Computational Statistics - Suggested Solution for Exam

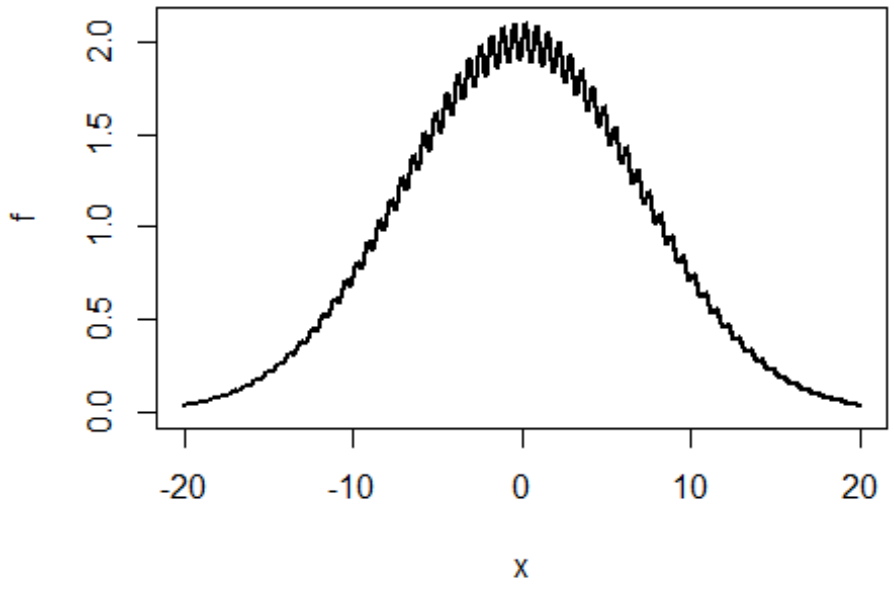
Frank Miller, IDA, Linköping University

2024-03-19

Assignment 1

Question 1.1

```
f <- function(x){
  exp(-x^2/100) * (2-0.1*cos(3*pi*(x+0.1)))
}
vf <- Vectorize(f)
xv <- -2000:2000/100
plot(c(min(xv), max(xv)), c(0, max(vf(xv))), type="n", xlab=expression(x),
ylab="f")
lines(xv, vf(xv), col=1, lwd=2)
```



Not requested:

```
optimize(f, c(-3, 3), maximum=TRUE)
```

```
## $maximum
## [1] 1.559289
```

```
##
## $objective
## [1] 2.049321
```

The main problem with Newton here is that it searches a local maximum usually close to the starting value. Here are a lot of local maxima. In general the algorithm cannot find a global maximum. However, one could use a grid of many starting values and run Newton many times. The solution with the highest value of f can then be the approximation for the global maximum. An advantage of simulated annealing: It is build for searching a global maximum since it accepts also deterioration in the initial phase of the algorithm. By this, it can escape from local maxima. A disadvantage is that simulated annealing is much slower than Newton in converging.

Question 1.2

The candidate for the next iteration is chosen in a neighbourhood of the current value to search locally around the current value. Specifically, we add a standard normally distributed value to the current value. The temperature is updated after each 100 iterations (multiplied with taufrac< 1).

```
simann <- function(xstart, taustart=10, maxiter=10000, taufac=0.95){
  tau   <- taustart           # current temperature
  x     <- xbest <- xstart    # current approximation
  fx    <- fxbest <- f(x)
  for (i in 1:maxiter){
    xcand  <- x + rnorm(1)
    fxcand <- f(xcand)
    if (fxcand>fx){
      x  <- xcand
      fx <- fxcand
      if (fxcand>fxbest){
        xbest  <- xcand
        fxbest <- fxcand
      }
    } else {
      if (runif(1) < exp((fxcand-fx)/tau)){
        x  <- xcand
        fx <- fxcand
      }
    }
    if (i/100-round(i/100)==0) tau <- tau*taufac
  }
  c(xbest, fxbest)
}
```

Question 1.3

A single run:

```r
set.seed(2024)
simann(5, taustart=0.1)
```

```
## [1] 0.232245 2.098862
```

The global maximum is at 0.23 with function value 2.0989. We count therefore, how many times we get a result x with $f(x) > 2.098$. We use start-temperature 10 and 0.1 and run the algorithm 100 times.

```r
succ <- 0
for (j in 1:100){
  xopt <- simann(5, taustart=10)
  succ <- succ + (xopt[2]>2.098)
}
succ/100
```

```
## [1] 0.82
```

```r
succ <- 0
for (j in 1:100){
  xopt <- simann(5, taustart=0.1)
  succ <- succ + (xopt[2]>2.098)
}
succ/100
```

```
## [1] 1
```

The result is that start-temperature 10 is too high (around 80-90% success rate); better is 0.1 with (almost) 100% success rate.

Using optim, one can apply simulated annealing the following way (and below using the quasi-Newton BFGS (not required, here).

```r
optim(5, f, method="SANN", control=list(fnscale=-1, temp=10, maxit=10000,
tmax=10000))
```

```
## $par
## [1] 0.241158
##
## $value
## [1] 2.098507
##
## $counts
## function gradient
##    10000       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
optim(5, f, method="BFGS", control=list(fnscale=-1))
```

```
## $par
## [1] 4.213307
##
## $value
## [1] 1.756937
##
## $counts
## function gradient
##        8        4
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Question 2

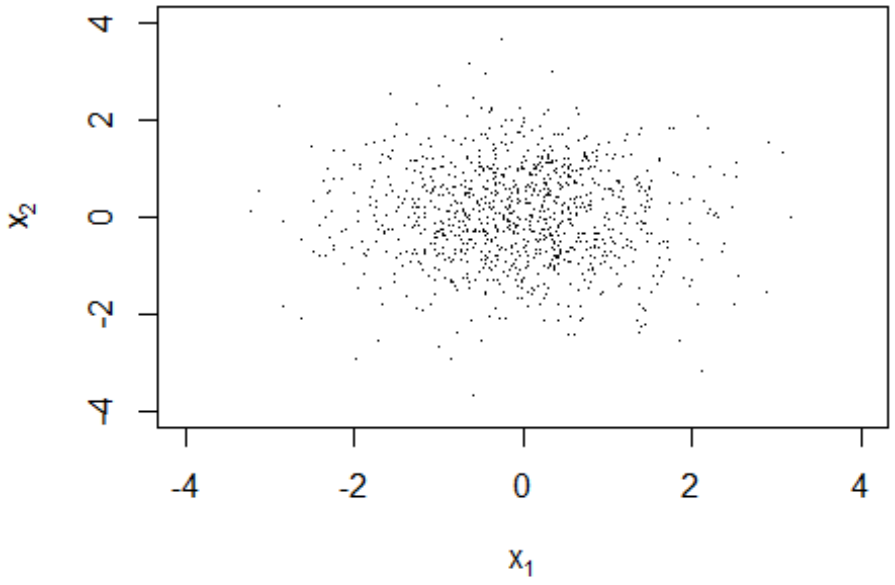
Question 2.1

A function for Gibbs sampling; we include here already plotting the generated sample since this is needed in 2.2.

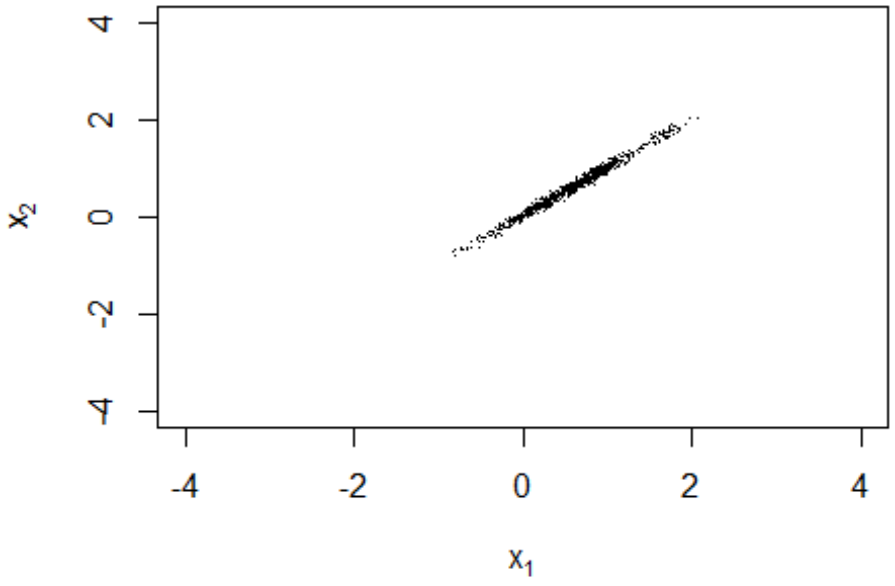
```r
gibbs <- function(xstart=c(0, 0), n=1000, rho=0){
  x   <- xstart
  x2n <- x[2]
  plot(c(-4, 4), c(-4, 4), type="n", xlab=expression(x[1]),
ylab=expression(x[2]))
  for (i in 1:1000){
    x1n  <- rnorm(1, mean=rho*x2n, sd=sqrt(1-rho^2))
    x2n  <- rnorm(1, mean=rho*x1n, sd=sqrt(1-rho^2))
    newx <- c(x1n, x2n)
    x <- cbind(x, newx)
    points(x1n, x2n, pch=".")
  }
  x
}
```

Question 2.2

```r
set.seed(240319)
res0 <- gibbs(rho=0)
```




```
res9 <- gibbs(rho=0.998)
```



The first case works well and the sample seems to represent the bivariate normal well. For high correlation, Gibbs can only make small steps in each iteration and does not manage to explore the whole distribution when $n = 1000$ samples are generated. Therefore, the distribution is not well represented by the sample (with very high n, this problem disappears).

Question 2.3

```
sum(colSums(res0)>0)/1000
```

```
## [1] 0.486
```

```
sum(colSums(res9)>0)/1000
```

```
## [1] 0.894
```

The second result shows that the generated sample does not represent the distribution well.

Question 2.4

We can generate two independent standard normal variables y_1 and y_2. Set $x_1 = y_1$ and $x_2 =$ an appropriate linear combination (=weighted sum) of y_1 and y_2 such that x_1 and x_2 have correlation ρ (this means $x_2 = \rho x_1 + \sqrt{1-\rho^2}x_2$). (x_1, x_2) is then a two-dimensional sample. Advantage of this alternative: One does not need to derive the conditional distributions in general (here however, they are already given). Problems like seen here with high correlation are avoided. Disadvantage of this alternative: When generalizing this method to higher dimensions, it becomes necessary to compute the root of a symmetric matrix which might slow down the simulation process.