Here’s MATLAB code for **each question from your Week-6 (Time Domain Analysis) – Exercises sheet**.
I’ll assume some reasonable expressions where the PDF figure is unclear, and I’ll clearly mark those places so you can easily edit them to match your exact graphs.

Q1 – Convolution of (x(t) = \dfrac{1}{1 + t^2}) with (u(t))

```
%% Q1: Convolution x(t) * u(t)
clear; clc; close all;

% Assumption from sheet: x(t) = 1/(1 + t^2)
% You can change this if your x(t) is slightly different.

dt = 0.001;                 % time step
t_min = -10;
t_max =  10;
t = t_min:dt:t_max;

% Define signals
x = 1 ./ (1 + t.^2);        % x(t)
u = double(t >= 0);         % unit step u(t)

% Plot x(t) and u(t)
figure;
subplot(3,1,1);
plot(t, x, 'LineWidth', 1.5);
grid on;
xlabel('t'); ylabel('x(t)');
title('Q1: x(t)');

subplot(3,1,2);
plot(t, u, 'LineWidth', 1.5);
grid on;
xlabel('t'); ylabel('u(t)');
title('Unit step u(t)');

% Numerical convolution x * u
y = conv(x, u) * dt;        % scale by dt for continuous-time approximation

% Time axis for convolution result
t_conv = (t(1) + t(1)) : dt : (t(end) + t(end));

subplot(3,1,3);
plot(t_conv, y, 'LineWidth', 1.5);
```

```
grid on;
xlabel('t'); ylabel('y(t)');
title('Convolution y(t) = x(t) * u(t)');

% Zoom into a more "appropriate" range if needed
xlim([-5 5]);
```

Q2
You're right – earlier I'd guessed a triangular (g(t)).
From your figure, the correct signals are:
- (x(t) = \sin t) for (0 \le t \le 2\pi), 0 otherwise
- (g(t) = 1) for (0 \le t \le 2\pi), 0 otherwise (rectangular pulse)

Here's MATLAB code **just for Q2**, with all three parts:

```
%% Q2: c(t) = x(t) * g(t) for given signals
clear; clc; close all;

dt = 0.001;

% Time axis large enough for convolution result [0,4*pi]
t = -1:dt:5*pi;          % you can adjust if you like

%% -----------------------------------------------------------------------
% 1) Original signals: x(t) time-limited, g(t) rectangular [0, 2*pi]
%    x(t) = sin(t) on [0, 2*pi], 0 otherwise
%    g(t) = 1      on [0, 2*pi], 0 otherwise
% ------------------------------------------------------------------------

x_fin = sin(t) .* (t >= 0   & t <= 2*pi);     % finite-duration x(t)
g     = 1      .* (t >= 0   & t <= 2*pi);     % given g(t)

% Convolution (continuous-time approximated)
c_fin = conv(x_fin, g) * dt;

% Time axis for convolution result
t_c = (t(1) + t(1)) : dt : (t(end) + t(end));

figure;
subplot(3,1,1);
plot(t, x_fin, 'LineWidth', 1.5);
grid on; xlabel('t'); ylabel('x(t)');
title('Q2: Original x(t) (time-limited)');

subplot(3,1,2);
```

```
plot(t, g, 'LineWidth', 1.5);
grid on; xlabel('t'); ylabel('g(t)');
title('g(t) = 1 on [0, 2\pi]');

subplot(3,1,3);
plot(t_c, c_fin, 'LineWidth', 1.5);
grid on; xlabel('t'); ylabel('c(t)');
xlim([0 4*pi]);          % support should be [0, 4*pi]
title('c(t) = x(t) * g(t), with finite-duration x(t)');

%% --------------------------------------------------------------------------
% 2) Case (i): x(t) is everlasting     (x(t) = sin(t) for all t)
% ---------------------------------------------------------------------------

t2 = -10:dt:10;          % bigger window to mimic everlasting
x_ever = sin(t2);        % everlasting sine in simulation
g2 = 1 .* (t2 >= 0 & t2 <= 2*pi);

c_ever = conv(x_ever, g2) * dt;
t2_c = (t2(1) + t2(1)) : dt : (t2(end) + t2(end));

figure;
subplot(3,1,1);
plot(t2, x_ever, 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('x(t)');
title('Case (i): x(t) everlasting = sin(t)');

subplot(3,1,2);
plot(t2, g2, 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('g(t)');
title('g(t) (same as before)');

subplot(3,1,3);
plot(t2_c, c_ever, 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('c(t)');
title('c(t) = x(t) * g(t) with everlasting x(t)');

%% --------------------------------------------------------------------------
% 3) Case (ii): x(t) everlasting AND frequency of x(t) is changed
%    Example: use x(t) = sin(2t) instead of sin(t)
% ---------------------------------------------------------------------------
```

```
x_ever2 = sin(2*t2);       % higher frequency
c_ever2 = conv(x_ever2, g2) * dt;    % convolve with same g(t)

figure;
subplot(3,1,1);
plot(t2, x_ever, 'LineWidth', 1.2); hold on;
plot(t2, x_ever2, '--', 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('x(t)');
legend('sin(t)','sin(2t)');
title('Everlasting x(t) with different frequencies');

subplot(3,1,2);
plot(t2_c, c_ever, 'LineWidth', 1.2); hold on;
plot(t2_c, c_ever2, '--', 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('c(t)');
legend('c(t) for sin(t)','c(t) for sin(2t)');
title('Effect of changing frequency of x(t) on c(t)');

subplot(3,1,3);
plot(t2, g2, 'LineWidth', 1.2);
grid on; xlabel('t'); ylabel('g(t)');
title('g(t) (for reference)');
```

Here's a simple MATLAB script that defines the two signals exactly as in the figure, computes (c(t)=x(t)*g(t)) numerically (continuous–time conv ≈ discrete conv), and plots all three.

```
clc; clear; close all;

%% Sampling grid (continuous-time approximation)
dt = 0.001;                  % time step
t  = -2:dt:10;               % overall time axis (covers both signals)

%% Define x(t)
% x(t) = 2 for -1 <= t <= 2, 0 otherwise
x = 2 * ( (t >= -1) & (t <= 2) );

%% Define g(t)
% g(t) = (2/3)(t-1) for 1 <= t <= 4   (ramp from 0 to 2)
%      = 3       for 5 <= t <= 7     (rectangular pulse)
%      = 0       otherwise
g = (2/3)*(t-1) .* ( (t >= 1) & (t <= 4) ) ...
  + 3         .* ( (t >= 5) & (t <= 7) );

%% Convolution c(t) = x(t) * g(t)
c_num = conv(x, g, 'full') * dt;                % scale by dt for CT conv
t_c   = (t(1) + t(1)) : dt : (t(end) + t(end));  % time axis for convolution

%% Plots
figure;
```

subplot(3,1,1);
plot(t, x, 'LineWidth', 1.8);
grid on;
xlabel('t (s)'); ylabel('x(t)');
title('x(t)');

subplot(3,1,2);
plot(t, g, 'LineWidth', 1.8);
grid on;
xlabel('t (s)'); ylabel('g(t)');
title('g(t)');

subplot(3,1,3);
plot(t_c, c_num, 'LineWidth', 1.8);
grid on;
xlabel('t (s)'); ylabel('c(t)');
title('c(t) = x(t) * g(t)');

If your teacher used a different height for the rectangle in (g(t)) (e.g. 2 instead of 3), just change the 3 in the definition of gto that value.

Q4 – Periodic input (x(t)) and impulse response (h(t) = t(u(t) - u(t-1.5)))

The figure suggests a periodic (x(t)). The text mentions **"If x(t) variation is t instead of t²"**, so I'll assume:

- One period of (x(t)) is defined on (-1 \le t \le 2)
- Original (x(t) = t^2) on this interval, 0 elsewhere, repeated periodically
- Alternate case: (x(t) = t) on the same interval.

You can adapt the exact shape/interval if your figure is different.

```
%% Q4: Periodic x(t) through system h(t) = t (u(t) - u(t-1.5))
clear; clc; close all;

dt = 0.001;
t = -5:dt:5;   % overall visualization range

% ----- Define one period of x(t) -----
T0 = 3;    % Period length (assume from -1 to 2 -> length 3)

% Helper: map any t to equivalent t within base interval [-1, 2)
t_mod = mod(t + 1, T0) - 1;   % Now t_mod is in [-1, 2)

% Case 1: x(t) = t^2 over [-1, 2)
x_period1 = (t_mod.^2) .* (t_mod >= -1 & t_mod <= 2);
x1 = x_period1;   % full periodic signal

% Case 2: x(t) = t over [-1, 2)
x_period2 = (t_mod) .* (t_mod >= -1 & t_mod <= 2);
x2 = x_period2;
```

```
% ---- Impulse response h(t) = t(u(t)-u(t-1.5)) ----
h_range = 0:dt:5;  % h(t) nonzero only on [0, 1.5]
h = h_range .* (h_range >= 0 & h_range <= 1.5);

% Convolution for case 1
y1 = conv(x1, h) * dt;
t_y1 = (t(1) + h_range(1)) : dt : (t(end) + h_range(end));

% Convolution for case 2
y2 = conv(x2, h) * dt;
t_y2 = t_y1;

% Plot x(t) and y(t) over -3 <= t <= 3
figure;
subplot(3,1,1);
plot(t, x1, 'LineWidth', 1.2);
grid on;
xlim([-3 3]);
xlabel('t'); ylabel('x(t)');
title('Q4: Periodic x(t) with t^2');

subplot(3,1,2);
plot(t_y1, y1, 'LineWidth', 1.2);
grid on;
xlim([-3 3]);
xlabel('t'); ylabel('y(t)');
title('y(t) = x(t) * h(t), with x(t) = t^2 (periodic)');

subplot(3,1,3);
plot(t_y2, y2, 'LineWidth', 1.2);
grid on;
xlim([-3 3]);
xlabel('t'); ylabel('y(t)');
title('y(t) = x(t) * h(t), with x(t) = t (periodic)');

% You can also build a table of "integration ranges" manually if needed for
theory.
```

If your base interval or formula of (x(t)) in Fig.1 is different, just change:

- T0, and
- the expression for x_period1 and x_period2.

Q5 – Stability classification and plotting forced & free responses

We'll use the Control System Toolbox:

- Represent each system as a transfer function (H(s)).
- Get poles to check stability.
- Plot:
  - **Forced response (ZSR)**: step response (or any chosen input with lsim).
  - **Free response**: response to initial conditions, with zero input (initial).

```
%% Q5: Stability & responses for different LTIC systems
clear; clc; close all;

% Requires Control System Toolbox

s = tf('s');

% (a) (D^2 + 8D + 12) y(t) = (D − 1) x(t)
H_a = (s − 1) / (s^2 + 8*s + 12);

% (b) D(D^2 + 3D + 2) y(t) = (D + 5) x(t)
H_b = (s + 5) / (s * (s^2 + 3*s + 2));

% (c) D^2 (D^2 + 2) y(t) = x(t)
H_c = 1 / (s^2 * (s^2 + 2));

% (d) (D + 1)(D^2 − 6D + 5) y(t) = (3D + 1) x(t)
H_d = (3*s + 1) / ((s + 1) * (s^2 − 6*s + 5));

% (e) (D + 1) (D^2 + 2D + 5)^2 y(t) = x(t)
H_e = 1 / ((s + 1) * (s^2 + 2*s + 5)^2);

% (f) (D + 1)(D^2 + 9) y(t) = (2D + 9) x(t)
H_f = (2*s + 9) / ((s + 1) * (s^2 + 9));

% (g) (D + 1)(D^2 + 9)^2 y(t) = (2D + 9) x(t)
H_g = (2*s + 9) / ((s + 1) * (s^2 + 9)^2);

% (h) (D^2 + 1)(D^2 + 4)(D^2 + 9) y(t) = (3D) x(t)
H_h = 3*s / ((s^2 + 1)*(s^2 + 4)*(s^2 + 9));

systems = {H_a, H_b, H_c, H_d, H_e, H_f, H_g, H_h};
labels  = {'a','b','c','d','e','f','g','h'};

% Time vector for simulations
```

```
t = 0:0.01:20;

for k = 1:length(systems)
    H = systems{k};
    label = labels{k};

    fprintf('System (%s):\n', label);
    poles_H = pole(H);
    zeros_H = zero(H);
    disp('  Poles:'); disp(poles_H.');
    disp('  Zeros:'); disp(zeros_H.');

    % Basic BIBO stability check (all poles in open left half-plane)
    if all(real(poles_H) < 0)
        fprintf('  -> BIBO stable (all poles in left half-plane)\n');
    elseif any(real(poles_H) > 0)
        fprintf('  -> BIBO unstable (pole in right half-plane)\n');
    else
        fprintf('  -> Marginal (poles on j-omega axis, none in RHP)\n');
    end

    fprintf('\n');

    % ----- Forced response (ZSR): use step response as example -----
    figure;
    subplot(2,1,1);
    step(H, t); grid on;
    title(sprintf('System (%s): Forced response (ZSR) to unit step', label));

    % ----- Free (natural) response: response to initial conditions -----
    % Convert to state-space and set some non-zero initial condition
    ssH = ss(H);
    nx = size(ssH.A,1);
    x0 = ones(nx,1);  % simple nonzero IC

    subplot(2,1,2);
    initial(ssH, x0, t); grid on;
    title(sprintf('System (%s): Free (natural) response (zero input)',
label));
end
```

This code:

- Prints poles and a simple verbal classification for BIBO stability.

- Shows, for each system:
    - **Step response** (forced/ZSR).
    - **Initial-condition response** (free/natural).

You can also test other inputs for ZSR using lsim(H, u, t) with any u(t).

If you want, next I can:

- Align the **exact piecewise definitions** of (x(t)) and (g(t)) once you tell me their formulas from the figures, and
- Add comments in the code that match each theoretical range (integrating limits) you had to derive by hand.