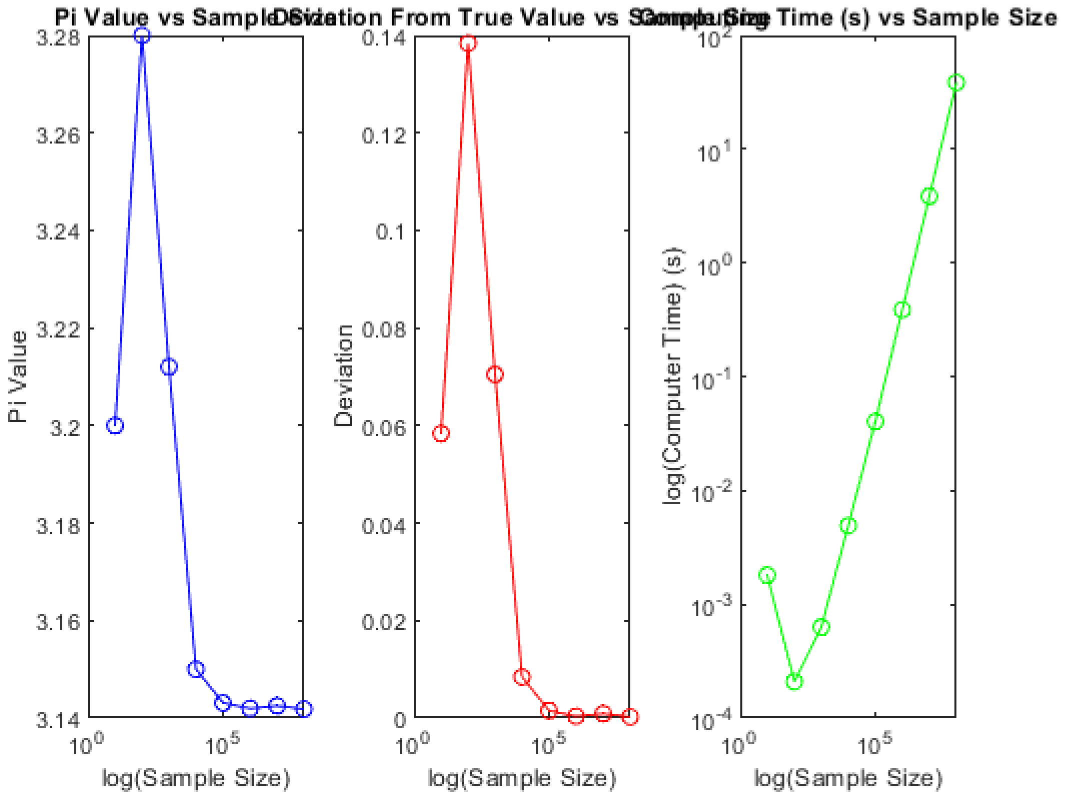
Table of Contents

Section 1: for loop

```
clear;

NEXP = 8;
piVal = 0;

times = [0];
piValues = [4 4];
sampleSize = [1 2];
deviations = [0];

fprintf('================================\nUsing for loop for
generation\n================================\n');

% conduct a trial for every power of 10 to NEXP
for exp = 1:NEXP
    % set numpoints
    piTotal = 0;
    numPoints = 10^exp;
    numPointsInCircle = 0;

    % start a timer to see how long the loop takes
    loopStart = tic;

    for i = 0:numPoints
        if generatePointAndCount()
            % calculate the fraction of points within the circle out of all
points generated
            proportion = numPointsInCircle/numPoints;
            numPointsInCircle = numPointsInCircle + 1;
            % calculate pi assuming that the proportion is roughly equal to
pi/4
            piVal = proportion * 4;
        end
    end

    avgPiVal = 2*piTotal/numPointsInCircle;

    % print result to console
    fprintf('monte carlo pi value for %d points: %f \n', numPoints, piVal);

    %end timer for loop
    loopTime = toc(loopStart);
    % store time, generated value, sample size, and deviation in respective
    % arrays
    times(exp) = loopTime;
    piValues(exp) = piVal;
    sampleSize(exp) = numPoints;
    deviations(exp) = abs(pi() - piVal);

end

% combine all data arrays into summary table
summaryTable = table(transpose(sampleSize), transpose(piValues),
transpose(deviations), transpose(times), 'VariableNames', ["Sample Size",
"Pi Values Calculated", "Deviation From True Value", "Time for
Calculation"]);
disp(summaryTable);

% plot figures showing requested statistics
tiledlayout(1,3);
nexttile;
semilogx(sampleSize, piValues, 'b-o');
title('Pi Value vs Sample Size');
xlabel('log(Sample Size)');
ylabel('Pi Value');

nexttile;
semilogx(sampleSize, deviations, 'r-o');
title('Deviation From True Value vs Sample Size')
xlabel('log(Sample Size)');
ylabel('Deviation');

nexttile;
loglog(sampleSize, times, 'g-o');
title('Computing Time (s) vs Sample Size');
xlabel('log(Sample Size)');
ylabel('log(Computer Time) (s)');
```

```
===============================
Using for loop for generation
===============================
monte carlo pi value for 10 points: 3.200000
monte carlo pi value for 100 points: 3.280000
monte carlo pi value for 1000 points: 3.212000
monte carlo pi value for 10000 points: 3.150000
monte carlo pi value for 100000 points: 3.143080
monte carlo pi value for 1000000 points: 3.141948
monte carlo pi value for 10000000 points: 3.142482
monte carlo pi value for 100000000 points: 3.141822
    Sample Size    Pi Values Calculated    Deviation From True Value    Time
for Calculation
    ___________    ____________________    _________________________
____________________

          10                   3.2
0.058407                  0.001807
```

```
         100                 3.28                       0.13841
0.0002085
        1000                3.212                      0.070407
0.0006308
       10000                 3.15                     0.0084073
0.0049223
       1e+05               3.1431
0.0014873                   0.040549
       1e+06               3.1419
0.00035535                    0.38649
       1e+07               3.1425
0.00088895                     3.8439
       1e+08               3.1418
0.00022967                     38.554
```



Section 2: while loop of specified precision

```
fprintf('================================\nUsing while loop for
generation\n================================\n');
theta = linspace(0, 2*pi(), 256);
circleX = cos(theta)+1;
circleY = sin(theta)+1;
generatedPoints=[rand(2,1)];
pointsInCircle = [[1;1]];
pointsOutsideCircle = [[0;0]];
```

```
piValues = [4];
piVal = 0;

SIG_FIGS = 4;

for sigFigs = 1:SIG_FIGS

    % While the calculated value does not have the desired number of correct
    % sig figs, continuously generate new points, append to original array,
    % plot, and recalculate the value. Break when the value is correct or the
    % set maximum number of tries is reached.
    while ~meetsSigFigs(sigFigs, piValues)
        generatedPoints = cat(2, generatedPoints, 2*rand(2,1));
        [piVal, pointsInCircle, pointsOutsideCircle]  =
getPiValFromCoords(generatedPoints, pointsInCircle, pointsOutsideCircle);
        piValues(numel(piValues)+1) = piVal;
        if meetsSigFigs(sigFigs, piValues)
            break;
        elseif numel(generatedPoints(1,:)) > 10^8
            disp('maximum tries expended')
            break;
        end
    end

    fprintf('Pi value with %d significant figure(s): %f \n', sigFigs, piVal);
    fprintf('Number of points required for %d significant figure(s): %d \n',
sigFigs, numel(generatedPoints(1,:)));
end
```

```
===============================
Using while loop for generation
===============================
Pi value with 1 significant figure(s): 3.225806
Number of points required for 1 significant figure(s): 31
Pi value with 2 significant figure(s): 3.280899
Number of points required for 2 significant figure(s): 89
Pi value with 3 significant figure(s): 3.125828
Number of points required for 3 significant figure(s): 453
Pi value with 4 significant figure(s): 3.135456
Number of points required for 4 significant figure(s): 3706
```

Functions

```
disp("")

% Check if the past 30 pi values have a consistent digit in the desired
% significant figure place; if they do, the number is assumed to be
% sufficiently precise
function hasFigs = meetsSigFigs(sigfigs, valueArray)
    hasFigs = false;
    minVal = numel(valueArray) - 30;
    difference = 10^(-sigfigs+1);
    if minVal <= 0
```

```
        return
    else
        for val = (minVal+1):numel(valueArray)
            if abs(valueArray(minVal) - valueArray(val)) > difference
                hasFigs = false;
                break;
            else
                hasFigs = true;
            end
        end
    end
end

% Function that accepts an array of randomly generated coordinates
% and provides a pi value based on the number that fall within
% a circle, as well as two arrays of points inside and outside the circle
function [valueToReturn, withinCircle, outsideCircle] =
getPiValFromCoords(coordArray, currentInsideCircle, currentOutsideCircle)
    intCount = 0.0;
    for coord = 1:numel(coordArray(1,:))
        if distanceToCenter([coordArray(1,coord),coordArray(2,coord)]) <= 1
            intCount = intCount + 1;
            withinCircle = cat(2, currentInsideCircle,
[coordArray(1,coord);coordArray(2,coord)]);
            outsideCircle = currentOutsideCircle;
        else
            outsideCircle = cat(2, currentOutsideCircle,
[coordArray(1,coord);coordArray(2,coord)]);
            withinCircle = currentInsideCircle;
        end
    end
    proportion = intCount/numel(coordArray(1,:));
    valueToReturn = proportion*4;
end

% Generate a random coordinate and determine if it falls within
% the inscribed circle via distance-to-center calculation; if it does, return
% true, and otherwise return false
function pointWithinCircle = generatePointAndCount()
    point = rand(1,2,1);
    if distanceToCenter(point) <= 1.0
        pointWithinCircle = true;
    else
        pointWithinCircle = false;
    end

end


% calculates distance between point argument (treated as [x;x])
% and center of square, taken to be [0.5;0.5]
function dist = distanceToCenter(point)
    dist = sqrt((1-point(1))^2 + (1-point(2))^2);
end
```

Published with MATLAB® R2024a