Find Best Fit of Curves Through Points

Fan Wang

2022-07-12

Contents

1 Fit Curves Through Points

Go to the **RMD**, **R**, **PDF**, or **HTML** version of this file. Go back to fan's REconTools research support package, R4Econ examples page, PkgTestR packaging guide, or Stat4Econ course page.

1.1 Polynomial Fit with N Sets of Points

There are three points defined by their x and y coordinates. We draw these points randomly, and we find a curve that fits through them. The fit is exact. If there are more than three sets of points, we will not be able to fit exactly. In the illustration before, first, we draw 3 sets of x and y points, then we draw 4 and 5 sets, and we compare the prediction results.

Note that we are not generating data from a particular set of quadratic (polynomial) parameters, we are just drawing random values. When we draw exactly three pairs of random x and y values, we can find some polynomial that provides an exact fit through the three points in 2d space.

We define the function here.

```r
#' Test polynomial fit to random draw x and y points
#'
#' @description Draw random sets of x and y points, fit a polynomial curve through
#' and compare predictions of y and actual y values.
#'
#' @param it_xy_pairs The number of x and y pair points
#' @param it_seed The random seed for drawing values
#' @param it_poly_fit An integer value of the order of polynomial
#' @param fl_mean The mean of the the random normal draw
#' @param fl_sd The standard deviation of the random normal draw
#' @returns
#' \itemize{
#'   \item rs_lm_poly - polynomial fit estimation results
#'   \item df_data_predict - N by 3 where N = \code{it_xy_pairs} and
#' columns are x, y, y-predict, and residual
#'   \item td_table_return - Display version of df_data_predict with title
#' }
#' @import stats, tibble, dplyr
#' @author Fan Wang, \url{http://fanwangecon.github.io}
ffi_lm_quad_fit <- function(it_xy_pairs = 3, it_seed = 123,
                             it_poly_fit = 2, fl_mean = 1, fl_sd = 1,
                             verbose = FALSE) {

  # 1. Generate three pairs of random numbers
  set.seed(it_seed)
  mt_rnorm <- matrix(
    rnorm(it_xy_pairs * 2, mean = fl_mean, sd = fl_sd),
    nrow = it_xy_pairs, ncol = 2
  )
  colnames(mt_rnorm) <- c("x", "y")
  rownames(mt_rnorm) <- paste0("p", seq(1, it_xy_pairs))
  df_rnorm <- as_tibble(mt_rnorm)

  # 2. Quadratic fit using ORTHOGONAL POLYNOMIAL
  # For predictions, lm(y ~ x + I(x^2)) and lm(y ~ poly(x, 2)) are the same,
  # but they have different parameters because x is transformed by poly().
  rs_lm_quad <- stats::lm(y ~ poly(x, it_poly_fit), data = df_rnorm)
  if (verbose) print(stats::summary.lm(rs_lm_quad))

  # 3. Fit prediction
  ar_y_predict <- stats::predict(rs_lm_quad)
  df_data_predict <- cbind(df_rnorm, ar_y_predict) %>%
    mutate(res = ar_y_predict - y)
  if (verbose) print(df_data_predict)

  # 4. show values
  st_poly_order <- "Quadratic"
  if (it_poly_fit != 2) {
    st_poly_order <- paste0(it_poly_fit, "th order")
  }
  td_table_return <- kable(df_data_predict,
    caption = paste0(
      st_poly_order, " Fit of ", it_xy_pairs, " Sets of Random (X,Y) Points"
    )
  ) %>%
    kable_styling_fc()

  return(list(
    rs_lm_quad = rs_lm_quad,
    df_data_predict = df_data_predict,
    td_table_return = td_table_return
  ))
}
```

In the first example below, we simulate 3 set of points and estimate quadratic exact fit.

```r
ls_ffi_lm_quad_fit <-
  ffi_lm_quad_fit(
    it_xy_pairs = 3, it_seed = 123,
    it_poly_fit = 2, fl_mean = 1, fl_sd = 1
  )
ls_ffi_lm_quad_fit$td_table_return
```

In the second example below, we simulate 4 set of points and estimate a quadratic non-exact fit.

Quadratic Fit of 3 Sets of Random (X,Y) Points

x	y	ar_y_predict	res
0.4395244	1.070508	1.070508	0
0.7698225	1.129288	1.129288	0
2.5587083	2.715065	2.715065	0

```
ls_ffi_lm_quad_fit <-
  ffi_lm_quad_fit(
    it_xy_pairs = 4, it_seed = 345,
    it_poly_fit = 2, fl_mean = 1, fl_sd = 1
  )
ls_ffi_lm_quad_fit$td_table_return
```

Quadratic Fit of 4 Sets of Random (X,Y) Points

x	y	ar_y_predict	res
0.2150918	0.9324684	0.9373687	0.0049003
0.7204856	0.3664796	1.5207575	1.1542779
0.8385421	0.0722760	-0.0080226	-0.0802987
0.7094034	2.7107710	1.6318915	-1.0788795

In the third example below, we simulate the same 4 sets of points as in the prior example, but now use a cubic polynomial to fit the data exactly.

```
ls_ffi_lm_cubic_fit <-
  ffi_lm_quad_fit(
    it_xy_pairs = 4, it_seed = 345,
    it_poly_fit = 3, fl_mean = 1, fl_sd = 1
  )
ls_ffi_lm_cubic_fit$td_table_return
```

3th order Fit of 4 Sets of Random (X,Y) Points

x	y	ar_y_predict	res
0.2150918	0.9324684	0.9324684	0
0.7204856	0.3664796	0.3664796	0
0.8385421	0.0722760	0.0722760	0
0.7094034	2.7107710	2.7107710	0