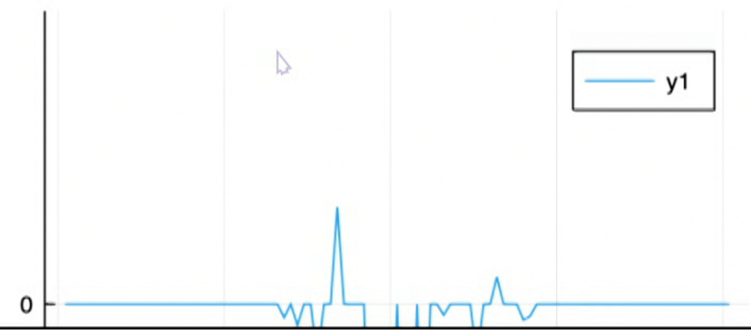
In [92]:
```
sol1 = solveAdvection(1.0, 100, 0.5);
sol2 = solveAdvection(1.0, 200, 0.5);
```

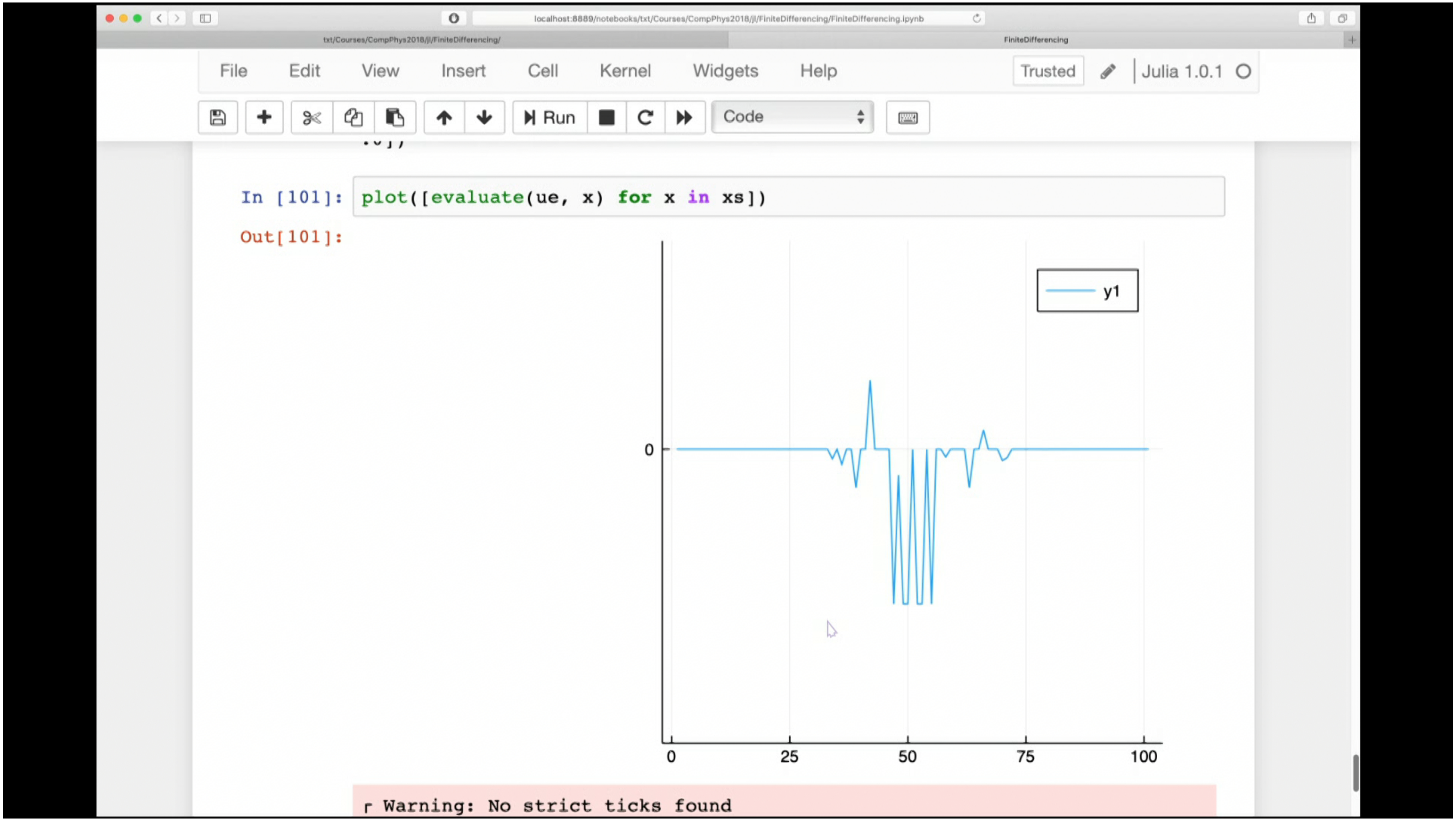
In [100]:
```
ue = refine(sol1.states[1].u) + (-1.0) * sol2.states[1].u
```

Out[100]:
```
PLCFun{Float64,Float64}([0.0, 2.79165e-12, 0.0, 7.12873e-12, 1.29247e-26,
1.78275e-11, 0.0, 4.36595e-11, 0.0, 1.04703e-10  …  1.04703e-10, 0.0, 4.3
6595e-11, 0.0, 1.78275e-11, 1.29247e-26, 7.12873e-12, 0.0, 2.79165e-12, 0
.0])
```

In [101]:
```
plot([evaluate(ue, x) for x in xs])
```

Out[101]:

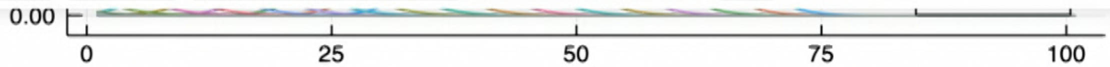



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.0.1
Run
Code
In [101]: plot([evaluate(ue, x) for x in xs])
Out[101]:
y1
0
0 25 50 75 100
Warning: No strict ticks found

localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

Run Code



Discretization Error, Convergence

In [92]:
```
sol1 = solveAdvection(1.0, 100, 0.5);
sol2 = solveAdvection(1.0, 200, 0.5);
```

In [106]:
```
(sol1.states[2].time, sol2.states[3].time)
```

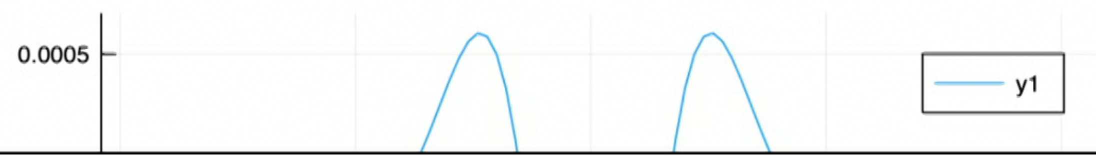
Out[106]:
```
(0.0, 0.0)
```

In [104]:
```
ue = refine(sol1.states[2].u) + (-1.0) * sol2.states[3].u
```

Out[104]:
```
PLCFun{Float64,Float64}([1.39583e-12, 7.19654e-12, 3.56437e-12, 1.81295e-11, 8.91376e-12, 4.47295e-11, 2.18298e-11, 1.08076e-10, 5.23513e-11, 2.55721e-10 … 1.08076e-10, 2.18298e-11, 4.47295e-11, 8.91376e-12, 1.81295e-11, 3.56437e-12, 7.19654e-12, 1.39583e-12, 3.63388e-12, 0.0])
```

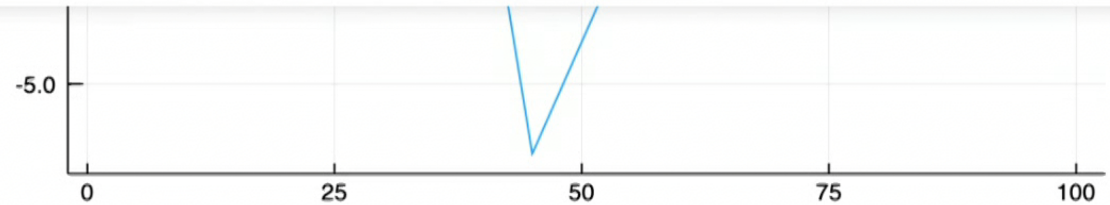
In [105]:
```
plot([evaluate(ue, x) for x in xs])
```

Out[105]:



localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1



Time evolution

In [107]:
```
function euler(rhs::Function, dt::T,
               s0::AdvectionState{T})::AdvectionState{T} where {T}
    r0 = rhs(s0)
    s1 = s0 + dt * r0
    AdvectionState{T}(s0.time + dt, s1.u)
end
```

Out[107]: euler (generic function with 1 method)

In [39]:
```
s1 = euler(rhsAdvection, 0.1, s0)
```

Out[39]: AdvectionState{Float64}(0.0, PLCFun{Float64,Float64}([6.24922e-7, 0.00154452, 0.168075, 0.841922, 0.367689, 0.0434666, 0.000386716, 1.56241e-7, 1.38879e-11]))

In [41]:
```
plot([evaluate(s0.u, x) for x in xs])
plot!([evaluate(s1.u, x) for x in xs])
```

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

In [116]:
```julia
(sol1.states[2].time, sol2.states[3].time)
```

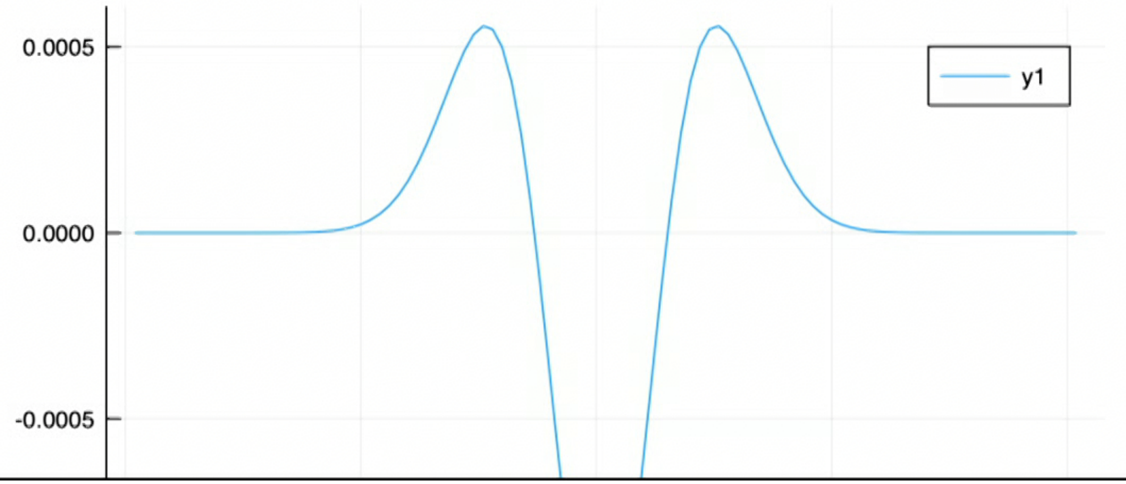
Out[116]:
```
(0.005, 0.005)
```

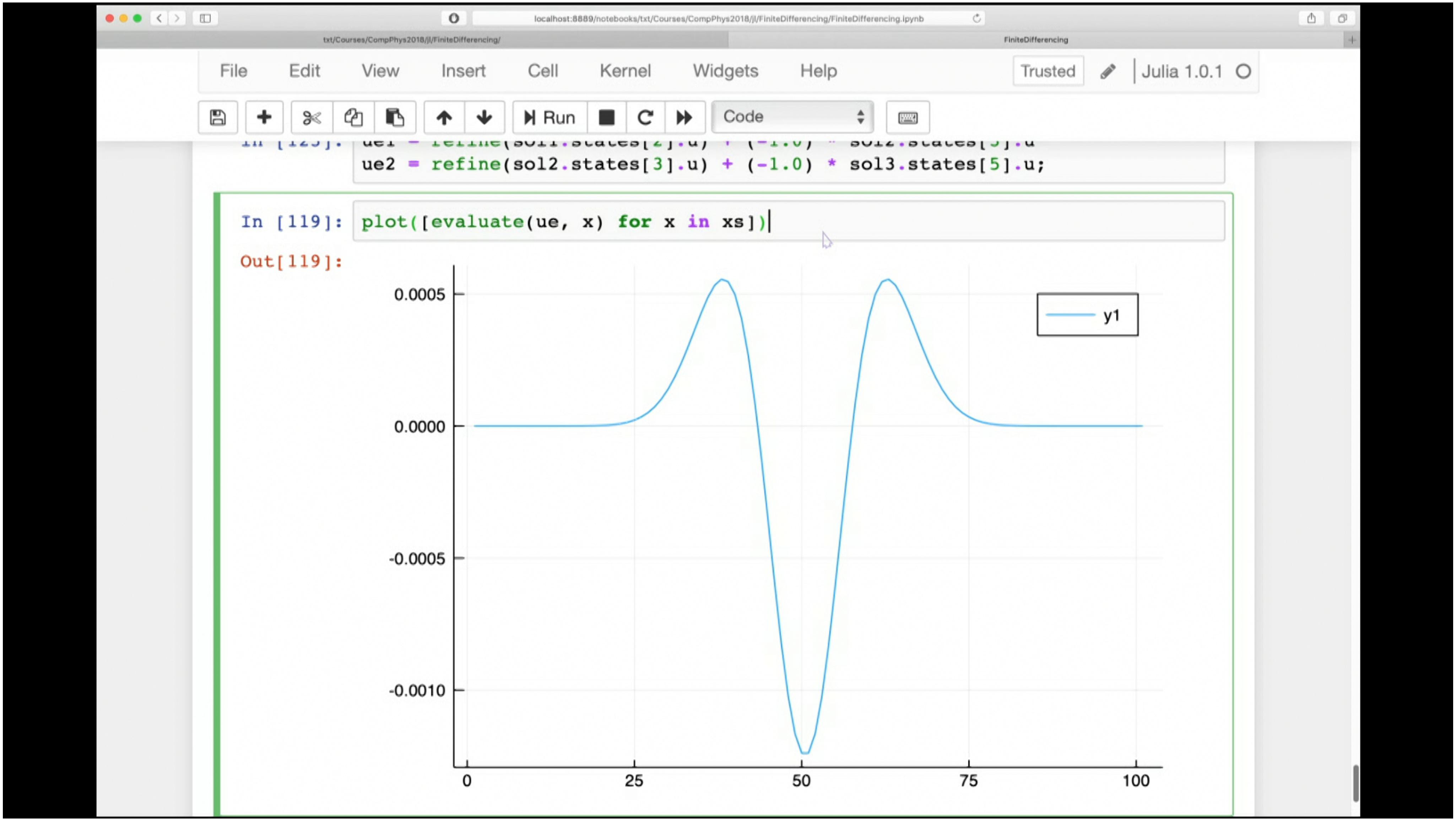
In [118]:
```julia
ue = refine(sol1.states[2].u) + (-1.0) * sol2.states[3].u
```

Out[118]:
```
PLCFun{Float64,Float64}([1.39583e-12, 7.19654e-12, 3.56437e-12, 1.81295e-11, 8.91376e-12, 4.47295e-11, 2.18298e-11, 1.08076e-10, 5.23513e-11, 2.55721e-10 … 1.08076e-10, 2.18298e-11, 4.47295e-11, 8.91376e-12, 1.81295e-11, 3.56437e-12, 7.19654e-12, 1.39583e-12, 3.63388e-12, 0.0])
```

In [119]:
```julia
plot([evaluate(ue, x) for x in xs])
```

Out[119]:




localhost:8889/notebooks/txt/Courses/CompPhys2018/jl/FiniteDifferencing/FiniteDifferencing.ipynb
txt/Courses/CompPhys2018/jl/FiniteDifferencing/
FiniteDifferencing
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Julia 1.0.1
Run
Code
ue2 = refine(sol2.states[3].u) + (-1.0) * sol3.states[5].u;
In [119]: plot([evaluate(ue, x) for x in xs])
Out[119]:
0.0005
0.0000
-0.0005
-0.0010
0
25
50
75
100
y1

File Edit View Insert Cell Kernel Widgets Help Trusted Julia 1.0.1

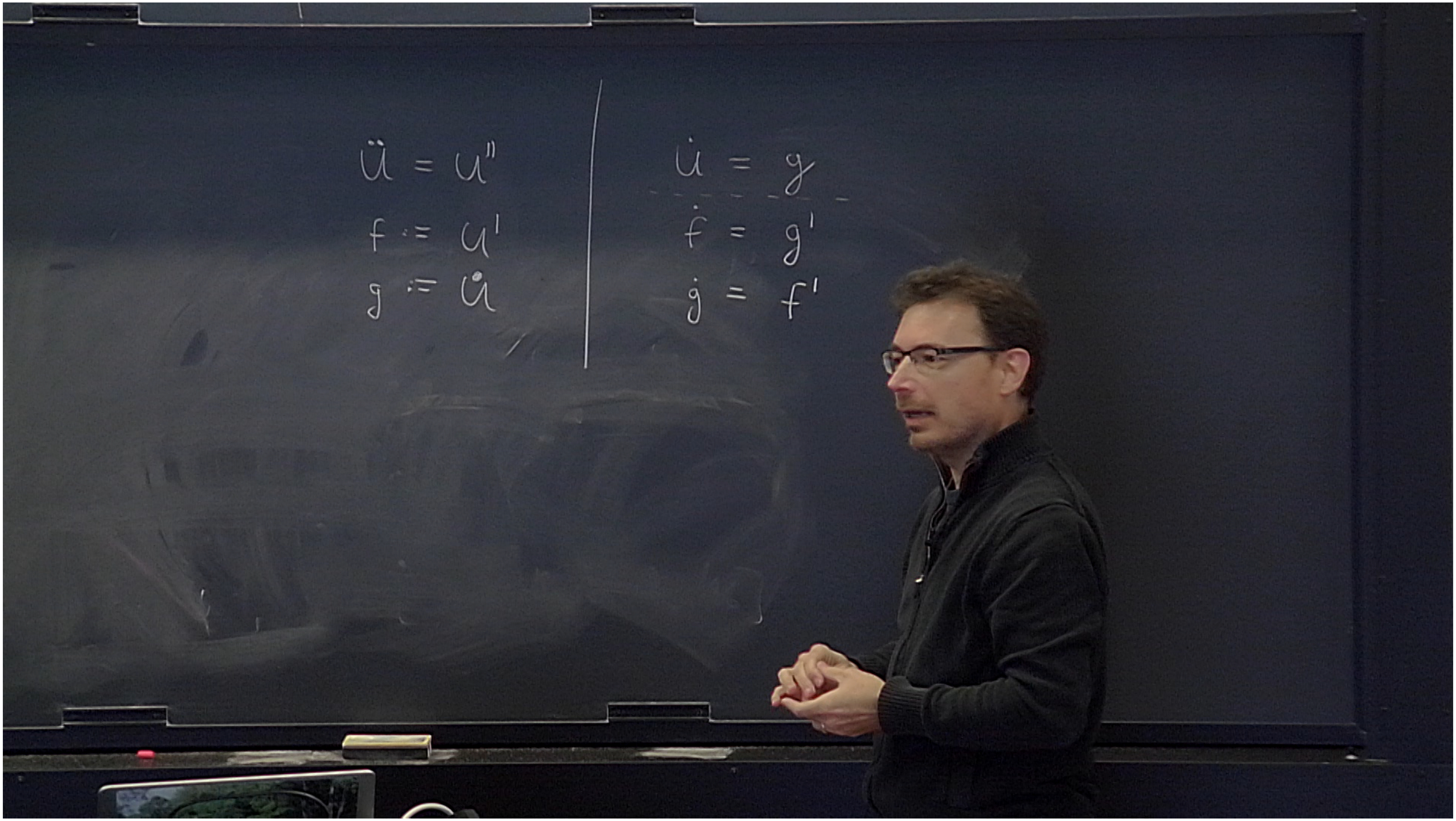
In [17]:
```julia
# The state vector describing how we evolve the advection equation in time
struct AdvectionState{T}
    time::T
    u::PLCFun{T, T}
end
```

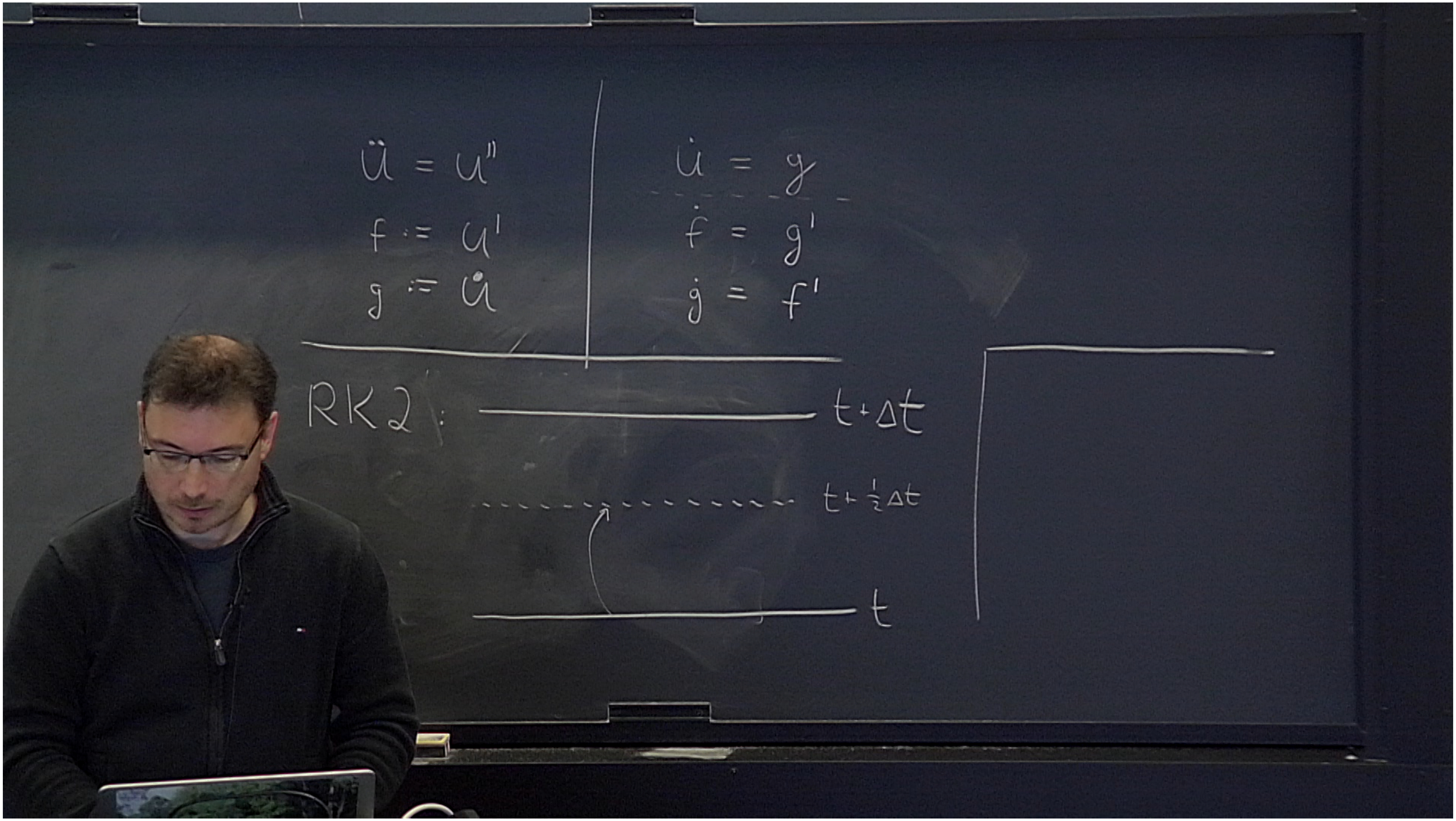
In [37]:
```julia
# State vectors can be scaled and added
function Base. *(a::T, s::AdvectionState{T})::AdvectionState{T} where {T}
    AdvectionState{T}(s.time, a * s.u)
end
function Base. +(s1::AdvectionState{T},
                 s2::AdvectionState{T})::AdvectionState{T} where {T}
    @assert abs(s1.time - s2.time) <= 100*eps(T)
    AdvectionState{T}(s1.time, s1.u + s2.u)
end
```

In [18]:
```julia
# A Gaussian, centred at x=1/2, with a width of 1/10
function gaussian(x::T)::T where {T}
    exp(- ((x - 0.5) * 10)^2)
end
```

Out[18]: gaussian (generic function with 1 method)

In [19]:
```julia
# Define initial conditions: A Gaussian at t=0
function initialGaussian(nlines::Int)::AdvectionState{Float64}
    t = 0
```


$\ddot{u} = u''$
$f := u'$
$g := \dot{u}$
$\dot{u} = g$
$\dot{f} = g'$
$\dot{g} = f'$


ü = u''
f := u'
g := u̇
u̇ = g
ḟ = g'
ġ = f'
RK2 :
t+Δt
t+½Δt
t