Project 1

Johannes Pohjolainen

```r
library(tidyverse)
```

Summary

In this project we will investigate how four different methods perform in solving initial value problems (IVP), using an example.

The IVP

Below, the IVP we will test our methods on is displayed.

$$y'(x) = cos(\pi x) + y(x)y(0) = 2$$

The results will be compared to the analytic solution $\frac{(3+2\pi^2)e^x+\pi sin(\pi x)-cos(\pi x)}{1+\pi^2}$.

Implementing the IVP

As we want to test our methods practically, we implement the IVP to have it ready for usage:

```r
dy <- function(x, y) {
  cos(pi*x) + y
}

y_sol <- function(x) ((3 + 2*pi^2)*exp(x) + pi * sin(pi * x) - cos(pi * x))/(1 + pi^2)

y_0 <- 2

n_points <- 10 * rep(2, 7) ^ c(0:6)
#plot(90:109, (90:109) %>% y_sol)
```

Methods

The methods that are to be evaluated are the: 1) Euler method 2) Heun's method 3) Midpoint method 4) 4th order runge-kutta method

Implementation of the methods

As all methods can be decribed as the $\hat{y}'$ part in $y_{i+1} = y_i + h\hat{y}'$, we only implement that part and then use a stepper that uses these as arguments.

```r
eulers <- function(x, y, h, dy) {
  dy(x, y)
}
#y_0 + eulers(0,2,0.01, dy)

midpoint <- function(x, y, h, dy) {
  dy(x + 1/2 * h, y + h/2 * dy(x, y))
}
#y_0 + midpoint(0,2,0.01, dy)

heun <- function(x, y, h, dy) {
  1/2 * (dy(x, y) + dy(x + h, y + h * dy(x, y)))
}
#y_0 + heun(0,2,0.01, dy)

fourth <- function(x, y, h, dy) {
  k1 <- dy(x, y)
  k2 <- dy(x + h/2, y + h/2 * k1)
  k3 <- dy(x + h/2, y + h/2 * k2)
  k4 <- dy(x + h, y + h * k3)
  1/6 * (k1 + 2 * k2 + 2 * k3 + k4)
}
#y_0 + fourth(0,2,0.01, dy)
```

Functions for using the methods

Now that all functions are implemented, we can create the stepper that iteratively uses the methods-functions from above to solve the IVP:

```r
approxer <- function(y_0, x_n, slope_f, n) {
  t_0 <- 0
  #always start from t = 0
  h <- (x_n - 0)/n

  x <- c(0)
  y <- c(y_0)
  for (i in 1:n) {
    x_i <- x[i] + h
    y_i <- y[i] + h * slope_f(x[i], y[i], h, dy)
    x <- c(x, x_i)
    y <- c(y, y_i)
  }
  list(x = x, y = y)
}
```

We want to compare these methods for different numbers of points. Hence we create the following method that creates a dataframe containing estimates from an array of points.

```r
check_points <- function(y_0, x_n, slope_f, n_list) {
  df <- data.frame()
  for (n in n_list) {
    approx_n <- approxer(y_0, x_n, slope_f, n)
    df <- data.frame(approx_n, estimates = as.character(n)) %>%
      bind_rows(df)
  }
  df
}
```

Graphical results

Now that we have all functions in place, it's time to create the dataframes needed to compare the methods:

```r
m1 <- check_points(y_0, 2, eulers, n_points) %>% mutate(method = "euler")
m2 <- check_points(y_0, 2, heun, n_points) %>% mutate(method = "heun's")
m3 <- check_points(y_0, 2, midpoint, n_points) %>% mutate(method = "midpoint")
m4 <- check_points(y_0, 2, fourth, n_points) %>% mutate(method = "4th order runge-kutta")
solution <- data.frame(x = seq(0,2,0.01), y = y_sol(seq(0,2,0.01)), estimates = "10", method = "analyti
for (n in n_points) {
  solution <- solution %>% mutate(estimates = as.character(n)) %>% bind_rows(solution)
}

all <- bind_rows(m1,m2,m3, m4, solution) %>%
  mutate(estimates = factor(estimates,
                            levels = factor(c(as.character(n_points)))))
```

Having the dataframe allows us to compare the methods solutions:

```r
all %>% ggplot(aes(x,y, color = method)) +
  geom_point(size = 0.1) + facet_wrap(~estimates)
```

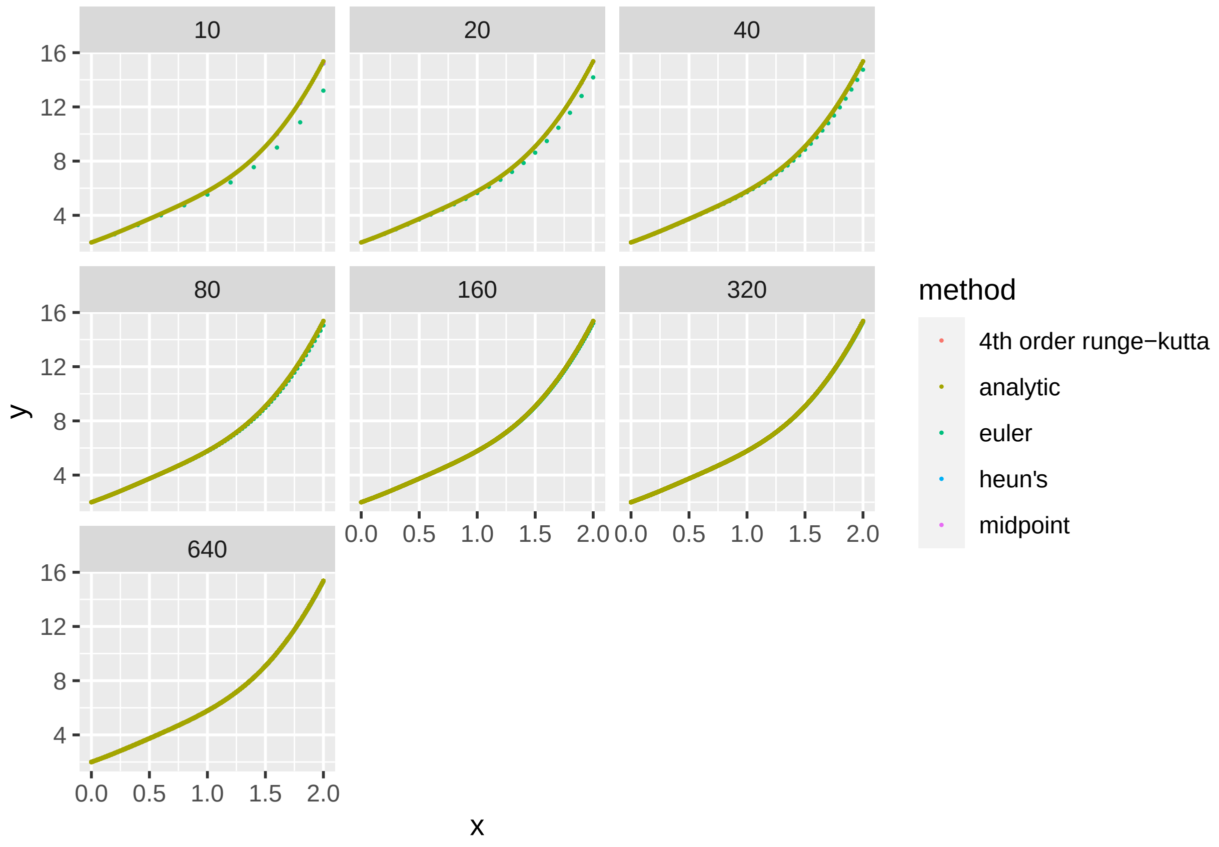

Figure 1: Fig 1: Displays the solutions for each method

From Fig 1 we see that the only method that is distingushable as beeing worse is euler's method. The others are so good that their differance is hard to see.

To try to see the differences more clearly, we plot the errors below:

```
all_error <- all %>%
  group_by(method, estimates) %>%
  filter(x == max(x)) %>%
  mutate(error = abs(y - tail(solution$y,1)))

ggplot(all_error, aes(estimates, error, color = method)) + geom_point()
```

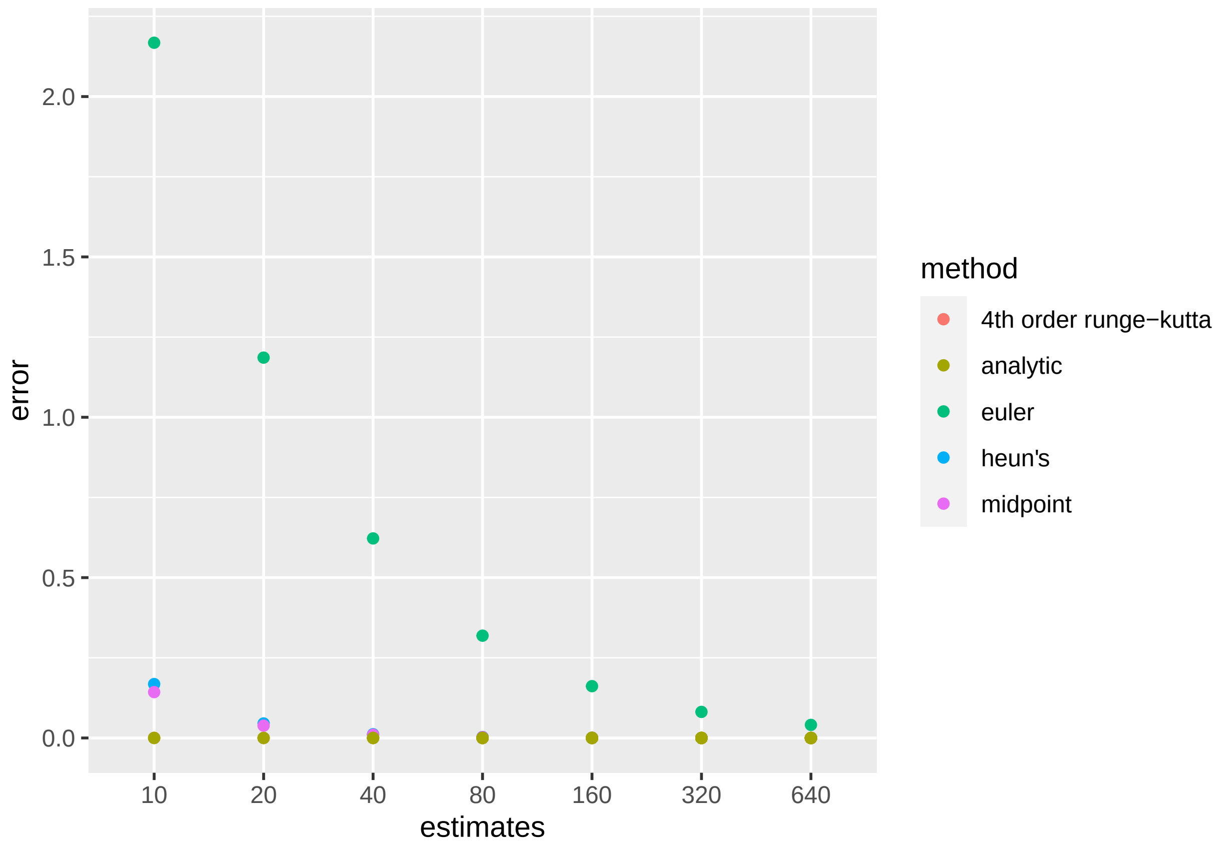

Figure 2: Fig 2: Displays the error at t = 2 for each method

```
rungers <- all_error %>% filter(as.character(method) == "4th order runge-kutta")
ratio <- rungers$error[2]/rungers$error[1]
ratio
```

```
## [1] 16.00794
```

Fig 2 allows the following ranking:

1. 4th order runge-kutta
2. midpoint
3. heun's
4. euler's

We see that the errors seem to decline non-linearly when increasing step-size. Euler's method apperas to have a first order global error as doubling the points halved the error. Heun's method seems to have had it's error cut into four, hence it seems to be of second order. The midpoint error seemed to behave similarly. The 4th order runge kutta is harder to see, which is why we calculated it. Doubling the points gives an error reduction of 1/16. Hence it seems to be a fourth order method.