­­­Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Лабораторная работа №4 по дисциплине  
«Вычислительная математика»

"Аппроксимация функции методом наименьших квадратов"

Выполнил: Сафронов Егор Михайлович

Группа: P3213

Преподаватель: Малышева Татьяна Алексеевна

г. Санкт-Петербург

2022

Цель работы:

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

Задание лабораторной работы:

1) Вычислительная реализация задачи:

а) Для заданной функции () построить наилучшие линейное и квадратичное приближения по 11 точкам указанного интервала.

b) Найти среднеквадратические отклонения. Ответы дать с тремя знаками после запятой.

c) Построить графики линейного и квадратичного приближений и заданной функции.

2) Программная реализация задачи:

a) Предусмотреть ввод исходных данных из файла/консоли  
b) Реализовать метод наименьших квадратов, исследуя все функции  
c) Предусмотреть вывод результатов в файл/консоль.  
d) Для линейной зависимости вычислить коэффициент корреляции Пирсона.  
e) Программа должна отображать наилучшую аппроксимирующую функцию.  
f) Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

Вычислительная реализация:

|  |  |
| --- | --- |
| X | Y |
| -2 | -0.4 |
| -1.8 | -0.496634 |
| -1.6 | -0.606428 |
| -1.4 | -0.714140 |
| -1.2 | -0.790306 |
| -1.0 | -0.8 |
| -0.8 | -0.725689 |
| -0.6 | -0.581170 |
| -0.4 | -0.397456 |
| -0.2 | -0.199920 |
| 0 | 0 |

**Квадратичная аппроксимация:**

Решение системы:

Итоговая аппроксимация:

|  |  |  |
| --- | --- | --- |
| X | Y | Epsilon |
| -2 | -0.353775 | 0.002137 |
| -1.8 | -0.528411 | 0.001010 |
| -1.6 | -0.655867 | 0.002444 |
| -1.4 | -0.736142 | 0.000484 |
| -1.2 | -0.769237 | 0.000444 |
| -1.0 | -0.755151 | 0.002011 |
| -0.8 | -0.693885 | 0.001011 |
| -0.6 | -0.585439 | 0.000018 |
| -0.4 | -0.429812 | 0.001047 |
| -0.2 | -0.227006 | 0.000734 |

Chart, line chart

Description automatically generated

Красная – f(x), синяя – P(x)

**Линейная аппроксимация:**

Решение системы:

Итоговая аппроксимация:

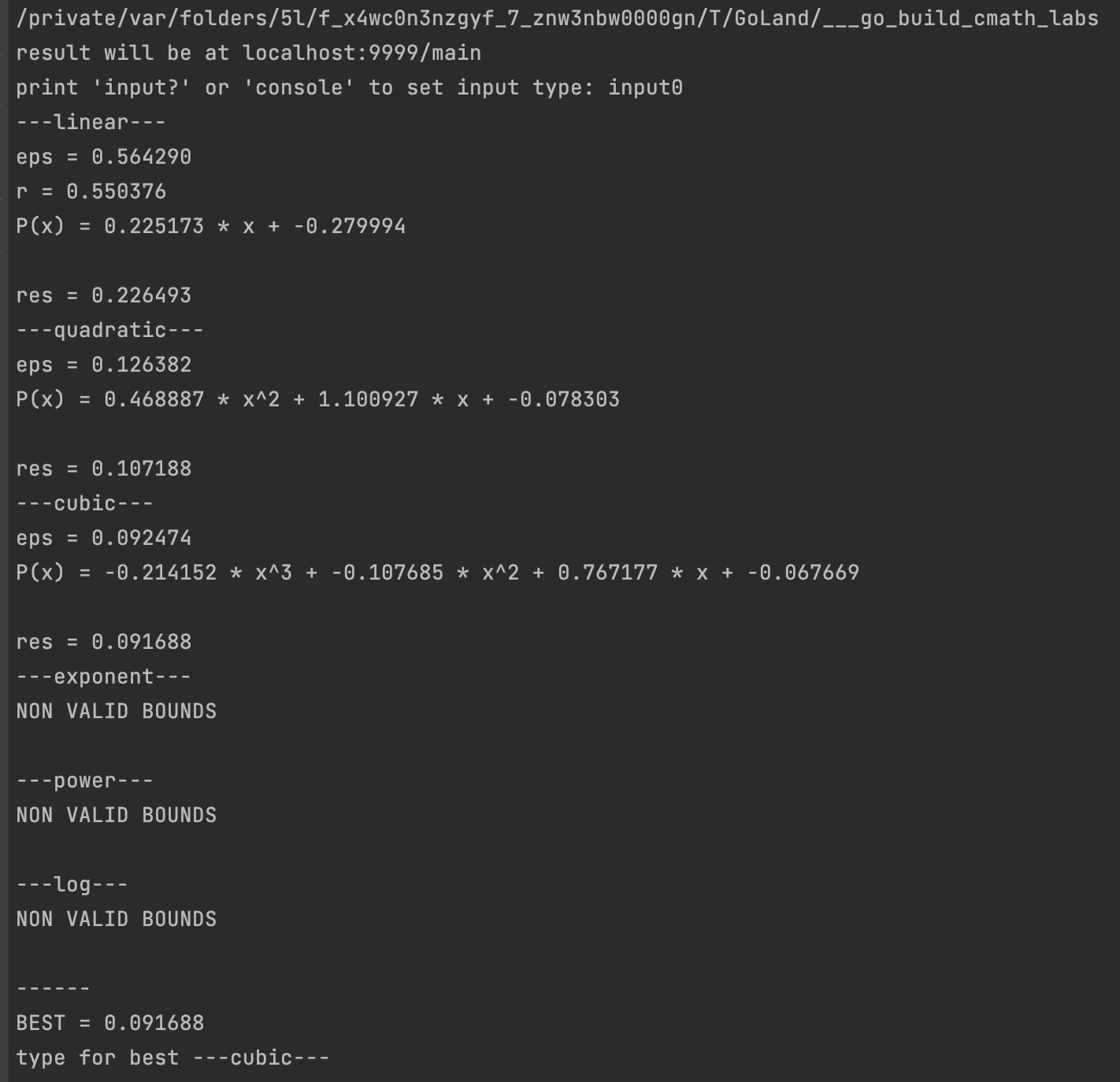
|  |  |  |
| --- | --- | --- |
| X | Y | Epsilon |
| -2 | -0.707628 | 0.094635 |
| -1.8 | -0.669952 | 0.030039 |
| -1.6 | -0.632277 | 0.000668 |
| -1.4 | -0.594601 | 0.014290 |
| -1.2 | -0.556925 | 0.054466 |
| -1.0 | -0.519249 | 0.078821 |
| -0.8 | -0.481574 | 0.059592 |
| -0.6 | -0.443898 | 0.018844 |
| -0.4 | -0.406222 | 0.000077 |
| -0.2 | -0.368547 | 0.028435 |

Chart, line chart

Description automatically generated

Красная – f(x), синяя – P(x)

Примеры работы программы:

A screenshot of a computer

Description automatically generated with medium confidence

Графики:

Chart, line chart

Description automatically generated

Реализация расчета:

func LinearApproximation(inputSeries [][]float64, size int) \*m.Series {  
 a := matrix{  
 {GlobalSums.x2, GlobalSums.x1, GlobalSums.xy},  
 {GlobalSums.x1, float64(size), GlobalSums.y},  
 }  
  
 linearAnswers, errMult, errNone := CalculateMatrix(a, 2)  
  
 if errMult != nil || errNone != nil {  
 fmt.Printf("---linear---\nTROUBLES SOLVING\n\n")  
 return m.NewSeries()  
 }  
  
 linearSeries := m.NewSeries()  
  
 sumx := 0.0  
 sumOldY := 0.0  
 for i := 0; i < size; i++ {  
 sumx += inputSeries[i][0]  
 sumOldY += inputSeries[i][1]  
 }  
 avgX := sumx / float64(size)  
 avgY := sumOldY / float64(size)  
 sumMult := 0.0  
 sumXForSquare := 0.0  
 sumYForSquare := 0.0  
  
 eps := 0.0  
 y2 := 0.0  
 sumy := 0.0  
 for i := 0; i < size; i++ {  
 y := linearAnswers[0]\*inputSeries[i][0] + linearAnswers[1]  
 linearSeries.Add(m.MakeValue(inputSeries[i][0], y))  
  
 sumMult += (inputSeries[i][0] - avgX) \* (inputSeries[i][1] - avgY)  
 sumXForSquare += (inputSeries[i][0] - avgX) \* (inputSeries[i][0] - avgX)  
 sumYForSquare += (inputSeries[i][1] - avgY) \* (inputSeries[i][1] - avgY)  
  
 y2 += y \* y  
 sumy += y  
 eps += math.Pow(math.Abs(y-inputSeries[i][1]), 2)  
 }  
  
 fmt.Printf("---linear---\neps = %f\n", eps)  
 fmt.Printf("r = %f\n", sumMult/math.Sqrt(sumXForSquare\*sumYForSquare))  
 fmt.Printf("P(x) = %f \* x + %f\n\n", linearAnswers[0], linearAnswers[1])  
 fmt.Printf("res = %f\n", math.Sqrt(eps/11))  
 if Best > math.Sqrt(eps/11) {  
 Best = math.Sqrt(eps / 11)  
 }  
 if Best > math.Sqrt(eps/11) {  
 Best = math.Sqrt(eps / 11)  
 Name = "---linear---"  
 }  
  
 return linearSeries  
}  
  
func QuadraticApproximation(inputSeries [][]float64, size int) \*m.Series {  
 a := matrix{  
 {float64(len(inputSeries)), GlobalSums.x1, GlobalSums.x2, GlobalSums.y},  
 {GlobalSums.x1, GlobalSums.x2, GlobalSums.x3, GlobalSums.xy},  
 {GlobalSums.x2, GlobalSums.x3, GlobalSums.x4, GlobalSums.x2y},  
 }  
  
 quadraticAnswers, errMult, errNone := CalculateMatrix(a, 3)  
 if errMult != nil || errNone != nil {  
 fmt.Printf("---quadratic---\nTROUBLES SOLVING\n\n")  
 return m.NewSeries()  
 }  
 quadraticSeries := m.NewSeries()  
 eps := 0.0  
 y2 := 0.0  
 sumy := 0.0  
 for i := 0; i < size; i++ {  
 y := quadraticAnswers[2]\*inputSeries[i][0]\*inputSeries[i][0] + quadraticAnswers[1]\*inputSeries[i][0] + quadraticAnswers[0]  
 quadraticSeries.Add(m.MakeValue(inputSeries[i][0], y))  
 y2 += y \* y  
 sumy += y  
 eps += math.Pow(math.Abs(y-inputSeries[i][1]), 2)  
 }  
  
 fmt.Printf("---quadratic---\neps = %f\n", eps)  
 fmt.Printf("P(x) = %f \* x^2 + %f \* x + %f\n\n", quadraticAnswers[2], quadraticAnswers[1], quadraticAnswers[0])  
 fmt.Printf("res = %f\n", math.Sqrt(eps/11))  
 if Best > math.Sqrt(eps/11) {  
 Best = math.Sqrt(eps / 11)  
 Name = "---quadratic---"  
 }  
 return quadraticSeries  
}  
  
func CubicApproximation(inputSeries [][]float64, size int) \*m.Series {  
 a := matrix{  
 {float64(len(inputSeries)), GlobalSums.x1, GlobalSums.x2, GlobalSums.x3, GlobalSums.y},  
 {GlobalSums.x1, GlobalSums.x2, GlobalSums.x3, GlobalSums.x4, GlobalSums.xy},  
 {GlobalSums.x2, GlobalSums.x3, GlobalSums.x4, GlobalSums.x5, GlobalSums.x2y},  
 {GlobalSums.x3, GlobalSums.x4, GlobalSums.x5, GlobalSums.x6, GlobalSums.x3y},  
 }  
 cubicAnswers, errMult, errNone := CalculateMatrix(a, 4)  
 if errMult != nil || errNone != nil {  
 fmt.Printf("---cubic---\nTROUBLES SOLVING\n\n")  
 return m.NewSeries()  
 }  
 cubicSeries := m.NewSeries()  
 eps := 0.0  
 y2 := 0.0  
 sumy := 0.0  
 for i := 0; i < size; i++ {  
 y := cubicAnswers[3]\*math.Pow(inputSeries[i][0], 3) + cubicAnswers[2]\*math.Pow(inputSeries[i][0], 2) + cubicAnswers[1]\*math.Pow(inputSeries[i][0], 1) + cubicAnswers[0]  
 cubicSeries.Add(m.MakeValue(inputSeries[i][0], y))  
 y2 += y \* y  
 sumy += y  
 eps += math.Pow(math.Abs(y-inputSeries[i][1]), 2)  
 }  
 fmt.Printf("---cubic---\neps = %f\n", eps)  
 fmt.Printf("P(x) = %f \* x^3 + %f \* x^2 + %f \* x + %f\n\n", cubicAnswers[3], cubicAnswers[2], cubicAnswers[1], cubicAnswers[0])  
 fmt.Printf("res = %f\n", math.Sqrt(eps/11))  
 if Best > math.Sqrt(eps/11) {  
 Best = math.Sqrt(eps / 11)  
 Name = "---cubic---"  
 }  
 return cubicSeries  
}

Выводы по работе:

Реализованы различные аппроксимации с использованием метода наименьших квадратов. Для данной функции, на данном участке, лучше всего работает аппроксимация к функции 3ей степени.