Carson Slater

6376 Homework 12 Solutions

```
source(here::here("helpers.R"))
```

Preliminary Citations

For problem 1(j), we were experiencing trouble getting the data to be wrangled into the format necessary to get the code from our assignment to run. In order to produce the vector field visualization, Jacob found that if you add a `t()` when generating the local linear approximation, our code then ran. Otherwise, I did this assignment on my own.

Question 1

(a)

```
f <- function(x) (x-1) * x * (x+1)
df <- function(x) 3*x^2 - 1
x <- 0.25
ep <- .Machine$double.eps^0.50

# analytical
df(x)
```

```
[1] -0.8125
```

```
# forward difference
(f(x + ep) - f(x))/ep
```

```
[1] -0.8125
```

```
# backward difference
(f(x) - f(x - ep))/ep
```

```
[1] -0.8125
```

```r
# central difference
(f(x + ep) - f(x - ep))/(2*ep)
```

```
[1] -0.8125
```

(b)

```r
d <- function(f, ep = .Machine$double.eps^0.50, method = "central") {
  switch (method,
    "central"  = function(x) (f(x + ep) - f(x - ep))/(2*ep),
    "forward"  = function(x) (f(x + ep) - f(x))/ep,
    "backward" = function(x) (f(x) - f(x - ep))/ep,
  )
}
```

```r
d(f)(.25)
```

```
[1] -0.8125
```

```r
f <- function(x) sin(1/x)
df <- function(x) cos(1/x)*-1/x^2

# test for function sin(1/x)
x0 <- .001
f2 <- function(x) sin(1/x)
c(
  "true" = cos(1/x0) * (-1)*(x0^-2),
  "central" = d(f2, method = "central")(x0),
  "forward" = d(f2, method = "forward")(x0),
  "backward" = d(f2, method = "backward")(x0)
)
```

```
     true   central   forward  backward
-562379.1 -562358.1 -568510.3 -556205.8
```

(c)

```r
f <- function(v) {
  x <- v[1]; y <- v[2]
  x^2 + y^2
}

df <- function(v){
  c(2*v[1], 2*v[2])
}

df(c(.25, .50))
```

```
[1] 0.5 1.0
```

```r
# partial with respect to x
(f(c(0.25 + ep, 0.5)) - f(c(0.25 - ep, 0.5)))/(2*ep)
```

```
[1] 0.5
```

```r
# partial with respect to y
(f(c(0.25, 0.5 + ep)) - f(c(0.25, 0.5 - ep)))/(2*ep)
```

```
[1] 1
```

(d)

```r
grad <- function(f, ep = .Machine$double.eps^0.50, method = "central") {
  function(v) {
    n <- length(v)
    e <- diag(n)
    df <- numeric(n)
    switch(method,
        "central"  = for (i in 1:n) df[i] <- (f(v + ep*e[,i]) - f(v - ep*e[,i]))/(2*ep),
        "forward"  = for (i in 1:n) df[i] <- (f(v + ep*e[,i]) - f(v))/ep,
        "backward" = for (i in 1:n) df[i] <- (f(v) - f(v - ep*e[,i]))/ep
    )
    df
  }
}
```

```r
grad(f)(c(0.25, 0.5))
```

```
[1] 0.5 1.0
```

```r
x0 <- c(.25, .25)
f2 <- function(v) sin(v[1]) * sin(v[2])
c(cos(x0[1])*sin(x0[2]), sin(x0[1])*cos(x0[2]))
```

```
[1] 0.2397128 0.2397128
```

```r
# testing gradient (non-vectorized)
grad(f2, method = "central")(x0)
```

```
[1] 0.2397128 0.2397128
```

```r
grad(f2, method = "forward")(x0)
```

```
[1] 0.2397128 0.2397128
```

```r
grad(f2, method = "backward")(x0)
```

```
[1] 0.2397128 0.2397128
```

(e)

```r
# vectorized gradient
gradient <- function(f, ep = .Machine$double.eps^0.50, method = "central") {
  function(mat) {
    if (is.null(dim(mat))) return(grad(f, ep, method)(mat))
    column_names <- colnames(mat)
    g <- apply(mat, 1, grad(f, ep, method)) |> t()
    colnames(g) <- column_names
    g
  }
}
```

```r
(mat <- cbind("x" = c(.25, .50, 0), "y" = c(.50, .25, 0)))
```

```
        x    y
[1,] 0.25 0.50
[2,] 0.50 0.25
[3,] 0.00 0.00
```

```r
apply(mat, 1, df) |> t()
```

```
       x   y
[1,] 0.5 1.0
[2,] 1.0 0.5
[3,] 0.0 0.0
```

```r
# testing gradient (vectorized)
gradient(f)(mat)
```

```
       x   y
[1,] 0.5 1.0
[2,] 1.0 0.5
[3,] 0.0 0.0
```

(f)

```r
# code from assignment
f <- function(v) {
  x <- v[1]; y <- v[2]
  c(x^2 + y^2, sin(x)*sin(y), x*y)
}
# symbolically
x0 <- c(.25, .50)
x <- x0[1]; y <- x0[2]
# true value, for reference
df <- function(v) {
  x <- v[1]; y <- v[2]
  rbind(
    c(2*x, 2*y),
    c(cos(x)*sin(y), sin(x)*cos(y)),
```

```
    c(y, x)
  )
}
```

```
# vectorized jacobian
jacobian <- function(f, ep = .Machine$double.eps^0.50, method = "central") {
  function(mat) {
    column_names <- colnames(mat)

    # for single vector inputs
    if (is.null(dim(mat))) {
      m <- length(f(mat)); n <- length(mat)
      my_matrix <- matrix(0, nrow = m, ncol = n)
      e <- diag(n)
      for (i in 1:m) {
      df <- numeric(n)
      my_matrix[i,] <- switch(method,
        "central"  = {
          for (j in 1:n) df[j] <- (f(mat + ep*e[,j])[i] - f(mat - ep*e[,j])[i]) / (2*ep)
          df
        },
        "forward"  = {
          for (j in 1:n) df[i] <- (f(mat + ep*e[,j])[i] - f(mat)[i]) / ep
          df
        },
        "backward" = {
          for (j in 1:n) df[j] <- (f(mat)[i] - f(mat - ep*e[,j])[i]) / ep
          df
        }
      )
    }
    colnames(my_matrix) <- column_names
    my_matrix
    } else {

      # for input matrices and a vectorized version of the function
      m <- length(f(mat[1,])); n <- ncol(mat); s <- nrow(mat)
      e <- diag(n)
      mat_list <- list()
      for (k in 1:s) {
        sheet <- matrix(0, nrow = m, ncol = n)
        for (i in 1:m) {
```

```
          df <- numeric(n)
          sheet[i,] <- switch(method,
            "central"  = {
              for (j in 1:n) df[j] <- (f(mat[k,] + ep*e[,j])[i] - f(mat[k,] - ep*e[,j])[i]) /
              df
            },
            "forward"  = {
              for (j in 1:n) df[i] <- (f(mat[k,] + ep*e[,j])[i] - f(mat[k,])[i]) / ep
              df
            },
            "backward" = {
              for (j in 1:n) df[j] <- (f(mat[k,])[i] - f(mat[k,] - ep*e[,j])[i]) / ep
              df
            }
          )
        }
        mat_list[[k]] <- sheet
      }
      final_array <- array(unlist(mat_list), dim = c(m, n, s))
      colnames(final_array) <- column_names
      final_array
    }
  }
}
```

```
# testing jacobian (non-vectorized)
df(x0)
```

```
          [,1]      [,2]
[1,] 0.5000000 1.0000000
[2,] 0.4645214 0.2171174
[3,] 0.5000000 0.2500000
```

```
jacobian(f)(x0)
```

```
          [,1]      [,2]
[1,] 0.5000000 1.0000000
[2,] 0.4645214 0.2171174
[3,] 0.5000000 0.2500000
```

```r
# testing jacobian (vectorized)
(mat <- cbind("x" = c(.25, .50, 0), "y" = c(.50, .25, 0)))
```

```
        x    y
[1,] 0.25 0.50
[2,] 0.50 0.25
[3,] 0.00 0.00
```

```r
apply(mat, 1, df, simplify = FALSE)
```

```
[[1]]
             x         y
[1,] 0.5000000 1.0000000
[2,] 0.4645214 0.2171174
[3,] 0.5000000 0.2500000

[[2]]
             x         y
[1,] 1.0000000 0.5000000
[2,] 0.2171174 0.4645214
[3,] 0.2500000 0.5000000

[[3]]
     x y
[1,] 0 0
[2,] 0 0
[3,] 0 0
```

```r
jacobian(f)(mat) |> asplit(3)
```

```
[[1]]
             x         y
[1,] 0.5000000 1.0000000
[2,] 0.4645214 0.2171174
[3,] 0.5000000 0.2500000

[[2]]
             x         y
[1,] 1.0000000 0.5000000
[2,] 0.2171174 0.4645214
```

```
[3,] 0.2500000 0.5000000

[[3]]
     x y
[1,] 0 0
[2,] 0 0
[3,] 0 0
```

(g)

```r
hessian <- function(f, ep = .Machine$double.eps^0.50, method = "central") {
  function(v) jacobian(gradient(f, ep, method))(v)
}
```

```r
f <- function(v) {
  x <- v[1]; y <- v[2]
  x^2 + y^2
}

hessian(f)(c(.25, .50))
```

```
     [,1] [,2]
[1,]    2    0
[2,]    0    2
```

(h)

```r
L <- function(f, x0, ep = .Machine$double.eps^0.50, method = "central") {
  function(x) f(x0) + gradient(f, ep, method)(x0)*(x - x0)
}
```

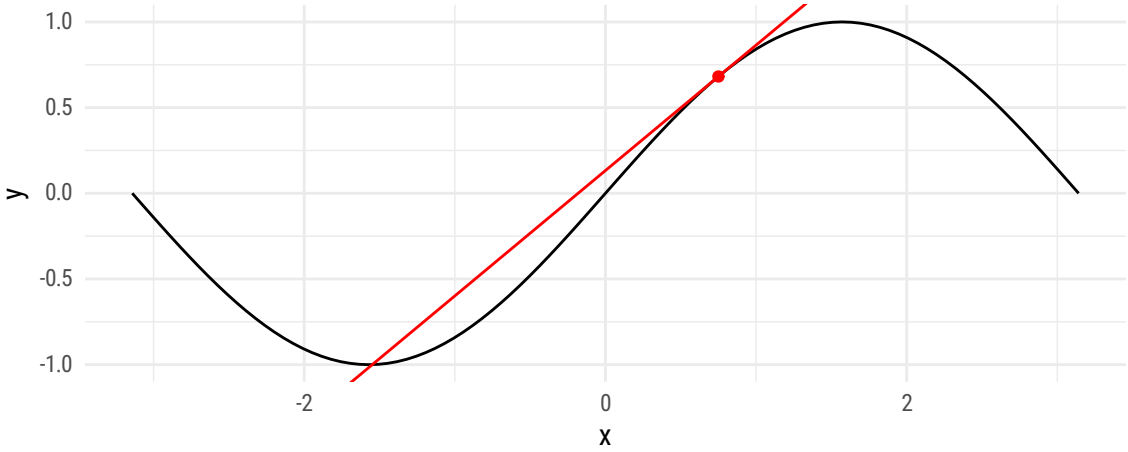
```r
f <- sin
x0 <- .75
x <- 1
f(x)
```

```
[1] 0.841471
```

```
L(f, x0)(x)
```

```
[1] 0.864561
```

```
ggplot() +
  geom_function(fun = f, xlim = c(-pi, pi)) +
  geom_function(fun = L(f, x0), xlim = c(-pi, pi), color = "red") +
  annotate("point", x = x0, y = L(f, x0)(x0), color = "red") +
  coord_cartesian(ylim = c(-1,1))
```



(i)

```
L <- function(f, x0, ep = .Machine$double.eps^0.50, method = "central") {
  function(x) f(x0) + ip(gradient(f, ep, method)(x0),(x - x0))
}
```

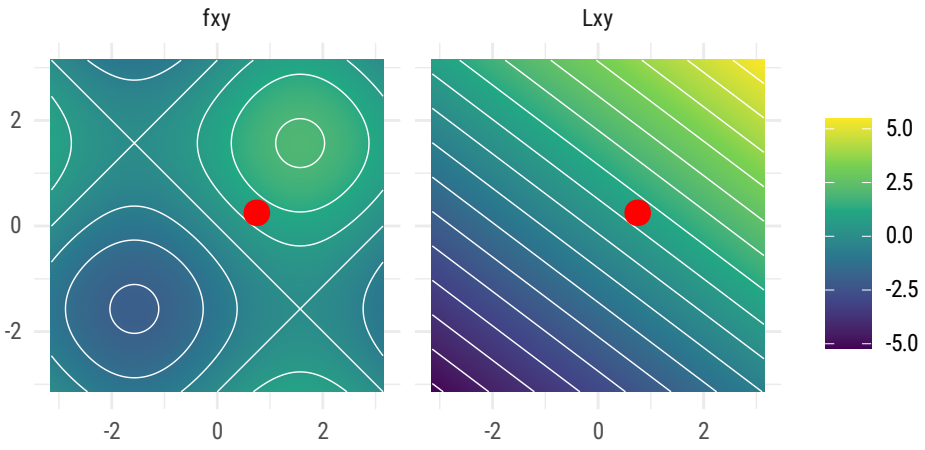
```
f <- function(v) {
  if (is.matrix(v)) return(apply(v, 1, f))
  x <- v[1]; y <- v[2]
  sin(x) + sin(y)
}

x0 <- c(.75, .25)
x <- c(1, 1)
f(x)
```

```
[1] 1.682942
```

```
L(f, x0)(x)
```

```
[1] 1.838649
```

```
expand_grid("x" = seq(-pi, pi, length.out = 251), "y" = x) |>
  mutate(
    fxy = apply(cbind(x,y), 1, f),
    Lxy = apply(cbind(x,y), 1, L(f, x0)),
  ) |>
  pivot_longer(c(fxy, Lxy), names_to = "f", values_to = "f_val") |>
  ggplot(aes(x, y)) +
    geom_raster(aes(fill = f_val)) +
    geom_contour(aes(z = f_val), color = "white", bins = 20, linewidth = .25) +
    annotate("point", x = x0[1], y = x0[2], color = "red", size = 4) + facet_wrap(~ f) +
    coord_equal() +
    theme(
      axis.title = element_blank(),
      legend.title = element_blank()
      )
```



(j)

```
L <- function(f, x0, ep = .Machine$double.eps^0.50, method = "central") {
  function(x) {
```

```
    if (length(dim(x)) == 2) apply(x, 1, \(x) f(x0) + (jacobian(f, ep, method)(x0) %*% (x - x
    else (f(x0) + (jacobian(f, ep, method)(x0) %*% (x - x0))) |> drop()
  }
}
```

```
f <- function(v) {
  if (is.matrix(v)) return(apply(v, 1, f) |> t())
  x <- v[1]; y <- v[2]
  c(sin(5*x), sin(5*y))
}
x0 <- c(1, 1)
x <- c(1, 1)

f(x)
```

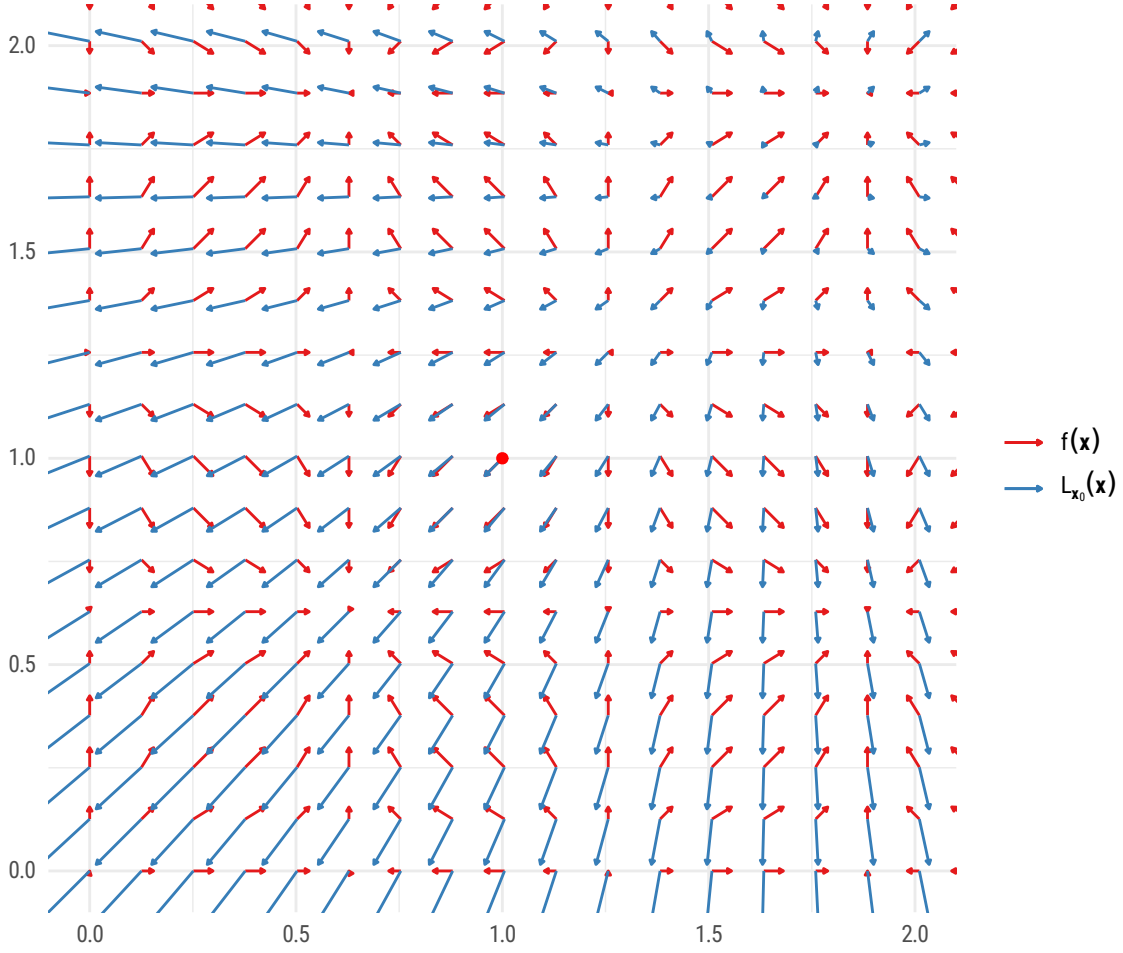
```
[1] -0.9589243 -0.9589243
```

```
L(f, x0)(x)
```

```
[1] -0.9589243 -0.9589243
```

```
s <- .05 # scale gradient for visuals
expand_grid("x" = seq(-pi, pi, length.out = 51), "y" = x) |>
  mutate(
    f = (s*f(cbind(x, y))) |> as.data.frame()  |> set_names(c("f_xend","f_yend")),
    L = (s*L(f, x0)(cbind(x, y))) |> t() |> as.data.frame() |> set_names(c("L_xend","L_yend")
    ) |>
  unpack(f) |>
  unpack(L) |>
  ggplot(aes(x, y)) +
  geom_segment(
    aes(xend = x + f_xend, yend = y + f_yend, color = "f"),
    arrow = arrow(length = unit(0.025, "inches"), type = "closed")
    )+
  geom_segment(
    aes(xend = x + L_xend, yend = y + L_yend, color = "L"),
    arrow = arrow(length = unit(0.025, "inches"), type = "closed")
    ) +
  annotate("point", x = x0[1], y = x0[2], color = "red") +
  scale_color_brewer(palette = 6, type = "qual",
```

```
                    breaks = c("f", "L"),
                    labels = list(expression(f(bold(x))),
                                  expression(L[bold(x)[0]](bold(x))))) +
coord_equal(xlim = c(0, 2), ylim = c(0, 2)) +
theme(legend.title = element_blank(), axis.title = element_blank())
```

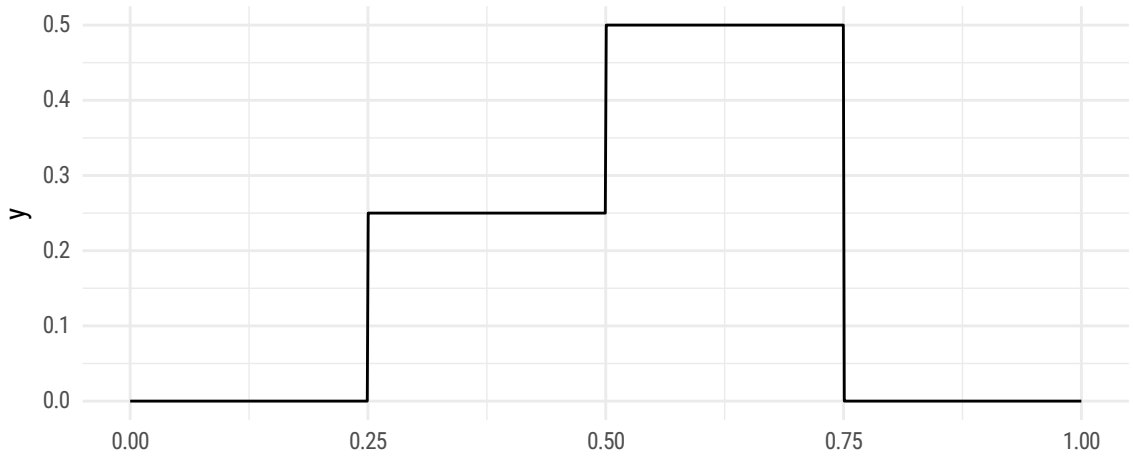


Question 2

(a)

```
# function to create integrand for coefficients
integrand <- function(a_coef = TRUE, f, L, k) {
  function(x) {
    if (a_coef == FALSE) sin(2*pi*k*x/L)*f(x)
    else cos(2*pi*k*x/L)*f(x)
  }
}

fourier_approx <- function(f, N, L) {
  function(x) {
    map_vec(x, .f = \(x) {
      a_terms <- c(
      (1/L)*integrate(f, 0, L)$value,
      map_vec(.x = 1:N,
              .f = \(k) cos(2*pi*k*x/L)*(2/L)*integrate(
                  integrand(a_coef = TRUE, f, L, k), 0, L
                )$value
              )
      )
      b_terms <- map_vec(1:N, .f = \(k) sin(2*pi*k*x/L)*(2/L)*integrate(
               integrand(a_coef = FALSE, f, L, k), 0, L
             )$value
           )
      sum(a_terms) + sum(b_terms)
      }
    )
  }
}
```

```
indicator <- function(x, L, U) (L <= x) & (x < U)
f <- function(x) .25*indicator(x, .25, .50) + .50*indicator(x, .50, .75)
ggplot() + geom_function(fun = f, xlim = c(0,1), n = 1e3)
```




```
f(.4)
```

```
[1] 0.25
```

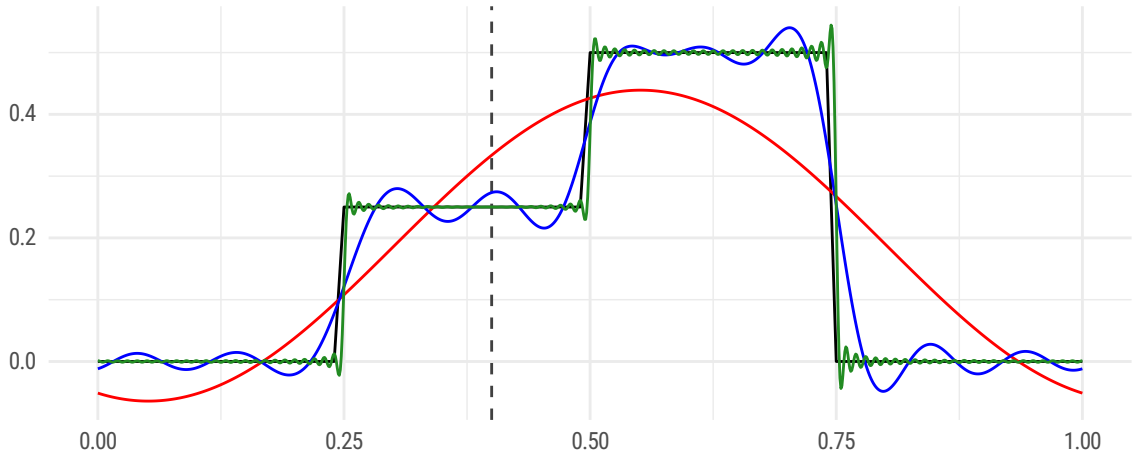
```
fourier_approx(f, 2, 1)(.4)
```

```
[1] 0.2581814
```

```
1:20 |> map_dbl( (\(N) fourier_approx(f, N, 1)(.4)) ) |> round(digits = 4)
```

```
 [1] 0.3339 0.2582 0.2575 0.2575 0.2098 0.2254 0.2467 0.2467 0.2734 0.2734
[11] 0.2516 0.2516 0.2401 0.2334 0.2493 0.2493 0.2494 0.2578 0.2501 0.2501
```

```
ggplot() +
  geom_function(fun = f) +
  geom_vline(xintercept = .40, color = "gray25", linetype = "dashed") +
  geom_function(fun = fourier_approx(f, 1, 1), color = "red", n = 1e3) +
  geom_function(fun = fourier_approx(f, 10, 1), color = "blue", n = 1e3) +
  geom_function(fun = fourier_approx(f, 100, 1), color = "forestgreen", n = 1e3) +
  scale_x_continuous(limits = c(0,1)) + theme(axis.title = element_blank())
```

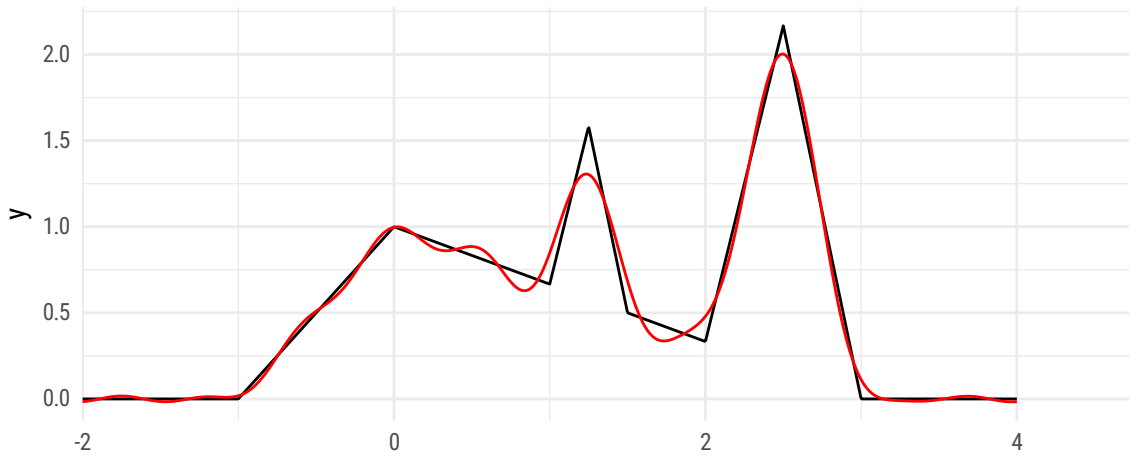


(b)

```
# changing it so it works on a specified interval (l, u)
fourier_approx_lu <- function(f, N, l, u) {
  function(x){
      L <- (u - l)
      map_vec(x, .f = \(x) {
        a_terms <- c(
          (1/L)*integrate(f, l, u)$value,
          map_vec(.x = 1:N,
                  .f = \(k) cos(2*pi*k*x/L)*(2/L)*integrate(
                      integrand(a_coef = TRUE, f, L, k), l, u
                    )$value
                  )
        )
        b_terms <- map_vec(1:N, .f = \(k) sin(2*pi*k*x/L)*(2/L)*integrate(
                        integrand(a_coef = FALSE, f, L, k), l, u
                      )$value
                    )
        sum(a_terms) + sum(b_terms)
      }
    )
  }
}
```

```
make_triangle_function_over_range <- function(a, b, h = 1, m = (a+b)/2) {
  function(x) {
```

```
    (a*h - h*x)/(a-m) * indicator(x, a, m) +
    (b*h - h*x)/(b-m) * indicator(x, m, b)
  }
}

f <- function(x) {
  make_triangle_function_over_range(1, 1.5, h = 1)(x) +
  make_triangle_function_over_range(-1, 3, m = 0)(x) +
  make_triangle_function_over_range(2, 3, 2)(x)
}

xlim <- c(-2,4)

# showing it works over a specified interval (l, u)
ggplot() +
  geom_function(fun = f, n = 1001) +
  geom_function(fun = fourier_approx_lu(f, 10, xlim[1], xlim[2]), color = "red", n = 1e3) +
  scale_x_continuous(limits = xlim, expand = expansion(c(0, .12)))
```



Question 3

(a)

```
x <- 1:3

x |> lp_norm()
```

```
[1] 3.741657
```

```
x |>
  dft(scale = TRUE) |>
  lp_norm()
```

```
[1] 3.741657
```

(b)

```
odd_even_perm_mat <- function(n) {
  p <- diag(n)
  rbind(p[seq(1, n, 2),], p[seq(2, n, 2),])
}
```

```
odd_even_perm_mat(4)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    0    1    0
[3,]    0    1    0    0
[4,]    0    0    0    1
```

```
odd_even_perm_mat(4) %*% 1:4
```

```
     [,1]
[1,]    1
[2,]    3
[3,]    2
[4,]    4
```

```
odd_even_perm_mat(5) %*% 1:5
```

```
     [,1]
[1,]    1
[2,]    3
[3,]    5
[4,]    2
[5,]    4
```

(c)

```
fft2 <- function(x) {
  n <- length(x)

  # compute xhats
  if (n == 1) x
  else {
    xo_hat <- x[seq(1, n, 2)] |> fft2()
    xe_hat <- x[seq(2, n, 2)] |> fft2()

    omega_xehat <- xe_hat * sapply(0:(floor(n/2)-1), \(j) (exp((-j*2*pi*1i)/n)))

    c(xo_hat + omega_xehat, xo_hat - omega_xehat)
  }
}
```

```
fft(1:8) |> round(digits = 3)
```

```
[1] 36+0.000i -4+9.657i -4+4.000i -4+1.657i -4+0.000i -4-1.657i -4-4.000i
[8] -4-9.657i
```

```
fft2(1:8) |> round(digits = 3)
```

```
[1] 36+0.000i -4+9.657i -4+4.000i -4+1.657i -4+0.000i -4-1.657i -4-4.000i
[8] -4-9.657i
```

(d)

```
bench::mark(
  dft(1:2^12),
  fft2(1:2^12),
  fft(1:2^12),
  check = FALSE,
  relative = TRUE
)
```

```
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
```

```
# A tibble: 3 x 6
  expression      min median `itr/sec` mem_alloc `gc/sec`
  <bch:expr>    <dbl>  <dbl>     <dbl>     <dbl>    <dbl>
1 dft(1:2^12)  23776. 20110.       1      13304.      1
2 fft2(1:2^12)  2935.  2509.      7.91      39.2     6.26
3 fft(1:2^12)      1      1   17457.         1       1.31
```

Question 4

(a)

```
logistic_seq <- function(n, seed, r = 4) {
  x <- c(seed, numeric(n-1))
  for (i in seq(2, n)) {
    x[i] <- r*x[i-1]*(1-x[i-1])
  }
  x
}
```

```
round(logistic_seq(10, .1, r = 4), 3)
```

```
 [1] 0.100 0.360 0.922 0.289 0.822 0.585 0.971 0.113 0.402 0.962
```

(b)

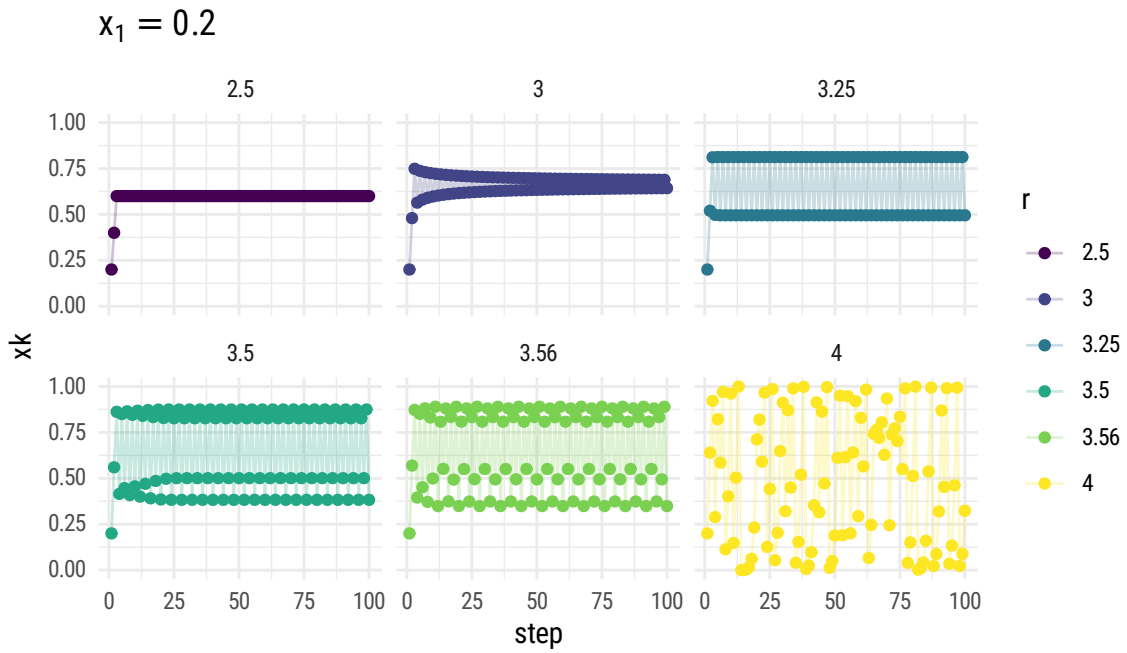
```
r <- c(2.5, 3, 3.25, 3.5, 3.56, 4)
list_of_seqs <- lapply(r,
                       \(r) data.frame(
                         "step" = 1:100,
                         "xk" = logistic_seq(100, 0.2, r),
                         "r" = as.factor(r)
                         )
                       )
lapply(list_of_seqs, tail, 2)
```

```
[[1]]
    step  xk   r
99    99 0.6 2.5
100  100 0.6 2.5

[[2]]
    step        xk r
99    99 0.6887757 3
100  100 0.6430912 3
```

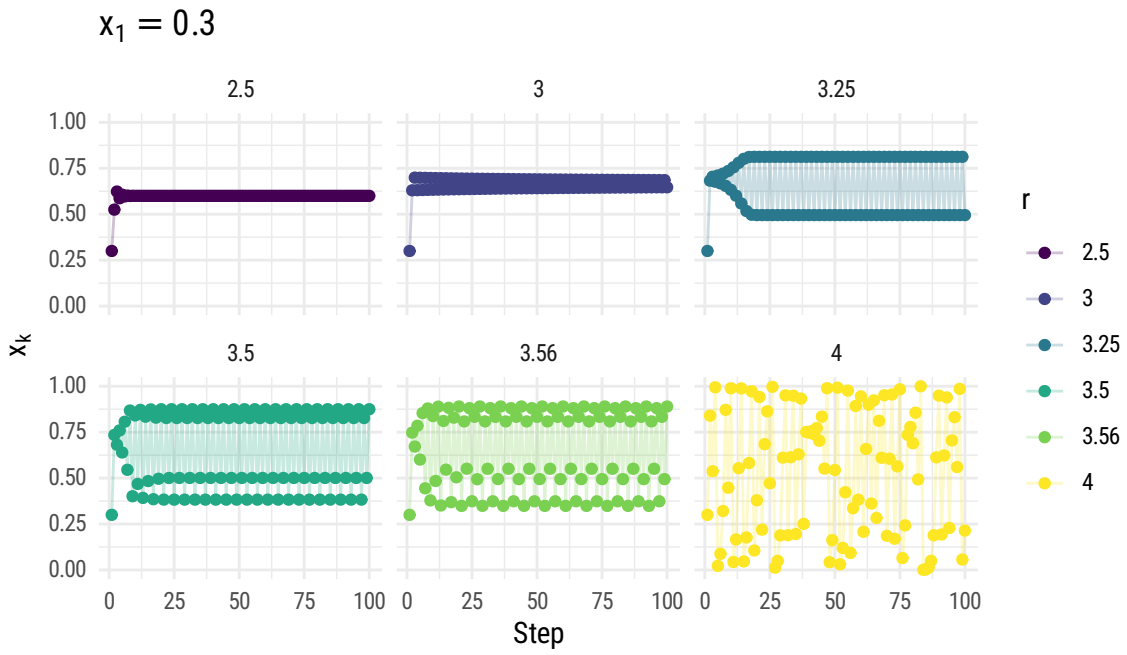
```
[[3]]
    step        xk    r
99    99 0.8124271 3.25
100  100 0.4952652 3.25

[[4]]
    step        xk   r
99    99 0.8749973 3.5
100  100 0.3828197 3.5

[[5]]
    step        xk    r
99    99 0.8898918 3.56
100  100 0.3488244 3.56

[[6]]
    step        xk r
99    99 0.0887950 4
100  100 0.3236418 4
```

(c)

```r
seqs <- bind_rows(list_of_seqs)

seqs |>
  ggplot(aes(step, xk, color = r)) +
  geom_point() +
  geom_line(alpha = 0.25) +
  labs(title = expression(x[1] == 0.2)) +
  scale_color_viridis_d() +
  facet_wrap(r ~ .)
```



(d)

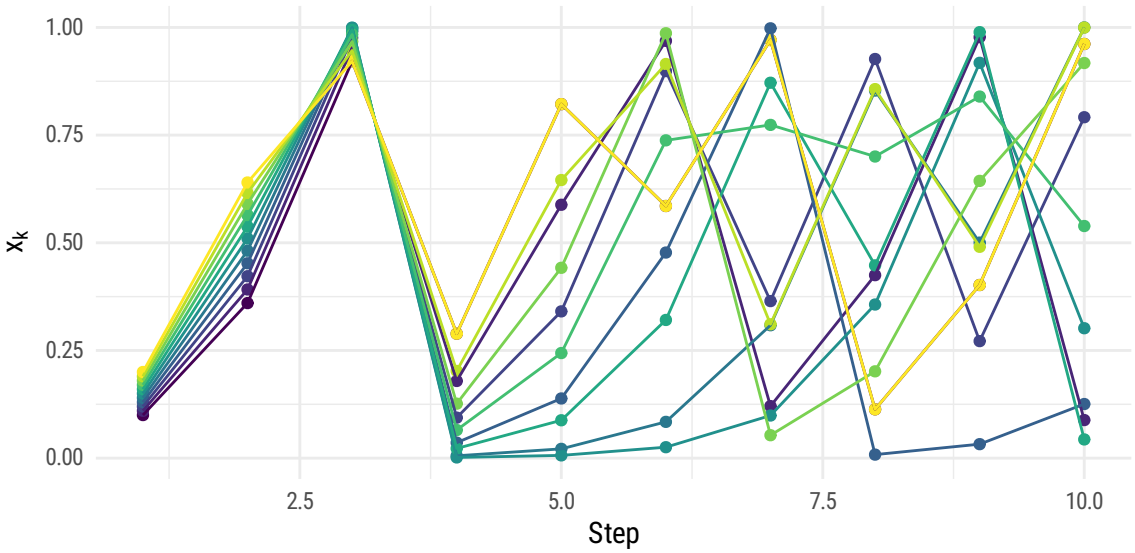


For some values of r, the sequence seems to converge to a pattern faster for a seed of 0.2 than 0.3, like $r = 3$. Alternatively, for some values of r, it is the opposite. For example, when

$r = 3.25$, the sequence seems to congerge to its oscillatory behavior quicker with a seed of 0.3 as opposed to 0.2.

(e)

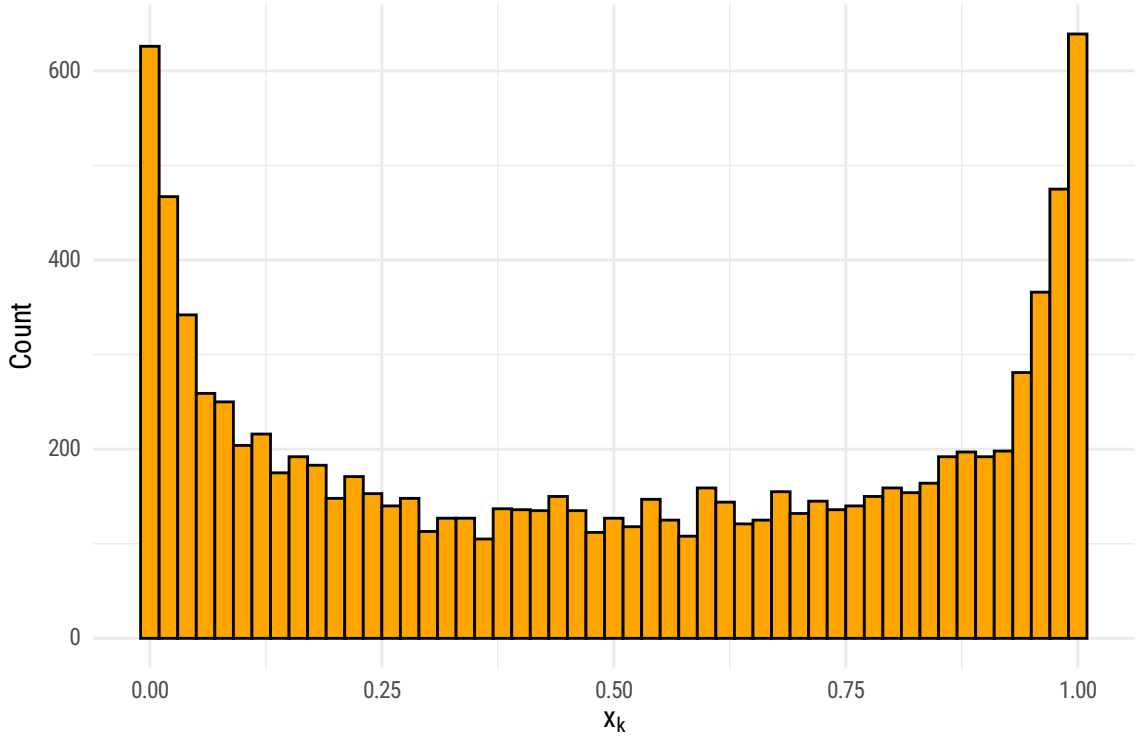
```
seqs <- lapply(seq(0.1, 0.2, 0.01),
               \(x1) data.frame(
                 "step" = 1:10,
                 "xk"   = logistic_seq(10, x1, 4),
                 "x1"   = as.factor(x1)
                 )
               ) |> bind_rows()

seqs |>
  ggplot(aes(step, xk, color = x1)) +
  geom_point() +
  geom_line() +
  labs(y = expression(x[k]),
       x = "Step") +
  scale_color_viridis_d() +
  guides(color = "none")
```



After about five steps, most of the sequences seem to vastly diverge. Until then though they are fairly similar.

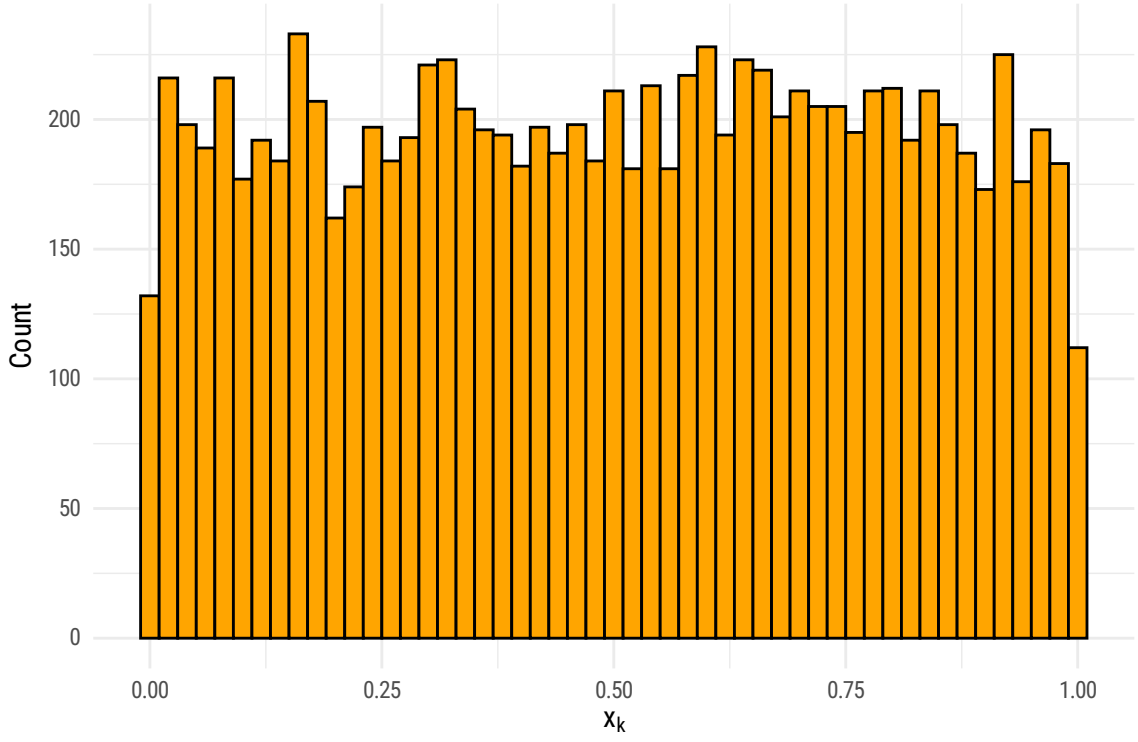
(f)

```
rchaos <- function(n, burn_in = 500) {
  seed <- (as.numeric(unclass(format(Sys.time(), "%OS3"))) %% n)/n
  logistic_seq((n + burn_in), seed, 4)[seq(burn_in + 1, n + burn_in)]
}

chaos <- data.frame(
  "step" = 1:10000,
  "xk"   = rchaos(10000)
  )

chaos |>
  ggplot(aes(xk)) +
    geom_histogram(color = "black",
                   fill = "orange",
                   binwidth = 0.02) +
    labs(y = "Count",
         x = expression(x[k]))
```



(g)

```
runif2 <- function(n) {
  pbeta(rchaos(n), 0.5, 0.5)
}

my_unif <- data.frame(
  "step" = 1:10000,
  "xk"   = runif2(10000)
  )

my_unif |>
  ggplot(aes(xk)) +
    geom_histogram(color = "black",
                   fill = "orange",
                   binwidth = 0.02) +
    labs(y = "Count",
         x = expression(x[k]))
```



(h)

It has been dually noted that this sampler suffers from some problems, as von Neumann describes. I will not use it in practice.