MEMS 1060: Homework #5

Due on March 4, 2021 at 6:00pm

Professor Sammak Th 6:00PM

Made using LaTeX/Inkscape. Source files available upon request.

Shane Riley

Problem 1

Given points for world population by year, find coefficients for an exponential fit ($p = b * e^{mx}$). Convert the equation to linear form, and use linear squares regression to find the constants. Plot the data and the fit, and estimate the world population in the year 1970 using the fit.

Solution

In order to make the equation linear, we take the natural log of both sides, yielding the following equation and relations:

$$\begin{aligned} ln(p) &= ln(b) + mx \\ y &= ln(p) \\ m &= a_1 \\ b &= e^{a_0} \\ y &= a_1 x + a_0 \end{aligned}$$

Using least squares regression, we find a_0 and a_1. When calling the homemade function to run the regression, notice that the natural log of population is inserted–this is because of the linearization.
With a_0 and a_1, we find m and b using the previously described relations:

$$\begin{aligned} m &= 0.010440[years^{-1}] \\ b &= 4.6315 * 10^{-9}[billions] \end{aligned}$$

With our exponential model specified, we simply plug in the year 1970 for x to find our approximation. This is done in MATLAB using an anonymous function.

$$\begin{aligned} p &= b * e^{mx} \\ p(1970) &= 3.958[billions] \end{aligned}$$

Using least-squares regression, we find $m = 0.010440[years^{-1}]$ and $b = 4.6315 * 10^{-9}[billions]$. With the model, we find $p(1970) = 3.958[billions]$.

Problem 2

Using point data for (x, v), where x is the falling distance from a drop tower, find the experimental value for acceleration due to gravity, g. Know that $v^2 = 2gx$.

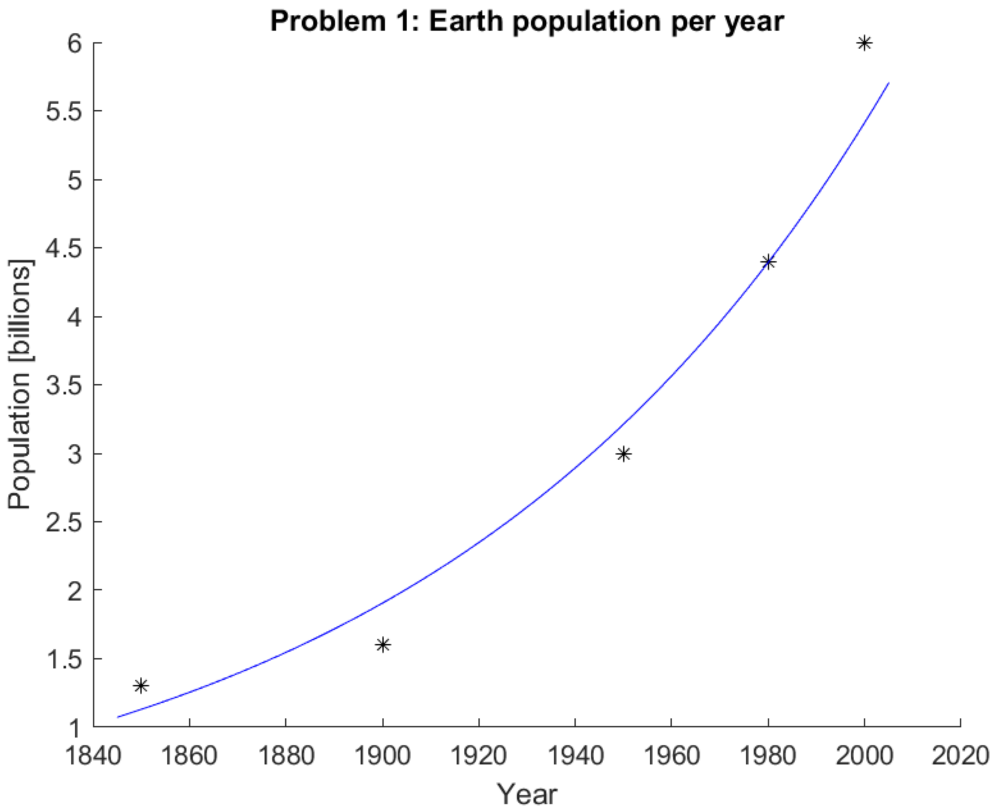
Solution

We can linearize the equation provided and determine g using least-squares regression. We will do this by squaring the provided velocity values. We are going to avoid fixing the y-intercept at 0, since we are dealing with experimental data and it is possible that the object was not perfectly dropped from rest.

$$\begin{aligned} v^2 &= 2gx + v_0^2 \\ y &= v^2 \\ a_1 &= 2g \\ a_0 &= v_0^2 \\ y &= a_1 x + a_0 \end{aligned}$$

Problem 2 continued on next page. . .

Figure 1: Exponential fit for world population



With the modified x and y, we find a_1 and a_0 using least-squares regression (see the source code for the implementation). Since $g = \frac{a_1}{2}$, we compute g using this information.
By using the data and least-squares regression (NOT fixing $v_0 = 0$), we find $g = 9.8509[m/s^2]$.

Problem 3

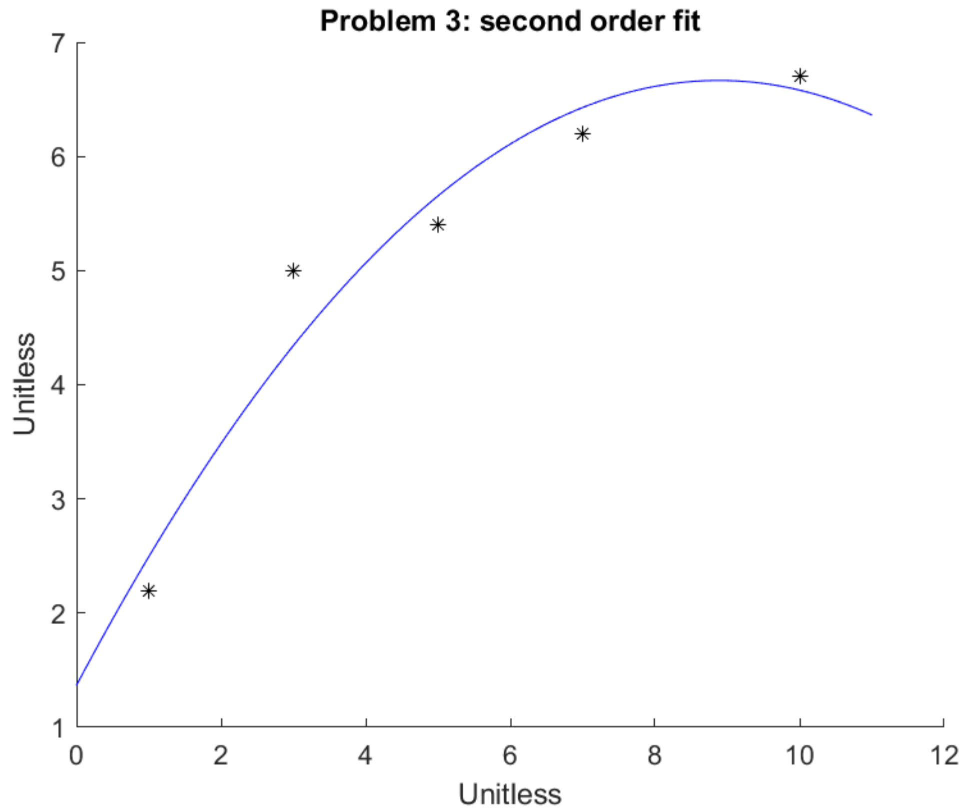
Given x and y point data, find coefficients for a second-order least-squares fit and plot it.
Solution

In order to handle nth-order polynomial regression, a linear system is constructed and solved for the coefficients. As an example, a second-order polynomial system looks like this:

$$\begin{bmatrix} n & S_x & S_{xx} \\ S_x & S_{xx} & S_{xxx} \\ S_{xx} & S_{xxx} & S_{xxxx} \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} S_y \\ S_{xy} \\ S_{xxy} \end{bmatrix}$$

All of the summations can be calculated using vector sums in MATLAB (the subscripts indicate the element summed through the vector), and n is simply the length of the a vector. Once the matrices are populated, the system is solved to find the coefficients. See the source code for more information.
Using polynomial regression, we find $a_0 = 1.3673$, $a_1 = 1.1935$, $a_2 = -0.06722$. The fit is plotted.

Figure 2: Polynomial fit.



Problem 4

Given point data for fuel economy per speed, use quadratic spline interpolation to approximate mpg for speeds 30[mph] and 65[mph].

Solution

Since there are five points provided, we know there will be four splines. Each spline has three coefficients (quadratic), meaning we have 12 degrees of freedom in total. Each of the four splines is fixed on each end to the points bounding it (2 constraints per spline–8 constraints). Additionally, the slopes of the splines should match where they connect (1 constraint per knot–3 constraints). Finally, we specify the second derivative to be zero at the beginning of our interval in order to fully constrain our system. This linear model is constructed and solved in a user-defined function in order to create the coefficients for all the splines. Then, the x-value to be interpolated is paired with spline in the same interval. Finally, the equation for said spline is used to interpolate for the y-value. Some general equations are described below for a specific point x (fixing value and ensuring first-derivative compatibility):

$$a_0x^2 + b_0x + c_0 = a_1x^2 + b_1x + c_1$$
$$2a_0x + b_0 = 2a_1x + b_1$$

Since we are fixing the second derivative as 0 at the beginning of the first spline, we are setting the a coefficient for the first spline as 0. See the source code.

Using quadratic spline interpolation, we find $mpg(30[mph]) = 29.317[mpg]$ **and** $mpg(65[mph]) = 30.429[mpg]$

```
%% MEMS 1060 Homework 5
% Author: Shane Riley
% Date: 3/4/2021
format long
%% Problem 1
% Given population/year data, find an eponential curve of best fit by
% linearizing and using linear least-squares regression. Estimate the
% population in 1970 and plot the regression with the points.
disp(" ");
disp("Problem 1");

years = [...
    1850
    1900
    1950
    1980
    2000];  % [years]
populations = [...
    1.3
    1.6
    3
    4.4
    6];  % [billions]

% p = b*exp(m x)
% ln both sides
% ln(p) = ln(b) + (m x)

ln_populations = log(populations);
a_p1 = myPolyReg(years, ln_populations, 1);

% Post process, get the estimate
m = a_p1(2);
b = exp(a_p1(1));
model = @(x) b*exp(m .* x);
pop_1970 = model(1970);

% Prints
disp("m:");
disp(m);
disp("b:");
disp(b);
disp("Population in 1970:");
disp(pop_1970);

% Plot
linyears = linspace(min(years) - 5, max(years) + 5, 100);
hold on
figure(1)
```

```
plot(years, populations, 'k*'); % plot input points
plot(linyears, model(linyears), 'b-'); % Plot fit
title("Problem 1: Earth population per year");
xlabel("Year");
ylabel("Population [billions]");
print("images/figure1",'-dpng');
hold off


%% Problem 2
% Given v(x) data points, find g using linear regression
disp(" ");
disp("Problem 2");

height = [...
    0
    5
    10
    15
    20
    25]; % [m]

fall_speed = [...
    0
    9.85
    14.32
    17.63
    19.34
    22.41]; % [m/s]

% v^2 = 2 g x

fall_speed_squared = fall_speed .^ 2;

a_p2 = myPolyReg(height, fall_speed_squared, 1); % first order fit

g_p2 = a_p2(2)/2;

% Prints
disp("Experimental value for g: ");
disp(g_p2);


%% Problem 3
% Given y(x) data points, find a second-order polynomial fit
disp(" ");
disp("Problem 3");

x = [...
```

```
    1
    3
    5
    7
    10];  % [-]

y = [...
    2.2
    5.0
    5.4
    6.2
    6.7];

% find a, where a(1) is lowest-order coefficient
a_p3 = myPolyReg(x,y,2);

% Print coefficients
for i=1:length(a_p3)
   disp("a" + num2str(i-1) + ":");
   disp(a_p3(i));
end

polynomial_fit = @(x) a_p3(1) + (a_p3(2) .* x) + (a_p3(3) .* (x .^2));

% Plot
linx_p3 = linspace(min(x) - 1, max(x) + 1, 100);
figure(2)
hold on
plot(x, y, 'k*');  % plot input points
plot(linx_p3, polynomial_fit(linx_p3), 'b-');  % Plot fit
title("Problem 3: second order fit");
xlabel("Unitless");
ylabel("Unitless");
print("images/figure2",'-dpng');
hold off



%% Problem 4
% Use quadratic splines interpolation to find mpg(30 mph), mpg(65 mph),
% given mpg(mph) data points
disp(" ");
disp("Problem 4");

drive_speed = [...
    11
    25
    40
    55
```

```
    70];  % [mph]

fuel_economy = [...
    13
    26
    28
    30
    24];  % [mpg]

disp("MPG at 30 mph: ");
disp(myQuadraticSplineInterp(drive_speed, fuel_economy, 30));

disp("MPG at 65 mph: ");
disp(myQuadraticSplineInterp(drive_speed, fuel_economy, 65));

%% Supporting functions

function a = myPolyReg(x, y, n_in)
% MYPOLYREG creates a vector of coefficients for n-th order polynomial fit
% using least-squares regression.
% x - input vector
% y - output vector (length matches x)
% n_in - order (n >= 1)
% a - coefficient vector (n=length(a))

% n_in is order; n is number of coefficients
n = n_in + 1;

% Build the A matrix
% TODO: make fewer redundant calculations
sx = @(x,n) sum(x.^(n));
A = zeros(n);
for i=1:n
    for j=1:n
        order = (i-1) + (j-1);
        A(i,j) = sx(x,order);
    end
end

% Build the B matrix
sxy = @(x,y,n) sum((x.^(n) .* y));
b = zeros(n,1);
for i=1:n
   order = i - 1;
   b(i) = sxy(x,y,order);
end

% Find coefficients
a = A\b;
```

```
end

function Yint = myQuadraticSplineInterp(x, y, Xint)
% MYQUADRATICSPLINEINTERP approximates y(Xint) given points x,y using
% quadratic splines
% x - input vector (MUST BE ASCENDING)
% y - output vector (length matches x)
% Xint - input for interpolation
% Yint - output for interpolation

num_splines = length(x) - 1;
n = length(x);
num_dof = (num_splines * 3);

% Specify matrices
A = zeros(num_dof);
b = zeros(num_dof, 1);

% Fix at points
for i=1:num_splines
    a_col = i*3 - 2;
    b_col = i*3 - 1;
    c_col = i*3;

    % Row 2i - 1: fix start point
    row = 2*i - 1;
    A(row,a_col) = x(i).^2;
    A(row,b_col) = x(i);
    A(row,c_col) = 1;
    b(row) = y(i);

    % Row 2i: fix end point

    row = 2*i;
    b(row) = y(i+1);
    A(row,a_col) = x(i+1).^2;
    A(row,b_col) = x(i+1);
    A(row,c_col) = 1;
end

% Differentiable at knots
row = num_dof - num_splines + 1;
for i=2:num_splines
    pre_a_col = 3*i - 5;
    post_a_col = 3*i - 2;
    pre_b_col = 3*i - 4;
    post_b_col = 3*i - 1;

    A(row, pre_a_col) = 2 * x(i);
```

```
    A(row, pre_b_col) = 1;
    A(row, post_a_col) = - 2 * x(i);
    A(row, post_b_col) = -1;
    b(row) = 0;

    row = row + 1;
end

% Fix a0 as 0
A(num_dof, 1) = 1;
b(num_dof) = 0;

coeff = A\b;

% Pick the spline number
for i=1:n
    if Xint < x(i+1)
        which_spline = i;
        break;
    end
end

a_int = coeff(3*which_spline - 2);
b_int = coeff(3*which_spline - 1);
c_int = coeff(3*which_spline);

model = @(x) (a_int .* (x.^2)) + (b_int .* x) + c_int;
Yint = model(Xint);

end
```