Analytical Formula Fit Curves Through Points

Fan Wang

2022-07-16

Contents

1 Polynomial Formulas for 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 Formulas for Quadratic Parameters and Three Points

There are three points defined by their x and y coordinates, find a curve that fits through them exactly.

First, we have three equations with three unknowns, A, B, and C.

$$
\begin{aligned}
y_1 &= A + B \cdot x_1 + C \cdot x_1^2 \\
y_2 &= A + B \cdot x_2 + C \cdot x_2^2 \\
y_3 &= A + B \cdot x_3 + C \cdot x_3^2
\end{aligned}
$$

Second, we rewrite the problem in matrix form.

$$
\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}
$$

Third, we solve the system of equations for the unknown vector $[A, B, C]$, via elementary row operations. The code below uses matlab to arrive at symbolic analytical solutions for for $[A, B, C]$.

```
clc
clear

% Define inputs
syms x11 x12 x21 x22 x31 x32 y1 y2 y3
mt_sm_z = [1,x11,x12;1,x21,x22;1,x31,x32];
ar_sm_y = [y1;y2;y3];

% Solve analytically
ar_sm_solu = linsolve(mt_sm_z, ar_sm_y)
```

```
% Randomly draw x and y values
rng(1234);
mt_rand = rand(3,2);
mt_rand = [0.1915, 0.6221, 0.4377;
  0.7854, 0.7800, 0.2726]';
[fl_x1, fl_x2, fl_x3] = deal(mt_rand(1,1), mt_rand(2,1), mt_rand(3,1));
[fl_y1, fl_y2, fl_y3] = deal(mt_rand(1,2), mt_rand(2,2), mt_rand(3,2));
[fl_x11, fl_x21, fl_x31] = deal(fl_x1, fl_x2, fl_x3);
[fl_x12, fl_x22, fl_x32] = deal(fl_x1^2, fl_x2^2, fl_x3^2);

% Numerically evaluate coefficients
ar_fl_solu = double(subs(ar_sm_solu, ...
  {x11, x12, x21, x22, x31, x32, y1, y2, y3}, ...
  {fl_x11,fl_x12,fl_x21,fl_x22,fl_x31,fl_x32,fl_y1,fl_y2,fl_y3}));
disp(['ar_fl_solu=', num2str(ar_fl_solu')])

% Y predictions
mt_fl_z = [1,fl_x11,fl_x12;1,fl_x21,fl_x22;1,fl_x31,fl_x32];
ar_fl_y_pred = mt_fl_z*ar_fl_solu;
ar_fl_x_actual = [fl_x1;fl_x2;fl_x3];
ar_fl_y_actual = [fl_y1;fl_y2;fl_y3];

% Compare results
tb_test = array2table([ar_fl_x_actual';ar_fl_y_actual';ar_fl_y_pred']');
cl_col_names = ["x_actual","y_actual", "y_predict"];
cl_row_names = strcat('obs_', string((1:3)));
tb_test.Properties.VariableNames = matlab.lang.makeValidName(cl_col_names);
tb_test.Properties.RowNames = matlab.lang.makeValidName(cl_row_names);
display(tb_test);
```

Fourth, the solutions are as follows.

$$A = \frac{x_1 x_2^2 y_3 - x_1^2 x_2 y_3 - x_1 x_3^2 y_2 + x_1^2 x_3 y_2 + x_2 x_3^2 y_1 - x_2^2 x_3 y_1}{x_1 x_2^2 - x_1^2 x_2 - x_1 x_3^2 + x_1^2 x_3 + x_2 x_3^2 - x_2^2 x_3}$$

$$B = \frac{-\left(x_1^2 y_2 - x_1^2 y_3 - x_2^2 y_1 + x_2^2 y_3 + x_3^2 y_1 - x_3^2 y_2\right)}{x_1 x_2^2 - x_1^2 x_2 - x_1 x_3^2 + x_1^2 x_3 + x_2 x_3^2 - x_2^2 x_3}$$

$$C = \frac{x_1 y_2 - x_1 y_3 - x_2 y_1 + x_2 y_3 + x_3 y_1 - x_3 y_2}{x_1 x_2^2 - x_1^2 x_2 - x_1 x_3^2 + x_1^2 x_3 + x_2 x_3^2 - x_2^2 x_3}$$

Fifth, given three pairs randomly drawn x and y points, we use the formulas just derived to find the parameters for the quadratic polynomial.

```
# Inputs X and Y
set.seed(123)
# Draw Randomly
mt_rnorm <- matrix(rnorm(6, mean = 1, sd = 1), nrow = 3, ncol = 2)
# # Fixed Values
# mt_rnorm <- matrix(c(
#   0.1915, 0.6221, 0.4377,
#   0.7854, 0.7800, 0.2726
# ), nrow = 3, ncol = 2)
```

```r
colnames(mt_rnorm) <- c("x", "y")
x1 <- mt_rnorm[1, 1]
x2 <- mt_rnorm[2, 1]
x3 <- mt_rnorm[3, 1]
y1 <- mt_rnorm[1, 2]
y2 <- mt_rnorm[2, 2]
y3 <- mt_rnorm[3, 2]

# X quadratic
x11 <- x1
x12 <- x1**2
x21 <- x2
x22 <- x2**2
x31 <- x3
x32 <- x3**2

# Shared denominator
fl_denominator <- (x11 * x22 - x12 * x21
  - x11 * x32 + x12 * x31
  + x21 * x32 - x22 * x31)

# Solve for A, B, and C exact fit quadratic coefficients
fl_A <- (x11 * x22 * y3 - x12 * x21 * y3
  - x11 * x32 * y2 + x12 * x31 * y2
  + x21 * x32 * y1 - x22 * x31 * y1) / fl_denominator

fl_B <- -(x12 * y2 - x12 * y3
  - x22 * y1 + x22 * y3
  + x32 * y1 - x32 * y2) / fl_denominator

fl_C <- (x11 * y2 - x11 * y3
  - x21 * y1 + x21 * y3
  + x31 * y1 - x31 * y2) / fl_denominator

# Display
st_display <- paste0(
  "A(intercept)=", round(fl_A, 3),
  ", B(lin)=", round(fl_B, 3),
  ", C(quad)=", round(fl_C, 3)
)
print(st_display)
```

```
## [1] "A(intercept)=1.105, B(lin)=-0.226, C(quad)=0.334"
```

Sixth, to check that the estimates are correct, we derive results from running quadratic estimation with the three points of data drawn. We use both polynomial and orthogonal polynomials below.

```r
# Estimation results
df_rnorm <- as_tibble(mt_rnorm)
# Linear and quadratic terms
rs_lm_quad <- stats::lm(y ~ x + I(x^2), data = df_rnorm)
print(stats::summary.lm(rs_lm_quad))
```

```
##
## Call:
```

```
## stats::lm(formula = y ~ x + I(x^2), data = df_rnorm)
##
## Residuals:
## ALL 3 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.1054        NaN     NaN      NaN
## x            -0.2264        NaN     NaN      NaN
## I(x^2)        0.3343        NaN     NaN      NaN
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 2 and 0 DF,  p-value: NA
```

```
# Using orthogonal polynomials
# vs. rs_lm_quad: different parameters, but same predictions
rs_lm_quad_otho <- stats::lm(y ~ poly(x, 2), data = df_rnorm)
print(stats::summary.lm(rs_lm_quad_otho))
```

```
##
## Call:
## stats::lm(formula = y ~ poly(x, 2), data = df_rnorm)
##
## Residuals:
## ALL 3 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.6383        NaN     NaN      NaN
## poly(x, 2)1   1.3109        NaN     NaN      NaN
## poly(x, 2)2   0.1499        NaN     NaN      NaN
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 2 and 0 DF,  p-value: NA
```

Seventh, now we compare between the predications based on analytical solutions and lm regression.

```
# Matrix of input values
mt_vals_xs <- t(
  matrix(c(1, x1, x1**2, 1, x2, x2**2, 1, x3, x3**2),
    nrow = 3, ncol = 3
  )
)

# Predictions from LM poly prediction
ar_pred_lm <- mt_vals_xs %*% as.numeric(rs_lm_quad$coefficients)
as_pred_lm_otho <- stats::predict(rs_lm_quad_otho)

# Predictions based on analytical solutions
ar_pred_sym <- mt_vals_xs %*% c(fl_A, fl_B, fl_C)

# Combine results
kable(
  cbind(
```

```
    df_rnorm, ar_pred_sym,
    ar_pred_lm, as_pred_lm_otho
  ) %>%
    mutate(res = ar_pred_lm - y),
  caption = paste0(
    "Quadratic Fit of 3 Sets of Random (X,Y) Points"
  )
) %>% kable_styling_fc()
```

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

x	y	ar_pred_sym	ar_pred_lm	as_pred_lm_otho	res
0.4395244	1.070508	1.070508	1.070508	1.070508	0
0.7698225	1.129288	1.129288	1.129288	1.129288	0
2.5587083	2.715065	2.715065	2.715065	2.715065	0

1.2 Formulas for Quadratic Parameters and Differences

Similar to the prior example, we remain interested in quadratic parameters, however, we no longer observe three sets of x and y values, instead, we observe differences in the y values. Specifically, we observe Δy_2 and Δy_3, not y_3, y_2, and y_1. We do still observe x_1, x_2, and x_3. We can no longer identify A, but we can still identify B and C.

$$
\begin{aligned}
y_2 - y_1 = \Delta y_2 = B \cdot (x_2 - x_1) + C \cdot (x_2^2 - x_1^2)\\
y_3 - y_2 = \Delta y_3 = B \cdot (x_3 - x_2) + C \cdot (x_3^2 - x_2^2)
\end{aligned}
$$

First, we rewrite the problem in matrix form.

$$
\begin{bmatrix}
\underbrace{x_2 - x_1}_{T} & \underbrace{x_2^2 - x_1^2}_{U}\\
\underbrace{x_3 - x_2}_{V} & \underbrace{x_3^2 - x_2^2}_{W}
\end{bmatrix}
\cdot
\begin{bmatrix}
B\\
C
\end{bmatrix}
=
\begin{bmatrix}
\underbrace{y_2 - y_1}_{R}\\
\underbrace{y_3 - y_2}_{S}
\end{bmatrix}
$$

Second, we solve the system of equations for the unknown vector $[B, C]$, via elementary row operations. The code below uses matlab to arrive at symbolic analytical solutions for for $[B, C]$. Derivations of the solutions are also shown here: Reduced Row Echelon Form with 2 Equations and 2 Unknowns.

```
clc
clear

% Define inputs
syms T U V W R S
mt_sm_z = [T, U; V, W];
ar_sm_y = [R; S];

% Solve analytically
ar_sm_solu = linsolve(mt_sm_z, ar_sm_y)

% Randomly draw x and y values
rng(1234);
mt_rand = rand(3,2);
% Use below to check not-exact fit, gap actual and predict of y
```

```matlab
mt_rand = [0.1915, 0.6221, 0.4377;
           0.7854, 0.7800, 0.2726]';
% Use below to check for exact fit 2nd 3rd points same
% mt_rand = [0.1915, 0.6221, 0.6221;
%  0.7854, 0.7800, 0.7800]';
[fl_x1, fl_x2, fl_x3] = deal(mt_rand(1,1), mt_rand(2,1), mt_rand(3,1));
[fl_y1, fl_y2, fl_y3] = deal(mt_rand(1,2), mt_rand(2,2), mt_rand(3,2));
[fl_x11, fl_x21, fl_x31] = deal(fl_x1, fl_x2, fl_x3);
[fl_x12, fl_x22, fl_x32] = deal(fl_x1^2, fl_x2^2, fl_x3^2);

% Define values of U V Q and S
fl_T = fl_x21 - fl_x11;
fl_U = fl_x22 - fl_x12;
fl_V = fl_x31 - fl_x21;
fl_W = fl_x32 - fl_x22;
fl_R = fl_y2 - fl_y1;
fl_S = fl_y3 - fl_y2;

% Numerically evaluate coefficients
ar_fl_solu = double(subs(ar_sm_solu, ...
  {T, U, V, W, R, S}, ...
  {fl_T, fl_U, fl_V, fl_W, fl_R, fl_S}));
disp(['ar_fl_solu=', num2str(ar_fl_solu')])

% Y difference predictions
mt_fl_z = [fl_T, fl_U;fl_V, fl_W];
ar_fl_y_diff_pred = mt_fl_z*ar_fl_solu;
ar_fl_x_diff_actual = [fl_T;fl_V];
ar_fl_x_sqr_diff_actual = [fl_U;fl_W];
ar_fl_y_diff_actual = [fl_R;fl_S];

% Compare results
tb_test = array2table( ...
  [ar_fl_x_diff_actual'; ar_fl_x_sqr_diff_actual'; ...
   ar_fl_y_diff_actual'; ar_fl_y_diff_pred']');
cl_col_names = ["x_diff_actual", "x_sqr_diff_actual", ...
                "y_diff_actual", "y_diff_predict"];
cl_row_names = strcat('diff_obs_', string((1:2)));
tb_test.Properties.VariableNames = matlab.lang.makeValidName(cl_col_names);
tb_test.Properties.RowNames = matlab.lang.makeValidName(cl_row_names);
display(tb_test);
```

Third, the solutions for the intercept and quadratic terms based on differences in y values and based on functions of the all x values are as follows.

$$
\begin{aligned}
B &= \frac{RW - SU}{TW - UV} \\
&= \frac{(y_2 - y_1) \cdot \left(x_3^2 - x_2^2\right) - (y_3 - y_2) \cdot \left(x_2^2 - x_1^2\right)}{(x_2 - x_1) \cdot (x_3^2 - x_2^2) - (x_2^2 - x_1^2) \cdot (x_3 - x_2)}
\end{aligned}
$$

$$
\begin{aligned}
C &= \frac{ST - RV}{TW - UV} \\
&= \frac{(y_3 - y_2) \cdot (x_2 - x_1) - (y_2 - y_1) \cdot (x_3 - x_2)}{(x_2 - x_1) \cdot (x_3^2 - x_2^2) - (x_2^2 - x_1^2) \cdot (x_3 - x_2)}
\end{aligned}
$$

Fourth, given three pairs randomly drawn x and y points, we use the formulas just derived to find the parameters for the best-fitting slope and quadratic parameters given observed y differences.

```r
# Inputs X and Y
set.seed(123)
# Draw Randomly
mt_rnorm <- matrix(rnorm(6, mean = 1, sd = 1), nrow = 3, ncol = 2)
# # Three fixed and different set of points
# mt_rnorm <- matrix(c(
#   0.1915, 0.6221, 0.4377,
#   0.7854, 0.7800, 0.2726
# ), nrow = 3, ncol = 2)

colnames(mt_rnorm) <- c("x", "y")
x1 <- mt_rnorm[1, 1]
x2 <- mt_rnorm[2, 1]
x3 <- mt_rnorm[3, 1]
y1 <- mt_rnorm[1, 2]
y2 <- mt_rnorm[2, 2]
y3 <- mt_rnorm[3, 2]

# X quadratic
x11 <- x1
x12 <- x1**2
x21 <- x2
x22 <- x2**2
x31 <- x3
x32 <- x3**2

# Define U and V, as well as Q and S
fl_T <- x21 - x11;
fl_U <- x22 - x12;
fl_V <- x31 - x21;
fl_W <- x32 - x22;
fl_R <- y2 -  y1;
fl_S <- y3 -  y2;

# Shared denominator
fl_denominator <- (fl_T*fl_W - fl_U*fl_V)

# Solve for A and B coefficients (not exact fit)
fl_B <- (fl_R * fl_W - fl_S * fl_U) / fl_denominator
fl_C <- (fl_S * fl_T - fl_R * fl_V) / fl_denominator
```

```r
# Display
st_display <- paste0(
  "B(lin)=", round(fl_B, 3),
  ", C(quad)=", round(fl_C, 3)
)
print(st_display)
```

```
## [1] "B(lin)=-0.226, C(quad)=0.334"
```

Fifth, to check that the estimates are correct, we derive results from running linear estimation with the three points of data drawn, using all level information. As prior, we use both polynomial and orthogonal polynomials below. Note that we can not run this regression in practice because we do not observe levels, just differences.

```r
# Estimation results
df_rnorm <- as_tibble(mt_rnorm)
# Linear and quadratic terms
rs_lm_quad <- stats::lm(y ~ x + I(x^2), data = df_rnorm)
print(stats::summary.lm(rs_lm_quad))
```

```
##
## Call:
## stats::lm(formula = y ~ x + I(x^2), data = df_rnorm)
##
## Residuals:
## ALL 3 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.1054        NaN     NaN      NaN
## x            -0.2264        NaN     NaN      NaN
## I(x^2)        0.3343        NaN     NaN      NaN
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 2 and 0 DF,  p-value: NA
```

```r
# Using orthogonal polynomials
# vs. rs_lm_quad: different parameters, but same predictions
rs_lm_quad_otho <- stats::lm(y ~ poly(x, 2), data = df_rnorm)
print(stats::summary.lm(rs_lm_quad_otho))
```

```
##
## Call:
## stats::lm(formula = y ~ poly(x, 2), data = df_rnorm)
##
## Residuals:
## ALL 3 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.6383        NaN     NaN      NaN
## poly(x, 2)1   1.3109        NaN     NaN      NaN
## poly(x, 2)2   0.1499        NaN     NaN      NaN
##
```

```
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 2 and 0 DF,  p-value: NA
```

Sixth, now we compare between the predications based on analytical solutions using differences in y, and lm regression. The fit of differences in y are exact.

```
# Matrix of input values
mt_vals_xs <- t(
  matrix(c(x2 - x1, x2^2 - x1^2, x3 - x2, x3^2 - x2^2),
    nrow = 2, ncol = 2
  )
)

# Predictions from LM poly prediction
ar_pred_lm <- mt_vals_xs %*% as.vector(rs_lm_quad$coefficients)[2:3]
as_pred_lm_otho_lvl <- stats::predict(rs_lm_quad_otho)
as_pred_lm_otho <- t(t(diff(as_pred_lm_otho_lvl)))

# Predictions based on analytical solutions
ar_pred_sym <- mt_vals_xs %*% c(fl_B, fl_C)

# Combine results
kable(
  cbind(
    as_tibble(apply(df_rnorm, 2, diff)), ar_pred_sym,
    ar_pred_lm, as_pred_lm_otho
  ) %>%
    mutate(res = ar_pred_lm - y),
  caption = paste0(
    "Quadratic Fit, given 3 Sets of Random (X,Y) Points' Differences in Y"
  )
) %>% kable_styling_fc()
```

Quadratic Fit, given 3 Sets of Random (X,Y) Points' Differences in Y

	x	y	ar_pred_sym	ar_pred_lm	as_pred_lm_otho	res
2	0.3302982	0.0587793	0.0587793	0.0587793	0.0587793	0
3	1.7888858	1.5857773	1.5857773	1.5857773	1.5857773	0

1.3 Formulas for Linear Fit with Three Points

We have three points, what are the optimizing intercept and slope parameters? We solve the following problem to minimize the sum-squared error given linear fit with the three points.

First, the minimizing objective function is:

$$O(A, B) = \left\{ (y_1 - A - Bx_1)^2 + (y_2 - A - Bx_2)^2 + (y_3 - A - Bx_3)^2 \right\}$$

Second, we solve for $min_{A,B} O(A, B)$. We take derivative of $O(A, B)$ with respect to A and B:

$$
\begin{aligned}
\frac{\partial O}{\partial A} &= -\left(y_1 - A - Bx_1\right) - \left(y_2 - A - Bx_2\right) - \left(y_3 - A - Bx_3\right) \\
&= A + Bx_1 - y_1 + A + Bx_2 - y_2 + A + Bx_3 - y_3 \\
&= 3A + \left(x_1 + x_2 + x_3\right) B - \left(y_1 + y_2 + y_3\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial O}{\partial B} &= -x_1\left(y_1 - A - Bx_1\right) - x_2\left(y_2 - A - Bx_2\right) - x_3\left(y_3 - A - Bx_3\right) \\
&= \left(x_1 + x_2 + x_3\right) A + \left(x_1^2 + x_2^2 + x_3^2\right) B - \left(x_1 y_1 + x_2 y_2 + x_3 y_3\right)
\end{aligned}
$$

Third, at the optimizing minimum (note quadratic), we now have two equations with two unknowns:

$$
\begin{aligned}
\left(y_1 + y_2 + y_3\right) &= 3A + \left(x_1 + x_2 + x_3\right) B \\
\left(x_1 y_1 + x_2 y_2 + x_3 y_3\right) &= \left(x_1 + x_2 + x_3\right) A + \left(x_1^2 + x_2^2 + x_3^2\right) B
\end{aligned}
$$

Fourth, we rewrite the problem in matrix form.

$$
\begin{bmatrix}
3 & \underbrace{x_1 + x_2 + x_3}_{U} \\
\underbrace{x_1 + x_2 + x_3}_{U} & \underbrace{x_1^2 + x_2^2 + x_3^2}_{V}
\end{bmatrix}
\cdot
\begin{bmatrix} A \\ B \end{bmatrix}
=
\begin{bmatrix}
\underbrace{y_1 + y_2 + y_3}_{Q} \\
\underbrace{x_1 y_1 + x_2 y_2 + x_3 y_3}_{S}
\end{bmatrix}
$$

Fifth, we solve the system of equations for the unknown vector $[A, B]$, via elementary row operations. The code below uses matlab to arrive at symbolic analytical solutions for for $[A, B]$.

```
clc
clear

% Define inputs
syms U V Q S
mt_sm_z = [3, U; U, V];
ar_sm_y = [Q; S];

% Solve analytically
ar_sm_solu = linsolve(mt_sm_z, ar_sm_y)

% Randomly draw x and y values
rng(1234);
mt_rand = rand(3,2);
% Use below to check not-exact fit, gap actual and predict of y
mt_rand = [0.1915, 0.6221, 0.4377;
  0.7854, 0.7800, 0.2726]';
% Use below to check for exact fit 2nd 3rd points same
% mt_rand = [0.1915, 0.6221, 0.6221;
%  0.7854, 0.7800, 0.7800]';
[fl_x1, fl_x2, fl_x3] = deal(mt_rand(1,1), mt_rand(2,1), mt_rand(3,1));
[fl_y1, fl_y2, fl_y3] = deal(mt_rand(1,2), mt_rand(2,2), mt_rand(3,2));
[fl_x11, fl_x21, fl_x31] = deal(fl_x1, fl_x2, fl_x3);
[fl_x12, fl_x22, fl_x32] = deal(fl_x1^2, fl_x2^2, fl_x3^2);

% Define values of U V Q and S
fl_U = fl_x11 + fl_x21 + fl_x31;
fl_V = fl_x12 + fl_x22 + fl_x32;
```

```
fl_Q = fl_y1 + fl_y2 + fl_y3;
fl_S = fl_y1*fl_x11 + fl_y2*fl_x21 + fl_y3*fl_x31;

% Numerically evaluate coefficients
ar_fl_solu = double(subs(ar_sm_solu, ...
  {U, V, Q, S}, ...
  {fl_U, fl_V, fl_Q, fl_S}));
disp(['ar_fl_solu=', num2str(ar_fl_solu')])

% Y predictions
mt_fl_z = [1,fl_x11;1,fl_x21;1,fl_x31];
ar_fl_y_pred = mt_fl_z*ar_fl_solu;
ar_fl_x_actual = [fl_x1;fl_x2;fl_x3];
ar_fl_y_actual = [fl_y1;fl_y2;fl_y3];

% Compare results
tb_test = array2table([ar_fl_x_actual';ar_fl_y_actual';ar_fl_y_pred']');
cl_col_names = ["x_actual","y_actual", "y_predict"];
cl_row_names = strcat('obs_', string((1:3)));
tb_test.Properties.VariableNames = matlab.lang.makeValidName(cl_col_names);
tb_test.Properties.RowNames = matlab.lang.makeValidName(cl_row_names);
display(tb_test);
```

Sixth, the solutions are as follows.

$$\begin{aligned}A &= \frac{QV - SU}{-U^2 + 3V} \\ &= \frac{(y_1 + y_2 + y_3) \cdot \left(x_1^2 + x_2^2 + x_3^2\right) - (x_1 y_1 + x_2 y_2 + x_3 y_3) \cdot (x_1 + x_2 + x_3)}{3 \cdot \left(x_1^2 + x_2^2 + x_3^2\right) - (x_1 + x_2 + x_3)^2}\end{aligned}$$

$$\begin{aligned}B &= \frac{3S - QU}{-U^2 + 3V} \\ &= \frac{3\,(x_1 y_1 + x_2 y_2 + x_3 y_3) - (y_1 + y_2 + y_3) \cdot (x_1 + x_2 + x_3)}{3 \cdot \left(x_1^2 + x_2^2 + x_3^2\right) - (x_1 + x_2 + x_3)^2}\end{aligned}$$

Seventh, given three pairs randomly drawn x and y points, we use the formulas just derived to find the parameters for the best-fitting y-intercept and slope values.

```
# Inputs X and Y
set.seed(123)
# Draw Randomly
mt_rnorm <- matrix(rnorm(6, mean = 1, sd = 1), nrow = 3, ncol = 2)
# # Three fixed and different set of points
# mt_rnorm <- matrix(c(
#   0.1915, 0.6221, 0.4377,
#   0.7854, 0.7800, 0.2726
# ), nrow = 3, ncol = 2)
# # Below, the 2nd and 3rd points are the same
# mt_rnorm <- matrix(c(
#   0.1915, 0.6221, 0.6221,
#   0.7854, 0.7800, 0.7800
# ), nrow = 3, ncol = 2)
```

```r
colnames(mt_rnorm) <- c("x", "y")
x1 <- mt_rnorm[1, 1]
x2 <- mt_rnorm[2, 1]
x3 <- mt_rnorm[3, 1]
y1 <- mt_rnorm[1, 2]
y2 <- mt_rnorm[2, 2]
y3 <- mt_rnorm[3, 2]

# X quadratic
x11 <- x1
x12 <- x1**2
x21 <- x2
x22 <- x2**2
x31 <- x3
x32 <- x3**2

# Define U and V, as well as Q and S
fl_U <- x11 + x21 + x31
fl_V <- x12 + x22 + x32
fl_Q <- y1 + y2 + y3
fl_S <- x11*y1 + x21*y2 + x31*y3

# Shared denominator
fl_denominator <- (3*fl_V - fl_U^2)

# Solve for A and B coefficients (not exact fit)
fl_A <- (fl_Q * fl_V - fl_S * fl_U) / fl_denominator

fl_B <- (3 * fl_S - fl_Q * fl_U) / fl_denominator

# Display
st_display <- paste0(
  "A(intercept)=", round(fl_A, 3),
  ", B(lin)=", round(fl_B, 3)
)
print(st_display)
```

```
## [1] "A(intercept)=0.617, B(lin)=0.813"
```

Eighth, to check that the estimates are correct, we derive results from running linear estimation with the three points of data drawn. We use both polynomial and orthogonal polynomials below.

```r
# Estimation results
df_rnorm <- as_tibble(mt_rnorm)
# Linear and quadratic terms
rs_lm_quad <- stats::lm(y ~ x, data = df_rnorm)
print(stats::summary.lm(rs_lm_quad))
```

```
##
## Call:
## stats::lm(formula = y ~ x, data = df_rnorm)
##
## Residuals:
##        1        2        3
##  0.09601 -0.11374  0.01773
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.61718    0.14533   4.247   0.1472
## x            0.81297    0.09296   8.746   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1499 on 1 degrees of freedom
## Multiple R-squared:  0.9871, Adjusted R-squared:  0.9742
## F-statistic: 76.48 on 1 and 1 DF,  p-value: 0.07248
```

```
# Using orthogonal polynomials
# vs. rs_lm_quad: different parameters, but same predictions
rs_lm_quad_otho <- stats::lm(y ~ poly(x, 1), data = df_rnorm)
print(stats::summary.lm(rs_lm_quad_otho))
```

```
##
## Call:
## stats::lm(formula = y ~ poly(x, 1), data = df_rnorm)
##
## Residuals:
##        1        2        3
##  0.09601 -0.11374  0.01773
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.63829    0.08654  18.931   0.0336 *
## poly(x, 1)   1.31089    0.14989   8.746   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1499 on 1 degrees of freedom
## Multiple R-squared:  0.9871, Adjusted R-squared:  0.9742
## F-statistic: 76.48 on 1 and 1 DF,  p-value: 0.07248
```

Ninth, now we compare between the predications based on analytical solutions and lm regression. Again, note that the fit is not exact.

```
# Matrix of input values
mt_vals_xs <- t(
  matrix(c(1, x1, 1, x2, 1, x3),
    nrow = 2, ncol = 3
  )
)

# Predictions from LM poly prediction
ar_pred_lm <- mt_vals_xs %*% as.vector(rs_lm_quad$coefficients)
as_pred_lm_otho <- stats::predict(rs_lm_quad_otho)

# Predictions based on analytical solutions
ar_pred_sym <- mt_vals_xs %*% c(fl_A, fl_B)

# Combine results
kable(
  cbind(
```

```r
    df_rnorm, ar_pred_sym,
    ar_pred_lm, as_pred_lm_otho
  ) %>%
    mutate(res = ar_pred_lm - y),
  caption = paste0(
    "Linear Fit of 3 Sets of Random (X,Y) Points"
  )
) %>% kable_styling_fc()
```

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

x	y	ar_pred_sym	ar_pred_lm	as_pred_lm_otho	res
0.4395244	1.070508	0.9744999	0.9744999	0.9744999	-0.0960085
0.7698225	1.129288	1.2430232	1.2430232	1.2430232	0.1137354
2.5587083	2.715065	2.6973381	2.6973381	2.6973381	-0.0177269