multiply

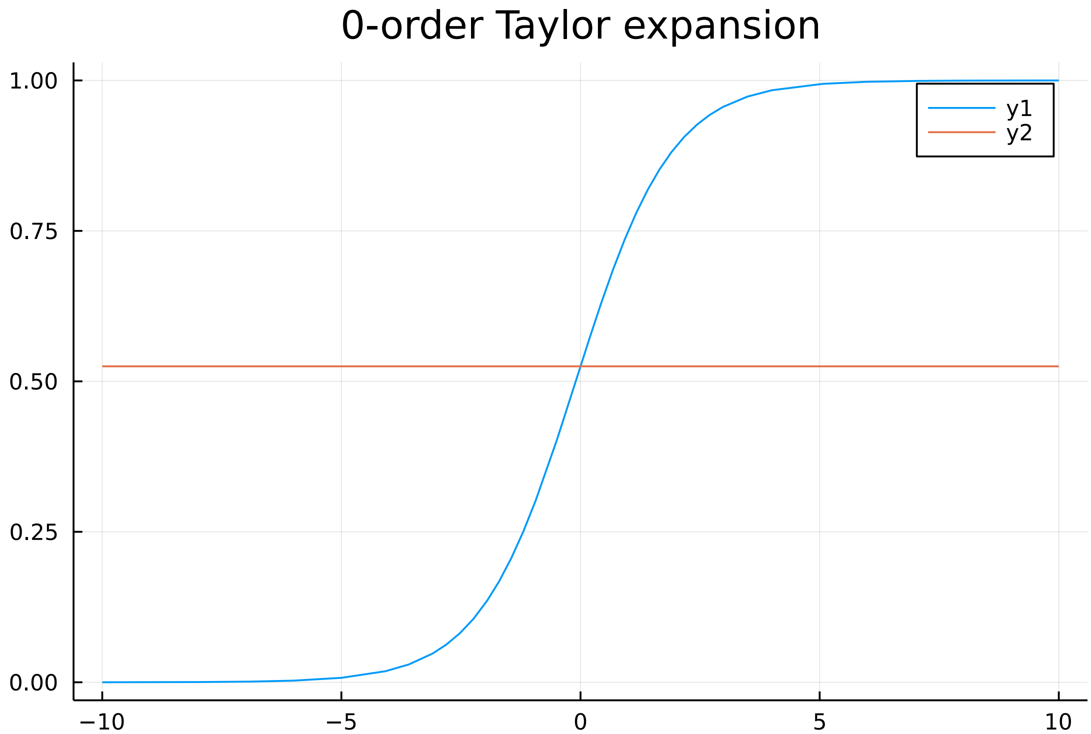
March 31, 2022

[1]:
```
# using Pkg; Pkg.add("GLM");
```

[2]:
```
# TODO: remove any mention of sigmoids

using DataFrames
using GLM
using LaTeXStrings
using Plots

function regress_convergence(h, Z)
    df = DataFrame(h=h, Z=Z) # Z = Ch^p
    fm = @formula(log(Z) ~ 1 + log(h)) # log(Z) = log(C)*1 + p*log(h)
    lr = lm(fm, df) # fit a straight line to the data
    lC, p = GLM.coef(lr) # retreive constant coefficients log(C) and p
    C = exp(lC)
    return p, C
end

rd(x) = round(x, digits=3) # formatting

function sigmoid(delta)
    s(x) = 1.0/(1.0+exp(-(x+delta)))
    return s
end

function d2sigmoid(delta)
    d2s(x) = 2*exp(-2*(x+delta))/(exp(-(x+delta))+1)^3 - exp(-(x+delta))/
 ↪(exp(-(x+delta))+1)^2
    return d2s
end

delta = 0.1 # translate so sigma^(k)(0) != 0

sigma = sigmoid(delta)
poly0(x) = sigma(0.0) # 0-order taylor expansion

plot(sigma, -10.0, 10.0)
```

```
plot!(poly0, -10.0, 10.0, title="0-order Taylor expansion")
```

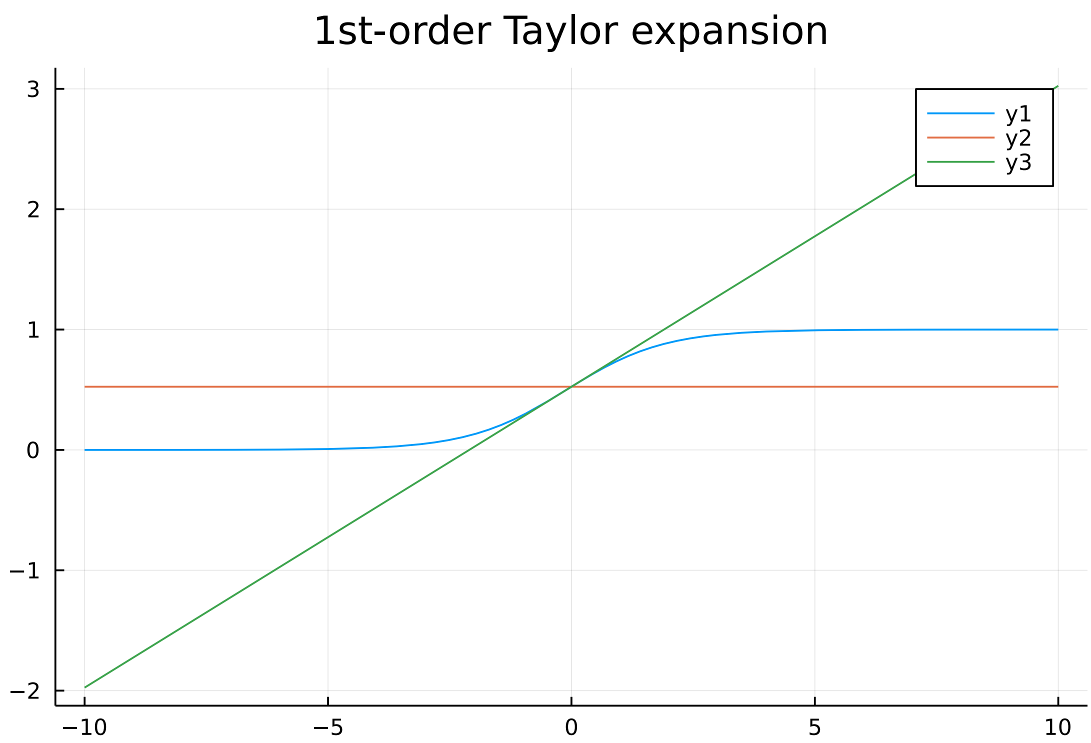
[2]:



[3]:

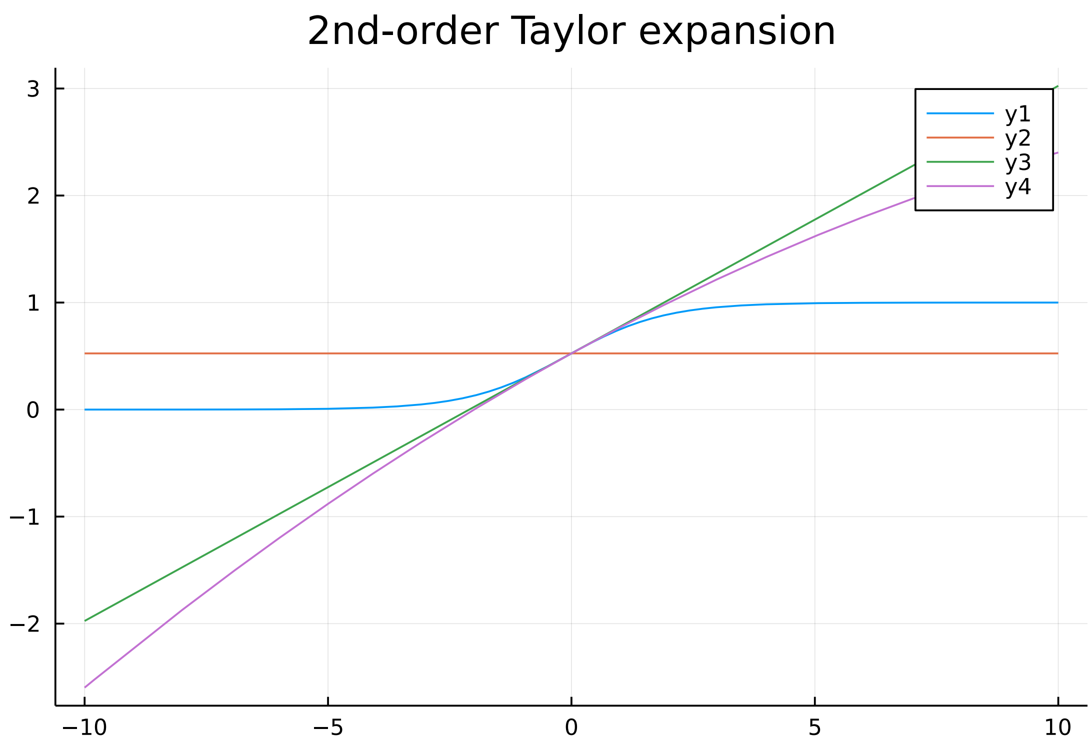
```
function ds(x)
    return exp(-x)/(exp(-x)+1)^2
end

poly1(x) = poly0(x) + x*ds(0)

plot!(poly1, -10.0, 10.0, title="1st-order Taylor expansion")
```

[3]:



[4]:
```
d2sigma = d2sigmoid(0.1)

poly2(x) = poly1(x) + (x^2/2)*d2sigma(0)

plot!(poly2, -10.0, 10.0, title="2nd-order Taylor expansion")
```

[4]:



[5]:
```
delta = 1.0 # 100

#s = sigmoid(delta)
#d2s = d2sigmoid(delta)
s(x) = sqrt(x+1.0)
d2s(x) = -(x+1.0)^(-3.0/2.0)/4.0

function m(input)
    u = input[1]
    v = input[2]
    return (s(+u+v)+s(-u-v)-s(+u-v)-s(-u+v))/(4*d2s(0))
end

d2s(0)
```

[5]: -0.25

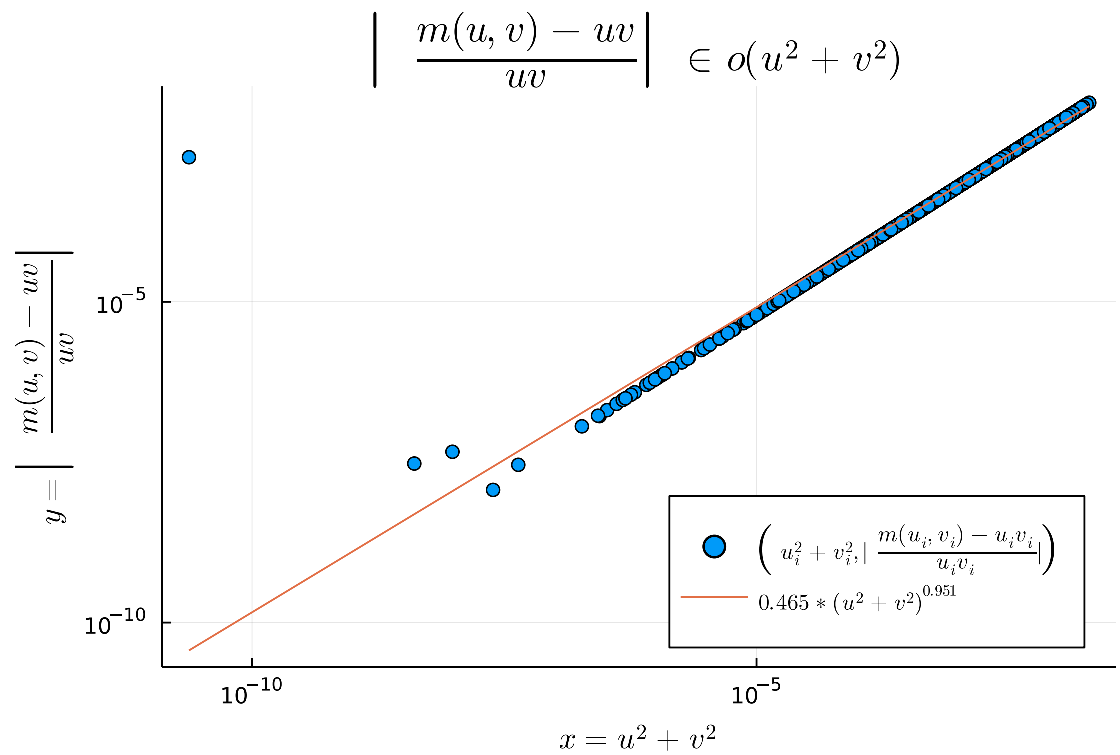
[6]:
```
# test
n = 2000
data = rand(n,5);
for i in 1:n
    u = rand()*10^(-1.0);
    v = u*rand() # rand()*10^(-1.0*rand(1:20));
```

```julia
    input = [u; v]
    data[i,1] = max(abs(u), abs(v)) #abs(u*v*(u^2+v^2))
    data[i,2] = abs((m(input)-u*v)/(u*v))
    data[i,3] = u
    data[i,4] = v
    data[i,5] = abs(1.0*(u^2+v^2))
end

p, C = regress_convergence(data[:,5], data[:,2])
```

[6]: (0.9507850080143745, 0.4654573876294935)

[7]:
```julia
O(h) = C*h^p
scatter(data[:,5], data[:,2], yaxis=:log, xaxis=:log, label=L"\left(u_i^2+v_i^2,|\frac{m(u_i,v_i)-u_iv_i}{u_iv_i}|\right)", legend=:bottomright, size=(600,400))
plot!(O, minimum(data[:,5]), maximum(data[:,5]), label=L"%$(rd(C))*(u^2+v^2)^{%$(rd(p))}", xlabel=L"x=u^2+v^2", ylabel=L"y = \left|\frac{m(u,v)-uv}{uv}\right|", title=L"\left|\frac{m(u,v)-uv}{uv}\right|\in \mathcal{o}(u^2+v^2)")
```

[7]:



[8]:
```julia
savefig("convergence")
```

```
b = 0.0

lambda = 1.0

function A1(x)
    W1 = [[+1.0, -1.0, +1.0, -1.0] [+1.0, -1.0, -1.0, +1.0]]
    b1 = [b, b, b, b]
    return lambda*(W1*x+b1)
end

function A2(x)
    W2 = [[1.0] [1.0] [-1.0] [-1.0]]
    b2 = [-b]
    return (W2*x+b2)/(4*d2s(0)*lambda^2)
end

function f(input)
    l1 = s.(A1(input))
    return A2(l1)
end

for i in 1:n
    u = data[i,3]
    v = data[i,4]
    input = [u; v]
    data[i,2] = abs((sum(f(input))-u*v)/(u*v))
#     data[i,5] = abs(sum(f(input))-u*v)
end

p2, C2 = regress_convergence(data[:,5], data[:,2])
```

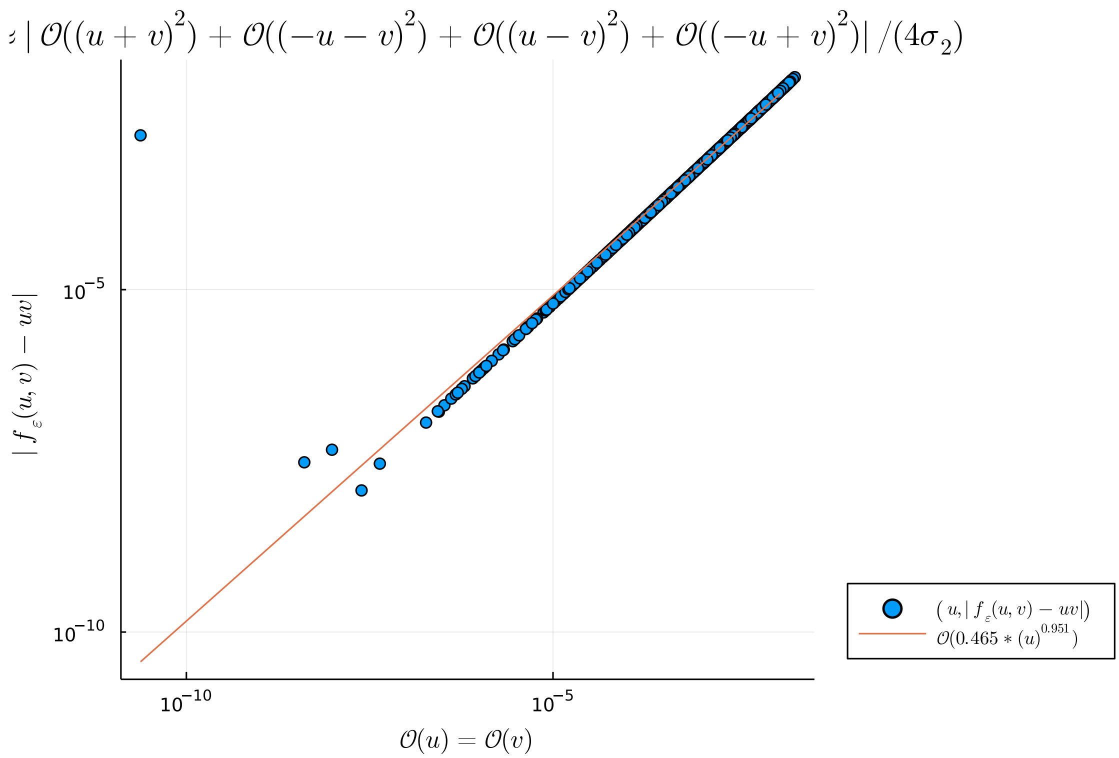
[8]:

```
(0.9507850080143745, 0.4654573876294935)
```

[9]:

```
O2(h) = C2*h^p2
scatter(data[:,5], data[:,2], yaxis=:log, xaxis=:log,␣
 ↪label=L"\left(u,|f_\varepsilon(u,v)-uv|\right)", legend=:outerbottomright,␣
 ↪size=(720, 480))
plot!(O, minimum(data[:,5]), maximum(data[:,2]),␣
 ↪label=L"\mathcal{O}(%$(rd(C2))*(u)^{%$(rd(p2))})",␣
 ↪xlabel=L"\mathcal{O}(u)=\mathcal{O}(v)", ylabel=L"|f_\varepsilon(u,v)-uv|",␣
 ↪title=L"y␣
 ↪\approx|\mathcal{O}((u+v)^2)+\mathcal{O}((-u-v)^2)+\mathcal{O}((u-v)^2)+\mathcal{O}((-u+v)^
 ↪(4\sigma_2)")
```

[9]:



$| \mathcal{O}((u+v)^2) + \mathcal{O}((-u-v)^2) + \mathcal{O}((u-v)^2) + \mathcal{O}((-u+v)^2)| /(4\sigma_2)$
$| f_\varepsilon(u,v) - uv|$
10^{-5}
10^{-10}
10^{-10}
10^{-5}
$\mathcal{O}(u) = \mathcal{O}(v)$
$(u, | f_\varepsilon(u,v) - uv|)$
$\mathcal{O}(0.465 * (u)^{0.951})$

[ ]: