Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 8 – Lösungsvorschlag

Aufgabe 9 Vereinbaren Sie eine abstrakte Basisklasse (ABC) `PDF` zur Modellierung von Wahrscheinlichkeitsdichtefunktionen (*PDFs*) $\mathbb{R} \to [0,1]$. Abgeleitete Klassen, die PDFs modellieren, sollen eine konstante Methode implementieren, die den Mittelwert der Verteilung als Rückgabewert vom Typ `double` liefert.

PDF

```
class PDF {
  public:
    virtual double getMean() const = 0;
    virtual ~PDF() = default;
};
```

Implementieren Sie Klassen für die folgenden Familien von PDFs inkl. Überladung des Funktionsauswertungsoperators:

Normalverteilung PDFs f mit zwei Parametern $\mu, \sigma \in \mathbb{R}$, Mittelwert μ und $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$.

Gleichverteilung PDFs f mit zwei Parametern $a, b \in \mathbb{R}$ mit $a < b$ (geeignete Fehlerbehandlung im Konstruktor), Mittelwert $a + \frac{b-a}{2}$ und $f(x) = \begin{cases} \frac{1}{b-a} & a \le x \le b \\ 0 & \text{sonst} \end{cases}$

Exponentialverteilung PDFs f mit Parameter $\lambda \in \mathbb{R}$, Mittelwert $\frac{1}{\lambda}$ und $f(x) = \lambda \exp(-\lambda x)$

Verteilungen

```
class NormV : public PDF {
  public:
    double mean, stddev;
    NormV (double mean_ = 0, double stddev_ = 1)
      : mean(mean_), stddev(stddev_) {}
    virtual double operator()(double x) const {
      return M_2_SQRTPI / (2 * M_SQRT2 * stddev) * exp(-((x - mean) * (x - mean)) /
      ↪  (2*stddev*stddev));
    }
    double getMean() const {
      return mean;
    }
};
class GleichV : public PDF {
  private:
    double from, to;

  public:
    GleichV(double from_, double to_): from(from_), to(to_) {
      if (to <= from) {
        ostringstream sstr;
```

```
        sstr << "NormV(" << from << ", " << to << ")";
        throw invalid_argument(sstr.str());
      }
    }
    virtual double operator()(double x) const {
      if (x < from || x > to)
        return 0;
      return 1 / (to - from);
    }
    double getMean() const {
      return from + (to - from) / 2;
    }
};
class ExpV : public PDF {
  public:
    double rate;
    ExpV(double rate_): rate(rate_) {}
    virtual double operator()(double x) const {
      if (x < 0)
        return 0;
      return rate * exp(- rate * x);
    }
    double getMean() const {
      return 1 / rate;
    }
};
```

Implementieren Sie eine Klasse `SumV` für Summen von einer oder mehr PDFs. Stellen Sie die Einhaltung der Invariante bzgl. der Anzahl der enthaltenen PDFs stets sicher. Zur Modellierung von `SumV` empfiehlt sich ein privates Attribut vom Typ `list<unique_ptr<PDF>>`. Implementieren Sie einen Konstruktor für `SumV`, der einen Wert vom Typ „Zeiger auf `PDF`“ akzeptiert. Implementieren Sie einen Additions-Zuweisungsoperator (`+=`) als Methode von `SumV` mit Parameter vom Typ „Zeiger auf `PDF`“. Der Mittelwert einer Summe von PDFs ist die Summe der Mittelwerte der einzelnen PDFs.

SumV

```
class SumV : public PDF {
  private:
    list<unique_ptr<PDF>> pdfs;

  public:
    explicit SumV(PDF* pdf) { *this += pdf; }
    SumV(const SumV&) = delete;

    SumV& operator+=(PDF* pdf) {
      pdfs.push_back(unique_ptr<PDF>{pdf});
      return *this;
    }

    double getMean() const {
      double sum = 0;
      for (const unique_ptr<PDF>& pdf: pdfs)
        sum += pdf->getMean();
      return sum;
    }
};
```

Implementieren Sie ein Hauptprogramm, das von der Standardeingabe zunächst den Namen einer PDF-Familie (`norm`, `gleich` oder `exp`) einliest. Es soll dann eine endliche Abbildung von Namen von PDF-Familien

auf geeignete Funktionsobjekte verwendet werden um die benötigten Parameter abzufragen und dann ein Objekt der passenden Klasse zu initialisieren. Dies soll iteriert werden solange das Einlesen des Namens einer PDF-Familie nicht fehlschlägt. Die Eingabe nicht implementierter Namen soll geeignet behandelt werden und die Eingabe weiterer PDFs nicht verhindern.

Die eingegebenen PDFs sollen jeweils direkt nach ihrer Erzeugung einem Objekt der Klasse `SumV` beigefügt werden und der Mittelwert der so konstruierten PDF am Ende einmal ausgegeben werden.

distributions.cpp

```
#include <sstream>
#include <cmath>
#include <memory>
#include <list>
#include <string>
#include <map>
#include <functional>
#include <iostream>

using namespace std;
```


```
const map<string, function<PDF*()>> pdfs{
  {"norm", []() -> PDF* {
      double mean, stddev;
      cout << "mean, stddev = ";
      cin >> mean >> stddev;
      return new NormV{mean, stddev};
    }
  },
  {"gleich", []() -> PDF* {
      double from, to;
      cout << "from, to = ";
      cin >> from >> to;
      return new GleichV{from, to};
    }
  },
  {"exp", []() -> PDF* {
      double rate;
      cout << "rate = ";
      cin >> rate;
      return new ExpV{rate};
    }
  }
};

int main() {
  unique_ptr<SumV> sum;
  while (true) {
```

```
    string fun;
    cout << "sum += ";
    if (!(cin >> fun)) {
      cout << endl;
      break;
    }

    try {
      PDF* pdf = (pdfs.at(fun))();
      if (sum)
        *sum += pdf;
      else
        sum = unique_ptr<SumV>{new SumV{pdf}};
    } catch (const out_of_range&) {
      cout << "Unknown distribution: " << fun << endl;
    }
  }

  cout << "sum.getMean() = " << sum->getMean() << endl;
}
```

Beispiel (Kontrollergebnis). Ihr Programm könnte sich verhalten, wie folgt:

```
sum += norm
mean, stddev = 3 1
sum += norm
mean, stddev = 1 1
sum += gleich
from, to = -1 1
sum += exp
rate = 0.1
sum +=
sum.getMean() = 14
```