APTS Statistical Computing: Lab 2

Anthony Lee

Here are some practical problems which aim to explore and reinforce some of the course material.

Numerical differentiation

In this question, let

$$f(x_1, x_2, x_3) = \frac{(x_1 x_2 \sin(x_3) + \exp(x_1 x_2))}{x_3},$$

which is the same function investigated in the automatic differentiation section of the lecture notes.

Question 1. Use finite differencing to approximate the gradient of f at $(x_1, x_2, x_3) = (1, 2, \pi/2)$. Use all values of h in $\{2^{-i} : i \in \{0, \ldots, 60\}\}$.

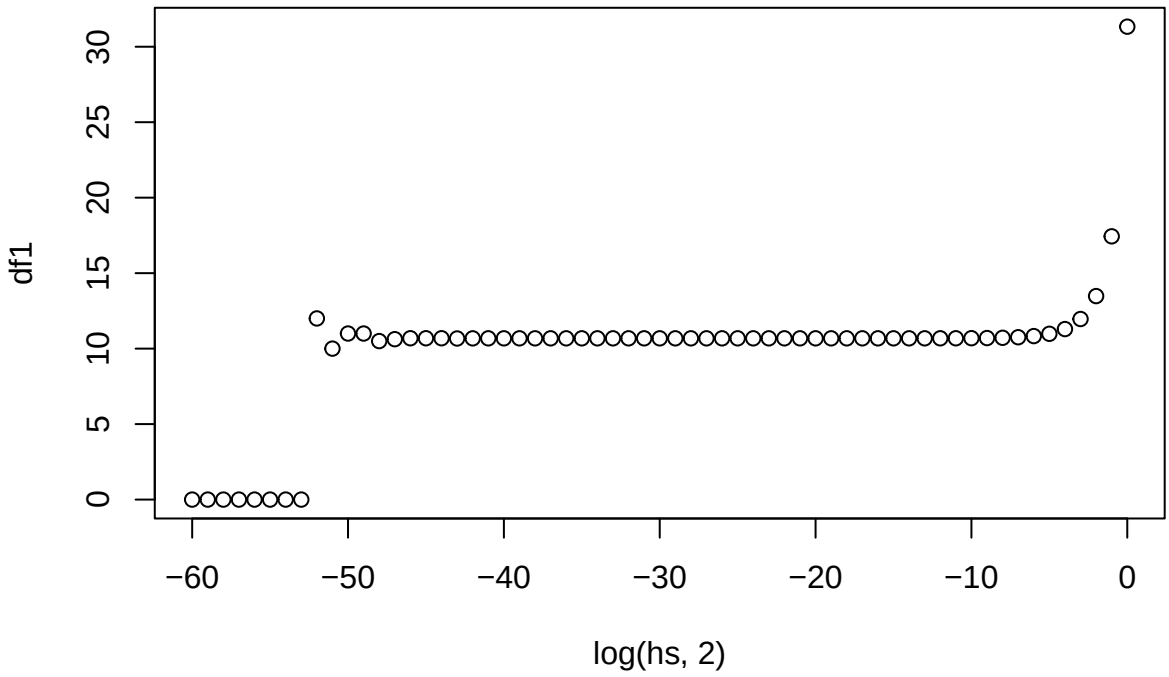
You may find it helpful to define functions of each x_i separately, with the rest fixed to the relevant values in x above, as suggested by the lecture notes. For example, we can define

```
f <- function(x) {
  (x[1]*x[2]*sin(x[3])+exp(x[1]*x[2]))/x[3]
}

x <- c(1,2,pi/2)

f1 <- Vectorize(function(x1) {
  f(c(x1,x[2:3]))
})
```

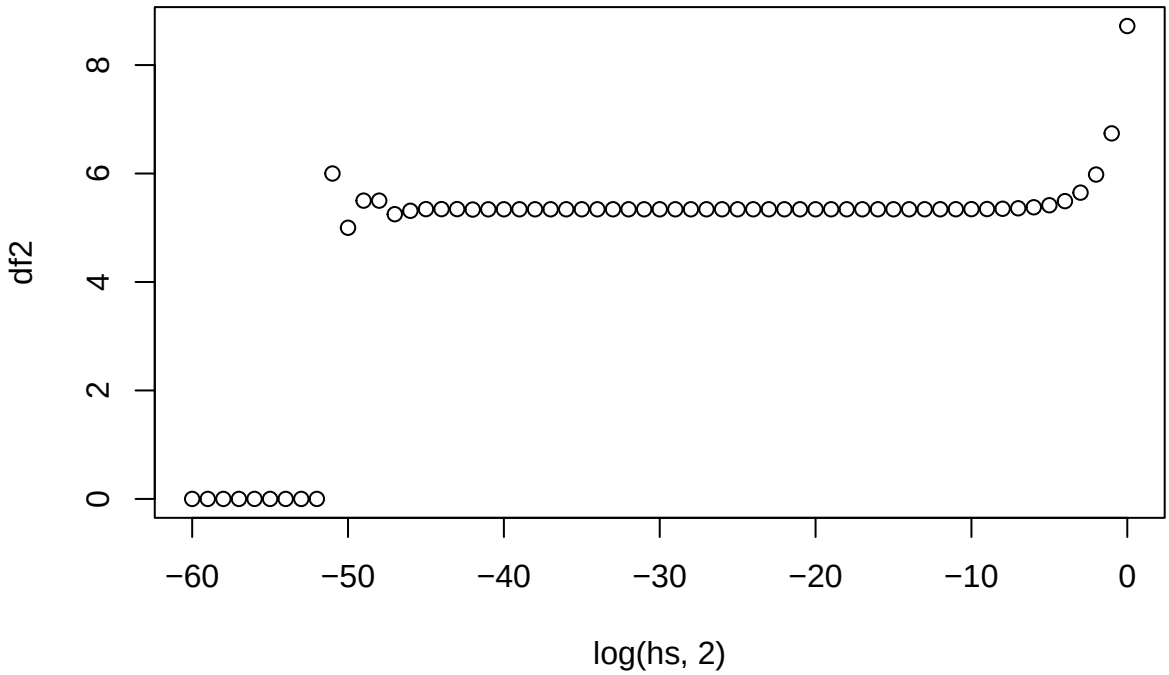
so that `f1` is the function of x_1 only, with $(x_2, x_3) = (2, \pi/2)$. Note that we use the `Vectorize` function so that we can input several values of `x1`, i.e. x_1 and the output will be the corresponding values of $f(x_1, x_2, x_3)$.

—***Solution***—

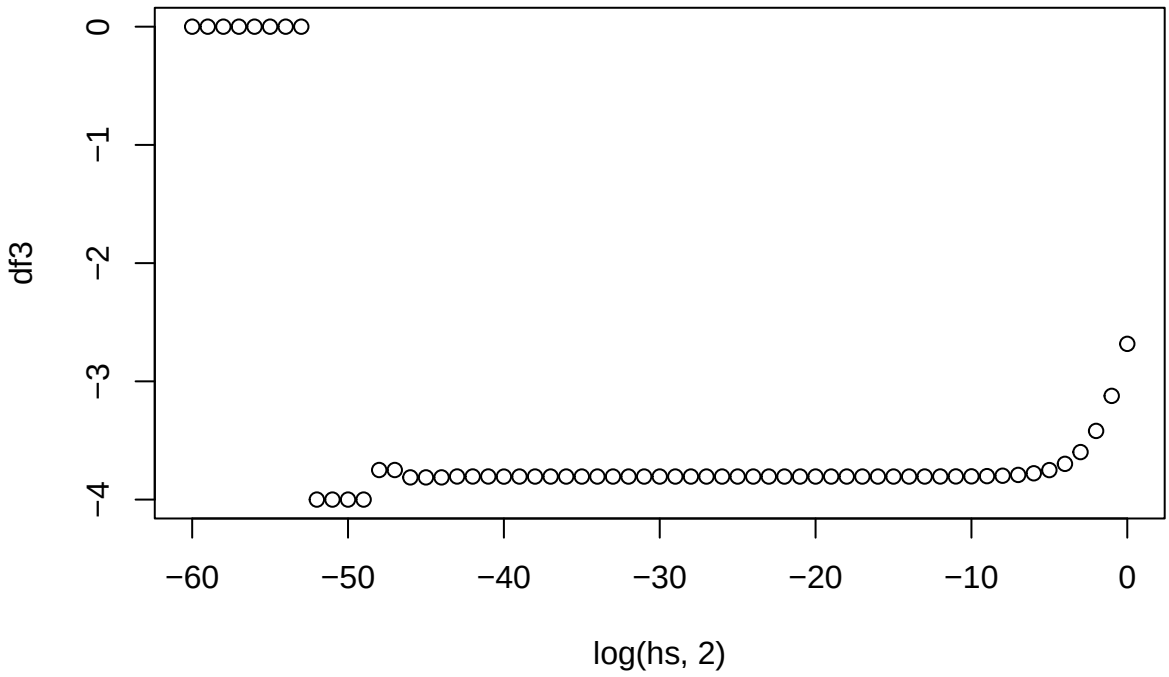
```
f2 <- Vectorize(function(x2) {
  f(c(x[1],x2,x[3]))
})

f3 <- Vectorize(function(x3) {
  f(c(x[1:2],x3))
})

hs <- 2^{-(0:60)}

df1 <- (f1(x[1]+hs)-f1(x[1]))/hs
df2 <- (f2(x[2]+hs)-f2(x[2]))/hs
df3 <- (f3(x[3]+hs)-f3(x[3]))/hs

plot(log(hs,2),df1)
```




```
plot(log(hs,2),df2)
```




```
plot(log(hs,2),df3)
```




```
df1[30]
```

```
## [1] 10.68128
```

```
df2[30]
```

```
## [1] 5.340639
```

```
df3[30]
```

```
## [1] -3.805241
```

We see that the finite difference approximations get better as h decreases until eventually becoming unstable. The actual estimates we report are from the middle range, but really any reasonable value of h gives essentially the same answer.

—***End of Solution***—

Note that one can also use the `pracma` package's `grad` function to compute gradients numerically. You may need to install the package to run the following code.

```
pracma::grad(f,x)
```

```
## [1] 10.681278  5.340639 -3.805241
```

This function, according to the documentation, uses the central difference formula mentioned in the lecture notes.

Numerical quadrature

Here we consider integrating a continuous function f using a composite rule.

We will initially look at integrating using a simple rule. The main idea is to approximate f using a degree $k-1$ interpolating polynomial p_{k-1}. This involves evaluating f at k points $x_1, \ldots, x_k$.

Polynomial interpolation

The interpolating polynomial is unique, has degree at most $k-1$, and it is convenient to express it as a Lagrange polynomial:

$$p_{k-1}(x) := \sum_{i=1}^{k} \ell_i(x) f(x_i),$$

where the ℓ_i are the Lagrange basis polynomials

$$\ell_i(x) = \prod_{j=1, j\neq i}^{k} \frac{x - x_j}{x_i - x_j} \qquad i \in \{1, \ldots, k\}.$$

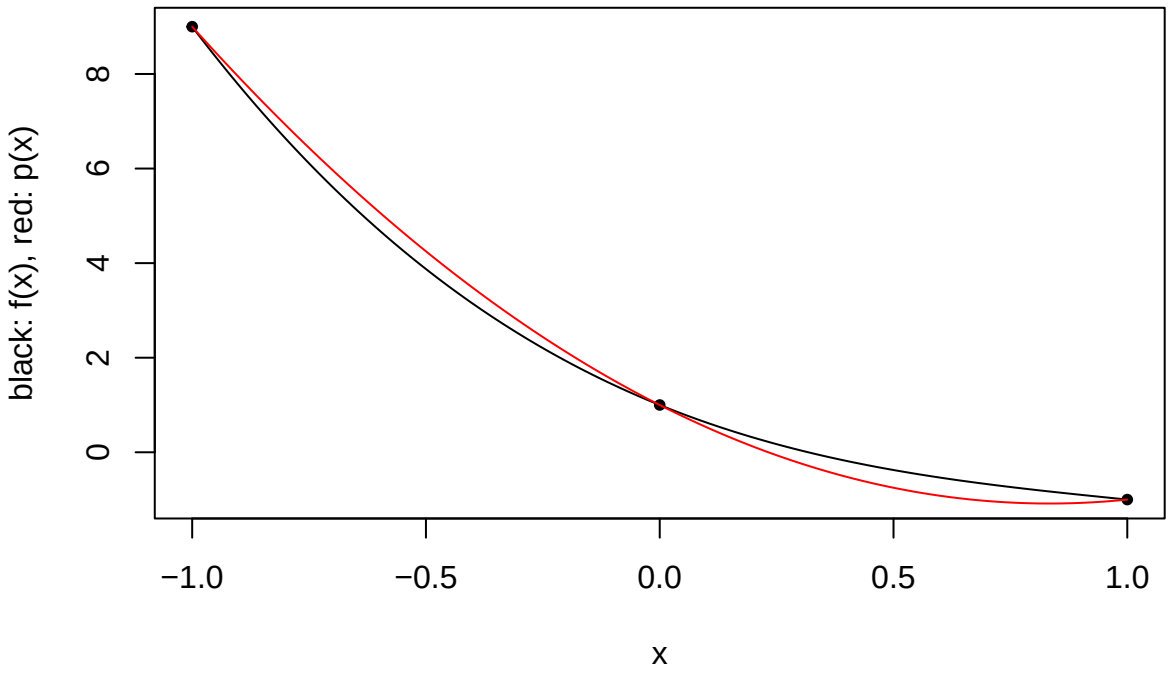
Question 2. Demonstrate empirically that if f is a degree $k-1$ polynomial then one perfectly fits f using any distinct $x_1, \ldots, x_k$. Also demonstrate that if f is not a polynomial then the placement of the $x_1, \ldots, x_k$ does matter.

The following code should be helpful.

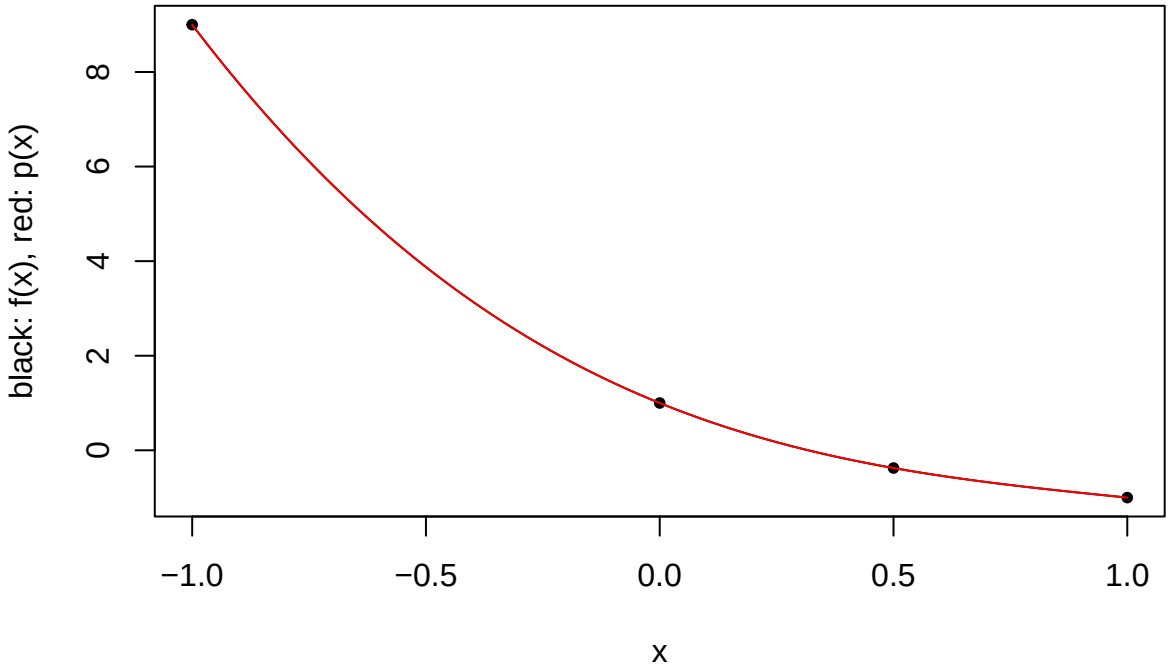
```
construct.interpolating.polynomial <- function(f, xs) {
  k <- length(xs)
  fxs <- f(xs)
  p <- function(x) {
    value <- 0
    for (i in 1:k) {
      fi <- fxs[i]
      zs <- xs[setdiff(1:k,i)]
      li <- prod((x-zs)/(xs[i]-zs))
      value <- value + fi*li
    }
    return(value)
  }
  return(p)
}

plot.polynomial.approximation <- function(f, xs, a, b) {
  p <- construct.interpolating.polynomial(f, xs)
  vs <- seq(a, b, length.out=500)
  plot(vs, f(vs), type='l', xlab="x", ylab="black: f(x), red: p(x)")
  points(xs, f(xs), pch=20)
  lines(vs, vapply(vs, p, 0), col="red")
}
```

—***Solution***—

```
a <- -1
b <- 1

f <- function(x) {
  return(-x^3 + 3*x^2 - 4*x + 1)
}

plot.polynomial.approximation(f, c(-1, 0, 1), a, b)
```

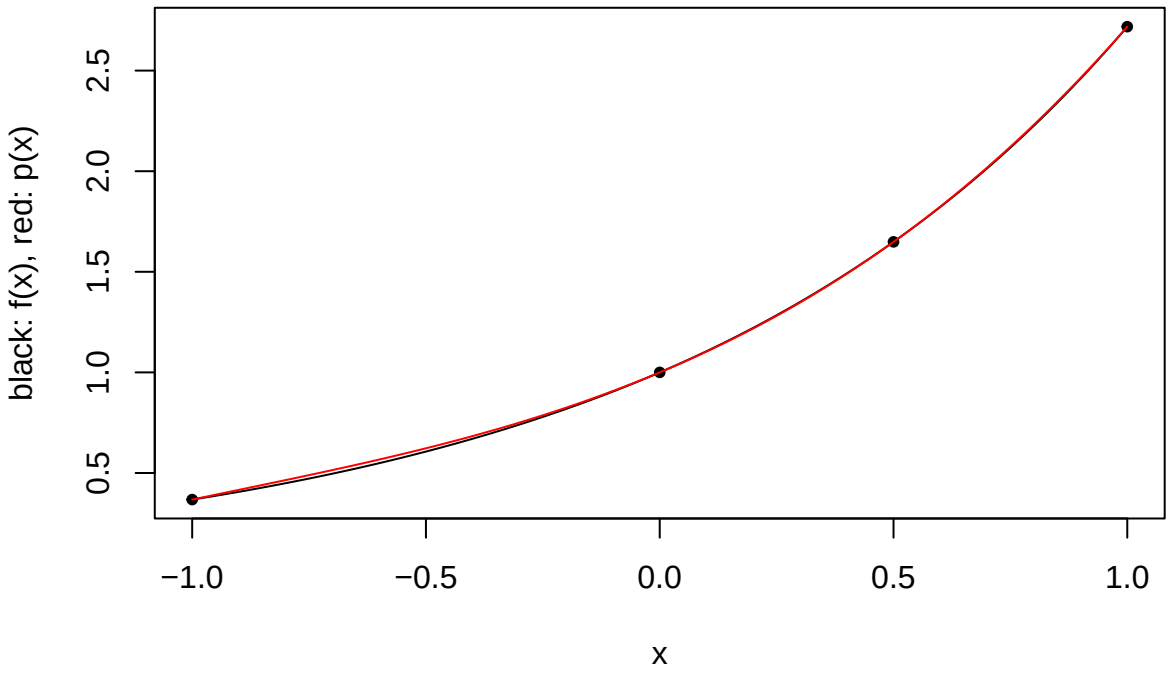



```
plot.polynomial.approximation(f, c(-1, 0, 0.5, 1), a, b)
```

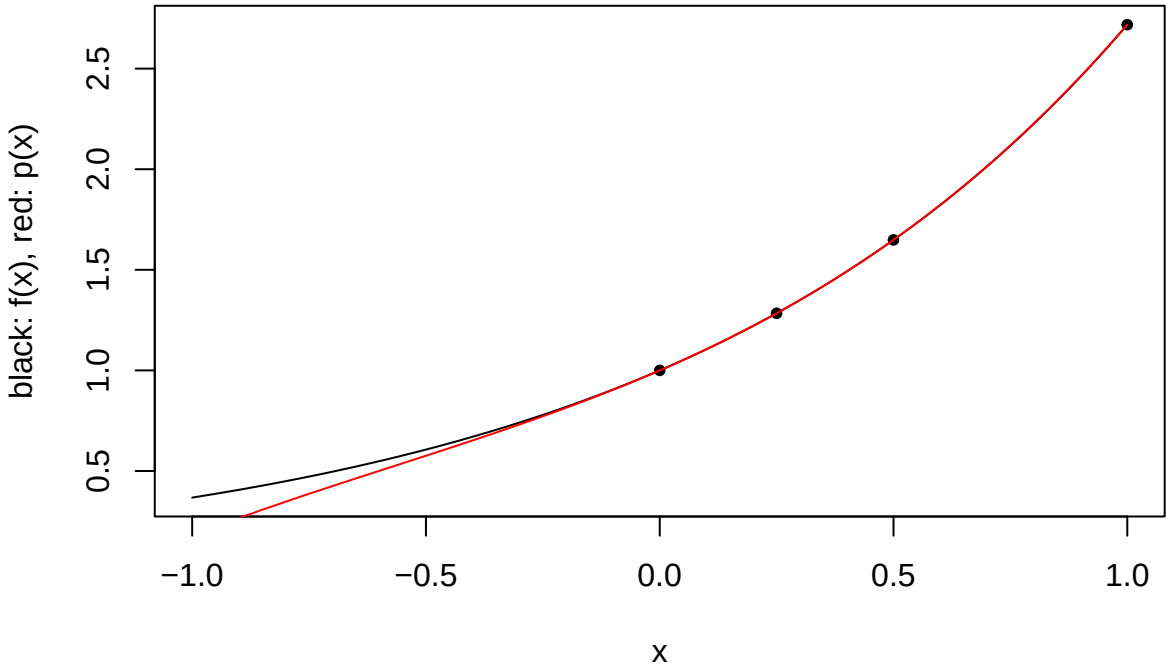


We see that indeed we retrieve (numerically) exactly the degree 3 polynomial with $k = 4$.

```
f <- exp

plot.polynomial.approximation(f, c(-1, 0, 0.5, 1), a, b)
```




```
plot.polynomial.approximation(f, c(0, 0.25, 0.5, 1), a, b)
```



We see that here the placement of the points does matter. In particular, the approximation error is larger in this example in places where we don't have a nearby function evaluation.

—***End of Solution***—

Polynomial integration

Now we use the fact that we can integrate p_{k-1} exactly. Hopefully this is close to the integral of f.

The main idea of the approximation is to write

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &\approx \int_a^b p_{k-1}(x)\mathrm{d}x \\ &= \int_a^b \sum_{i=1}^k \ell_i(x) f(x_i)\mathrm{d}x \\ &= \sum_{i=1}^k f(x_i) \int_a^b \ell_i(x)\mathrm{d}x \\ &= \sum_{i=1}^k w_i f(x_i),\end{aligned}$$

where for $i \in \{1, \ldots, k\}$, $w_i := \int_a^b \ell_i(x)\mathrm{d}x$ and we recall that $\ell_i(x) = \prod_{j=1, j\neq i}^k \frac{x-x_j}{x_i-x_j}$.

We will consider the case where the interpolation points are a, $(a+b)/2$ and b, corresponding to $k = 3$. We obtain the weights as

$$w_1 = w_2 = \frac{b-a}{6}, \qquad w_3 = \frac{2(b-a)}{3},$$

giving rise to the approximation

$$\int_a^b p_{k-1}(x)\mathrm{d}x = \frac{b-a}{6}\left\{f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right\}.$$

Question 3. Use the above to approximate the integral of cos over $[-1, 1]$ and $[-5, 5]$, and compare with the true value

$$\int_a^b \cos(x)\mathrm{d}x = \sin(b) - \sin(a).$$

—***Solution***—

```
integral.approx <- function(f,a,b) {
  (b-a)/6*(f(a)+4*f((a+b)/2)+f(b))
}
integral.approx(cos,-1,1)
```

```
## [1] 1.693535
```

```
sin(1)-sin(-1)
```

```
## [1] 1.682942
```

```
integral.approx(cos,-5,5)
```

```
## [1] 7.612207
```

```
sin(5)-sin(-5)
```

```
## [1] -1.917849
```

We see that the first approximation is not too bad but the second approximation is very inaccurate. One expects that this is because the polynomial approximation is also very inaccurate.

—***End of Solution***—

Composite polynomial integration

We have a couple of options when the approximation is inaccurate. One is to increase k, but this leads to quite complicated expressions and ultimately only makes sense for sufficiently smooth functions.

The alternative is to approximate the function f by different degree $k-1$ polynomials in subintervals. That is we write

$$f = \sum_{i=1}^{m} f \cdot \mathbf{1}_{A_i} =: \sum_{i=1}^{m} f_i,$$

where the A_i partition $[a,b]$ and in particular we can take $A_i = [a+(i-1)h, a+ih]$ with $h = (b-a)/m$.

Question 4. Use the above to approximate the integral of cos over $[-5,5]$ for different values of m, and compare with the true value

$$\int_a^b \cos(x)\mathrm{d}x = \sin(b) - \sin(a).$$

—***Solution***—

```
composite.integral.approx <- function(f,a,b,m) {
  h <- (b-a)/m
  s <- 0
  for (i in 1:m) {
    s <- s + integral.approx(f,a+(i-1)*h,a+i*h)
  }
  s
}

sin(5)-sin(-5)
```

```
## [1] -1.917849
```

```
composite.integral.approx(cos,-5,5,1)
```

```
## [1] 7.612207
```

```
composite.integral.approx(cos,-5,5,2)
```

```
## [1] -3.20152
```

```
composite.integral.approx(cos,-5,5,4)
```

```
## [1] -1.949644
```

```
composite.integral.approx(cos,-5,5,8)
```

```
## [1] -1.919553
```

```
composite.integral.approx(cos,-5,5,16)
```

```
## [1] -1.917951
```

```
composite.integral.approx(cos,-5,5,32)
```

```
## [1] -1.917855
```

```
composite.integral.approx(cos,-5,5,64)
```

```
## [1] -1.917849
```

—End of Solution—

Of course, you can and should use robustly implemented algorithms in practice, if there is no reason to do otherwise.

```
integrate(cos,-5,5)
```

```
## -1.917849 with absolute error < 1e-11
```

Importance sampling

We consider a simple Bayesian logistic regression problem with 7 predictors and an intercept. The model is

$$Y \sim \text{Bernoulli}(p(x_i;\beta)),$$

where

$$p(x_i;\beta) = \frac{1}{1+\exp(-\beta^T x_i)}.$$

Below we generate the predictors.

```
set.seed(2024)
n <- 200
p <- 8
X <- cbind(1,matrix(rnorm(n*(p-1)),n,p-1))
```

Now we sample randomly a true coefficient vector and some responses.

```
beta.true <- rnorm(p)
beta.true
```

```
## [1] -0.2465327  1.0457681 -0.5950995 -2.4267081  1.0369776  1.6367101 -1.0683578
## [8] -1.5419420
```

```
ps <- 1/(1+exp(-X%*%beta.true))
ys <- as.numeric(runif(n) < ps)
```

We can perform maximum likelihood estimation using the `glm.fit` function as follows:

```
df <- data.frame(response=ys,predictors=X)
model <- glm.fit(X, ys,family=binomial(link='logit'),intercept=FALSE)
```

but for the sake of this lab let us instead try to approximate the posterior distribution assuming each coefficient is *a priori* an independent standard normal random variable. That is, the prior distribution has density

$$\pi_0(\beta) = \prod_{i=1}^{p} N(\beta_i;0,1),$$

so that the posterior distribution is

$$\pi(\beta) \propto \pi_0(\beta) L(\beta; x_{1:n}, y_{1:n}) = \pi_0(\beta) \prod_{i=1}^{n} p(x_i;\beta)^{y_i} \{1-p(x_i;\beta)\}^{1-y_i}.$$

We can find the maximum *a posteriori* estimate, as well as the Hessian matrix associated with the log-posterior at this value of β as follows

```
log.prior <- function(beta) {
  sum(dnorm(beta,log=TRUE))
}

log.likelihood <- function(beta) {
  ps <- 1/(1+exp(-X%*%beta))
  sum(ys*log(ps)+(1-ys)*log(1-ps))
}

log.posterior <- function(beta) {
  log.prior(beta) + log.likelihood(beta)
}

optim.out <- optim(rep(0,p), log.posterior, method = "L-BFGS-B", hessian=TRUE, control=list(fnscale=-1))
H <- optim.out$hessian
beta.map <- optim.out$par
Sigma.approx <- solve(-H)
beta.map
```

```
## [1] -0.07185713  0.67251415 -0.53638139 -2.00813671  1.10551283  1.48005468
## [7] -0.95072434 -1.63515861
```

```
Sigma.approx
```

```
##               [,1]         [,2]          [,3]         [,4]          [,5]
## [1,]  0.0456296958 -0.004524148 -0.0018225182  0.004227274  0.0016661559
## [2,] -0.0045241480  0.052322902 -0.0027318041 -0.019747744  0.0126441307
## [3,] -0.0018225182 -0.002731804  0.0551743816  0.018495846 -0.0009274784
## [4,]  0.0042272744 -0.019747744  0.0184958461  0.102840364 -0.0239618059
## [5,]  0.0016661559  0.012644131 -0.0009274784 -0.023961806  0.0622587630
## [6,]  0.0003699451  0.023566146 -0.0175668308 -0.042344673  0.0211803012
## [7,] -0.0017093810 -0.013712325 -0.0003087388  0.024823573 -0.0145721982
## [8,] -0.0080932064 -0.007214051  0.0119883390  0.048323347 -0.0190354588
##               [,6]          [,7]         [,8]
## [1,]  0.0003699451 -0.0017093810 -0.008093206
## [2,]  0.0235661460 -0.0137123247 -0.007214051
## [3,] -0.0175668308 -0.0003087388  0.011988339
## [4,] -0.0423446734  0.0248235732  0.048323347
## [5,]  0.0211803012 -0.0145721982 -0.019035459
## [6,]  0.0792770063 -0.0209594247 -0.034060687
## [7,] -0.0209594247  0.0600375171  0.011031647
## [8,] -0.0340606868  0.0110316469  0.083280055
```

The quantity `beta.map` is the MAP estimate and `Sigma` can be viewed as an estimate of the posterior covariance. Indeed, if the Laplace approximation is accurate then the posterior is close to a $N(\beta_{\text{MAP}}, \Sigma)$ distribution. We will see how this approximation can be used in importance sampling.

Question 5. Complete the body of the function `is.gaussian` so that `bs` is a matrix where each of the N rows is a β sample of length p and `ws` contains the associated weights for each β. The weights should sum to 1. The proposal distribution should be normal with mean `mu` and covariance matrix `Sigma` and the `log.target` function computes the log posterior density up to a normalizing constant.

It should be helpful to know that if $X \sim N(\mu, \Sigma)$ then $X = \mu + AZ$ where A is any matrix such that $AA^T = \Sigma$. The log density of $N(\mu, \Sigma)$ is

$$\log N(x;\mu,\Sigma) = C - \frac{1}{2}\log\det(\Sigma) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu),$$

where C is a constant.

```
is.gaussian <- function(log.target, mu, Sigma, N) {
  p <- length(mu)
  bs <- matrix(0,N,p)
  log.weights <- rep(0,N)
  ###
  ### more code here
  ###
  ws <- exp(log.weights-max(log.weights))
  ws <- ws/sum(ws)
  return(list(bs=bs,ws=ws,ess=1/sum(ws^2)))
}
```

—***Solution***—

```
is.gaussian <- function(log.target, mu, Sigma, N) {
  p <- length(mu)
  bs <- matrix(0,N,p)
  log.weights <- rep(0,N)
  A <- t(chol(Sigma))
  Sigma.inv <- solve(Sigma)
  Sigma.logdet <- determinant(Sigma,logarithm = TRUE)$modulus[1]
  for (i in 1:N) {
    b <- mu + A%*%rnorm(p)
    bs[i,] <- b
    lQd <- -0.5*Sigma.logdet-0.5*t(b-mu)%*%Sigma.inv%*%(b-mu)
    lPd <- log.target(b)
    log.weights[i] <- lPd - lQd
  }
  ws <- exp(log.weights-max(log.weights))
  ws <- ws/sum(ws)
  return(list(bs=bs,ws=ws,ess=1/sum(ws^2)))
}
```

—***End of Solution***—

You can test your code using the following function.

```
## approximates the mean and covariance matrix associated with the weighted
## samples in output
approx.mean.var <- function(output) {
  p <- length(output$bs[1,])
  bs <- output$bs
  ws <- output$ws
  ms <- rep(0,p)
  for (i in 1:p) {
    ms[i] <- sum(ws*bs[,i])
  }
  vs <- matrix(0,p,p)
  for (i in 1:p) {
    for (j in 1:p) {
      vs[i,j] <- sum(ws*bs[,i]*bs[,j])-ms[i]*ms[j]
```

```
    }
  }
  return(list(ms=ms,vs=vs))
}
```

```
test.mu <- c(1,2)
test.Sigma <- matrix(c(1,0.2,0.2,0.8),2,2)
test.Sigma.logdet <- determinant(test.Sigma,logarithm = TRUE)$modulus[1]
test.Sigma.inv <- solve(test.Sigma)
test.log.target <- function(b) {
  -0.5*test.Sigma.logdet-0.5*t(b-test.mu)%*%test.Sigma.inv%*%(b-test.mu)
}

test.output <- is.gaussian(test.log.target, c(-1,1), matrix(c(2,-0.5,-0.5,2),2,2), 100000)
approx.mean.var(test.output)
```

```
## $ms
## [1] 1.042352 2.028940
##
## $vs
##            [,1]      [,2]
## [1,] 1.0665911 0.2616807
## [2,] 0.2616807 0.8351101
```

Question 6. Now use your Gaussian importance sampling code to investigate the importance sampling algorithms associated with different choices of μ and Σ. In particular:

- $\mu = 0$, $\Sigma = I_p$
- $\mu = \beta_{\text{MAP}}$, $\Sigma = I_p$
- $\mu = \beta_{\text{MAP}}$, $\Sigma = \text{diag}(\Sigma_{\text{approx}})$
- $\mu = \beta_{\text{MAP}}$, $\Sigma = \Sigma_{\text{approx}}$.

It is often useful to look at the effective sample size as a measure of the quality of the samples.

—***Solution***—

```
N <- 100000
simple <- is.gaussian(log.posterior,rep(0,p),diag(p),N)
simple$ess
```

```
## [1] 3.877375
```

```
approx.mean.var(simple)$ms
```

```
## [1]  0.1180896  0.8272556 -0.4113392 -1.5109677  1.0140454  1.1732067 -0.6391444
## [8] -1.3323144
```

```
centred <- is.gaussian(log.posterior,beta.map,diag(p),N)
centred$ess
```

```
## [1] 30.49446
```

```
approx.mean.var(centred)$ms
```

```
## [1] -0.009608475  0.684061808 -0.538096084 -2.040697155  1.151022490
## [6]  1.579248961 -1.022537973 -1.685236895
```

```
centred.scaled <- is.gaussian(log.posterior,beta.map,diag(diag(Sigma.approx)),N)
centred.scaled$ess
```

```
## [1] 11033.37
```

```
approx.mean.var(centred.scaled)$ms
```

```
## [1] -0.07641854  0.69829609 -0.55534479 -2.09209298  1.15945912  1.54341104
## [7] -0.99713316 -1.71194082
```

```
Laplace <- is.gaussian(log.posterior,beta.map,Sigma.approx,N)
Laplace$ess
```

```
## [1] 70625.07
```

```
approx.mean.var(Laplace)$ms
```

```
## [1] -0.0744150  0.7100549 -0.5627055 -2.1148314  1.1736061  1.5613853 -1.0100201
## [8] -1.7284975
```

We see that each successive change improves the quality of the samples, as measured by the effective sample size. The benefit of using the approximation of the full covariance of the posterior is quite dramatic in comparison to only scaling the variances appropriately.

—***End of Solution***—