Równania nieliniowe - projekt

Natalia Wojtania i Grzegorz Chojnacki

17 grudnia 2020

1 Zadanie

1.1 Tytuł

Tytuł zadania to "Przybliżone rozwiązywanie równań nieliniowych".

1.2 Treść

Napisz program, który rozwiązuje równanie:

$$4x^3 + 5x^2 + 6x - 7 = 0$$

metodą siecznych.

1.3 Metoda

W programie należy wykorzystać metodę siecznych.

1.3.1 Opis metody

Metoda siecznych (interpolacji liniowej) polega na przyjęciu, że funkcja ciągła na dostatecznie małym odcinku w przybliżeniu zmienia się w sposób liniowy. Można wtedy na odcinku $[a, b]$ krzywą $y = f(x)$ zastąpić sieczną. Za przybliżoną wartość pierwiastka przyjmuje się punkt przecięcia siecznej z osią odciętych OX. Miejsce przecięcia tej prostej z osią x jest przybliżonym wynikiem szukanego miejsca zerowego, o ile różnica bezwzględna wartości z dwóch ostatnich iteracji jest mniejsza od założonej dokładności. Metoda ta wymaga ustalenia na przedziale $[a, b]$ dwóch punktów startowych x_0 i x_1.

Metodę siecznych dla funkcji $f(x)$, mającej pierwiastek w przedziale $[a, b]$ można zapisać następującym wzorem rekurencyjnym:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, k \geqslant 1$$

i

$$x_0 = a, x_1 = b,$$

gdzie w każdym kroku x_{k+1} to miejsce zerowe siecznej wykresu $y = f(x)$ w punktach $(x_{k-1}, f(x_{k-1}))$ oraz$(x_k, f(x_k))$, czyli prostej

$$y = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k) + f(x_k)$$

1.3.2 Przykład

Dla równania $x^3 - 2x - 5 = 0$ rozważanego na przedziale $[a, b] = [2, 3]$ przybliżyć rozwiązanie wartością x_k wyznaczoną metodą siecznych z
$x_0 = a, x_1 = b$ oraz dokładnością $\epsilon = 0.5$.

$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$

Mamy $x_0 = 2$, i $x_1 = 3, f(x_1) = f(3) = 16, f(x_0) = f(2) = -1$
Stąd $x_2 = 3 - \frac{16(3-2)}{16-(-1)} = 2\frac{1}{17}, |x_2 - x_1| = 0,94 > \epsilon$ $f(x_2) = f(2\frac{1}{17}) = -0.39, x_3 = 2\frac{1}{17} - \frac{-0.39(-0,94)}{-0.39-16} = 2.08, |x_3 - x_2| = 0,94 < \epsilon$
Zatem przybliżone rozwiązanie $x_k = 2\frac{1}{17}$ oraz $k = 3$!!!!!

2 Opis implementacji algorytmu

Implementacja realizująca metodę siecznych.

2.1 Dane wejściowe

Na wejściu program pobiera od użytkownika wartość wyrażającą dokładność rozwiązania(epsilon) $\epsilon \in (0, 1)$.

2.2 Przebieg działania

Program wyświetla komunikat: 'Wprowadź dokładność rozwiązania $\epsilon \in (0, 1)$ '. Jeśli została wprowadzona prawidłowa wartość dokładności, to program poprzez funkcję calculate wylicza przybliżone rozwiązanie i dzięki funkcji refresh wyświetla je wraz z liczbą kroków. Próba wprowadzenia nieprawidłowych danych, które weryfikowane są w programie w funkcji 'refresh' skutkuje wyświetleniem stosownego ostrzeżenia.

Następnie funkcja *calculate* klasy *SecantMethod* zajmuje się wyliczeniem przybliżonego rozwiązania.
Funkcja *getNext*, której argumentami są a i b odpowiednio oznaczające x_{k-1} oraz x_k zwraca wartość poszczególnego x_{k+1}.
Funkcja *isGoodEnough* sprawdza czy różnica $|x_k - x_{k-1}|$ jest mniejsza od podanej przez użytkownika dokładności. Jeśli tak, to kończymy przekazując wynik oraz ilość kroków. W przeciwnym wypadku liczone jest kolejne przybliżenie tak długo, aż warunek zostanie spełniony.

Wynikiem działania programu jest przybliżone rozwiązanie równania: x_k oraz liczba wykonanych kroków: k.

2.3 Najważniejsze fragmenty programu

newtonEvaluator.js

```
class NewtonEvaluator {
  constructor(points) { this.points = points }

  getPolynomial() {
    if (this.points.length === 1) return new Polynomial([this.points[0].y])

    const [b0, ...bs] = this.getBs()
    const polynomials = init(this.points)
      .map(Polynomial.point)
      .map(getListSlicesFromStart)
      .map(group => group.reduce(Polynomial.product))

    return polynomials
      .map((polynomial, index) => polynomial.multiply(bs[index]))
      .reduce(Polynomial.sum)
      .add(b0)
  }

  getBs() {
    const pointProduct = (points) => points
      .map(Polynomial.point)
      .reduce(Polynomial.product)

    const P = memoized((points) => {
      if (points.length === 1) return new Polynomial([points[0].y])
      else {
        const rest = init(points)
        return P(rest).add(pointProduct(rest).multiply(getB(points)))
      }
    })

    const getB = memoized((points) => {
      if (points.length === 1) return points[0].y
      else {
```

```
        const [current, rest] = [last(points), init(points)]
        return (current.y - P(rest).at(current.x)) /
         pointProduct(rest).at(current.x)
      }
    })

    return this.points.map(getListSlicesFromStart).map(getB)
  }
}
```

polynomial.js

```
class Polynomial {
  static one  = new Polynomial([1])
  static zero = new Polynomial([0])
  static point = (p) => new Polynomial([-p.x, 1])
  static product = (acc, polynomial) => acc.multiply(polynomial)
  static sum     = (acc, polynomial) => acc.add(polynomial)

  terms = []

  constructor(terms) {
    const trimmed = terms => {
      if (terms.length === 0) return [0]
      else if (last(terms) !== 0) return terms
      else return trimmed(init(terms))
    }

    if (terms.length === 0) throw new Error('Term list is empty')
    else this.terms = trimmed(terms)
  }

  at(x) {
    return this.terms.reduceRight((acc, term) => acc * x + term)
  }

  add(that) {
```

```
    if (typeof that == 'number') return this.addNumber(that)
    else return this.addPolynomial(that)
  }

  addNumber(that) {
    const [head, ...tail] = this.terms
    return new Polynomial([head + that, ...tail])
  }

  addPolynomial(that) {
    const [longer, shorter] = (this.terms.length >= that.terms.length)
    ? [this, that]
    : [that, this]


    const addedTerms = zip(longer.terms, shorter.terms).map(pairSum)

    return new Polynomial(addedTerms)
  }

  multiply(that) {

    const padLeft = (arr, padding) => new Array(padding).fill(0).concat(arr)

    const multiplyByTerm = (thisTerm, power, that) => {
      const multiplied = that.terms.map(thatTerm => thatTerm * thisTerm)
      return padLeft(multiplied, power)
    }

    if (typeof that === "number") return this.multiply(new Polynomial([that]))
    else return this.terms
      .map((term, power) => multiplyByTerm(term, power, that))
      .map(terms => new Polynomial(terms))
      .reduce(Polynomial.sum)
  }

}
```

2.4 Widok działania programu