БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа № 2
«Интерполяционный кубический сплайн»

Выполнил
Студент 11 группы
Сергиенко Лев Эдуардович

Минск, 2024

Алгоритм построения интерполяционного кубического сплайна

1. Построение равноотстоящих узлов

$$x_i = a + i\frac{b-a}{n}, i = \overline{0,n}$$, где n = 15, a = -3, b = 3

2. Составление СЛАУ

$$\frac{h_i}{6}M_{i-1} + \frac{h_i + h_{i+1}}{3}M_i + \frac{h_{i+1}}{6}M_{i+1} = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}, i = \overline{1, N-1}$$

3. Дополнительные условия

$$\frac{h_1}{3}M_0 + \frac{h_1}{6}M_1 = \frac{f_1 - f_0}{h_1} - f_0'; \frac{h_N}{6}M_{N-1} + \frac{h_N}{3}M_N = f_N' - \frac{f_N - f_{N-1}}{h_N}$$

4. Решение СЛАУ методом прогонки

$$d_{i+1} = e_i * \left(-\frac{c_i}{d_i}\right), y_{i+1} = y_i * \left(-\frac{c_i}{d_i}\right), c_i = 0,\ i = \overline{0..N-1}$$

$$y_{i-1} = y_i * \left(-\frac{e_{i-1}}{d_i}\right), e_{i-1} = 0, i = \overline{N..1}$$

5. Формула кубического сплайна

$$S_{3,i} = \frac{M_{i-1}(x_i - x)^3}{6h} + \frac{M_i(x - x_{i-1})^3}{6h} + \left(f_{i-1} - \frac{h^2}{6}M_{i-1}\right)\frac{(x_i - x)}{h} + \left(f_i - \frac{h^2}{6}M_i\right)\frac{(x - x_{i-1})}{h},$$
$$i = \overline{1..N}$$

Погрешность интерполяции

$$max_{i=0}^{100}|S_3(x_i) - f_1(x_i)| = 0.000068203841748$$

Графики функции f1(x) и интерполяционного кубического сплайна S3(x)

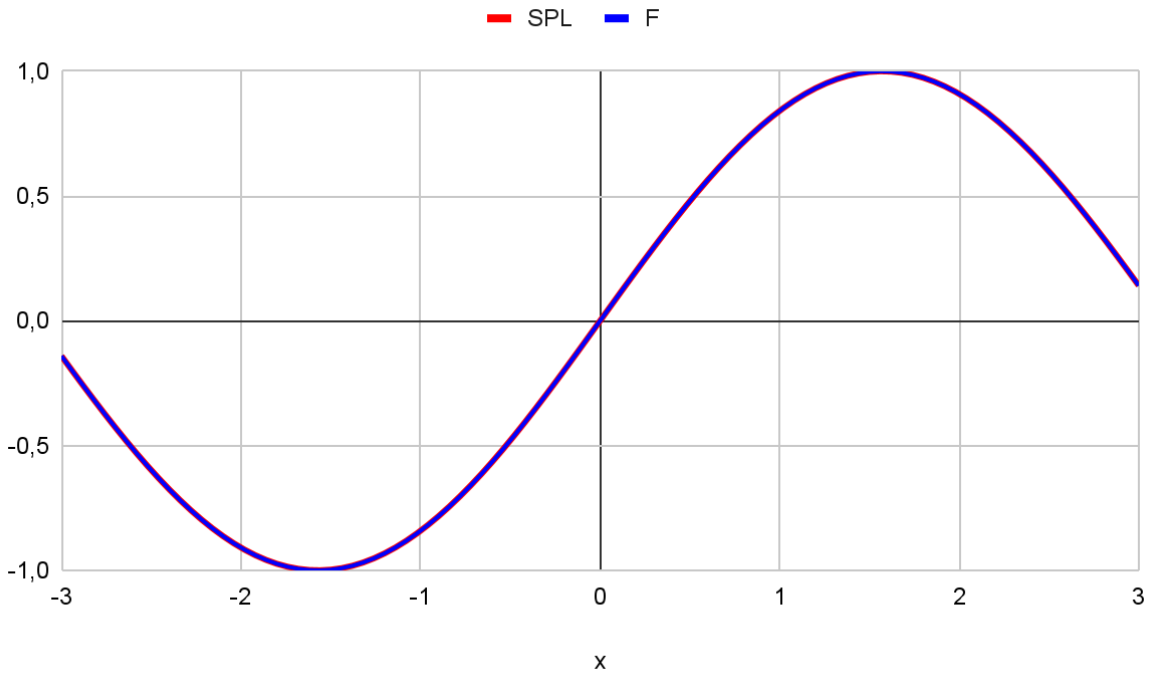
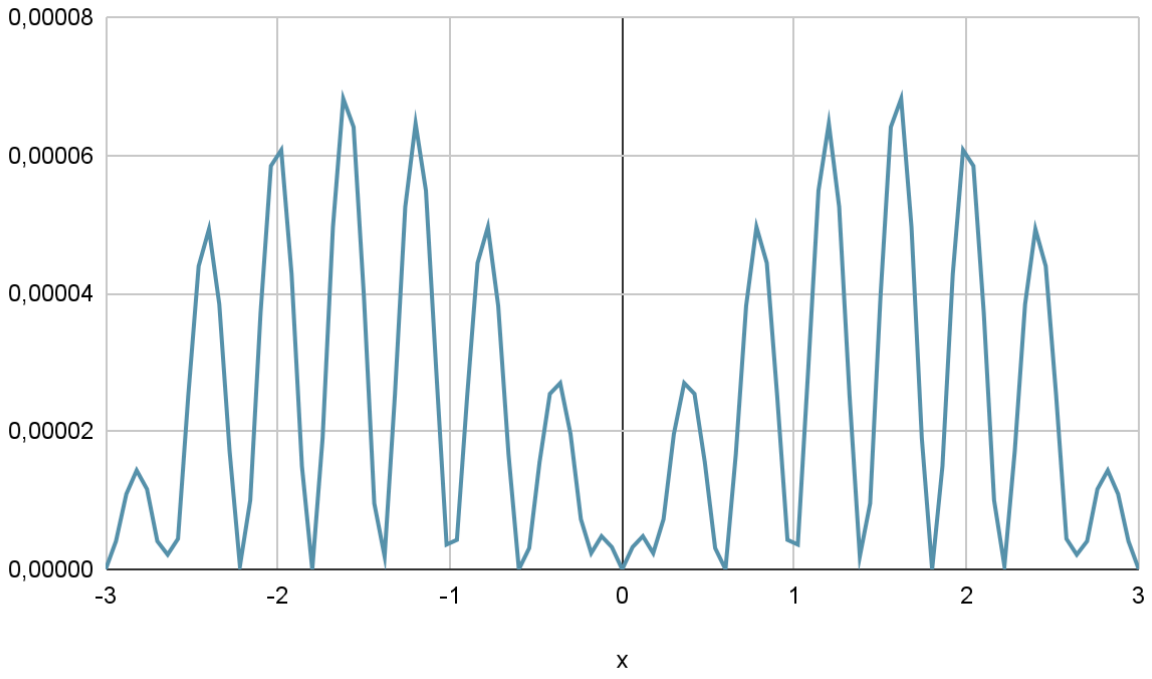


График погрешности интерполирования кубическим сплайном



Выводы

В результате проведенной работы я построил кубический сплайн для интерполяции функции sin(x), используя равноотстоящие узлы.

Полученная погрешность интерполяции составила всего лишь 0.0000682. Это говорит о том, что сплайн очень близок к исходной функции в выбранных точках.

Результаты говорят о том, что использованный кубический сплайн хорошо интерполирует функцию sin(x) на равноотстоящих узлах с небольшой погрешностью.

Листинг программы с комментариями

Структура программы:

```
├── internal/
│   ├── equationsSolving/
│   │   └── tridiagonal.go
│   ├── spline/
│   │   └──cubicSpline.go
│   └── node/
│       ├── builder.go
│       └── node.go
└── main.go
```

tridiagonal.go

```go
package equationsSolving

// SolveTridiagonal решает систему линейных уравнений с трехдиагональной матрицей
// методом прогонки. Принимает на вход матрицу коэффициентов A и столбец свободных членов B.
// Возвращает вектор решений x.
func SolveTridiagonal(A, B [][]float64) []float64 {
  // Получаем размерность системы
  n := len(B)

  // Инициализируем временные массивы для коэффициентов
  a := make([]float64, n)
  b := make([]float64, n)

  // Инициализируем вектор решений
  x := make([]float64, n)

  // Начальные значения прогоночных коэффициентов
  a[0] = A[0][0]
  b[0] = B[0][0]

  // Прямой ход метода прогонки
  for i := 1; i < n; i++ {
    // Вычисляем прогоночные коэффициенты
```

```
    a[i] = A[i][i] - (A[i][i-1]*A[i-1][i])/a[i-1]
    b[i] = B[i][0] - (A[i][i-1]*b[i-1])/a[i-1]
  }

  // Обратный ход метода прогонки
  x[n-1] = b[n-1] / a[n-1]
  for i := n - 2; i >= 0; i-- {
    // Вычисляем значения переменных на обратном ходе
    x[i] = (b[i] - A[i][i+1]*x[i+1]) / a[i]
  }

  return x
}
```

cubicSpline.go

```
package spline

import (
  "errors"
  "mv2.2/internal/equationsSolving"
  "mv2.2/internal/node"
)

// CubicSpline представляет кубическую сплайн-интерполяцию для заданного набора узлов.
type CubicSpline struct {
  nodes  []node.Node // Слайс узлов для интерполяции.
  coeffs []float64   // Коэффициенты для кубической сплайн-интерполяции.
}

// New создает новую кубическую сплайн-интерполяцию на основе заданной производной и набора узлов.
func New(Df func(float64) float64, nodes []node.Node) CubicSpline {
  // Создаем слайсы для матриц A и B для решения СЛАУ.
  var (
    A = make([][]float64, len(nodes))
    B = make([][]float64, len(nodes))
  )

  // Заполняем матрицы A и B для всех узлов, кроме первого и последнего.
  for i := 1; i < len(nodes)-1; i++ {
    A[i] = make([]float64, len(nodes))
    B[i] = make([]float64, 1)

    A[i][i-1] = (nodes[i].X - nodes[i-1].X) / 6
    A[i][i] = (nodes[i+1].X - nodes[i-1].X) / 3
```

```
    A[i][i+1] = (nodes[i+1].X - nodes[i].X) / 6

    B[i][0] = (nodes[i+1].Y-nodes[i].Y)/(nodes[i+1].X-nodes[i].X) -
(nodes[i].Y-nodes[i-1].Y)/(nodes[i].X-nodes[i-1].X)
  }

  // Заполняем матрицы A и B для первого узла.
  A[0] = make([]float64, len(nodes))
  B[0] = make([]float64, 1)

  A[0][0] = (nodes[1].X - nodes[0].X) / 3
  A[0][1] = A[0][0] / 2
  B[0][0] = (nodes[1].Y-nodes[0].Y)/(nodes[1].X-nodes[0].X) - Df(nodes[0].X)

  // Заполняем матрицы A и B для последнего узла.
  A[len(nodes)-1] = make([]float64, len(nodes))
  B[len(nodes)-1] = make([]float64, 1)

  A[len(nodes)-1][len(nodes)-2] = (nodes[len(nodes)-1].X - nodes[len(nodes)-2].X) / 6
  A[len(nodes)-1][len(nodes)-1] = A[len(nodes)-1][len(nodes)-2] * 2
  B[len(nodes)-1][0] = Df(nodes[len(nodes)-1].X) -
(nodes[len(nodes)-1].Y-nodes[len(nodes)-2].Y)/(nodes[len(nodes)-1].X-nodes[len(no
des)-2].X)

  // Решаем СЛАУ с помощью метода прогонки и возвращаем
результирующую кубическую сплайн-интерполяцию.
  return CubicSpline{
    nodes:  nodes,
    coeffs: equationsSolving.SolveTridiagonal(A, B),
  }
}

// Solve вычисляет значение кубического сплайна в точке x.
func (cs CubicSpline) Solve(x float64) (float64, error) {
  // Ищем индекс интервала, в который попадает x
  i := -1
  for k := 1; k < len(cs.nodes); k++ {
    if x <= cs.nodes[k].X {
      i = k
      break
    }
  }

  // Проверяем, что x находится в пределах заданных узлов данных
  if i == -1 {
```

```
    return 0, errors.New("the argument is out of range")
  }

  // Расчет значений для кубического сплайна
  h := cs.nodes[i].X - cs.nodes[i-1].X
  result :=
cs.coeffs[i-1]*(cs.nodes[i].X-x)*(cs.nodes[i].X-x)*(cs.nodes[i].X-x)/(6*h)+cs.coeffs[i]*(x-cs
.nodes[i-1].X)*(x-cs.nodes[i-1].X)*(x-cs.nodes[i-1].X)/(6*h) +
(cs.nodes[i-1].Y-h*h/6*cs.coeffs[i-1])*(cs.nodes[i].X-x)/h +
(cs.nodes[i].Y-h*h/6*cs.coeffs[i])*(x-cs.nodes[i-1].X)/h

  return result, nil
}
```

builder.go

```
package node

import (
  "math"
)

// BuildEquidistantNodes строит равноотстоящие узлы
// f - интерполируемая функция, a и b - границы интервала, n - степнь
многочлена
// Возвращает массив узлов Node, представляющих точки интерполяции
func BuildEquidistantNodes(f func(float64) float64, a float64, b float64, n int)
[]Node {
  // Создаем слайс для хранения узлов
  nodes := make([]Node, n+1)
  // Вычисляем шаг между узлами
  h := (b - a) / float64(n)

  // Заполняем массив узлами, где X - равномерно распределенные точки, Y -
значение функции в этих точках
  for i := 0; i <= n; i++ {
    x := a + float64(i)*h
    nodes[i] = Node{X: x, Y: f(x)}
  }

  // Возвращаем массив узлов для использования при интерполяции
  return nodes
}

// BuildChebyshevNodes строит чебышѐвские узлы
// Возвращает массив узлов Node, представляющих точки интерполяции
```

```
func BuildChebyshevNodes(f func(float64) float64, a float64, b float64, n int)
[]Node {
  // Создаем слайс для хранения узлов
  nodes := make([]Node, n+1)

  // Заполняем массив узлами, где X - точки, определенные методом
Чебышева
  for i := 0; i <= n; i++ {
    x := (a+b)/2 + (b-a)/2*math.Cos(math.Pi*(2*float64(i)+1)/(2*float64(n)+2))
    nodes[i] = Node{X: x, Y: f(x)}
  }

  return nodes
}
```

node.go

```
package node

// Node представляет узел для интерполяции в двумерном пространстве.
type Node struct {
  X float64 // Координата X узла в пространстве для интерполяции.
  Y float64 // Координата Y узла в пространстве для интерполяции.
}
```

main.go

```
package main

import (
  "fmt"
  "math"
  "mv2.2/internal/node"
  "mv2.2/internal/spline"
  "os"
)

const (
  a      = -3.0
  b      = 3.0
  N      = 15
  points = 100
)

var (
  f = func(x float64) float64 {
    return math.Sin(x)
  }
```

```
  Df = func(x float64) float64 {
    return math.Cos(x)
  }
)

func main() {
  spl := spline.New(Df, node.BuildEquidistantNodes(f, a, b, N))

  saveToFile("spline", spl, f)
}

// saveToFile сохраняет результаты интерполяции в файл и вычисляет
погрешность
func saveToFile(filename string, spl spline.CubicSpline, f func(float64) float64) {
  file, err := os.Create(filename)
  if err != nil {
    fmt.Println("Error creating file:", err)
    return
  }
  defer file.Close()

  // Вычисление шага для точек интерполяции
  step := (b - a) / points
  interErr := 0.0

  // Запись точек интерполяции и реальных значений функции в файл
  for i := 0; i <= points; i++ {
    x := a + float64(i)*step
    y, err := spl.Solve(x)

    if err != nil {
      panic(err)
    }

    yReal := f(x)

    // Обновление максимальной погрешности
    interErr = math.Max(interErr, math.Abs(yReal-y))

    //Запись в файл в формате "x y интерполяция y реальное значение"
    _, err = file.WriteString(fmt.Sprintf("%.2f %.10f %.10f %.15f\n", x, y, yReal,
math.Abs(y-yReal)))
    if err != nil {
      fmt.Println("Error writing to file:", err)
      return
```

```
    }
  }

  //Запись погрешности интерполяции в файл
  _, err = file.WriteString(fmt.Sprintf("%.15f", interErr))
  if err != nil {
    fmt.Println("Error writing to file:", err)
    return
  }
}
```