实验题目1 拉格朗日(Lagrange)插值

代码实现

In [113…
```
using Printf
using LinearAlgebra
using Plots
```

In [114…
```
# Lagrange Interpolation method
function lagrange(xs, fxs, x::Number)
    # plot(xs,ys)
    num = size(xs, 1)
    y, i = 0.0, 1
    while i <= num
        li = 1.0
        # for j in filter(x -> x != i, 1:num)  # much slower than continue
        for j in 1:num
            if j == i
                continue
            end
            li *= (x - xs[j]) / (xs[i] - xs[j])
        end
        y += li * fxs[i]
        i += 1
    end
    x, y
end
function lagrange(xs, fxs, x::Vector)
    # plot(xs,ys)
    num = size(xs, 1)
    y, i = zeros(size(x, 1)), 1
    li = zeros(size(x, 1))  # 两种写法孰优孰劣还不好说，得写代码来验证
    while i <= num
        li .= 1.0
        # li = repeat([1.0], size(x, 1))
        # for j in filter(x -> x != i, 1:num)  # much slower than continue
        for j in 1:num
            if j == i
                continue
            end
            li = li .* (x .- xs[j]) / (xs[i] - xs[j])
        end
        y = y + li .* fxs[i]
        i += 1
    end
    x, y
end
```

```
lagrange (generic function with 2 methods)
```

In [115…
```
function show_error(f::Function, title::String, series_x, series_y)
    errors = abs.(f.(series_x) - series_y) ./ f.(series_x)
    plot(series_x, errors, label="relative error", title=title, legend=:outertoprigh
    # plot!(ylim=(0, 1))
end
```

```
show_error (generic function with 2 methods)
```

测试代码

Test 1 - Simple

In [116…
```
xs = [0, 2, 3, 5, 6]
ys = [1, 3, 2, 5, 6]
# xs = [x for x in 1:1000]
# ys = [y^2 for y in 1:1000]
# 多项式插值法的弊端，当n过大会导致失真
xs = [0.4, 0.55, 0.65, 0.80]
ys = [0.41075, 0.57815, 0.69675, 0.88811]
@time lagrange(xs, ys, 0.55)
```

```
  0.014786 seconds (19.91 k allocations: 1.103 MiB, 99.81% compilation time)
(0.55, 0.57815)
```

实验题目

问题 1

拉格朗日插值多项式的次数n越大越好吗?

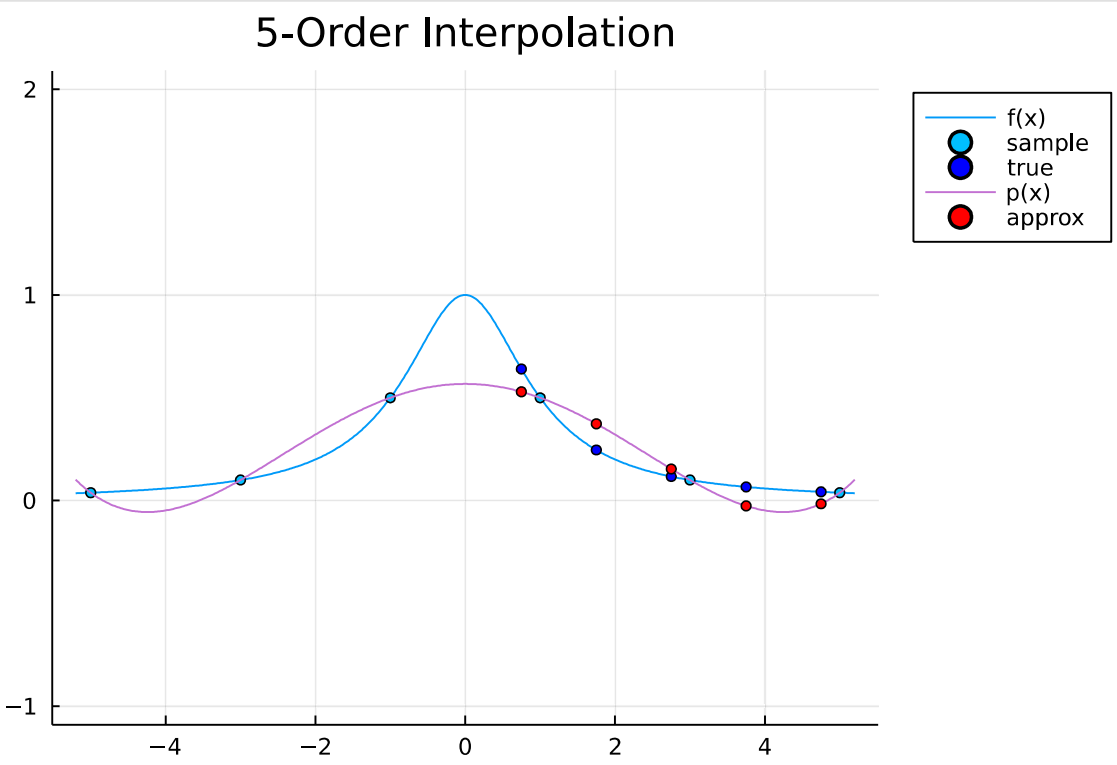
> 不是，若是次数过高，会出现Runge现象，插值多项式在距离已知点位置较远处会剧烈震荡，直观呈现如下列问题所作示意图

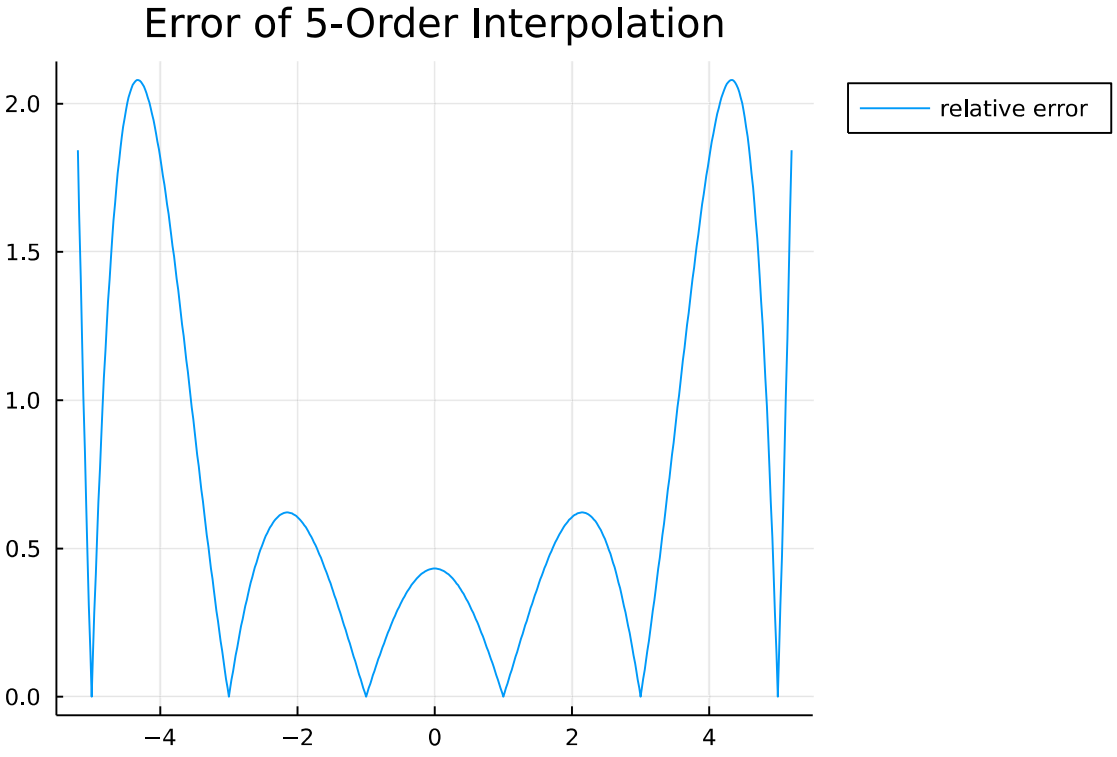
In [117…
```
function show_result(f::Function, split_nums::Vector, test_x::Vector, xlim::Vec
    for n in split_nums
        x_min, x_max = xlim
        x_range = x_min-0.2:0.02:x_max+0.2
        xs = x_min:(x_max-x_min)/n:x_max
        ys = f.(xs)

        # (xlim=(xMin,xMax), ylim=(yMin, yMax), yflip = false)
        # https://stackoverflow.com/questions/53230969/how-to-scale-a-plot-in-julia-
        plot(x_range, f.(x_range), label="f(x)")  # plot f(x)
        plot!(legend=:outertopright, title="$n-Order Interpolation")
        plot!(ylim=ylim, yflip=false)  # add ylim
        plot!(xs, ys, seriestype=:scatter, markersize=3, msw=1, color=:deepskyblue,

        test_y = f.(test_x)
        plot!(test_x, test_y, seriestype=:scatter, markersize=3, msw=1, color=:blue,
        _, pred_y = lagrange(xs, ys, test_x)
        # @time _, pred_y = lagrange(xs, ys, test_x)
        println("$n-Order Interpolation:")
        println("test_x: $test_x")
        println("test_y: $test_y")
        println("pred_y: $pred_y")
        series_x = Vector(x_range)
        _, series_y = lagrange(xs, ys, series_x)  # compute the interporlation funct
        plot!(series_x, series_y, label="p(x)")  # add p(x) function curve
        display(plot!(test_x, pred_y, seriestype=:scatter, markersize=3, msw=1, colo
        display(show_error(f, "Error of $n-Order Interpolation", series_x, series_y)
    end
end
```

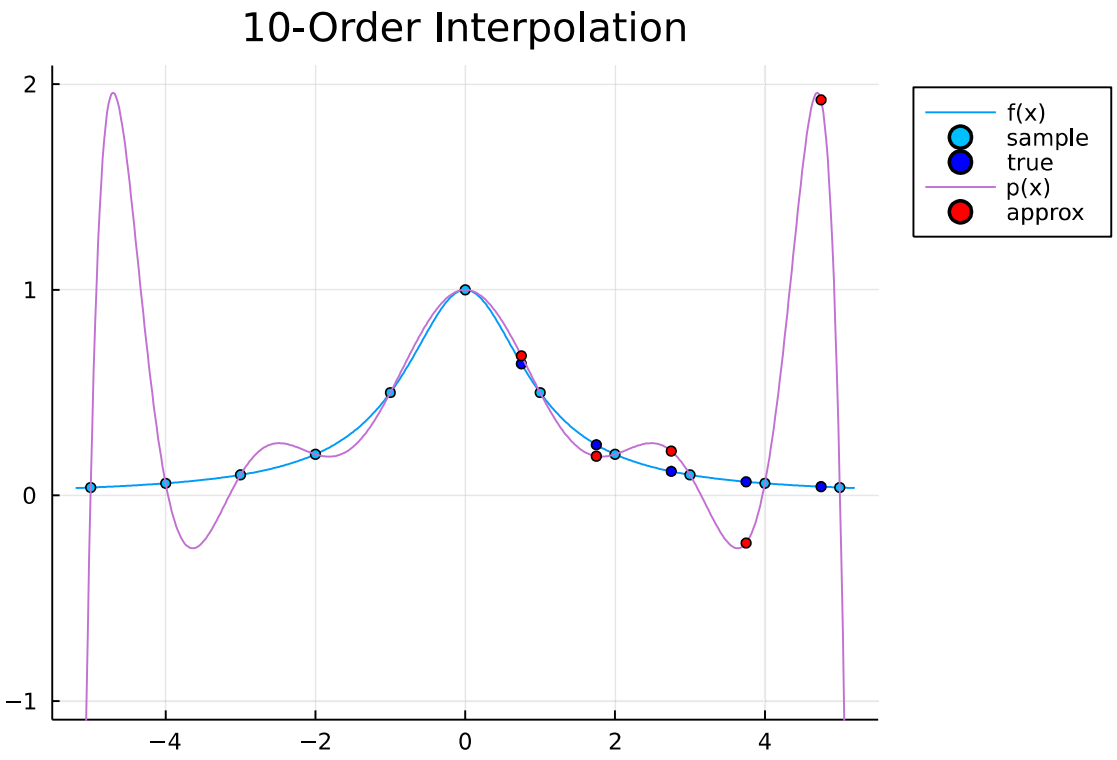
```
show_result (generic function with 2 methods)
```

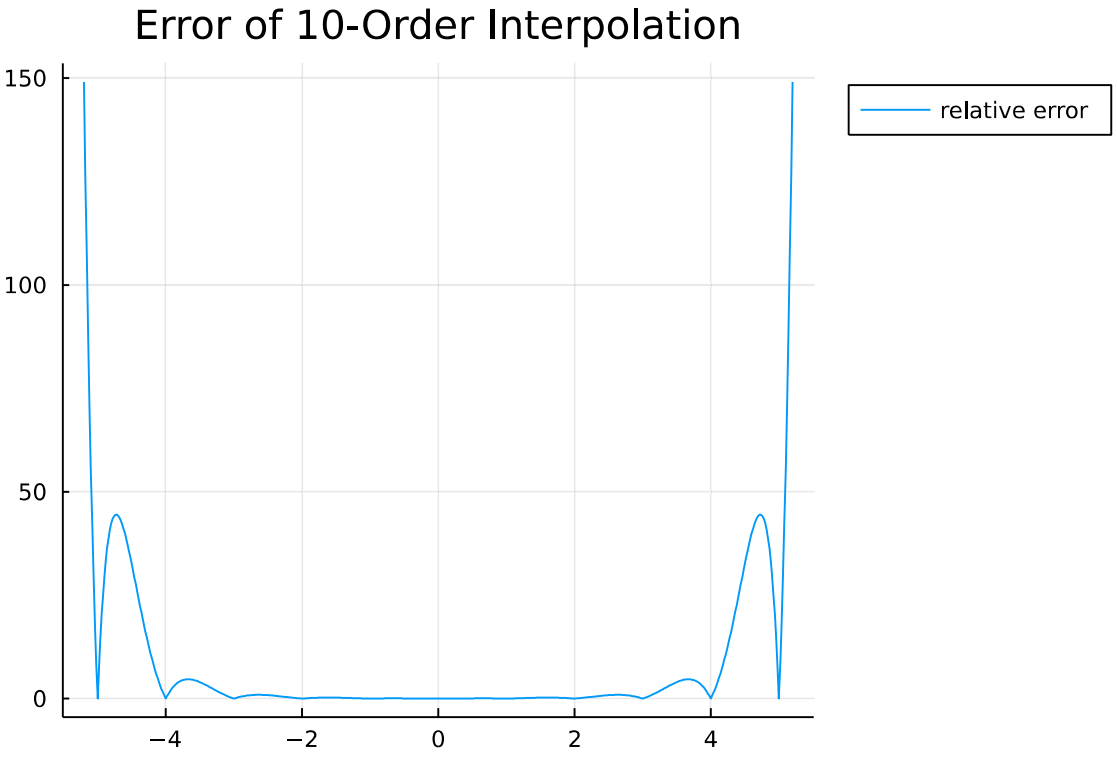
In [118…
```
f(x) = 1 / (1 + x^2)
split_nums = [5, 10, 20]
test_x = [0.75, 1.75, 2.75, 3.75, 4.75]
xlim = [-5, 5]
```

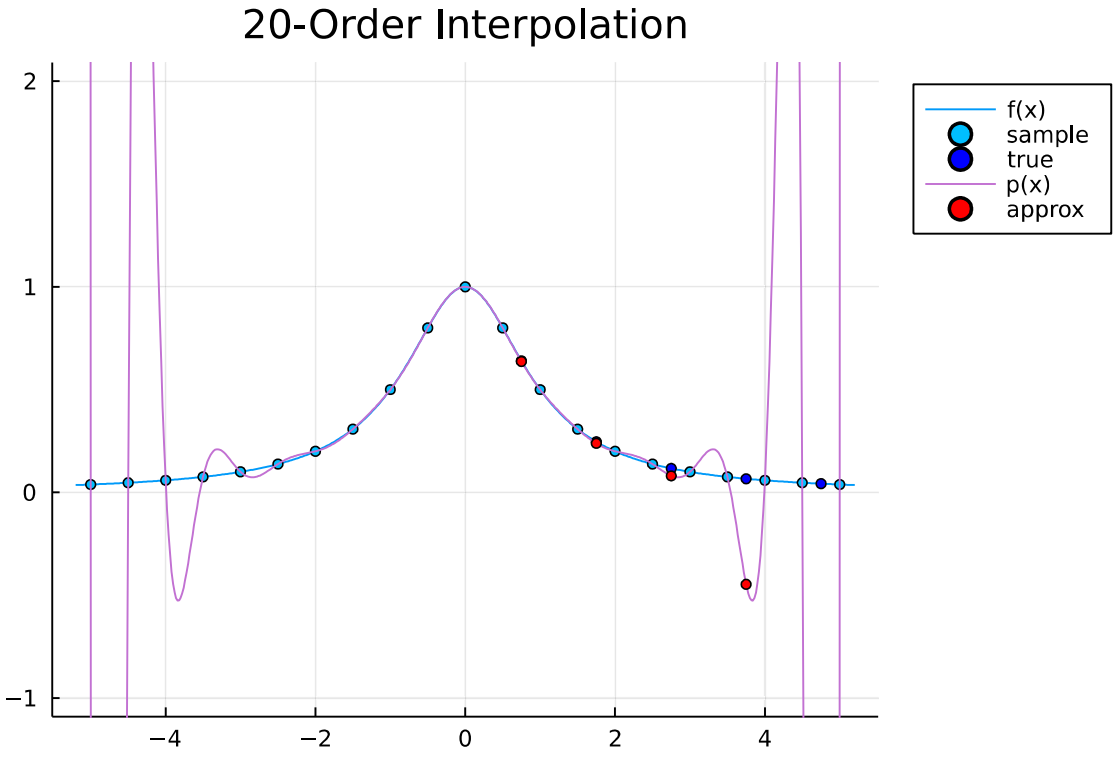
```
ylim = [-1, 2]
show_result(f, split_nums, test_x, xlim, ylim)
```

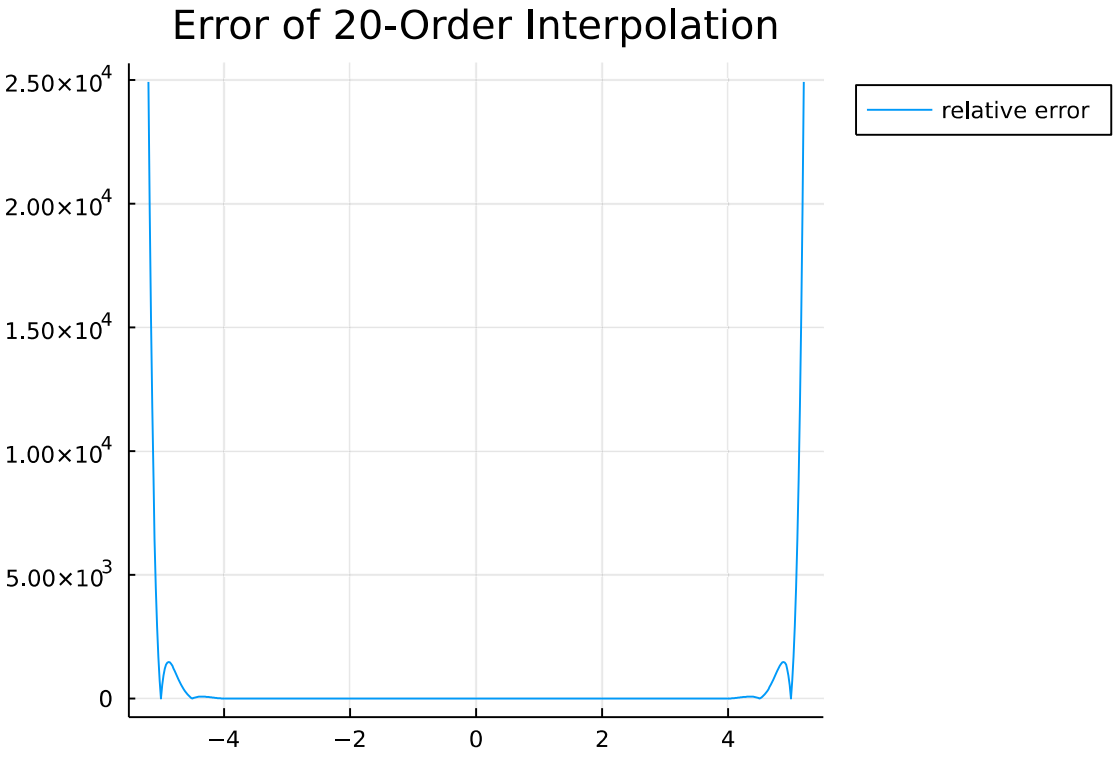





10-Order Interpolation
f(x)
sample
true
p(x)
approx
2
1
0
−1
−4
−2
0
2
4


Error of 10-Order Interpolation
relative error
150
100
50
0
−4
−2
0
2
4

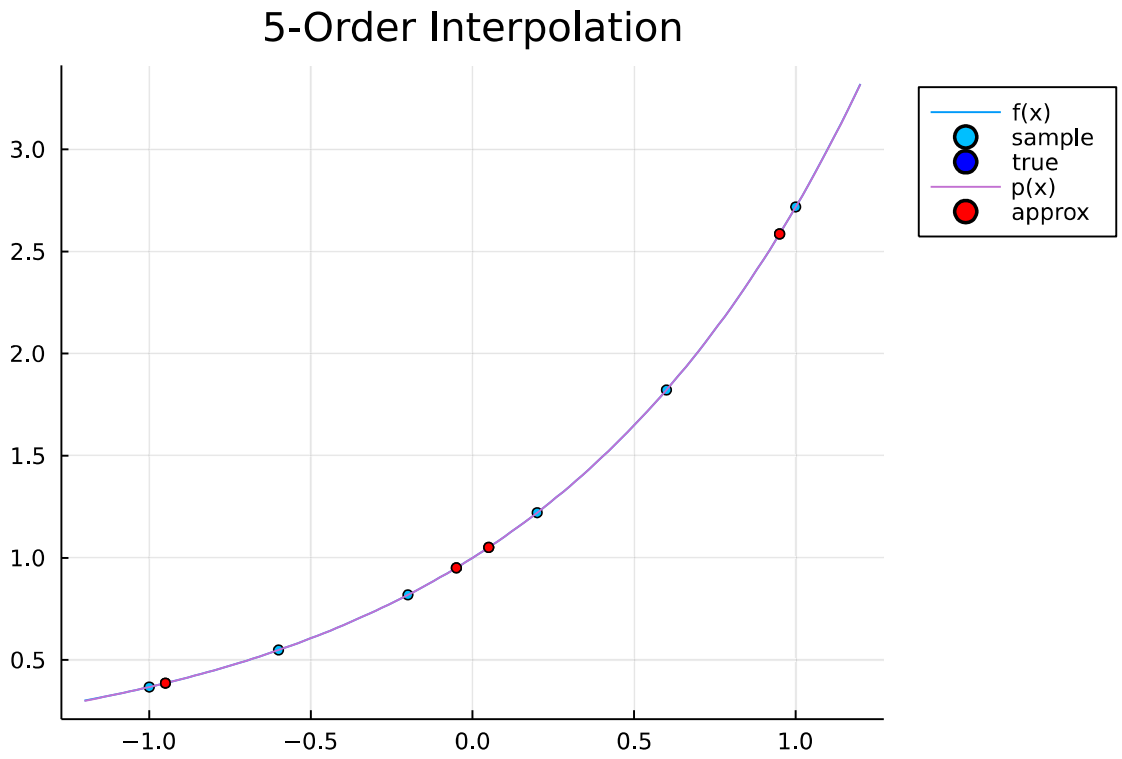

20-Order Interpolation
f(x)
sample
true
p(x)
approx
2
1
0
−1
−4
−2
0
2
4

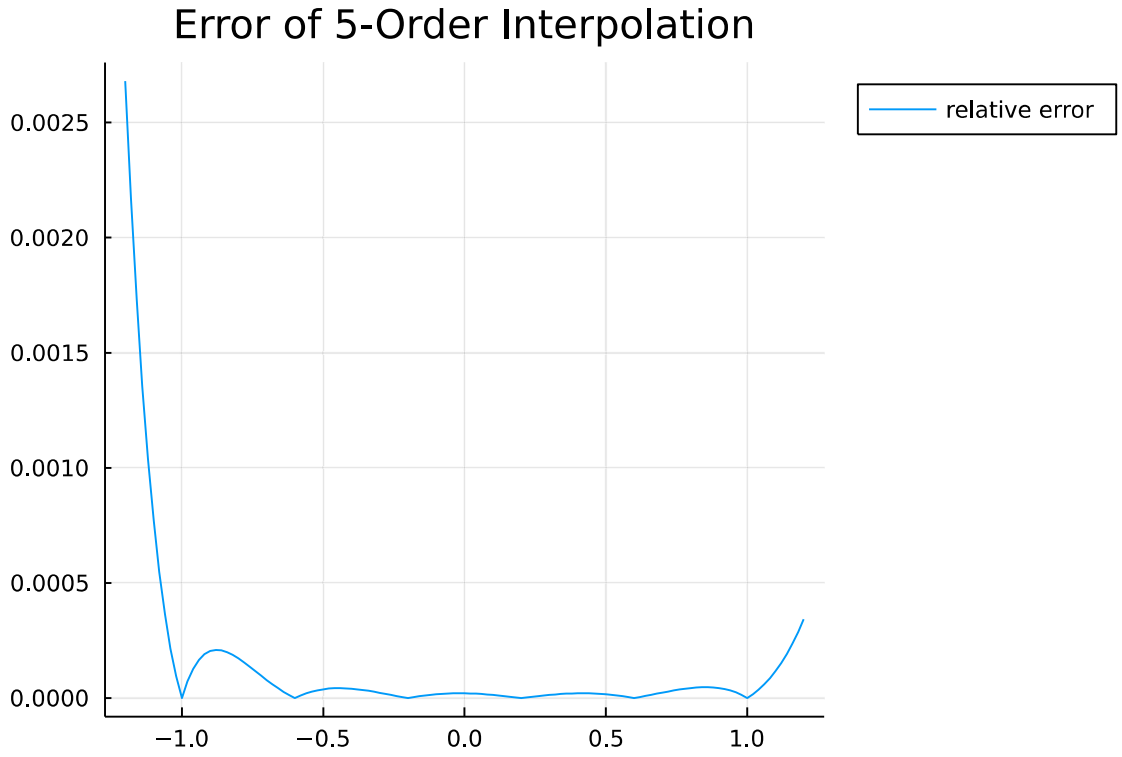

Error of 20-Order Interpolation
relative error
2.50×10^4
2.00×10^4
1.50×10^4
1.00×10^4
5.00×10^3
0
−4
−2
0
2
4

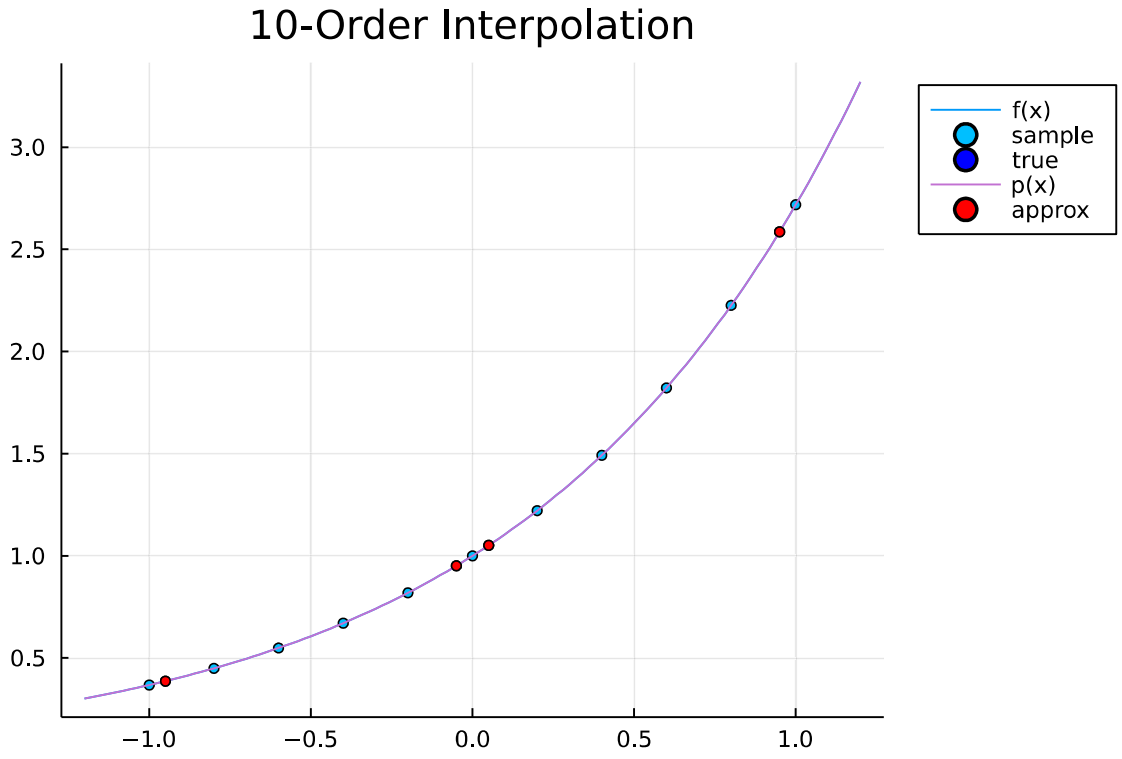
```
5-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [0.64, 0.24615384615384617, 0.11678832116788321, 0.06639004149377593, 0.0424
40318302387266]
pred_y: [0.528973858173077, 0.3733248197115384, 0.15373347355769232, -0.025954026442
307696, -0.015737680288461536]
10-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [0.64, 0.24615384615384617, 0.11678832116788321, 0.06639004149377593, 0.0424
40318302387266]
pred_y: [0.678989577293396, 0.19058046675375687, 0.21559187891256765, -0.23146174989
674442, 1.9236311497192042]
20-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [0.64, 0.24615384615384617, 0.11678832116788321, 0.06639004149377593, 0.0424
40318302387266]
pred_y: [0.6367553359164332, 0.23844593373813264, 0.08065999342165572, -0.4470519607
0883363, -39.95244903304101]
```

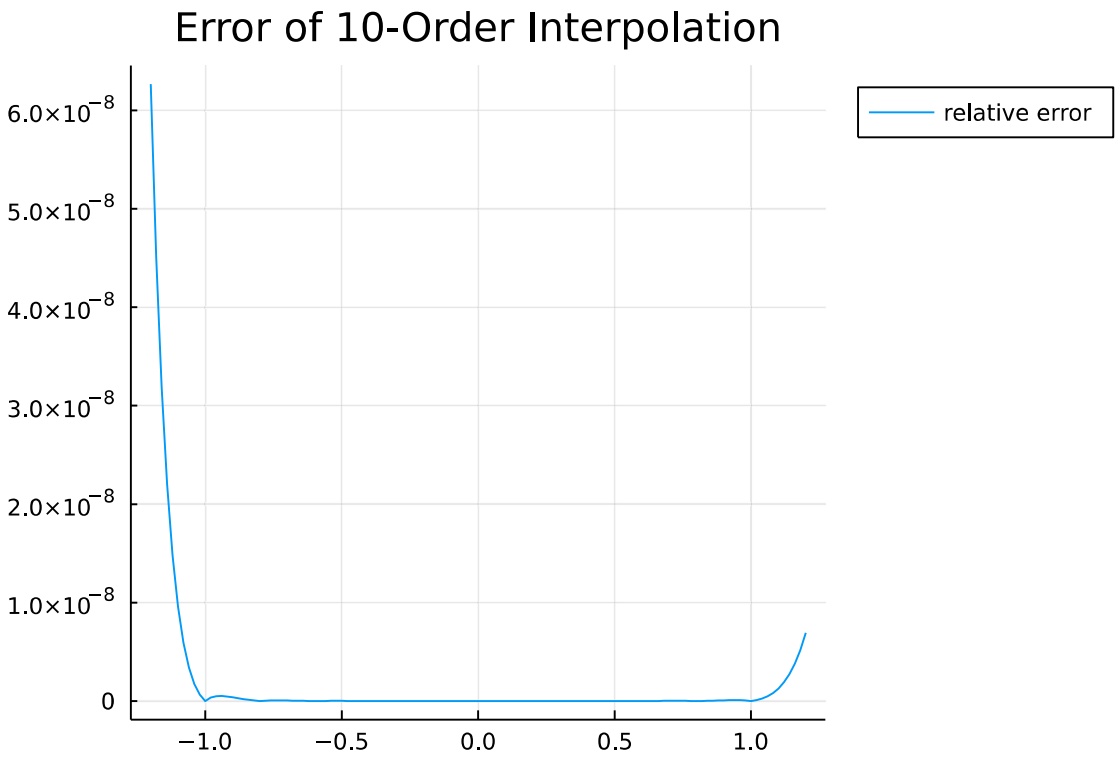
In [119…

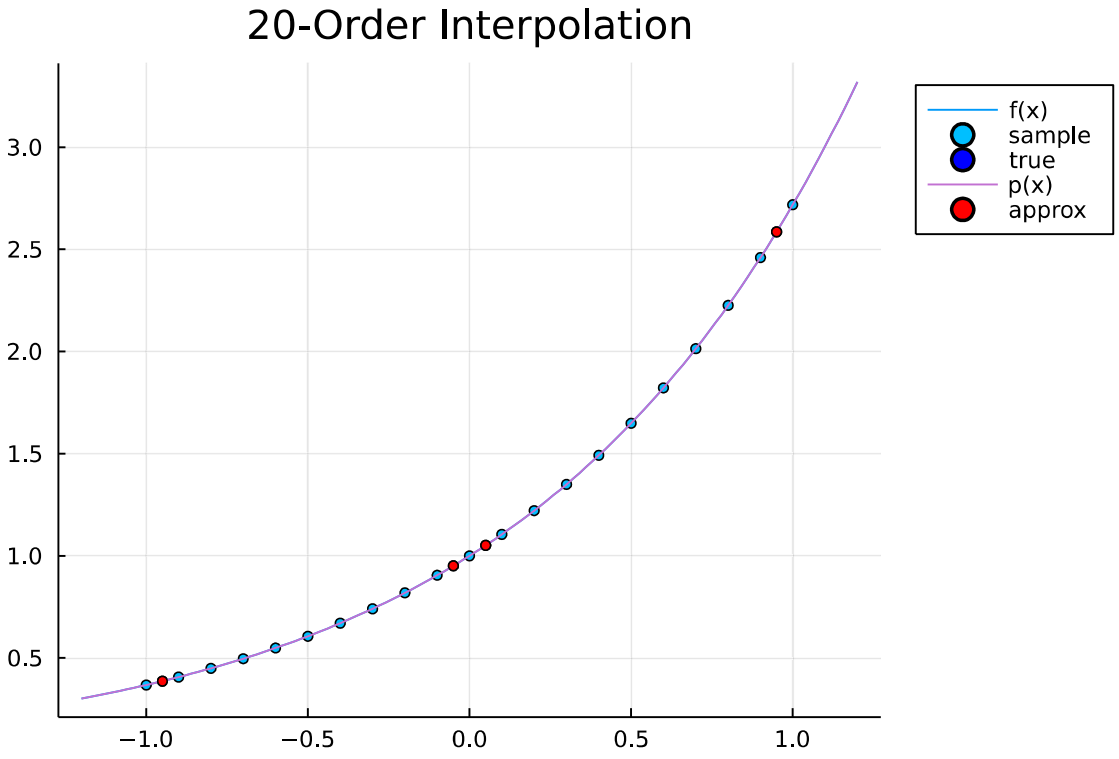
```
f(x) = exp(x)
split_nums = [5, 10, 20]
test_x = [-0.95, -0.05, 0.05, 0.95]
xlim = [-1, 1]
# ylim = [-1, 10]  # the good-looking ylim is defined manually
ylim = []
show_result(f, split_nums, test_x, xlim, ylim)
```

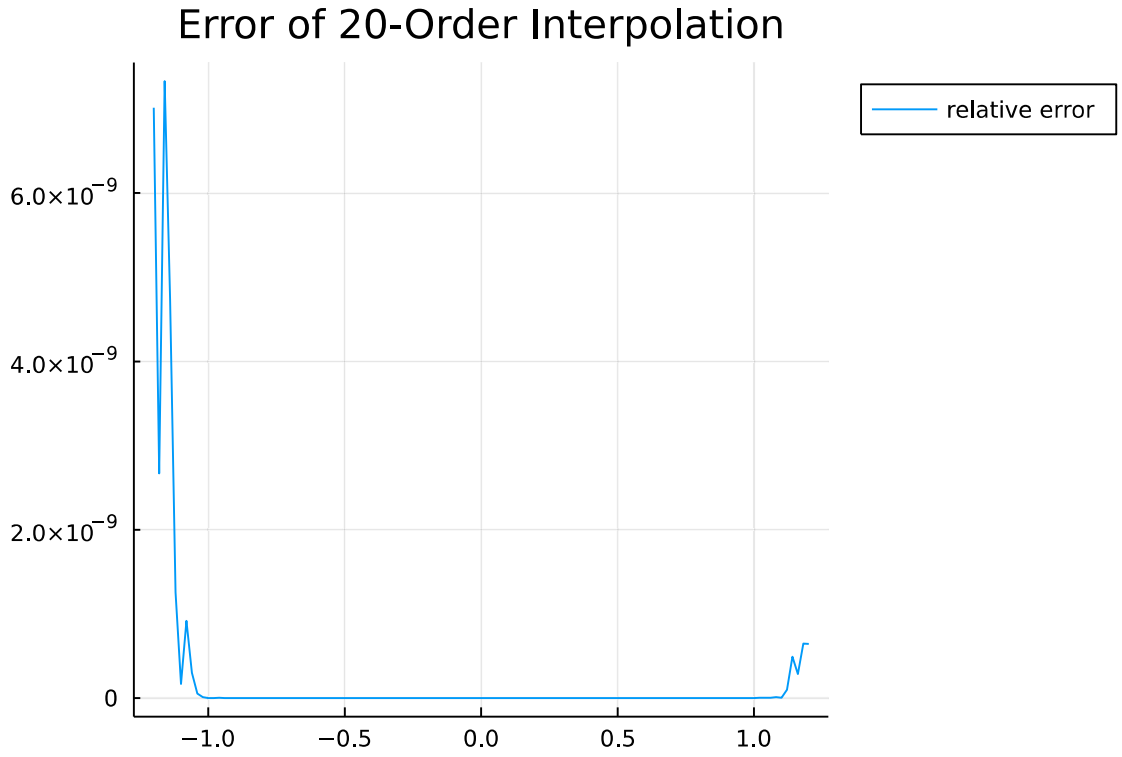



Error of 5-Order Interpolation
relative error
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−1.0
−0.5
0.0
0.5
1.0


10-Order Interpolation
f(x)
sample
true
p(x)
approx
3.0
2.5
2.0
1.5
1.0
0.5
−1.0
−0.5
0.0
0.5
1.0


Error of 10-Order Interpolation
relative error
6.0×10^−8
5.0×10^−8
4.0×10^−8
3.0×10^−8
2.0×10^−8
1.0×10^−8
0
−1.0
−0.5
0.0
0.5
1.0


20-Order Interpolation
f(x)
sample
true
p(x)
approx
3.0
2.5
2.0
1.5
1.0
0.5
−1.0
−0.5
0.0
0.5
1.0




```
5-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.38674102345450123, 0.951229424500714, 1.0512710963760241, 2.5857096593158
46]
pred_y: [0.38679815885799357, 0.9512483333804465, 1.051290275808478, 2.5857845509846
147]
10-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.38674102345450123, 0.951229424500714, 1.0512710963760241, 2.5857096593158
46]
pred_y: [0.3867410232556747, 0.9512294244990149, 1.0512710963777372, 2.5857096595487
61]
20-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.38674102345450123, 0.951229424500714, 1.0512710963760241, 2.5857096593158
46]
pred_y: [0.3867410234532931, 0.9512294245007142, 1.0512710963760243, 2.5857096593151
82]
```

问题 2

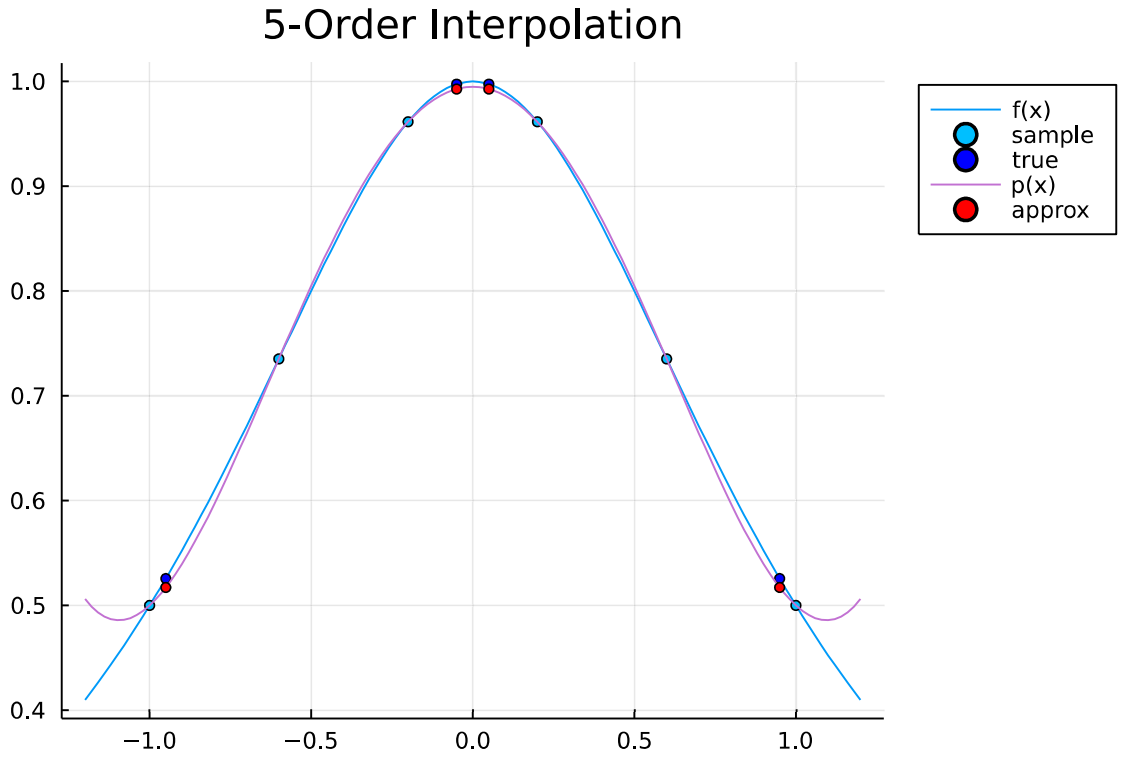
插值区间越小越好吗?

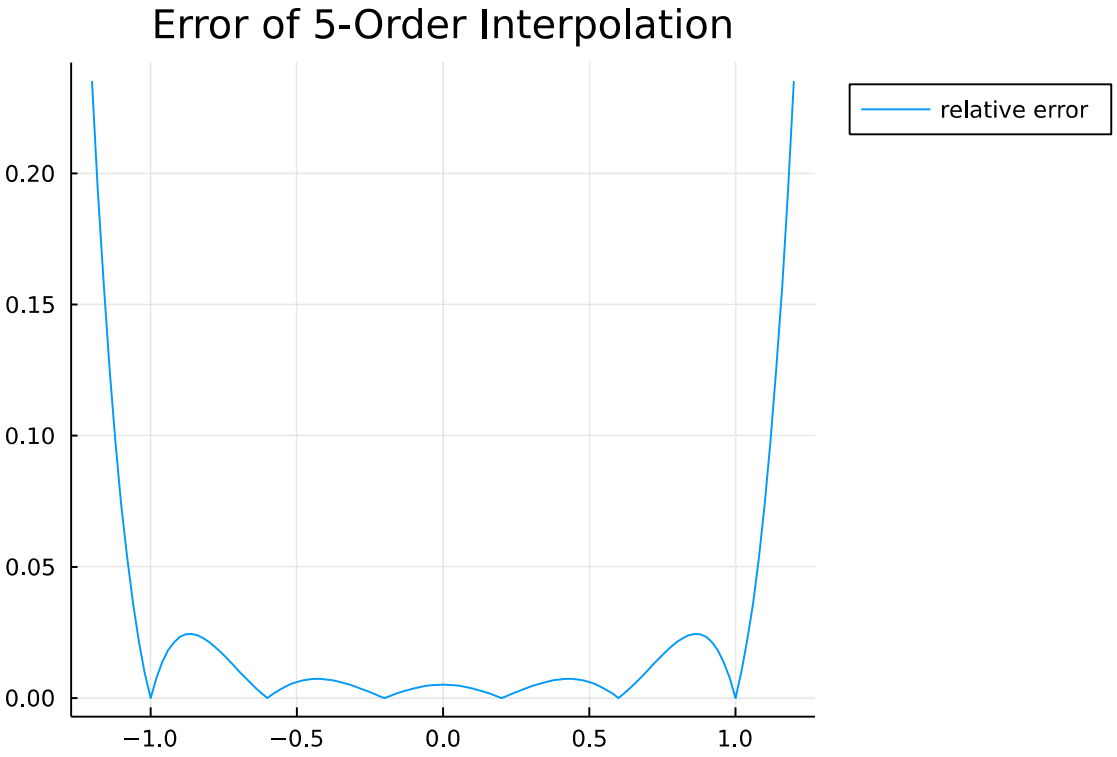
> 不一定，从精度上考虑虽然有一定的合理性，但插值节点过于密集时，一方面计算量增大却没提高对于精度计算的收益，另一方面区间缩短、节点增加并不能保证两节点间能很好的逼近函数，反而有可能出现Runge现象。但合理的对区间长度进行选择，同时采用低次插值来避免Runge现象，能够得到较好的拟合效果。
>
> 不过，实例中对于函数$f(x)=\frac{1}{1+x^2}$，较短区间的插值效果比长区间插值更好
>
> 而函数$f(x)=e^x$无论是长区间还是短区间插值，都能得到相对较好的拟合效果，但短区间插值相对误差更低

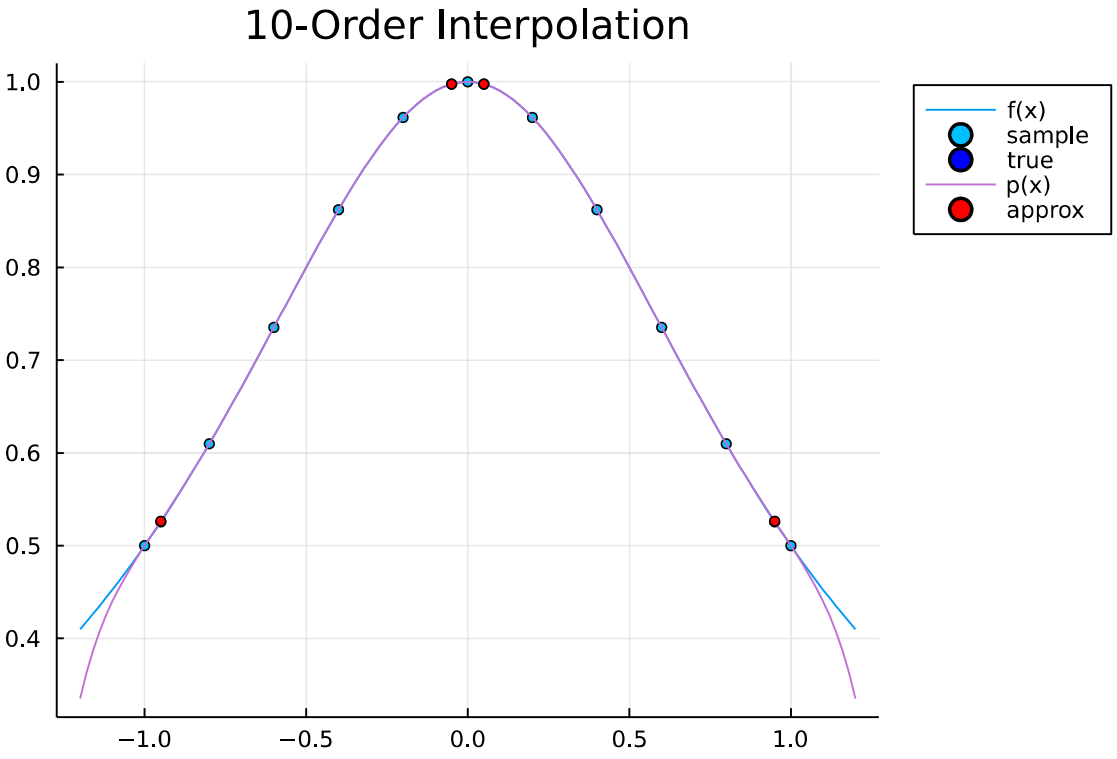
In [120…

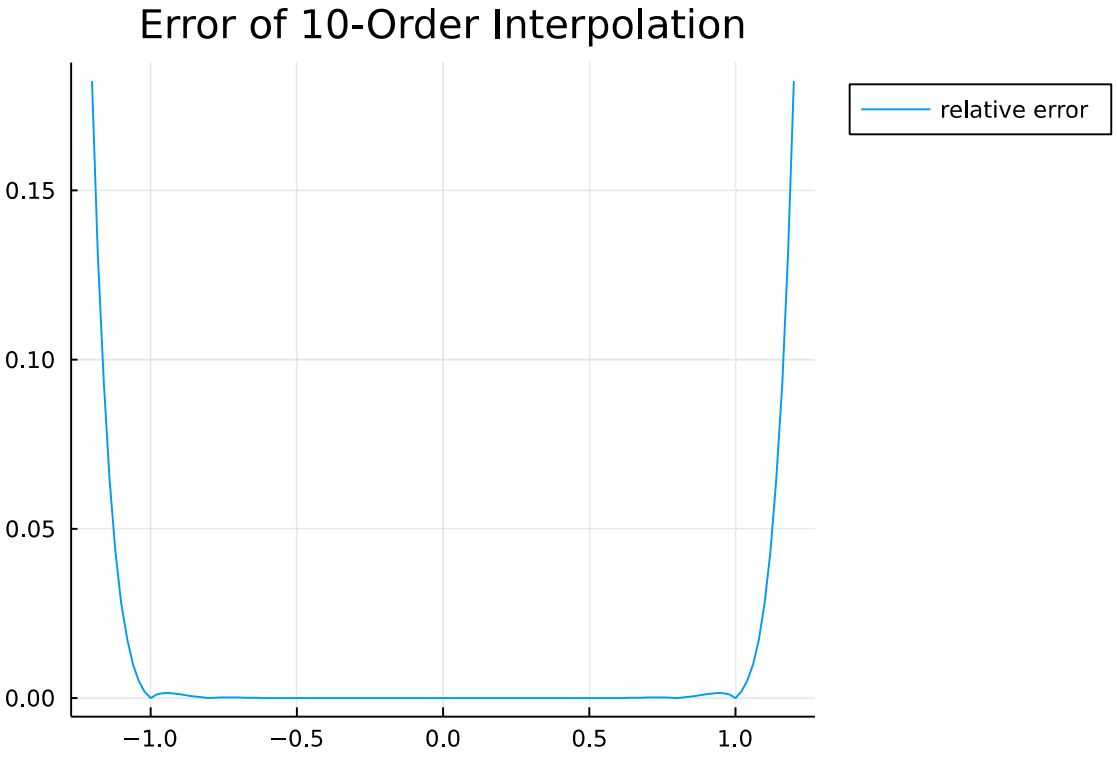
```
f(x) = 1 / (1 + x^2)
```

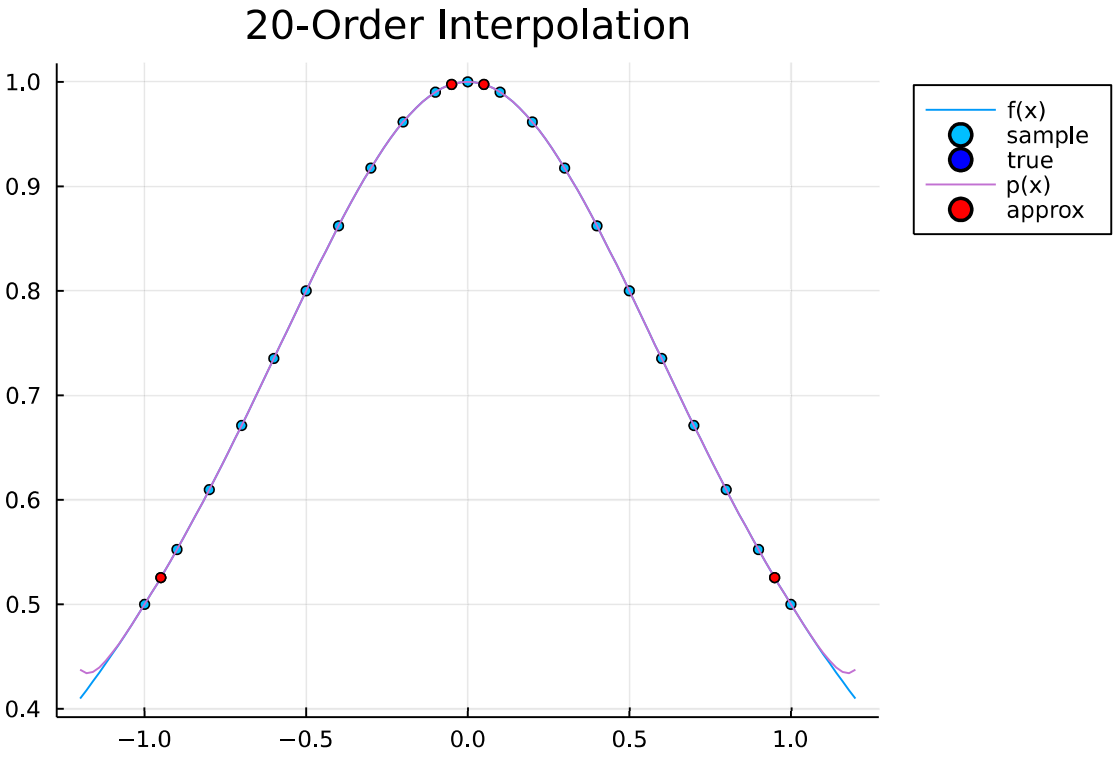
```
split_nums = [5, 10, 20]
test_x = [-0.95, -0.05, 0.05, 0.95]
xlim = [-1, 1]
# ylim = [-1, 2]
ylim = []
println("f(x) = 1 / (1 + x^2)")
show_result(f, split_nums, test_x, xlim, ylim)

f(x) = exp(x)
split_nums = [5, 10, 20]
test_x = [0.75, 1.75, 2.75, 3.75, 4.75]
xlim = [-5, 5]
# ylim = [-1, 10]  # the good-looking ylim is defined manually
ylim = []
println("f(x) = exp(x)")
show_result(f, split_nums, test_x, xlim, ylim)
```

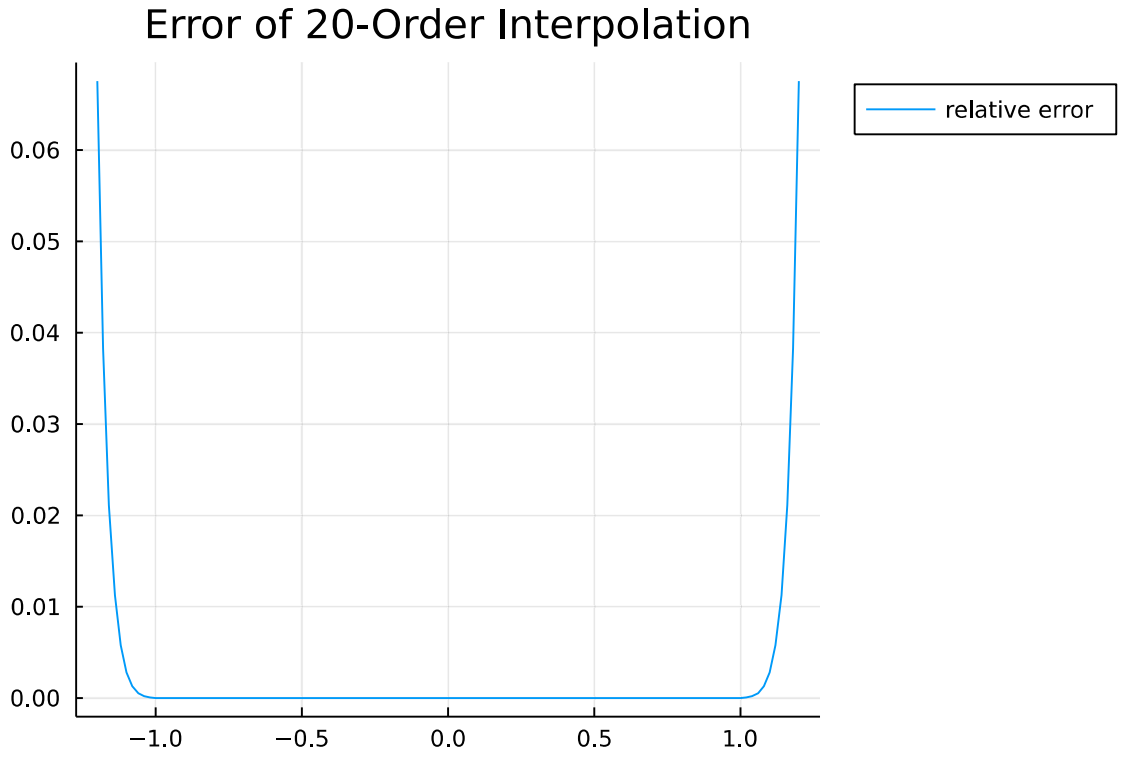


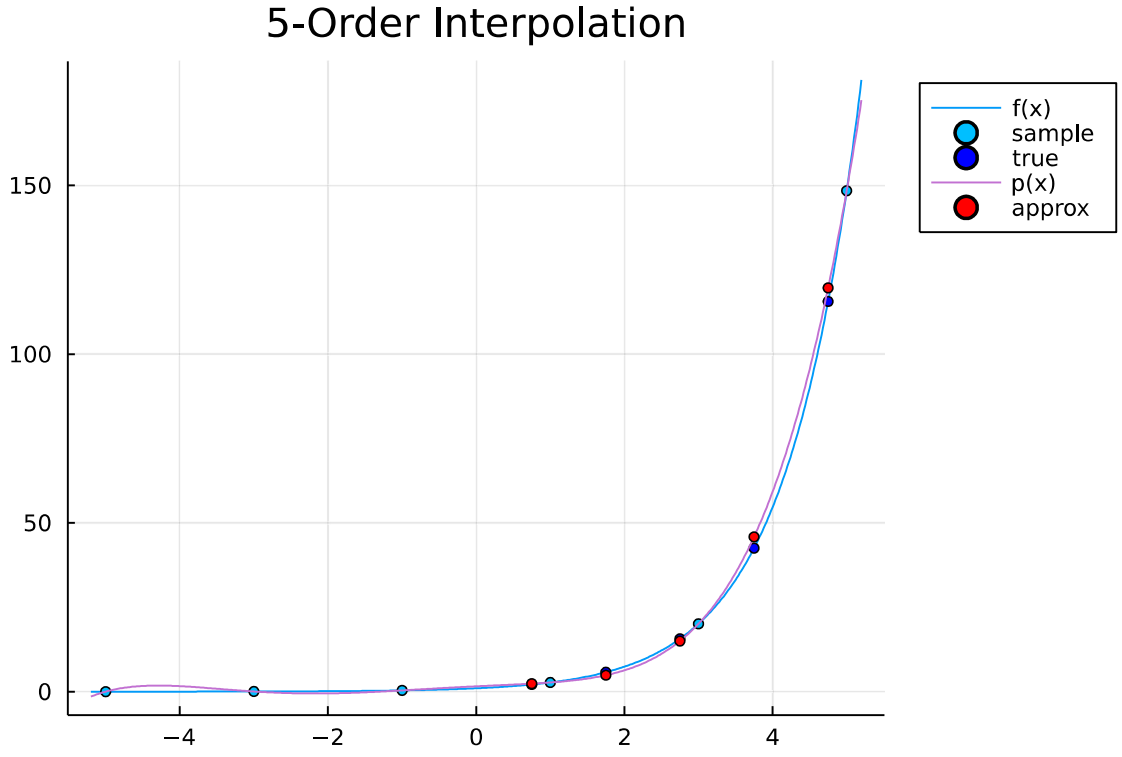

Error of 5-Order Interpolation
relative error
0.20
0.15
0.10
0.05
0.00
−1.0
−0.5
0.0
0.5
1.0

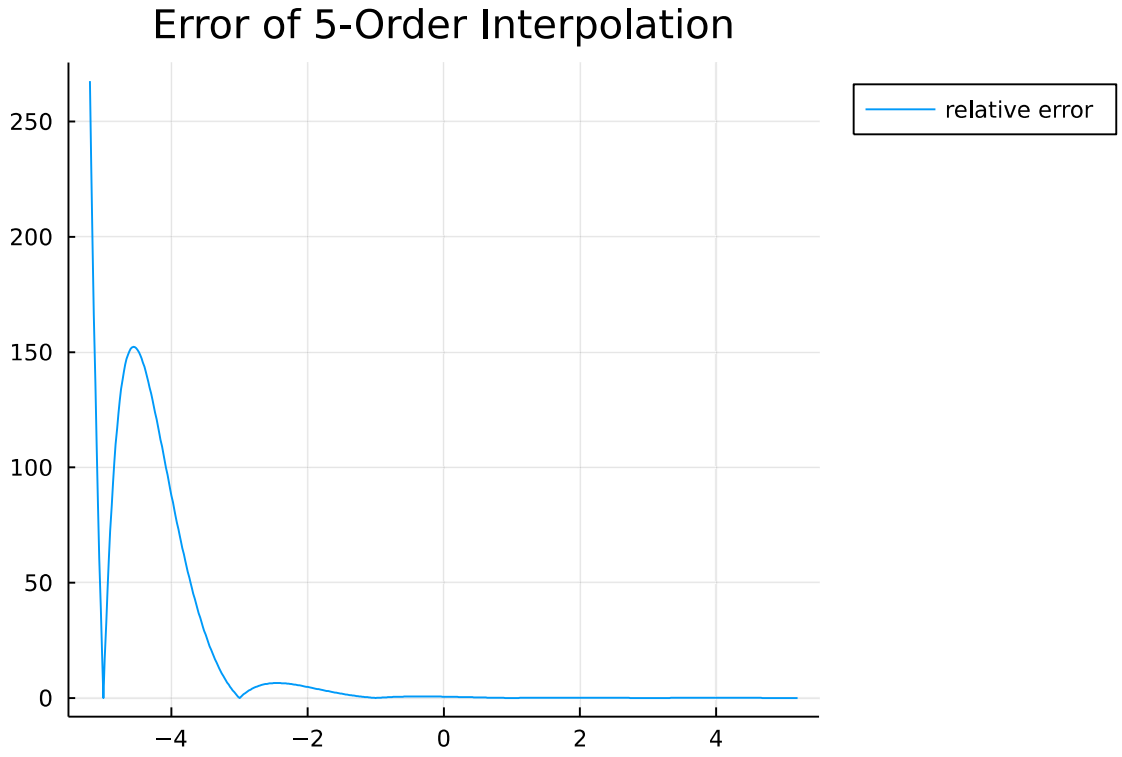

10-Order Interpolation
f(x)
sample
true
p(x)
approx
1.0
0.9
0.8
0.7
0.6
0.5
0.4
−1.0
−0.5
0.0
0.5
1.0

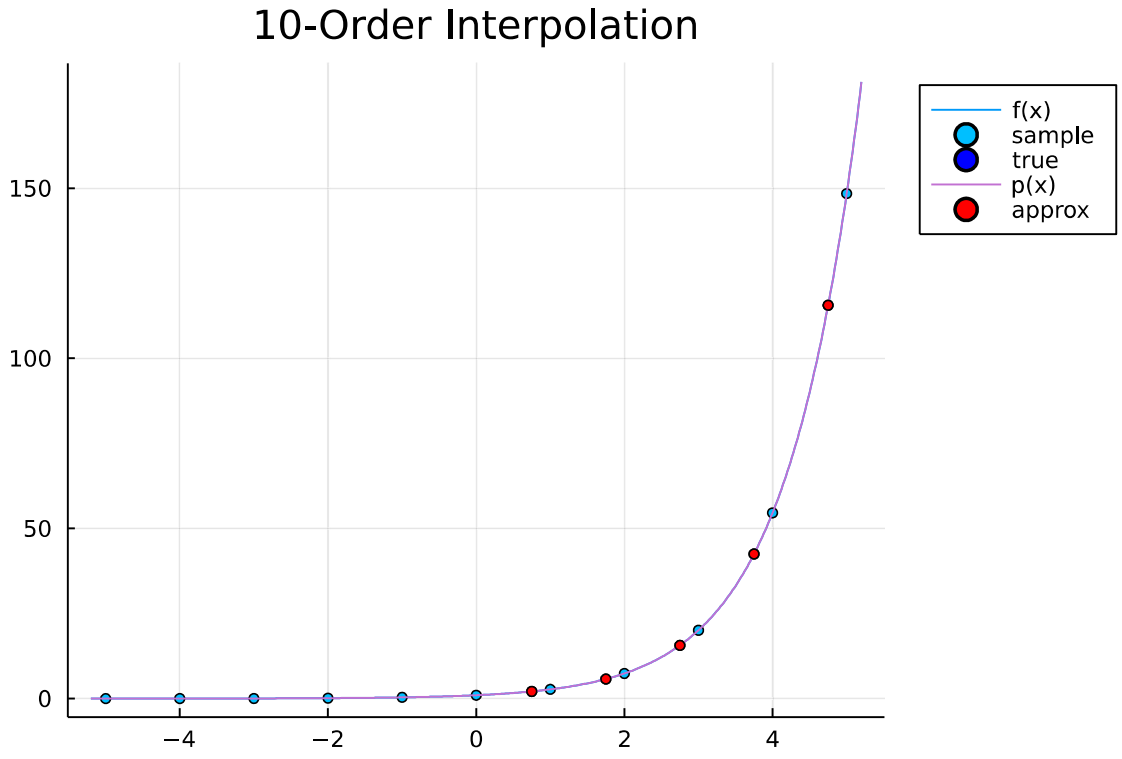

Error of 10-Order Interpolation
relative error
0.15
0.10
0.05
0.00
−1.0
−0.5
0.0
0.5
1.0

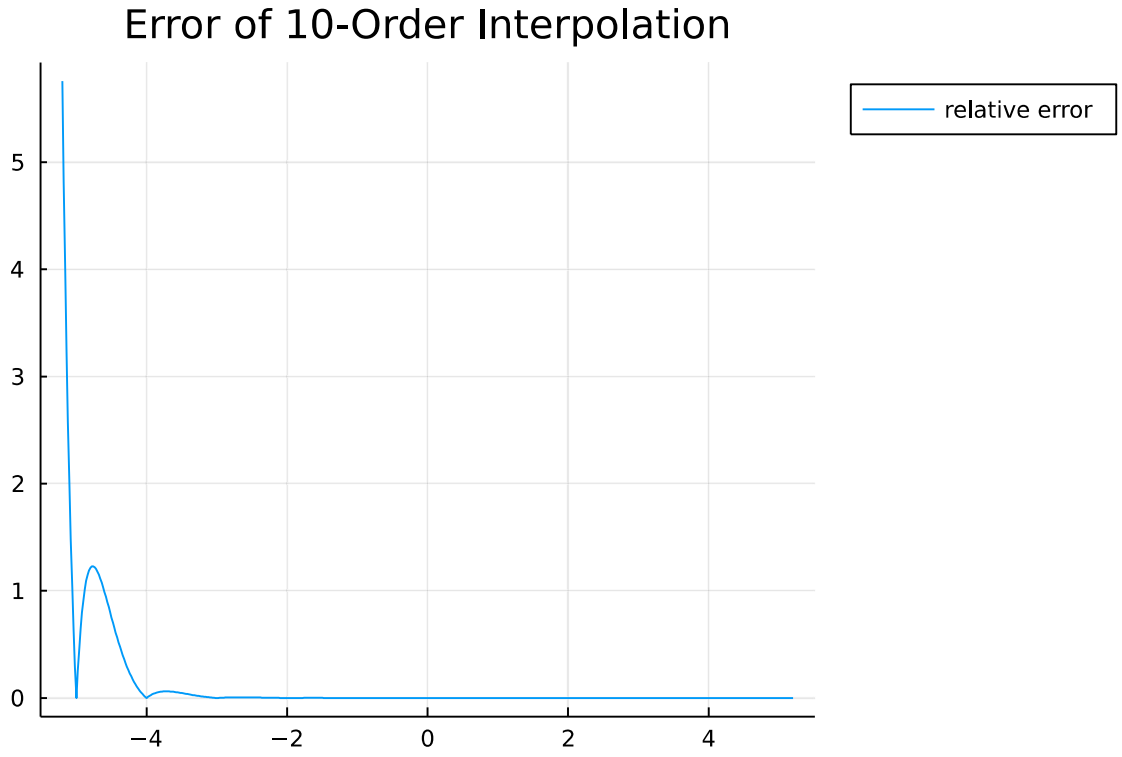

20-Order Interpolation
f(x)
sample
true
p(x)
approx
1.0
0.9
0.8
0.7
0.6
0.5
0.4
−1.0
−0.5
0.0
0.5
1.0

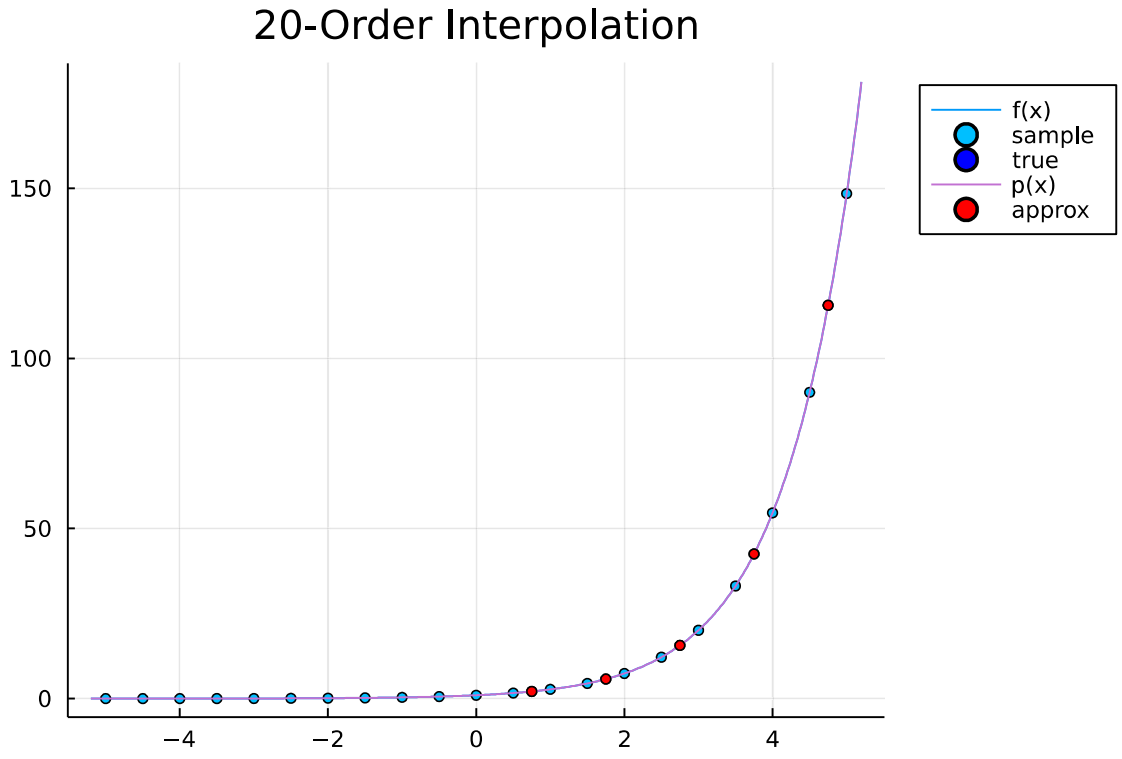

Error of 20-Order Interpolation
relative error
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−1.0
−0.5
0.0
0.5
1.0

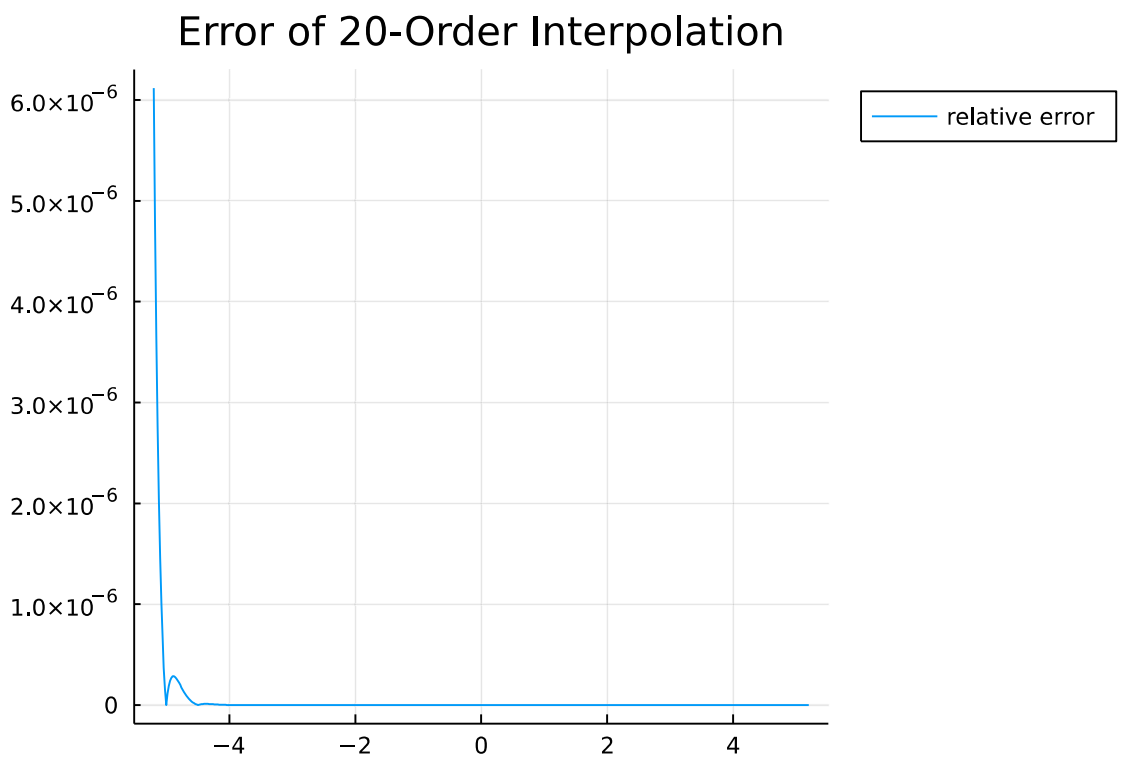

5-Order Interpolation
f(x)
sample
true
p(x)
approx
150
100
50
0
−4
−2
0
2
4


Error of 5-Order Interpolation
relative error
250
200
150
100
50
0
−4
−2
0
2
4


10-Order Interpolation
f(x)
sample
true
p(x)
approx
150
100
50
0
−4
−2
0
2
4


Error of 10-Order Interpolation
relative error
5
4
3
2
1
0
−4
−2
0
2
4


20-Order Interpolation
f(x)
sample
true
p(x)
approx
150
100
50
0
−4
−2
0
2
4




```
f(x) = 1 / (1 + x^2)
5-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.5256241787122208, 0.9975062344139651, 0.9975062344139651, 0.5256241787122
208]
pred_y: [0.517147288602941, 0.992790670955882, 0.9927906709558821, 0.517147288602941
3]
10-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.5256241787122208, 0.9975062344139651, 0.9975062344139651, 0.5256241787122
208]
pred_y: [0.5264079831033747, 0.9975068566175541, 0.9975068566175546, 0.5264079831033
778]
20-Order Interpolation:
test_x: [-0.95, -0.05, 0.05, 0.95]
test_y: [0.5256241787122208, 0.9975062344139651, 0.9975062344139651, 0.5256241787122
208]
pred_y: [0.5256203657463834, 0.997506234424685, 0.997506234424685, 0.525620365746887
5]
f(x) = exp(x)
5-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [2.117000016612675, 5.754602676005731, 15.642631884188171, 42.5210820000627
8, 115.58428452718766]
pred_y: [2.373956870299596, 4.871634881082521, 15.008060923792579, 45.8622567757434
5, 119.62100705608145]
10-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [2.117000016612675, 5.754602676005731, 15.642631884188171, 42.5210820000627
8, 115.58428452718766]
pred_y: [2.1171358945697327, 5.754367325591778, 15.643248061194516, 42.5184309763158
84, 115.60736006305405]
20-Order Interpolation:
test_x: [0.75, 1.75, 2.75, 3.75, 4.75]
test_y: [2.117000016612675, 5.754602676005731, 15.642631884188171, 42.5210820000627
8, 115.58428452718766]
pred_y: [2.1170000166127365, 5.754602676005899, 15.64263188418928, 42.5210820000842
4, 115.58428452936789]
```

问题 4

考虑拉格朗日插值问题，内插比外推更可靠吗?

> 不一定，这取决于函数的性质，但通常我们认为对于连续函数内插的可靠程度更高。
>
> 外推等价于根据已知点预测完全未知点的函数值，但我们所得的插值多项式不含有任何有关待拟合函数的已知点外的信息，根据多项式函数的特性进行外推是不合理的
>
> 而考虑到连续函数，内插则不会对于函数的拟合存在无根据的外推过程，有更高的可靠程度
>
> 从实验结果来看，第一个实例体现的是外推的严重错误，尽管第二个实例中外推所得误差稍小于内插结果，但在事实上这只是所选区间拟合的巧合，而内插误差虽然略高，却也具有相当低的误差和相当高的可靠程度

In [121…

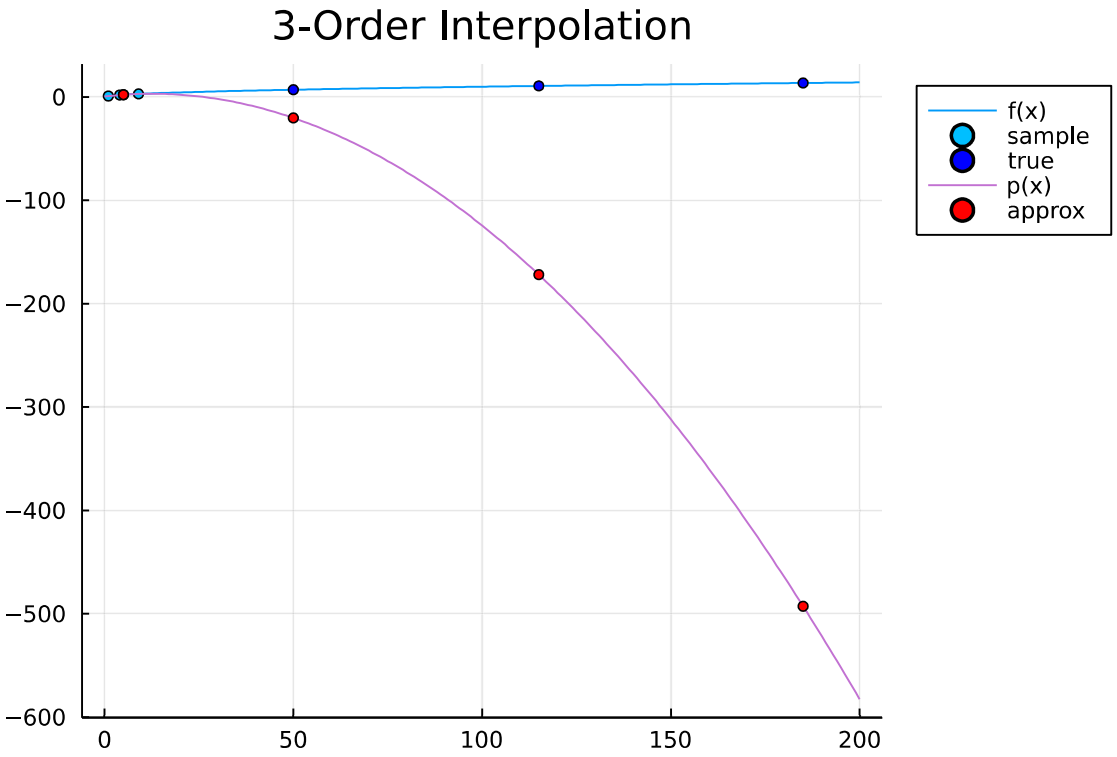
```
function show_result(f::Function, split_nums::Nothing, split_xs::Vector, test_x
    x_min, x_max = xlim
    x_range = x_min:1:x_max+0.2  # x_min cannot be negative
    xs = split_xs
    ys = f.(xs)

    plot(x_range, f.(x_range), label="f(x)")  # plot f(x)
    plot!(legend=:outertopright, title="$(size(split_xs,1))-Order Interpolation")
    plot!(ylim=ylim, yflip=false)  # add ylim
    plot!(xs, ys, seriestype=:scatter, markersize=3, msw=1, color=:deepskyblue, labe

    test_y = f.(test_x)
    plot!(test_x, test_y, seriestype=:scatter, markersize=3, msw=1, color=:blue, lab
    _, pred_y = lagrange(xs, ys, test_x)
    # @time _, pred_y = lagrange(xs, ys, test_x)
    println("test_x: $test_x")
    println("test_y: $test_y")
    println("pred_y: $pred_y")

    series_x = Vector(x_range)
    _, series_y = lagrange(xs, ys, series_x)  # compute the interporlation function
    plot!(series_x, series_y, label="p(x)")  # add p(x) function curve
    display(plot!(test_x, pred_y, seriestype=:scatter, markersize=3, msw=1, color=:r
    display(show_error(f, "Error of $(size(split_xs,1))-Order Interpolation", series
end
```

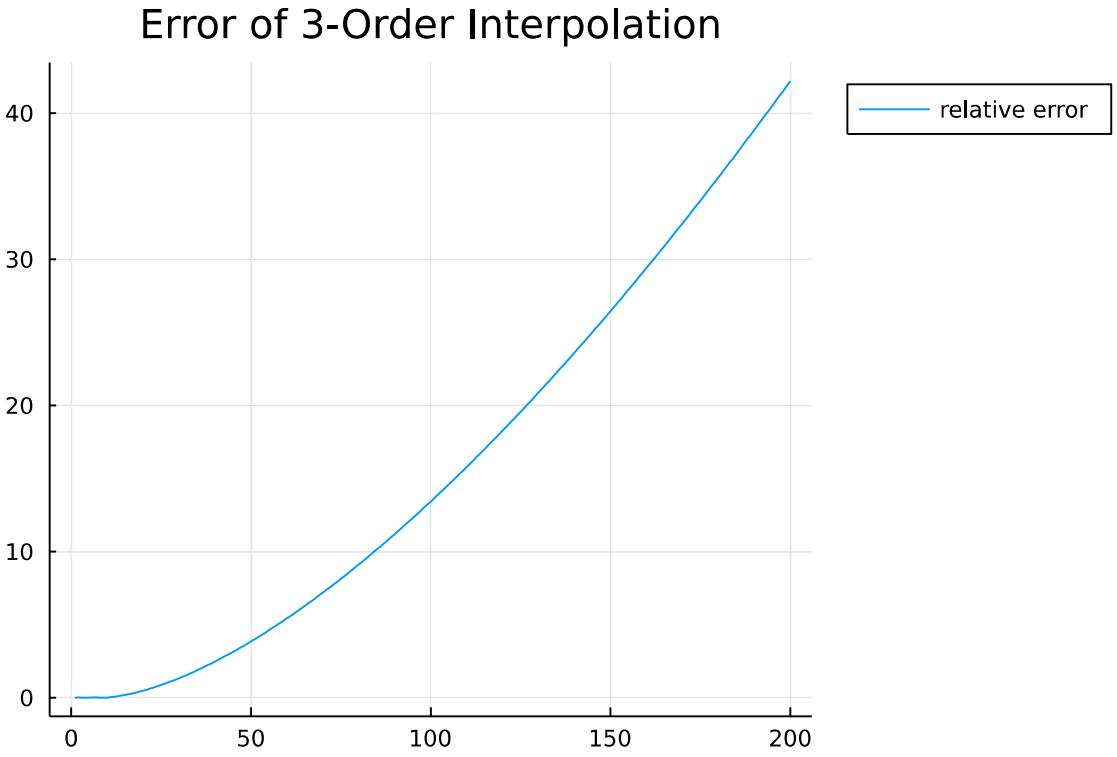
```
show_result (generic function with 2 methods)
```

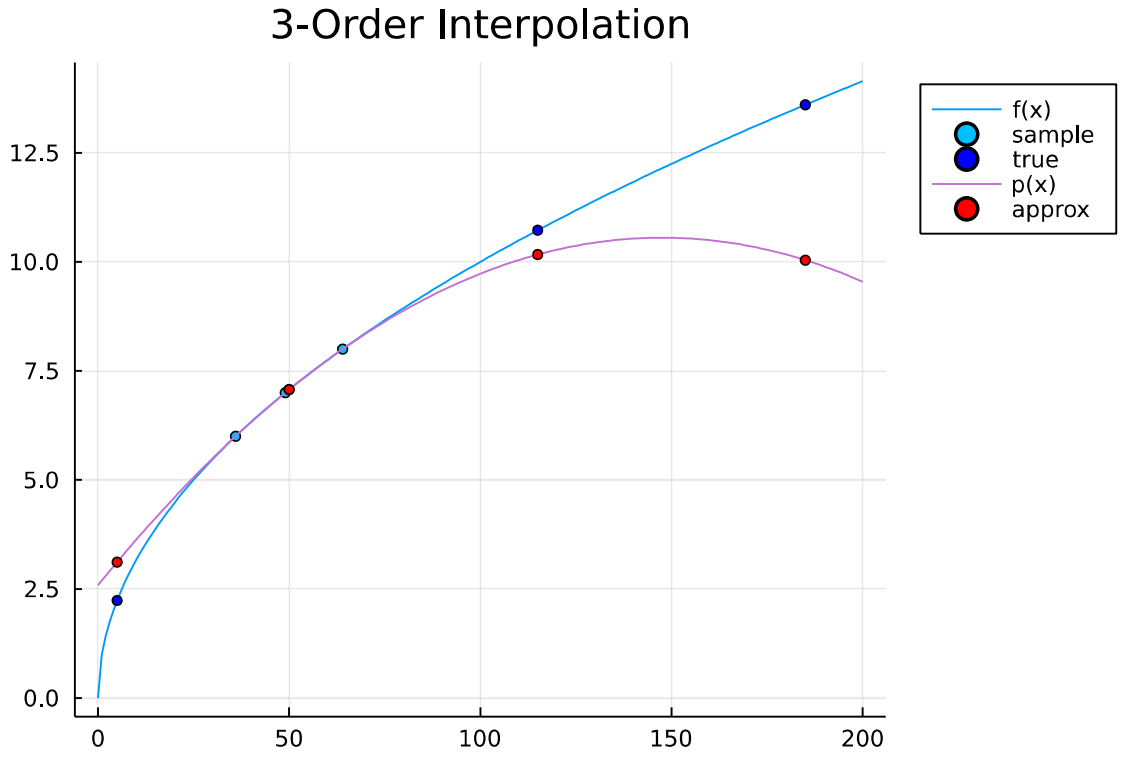
In [122…

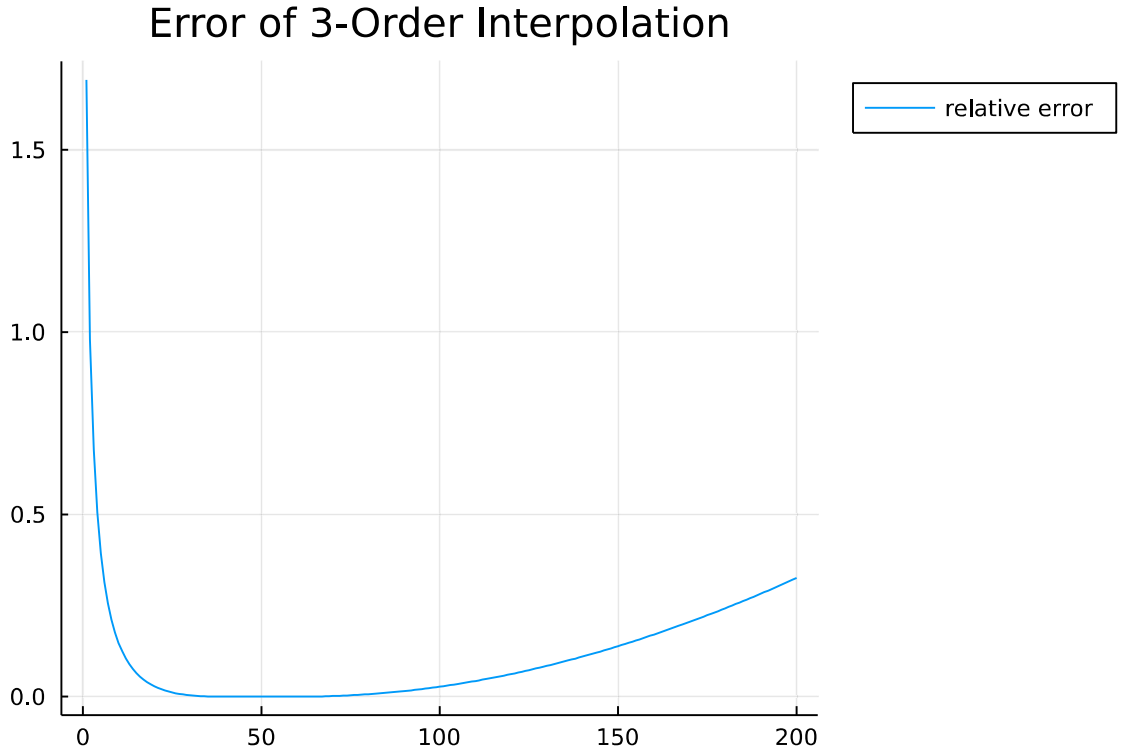
```
f(x) = sqrt(x)
split_xs = [1, 4, 9]
test_x = [5, 50, 115, 185]
xlim = [0, 200]
# ylim = [-1, 2]
ylim = []
println("f(x) = sqrt(x)")
show_result(f, nothing, split_xs, test_x, xlim, ylim)

f(x) = sqrt(x)
split_xs = [36, 49, 64]
test_x = [5, 50, 115, 185]
xlim = [0, 200]
# ylim = [-1, 2]
```

```
ylim = []
println("f(x) = sqrt(x)")
show_result(f, nothing, split_xs, test_x, xlim, ylim)
```







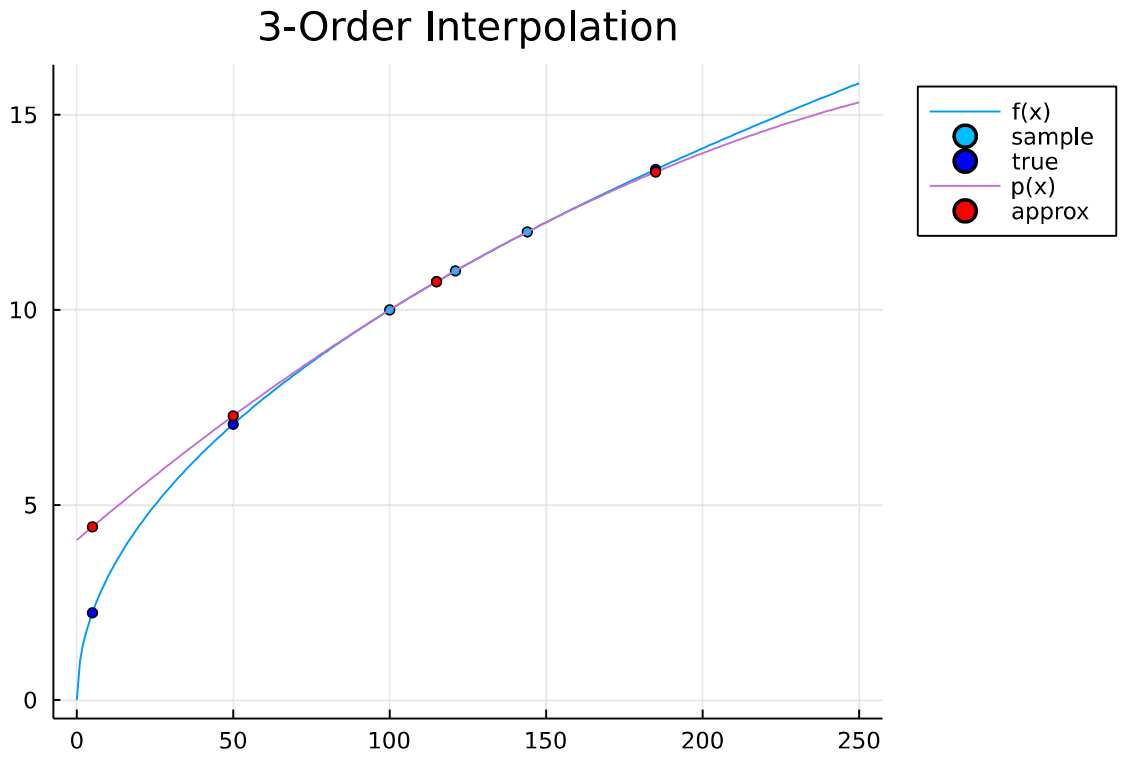


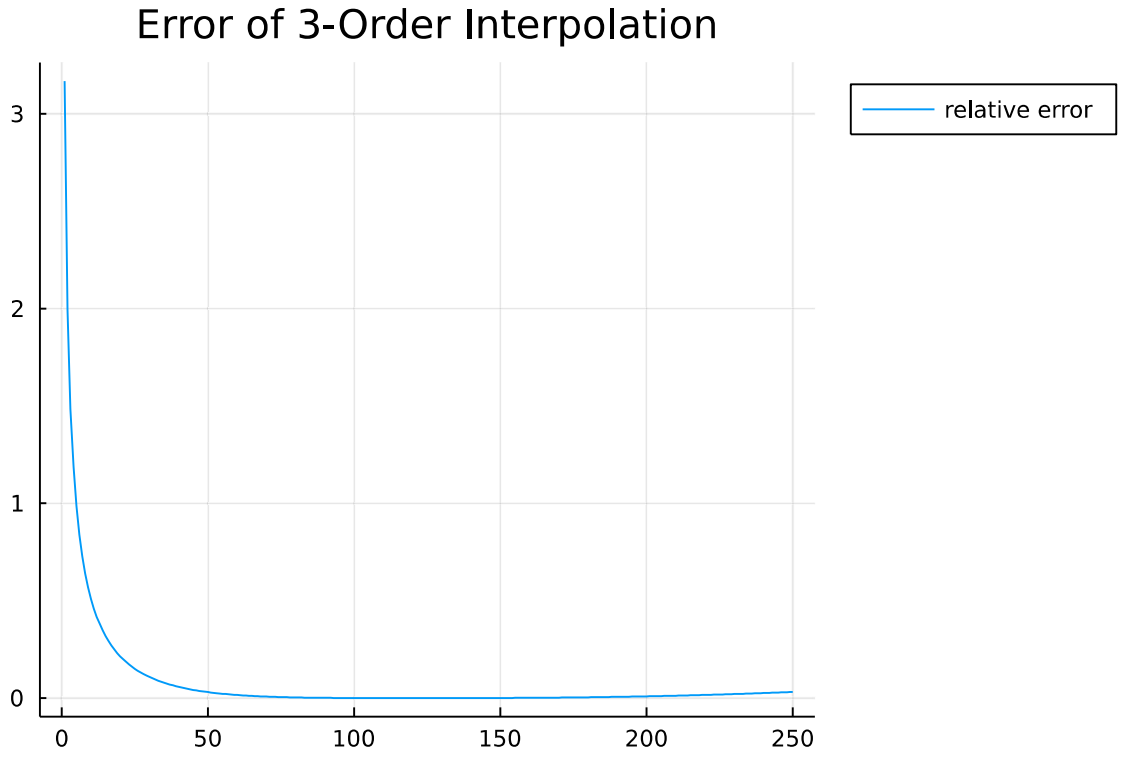

```
f(x) = sqrt(x)
test_x: [5, 50, 115, 185]
test_y: [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.60147050873544
4]
pred_y: [2.2666666666666666, -20.23333333333332, -171.89999999999998, -492.733333333
3331]
f(x) = sqrt(x)
test_x: [5, 50, 115, 185]
test_y: [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.60147050873544
4]
pred_y: [3.1157509157509153, 7.0717948717948715, 10.167032967032995, 10.038827838827
785]
```

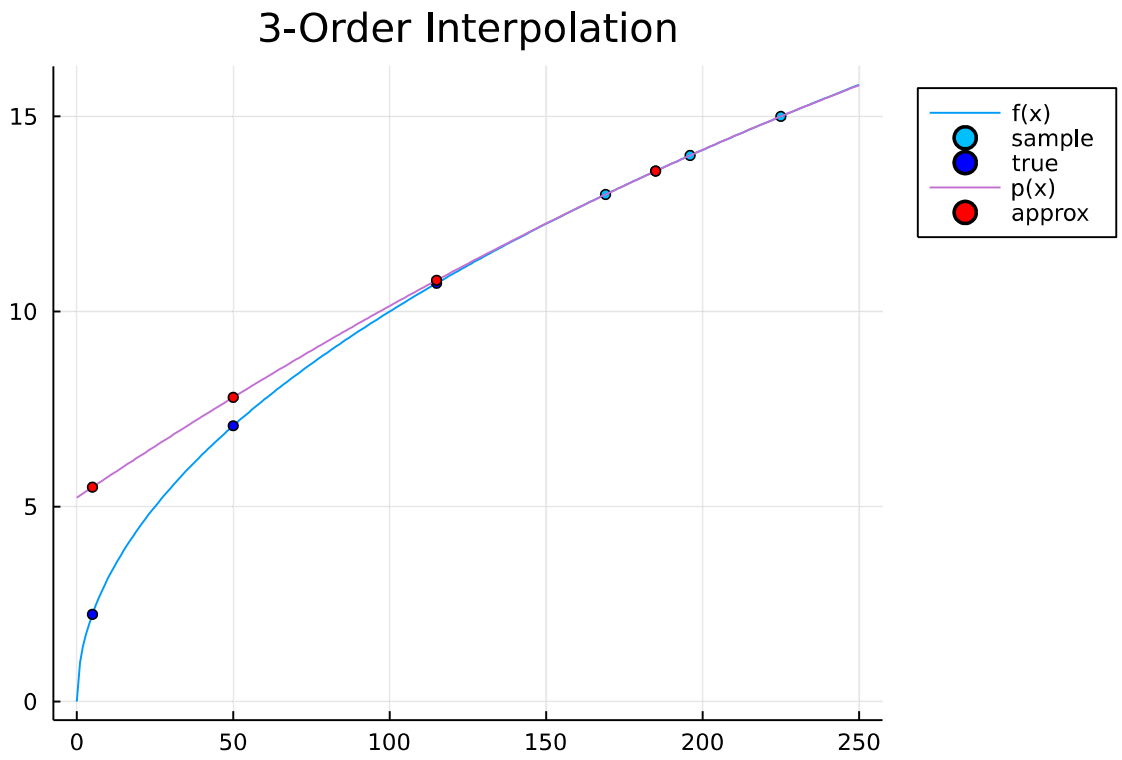
In [123…

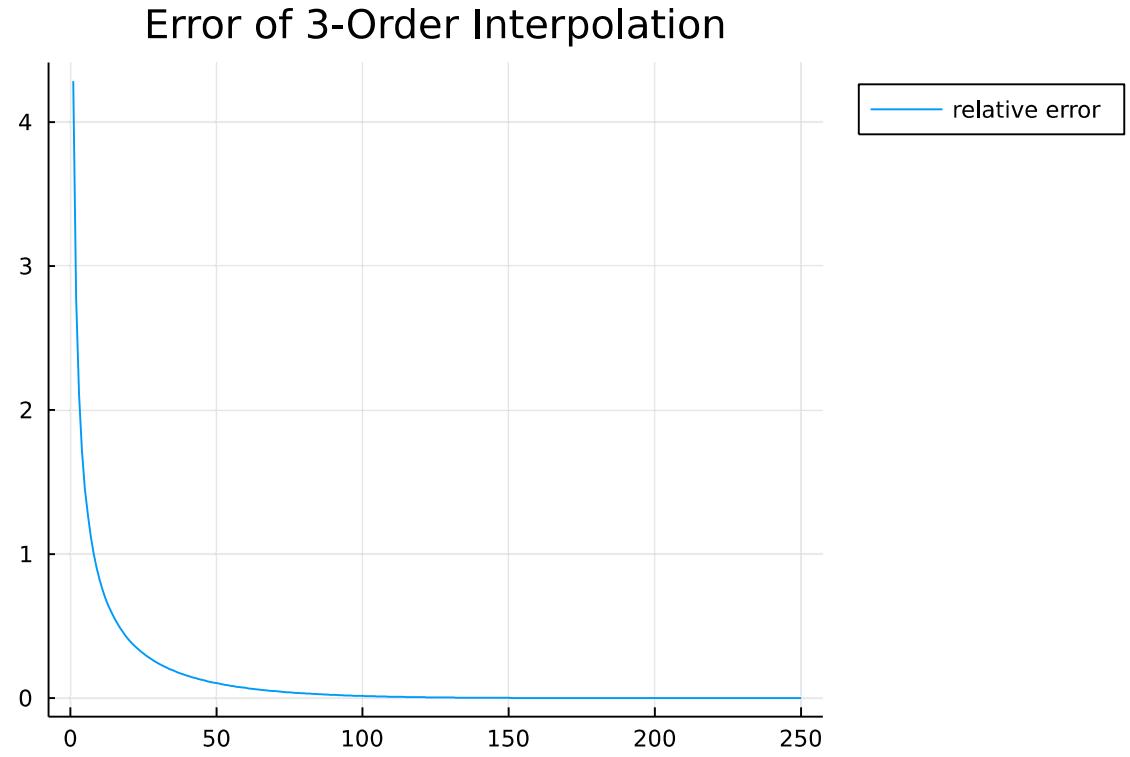
```
f(x) = sqrt(x)
split_xs = [100, 121, 144]
```

```
test_x = [5, 50, 115, 185]
xlim = [0, 250]
# ylim = [-1, 2]
ylim = []
println("f(x) = sqrt(x)")
show_result(f, nothing, split_xs, test_x, xlim, ylim)

f(x) = sqrt(x)
split_xs = [169, 196, 225]
test_x = [5, 50, 115, 185]
xlim = [0, 250]
# ylim = [-1, 2]
ylim = []
println("f(x) = sqrt(x)")
show_result(f, nothing, split_xs, test_x, xlim, ylim)
```




Error of 3-Order Interpolation
relative error
3
2
1
0
0
50
100
150
200
250


3-Order Interpolation
f(x)
sample
true
p(x)
approx
15
10
5
0
0
50
100
150
200
250




```
f(x) = sqrt(x)
test_x: [5, 50, 115, 185]
test_y: [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.60147050873544
4]
pred_y: [4.439111613024664, 7.284961415396204, 10.722755505364201, 13.53566723131940
8]
f(x) = sqrt(x)
test_x: [5, 50, 115, 185]
test_y: [2.23606797749979, 7.0710678118654755, 10.723805294763608, 13.60147050873544
4]
pred_y: [5.4971720488960045, 7.80012771392083, 10.800492610837438, 13.60062032475825
5]
```

思考题

1. 对实验 1 存在的问题，应如何解决?

   当插值多项式次数过高的时候会出现Runge现象，插值多项式在距离已知点位置较远处会剧烈震荡，越靠近端点，逼近的效果越差，这表明了节点的密集不一定能保证在两节点间插值函数逼近程度的上升。

   这一问题的解决方案主要有两类。

   一个是从插值函数的二阶导数剧烈变化出发，修改插值条件对插值函数的二阶导数进行限制，如使用Hermite型插值；

   另一种是将长区间划分为若干个小区间，在每一个小区间上分别做低次插值来避免Runge现象，逼近效果要比在整个区间上用告辞光滑差值效果更好，即使用分段插值和样条插值。

   > 参考自教材《数值分析原理》吴勃英 123页

2. 对实验 2 存在的问题的回答，试加以说明

首先不一定，从精度上考虑虽然有一定的合理性，但插值节点过于密集时，一方面计算量增大却没提高对于精度计算的收益，另一方面区间缩短、节点增加并不能保证两节点间能很好的逼近函数，反而有可能出现Runge现象。但合理的对区间长度进行选择，同时采用低次插值来避免Runge现象，能够得到较好的拟合效果。

不过，实例中对于函数$f(x)=\frac{1}{1+x^2}$，较短区间的插值效果比长区间插值更好

而函数$f(x)=e^x$无论是长区间还是短区间插值，都能得到相对较好的拟合效果，但短区间插值相对误差更低

3. 略

4. 如何理解插值问题中的内插和外推?

   通常我们认为对于连续函数内插的可靠程度高于外推，因为对于未知的连续函数而言，我们无法预知任何在已知点信息之外的有关函数的信息，无法简单通过多项式插值来对函数趋势进行判断。

   外推等价于根据已知点预测完全未知点的函数值，但我们所得的插值多项式不含有任何有关待拟合函数的已知点外的信息，根据多项式函数的特性进行外推是不合理的

   而考虑到连续函数，内插则不会对于函数的拟合存在无根据的外推过程，同时是有限的利用已知插值节点的关系，故有更高的可靠程度

   从实验结果来看，第一个实例体现的是外推的严重错误，尽管第二个实例中外推所得误差稍小于内插结果，但在事实上这只是所选区间拟合的巧合，而内插误差虽然略高，却也具有相当低的误差和相当高的可靠程度

实验题目2 龙贝格(Romberg)积分法

参考资料

julia 数值积分 https://blog.csdn.net/m0_37816922/article/details/103475445

julia SymPy.jl https://github.com/JuliaPy/SymPy.jl

SymPy.jl example https://docs.juliahub.com/SymPy/KzewI/1.0.29/introduction/

SymPy.jl assumption https://docs.sympy.org/dev/modules/core.html#module-sympy.core.assumptions

julia calculus https://docs.juliahub.com/CalculusWithJulia/AZHbv/0.0.16/

代码实现

In [19]:
```
using Printf
using Plots
using SymPy
```

In [20]:
```
function romberg(f::Function, xlim, iternum, ϵ)
    a, b = xlim
    n = iternum

    m = 1
    h = (b - a)    # bug here
    T1 = 1 / 2 * h * (f(a) + f(b))
    @printf("\nT:%16.9f", T1)
    # missing term: f(0.5)
    # starting point error, missed i=1
    T2, C1, C2, S1, S2, R1, R2 = zeros(7)
    for i = 1:n
        ii = 2^(i - 1)
        tmpsum = 0.0
        for k = 1:ii
            tmpsum += f(a + (k - 1 / 2) * h)
        end
        T2 = 1 / 2 * T1 + 1 / 2 * h * tmpsum
        @printf("\nT:%16.9f", T2)

        S2 = 1 / 3 * (4T2 - T1)
        @printf("\tS:%16.9f", S2)

        if m > 1
            C2 = 1 / 15 * (16S2 - S1)
            @printf("\tC:%16.9f", C2)
        end

        if m > 2
            R2 = 1 / 63 * (64C2 - C1)
            @printf("\tR:%16.9f", R2)
        end
        if m > 3
            tol = abs(R2 - R1)
            if tol < ϵ
```

```
                println("\nAccuracy requirement satisfied.\n")
                return
            end
        end
        # @label PARAM_UPDATE
        R1, C1, S1, T1 = R2, C2, S2, T2
        h /= 2
        m += 1
    end
    m
end
```

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

测试代码

Test 1 - Simple

In [21]:
```
# f(x) = x^2 * exp(x)
# f(x) = 1/x
# ϵ = 1e-6
# xlim = 1, 3

# romberg(f,xlim,10,ϵ)
```

In [22]:
```
# # compute definite integration
# @syms x y z
# f = 1 / x
# @time integrate(f, (x, 1, 3)) |> float
```

In [23]:
```
# f = 1 / (1 + x^4)
# @time integrate(f, (x, 1, 3)) |> float
```

实验题目

问题 1

In [24]:
```
iter_num = 30


f(x) = x^2 * exp(x)
ϵ = 1e-6
xlim = 0, 1
println("f(x) = x^2 * exp(x)")
romberg(f, xlim, iter_num, ϵ)

f(x) = exp(x)sin(x)
ϵ = 1e-6
xlim = 1, 3
println("f(x) = exp(x)sin(x)")
romberg(f, xlim, iter_num, ϵ)


f(x) = 4 / (1 + x^2)
ϵ = 1e-6
xlim = 0, 1
println("f(x) = 4 / (1 + x^2)")
romberg(f, xlim, iter_num, ϵ)

f(x) = 1 / (x + 1)
```

```
ϵ = 1e-6
xlim = 0, 1
println("f(x) = 1 / (x + 1)")
romberg(f, xlim, iter_num, ϵ)
```

```
f(x) = x^2 * exp(x)

T:      1.359140914
T:      0.885660616     S:      0.727833850
T:      0.760596332     S:      0.718908238     C:      0.718313197
T:      0.728890177     S:      0.718321459     C:      0.718282340     R:      0.718
281850
T:      0.720935779     S:      0.718284313     C:      0.718281837     R:      0.718
281829
Accuracy requirement satisfied.

f(x) = exp(x)sin(x)

T:      5.121826420
T:      9.279762907     S:     10.665741736
T:     10.520554284     S:     10.934151409     C:     10.952045388
T:     10.842043468     S:     10.949206529     C:     10.950210203     R:     10.950
181074
T:     10.923093890     S:     10.950110697     C:     10.950170975     R:     10.950
170352
T:     10.943398421     S:     10.950166598     C:     10.950170325     R:     10.950
170315
Accuracy requirement satisfied.

f(x) = 4 / (1 + x^2)

T:      3.000000000
T:      3.100000000     S:      3.133333333
T:      3.131176471     S:      3.141568627     C:      3.142117647
T:      3.138988494     S:      3.141592502     C:      3.141594094     R:      3.141
585784
T:      3.140941612     S:      3.141592651     C:      3.141592661     R:      3.141
592638
T:      3.141429893     S:      3.141592654     C:      3.141592654     R:      3.141
592654
Accuracy requirement satisfied.

f(x) = 1 / (x + 1)

T:      0.750000000
T:      0.708333333     S:      0.694444444
T:      0.697023810     S:      0.693253968     C:      0.693174603
T:      0.694121850     S:      0.693154531     C:      0.693147901     R:      0.693
147478
T:      0.693391202     S:      0.693147653     C:      0.693147194     R:      0.693
147183
Accuracy requirement satisfied.
```

思考题

1. 略

2. 在实验 1 中二分次数和精度的关系如何?

二分次数越多所求的精度越高，通常预设较大的二分次数来确保计算结果有足够的精度，同时也设定早停需要满足的精度要求，避免达到所需精度之后继续计算导致增加的运算量

3. 略

4. 略

实验题目3 四阶龙格-库塔(Runge-Kutta)方法

参考资料

julia ordinary differential equations tutorial
https://diffeq.sciml.ai/stable/tutorials/ode_example/

intro to solving differential equations in julia https://www.youtube.com/watch?v=KPEqYtEd-zY

julia ode solver type: Runge-Kutta https://diffeq.sciml.ai/stable/solvers/ode_solve/#Explicit-Runge-Kutta-Methods

julia ode problem type https://diffeq.sciml.ai/stable/types/ode_types/#ode_prob

julia ode speed up perf
https://diffeq.sciml.ai/stable/features/performance_overloads/#performance_overloads

julia ode common solver option
https://diffeq.sciml.ai/stable/basics/common_solver_opts/#solver_options

代码实现

In [1]:
```
using DifferentialEquations
using Plots
using LaTeXStrings
using Statistics
using ImplicitEquations
```

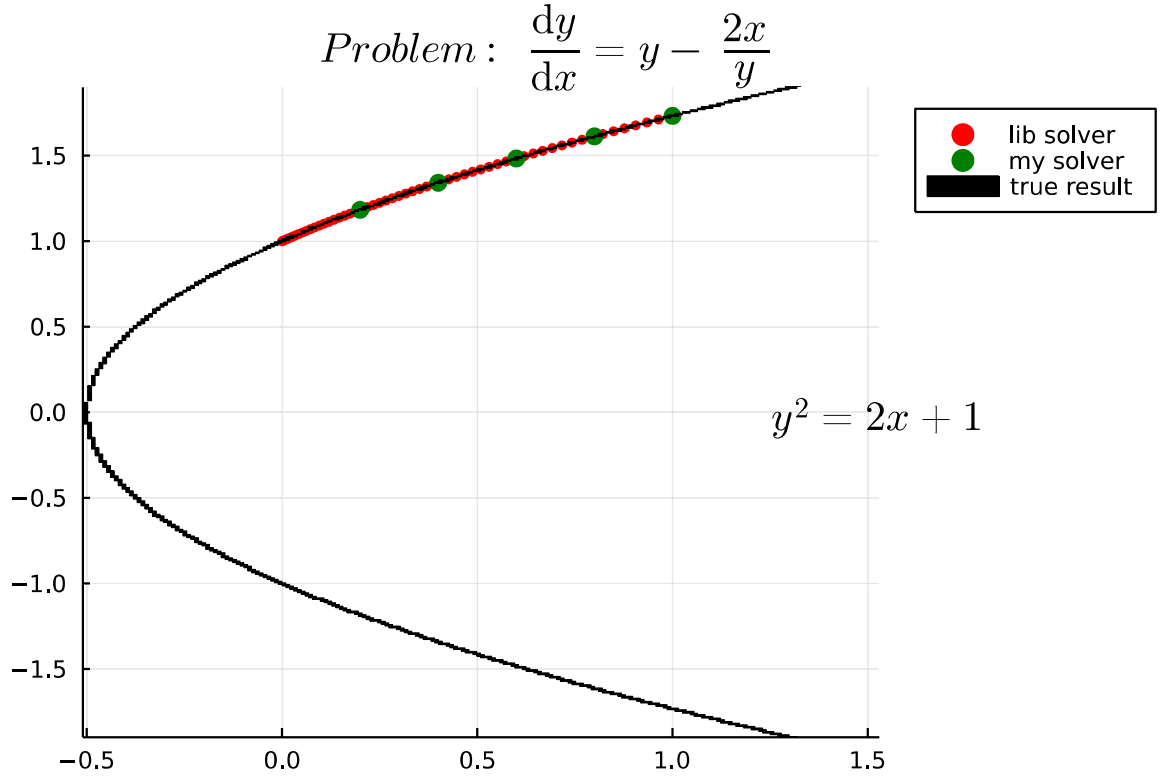
In [2]:
```
function rungekutta(f::Function, xspan, y0, num)
    a, b = xspan
    x0 = a
    h = (b - a) / num
    xs, ys = zeros(num), zeros(num)
    for n = 1:num
        K1 = h * f(x0, y0)
        K2 = h * f(x0 + h / 2, y0 + K1 / 2)
        K3 = h * f(x0 + h / 2, y0 + K2 / 2)
        K4 = h * f(x0 + h, y0 + K3)
        x1 = x0 + h
        y1 = y0 + 1 / 6 * (K1 + 2K2 + 2K3 + K4)
        xs[n], ys[n] = x0, y0 = x1, y1
    end
    println("Runge-Kutta:")
    println("x: $xs")
    println("y: $ys")
    xs, ys
end
```

Out[2]:
```
rungekutta (generic function with 1 method)
```

测试代码

Test 1 - Simple

In [3]:
```
f(y, p, x) = y - 2x / y
xspan = (0.0, 1.0)
y0 = 1.0
prob = ODEProblem(f, y0, xspan)
alg = RK4()
sol = solve(prob, alg, reltol=1e-8, abstol=1e-8)
plot(title=L"~~~~~~~~~~~~ Problem:\ \frac{\mathrm{d} y}{\mathrm{d} x}=y-\frac{2x}{y}
plot!(sol.t, sol.u, seriestype=:scatter, markersize=3, msw=0, color=:red, label="li

f(x, y) = y - 2x / y
# xspan = (0.0, 1.0)
# y0 = 1.0
num = convert(Integer, 1.0 / 0.2)
xs, ys = rungekutta(f, xspan, y0, 5)

p = plot!(xs, ys, seriestype=:scatter, markersize=5, msw=0, color=:green, label="my
# display(p)
f(x, y) = y^2 - 2x - 1
p = plot!(f == 0.0, color=:green, linewidth=0.1, label="true result")  # \Equal[Tab
p = plot!(legend=:outertopright, xlim=(-0.51, 1.53), ylim=(-1.9, 1.9))
x = xlims(p)[2]
y = mean(ylims(p))
ymax = ylims(p)[2]
annotate!(x, y, L"y^2=2x+1", :black)
display(p)
```




```
Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [1.183229287445307, 1.3416669298526065, 1.4832814583502616, 1.6125140416775265,
1.7321418826911932]
```

实验题目

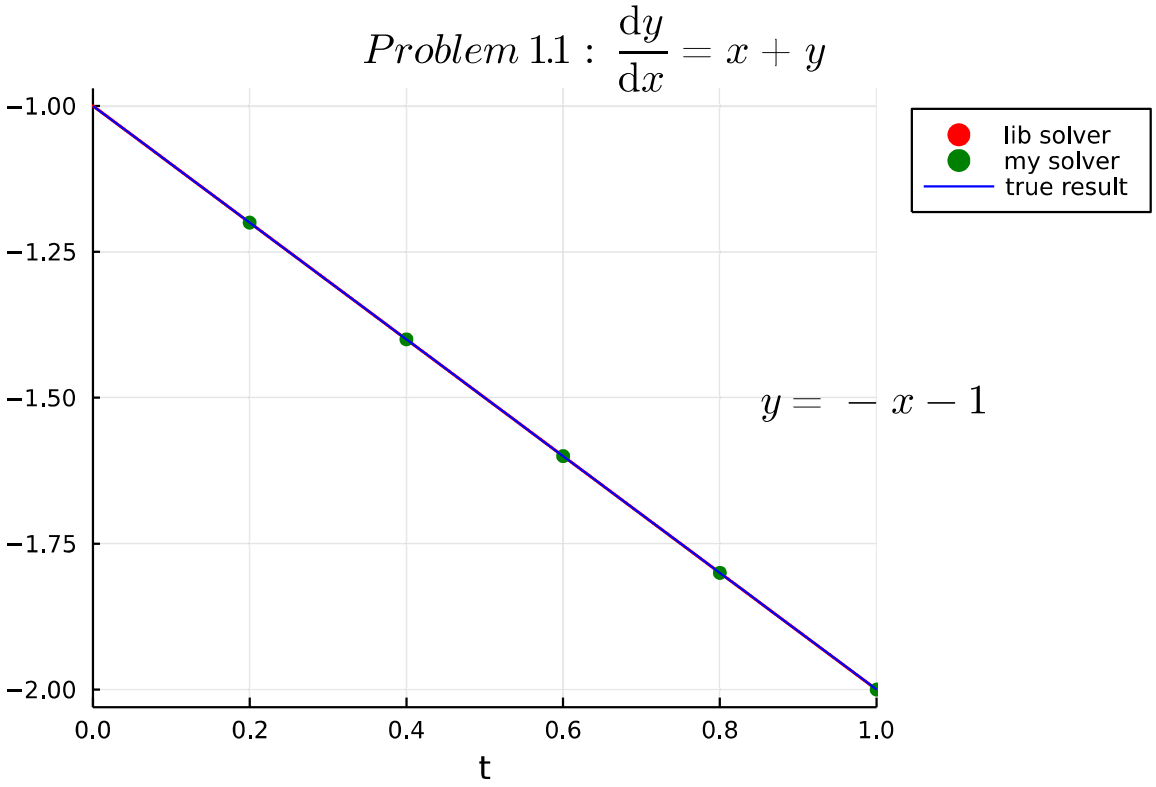
In [4]:
```
function show_plot(p, f::Function, tspan, u0::Float64, reltol, abstol, dense::Bo
    prob = ODEProblem(f, u0, tspan)
    alg = RK4()
```

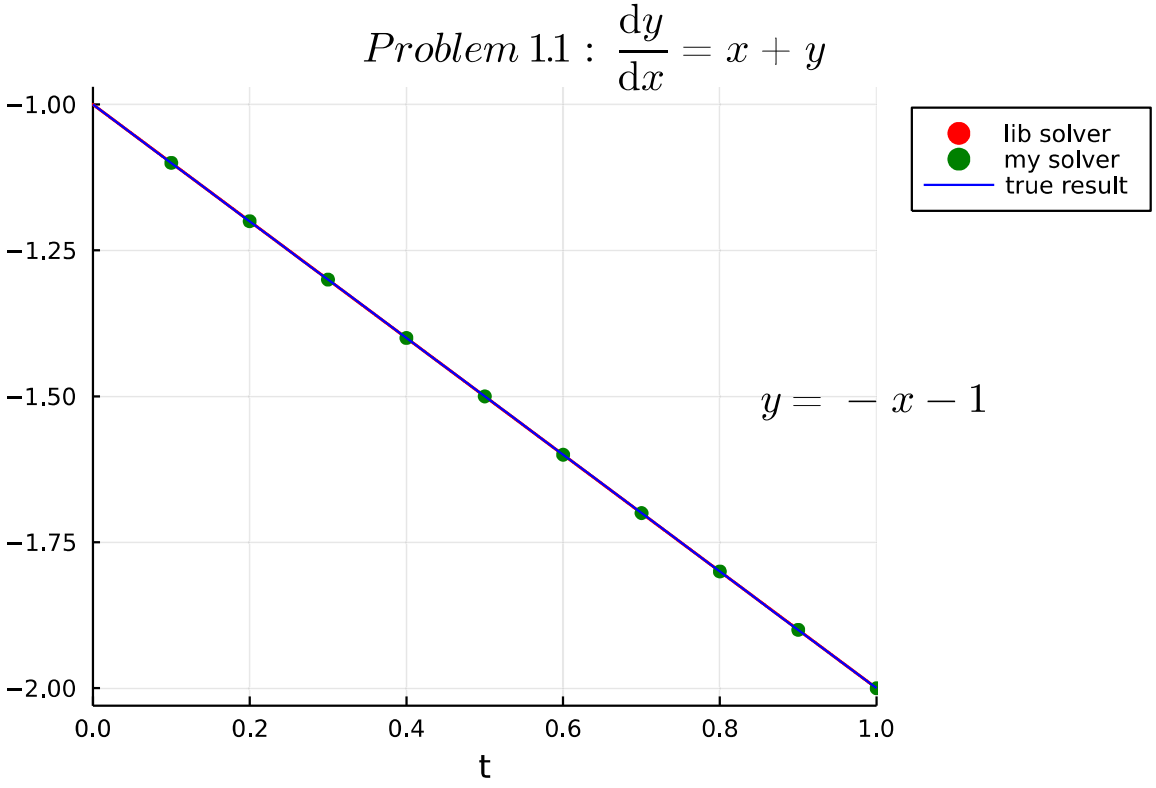
```
    sol = solve(prob, alg, reltol=1e-8, abstol=1e-8)
    if dense
        plot!(sol, seriestype=:scatter, markersize=1, msw=0, color=:red, label="lib
    else
        plot!(sol.t, sol.u, seriestype=:scatter, markersize=2, msw=0, color=:red, l
    end
end
function show_plot(p, f::Function, xspan, y0::Float64, iternum::Integer)
    xs, ys = rungekutta(f, xspan, y0, iternum)
    plot!(xs, ys, seriestype=:scatter, markersize=4, msw=0, color=:green, label="my
end
function show_plot(p, f::Function, show::Bool, text)
    x = xlims(p)[2]
    y = mean(ylims(p))
    annotate!(x, y, text, :black)
    if show
        p = plot!(f, color=:blue, label="true result")
    else
        p = plot!(f, color=:blue, label="true result")
    end
    p
end
function show_result(f1::Function, f2::Function, f3::Function, xspan, y0, iterr
    println("\n\n" * title)
    for iternum in iternums
        print("\nIternum: $iternum\t")
        p = plot(legend=:outertopright, title=L"~~~~~~~~~~~~~~~~~~~~~" * title)
        p = show_plot(p, f1, xspan, y0, 1e-8, 1e-8, dense)
        p = show_plot(p, f2, xspan, y0, iternum)
        p = show_plot(p, f3, show, text)
        display(p)
    end
end
```

Out[4]: show_result (generic function with 1 method)

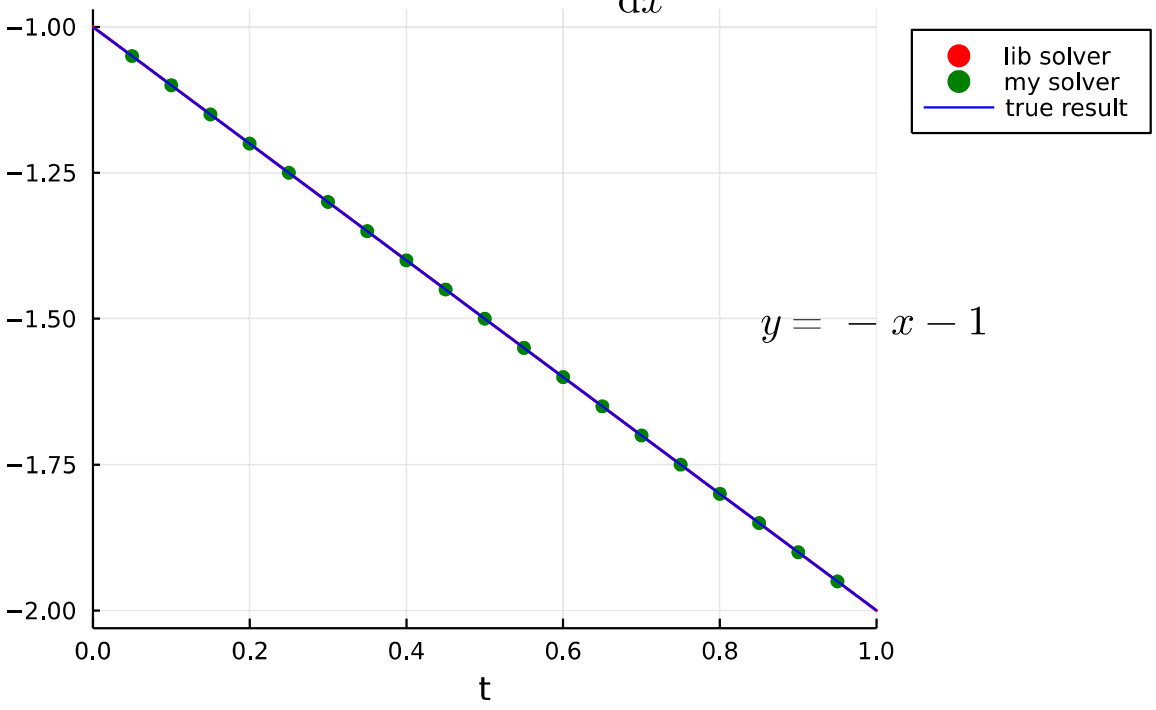
问题 1

In [5]:
```
iternums = [5, 10, 20]

f1(y, p, x) = x + y    # lib RK4() solver
xspan = (0.0, 1.0)
y0 = -1.0
f2(x, y) = x + y       # my rungekutta() solver
f3(x) = -x - 1         # true result
show_result(f1, f2, f3, xspan, y0, iternums, true, true, L"Problem\ 1.1: \frac{\math

f1(y, p, x) = -y^2
xspan = (0.0, 1.0)
y0 = 1.0
f2(x, y) = -y^2
f3(x) = 1 / (x + 1)
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 1.2: \frac{\mat
```

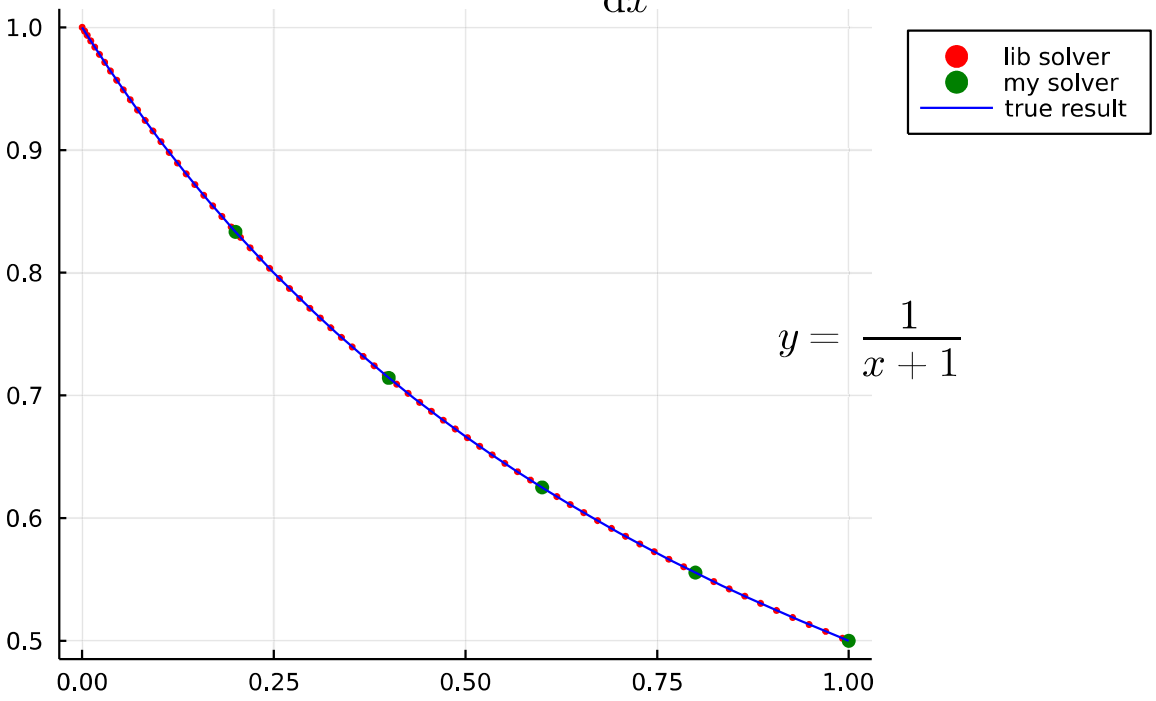

$Problem\ 1.1: \frac{dy}{dx} = x + y$
lib solver
my solver
true result
$y = -x - 1$
−1.00
−1.25
−1.50
−1.75
−2.00
0.0
0.2
0.4
0.6
0.8
1.0
t

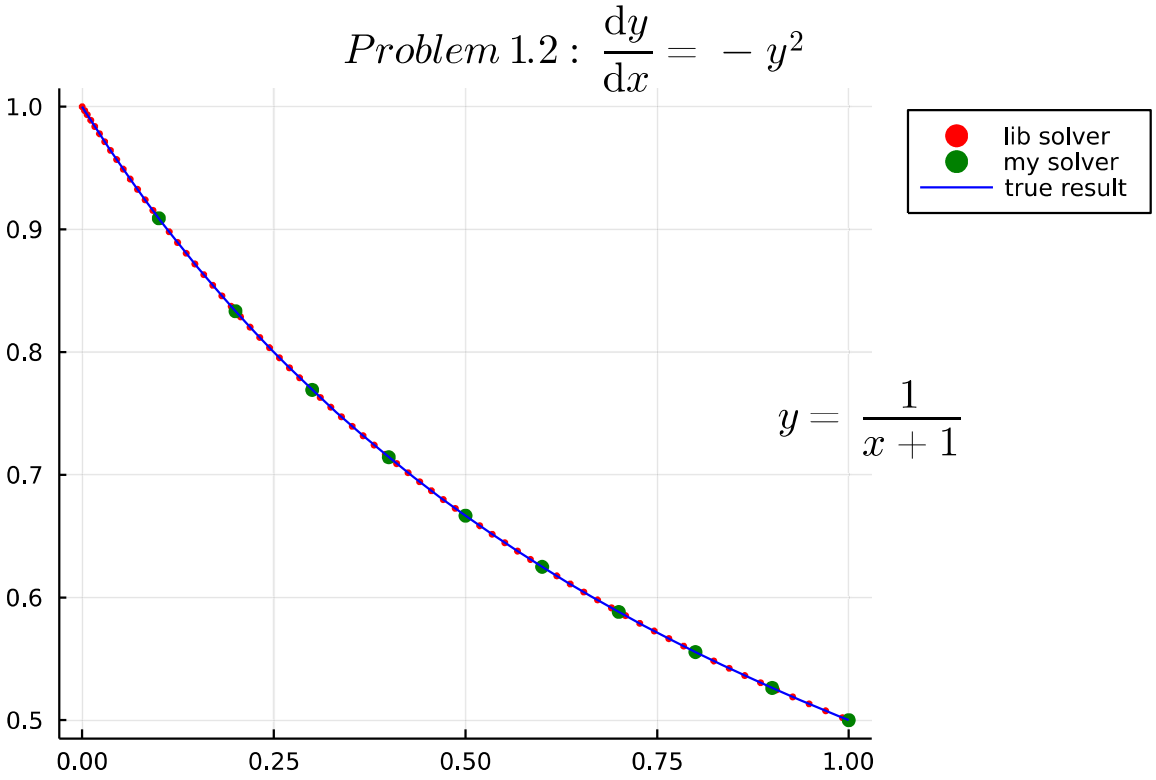

$Problem\ 1.1: \frac{dy}{dx} = x + y$
lib solver
my solver
true result
$y = -x - 1$
−1.00
−1.25
−1.50
−1.75
−2.00
0.0
0.2
0.4
0.6
0.8
1.0
t

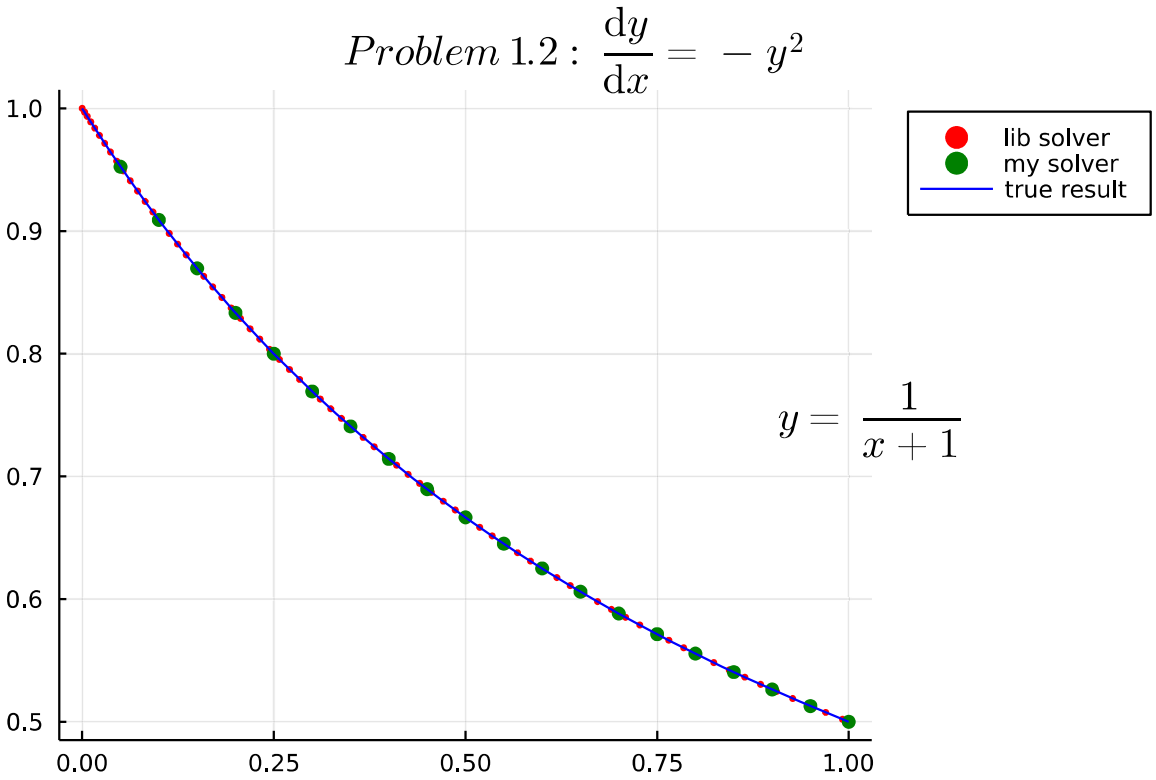
$$Problem\ 1.1:\ \frac{\mathrm{d}y}{\mathrm{d}x} = x + y$$



$$Problem\ 1.2:\ \frac{\mathrm{d}y}{\mathrm{d}x} = -y^2$$




$Problem\ 1.2: \frac{\mathrm{d}y}{\mathrm{d}x} = -y^2$
lib solver
my solver
true result
$y = \frac{1}{x+1}$
1.0
0.9
0.8
0.7
0.6
0.5
0.00
0.25
0.50
0.75
1.00


$Problem\ 1.2: \frac{\mathrm{d}y}{\mathrm{d}x} = -y^2$
lib solver
my solver
true result
$y = \frac{1}{x+1}$
1.0
0.9
0.8
0.7
0.6
0.5
0.00
0.25
0.50
0.75
1.00

$Problem\ 1.1: \frac{\mathrm{d} y}{\mathrm{d} x} = x + y$

Iternum: 5 Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [-1.2, -1.4, -1.5999999999999999, -1.7999999999999998, -1.9999999999999998]

Iternum: 10 Runge-Kutta:
x: [0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0.7999999999999999, 0.8999999999999999, 0.9999999999999999]
y: [-1.1, -1.2000000000000002, -1.3000000000000003, -1.4000000000000004, -1.5000000000000004, -1.6000000000000005, -1.7000000000000006, -1.8000000000000007, -1.9000000000000008, -2.000000000000001]

Iternum: 20 Runge-Kutta:
x: [0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.44999999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.7000000000000001, 0.7500000000000001, 0.8000000000000002, 0.8500000000000002, 0.9000000000000002, 0.9500000000000003, 1.0000000000000002]
y: [-1.05, -1.1, -1.1500000000000001, -1.2000000000000002, -1.2500000000000002, -1.3000000000000003, -1.3500000000000003, -1.4000000000000004, -1.4500000000000004, -1.5000000000000004, -1.5500000000000005, -1.6000000000000005, -1.6500000000000006, -1.7000000000000006, -1.7500000000000007, -1.8000000000000007, -1.8500000000000008, -1.9000000000000008, -1.9500000000000008, -2.000000000000001]

$Problem\ 1.2: \frac{\mathrm{d} y}{\mathrm{d} x} = -y^2$

Iternum: 5 Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [0.8333390356230387, 0.7142921304635431, 0.6250058936085341, 0.5555606879341864, 0.5000044061582258]

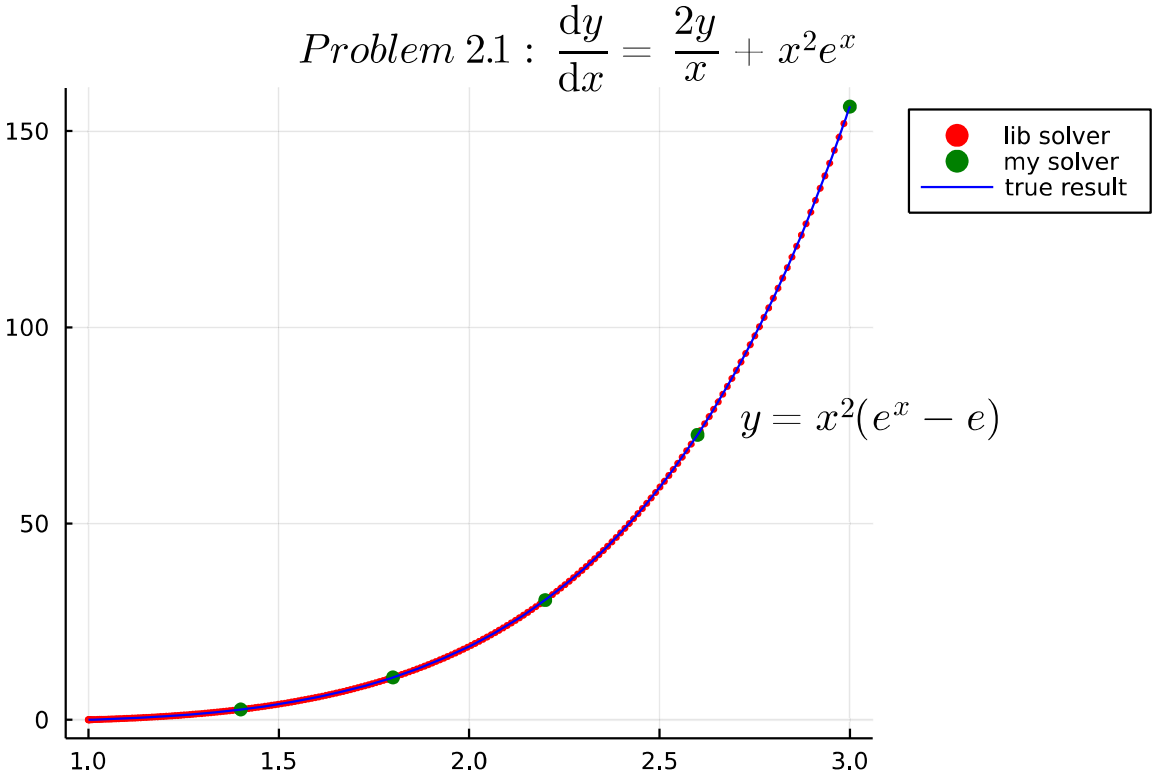
Iternum: 10 Runge-Kutta:
x: [0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0.7999999999999999, 0.8999999999999999, 0.9999999999999999]
y: [0.9090911863322196, 0.8333337288430721, 0.7692312057532864, 0.7142861538927614, 0.6666670910658625, 0.6250004009491739, 0.5882356685808391, 0.5555559031832142, 0.5263161112642581, 0.500000297580231]

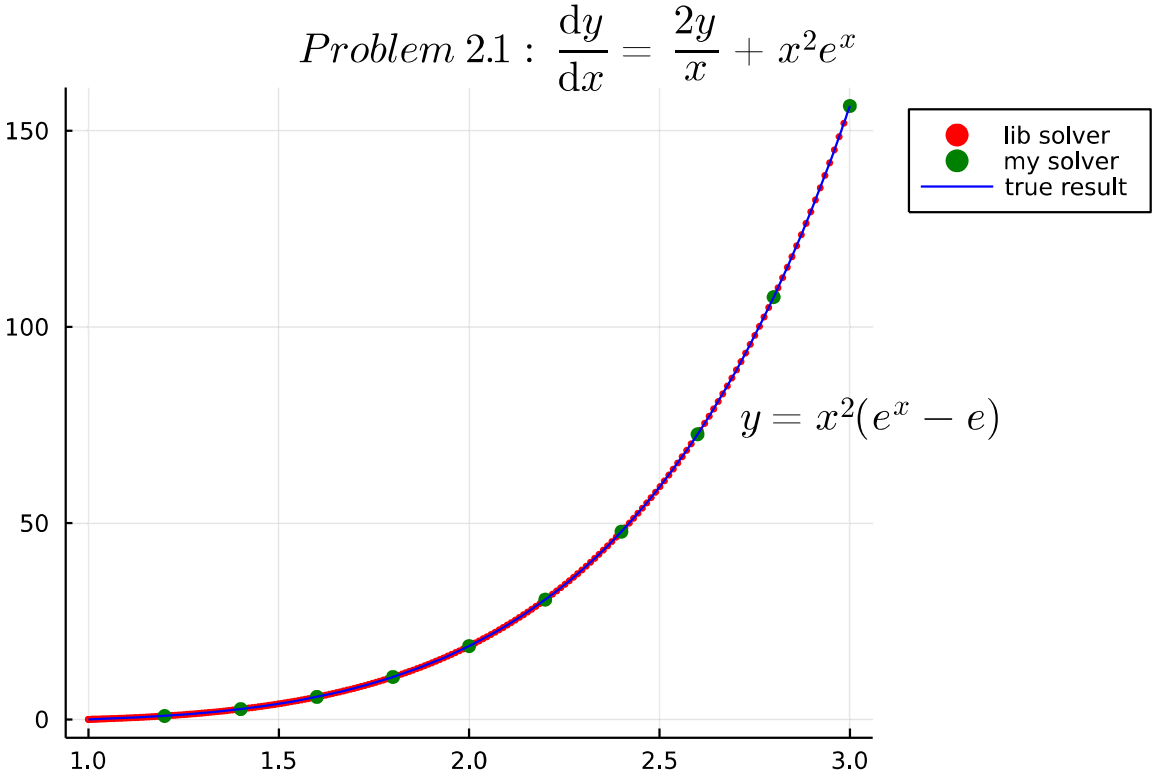
Iternum: 20 Runge-Kutta:
x: [0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.44999999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.7000000000000001, 0.7500000000000001, 0.8000000000000002, 0.8500000000000002, 0.9000000000000002, 0.9500000000000003, 1.0000000000000002]
y: [0.9523809630269818, 0.9090909268125394, 0.8695652397267474, 0.8333333585722669, 0.8000000269481106, 0.7692307970520927, 0.7407407688506038, 0.7142857422771115, 0.6896552000061802, 0.6666666936699812, 0.6451613166117316, 0.6250000254966355, 0.6060606307199913, 0.5882353179191637, 0.571428594368805, 0.5555555776432456, 0.5405405617928818, 0.5263158099136128, 0.5128205324747149, 0.5000000188974524]

问题 2

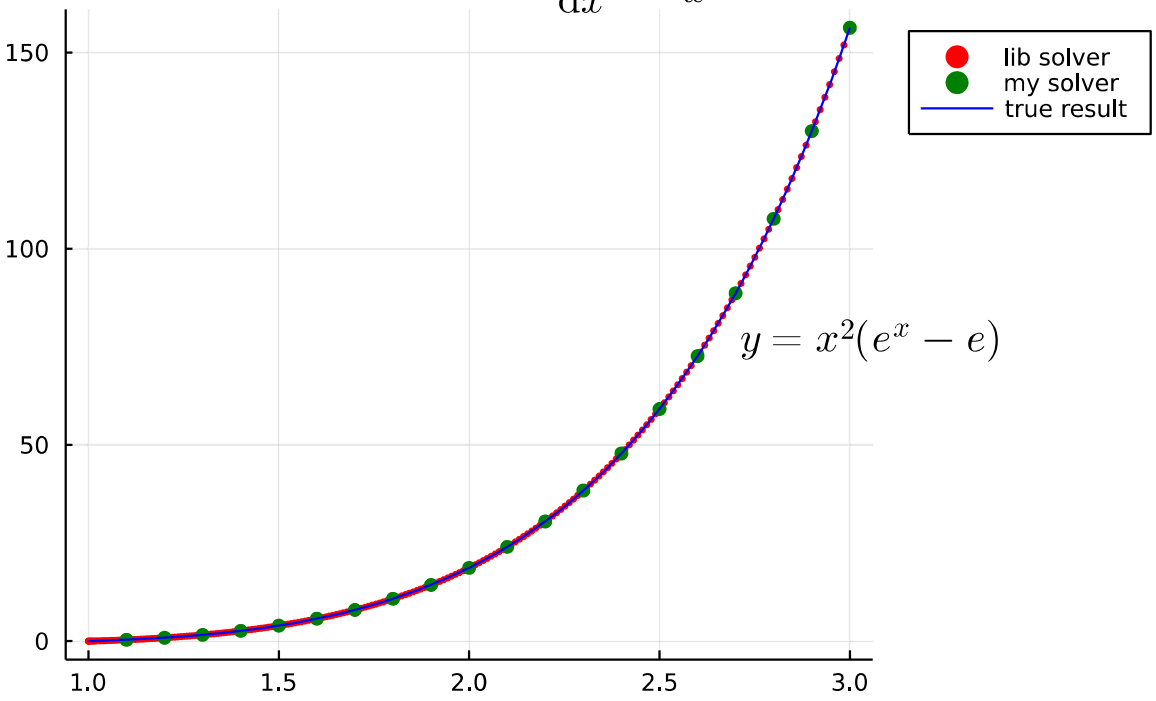
In [6]:
```
iternums = [5, 10, 20]

f1(y, p, x) = 2 * y / x + x^2 * exp(x)
xspan = (1.0, 3.0)
# !不需要换元，也不需要改xspan，当x最左边的值确定的时候，就是初值的位置
y0 = 0.0
f2(x, y) = 2 * y / x + x^2 * exp(x)
f3(x) = x^2 * (exp(x) - exp(1))
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 2.1:\frac{\math
```

```
f1(y, p, x) = (y^2 + y) / x
xspan = (1.0, 3.0)
y0 = -2.0
f2(x, y) = (y^2 + y) / x
f3(x) = 2x / (1 - 2x)
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 2.2:\frac{\math
```

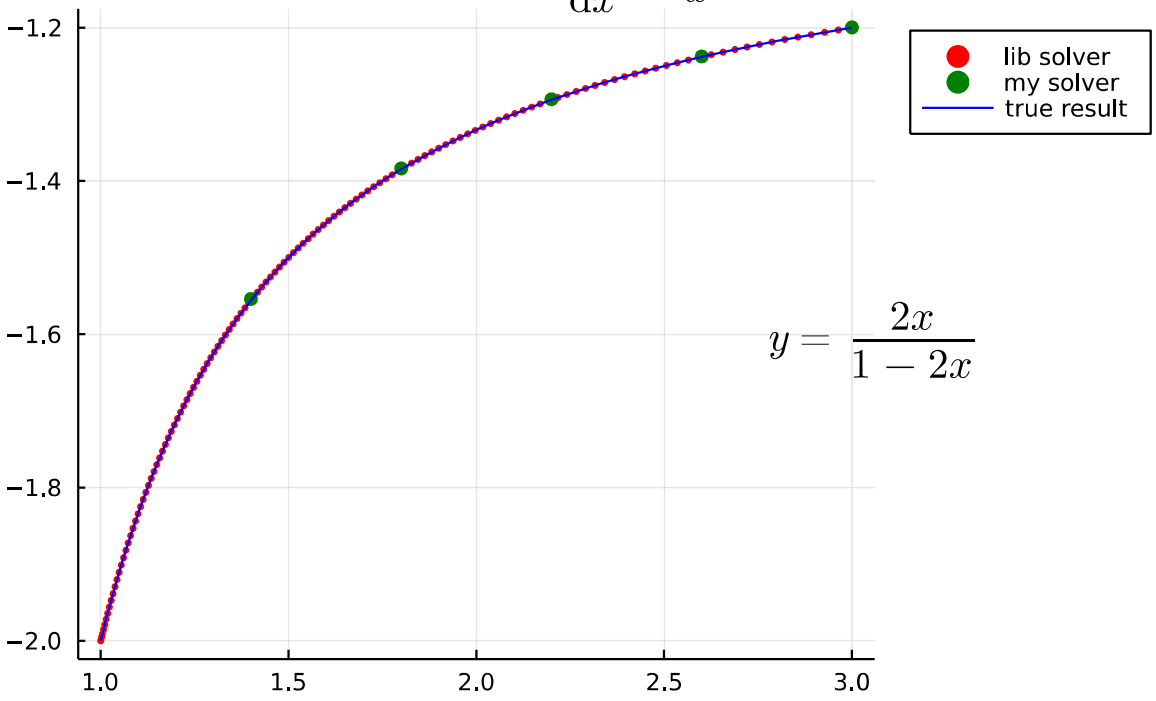


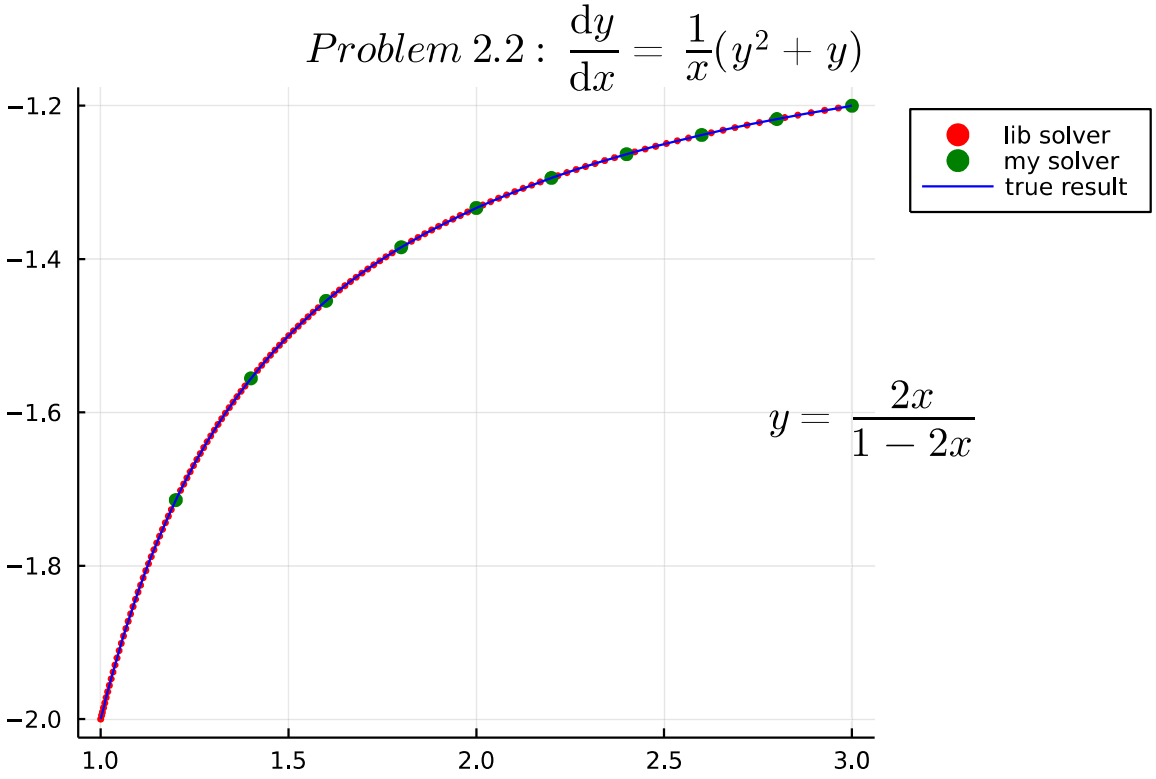


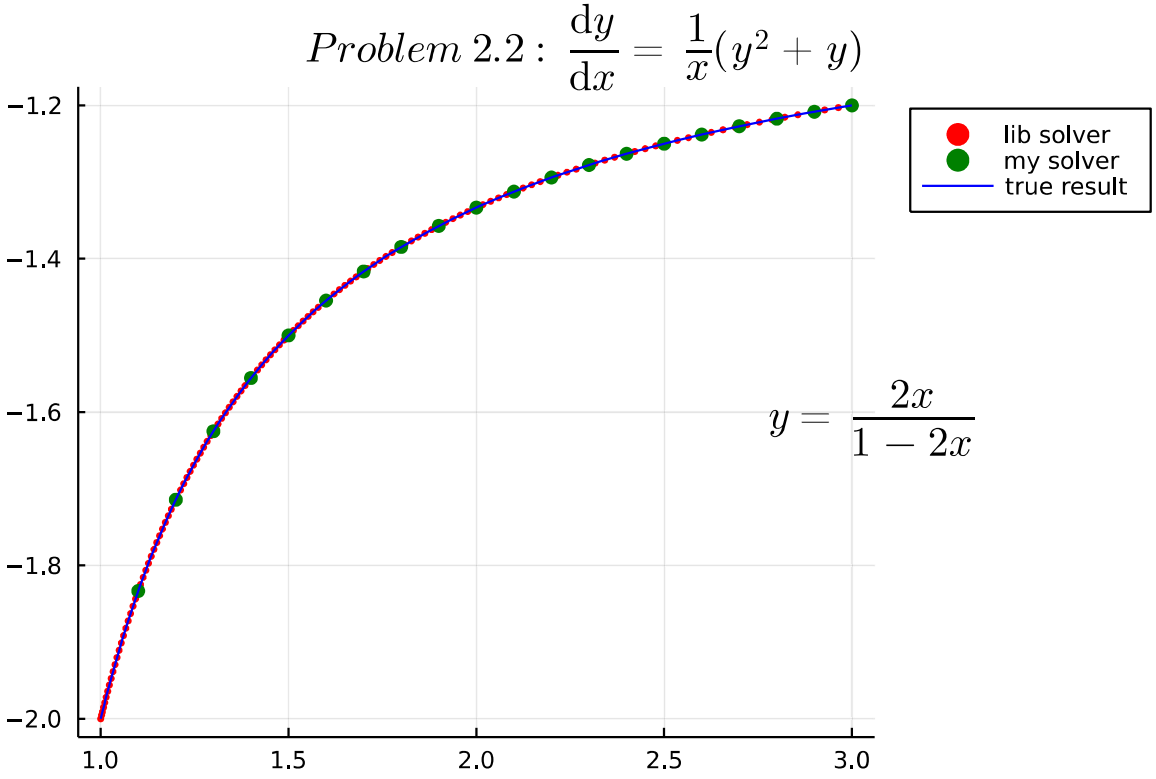
$$Problem\ 2.1: \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{2y}{x} + x^2e^x$$



$$Problem\ 2.2: \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{1}{x}(y^2 + y)$$




$Problem\ 2.2: \frac{dy}{dx} = \frac{1}{x}(y^2 + y)$
lib solver
my solver
true result
$y = \frac{2x}{1-2x}$
−1.2
−1.4
−1.6
−1.8
−2.0
1.0
1.5
2.0
2.5
3.0


$Problem\ 2.2: \frac{dy}{dx} = \frac{1}{x}(y^2 + y)$
lib solver
my solver
true result
$y = \frac{2x}{1-2x}$
−1.2
−1.4
−1.6
−1.8
−2.0
1.0
1.5
2.0
2.5
3.0

$Problem\ 2.1:\frac{\mathrm{d}\ y}{\mathrm{d}\ x}=\frac{2y}{x}+x^2\ e^x$

```
Iternum: 5      Runge-Kutta:
x: [1.4, 1.7999999999999998, 2.1999999999999997, 2.5999999999999996, 2.9999999999999996]
y: [2.613942792503426, 10.776313166418575, 30.491654203794223, 72.58559860601221, 156.22519827584796]

Iternum: 10     Runge-Kutta:
x: [1.2, 1.4, 1.5999999999999999, 1.7999999999999998, 1.9999999999999998, 2.1999999999999997, 2.4, 2.6, 2.8000000000000003, 3.0000000000000004]
y: [0.8663791119740196, 2.619740520468712, 5.719895279538559, 10.79201759748925, 18.6808523645173, 30.521598135366503, 47.83236583269365, 72.634503537672, 107.60885199118545, 156.2982574428725]

Iternum: 20     Runge-Kutta:
x: [1.1, 1.2000000000000002, 1.3000000000000003, 1.4000000000000004, 1.5000000000000004, 1.6000000000000005, 1.7000000000000006, 1.8000000000000007, 1.9000000000000008, 2.000000000000001, 2.100000000000001, 2.200000000000001, 2.300000000000001, 2.4000000000000012, 2.5000000000000013, 2.6000000000000014, 2.7000000000000015, 2.8000000000000016, 2.9000000000000017, 3.0000000000000018]
y: [0.3459102873064402, 0.8666216927288839, 1.6071813476640324, 2.620311305871806, 3.9676018979880405, 5.720879324244457, 7.963771792604615, 10.793501783648523, 14.32293572758867, 18.6829265676522, 24.02498941966458, 30.524355889829184, 38.38345866002602, 47.83590478093721, 59.15100382752139, 72.6389257808317, 88.65657333091889, 107.61426438930823, 129.9833331156541, 156.30477188083754]
```

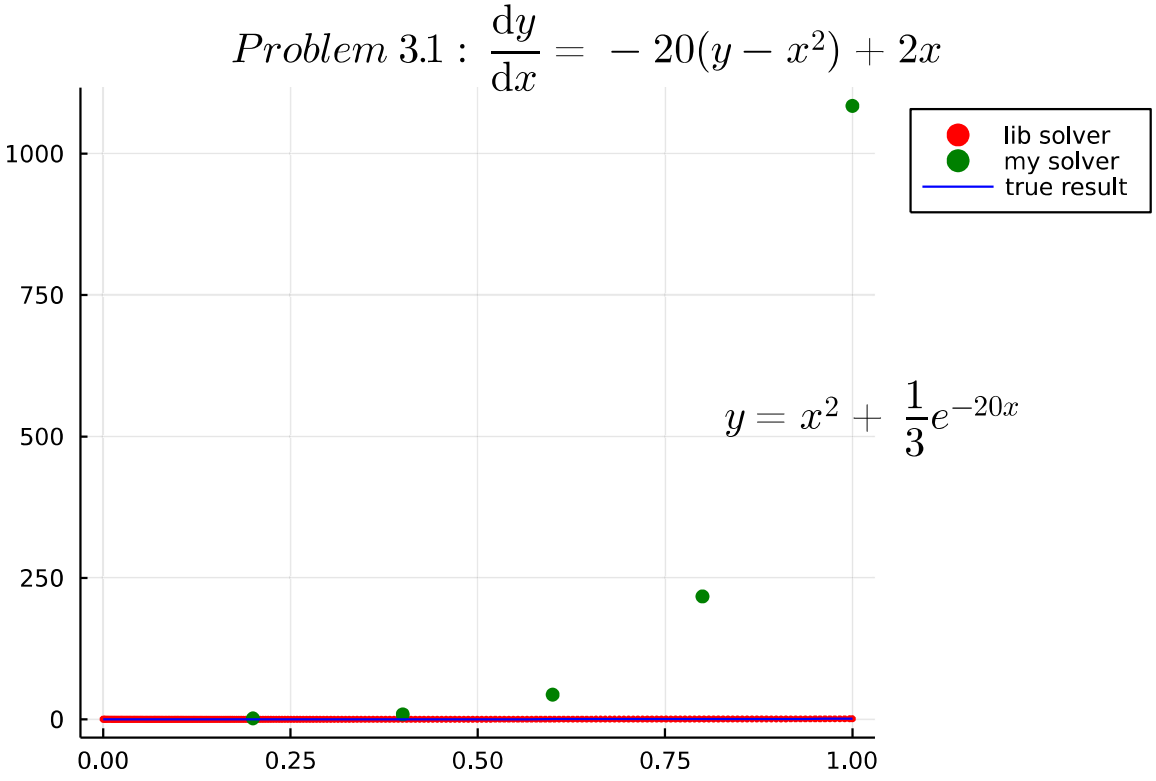
$Problem\ 2.2:\frac{\mathrm{d}\ y}{\mathrm{d}\ x}=\frac{1}{x}(y^2+y)$

```
Iternum: 5      Runge-Kutta:
x: [1.4, 1.7999999999999998, 2.1999999999999997, 2.5999999999999996, 2.9999999999999996]
y: [-1.5539889980952382, -1.3836172899114931, -1.2934015269193302, -1.237540157935232, -1.1995479584579267]

Iternum: 10     Runge-Kutta:
x: [1.2, 1.4, 1.5999999999999999, 1.7999999999999998, 1.9999999999999998, 2.1999999999999997, 2.4, 2.6, 2.8000000000000003, 3.0000000000000004]
y: [-1.714245180451154, -1.5555228848496194, -1.4545197492007562, -1.384594506286678, -1.3331585607527364, -1.2941026605729438, -1.263144798904635, -1.2380836211681467, -1.2173808733204385, -1.1999905397087856]

Iternum: 20     Runge-Kutta:
x: [1.1, 1.2000000000000002, 1.3000000000000003, 1.4000000000000004, 1.5000000000000004, 1.6000000000000005, 1.7000000000000006, 1.8000000000000007, 1.9000000000000008, 2.000000000000001, 2.100000000000001, 2.200000000000001, 2.300000000000001, 2.4000000000000012, 2.5000000000000013, 2.6000000000000014, 2.7000000000000015, 2.8000000000000016, 2.9000000000000017, 3.0000000000000018]
y: [-1.8333328294259301, -1.7142851698413297, -1.6249995001712725, -1.5555551110526054, -1.4999996057103289, -1.454545102841952, -1.416666350536796, -1.3846150982409504, -1.3571425958336794, -1.333333093327103, -1.3124997782659393, -1.294117441136210, -1.2777775856503875, -1.2631577147371074, -1.24999983073609, -1.2380950783953766, -1.22727257614238, -1.2173911609365085, -1.2083331969086475, -1.1999998699271444]
```

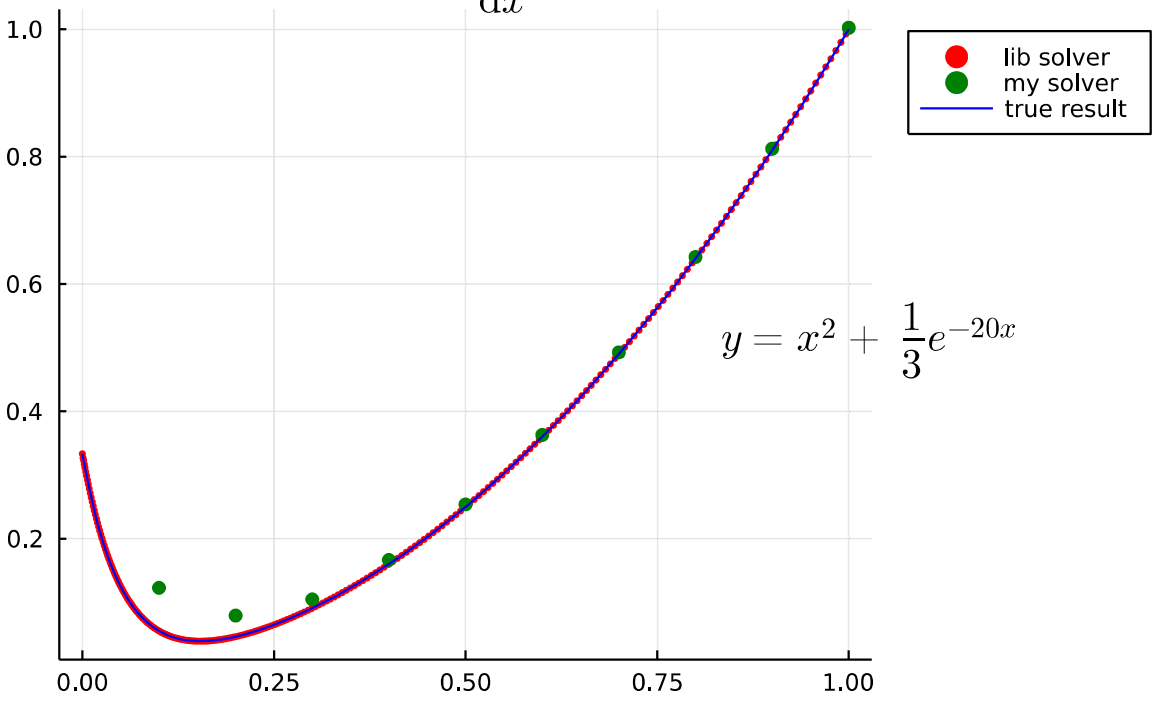
问题 3

In [7]:
```
f1(y, p, x) = -20(y - x^2) + 2x
xspan = (0.0, 1.0)
y0 = 1 / 3
f2(x, y) = -20(y - x^2) + 2x
f3(x) = x^2 + 1 / 3 * exp(-20x)
```

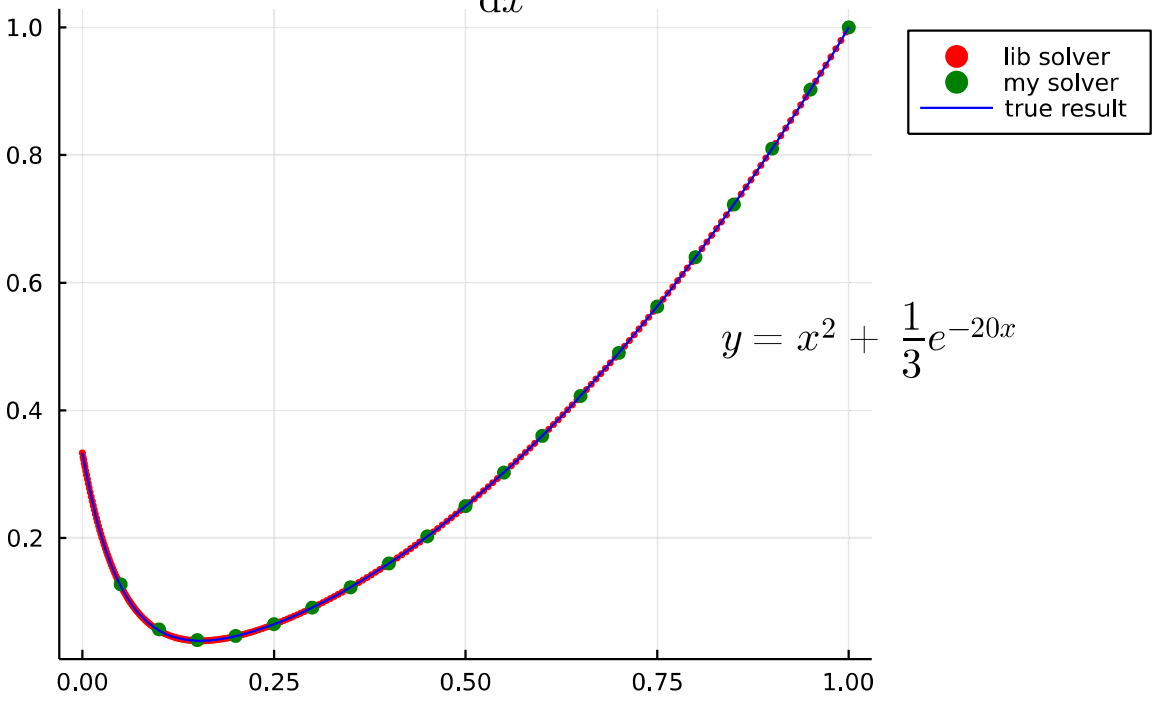
```
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 3.1: \frac{\mat


f1(y, p, x) = -20y + 20sin(x) + cos(x)
xspan = (0.0, 1.0)
y0 = 1.0
f2(x, y) = -20y + 20sin(x) + cos(x)
f3(x) = exp(-20x) + sin(x)
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 3.2: \frac{\mat


f1(y, p, x) = -20(y - exp(x)sin(x)) + exp(x) * (sin(x) + cos(x))
xspan = (0.0, 1.0)
y0 = 0.0
f2(x, y) = -20(y - exp(x)sin(x)) + exp(x) * (sin(x) + cos(x))
f3(x) = exp(x) * sin(x)
show_result(f1, f2, f3, xspan, y0, iternums, true, false, L"Problem\ 3.3: \frac{\mat
```



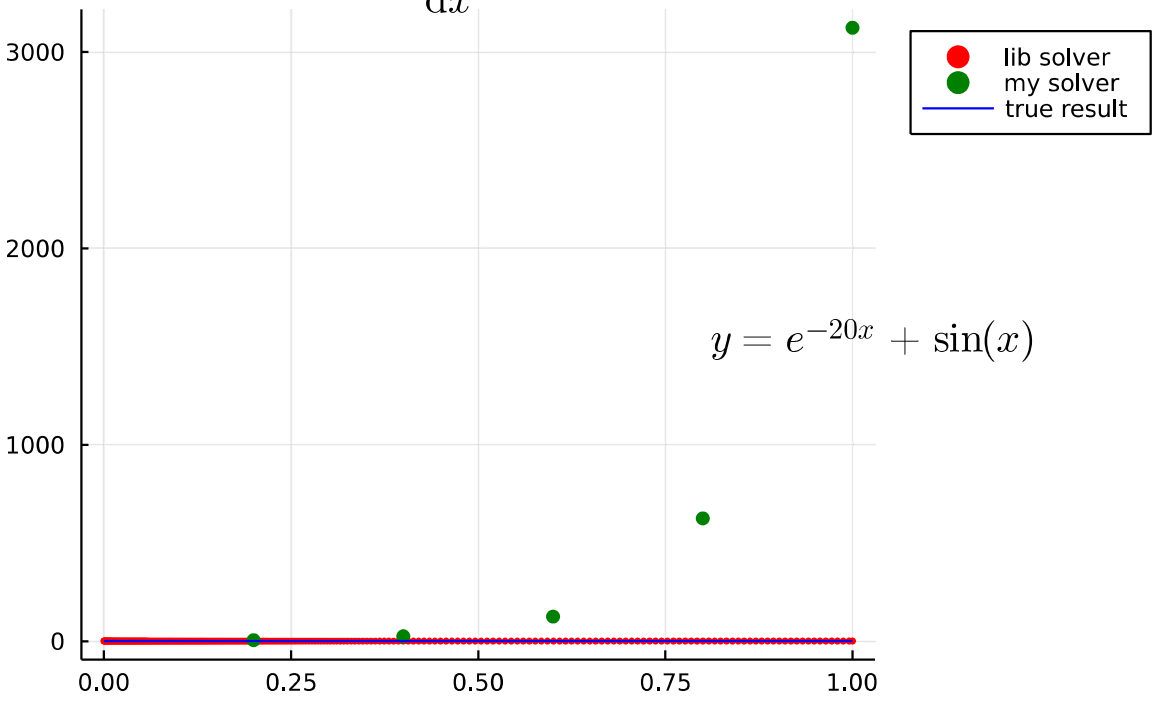
$$Problem\ 3.1: \frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - x^2) + 2x$$



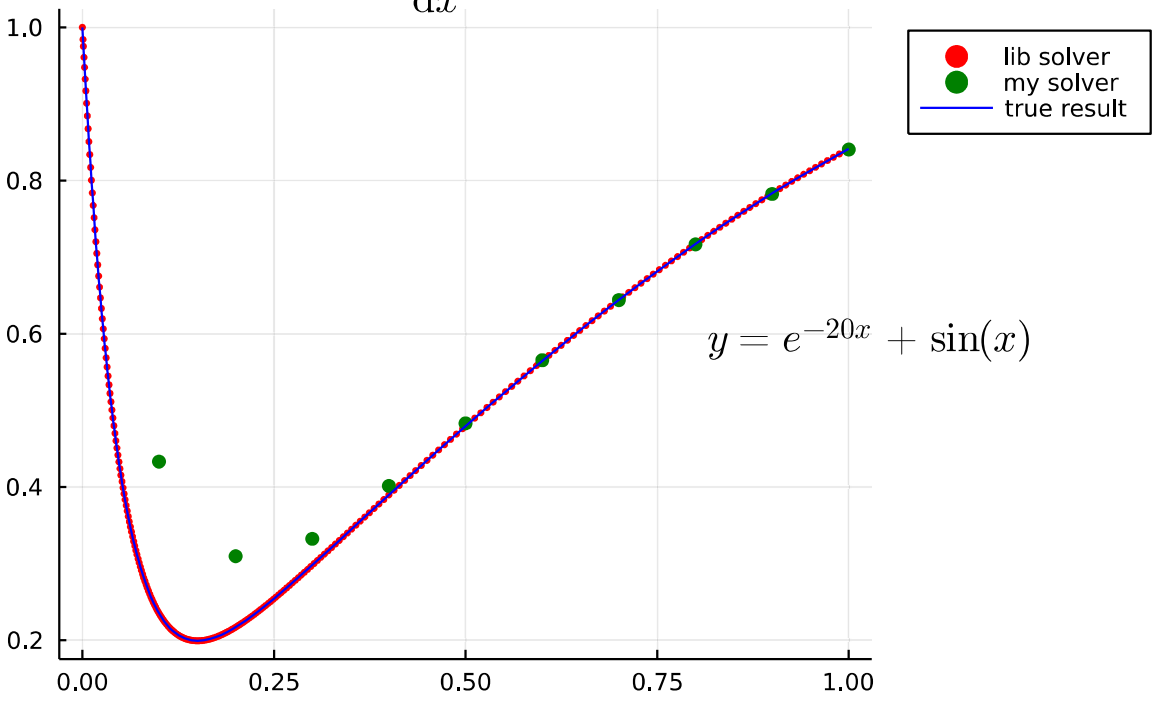
$$Problem\ 3.1: \frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - x^2) + 2x$$



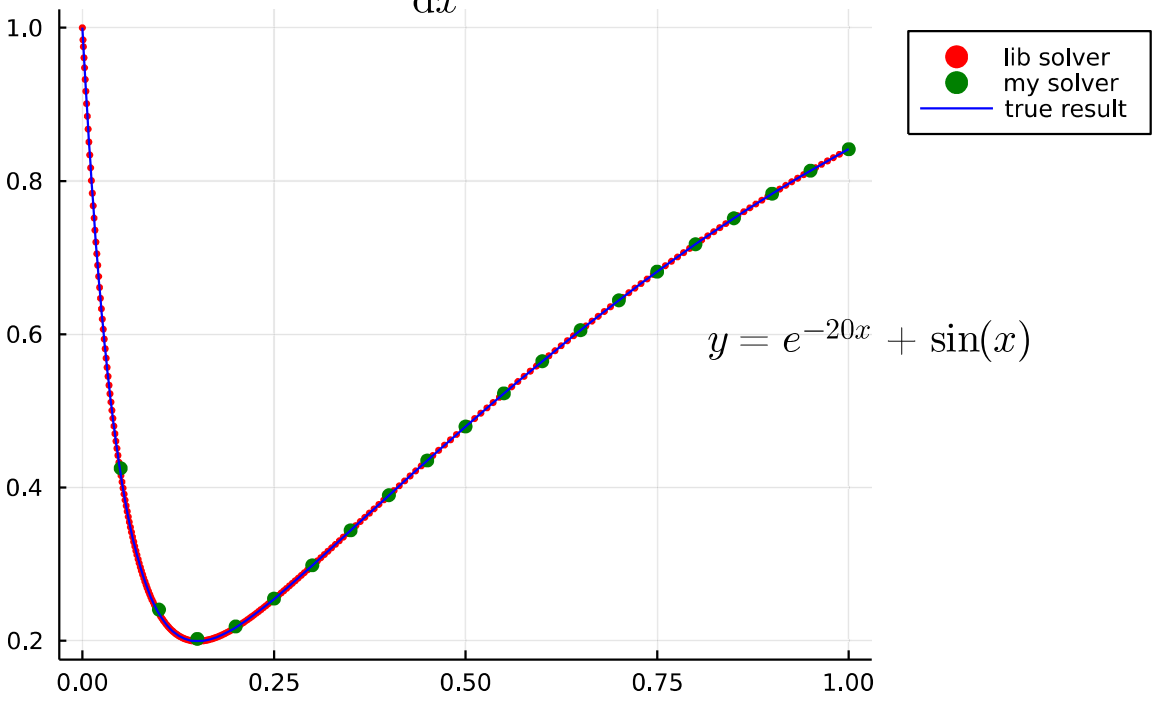
$$Problem\ 3.2: \frac{\mathrm{d}y}{\mathrm{d}x} = -20y + 20\sin(x) + \cos(x)$$



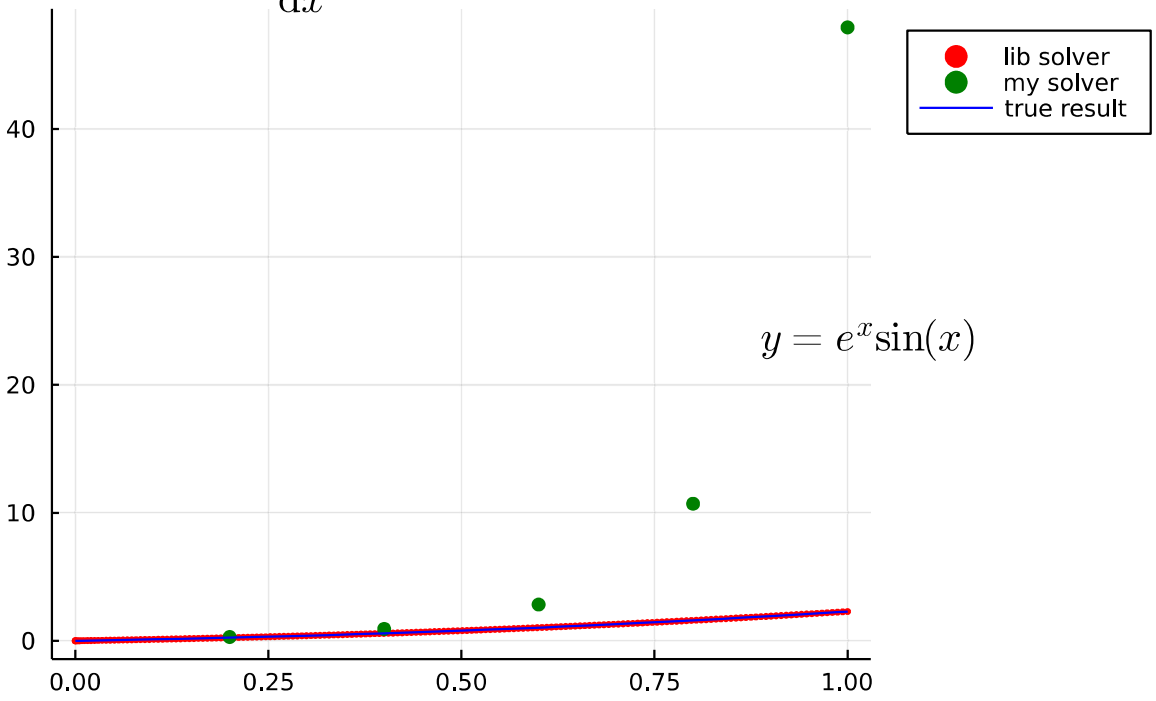
$$Problem\ 3.2: \frac{\mathrm{d}y}{\mathrm{d}x} = -20y + 20\sin(x) + \cos(x)$$



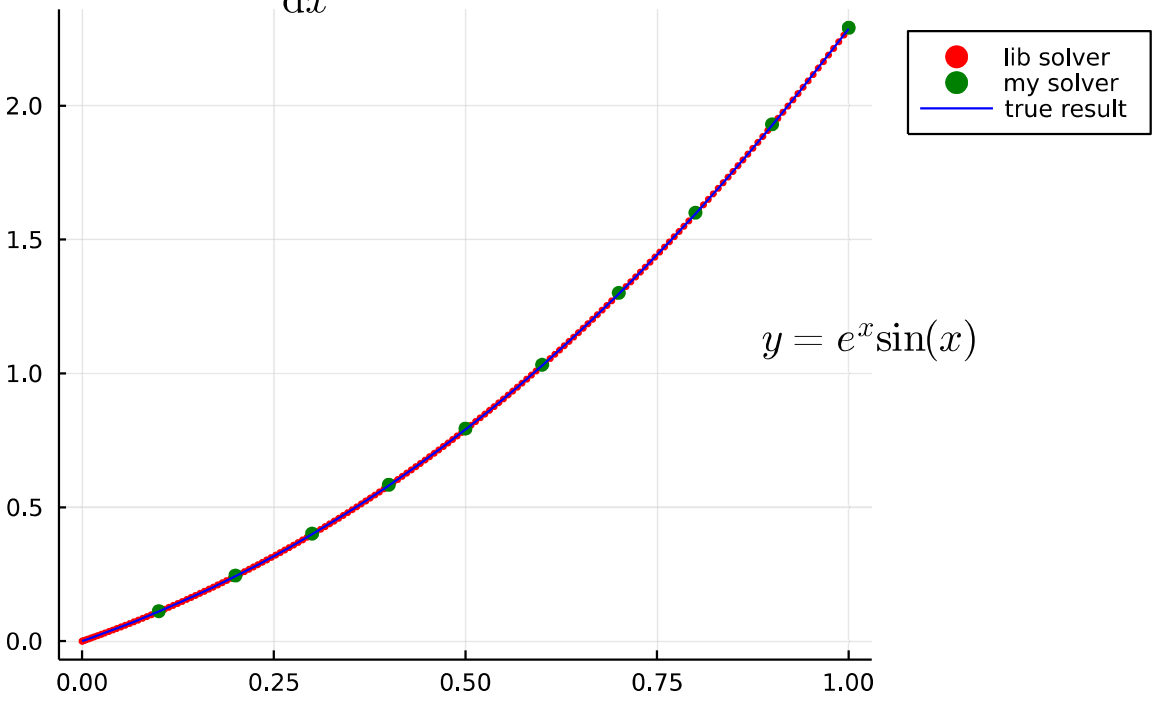
$$Problem\ 3.2: \frac{\mathrm{d}y}{\mathrm{d}x} = -20y + 20\sin(x) + \cos(x)$$



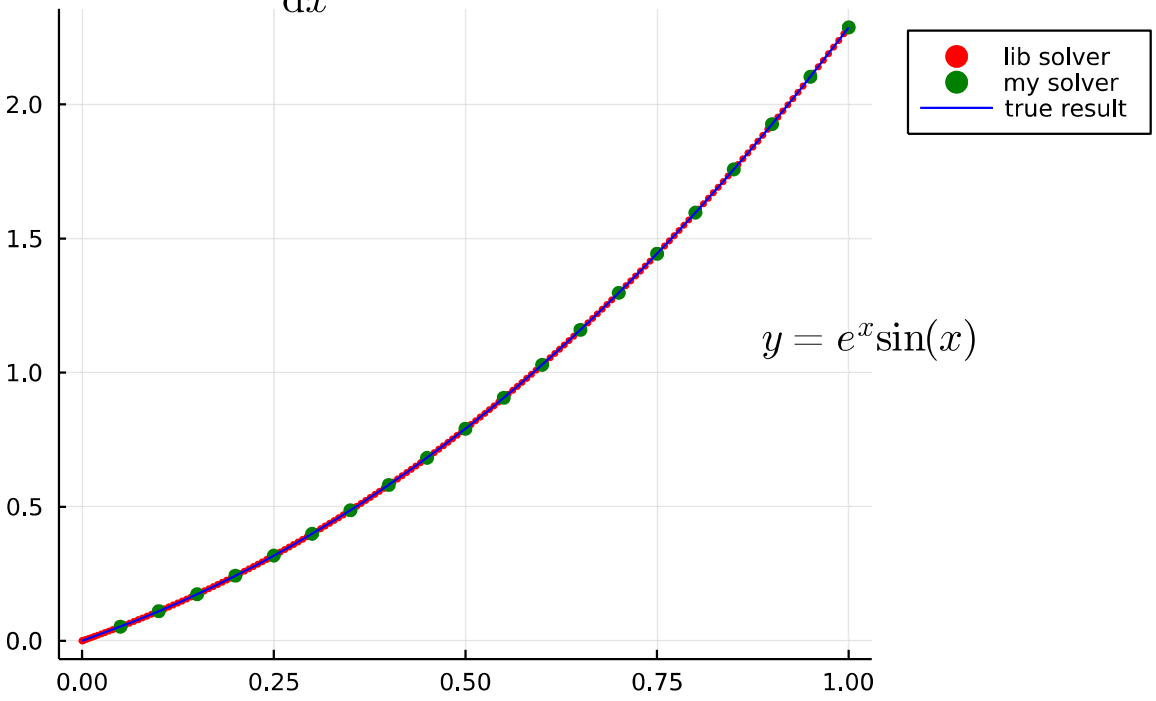
$$Problem\ 3.3: \frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - e^x\sin(x)) + e^x(\sin(x) + \cos(x))$$



$$Problem\ 3.3: \frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - e^x\sin(x)) + e^x(\sin(x) + \cos(x))$$



$$Problem\ 3.3: \frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - e^x\sin(x)) + e^x(\sin(x) + \cos(x))$$



$Problem\ 3.1: \frac{\mathrm{d} y}{\mathrm{d} x}=-20(y-x^2)+2x$

Iternum: 5 Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [1.7600000000000002, 8.813333333333336, 43.68000000000001, 217.29333333333338, 1084.3200000000002]

Iternum: 10 Runge-Kutta:
x: [0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0.7999999999999999, 0.8999999999999999, 0.9999999999999999]
y: [0.12277777777777779, 0.07925925925925928, 0.10475308641975309, 0.16658436213991773, 0.25386145404663923, 0.36295381801554644, 0.4926512726718488, 0.6425504242239496, 0.8125168080746498, 1.00250560269155]

Iternum: 20 Runge-Kutta:
x: [0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.4499999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.7000000000000001, 0.7500000000000001, 0.8000000000000002, 0.8500000000000002, 0.9000000000000002, 0.9500000000000003, 1.0000000000000002]
y: [0.12755208333333334, 0.056946614583333346, 0.04015706380208334, 0.04667348225911459, 0.0650546391805013, 0.09101007302602132, 0.12293086071809133, 0.16021365610261756, 0.2026322043718149, 0.2501016599727639, 0.30259020582311974, 0.3600859105170032, 0.42258429977720957, 0.49008369574978694, 0.5625834692395035, 0.6400833842981473, 0.7225833524451387, 0.8100833405002607, 0.9025833360209314, 1.000083334341183]

$Problem\ 3.2: \frac{\mathrm{d} y}{\mathrm{d} x}=-20y+20\sin(x)+\cos(x)$

Iternum: 5 Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [5.197338106220029, 25.376170704380762, 125.48681526112966, 625.3120955171343, 3123.7951509471586]

Iternum: 10 Runge-Kutta:
x: [0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0.7999999999999999, 0.8999999999999999, 0.9999999999999999]
y: [0.43313899649719434, 0.309660468004797, 0.33232466670513466, 0.4014139712639834, 0.4830743414705462, 0.565435279659871, 0.6439890044827506, 0.7167223470605888, 0.7824991512012693, 0.840525720595564]

Iternum: 20 Runge-Kutta:
x: [0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.4499999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.7000000000000001, 0.7500000000000001, 0.8000000000000002, 0.8500000000000002, 0.9000000000000002, 0.9500000000000003, 1.0000000000000002]
y: [0.4249785186019 45, 0.24045622213059903, 0.20216843904311116, 0.2184386634123050 6, 0.25481165110875065, 0.2982910222151891, 0.3439285510509592, 0.38979533635034436, 0.4350961733328589, 0.4794626228591924, 0.5226880879214766, 0.5646286383723231, 0.6051659863042762, 0.6441937625688693, 0.6816125254669342, 0.7173280378596233, 0.7512507634216402, 0.783295813201471, 0.8133830538368221, 0.8414372688602679]

$Problem\ 3.3: \frac{\mathrm{d} y}{\mathrm{d} x}=-20(y-e^x \sin(x))+e^x (\sin(x) + \cos(x))$

Iternum: 5 Runge-Kutta:
x: [0.2, 0.4, 0.6000000000000001, 0.8, 1.0]
y: [0.29864621275013403, 0.9272198700273476, 2.835477338896381, 10.710885330937327, 47.941446381632616]

Iternum: 10 Runge-Kutta:
x: [0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0.7999999999999999, 0.8999999999999999, 0.9999999999999999]

```
y: [0.11205510913037421, 0.2451165144244346, 0.4017780966782987, 0.5840969565792278,
0.793822052967138, 1.0324183053426443, 1.3010149883505369, 1.6003210120174918, 1.930
5210337840668, 2.291156923060078]

Iternum: 20     Runge-Kutta:
x: [0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.449
99999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.70000000000000
01, 0.7500000000000001, 0.8000000000000002, 0.8500000000000002, 0.9000000000000002,
0.9500000000000003, 1.0000000000000002]
y: [0.05259503995574239, 0.11040898628183947, 0.17370939051652695, 0.242749000926038
02, 0.31777169155820517, 0.3990135524673282, 0.48670206962488405, 0.581054489098230
5, 0.6822757724973431, 0.7905562930220894, 0.9060693250287787, 1.0289683441295572,
1.1593841416649548, 1.2974217527850616, 1.4431571960299754, 1.5966340222275905, 1.75
78596709840264, 1.9268016337536287, 2.1033834233412323, 2.2874803506747066]
```

思考题

1. 对实验 1，数值解和解析解相同吗？为什么？试加以说明。

   是相同的，因为解是线性函数，能够通过所得数值解的两个点确定直线的方程，即等价于得到了解析解

2. 对实验 2，N 越大越精确吗？试加以说明。

   从实验的结果来看，并不是，因为当n=5的时候已经获得足够精确的数值解了，再增大n的值只是增加了计算量，却不能再明显提高结果的精度，得不偿失

3. 对实验 3，N 较小会出现什么现象？试加以说明

   当n较小的时候所得数值解和正确结果相差较大，结果失真，说明在一定条件下确实需要更大的n来更好的获得数值解。具体的条件取决于待求解微分方程性质。

实验题目4 牛顿(Newton)迭代法

代码实现

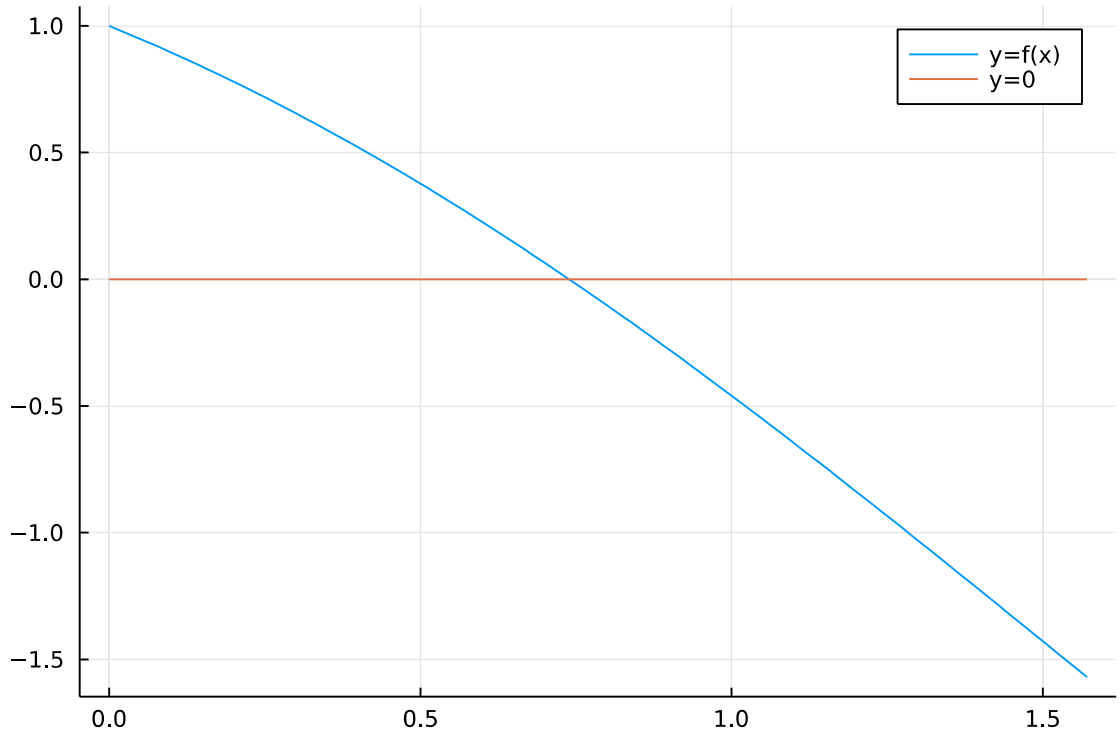
In [79]:
```
using Printf
using Plots
using NLsolve
```

In [80]:
```
# newton method
function newton(f::Function, df::Function, ϵ1, ϵ2, N, x0)
    n = 1
    while n <= N
        F = f(x0)
        DF = df(x0)
        if abs(F) < ϵ1
            @printf("iter:%3d\troot:%18.12f\n", n - 1, x0)
            return x0
        end
        if abs(DF) < ϵ2
            @printf("Reach a critical point!\n")
            return
        end
        x1 = x0 - F / DF
        Tol = abs(x1 - x0)
        if Tol < ϵ1
            @printf("iter:%3d\troot:%18.12f\n", n - 1, x1)
            return x1
        end
        n = n + 1
        x0 = x1
    end
    @printf("Fail to converge within %d iterations!\n", N)
end
# multi-root newton method
function newton(f::Function, df::Function, ϵ1, ϵ2, N, x0, λ)
    n = 1
    while n <= N
        F = f(x0)
        DF = df(x0)
        if abs(F) < ϵ1
            @printf("iter:%3d\troot:%18.12f\n", n - 1, x0)
            return x0
        end
        if abs(DF) < ϵ2
            @printf("Reach a critical point!\n")
            return
        end
        x1 = x0 - λ * F / DF
        Tol = abs(x1 - x0)
        if Tol < ϵ1
            @printf("iter:%3d\troot:%18.12f\n", n - 1, x1)
            return x1
        end
        n = n + 1
        x0 = x1
    end
    @printf("Fail to converge within %d iterations!\n", N)
end
```

```
newton (generic function with 2 methods)
```

问题 1

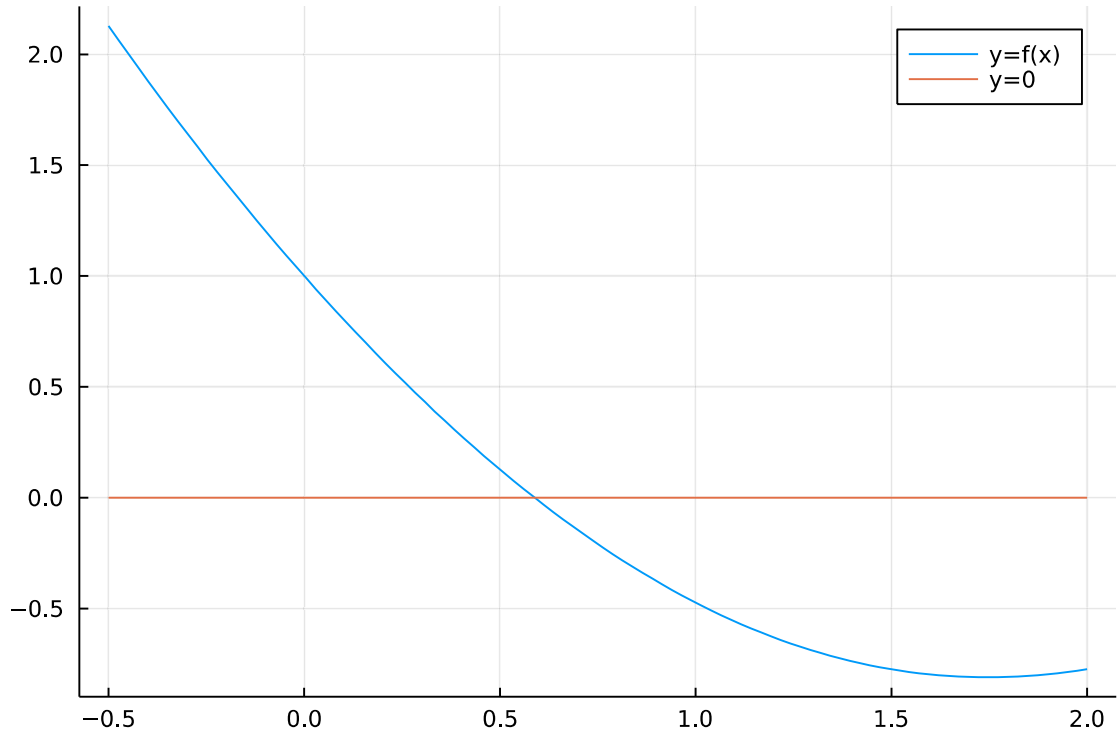
In [81]:
```
f(x) = cos(x) - x
df(x) = -sin(x) - 1
x = range(start=0, stop=pi / 2, length=100)
c(x) = 0
y = [f.(x), c.(x)]
label = ["y=f(x)" "y=0"]
display(plot(x, y, label=label))
# https://discourse.julialang.org/t/basic-usage-of-nlsolve-for-scalar-problems/28489
# https://github.com/JuliaNLSolvers/NLsolve.jl
function f!(r, x)
    r .= f.(x)
end
function j!(J, x)
    (s1, s2) = size(J)
    J .= zeros(s1, s1)
    for i in 1:s1
        J[i, i] = df(x[i])
    end
end
ϵ1 = 1e-6
ϵ2 = 1e-4
N = 10
x0 = 0.785398163  # pi/4
@time sol = nlsolve(f!, j!, [x0]; method=:newton)
# println(sol)
r1 = sol.zero
@time r2 = newton(f, df, ϵ1, ϵ2, N, x0)
println("library root solver:\t$r1")
println("single root solver:\t$([r2])")
```



```
  0.090331 seconds (134.57 k allocations: 6.886 MiB, 31.24% gc time, 99.89% compilat
ion time)
iter:  2        root:    0.739085178106
  0.013239 seconds (8.03 k allocations: 436.511 KiB, 99.04% compilation time)
library root solver:    [0.7390851332151611]
single root solver:     [0.7390851781060086]
```

In [82]:

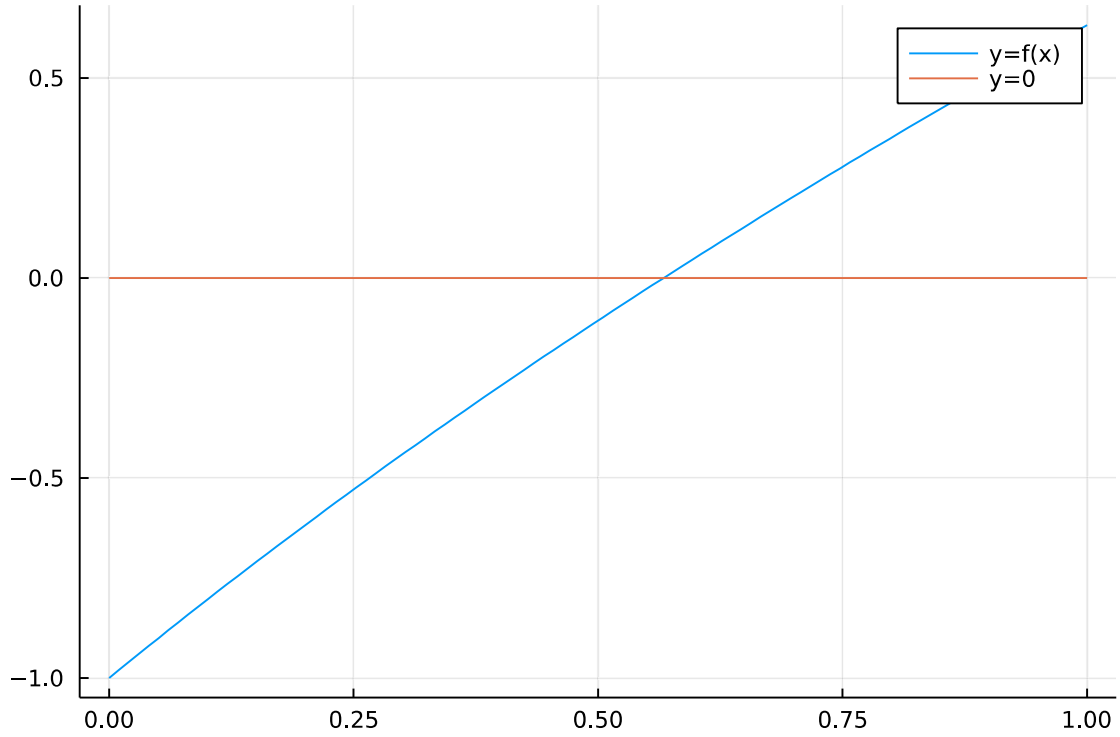
```
f(x) = exp(-x) - sin(x)
df(x) = -exp(-x) - cos(x)
x = range(start=-1 / 2, stop=2, length=100)
c(x) = 0
y = [f.(x), c.(x)]
label = ["y=f(x)" "y=0"]
display(plot(x, y, label=label))
function f!(r, x)
    r .= f.(x)
end
function j!(J, x)
    (s1, s2) = size(J)
    J .= zeros(s1, s1)
    for i in 1:s1
        J[i, i] = df(x[i])
    end
end
ϵ1 = 1e-6
ϵ2 = 1e-4
N = 10
x0 = 0.6
@time sol = nlsolve(f!, j!, [x0]; method=:newton)
# println(sol)
r1 = sol.zero
@time r2 = newton(f, df, ϵ1, ϵ2, N, x0)
println("library root solver:\t$r1")
println("single root solver:\t$([r2])")
```




```
  0.059891 seconds (129.01 k allocations: 6.544 MiB, 99.86% compilation time)
iter:  2        root:    0.588532742848
  0.012462 seconds (8.18 k allocations: 440.729 KiB, 99.11% compilation time)
library root solver:    [0.588532742847979]
single root solver:     [0.588532742847979]
```

问题 2

In [83]:
```
f(x) = x - exp(-x)
df(x) = 1 + exp(-x)

x = range(start=0, stop=1, length=100)
c(x) = 0
y = [f.(x), c.(x)]
label = ["y=f(x)" "y=0"]
display(plot(x, y, label=label))
function f!(r, x)
    r .= f.(x)
end
function j!(J, x)
    (s1, s2) = size(J)
    J .= zeros(s1, s1)
    for i in 1:s1
        J[i, i] = df(x[i])
    end
end

ϵ1 = 1e-6
ϵ2 = 1e-4
N = 10
x0 = 0.5

@time sol = nlsolve(f!, j!, [x0]; method=:newton)
# println(sol)
r1 = sol.zero
@time r2 = newton(f, df, ϵ1, ϵ2, N, x0)
println("library root solver:\t$r1")
println("single root solver:\t$([r2])")
```




```
  0.059150 seconds (134.73 k allocations: 6.888 MiB, 99.88% compilation time)
iter:  2        root:    0.567143165035
  0.011558 seconds (8.06 k allocations: 436.974 KiB, 99.16% compilation time)
library root solver:    [0.5671432904097811]
single root solver:     [0.5671431650348622]
```

In [84]:
```
f(x) = x^2 - 2x * exp(-x) + exp(-2x)
# df(x) = 2x - (2exp(-x)-2x*exp(-x)) -2exp(-2x)
df(x) = 2(x - exp(-x)) * (1 + exp(-x))
x = range(start=0, stop=1, length=100)
c(x) = 0
y = [f.(x), c.(x)]
label = ["y=f(x)" "y=0"]
plot(x, y, label=label)
function f!(r, x)
    r .= f.(x)
end
function j!(J, x)
    (s1, s2) = size(J)
    J .= zeros(s1, s1)
    for i in 1:s1
        J[i, i] = df(x[i])
    end
end
ϵ1 = 1e-6
ϵ2 = 1e-4
N = 20
x0 = 0.5
λ = 2
@time sol = nlsolve(f!, j!, [x0]; method=:newton)
# println(sol)
r1 = sol.zero
@time r2 = newton(f, df, ϵ1, ϵ2, N, x0)
@time r3 = newton(f, df, ϵ1, ϵ2, N, x0, λ)  # multi-root newton method
println("library root solver:\t$r1")
println("sigle root solver:\t$([r2])")
println("multi-root solver:\t$([r3])")
```

```
  0.061985 seconds (145.75 k allocations: 7.441 MiB, 99.87% compilation time)
iter:  7        root:    0.566605704128
  0.012734 seconds (8.72 k allocations: 463.742 KiB, 98.70% compilation time)
iter:  2        root:    0.567143165035
  0.012515 seconds (9.09 k allocations: 481.673 KiB, 98.24% compilation time)
library root solver:    [0.5671096851375182]
sigle root solver:      [0.566605704128146]
multi-root solver:      [0.5671431650348469]
```

思考题

1. 对问题 1 确定初值的原则是什么？实际计算中应如何解决？

   选择一个有根区间，本例中容易得到

   $f(x)=\cos(x)-x$，有$f(0)=1>0, f(\frac{\pi}{2})=-\frac{\pi}{2}<0$，取区间中点即$x=\frac{\pi}{4}$为初值

   $f(x)=e^{-x}-\sin(x)$，有$f(0)=1>0, f(1.2)\approx -0.631<0$，同样取区间中点即$x=0.6$为初值

   实际计算中，根据其他算法求出多个精度较粗的有根区间，然后使用牛顿法逼近获得较为精确的数值解。

   通常，对于我们而言，可能在给定区间直接作出函数图像是最简单最可行的方式。

   > https://computingskillset.com/solving-equations/how-to-find-the-initial-guess-in-newtons-method/

> https://math.stackexchange.com/questions/743373/how-to-choose-the-starting-point-in-newtons-method

在我所查找的资料中提及，对于更一般的情形，试图通过程序自动化来计算函数的根的话，情况会变得更加复杂，涉及到多个领域的研究。

> https://en.wikipedia.org/wiki/Newton_fractal
>
> How to find all roots of complex polynomials by Newton's method

2. 对问题 2 如何解释在计算中出现的现象？试加以说明

本例中，

$(1) f_1(x) = x - e^{-x} = 0$

```
iter:  2 root:    0.567143165035

library root solver: [0.5671432904097811]  0.059150 seconds (134.73 k
allocations: 6.888 MiB, 99.88% compilation time)
single root solver:  [0.5671431650348622]  0.011558 seconds (8.06 k
allocations: 436.974 KiB, 99.16% compilation time)
```

$(2) f_2(x) = x^2 - 2xe^{-x} + e^{-2x} = (x - e^{-x})^2 = {f_1}^2(x) = 0$

```
iter:  7 root:    0.566605704128
iter:  2 root:    0.567143165035

library root solver: [0.5671096851375182]  0.061985 seconds (145.75 k
allocations: 7.441 MiB, 99.87% compilation time)
sigle root solver:   [0.566605704128146]   0.012734 seconds (8.72 k
allocations: 463.742 KiB, 98.70% compilation time)
multi-root solver:   [0.5671431650348469]  0.012515 seconds (9.09 k
allocations: 481.673 KiB, 98.24% compilation time)
```

显然，方程(2)在方程(1)的根位置有重根，可以看到直接应用牛顿迭代法计算轮数为7轮，耗时通常情况下稍微增加，但由于当前实例计算简单、精度要求低，耗时变化不明显，不过能看到在给定精度要求情况下所得精度低于无重根牛顿迭代法。

由理论课知识可知，当存在重根时牛顿迭代法的收敛速度为线性收敛。在后续使用修正的牛顿法可以使收敛速度重新达到平方收敛，耗时几乎与无重根时一致，迭代次数和精度也相同。

除此以外，在本实验中，直接手写的牛顿法运行效率意外的高于库函数直接使用的效率，耗时更短，内存消耗更小，这是让人十分意外的。当然，事实上在同一数量级时间的差异并不大，而库函数代码通用于解非线性系统，仍然是求解实际问题时的优选。

不过，在后期由julia控制台直接运行时所消耗的时间并非如此，考虑到可能是使用IJulia在Jupyter Notebook环境下运行的时间损耗，如下所示。

```
julia> @time sol = nlsolve(f!,j!,[pi/4];method = :newton)
  0.000031 seconds (42 allocations: 2.578 KiB)
Results of Nonlinear Solver Algorithm
 * Algorithm: Newton with line-search
 * Starting Point: [0.7853981633974483]
 * Zero: [0.739085133215161]
 * Inf-norm of residuals: 0.000000
```

```
  * Iterations: 3
  * Convergence: true
    * |x - x'| < 0.0e+00: false
    * |f(x)| < 1.0e-08: true
  * Function Calls (f): 4
  * Jacobian Calls (df/dx): 4
```

3. 略

实验题目5 高斯(Gauss)列主元消去法

代码实现

In [39]:
```
using Printf
using LinearAlgebra
```

In [40]:
```
# from: https://stackoverflow.com/questions/58667332/is-there-a-way-to-swap-columns-
function swapcols!(X::AbstractMatrix, i::Integer, j::Integer)
    @inbounds for k = 1:size(X, 1)
        X[k, i], X[k, j] = X[k, j], X[k, i]
    end
end
# from: https://discourse.julialang.org/t/swap-cols-rows-of-a-matrix/47904/9
function _swapcol!(x, i, j)
    for k in axes(x, 1)  # <- give dimension as input to axes function
        x[k, i], x[k, j] = x[k, j], x[k, i]
    end
end
```

```
_swapcol! (generic function with 1 method)
```

In [41]:
```
function swaprows!(X::AbstractMatrix, i::Integer, j::Integer)
    @inbounds for k = 1:size(X, 2)
        X[i, k], X[j, k] = X[j, k], X[i, k]
    end
end
```

```
swaprows! (generic function with 1 method)
```

In [42]:
```
# https://stackoverflow.com/questions/45396685/what-does-an-exclamation-mark-mean-af
# https://people.richland.edu/james/lecture/m116/matrices/pivot.html
function pivoting!(A::Matrix{Float64}, k::Integer, n::Integer)
    val, idx = findmax(A[k:n, k])
    idx += k - 1  # index must add previous length that omitted by slice operator
    return val, idx
end
function pivoting!(A::Matrix{Float64}, b::Vector{Float64}, k::Integer, n::Int
    s = [maximum(A[i, k:n]) for i in k:n]
    if 0 in s
        println("Cannot solve a singular matrix!")
        return
    end
    if implicit
        val, idx = findmax(A[k:n, k] ./ s[1:n-k+1])
    else
        A[k:n, k:n] = A[k:n, k:n] ./ s
        b[k:n] = b[k:n] ./ s
        val, idx = findmax(A[k:n, k])
    end
    idx += k - 1  # index must add previous length that omitted by slice operator
    return val, idx
end
```

```
pivoting! (generic function with 2 methods)
```

In [43]:
```
# Gauss列主元消去法
# Todo: modify it using . operator
function gauss(n, A::Matrix{Float64}, b::Vector{Float64})
    for k = 1:n-1
```

```
        # select pivot in columns
        val, idx = pivoting!(A, k, n)
        if val == 0
            println("Cannot solve a singular matrix!")
            return
        end
        # swap rows
        if idx != k
            swaprows!(A, idx, k)
            b[idx], b[k] = b[k], b[idx]
        end
        # elimination
        for i = k+1:n
            m = A[i, k] / A[k, k]
            A[i, :] -= A[k, :] * m
            b[i] -= b[k] * m
        end
    end
    if A[n, n] == 0
        println("Cannot solve a singular matrix!")
        return
    end
    # https://stackoverflow.com/questions/62142717/julia-quick-way-to-initialise-an-
    x = similar(b, Float64)
    x[n] = b[n] / A[n, n]
    for k = n-1:-1:1  # the usage of reverse sequence
        x[k] = (b[k] - dot(A[k, k+1:n], x[k+1:n])) / A[k, k]  # something really an
    end
    x
end
```

```
gauss (generic function with 2 methods)
```

In [44]:

```
# Gauss列主元消去法
# Todo: modify it using . operator
function gauss(n, A::Matrix{Float64}, b::Vector{Float64}, implicit::Bool)
    for k = 1:n-1
        # select pivot in columns
        val, idx = pivoting!(A, b, k, n, implicit)
        if val == 0
            println("Cannot solve a singular matrix!")
            return
        end
        # swap rows
        if idx != k
            swaprows!(A, idx, k)
            b[idx], b[k] = b[k], b[idx]
        end
        # elimination
        for i = k+1:n
            m = A[i, k] / A[k, k]
            A[i, :] -= A[k, :] * m
            b[i] -= b[k] * m
        end
    end
    if A[n, n] == 0
        println("Cannot solve a singular matrix!")
        return
    end
    # https://stackoverflow.com/questions/62142717/julia-quick-way-to-initialise-an-
    x = similar(b, Float64)
    x[n] = b[n] / A[n, n]
    for k = n-1:-1:1  # the usage of reverse sequence
        x[k] = (b[k] - dot(A[k, k+1:n], x[k+1:n])) / A[k, k]  # something really an
```

```
    end
    x
end
```

```
gauss (generic function with 2 methods)
```

测试代码

Test 1 - Correctness

In [45]:
```
# test random result of standard library
# test pass
for i in 1:5
    M = rand(300, 300)
    v = rand(300)
    A, b = copy(M), copy(v)  # ? Todo: move this line into try block will extend com
    try
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b), 2))\t")

        A, b = copy(M), copy(v)
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b, false), 2))\t")  # impli

        A, b = copy(M), copy(v)
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b, true), 2))\t")  # implic
    catch SigularException
        println("Cannot solve a singular matrix!")
    end
    println()
end
```

```
9.318988586837023e-13    0.275640 seconds (278.72 k allocations: 444.834 MiB, 14.4
4% gc time, 32.70% compilation time)
1.4865593573213545e-12   0.493795 seconds (475.89 k allocations: 665.127 MiB, 10.6
8% gc time, 47.91% compilation time)
1.893917465969191e-12    0.213457 seconds (227.26 k allocations: 515.581 MiB, 16.8
5% gc time)

3.5673692065171883e-11   0.160950 seconds (180.32 k allocations: 439.810 MiB, 19.1
3% gc time)
1.1903308858224424e-10   0.241354 seconds (228.37 k allocations: 653.650 MiB, 13.2
0% gc time)
7.475839084501319e-11    0.198274 seconds (227.26 k allocations: 515.581 MiB, 15.8
1% gc time)

2.484235319254942e-13    0.155229 seconds (180.31 k allocations: 439.810 MiB, 18.2
5% gc time)
2.3602954910384214e-13   0.244467 seconds (228.37 k allocations: 653.650 MiB, 15.8
0% gc time)
2.6888000866323927e-13   0.200813 seconds (227.26 k allocations: 515.581 MiB, 15.3
3% gc time)

1.981893077413214e-12    0.157618 seconds (180.31 k allocations: 439.810 MiB, 17.8
1% gc time)
2.6310099338685277e-12   0.235614 seconds (228.37 k allocations: 653.650 MiB, 15.0
1% gc time)
3.4283323739836462e-12   0.207191 seconds (227.26 k allocations: 515.581 MiB, 15.0
8% gc time)

7.446480464646344e-12    0.171749 seconds (180.32 k allocations: 439.810 MiB, 18.8
0% gc time)
9.250087276929403e-12    0.412266 seconds (228.37 k allocations: 653.650 MiB, 45.6
6% gc time)
9.312773369641041e-12    0.202125 seconds (227.26 k allocations: 515.581 MiB, 17.1
1% gc time)
```

Test 2 - Performance

In [46]:

```
# x = [@elapsed(rand(i,i)\rand(i)) for i = 10:15]
# for i = 10:15
#     A, b = rand(i,i), rand(i)
#     @time A\b
# end
# # compute moving average
# # https://stackoverflow.com/questions/28820904/how-to-efficiently-compute-average-
# # n=1;
# # curAvg = 0;
# # loop{
# #   curAvg = curAvg + (newNum - curAvg)/n;
# #   n++;
# # }
# n = 2000
# curAvg = zeros(n)
# # curAvg[i] = 0
# for i = 2:n
#     curAvg[i] = curAvg[i-1] + (@elapsed(rand(i,i)\rand(i)) - curAvg[i-1])/(i-1)
# end
# plot(curAvg)

# function ma(n)
#     curAvg = zeros(n)
#     # curAvg[i] = 0
#     for i = 2:n
```

```
#         curAvg[i] = curAvg[i-1] + (@elapsed(rand(i,i)\rand(i)) - curAvg[i-1])/(i-1
#     end
#     plot(curAvg)
# end
```

In [47]:
```
# test random result of standard library
# test pass
for i in 1:5
    M = rand(300, 300)
    v = rand(300)
    A, b = copy(M), copy(v)
    try
        @time A \ b

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b)

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b, false)  # implicit=false

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b, true)  # implicit=true
    catch SigularException
        println("Cannot solve a singular matrix!")
    end
    println()
end
```

```
  0.005614 seconds (4 allocations: 708.172 KiB)
  0.157441 seconds (180.30 k allocations: 439.115 MiB, 18.55% gc time)
  0.230287 seconds (228.35 k allocations: 652.955 MiB, 14.68% gc time)
  0.192803 seconds (227.24 k allocations: 514.886 MiB, 18.65% gc time)

  0.006017 seconds (4 allocations: 708.172 KiB)
  0.147328 seconds (180.30 k allocations: 439.115 MiB, 18.96% gc time)
  0.228614 seconds (228.35 k allocations: 652.955 MiB, 15.55% gc time)
  0.197291 seconds (227.24 k allocations: 514.886 MiB, 17.19% gc time)

  0.005568 seconds (4 allocations: 708.172 KiB)
  0.159448 seconds (180.30 k allocations: 439.115 MiB, 19.94% gc time)
  0.226598 seconds (228.35 k allocations: 652.955 MiB, 14.99% gc time)
  0.194048 seconds (227.24 k allocations: 514.886 MiB, 14.52% gc time)

  0.005533 seconds (4 allocations: 708.172 KiB)
  0.153294 seconds (180.30 k allocations: 439.115 MiB, 19.95% gc time)
  0.223405 seconds (228.35 k allocations: 652.955 MiB, 14.09% gc time)
  0.194211 seconds (227.24 k allocations: 514.886 MiB, 15.78% gc time)

  0.006003 seconds (4 allocations: 708.172 KiB)
  0.146435 seconds (180.30 k allocations: 439.115 MiB, 19.11% gc time)
  0.225317 seconds (228.35 k allocations: 652.955 MiB, 15.15% gc time)
  0.194285 seconds (227.24 k allocations: 514.886 MiB, 17.41% gc time)
```

Test 3 - Special Matrix

Bug: 处理上三角矩阵计算有异常

Todo: 应当增加对于特殊矩阵的测试，待测试列表同Julia Special Matrix:

- [ ] Symmetric
- [ ] Hermitian

- [ ] UpperTriangular
- [ ] UnitUpperTriangular
- [ ] LowerTriangular
- [ ] UnitLowerTriangular
- [ ] UpperHessenberg
- [ ] Tridiagonal
- [ ] SymTridiagonal
- [ ] Bidiagonal
- [ ] Diagonal
- [ ] UniformScaling

基本语法为 `M = SpecialMatrix(rand(300,300))`

实验题目

问题 1

In [48]:
```
function show_result(A, b)
    n = size(A, 1)
    A = Float64.(A)
    b = Float64.(b)
    display("input matrix:")
    # https://www.geeksforgeeks.org/creating-array-with-repeated-elements-in-julia-r
    # http://www.jlhub.com/julia/manual/en/function/repeat
    display([A repeat([], inner=(n, 1)) b])
    display("result by standard library:")
    display(@time A \ b)
    display("result by my gauss method:")
    display(@time gauss(n, A, b))
    # display(@time gauss(n, A, b, false))  # implicit=false
    # display(@time gauss(n, A, b, true))  # implicit=true
end
```

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

In [49]:
```
A = [0.4096 0.1234 0.3678 0.2943
    0.2246 0.3872 0.4015 0.1129
    0.3645 0.1920 0.3781 0.0643
    0.1784 0.4002 0.2786 0.3927]
b = [1.1951; 1.1262; 0.9989; 1.2499]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 0.4096  0.1234  0.3678  0.2943  │  1.1951
 0.2246  0.3872  0.4015  0.1129  │  1.1262
 0.3645  0.192   0.3781  0.0643  │  0.9989
 0.1784  0.4002  0.2786  0.3927  │  1.2499
"result by standard library:"
4-element Vector{Float64}:
 0.9999999999999994
 1.0
 1.0000000000000004
 1.0000000000000002
"result by my gauss method:"
```

```
4-element Vector{Float64}:
 1.0000000000000027
 1.0000000000000018
 0.9999999999999971
 0.9999999999999992
  0.000012 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [50]:
```
A = [136.01 90.860 0 0
    90.860 98.810 -67.590 0
    0 -67.590 132.01 46.260
    0 0 46.260 177.17]
b = [226.87; 122.08; 110.68; 223.43]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 136.01   90.86    0.0     0.0    │  226.87
  90.86   98.81  -67.59    0.0    │  122.08
   0.0   -67.59  132.01   46.26   │  110.68
   0.0     0.0   46.26   177.17   │  223.43
"result by standard library:"
4-element Vector{Float64}:
 1.0000000000000704
 0.9999999999998946
 0.9999999999999408
 1.0000000000000155
"result by my gauss method:"
4-element Vector{Float64}:
 1.000000000000133
 0.9999999999998009
 0.999999999999888
 1.0000000000000293
  0.000010 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [51]:
```
A = [1 1/2 1/3 1/4
     1/2 1/3 1/4 1/5
     1/3 1/4 1/5 1/6
     1/4 1/5 1/6 1/7]
b = [25 / 12; 77 / 60; 57 / 60; 319 / 420]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 1.0       0.5       0.333333  0.25      │  2.08333
 0.5       0.333333  0.25      0.2       │  1.28333
 0.333333  0.25      0.2       0.166667  │  0.95
 0.25      0.2       0.166667  0.142857  │  0.759524
"result by standard library:"
4-element Vector{Float64}:
 0.9999999999999779
 1.000000000000241
 0.9999999999994366
 1.0000000000003588
"result by my gauss method:"
4-element Vector{Float64}:
 0.9999999999999927
 1.0000000000000688
 0.9999999999998556
 1.0000000000000846
  0.000009 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [52]:
```
A = [10 7 8 7
```

```
    7 5 6 5
    8 6 10 9
    7 5 9 10]
b = [32; 23; 33; 31]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 10.0  7.0   8.0   7.0  |  32.0
  7.0  5.0   6.0   5.0  |  23.0
  8.0  6.0  10.0   9.0  |  33.0
  7.0  5.0   9.0  10.0  |  31.0
"result by standard library:"
4-element Vector{Float64}:
 1.000000000000083
 0.9999999999998619
 1.000000000000035
 0.999999999999979
"result by my gauss method:"
4-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
  0.000014 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

问题 2

In [53]:
```
A = [197 305 -206 -804
     46.8 71.3 -47.4 52.0
     88.6 76.4 -10.8 802
     1.45 5.90 6.13 36.5]
b = [136; 11.7; 25.1; 6.60]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 197.0   305.0  -206.0   -804.0  |  136.0
  46.8    71.3   -47.4     52.0  |   11.7
  88.6    76.4   -10.8    802.0  |   25.1
   1.45    5.9     6.13    36.5  |    6.6
"result by standard library:"
4-element Vector{Float64}:
  0.9536791069017718
  0.32095684552110354
  1.0787080757932384
 -0.09010850953957893
"result by my gauss method:"
4-element Vector{Float64}:
  0.9536791069017717
  0.3209568455211036
  1.0787080757932384
 -0.09010850953957895
  0.000009 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [54]:
```
A = [0.5398 0.7161 -0.5554 -0.2982
    0.5257 0.6924 0.3565 -0.6255
    0.6465 -0.8187 -0.1872 0.1291
    0.5814 0.9400 -0.7779 -0.4042]
b = [0.2058; -0.0503; 0.1070; 0.1859]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 0.5398   0.7161  -0.5554  -0.2982  |   0.2058
 0.5257   0.6924   0.3565  -0.6255  |  -0.0503
 0.6465  -0.8187  -0.1872   0.1291  |   0.107
 0.5814   0.94    -0.7779  -0.4042  |   0.1859
"result by standard library:"
4-element Vector{Float64}:
 0.5161772979585422
 0.4152194728301359
 0.10996610286788958
 1.0365392233362019
"result by my gauss method:"
4-element Vector{Float64}:
 0.5161772979585416
 0.41521947283013527
 0.10996610286788916
 1.0365392233362005
  0.000011 seconds (3 allocations: 384 bytes)
  0.000007 seconds (34 allocations: 3.031 KiB)
```

In [55]:
```
A = [10 1 2
    1 10 2
    1 1 5]
b = [13; 13; 7]
show_result(A, b)
```

```
"input matrix:"
3×5 Matrix{Any}:
 10.0   1.0  2.0  |  13.0
  1.0  10.0  2.0  |  13.0
  1.0   1.0  5.0  |   7.0
"result by standard library:"
3-element Vector{Float64}:
 1.0
 0.9999999999999998
 1.0
"result by my gauss method:"
3-element Vector{Float64}:
 1.0
 0.9999999999999998
 1.0000000000000002
  0.000014 seconds (3 allocations: 288 bytes)
  0.000006 seconds (19 allocations: 1.453 KiB)
```

In [56]:
```
A = [4 -2 -4
    -2 17 10
    -4 10 9]
b = [-2; 25; 15]
show_result(A, b)
```

```
"input matrix:"
3×5 Matrix{Any}:
  4.0  -2.0  -4.0  |  -2.0
 -2.0  17.0  10.0  |  25.0
 -4.0  10.0   9.0  |  15.0
"result by standard library:"
3-element Vector{Float64}:
 1.0
 1.0
 1.0
"result by my gauss method:"
```

```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
  0.000011 seconds (3 allocations: 288 bytes)
  0.000007 seconds (19 allocations: 1.453 KiB)
```

总结

Todo: 直接法效率评价，写一个迭代法来算，用于比较效率，库函数使用的方法

Todo: 参考资料链接放在文末

Todo: 特殊矩阵的测试代码，剩余bug修复