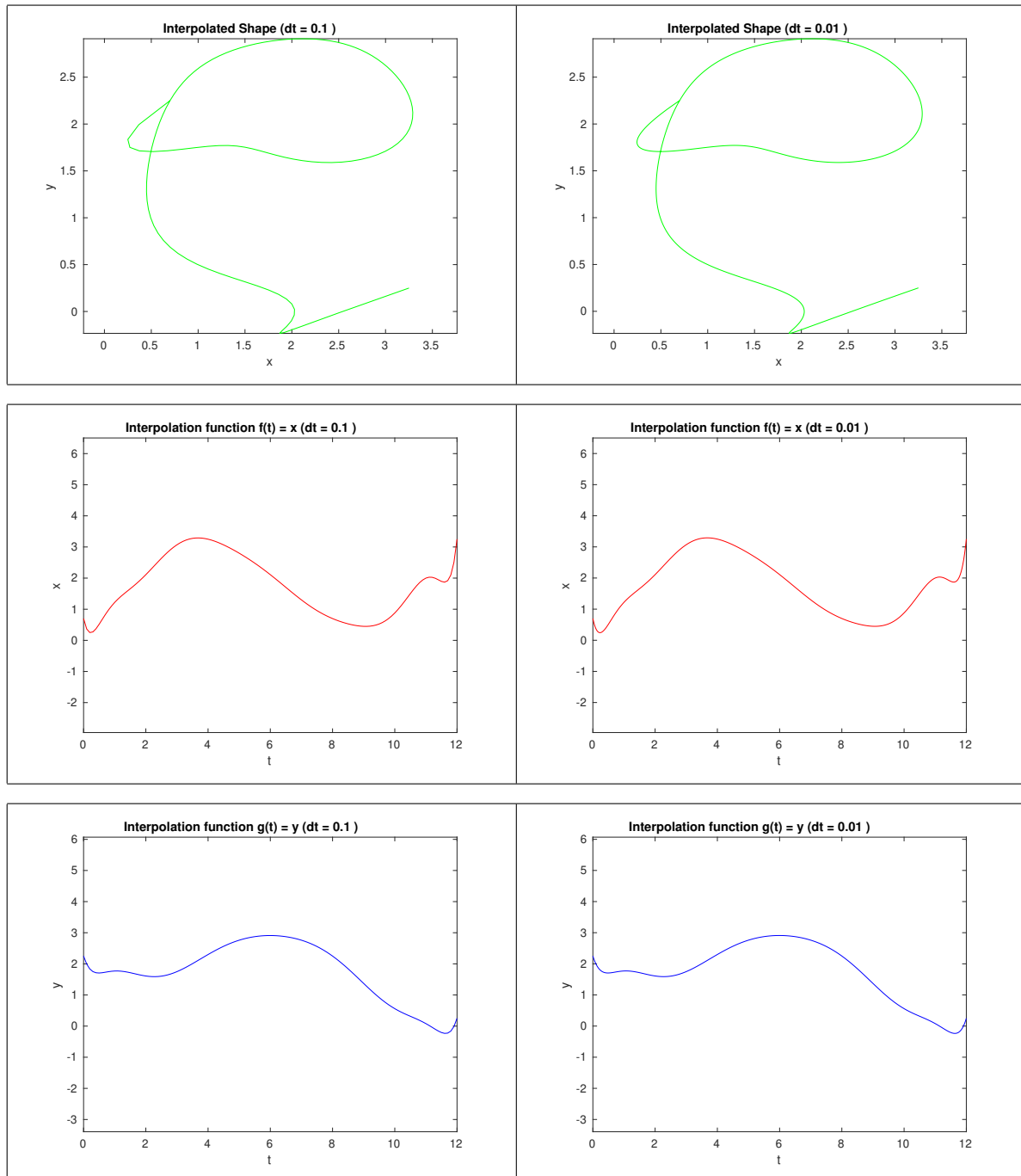


Q1. The root of the equation $x - e^{-x} = 0$ is obtained by using iterated inverse interpolation and Neville's Algorithm. Neville's Algorithm was implemented using a recursive function. To calculate the relative error, the actual root of the equation is taken as the one obtained by using the Newton-Raphson Method (with a tolerance of 10^{-10}) upto 10 decimal digits. The results are listed below.

Approximate root (obtained using iterated interpolation)	0.567143
Actual root (obtained using Newton's Method)	0.5671432904
Relative Error (%)	0.000118

Q2. Parametric interpolation was used to interpolate the desired shape with two different step sizes: (i) $dt = 0.1$ and (ii) $dt = 0.01$. However, improvement in interpolation quality was not very significant. The interpolated shape as well as interpolation functions $f(t)$ and $g(t)$ are also shown below.



We can see that the improvement in interpolated shape quality is very very minimal. Further reduction of step size too did not yield any significant improvement in image quality.

Q3. Problem 2 was repeated using natural cubic splines. The coefficients for each of the 11 splines for the interpolation functions $f(t)$ and $g(t)$ are reported below.

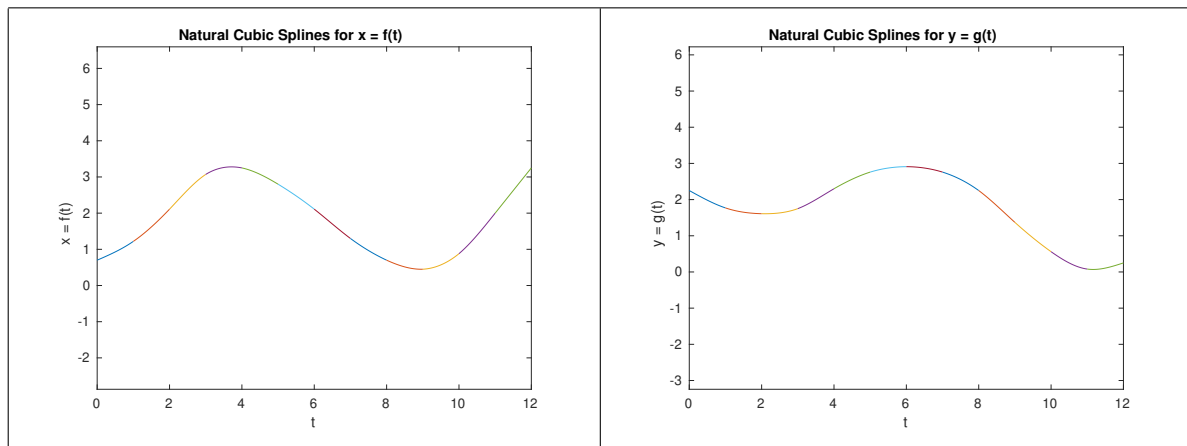
Coefficients of natural cubic splines ($x = f(t)$)

j	a_j	b_j	c_j	d_j
0	0.7	0.43658	0	0.083416
1	1.22	0.68683	0.25025	-0.047082
2	2.11	1.0552	0.109	-0.20417
3	3.07	0.6187	-0.50352	0.064819
4	3.25	-0.21963	-0.30906	0.078692
5	2.8	-0.60776	-0.072982	-0.0092611
6	2.11	-0.7899	-0.10077	0.080667
7	1.3	-0.73766	0.14124	-0.003575
8	0.7	-0.45504	0.13051	0.074527
9	0.45	0.059072	0.35409	0.016837
10	0.88	0.85148	0.4046	-0.13608
11	2	1.2524	-0.0036506	0.0012169

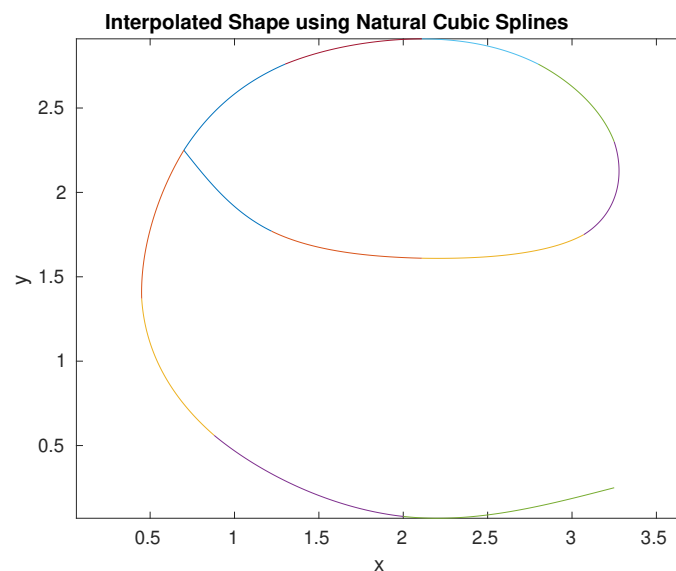
Coefficients of natural cubic splines ($y = g(t)$)

j	a_j	b_j	c_j	d_j
0	2.25	-0.55228	0	0.07228
1	1.77	-0.33544	0.21684	-0.041398
2	1.61	-0.018237	0.092645	0.065591
3	1.75	0.38795	0.28942	-0.12737
4	2.3	0.57696	-0.092678	-0.024286
5	2.76	0.30495	-0.16554	0.010583
6	2.91	-0.0055208	-0.13379	-0.01069
7	2.76	-0.31899	-0.16586	-0.02515
8	2.25	-0.74627	-0.24131	0.10758
9	1.37	-0.89937	0.081421	0.0079496
10	0.56	-0.70391	0.10527	0.11864
11	0.08	-0.13746	0.46118	-0.15373

Plots of interpolation functions $f(t)$ and $g(t)$ follow. The splines are differentiated by color.



The interpolated shape can be seen in the figure below.



Comments: It is clearly visible that natural cubic splines yield a much better interpolated shape than parametric interpolation done in Q2. The differences are pretty much obvious as we compare the images. The interpolation functions obtained by using natural cubic splines are much smoother with lesser variations in curvature as compared to the ones obtained by parametric interpolation.

Parametric interpolation approximates the shape using a polynomial of order 12. The intermediate data points are points on this polynomial. A polynomial of such a high order will have many local maxima and minima as the data are not monotonous. This also means that there will be points where the curvature changes sign. But in the context of our problem, where we are printing letters, we already know that the entire letter is one smooth stroke, and parametric interpolation fails to imitate that. Spline interpolation achieves this by interpolating each interval with a lower order polynomial that has a better resemblance to the smooth strokes in the shape.

Q4. A linear least squares fit to the equation $y = be^{-2\pi ax}$ is achieved by the following change of variable.

$$y = be^{-2\pi ax}$$

$$\implies \ln(y) = \ln(b) + (-2\pi a)x$$

A least squares fit is then obtained between $\ln(y)$ and x . The slope and intercept of the resulting line can then be transformed to obtain the parameters a and b .

$$b = e^{\text{intercept}}$$

$$a = \text{slope} / (-2\pi)$$

The obtained values of the parameters for the linear least-squares fit are given below:

$$a = 0.0061 \text{ and } b = 16.8640.$$

Attempting to construct a non-linear fit to this data using the given data directly is not possible since the parameters a and b cannot be found exactly for an exponential least squares fit. It might be possible to obtain approximate values of the parameters using iterative solvers or optimization strategies. Assuming that we are able to find the values of a and b for the non-linear least squares fit, these values will be **different** from the values we obtained previously for a linear fit achieved by a change of variable.

The error for the linear fit at the i^{th} data point is

$$(E_{\text{linear}})_i = (\ln y_i - \ln b_1 + 2\pi a_1 x_i)^2$$

where a_1 and b_1 are the least-squares fit parameters for the linear model.

The error for the non-linear fit at the i^{th} data point is

$$(E_{\text{non-linear}})_i = (y_i - b_2 e^{-2\pi a_2 x_i})^2$$

where a_2 and b_2 are the least-squares fit parameters for the non-linear model.

Given that a_1 and b_1 minimize the overall error $\sum (E_{\text{linear}})_i$; a_2 and b_2 minimize the overall error $\sum (E_{\text{non-linear}})_i$, it cannot be proven that $a_1 = a_2$ and $b_1 = b_2$.

MATLAB Code - Q1**filename - hw3p1.m**

```
%
% DS 288 Homework 3, Problem 1
% Nikhil Jayswal, SR - 16961
%

clear; clc;
close all

%% test data
% x = [1.0 1.3 1.6 1.9 2.2];
% y = [0.7651977 0.6200860 0.4554022 0.2818186 0.1103623];
% y(1.5) = 0.5118200

%% given data
x = [0.3 0.4 0.5 0.6];
y = [0.740818 0.670320 0.606531 0.548812];

% using neville's method, find root of  $x - y = 0$ 
eval_point = 0;
%tic
    guess = neville(eval_point, y-x, x);
%toc

fprintf('Root obtained by Neville''s method is %.6f.\n', guess)

% calculate actual root using Newton's Method
init_root = 0.5;
tol = 1e-10;
while (1 > 0)
    a = init_root - exp(-init_root);
    b = 1 + exp(-init_root);
    root = init_root - a/b;
    if abs(root - init_root) < tol
        break
    end
    init_root = root;
end
```

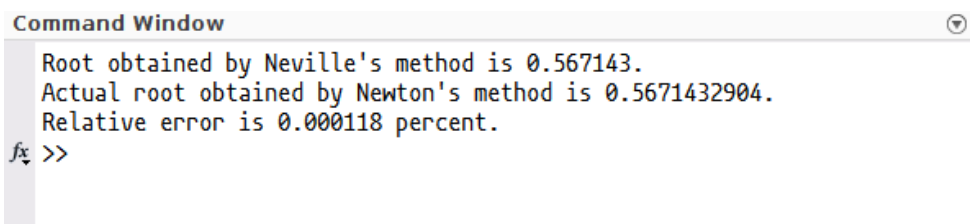
```
rel_error = 100 * abs(root - guess)/root;
fprintf('Actual root obtained by Newton's method is %.10f.\n', root)
fprintf('Relative error is %.6f percent.\n', rel_error)
```

filename - neville.m

```
function [value] = neville(eval_point, xval, yval)
%
% function that implements Neville's Algorithm
% using recursion
%

    if length(xval) == 1
        value = yval;
    else
        x1 = xval(1:end-1);
        x2 = xval(2:end);
        y1 = yval(1:end-1);
        y2 = yval(2:end);
        value = (eval_point - xval(1))*neville(eval_point, x2, y2) ...
                - (eval_point - xval(end))*neville(eval_point, x1, y1);
        value = value/(xval(end) - xval(1));
    end
end
```

Command Window Output - Q1

A screenshot of the MATLAB Command Window. The title bar says "Command Window". The output text is: "Root obtained by Neville's method is 0.567143.", "Actual root obtained by Newton's method is 0.5671432904.", and "Relative error is 0.000118 percent." Below the output, there is a prompt "fx >>" with a cursor.

```
Command Window
Root obtained by Neville's method is 0.567143.
Actual root obtained by Newton's method is 0.5671432904.
Relative error is 0.000118 percent.
fx >>
```

MATLAB Code - Q2**filename - hw3p2.m**

```
%
% DS 288 Homework 3, Problem 2
% Nikhil Jayswal, SR - 16961
%

clc; clear;
close all

%% given data
t = (0.0:1.0:12.0);
x = [0.70 1.22 2.11 3.07 3.25 ...
      2.80 2.11 1.30 0.70 0.45 0.88 2.00 3.25];
y = [2.25 1.77 1.61 1.75 2.30 ...
      2.76 2.91 2.76 2.25 1.37 0.56 0.08 0.25];

%% obtain interpolation functions
%tic
dt = 0.5;
tp = 0:dt:12;
xp = zeros(size(tp));
yp = zeros(size(tp));
for i = 1:length(tp)
    xp(i) = neville(tp(i), t, x);
    yp(i) = neville(tp(i), t, y);
end

%% make plots
figure
plot(tp, xp, '-r')
axis equal
titlestr = ['Interpolation function f(t) = x (dt = ', num2str(dt), ' )'];
title(titlestr)
xlabel('t')
ylabel('x')
figure
plot(tp, yp, '-b')
axis equal
titlestr = ['Interpolation function g(t) = y (dt = ', num2str(dt), ' )'];
```



```
title(titlestr)
xlabel('t')
ylabel('y')
figure
plot(xp, yp, '-g')
axis equal
titlestr = ['Interpolated Shape (dt = ', num2str(dt), ' )'];
title(titlestr)
xlabel('x')
ylabel('y')
%toc
```

MATLAB Code - Q3**filename - hw3p3.m**

```
%
% DS 288 Homework 3, Problem 3
% Nikhil Jayswal, SR - 16961
%

clc; clear;
close all

%% given data
t = (0.0:1.0:12.0);
x = [0.70 1.22 2.11 3.07 3.25 ...
     2.80 2.11 1.30 0.70 0.45 0.88 2.00 3.25];
y = [2.25 1.77 1.61 1.75 2.30 ...
     2.76 2.91 2.76 2.25 1.37 0.56 0.08 0.25];

%% obtain splines for x = f(t) and y = g(t)
%tic
[bx, cx, dx] = ncspline(t, x);
[by, cy, dy] = ncspline(t, y);

%% make plots
dt = 0.01;
for i = 1:length(t) - 1
    tp = t(i):dt:t(i+1);
    xp = zeros(length(tp), 1);
    yp = zeros(length(tp), 1);
    for j = 1:length(tp)
        h = tp(j) - t(i);
        xp(j) = x(i) + bx(i)*h + cx(i)*h^2 + dx(i)*h^3;
        yp(j) = y(i) + by(i)*h + cy(i)*h^2 + dy(i)*h^3;
    end
    figure(1)
    plot(tp, xp)
    hold on
    figure(2)
    plot(tp, yp)
    hold on
    figure(3)
```

```
        plot(xp, yp)
        hold on
    end
    figure(1)
    axis equal
    title('Natural Cubic Splines for x = f(t)')
    xlabel('t')
    ylabel('x = f(t)')
    figure(2)
    axis equal
    title('Natural Cubic Splines for y = g(t)')
    xlabel('t')
    ylabel('y = g(t)')
    figure(3)
    axis equal
    xlabel('x')
    ylabel('y')
    title('Interpolated Shape using Natural Cubic Splines')
    %toc
```

filename - ncspline.m

```
function [b, c, d] = ncspline(x, y)
%
% function that returns a natural cubic spline
% that fits the given data points
%
    h = zeros(1, length(x) - 1);
    for i = 1:length(h)
        h(i) = x(i+1) - x(i);
    end
    alpha = zeros(size(h));
    for j = 2:length(h)
        alpha(j) = (3*(y(j+1) - y(j))/h(j)) - ...
            (3*(y(j) - y(j-1))/h(j-1));
    end
    l = zeros(size(x));
    mu = zeros(size(h));
    z = zeros(size(x));
    l(1) = 1;
    mu(1) = 0;
```

```
z(1) = 0;
for j = 2:length(h)
    l(j) = 2*(x(j+1) - x(j-1)) - h(j-1)*mu(j-1);
    mu(j) = h(j)/l(j);
    z(j) = (alpha(j) - h(j-1)*z(j-1))/l(j);
end
l(end) = 1;
z(end) = 0;
c = zeros(size(x));
c(end) = 0;
b = zeros(size(h));
d = zeros(size(h));
k = length(h);
while k > 0
    c(k) = z(k) - mu(k)*c(k+1);
    b(k) = ((y(k+1) - y(k))/h(k)) - h(k)*(c(k+1) + 2*c(k))/3;
    d(k) = (c(k+1) - c(k))/(3*h(k));
    k = k - 1;
end
c = c(1:end-1);
end
```

MATLAB Code - Q4**filename - hw3p4.m**

```
%
% DS 288 Homework 3, Problem 4
% Nikhil Jayswal, SR - 16961
%

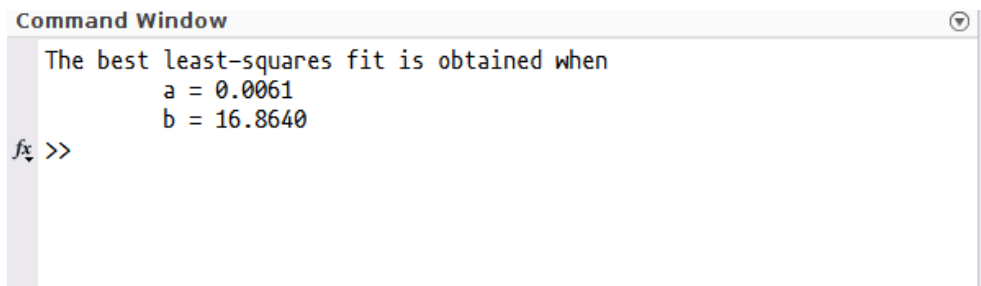
clc; clear;
close all

%% given data
x = [0 16 32];
y = [17 9 5];

%% obtain a least squares fit
%  $\ln(y) = \ln(b) + (-2\pi a)x$ 
yp = log(y);
n = length(x);
sum_x = sum(x);
sum_y = sum(yp);
sum_xy = sum(x.*yp);
sum_xx = sum(x.^2);
slope = (n*sum_xy - sum_x*sum_y)/(n*sum_xx - sum_x^2);
intercept = (sum_xx*sum_y - sum_xy*sum_x)/(n*sum_xx - sum_x^2);
a = slope/(-2*pi);
b = exp(intercept);

%% print results
fprintf('The best least-squares fit is obtained when \n')
fprintf('\t a = %.4f\n', a)
fprintf('\t b = %.4f\n', b)
```

Command Window Output - Q4

A screenshot of the MATLAB Command Window. The window title is "Command Window". The text inside shows the output of a least-squares fit: "The best least-squares fit is obtained when", followed by "a = 0.0061" and "b = 16.8640" on separate lines. At the bottom left, the prompt "fx >>" is visible.

```
Command Window
The best least-squares fit is obtained when
    a = 0.0061
    b = 16.8640
fx >>
```