Problem: Solve the equation $z = (1 - \exp(-p\cdot z))/(p\cdot z)$ for $z \ge 0$ as a function of $p \ge 0$. In particular, we are interested in z when p is extremely tiny and roundoff corrupts the equation by introducing spurious roots z instead of the one true root $z = 1 - p/2 + 5p^2/12 - \dots$.

. . . .

To obtain numbers of reasonable size when p is tiny, we shall recast the equation in terms of the number $u := 1$ ulp of 1 , which is the difference between 1 and the floating–point number next less than 1 . Let $p := P\cdot u$. Now we seek the set of roots $z = Z(P)$ of the equation

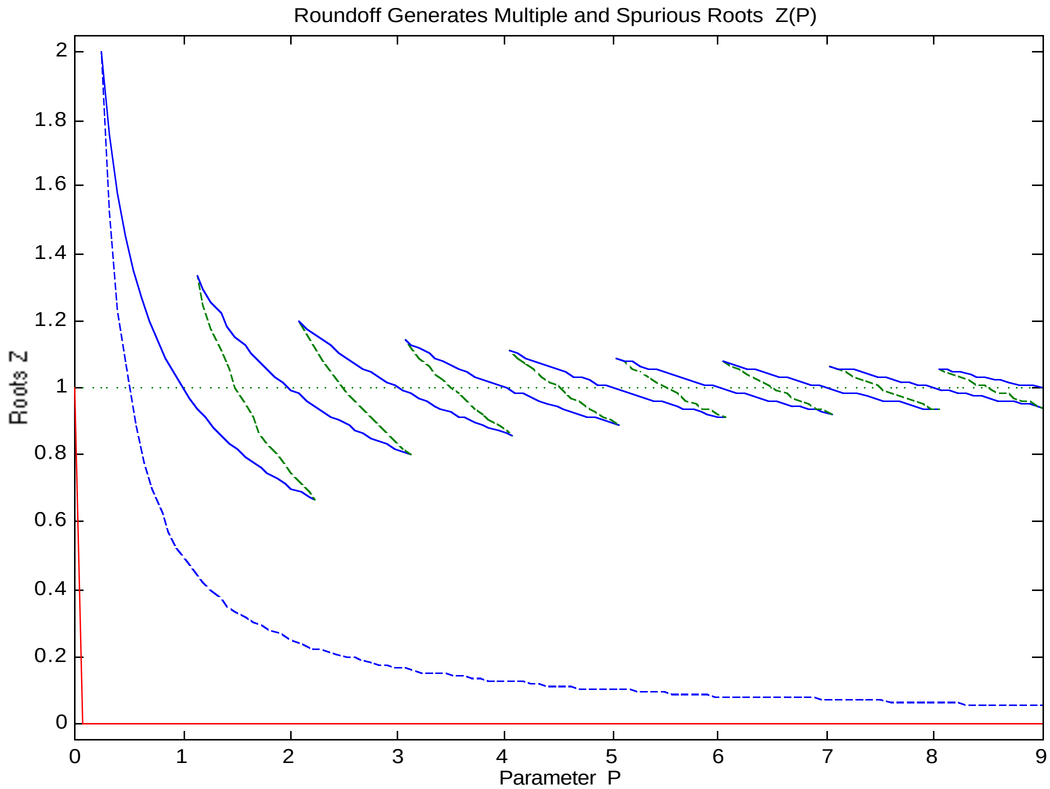
$$z = (1 - \text{Rounded}\big(\exp(-P\cdot z\cdot u)\big))/(P\cdot z\cdot u) .$$

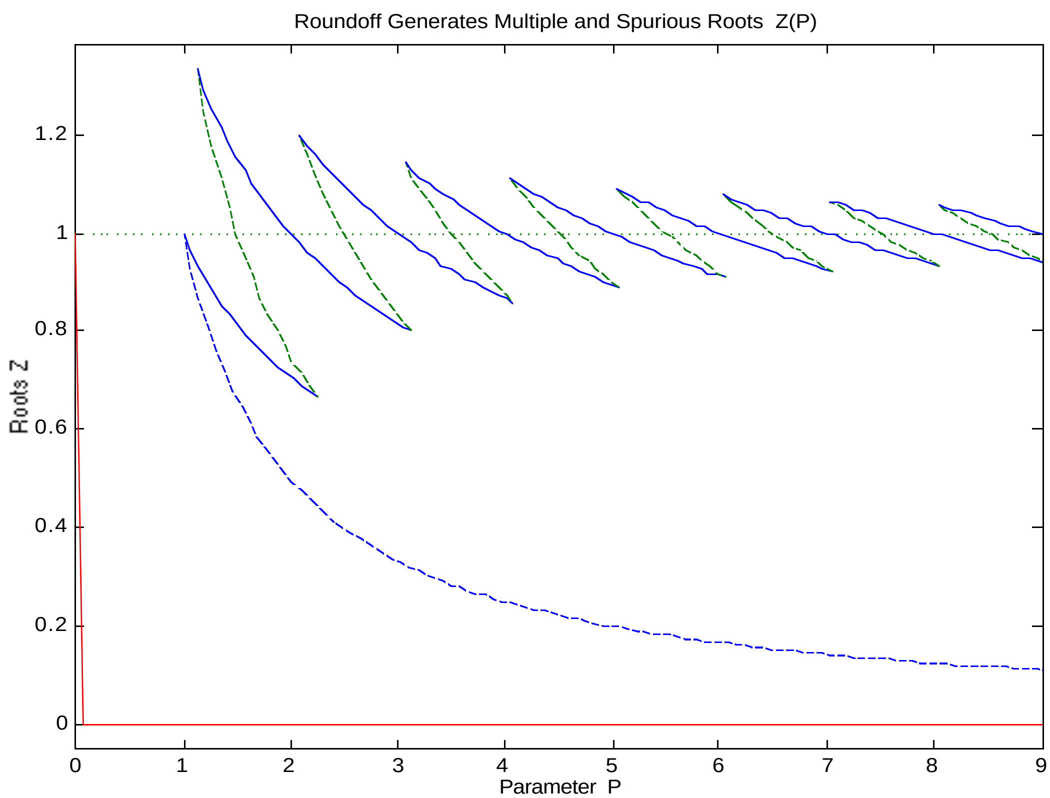
Next, for small integer values $k = 0, 1, 2, 3, \dots, 100$ in turn, define $y_{-1} := 0$ and y_k to satisfy " $\exp(-y\cdot u)$ rounds to $1 - k\cdot u$ throughout $y_{k-1} < y < y_k$." Were $\exp(\dots)$ correctly rounded we'd find $\text{Rounded}\big(\exp(-y\cdot u)\big) = 1 - k\cdot u$ just when $1 - (k+1/2)\cdot u < \exp(-y\cdot u) < 1 - (k-1/2)\cdot u$, which would determine $y_k = -\ln(1 - (k+1/2)\cdot u)/u \approx k + 1/2 + (k+1/2)^2\cdot u/2 + \dots$. In fact, the last equation merely approximates y_k because $\exp(\dots)$ is not quite correctly rounded; still, any respectable implementation of $\exp(\dots)$ should be monotonic in the sense that its rounded value does not decrease when its argument increases, so y_k should be well-defined and monotonic too: $0 = y_{-1} < y_0 < y_1 < y_2 < \dots$. These values have to be computed by applying binary chop to "solve" $(1 - \exp(-y_k\cdot u))/u = k + 1/2$ for y_k on your computer.

Roundoff in $\exp(\dots)$ turns the equation to be solved into $z = k/(P\cdot z)$ while $y_{k-1} < P\cdot z < y_k$. In other words, a root is $z = Z_k(P) := \sqrt{k/P}$ while $y_{k-1}^2/k < P < y_k^2/k$, and on that interval $Z_k(P)$ is a decreasing function: $k/y_{k-1} > Z_k(P) > k/y_k$. (The case $k = 0$ is a special case; $Z_0(0) = 1$ and $Z_0(P) := 0$ for all $P > 0$ although the equation involves $0/0$ then.) But numerical root-finders find more "roots" z generated by the jumps in the rounded values of $\exp(\dots)$ as follows:

Let ø stand for any sufficiently tiny positive number. Then $\text{Rounded}\big(\exp(-(y_k - ø)\cdot u)\big) = 1 - k\cdot u$ and $\text{Rounded}\big(\exp(-(y_k + ø)\cdot u)\big) = 1 - (k+1)\cdot u$. Therefore, while $x \approx (y_k \pm ø)/P$ we find that the computed value of $f(x) := x - (1 - \text{Rounded}\big(\exp(-P\cdot x\cdot u)\big))/(P\cdot x\cdot u)$ jumps down from very nearly $f((y_k - ø)/P) \approx y_k/P - k/y_k > 0$ to very nearly $f((y_k + ø)/P) \approx y_k/P - (k+1)/y_k < 0$ provided $y_k^2/(k+1) < P < y_k^2/k$. On this interval the sign-changing jump of $f(x)$ generates another spurious "root" $z = S_k(P) := y_k/P$ that also decreases monotonically: $(k+1)/y_k > S_k(P) > k/y_k$.

The graphs of Z_k and S_k on their respective intervals connect each other alternately to form a single zig-zag curve. See page 25 of "Personal Calculator Has Key to Solve Any Equation $f(x) = 0$." *Hewlett–Packard Journal* **30** #12 (Dec. 1979) pp. 20-26. (A scanned copy is at `http://www.cs.berkeley.edu/~wkahan/Math128/SOLVEkey.pdf` .) The following figure, produced by Matlab 5 on a µ68040-based Macintosh Quadra 950 shows true root $Z(P)$ as a nearly horizontal dotted line; the roots $Z_k(P)$ are shown solid red and blue, and $S_k(P)$ dashed or grey. The figure after that was produced by the same Matlab 5 program on a Power Mac 8500, whose $\exp(\dots)$ is less accurate; its missing legend on the left is a Matlab -> PICT -> PDF bug. On both computers, numerical root-finders can find as many as five "roots" instead of one.


Roundoff Generates Multiple and Spurious Roots Z(P)
Roots Z
Parameter P
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9


Roundoff Generates Multiple and Spurious Roots Z(P)
Roots Z
Parameter P
1.2
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9

```
function  y = solvezag(R, pts)
%  solvezag(R)  exhibits a zig-zag graph of the nonnegative roots  z  of the
%  equation  z = ( 1 - exp(-u*P*z) )/(u*P*z)  where  u = eps/2  and  0 < P < R
%  and roundoff corrupts the equation  ( mainly by corrupting  exp(...) ) .
%  Restriction:  2 < R < 100 .  y = solvezag(R)  returns a column of the first
%  several  ( about  R )  points  y  where  ( 1 - exp(-y*u) )/u  jumps;  they
%  should be very near the consecutive half-integers  0.5, 1.5, 2.5, ... .
%  solvezag(R, pts)  plots at a density of  pts/R  instead of  128/R .  And
%  R = 10  by default if omitted.
if  ( nargin < 2 ),  pts = 128 ;  end
if  ( nargin < 1 ),  R = 10 ;  end
if  (R<2)|(R>100),  error( ' solvezag( R  out of range )' ),  end
K = round(R) ;  y = yk(K) ;
h = R/pts ;  P = h*[0:pts]' ;  Pend = P(1+pts) ;
u = eps/2 ;  Zt = 1 - 0.5*u*P.*(1 - (5/6)*u*P) ;  % ...  Zt = true root.
Z0 = 0*P ;  Z0(1) = 1 ;  % ...  Z0 = degenerate root.
SP0 = [ y(1)^2 : h : Pend ]' ;  S0 = y(1)./SP0 ;  % ...  S0 = 1st spurious root
plot( SP0,S0,'--', P,Zt,'.',  P,Z0 )
ytop = max( [ S0(1), 2/y(2) ]) + 0.05 ;
axis( [0, R-1, -0.05, ytop ] ) ;  hold on ;
for  k = 1:(K-1) ,  % ...  superpose graphs of  "roots"  Zk  and  Sk .
     pl = y(k)*y(k)/k ;  pr = y(k+1)*y(k+1)/k ;  % ...  ends of range for  Zk
     prl = pr - pl ;  ptsk = round( prl/h ) + 1 ;
     ZPk = pl + (prl/ptsk)*[0:ptsk]' ;
        Zk = sqrt( k ./ ZPk ) ;
        pl = y(k+1)*y(k+1)/(k+1) ;  %  adjust left end of range for  Sk
        prl = pr - pl ;  ptsk = round( prl/h ) + 1 ;
     SPk = pl + (prl/ptsk)*[0:ptsk]' ;
        Sk = y(k+1) ./ SPk ;
        plot( ZPk,Zk, '-',  SPk,Sk, '--')
   end  % ...  k
hold off ,  xlabel(' Parameter  P'),  ylabel(' Roots  Z' )
title( ' Roundoff Generates Multiple and Spurious Roots  Z(P) ')

function  y = yk(K)
%  yk(K) = [ yc(0.5), yc(1.5), yc(2.5), ..., yc(K+0.5) ]'  for  yc  below,
%  and for  nonnegative integer  K < 101 .
k = round(K)+1 ;
if  (k<1)|(k>101),  error('  yk( K  out of range )'),  end
y = zeros( k, 1) ;
for  j = [1:k]
      y(j) = yc(j - 0.5) ;
   end

function  y = yc(c)
%  yc(c) = solution  y  of  ef(y) = c ,  which see below,  by binary chop.
y = c-1.25 ;  fl = ef(y)-c ;  if  fl==0,  return,  end
yl = y ;
y = c+1.25 ;  fr = ef(y)-c ;  if  fr==0,  return,  end
yr = y ;
if  fl*fr > 0 ,  error('Oops!  Missed the sign reversal!'), end
y = (yl + yr)*0.5 ;
while  (y ~= yl) & (y ~= yr)
     f = ef(y)-c ;  if  f==0,  return,  end
        if  f*fr > 0 ,  yr = y ;  fr = f ;
               else      yl = y ;  fl = f ;  end
     y = (yl + yr)*0.5 ;
  end

function  y = ef(x)
%  ef(x) = (1 - exp(-u*x))/u   where  u = eps/2 .
u = eps/2 ;
y = (1 - exp(-u*x))/u ;
```