Московский авиационный институт
(Национальный Исследовательский Институт)

Институт №8 информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу по курсу «Численные методы» на тему «Интерполяция экспоненциальными сплайнами»

Студент: Суханов Е.А

Группа: М8О-406Б-19

Оценка:

Преподаватель: Ревизников Д.Л.

Подпись:

Москва
2022

Описание

Нужно реализовать интерполяцию экспоненциальными сплайнами. Ограничимся интерполяцией только монотонных функций.

Основное отличие от обычного кубического сплайна в том, что вместо полинома используется экспонента следующего вида $C+B\exp(Ax)$, где нам нужно определить коэффициенты.

Строить экспоненты я буду по трем точкам $(x_1; y_1),(x_2; y_2),(x_3; y_3)$.

Имеем систему уравнений (1):

$$y_1=C+B\exp(Ax_1)$$

$$y_2=C+B\exp(Ax_2)$$

$$y_3=C+B\exp(Ax_3)$$

Если аргументы находятся на одинаковом расстоянии друг от друга, то я использую следующие формулы для нахождения коэффициентов:

$$A=\frac{\ln\frac{y_3-y_2}{y_2-y_1}}{x_3-x_2} \quad C=\frac{y_2^2-y_1y_3}{2*y_2-y_1-y_3} \quad B=(y_1-C)\exp(-Ax_1)$$

Эти формулы выводятся из системы уравнений (1) при условии, что аргументы находятся на равном расстоянии друг от друга.

В случае, если точки находятся на произвольном расстоянии друг от друга, то придется находить коэффициент А численно. Я буду использовать метод касательных. Остальные коэффициенты выводятся из системы уравнений.

Стоит обратить внимание, что данный метод интерполяции будет работать только для монотонных функций. Кроме этого, здесь есть проблема склейки сплайнов. А именно, не соблюдается непрерывность производной в узлах интерполяции.

Что бы решить эту проблему, я использовал функцию склейки, которая гарантирует непрерывность производной в узлах интерполяции.

Она выглядит следующим образом:

$$\frac{(x_2-x)F_1(x)+(x-x_1)F_2(x)}{x_2-x_1}$$

Где F – это конфликтные экспоненциальные сплайны.

Сравнение с кубическим сплайном

Интерполяция экспоненциальными сплайнами дает лучший результат для данных, которые имеют экспоненциальную природу. Например какие-нибудь физические наблюдения.

Основное преимущество перед кубическим заключается в избавлении от перегибов. Это особенно заметно, когда у нас мало интерполяционных узлов.

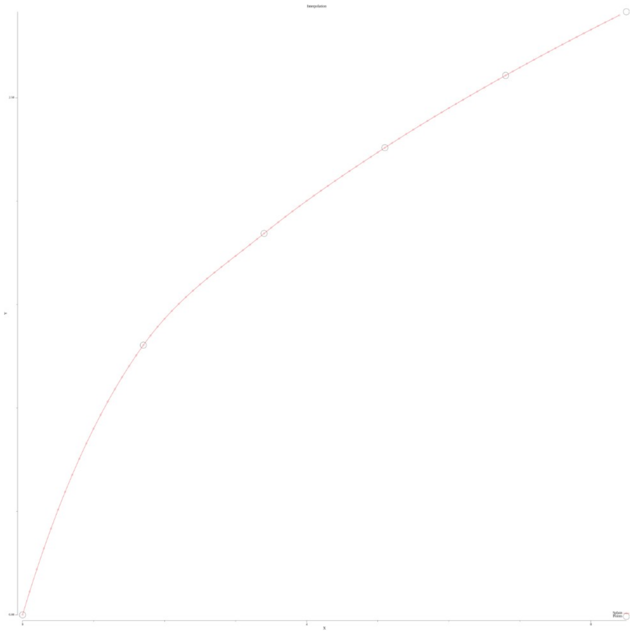

Рис 1: Интерполяция экспоненциальными сплайнами

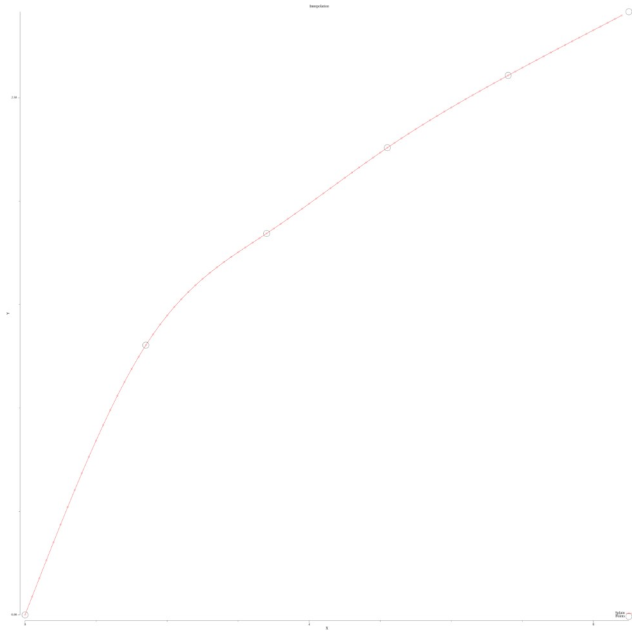

Рис 2: Интерполяция кубическими сплайнами

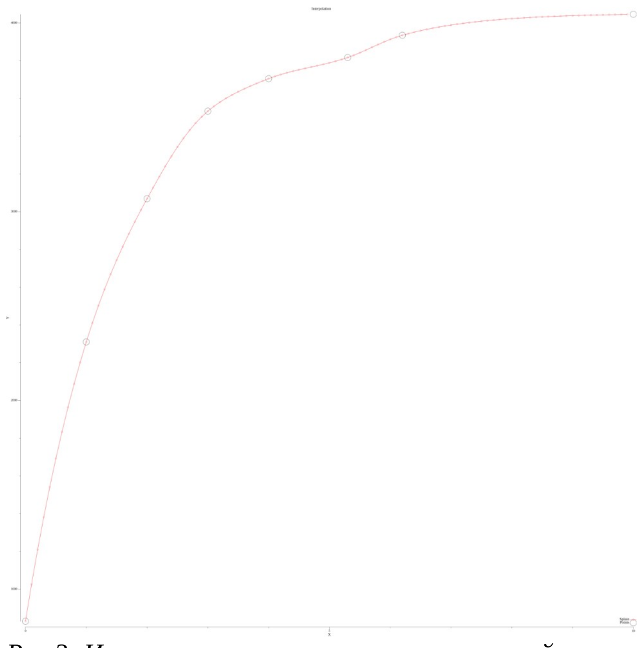

Рис 3: Интерполяция экспоненциальными сплайнами

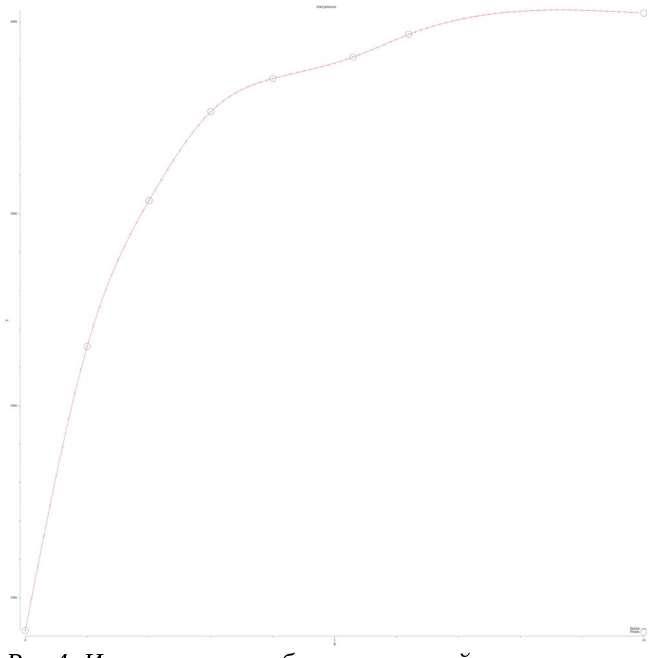

Рис 4: Интерполяция кубическими сплайнами

Листинг

За основу я взял лабораторную работу по сплайнам.

main.go:

```go
package main

import (
  "encoding/csv"
  "fmt"
  "math"
  "os"

  "github.com/Reterer/number_methods/internal/run_through"
  "github.com/Reterer/number_methods/internal/utils"
  "github.com/Reterer/number_methods/pkg/matrix"
  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
)

type Point struct {
  x, y float64
}

type ftype func(float64) float64

func MakeSplainInterpolation(points []Point) func(float64) float64 {
  n := len(points) - 1

  c := make([]float64, n)
  {
    mat := matrix.MakeRealMatrix(n-1, n-1)
    b := matrix.MakeRealMatrix(n-1, 1)
    for i := 0; i < n-1; i++ {
      hc := points[i+2].x - points[i+1].x
      hp := points[i+1].x - points[i].x

      if i > 0 {
        mat.SetEl(i, i-1, hp)
      }

      mat.SetEl(i, i, 2*(hp+hc))

      if i < n-2 {
        mat.SetEl(i, i+1, hp)
      }

      fc := points[i+2].y - points[i+1].y
      fp := points[i+1].y - points[i].y
```

```
      b.SetEl(i, 0, 3*(fc/hc-fp/hp))

    }

    utils.PrintMatrix(mat)
    utils.PrintMatrix(b)
    c_2n := run_through.Do(mat, b)
    utils.PrintMatrix(c_2n)

    for i := 0; i < n-1; i++ {
      c[i+1] = c_2n.GetEl(i, 0)
    }
  }

  a := make([]float64, n)
  for i := 0; i < n; i++ {
    a[i] = points[i].y
  }

  b := make([]float64, n)
  for i := 0; i < n-1; i++ {
    fcurr := points[i+1].y - points[i].y
    hcurr := points[i+1].x - points[i].x
    b[i] = fcurr/hcurr - 1./3.*hcurr*(c[i+1]+2*c[i])
  }
  b[n-1] = (points[n].y-points[n-1].y)/(points[n].x-points[n-1].x) -
2./3.*(points[n].x-points[n-1].x)*c[n-1]

  d := make([]float64, n)
  for i := 0; i < n-1; i++ {
    hcurr := points[i+1].x - points[i].x
    d[i] = (c[i+1] - c[i]) / (3 * hcurr)
  }
  d[n-1] = -c[n-1] / (3 * (points[n].x - points[n-1].x))

  fmt.Println("A: ", a)
  fmt.Println("B: ", b)
  fmt.Println("C: ", c)
  fmt.Println("D: ", d)

  return func(x float64) float64 {
    // find interval
    i := 0
    for ; points[i+1].x < x; i++ {
    }
    dx := x - points[i].x
    return a[i] + b[i]*dx + c[i]*dx*dx + d[i]*dx*dx*dx
  }
}

func expfunc(a, b, c, x float64) float64 {
```

```go
	return c + b*math.Exp(a*x)
}

func MakeExpInterpolation(points []Point) func(float64) float64 {
	eps := 0.0000001
	n := len(points) - 2
	// Вместо полинома будем использовать экспоненту вида y = C + B*exp(A*x)

	c := make([]float64, n)
	a := make([]float64, n)
	b := make([]float64, n)

	for i := 0; i < n; i++ {
		y1 := points[i].y
		y2 := points[i+1].y
		y3 := points[i+2].y
		x1 := points[i].x
		x2 := points[i+1].x
		x3 := points[i+2].x

		if math.Abs(x2-x1-x3+x2) < eps {
			// Если точки на равном расстоянии
			z := (2*y2 - y1 - y3)
			c[i] = (y2*y2 - y1*y3) / z
			r := (y3 - y2) / (y2 - y1)
			a[i] = math.Log(r) / (x3 - x2)
			b[i] = (y1 - c[i]) * math.Exp(-a[i]*x1)
		} else {
			// Иначе используем более сложный метод нахождения коэф.
			dif := (y3 - y2) * (x2 - x1) / ((y2 - y1) * (x3 - x2))
			Amin := math.Log(dif) / (x3 - x1)
			A0 := 2 * Amin

			for n := 10; n > 0; n-- {
				u := math.Exp(A0 * (x3 - x2))
				v := math.Exp(-A0 * (x2 - x1))
				F := (y2-y1)*(u-1) + (y3-y2)*(v-1)
				FF := (y2-y1)*(x3-x2)*u - (y3-y2)*(x2-x1)*v
				dA := -F / FF
				A0 = A0 + dA
				if math.Abs(dA/Amin) < eps {
					break
				}
			}
			a[i] = A0
			b[i] = (y1 - y2) / (math.Exp(A0*x1) - math.Exp(A0*x2))
			c[i] = y1 - b[i]*math.Exp(A0*x1)
		}
	}

	return func(x float64) float64 {
```

```
    // find interval
    i := 0
    for ; points[i].x < x; i++ {
    }

    // Граничные случаи
    if i <= 1 {
      return c[0] + b[0]*math.Exp(a[0]*x)
    } else if i >= len(points)-1 {
      return c[n-1] + b[n-1]*math.Exp(a[n-1]*x)
    }

    // Будем использовать склеивание функций
    // с сохранением непрерывности первой производной
    f1 := i - 2
    f2 := i - 1
    x1 := points[i-1].x
    x2 := points[i].x

    g := ((x2-x)*expfunc(a[f1], b[f1], c[f1], x) + (x-x1)*expfunc(a[f2], b[f2],
c[f2], x)) / (x2 - x1)
    return g
  }
}

func readFromFile(filePath string) []Point {
  f, err := os.Open(filePath)
  if err != nil {
    panic("Unable to read input file " + filePath + " " + err.Error())
  }
  defer f.Close()

  csvReader := csv.NewReader(f)
  records, err := csvReader.ReadAll()
  if err != nil {
    panic("Unable to parse file as CSV for " + filePath + " " + err.Error())
  }

  points := make([]Point, len(records))
  for i := 0; i < len(records); i++ {
    _, err := fmt.Sscanf(records[i][0], "%f", &points[i].x)
    if err != nil {
      panic(err.Error())
    }
    _, err = fmt.Sscanf(records[i][1], "%f", &points[i].y)
    if err != nil {
      panic(err.Error())
    }
  }
  return points
}
```

```go
func genPlot(path string, sf ftype, points []Point, a float64, b float64, h
float64) {
  p := plot.New()

  p.Title.Text = "Interpolation"
  p.X.Label.Text = "X"
  p.Y.Label.Text = "Y"

  steps := int((b - a) / h)
  s_p := make(plotter.XYs, steps)
  x := a
  for step := 0; step < steps; step++ {
    s_p[step].X = x
    s_p[step].Y = sf(x)

    x += h
  }
  err := plotutil.AddLinePoints(p,
    "Splain", s_p)
  if err != nil {
    panic(err)
  }

  // Scatter
  scatter_data := make(plotter.XYs, len(points))
  for i := 0; i < len(points); i++ {
    scatter_data[i].X = points[i].x
    scatter_data[i].Y = points[i].y
  }
  s, err := plotter.NewScatter(scatter_data)
  if err != nil {
    panic(err)
  }
  s.GlyphStyle.Radius = 10
  p.Add(s)
  p.Legend.Add("Points", s)
  // Save the plot to a PNG file.
  if err := p.Save(2000, 2000, path); err != nil {
    panic(err)
  }
}

func main() {
  if len(os.Args) < 2 {
    panic("Аргументов должно быть два")
  }
  inputFile := os.Args[1]
  outputFile := os.Args[2]

  points := readFromFile(inputFile)
```

```
  {
    sf := MakeSplainInterpolation(points)
    // eps := math.Abs(f(0.8) - lf(0.8))
    genPlot("polinome_"+outputFile, sf, points, points[0].x, points[len(points)-
1].x, 0.1)
  }
  {
    sf := MakeExpInterpolation(points)
    // eps := math.Abs(f(0.8) - lf(0.8))
    genPlot("exp_"+outputFile, sf, points, points[0].x, points[len(points)-1].x,
0.1)
  }

}
```

Выводы

Экспоненциальные сплайны хорошо работают там, где наблюдается экспоненциальная природа данных.

Кроме этого, в отличие от полиномиальных сплайнов, у них менее явная точка перегиба.

Тем не менее, реализованный мною алгоритм работает только для монотонных функций, тогда как кубические сплайны могут работать для немонотонных.

Список источников

1. А. С. Ильин, Алгоритм интерполяции возрастающей функции экспоненциальными сплайнами, Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление, 2015, выпуск 2, 41–48