1. Runge-Kutta algorithm in MATLAB to simulate the free system

```
function fitzhugh_nagumo_no_impulse
clc

ti = 0; % Initial time of simulation, in msec
tf = input('Enter final time of simulation in msec: ');
n = input('Enter number of forward steps in time to take: ');
h = (tf - ti)/n; % Time step size, in msec
xi = -2 + (2 - (-2))*rand(1,50);
% Random initial conditions for x ranging between -2 and 2
yi = -1 + (1 - (-1))*rand(1,50);
% Random initial conditions for y ranging between -0.05 and 0.05
x = zeros(n,length(xi)); % Initialize vector of values of x
y = zeros(n,length(yi)); % Initialize vector of values of y
t = zeros(n,1); % initialize vector of values of time t
x(1,:) = xi; % Initial values of x
y(1,:) = yi; % Initial values of y
t(1) = ti; % Initial value of t
a = input('Enter value of a: ');
b = input('Enter value of b: ');

for i = 1:(length(xi) - 1)
    for j = 1:(n - 1)

    % Compute the first through fourth RK4 coefficients for x and y
        k1x = coefficientx(x(j,i), y(j,i), h);
        k1y = coefficienty(x(j,i), y(j,i), h, a, b);
        k2x = coefficientx(x(j,i) + k1x/2, y(j,i), h);
        k2y = coefficienty(x(j,i), y(j,i) + k1y/2, h, a, b);
        k3x = coefficientx(x(j,i) + k2x/2, y(j,i), h);
        k3y = coefficienty(x(j,i), y(j,i) + k2y/2, h, a, b);
        k4x = coefficientx(x(j,i) + k3x, y(j,i), h);
        k4y = coefficienty(x(j,i), y(j,i) + k3y, h, a, b);

        % Compute the next values of x, y, and t
        x(j+1,i) = x(j,i) + (k1x + (2*k2x) + (2*k3x) + k4x)./6;
        y(j+1,i) = y(j,i) + (k1y + (2*k2y) + (2*k3y) + k4y)./6;
        t(j+1) = t(j) + h;
    end
end

subplot(3,1,1), plot(t,x), xlabel('Time (msec)'), ylabel('x'), ylim([-3 3])
subplot(3,1,2), plot(t,y), xlabel('Time (msec)'), ylabel('y'), ylim([-1 3])
subplot(3,1,3), plot(x,y), xlabel('x'), ylabel('y'), xlim([-3 3]), ylim([-1 3])

% Function to calculate RK4 coefficients for x
```

```
function result = coefficientx(x,y,h)
result = h.*(y + x - (x.^3)./3);

% Function to calculate RK4 coefficients for y
function result = coefficienty(x,y,h,a,b)
result = h.*(-x + a - b.*y);
```

2. Runge-Kutta algorithm in MATLAB to simulate the square wave-forced system

```
function fitzhugh_nagumo_square_wave
clc

ti = 0; % Initial time of simulation, in msec
tf = input('Enter final time of simulation in msec: ');
n = input('Enter number of forward steps in time to take: ');
h = (tf - ti)/n; % Time step size, in msec
% Random initial conditions for x ranging between -2 and 2
xi = -2 + (2 - (-2))*rand(1,10);
% Random initial conditions for y ranging between -0.05 and 0.05
yi = -1 + (1 - (-1))*rand(1,10);
x = zeros(n,length(xi)); % Initialize vector of values of x
y = zeros(n,length(yi)); % Initialize vector of values of y
t = zeros(n,1); % initialize vector of values of time t
x(1,:) = xi; % Initial values of x
y(1,:) = yi; % Initial values of y
t(1) = ti; % Initial value of t
a = input('Enter value of a: ');
b = input('Enter value of b: ');
A = input('Enter the amplitude of the square wave in mV: ');
T = input('Enter half of the period of the square wave in msec: ');
k = input('Enter number of terms in the series: ');

for i = 1:(length(xi) - 1)
    for j = 1:(n - 1)

    % Compute the first through fourth RK4 coefficients for x and y
        k1x = coefficientx(x(j,i), y(j,i), h) +
        h.*squarewave(A, T, k, t(j) + h/2);
        k1y = coefficienty(x(j,i), y(j,i), h, a, b);
        k2x = coefficientx(x(j,i) + k1x/2, y(j,i), h) +
        h.*squarewave(A, T, k, t(j) + h/2);
        k2y = coefficienty(x(j,i), y(j,i) + k1y/2, h, a, b);
        k3x = coefficientx(x(j,i) + k2x/2, y(j,i), h) +
        h.*squarewave(A, T, k, t(j) + h/2);
        k3y = coefficienty(x(j,i), y(j,i) + k2y/2, h, a, b);
        k4x = coefficientx(x(j,i) + k3x, y(j,i), h) +
        h.*squarewave(A, T, k, t(j) + h);
        k4y = coefficienty(x(j,i), y(j,i) + k3y, h, a, b);

        % Compute the next values of x, y, and t
        x(j+1,i) = x(j,i) + (k1x + (2*k2x) + (2*k3x) + k4x)./6;
        y(j+1,i) = y(j,i) + (k1y + (2*k2y) + (2*k3y) + k4y)./6;
        t(j+1) = t(j) + h;
```

```
    end
end

subplot(3,1,1), plot(t,x), xlabel('Time (msec)'), ylabel('x')
subplot(3,1,2), plot(t,y), xlabel('Time (msec)'), ylabel('y')
subplot(3,1,3), plot(x,y), xlabel('x'), ylabel('y')

% Function to calculate RK4 coefficients for x
function result = coefficientx(x,y,h)
result = h.*(y + x - (x.^3)./3);

% Function to calculate RK4 coefficients for y
function result = coefficienty(x,y,h,a,b)
result = h.*(-x + a - b.*y);

% Square wave expressed as a Fourier sine series
function result = squarewave(A,T,k,t)
x = 0;
for i = 1:1:k
    x = x + ((4.*A)./pi).*(1./(2.*i - 1)).*sin((2.*i - 1).*pi.*t./T);
end
result = x;
```

3. Runge-Kutta algorithm in MATLAB to simulate the constant- and cosine wave-forced systems

```
function fitzhugh_nagumo_nonzero_impulse
clc

ti = 0; % Initial time of simulation, in msec
tf = input('Enter final time of simulation in msec: ');
n = input('Enter number of forward steps in time to take: ');
h = (tf - ti)/n; % Time step size, in msec
% Random initial conditions for x ranging between -2 and 2
xi = -2 + (2 - (-2))*rand(1,10);
% Random initial conditions for y ranging between -0.05 and 0.05
yi = -1 + (1 - (-1))*rand(1,10);
x = zeros(n,length(xi)); % Initialize vector of values of x
y = zeros(n,length(yi)); % Initialize vector of values of y
t = zeros(n,1); % initialize vector of values of time t
x(1,:) = xi; % Initial values of x
y(1,:) = yi; % Initial values of y
t(1) = ti; % Initial value of t
a = input('Enter value of a: ');
b = input('Enter value of b: ');
A = input('Enter the amplitude of the square wave in mV: ');
A = input('Enter amplitude of cosine wave in mV: ');
w = input('Enter frequency of cosine wave in 1/msec: ');

for i = 1:(length(xi) - 1)
    for j = 1:(n - 1)

    % Compute the first through fourth RK4 coefficients for x and y
        k1x = coefficientx(x(j,i), y(j,i), h, A, w, t(j) + h/2);
        k1y = coefficienty(x(j,i), y(j,i), h, a, b);
        k2x = coefficientx(x(j,i) + k1x/2, y(j,i), h, A, w, t(j) + h/2);
        k2y = coefficienty(x(j,i), y(j,i) + k1y/2, h, a, b);
        k3x = coefficientx(x(j,i) + k2x/2, y(j,i), h, A, w, t(j) + h/2);
        k3y = coefficienty(x(j,i), y(j,i) + k2y/2, h, a, b);
        k4x = coefficientx(x(j,i) + k3x, y(j,i), h, A, w, t(j) + h);
        k4y = coefficienty(x(j,i), y(j,i) + k3y, h, a, b);

        % Compute the next values of x, y, and t
        x(j+1,i) = x(j,i) + (k1x + (2*k2x) + (2*k3x) + k4x)./6;
        y(j+1,i) = y(j,i) + (k1y + (2*k2y) + (2*k3y) + k4y)./6;
        t(j+1) = t(j) + h;
    end
end

subplot(3,1,1), plot(t,x), xlabel('Time (msec)'), ylabel('x')
```

```
subplot(3,1,2), plot(t,y), xlabel('Time (msec)'), ylabel('y')
subplot(3,1,3), plot(x,y), xlabel('x'), ylabel('y')

% Function to calculate RK4 coefficients for x
function result = coefficientx(x,y,h,A,w,t)
result = h.*(y + x - (x.^3)./3 + A.*(cos(w.*t)));

% Function to calculate RK4 coefficients for y
function result = coefficienty(x,y,h,a,b)
result = h.*(-x + a - b.*y);
```