Project C - continued fractions

Table of Contents

Candidate number: 1075844

Exercise C1

```
% Initialise the command window and clear existing figures.
clear
clf

type get_cont_frac.m
% Use the function to calculate continued fractions for any real number.

% Calculate and display the coefficients in the continued fractions with
% n = 14 for the given numbers.
a1 = get_cont_frac(exp(1),14)
a2 = get_cont_frac(pi,14)
a3 = get_cont_frac((1+sqrt(5))/2,14)
a4 = get_cont_frac(sqrt(2),14)
a5 = get_cont_frac(3.0578,14)


% The function takes in two values: n, the maximum number of
% iterations / calculate up to a_n; and x, the real number that will be
% written as a continued fraction.
% It then outputs an array consisting of a_i, i = 0,1,2...n, in (5.1).

function a = get_cont_frac(x,m)

    % Find a_0.
    a = floor(sym(x));

    % Note: sym(x) is used and it does not significantly slow down the
    % programme, so it can be kept for small n.


    % Find a_1, a_2, ... a_n
    for i = 2:m+1
        x = 1 / (sym(x) - a(i-1)); % change the value of x in each iteration
        if x == Inf
            break
        % If x == Inf, then sym(x) - a(i-1) == 0, which means sym(x) == a(i-1)
```

```
        % is an integer. So the continued fraction is finite and should end.
        end
        a(i) = floor(sym(x));
    end
end


a1 =

[2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10]


a2 =

[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1]


a3 =

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]


a4 =

[1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]


a5 =

[3, 17, 3, 3, 9, 3]
```

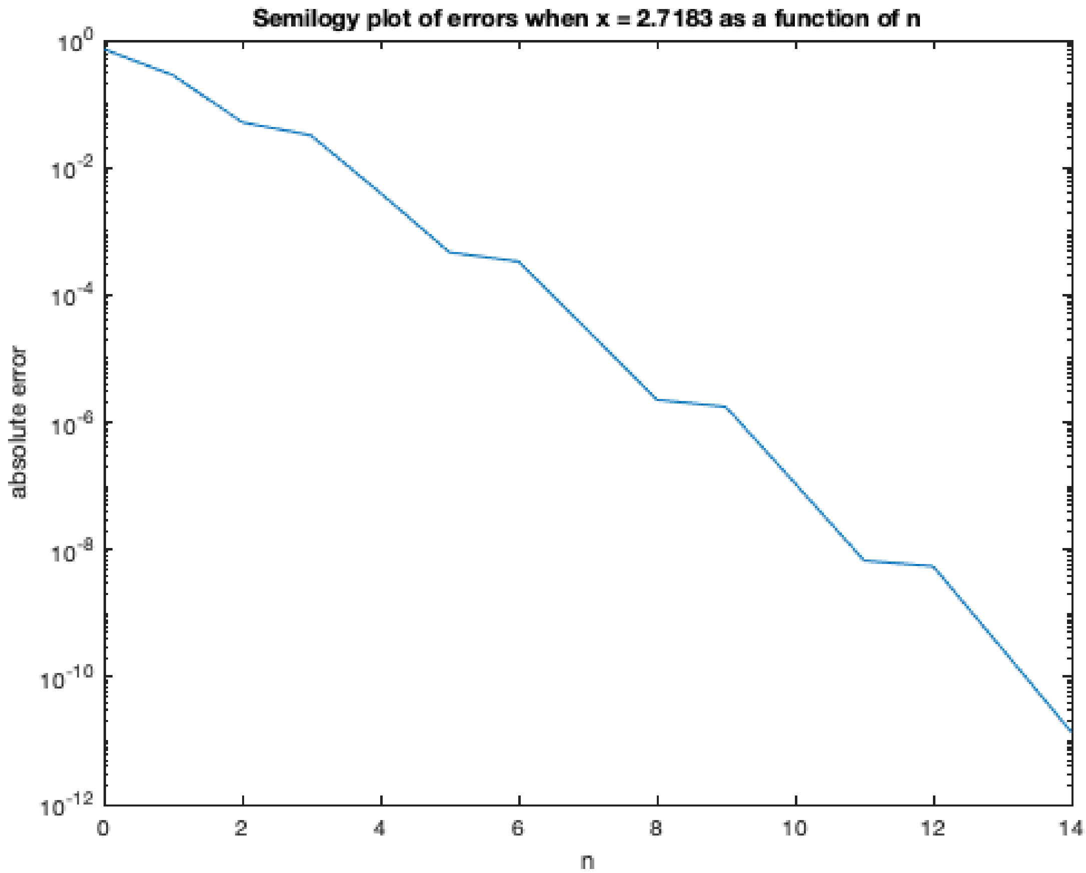
Exercise C2

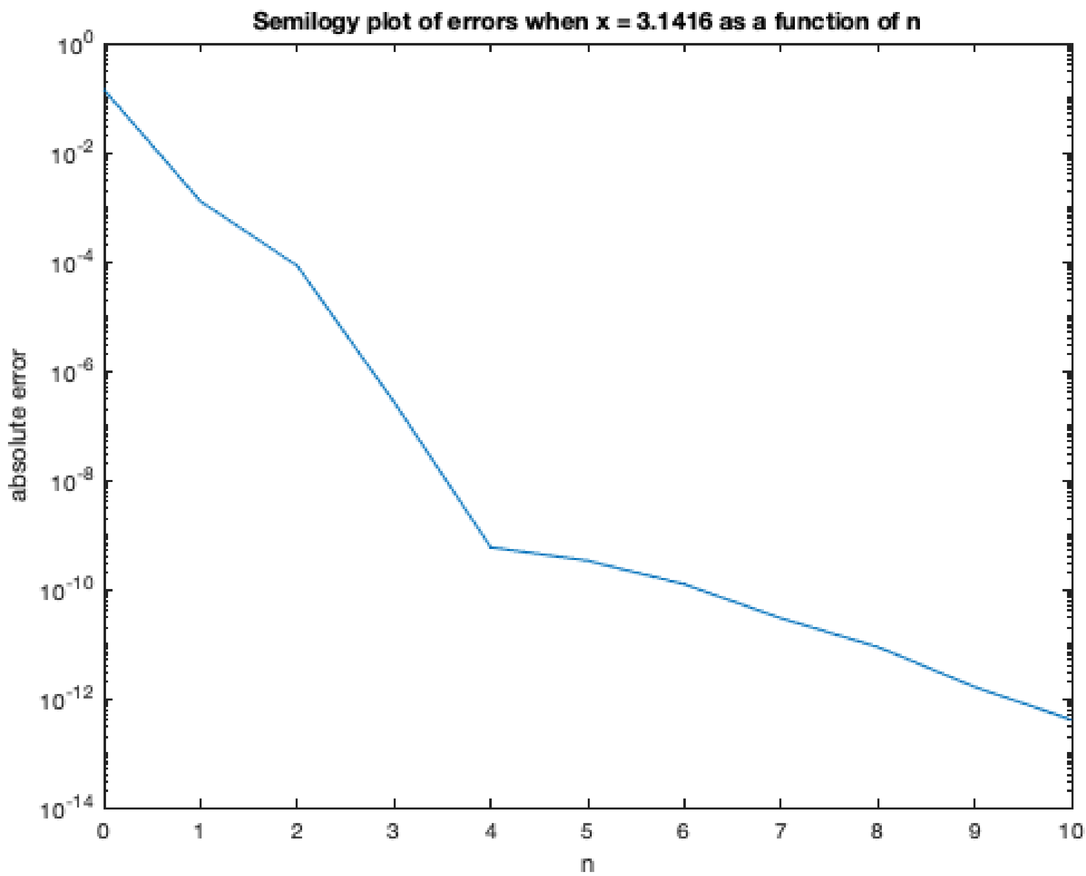
```
type convergents.m

% Calculate and display convergents for each of the numbers in C1.
% Only the ratio r_k = p_k / q_k is needed. But for the 3rd number, phi,
% the list of q_n is required for part C3.
[~,~,r1] = convergents(a1); r1
[~,~,r2] = convergents(a2); r2
[~,q3,r3] = convergents(a3); r3
[~,~,r4] = convergents(a4); r4
[~,~,r5] = convergents(a5); r5

type errors_plot_14.m

% Use the function to plot semilogy graphs of n for each number
figure(1)
errors_plot_14(exp(1))

figure(2)
errors_plot_14(pi)
```

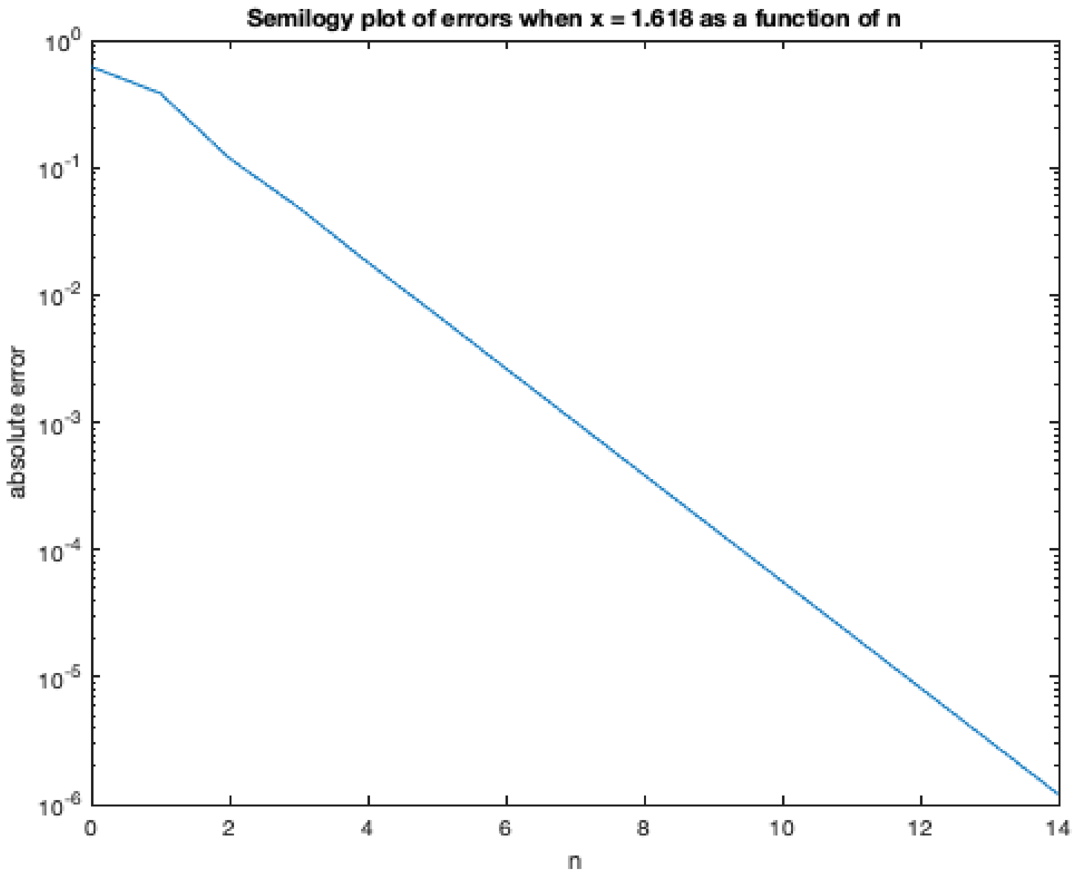
```
figure(3)
errors_plot_14((1+sqrt(5))/2)

figure(4)
errors_plot_14(sqrt(2))

figure(5)
errors_plot_14(3.0578)


% This function takes in any list a, likely being generated from the
% funciton 'continued_fraction', and outputs three lists, p_k, q_k,
% and the convergents r_k = p_k / q_k

function [p,q,r] = convergents(a)

    % Find the length of array a, which should also be the length of p and q.
    m = length(a);
    % Initialise p and q.
    p = zeros(1, m);
    q = zeros(1, m);

    % Assign initial values p_0, q_0, p_1 and q_1.
    p(1) = a(1);
    p(2) = a(1) * a(2) + 1;
    q(1) = 1;
    q(2) = a(2);
    % Due to the indexing convention, p(i) here is actually p_(i-1).

    % Calculate p_k and q_k
    for i = 3:m
        p(i) = a(i)*p(i-1)+p(i-2);
        q(i) = a(i)*q(i-1)+q(i-2);
    end

    % Find convergents.
    r = p ./ q;
end


r1 =

  Columns 1 through 7

    2.0000    3.0000    2.6667    2.7500    2.7143    2.7188    2.7179

  Columns 8 through 14

    2.7183    2.7183    2.7183    2.7183    2.7183    2.7183    2.7183

  Column 15

    2.7183
```

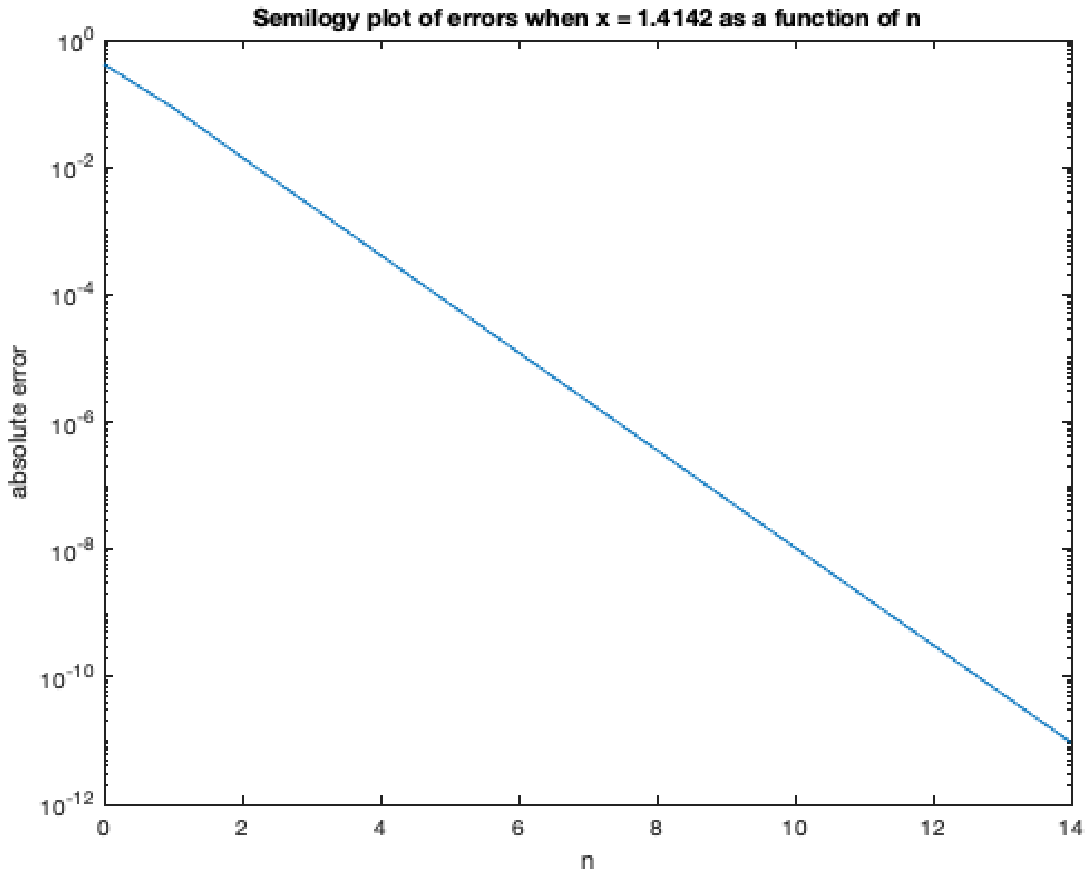
```
r2 =

  Columns 1 through 7

    3.0000    3.1429    3.1415    3.1416    3.1416    3.1416    3.1416

  Columns 8 through 14

    3.1416    3.1416    3.1416    3.1416    3.1416    3.1416    3.1416

  Column 15

    3.1416


r3 =

  Columns 1 through 7

    1.0000    2.0000    1.5000    1.6667    1.6000    1.6250    1.6154

  Columns 8 through 14

    1.6190    1.6176    1.6182    1.6180    1.6181    1.6180    1.6180

  Column 15

    1.6180


r4 =

  Columns 1 through 7

    1.0000    1.5000    1.4000    1.4167    1.4138    1.4143    1.4142

  Columns 8 through 14

    1.4142    1.4142    1.4142    1.4142    1.4142    1.4142    1.4142

  Column 15

    1.4142


r5 =

    3.0000    3.0588    3.0577    3.0578    3.0578    3.0578


% This function takes in a real number x and outputs a semilogy plot of
% absolute errors (between x and the convergent) against n.
```

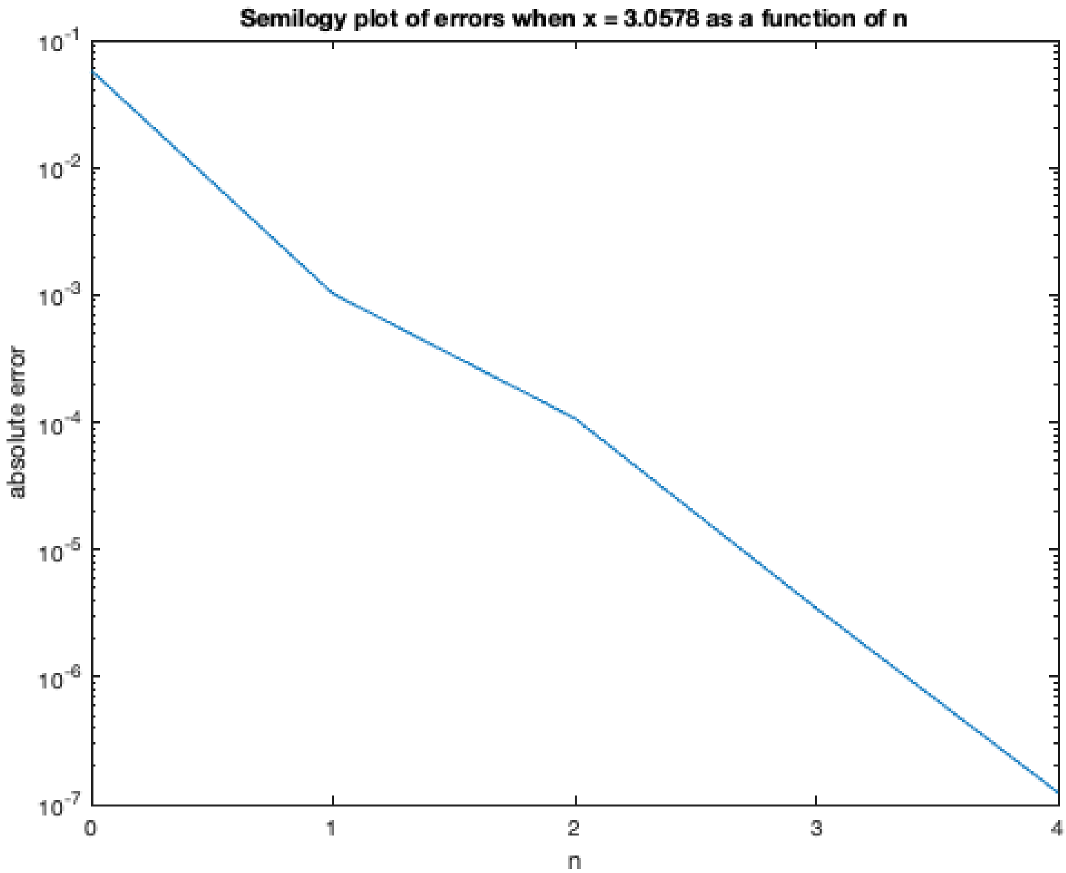
```
function errors_plot_14(x)

    % Compute the continued fraction of x up to a_14.
    % Find the convergent for each n = 0,1,...14
    a = get_cont_frac(sym(x),14);
    [~,~,r] = convergents(a);

    % I want to start from n = 0.
    n = 0:length(a)-1;

    % Find aboslute errors between the convergent and x for each n.
    y = abs(sym(x)-r);

    % Then make a semilogy plot of absolute errors agains n.
    semilogy(n,y)

    % I also want my horizontal axis to display integer values only.
    curtick = get(gca, 'xTick');
    xticks(unique(round(curtick)));

    title(['Semilogy plot of errors when x = ', num2str(x), ' as a function of
 n'])
    xlabel('n')
    ylabel('absolute error')

end
```


Semilogy plot of errors when x = 2.7183 as a function of n
absolute error
n
10^0
10^-2
10^-4
10^-6
10^-8
10^-10
10^-12
0
2
4
6
8
10
12
14


Semilogy plot of errors when x = 3.1416 as a function of n
absolute error
n
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
0
1
2
3
4
5
6
7
8
9
10


Semilogy plot of errors when x = 1.618 as a function of n
absolute error
n
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
2
4
6
8
10
12
14


Semilogy plot of errors when x = 1.4142 as a function of n
absolute error
n
10^0
10^-2
10^-4
10^-6
10^-8
10^-10
10^-12
0
2
4
6
8
10
12
14



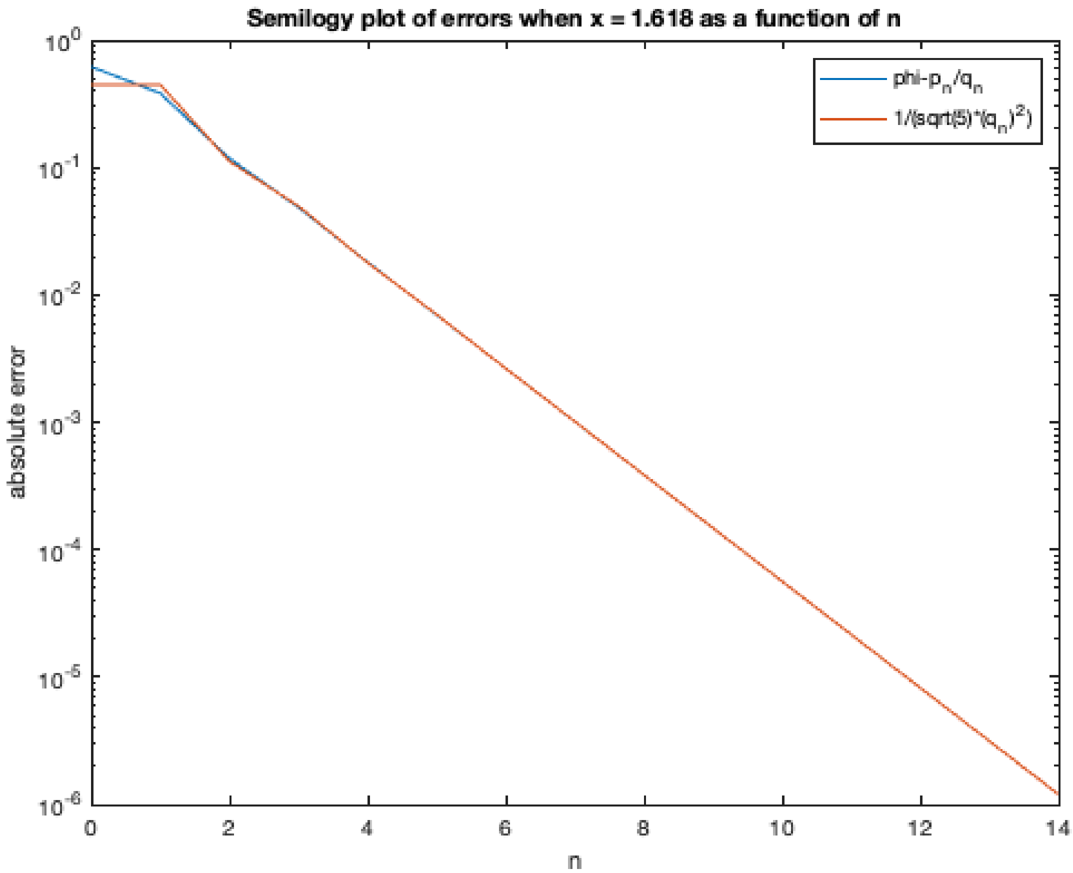
Exercise C2 last part, alternatively:

```
type errors_plot.m

% The rationale behind is 'I dont know how big my n should be'. Instead of
% using n = 14, this function would plot semilogy of absolute errors
% against n up to the absolute error is less than or equal to  1e-12.


% This function is similar to errors_plot_14. They only differ from the
% value of n.
function y = errors_plot(x)

    error = Inf;
    n = 1;
    y = [];

    % A while loop is used to make sure error <= 1e-12.
    while error > 1e-12
        a = get_cont_frac(sym(x),n);
        [~,~,r] = convergents(a);
        error = abs(r(end)- sym(x));
        y(end+1) = error;
```

```
        n = n + 1;
    end

    z = linspace(0,length(y)-1,length(y));
    semilogy(z,y)
    curtick = get(gca, 'xTick');
    xticks(unique(round(curtick)));
    title(['Semilogy plot of errors when x = ', num2str(x), ' as a function of
 n'])
    xlabel('n')
    ylabel('absolute error')

end
```

Exercise C3

```
% Calculate a new matrix with its entries equal to 1/(sqrt(5)*(q_n)^2).
new_q3 = sym(1 ./ ((q3 .^ 2) .* sqrt(5)));

figure(3);
hold on
% Again, I want to start from n = 0.
n = 0:14;
semilogy(n,new_q3)
legend('phi-p_{n}/q_{n}','1/(sqrt(5)*(q_{n})^2)')

% Hence we can see from the graph that the two curves almost overlap,
% which means the convergents of phi are almost exactly 1/(sqrt(5)*(q_n)^2)
```



Exercise C4

```
a6 = get_cont_frac(sqrt(19),14)
% The sequence of repeated numbers are: [2,1,3,1,2,8].
% a_0 = 4 and the last number in the repeated sequence is indeed 2*4=8.
```

```
a6 =

[4, 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2, 8, 2, 1]
```

Exercise C5

```
type pell_solver.m

% Calculate solutions for D = 19 and D = 17.
[x_19,y_19] = pell_solver(10,19)
[x_17,y_17] = pell_solver(10,17)
```

```
% The function takes in two values, l, the maximum m that I want to try,
```

```
% and D, the integer in Pell's Equation.
% It then outputs a tuple (x,y) that is a solution to Pell's equation
% x^2 - Dy^2 = 1.

% Note that two variables are used instead of one, because we can't make
% sure the pattern repeats within any length... If the initial guess of l
% does not work, NaN will be the output, which means we have to try a larger
% l.

function [x,y] = pell_solver(l,D)

    % First, find the array of coefficients
    a = get_cont_frac(sym(sqrt(D)),l);

    % Find the a_m in a that is equal to 2*a_0.
    for i = 2:l+1
        if a(i) == 2 * a(1)
            m = i - 1;
            break
        end
        m = NaN; % If you found m = NaN, then a larger l is needed.
    end

    % Find the convergents, arrays of p_n and q_n.
    [p,q,~] = convergents(a);

    % If m is even, x = p_{m-1} = the mth element in the array of p, and
    % y = q_{m-1} = the mth element in the array of q.
    if mod(m,2) == 0
        x = p(m);
        y = q(m);
    else
    % If m is odd, rewrite m as 2*m the values of p and q follows.
        m = 2*m;
        x = p(m);
        y = q(m);
    end

end


x_19 =

   170


y_19 =

    39


x_17 =

    33
```

```
y_17 =

     8
```

Published with MATLAB® R2022b