Problem 1.

(60 points)
Consider the initial value problem (IVP) for the ordinary differential equation (ODE):

$$y'(t) = \frac{dy}{dt} = \sin(y(t)), t \in [0, 1]$$
$$y(0) = 1$$

Let's partition the domain [0, 1] into n sub-intervals equally with mesh size $h = 1/n$, i.e., $0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = 1$ with $t_k = k/n$ for $k = 0, 1, \ldots, n$. We will find approximation of the solution $y(t)$ at the grid points $\{t_0, t_1, \ldots, t_n\}$ as $y_k \approx y(t_k)$ for $k = 0, 1, \ldots, n$. It is obvious that $y_0 = 1$ by the initial condition.
Implement the following methods with Matlab (using $n = 20, 40$) and plot the results.

(a) At each point t_k, approximate the derivative $y'(t_k)$ with forward difference to get

$$y_{k+1} = y_k + h\sin(y_k)$$

(b) At each point t_k, approximate the derivative $y'(t_k)$ with backward difference to get

$$y_{k+1} = y_k + h\sin(y_{k+1})$$

To obtain update y_{k+1}, use Fixed point iteration to solve the nonlinear equation for y_{k+1} (using $y_k + h\sin(y_k)$ as initial guess in the Matlab routine for fixed point iteration).

(c) Take integral of the ODE from t_k to t_{k+1} to get

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} \sin(y(t))dt,$$

and use Trapezoidal rule to approximate the above integral to get

$$y_{k+1} = y_k + h\frac{\sin(y_k) + \sin(y_{k+1})}{2}.$$

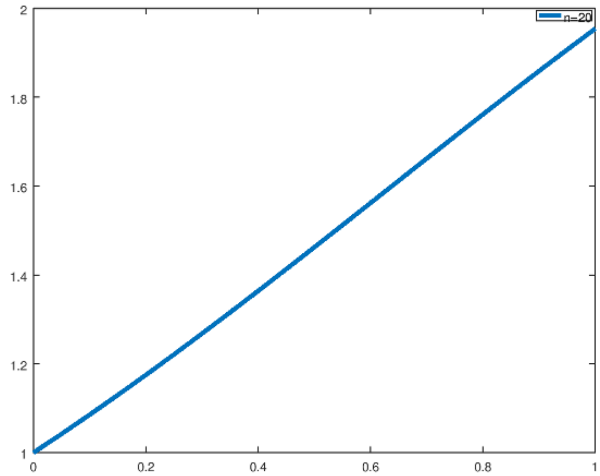
To obtain update y_{k+1}, use Fixed point iteration to solve the nonlinear equation for y_{k+1} (using $y_k + h\sin(y_k)$ as initial guess in the Matlab routine for fixed point iteration).

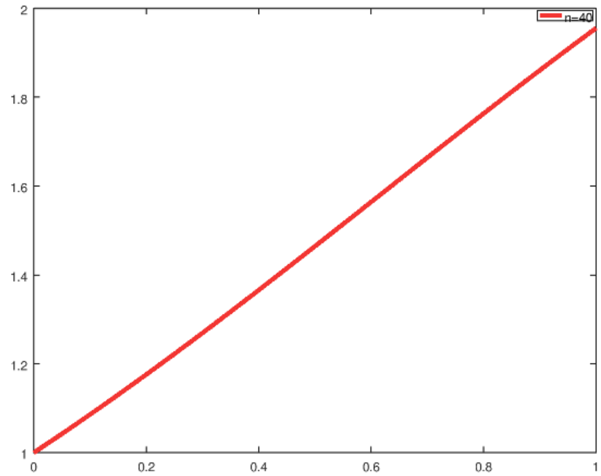
Solution

Part (a)

```
function [x, y] = p1a(n)

% step size
h = 1/n;
x = 0:h:1;

% initial condition
y = [1];

% forward difference
ynext = @(yk) yk + h * sin(yk);

% skip the initial condition
for _x = h:h:1
        yk = y(length(y))
        y = [y ynext(yk)]
end

end
```

Below we can see the results of plotting the results of $p1a(20)$ and $p1a(40)$. Where left (blue) is $n = 20$ and the right (red) is $n = 40$.

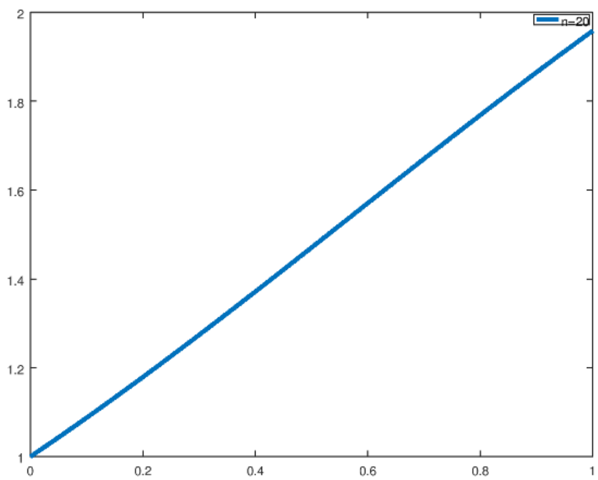


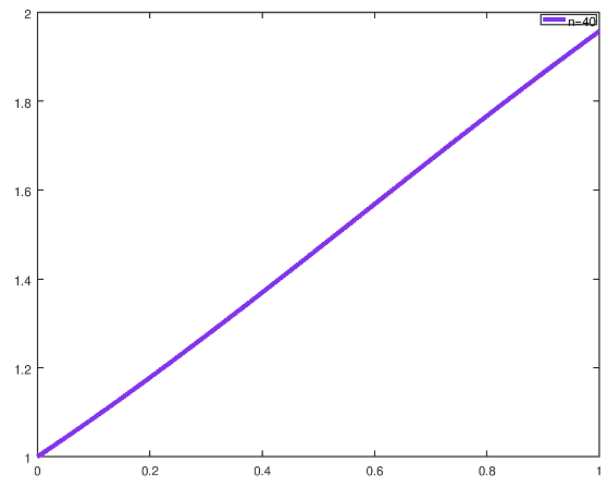


Part (b)

```
function [x, y] = p1b(n)

% step size
h = 1/n;
x = 0:h:1;

% initial condition
y = [1];

% backward difference
guess = @(yk) yk + h * sin(yk);
ynext = @(yk, ykk) yk + h * sin(ykk);

% skip the initial condition
for _x = h:h:1
        yk = y(length(y));
        g = @(guess) yk + h * sin(guess);
        [ykk _ykk] = fixedpoint(g, guess(yk), 1e-6, 1000);

        y = [y ynext(yk, ykk(length(ykk)))];
end

end
```

Again we get very similar looking results by running this method with step sizes $n = 20, 40$. Below you can see a plot of the results of calling the above Matlab code. The left (blue) is for $n = 20$ and the right (purple) for $n = 40$.

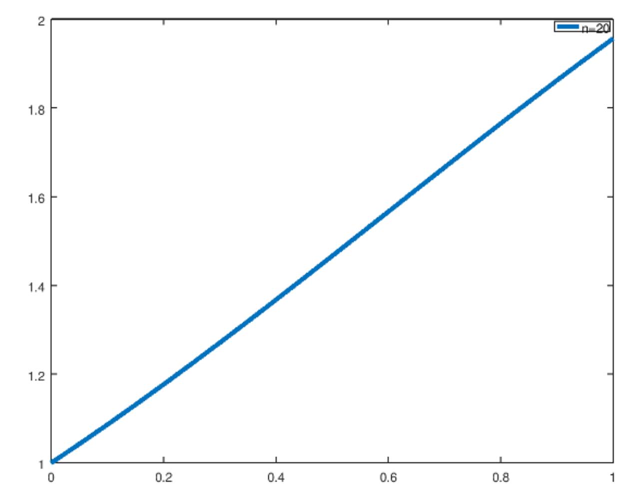


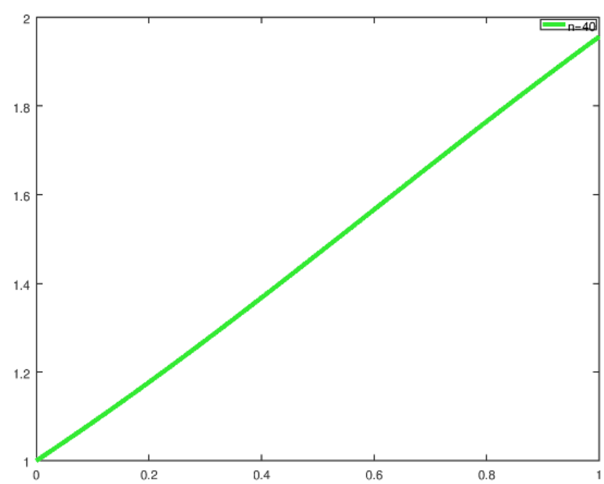


Part (c)

```
function [x, y] = p1c(n)

% step size
h = 1/n;
x = 0:h:1;

% initial condition
y = [1];

% trapezoidal rule
guess = @(yk) yk + h * sin(yk);
ynext = @(yk, ykk) yk + 0.5 * h * (sin(yk) + sin(ykk));

% skip the initial condition
for _x = h:h:1
        yk = y(length(y));
        g = @(guess) yk + h * sin(guess);
        [ykk _ykk] = fixedpoint(g, guess(yk), 1e-6, 1000);

        y = [y ynext(yk, ykk(length(ykk)))];
end

end
```

The code for this method is very similar to part b, only requiring an update to the function which computes the next value (though it takes the same parameters). Below you can see a plot af the results of calling the above Matlab code. The left (blue) is for $n = 20$ and the right (green) for $n = 40$.





Problem 2.

(40 points)
Consider the Boundary value problem for the Poisson equation:

$$-u''(x) = f(x), x \in [0,1]$$
$$u(0) = \alpha, u(1) = \beta.$$

Let's partition the domain $[0,1]$ into n subintervals equally with mesh size $h = 1/n$, i.e. $0 = x_0 < x_1 < \cdots < x_{n-1} < x_n = 1$, with $x_k = k/n$ for $k = 0, 1, \ldots, n$. We will find an approximation of the solution $u(x)$ at the grid points $\{x_0, x_1, \ldots, x_n\}$ as $u_k \approx u(x_k)$ for $k = 0, 1, \ldots, n$. It is obvious that $u_0 = \alpha, u_n = \beta$ by the Boundary condition.
At each point x_k for $k = 1, 2, \ldots, n-1$, approximate the second derivative $u''(x_k)$ using three-point central difference

$$u''(x_k) \approx \frac{u(x_{k+1}) - 2u(x_k) + u(x_{k-1})}{h^2}$$

With the above approximation to set up a linear system to determine $\{u_1, u_2, \ldots, u_{n-1}\}$, i.e.,

$$A\mathbf{u} = \mathbf{b}$$

with

$$A = \begin{bmatrix} 2 & -1 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & -1 & 2 & -1 \\ 0 & \cdots & \cdots & -1 & 2 \end{bmatrix}, u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}, b = h^2 \begin{bmatrix} f_1 + u_0/h^2 \\ f_2 \\ \vdots \\ f_{n-2} \\ f_{n-1} + u_n/h^2 \end{bmatrix}$$

Implement the above method with Matlab and plot the results. Choose $f(x) = 4\sin(2x), \alpha = 0, \beta = \sin(2), n = 20, 40$.

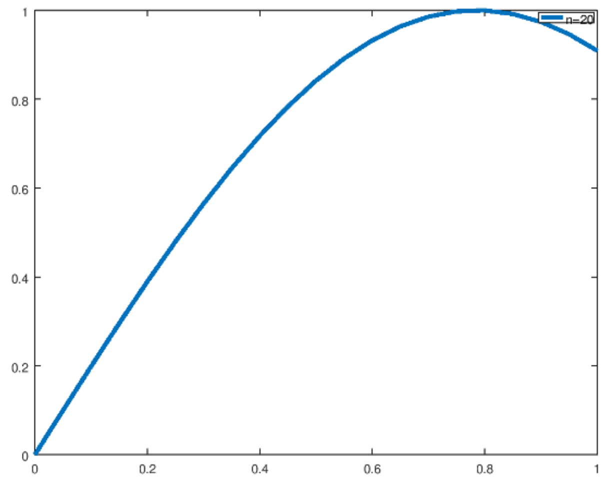
(a) Find the maximum and minimum eigenvalues (in magnitude) of A.

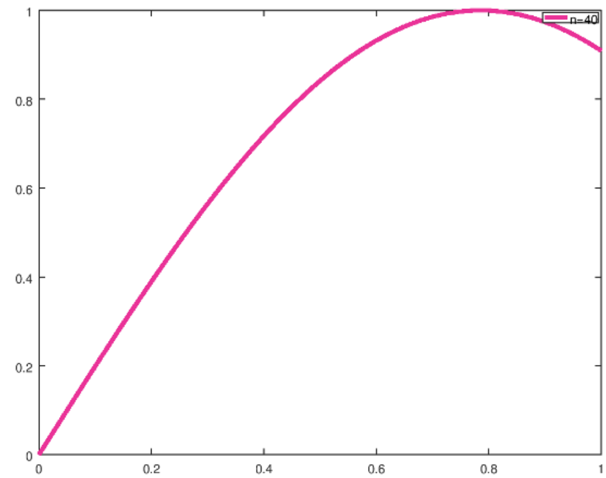
(b) solve the linear system by Gaussian Elimination with Backward substitution.

Solution

```
function [x, u] = p2(f, alpha, beta, n)

% want to approximate u(x)
h = 1/n;
k = 0:n;
x = k/n;

% boundary condition
u0 = alpha;
un = beta;

% construct the linear system we need to solve
% this will give us u_1 through u_{n-1}

A = diag(repmat(2, n-1, 1));
B = shift(resize(diag(repmat(-1, n-2, 1)), n-1, n-1), 1);
A += B + B';

b = [];
for k=2:n
        b = [b f(x(k))];
end

b(1) += u0 / (h^2);
b(length(b)) += un / (h^2);
b *= (h^2);

% Au = b => u = A^{-1}b
u = inv(A) * b';
% add the boundaries to the result
u = [u0 u' un];

end
```

Running the above code using the provided f, α, β values and plotting the output produces the following two graphs. The left (blue) is for $n = 20$ where the right (red) is for $n = 40$.





Part (a)

```
function [x, u] = p2eig(n)

% generate the matrix A
A = diag(repmat(2, n-1, 1));
B = shift(resize(diag(repmat(-1, n-2, 1)), n-1, n-1), 1);
A += B + B';

% compute eigenvalues
e = eig(A);

% we want max/min in magnitude
mags = abs(e);
% find the indecies of the max/min magnituse
% we will use these to look up the actual values in e
[_max maxidx] = max(mags);
[_min minidx] = min(mags);

printf('for n=%d\n', n);
printf('minimum eigenvalue: %f\n', e(maxidx(1)))
printf('maximum eigenvalue: %f\n', e(minidx(1)))

end
```

The above script uses the Matlab built in *eig* function to compute the eigenvalues of the matrix A generated by the input n. The script then uses this list to find the maximum and minimum eigenvalues in magnitude. Running the script with $n = 20, 40$ produces the following results...

```
for n=20
minimum eigenvalue: 3.975377
maximum eigenvalue: 0.024623

for n=40
minimum eigenvalue: 3.993835
maximum eigenvalue: 0.006165
```

Part (b)