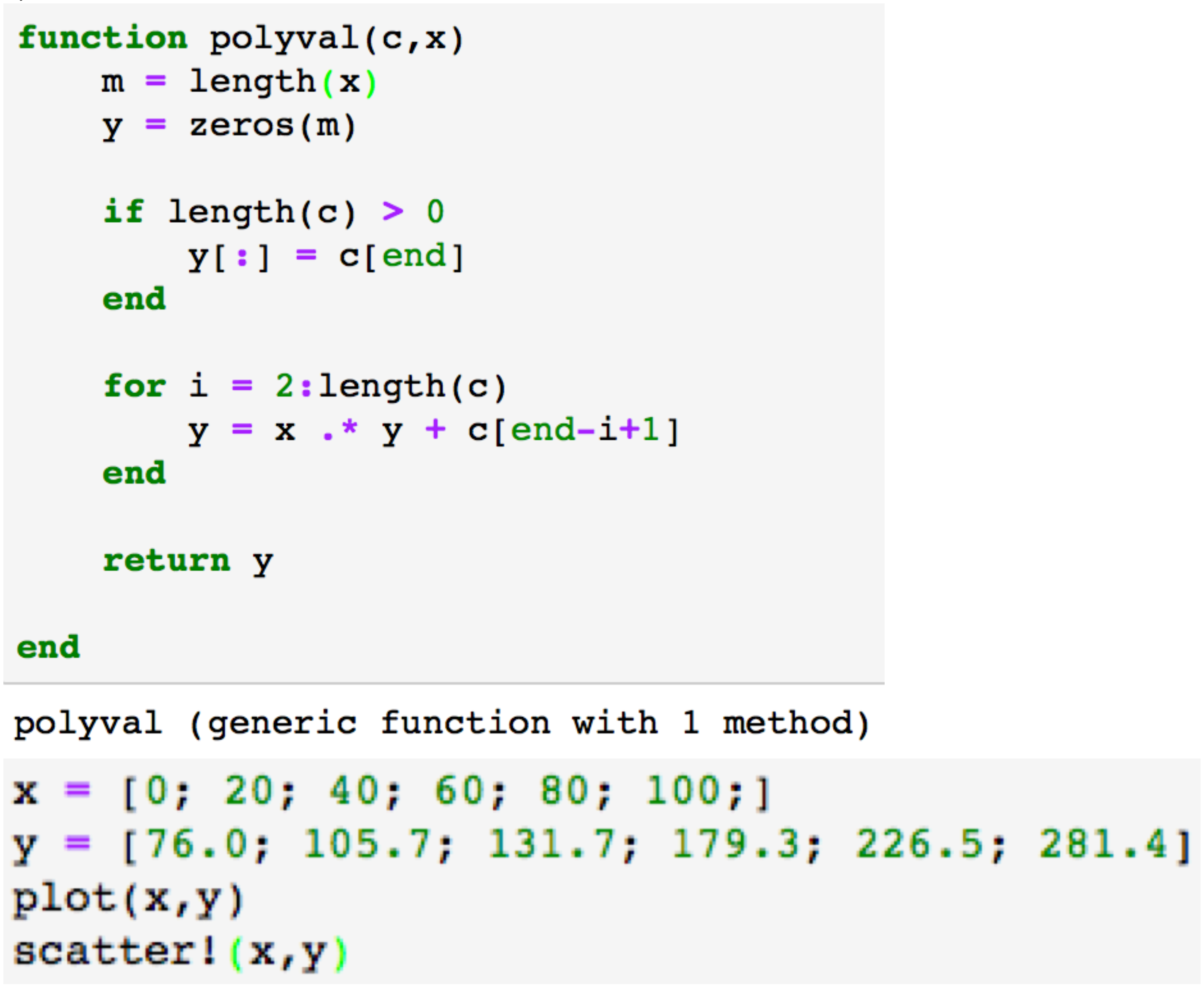
Tian Qiu CS314 HW5

Problem 1

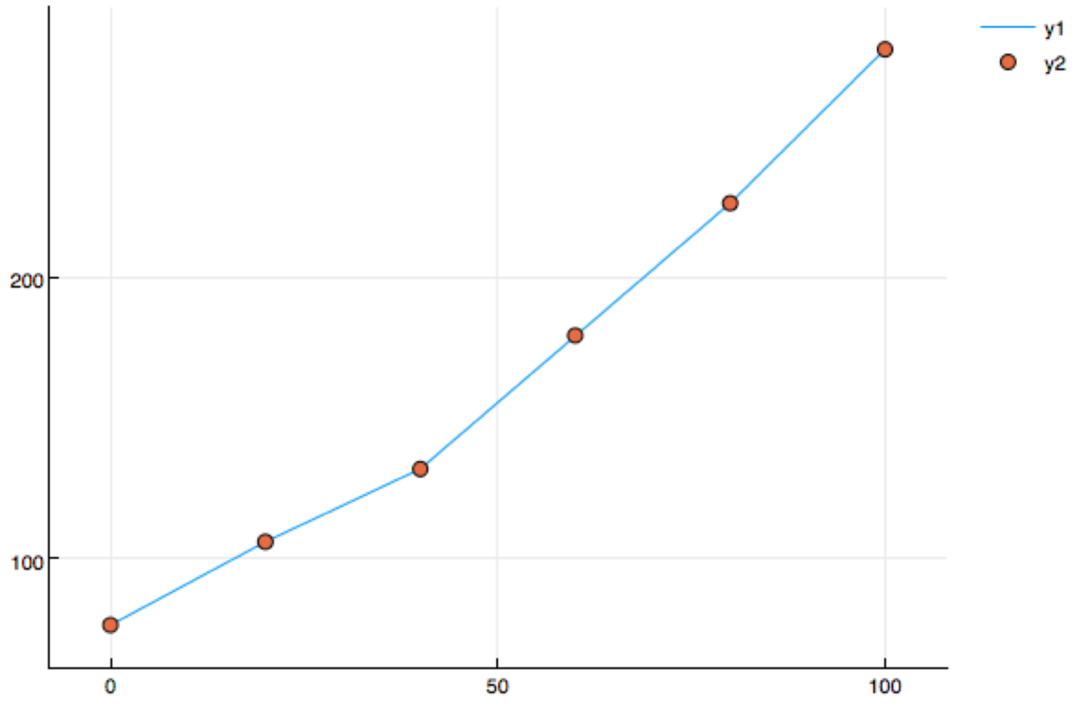
1. a).

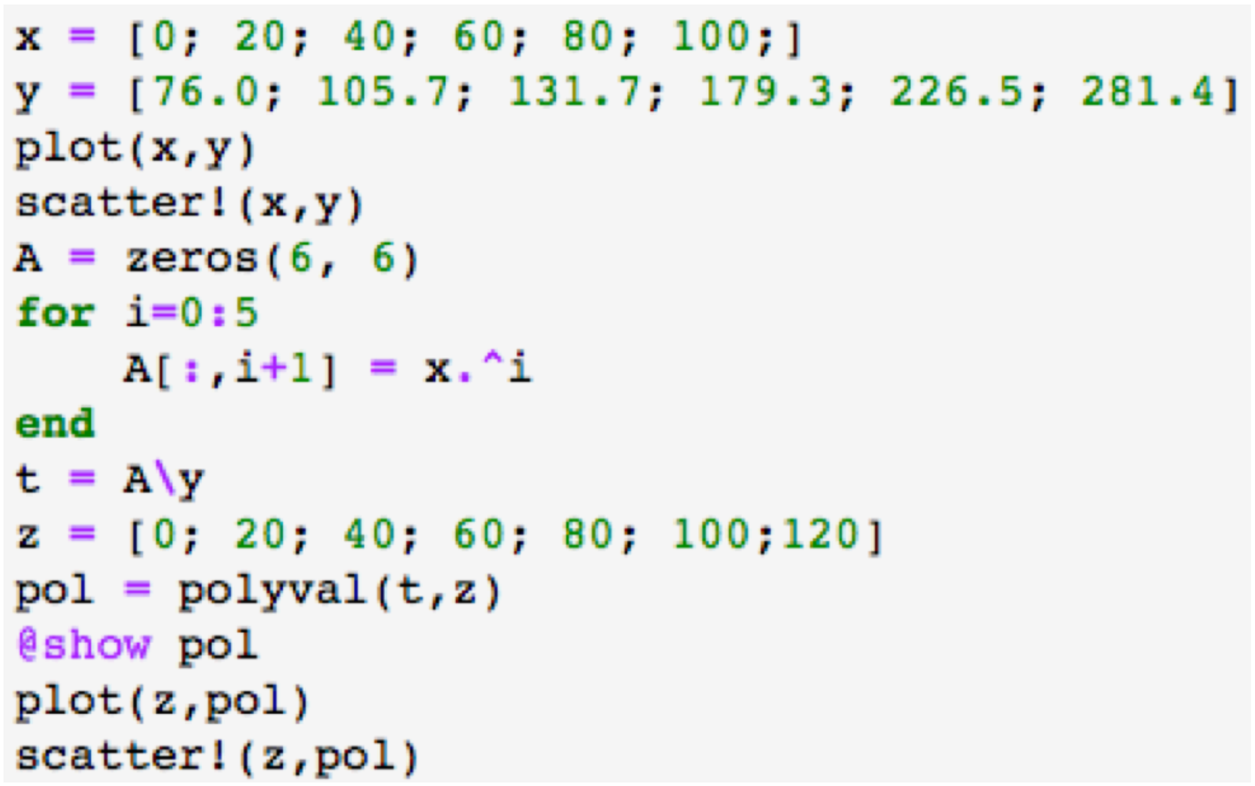


```
function polyval(c,x)
    m = length(x)
    y = zeros(m)

    if length(c) > 0
        y[:] = c[end]
    end

    for i = 2:length(c)
        y = x .* y + c[end-i+1]
    end

    return y

end
```

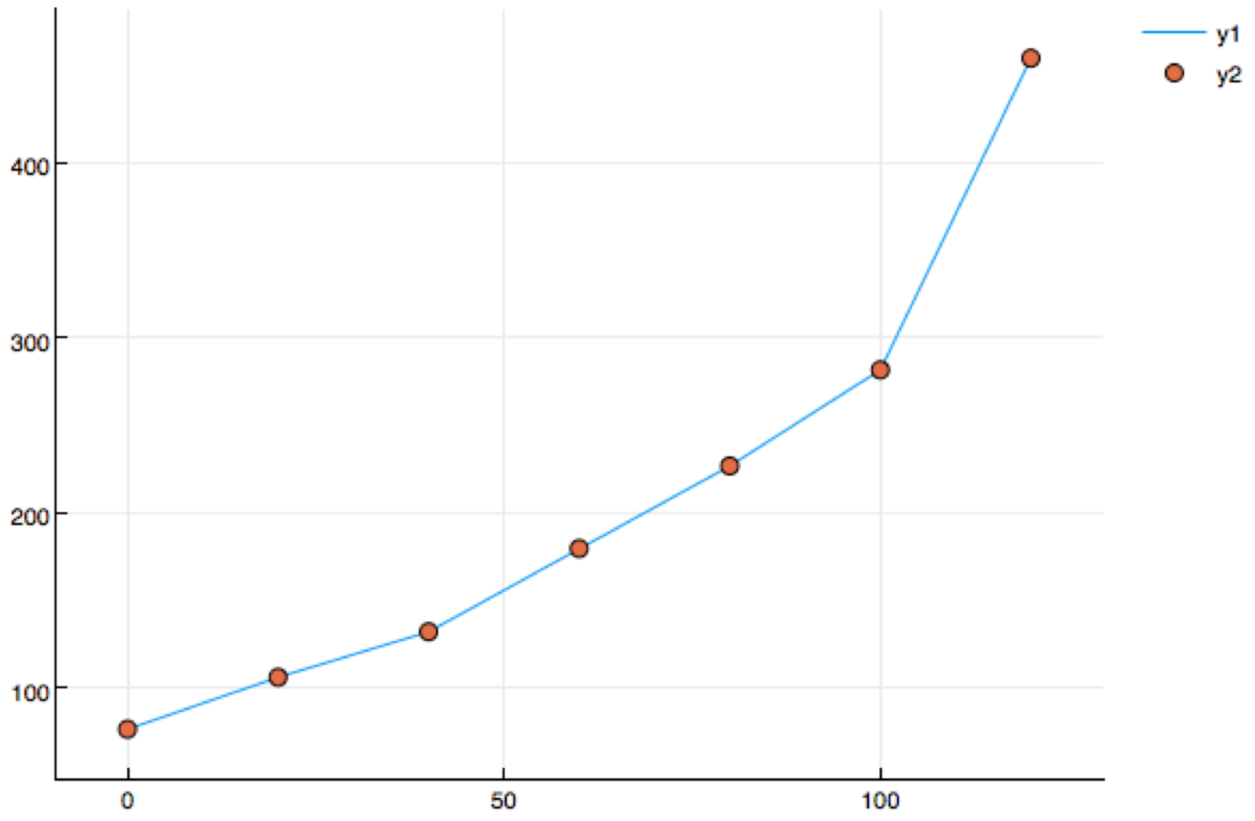
```
polyval (generic function with 1 method)
```

```
x = [0; 20; 40; 60; 80; 100;]
y = [76.0; 105.7; 131.7; 179.3; 226.5; 281.4]
plot(x,y)
scatter!(x,y)
```





```
x = [0; 20; 40; 60; 80; 100;]
y = [76.0; 105.7; 131.7; 179.3; 226.5; 281.4]
plot(x,y)
scatter!(x,y)
A = zeros(6, 6)
for i=0:5
    A[:,i+1] = x.^i
end
t = A\y
z = [0; 20; 40; 60; 80; 100;120]
pol = polyval(t,z)
@show pol
plot(z,pol)
scatter!(z,pol)
```



I think it is a reasonable way to estimate the population of the US in the future years. However, the data points are quite small. Based on this these data cannot give us an accurate estimation even we used the right method.

The population in 2020 based on interpolation will be around 460 million.

1 b). F(x) =

$$76.0 * \frac{(x-20)(x-40)}{(0-20)(0-40)} + 105.7 * \frac{(x-0)(x-40)}{(20-0)(20-40)} + 131.7 * \frac{(x-0)(x-20)}{(40-0)(40-20)}$$

2.

3.

```
function barylag(x,y,xx)
    # direct port of
    #
http://www.mathworks.com/matlabcentral/fileexcha
nge/...
    #    4478-barycentric-lagrange-
interpolating-polynomials-...
    #    and-lebesgue-constant/content/barylag.m
    # to Julia. Not that better implmentations
in Julia are possible.
    M = length(x)
    N = length(xx)
    @assert M == length(y)
    X = repmat(x,1,M)
    W = repmat(1./prod(X-X'+eye(M),1),N,1)
    xdist=repmat(xx,1,M)-repmat(x',N,1)
    fixi,fixj = findnz(xdist.==0)
    H=W./xdist
    p=vec((H*y)./sum(H,2))
    p[fixi] = y[fixj]
    return p
end
```

Problem 2

The Chebyshev interpolation points in (8.15) are defined for the interval [−1, 1]. Suppose we wish to approximate a function on an interval [a, b]. Write down the linear transformation that maps the interval [−1, 1] to [a, b], with (a) = −1 and (b) = 1.

1. $l(x) = -1 + 2 * \frac{(x-a)}{(b-a)}$

$$= -1 + \frac{2x - 2a}{(b-a)}$$

$$= \frac{2x-a-b}{(b-a)}$$

2.

```
function chebspace(a,b,n)
    Cpoints = zeros(Float64, n)
    if n == 1
        return b
    else
        for i=0:n-1
            Cpoints[i+1] = a+(b-a)/2 + (b-a)/(n-1)*cos(pi*(i/(n)));
        end
    end
    return Cpoints
end
```

```
function chebspace(a,b)
    return chebspace(a,b,100)
end
```

Problem 3

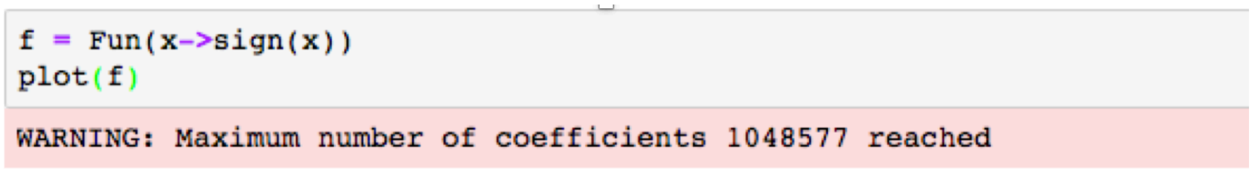
I have learnt that the polynomial interpolation is a method to interpolate the given data set by a polynomial. Even when the data set is not continuous, the polynomial interpolation can go exactly through these points. For polynomial interpolation, we have many methods to accomplish this task like Vandermonde system, least squares and Lagrange interpolation. Also, we learnt the interpolation at Chebyshev points and the method to judge the error of polynomial interpolation.

I definitely used the polynomial interpolation before. One example is that in introduction to statistics course, we used least square method to get the linear interpolation. Also, we learnt Lagrange interpolation in high school chemical experiment course to calculate the equations of a data set which contains experimental data and control data.

Problem 4

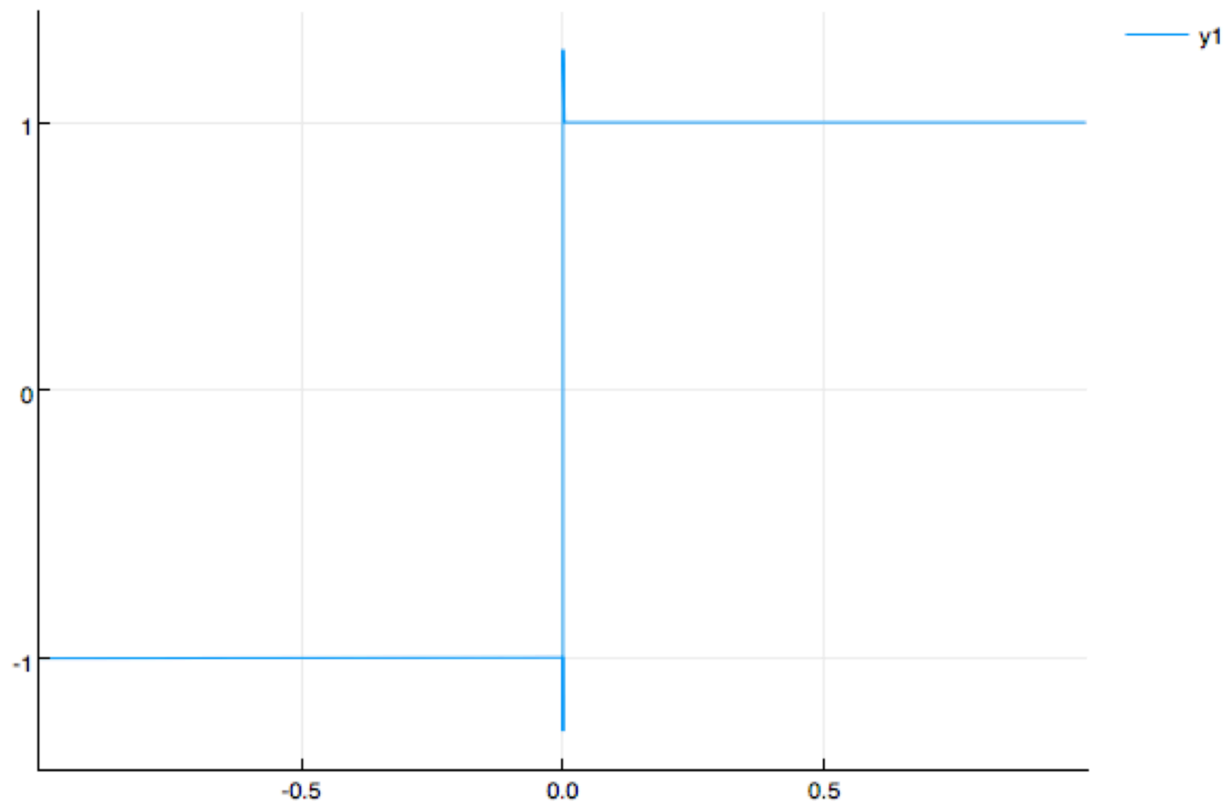
1. Done.

2.



```
f = Fun(x->sign(x))
plot(f)
```

```
WARNING: Maximum number of coefficients 1048577 reached
```



To use Weistrauss approximation theorem, we need the function to be continuous from -1 to 1. As we can see from the graph, it is a continuous function defined on [-1, 1]. So, the Weistrauss approximation theorem can be used.

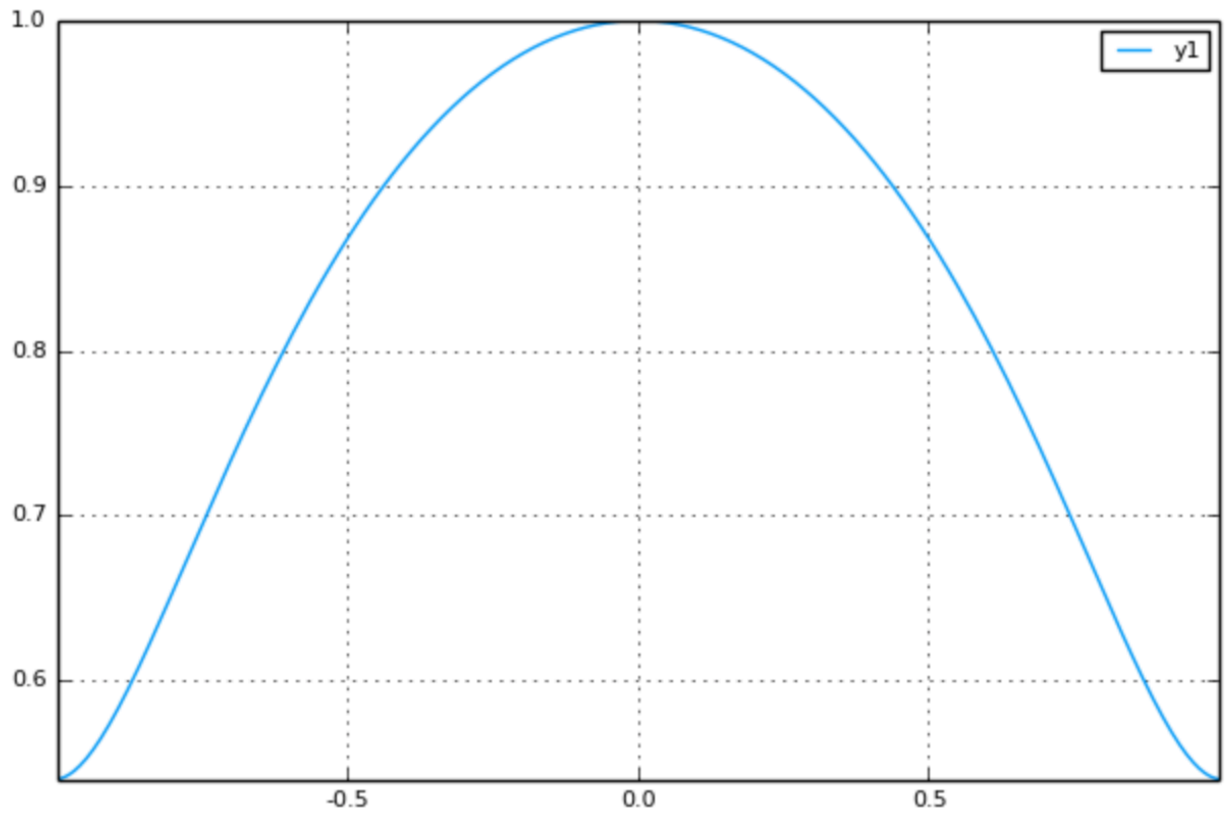
3.

```
using Plots
Pkg.add("Plots")
```

```
INFO: Nothing to be done
INFO: METADATA is out-of-date — you may not have the latest version of P
lots
INFO: Use `Pkg.update()` to get the latest versions of your packages
```

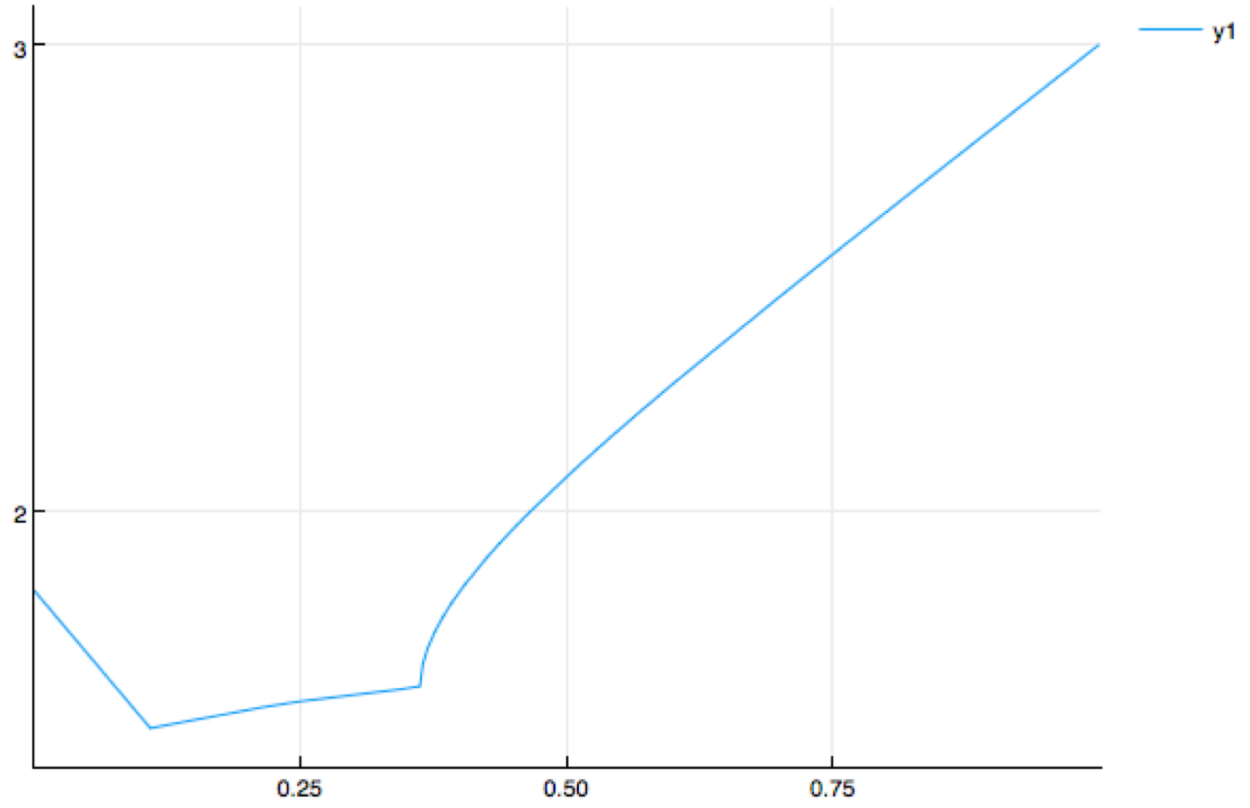
```
using ApproxFun
```

```
f = Fun(x->cos(sin(tan(x))))
plot(f)
```



4.

```
A = [1 2 0; 0 2 1; 1 0 2.0]
B = [1 1 0; 1 -1 1; -1 1 1.0]
f = Fun(t -> maximum(abs(eigvals(t*A + (1-t)*B))),[0,1])
plot(f)
```



5. Skip

Problem 5

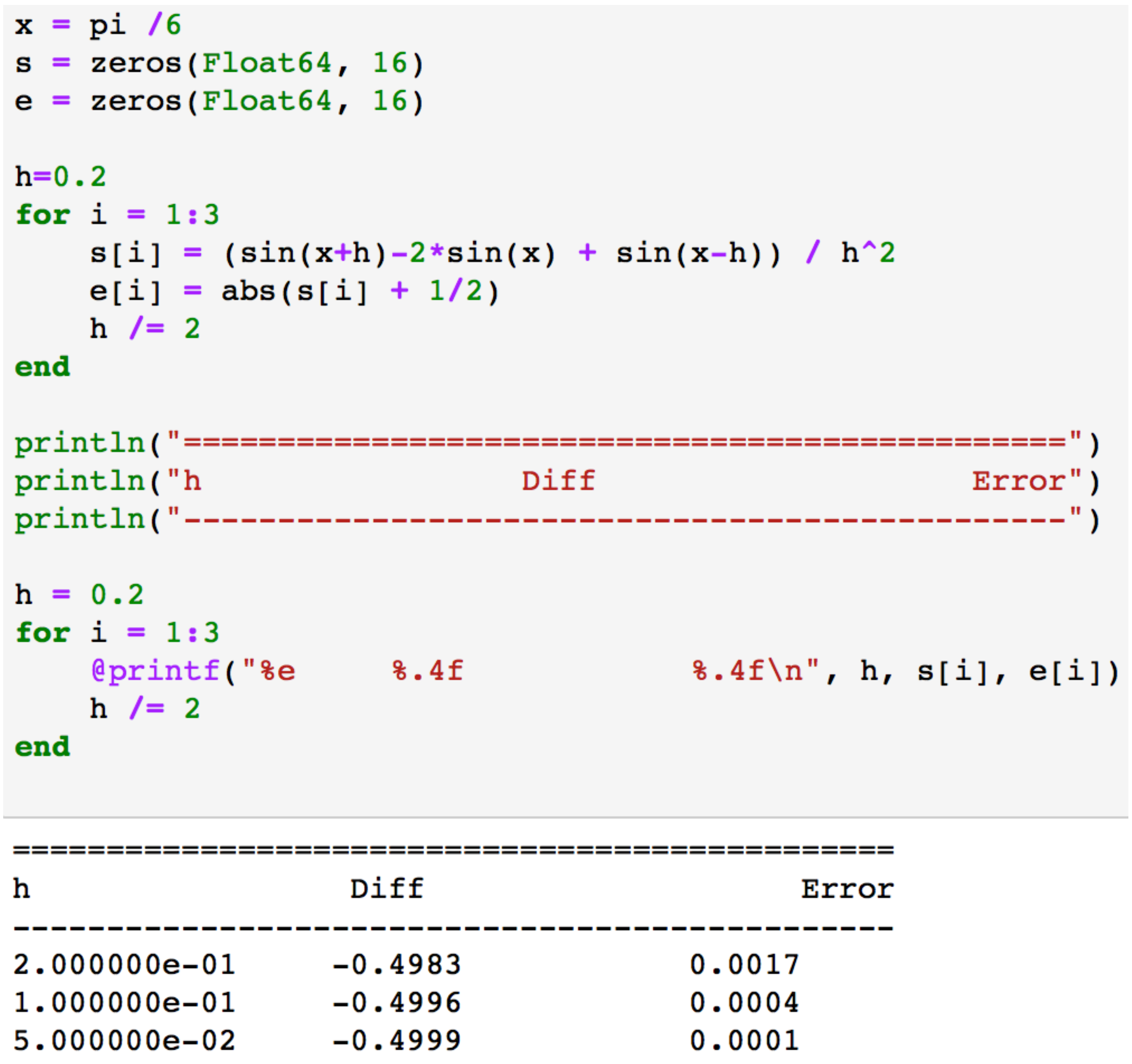
1.

```julia
x = pi /6
s = zeros(Float64, 16)
e = zeros(Float64, 16)

h=0.1
for i = 1:16
    s[i] = (sin(x+h)-2*sin(x) + sin(x-h)) / h^2
    e[i] = abs(s[i] + 1/2)
    h /= 10
end

println("===============================================")
println("h                 Diff                    Error")
println("-----------------------------------------------")

h = 0.1;
for i = 1:16
    @printf("%e     %.4f            %.4f\n", h, s[i], e[i])
    h /=10;
end
```

```
===============================================
h                 Diff                    Error
-----------------------------------------------
1.000000e-01     -0.4996            0.0004
1.000000e-02     -0.5000            0.0000
1.000000e-03     -0.5000            0.0000
1.000000e-04     -0.5000            0.0000
1.000000e-05     -0.5000            0.0000
1.000000e-06     -0.4999            0.0001
1.000000e-07     -0.4940            0.0060
1.000000e-08     -1.1102            0.6102
1.000000e-09     55.5112            56.0112
1.000000e-10     0.0000            0.5000
1.000000e-11     0.0000            0.5000
1.000000e-12     0.0000            0.5000
1.000000e-13     5551115123.1258            5551115123.6258
1.000000e-14     -555111512312.5781            555111512312.0781
1.000000e-15     0.0000            0.5000
1.000000e-16     -5551115123125782.0000            5551115123125782.0000
```

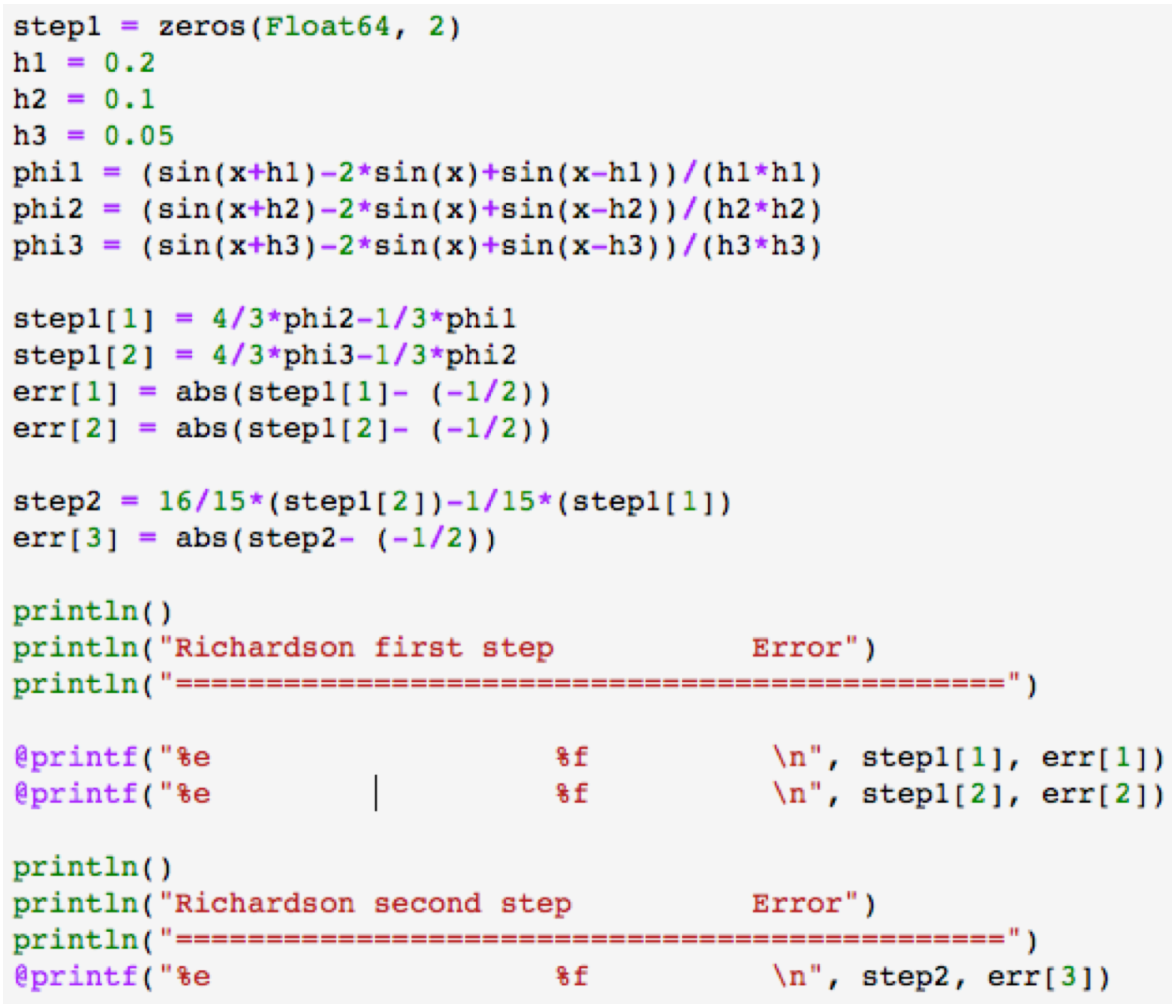
As we can see in the table, the best approximation is when h is around 10^{-2} *to* 10^{-5}. When h gets lesser, the error becomes larger and larger. Because for the equation s[i] = …. / h^2, when h gets lesser, h^2 will dominate the whole equation and make the error larger.

2.

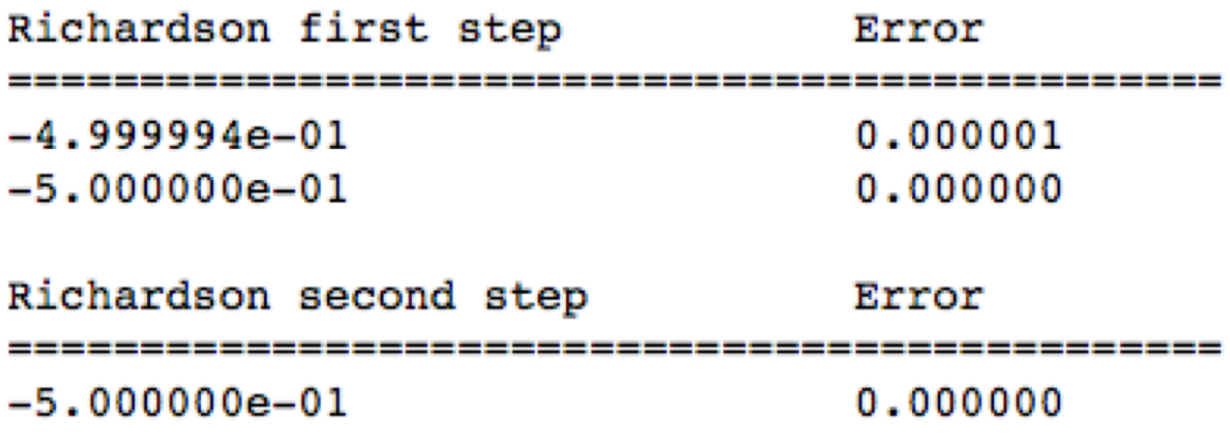


```julia
x = pi /6
s = zeros(Float64, 16)
e = zeros(Float64, 16)

h=0.2
for i = 1:3
    s[i] = (sin(x+h)-2*sin(x) + sin(x-h)) / h^2
    e[i] = abs(s[i] + 1/2)
    h /= 2
end

println("===============================================")
println("h                 Diff                    Error")
println("-----------------------------------------------")

h = 0.2
for i = 1:3
    @printf("%e     %.4f            %.4f\n", h, s[i], e[i])
    h /= 2
end
```

```
===============================================
h                 Diff                    Error
-----------------------------------------------
2.000000e-01     -0.4983            0.0017
1.000000e-01     -0.4996            0.0004
5.000000e-02     -0.4999            0.0001
```



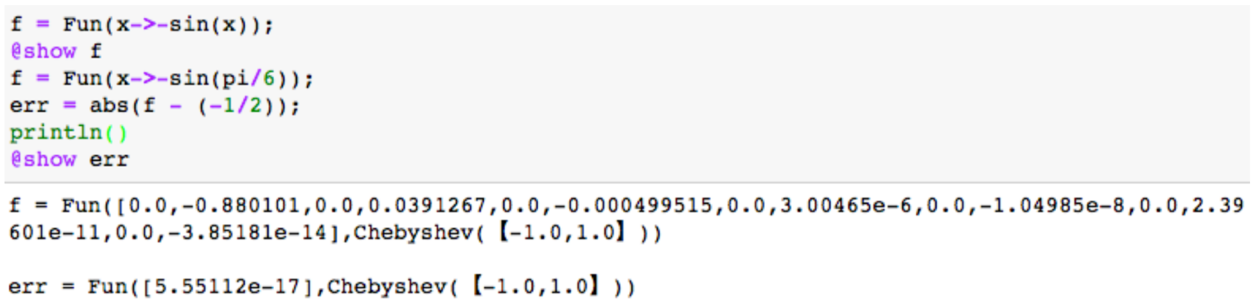
```
step1 = zeros(Float64, 2)
h1 = 0.2
h2 = 0.1
h3 = 0.05
phi1 = (sin(x+h1)-2*sin(x)+sin(x-h1))/(h1*h1)
phi2 = (sin(x+h2)-2*sin(x)+sin(x-h2))/(h2*h2)
phi3 = (sin(x+h3)-2*sin(x)+sin(x-h3))/(h3*h3)

step1[1] = 4/3*phi2-1/3*phi1
step1[2] = 4/3*phi3-1/3*phi2
err[1] = abs(step1[1]- (-1/2))
err[2] = abs(step1[2]- (-1/2))

step2 = 16/15*(step1[2])-1/15*(step1[1])
err[3] = abs(step2- (-1/2))

println()
println("Richardson first step           Error")
println("==============================================")

@printf("%e                   %f           \n", step1[1], err[1])
@printf("%e                   %f           \n", step1[2], err[2])

println()
println("Richardson second step          Error")
println("==============================================")
@printf("%e                   %f           \n", step2, err[3])
```



```
Richardson first step            Error
=============================================
-4.999994e-01                    0.000001
-5.000000e-01                    0.000000

Richardson second step           Error
=============================================
-5.000000e-01                    0.000000
```

The order of accuracy for Richardson extrapolation after one step is $O(h^4)$. After two steps is $O(h^6)$.

3.



```
f = Fun(x->-sin(x));
@show f
f = Fun(x->-sin(pi/6));
err = abs(f - (-1/2));
println()
@show err
```

```
f = Fun([0.0,-0.880101,0.0,0.0391267,0.0,-0.000499515,0.0,3.00465e-6,0.0,-1.04985e-8,0.0,2.39
601e-11,0.0,-3.85181e-14],Chebyshev( [-1.0,1.0] ))

err = Fun([5.55112e-17],Chebyshev( [-1.0,1.0] ))
```

The degree of the interpolation polynomial that it produces for f is 13.
The error in its approximation is 5.55e-17.

4.

$$f'(x) \simeq \frac{1}{2h}[-3f(x) + 4f(x+h) - f(x+2h)]$$

$$\simeq \frac{1}{2h}\left[-3f(x) + 4\left[f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\epsilon 1) + O(h^4)\right] - \left[f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) + \frac{8h^3}{6}f'''(\epsilon 2) + O(h^4)\right]\right]$$

$$(because\ the\ complexicty\ O(h^4) term\ got\ deduced.)$$

$$\simeq \frac{1}{2h}\left[2hf'(x) + \frac{4h^3}{6}f'''(\epsilon 1) + \frac{8h^3}{6}f'''(\epsilon 2)\right]$$

$$\simeq f'(x) + \frac{h^2}{3}f'''(\epsilon 1) + \frac{2h^2}{3}f'''(\epsilon 2)$$

Error term: $\frac{h^2}{3}f'''(\epsilon 1) + \frac{2h^2}{3}f'''(\epsilon 2)$.