Problem 1.

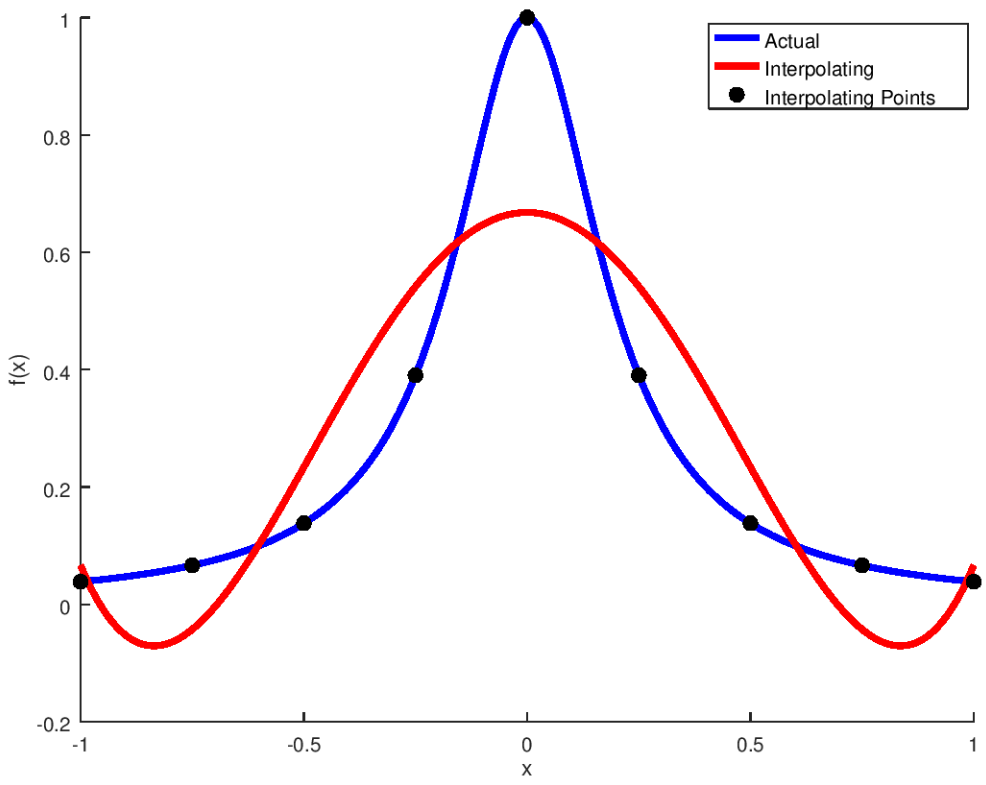
Use Matlab (or some other program) to calculate the polynomial which interpolates the function

$$f(x) = \frac{1}{1+25x^2} \tag{1}$$

at equally spaced points between -1 and 1, in steps of 0.25. You can use Matlab built-in to find the polynomial.

Plot the original function (in default color blue) and the interpolating polynomial (in red), and put circles around the interpolation points. Make sure you use enough points so that you get a smooth-looking curve.

Solution



```
function p1

clear all;
close all;
hold on;

```

```
f = @(x) 1 ./ (1 + 25 * x .^ 2);

% Actual curve.
_x = linspace(-1, 1, 200);
_y = f(_x);

% Interpolating points.
x = -1:0.25:1;
y = f(x);

% Fit a polynomial to the interpolating points.
p = polyval(polyfit(x, y, 5), _x);

plot(_x, _y, 'b', 'DisplayName', 'Actual', 'LineWidth', 3);
plot(_x,  p, 'r', 'DisplayName', 'Interpolating', 'LineWidth', 3);
scatter(x, y, 50, 'k', 'filled', 'DisplayName', 'Interpolating Points');
legend('show');
xlabel('x');
ylabel('f(x)');

end
```

Problem 2.

(a) Given a point x_0 and a step size h, use Lagrange polynomials to find the quadratic polynomial which interpolates a function f(x) at the points x_0-2h, x_0, and x_0+h.

(b) Integrate the polynomial on [x_0-2h, x_0+h] to produce an integration formula. Use the formula to estimate $\int_0^3 e^{-x}dx$.

Solution

Part (a)

Denote the Lagrange polynomial P(t) given by

$$\begin{aligned} P(t) &= f(x_0 - 2h)\frac{(t-t_1)(t-t_2)}{(t_0-t_1)(t_0-t_2)} \\ &+ f(x_0)\frac{(t-t_0)(t-t_2)}{(t_1-t_0)(t_1-t_2)} \\ &+ f(x_0+h)\frac{(t-t_0)(t-t_1)}{(t_2-t_0)(t_2-t_1)} \end{aligned}$$

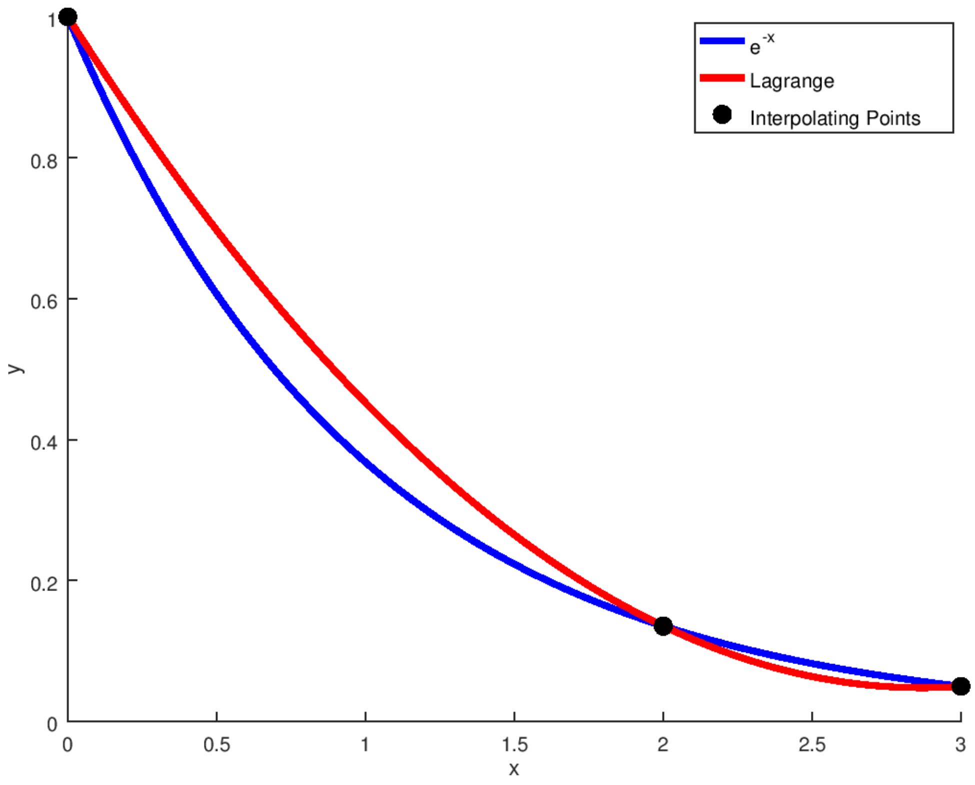
In this case we have $t_0 = x_0 - 2h$, $t_1 = x_0$, and $t_2 = x_0 + h$.
Next we make the necessary substitutions and simplify the different parts of P(t).

$$\begin{aligned} (t-t_1)(t-t_{)} &= (t-x_0)(t-(x_0+h)) \\ (t-t_0)(t-t_2) &= (t-x_0+2h)(t-x_0-h) \\ (t-t_0)(t-t_1) &= (t-x_0+2h)(t-x0) \\ (t_0-t_1)(t_0-t_2) &= ((x_0-2h)-x_0)((x_0-2h)-(x_0+h)) \\ &= (-2h)(x_0-2h-x_0-h) \\ &= (-2h)(-3h) \\ &= 6h^2 \\ &= (t-x_0)(t-x_0-h) \\ (t_1-t_0)(t_1-t_2) &= (x_0-(x_0-2h))(x_0-(x_0+h)) \\ &= (2h)(-h) \\ &= -2h^2 \\ (t_2-t_0)(t_2-t_1) &= ((x_0+h)-(x_0-2h))((x_0-2h)-x0) \\ &= (3h)(h) \\ &= 3h^2 \end{aligned}$$

Thus we arrive at the following equation

$$\begin{aligned} P(t) = &\frac{f(x_0 - 2h)}{6h^2}(t - x_0)(t - x_0 - h) \\ &+ \frac{f(x_0)}{-2h^2}(t - x_0 + 2h)(t - x_0 - h) \\ &+ \frac{f(x_0 + h)}{3h^2}(t - x_0 + 2h)(t - x_0) \end{aligned}$$

In the GNU Octave code below, P(t) is broken up into parts L0, L1, L2 which are defined on lines 26-28 of the code. The graph below shows an example case of the quadratic Lagrange polynomial for the function e^{-x} with $x_0 = 2$, $h = 1$. The original function is drawn in blue, while the Lagrange polynomial drawn is in red.



Part (b)

In order to integrate P(t), first expand each term to produce

$$\begin{aligned}P(t) = &\frac{f(x_0-2h)}{6h^2}(-ht+hx_0+t^2+x_0^2)\\ &+\frac{f(x_0)}{-2h^2}(-2h^2+ht-hx_0+t^2-2tx_0+x_0^2)\\ &+\frac{f(x_0+h)}{3h^2}(2ht-2hx_0+t^2-2tx_0+x_0^2)\end{aligned}$$

Integrating both sides produces

$$\begin{aligned}\int_{x_0-2h}^{x_0+h} P(\tau)d\tau = &\frac{f(x_0-2h)}{6h^2}\int_{x_0-2h}^{x_0+h}(-ht+hx_0+t^2+x_0^2)d\tau\\ &+\frac{f(x_0)}{-2h^2}\int_{x_0-2h}^{x_0+h}(-2h^2+ht-hx_0+t^2-2tx_0+x_0^2)d\tau\\ &+\frac{f(x_0+h)}{3h^2}\int_{x_0-2h}^{x_0+h}(2ht-2hx_0+t^2-2tx_0+x_0^2)d\tau\end{aligned}$$

$$\begin{aligned}\int_{x_0-2h}^{x_0+h} P(\tau)d\tau = &\frac{f(x_0-2h)}{6h^2}(-\frac{h}{2}\tau^2+hx_0\tau+\frac{1}{3}\tau^3-x_0\tau^2+x_0^2\tau)|_{x_0-2h}^{x_0+h}\\ &+\frac{f(x_0)}{-2h^2}(-2h^2\tau+\frac{h}{2}\tau^2-hx_0\tau+\frac{1}{3}\tau^3-x_0\tau^2+x_0^2\tau)|_{x_0-2h}^{x_0+h}\\ &+\frac{f(x_0+h)}{3h^2}(\mathrm{h}\tau^2-2hx_0\tau+\tfrac{1}{3}\tau^3-x_0\tau^2+x_0^2\tau)|_{x_0-2h}^{x_0+h}\end{aligned}$$

Each indefinite is given by intL0, intL1, & intL2 in the Octave code below. Each integral on $[x_0-2h, x_0+h]$ is evaluated at lines 44-46 and assigned to evalL0, evalL1, & evalL2. These are then multiplied by their corresponding constants producing k0, k1, and k2 which are then summed to produce the result of the integral of P(t) on the given bounds. Line 54 of the Octave code produces the actual integral of f using one of Octaves built in numerical integration methods. Line 55 of the Octave code produces the estimate of the integral of f, computed by evaluating the integral of P(t).

$$\int_0^3 e^{-x}dx = 0.95021$$
$$\int_0^3 P(t)dt = 1.0545$$

The estimate of the integral via the Lagrange polynomial has an error of 0.10429.

```
1 function p2
2
3 clear all;
```

```matlab
close all;
hold on;

% Constants used for testing.
h = 1.0;
x0 = 2.0;

% Example function to test our interpolation method.
f = @(x) e .^ (-x);
_x = linspace(0, 3, 200);

% List of our interpolation points.
x = [x0 - 2*h, x0, x0 + h];

% Plot our actual curve, and the points we are interpolating around.
plot(_x, f(_x), 'b', 'DisplayName', 'e^{-x}', 'LineWidth', 3);

% Next define the components of our interpolating function.
F0 = f(x0 - 2*h);
F1 = f(x0);
F2 = f(x0 + h);

L0 = @(x) (x - x0) .* (x - x0 - h) .* (1 / (6 * (h .^ 2)));
L1 = @(x) (x - x0 + 2 * h) .* (x - x0 - h) .* (1 / (-2 * (h .^ 2)));
L2 = @(x) (x - x0 + 2 * h) .* (x - x0) .* (1 / (3 * (h .^ 2)));

% Here is the actual interpolating polynomial.
P = @(x) F0 * L0(x) + F1 * L1(x) + F2 * L2(x);
plot(_x, P(_x), 'r', 'DisplayName', 'Lagrange', 'LineWidth', 3);
scatter(x, f(x), 75, 'k', 'filled', 'DisplayName', 'Interpolating Points');

% Compute the indefinate integral for P(x).
intL0 = @(t) (-h/2.0) .* (t .^ 2) + (h .* x0 .* t) + (1/3) .* (t .^ 3)  -
 ↪  (x0 .* (t .^ 2)) + ((x0 .^ 2) .* t);
intL1 = @(t) (-2 * (h .^ 2) .* t) + (h/2) .* (t .^ 2) - (h * x0) .* t +
 ↪  (1/3) .* (t .^ 3) - (x0 .* (t .^ 2)) + (x0 .^ 2) .* t;
intL2 = @(t) h .* (t .^ 2) - (2 * h * x0) .* t + (1/3) .* (t .^ 3) - x0 .*
 ↪  (t .^ 2) + (x0 .^ 2) .* t;

% Evaluate the above integrals over the bound (a, b).
a = x0 - 2 * h;
b = x0 + h;

evalL0 = intL0(b) - intL0(a);
```

```
evalL1 = intL1(b) - intL1(a);
evalL2 = intL2(b) - intL2(a);

% Calculate the partial sums that can be summed to estimate our integral.
k0 = (1 / ( 6 * (h ^ 2))) * f(x0 - 2 * h) * evalL0;
k1 = (1 / (-2 * (h ^ 2))) * f(x0) * evalL1;
k2 = (1 / ( 3 * (h ^ 2))) * f(x0 + h) * evalL2;

% Show the estimated integral against the actual error.
quad(f, 0, 3) % Numerical integration of f on the interval (0, 3).
(k0 + k1 + k2)

legend('show');
xlabel('x');
ylabel('y');

end
```

Problem 3.

For k=2, Iserles calls these the Nystrom and Milne methods. That is, start with

$$y(t_{n+2}) = y(t_n) + \int_{t_n}^{t_{n+2}} f(\tau, y(\tau))d\tau, \tag{2}$$

and interpolating at t_n, t_{n+1} (explicit) or t_n, t_{n+1}, t_{n+2} (implicit).
Find the leading error term, and the order of the method, for both cases.

Solution

Problem 4.

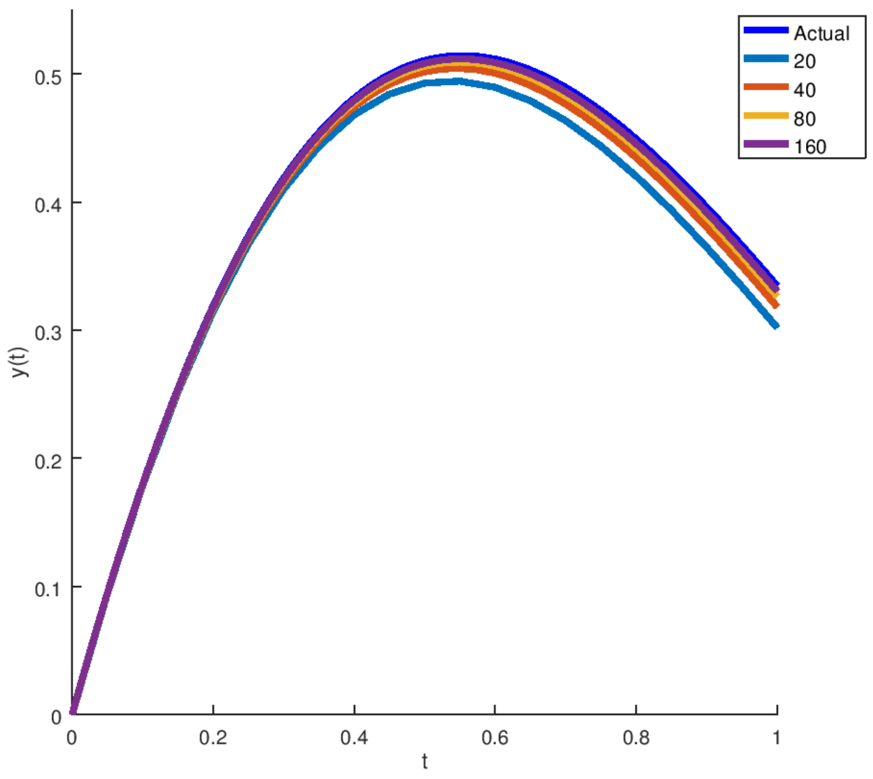
The IVP

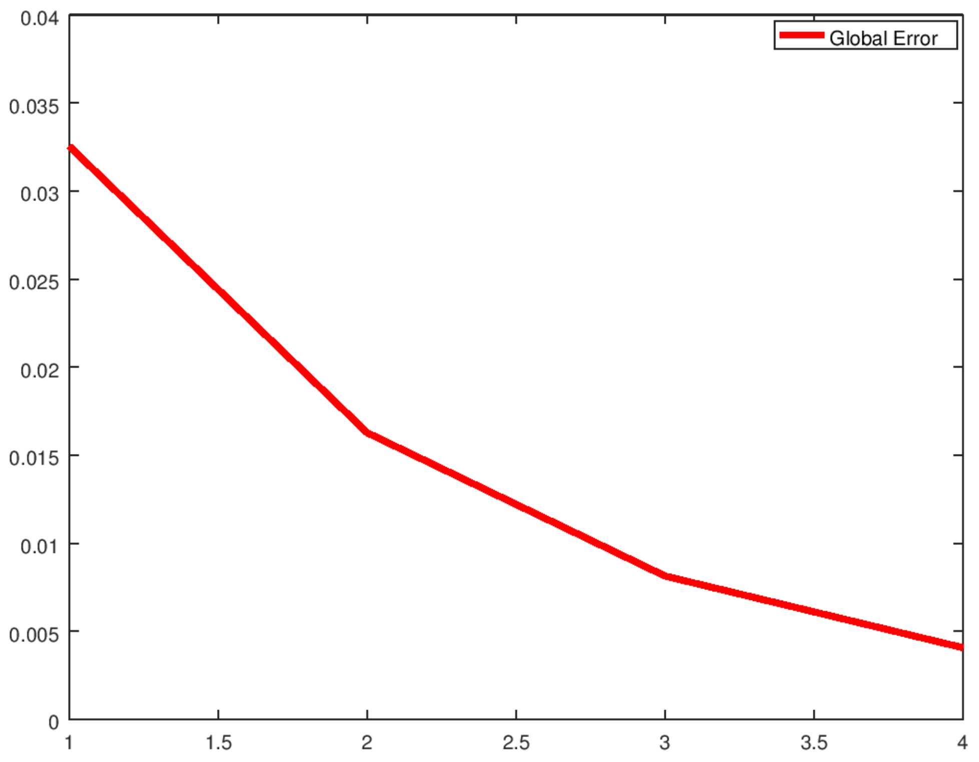
$$y' = -y + e^{-t}\cos(2t) \tag{3}$$

$$y(0) = 0 \tag{4}$$

has the true solution y(t) = $e^{-t}\sin(2t)$. Solve it numerically, using Euler's method with 20, 40, 80, 160 steps from 0 to 1, and compute the global error at the endpoint for each stepsize. Verify that the error goes down approximately linearly.

Solution





Note that the error term (graphed above) descends approximately linearly with respect to the step size.

```
function p4

clear all;
close all;
hold on;
axis([0, 1, 0, 0.55], 'square');

y  = @(t) (e .^ -t) .* (sin(2 .* t));
dy = @(t, y) -y + (2 .* e .^ -t) .* cos(2 .* t);

_x = linspace(0, 1, 200);
_y = y(_x);
plot(_x, y(_x), 'DisplayName', 'Actual', 'b', 'LineWidth', 3);

% Where to start and stop the approximation.
start = 0;
stop = 1;

ERR = zeros(4, 1);
```

```
steps = 20 .* 2 .^ (0:3);

for idx = 1:4

    h = (stop - start)/steps(idx);
    yk = zeros(steps + 1, 1);
    x = start:h:stop;

    yk(1) = 0; % Given initial point.
    for k = 1:steps(idx)
            tk = start + h * k;
            yk(k+1) = yk(k) + h * dy(tk, yk(k));
    end

    plot(x, yk, 'DisplayName', num2str(steps(idx)), 'LineWidth', 3);
    ERR(idx) = norm(yk - y(start:h:stop)', inf);
end

xlabel('t');
ylabel('y(t)');
legend('show');

%%%%%%%%%%%%%%%%%%%%%%
%% Graph the error. %%
%%%%%%%%%%%%%%%%%%%%%%
hold;
plot(1:4, ERR, 'r', 'LineWidth', 3, 'DisplayName', 'Global Error');
legend('show');

end
```

Problem 5.

Solve

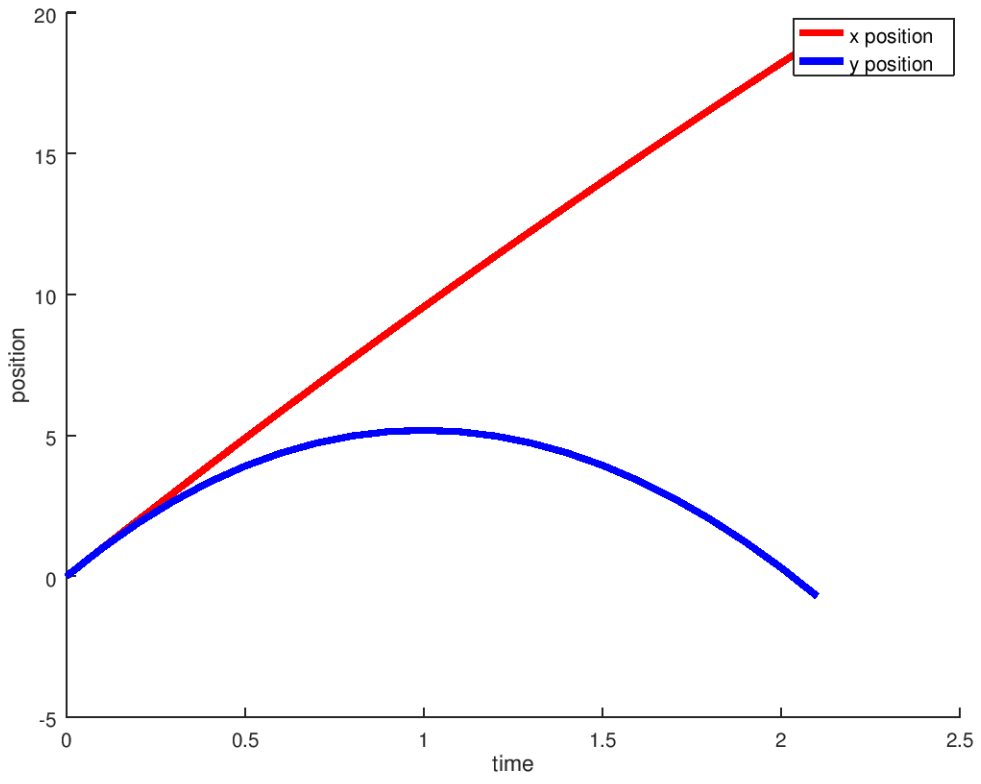
$$y''(t) = \begin{bmatrix} 0 \\ -10 \end{bmatrix} - 0.1y'(t),$$

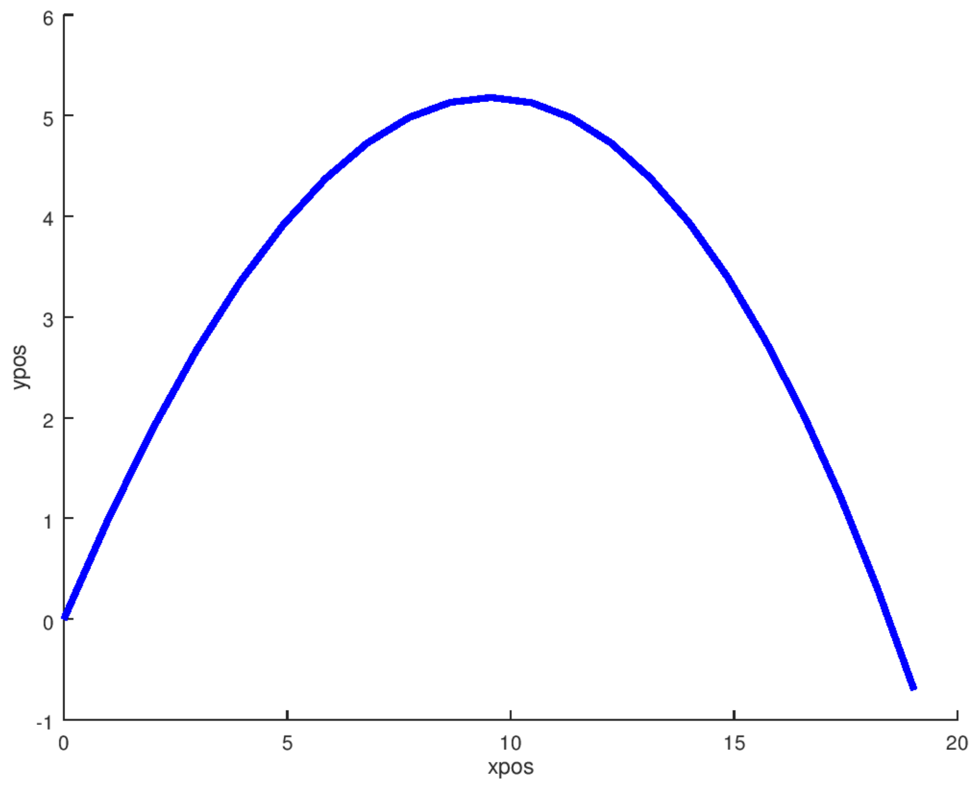
$$y(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$y'(0) = \begin{bmatrix} 10 \\ 10 \end{bmatrix}$$

by using Euler's method with stepsize h = 0.1, until y_2 becomes 0 or negative.
Plot y_1 and $y_)$ as functions of t (horizontal and vertical distance as functions of time), and also plot y_2 versus y_1. That will show the path of the object. It should look a bit like a parabola, but compressed on the right because of friction.

Solution





```
function p5

clear all;
close all;
hold on;

% Numerical solution initialized by the given initial values.
y1 = [[0; 0]];                  % y1 = y
y2 = [[10; 10]];        % y2 = y'

dy2 = @(t, y2) [0; -10] - (0.1 .* y2);
h = 0.1;        % Constant step size.
_t = [0];        % List of time steps for plotting.

% Need at least one pass to get off the ground...
do
        % Get the current time step.
        t = _t(length(_t));

        % Previous elements we are stepping off from.
        prev_y2 = y2(:, length( y2(2,:) ));
        prev_y1 = y1(:, length( y1(2,:) ));

        % Step the simulation.
        y1 = [y1, prev_y1 + h .* prev_y2];
        y2 = [y2, prev_y2 + h .* dy2(t, prev_y2)];

        % Step our method forward.
        _t = [_t, t+h];

% Until we reach the ground again.
until y1(2, length(y1(1,:))) <= 0 % Octave not Matlab :)

% Plot horizontal and vertical position with respect to to time.
xlabel('time');
ylabel('position');
xplot = plot(_t, y1(1,:), 'LineWidth', 3, 'r');
yplot = plot(_t, y1(2,:), 'LineWidth', 3, 'b');
legend('x position', 'y position');

% Plot horizontal against vertical position.
hold;
xlabel('xpos');
ylabel('ypos');
plot(y1(1,:), y1(2,:), 'LineWidth', 3, 'b');

end
```