```julia
using Colors, Plots, Polynomials, Images, ImageView, PlutoUI, LinearAlgebra, DSP, AbstractFFTs,
Interpolations, CoordinateTransformations, Statistics
```

References

- **fourier series harmonic approximation**

imagesc (generic function with 1 method)

```julia
function imagesc(x; degree=1, colorScheme="oranges")
    # using Polynomials, Colors
    # Blues, Greens, Grays, Oranges, Purples, Reds
    xvec = sort(vec(x))
    p = fit(convert.(Float64, xvec), 1:length(xvec), degree)

    y1 = floor.(Int, p.(x))
    y = (y1 .- minimum(y1)) .+ 1

    cmap = colormap(colorScheme, maximum(y))
    yvec = [cmap[i] for i in y]

    if ndims(x) == 2
        out = reshape(yvec, size(x))
    else
        out = reshape(yvec, (1, length(yvec)))
    end

    return out
end
```

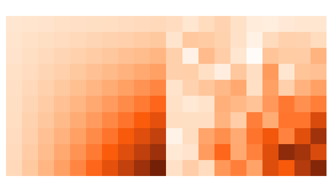
outer (generic function with 1 method)

```julia
outer(v, w) = [x*y for x in v, y in w]
```

```julia
@bind n Slider(2:100, default=10)
```

```julia
@bind k Slider(0:25, default=10)
```

```julia
begin
    d1 = outer(1:n, 1:n);
    d2 = d1 .+ k*randn(n, n);
end;
```



```
imagesc([d1 d2])
```

θ = Float64[-3.14159, -3.13545, -3.12931, -3.12317, -3.11702, -3.11088, -3.10474, -

```
θ = range(-π, π, length=1024)
```

r = Float64[0.0, 9.43077e-6, 3.77227e-5, 8.48748e-5, 0.000150885, 0.000235751, 0.000

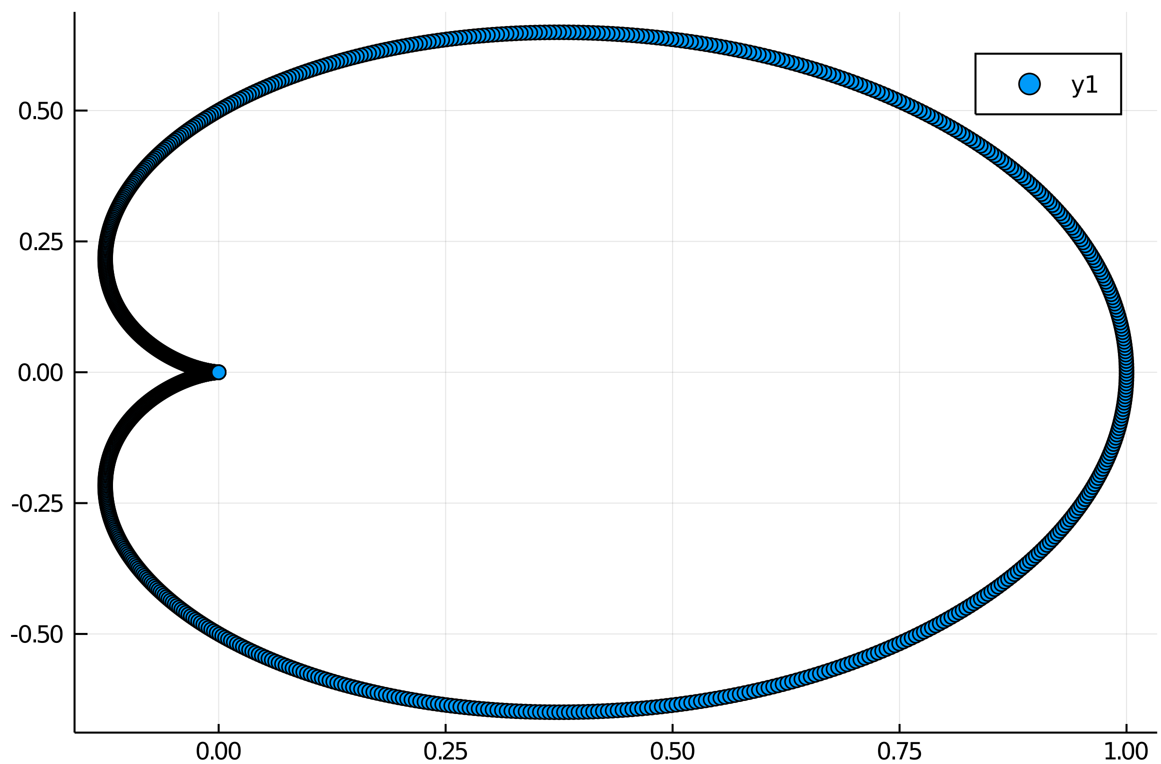
```
r = DSP.Windows.hanning(1024)
```

x = Float64[-0.0, -9.43059e-6, -3.77199e-5, -8.48604e-5, -0.00015084, -0.00023564,

```
x = r .* cos.(θ)
```

y = Float64[-0.0, -5.79227e-8, -4.63368e-7, -1.56379e-6, -3.70653e-6, -7.2387e-6, -

```
y = r .* sin.(θ)
```




```
scatter(x, y)
```

fourierSeries (generic function with 1 method)

```
function fourierSeries(period, N)
    T = length(period)
    tt = collect(range(1, T, step=1))
    result = Array{Float64,2}(undef, N+1, 2)
    for n in 1:N+1
        an = sum(2/T .* (period .* cos.(2*π*n*tt/T)))
        bn = sum(2/T .* (period .* sin.(2*π*n*tt/T)))
        result[n,1] = an
        result[n,2] = bn
    end

    return result
end
```

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

```
function reconstruct(P, anbn)
    result = 0
    t = collect(1:P)
    for n in 1:length(anbn[:,1])
        if n == 1
            anbn[n,1] = anbn[n,1]/2
        end
        result = result .+ anbn[n,1] .* cos.(2*π*n .* t/P) + anbn[n,2] .* sin.(2*π*n .* t/P)
    end
    return result
end
```

```
func4 (generic function with 1 method)
```

```
begin
    func1(t) = Float64.(abs.((t.%1).-0.25) .< 0.25) - Float64.(abs.((t.%1).-0.75) .< 0.25)
    func2(t) = t .% 1
    func3(t) = Float64.(abs.((t.%1).-0.5) .< 0.25) + 8 .* (abs.((t.%1).-0.5)) .* (abs.((t.%1).-0.5).
<0.25)
    func4(t) = ((t.%1).-0.5).^2
end
```

```
Fs = 10000
```

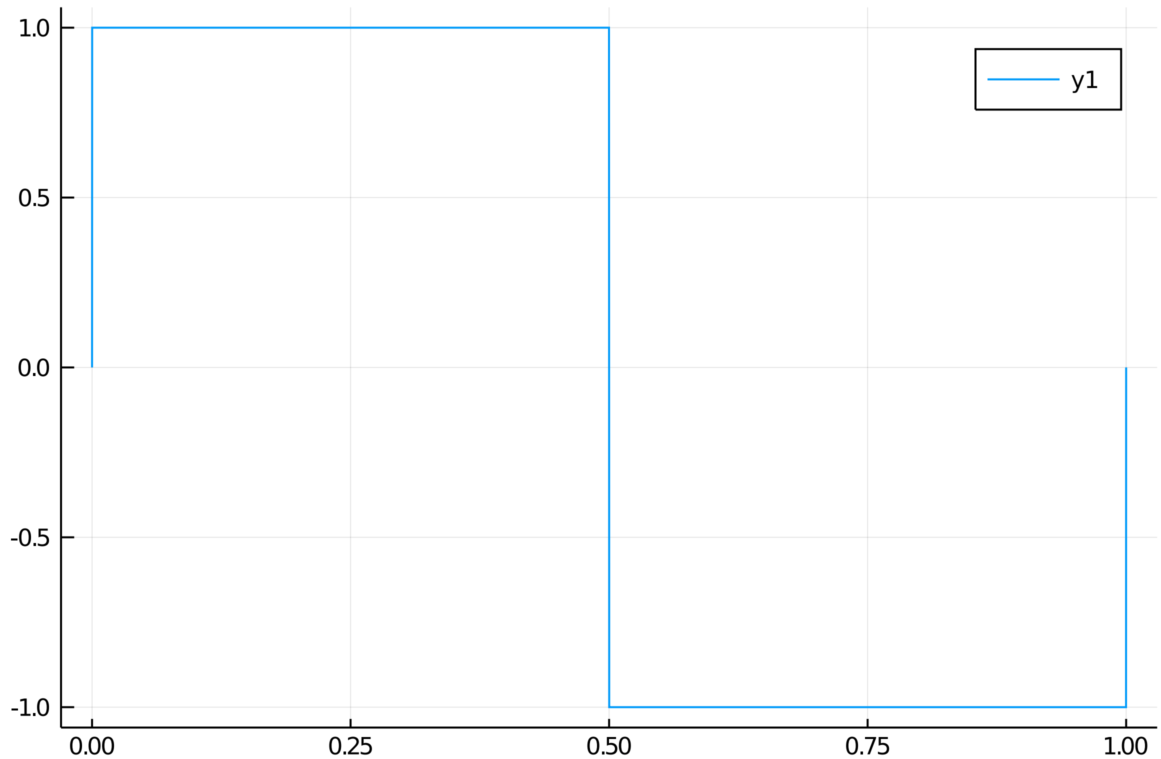
```
Fs = 10000
```

```
t = Float64[0.0, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, (
```

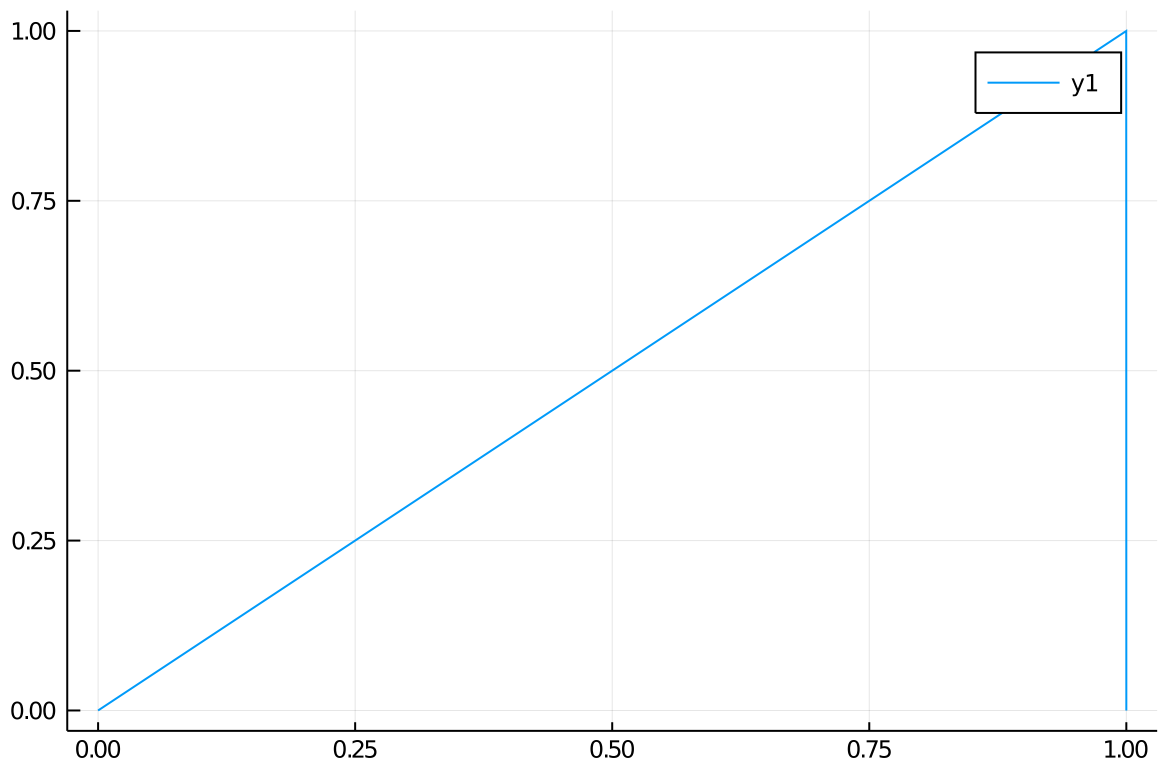
```
t = collect(range(0, 1, step=1/Fs))
```

```
11×2 Array{Float64,2}:
 -0.00039995995466629575   1.2732393458074247
  1.183997067055742e-10   -1.884578099131673e-7
 -0.0003999595600005512    0.4244125847953966
  4.735985908999041e-10   -3.7691528520511497e-7
 -0.0003999587706698393    0.25464691430945674
  1.0655959170602358e-9   -5.653720909909454e-7
 -0.00039995758667499964   0.18188997104254778
  1.894390690322656e-9    -7.538278924690278e-7
 -0.0003999560080177045    0.14146927018317032
  2.959981128258793e-9    -9.422823551291648e-7
 -0.00039995403469955323   0.1157468613285123
```

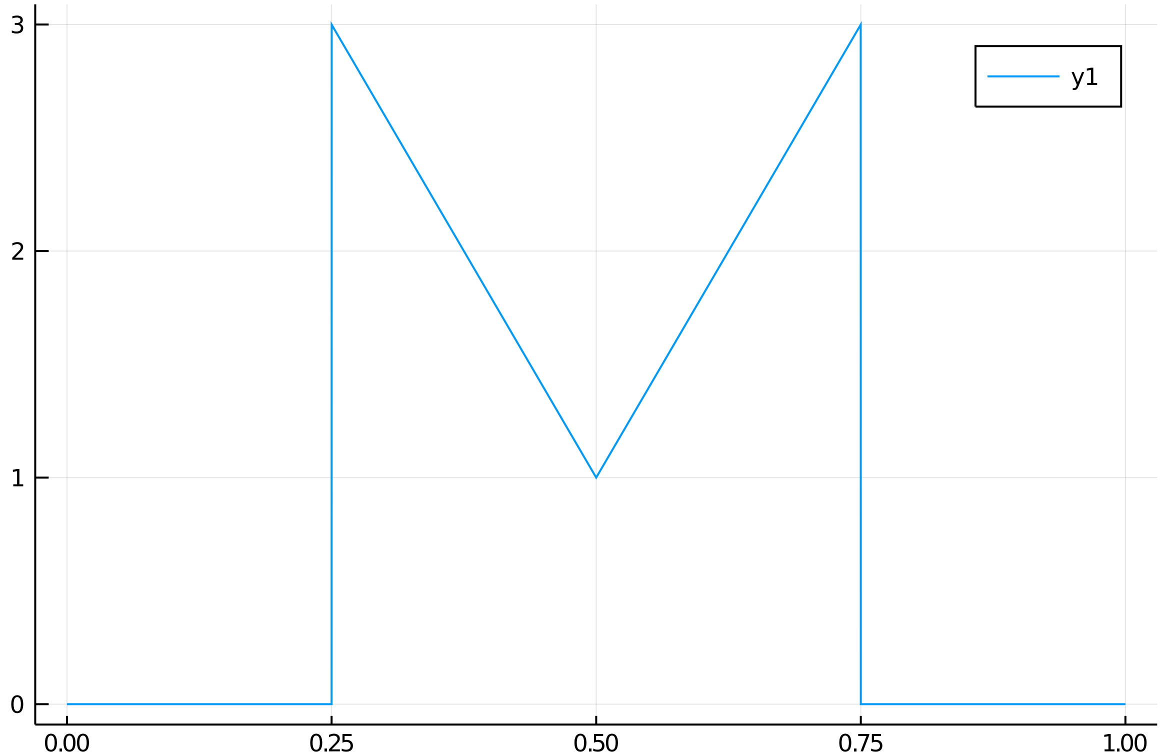
```
fourierSeries(func1(t), 10)
```



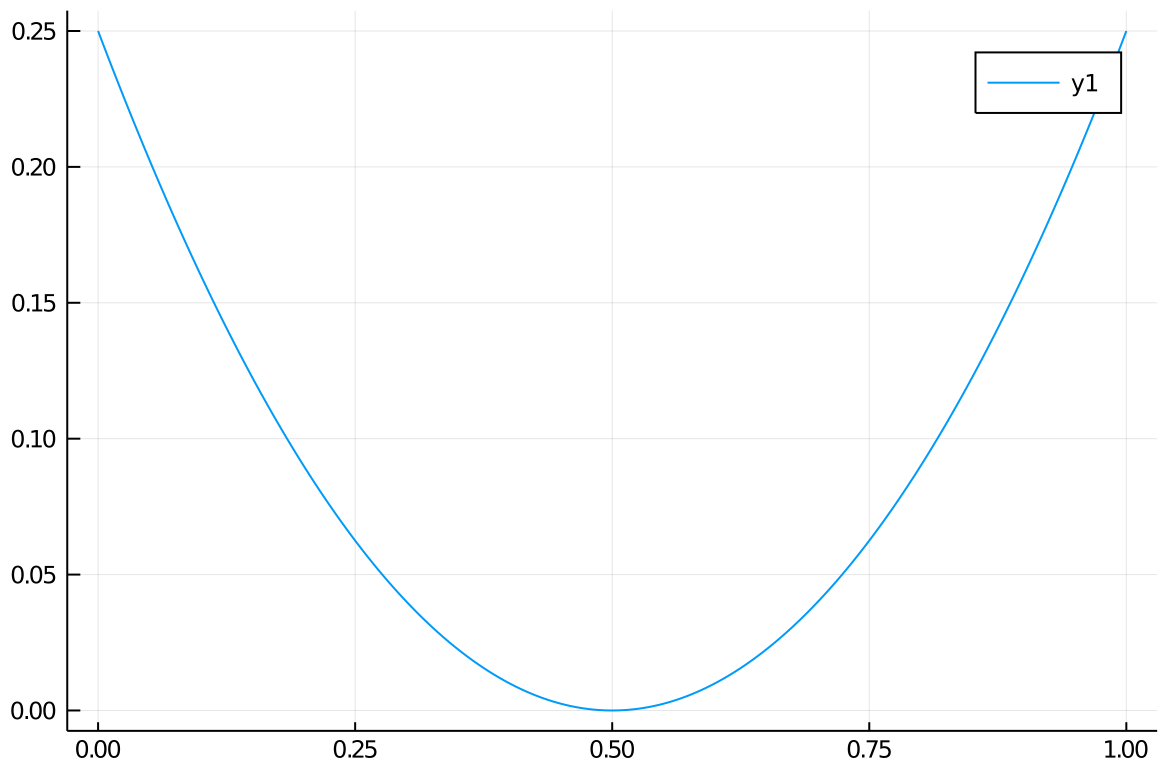

```
plot(t, func1(t))
```




```
plot(t, func2(t))
```




```
plot(t, func3(t))
```




```
plot(t, func4(t))
```

```
fi = Float64[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
```

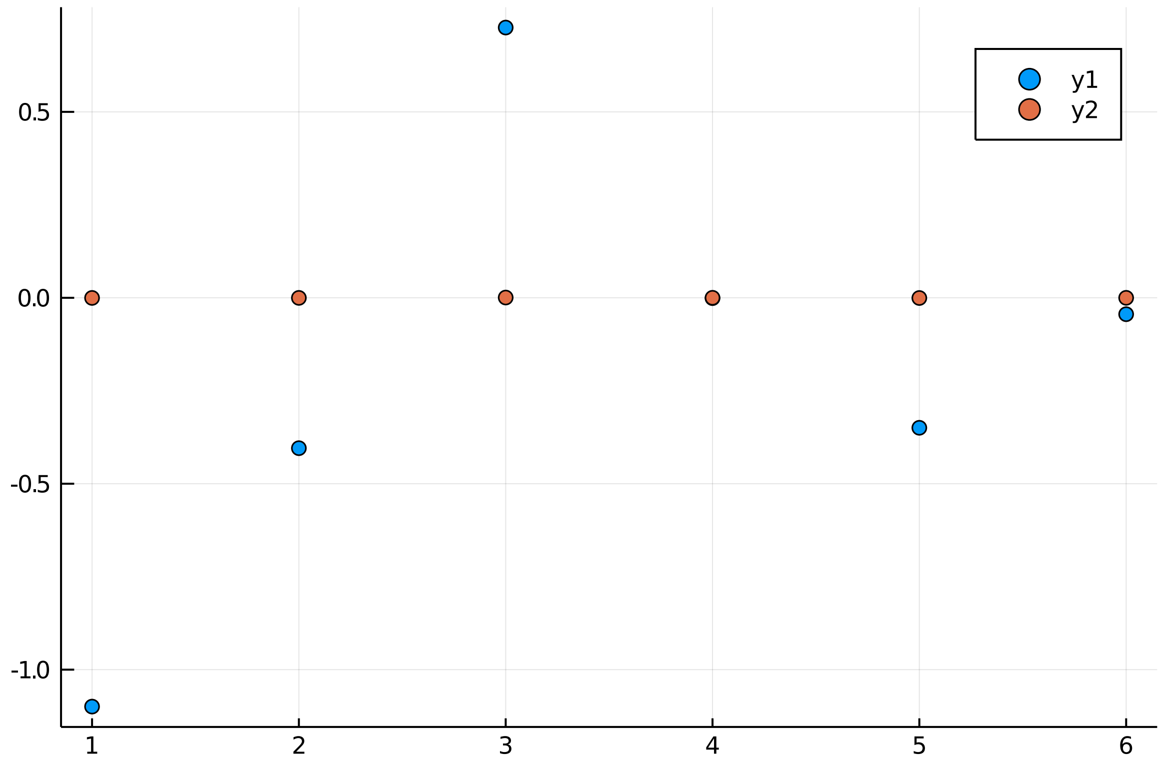
```
fi = func3(t)
```

```
F = 6×2 Array{Float64,2}:
 -1.0993358534666076     -0.00034533202226538773
 -0.404425316622405      -0.0002540825459030893
  0.7267336565191823      0.0006848620514848802
 -0.0008999190015989283  -1.130759088779909e-6
 -0.3495699463163159     -0.0005490487342872385
 -0.04413619593045691    -8.318654919287072e-5
```

```
F = fourierSeries(fi, 5)
```

```
Array{Float64,2}
```

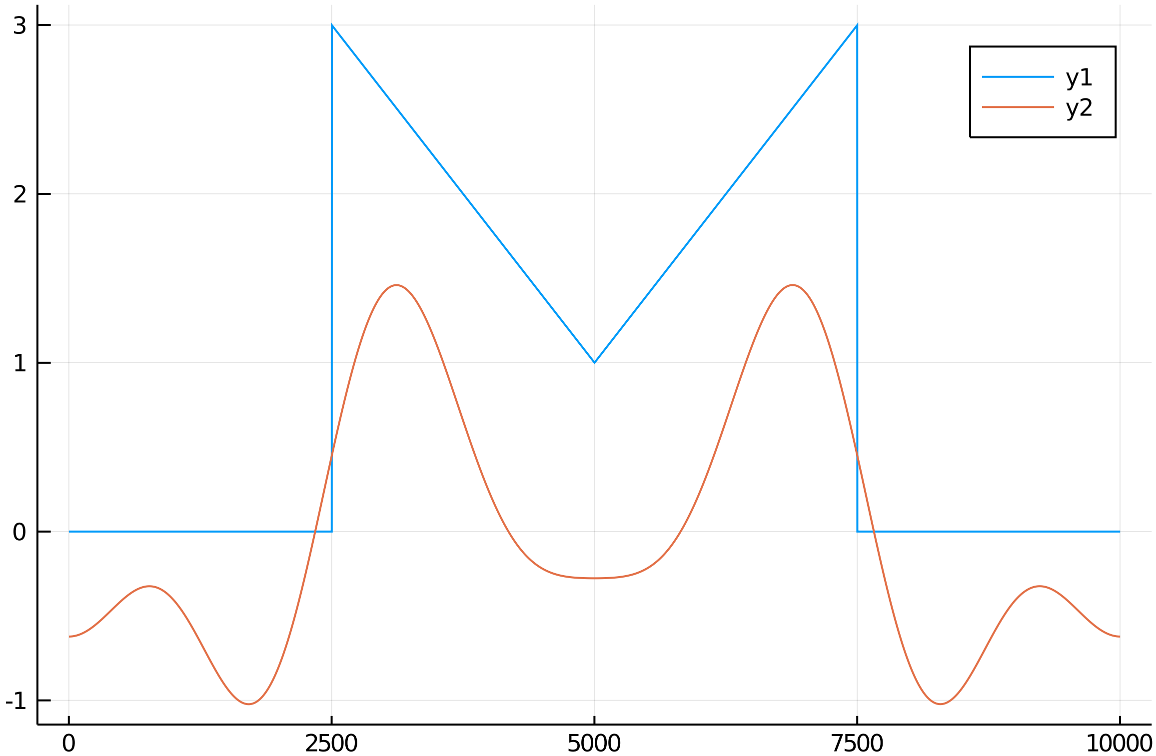
```
typeof(F)
```




```
scatter(F)
```

```
fo = Float64[-0.621966, -0.621964, -0.621959, -0.621952, -0.621943, -0.621931, -0.6:
```

```
fo = reconstruct(length(t), F)
```




```
plot([fi, fo])
```

θ2 = Float64[-3.14159, -3.14096, -3.14034, -3.13971, -3.13908, -3.13845, -3.13782,

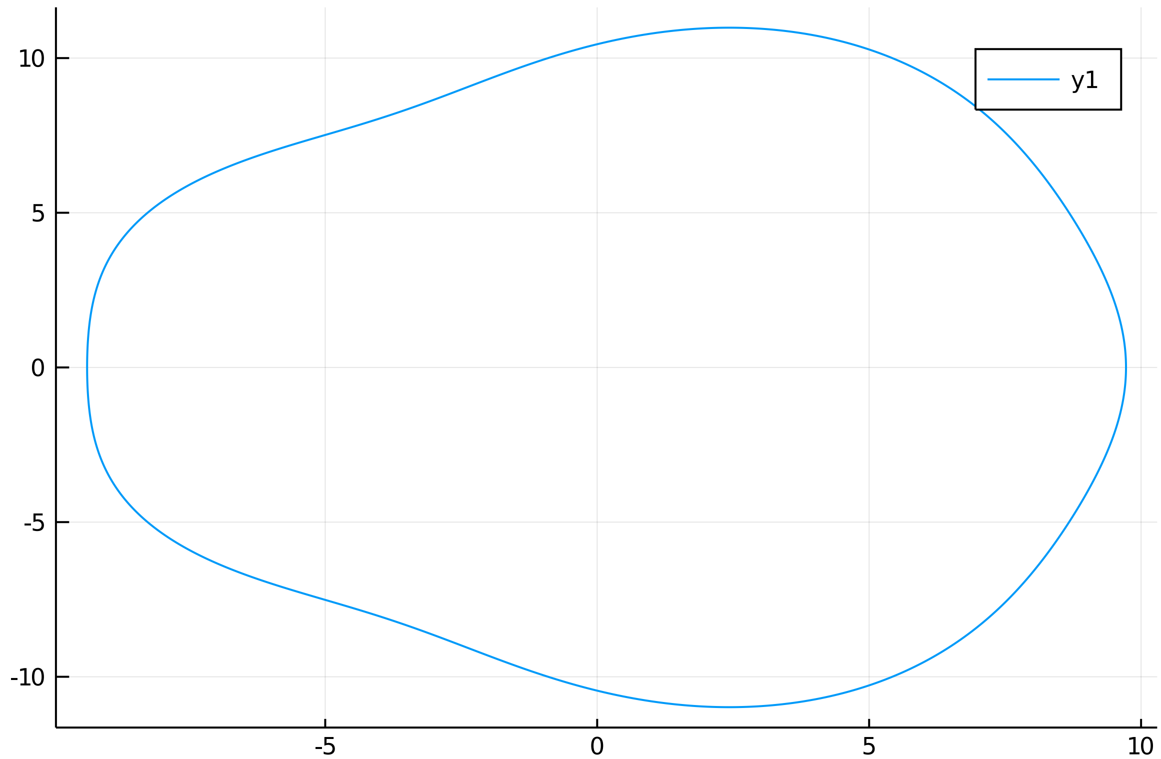
```
θ2 = range(-π, π, length=length(fi))
```

xi = Float64[-9.37803, -9.37803, -9.37803, -9.37803, -9.37803, -9.37802, -9.37802,

```
xi = (10 .+ fo) .* cos.(θ2)
```

yi = Float64[-1.14848e-15, -0.00589239, -0.0117848, -0.0176772, -0.0235696, -0.02946

```
yi = (10 .+ fo) .* sin.(θ2)
```



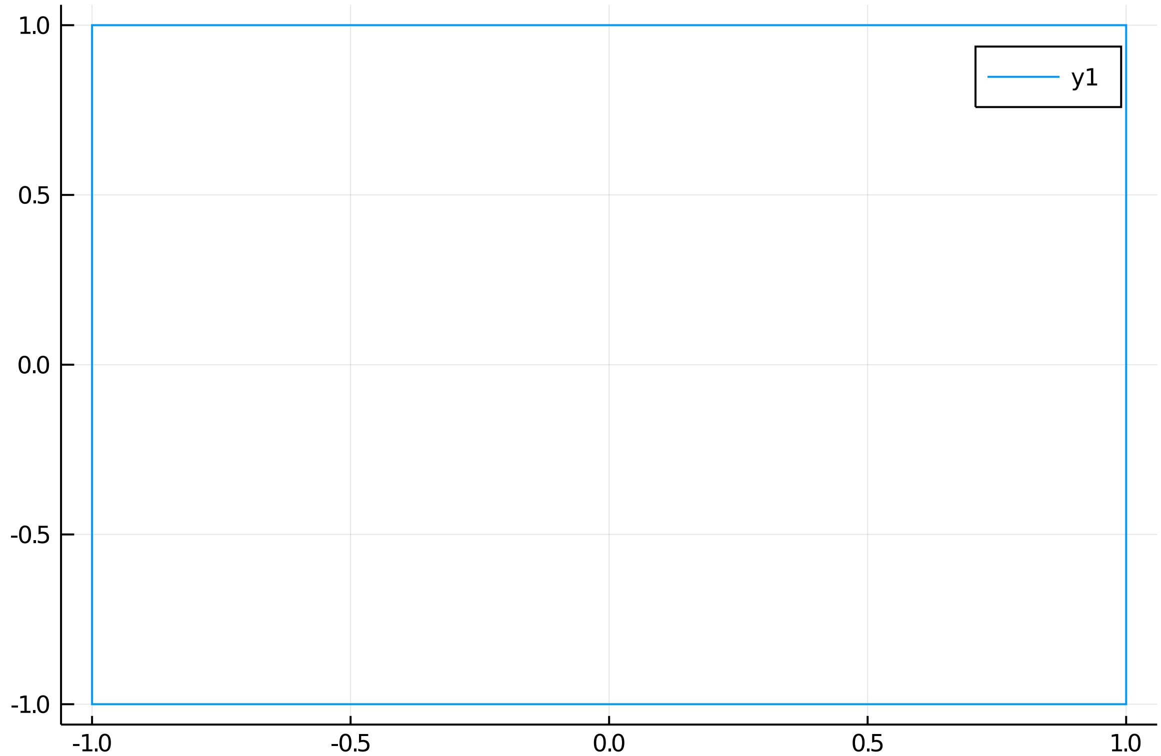

```
plot(xi, yi)
```

squareShapeX = Float64[-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0

```
squareShapeX = [-1 .* ones(10); range(-1, 1, length=20); ones(20); range(1, -1, length=20); -1 .*
ones(10)]
```

squareShapeY = Float64[0.0, -0.111111, -0.222222, -0.333333, -0.444444, -0.555556,

```
squareShapeY = [range(0, -1, length=10); -1 .* ones(20); range(-1, 1, length=20); ones(20); range(1, 0,
length=10)]
```




```
plot(squareShapeX, squareShapeY)
```

squareShapeR = Float64[1.0, 1.00615, 1.02439, 1.05409, 1.09432, 1.14396, 1.20185, 1

```
squareShapeR = [sqrt(squareShapeX[p]^2+squareShapeY[p]^2) for p in 1:length(squareShapeX)]
```

squareShapeθ = Float64[3.14159, 3.03094, 2.92292, 2.81984, 2.72337, 2.63449, 2.5535

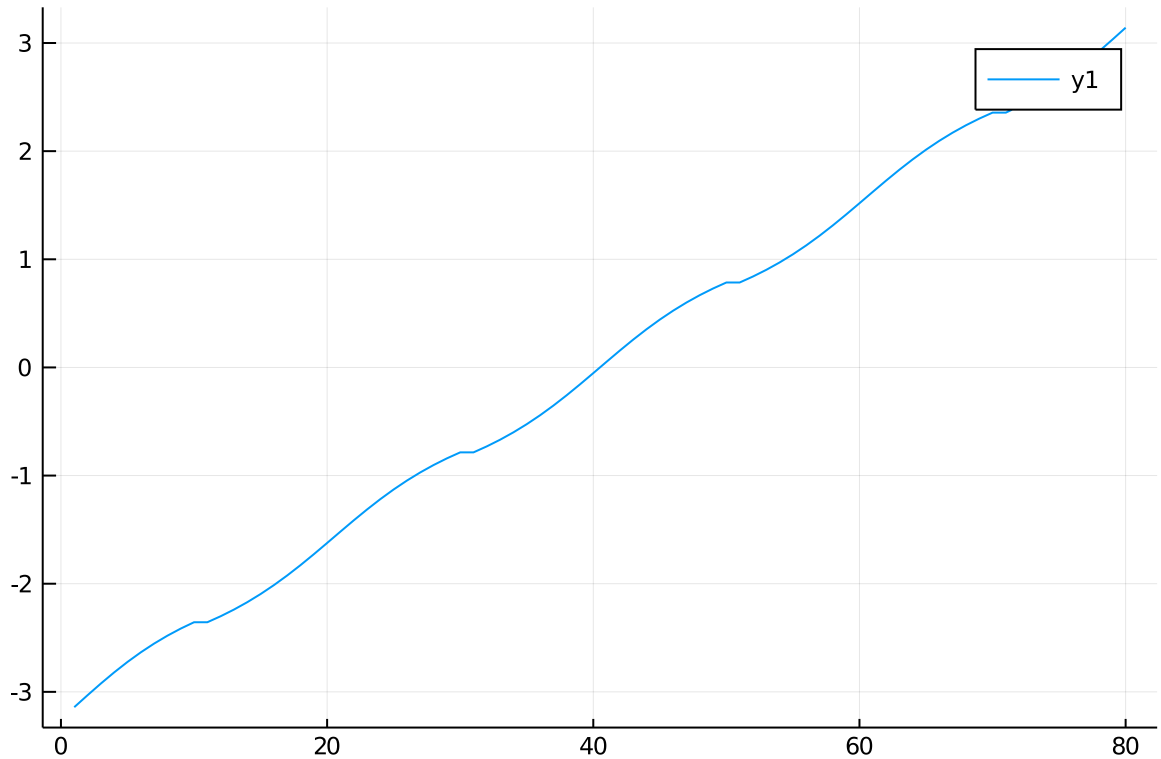
```
squareShapeθ = [acos(squareShapeX[p]/squareShapeR[p]) for p in 1:length(squareShapeX)]
```

(0.0525831, 3.14159)

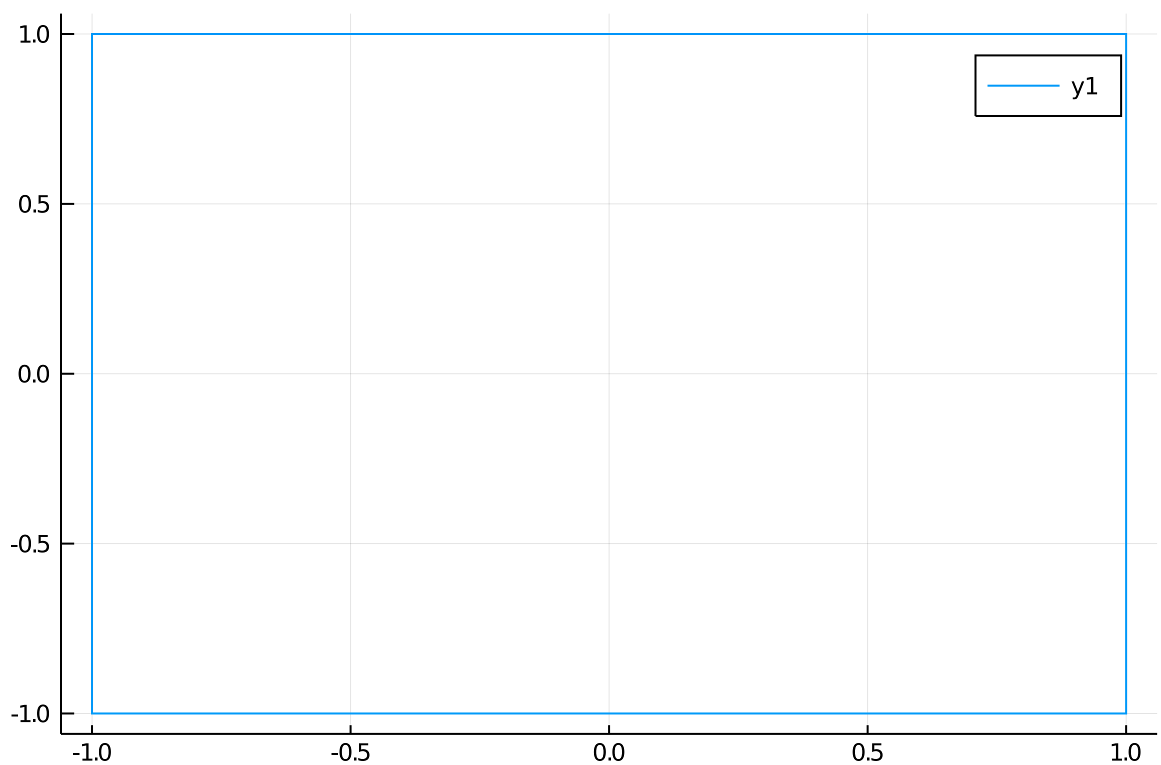
```
Enter cell code...
```

Float64[-3.14159, -3.03094, -2.92292, -2.81984, -2.72337, -2.63449, -2.55359, -2.4

```
squareShapeθ[1:length(squareShapeθ)÷2] .*= -1
```




```
plot(squareShapeθ)
```



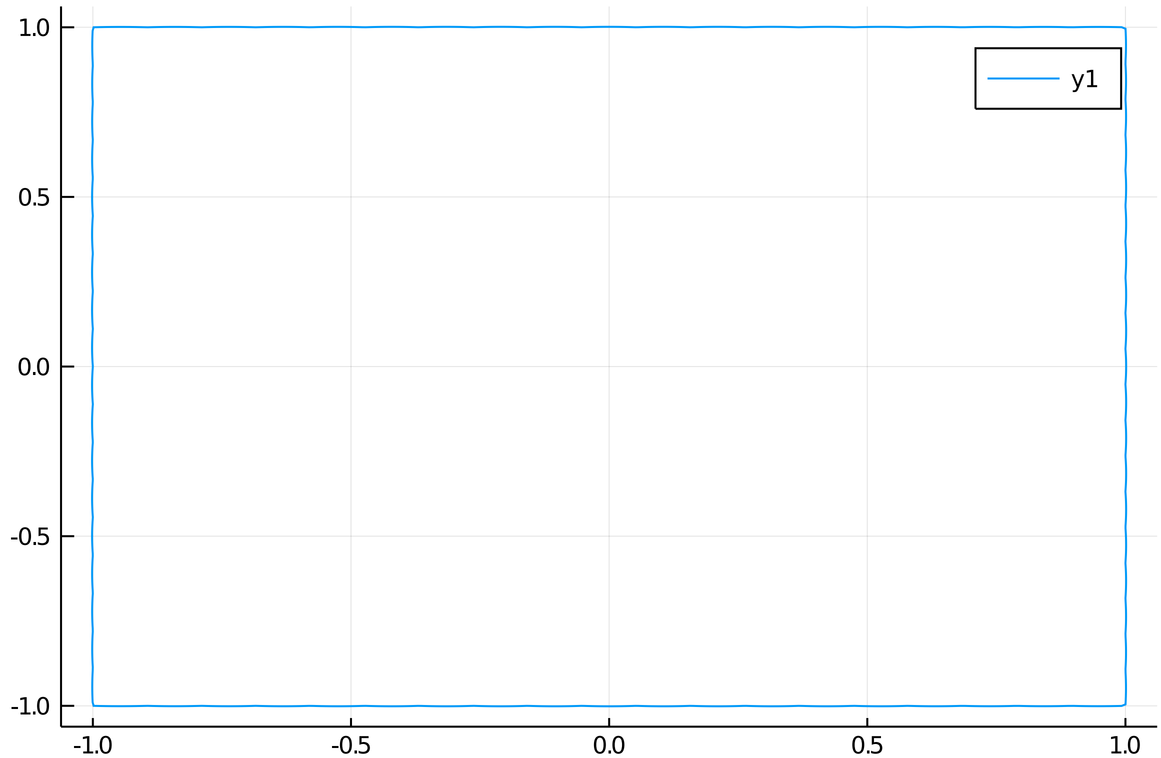

```
plot(squareShapeR .* cos.(squareShapeθ), squareShapeR .* sin.(squareShapeθ))
```

```
squareShapeθi = Float64[-3.14159, -3.13545, -3.12931, -3.12317, -3.11702, -3.11088,
```

```
squareShapeθi = range(-π, π, length=1024)
```

```
squareShapeRi =  Float64[1.0,  1.00034,  1.00068,  1.00102,  1.00137,  1.00171,  1.00205,
```

```
· squareShapeRi = LinearInterpolation(squareShapeθ, squareShapeR).(squareShapeθi)
```



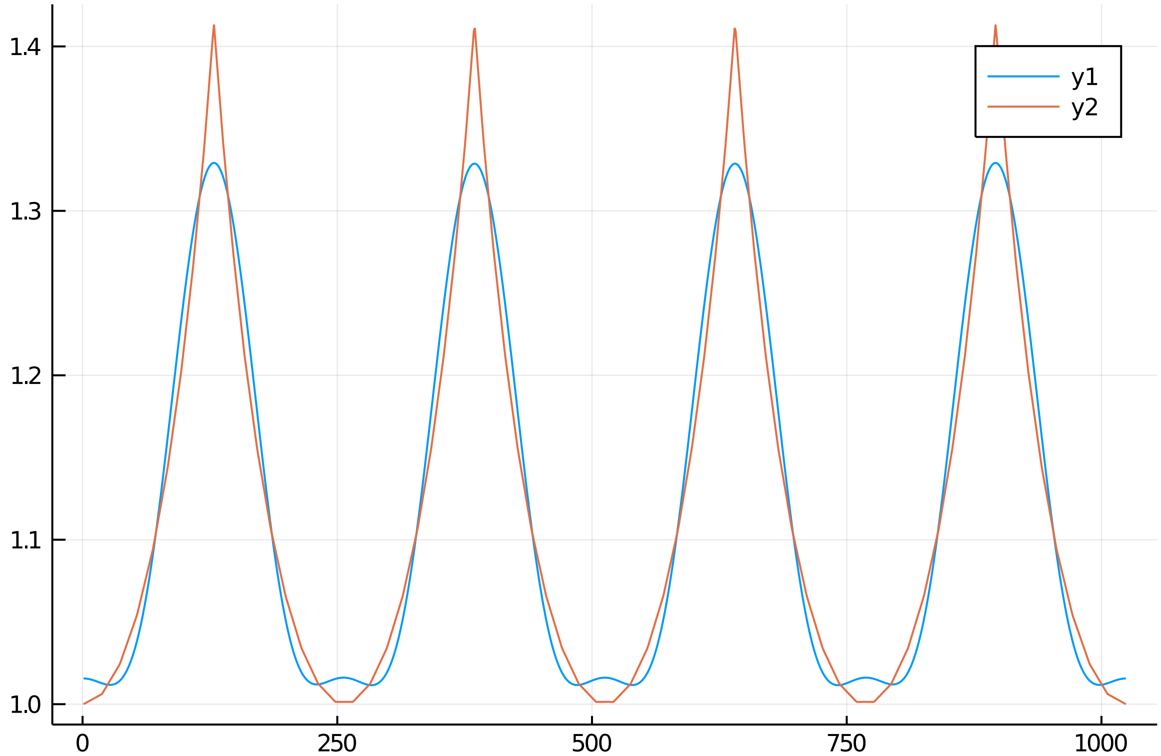

```
· plot(squareShapeRi .* cos.(squareShapeθi), squareShapeRi .* sin.(squareShapeθi))
```

```
squaredHarmonics = 11×2 Array{Float64,2}:
                    -0.00020749423397290823  -6.365863343450282e-7
                    -0.0003028717873173781   -1.8584213347057512e-6
                    -0.0006045551067440269   -5.564412637878708e-6
                    -0.15645375221884686     -0.00192007278824769
                     0.0006509658778407613    9.986474818874888e-6
                     0.00040790133825275763   7.5094019908403425e-6
                     0.0005003324144285763    1.0746656553341594e-5
                     0.049376768948812745     0.0012121316418840942
                    -0.0003630162499221896   -1.0026027244037101e-5
                    -0.0002281434413228386   -7.001549965484052e-6
                    -0.000309483010721737    -1.0448268646425291e-5
```

```
· squaredHarmonics = fourierSeries(squareShapeRi, 10)
```

```
squareShapeRiRecon =  Float64[-0.10743,  -0.107453,  -0.107499,  -0.107568,  -0.107658,  -
```

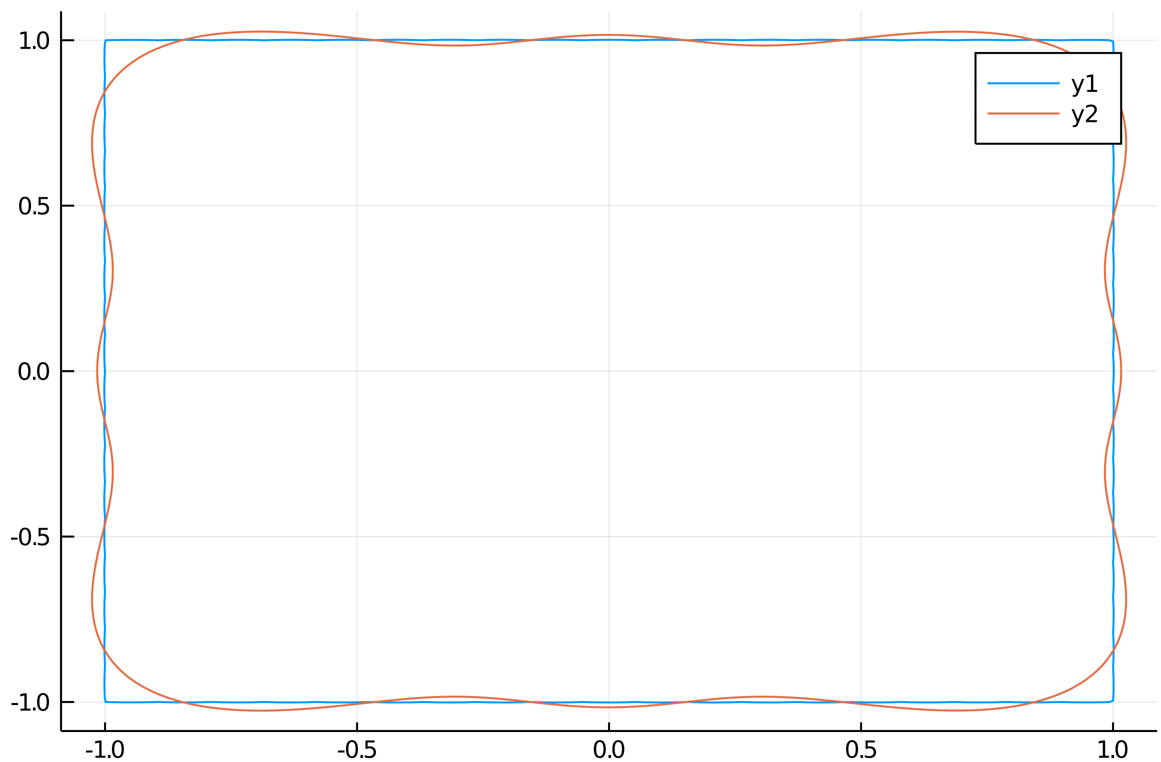
```
· squareShapeRiRecon = reconstruct(length(squareShapeRi), squaredHarmonics)
```




```
plot([squareShapeRiRecon .+ mean(squareShapeRi), squareShapeRi])
```

```
Float64[1.01565, 1.01562, 1.01558, 1.01551, 1.01542, 1.01531, 1.01518, 1.01503, 1.
```

```
squareShapeRiRecon .+= mean(squareShapeRi)
```




```
begin
    plot(squareShapeRi .* cos.(squareShapeθi), squareShapeRi .* sin.(squareShapeθi))
    plot!(squareShapeRiRecon .* cos.(squareShapeθi), squareShapeRiRecon .* sin.(squareShapeθi))
```

```
end
```