数値計算法　演習課題 2　 提出日：2024 年 5 月 18 日
202310330　 長田悠生

課題 1

(1-1)

(1-1) のソースコード

```
module MonteCarloModule
    using Random
    Random.seed!(0)
    function MonteCarlo(n::Int64)::Float64
        x::Vector{Float64} = zeros(Float64, n)
        y::Vector{Float64} = zeros(Float64, n)
        for i::Int64 = 1:n
            x[i] = rand()
            y[i] = rand()
        end
        r::Vector{Float64} = zeros(Float64, n)
        for i::Int64 = 1:n
            r[i] = x[i]^2 + y[i]^2
        end
        m::Int64 = 0
        for i::Int64 = 1:n
            if r[i] <= 1
                m = m + 1
            end
        end
        p::Float64 = 4*m/n
        return p
    end

    function monte_carlo_func(n::Int64)::Float64
        if n <= 0
            throw("n have to larger than 0.")
        end
        return MonteCarlo(n)
    end
end
```

数値計算法　演習課題 2　 提出日：2024 年 5 月 18 日
202310330　 長田悠生

前ページの関数について具体的な n の値を入れてみたときのソースコードと出力結果です。前ページのソースコードと以下のソースコードは同一のファイルに書いています。

上記の関数に具体的な n の値を入れてみたときのソースコード

```
using .MonteCarloModule

p_1::Float64 = MonteCarloModule.monte_carlo_func(1)
println("p_1 = $p_1")

p_100::Float64 = MonteCarloModule.monte_carlo_func(100)
println("p_100 = $p_100")

p_0::Float64 = MonteCarloModule.monte_carlo_func(0)
println("p_0 = $p_0")
```

実行結果

```
$ julia --project ./src/report1-1.jl
p_1 = 4.0
p_100 = 3.28
ERROR: LoadError: "n have to larger than 0."
Stacktrace:
    [1] monte_carlo_func(n::Int64)
    @ Main.MonteCarloModule ~/Desktop/julia-numeric-calculation/
       class2_report/src/report1-1.jl:27
    [2] top-level scope
    @ ~/Desktop/julia-numeric-calculation/class2_report/src/
       report1-1.jl:41
in expression starting at /home/hello/Desktop/julia-numeric-
   calculation/class2_report/src/report1-1.jl:41
```

(1-2)

(1-2) のソースコード

```
module MonteCarloModule
    using Random
    Random.seed!(0)
    function MonteCarlo(n::Int64)::Float64
        x::Vector{Float64} = zeros(Float64, n)
        y::Vector{Float64} = zeros(Float64, n)
        for i::Int64 = 1:n
            x[i] = rand()
            y[i] = rand()
        end
        r::Vector{Float64} = zeros(Float64, n)
        for i::Int64 = 1:n
            r[i] = x[i]^2 + y[i]^2
        end
        m::Int64 = 0
        for i::Int64 = 1:n
            if r[i] <= 1
                m = m + 1
            end
        end
        p::Float64 = 4*m/n
        return p
    end

    function monte_carlo_func(n::Int64)::Float64
        if n <= 0
            throw("n have to larger than 0.")
        end
        return MonteCarlo(n)
    end

    struct MonteCarloData
        n_vec::Array{Float64}
```

```
        result_vec::Array{Float64}
    end

    function monte_carlo_data_func(size::Int64)::MonteCarloData
        n_vec::Array{Int64} = zeros(Int64, size)
        result_vec::Array{Float64} = zeros(Float64, size)
        for i::Int64 = range(1, size, size)
            n::Int64 = 10^i
            result::Float64 = monte_carlo_func(n)
            n_vec[i] = n
            result_vec[i] = abs(result - pi)
        end
        data::MonteCarloData = MonteCarloData(n_vec, result_vec)
        return data
    end
end

using .MonteCarloModule
using Plots

data::MonteCarloModule.MonteCarloData = MonteCarloModule.
    monte_carlo_data_func(6)

plot(data.n_vec, data.result_vec, xaxis=:log)
savefig("report1-2.png")
```

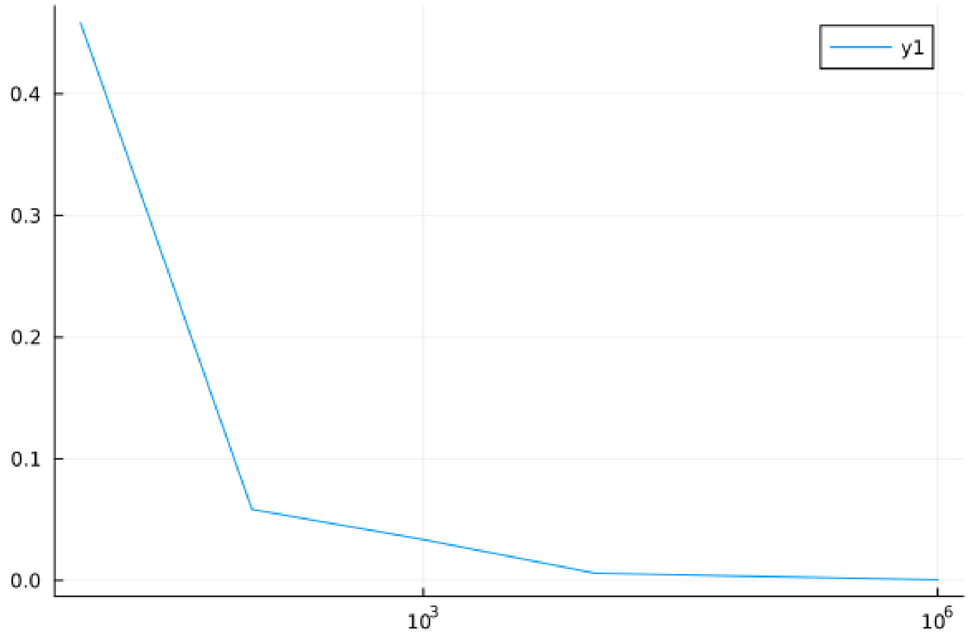

図 1　実行結果のグラフ

課題 2

(2-1)

課題 2 では $roots(Polynomial([c,b,a]))$ で計算した結果を正しいものと仮定しているので、$\frac{-b+\sqrt{b^2-4ac}}{2a} = (-b + sqrt(b^2.0 - 4.0 * a * c))/(2.0 * a)$、$\frac{-b-\sqrt{b^2-4ac}}{2a} = (-b - sqrt(b^2.0 - 4.0 * a * c))/(2.0 * a)$ とすると、残差の計算式を $\left|f\left(\frac{-b-\sqrt{b^2-4ac}}{2a}\right) - 0\right|$ と $\left|f\left(\frac{-b+\sqrt{b^2-4ac}}{2a}\right) - 0\right|$ ではなく、$\left|f\left(\frac{-b-\sqrt{b^2-4ac}}{2a}\right) - f(roots(Polynomial([c,b,a]))[1])\right|$ と $\left|f\left(\frac{-b+\sqrt{b^2-4ac}}{2a}\right) - f(roots(Polynomial([c,b,a]))[2])\right|$ にしている。

(2-1) のソースコード

```
module QuadraticEquation
using Polynomials

struct Term
    c::Float64
    b::Float64
```

```
        a::Float64
    end

    struct Solution
        true_value::Array{Float64}
        approximation_value::Array{Float64}
    end

    function approximation_func(term::Term)::Array{Float64}
        return [(-term.b - sqrt(term.b^2.0 -4.0*term.a*term.c
            ))/(2.0*term.a), (-term.b + sqrt(term.b^2.0 -4.0*term
            .a*term.c))/(2.0*term.a)]
    end

    function true_func(term::Term)::Array{Float64}
        coefficient::Array{Float64} = [term.c, term.b, term.a]
        return roots(Polynomial(coefficient))
    end

    function absolute_error_func(solution::Solution)::Array{
       Float64}
        return [
            abs(solution.true_value[1] - solution.
                approximation_value[1]),
            abs(solution.true_value[2] - solution.
                approximation_value[2])
        ]
    end

    function relative_error_func(solution::Solution)::Array{
       Float64}
        return [
            abs(solution.approximation_value[1] - solution.
                true_value[1]) / abs(solution.true_value[1]),
            abs(solution.approximation_value[2] - solution.
                true_value[2]) / abs(solution.true_value[2]),
        ]
```

```
    end

    function quadratic_equation(term::Term, x::Float64)::Float64
        return term.a*x^2 + term.b*x + term.c
    end

    function rest_func(term::Term, solution::Solution)::Array{
       Float64}
        return [
            abs(quadratic_equation(term, solution.
               approximation_value[1]) - quadratic_equation(term
               , solution.true_value[1])),
            abs(quadratic_equation(term, solution.
               approximation_value[2]) - quadratic_equation(term
               , solution.true_value[2])),
        ]
    end
end

import .QuadraticEquation

values::QuadraticEquation.Term = QuadraticEquation.Term(1.0,
   -124.0, 1.0)

println("===解の公式を使って解を求める===")
approximation_value::Array{Float64} = QuadraticEquation.
   approximation_func(values)
println(approximation_value)

println("===正確な解を求める===")
true_value::Array{Float64} = QuadraticEquation.true_func(values)
println(true_value)

solution::QuadraticEquation.Solution = QuadraticEquation.
   Solution(true_value, approximation_value)

println("===絶対誤差===")
```

```
absolute_error::Array{Float64} = QuadraticEquation.
    absolute_error_func(solution)
println(absolute_error)

println("===相対誤差===")
relative_error::Array{Float64} = QuadraticEquation.
    relative_error_func(solution)
println(relative_error)

true_result_1 = QuadraticEquation.quadratic_equation(values,
    solution.true_value[1])
true_result_2 = QuadraticEquation.quadratic_equation(values,
    solution.true_value[2])
approximation_result_1 = QuadraticEquation.quadratic_equation(
    values, solution.approximation_value[1])
approximation_result_2 = QuadraticEquation.quadratic_equation(
    values, solution.approximation_value[2])
println("===2次方程式の解を求める===")
println("x = $(solution.true_value[1]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $true_result_1")
println("x = $(solution.true_value[2]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $true_result_2")
println("x = $(solution.approximation_value[1]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $approximation_result_1")
println("x = $(solution.approximation_value[2]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $approximation_result_2")

println("===残差===")
rest_value = QuadraticEquation.rest_func(values, solution)
println(rest_value)
```

実行結果

```
$ julia --project ./src/report2-1.jl
===解の公式を使って解を求める===
[0.008065040684527958, 123.99193495931547]
===正確な解を求める===
[0.008065040684526154, 123.99193495931547]
===絶対誤差===
[1.8041124150158794e-15, 0.0]
===相対誤差===
[2.2369538922194248e-13, 0.0]
===2次方程式の解を求める===
x = 0.008065040684526154 のとき、
1.0*x^2 + -124.0*x + 1.0 = 1.1102230246251565e-16
x = 123.99193495931547 のとき、
1.0*x^2 + -124.0*x + 1.0 = 0.0
x = 0.008065040684527958 のとき、
1.0*x^2 + -124.0*x + 1.0 = -2.2359891715950653e-13
x = 123.99193495931547 のとき、
1.0*x^2 + -124.0*x + 1.0 = 0.0
===残差===
[2.2370993946196904e-13, 0.0]
```

(2-2)

課題 2 では $roots(Polynomial([c, b, a]))$ で計算した結果を正しいものと仮定しているので、$\alpha = \frac{-b+\sqrt{b^2-4ac}}{2a}$、$\beta = \frac{c}{a\alpha}$ とすると、残差の計算式を $\left|f\left(\frac{-b-\sqrt{b^2-4ac}}{2a}\right)-0\right|$ と $\left|f\left(\frac{-b+\sqrt{b^2-4ac}}{2a}\right)-0\right|$ ではなく、$\left|f(\beta)-f(roots(Polynomial([c, b, a]))[1])\right|$ と $\left|f(\alpha)-f(roots(Polynomial([c, b, a]))[2])\right|$ にしている。

実行結果

```
module QuadraticEquation
    using Polynomials

    struct Term
        c::Float64
```

```
    b::Float64
    a::Float64
end\fra

struct Solution
    true_value::Array{Float64}
    approximation_value::Array{Float64}
end

function approximation_func(term::Term)::Array{Float64}
    return [(-term.b - sqrt(term.b^2.0 -4.0*term.a*term.c
        ))/(2.0*term.a), (-term.b + sqrt(term.b^2.0 -4.0*term
        .a*term.c))/(2.0*term.a)]
end

function true_func(term::Term)::Array{Float64}
    coefficient::Array{Float64} = [term.c, term.b, term.a]
    return roots(Polynomial(coefficient))
end

function absolute_error_func(solution::Solution)::Array{
   Float64}
    return [
        abs(solution.true_value[1] - solution.
            approximation_value[1]),
        abs(solution.true_value[2] - solution.
            approximation_value[2])
    ]
end

function relative_error_func(solution::Solution)::Array{
   Float64}
    return [
        abs(solution.approximation_value[1] - solution.
            true_value[1]) / abs(solution.true_value[1]),
        abs(solution.approximation_value[2] - solution.
            true_value[2]) / abs(solution.true_value[2]),
```

```
        ]
    end

    function quadratic_equation(term::Term, x::Float64)::Float64
        return term.a*x^2 + term.b*x + term.c
    end

    function rest_func(term::Term, solution::Solution)::Array{
        Float64}
        return [
            abs(quadratic_equation(term, solution.
                approximation_value[1]) - quadratic_equation(term
                , solution.true_value[1])),
            abs(quadratic_equation(term, solution.
                approximation_value[2]) - quadratic_equation(term
                , solution.true_value[2])),
        ]
    end

    function prevent_digit_loss_func(term::Term)::Array{Float64}
        alpha::Float64 = (-term.b + sqrt(term.b^2.0 -4.0*term.a*
            term.c))/(2.0*term.a)
        beta::Float64 = term.c / (alpha * term.a)
        return [beta, alpha]
    end

end

import .QuadraticEquation

values::QuadraticEquation.Term = QuadraticEquation.Term(1.0,
    -124.0, 1.0)

println("===桁落ちを防いで解を求める===")
prevent_digit_loss::Array{Float64} = QuadraticEquation.
    prevent_digit_loss_func(values)
println(prevent_digit_loss)
```

```
println("===正確な解を求める===")
true_value::Array{Float64} = QuadraticEquation.true_func(values)
println(true_value)

prevent_digit_loss_solution::QuadraticEquation.Solution =
    QuadraticEquation.Solution(true_value, prevent_digit_loss)

println("===絶対誤差===")
prevent_digit_loss_absolute_error::Array{Float64} =
    QuadraticEquation.absolute_error_func(
    prevent_digit_loss_solution)
println(prevent_digit_loss_absolute_error)

println("===相対誤差===")
prevent_digit_loss_relative_error::Array{Float64} =
    QuadraticEquation.relative_error_func(
    prevent_digit_loss_solution)
println(prevent_digit_loss_relative_error)

true_result_1 = QuadraticEquation.quadratic_equation(values,
    true_value[1])
true_result_2 = QuadraticEquation.quadratic_equation(values,
    true_value[2])
prevent_digit_loss_result_1 = QuadraticEquation.
    quadratic_equation(values, prevent_digit_loss[1])
prevent_digit_loss_result_2 = QuadraticEquation.
    quadratic_equation(values, prevent_digit_loss[2])
println("===2次方程式の解を求める===")
println("x = $(true_value[1]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $true_result_1")
println("x = $(true_value[2]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $true_result_2")
println("x = $(prevent_digit_loss[1]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
```

```
    $prevent_digit_loss_result_1")
println("x = $(prevent_digit_loss[2]) のとき、")
println("$(values.a)*x^2 + $(values.b)*x + $(values.c) =
    $prevent_digit_loss_result_2")

println("===残差===")
prevent_digit_loss_rest_value = QuadraticEquation.rest_func(
    values, prevent_digit_loss_solution)
println(prevent_digit_loss_rest_value)
```

実行結果

```
$ julia --project ./src/report2-2.jl
===桁落ちを防いで解を求める===
[0.008065040684526154, 123.99193495931547]
===正確な解を求める===
[0.008065040684526154, 123.99193495931547]
===絶対誤差===
[0.0, 0.0]
===相対誤差===
[0.0, 0.0]
===2次方程式の解を求める===
x = 0.008065040684526154 のとき、
1.0*x^2 + -124.0*x + 1.0 = 1.1102230246251565e-16
x = 123.99193495931547 のとき、
1.0*x^2 + -124.0*x + 1.0 = 0.0
x = 0.008065040684526154 のとき、
1.0*x^2 + -124.0*x + 1.0 =  1.1102230246251565e-16
x = 123.99193495931547 のとき、
1.0*x^2 + -124.0*x + 1.0 =  0.0
===残差===
[0.0, 0.0]
```

(2-3)

桁落ちは、絶対値が近い大きさの小数同士の減算を行ったときに、有効数字が減る減少である。桁落ちを防ぐ計算をしない場合でも $\frac{-b+\sqrt{b^2-4ac}}{2a}$ では誤差が生じておらず、$\frac{-b-\sqrt{b^2-4ac}}{2a}$ で誤差が生じているため、$-b$ と $\sqrt{b^2-4ac}$ との減算で桁落ちが発生したと考えられる。また、桁落ちは絶

対値が近い大きさの小数同士の減算を行ったときに有効数字が減るので、$|-b|$ と $|\sqrt{b^2-4ac}|$ の大きさが近いと桁落ちが発生する。従って、b^2 と $4ac$ との差がとても大きいときに桁落ちが発生する可能性があると考えられる。

参考文献

[1] Chapter 8: Handling errors、Learn Julia the Hard Way https://scls.gitbooks.io/ljthw/content/_chapters/11-ex8.html

[2] Julia を基礎からゆっくりと（その 08/18)、Hatena Blog https://power-of-awareness.com/entry/2022/10/17/120000

[3]【Julia】 基礎: 型、Qiita https://qiita.com/Krypf/items/145ad24911ca51117e29

[4] Julia の型定義、Qiita https://qiita.com/bonten999/items/4fa3723884a5684c464c

[5] いまから使える Julia 言語、Julia in Physics 2021 Online https://bicycle1885.org/slides/2021/juliainphysics2021/

[6] コンピュータソフトウェアの LaTeX スタイルファイルで lstlisting のキャプションが崩れる問題、Zenn https://zenn.dev/niyaton/articles/9fa4daaab48d16

[7] Julia で Plots のグラフが表示されない、teratail https://teratail.com/questions/366881

[8] 早わかり すぐに使える julia 1.0、早わかり すぐに使える julia 1.0 https://cheatsheet.juliadocs.org/ja/

[9] Julia 早引きノート［01］変数・定数の使い方、Qiita https://qiita.com/ttabata/items/a1ada2c0cba03672e105

[10] 桁落ち、ZDNET https://japan.zdnet.com/glossary/exp/%E6%A1%81%E8%90%BD%E3%81%A1/?s=4

[11] title、IT 用語辞典 e-Words https://e-words.jp/w/%E6%A1%81%E8%90%BD%E3%81%A1.html

[12] chi-feng/preamble、GitHub https://gist.github.com/chi-feng/6589066

[13]【LaTeX】ソースコードの張り付け方、お茶の葉 https://www.ochappa.net/posts/latex-code-b

[14] listings (download here) 、xyoshiki.web http://xyoshiki.web.fc2.com/tex/listings.html

[15] Comprehensive TeX Archive Network、CTAN https://www.ctan.org/tex-archive/macros/latex/contrib/listings

[16] LaTex に日本語を含むソースコードの記載、Qiita https://qiita.com/izayoi5776/items/67a65dae5f2ce5ff84a3

[17] Latex に日本語を含んだソースコードを載せる、Hatena Blog https://contents-open.hatenablog.com/entry/2021/07/24/140759

[18] LaTeX の Listing 対応言語一覧 、晴耕雨読 https://tex2e.github.io/blog/latex/listing-predefined-languages