Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу "Численные методы"
по теме "Вычисление многократных интегралов с использованием квадратурных формул и метода Монте-Карло"

Студент: А. А. Чакирян
Преподаватель: Ю. В. Сластушенский
Группа: М8О-408Б-17

Дата:
Оценка:
Подпись:

Москва 2020

1 Цель работы

Реализация, анализ, сравнительная характеристика, оценка погрешности метода многократного интегрирования с использованием квадратурных формул и метода Монте-Карло.

2 Алгоритм

2.1 Квадратурные формулы

Алгоритм был реализован рекурсивно для интеграла каждой вложенности. Переполнение стека при решении задачи не грозит, потому что время выполнения растет гораздо быстрее чем глубина стека.

2.2 Метод Монте-Карло

Генерируем много точек, считаем сколько точек попало "под график". Интеграл примерно равен отношению количества точек попавших под график к общему числу точек.

3 Пример работы

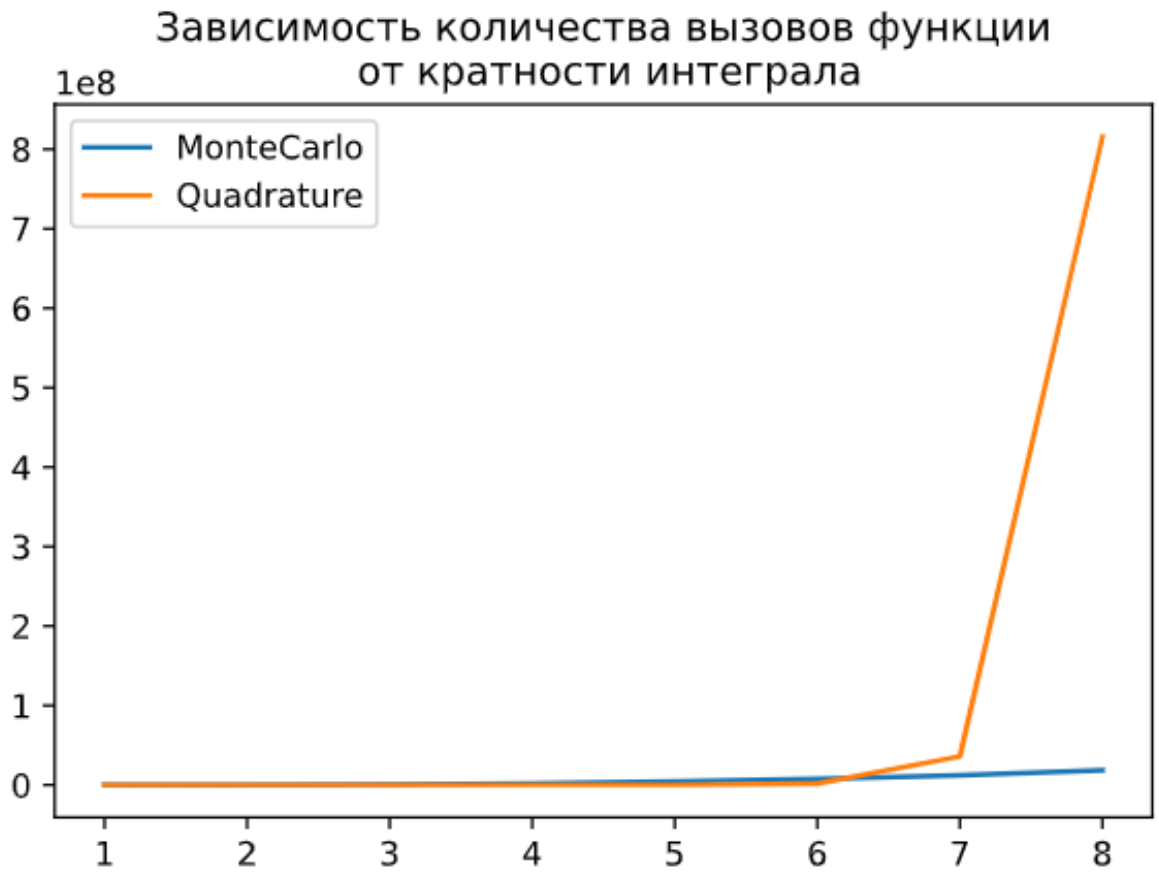
Пример запуска программы для вычисления интеграла

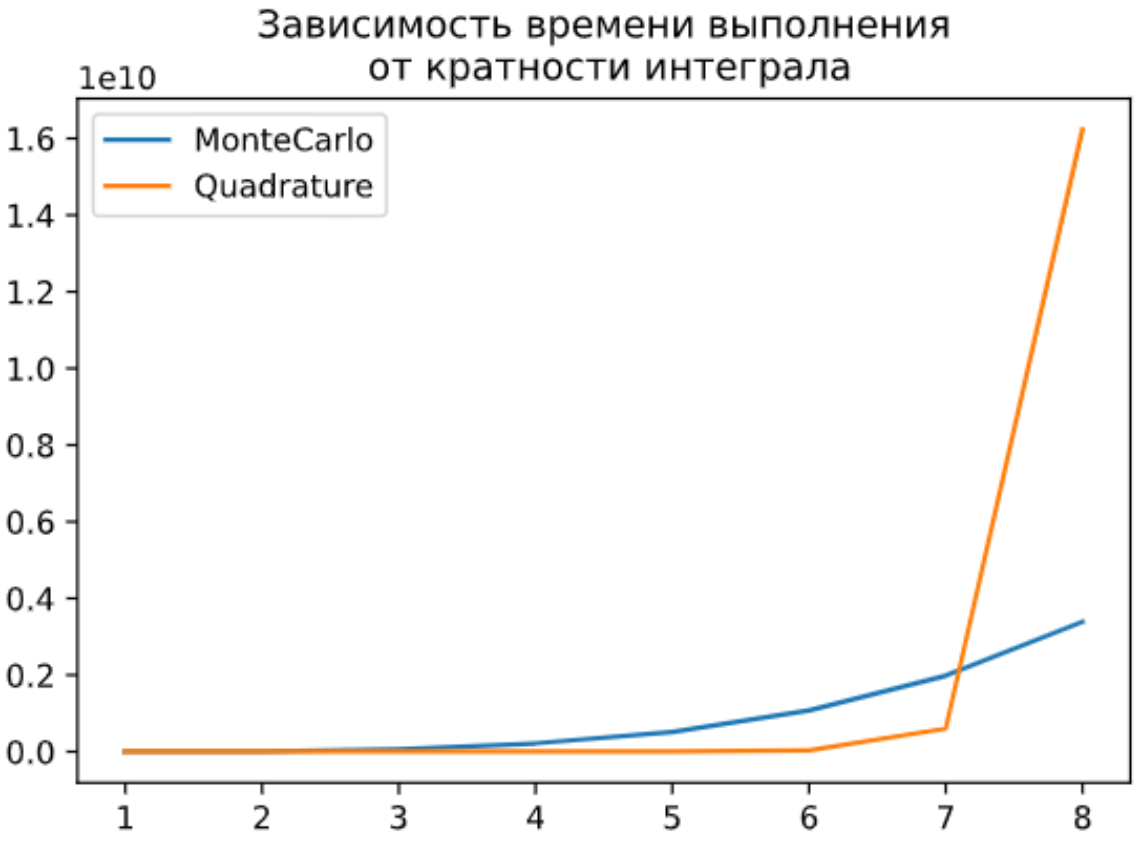
$$\int_0^1 \cdots \int_{2n}^{2n+1} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j + 1 \right) dx_n \dots dx_1$$

```
2020-11-22T17:43:45.920+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 1, "sum": 0.32966674756872066, "invokeCount": 7500,
"executionTime": "287.4µs", "error": 0.003666585764612651}
2020-11-22T17:43:45.920+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 1, "sum": 0.3400000000000001, "invokeCount": 6,
"executionTime": "600ns", "error": 0.006666666666666765}
2020-11-22T17:43:45.931+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 2, "sum": 6.659715965962508, "invokeCount": 195000,
"executionTime": "11.5643ms", "error": 0.006950700704159374}
2020-11-22T17:43:45.932+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 2, "sum": 6.675958000050223, "invokeCount": 49,
"executionTime": "1.7µs", "error": 0.009291333383556477}
2020-11-22T17:43:45.999+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 3, "sum": 27.00299844687684, "invokeCount": 802500,
```

```
"executionTime": "67.0145ms", "error": 0.002998446876841143}
2020-11-22T17:43:45.999+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 3, "sum": 27.0078125, "invokeCount": 729,
"executionTime": "13.7µs", "error": 0.0078125}
2020-11-22T17:43:46.221+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 4, "sum": 69.3378907820623, "invokeCount": 2070000,
"executionTime": "222.0327ms", "error": 0.004557448728974123}
2020-11-22T17:43:46.221+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 4, "sum": 69.34127351487209, "invokeCount": 10000,
"executionTime": "186.7µs", "error": 0.007940181538756974}
2020-11-22T17:43:46.756+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 5, "sum": 141.66563284922614, "invokeCount": 4237501,
"executionTime": "534.805ms", "error": 0.0010338174405148948}
2020-11-22T17:43:46.759+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 5, "sum": 141.67500000000004, "invokeCount": 161051,
"executionTime": "2.5016ms", "error": 0.008333333333382598}
2020-11-22T17:43:47.932+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 6, "sum": 252.00121908891697, "invokeCount": 7545001,
"executionTime": "1.172847s", "error": 0.0012190889169687 58}
2020-11-22T17:43:47.962+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 6, "sum": 252.0100000000001, "invokeCount": 1771561,
"executionTime": "30.4821ms", "error": 0.010000000000104592}
2020-11-22T17:43:49.998+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 7, "sum": 408.3367726678104, "invokeCount": 12232501,
"executionTime": "2.0352464s", "error": 0.0034393344770933254}
2020-11-22T17:43:50.677+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 7, "sum": 408.34326526992356, "invokeCount": 35831808,
"executionTime": "679.3044ms", "error": 0.009931936590248824}
2020-11-22T17:43:54.293+0300    INFO    MonteCarlo      integrate/main.go:148
calculate       {"n": 8, "sum": 618.6651086909142, "invokeCount": 18540001,
"executionTime": "3.616292s", "error": 0.0015579757524619708}
2020-11-22T17:44:08.865+0300    INFO    Quadrature      integrate/main.go:176
calculate       {"n": 8, "sum": 618.6739237583992, "invokeCount": 815730721,
"executionTime": "14.5717169s", "error": 0.007257091732526533}
```

4 Сравнение и анализ





Квадратуры подходят для интегрирования при малой кратности интеграла. С увеличением кратности, резко возрастает время выполнения. При больших входных параметрах метод Монте-Карло показывает лучший результат.

Приложение

Листинг кода

main.go

```go
package main

import (
        "encoding/json"
        "flag"
        "fmt"
        "math"
        "os"
        "time"

        "github.com/MayaMIkachan/numerical-methods/internal/integrate"
        "go.uber.org/atomic"
        "go.uber.org/zap"
        "go.uber.org/zap/zapcore"
)

func f(args []float64) float64 {
        var sum float64
        for _, value := range args {
                sum += value * value
        }
        return sum
}

func main() {
        outputPath := flag.String("output", "report.json", "output
            path")
        flag.Parse()

        log, err := newLogger()
        if err != nil {
                errorf("%v\n", err)
        }

        var (
                points = []int{
                        7500,
                        195000,
                        802500,
                        2070000,
                        4237501,
                        7545001,
                        12232501,
                        18540001,
                        26707500,
                        36975001,
```

```
46                  }
47                  h = []float64{
48                          0.2,
49                          0.166667,
50                          0.125,
51                          0.111111,
52                          0.1,
53                          0.1,
54                          0.0909091,
55                          0.0833333,
56                          0.0769231,
57                          0.0769231,
58                  }
59                  refValues = []float64{1.0 / 3.0, 20.0 / 3.0, 27.0,
                        208.0 / 3.0, 425.0 / 3.0, 252.0, 1225.0 / 3.0,
                        1856.0 / 3, 891.0, 3700.0 / 3.0}
60          )
61
62          stats := []report{}
63          for i := 1; i <= 9; i++ {
64                  var (
65                          a    = make([]float64, i)
66                          b    = make([]float64, i)
67                          grid = make([]float64, i)
68                  )
69
70                  for j := 0; j < i; j++ {
71                          a[j] = float64(2 * j)
72                          b[j] = float64(2*j + 1)
73                          grid[j] = h[i-1]
74                  }
75                  stat, err := calculate(
76                          log,
77                          i,
78                          a,
79                          b,
80                          points[i-1],
81                          grid,
82                          refValues[i-1],
83                  )
84                  if err != nil {
85                          errorf("%v\n", err)
86                  }
87                  stats = append(stats, stat)
88          }
89
90          marshaled, err := json.Marshal(stats)
91          if err != nil {
92                  errorf("%v\n", err)
93          }
94          file, err := os.Create(*outputPath)
```

```
95          if err != nil {
96                  errorf("%v\n", err)
97          }
98          if _, err := file.Write(marshaled); err != nil {
99                  errorf("%v\n", err)
100         }
101 }
102
103 func newLogger() (*zap.Logger, error) {
104         conf := zap.NewDevelopmentConfig()
105         conf.EncoderConfig.EncodeLevel = zapcore.
                CapitalColorLevelEncoder
106         return conf.Build()
107 }
108
109 type stat struct {
110         InvokeCount   int64
111         ExecutionTime time.Duration
112 }
113
114 type report struct {
115         MonteCarlo stat
116         Quadrature stat
117 }
118
119 func calculate(
120         log *zap.Logger,
121         n int,
122         a []float64,
123         b []float64,
124         points int,
125         h []float64,
126         value float64,
127 ) (report, error) {
128         r := report{}
129
130         {
131                 log := log.Named("MonteCarlo")
132                 f, counter := withInvokeCounter(f)
133                 start := time.Now()
134                 sum, err := integrate.MonteCarlo(
135                         f,
136                         a,
137                         b,
138                         points,
139                 )
140                 if err != nil {
141                         log.Error("calculate", zap.Error(err))
142                         return report{}, err
143                 }
144                 r.MonteCarlo = stat{
```

```
145                             InvokeCount:   counter.Load(),
146                             ExecutionTime: time.Since(start),
147                     }
148                     log.Info(
149                             "calculate",
150                             zap.Int("n", n),
151                             zap.Float64("sum", sum),
152                             zap.Int64("invokeCount", r.MonteCarlo.
                                    InvokeCount),
153                             zap.Duration("executionTime", r.MonteCarlo.
                                    ExecutionTime),
154                             zap.Float64("error", math.Abs(value-sum)),
155                     )
156             }
157
158             {
159                     log := log.Named("Quadrature")
160                     f, counter := withInvokeCounter(f)
161                     start := time.Now()
162                     sum, err := integrate.Quadrature(
163                             f,
164                             a,
165                             b,
166                             h,
167                     )
168                     if err != nil {
169                             log.Error("calculate", zap.Error(err))
170                             return report{}, err
171                     }
172                     r.Quadrature = stat{
173                             InvokeCount:   counter.Load(),
174                             ExecutionTime: time.Since(start),
175                     }
176                     log.Info(
177                             "calculate",
178                             zap.Int("n", n),
179                             zap.Float64("sum", sum),
180                             zap.Int64("invokeCount", r.Quadrature.
                                    InvokeCount),
181                             zap.Duration("executionTime", r.Quadrature.
                                    ExecutionTime),
182                             zap.Float64("error", math.Abs(value-sum)),
183                     )
184             }
185
186             return r, nil
187     }
188
189     func withInvokeCounter(f func([]float64) float64) (func([]float64)
        float64, *atomic.Int64) {
190             counter := atomic.NewInt64(0)
```

```
191
192             return func(args []float64) float64 {
193                     counter.Add(1)
194                     return f(args)
195             }, counter
196     }
197
198     func errorf(format string, args ...interface{}) {
199             fmt.Fprintf(os.Stderr, format, args...)
200             os.Exit(1)
201     }
```

internal/integrate/integrate.go

```
1   package integrate
2
3   import (
4           "errors"
5           "math"
6           "math/rand"
7   )
8
9   var (
10          // ErrWrongDimensions is ...
11          ErrWrongDimensions = errors.New("wrong dimensions")
12  )
13
14  // MonteCarlo returns multidim integral
15  func MonteCarlo(
16          f func([]float64) float64,
17          left, right []float64,
18          points int,
19  ) (float64, error) {
20          n := len(left)
21          if n != len(right) {
22                  return 0, ErrWrongDimensions
23          }
24
25          var (
26                  x      = make([]float64, n)
27                  sum    = 0.0
28                  volume = 1.0
29          )
30
31          for i := 0; i < n; i++ {
32                  volume *= right[i] - left[i]
33          }
34
35          for i := 0; i < points; i++ {
36                  randomVector(left, right, x)
37                  sum += f(x)
38          }
```

```
39
40          return sum * volume / float64(points), nil
41  }
42
43  func randomVector(a, b, v []float64) {
44          n := len(a)
45
46          for i := 0; i < n; i++ {
47                  v[i] = a[i] + rand.Float64()*(b[i]-a[i])
48          }
49  }
50
51  // Quadrature returns multidim integral
52  func Quadrature(
53          f func([]float64) float64,
54          left, right []float64,
55          h []float64,
56  ) (float64, error) {
57          n := len(left)
58          if n != len(right) || n != len(h) {
59                  return 0, ErrWrongDimensions
60          }
61
62          var (
63                  calculate func() float64
64                  k         int
65                  x         = make([]float64, n)
66          )
67          copy(x, left)
68
69          calculate = func() float64 {
70                  var (
71                          sum   = 0.0
72                          w     = right[k] - left[k]
73                          count = int(math.Round(w / h[k]))
74                  )
75                  if k < n-1 {
76                          k++
77                          sum += 0.5 * calculate()
78
79                          for i := 1; i < count; i++ {
80                                  x[k] = left[k] + float64(i)*h[k]
81                                  k++
82                                  sum += calculate()
83                          }
84
85                          x[k] = right[k]
86                          k++
87                          sum += 0.5 * calculate()
88                  } else {
89                          sum += 0.5 * f(x)
```

```
                        for i := 1; i < count; i++ {
                                x[k] = left[k] + float64(i)*h[k]
                                sum += f(x)
                        }
                        x[k] = right[k]
                        sum += 0.5 * f(x)
                }
                sum *= h[k]

                x[k] = left[k]
                k--

                return sum
        }

        return calculate(), nil
}
```