Part 1: Report

Question 1: Bisection Method

For the bisection method I reorganized the equation to be set to zero, as this ensures that p is known to be 0 and I could check my approximation against an error tolerance to terminate the loop, which I set to **10^-5**. On the **16th** iteration of the algorithm (i = 15) the solution was found to be **–0.6518**.
Initial interval **a0 = -1** and **b0 = 0** was chosen because f(a0) is negative, and f(b0) is positive, ensuring that f(p) = 0 exists within this interval, and each approximation was calculated using the midpoint rather than the golden ratio.
In the below table of the first 11 iterations, I've included the values that were gained by plugging in the approximated value (from the midpoint) into the function as this displays how close to zero each approximation was, and thus how close to p the value was getting:

Iteration #	Approx. Value	F(Value)
0	-0.5000	0.53125
1	-0.7500	-0.5380859375
2	-0.6250	0.117279052734
3	-0.6875	-0.174601554871
4	-0.6563	-0.020474582911
5	-0.6406	0.050358883105
6	-0.6484	0.015442326578
7	-0.6523	-0.002389686596
8	-0.6504	0.006557754775
9	-0.6514	0.002091914678
10	-0.6519	-0.00014691307

Question 2: Fixed Point Iteration

Fixed point iteration was a difficult method to perform due to the need to find a good iterating function, and an appropriate interval that would ensure convergence at g(x) = x. This method converged after **6** iterations to the value -**0.651823453823.** The iteration function **g(x) = x - ((x^5 - x^4 + x^3 - x^2 + 1) / (5*x^4 - 4*x^3 + 3*x^2 - 2*x))** was used, as this function is guaranteed to converge with the line y = x when an interval of [-1, -0.4] is chosen since:
g(-1) = -0.78, which is above the line y = x and within the interval [-1, -0.4]
g(-0.4) = -0.85, which is below the line y = x and within the interval [-1, -0.4]
$|g'(-1)|$ = 0.85, which is a constant k such that $0 < k < 1$
$|g'(-0.4)|$ = 0.85, which = k, and $0 < k < 1$
These findings, along with the knowledge that g(x) is continuous along the interval, indicate that there must be a point within the interval where g(x) = x by the **Fixed-Point Theorem**. Initial value **p0 = -1** was chosen since it is within the interval evaluated above, and is the same initial value chosen with other methods, allowing for better comparison. **Note:** table values below were found by running the algorithm without an error check, whereas code output in Part 2 only runs until the 6th iteration, where it exits because the fixed point is found.

Iteration #	Value
0	-0.785714285714
1	-0.678004746268
2	-0.653006305773
3	-0.651825971642
4	-0.651823453835
5	-0.651823453823
6	-0.651823453823
7	-0.651823453823
8	-0.651823453823
9	-0.651823453823
10	-0.651823453823

Question 3: Newton's Method

When using the same error tolerance as Question 1 (**10^-5**) the approximation **-0.651823453823** is found within **6** iterations using Newton's Method. Initial value **p0 = -1** was chosen because it was known to be close to the value and is consistent with other initial values chosen in other questions; 0 was not an appropriate initial value because f'(0) = 0, which you cannot divide by. This method initially took some time to compute each value, as both f(x) and f'(x) had to be computed each iteration, so to reduce the computation time I rounded the values to 12 decimal points, which successfully improved performance. Note: table values below, like in question 2, were found by disabling the error checking and allowing all iterations to run.

Iteration #	Value
0	-0.785714285714
1	-0.678004746268
2	-0.653006305773
3	-0.651525971642
4	-0.651823453835
5	-0.651823453823
6	-0.651823453823
7	-0.651823453823
8	-0.651823453823
9	-0.651823453823
10	-0.651823453823

Question 4: Secant Method

Using the secant method, the approximation -**0.65182345382** was found after **12** iterations using the same error tolerance of **10^-5**. Initial values **p0 = 0** and **p1= -1** were chosen because f(p0) and f(p1) have opposite signs, ensuring that f(x) intersects with the x axis, and that this method will find where the function intersects with the x axis.
When I first ran the algorithm using the initial value -0.5 instead of 0 for p0 (f(-0.5) is the same sign as f(0) so these are both values that could be chosen) the sequence converged within **7** iterations, but I received an error on the 11th iteration because it could not divide by zero (since fp1 and fp0 were the same,). Thus, I chose p0 = 0 in order to be able to display all values, and to keep my initial value choices consistent through the different methods:

Iteration #	Value
0	0.0000000000000
1	-1.000000000000
2	-0.250000000000
3	-0.425579655946
4	-0.975396421235
5	-0.539229121216
6	-0.598844440129
7	-0.664299818788
8	-0.650590704997
9	-0.651795985959
10	-0.651823514947

Question 5: Method Comparison

Iteration #	Bisection	Fixed-Point Iteration	Newton's Method	Secant Method
0	-0.5000	-0.785714285714	-0.785714285714	0.0000000000000
1	-0.7500	-0.678004746268	-0.678004746268	-1.000000000000
2	-0.6250	-0.653006305773	-0.653006305773	-0.250000000000
3	-0.6875	-0.651825971642	-0.651525971642	-0.425579655946
4	-0.6563	-0.651823453835	-0.651823453835	-0.975396421235
5	-0.6406	-0.651823453823	-0.651823453823	-0.539229121216
6	-0.6484	-0.651823453823	-0.651823453823	-0.598844440129
7	-0.6523	-0.651823453823	-0.651823453823	-0.664299818788
8	-0.6504	-0.651823453823	-0.651823453823	-0.650590704997
9	-0.6514	-0.651823453823	-0.651823453823	-0.651795985959
10	-0.6519	-0.651823453823	-0.651823453823	-0.651823514947
Iteration Converged On	**16**	**6**	**6**	**12**

Using the interval [-1, 0] and/or the initial value of –1 or 0 the methods with the fastest convergence rate are Fixed-Point iteration and Newton's Method at 6 iterations, the Secant Method second with 12, and Bisection third with 16. Generally, one would expect that Newton's Method would converge the fastest out of all methods. The reason that Fixed-Point iteration converged at the same pace I believe is because of the similarities in the cases: initial point p0 = -1 was chosen for both methods, and the iterating function derived for fixed point iteration:

$$\mathbf{g(x)} = x - ((x^5 - x^4 + x^3 - x^2 + 1) / (5*x^4 - 4*x^3 + 3*x^2 - 2*x)) = \mathbf{x - (f(x) / f'(x))}$$

is in the same form as the Newton's Method formula **f(xn+1) = xn – f(xn) / f'(xn).** All methods required finding the appropriate interval and initial values: the simplest method was Bisection, then Newton's and Secant method, fixed-point iteration required the most complex set-up as an iterating function that converged with y = x and did not result in imaginary numbers was quite difficult to find. Therefore, because of its convergence speed, simple to use formula, and for this function differentiation was simple and straightforward; Newton's Method is the best.

Part 2: Code

Hyperlinks lead to the published code from Matlab and display their output. Each question was done in a separate file.

Question 1: Bisection Method

```
%------------------Q1: bisection method------------------
%equation:x^2 + x^4 + 6 = x^3 + x^5 + 7

%function
syms x
f = x^5 - x^4 + x^3 - x^2 + 1;
%pi = calculated estimation, p = value being estimated
pi = 1;
p = 0;
%interval [a,b]
a = -1;
b = 0;
%error tolerance
tolerance = 0.00001;

for i = 0:20
    error = abs(pi - p);
    %check if the solution is found to be within the set tolerance
    if(error <= tolerance)
        disp(i)
        disp("approximation found");
        disp(mid);
        break;
    end
    %Find interval midpoint
    mid = ((b - a) / 2) + a ;
    %f(mid)
    pi = subs(f, x, mid);
    pi = round(pi, 12);
    if(i < 11)
        disp(i);
        disp(mid);
    end
    %redefine interval to bisect previous interval
    if(pi < 0)
        a = mid;
    elseif(pi > 0)
       b = mid;
    end
end
```

Question 2: Fixed Point Iteration

```
%------------------Q2: Fixed Point Iteration------------------

%equation:
%---------------------- x^2 + x^4 + 6 = x^3 + x^5 + 7 ----------------------
%function f(x) = 0  = g(x) = x
syms x

g = x - ((x^5 - x^4 + x^3 - x^2 + 1) / (5*x^4 - 4*x^3 + 3*x^2 - 2*x));
tolerance = 0.00001;
p0 = -1;

for i = 0:10
    p = subs(g, x, p0);
    p = round(p, 12);
    disp(i);
    disp(p);
    %to display all 11 iterations comment out lines 18-21
    if (abs(p - p0) < tolerance)
        disp("fixed point found");
        break;
    end
    p0 = p;
end

```

Question 3: Newton's Method

```
%------------------Q3: Newton's Method------------------

%equation:
%---------------------- x^2 + x^4 + 6 = x^3 + x^5 + 7 ---------------------
%function f(x) = 0
syms x
f = x^5 - x^4 + x^3 - x^2 + 1;
%f'(x)
f_deriv = 5*x^4 - 4*x^3 + 3*x^2 - 2*x;
%initial approximation
p0 = -1;
%error tolerance
tolerance = 0.00001;

for i = 0:10
    fp = subs(f, x, p0);
    fd_p = subs(f_deriv, x, p0);
    p = p0 - (fp / fd_p);
    p = round(p,12);
    %to display all 11 iterations comment out lines 21-26
    if(abs(p-p0) < tolerance)
        disp("approximation found");
        disp(i);
        disp(p);
        break;
    end
    disp(i);
    disp(p);
    p0 = p;
end

```

Question 4: Secant Method

```
%------------------Q4: Secant Method------------------

%equation:
%---------------------- x^2 + x^4 + 6 = x^3 + x^5 + 7 ----------------------
%function f(x) = 0
syms x
f = x^5 - x^4 + x^3 - x^2 + 1;

%initial approximation
p0 = 0;
p1 = -1;
%f(p0) and f(p1)
fp0 = subs(f,x,p0);
fp1 = subs(f,x,p1);
%error tolerance
tolerance = 0.00001;

for i = 2:15
    p = p1 - ((fp1 * (p1 - p0)) / (fp1 - fp0));
    p = round(p,12);
    e = p - p1;
    disp(i);
    if(abs(e) < tolerance)
        disp("approx found:");
        disp(p);
        break;
   end
    %display first 11 iterations
    if(i < 11)
        disp(p);
    end
    %reassign for next iteration
    p0 = p1;
    p1 = p;
    fp0 = fp1;
    fp1 = subs(f,x,p);
end
if(i == 20)
    disp("approx not found");
end
```