Monte-Carlo-Simulation

Aufgabe 1

```
zi = function(xi, yi) {
  out = 4 * ifelse(xi^2 + yi^2 <= 1, 1, 0)
  return(out)
}

simpi = function(n) {
  xi = runif(n, -1, 1)
  yi = runif(n, -1, 1)
  z = zi(xi, yi)
  out = mean(z)
  return(out)
}
```

```
pimat = matrix(NA, nrow = 8, ncol = 3)
for (k in seq(8)) {
  z = simpi(10^k)
  pimat[k, 1] = k
  pimat[k, 2] = z
  pimat[k, 3] = round(abs(pi - z), 6)
}
df_pi = data.frame(pimat)
colnames(df_pi) = c("k", "z", "e")
kable(df_pi, align = "c")
```

k	z	e
1	3.200000	0.058407
2	3.120000	0.021593
3	3.144000	0.002407
4	3.138800	0.002793
5	3.138960	0.002633
6	3.141484	0.000109
7	3.141327	0.000266
8	3.141677	0.000084

Um mit Chebyshev abschätzen zu können benötigen wir Erwartungswert und Varianz der Zufallsvariablen z_i. Der Erwartungswert sollte mit π übereinstimmen. Wir rechnen nach. Sei dazu z_1 die Approximation von π mittels des Einheitskreises im Einheitsquadrat eingeschränkt auf den 1. Quadranten.

$$\mathrm{E}(z_i) = \mathrm{E}(4z_1) = 4\,\mathrm{E}(z_i) = 4\int_0^1 \sqrt{1-x^2}\;dx = 4\frac{\pi}{4} = \pi \tag{1}$$

$$\begin{aligned}
\mathrm{V}(z_i) &= \mathrm{V}(4z_1) = 16\,\mathrm{V}(z_i) && (2)\\
&= 16\int_0^1 \sqrt{1-x^2}^2\;dx - 16\left(\int_0^1 \sqrt{1-x^2}\;dx\right)^2 && (3)\\
&= 16\int_0^1 1-x^2\;dx - 16\frac{\pi^2}{16} = 16\left[x-\frac{1}{3}x^3\right]_0^1 - \pi^2 && (4)\\
&= \frac{32}{3} - \pi^2 \approx 0.797 && (5)
\end{aligned}$$

Damit erhalten wir dann für $\bar{z}$:

$$\mathrm{E}(\bar{z}) = \mathrm{E}\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right) = \frac{1}{n}n\,\mathrm{E}(z_i) = \mathrm{E}(z_i) = \pi \tag{6}$$

$$\mathrm{V}(\bar{z}) = \mathrm{V}\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n} z_i\right) = \frac{n}{n^2}\,\mathrm{V}(z_i) = \frac{1}{n}\,\mathrm{V}(z_i) \approx \frac{0.797}{n} \tag{7}$$

Nach Chebyshev gilt nun für einen beliebigen Fehler ε und $\sigma^2 = \mathrm{V}(z_i)$:

$$\mathrm{P}\left(|\bar{z}-\pi| \geq \varepsilon\right) \leq \frac{\mathrm{V}(\bar{z})}{\varepsilon^2} \tag{8}$$

$$\implies \mathrm{P}\left(|\bar{z}(n)-\pi| \geq \varepsilon\right) \leq \frac{\sigma^2}{n\varepsilon^2} \tag{9}$$

Nun gilt es zu bedenken, dass die Abschätzung zwar für alle $\varepsilon > 0$ gilt, aber nur für $n\varepsilon^2 > \sigma^2$ wirklich Sinn macht. Sprich es macht nur Sinn zu prüfen, wie hoch die Chance ist, dass man ein Vielfaches der Standardabweichung vom Mittelwert abweicht. Wie man an der Abschätzung erkennen kann, sinkt die Genauigkeit, dass wir um mehr als einen gegebenen Fehler abweichen mit zunehmenden n. Das heißt insbesondere, dass mit höheren n der Fehler sinkt. Dies können wir anhand der Tabelle bestätigen.

A2

a)

```
simsnd = function(n) {
  x = runif(n, -1.96, 1.96)
  phi = dnorm(x)
  out = 2 * 1.96 * mean(phi)
  return(out)
}
```

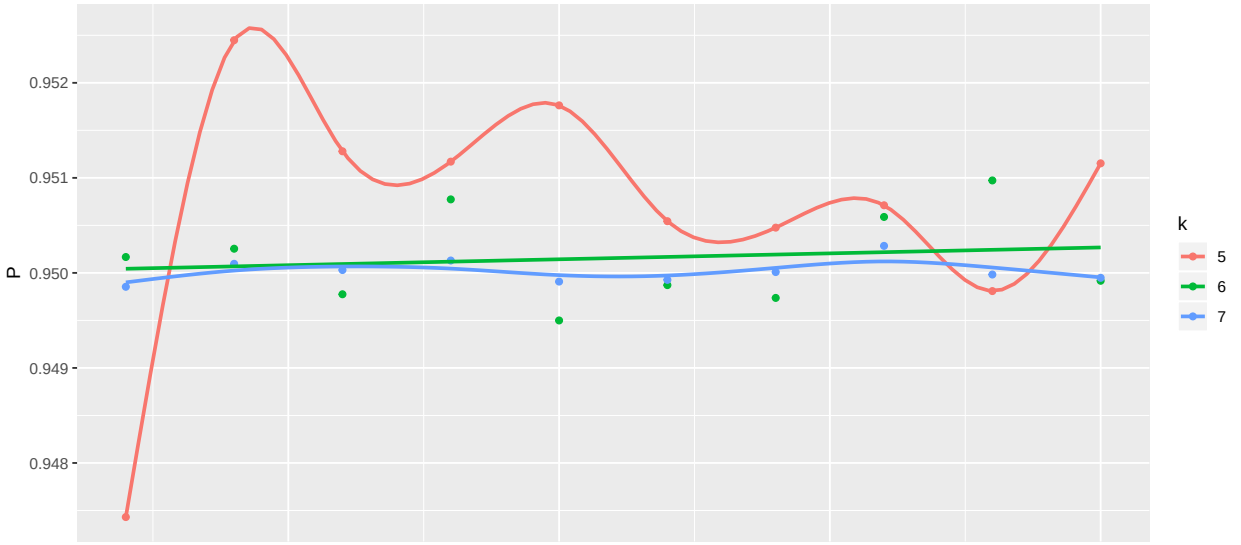
```
sndmat = matrix(NA, nrow = 8, ncol = 3)
for (k in seq(8)) {
  z = simsnd(10^k)
  sndmat[k, 1] = k
  sndmat[k, 2] = z
  sndmat[k, 3] = round(abs(0.95 - z), 6)
}
df_snd = data.frame(sndmat)
colnames(df_snd) = c("k", "z", "e")
kable(df_snd, align="c")
```

k	z	e
1	0.5535311	0.396469
2	0.9300504	0.019950
3	0.9716125	0.021613
4	0.9544350	0.004435
5	0.9497721	0.000228
6	0.9499837	0.000016
7	0.9499067	0.000093
8	0.9499762	0.000024

Hier sehen wir , dass es ab etwa 10^6 Versuchen sehr präzise wird.

b)

```
nexp = 10
M = matrix(NA, nrow = nexp, ncol = 4)
for (k in seq(5,7)) {
  for (i in seq(nexp)) {
    z = simsnd(10^k)
    M[i, 1] = i
    M[i,k - 3] = z
  }
}
df = data.frame(M)
colnames(df) = c("i", "5", "6", "7")
ggdf = gather(df, "k", "P", seq(2,4))
gg = ggplot(
  data = ggdf,
  mapping = aes(x = i, y = P, group = k, col = k)
)
gg = gg + xlab("Iteration") + ylab("P")
gg = gg + geom_point()
gg = gg + stat_smooth(method = "gam", formula = y ~ s(x, bs='cr'), se = FALSE)
gg = gg + theme(
  axis.title.x=element_blank(),
  axis.text.x=element_blank(),
  axis.ticks.x=element_blank()
)
gg
```



In der Abbildung können wir erkennen, dass für $k = 5 \implies n = 10^5$ es noch eine signifikante Streuung um den erwarteten Wert von $P = 0.95$ gibt. Um dies graphisch besser darzustellen haben wir einen kubischen Regressionspline durch die Punkte gezogen. Man erkennt für $k = 5$ deutliche Abweichungen, während für $k = \{6, 7\}$ der Spline eher eine Gerade am erwarteten Wert ist. Erinnern wir uns an die Abschätzung mit Chebyshev aus Aufgabe 1, so sollten wir auch hier eine ähnliche Abschäzung erhalten (da analoges Vorgehen) und da die Wahrscheinlichkeit von Mittelwert abzuweichen mit dem Produkt aus Fehler und n sinkt, braucht es schon ein hohes n, um möglichst kleine Fehler zu garantieren.

A3

```
u1 = function(x,y) {
  if (x^2 / 49 + y^2 / 9 - 1 <= 0) {
    if (abs(x) >= 4 && y >= -(3 * sqrt(33))/7 && y <= 0) {
      return(1)
    } else {
      if (abs(x) >= 3 && y >= 0) {
        return(1)
      }
    }
  }
  return(0)
}
u2 = function(x,y) {
  if (y >= -3 && y <= 0 && -4 <= x && x <= 4) {
    if (-((3*sqrt(33)-7)*x^2)/112 + abs(x)/2 + sqrt(1-(abs(abs(x)-2)-1)^2) -y - 3 <= 0) {
      return(1)
    }
  }
  return(0)
}
u3 = function(x,y) {
  if (y >= 0 && 3/4 <= abs(x) && abs(x) <= 1) {
    if (-8*abs(x) - y + 9 >= 0) {
      return(1)
    }
  }
  return(0)
}
u4 = function(x,y) {
  if (1/2 <= abs(x) && abs(x) <= 3/4 && 3 * abs(x) - y + 3/4 >= 0 && y >= 0) {
    return(1)
  }
  return(0)
}
u5 = function(x,y) {
  if (abs(x) <= 1/2 && y >= 0 && 9/4 - y >= 0) {
    return(1)
  }
  return(0)
}
u6 = function(x,y) {
  if (1 <= abs(x) && abs(x) <= 3 && y >= 0) {
    if (-abs(x)/2 - 3/7 * sqrt(10)*sqrt(4 - (abs(x) -1)^2) -y + 6*sqrt(10)/7 + 3/2 >= 0) {
      return(1)
    }
  }
  return(0)
}

u = function(x,y) {
  if (u1(x,y) == 1) {
    return(1)
```

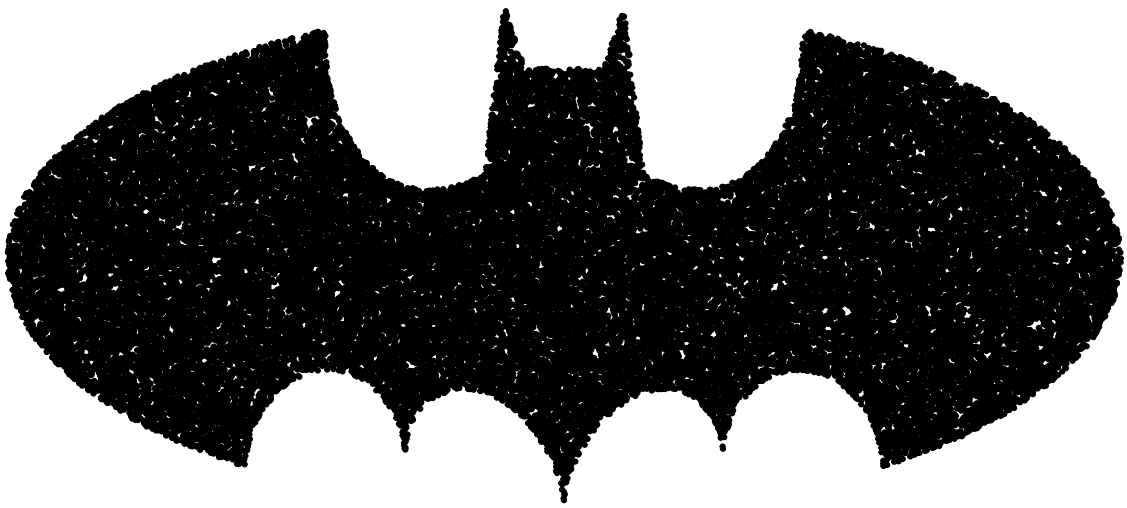
```
  }
  if (u2(x,y) == 1) {
    return(1)
  }
  if (u3(x,y) == 1) {
    return(1)
  }
  if (u4(x,y) == 1) {
    return(1)
  }
  if (u5(x,y) == 1) {
    return(1)
  }
  if (u6(x,y) == 1) {
    return(1)
  }
  return(0)
}

batman = function(n) {
  x = runif(n, -7, 7)
  y = runif(n, -4, 4)
  df = data.frame(x = double(), y = double(), z = double())
  for (i in seq(n)) {
    z = u(x[i], y[i])
    df = rbind(df, c(x[i], y[i], z))
  }
  colnames(df) = c("x", "y", "z")
  return(df)
}
```

```
n = 10^5
df = batman(n)
```

a)

```
gg = ggplot(
  data = df[df$z == 1,],
  mapping = aes(
    x = x,
    y = y
  )
)
gg = gg + geom_point(size = 1)
gg = gg + theme(
  axis.title.x=element_blank(),
  axis.text.x=element_blank(),
  axis.ticks.x=element_blank(),
  axis.title.y=element_blank(),
  axis.text.y=element_blank(),
  axis.ticks.y=element_blank(),
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  panel.background = element_blank()
)
gg
```



Es ist Batman. Für `geom_point(size = 1)` sogar Space Batman.

b)

```
meanz = mean(df$z)
A = 14 * 8 * meanz
A = round(A,2)
```

Der Space Batman hat einen approximativen Flächeninhalt von $A = 48.67$.