1.1
(a) `3.1416`
(b) `0.5000`
(c) `3.0000`
(d) error, it should be typed `cos(2*pi/3)`.
(e) `10`
(f) `14`
(g) `24`
(h) `4`
(i) `1`
(j) `162`
(k) `32`
(l) `4096`
(m) `3.1416`
(n) `5100`
(o) `2.2204e-16`
(p) `0`
(q) `1.9999`
(r) `3`
(s) `0 + 3.0000i`
(t) `-1`
(u) `-8.0000 + 6.0000i`
(v) `-1.0000 + 0.0000i`
(w) `4.6052` This is the natural log. Try `exp(4.6052)` to be sure.
(x) `2`
(y) `c = 300000000`
(z) `100000000`, notice that you can define your own constants.

1.2 In the end we have $x = 0.7391$ which is the value of x for which $x = \cos x$.

1.3 The six expressions could be

```
x>1/2 && x<1                  x<1 && y<1                x>1 || y>1
y>x && y<2x              y>x && y>-2x && y<1             x^2+y^2<1
```

1.4
(a)
```
    >> t = -2:0.01:2;
    >> y = 1./(1+t.^2);
    >> plot(t,y)
```
(b)
```
    >> theta = 0:0.01:(3*pi/2);
    >> x = 2*cos(theta);
    >> y = 2*sin(theta);
    >> plot(x,y);
```
(c)
```
    >> theta = 0:0.01:(2*pi);
    >> x = 2*cos(theta);
    >> y = 3*sin(theta);
    >> plot(x,y);
```
(d)
```
    >> theta = 0:0.01:(20*pi);
    >> x = theta.*cos(theta);
    >> y = theta.*sin(theta);
    >> plot(x,y);
```

1.5 Recall that for freefall

$$x(t) = x_0 + v_{x0}t = 0 + (30\tfrac{\mathrm{m}}{\mathrm{s}})\cos(55^\circ)t = (30\tfrac{\mathrm{m}}{\mathrm{s}})\cos(55^\circ)t$$
$$y(t) = y_0 + v_{y0}t - \tfrac{1}{2}gt^2 = (10\mathrm{m}) + (30\tfrac{\mathrm{m}}{\mathrm{s}})\sin(55^\circ)t - \tfrac{1}{2}(9.8\tfrac{\mathrm{m}}{\mathrm{s}^2})t^2$$

So we can compute the trajectory as follows.

```
    >> t = 0:0.01:6
    >> x = 30*cos(55*pi/180)*t;
    >> y = 10+30*sin(55*pi/180)*t - (9.8/2)*t.^2;
    >> plot(x,y)
```

Then we can zoom in on the plot to find that y is zero at about $x = 92.809$m.

1.6 The center of the first interval is at $\frac{dx}{2}$ since the left edge is at 0 and the right edge is at dx. The center of the next interval is just dx more than the center of the first. Likewise for the rest. So we can get the centers via the expression. `x = (dx/2):dx:1`. The rest of the calculation is the same, and the error is in the end 0.011%.

1.7

1.8 First we note that the acceleration is $\frac{dv}{dt} = a = \frac{F}{m}$ and that

$$\Delta v = \int_{t_i}^{t_f} \frac{dv}{dt}\,dt = \int_{7\mathrm{s}}^{13\mathrm{s}} \frac{(20\mathrm{N})\ln(t/(6\mathrm{s}))}{(10\mathrm{kg})}dt = \int_{7\mathrm{s}}^{13\mathrm{s}} (2\tfrac{\mathrm{m}}{\mathrm{s}^2})\ln(t/(6\mathrm{s}))dt$$

```
    >> dt = 0.00001;
    >> t = 7:dx:13;
    >> a = 2*log(t/6);
    >> dv = sum(a*dt)
    dv5.9448 =
                  So
```

the change in velocity is about $5.94\frac{\mathrm{m}}{\mathrm{s}}$.

1.9 First we need to define our currents. Let I_1 be the current downward through the 8Ω resistor. Let I_2 be the current to upward through the 5Ω resistor. Let i_3 be the current to the left through the 3Ω resistor. Kirchhoff's junction rule at the upper node is

$$0 = I_1 - I_2 - I_3$$
$$0 = 1I_1 - 1I_2 - 1I_3$$

while the Kirchhoff's loop rule going clockwise around the left side "window" is

$$(8\Omega)I_1 + (5\Omega)I_2 + (1.0\Omega)I_2 - 4\text{V} = 0$$

or

$$(4\text{A}) = 8I_1 + 6I_2 + 0I_3$$

while the Kirchhoff's loop rule going clockwise around the outside loop is

$$(8\Omega)I_1 + (3\Omega)I_3 + (1.0\Omega)I_3 - (12\text{V}) = 0$$
$$(12\text{A}) = 8I_1 + 0I_2 + 4I_3$$

bringing the three Kirchhoff's equations together we find

$$0 = 1I_1 - 1I_2 - 1I_3$$
$$4 = 8I_1 + 6I_2 + 0I_3$$
$$12 = 8I_1 + 0I_2 + 4I_3$$

or

$$\begin{bmatrix} 0 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ 8 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

So we can use find the currents by

```
    >> M = [1,-1,-1; 8,6,0; 8,0,4];
    >> >> V = [0;4;12];
    >> inv(M)*V
    ans =
          0.8462
       -0.4615
        1.3077
```

2.1

```
function [A, V] = canSize(r,h)
    A_top = pi*r^2;
    V = A_top * h;
    A = 2*pi*r*h + 2*A_top;
end
```

2.2

```
function [xp, xm] = quadratic(a,b,c)
% Computes the two solutions to the
% quadratic equation ax^2 + bx + c = 0
    xp = (-b+sqrt(b^2-4*a*c))/(2*a);
    xm = (-b-sqrt(b^2-4*a*c))/(2*a);
end
```

2.3

```
function y = cosP(x)
    y = cos(x);
    if y<0
        y = 0;
    end
end
```

2.4 The program will print out 1, 2, 3, 4 and then return the value 4.

2.5

```
function x = solvCos(threshold)
    x0 = 1;
    x = cos(x0);
    while abs(x-x0)>threshold
        x0 = x;
        x = cos(x0);
    end
end
```

2.6

```
function x = plotSolvCos(N)
    x = NaN(1,N);
    x(1) = 1;
    for n=2:N
        x(n) = cos(x(n-1));
    end
    figure(1)
    plot(x,'o')
    figure(2)
    semilogy(abs(x-x(N)),'o')
end
```

2.7

```
function T = waterHeater(d,L,alpha,Te,T_int,t_f)
% d     Wall thickness in centemeters.
% L     Length of one side of cube.
% alpha Ratio of wall that is glass
% Te    Temperature of environment
% T0    Initial temperature of the water
% t_f   The total time of the simulation in days.
    Ug = 2.25;      % U-factor of glass
    Uw = 0.66*5/d;  % U-factor of walls
    c = 4186;       % specific heat of water
    rho = 1000;     % density of water
    m = rho*L^3;    % mass of water
    mc = m*c;       % heat capacity of water
    Ag = alpha*L^2;   % area of glass.
    Aw = 6*L^2 - Ag;  % area of non-glass walls.
    phi = 1000;     % intensity of solar radiation
    day = 60*60*24; % one day in seconds
    dt = 600;       % time step in seconds (ten mintues)
    t = 0:dt:(t_f*day);
    Nt = length(t); % the number of time steps to simulation
    T = NaN(1,Nt);  % prealocate memory for temperatures
    T(1) = T_int;   % Assign starting temperature.

    for n = 1:Nt-1  % a loop over time.
        theta = 2*pi*t(n)/day;  % the angle of the sun.
        Pin = phi*Ag*cosP(theta);
        Pout = (Aw*Uw + Ag*Ug)*(T(n)-Te);
        f=(Pin-Pout)/mc;
        T(n+1) = T(n) + f*dt; % update the temperature
    end

    plot(t/day,T);  % plot the results.
    xlabel('time [days]')
    ylabel('Temperature [C]');
end

function y = cosP(x)
    y = max(0,cos(x));
end
```

(a) The temperature swing is 3.6220 when the walls are 5cm. With a thickness of 10 cm it is effectively the same 3.6221. So changing the thickness does not change the temperature fluctuation.
(b) If we cut the window to 1/2 the to area the the fluctuation reduces to 1.8110.

2.8

```
function T = solarHouse(Te,T_init,t_f,d,Ae,At,Aw)
% Te        Temperature of environment.             [C]
% T_init    Initial temperature of the water.       [C]
% t_f       The total time of the simulation.       [days]
% d         Slab thickness.                         [m]
% Ae        Area of glass on east side of building. [m^3]
% At        Area of glass on top  side of building. [m^3]
% Aw        Area of glass on west side of building. [m^3]
    Ug = 2.25;          % U-factor of glass.        [W/m^2*C]
    Uw = 0.33;          % U-factor of walls.        [W/m^2*C]
    c = 750;            % specific heat of water    [J/kg*C]
    m = 2400*10*10*d;   % mass of concrete          [kg]
    mc = m*c;           % heat capacity of slab     [J/C]
    A = 10*10 + 4*10*3; % exposed surface area.     [m^2]
    Ag = Ae + At + Aw;  % area of glass.            [m^2]
    An = A - Ag;        % area of non-glass surface.[m^2]
    phi = 1000;         % intensity of radiation.   [W/m^2]
    day = 60*60*24;     % one day in seconds.       [sec]
    dt = 600;           % time step in seconds.     [sec]
    t = 0:dt:(t_f*day); % the time vector.          [sec]
    Nt = length(t);     % the number of time steps to simulation.
    T = NaN(1,Nt);      % prealocate memory for temperatures.
    T(1) = T_init;      % Assign starting temperature.

    for n = 1:Nt-1                          % a loop over time.
        theta = 2*pi*t(n)/day;              % the angle of the sun.
        if cos(theta)>0                     % check to see if it is daytime.
            Pin =       phi*At*cosP(theta);
            Pin = Pin + phi*Ae*cosP(theta+pi/2);
            Pin = Pin + phi*Aw*cosP(theta-pi/2);
        else                                % at night there is no solar gain.
            Pin = 0;
        end
        Pout = (An*Uw+Ag*Ug)*(T(n)-Te);  % compute the conductive loss.
        f=(Pin-Pout)/mc;
        T(n+1) = T(n) + f*dt;               % update the temperature.
    end

    plot(t/day,T);                          % plot the results.
    xlabel('time [days]')
    ylabel('Temperature [C]');
end

function y = cosP(x)
    y = max(0,cos(x));
end
```

2.9 The error after 2 seconds is 980mm, 98mm, 9.8mm, 0.98mm for $dt = 0.1$, 0.01, 0.001, 0.0001.

```
function error_final = ch2pr9(y0,v0,t_final,dt)
t = 0:dt:t_final;
N = length(t);
y = NaN(1,N);
v = NaN(1,N);
y(1) = y0;
v(1) = v0;
a = -9.8;
for n = 1:(N-1)
    y(n+1) = y(n) + v(n)*dt;
    v(n+1) = v(n) + a*dt;
end
y_theory = y0 + v0*t +0.5*a*t.^2;
figure(1)
plot(t,y,t,y_theory);
xlabel('time [s]')
ylabel('height [m]')

figure(2)
err = y-y_theory;
plot(t,err,'o');
title('error')
xlabel('time [s]')
ylabel('error [m]')

error_final = err(N);
end
```

2.10

```
function y = ch2pr10(A,t_final,dt)
t = 0:dt:t_final;
N = length(t);
y = NaN(1,N);
v = NaN(1,N);
y(1) = A;
v(1) = 0;
for n = 1:(N-1)
    a = -y(n)^5;
%     a = -sqrt(abs(y(n)))*sign(y(n));
    y(n+1) = y(n) + v(n)*dt;
    v(n+1) = v(n) + a*dt;
end
figure(1)
plot(t,y);
xlabel('time [s]')
ylabel('height [m]')

end
```

(a) Zooming in on the graph at the second peak we see that the period is about 8.4 seconds.
(b) Zooming in on the graph at the second peak we see that the period is about 2.9 seconds. Much shorter than the period in the previous part.

2.11 To Be Done

2.14 The integer part can be represented just fine in any basis. So let x be the fractional part of the expansion. Suppose that $-N$ is the smallest power needed in the expansion of x so that

$$x = \sum_{n=-N}^{-1} a_n 2^n = \sum_{1}^{N} a_n \frac{1}{2^n}$$

Now notice that $2 = \frac{10}{5}$ So that

$$x = \sum_{n=1}^{N} a_n \frac{5^n}{10^n} = \sum_{n=1}^{N} a_n 5^n \frac{1}{10^n}$$

So while this is not a proper expansion since the numbers $a_n 5^n$ are not necessarily between 0 and 9, in the cases where $a_n 5^n > 9$ the extra could be pushed up to a smaller place n. So for example

$$0.000001_2 = \frac{1}{2^6} = \frac{5^6}{10^6} = \frac{15625}{10^6} = 0.015625_{10}$$

3.1 The code is written in terms of two sub-functions that are called by the main function `rk2HO`. First the main.

```
function rk2HO(Ne,Nr)
    omega = 2*pi;
    [xe,~,te] = rkHO(Ne,omega,1);
    [xr,~,tr] = rkHO(Nr,omega,2);
    figure(1)
    plot(te,xe,tr,xr);
    legend('euler','RK2','location','northwest');
    graphToPDF('RK2.pdf',4,3)
    figure(2)
    plot(te,xe-cos(omega*te),tr,xr-cos(omega*tr));
    title('error versus time');
    legend('euler','RK2','location','northwest');
    graphToPDF('errorRK2.pdf',4,3)
end
```

Then the function called by the main function.

```
function [x,v,t] = rkHO(Npp,omega,order)
% Npp is number of time steps per period.
% omega is the angular fequency of the oscillation. [radians per second]
% order is 1 for Euler and 2 or 4 for Runge-Kutta second or fourth order.
period = 2*pi/omega;    % calculate the period of oscillation
dt = period/Npp;        % calculate time step
N = 10*Npp+1;           % run ten periods.
t = NaN(1,N);           % Declare arrays.
x = NaN(1,N);
v = NaN(1,N);
t(1) = 0;               % initial time of zero;
x(1) = 1;               % initial displacement of 1 unit.
v(1) = 0;               % initial velocity of zero.

for n = 1:N-1   % loop over time steps
    [x(n+1),v(n+1),t(n+1)] = rkStep(x(n),v(n),t(n),dt,omega,@accHO,order);
end

end
```

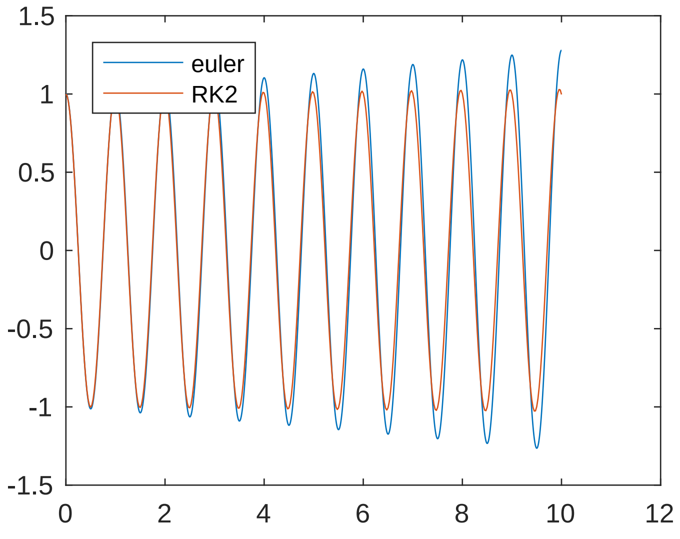
Then another function that is called by `rkHO`.

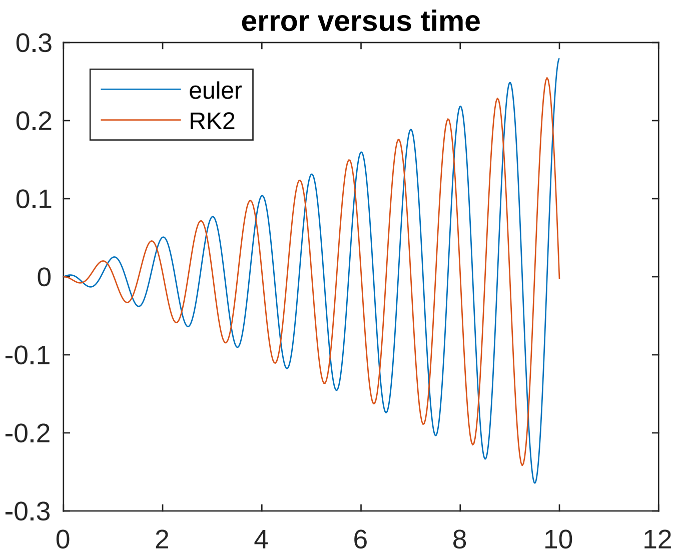
```
function [rn,vn,tn] = rkStep(r,v,t,dt,par,accFunc,order)
if order == 1
    rn = r + v*dt;
    vn = v + accFunc(r,v,t,par)*dt;
elseif order == 2
    rmid = r + v*dt/2;
    vmid = v + accFunc(r,v,t,par)*dt/2;
    tmid = t + dt/2;
    rn = r + vmid*dt;
    vn = v + accFunc(rmid,vmid,tmid,par)*dt;
elseif order == 4
    dra = v*dt;
    dva = accFunc(r,v,t,par)*dt;
    drb = (v+dva/2)*dt;
    dvb = accFunc(r+dra/2,v+dva/2,t+dt/2,par)*dt;
    drc = (v+dvb/2)*dt;
    dvc = accFunc(r+drb/2,v+dvb/2,t+dt/2,par)*dt;
    drd = (v+dvc)*dt;
    dvd = accFunc(r+drc,v+dvc,t+dt,par)*dt;
    rn = r + dra/6 + drb/3 + drc/3 + drd/6;
    vn = v + dva/6 + dvb/3 + dvc/3 + dvd/6;
else
    error('That order is not implemented, choose 1,2,or 4.')
end
tn = t + dt;
end
```

and lastly a fourth function called by `rkHO`.

```
function acc = accHO(x,v,t,omega)
 acc = -omega^2*x;
end
```

The results are show below for the case of $Nr = 40$ and $Ne = 800$.



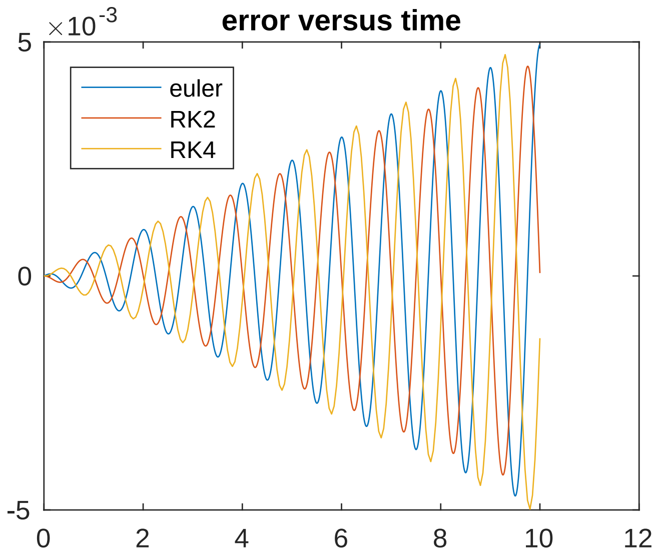


In the first graph it is obvious that the amplitude of the Euler method solution is diverging. What is not obvious is that the phase of the Runge-Kutta is diverging. If we subtract the theoretical solution $x = \cos\omega$ then we can see the actual error. This is the second graph. We see that the to methods have a similar error in this case.

3.2 We use the same sub-functions as in the last problem and a slightly altered main function.

```
function rk4HO(Ne,N2,N4)
    omega = 2*pi;
    [xe,~,te] = rkHO(Ne,omega,1);
    [x2,~,t2] = rkHO(N2,omega,2);
    [x4,~,t4] = rkHO(N4,omega,4);

    figure(1)
    plot(te,xe,t2,x2,t4,x4);
    legend('euler','RK2','RK4','location','northwest');
    graphToPDF('RK4.pdf',4,3)

    figure(2)
    plot(te,xe-cos(omega*te),t2,x2-cos(omega*t2),t4,x4-cos(omega*t4));
    title('error versus time');
    legend('euler','RK2','RK4','location','northwest');
    graphToPDF('errorRK4.pdf',4,3)
end
```

With `Ne=40000` and `Nr2=300` and `Nr4=20` we get the following errors, which are more or less the same magnitude.



If we count one step of the Euler method as one unit of computation, then the Euler method uses 40000 units, while the second order Runge-Kutta uses 2*300=600 units, and the fourth order Runge-Kutta uses 4*20 = 80 units. Thus the the fourth order is singificantly more efficient than the second order and vastly more efficient than Euler's method. So the moral of the story is, don't use Euler's method unless accuracy is not important to you.

3.4 We know that $s = \omega_0 t$ so that $\Delta s = \omega_0 \Delta t$ or $\Delta t = \frac{\Delta s}{\omega_0}$. The period is a time interval so

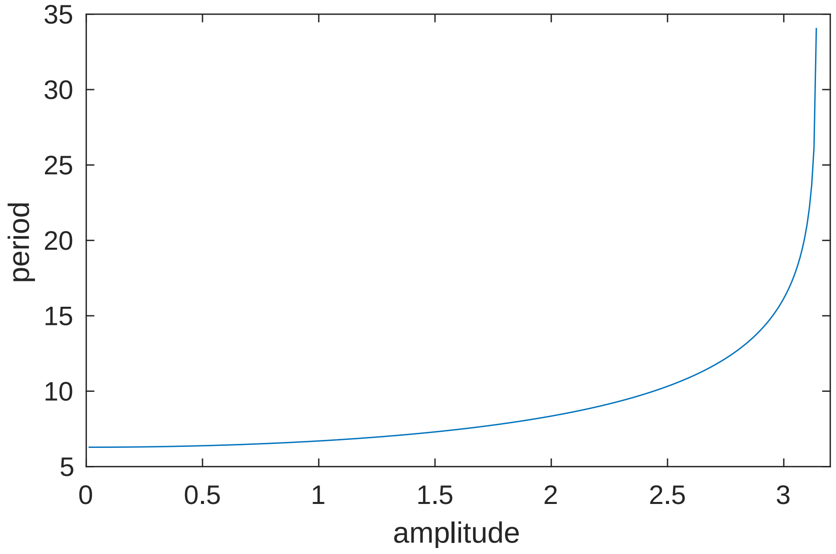
$$T = \frac{2.7}{5.0\ \mathrm{s}^{-1}} = 0.54\mathrm{s}$$

3.5 The following three functions implements the solution.

```
function a = accPP(x,~,~,~)
    a = -sin(x);
end
```

```
function graphPeriod(A,dt,acc,par)
    N = length(A);
    period = NaN(1,N);
    for n = 1:N
        period(n) = findPeriod(A(n),dt,acc,par);
    end
    plot(A,period)
    xlabel('amplitude')
    ylabel('period')
end

function period = findPeriod(A,dt,acc,par)
    x = A;
    v = 0;
    t = 0;
    count = 0;
    tc = NaN(1,2);

    while count<2
        [xn,v,tn] = rkStep(x,v,t,dt,par,acc,4);
        if x>= 0 && xn<0
            count = count+1;
            s = (xn-x)/(tn-t);
            b = x - s*t;
            tc(count) = -b/s;
        end
        x = xn;
        t = tn;
    end
    period = tc(2)-tc(1);
end
```

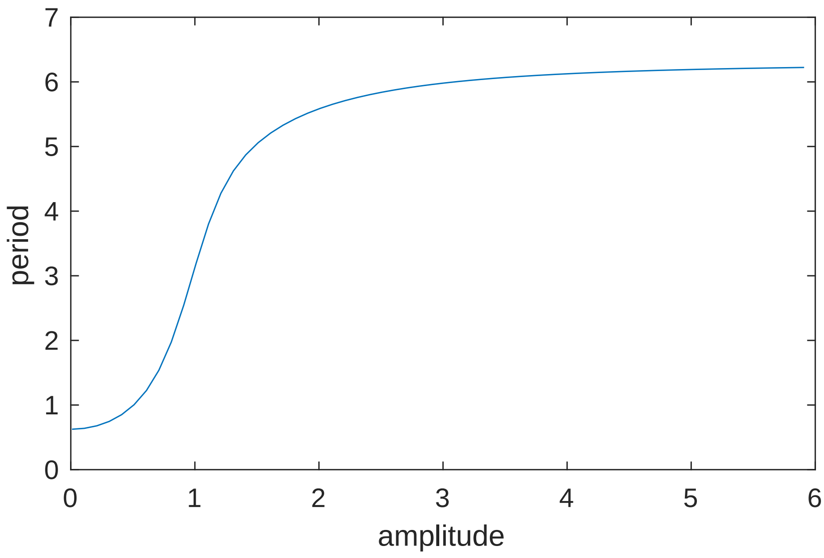
Calling the function as
`graphPeriod(0.01:0.01:pi,0.01,@accPP,NaN)`
we get the following graph.



3.6 With the acceleration function defined as follows we can use the same program as in the last problem.

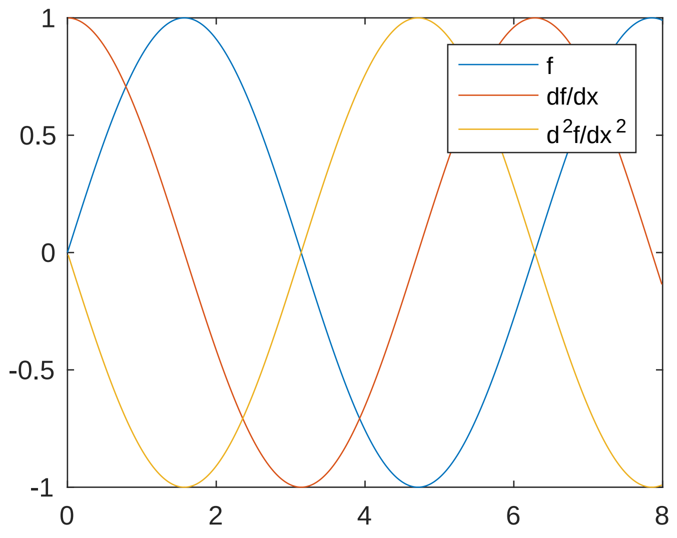
```
function a = accOdd(x,~,~,~)
    a = -x*(1+100*exp(-5*x^2));
end
```

Calling the function as
`graphPeriod(0.01:0.1:6,0.01,@accOdd,NaN)`
we get the following graph.



4.1

```
function [fp,fpp] = graphDer(func,x)
    N = length(x);
    dx = x(2)-x(1);
    f = func(x);
    fp = NaN(1,N);
    fpp = NaN(1,N);
    for n = 2:N-1
        fp(n) = (f(n+1)-f(n-1))/(2*dx);
        fpp(n) = (f(n+1)-2*f(n)+f(n-1))/dx^2;
    end
    plot(x,f,x,fp,x,fpp)
    legend('f','df/dx','d^2f/dx^2')
end
```

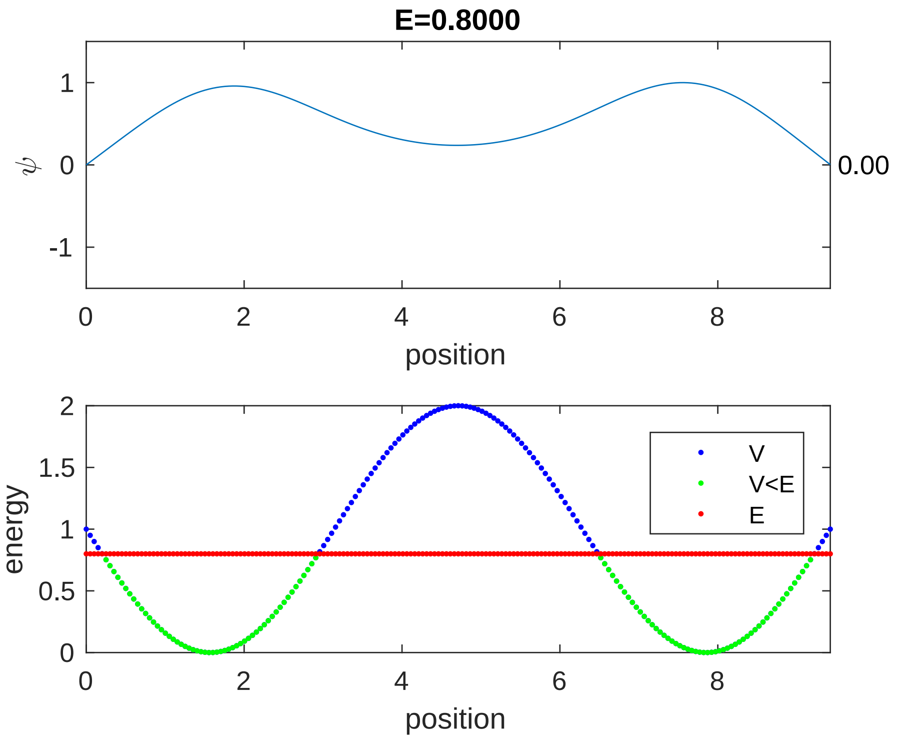


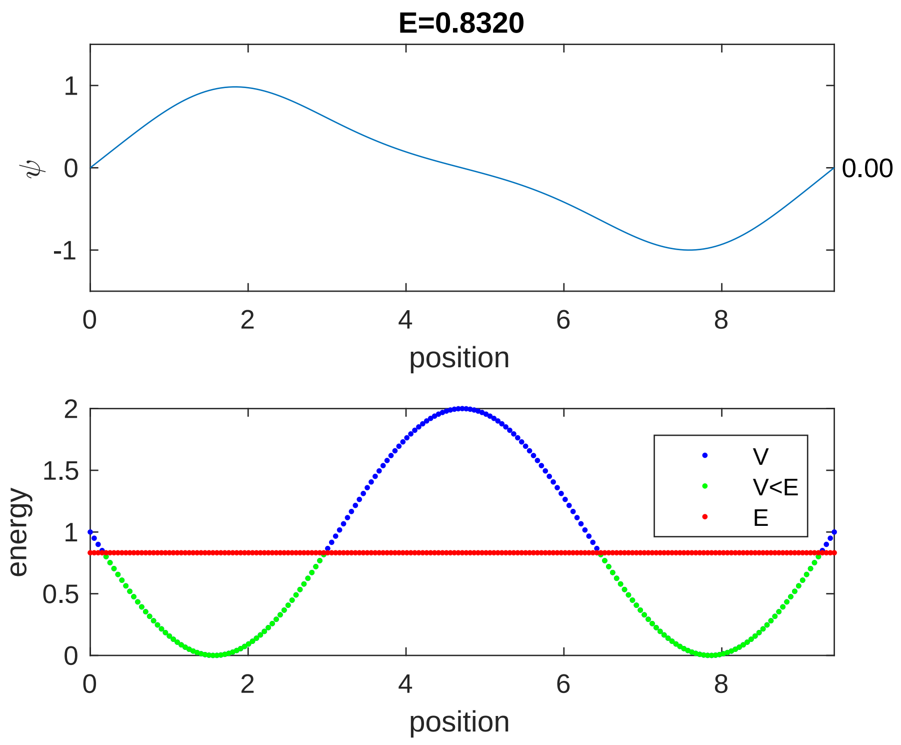
4.2 The code below solved this problem.

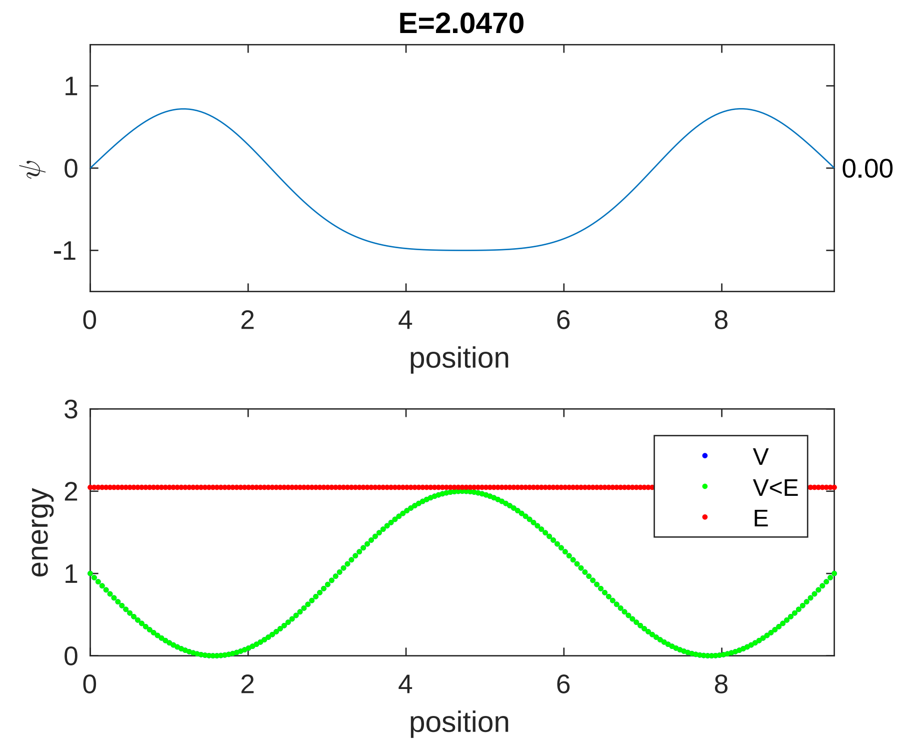
```
function [psi,x] = compPsi(pot,dx,xa,xb,type,E)
    N = round((xb-xa)/dx)+1;    % The number of positions.
    dx = (xb-xa)/(N-1);         % recompute dx
    dx2 = dx^2;
    x = xa+(0:N-1)*dx;
    N = length(x);
    psi = NaN(1,N);
    V = pot(x);

    switch type
        case 'even'
        psi(1) = 1;
        psi(2) = psi(1) + 0.5*(pot(0)-E)*psi(1)*dx2;
        case 'odd'
        psi(1) = 0;
        psi(2) = dx;
        case 'finite'
        psi(1) = 0;
        psi(2) = dx;
        case 'limit'
        psi(1) = dx;
        kappa = sqrt(V(1)-E);
        psi(2) = exp(kappa*dx)*psi(1);
    end

    for n = 2:N-1
        psi(n+1) = 2*psi(n) - psi(n-1) + (V(n)-E)*psi(n)*dx2;
    end

    figure(1)
    ind = find(V<E);                            % classically allowed region
    subplot(2,1,2)
    plot(x,V,'.r',x(ind),V(ind),'.g',x,x*0+E,'.b')
    xlim([xa,xb]);
    legend('V','V<E','E')
    ylabel('energy')
    xlabel('position')

    subplot(2,1,1)
    psi_max = max(abs(psi(ind)));    % find the maximum over classically allowed region
    psi = psi/psi_max;                          % a normalization of sorts
    plot(x,psi)
    xlim([xa,xb]);
    ylim([-1.1,1.1]);
%    ylim([min(psi(ind)),max(psi(ind))]);
    ylabel('\psi')
    xlabel('position')
    kappa = sqrt(V(N)-E);
    theta = kappa*xb;
    phi = kappa*dx;
    A = (psi(N-1)*exp(-phi)-psi(N))/(exp( theta)*(exp(-phi)-exp( phi)));
    B = (psi(N-1)*exp( phi)-psi(N))/(exp(-theta)*(exp( phi)-exp(-phi)));
```

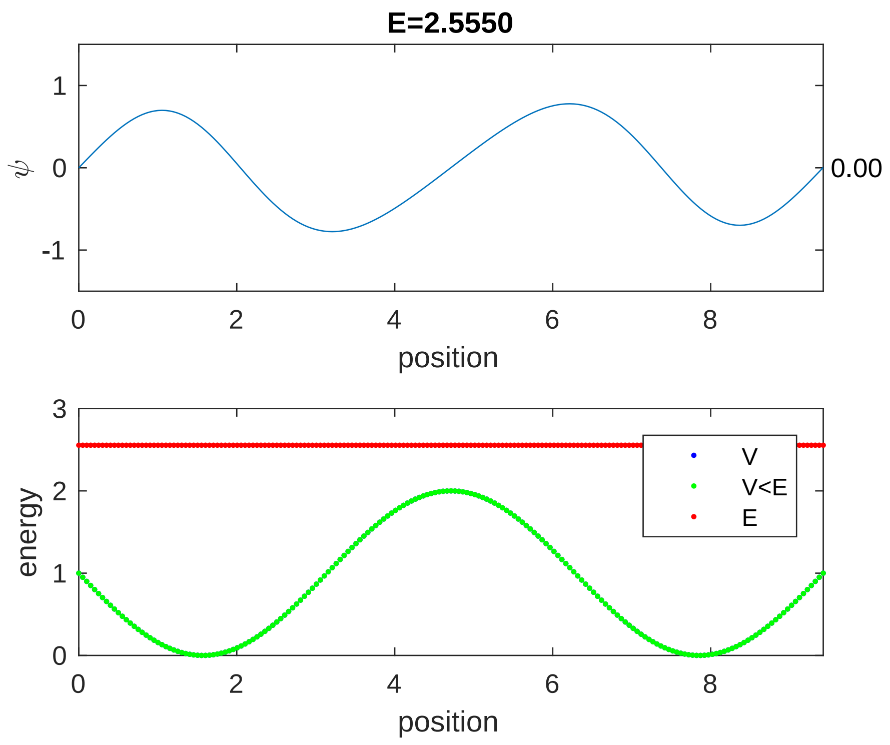
```
        ABmax = max(abs(A),abs(B));
        B = B/ABmax;
        A = A/ABmax;
        title(sprintf('E=%0.7f, A=%0.1e, B=%0.1e',E,A,B))
        text(xb,psi(N),sprintf(' %0.2f\n %0.1e\n %0.1e',psi(N),A,B))


    end
```

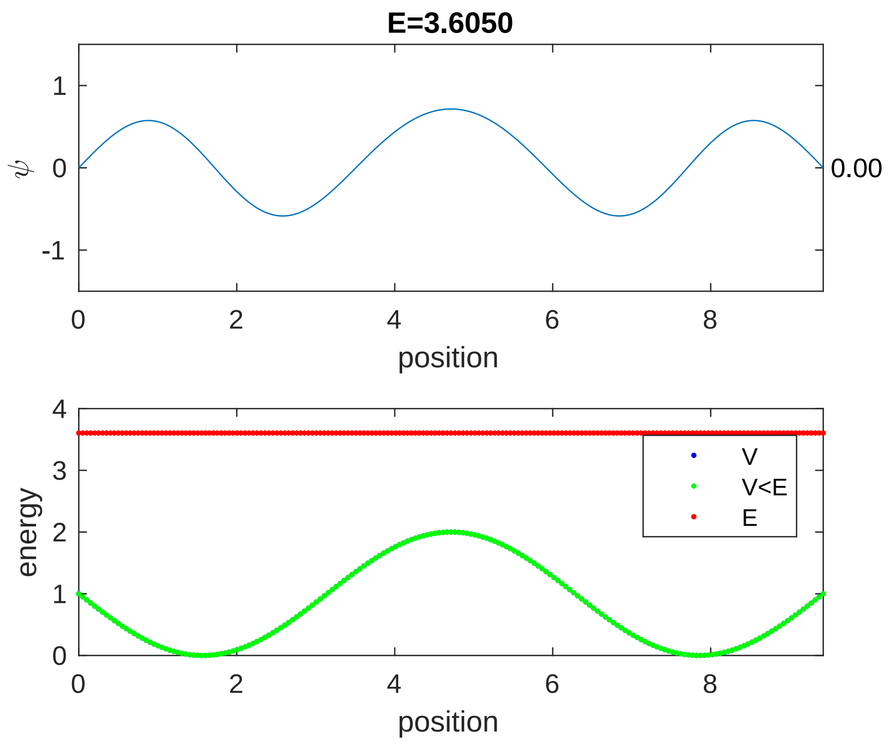
By trial and error the values of E as shown in the graphs below in the graphs were found.




E=0.8320
ψ
position
0.00
energy
position
V
V<E
E


E=2.0470
ψ
position
0.00
energy
position
V
V<E
E

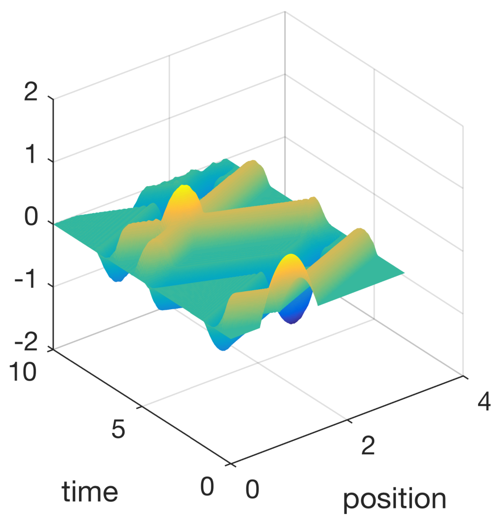

E=2.5550
ψ
0.00
position
energy
V
V<E
E
position

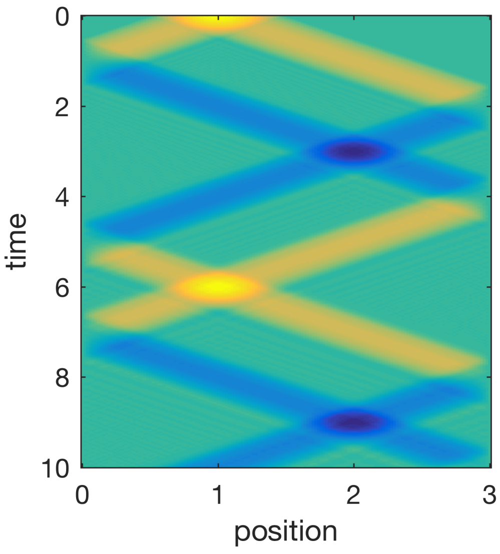

E=3.6050
ψ
0.00
position
energy
V
V<E
E
position

4.3

4.4

4.5 With the code below we can simulate this system.
Calling `waveEq1D(0,3,0.02,0,10,[1,0.5],@intCondParab)` simulates the system for a time $T = 10$, with $dx = 0.02$, $L = 3$, $a = 1$, and $b = 0.5$. It produces the following graph.



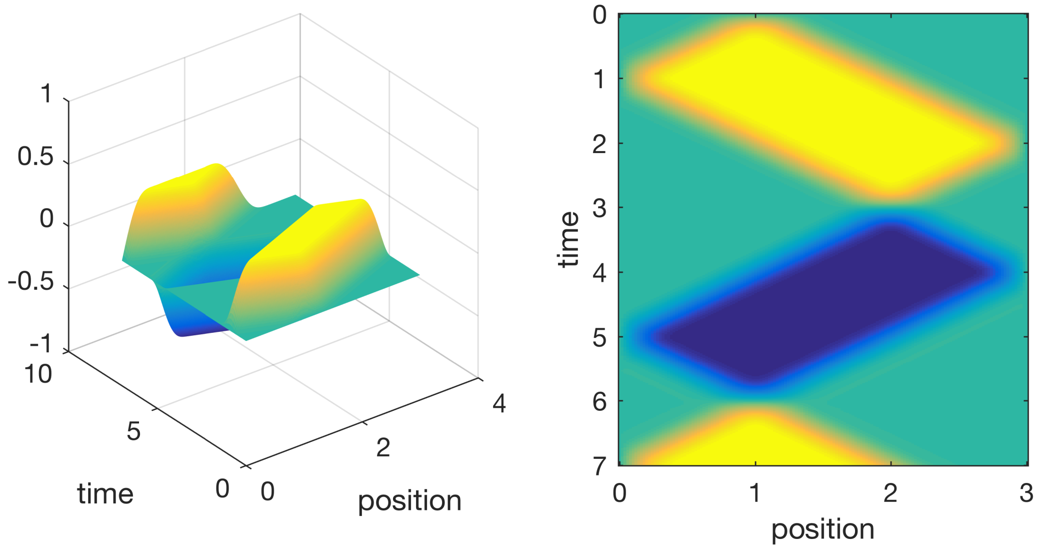


(continued)

```
function waveEq1D(xi,xf,dx,beta,T,par,intCond)
    Nx = round((xf-xi)/dx);
    dx = (xf-xi)/Nx;
    Nx = Nx+1;
    x = xi+(0:Nx-1)*dx;
    c = 1;
    alpha = 0.1;
    dt = sqrt(alpha)*dx/c;
    A = 2/(1+beta*dt);
    B = (1-beta*dt)/(1+beta*dt);
    C = alpha/(1+beta*dt);

    Nt = round(T/dt)+1;
    t  = (0:Nt-1)*dt;
    psi = zeros(Nt,Nx);
    [f,g] = intCond(x,c,par);
    psi(1,:) = f;
    psi(2,:) = f + g*dt + 0.5*alpha*lap(f);

    for k = 2:Nt-1
        psi(k+1,:) = A*psi(k,:) - B*psi(k-1,:) + C*lap(psi(k,:));
    end
    subplot(1,2,1)
    surf(x,t,psi,'edgecolor','none')
    xlabel('position')
    ylabel('time')

    subplot(1,2,2)
    imagesc(x,t,psi)
    xlabel('position')
    ylabel('time')
end

function L = lap(f)
    N = length(f);
    L = zeros(1,N);
    n = 2:(N-1);
    L(n) = f(n-1) - 2*f(n) + f(n+1);
end
```

```
function [f,g] = intCondParab(x,c,par)
% c is the speed of the wave
% f is the intial displacement
% g is the intial velocity
   a = par(1);
   b = par(2);

   f = 1-(x/b - a/b).^2;
   f(abs(x-a)>b) = 0;

   g = 0;
end
```

4.6 Calling `waveEq1D(0,3,0.02,0,8,[1,0.5],@intCondStrike)` produces

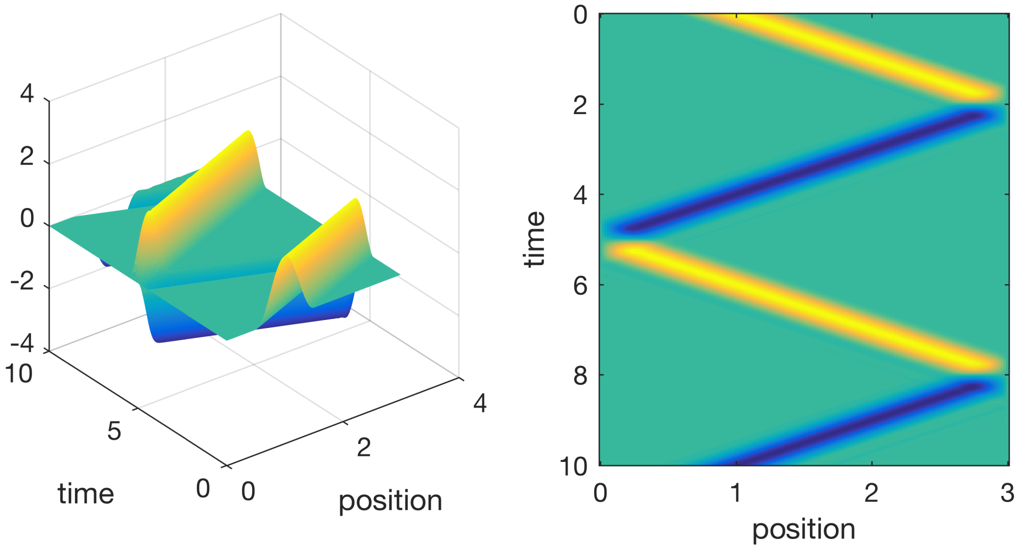


```
function [f,g] = intCondStrike(x,c,par)
% c is the speed of the wave
% f is the intial displacement
% g is the intial velocity
    a = par(1);
    b = par(2);

    f = 0;

    g = 1+cos(pi*(x-a)/b);
    g(abs(x-a)>b) = 0;
end
```

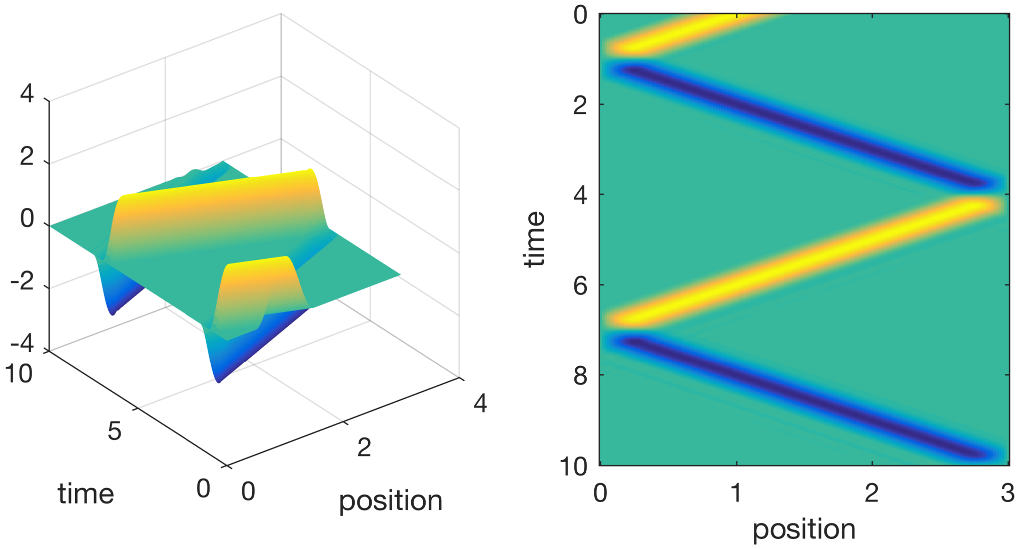
4.7 In both parts we use the following function

```
function [f,g] = intCondTravel(x,c,par)
   a = par(1);
   b = par(2);
   s = par(3);

   f = 1+cos(pi*(x-a)/b);
   f(abs(x-a)>b) = 0;

   g = s*c*(pi/b)*sin(pi*(x-a)/b);
   g(abs(x-a)>b) = 0;
end
```

(a) Calling `waveEq1D(0,3,0.02,0,8,[1,0.5,+1],@intCondTravel)` produces



(b) Calling `waveEq1D(0,3,0.02,0,8,[1,0.5,-1],@intCondTravel)` produces



4.8

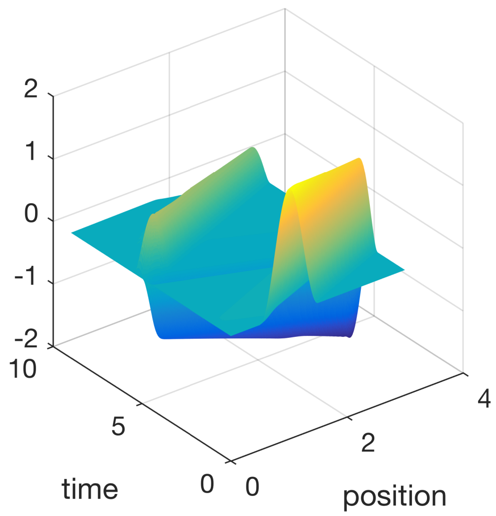
(a)

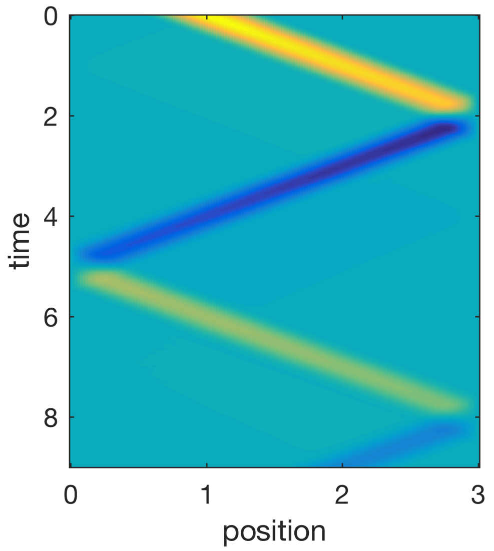
$$\begin{aligned}
\frac{\psi_n^{k-1} - 2\psi_n^k + \psi_n^{k+1}}{\Delta t^2} &= c^2 \frac{\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k}{\Delta x^2} - 2\beta \frac{\psi_n^{k+1} - \psi_n^{k-1}}{2\Delta t} \\
\psi_n^{k-1} - 2\psi_n^k + \psi_n^{k+1} &= \alpha\left(\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right) - \beta\Delta t\left(\psi_n^{k+1} - \psi_n^{k-1}\right) \\
(1+\beta\Delta t)\psi_n^{k+1} &= 2\psi_n^k - (1-\beta\Delta t)\psi_n^{k-1} + \alpha\left(\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right) \\
\psi_n^{k+1} &= \frac{2\psi_n^k - (1-\beta\Delta t)\psi_n^{k-1} + \alpha\left(\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right)}{1+\beta\Delta t} \\
&= A\psi_n^k - B\psi_n^{k-1} + C\left(\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right)
\end{aligned}$$

with

$$A = \frac{2}{1+\beta\Delta t}, \qquad B = \frac{1-\beta\Delta t}{1+\beta\Delta t}, \qquad C = \frac{\alpha}{1+\beta\Delta t}.$$

So we see that the recursion relation is nearly the same with $2 \to A$, $1 \to B$ and $\alpha \to C$.
(b) This is already implemented in the `waveEq1D()` code. So calling
`waveEq1D(0,3,0.02,0.15,9,[1,0.5,+1],@intCondTravel)`
simulates damping with $\beta = 0.15$.





4.9 To Be Done

4.10 To Be Done

4.11 The equation $\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi - 2\beta\frac{\partial\psi}{\partial t}$ becomes

$$\frac{\psi^{k+1}_{n,m} - 2\psi^{k}_{n,m} + \psi^{k-1}_{n,m}}{\Delta t^2} = c^2\frac{\mathcal{L}^k_{n,m}}{\Delta x^2} - 2\beta\frac{\psi^{k+1}_{n,m} - \psi^{k-1}_{n,m}}{2\Delta t}$$

or

$$\psi^{k+1}_{n,m} - 2\psi^{k}_{n,m} + \psi^{k-1}_{n,m} = \frac{c^2\Delta t^2}{\Delta x^2}\mathcal{L}^k_{n,m} - \beta\Delta t\ \psi^{k+1}_{n,m} + \beta\Delta t\ \psi^{k-1}_{n,m}$$

or

$$(1+\beta\Delta t)\psi^{k+1}_{n,m} - 2\psi^{k}_{n,m} + (1-\beta\Delta t)\psi^{k-1}_{n,m} = \frac{c^2\Delta t^2}{\Delta x^2}\mathcal{L}^k_{n,m}$$

Solving for $\psi^{k+1}_{n,m}$ we find

$$\psi^{k+1}_{n,m} = \frac{1}{1+\beta\Delta t}\left[2\psi^{k}_{n,m} - (1-\beta\Delta t)\psi^{k-1}_{n,m} + \frac{c^2\Delta t^2}{\Delta x^2}\mathcal{L}^k_{n,m}\right]$$

If β is not a constant for computational efficiency it is better to write this as

$$\psi^{k+1}_{n,m} = A_{n,m}\psi^{k}_{n,m} - B_{n,m}\psi^{k-1}_{n,m} + C_{n,m}\mathcal{L}^k_{n,m}$$

with

$$\begin{aligned} A_{n,m} &= \frac{2}{1+\beta_{n,m}\Delta t} \\ B_{n,m} &= \frac{1-\beta_{n,m}\Delta t}{1+\beta_{n,m}\Delta t} \\ C_{n,m} &= \frac{c^2_{n,m}\Delta t^2}{\Delta x^2}\frac{1}{1+\beta_{n,m}\Delta t} \end{aligned}$$

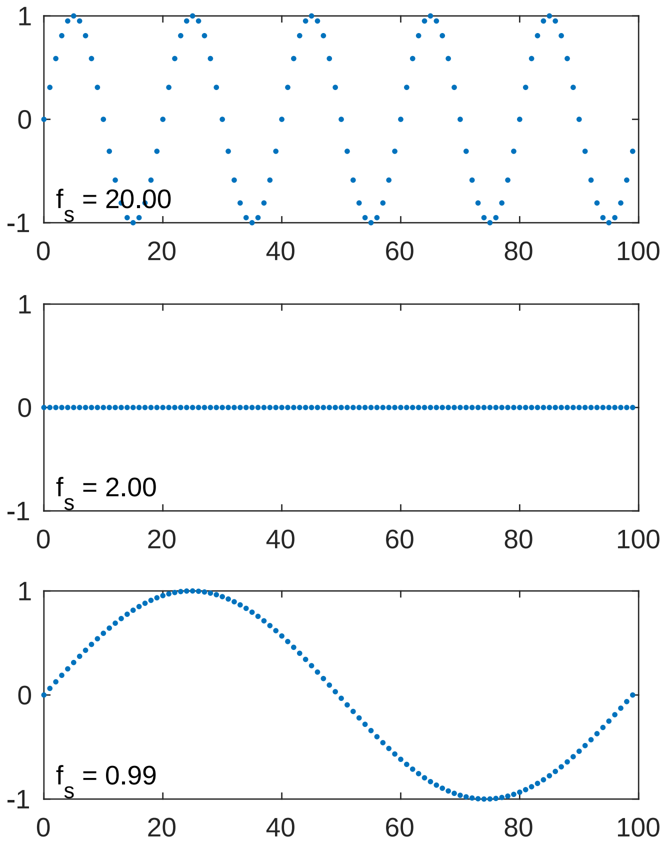
keeping in mind that these three matrices are constants it is best to compute them once before the time loop.

4.12 To Be Done

4.13 To Be Done

5.1 We can compute the values using

$$x_n = x(t_n) = \sin(2\pi t_n) = \sin(2\pi n\Delta t) = \sin(2\pi n/f_s).$$



```
function sampleSin()
N = 100;
fs = [20,2,0.99];
n = 0:N-1;
for k = 1:3
    t = n/fs(k);
    x = sin(2*pi*t);
    subplot(3,1,k)
    plot(n,x,'.')
    ylim([-1,1])
    text(2,-0.8,sprintf('f_s = %0.2f',fs(k)))
end
end
```

5.2 The idea is to add or subtract a multiple of f_s from the signal frequency until you get a frequency between $-f_s/2$ and $f_s/2$.
(a) $x = \sin(2\pi(130\text{Hz})t) \longrightarrow x_{\text{alias}} = \sin(2\pi(130 - 200\text{Hz})t) = \sin(2\pi(-70\text{Hz})t)$.
(b) $x = \sin(2\pi(237\text{Hz})t) \longrightarrow x_{\text{alias}} = \sin(2\pi(37\text{Hz})t)$.
(c) $x = \sin(2\pi(320\text{Hz})t) \longrightarrow x_{\text{alias}} = \sin(2\pi(-80\text{Hz})t)$.
(d) $x = \sin(2\pi(450\text{Hz})t) \longrightarrow x_{\text{alias}} = \sin(2\pi(50\text{Hz})t)$.
(e) $x = \sin(2\pi(-140\text{Hz})t) \longrightarrow x_{\text{alias}} = \sin(2\pi(60\text{Hz})t)$.
(f) $x = \cos(2\pi(237\text{Hz})t) \longrightarrow x_{\text{alias}} = \cos(2\pi(37\text{Hz})t)$.

5.3

$$\begin{aligned} e^{i2\pi(f\pm f_s)t_k} &= e^{i2\pi f t_k} e^{\pm i2\pi f_s t_k} \\ &= e^{i2\pi f t_k} e^{\pm i2\pi k} \\ &= e^{i2\pi f t_k} \left(e^{\pm i2\pi}\right)^k \\ &= e^{i2\pi f t_k} \left(1\right)^k \\ &= e^{i2\pi f t_k} \end{aligned}$$