Article title: Relationship Between Adherence Rate Threshold and Drug ‘Forgiveness’

Authors: Alan Morrison,* Melissa E. Stauffer, Anna S. Kaufman

*Corresponding author. ScribCo, Effort, Pennsylvania, USA; Email: i2scribbled@verizon.net
1. Calculation of probability of remaining within the therapeutic range

For a drug given once daily (I = 1) for a period of N days (30 by default), we wish to calculate \( nP_m \), the probability of remaining within the therapeutic range for the entire duration of treatment, given a number, m, of missed doses, for values of D of 2, 3, 4... i.e., values of F of 1, 2, 3... This probability is equal to the probability of there being no instances of \( >F \) consecutive missed doses and, for a 30-day period if m doses are missed, is given by \( 30P_m = 30S_m/30C_m \). Since the formula for \( 30C_m \) is known, we wish to compute \( 30S_m \), the number of combinations of m missed doses in N days with no missed doses on \( >F \) consecutive days.

The value of \( 30S_m \) can be calculated in two ways. First, by direct computation using a computer program. A program for this is given in Appendix B. Second, by inspection of combinations of missed doses for small values of N and m in order to observe a common pattern, and inference of the observed pattern to values of N = 30 and m = 1-30. Here, we illustrate the second approach for F = 1.

2. Calculation of \( 30S_m \) for F = 1

When F = 1, the serum drug level will remain within the therapeutic range for 1 day after a missed dose but then fall below the therapeutic range. Then, missed doses will not result in the serum drug level falling below the therapeutic range unless they occur on 2 or more consecutive days. If m = 1, there is no possibility of consecutive missed doses, so that \( 30S_1 = 30 \) and \( 30P_1 = 1.0 \), since \( 30C_1 = 30 \). Furthermore, when \( m > N/2 \), \( nP_m = 0 \), because there is no possibility of not having missed doses on consecutive days. For values of m between 1 and N/2, \( nP_m \), the probability of remaining within the therapeutic range is the probability of there being no consecutively missed doses, which is a function of \( nS_m \).

\( nS_m \) can be calculated as follows. First, take a simple example, with N=5 and m = 2 \((m_1 \text{ and } m_2)\): see Figure 3 below, which shows all possible combinations, \( 5C_2 \) (left panel), the subset of combinations with consecutive \( m_1 \text{ and } m_2 \) combinations (middle panel), and the subset with no consecutive \( m_1 \text{ and } m_2 \) combinations, i.e., \( 5S_2 \) (right panel).
Fig. 3 Left (a), all possible combinations of 2 missed doses, \( m_1 \) and \( m_2 \), in \( N = 5 \) positions \( (5\text{C}_2) \). Middle (b), subset of combinations of \( 5\text{C}_2 \) with consecutive missed doses \((5\text{T}_2)\). Right (c), subset of combinations of \( 5\text{C}_2 \) with no adjacent missed doses \((5\text{S}_2)\).

Referring to panel (c) of Figure 3, beginning with \( m_1 \) in position 1, there are 3 possible positions for \( m_2 \) that do not immediately follow \( m_1 \). With \( m_1 \) in position 2, there are 2 non-consecutive positions for \( m_2 \). With \( m_1 \) in position 3, there is only 1 position, where \( m_2 \) does not follow \( m_1 \) consecutively. Hence, there are \( 3 + 2 + 1 = 6 \) combinations of \( m_1 \) and \( m_2 \) in which there are no consecutive missed doses. The following is an example with \( m=3 \) and \( N=7 \) (Fig. 4).

Fig. 4 Combinations with no adjacent missed doses for \( m=3 \) and \( N=7 \)
It can be seen in Figure 4 that the number of combinations of non-consecutive $m_1$, $m_2$, and $m_3$ is $(3+2+1) + (2+1) + (1) = 10$.

By continuing this process, we can construct the partial matrix of $N$ rows and $m$ columns shown in Figure 5.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$N$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>15</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>21</td>
<td>20</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>15</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>35</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5 Partial matrix of values of $nS_m$ for $m = 2$ to 6 and $N = 1$ to 13, obtained by inspection. The entire matrix for all values of $m$ and $N$ can be calculated from the formula

$$nS_m = (n-1)S_m + (n-2)S_{m-1}$$

with columns for $m = 1$ and $m = 2$ as seed values.

It can be seen by inspection from Figure 5 that the following general rule applies:

$$nS_m = (n-1)S_m + (n-2)S_{m-1}$$

This recursive formula enables computation of the entire matrix for all values of $N$ and $m$. Furthermore, the series of integers in the columns in Figure 5 can be identified in the on-line encyclopedia of integer sequences (http://oeis.org/) as binomial coefficients, i.e., combinations. Specifically,

$$nS_m = (n-(m-1))C_{(N-(2m-1))}$$
This formula can be also used to calculate \(30 S_m\) for all values of \(m\). In addition, the above formula for \(NS_m\) as the sum of \((N-1)S_m\) and \((N-2)S_{(m-1)}\) can be rewritten as

\[NC_m = (N-1)C_m + (N-1)C_{(m-1)}\]

...which is the formula for Pascal’s Identity.

3. Program for calculating \(N T_m\)
The program consists of a Perl script that parses every possible combination of \(m\) misses in \(N\) daily doses and identifies combinations that contain streaks of \(S\) consecutive misses. The command line contains options for ranges of \(N\), \(m\), and \(S\), and also options for identifying combinations that contain \(\geq m\) misses or more than one instance of a streak of \(S\) consecutive misses. The output is the number of combinations that meet the criteria specified on the command line, i.e., the number of combinations that have the specified number of consecutive misses \((N T_m)\).

The command line options used for this analysis were:

```
-doses 30 -missed 2 -missedHigh 16 -streak 2 -streakHigh 4 -streakFuzzy
```

A similar command was used to obtain results for streaks of 2-4 consecutive misses within patterns of 17-26 misses over 30 days:

```
-doses 30 -missed 17 -missedHigh 26 -streak 2 -streakHigh 4 -streakFuzzy
```

The code is given below and the script is available on request.
#!/usr/bin/perl -w

md3.pl

Calculates number of occurrences of dosages missed in a row based on a period and the number of total misses in that period.

Parameters:
--doses-integer Required. Number of doses in the period. Max 32.
--dosesHigh-integer Optional. If specified indicates high end of a range of doses to process. Max 32.
--missed-integer Required. Number of doses missed in the period. Base of range if high range indicated.
--missedHigh-integer Optional. If specified indicates high end of a range of missed doses.
--missedfuzzy Optional. Flag if specified indicates missed processing is at least value or higher.
  Default is exact match.
--streak-integer Required. Number of doses missed in a row to be looked for. Base of range if high range indicated.
--streakHigh-integer Optional. If specified indicates high end of a range of streaks.
--streakfuzzy Optional. Flag if specified indicates streak processing is any number of streaks.
  Default is one streak only in dosage pattern.
--help Optional. Prints usage and exits.
--debug Optional. Enables debugging.

History
2016-11-24 2.1.0 BCG Changed loop pattern match on multi doses to use shifts rather than exponentiation
2016-11-21 2.0.0 BCG Added "--dosesHigh" parameter to process range of doses in single pass
  Major redesign - use predictive pattern generation to minimize coverage needed.
2016-11-18 1.2.0 BCG Modified countdown print to use STDERR allowing summary output routing to a file.
2016-11-14 1.1.0 BCG Performance optimization - reduce streak loop size for eligible patters as streak's can't
  be longer than the number of misses in the pattern
2016-11-13 1.0.0 BCG Delivered version
2016-11-10 0.5.0 BCG Adding range processing for both missed and streak values.
2016-11-10 0.1.0 BCG New

use strict;
use warnings;

use Getopt::Long;
use File::Basename;
use File::Path;

## Runtime environment setup
# capture script name, allow for Windows paths

( my $scriptName = basename($0) ) =~ s/\.[^.]\+$//;

## Parameter setup

my $fHelp = '';  
my ( $dosages, $missed, $streak ) = ( -1, -1, -1 );  
my ( $dosagesHigh,$missedHigh, $streakHigh ) = ( 0, 0, 0 );  
my ( $fMissedFuzzy, $fStreakFuzzy ) = ( '', '' );  
my $fDebug = '';  

## Process command line arguments

GetOptions(
  'help' => \$fHelp,  
  'doses-i' => \$dosages,  
  'dosesHigh-i' => \$dosagesHigh,  
  'missed-i' => \$missed,  
  'missedHigh-i' => \$missedHigh,  
  'missedfuzzy' => \$fMissedFuzzy,  
  'streak-i' => \$streak,  
  'streakHigh-i' => \$streakHigh,  
  'streakfuzzy' => \$fStreakFuzzy,  
  'debug' => \$fDebug,
);  

## Check for proper usage

if ( $fHelp ||  
( $dosages < 0 || $dosages > 62 ) ||  
( $dosagesHigh && ( ( $dosagesHigh > 62 ) || ( $dosagesHigh <= $dosages ) ) ) ||  
( $missed < 2 || $missed > $dosages ) ||  
( $streak < 1 || $streak > $missed ) ||  
( $missedHigh && ( ( $missedHigh > $dosages ) || ( $missedHigh <= $missed ) ) ) ||  
( $streakHigh &&  
  ( $missedHigh && ( ( $streakHigh > $missedHigh ) ) ) ||  
  ( !$missedHigh && ( ( $streakHigh > $missed ) ) ) ||  
  ( $streakHigh && ( $streakHigh <= $streak ) )  
)  
)  
Usage();  
exit 1;  

## Process the data
# Set processing ranges
if ( ! $streakHigh ) { $streakHigh = $streak };  
if ( ! $missedHigh ) { $missedHigh = $missed };  
if ( ! $dosagesHigh ) { $dosagesHigh = $dosages };  

# Prep results array
my @results;  
# four dimensions - doses/missed/streak/results  
# where results[d][m][s][0] is matching missed patterns,  
# results[d][m][s][1] is matching streak patterns

# Process by pattern generation  
# Work from longest pattern down - take advantage of fuzzy processing
my $patternsTotal = 2**$dosagesHigh;  
for ( my $misses = $missedHigh; $misses >= $missed; $misses -- ) {  
    # initialize pattern generator with lowest "missed" pattern
    my $pattern = 2**$misses - 1;  
    while ( $pattern < $patternsTotal ) {  
        # misses already known, skip bit counting section from md2.pl
        # work it as a known pattern of interest
        # add it to count for all streaks/doses/missed ranges with fuzzy
        # check and initialize streak match count if needed
        for ( my $d = $dosagesHigh; $d >= $dosages && $pattern < 2**$d; $d -- ) {  
            for ( my $m = $misses; $m >= $missed; $m -- ) {  
                for ( my $s = $streak; $s <= $misses; $s ++ ) {  
                    if ( defined ( $results[$d][$m][$s][0] ) ) {  
                        $results[$d][$m][$s][0] ++;  
                    } else {  
                        $results[$d][$m][$s][0] = 1;  
                        $results[$d][$m][$s][1] = 0;  
                    }  
                }  
            }  
        }  
    }  
}

last if ( ! $fMissedFuzzy );  
# ensures one missed processed but (  
# only one if exact match  
}  
# misses
for ( my $s = $misses; $s >= $streak; $s -- ) {
    if ( $fDebug ) { print "Missed $misses, checking streak $s, fuzzy [$fStreakFuzzy]\n"; }
    next if ! CheckStreak( $pattern, $s, $fStreakFuzzy ); # desired streak not found so skip
    if ( $fDebug ) { print "  found \n"; }
    # streak is interesting - record results for all misses of interest
    # counts against current "misses" only or all misses up to missedHigh if fuzzy
    # counts against current streak and all streaks below if fuzzy, else only against the current streak
    # counts against the dosage range if the streak is valid for that range
    for ( my ( $d, $dLimit ) = ( $dosagesHigh, 2<<(1+$dosagesHigh-1) ); $d >= $dosages && $pattern < $dLimit; $d --, $dLimit >>= 1 ) {
        for ( my $mMatch = $misses; $fMissedFuzzy ? $mMatch >= $missed : $mMatch == $misses; $mMatch -- ) {
            for ( my $sMatch = $s; $fStreakFuzzy ? $sMatch >= $streak : $sMatch == $s; $sMatch -- ) {
                $results[$d][$mMatch][$sMatch][1] ++;
            }
        }
    } last if ( $fStreakFuzzy );
}

## for-streak loop

## advance to next bit pattern with same number of misses - HAKMEM 175

use integer;
my $lowestBit = $pattern & -$pattern;
my $leftBits = $pattern + $lowestBit;
my $changedBits = $pattern ^ $leftBits;
my $rightBits = ( $changedBits / $lowestBit ) >> 2;
$pattern = $leftBits | $rightBits;
### while pattern loop

### for misses loop

print STDERR "\n\n";  # if out put to screen makes it prettier

## print results - input section

print "SUMMARY RESULTS\n\n";

for ( my $d = $dosages; $d <= $dosagesHigh;  $d ++ ) {
    print "+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*\n"
    print "\n"
    print "Doses during period: $d\n"
    print "Misses during period: $missed\n"
    if ( $missed != $missedHigh ) {
        print " to $missedHigh\n"
    }
    print ", searching for ";
    print ( $fMissedFuzzy ? "at least" : "exact" ) ;
    print " missed value\n"
    if ( $missed != $missedHigh ) {
        print "s in the range\n"
    }
    print " with\n"
    print "missed streak length;\n"
    if ( $streak != $streakHigh ) {
        print "$streak\n"
    }
    print " of: \n"
    if ( $streak == $streakHigh ) {
        print "\n"
    }
    print "$streak\n"
    if ( $streak != $streakHigh ) {
        print " to $streakHigh\n"
    }
    print " that occur\n"
    if ( $streak == $streakHigh ) {
        print "$streak\n"
    }
    print ( $fStreakFuzzy ? "at least" : "exactly" ) ;
    print " one time during the adherence pattern\n"
    print "\n";

    print "\n";
printf ( "Total adherence patterns checked:   %d
", 2**$d );

# print adherence populations
print "\n";
print " ** Adherence patterns : Missed criteria population ** \n";
printf ( "%-16s %-16s\n", "MISSING DOSES", "PATTERNS FOUND" );
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d %-16d\n", $m, defined $results[$d][$m][$streak][0] ? $results[$d][$m][$streak][0] : 0 );
}

# print adherence streaks
print "\n";
print " ** Adherence patterns : Occurrences of missed streaks ** \n";
printf ( "%-16s\n", "MISSSES | STREAKS" );
for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
   printf ( "%-16d\n", $s );
}
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d\n", $m );
   for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
      printf("%-16d", defined $results[$d][$m][$s][1] ? $results[$d][$m][$s][1] : 0 );
   }
   print "\n";
}

exit 0;

####
##  CheckStreak:
##

sub CheckStreak {
   my $pattern = shift;       # bit pattern to scan
   my $streak = shift;        # size of streak to find
   my $fuzzy = shift;         # fuzzy means any number of streaks

   ## Streak check starts with a bit pattern of "$streak" bits at lowest position
   ## then shifts left checking for matching patterns
   ## trick - streak is started based on the lowest 1-bit because
   ## everything to the right is 0's and can't be in the streak.
   ## that way we cut off the scan as the interesting bits shift left in the pattern

printf ( "Total adherence patterns checked:   %d\n", 2**$d );

# print adherence populations
print "\n";
print " ** Adherence patterns : Missed criteria population ** \n";
printf ( "%-16s %-16s\n", "MISSING DOSES", "PATTERNS FOUND" );
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d %-16d\n", $m, defined $results[$d][$m][$streak][0] ? $results[$d][$m][$streak][0] : 0 );
}

# print adherence streaks
print "\n";
print " ** Adherence patterns : Occurrences of missed streaks ** \n";
printf ( "%-16s\n", "MISSSES | STREAKS" );
for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
   printf ( "%-16d\n", $s );
}
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d\n", $m );
   for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
      printf("%-16d", defined $results[$d][$m][$s][1] ? $results[$d][$m][$s][1] : 0 );
   }
   print "\n";
}

print "\n";
}

exit 0;

####
##  CheckStreak:
##

sub CheckStreak {
   my $pattern = shift;       # bit pattern to scan
   my $streak = shift;        # size of streak to find
   my $fuzzy = shift;         # fuzzy means any number of streaks

   ## Streak check starts with a bit pattern of "$streak" bits at lowest position
   ## then shifts left checking for matching patterns
   ## trick - streak is started based on the lowest 1-bit because
   ## everything to the right is 0's and can't be in the streak.
   ## that way we cut off the scan as the interesting bits shift left in the pattern

printf ( "Total adherence patterns checked:   %d\n", 2**$d );

# print adherence populations
print "\n";
print " ** Adherence patterns : Missed criteria population ** \n";
printf ( "%-16s %-16s\n", "MISSING DOSES", "PATTERNS FOUND" );
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d %-16d\n", $m, defined $results[$d][$m][$streak][0] ? $results[$d][$m][$streak][0] : 0 );
}

# print adherence streaks
print "\n";
print " ** Adherence patterns : Occurrences of missed streaks ** \n";
printf ( "%-16s\n", "MISSSES | STREAKS" );
for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
   printf ( "%-16d\n", $s );
}
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d\n", $m );
   for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
      printf("%-16d", defined $results[$d][$m][$s][1] ? $results[$d][$m][$s][1] : 0 );
   }
   print "\n";
}

print "\n";
}

exit 0;

####
##  CheckStreak:
##

sub CheckStreak {
   my $pattern = shift;       # bit pattern to scan
   my $streak = shift;        # size of streak to find
   my $fuzzy = shift;         # fuzzy means any number of streaks

   ## Streak check starts with a bit pattern of "$streak" bits at lowest position
   ## then shifts left checking for matching patterns
   ## trick - streak is started based on the lowest 1-bit because
   ## everything to the right is 0's and can't be in the streak.
   ## that way we cut off the scan as the interesting bits shift left in the pattern

printf ( "Total adherence patterns checked:   %d\n", 2**$d );

# print adherence populations
print "\n";
print " ** Adherence patterns : Missed criteria population ** \n";
printf ( "%-16s %-16s\n", "MISSING DOSES", "PATTERNS FOUND" );
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d %-16d\n", $m, defined $results[$d][$m][$streak][0] ? $results[$d][$m][$streak][0] : 0 );
}

# print adherence streaks
print "\n";
print " ** Adherence patterns : Occurrences of missed streaks ** \n";
printf ( "%-16s\n", "MISSSES | STREAKS" );
for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
   printf ( "%-16d\n", $s );
}
for ( my $m = $missed; $m <= $missedHigh; $m ++ ) {
   printf ( "%-16d\n", $m );
   for ( my $s = $streak; $s <= $streakHigh; $s ++ ) {
      printf("%-16d", defined $results[$d][$m][$s][1] ? $results[$d][$m][$s][1] : 0 );
   }
   print "\n";
}

print "\n";
}
my $lowestBit = $pattern & -$pattern;
my $match = 2**$streak - 1;
while ( ! ( ( $match & -$match ) & $lowestBit ) ) {
  $match <<= 1;
}
my $found = 0;
while ( $match <= $pattern ) {
  if ( $fDebug ) { printf( "  CheckStreak: pattern %b   match %b\n", $pattern, $match ); }
  if ( ( $pattern & $match ) == $match ) {
    # found a streak
    $found ++;
    if ( $fuzzy ) { # mark it found
      return 1;
    } elsif ( $found > 1 ) { # only wanted one
      return 0;
    }
  }
  # loop continues to cover case of not fuzzy over entire pattern
  $match = $match << 1;
  # shift bit pattern for next position to check
  # until the bit pattern is beyond bit pattern to search
}
return $found;
# found has either 0 or 1 at this point

## Usage: Display usage message

sub Usage {
  print STDERR "Usage: $0 [--help] --doses <number> [ --dosesHigh <number> ]\n";
  print STDERR " --missed <number> [ --missedHigh <number> ][ --missedFuzzy ]\n";
  print STDERR " --streak <number> [ --streakHigh <number> ][ --streakFuzzy ]\n";
  print STDERR " where doses must be positive and at most 62\n";
  print STDERR " dosesHigh if specified is the high end of a range of dose populations to process\n";
  print STDERR " NOTE: if a range of doses is processed, missed/streak values must fit in the range\n";
  print STDERR " for all dose counts\n";
  print STDERR " missed is doses missed over the interval\n";
  print STDERR " missedHigh if specified is the high end of a range of doses missed\n";
  print STDERR " missedFuzzy if specified means process missed doses as equal to or greater than indicated value\n";
  print STDERR " default is to match missed doses exactly for each run\n";
  print STDERR " streak is number of doses missed in a row\n";
}
print STDERR "streakHigh if specified is the high end of a range of streaks to check\n";
print STDERR "streakFuzzy if specified means process any number of streaks within the eligible missed pattern\n";
print STDERR "default is to process missed patterns with only one matching missed streak\n";
print STDERR "\n\n";

if ( $fDebug ) {
    print STDERR "doses [\$dosages]\n";
    print STDERR "dosesHigh [\$dosagesHigh]\n";
    print STDERR "missed [\$missed]\n";
    print STDERR "missedHigh [\$missedHigh]\n";
    print STDERR "missedFuzzy [\$fMissedFuzzy]\n";
    print STDERR "streak [\$streak]\n";
    print STDERR "streakHigh [\$streakHigh]\n";
    print STDERR "streakFuzzy [\$fStreakFuzzy]\n";
    print STDERR "help [\$fHelp]\n";
    }
}