Równanie potencjału grawitacyjnego rozwiązane metodą elementów skończonych

Filip Dziurdzia

21 Styczeń 2023

Część teoretyczna

Opis problemu

Zadane równanie do rozwiązanie MES:

$$\frac{d^2\Phi}{dx^2} = 4\pi G\rho(x) \tag{1}$$

$$\Phi(0) = 5$$

$$\Phi(3) = 4$$

$$\rho(x) = \begin{cases} 0 \ dla \ x \in [0,1] \\ 1 \ dla \ x \in (1,2] \\ 0 \ dla \ x \in (2,3] \end{cases}$$

gdzie szukaną funkcją jest

$$\Phi(x) \in \mathbb{R}$$

$$x \in [0,3]$$

Równanie słabe (wariacyjne)

Równanie możemy zapisać w postaci

$$\Phi''(x) = 4\pi G\rho(x) \tag{2}$$

Następnie przyjmujemy przestrzeń V, która zeruje się na brzegach oraz przemnażamy obie strony równania przez funkcję testową v ∈ V, po czym otrzymujemy

$$\int_0^3 \Phi(x)v(x)dx = \int_0^3 3\pi G\rho(x)v(x)dx \tag{3}$$

Przekształcamy równanie całkując lewą stronę przez części

$$\Phi'(x)v(x)\Big|_0^3 - \int_0^3 \Phi'(x)v'(x)dx = \int_0^3 4\pi G\rho(x)v(x)dx$$

Ponieważ przestrzeń zeruje się na brzegach, wiemy że $v(0) = 0$ i $v(3) = 0$

$$\Phi'(x)v(x)\Big|_0^3 = 0$$

Stąd zostaje nam postać

$$-\int_0^3 \Phi'(x)v'(x)dx = \int_0^3 4\pi G\rho(x)v(x)dx \tag{4}$$

W podanych danych 1 warunki Dirichleta $\Phi(0) = 5$ i $\Phi(3) = 4$ są niezerowe, stąd musimy zastosować shift Dirichleta

$$\begin{aligned} \tilde{\Phi}(x) &= 5 - \frac{x}{3} \\ \Phi(x) &= \tilde{\Phi}(x) + w(x), \quad w \in V \\ \Phi(x) &= 5 - \frac{x}{3} + w(x) \\ \Phi'(x) &= w'(x) - \frac{1}{3} \end{aligned} \tag{5}$$

Następnie podstawiamy do równania 4

$$-\int_0^3 \left(w'(x) - \frac{1}{3}\right) v'(x)dx = 4\pi G \int_0^3 \rho(x)v(x)dx$$

$$-\int_0^3 w'(x)v'(x)dx \ + \ \frac{1}{3}\int_0^3 v'(x)dx = 4\pi G \int_0^3 \rho(x)v(x)dx$$

$$-\int_0^3 w'(x)v'(x)dx = 4\pi G \int_0^3 \rho(x)v(x)dx - \frac{1}{3}\int_0^3 v'(x)dx \tag{6}$$

Korzystając z danych 1 podstawiamy $\rho(x)$ do całki w równaniu 6

$$4\pi G \left(\int_0^1 0v(x)dx + \int_1^2 1v(x)dx + \int_2^3 0v(x)dx\right)$$

$$4\pi G \int_1^2 v(x)dx \tag{7}$$

Wprowadzamy pomocnicze oznaczenia

$$B(w, v) = -\int_0^3 w'(x)v'(x)dx$$

$$L(v) = 4\pi G \int_1^2 v(x)dx - \frac{1}{3}\int_0^3 v'(x)dx$$

Otrzymując

$$B(w, v) = L(v) \tag{8}$$

Metoda Galerkina

Metoda Galerkina pozwali uzyskać przybliżony wynik zadanego równania
Niech $w_h \in V_h \subset V$

$$w \approx w_h = \sum_{0}^{n} w_i e_i = \sum_{1}^{n-1} w_i e_i$$

Przyjmujemy n jako liczbę punktów podziału (w tym krańce dziedziny), a e_i jako funkcje bazowyme, które generują przestrzeń V_h

$$e_i = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & dla\ x \in (x_{i-1},\ x_i) \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & dla\ x \in (x_i,\ x_{i+1}) \end{cases}$$

$$V_h = [e_1, e_2, e_3, ...e_{n-1}]$$

Podstawiamy do równania 8

$$B\left(\sum_{1}^{n-1} w_i e_i, v\right) = L(v)$$

$$\sum_{1}^{n-1} w_i B(e_i, v) = L(v)$$

Zauważając, że $w_h \in V_h \subset V \ni v$ możemy zapisać v jako e_j dla $j = 1, 2, 3, ...n-1$

$$\sum_{1}^{n-1} w_i B(e_i, e_j) = L(e_j) \tag{9}$$

Dostajemy układ n-1 równań, który możemy zapisać w postaci macierzowej

$$\begin{bmatrix} B(e_1, e_1) & \dots & B(e_{n-1}, e_1) \\ \vdots & \ddots & \vdots \\ B(e_1, e_{n-1}) & \dots & B(e_{n-1}, e_{n-1}) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_{n-1} \end{bmatrix} = \begin{bmatrix} L(e_1) \\ \vdots \\ L(e_{n-1}) \end{bmatrix}$$

Podstawiając do równania 5, możemy funkcję Φ jako

$$\Phi(x) = 5 - \frac{x}{3} + \sum_{1}^{n-1} w_i e_i \tag{10}$$

Część programistyczna

Kod w języku Julia

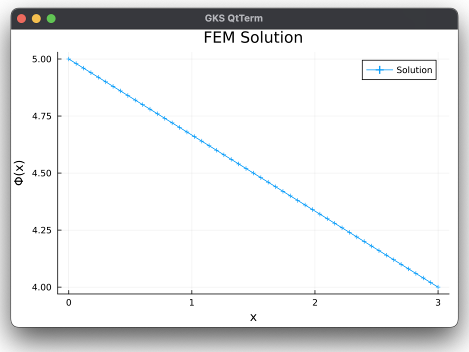
```julia

# FOR INSTALLING PLOTS PACKAGE FOR VISUALIZATION
# using Pkg
# Pkg.add("Plots")

using Plots

# CONSTANT VALUES

# constant G
const G = 6.67408e-11
# const G = 20 # for testing
# domain of the problem
const range = [0, 3]
# input of parameter for the number of divisions !!!
print("Enter the number of divisions: ")
const n = parse(Int64, readline()) # input
# distance between each division
const h = (range[2] - range[1])/n
# weights 1 and 2 for the Gauss Legendre Quadrature (GLQ) - 4 points formula
const w1 = (18 + sqrt(30))/36
# weights 3 and 4 for the Gauss Legendre Quadrature (GLQ) - 4 points formula
const w2 = (18 - sqrt(30))/36
# points 1 and 2 for the Gauss Legendre Quadrature (GLQ) - 4 points formula
const x12 = sqrt((3/7)-(2/7)*sqrt(6/5))
# point 3 for the Gauss Legendre Quadrature (GLQ) - 4 points formula
const x3 = sqrt((3/7)+(2/7)*sqrt(6/5))
# point 4 for the Gauss Legendre Quadrature (GLQ) - 4 points formula
const x4 = -sqrt((3/7)+(2/7)*sqrt(6/5))

# function rho based on the given data in the problem
function rho(x::Number)
    if x > 1 && x <= 2
        return 1
    else
        return 0
    end
end

# Auxiliary function to calculate the new x for
# the Gauss Legendre Quadrature (GLQ) - 4 points formula
function calculateNewX(x::Number, r1::Number, r2::Number)
```

```julia
    return (r2 - r1)/2 * x + (r1 + r2)/2
end


# Numerical Integration Gauss Legendre Quadrature (GLQ) - 4 points formula
function integral(f, r1::Number, r2::Number)
    # f is the function, r1 and r2 are the limits of integration
    # return the result of the integration
    return ((r2 - r1)/2 *
    (w1*f(calculateNewX(x12, r1, r2)) + w1*f(calculateNewX(x12, r1, r2))
    + w2*f(calculateNewX(x3, r1, r2)) + w2*f(calculateNewX(x4, r1, r2))))
end

function e_i(i::Int64)
    return function(x)
        if x > h*i && x < h*(i+1)
            return (h*(i+1) - x)/h
        elseif x > h*(i-1) && x <= h*i
            return (x - h*(i-1))/h
        else
            return 0
        end
    end
end

function e_i_prime(i::Int64)
    return function(x)
        if x > h*i && x < h*(i+1)
            return -1/h
        elseif x > h*(i-1) && x <= h*i
            return 1/h
        else
            return 0
        end
    end
end

function FEM(n)
    # Preparing matrices
    A = zeros(n-1, n-1)
    b = zeros(n-1)

    # Filling the matrix A

    # Filling diagonal
    f1(x) = e_i_prime(1)(x) * e_i_prime(1)(x)
```

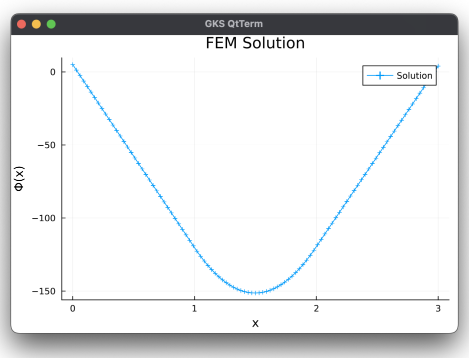
```
    tmp = integral(f1, h*0, h*2)
    for i in 1:n-1
        A[i, i] = -tmp
    end
    # Filling symetrical triangles
    for i in 1:n-2
        f2(x) = e_i_prime(i)(x)*e_i_prime(i+1)(x)
        tmp = -integral(f2, h*i, h*(i+1))
        A[i, i+1] = tmp
        A[i+1, i] = tmp
    end

    # Filling the vector b
    for i in 1:n-1
        f3(x) = e_i(i)(x) * rho(x)
        f4(x) = -(1/3) * e_i_prime(i)(x)
        b[i] = 4*pi*G*(integral(f3, h*(i-1), h*i) + integral(f3, h*i, h*(i+1)))
        + integral(f4, h*(i-1), h*i) + integral(f4, h*i, h*(i+1))
    end

    # Solving the system
    B = A\b

    # Function for the solution
    function solutionf(x)
        solution = 5 - (x/3)
        for i = 1:n-1
            solution += B[i]*e_i(i)(x)
        end
        return solution
    end

    # Plotting the solution
    X = [h*i for i in 0:n]
    Y = zeros(Float64, n+1)
    Y[1] = 5
    Y[n+1] = 4
    for i in 2:n
        Y[i] = solutionf(h*(i-1))
    end

    solutionPlot = plot(X, Y, label="Solution",
    title="FEM Solution", xlabel="x", ylabel="\Phi(x)",
    markershape = :auto, markersize = 3, legend=:topright)
    display(solutionPlot)
```

```
end

# Running the code
FEM(n)

# So the user can see the plot
print("Press enter to exit")
readline()
```

Otrzymane wykresy



Rysunek 1: Rozwiązanie przy n=50



Rysunek 2: Przypadek testowy dla G=20 przy n=100