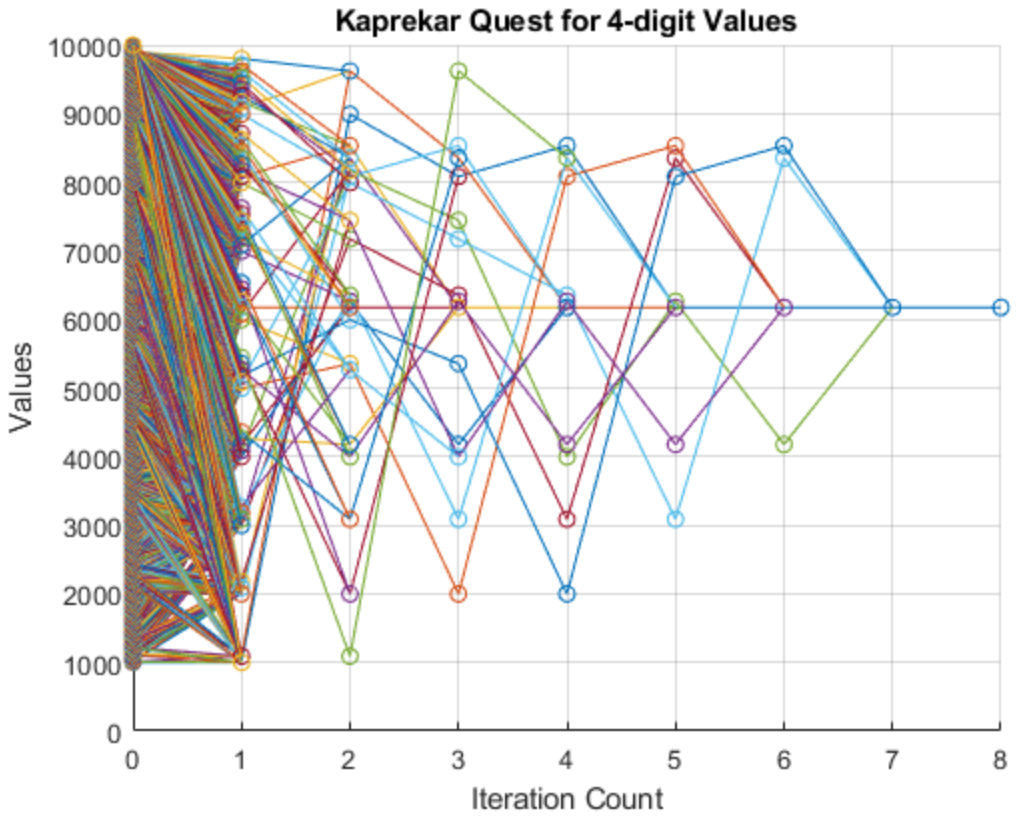
Table of Contents


```
% Pierce Zhang, CMOR220, Fall 2023, Project 4: Kaprekar's Constant
% Kaprekar.m
% Plot Kaprekar's quest for all valid four/five-digit integer starting values
% Last modified: 2 October 2023

% Driver method
function Kaprekar
```

SECTION ONE: KAPREKAR PATHS FOR FOUR-DIGIT VALUES

Naive plotting strategy

```
figure();
hold on;
grid on;

plot(0:8,quest(4,8),'-o');
title("Kaprekar Quest for 4-digit Values");
ylabel("Values"); xlabel("Iteration Count");
clear;
```

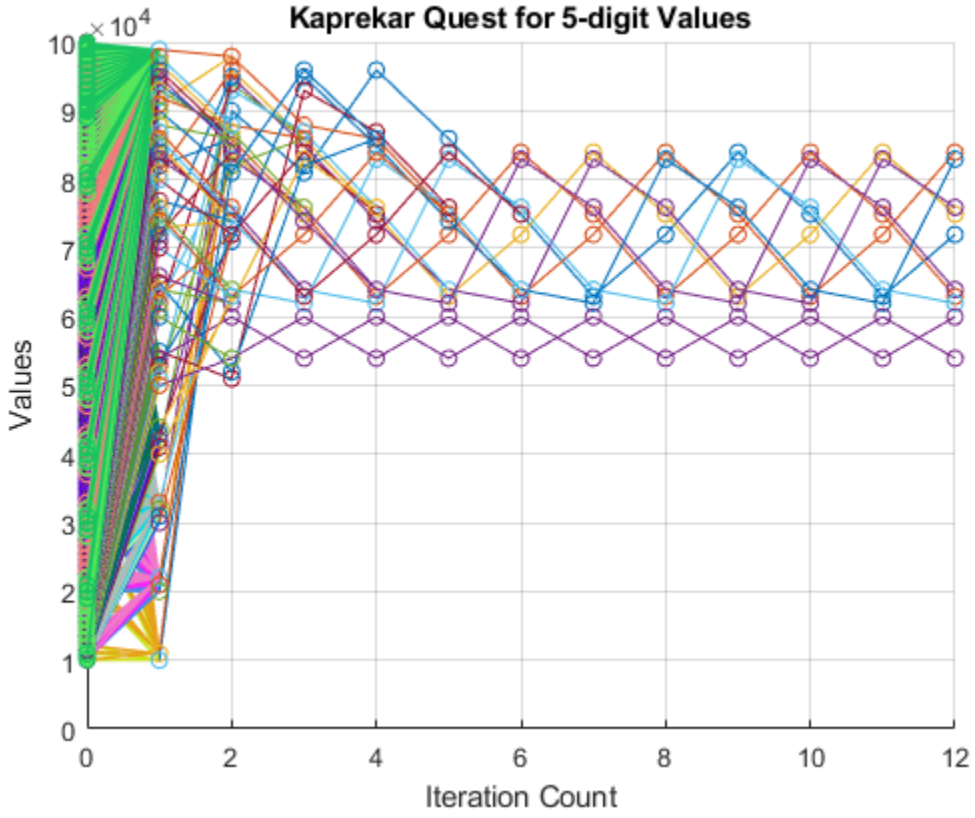


SECTION TWO: KAPREKAR PATHS FOR FIVE-DIGIT VALUES

The plotting operation of the five-digit Kaprekar paths utilizes two different optimizations. The first optimization is DP on the generation of the data matrix itself to ensure that there are no redundant paths throughout the entire matrix. The second optimization is to combine paths that terminate in the same point into the same line object, which makes it easier for MATLAB to render from a low-level standpoint. This is achieved using a zig-zag pattern based on the "entry" layer (column 2) of the dataset which in term is generated from discriminating the rows based on the unique values of the second layer. This optimization allows the five-digit values to be plotted in their entirety on my computer (13th Gen Intel i7-13700 / 16GB RAM). However, it may still not be enough for yours.

```
figure();
hold on;
grid on;

data = quest(5,12);
unique_entry_layer = unique(data(:,2)); % entry layer is the most crowded
% discriminate entire dataset based on unique values in entry layer
entry_layer_table = cell(1, length(unique_entry_layer)); % store in cell
for i = 1:length(unique_entry_layer)
    rows = data(:,2) == unique_entry_layer(i);
    entry_layer_table{i} = data(rows,:);
end

for i=1:length(entry_layer_table)
```

```
    B = []; % B will be plotted; A is auxiliary to extract data
    A = entry_layer_table{1,i};
    for j=1:length(A)
        B = [B ; 0 A(j,1)]; % zig-zag pattern
        B = [B ; 1 A(j,2)];
    end
    line(B(:,1),B(:,2),"Marker","o",'Color',[rand,rand,rand])
end
plot(1:12,data(:,2:end),'-o');
title("Kaprekar Quest for 5-digit Values");
ylabel("Values"); xlabel("Iteration Count");
```



QUESTIONS ANSWERED

Description: the figure shows the value of an integer, having started at the number posted on the y-axis, after the Kaprekar process has been run on it x-value times. There are 10 Kaprekar holes for five-digit numbers. They are:53955, 59994,61974,62964,63954,71973,74943,75933,82962,and 83952. There are no Kaprekar constants for five-digit numbers. However, there is a Kaprekar constant for four-digit numbers, which is 6174.

```
end

% Inputs: x, number of digits
% Outputs: d, first disallowed number with x digits
function [d] = dis(x)
    d = 0;
    for i = 0:x-1
```

```
        d = d + 10^i;
    end
end

% Inputs: n, number of digits, x, integer to iterate on Kaprekar's process
% Outputs: n_next, next integer of Kaprekar's process from x as a starter
function [n_next] = kaprekar_iter(n, x)
    % Performs the next iteration of Kaprekar's process
    string_num = num2str(x);
    if length(string_num) < n
        string_num = strcat('0',string_num);
    end
    asc = str2double(sort(string_num,'ascend'));
    desc = str2double(sort(string_num,'descend'));
    n_next = desc - asc;
end

% Inputs: n, number of digits, maxiter, number of iterations to use
% Outputs: data, a matrix containing the record of Kaprekar's process for
% all valid n-digit numbers
function [data] = quest(n, maxiter)
% Performs the Kaprekar Process on all x-digit integers
% for a number of iterations defined by maxiter.
% quest also stores each step of the Kaprekar Process in a
% data matrix, which will ultimately be a record of the
% path that each x-digit integer takes through the process
    V = 10^(n-1):1:10^n-1;
    D = dis(n);
    V(mod(V,D) == 0) = [];

    % Preallocate dp to ensure size bounds
    dp = NaN(1,10^n);
    seen = zeros(1,10^n);
    data = zeros(length(V),maxiter+1);
    data(:,1) = transpose(V); % first column

    % Use dynamic programming (dp) to memoize past states as an
    % optimization. This is the first optimization.
    for i=1:maxiter
        seen(:) = 0; % Reset "set"
        for j=1:length(V)
            if isnan(data(j,i)) % If data is NaN, indicating already
                % visited in previous calculation
                data(j,i+1) = NaN;
            elseif seen(data(j,i)) == 1 % Data seen in channel
                data(j,i+1) = NaN;
            elseif isnan(dp(data(j,i))) % If data not in dp
                data(j,i+1) = kaprekar_iter(n, data(j,i));
                dp(data(j,i)) = data(j,i+1);
                seen(data(j,i)) = 1;
            else
                data(j,i+1) = dp(data(j,i)); % Data in dp for last channel,
                % insert then block channel from further calculations
                seen(data(j,i)) = 1;
```

```
            end
        end
    end
end
```

Published with MATLAB® R2023a