Równanie Transportu Ciepła

Metoda Różnic Skończonych

Dominik Pilipczuk

Projekt obliczeniowy na przedmiot
Równania Różniczkowe i Różnicowe

Akademia Górniczo Hutnicza w Krakowie
21.01.2023

1 Wprowadzenie

Celem projektu było rozwiązanie równania różniczkowego, podanego przez prowadzącego, metodą elementów skończonych. Zadanym problemem było równanie transportu ciepła w postaci:

$$-k(x)\frac{d^2u(x)}{dx^2} = 0 \tag{1}$$

gdzie:

$$k(x) = \begin{cases} 1 & \text{dla } x \in [0,1] \\ 2 & \text{dla } x \in (1,2] \end{cases}$$

Samo równanie opisuje przepływ ciepła w układzie, uwzględniające różnice temperatury i właściwości cieplne materiałów. Jest ono ważne w wielu dziedzinach, takich jak inżynieria cieplna czy fizyka.

Wraz z samym równaniem podane były warunki brzegowe:

$$u(2) = 0 \tag{2}$$

oraz:

$$\frac{du(0)}{dx} - u(0) = 20 \tag{3}$$

Łatwo zauważyć, że warunek (2) jest warunkiem Dirichleta dla $x = 2$, zaś warunek (3) jest warunkiem Cauchy'ego. Ponadto, dana jest dziedzina funkcji $u(x)$:

$$[0,2] \ni x \to u(x) \in \mathbb{R}$$

2 Obliczenia

Przed wykorzystaniem narzędzi komputerowych do wyznaczenia funkcji u, należy uprościć równanie (1), obliczając wzór na składniki a i L:

$$-k \cdot u''(x) = 0$$

Dzielimy przez $-k(x) \neq 0$

$$u'' = 0$$

Wprowadzamy funkcję v, taką, że $v(2) = 0$, oraz całkujemy całe równanie:

$$\int_0^2 u'' \cdot v \, dx = 0$$

Całkując przez części:

$$[u' \cdot v]_0^2 - \int_0^2 u' \cdot v' \, dx = 0$$

$$u'(2) \cdot v(2) - u'(0) \cdot v(0) - \int_0^2 u' \cdot v' \, dx = 0$$

Z (3) wyznaczamy wzór na u', oraz odwracamy znaki:

$$\int_0^2 u' \cdot v' \, dx + 20 \cdot v(0) - u(0) \cdot v(0) = 0$$

$$\int_0^2 u' \cdot v' \, dx - u(0) \cdot v(0) = -20 \cdot v(0)$$

Wyznaczamy $a(u, v)$ oraz $L(v)$:

$$a(u, v) = \int_0^2 u' \cdot v' \, dx - u(0) \cdot v(0) \tag{4}$$

$$L(v) = -20 \cdot v(0) \tag{5}$$

3 Implementacja

Implementacji algorytmu MRS dokonałem w języku Rust, ponieważ zapewnia on bezpieczeństwo, wydajność i elastyczność.

Przy pisaniu programu wspomagałem się trzema bibliotekami odpowiedzialnymi za: wyliczanie całek, rozwiązanie układu macierzy oraz za stworzenie wykresu.

3.1 Funkcje pomocnicze

Zadeklarowałem parę pomocniczych funkcji, które po krótce opiszę.

```
fn get_a(u_d: impl Fn(f64) -> f64,
        v_d: impl Fn(f64) -> f64,
        u: impl Fn(f64) -> f64,
        v: impl Fn(f64) -> f64,
        a: f64, b: f64) -> f64 {

    let quad: GaussLegendre =
                GaussLegendre::init(4);

    quad.integrate(a, b,
                |x| u_d(x)*v_d(x)) - u(0.)*v(0.)
}
```

get_a - zwraca $a(u, v)$

```
fn get_l(v: impl Fn(f64) -> f64) -> f64 {
        -20.0*v(0.0)
}
```

get_l - zwraca $L(v)$

```
fn u_i(i_: usize) -> impl Fn(f64) -> f64{
        let n: f64 = N as f64;
        let i = i_ as f64;
        move |x| {
                if x > x_i(i - 1.) && x <= x_i(i) {
                        return n/2.0*x - i + 1.0
                }
                if x > x_i(i) && x < x_i(i+1.) {
                        return -n/2.0*x + i + 1.0
                };
                return 0.0;
        }
}
```

u_i - zwraca $u_i(x)$

```
fn ud_i(i_: usize) -> impl Fn(f64) -> f64 {
        let n = N as f64;
        let i = i_ as f64;
        move |x| {
                if x > x_i(i-1.) && x <= x_i(i) {
                        return n/2.0;
                }
                if x > x_i(i) && x < x_i(i+1.) {
                        return -n/2.0;
                };
                return 0.0;
        }
}
```

ud_i - zwraca $u_i'(x)$

```
fn x_i(i: f64) -> f64 {
        2.0*(i as f64)/(N as f64)
}
```

x_i - zwraca x_i

Ponadto, znajdują się funkcję tworzące wykres, w które szczegóły nie będę się zagłębiał.

3.2 Funkcja główna

```
fn main() {
  // Macierz wypelniona zerami
  let mut a: Array2<f64> =
    Array2::<f64>::zeros((N+1, N+1));

  let n = N as f64;
  // Wypelniamy macierz zgodnie z
  // przykladem na zajeciach
  for i in 0..N {
    for j in 0..=N {
      let s: f64;
      let e: f64;
      let diff = i.abs_diff(j);
      if diff > 1 { continue; }
      if diff == 1 {
        s = 2. * f64::max(0.,
          f64::min(i as f64, j as f64) / n);

        e = 2. * f64::min(1.,
          f64::max(i as f64, j as f64) / n);
      } else {
        s = 2. * f64::max(0., (i as f64 - 1.) / n);
        e = 2. * f64::min(1., (i as f64 + 1.) / n);
            }


      a[[i, j]] = get_a(ud_i(j), ud_i(i),
        u_i(j), u_i(i), s, e);
    }
  }

  a[[N, N]] = 1.;

  // Macierz B
  let mut b: Array1<f64> =
    Array1::<f64>::zeros(N+1);
```

```

  for i in 0..N {
    b[i] = get_l(u_i(i));
  }
  b[N] = 0.;

  // Rozwiazujemy uklad macierzy
  let res = a.solve_into(b).unwrap();

  // tworzymy punkty
  let x: Vec<f64> = (0..=2000)
  .map(|x| x as f64*0.001).
  collect::<Vec<f64>>();

  let mut y = vec![0f64; x.len()];

  for i in 0..x.len() {
    for j in 0..res.len() {
      let e = u_i(j);
      y[i] = y[i] + res[j] * e(x[i])
    }
  }

  // rysujemy wykres
  plot(x, y);
}
```

Funckja *main*() działa analogicznie do przykładów podanych na zajęciach. Tworzy i wypełnia macierz A i B po czym rozwiązuje układ równań $A \cdot X = B$ i ukazuje wyniki na wykresie.

4 Wyniki