Unit Test Demo

2023-09-14

The purpose of having unit tests is to ensure future feature development does not affect current analyses. How unit tests work is that it compares the results from the code (usually after some changes made) with previous known results to ensure the output is not changed when it’s not supposed to.

Use `fitexp3` as an example, the current fit for exponential 3 with some simulated data is as below, and we want to make sure this does not change as we add more changes to the package.

```
data("signatures")
conc=as.numeric(str_split(signatures[1,"conc"],"\\|")[[1]])
resp=as.numeric(str_split(signatures[1,"resp"],"\\|")[[1]])
fitexp3(conc, resp)
```

```
## $success
## [1] 1
##
## $aic
## [1] -2.325414
##
## $cov
## [1] 1
##
## $rme
## [1] 0.1204857
##
## $modl
## [1] 0.005237419 0.012028715 0.025721741 0.059355760 0.302734925 0.128144012
## [7] 0.685719829
##
## $a
## [1] 2.786124
##
## $b
## [1] 269.5222
##
## $p
## [1] 0.6895939
##
## $er
## [1] -2.441218
##
## $a_sd
## [1] 11.63993
##
## $b_sd
## [1] 1052.334
##
## $p_sd
## [1] 0.5020625
##
## $er_sd
## [1] 0.3799298
##
## $pars
## [1] "a"  "b"  "p"  "er"
##
## $sds
## [1] "a_sd"  "b_sd"  "p_sd"  "er_sd"
```

To create a test for a function, we can use `test_that` function from `testthat` package. The test will run the code above and compare the result to what we expect to see (results from above). The following chunk is the test created for `fitexp3` and it checks for the fitted values for all model parameters. This code can be ran directly and the output would be either you pass or did not pass the test.

```
test_that("fitexp3 works", {
  data("signatures")
  conc=as.numeric(str_split(signatures[1,"conc"],"\\|")[[1]])
  resp=as.numeric(str_split(signatures[1,"resp"],"\\|")[[1]])

  expect_equal(fitexp3(conc, resp)$a, 2.786, tolerance = 1e-3)
  expect_equal(fitexp3(conc, resp)$b, 269.522, tolerance = 1e-3)
  expect_equal(fitexp3(conc, resp)$p, 0.6895, tolerance = 1e-3)
})
```

```
## Test passed 🥳
```

We can also use `devtools::test()` to run all tests in the package at once. The result will show have many tests are ran and how many passed. For any failed tests, the result will show you the what is the actual output and what is the expected output.

```
# this markdown file is not in the package directory yet so I specify a path
# just run devtools::test() while in the package directory
devtools::test("../CompTox-ToxCast-tcplFit2")
```

```
## ℹ Testing tcplfit2
## ✔ | F W S  OK | Context
## 
⠏ |         0 | concRespCore
⠋ |         1 | concRespCore
⠹ |         3 | concRespCore
✔ |         4 | concRespCore [1.0s]
## 
⠏ |         0 | fitexp2
⠋ |         1 | fitexp2
⠙ |         2 | fitexp2
✔ |         2 | fitexp2
## 
⠏ |         0 | fitexp3
⠋ |         1 | fitexp3
⠙ |         2 | fitexp3
⠹ |         3 | fitexp3
✔ |         3 | fitexp3
## 
⠏ |         0 | fitexp4
⠋ |         1 | fitexp4
✔ |         2 | fitexp4
## 
⠏ |         0 | fitexp5
⠋ |         1 | fitexp5
⠙ |         2 | fitexp5
⠹ |         3 | fitexp5
✔ |         3 | fitexp5
## 
⠏ |         0 | fitgnls
⠋ |         1 | fitgnls
⠙ |         2 | fitgnls
⠹ |         3 | fitgnls
⠸ |         4 | fitgnls
⠼ |         5 | fitgnls
✔ |         5 | fitgnls [2.1s]
## 
⠏ |         0 | fithill
⠋ |         1 | fithill
⠙ |         2 | fithill
⠹ |         3 | fithill
✔ |         3 | fithill
## 
⠏ |         0 | fitpoly1
⠋ |         1 | fitpoly1
✔ |         1 | fitpoly1
## 
⠏ |         0 | fitpoly2
⠋ |         1 | fitpoly2
⠙ |         2 | fitpoly2
✔ |         2 | fitpoly2
## 
⠏ |         0 | fitpow
```

```
⠋ |          1 | fitpow
✓ |          2 | fitpow
##
⠏ |          0 | tcplfit2_core
⠋ |          1 | tcplfit2_core
⠹ |         13 | tcplfit2_core
✓ |         23 | tcplfit2_core
##
⠏ |          0 | tcplhit2_core
⠋ |          1 | tcplhit2_core
✓ |          4 | tcplhit2_core
##
## ══ Results ═════════════════════════════════════════════════════════════════════
## Duration: 8.9 s
##
## [ FAIL 0 | WARN 0 | SKIP 0 | PASS 54 ]
```

Let say I'm working on adding a new argument to allow users to choose normal as error distribution. The default for this argument would still be t-distribution as is right now, and adding this feature should not change the fitting result if normal error is not used.

For instance, I accidentally hard code the `fitexp3` to use normal for error distribution, it will return different fitted parameter values and cause the test to fail.

```
# this is an example code that contains a bug
# running this code will mask the function in the original package
# reminder to re-load the package if you ran the this chunk

fitexp3 = function(conc, resp, bidirectional = TRUE, verbose = FALSE, nofit = FALSE, dmin = .3,
                   errfun = "dt4", ...){
  ## adding a new errfun argument, defaults to t-distribution

  fenv <- environment()
  #initialize myparams
  pars <- paste0(c("a", "b", "p", "er"))
  sds <- paste0(c("a", "b", "p","er"), "_sd")
  myparams = c("success", "aic", "cov", "rme", "modl", pars, sds, "pars", "sds")

  #returns myparams with appropriate NAs
  if(nofit){
    out = as.list(rep(NA_real_, length(myparams)))
    names(out) = myparams
    out[["success"]] = out[["cov"]] = NA_integer_
    out[["pars"]] = pars
    out[["sds"]] = sds
    return(out)
  }

  #median at each conc, for multi-valued responses
  rmds <- tapply(resp, conc, median)
  #get max response and corresponding conc
  if(!bidirectional) mmed = rmds[which.max(rmds)] else mmed = rmds[which.max(abs(rmds))] #shorte
ned this code
  mmed_conc <- as.numeric(names(mmed)) #fixed this bug

  resp_max <- max(resp)
  resp_min <- min(resp)
  conc_min <- min(conc)
  conc_max <- max(conc)

  er_est <- if ((rmad <- mad(resp)) > 0) log(rmad) else log(1e-16)

  ###--------------------- Fit the Model ----------------------###
  ## Starting parameters for the Model
  a0 = mmed #use largest response with desired directionality
  if(a0 == 0) a0 = .01  #if 0, use a smallish number
  g <- c(a0, # y scale (a)
         conc_max, # x scale (b); curve scaled to highest resp and max conc
         1.2,       # power(p)
         er_est )# logSigma (er)


  ## Generate the bound matrices to constrain the model.
  #                a    b    p    er
  Ui <- matrix(c( 1,   0,   0,   0,
                 -1,   0,   0,   0,
                 0,   1,   0,   0,
                 0,  -1,   0,   0,
                 0,   0,   1,   0,
                 0,   0,  -1,   0),
               byrow = TRUE, nrow = 6, ncol = 4)

  if(!bidirectional){
    bnds <- c(1e-8*abs(a0), -1e8*abs(a0), # a bounds
              1e-2*conc_max, -1e8*conc_max, # b bounds (lower bound avoids overflow at max conc, 
max power)
              dmin, -8) # p bounds (p > 1, following bmd guidelines)
  } else {
    bnds <- c(-1e8*abs(a0), -1e8*abs(a0), # a bounds
              1e-2*conc_max, -1e8*conc_max, # b bounds (lower bound avoids overflow at max conc, 
max power)
             dmin, -8) # p bounds (p > 1, following bmd guidelines)
  }

  Ci <- matrix(bnds, nrow = 6, ncol = 1)

  ## Optimize the model
  fit <- try(constrOptim(g,
                         tcplObj,
                         ui = Ui,
                         ci = Ci,
                         mu = 1e-6,
                         method = "Nelder-Mead",
                         control = list(fnscale = -1,
                                        reltol = 1e-10,
                                        maxit = 6000),
                         conc = conc,
                         resp = resp,
                         fname = "exp3",
                         errfun = "dnorm"), ## should be errfun = errfun
              silent = !verbose)


  ## Generate some summary statistics
  if (!is(fit, "try-error")) { # The model fit the data
    if(verbose) cat("Exp3 >>>",fit$counts[1],fit$convergence,"\n")

    success <- 1L
    aic <- 2*length(fit$par) - 2*fit$value # 2*length(fit$par) - 2*fit$value
    mapply(assign,
           c(pars),
           fit$par,
           MoreArgs = list(envir = fenv))

    ## Calculate rmse for gnls
    modl <- exp3(fit$par, conc)
    rme <- sqrt(mean((modl - resp)^2, na.rm = TRUE))
    ## Calculate the sd for the gnls parameters
    fit$cov <- try(solve(-hessian(tcplObj,
                                  fit$par,
                                  conc = conc,
                                  resp = resp,
                                  fname = "exp3",
                                  errfun = "dnorm")), ## should be errfun = errfun
                    silent = !verbose)

    if (!is(fit$cov, "try-error")) { # Could invert gnls Hessian

      cov <- 1L
      diag_sqrt <- suppressWarnings(sqrt(diag(fit$cov)))
      if (any(is.nan(diag_sqrt))) {
        mapply(assign,
               sds,
               NaN,
               MoreArgs = list(envir = fenv))
      } else {
        mapply(assign,
               sds,
               diag_sqrt,
               MoreArgs = list(envir = fenv))
      }

    } else { # Could not invert gnls Hessian

      cov <- 0L
      mapply(assign,
             c(sds),
             NA_real_,
             MoreArgs = list(envir = fenv))

    }

  } else { # Curve did not fit the data

    success <- 0L
    aic <- NA_real_
    cov <- NA_integer_
    rme <- NA_real_
    modl = NA_real_

    mapply(assign,
           c(pars, sds),
           NA_real_,
           MoreArgs = list(envir = fenv))

  }

  return(mget(myparams))
```

```
}
```

This test will fail and the output will show you what the code returned v.s. what it expect.

```
test_that("fitexp3 works", {
  data("signatures")
  conc=as.numeric(str_split(signatures[1,"conc"],"\\|")[[1]])
  resp=as.numeric(str_split(signatures[1,"resp"],"\\|")[[1]])

  expect_equal(fitexp3(conc, resp)$a, 2.786, tolerance = 1e-3)
  expect_equal(fitexp3(conc, resp)$b, 269.522, tolerance = 1e-3)
  expect_equal(fitexp3(conc, resp)$p, 0.6895, tolerance = 1e-3)
})
```

```
## ── Failure ('<text>:6:3'): fitexp3 works ───────────────────────────────────────
## fitexp3(conc, resp)$a not equal to 2.786.
## 1/1 mismatches
## [1] 8.05 - 2.79 == 5.27
##
## ── Failure ('<text>:7:3'): fitexp3 works ───────────────────────────────────────
## fitexp3(conc, resp)$b not equal to 269.522.
## 1/1 mismatches
## [1] 961 - 270 == 691
##
## ── Failure ('<text>:8:3'): fitexp3 works ───────────────────────────────────────
## fitexp3(conc, resp)$p not equal to 0.6895.
## 1/1 mismatches
## [1] 0.719 - 0.69 == 0.0295
```

```
## Error in `reporter$stop_if_needed()`:
## ! Test failed
```

Running all tests as we making changes to the package can ensure the current analyses are not unintentionally affected.