Contents

- generate collocation points
- create matrix system of equations

```
function c = lsc(d, y, m, p, q, f, ua, ub)

    % Nikhil Jayswal
    % MATH 3890
    % Machine Problem 5
    % 22 Feb 2021

    % d = degree of spline
    % y = extended knot vector
    % m = number of equally spaced points in the interior
    %     of subinterval of partition
    %
    % solve u'' + pu' + qu = f
    % boundary conditions - s(a) = ua, s(b) = ub
    %
```

generate collocation points

interior knots

```
    dim = length(y) - d - 1;
    iknots = y(d+2:dim);
    % 'm' points in between each pair of interior knots
    cpts = [];
    for i = 1:length(iknots)-1
        a = iknots(i);
        b = iknots(i + 1);
        tmp = linspace(a, b, m+2);
        tmp = tmp(1:end-1);
        cpts = [cpts, tmp];
    end
    cpts = [cpts, iknots(end)];
```

create matrix system of equations

at each collocation point, satisfy the bvp s = c_i*N_i [N" + pN' + qN][c] = [f] [K][c] = [f] # of rows = # of collocation points # of cols = # of coefficients c_i = dimension of spline space

```
    K = zeros(length(cpts), dim);
    for i = 1:length(cpts)
```

```
    % find interval containing cpts(i)
    l = findinterval(dim, y, cpts(i));

    % assemble row #i of [K]
    % get all b-splines with non-zero value at the point cpts(i)
    b = bspl(d, l, cpts(i), y);
    for j = 1:dim
        if j < (l-d)
            K(i, j) = 0;
        end
        if j > l
            K(i, j) = 0;
        end
    end
    K(i, l-d:l) = q(cpts(i))*b;


    % get d/dt of b-splines
    b = bsplder(d, y, l, cpts(i), 1);
    for j = 1:dim
        if j < (l-d)
            K(i, j) = K(i, j) + 0;
        end
        if j > l
            K(i, j) = K(i, j) + 0;
        end
    end
    K(i, l-d:l) = K(i, l-d:l) + p(cpts(i))*b;


    % get d2/dt2 of b-splines
    b = bsplder(d, y, l, cpts(i), 2);
    for j = 1:dim
        if j < (l-d)
            K(i, j) = K(i, j) + 0;
        end
        if j > l
            K(i, j) = K(i, j) + 0;
        end
    end
    K(i, l-d:l) = K(i, l-d:l) + b;
end

% assemble [f] vector
fvector = f(cpts);
```

```
    % enforce boundary conditions
    % dirichlet boundary conditions
    % hence c_1 = ua, c_n = ub
    % reduce [K] and change fvector
    fvector = fvector' - ua*K(:, 1) - ub*K(:, dim);
    K = K(:, 2:dim-1);

    % get least squares solution for coefficients - c_2, ..., c_(n-1)
    c = K\fvector;
    % include ua and ub in c
    c = [ua; c; ub];

end
```