2.1)

```
% Combined script for angvel2skew, skew2angvel, and unit test
% angvel2skew.m
function W_hat = angvel2skew(w)
   % Create a 3x3 skew-symmetric matrix from a 3-element vector w
   W_hat = [0, -w(3), w(2);
            w(3), 0, -w(1);
           -w(2), w(1), 0];
end
% skew2angvel.m
function w = skew2angvel(W_hat)
   % Extract the angular velocity vector w from a 3x3 skew-symmetric matrix W_hat
   w = [W_hat(3,2); W_hat(1,3); W_hat(2,1)];
end
% Unit test for angvel2skew and skew2angvel functions
% Generate a random angular velocity vector w
w = rand(3, 1);
% Call angvel2skew to convert w to a skew-symmetric matrix W_hat
W_hat = angvel2skew(w);
% Call skew2angvel to convert the skew-symmetric matrix W_hat back to an angular velocity vector w_prime
w_prime = skew2angvel(W_hat);
% Check if w and w_prime are approximately equal (within a tolerance)
tolerance = 1e-6;
is_equal = all(abs(w - w_prime) < tolerance);
if is_equal
   disp('Test is passed as angvel2skew and skew2angvel are inverse to each other.');
else
   disp('Test is failed as angvel2skew and skew2angvel are not inverse to each other.');
end
```

2.2)

```
% Combined script for twist2rbvel, rbvel2twist, and unit test
% twist2rbvel.m
function V_hat = twist2rbvel(V)
    % Create a 4x4 rigid body velocity matrix in homogeneous coordinates from a 6-element twist vector V
    V_hat = [skewSymmetricMatrix(V(1:3)), V(4:6);
             zeros(1, 4)];
end
function W_hat = skewSymmetricMatrix(w)
    % Create a 3x3 skew-symmetric matrix from a 3-element vector w
    W_hat = [0, -w(3), w(2);
             w(3), 0, -w(1);
            -w(2), w(1), 0];
end
% rbvel2twist.m
function V = rbvel2twist(V_hat)
    % Extract the 6-element twist vector V from a 4x4 rigid body velocity matrix in homogeneous coordinates V_hat
    V = [V_hat(1:3, 4); unskewSymmetricMatrix(V_hat(1:3, 1:3))];
end
function w = unskewSymmetricMatrix(W_hat)
    % Extract a 3-element vector w from a 3x3 skew-symmetric matrix W_hat
    w = [W_hat(3, 2); W_hat(1, 3); W_hat(2, 1)];
end
% Unit test for twist2rbvel and rbvel2twist functions
% Generate a random twist vector V
V = rand(6, 1);
% Call twist2rbvel to convert V to a 4x4 rigid body velocity matrix in homogeneous coordinates V_hat
V_hat = twist2rbvel(V);
```

```
% Call rbvel2twist to convert the rigid body velocity matrix V_hat back to a twist vector V_prime
V_prime = rbvel2twist(V_hat);
% Check if V and V_prime are approximately equal (within a tolerance)
tolerance = 1e-6;
is_equal = all(abs(V - V_prime) < tolerance);
if is_equal
   disp('Test is passed as twist2rbvel and rbvel2twist are inverse of each other.');
else
   disp('Test is failed as twist2rbvel and rbvel2twist are not inverse of each other.');
end
```

2.3)

```
% tform2adjoint.m

function Adg = tform2adjoint(g)

    % Extract the rotation matrix R and translation vector p from the homogeneous transformation matrix g

    R = g(1:3, 1:3);

    p = g(1:3, 4);

    % Create the 6x6 adjoint transformation matrix Adg

    Adg = zeros(6, 6);

    % Populate the upper-left 3x3 block with R

    Adg(1:3, 1:3) = R;

    % Populate the lower-right 3x3 block with R

    Adg(4:6, 4:6) = R;

    % Compute the 3x3 skew-symmetric matrix p_hat from the translation vector p

    p_hat = [0, -p(3), p(2);

             p(3), 0, -p(1);

            -p(2), p(1), 0];

    % Populate the upper-right 3x3 block with p_hat

    Adg(1:3, 4:6) = p_hat;

end
```

2.4)

```
% compare_twist.m
function compare_twist()
    % Generate a random 4x4 homogeneous transformation matrix g
    g = random_homogeneous_matrix();
    % Compute the spatial velocity Vs_hat and body velocity Vb_hat
    Vs_hat = compute_spatial_velocity(g);
    Vb_hat = compute_body_velocity(g);
    % Convert the spatial velocity Vs_hat to a spatial twist Vs
    Vs = rbvel2twist(Vs_hat);
    % Convert the body velocity Vb_hat to a body twist Vb
    Vb = rbvel2twist(Vb_hat);
    % Display the generated transformation matrix g
    disp('Generated Transformation Matrix g:');
    disp(g);
    % Display the spatial and body twists
    disp('Spatial Twist Vs:');
    disp(Vs);
    disp('Body Twist Vb:');
    disp(Vb);
    % Check if the spatial twist and body twist are approximately equal (within a tolerance)
    tolerance = 1e-6;
    is_equal = all(abs(Vs - Vb) < tolerance);
    if is_equal
        disp('Test is passed as Spatial Twist and Body Twist are equal.');
    else
        disp('Test is Failed as Spatial Twist and Body Twist are not equal.');
    end
end
```

```
function g = random_homogeneous_matrix()
    % Generate a random 4x4 homogeneous transformation matrix
    R = random_rotation_matrix();
    p = rand(3, 1);
    g = eye(4);
    g(1:3, 1:3) = R;
    g(1:3, 4) = p;
end
function R = random_rotation_matrix()
    % Generate a random 3x3 rotation matrix
    theta = rand() * 2 * pi;
    v = rand(3, 1);
    v = v / norm(v);
    K = [0, -v(3), v(2);
         v(3), 0, -v(1);
        -v(2), v(1), 0];
    R = eye(3) + sin(theta) * K + (1 - cos(theta)) * K^2;
end
function Vs_hat = compute_spatial_velocity(g)
    % Compute the spatial velocity Vs_hat from the transformation matrix g
    Vs_hat = eye(4);
    Vs_hat(1:3, 4) = g(1:3, 4);
end
function Vb_hat = compute_body_velocity(g)
    % Compute the body velocity Vb_hat from the transformation matrix g
    Vb_hat = g;
    Vb_hat(1:3, 4) = [0; 0; 0];
end
```

2.5)

```
% compare_twist.m
function [g, Vs_Adg, Vs, Vb] = compare_twist()
    % Generate a random 4x4 homogeneous transformation matrix g
    g = random_homogeneous_matrix();
    % Compute the spatial velocity Vs_hat and body velocity Vb_hat
    Vs_hat = compute_spatial_velocity(g);
    Vb_hat = compute_body_velocity(g);
    % Convert the spatial velocity Vs_hat to a spatial twist Vs
    Vs = rbvel2twist(Vs_hat);
    % Convert the body velocity Vb_hat to a body twist Vb
    Vb = rbvel2twist(Vb_hat);
    % Compute the adjoint transformation matrix Adg
    Adg = tform2adjoint(g);
    % Compute Vs_Adg by applying the adjoint transformation
    Vs_Adg = Vs * Adg;
    % Display the generated transformation matrix g
    disp('Generated Transformation Matrix g:');
    disp(g);
    % Display Vs, Vs_Adg, and Vb
    disp('Spatial Twist Vs:');
    disp(Vs);
    disp('Vs_Adg (Spatial Twist after Adjoint Transformation):');
    disp(Vs_Adg);
    disp('Body Twist Vb:');
    disp(Vb);
    % Check if Vs and Vs_Adg are approximately equal (within a tolerance)
    tolerance = 1e-6;
    is_equal = all(abs(Vs - Vs_Adg) < tolerance);
```

```
    if is_equal
        disp('Test is passed as Vs and Vs_Adg are identical.');
    else
        disp('Test is failed as Vs and Vs_Adg are not identical.');
    end
end
% Rest of the functions (random_homogeneous_matrix, random_rotation_matrix,
% compute_spatial_velocity, compute_body_velocity, tform2adjoint) remain the same as in the previous answer.
```