Basic Features

```
library(alloscore)
library(dplyr)
library(ggplot2)
library(tibble)
```

This vignette runs through some of the new functionality added in this version of `alloscore`.

We start with an exponential data generating process across $N = 10$ targets and use the convenience function `add_pdqr_funs` to construct a log-normal forecaster that tries to match the exponential mean. Only cdfs ("p") and quantile functions ("q") need to be added for `alloscore` to operate but I'll add densities too for plotting.

```
N <- 10
y_mean <- 50
y_gen <- function(Ny=N) rexp(Ny, rate = 1/y_mean)

make_m_lnorm <- function(N) {
tibble(
  target_names = LETTERS[1:N],
    dist = "lnorm",
    sdlog = 1 + rpois(N, 8),
  meanlog = log(y_mean) - .5*sdlog,
) %>% add_pdqr_funs(types = c("p", "d", "q"))
}

m_lnorm <- make_m_lnorm(N)
y <- y_gen()
```

We can now score for an array of Ks in one step, returning a data frame of list columns containing the score, the components of the score, the allocations, and information about the optimization proceedure for each K...

```
a1 <- alloscore(df = m_lnorm, y = y, K = 60:70)
a1$xdf[[2]] # allocations for K = 61
#> # A tibble: 10 x 7
#>    target_names     x     y oracle components_raw components_oracle components
#>    <chr>        <dbl> <dbl>  <dbl>          <dbl>             <dbl>      <dbl>
#>  1 A             3.45 82.3  12.9           78.9               69.5       9.43
#>  2 B             3.06 34.8   5.45          31.8               29.4       2.39
#>  3 C             3.21 76.6  12.0           73.4               64.6       8.77
#>  4 D             3.82 11.2   1.75           7.36               9.43     -2.07
#>  5 E             4.41 37.1   5.81          32.7               31.3       1.40
#>  6 F             3.21 16.0   2.51          12.8               13.5      -0.706
#>  7 G             3.06 59.5   9.31          56.5               50.2       6.25
#>  8 H             3.06 38.9   6.08          35.8               32.8       3.02
#>  9 I            30.3  29.8   4.66           0                 25.1     -25.1
#> 10 J             3.45  3.58  0.560          0.130              3.02     -2.89
a1 %>% select(K, score) # scores for K in 60:70
```

```
#> # A tibble: 11 x 2
#>        K score
#>    <int> <dbl>
#>  1    60 0.413
#>  2    61 0.459
#>  3    62 0.505
#>  4    63 0.635
#>  5    64 0.788
#>  6    65 0.940
#>  7    66 1.09
#>  8    67 1.25
#>  9    68 1.40
#> 10    69 1.55
#> 11    70 1.70
```

or via pipes with the necessary parameters added at each step:

```
a1_piped <- m_lnorm %>% allocate(K = 60:70) %>% alloscore(y = y)
a1_piped %>% select(K, score)
#> # A tibble: 11 x 2
#>        K score
#>    <int> <dbl>
#>  1    60 0.413
#>  2    61 0.459
#>  3    62 0.505
#>  4    63 0.635
#>  5    64 0.788
#>  6    65 0.940
#>  7    66 1.09
#>  8    67 1.25
#>  9    68 1.40
#> 10    69 1.55
#> 11    70 1.70
```

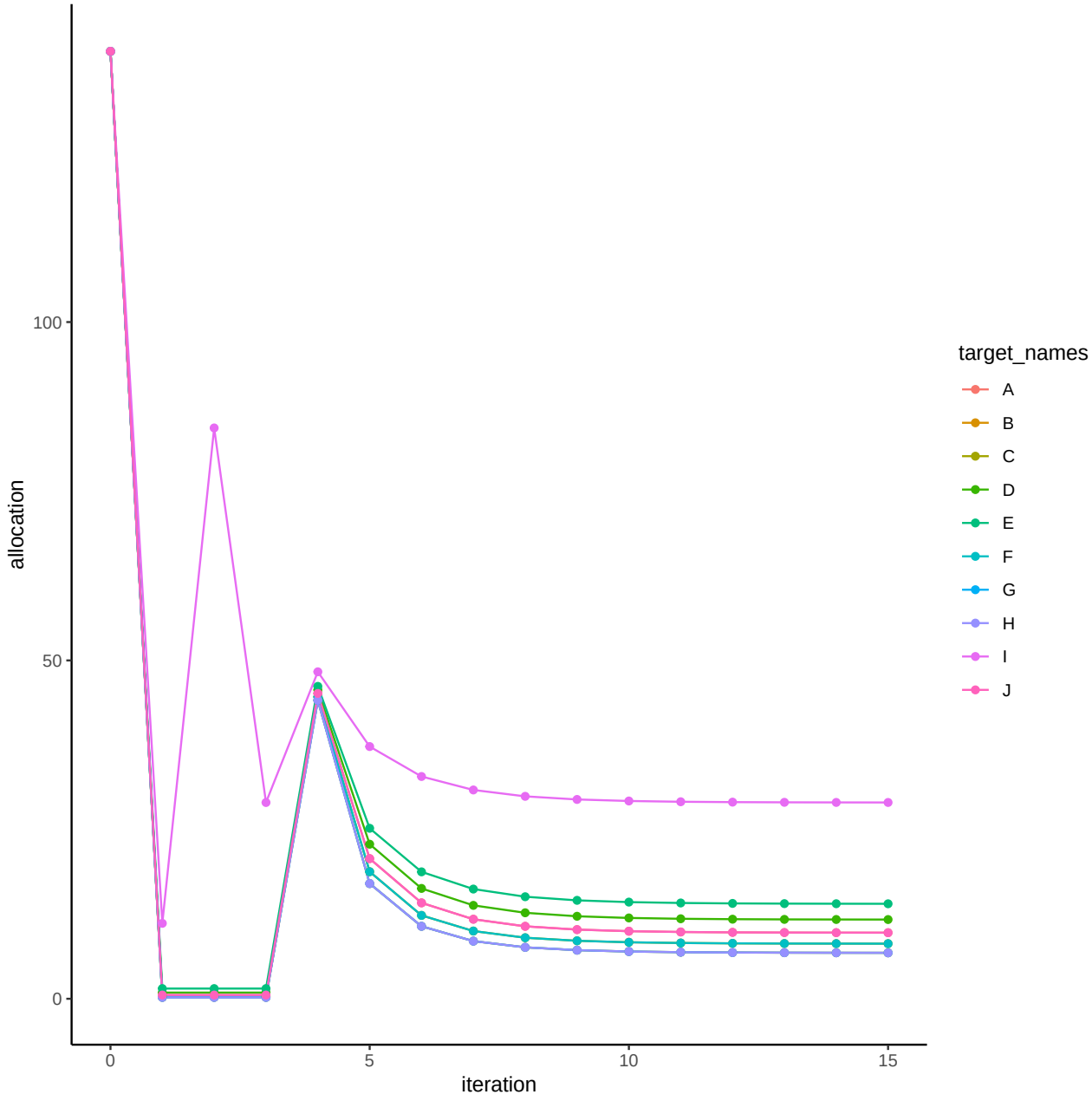
This works by giving the output of `allocate` an `allocated` class attribute and then calling an `alloscore` method for the `allocated` object. I'm trying to decide whether it makes sense to have a parallel system for oracles or leave oracle comparison in the jury-rigged state it is in now.

The `xs` column contains data frames for each K recording the iteration history of each allocation. I'll try to wrap this into a plot method soon:

```
# history for K = 69
(iters9 <- a1$xs[[9]])
#> # A tibble: 10 x 17
#>    target_names    qs    `1`    `2`    `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`  `11`  `12`
#>    <chr>        <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#>  1 A              140  0.554  0.554  0.554  45.1  20.7  14.2 11.8  10.7  10.2   9.99  9.87  9.81
#>  2 B              140  0.204  0.204  0.204  44.1  17.0  10.7  8.52  7.60  7.18  6.98  6.88  6.83
#>  3 C              140  0.336  0.336  0.336  44.6  18.8  12.3 10.0   9.02  8.57  8.35  8.24  8.19
#>  4 D              140  0.914  0.914  0.914  45.6  22.8  16.3 13.8  12.7  12.2  11.9  11.8  11.8
#>  5 E              140  1.51   1.51   1.51   46.2  25.2  18.8 16.2  15.1  14.5  14.3  14.2  14.1
#>  6 F              140  0.336  0.336  0.336  44.6  18.8  12.3 10.0   9.02  8.57  8.35  8.24  8.19
#>  7 G              140  0.204  0.204  0.204  44.1  17.0  10.7  8.52  7.60  7.18  6.98  6.88  6.83
#>  8 H              140  0.204  0.204  0.204  44.1  17.0  10.7  8.52  7.60  7.18  6.98  6.88  6.83
#>  9 I              140 11.1   84.4   29.0    48.3  37.3  32.8 30.9  29.9  29.5  29.2  29.1  29.1
#> 10 J              140  0.554  0.554  0.554  45.1  20.7  14.2 11.8  10.7  10.2   9.99  9.87  9.81
```

```
#> # i 3 more variables: `13` <dbl>, `14` <dbl>, `15` <dbl>
```

```
iters9 %>% rename(`0` = qs) %>%
  tidyr::pivot_longer(
    cols = -c(target_names),
    names_to = "iteration",
    values_to = "allocation") %>%
  mutate(
    iteration = as.numeric(iteration)
    ) %>%
  ggplot(aes(x = iteration, y = allocation, color = target_names)) +
  geom_line() + geom_point() + theme_classic()
```



The generalize piecwise lnear loss functions used for each target and the parameters used to construct them

are stored in an attribute:

```
attr(a1, "gpl_df")
#> # A tibble: 10 x 8
#>    g     target_names kappa alpha O     U     offset gpl_loss_fun
#>  * <chr> <chr>        <dbl> <dbl> <lgl> <lgl>  <dbl> <list>
#>  1 x     A                1     1 NA    NA         0 <fn>
#>  2 x     B                1     1 NA    NA         0 <fn>
#>  3 x     C                1     1 NA    NA         0 <fn>
#>  4 x     D                1     1 NA    NA         0 <fn>
#>  5 x     E                1     1 NA    NA         0 <fn>
#>  6 x     F                1     1 NA    NA         0 <fn>
#>  7 x     G                1     1 NA    NA         0 <fn>
#>  8 x     H                1     1 NA    NA         0 <fn>
#>  9 x     I                1     1 NA    NA         0 <fn>
#> 10 x     J                1     1 NA    NA         0 <fn>
```

O and U are mostly still just placeholders since I haven't yet adapted `allocate` to accept them, but this should be simple.

Also, `alloscore` (and `allocate`) can take the forecast (and gpl) info as individual arguments:

```
K <- c(10, 20, 30)
as_indiv <- alloscore(F = m_lnorm$F, Q = m_lnorm$Q, K = K, y = y)
as_df <- m_lnorm %>% allocate(K = K) %>% alloscore(y = y)

full_join(
  as_indiv %>% select(K, score),
  as_df %>% select(K, score),
  by = "K"
)
#> # A tibble: 3 x 3
#>       K   score.x   score.y
#>   <dbl>     <dbl>     <dbl>
#> 1    10 -1.63e- 3 -1.63e- 3
#> 2    20  5.68e-14  5.68e-14
#> 3    30  0         0
```

```
oa <- oracle_allocate(m_lnorm, y = y, K = 10:15)
oa1 <- oracle_allocate(y = y, K = 10:15)
oa1 <- oracle_allocate(y = y, K = 10, g = "log(1+x)")
```

FWIW, the time savings seems like it's around 2/3, but my bet is this will increase as we get into hub analyses...

```
N <- 50
m_lnorm2 <- make_m_lnorm(N)
Ks <- seq(25, 300, length.out = 100)
{
cat("new way: ")
start.time <- Sys.time()
allocate(df = m_lnorm2, K = Ks, eps_K = .0001)
end.time <- Sys.time()
print(end.time-start.time)
}
#> new way: Time difference of 7.928304 secs
```

```
{
cat("bulldozer way: ")
start.time <- Sys.time()
for (Kind in Ks) {
  allocate(df = m_lnorm2, K = Kind, eps_K = .0001)
}
end.time <- Sys.time()
print(end.time-start.time)
}
#> bulldozer way: Time difference of 23.3204 secs
```