Supplementary information:

Quantal Basis of Secretory Granule Biogenesis and Inventory Maintenance, the Surreptitious Nano-machine Behind It

Ilan Hammel[1] and Isaac Meilijson[2]

1 Sackler Faculty of Medicine, Department of Pathology, Tel Aviv University, Tel Aviv 6997801, Israel

2 Raymond and Beverly Sackler Faculty of Exact Sciences, School of Mathematical Sciences, Department of Statistics and Operations Research, Tel Aviv University, Tel Aviv 6997801, Israel

E-mail address:

Ilan Hammel ilanh@patholog.tau.ac.il

Isaac Meilijson isaco@math.tau.ac.il

Correspondence may be addressed to either of the two authors.

Telephone and fax:

Ilan Hammel: +972-3-640-8408 (Tel) +972-3-640-9141 (Fax)

Isaac Meilijson: +972-3-640-8826 (Tel) +972-3-640-9357 (Fax)

Supplementary information chapters: **Page Number**

Program lists:

1 The time at which simple random walk achieves a given draw-up or reaches a given level. The CUSUM method and integrate-and-fire neuronal management are so fundamental that we opt for illustrating first the ideas involved on a stochastic case where Markovian recursive techniques can be elementarily applied, yet yield exact answers.

2 The time to reach a level. Consider (the simple) random walk starting at zero, whose increments X_i are either +1 (with probability p>1/2) or -1 (with the complementary probability 1-p). Let T be the first hitting time of +1. This time is 1 plus a remainder. If the first increment is +1, the remainder is zero. If -1, the random walk is at level -1 and the remainder is distributed like the sum of two independent copies of T, the time to re-reach zero from -1 plus the time to reach +1 from 0. Hence, **T has the same distribution as *1 + J∗(T1+T2)***, where J is the obvious indicator. This leads to *E[T] = 1+E[J[]∗2∗E[T] = 1+2∗(1-p)∗E[T]*, or equivalently ***E[T] = 1/(2∗p-1) = 1/E[increment]***. Now we appeal to the method of partitioning variance (the same as calculating the moment of inertia of a body composed of sub-bodies) as *Var[T] = E[Var[T|J]] + Var[E[T|J]]* to evaluate *Var[T]* from E[T|J] = 1+J∗2∗E[T] and Var[T|J] = (J^2)∗2∗Var[T] as Var[T] = 2∗Var[T]∗(1-p) + 4∗$E[T]^2$∗p∗(1-p). *I.e., (2∗p-1)∗Var[T] =* $4*E[T]^2*p*(1-p)$ *or* **Var[T] = 4∗p∗(1-p)/$(2*p-1)^3$ = Var[increment]/$(E[increment])^3$**. Clearly, then, the mean and variance of the time T_n to reach some positive integer threshold n are $E[T_n]=n*E[T]=n/E[increment]$ and $Var[T_n]=n*Var[T]= n*Var[increment]/ (E[increment])^3$ respectively.

After this elementary but exact illustrative derivation, here is a general argument. If S_m is a random walk starting at zero, whose increments X_i have mean M and variance V, then the mean-zero processes $S_m – m*M$ and $(S_m – m*M)^2 – m*V$ are *Martingales*, and as such preserve the mean value zero when deterministic time m is replaced by random stopping time τ (with some restrictions). Let τ_n be the first time m when $S_m \geq n$ (which we approximate by $S_m = n$ ignoring the excess). Then $0 \approx n - E[\tau_n]*M$ and $0 = n^2 - 2*n*E[\tau]*M + E[\tau_n^2]*M^2 - E[\tau_n]*V$, from which the earlier presented formulas $E[\tau_n] = n/M$ and $Var[\tau_n] = n*V/M^3$ are recovered.

3 The time to achieve a draw-up. Returning to simple random walk, let DU_n be the fists time at which the random walk is n units above its minimal value so far. For example, DU_1 is the first time the increment is +1, a geometrically distributed random variable with mean $E[DU_1] = 1/p$, bounded from above by 2 and much smaller than $E[T_1] = 1/(2*p-1)$ for p just above ½, even if we expect $E[DU_n]$ and $E[T_n]$ to become close for large n. Here is an argument for an easy exact evaluation of the sequence $A_n = E[DU_n]$. It takes mean time A_{n-1} to attain draw-up $n-1$, at which time the random walk is at the upper end MIN+n-1 of the interval it has visited *[MIN MIN+n-1]=[minimum_value current_value]*. The random walk now proceeds (for a length of time whose mean we denote by S_{n-1}) until reaching *MIN* or *MIN+n*, whichever comes first. If *MIN+n* is reached first, the draw-up is n. If *MIN* is reached first, the quest for draw-up n starts all over. Let Q_{n-1} be the probability that the random walk (starting at zero playing the role of *MIN+n-1*) will reach level +1 before reaching level $-(n-1)$. By the above argument, $A_n = A_{n-1}+S_{n-1}+(1-Q_{n-1})*A_n$ so $A_n = (A_{n-1}+S_{n-1})/Q_{n-1}$. This is a recursive formula for the sequence A based on the sequences S and Q, much quoted under the name *Gambler's Ruin Problem*: A gambler with initial fortune k bets $1 at a move in Red & Black until reaching a goal N $(>k)$ or going broke. If the winning probability per move is p, then the probability of reaching the goal is $P_k = (1-((1-p)/p)^k)/(1-((1-p)/p)^N)$ and the expected number of moves of the entire game is $(N*P_k-k)/(2*p-1)$.

In our application, $N=n$ and $k=n-1$ so $Q_{n-1} = (1-((1-p)/p)^{n-1})/(1-((1-p)/p)^n)$ and $S_{n-1} = (n*Q_{n-1}-n+1)/(2*p-1)$. *Now the recursive relation* $A_n = (A_{n-1}+S_{n-1})/Q_{n-1}$ *can be re-written as* $(A_n-n/(2*p-1))/(1-((1-$

$p)/p)^n) = (A_{n-1}-(n-1)/(2*p-1))/(1-((1-p)/p)^{n-1})$, *i.e.,* it has the same value for all n (!). Hence, it is equal to its value at $n=1$: $(A_1-1/(2*p-1))/(1-((1-p)/p)^1) = ((1/p)-1/(2*p-1))/(1-((1-p)/p)^1) = -(1-p)/(2*p-1)^2$. *Finally,*

$$A_n = n/(2*p-1) + ((1-p)/(2*p-1)^2)*((1-p)/p)^n - (1-p)/(2*p-1)^2 .$$

If $p>1/2$, A_n grows indeed *linearly* like $n/(2*p-1)$ but if $p<1/2$, A_n grows *exponentially* like $((1-p)/p)^n$. This is the rationale for the CUSUM statistic method of detection: By a minor delay in the detection of the post-change mode, it is possible to reduce drastically the rate of false detection.

4 The general case: Lundberg's bound for the expected time to a draw-down. Let the increments V of a random walk have positive mean E[V] and *adjustment coefficient* α (the unique positive number α for which $E[exp(-\alpha*V)]=1$, a useful concept in actuarial risk theory (Asmussen 2000). Its inverse $1/\alpha$ is termed *index of riskiness* of V by Aumann and Serrano 2008). Then (Asmussen 2000, Meilijson 2009) the expected maximal height achieved by the random walk prior to experiencing draw-down DD is at least $(exp(\alpha*DD)-1)/\alpha$. Subtract DD to obtain the expected height upon achieving drawn-down DD and divide by the expected increment to convert height to time, to obtain the lower bound $(exp(\alpha*DD)-1- \alpha*DD)/(\alpha*E[V])$ for the expected time to achieve draw-down DD. This bound is quite tight: Meilijson (2009) found yet a third constant $C>1$ defined by the distribution of X such that the expected time is *at most* $(C*exp(\alpha*DD)-1- \alpha*DD)/E[V]$. Log likelihood ratio random walks are special: if X has density f and $V=-log(g(X)/f(X))$ then $\alpha=1$, whatever the densities f and g are. For any such case, then, the expected basal-mode number of granules to a draw-up DU is approximately $exp(DU)/KLD(F,G)$. Hence, if this expected number is to be MGFA (mean granules to a false alarm), then $DU\approx log(MGFA*KLD(F,G))$. Switching now hats from basal to evoked mode and applying a formula above (in this Appendix), we obtain the formula *MGD = E[T_{DU}] ≈ DU/E[increment]=DU/KLD(G,F) ≈ log (MGFA*KLD(F,G))/KLG(G,F)* stated in Section 4.

Appendix References

Asmussen S. (2000) Ruin Probabilities. Advanced Series on Statistical Science & Applied Probability. World Scientific, River Edge, NJ. MR1794582

Aumann JR, Serrano R. An economic index of riskiness. *J Polit Econo*. 2008;**116**:810–836.

Meilijson I. On the adjustment coefficient, drawdowns and Lundberg-type bounds for random walk. *Ann Appl Probab*. 2009;**19**:1015-1025.

```
% HM2014_simuLIFO.m by Ilan Hammel and Isaac Meilijson, June 2014

% This program simulates unit-addition quantal granule dynamics.
% It should be preceeded by HM2014_mixedsecret.m
% input batchsize (or uncooment line 38)
% input simulation sample size ITER
% maximal quantal size QN, desired fraction evoksecret of evoked
% secretion out of the list in HM2014_mixedsecret.m, where
% evoksecret=1 means pure basal secretion and evoksecret=2 is the
% smallest fraction in the menu, etc.
% Example: batchsize=25;QN=20;evoksecret=3;ITER=1000000;

% Granule quantal size is from 1 to QN. In order to induce a finite
% representation of states, polymers of size QN can't grow. Let QN be
% large enough so that size QN is in practice not reached.

% NEXT EVENT SIMULATION dynamics are as follows:
% We list all MM events that can happen and write down the rate of
each,
% r(1), r(2), ..., r(MM), with R=sum r(i) and pp(i)=r(i)/R. Then the
next
% event will occur at an exponentially distributed time with
parameter R.
% This time is simulated as -(1/R)*log(U1) where U1 is U[0,1]
distributed,
% and the name of the event is i if
% pp(1)+...+pp(i-1)<U2<=pp(1)+...+pp(i-1)+pp(i), where U2 is U[0,1]
% distributed, independent of U1.
% The state is updated after each event.

% The "events that can happen" are
% ARRIVAL (addition of one monomer to state(1)).
% EXIT from any polymer size present. (Subtract 1 from state(.))
% GROWTH at any polymer size present except QN. (Subtract 1 from
state(.)
% and add 1 to state(.+1))

% The vector V is the "standard", constant rate vector given by ARR
rate
% and the terms lam*n^beta and mu*n^gamma. This vector corresponds to
the
% columns of M. The actual rates are given by the products
M(i,n)*V(n).

% The rest is book-keeping
clear count count1 countex countex1 ctime dtime evoktime evoktrig
clear hcount hrates hstate lam mu name namec probs  propnew rates
```

```
clear state statec
musim=paramix(evoksecret,2);
mu=musim;
lamsim=paramix(evoksecret,1);
lam=lamsim;
%batchsize=20;
arriv1=0;arriv2=0;arriv3=0;
bet=-(2/3)*(Kbeta-1);gam=-(2/3)*(Kgamma-1);
state=zeros(1,QN);
time=0;
% propevok in 38 and paramixexit in 40 can be adjusated to a desired
evoked
% secretion fraction
propevok=evokprop(evoksecret);
eps2=-log(1-1/batchsize);
eps1=(propevok/batchsize)/(1+paramixexit(evoksecret));
ITER1=ITER+100;
count=zeros(1,QN);
ctime=zeros(ITER,1);
statec=zeros(ITER,QN);
count1=count;
evoktime=zeros(ITER,6);
namec=ctime;
dtime=ctime;
evoktrig=(rand(ITER,1)<eps1);
DURAT=0;NURAT=0;
for i=1:ITER
    if evoktrig(i)==1
        durat=1+fix(-
log(rand(1,1))/eps2);DURAT=DURAT+durat;NURAT=NURAT+1;
        evoktime(i:i+durat-1)=ones(durat,1);
    end
end
REX=mu*((1:QN).^gam);
REX1=nu*(1-propevok)*(ones(1,QN));
RGR=lam*(1:QN).^bet;
RAR=nu;RGR(QN)=0;
for i=1:ITER
    rates=[RAR*(i<ITER1)*(1-evoktime(i)) (REX+(1000*REX1/sum(state)-
REX)*evoktime(i)).*state (i<ITER1)*RGR.*state*(1-evoktime(i))];
    rate=rates*ones(2*QN+1,1);
    probs=cumsum([rates/rate]);
    U=rand;V=rand;
    namec(i)=1+sum((U>probs));
    name=namec(i);
    if i>100
        dtime(i)=-(1/rate)*log(V);
```

```
        ctime(i)=ctime(max(1,i-1))+dtime(i)*(1-0*evoktime(i));
    end
    if name==1
        state(1)=state(1)+1;
        arriv1=arriv1+1;
    elseif name<QN+2
        state(name-1)=max(0,state(name-1)-1);
        count(name-1)=count(name-1)+1;
        count1(name-1)=count1(name-1)+evoktime(i);
    elseif name>QN+1 & name<2*QN+1
        state(name-1-QN)=max(0,state(name-1-QN)-1);
        state(name-QN)=state(name-QN)+1;
        arriv2=arriv2+1;
    end
statec(i,:)=state;
end
countex=count/sum(count);
countex1=count1/sum(count1);

hstate=zeros(QN+1,QN);hcount=hstate;
hstate(1,:)=state;
for ii=1+ITER:ITER1+ITER
    hrates=[[0 1 zeros(1,QN-1)]'*RAR (ones(QN+1,1)*REX).*hstate
(ones(QN+1,1)*RGR).*hstate];
    hrate=ones(1,QN+1)*hrates*ones(2*QN+1,1);
    hprobs=[hrates/hrate];
    U=rand;PP=0;QQ=0;
    V=rand;dtime(ii)=-(1/hrate)*log(V);
    %ctime(ii)=ctime(max(1,ii-1))+dtime(ii);
    for i=1:QN+1
        for j=1:2*QN+1
            PP=PP+hprobs(i,j);
            if U<PP & QQ==0
                QQ=1;i0=i;j0=j;
            end
        end
    end
    if j0==1
        hstate(2,1)=hstate(2,1)+1;
        arriv1=arriv1+1;arriv3=arriv3+1;
    elseif j0<QN+2
        hstate(i0,j0-1)=hstate(i0,j0-1)-1;
        hcount(i0,j0-1)=hcount(i0,j0-1)+1;
    elseif j0>QN+1
        hstate(i0,j0-1-QN)=hstate(i0,j0-1-QN)-1;
        hstate(i0+1,j0-QN)=hstate(i0+1,j0-QN)+1;
        arriv2=arriv2+1;arriv3=arriv3+1;
```

```
    end
    sum2=sum(sum(hstate.*(((1:QN+1)-1)'*ones(1,QN))));
    sum1=sum(sum(hstate.*(ones(QN+1,1)*(1:QN))));
    propnew(ii-ITER)=sum2/sum1;
end

megrsi=sum((1:QN).*(mean(statec))/sum(mean(statec)));
countex=count/sum(count);

Ba='SIZE   ';
Bb='STAT   ';
Bc='EXIT   ';

%if jk==1
    disp(' ')
    disp('Empirical stationary and exit distributions')
    disp('-------------------------------------------')
    disp([vertcat([Ba ; Bb ; Bc]) num2str([(1:QN) ;
(mean(statec))/sum(mean(statec)) ; countex])])
    disp(' ')
    disp('   3600mu   3600lambda secr-rate  Golgi-rate Kbeta
Kgamma  mean-gr-si  burst size')
    disp([3600*mu   3600*lam   nu    [arriv1+arriv2]/max(ctime) 1-
1.5*bet 1-1.5*gam megrsi batchsize])
    %disp('-----------------------------------')
%end
%clear
figure(3),figure(3),plot(ctime(1:ITER)/(3600*24),statec),grid,zoom
%clear
figure(2),figure(2),plot(ctime(1:ITER)/(3600*24),sum(statec')'),grid,
zoom
%clear
figure(2),figure(2),plot(ctime(1:ITER)/(3600*24),[sum(statec')'
statec]),grid,zoom

%
QN=20;Kbeta=5;Kgamma=4;mol=4;nu=1;N=2000;batchsize=25;evoksecret=3;IT
ER=1000000;HM2014_mixedsecret,HM2014_simuLIFO
% figure(1),semilogx(ctime/3600,statec),grid,zoom
```

```
% HM2014_mixedsecret.m by Ilan Hammel and Isaac Meilijson, June 2014
%input mol (mu over lambda)
%      nu (birth rate of mature unit granules = secretion rate)
%      Kbeta and Kgamma
%      N (cell mean content in steady state)
% in-program: evokprop (proportion of evoked secretion in total
secretion)
clear basalexit basalstat dfg evokprop lam lamix meansjn
clear mixexit mumix mumixraw mustat NN numix nustat paramix
clear paramixexit sjn1 sjn2 surviv survival stdsjn
TOP=50; % maximal quantal size
KKK=5; % number of options for fraction of evoked secretion
mixexit=zeros(KKK,TOP);
nustat(1)=nu;
survival(1:TOP+1,1)=ones(TOP+1,1);
surviv(1)=1;
mustat=mol*(1:TOP).^((-2/3)*(Kgamma-1));
lam=(1:TOP).^((-2/3)*(Kbeta-1));
for i=1:TOP
    nustat(i+1)=nustat(i)*lam(i)/(lam(i)+mustat(i));
    NN(i)=nustat(i)/(lam(i)+mustat(i));
    surviv(i+1)=surviv(i)*lam(i)/(lam(i)+mustat(i));
    basalexit(i)=surviv(i)-surviv(i+1);
end
lam=lam*sum(NN)/N;
mustat=mustat*sum(NN)/N;
NN=NN*N/sum(NN);
basalstat=NN/N;
lamix=lam;
mumixraw=mustat;
for k=1:KKK,evokprop(k)=2^(k-2)/100;
    evokprop(1)=0;
mumix=mumixraw+nu*evokprop(k)/sum(NN);
numix(1)=nu;
for i=1:TOP
    numix(i+1)=numix(i)*lamix(i)/(lamix(i)+mumix(i));
    NN(i)=numix(i)/(lamix(i)+mumix(i));
end
I=1;
while sum(NN)<N
    I=I+1;S1=sum(NN);
    lamix=lamix*.999;mumixraw=mumixraw*.999;
    mumix=mumixraw+nu*evokprop(k)/sum(NN);
for i=1:TOP
    numix(i+1)=numix(i)*lamix(i)/(lamix(i)+mumix(i));
    NN(i)=numix(i)/(lamix(i)+mumix(i));
    survival(k,i+1)=survival(k,i)*lamix(i)/(lamix(i)+mumix(i));
```

```
    mixexit(k,i)=survival(k,i)-survival(k,i+1);
    mixexit(1,i)=surviv(i)-surviv(i+1);
    sjn1(k,i)=sum(1./(lamix(1:i)+mumix(1:i)));
    sjn2(k,i)=sum((1./(lamix(1:i)+mumix(1:i))).^2);
    sjn1(1,i)=sum(1./(lam(1:i)+mustat(1:i)));
    sjn2(1,i)=sum((1./(lam(1:i)+mustat(1:i))).^2);
end
end
S2=sum(NN);
mixstat(k,:)=NN/sum(NN);
paramix(k,:)=[lamix(1) mumixraw(1) mixstat(k,:)*(1:TOP)'];
paramixexit(k)=mixexit(k,:)*(1:TOP)';
end
top=sum((N*basalstat>.05));
% Kullback-Leibler Divergence - expected increments of random walk
kldstex=sum(mixstat(1,1:top).*log(mixstat(1,1:top)./mixexit(1,1:top))
);
kldexst=sum(mixexit(1,1:top).*log(mixexit(1,1:top)./mixstat(1,1:top))
);
% Second moments of increments of random walk
kld2stex=sum(mixstat(1,1:top).*(log(mixstat(1,1:top)./mixexit(1,1:top
))).^2);
kld2exst=sum(mixexit(1,1:top).*(log(mixexit(1,1:top)./mixstat(1,1:top
))).^2);
% Variance of increments of random walk
varkldstex=kld2stex-kldstex^2;
varkldexst=kld2exst-kldexst^2;

if jk==1
disp('      ')
disp('    mu/lambda nu (p/sec) Kbeta   Kgamma  KLD(St,Ex) KLD(Ex,St)
STD(St,Ex)')
disp('    ------------------------------------------------------------
----------')
disp([mol nu Kbeta Kgamma kldstex kldexst sqrt(varkldstex)])
disp('      ')
disp('Stat distr under menu of evoked fraction out of total
secretion')
disp('----------------------------------------------------------------
')
disp('evoked fraction 0%        1%        2%         4%         8%')
disp([(1:top) ; mixstat(:,1:top)]')
disp('      ')
disp('Exit distr under menu of evoked fraction out of total
secretion')
disp('----------------------------------------------------------------
')
```

```
disp('evoked fraction 0%        1%        2%         4%        8%')
disp([(1:top) ; mixexit(:,1:top)]')
end

mgsbasalstat=basalstat*(1:TOP)';
mgsbasalmix=mixstat*(1:TOP)';
mgsbasalexit=basalexit*(1:TOP)';
paramixexit(1)=mgsbasalexit;
mgsexitmix=mixexit*(1:TOP)';
meansjn=sum(mixexit'.*sjn1');
stdsjn=sqrt(sum(mixexit'.*sjn2'));
Aa='lambda p/grn p/hour   ';
Ba='mu     p/grn p/hour   ';
Da='mean gr size stat     ';
Ea='mean gr size exit     ';
Fa='mean sojourn (hours)  ';
Ga='stdv sojourn (hours)  ';
%if jk==1
disp('lambda and mu (p/grn p/hour), mean stat and exit grn size, mean
and std of sojourn (in hours)')
disp('---------------------------------------------------------------------
---------------------------')
disp('evoked fraction         0%           1%          2%
4%          8%')
disp([vertcat([Aa ; Ba ; Da ; Ea ; Fa ; Ga]) num2str(diag([3600 3600
1 1 1/(3600) 1/(3600)])*[[paramix paramixexit' meansjn' stdsjn']]')])
%end
disp('---------------------------------------------------------------------
---------------------------')
dfg=[kldstex sqrt(varkldstex) kldexst sqrt(varkldexst)];
mgd1=log(10^5*(6.6974+kldexst))/(1.4026+kldstex);
stdgd1=sqrt(mgd1*(varkldstex+.9^2)/(1.4026+kldstex)^2);
mgd2=log(10^5*(94.3948+kldexst))/(3.6152+kldstex);
stdgd2=sqrt(mgd2*(varkldstex+.99^2)/(3.6152+kldstex)^2);
disp('Mean, STD and burst size to detect evoked state')
disp('    MGD_R10   STD_R10  MGD+2*STD MGD_R100  STD_R100 MGD+2*STD')
disp([mgd1 stdgd1 mgd1+2*stdgd1 mgd2 stdgd2 mgd2+2*stdgd2])
% Example of input:
% mol=4;nu=1;Kbeta=7;Kgamma=7;N=2000;
```