FinM 331/Stat 339 Financial Data Analysis, Winter 2010

Floyd B. Hanson, Visiting Professor

Email: fhanson@uchicago.edu

Master of Science in Financial Mathematics Program
University of Chicago

Lecture 6

6:30-9:30 pm, 08 February 2010, Ryerson 251 in Chicago

7:30-10:30 pm, 08 February 2010 2010 at UBS in Stamford

7:30-10:30 am, 09 February 2010 at Spring in Singapore

6. *More Maximum Likelihood Estimation, But First Some Problem Review:*

- ***6.1PS: Lecture5-page28, Revison of `Robustfit` Results for WLS Fit Example with Normal-Poisson Error:***

Since is seemed that **`MATLAB's robustfit`** function was not working as well as advertised, we will revise the implementation of the weighting functions with the non-default form;

```
b=robustfit(x,y,wfun,tune);
```
(6.1)

where as we previously used the default form,

```
[prob,statrob] = robustfit(x,y);
```
(6.2)

in which the **scaled residual weight function `wfun='bisquare'`**,

$$w = (\text{abs}(r) < 1).*(1 - r.^2).^2; \tag{6.3}$$

where

$$r = \text{resid}/(\text{tune}*s*\sqrt{1-h}); \tag{6.4}$$

`h` is the hat function (leverage) vector, with a tuning value **`tune=4.685`**. This means that the outlier residuals $(\text{abs}(r) \geq 1)$ had zero weight.

So, a revised computation uses the weight function **wfun='fair'**,

$$w = 1./(1 + \mathbf{abs}(r)); \tag{6.5}$$

which has unrestricted range, with tuning parameter selected higher at **tune=2*4.685** and lower at the **'fair'** default **tune=1.4** for two contrasting cases. Also, a larger sample size of **(**n=500) replaces the prior size of **(**n=100). The two constrasting **'fair'** cases are closer to the OLS **regress** fit, but all are below the true straight line.

Another problem seemed to be some bias due to the fact that the zero mean, constant variance jump-diffusion noise was not symmetric, in that the jumps were all positive. This was changed to up and down jumps using compound Bernoulli-Poisson noise,

$$e_i = \sigma_e * \mathtt{randn} + \nu_e * \mathtt{unifrnd}(-1, +1) * \mathtt{binornd}(1, \Lambda_e); \tag{6.6}$$

and this seemed to make all statistical code have similar results. You can check that $\mathbf{E}[e_i] = 0$ and $\mathbf{Var}[e_i] = \sigma_e^2 + \nu_e^2 \Lambda_e / 3$.

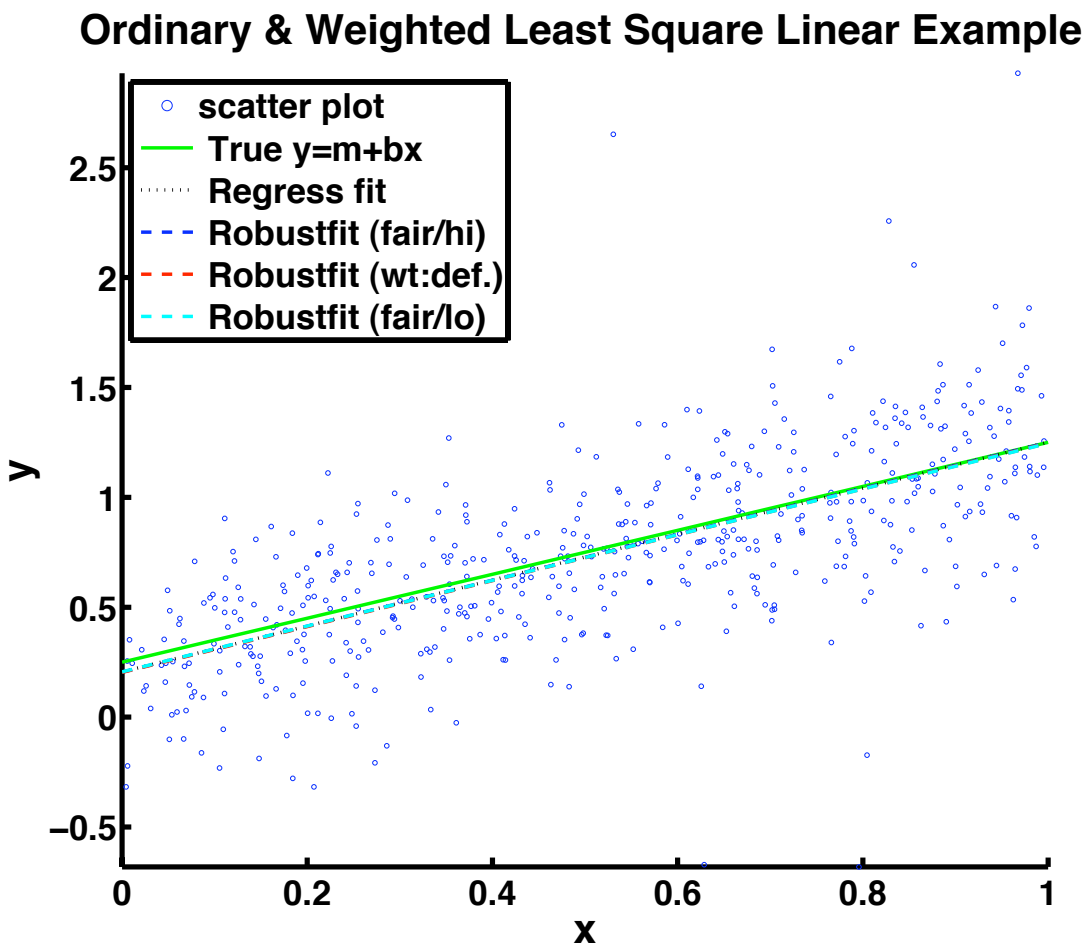

Figure 6.1: Revised figure for weighted least squares **`robustfit`**, with similar results for a straight line fit with **`scatterplot`** data (**`n=500`**) from **combined normal-Poisson random error**. A sample plot was chosen from several simulations that was not as good matching the true value.

○ *MATLAB Code for Ordinary & Weighted Least Squares Application with Normal-Poisson Error:*

```
function RegRobtest2
clc
fprintf('\nRegRobtest2 Output (%s):',datestr(now));
m = 1; b = 0.25; % True parm values;
fprintf('\nTrue: b = %7.4f; m = %7.4f;',b,m);
ptrue = [b;m];
n = 500;
fprintf('\nSample: n = %i;',n);
x = rand(n,1); % Simulated Uniform x-data;
sigma_e = 0.30; nu_e = 1.5; Lambda_e = 0.05;
% JD-Simulation Zero-Mean error:
err = sigma_e*randn(n,1) ...
    + nu_e*(poissrnd(Lambda_e,n,1)-Lambda_e*ones(n,1));
fprintf('\nsigma_e = %7.4f; nu_e = %7.4f; Lambda_e = %7.4f; var_e = %7.4f'...
    ,sigma_e,nu_e,Lambda_e,sigma_e^2+Lambda_e*nu_e^2);
errmean = mean(err); errvar = var(err);
fprintf('\nmean(err) = %9.3e; var(err) = %9.3e;',errmean,errvar);
y = b+m*x+err; % Simulated y-data with linear model;
TSS = (n-1)*var(y);
fprintf('\nTSS = %9.3e',TSS);
A = [ones(n,1) x];
fprintf('\nsize(A)=[%i,%i]; size(y)=[%i,%i];',size(A),size(y));
```

```
[preg,pci,res,resci,statreg] = Regress(y,A); % Multlinear Regression;
breg = preg(1); mreg = preg(2);
yhatreg = breg+mreg*x;
fprintf('\nRelResReg =%9.3e;',sqrt(norm(y-yhatreg)/norm(y)));
[probdef] = robustfit(x,y); % Def. Weighted Least Sqs Method;
[prob,statrob] = robustfit(x,y,'fair',9.370); % WLS Method, deftune*2;
[problo] = robustfit(x,y,'fair',2.3425); % WLS, deftune/2;
yhatrob = prob(1)+prob(2)*x;
fprintf('\nRelResRob =%9.3e;',sqrt(norm(y-yhatrob)/norm(y)));
fprintf('\nTrue:      b =%7.4f; m =%7.4f;',b,m);
fprintf('\nRegress:   b =%7.4f; m =%7.4f;',breg,mreg);
fprintf('\nRobustfit: b =%7.4f; m =%7.4f;',prob(1),prob(2));
fprintf('\nsqrt(norm(preg-ptrue)/norm(ptrue)) =%7.4f;' ...
    ,sqrt(norm(preg-ptrue)/norm(ptrue)));
fprintf('\nsqrt(norm(prob-ptrue)/norm(ptrue)) =%7.4f;' ...
    ,sqrt(norm(prob-ptrue)/norm(ptrue)));
fprintf('\nsqrt(norm(preg-prob)/norm(prob))   =%7.4f;' ...
    ,sqrt(norm(preg-prob)/norm(prob)));
SSEreg = sum((y-yhatreg).^2); % Also, RSS=ResSumSqs
SSErob = sum((y-yhatrob).^2); % Also, RSS=ResSumSqs
fprintf('\nRSS: SSEreg =%9.3e; SSErob =%9.3e;',SSEreg,SSErob);
Rsqreg = 1-SSEreg/TSS; Rsqrob = 1-SSErob/TSS;
fprintf('\nRsqreg =%9.3e; Rsqrob =%9.3e;',Rsqreg,Rsqrob);
fprintf('\nstatreg: R^2 =%7.4f; F = %7.4f;' ...
```

```
    ,statreg(1,1),statreg(1,2));
fprintf(' P-value)(F) =%7.4f; Var(error) =%7.4f;' ...
    ,statreg(1,1),statreg(1,1));
fprintf('\nstatrob Sigmas: OLS_s=%7.4f; Robust_s=%7.4f;' ...
    ,statrob.ols_s,statrob.robust_s);
fprintf(' MAD_s=%7.4f; final_s=%7.4f;' ...
    ,statrob.mad_s,statrob.s);
fprintf('\nstatrob: SE_p = [%7.4f; %7.4f];',statrob.se);
fprintf('\nstatrob: Corr_ps=[%7.4f, %7.4f; %7.4f, %7.4f];' ...
    ,statrob.coeffcorr);
statrob,
%
xg = 0:0.1:1;
ytrue = b+m*xg;
yhatregg = breg+mreg*xg;
yhatrobdefg = probdef(1)+probdef(2)*xg;
yhatroblog = problo(1)+problo(2)*xg;
yhatrobg = prob(1)+prob(2)*xg;
%
figure(1); nfig = 1;
scrsize = get(0,'ScreenSize'); % figure spacing for target screen
ss = [5.0,4.5,4.0,3.5]; % figure spacing factors
scatter(x,y,8); hold on;
plot(xg,ytrue,'-g',xg,yhatregg,':k',xg,yhatrobg,'--b' ...
    ,xg,yhatrobdefg,'--r',xg,yhatroblog,'--c','LineWidth',2);
```

```
axis tight; hold off;
title('Ordinary & Weighted Least Square Linear Example:'...
    ,'Fontsize',24,'FontWeight','Bold');
xlabel('x','Fontsize',24,'FontWeight','Bold');
ylabel('y','Fontsize',24,'FontWeight','Bold');
legend('scatter plot',' True y=m+bx',' Regress fit' ...
    ,' Robustfit (fair/hi)',' Robustfit (wt:def.)' ...
    ,' Robustfit (fair/lo)','Location','NorthWest');
set(gca,'Fontsize',20,'FontWeight','Bold','LineWidth',3);
set(gcf,'Color','White','Position' ...
   ,[scrsize(3)/ss(nfig) 60 scrsize(3)*0.60 scrsize(4)*0.80]); %[l,b,w,h]
fprintf('\n ');
================ OUTPUT ======================
RegRobtest2 Output (02-Feb-2010 18:34:00):
True: b =  0.2500; m =  1.0000;
Sample: n = 500;
sigma_e =  0.3000; nu_e =  1.5000; Lambda_e =  0.0500; var_e =  0.2025
mean(err) = -2.947e-02; var(err) = 2.208e-01;
TSS = 1.535e+02
size(A)=[500,2]; size(y)=[500,1];
RelResReg =7.185e-01;
RelResRob =7.189e-01;
True:      b = 0.2500; m = 1.0000;
Regress:   b = 0.2228; m = 0.9955;
Robustfit: b = 0.1993; m = 1.0036;
```

```
sqrt(norm(preg-ptrue)/norm(ptrue)) = 0.1636;
sqrt(norm(prob-ptrue)/norm(ptrue)) = 0.2221;
sqrt(norm(preg-prob)/norm(prob))   = 0.1558;
RSS: SSEreg =1.102e+02; SSErob =1.104e+02;
Rsqreg =2.822e-01; Rsqrob =2.810e-01;
statreg: R^2 = 0.2822; F = 195.7897; P-value)(F) = 0.2822;
Var(error) = 0.2822;
statrob Sigmas: OLS_s= 0.4704; Robust_s= 0.4106; MAD_s= 0.3186;
final_s= 0.4111;
statrob: SE_p = [ 0.0361;  0.0622];
statrob: Corr_ps=[ 1.0000, -0.8609; -0.8609,  1.0000];
 >>
```

• ***6.2PS: Review of Homework 3, Problem 2(a), the N-Day Count:***

Theorem 3.1. LVaR $\sqrt{N}$ Factor from Daily Basis to N-days:

Let the k–day log-returns be IID, distributed normally, with zero-mean and $\sigma^2 k\Delta t$-variance, where Δt is one trading day in years and k is an integer, then for the log-VaR at risk level α satisfies

$$\mathbf{LVaR}_N(\alpha) = \sqrt{N} \cdot \mathbf{LVaR}_1(\alpha). \tag{6.7}$$

Proof: Let $\mathbf{LR}_{k,i}$ be k-day log-turns, then the distribution is normally distributed as

$$F_{\mathbf{LR}_{k,i}}(z) = F_Z^{(n)}(z; 0, \sigma^2 k\Delta t) \tag{6.8}$$

and the $\mathbf{LVaR}_k(\alpha)$ for $\mathbf{LR}_{k,i}$is defined by

$$\begin{aligned} \alpha &= \mathbf{Prob}[\mathbf{LR}_{k,i} < -\mathbf{LVaR}_k(\alpha)] \\ &= F_Z^{(n)}(-\mathbf{LVaR}_k(\alpha); 0, \sigma^2 k\Delta t) \\ &= F_Z^{(n)}(-\mathbf{LVaR}_k(\alpha)/(\sigma\sqrt{k\Delta t}); 0, 1), \end{aligned} \tag{6.9}$$

upon standardization.

Thus, setting k to N or 1 and letting

$$\frac{Z_k}{\sqrt{k}} \equiv \frac{\mathbf{LVaR}_k(\alpha)}{(\sigma\sqrt{k\Delta t})}, \tag{6.10}$$

$$F_Z^{(n)}(-Z_N/\sqrt{N};0,1) = \alpha = F_Z^{(n)}(-Z_1;0,1), \tag{6.11}$$

inverting while making reasonable assumption that $\alpha < 0.5$,

$$-Z_N/\sqrt{N} = (F_Z^{(n)})^{-1}(\alpha;0,1) = -Z_1, \tag{6.12}$$

and restoring the original variables,

$$\mathbf{LVaR}_N(\alpha) = \sqrt{N}\cdot\mathbf{LVaR}_1(\alpha). \quad \square \tag{6.13}$$

{*Remark: The case of non-zero mean distribution is left as an exercise.*}

Homework 3, Problem 2, Part (b):

Using the same k-day IID normal distribution with variance $\sigma^2 k\Delta t$ and zero-mean of the theorem to find the number of days k that it would take to make the cumulative tail probability (i.e., the probability on $(-\infty, -|LR_i|]$) at least 0.25, if possible, for the both the minimum and maximum 2009 daily log-returns in Problem 1, using the the variance and average trading day Δt in year units.

Let the extreme tail probability be

$$F_Z^{(n)}(-|\mathrm{LR}^{(m)}|; 0, \sigma^2 k\Delta t) = 0.25; \tag{6.14}$$

where $\mathrm{LR}^{(m)}$ is the daily ($k=1$) log-return $\min_i(\mathrm{LR}_{1,i})$ or $\max_i(\mathrm{LR}_{1,i})$. Since the data is daily returns $LR = [LR_{1,i}]_{n\times 1}$, then the MATLAB $\texttt{std}(\mathrm{LR}) \simeq \sigma\sqrt{\Delta t}$ gives an estimate of $\sigma \simeq \sqrt{n}\cdot\texttt{std}(\mathrm{LR})$ using $\Delta t = 1/n$ is the trading day in trading year units.

Converting, the k-day distribution to standard form, we have

$$F_Z^{(n)}\left(-\frac{|LR^{(m)}|}{\sigma\sqrt{k\Delta t}};0,1\right)=0.25; \tag{6.15}$$

and inverting

$$-\frac{|LR^{(m)}|}{\sigma\sqrt{k\Delta t}}=F_Z^{-1}(0.25;0,1); \tag{6.16}$$

or as the estimate of the number of days,

$$\begin{aligned} k=\widehat{k}^{(m)}(0.25) &= \frac{|LR^{(m)}|^2}{\left(\sigma\sqrt{\Delta t}F_Z^{-1}(0.25;0,1)\right)^2} \\ &\simeq \frac{|LR^{(m)}|^2}{\left(\mathtt{std(LR)}F_Z^{-1}(0.25;0,1)\right)^2}; \end{aligned} \tag{6.17}$$

Assuming that the k-day distribution is valid for k days and that this inverse solution exists.

○ *MATLAB Code for K-day Estimation for 2008 S&P500 Index Log-Return Extremes:*

```
function N_DayProblem10HW3P2b
% Get Histogram for S&P500 ^GSPC (Yahoo Finance) for Year 2008
%   Dates 2007/12/31-2008/12/31, Daily Adjusted Closings.
clc
fprintf('\nN_DayProblem10HW3P2b Output (%s):\n',datestr(now));
% Get S&P500 ^GSPC Adjusted Closings for 2008 From Yahoo Finance;
S = textread('GSPC2008adjC.txt','%f'); % Xcel.cvs deleted to 1 column.
L = length(S);
fprintf('\nlength(S)=%3i; mean(S)=%6.1f; std(S)=%5.1f;\n'...
    ,L,mean(S),std(S));
%
LR = log(S(2:L))-log(S(1:L-1)); % Note: Vector Log Difference!
LRlen = length(LR); LRmean = mean(LR); LRstd = std(LR);
fprintf('\nlength=%i; mean(LR)=%8.6f; std(LR)=%7.5f;' ...
    ,LRlen,LRmean,LRstd);
Dt = 1/LRlen; sigma = sqrt(LRlen)*LRstd;
fprintf('\nDt=%8.6f; sigma=%8.6f;',Dt,sigma);
LRmin = min(LR); LRmax = max(LR);
fprintf('\nLRmin=%8.6f; LRmax=%8.6f;',LRmin,LRmax);
alpha = 0.25;
FZinv = norminv(alpha,0,1);
kmin = LRmin^2/(LRstd*FZinv)^2;
```

```
kmax = LRmax^2/(LRstd*FZinv)^2;
fprintf('\nalpha=%7.4f; kmin=%7.4f; kmax=%7.4f;\n',alpha,kmin,kmax);

fprintf('\n+Probability Alpha Dependence:');
for alp = 0.1:0.05:0.5
    FZinv = norminv(alp,0,1);
    kmin = LRmin^2/(LRstd*FZinv)^2;
    kmax = LRmax^2/(LRstd*FZinv)^2;
    fprintf('\nalpha=%4.2f; kmin=%7.2f; kmax=%7.2f;',alp,kmin,kmax);
end
fprintf('\n ');


================= OUTPUT ========================
N_DayProblem10HW3P2b Output (03-Feb-2010 16:41:50):

length(S)=254; mean(S)=1221.0; std(S)=191.7;

length=253; mean(LR)=0.001921; std(LR)=0.02583;
Dt=0.003953; sigma=0.410822;
LRmin=-0.109572; LRmax=0.094695;
alpha= 0.2500; kmin=39.5605; kmax=29.5473;
```

```
+Probability Alpha Dependence:
alpha=0.10; kmin=  10.96; kmax=   8.18;
alpha=0.15; kmin=  16.75; kmax=  12.51;
alpha=0.20; kmin=  25.41; kmax=  18.98;
alpha=0.25; kmin=  39.56; kmax=  29.55;
alpha=0.30; kmin=  65.45; kmax=  48.88;
alpha=0.35; kmin= 121.22; kmax=  90.54;
alpha=0.40; kmin= 280.40; kmax= 209.43;
alpha=0.45; kmin=1139.75; kmax= 851.27;
alpha=0.50; kmin=    Inf; kmax=    Inf;
 >>
```

○ ***Stories Behind the K-Day Estimation for 2008 S&P500 Index Log-Return Extremes (Winter 2009 Lecture 3):***

1. Joe Nocera, Risk Mismanagement, New York Times Magazine, January 4, 2009, `http://www.nytimes.com/2009/01/04/magazine/04risk-t.html?pagewanted=1&ref=business`.
   {*Commentary: This was a good, recent Times article on the uses and abuses of* **`VaR`** *and the* ***Black-Scholes model****, in particular, the uses of normal distributions rather than nonnormal distributions for calculations. As a magazine article it was popularized for general audiences, so there is no math or graphical illustrations except for investor cartoons. In spite of practitioner criticisms on both sides it is still worth reading for historical and more recent background.* }

2. Anonomous (posted byYves Smith) Woefully Misleading Piece on Value at Risk in New York Times, naked capitalism website, January 4, 2009, ***http://www.nakedcapitalism.com/2009/01/woefully-misleading-piece-on-value-at.html*** . *{Commentary: This is a anonymous and likely practitioner's trashing of Nocera's popularized article on the problems of the* ***Black-Scholes model*** *and* ***VaR****, with much more emphasis on the strongly nonnormal and fat-tail view of the market. This article is a little more technical, with a few graphs on skewness and Cauchy distribution fat tails, as well as links to related articles elsewhere. }*

3. Paul De Grauwe, Leonardo Iania, Pablo Rovira Kaltwasser, HOW ABNORMAL WAS THE STOCK MARKET IN OCTOBER 2008?, EVRO Eurointelligence website, November 11, 2008, ***http://www.eurointelligence.com/article.581+M5f21b8d26a3.0.html***

{Commentary: This article is one of the cross-referenced by Anonymous, that comes with some striking illustrations about the extreme rarity of some of the changes of October 2008, some that from a normal distribution that should happen only once in a number of years that is astronomically largely than the known age of the universe. Also displayed are the great differences in the actual Dow Jones Industrial Average time trajectories 1928 to 2008 and the "toy" normal process trajectories. }

However, the class should have shown that the results are ridiculous on their homework for reasonable probabilities.

4. *Extreme Rarity of October 2008 Events (De Grauw, etal., 2008):*

A *non-Normal* October

Date	Percentage Change (1)	Average Frequency under Normal Law (2)
07/10/2008	-5.11%	Once in 5,345 Years
09/10/2008	-7.33%	Once in 3,373,629,757 Years
13/10/2008	11.08%	Once in 603,033,610,921,669,000,000,000 (3) Years
15/10/2008	-7.87%	Once in 171,265,623,633 Years
22/10/2008	-5.86%	Once in 117,103 Years
28/10/2008	10.88	Once 73,357,946,799,753,900,000,000 (3) Years

Figure 6.2: De Grauwe, etal. table of the largest DJIA movements in October 2008. Note, the largest rally (10/13) has an estimated frequency of $\mathbf{6 \cdot 10^{23}}$ years, while the largest crash (10/15) is at $\mathbf{1 \cdot 10^{11}}$ years. The current estimate of the known age of the universe is about $\mathbf{2 \cdot 10^{10}}$years. More ridiculous than the normal model itself?

5. ***Dow Jones Industrial Average (DJIA) time trajectories during the years 1928 to 2008 (De Grauwe, etal., 2008):***

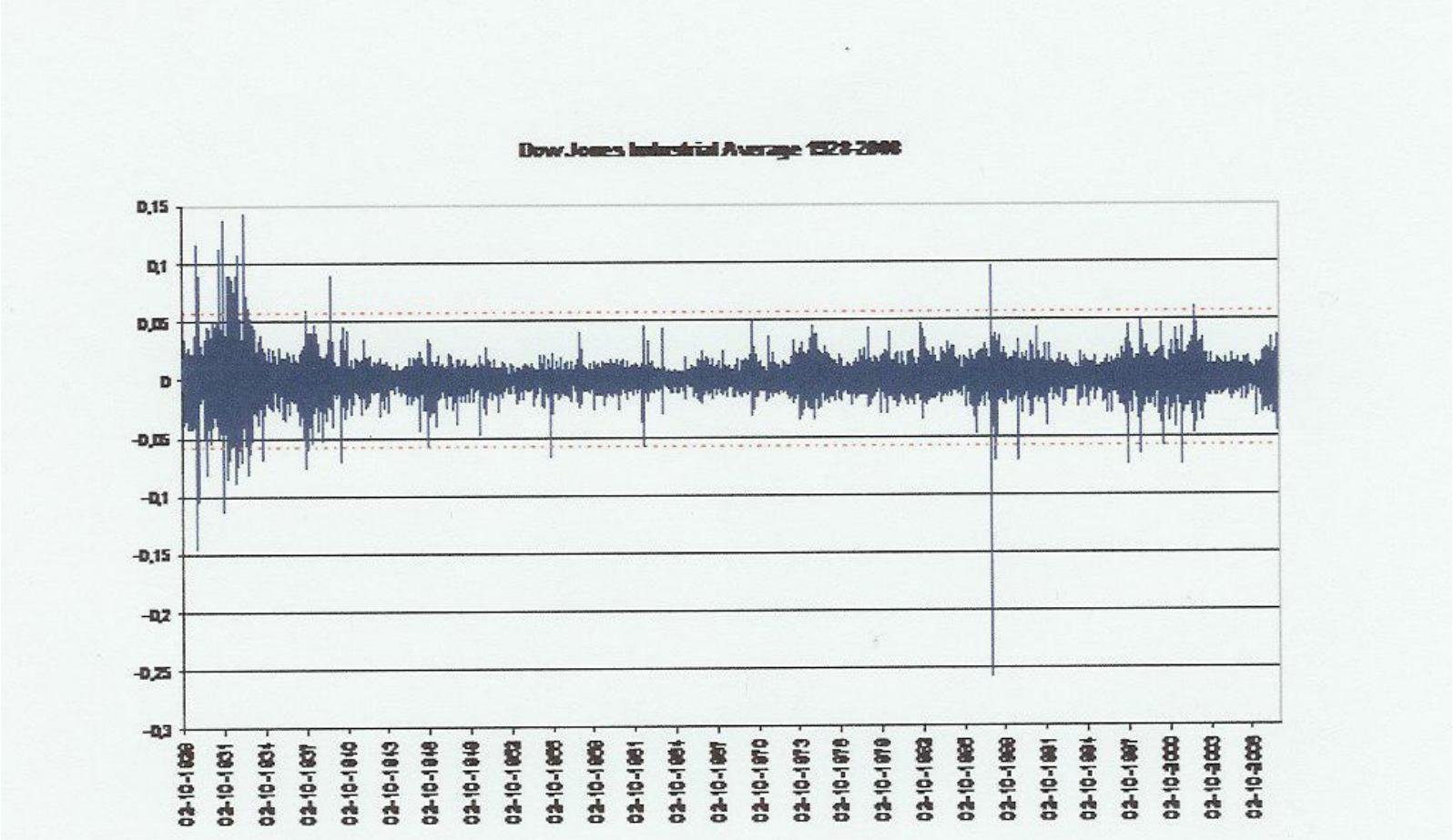

Figure 6.3: De Grauwe , etal. 2008 graph of the DJIA movements in 1928 to 2008, with October tickmarks. Note the biggest spike in 1987.

• *6.3 General Maximum Likelihood Function in* **MATLAB:**

[chat,cci]=mle(Y,'distribution','DistName','alpha', alpha): This is a maximum likelihood estimator for a large number of specialized distributions. The required input is data **Y**, but the optional input is the **'distribution'** parameter paired with the name value **'DistName'** which can be **'bernoulli'**, **'binomial'**, **'exponential'**, **'generalized pareto'**, **'lognormal'**, **'normal'** (default), **'poisson'**, **'uniform'** and others. The output is the estimated parameters **chat** and the parameter confidence interval **cci** at complementary level **alpha**, but if pair **'alpha', alpha** is omitted then the CI is at the MATLAB default of **0.05** or 95% CI. Many of the **mle** special distributions are prepackaged, such as **binofit**, **normfit** or **poissfit**. The auxiliary function **mlecov** with similar arguments outputs the parameter estimate covariance matrix.

○ *6.4 MLE for Linear Diffusion with Variable Trading-Day Intervals:*[a]

In the previous example of the asset price in a linear diffusion environment, it was assumed that the time between trading days. Δt, was constant and while this is the usual assumption among market practitioners justified by hand-waiving as well as more substantial arguments, **there are non-constant time differences for the many weekends and fewer holidays**.

---

[a]Continuing from the end of Lecture 5 with more variations on MLE for financial applications. This section is original as far as my experience indicates.

Returning to the discrete log-return asset pricing model and replacing $\Delta t > 0$ and $\Delta t_i > 0$ for $i = 1:n$ data observations,

$$\mathbf{LR}_i = \Delta \mathbf{X}_i = \Delta \log(A_i) = \widetilde{m}\Delta t_i + \sqrt{\widetilde{v}\Delta t_i}\, Z_i \tag{6.18}$$

where, $\widetilde{m} \equiv \mu - \sigma^2/2$ and $\widetilde{v} \equiv \sigma^2 > 0$ are log-coefficients from using $\Delta W_i = \sqrt{\Delta t_i}\, Z_i$ with $Z_i \overset{\text{dist}}{\underset{\text{IID}}{=}} \mathcal{N}(0,1)\ \forall\, i$. The the standard RV X has been changed to $\Delta \mathbf{X}_i$ for the consistency between increments in the variable time-step revised results.

The log-return distribution becomes,

$$\begin{aligned} F_{\Delta \mathbf{X}_i}(\Delta \mathbf{x}_i) &\overset{\text{alg}}{=} \mathbf{Prob}[Z_i \le (\Delta \mathbf{x}_i - \widetilde{m}\Delta t_i)/\sqrt{\widetilde{v}\Delta t_i}] \\ &\overset{\mathcal{N}}{=} F^{(\mathbf{n})}_{Z_i}\big((\Delta \mathbf{x}_i - \widetilde{m}\Delta t_i)/\sqrt{\widetilde{v}\Delta t_i}; 0, 1\big)\,. \end{aligned} \tag{6.19}$$

Upon differentiation with respect to $\Delta \mathrm{x}_i$ yields the ith likelihood function or density function form,

$$\begin{aligned}
\mathrm{LH}_i(\widetilde{m},\widetilde{v}) &= \frac{\partial F_{\Delta \mathrm{x}_i}}{\partial \Delta \mathrm{x}_i}(\Delta \mathrm{x}_i) = f_{\Delta \mathrm{x}_i}(\Delta \mathrm{x}_i) \\
&= \frac{\partial F^{(\mathrm{n})}_{Z_i}}{\partial \Delta \mathrm{x}_i}\left(\frac{(\Delta \mathrm{x}_i - \widetilde{m}\Delta t_i)}{\sqrt{\widetilde{v}\Delta t_i}};0,1\right) \\
&= \frac{1}{\sqrt{\widetilde{v}\Delta t_i}} f^{(\mathrm{n})}_{Z_i}\left(\frac{(\Delta \mathrm{x}_i - \widetilde{m}\Delta t_i)}{\sqrt{\widetilde{v}\Delta t_i}};0,1\right) \\
&\overset{\mathcal{N}}{=} \frac{1}{\sqrt{2\pi\widetilde{v}\Delta t_i}} \exp\left(-\frac{(\Delta \mathrm{x}_i - \widetilde{m}\Delta t_i)^2}{2\widetilde{v}\Delta t_i}\right).
\end{aligned} \tag{6.20}$$

Recall, the Z_i are IID normal, so the **total likelihood (TLH)** is the product of all the individual density for the log-return data count for $i = 1:n$,

$$\begin{aligned}\mathbf{TLH}_n(\widetilde{m},\widetilde{v}) &= f_{\vec{\Delta \mathbf{X}}}(\vec{\Delta \mathbf{x}}) \overset{\text{IID}}{=} \prod_{i=1}^{n} f_{\Delta \mathbf{X}_i}(\Delta \mathbf{x}_i) \\ &\overset{\mathcal{N}}{=} \prod_{i=1}^{n} \frac{\exp\left(-0.5(\Delta \mathbf{x}_i - \widetilde{m}\Delta t_i)^2/(\widetilde{v}\Delta t_i)\right)}{\sqrt{2\pi\widetilde{v}\Delta t_i}}.\end{aligned} \tag{6.21}$$

Taking logarithms, $\mathbf{LLH}_n = \log(\mathbf{TLH}_n)$ turning the products into sums, to get the **log-likelihood (LLH) function**,

$$\begin{aligned}\mathbf{LLH}_n(\widetilde{m},\widetilde{v}) &= \log\left(f_{\vec{\Delta \mathbf{X}}}(\vec{\Delta \mathbf{x}})\right) = \sum_{i=1}^{n} \log(f_{\Delta \mathbf{X}_i}(\Delta \mathbf{x}_i)) \\ &= -\sum_{i=1}^{n}\left(\frac{(\Delta \mathbf{x}_i - \widetilde{m}\Delta t_i)^2}{2\widetilde{v}\Delta t_i} + \frac{1}{2}\log(2\pi\widetilde{v}\Delta t_i)\right),\end{aligned} \tag{6.22}$$

which again is a least squares objective modified by the log of a normalization term.

Seeking critical points,

$$\frac{\partial \mathrm{LLH}_n}{\partial \widetilde{m}}(\widetilde{m},\widetilde{v}) = \sum_{i=1}^{n} \frac{(\Delta \mathrm{x}_i - \widetilde{m}\Delta t_i)\Delta t_i}{\widetilde{v}\Delta t_i} \stackrel{*}{=} 0, \tag{6.23}$$

and

$$\frac{\partial \mathrm{LLH}_n}{\partial \widetilde{v}}(\widetilde{m},\widetilde{v}) = \sum_{i=1}^{n} \left(\frac{(\Delta \mathrm{x}_i - \widetilde{m}\Delta t_i)^2}{2\widetilde{v}^2 \Delta t_i} - \frac{1}{2\widetilde{v}} \right) \stackrel{*}{=} 0, \tag{6.24}$$

gives the simultaneous estimates,

$$\widehat{m} = \widetilde{m}^* = \sum_{i=1}^{n} \Delta \mathrm{x}_i \Big/ \sum_{i=1}^{n} \Delta t_i \equiv \overline{\Delta \mathrm{x}} / \overline{\Delta t} \tag{6.25}$$

and

$$\begin{aligned} \widehat{v} = \widetilde{v}^* &= \frac{1}{n} \sum_{i=1}^{n} \frac{(\Delta \mathrm{x}_i - \widetilde{m}^* \Delta t_i)^2}{\Delta t_i} \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{\Delta \mathrm{x}_i}{\sqrt{\Delta t_i}} - \frac{\overline{\Delta \mathrm{x}}}{\sqrt{\overline{\Delta t}}} \sqrt{\frac{\Delta t_i}{\overline{\Delta t}}} \right)^2 \equiv \widehat{\sigma}^2_{\Delta \mathrm{x}/\sqrt{\Delta t}}, \end{aligned} \tag{6.26}$$

a somewhat unusual definition of estimated variance of a **normalized linear combination (NLC)**, but fits the model application.

Finally, converting back to standard model coefficient,

$$\widehat{\sigma} = \widehat{\sigma}_{\Delta x/\sqrt{\Delta t}} \quad \& \quad \widehat{\mu} = \overline{\Delta x}/\overline{\Delta t} + \widehat{\sigma}^2_{\Delta x/\sqrt{\Delta t}}\Big/2, \tag{6.27}$$

the latter form seems to contradict the practice of throwing out the mean $\widetilde{m}$ since $\widehat{\sigma}^2_{\Delta x/\sqrt{\Delta t}}\Big/2 > 0$.

Notice that when $\Delta t_i = \Delta t$, **a constant**,

$$\widehat{\sigma}^2_{\Delta x/\sqrt{\Delta t}} = \widehat{\sigma}^2_{\Delta x}/\Delta t \tag{6.28}$$

so then

$$\widehat{\sigma} = \widehat{\sigma}_{\Delta x}/\Delta t \quad \& \quad \widehat{\mu} = \left(\overline{\Delta x} + \widehat{\sigma}^2_{\Delta x}/2\right)/\Delta t. \tag{6.29}$$

Hence, there is not a great deal of difference as Hull (2006) comments on in his book, so the time between trading days is taken to be a constant.

• 6.5 MLE for Linear Jump-Diffusion with Compound Poisson and Otherwise Constant Coefficients:

Consider the asset price **compound jump-diffusion model**

$$dA(t) = A(t)(\mu dt + \sigma dW(t) + \nu(Q)dP(t)), \tag{6.30}$$

where $\sigma > 0$, $\nu(Q) > -1$ is the **jump-amplitude**, $dP(t) = dP(t;Q)$ is the **differential Poisson jump process**, such that **only one jump is likely** with jump-rate λ in the infinitesimally small time step dt, and $\mathrm{E}[dP(t;Q)] = \lambda f_Q(q;a,b)dqdt > 0$. The **underlying Poisson jump-amplitude random variables** Q are IID RVs with density $f_Q(q;a,b)$ on (a,b). The combination of random jump process and jump-amplitude, $\nu(Q)dP(t)$ is called a **compound Poisson process**. Using the doubly-stochastic form of the chain rule for jump-diffusion processes[a] with independent continuous and jump-changes, the logarithmic change of variables $X(t) = \log(A(t))$ results in

$$dX(t) = (\mu - \sigma^2/2)dt + \sigma dW(t) + \log(1+\nu(Q))dP(t)). \tag{6.31}$$

[a] Hanson ('07) book, Chapters 4-5.

However, the **given FinM 331/Stat 339 class problem** begins with the approximate discretized **Gaussian-Poisson mixture model** for **sufficiently small constant time-steps Δt** producing the log-return,

$$\mathrm{LR}_i = \Delta \mathrm{X}_i = \widetilde{m}\Delta t + \sqrt{\widetilde{v}\Delta t}Z_i + Q_i B_i(p_b), \tag{6.32}$$

where $\widetilde{m} \equiv \mu - \sigma^2/2$, $\widetilde{v} \equiv \sigma^2$ and letting

$$Q_i \equiv \log(1+\nu(Q_i)) \text{ or } \nu(Q_i) = \exp(Q_i) - 1 \tag{6.33}$$

has been selected for underlying jump-amplitude simplicity. Also, we have replaced the assumed **zero-one jump** incremental Poisson process ΔP_i by its equivalent, the **Bernoulli process**[a], $B_i(p_b)$, with parameter $p_b \equiv \frac{\Lambda}{1+\Lambda} \ll 1$[b], $\Lambda \equiv \lambda \Delta t$, which is the probability of one jump, while the complementary probability is $1-p_b = \frac{1}{1+\Lambda}$ for no jump, as if this were for an unfair (i.e., $p_b \neq 1/2$) coin-flips.

[a] Recall call that **`MATLAB Statistics Toolbox`** does not have a Bernoulli RNG because the binomial RNG with parameter **`n=1`**, **`binornd(1,pb,M,N)`**, is equivalent.

[b] Alternatively, $p_b = \Lambda$ is used, but then $1-p_b = 1-\Lambda$, which is positive as a probability only for $\Lambda < 1$ and that is an inconsistent formulation that is often used.

The log-return distribution is found using **Law of Total Probability (LTP)**[a], which basically states that if X is a RV and $\{Y_k\}_{k=1}^{\infty}$ is a countable set of discrete RVs, then

$$\mathbf{Prob}[X \leq x] \overset{\mathbf{LTP}}{=} \sum_{k=1}^{\infty} \mathbf{Prob}[X \leq x | Y_k] \cdot \mathbf{Prob}[Y_k]. \tag{6.34}$$

Letting the Gaussian or general normal part be

$$\Delta G_i \equiv \widetilde{m}\Delta t + \sqrt{\widetilde{v}\Delta t} Z_i \tag{6.35}$$

which has distribution $F^{(n)}_{\Delta G_i}(x; \widetilde{m}\Delta t, \widetilde{v}\Delta t)$, then

$$\begin{aligned} F^{(\mathbf{jd})}_{\Delta \mathbf{X}_i}(x) &\equiv \mathbf{Prob}[\Delta \mathbf{X}_i \leq x] = \mathbf{Prob}[\Delta G_i + Q_i B_i(p_b) \leq x] \\ &\overset{\mathbf{LTP}}{=} \sum_{k=0}^{1} \mathbf{Prob}[\Delta G_i + Q_i B_i(p_b) \leq x |\, B_i(p_b) = k] \\ &\quad \cdot \mathbf{Prob}[B_i(p_b) = k] \\ &= (1-p_b)\mathbf{Prob}[\Delta G_i \leq x] + p_b \mathbf{Prob}[\Delta G_i + Q_i \cdot 1 \leq x]. \end{aligned} \tag{6.36}$$

[a] Hanson (2007) Online Appendix B, Sect. B.3.2, pp. B29-B30

• *6.6 MLE for Linear Jump-Diffusion with Simple Poisson and Simpler, Single Jump-Amplitude $Q = q_0$:*

If the $Q_i = q_0$, where q_0 is a single fixed value, i.e., Q_i is discretely distributed, and we have a **simple Bernoulli-Poisson process** rather than a compound one. Returning to (6.36) with a single, discrete jump $Q_i = q_0$,

$$\begin{aligned} F^{(\mathrm{jd})}_{\Delta \mathrm{X}_i}(x) &= (1-p_b)\mathrm{Prob}[\Delta G_i \le x] + p_b \mathrm{Prob}[\Delta G_i + q_0 \le x] \\ &= (1-p_b) F^{(n)}_{\Delta G_i}(x; \widetilde{m}\Delta t, \widetilde{v}\Delta t) \\ &\quad + p_b F^{(n)}_{\Delta G_i}(x; \widetilde{m}\Delta t + q_0, \widetilde{v}\Delta t), \end{aligned} \tag{6.37}$$

a binary mixture of fixed Gaussian distributions.

Differentiating produces in the ith likelihood function or density function mixture form,

$$\begin{aligned}\mathbf{LH}_i(x_i;\widetilde{m},\widetilde{v},\boldsymbol{\lambda},q_0) &= f^{(\mathbf{jd})}_{\boldsymbol{\Delta}\mathbf{X}_i}(x_i)\\ &= (1-p_b)f^{(n)}_{\boldsymbol{\Delta}G_i}(x_i;\widetilde{m}\boldsymbol{\Delta}t,\widetilde{v}\boldsymbol{\Delta}t)\\ &\quad +p_b f^{(n)}_{\boldsymbol{\Delta}G_i}(x_i;\widetilde{m}\boldsymbol{\Delta}t+q_0,\widetilde{v}\boldsymbol{\Delta}t),\end{aligned} \tag{6.38}$$

which depends on four (4) unknown parameters.

Applying the general independence of the component processes in the simple Bernoulli-Poisson jump-diffusion yields the total data likelihood function,

$$\begin{aligned}\mathbf{TLH}_n(\vec{x};\widetilde{m},\widetilde{v},\boldsymbol{\lambda},q_0) &= f^{(\mathbf{jd})}_{\vec{\boldsymbol{X}}}(\vec{x}) \overset{\mathbf{IID}}{=} \textstyle\prod_{i=1}^n f^{(\mathbf{jd})}_{\boldsymbol{X}_i}(x_i)\\ &= \textstyle\prod_{i=1}^n \Big((1-p_b)f^{(n)}_{\boldsymbol{\Delta}G_i}(x_i;\widetilde{m}\boldsymbol{\Delta}t,\widetilde{v}\boldsymbol{\Delta}t)\\ &\quad +p_b f^{(n)}_{\boldsymbol{\Delta}G_i}(x_i;\widetilde{m}\boldsymbol{\Delta}t+q_0,\widetilde{v}\boldsymbol{\Delta}t)\Big).\end{aligned} \tag{6.39}$$

Taking logs only reduces the product symbol to a sum, leading to a complicated log-likelihood function,

$$\begin{aligned}\mathbf{LLH}_n(\vec{x};\widetilde{m},\widetilde{v},\boldsymbol{\lambda},q_0) &= \sum_{i=1}^{n}\log\Big(f^{(\mathbf{jd})}_{\boldsymbol{\Delta}\mathbf{X}_i}(\boldsymbol{\Delta}\mathbf{x}_i)\Big)\\ &= \sum_{i=1}^{n}\log\Big(\\ &\quad (1-p_b)\,f^{(n)}_{\Delta G_i}(x_i;\widetilde{m}\Delta t,\widetilde{v}\Delta t)\\ &\quad +p_b f^{(n)}_{\Delta G_i}(x_i;\widetilde{m}\Delta t+q_0,\widetilde{v}\Delta t)\Big),\end{aligned} \tag{6.40}$$

clearly too big a problem to try to think about solving analytically, but to think about solving computationally with approximations.

{*Remark: Since it is not a single Gaussian or normal, the* `MATLAB` *functions like* `normfit` *or any of the* `mle` *options will not work, so a general optimizer is needed.*}

• *6.7 Numerical Optimization and* `fminsearch` *General Direct Search:*[a]

Since the simple Poisson jump-diffusion model is already **sufficiently computationally complex** and it is unlikely that the one-jump probability p_b will be small compared to the zero-jump probability $1-p_b$, recalling our multijump results for 2008 and 2009 S&P 500 Index log-returns.

In addition, having discrete data, we have a **need for efficient derivative-free and nonsmooth optimum solvers**.

---

[a] See `help fminsearch` in `MATLAB command` window or search for `fminsearch` in `MATLAB help` window or for much more information see the ***Optimization Toolbox(TM) 4 User's Guide*** as ***http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf***.

○ *MATLAB Optimization Decision Table, Part 1:*

Optimization Decision Table

The following table is designed to help you choose a solver. It does not address multiobjective optimization or equation solving. There are more details on all the solvers in Problems Handled by Optimization Toolbox Functions

Use the table as follows:

1. Identify your objective function as one of five types:
   - Linear
   - Quadratic
   - Sum-of-squares (Least squares)
   - Smooth nonlinear
   - Nonsmooth
2. Identify your constraints as one of five types:
   - None (unconstrained)
   - Bound
   - Linear (including bound)
   - General smooth
   - Discrete (integer)
3. Use the table to identify a relevant solver.

In this table:

- Blank entries means there is no Optimization Toolbox solver specifically designed for this type of problem.
- * means relevant solvers are found in Genetic Algorithm and Direct Search Toolbox functions (licensed separately from Optimization Toolbox solvers).
- `fmincon` applies to most smooth objective functions with smooth constraints. It is not listed as a preferred solver for least squares or linear or quadratic programming because the listed solvers are usually more efficient.
- The table has suggested functions, but it is not meant to unduly restrict your choices. For example, `fmincon` is known to be effective on some non-smooth problems.
- The Genetic Algorithm and Direct Search Toolbox function `ga` can be programmed to address discrete problems. It is not listed in the table because additional programming is needed to solve discrete problems.

Figure 6.4: From Optimization Toolbox, ***Choosing a Solver***.

- ***MATLAB Optimization Decision Table, Part 2:***

Solvers by Objective and Constraint

Constraint Type	Objective Type				
	Linear	Quadratic	Least Squares	Smooth nonlinear	Nonsmooth
None	n/a (*f* = const, or min = −∞)	quadprog, Theory, Examples	\, lsqcurvefit, lsqnonlin, Theory, Examples	fminsearch, fminunc, Theory, Examples	fminsearch, *
Bound	linprog, Theory, Examples	quadprog, Theory, Examples	lsqcurvefit, lsqlin, lsqnonlin, lsqnonneg, Theory, Examples	fminbnd, fmincon, fseminf, Theory, Examples	*
Linear	linprog, Theory, Examples	quadprog, Theory, Examples	lsqlin, Theory, Examples	fmincon, fseminf, Theory, Examples	*
General smooth	fmincon, Theory, Examples	fmincon, Theory, Examples	fmincon, Theory, Examples	fmincon, fseminf, Theory, Examples	*
Discrete	bintprog, Theory, Example				

Figure 6.5: From Optimization Toolbox, ***Choosing a Solver***.

○ **MATLAB `fminsearch` *Details:***

`[x,fval,exitflag,output]=fminsearch(@f,x0,options);`

Solves the local minimum problem for a scalar valued function **`f`** of a single, vector argument **`x`**, with minimum location[a] at

$$x^* \simeq \mathbf{argmin}_x[f(x)], \tag{6.41}$$

given an initial multivariable start **`x0`** and objective function **`f`** appearing as the first argument as the pointer or handle **`@f`** pointing to a subfunction within the main function m-file. The **`fval`** $\simeq \mathbf{min}_x[f(x)] \simeq f(x^*)$.

1. Parameter or other variable arguments must be passed indirectly and not with the single argument **`x`**. It is recommended that the **`global`** declaration be used, e.g., **`global a b c`** placed in both **`f`** and main function codes, where **`(a,b,c)`** is a known parameter and variable set.[b]

[a] For **maxima** of $g(x)$, $x* = \mathbf{argmin}_x[f(x)]$ where $f(x) = -g(x)$, using a minimizer functions. For **MLE** computations, the unlikely term **negative-likelihood** is used.

[b] ***Contrary to MATLAB advice, avoid anonymous function dynamic input in command window, except for testing, and nested function input, as its documentation demonstrates their messy approaches.***

2. **Also for parameter estimation, the meaninngs are reversed, `x=` parameter vector and $[a, b, c] =$ data vector.**
3. The entities that are `global` variables must have the same name in both calling function and the objective subfunction, so order and count in global declarations does not matter, in fact `mlint` will complain if a parameter or variable is not used in the current function or subfunction.
4. ***Constraints*** can be embedded in function directly or by sufficiently large values not mistaken for a minimum.
5. Function `f` must have a proper form for a ***genuine minimum problem***, e.g., negative of the maximum likelihood objective and in that application `x` is the unknown parameter vector, while the usual state data vector is passed to `f` along with known parameters.
6. `fminsearch` is a general or ***derivative-free*** function and is not bothered by discontinuities except near the minimum location, i.e., is very robust.

Other input and output arguments are

1. `options` is an optional input structure argument that is usually set by the `optimset` function to handle fields such as `Display, FunValCheck, MaxFunEvals, MaxIter, OutputFcn, PlotFcns, TolFun, TolX`, but see `help optimset` for more information. Use `optimset fminsearch` in the command window to get a listing of the default settings for these options, e.g., both options `TolFun` and `TolX` have default values of 1.e-4, while options `MaxFunEvals` and `MaxIter` have default values of `200*length(x)`. Most other values are set to null, `[ ]`. These values can be reset if desired by parameter-value pairs, separated by commas, executed in the command window, e.g.,

```
options = optimset('Display', 'Iter',
'MaxIter', 500, 'TolFun', 1e-6, 'TolX', 1e-8)
```

   Notice that script arguments are in single quotes while numerical values are without quotes.

2. **`x`** is the fminsearch output local minimum location (depends on **`x0`**) .
3. **`fval`** is the minimal function value **`f(x)`** for output **`x`**,
4. **`exitflag`** is the value of a flag on the condition of the exit of fminsearch, i.e., **`+1`** if convergence, **`0`** if maximum function evaluations **`MaxFunEvals`** or iterations **`MaxIter`**attained or if **`-1`** output ended by objective **`f`**.
5. **`output`** is an output structure with elements **`output.algorithm`** saying what algorithm was used, **`output.funcCount`** is the number of function evaluations, **`output.iterations`** is the number of iterations, and **`output.messsage`** is exit message.

P.S. For **general nonlinear least square objectives** of the form

$$x^* = \min_x \left[\|\vec{f}(\vec{x})\|^2 \right] = \min_x \left[\sum_{i=1}^{n} (f_i(\vec{x}))^2 \right], \qquad (6.42)$$

then least squares, nonlinear optimizer **`lsqnonlin`** should be used, but is not derivative-free and is for a continuous **`f`**, as is **`fminunc`** for large scale unconstrained problems.

- ***Algorithm of `fminsearch` — Nelder and Mead's Downhill Simplex Method:***[a]

Let m be the dimension of the unknown, e.g., parameter, vector $\vec{x} = [x_i]_{m\times 1}$, then form a ***simplex*** of $m+1$ vertices with locations $\{\vec{x}_j = [x_{i,j}]_{m\times 1}$ for $j = 1:m+1\}$, i.e., similar to polygons in the plane where the triangle is a simplex, but in multidimensions.

1. The values of all vertices are calculated, $F_j = f(\vec{x}_j)$ for $j = 1:m+1$, then ***ordered*** the $\vec{x}_j$ such that $F_j \le F_{j+1}$;
2. The vector $\vec{x}_{m+1}$ of the largest value is replaced by a ***reflection***, $\vec{r} = 2\overline{x}_m - \vec{x}_{m+1}$, about the center, $\overline{x}_m = \sum_{k=1}^{m} \vec{x}_k/m$, of the rest of simplex;

---

[a] See section **`fminsearch algorithm`** in **`help Unconstrained Nonlinear Optimization`** page of the **`Optimization Toolbox`**, but has been in regular MATLAB; J. Lagarias, J. Reeds, M. H. Wright and P. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," SIAM J. Optmization, vol. 9, No. 1, pp. 112-147, 1998; J. A. Nelder and R. Mead, A Simplex Method for Function Minimization, Computer Journal, vol, 7, pp. 308-313, 1965.

3. If the ***reflection*** value is in the remainder $F_1 \leq f(\vec{r}) < F_m]$ then $\vec{r}$ replaces $\vec{x}_{m+1}$ and a new iteration begins;
4. If $f(\vec{r}) < F_1$ then the reflection becomes an ***expansion***, $\vec{s} = 2\vec{r} - \overline{x}_m$ and if $f(\vec{s}) < f(\vec{r})$, an improvement, then $\vec{s}$ is the replacement for the largest, else $\vec{r}$ is that replacement, and in either case go to a new iteration where either replace will become the new $\vec{x}_1$;
5. Otherwise there are several variations of contractions (***contraction outside*** ($\vec{c}$), ***contraction inside*** ($\vec{cc}$), ***shrink*** ($\vec{v}$)).

The iterations continue until a combination of tolerances, **`TolFcn`** and **`Tolx`**, is satisfied, unless the count limits **`MaxFunEvals`** or **`MaxIter`** are reached first.

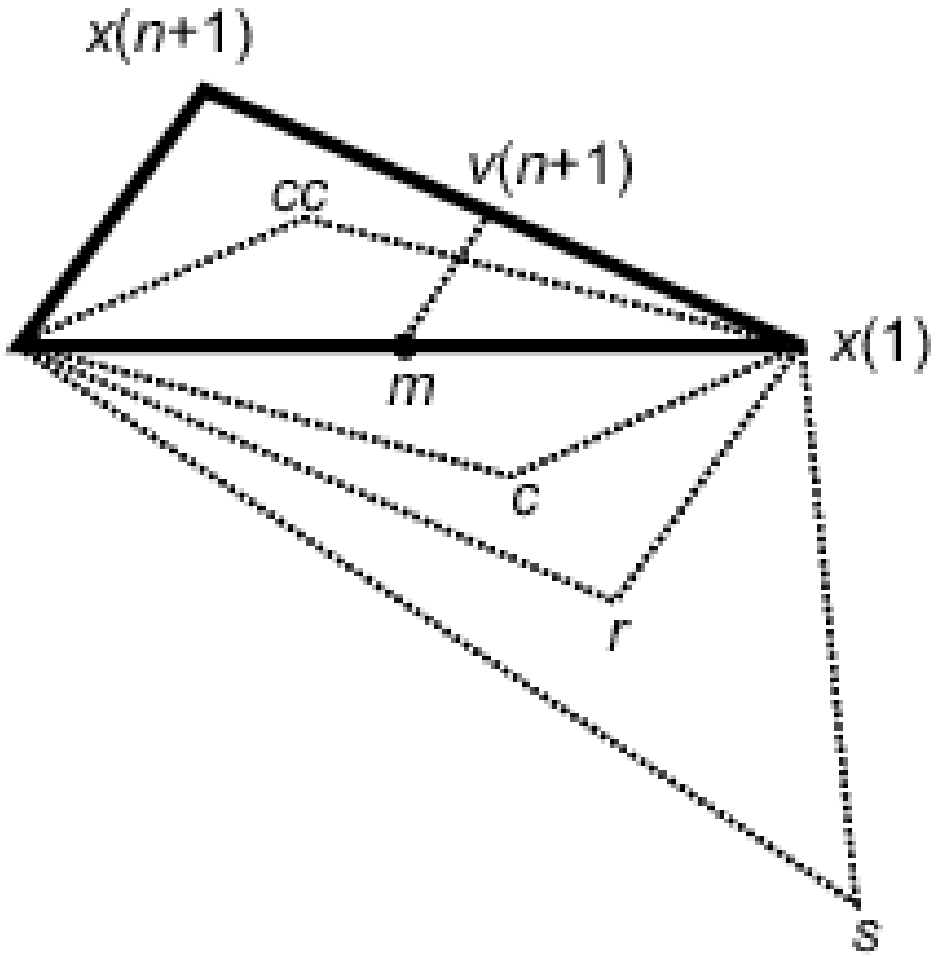

Figure 6.6: **MATLAB Down-Hill Simplex Algorithm Illustration:** *Fminsearch Algorithm, Unconstrained Nonlinear Optimization page, Optimization Toolbox, 2008.* Triangular with $(x(1), x(2), x(n+1) \approx x(3))$.

• *6.8 MLE for Linear Jump-Diffusion with a Zero-One Jump Compound Poisson Continued with Jump-Amplitude Distribution:*

Previously in Eq. (6.36), we considered the **zero-one jump-amplitude** Q_i in a **binary Gaussian mixture** that was not simplistic as the single discrete jump q_0 as in (6.37), but there are still the underlying Poisson jump-amplitude IID RVs, Q_i, in the jump-diffusion distribution,

$$\begin{aligned} F^{(\mathrm{jd})}_{\Delta \mathrm{X}_i}(x) &= (1-p_b)\mathrm{Prob}[\Delta G_i \le x] + p_b \mathrm{Prob}[\Delta G_i + Q_i \le x] \\ &= (1-p_b)F^{(n)}_{\Delta G_i}(x; \widetilde{m}\Delta t, \widetilde{v}\Delta t) + p_b F_{\Delta G_i + Q_i}(x), \end{aligned} \tag{6.43}$$

where $p_b = \Lambda/(1+\Lambda)$ is the one-jump probability, while $(1-p_b)$ is the zero-jump probability, and $\Delta G_i = \widetilde{m}\Delta t + \sqrt{\widetilde{v}\Delta t}Z_i$ is the dominating Gaussian term. The underlying the Q_i are usually continuous IID RVs and thus do not nicely fit into **LTP** form.

We have the linear combination of two distributions in (6.43), one is an obvious nonstandard normal or Gaussian distribution, but the other is a distribution for the sum of two independent RVs, say $Y = Q_i$ for the jump-amplitude and $Z = \Delta G_i$ for the Gaussian. However, the sum provides a constraint to their independence as we shall see.

When we have two independent RVs, Y and Z, then their constrained sum $Y+Z \leq x$ connects their distributions to a integral relation called a **convolution**[a] of distributions, denoted with a symmetry property by $(F_Z * f_Y)(x) = (F_Y * f_Z)(x)$ and follows from these steps:

$$\begin{aligned} \mathbf{Prob}[Y+Z \leq x] &\stackrel{\mathbf{I}}{=} \int_{-\infty}^{\infty} dy f_Y(y) \int_{-\infty}^{\infty} dz f_Z(z) 1_{\{y+z \leq x\}} \\ &= \int_{-\infty}^{\infty} dy f_Y(y) \int_{-\infty}^{x-y} dz f_Z(z) \\ &\stackrel{\mathbf{dist}}{=\!=\!=}_{\mathbf{def}} \int_{-\infty}^{\infty} F_Z(x-y) f_Y(y) dy \equiv (F_Z * f_Y)(x). \end{aligned} \tag{6.44}$$

[a] See Hanson (2007), Online Appendix B, p. B30ff.

Consequently, with

$$\mathbf{Prob}[\Delta G_i \le x] = F^{(n)}_{\Delta G_i}(x) = F^{(n)}_{\Delta G_i}(x; \widetilde{m}\Delta t, \widetilde{v}\Delta t),$$

$$\begin{aligned}
F^{(\mathrm{jd})}_{\Delta X_i}(x) &= (1-p_b)F^{(n)}_{\Delta G_i}(x) + p_b(F^{(n)}_{\Delta G_i} * f_Q)(x) \\
&= (1-p_b)F^{(n)}_{\Delta G_i}(x) + p_b \int_a^b F^{(n)}_{\Delta G_i}(x-q) f_Q(q) dq \\
&= (1-p_b)F^{(n)}_{\Delta G_i}(x) + p_b \int_a^b dq f_Q(q) \int_{-\infty}^{x-q} dz f^{(n)}_{\Delta G_i}(z) \\
&\overset{\text{int}}{=\!=\!\!\rightarrow} (1-p_b)F^{(n)}_{\Delta G_i}(x) + p_b \Big(\int_{-\infty}^{x-b} dz f^{(n)}_{\Delta G_i}(z) \int_a^b dq f_Q(q) \\
&\quad + \int_{x-b}^{x-a} dz f^{(n)}_{\Delta G_i}(z) \int_a^{x-z} dq f_Q(q) \Big) \\
&= (1-p_b)F^{(n)}_{\Delta G_i}(x) + p_b \Big(F^{(n)}_{\Delta G_i}(x-b) \cdot 1 \\
&\quad + \int_{x-b}^{x-a} dz f^{(n)}_{\Delta G_i}(z) \int_a^{x-z} dq f_Q(q) \Big).
\end{aligned} \tag{6.45}$$

We cannot do much more until be specify the jump-amplitude density $f_Q(q) \equiv f_Q(q; a, b)$.

Technique Alert — Double Integral Interchanges:

Upon substituting the density integral for the distribution $F^{(n)}_{\Delta G_i}(x-b)$,

$$\int_a^b dq f_Q(q) \int_{-\infty}^{x-q} dz f^{(n)}_{\Delta G_i}(z) \overset{\text{int}}{\underset{\leftrightarrow}{=}} \int_{-\infty}^{x-b} dz f^{(n)}_{\Delta G_i}(z) \int_a^b dq f_Q(q) + \int_{x-b}^{x-a} dz f^{(n)}_{\Delta G_i}(z) \int_a^{x-z} dq f_Q(q). \tag{6.46}$$

The original domain is a single trapezoid $z \in (-\infty, x-q)$ on $q \in (a,b)$ and changing the order of integrations changes the domain into two pieces, a rectangle $q \in (a,b)$ on $z \in (-\infty, x-b)$ plus a triangle $q \in (a, x-z)$ on $z \in (x-b, x-a)$ with q-integral nested inside the zq-double integral.

The simplest fat-tail distribution is the **uniform distribution**, and it all tail, on (a,b), $a<0<b$ for bear and bull tails, and its density is

$$f_Q(q;a,b)=\frac{1}{b-a}\left\{\begin{array}{ll} 1, & q\in(a,b) \\ 0, & else \end{array}\right\}. \tag{6.47}$$

Thus, the **uniform-amplitude jump-diffusion (UJD)** distribution becomes, after simplifying a normal integral with a linear coefficient,

$$\begin{aligned} F^{(\mathbf{ujd})}_{\Delta \mathbf{X}_i}(x) &= (1-p_b)F^{(n)}_{\Delta G_i}(x)+p_b\left(F^{(n)}_{\Delta G_i}(x-b)\right. \\ &\quad \left.+\int_{x-b}^{x-a} dz f^{(n)}_{\Delta G_i}(z)\frac{x-z-a}{b-a}\right) \\ &= (1-p_b)F^{(n)}_{\Delta G_i}(x)+p_b\frac{b+\widetilde{m}\Delta t-x}{b-a}F^{(n)}_{\Delta G_i}(x-b) \\ &\quad +p_b\frac{x-\widetilde{m}\Delta t-a}{b-a}F^{(n)}_{\Delta G_i}(x-a) \\ &\quad +p_b\frac{\widetilde{v}\Delta t}{b-a}\left(f^{(n)}_{\Delta G_i}(x-b)-f^{(n)}_{\Delta G_i}(x-a)\right). \end{aligned} \tag{6.48}$$

Technique Alert — Normal Integral Over Linear Term:

Upon collecting the Gaussian deviation argument, $(z-\widetilde{m}\Delta t)$, to obtain an exact integral for the density,

$$\int_{x-b}^{x-a} dz\, f^{(n)}_{\Delta G_i}(z)\frac{x-z-a}{b-a} = \int_{x-b}^{x-a} dz\, f^{(n)}_{\Delta G_i}(z)\frac{(x-\widetilde{m}\Delta t-a)-(z-\widetilde{m}\Delta t)}{b-a}$$

$$= \frac{x-\widetilde{m}\Delta t-a}{b-a}\left(F^{(n)}_{\Delta G_i}(x-a)-F^{(n)}_{\Delta G_i}(x-b)\right) \quad (6.49)$$

$$+\frac{\widetilde{v}\Delta t}{b-a}\left(f^{(n)}_{\Delta G_i}(x-b)-f^{(n)}_{\Delta G_i}(x-a)\right).$$

Differentiating, the jump-diffusion distribution, using the easier first equality of (6.48), becomes

$$f^{(\mathbf{ujd})}_{\Delta \mathbf{X}_i}(x) = (1-p_b) f^{(n)}_{\Delta G_i}(x) + \frac{p_b}{b-a}\left(F^{(n)}_{\Delta G_i}(x-a) - F^{(n)}_{\Delta G_i}(x-b)\right), \tag{6.50}$$

after several cancellations. The last term is in a difference form that is the **secant approximation of a derivative**,

$$f^{(sn)}_{Z}(z_1, z_2) \equiv \frac{F^{(n)}_{Z}(z_2) - F^{(n)}_{Z}(z_1)}{z_2 - z_1}, \tag{6.51}$$

so is called the **secant-normal (SN) density**.[a] Here, we have

$$f^{(\mathbf{ujd})}_{\Delta \mathbf{X}_i}(x) = (1-p_b) f^{(n)}_{\Delta G_i}(x) + p_b f^{(sn)}_{\Delta G_i}(x-b, x-a). \tag{6.52}$$

[a] Hanson (2007), p. 156. See also, online Errata. Also note that as $z_2 \to z_1$, $f^{(sn)}_{Z}(z_1, z_2) \to f^{(n)}_{Z}(z_1)$.

From Eq. (6.40), the log-likelihood function for UJD is

$$\begin{aligned}\mathbf{LLH}_n(\vec{x};\vec{p}) &= \sum_{i=1}^n \log\Big(f^{(\mathbf{ujd})}_{\mathbf{\Delta X}_i}(x_i)\Big)\\ &= \sum_{i=1}^n \log\Big((1-p_b) f^{(n)}_{\Delta G_i}(x_i;\widetilde{m}\Delta t,\widetilde{v}\Delta t)\\ &\quad + p_b f^{(sn)}_{\Delta G_i}((x-b,x-a;\widetilde{m}\Delta t\,\widetilde{v}\Delta t)\Big),\end{aligned} \tag{6.53}$$

where the yearly trading day Δt and and log-returns $\vec{x}=[x_i]_{n\times 1}$ are assumed to be simply known directly from the market data and the **unknown UJD parameter set** $\vec{p}=[\widetilde{m},\widetilde{v},\lambda,a,b]$ need to be estimated by maximum likelihood or better methods.

In the **negative log-likelihood (NLLH)** form for minimizing code,

$$\mathbf{NLLH}_n(\vec{x};\vec{p}) = -\mathbf{LLH}_n(\vec{x};\vec{p}). \tag{6.54}$$

• *6.9 MLE for Linear Jump-Diffusion with a K-Jump Compound Poisson Continued with Jump-Amplitude Distribution:*

For market data that has a lower frequency than daily data,[a] e.g., weekly or monthly data, then the jump-diffusion has a **higher probability of more then one jump in $(t, t + k\Delta t)$**, where k is the number of days between data collections, and the jump probability is governed by the **full Poisson process** and not the Bernoulli process special case. So, instead of the Bernoulli jump-diffusion in (6.32), we get from (6.31) a compound Poisson jump-diffusion,

$$\mathrm{LR}_i = \Delta \mathrm{X}_i = \widetilde{m}\Delta t + \sqrt{\widetilde{v}\Delta t}Z_i + \sum_{j=1}^{\Delta P_i} Q_j, \tag{6.55}$$

where $\widetilde{m} \equiv \mu - \sigma^2/2$, $\widetilde{v} \equiv \sigma^2$ and ΔP_i is the Poisson increment counter for time-step Δt in place of "$k\Delta t$" and parameter $\Lambda = \lambda\Delta t$.

[a] Recall that the daily data has a $\Delta t \simeq 0.04$ that is only marginally small in trading year units.

Again, letting the Gaussian or general normal part be

$$\Delta G_i \equiv \widetilde{m}\Delta t + \sqrt{\widetilde{v}\Delta t}Z_i, \tag{6.56}$$

then the jump-diffusion log-return distribution, assuming an reasonable, finite upper limit K for the jump-count, will be

$$\begin{aligned} F^{(\mathrm{jd})}_{\Delta \mathrm{X}_i}(x) &\equiv \mathbf{Prob}[\Delta \mathrm{X}_i \le x] = \mathbf{Prob}\left[\Delta G_i + \textstyle\sum_{j=1}^{\Delta P_i} Q_j \le x\right] \\ &\overset{\mathrm{LTP}}{=} \textstyle\sum_{k=0}^{K} \mathbf{Prob}\left[\Delta G_i + \sum_{j=1}^{\Delta P_i} Q_j \le x \,\middle|\, \Delta P_i = k\right] \\ &\quad \cdot \mathbf{Prob}[\Delta P_i = k] \\ &= \textstyle\sum_{k=0}^{K} F_{\Delta P_i}(k)\mathbf{Prob}\left[\Delta G_i + \sum_{j=1}^{k} Q_j \le x\right]. \end{aligned} \tag{6.57}$$

where the count-limited Poisson discrete distribution, for $k \le K$, is

$$F_{\Delta P_i}(k) = \mathbf{Prob}[\Delta P_i = k] = \frac{\Lambda^k}{k! C_K}, \quad \text{where } C_K = \sum_{j=0}^{K} \frac{\Lambda^j}{j!}, \tag{6.58}$$

the C_K being the proper renormalization constant. One way of choosing K, by analogy with confidence intervals, is to require $F_{\Delta P_i}(K+1) < \alpha$ for some small, positive α.

Just as the $K=1$ case (6.45) leads to a single convolution distribution, the general K jump-case leads to a K-nested convolution,

$$
\begin{aligned}
F^{(\mathrm{jd})}_{\Delta \mathrm{X}_i}(x) &\equiv \mathrm{Prob}[\Delta \mathrm{X}_i \le x] = \mathrm{Prob}\left[\Delta G_i + \sum_{j=1}^{\Delta P_i} Q_j \le x\right] \\
&\overset{\mathrm{LTP}}{=} \sum_{k=0}^{K} \mathrm{Prob}\left[\Delta G_i + \sum_{j=1}^{\Delta P_i} Q_j \le x \,\middle|\, \Delta P_i = k\right] \mathrm{Prob}[\Delta P_i = k] \\
&= \sum_{k=0}^{K} F_{\Delta P_i}(k)\ \mathrm{Prob}\left[\Delta G_i + \sum_{j=1}^{k} Q_j \le x\right] \\
&= \sum_{k=0}^{K} F_{\Delta P_i}(k) \left(F^{(n)}_{\Delta G_i} (*f_Q)^K\right)(x)
\end{aligned}
\tag{6.59}
$$

where the nested convolution can be expanded several ways. For instance, with $K=2$, $(F^{(n)}_{\Delta G_i}(*f_Q)^2)(x) = (F^{(n)}_{\Delta G_i} * (f_Q * f_Q))(x)$ or $(F^{(n)}_{\Delta G_i}(*f_Q)^2)(x) = ((F^{(n)}_{\Delta G_i} * f_Q) * f_Q)(x)$, where $(f_Q * f_Q)(x)$ is the 2-jump distribution.[a]

[a] Hanson(2007), p. 157, the $K=2$ distribution is given and has be used for JD options.

By differentiation, the JD density is

$$f^{(\mathrm{jd})}_{\Delta \mathrm{X}_i}(x) = \sum_{k=0}^{K} F_{\Delta P_i}(k)\Big(f^{(n)}_{\Delta G_i}(*f_Q)^K\Big)(x) \tag{6.60}$$

Then, from Eq. (6.53), the log-likelihood function for the K-jump JD, assuming the same uniform JD and parameters, is

$$\begin{aligned}\mathbf{LLH}_n(\vec{x};\vec{p}) &= \textstyle\sum_{i=1}^{n} \log\Big(f^{(\mathrm{ujd})}_{\Delta \mathrm{X}_i}(x_i)\Big) \\ &= \textstyle\sum_{i=1}^{n} \log\Big(\sum_{k=0}^{K} F_{\Delta P_i}(k)\Big(f^{(n)}_{\Delta G_i}(*f_Q)^K\Big)(x)\Big),\end{aligned} \tag{6.61}$$

where the trading day Δt in years and and log-returns $[x_i]_{n\times 1}$ are assumed to be simply known directly from the market data and the unknown UJD parameter set $\vec{p}=[\widetilde{m},\widetilde{v},\lambda,a,b]$ need to be estimated by maximum likelihood or better methods.

It is important to keep the maximum jump-count K down to a few jump in the model, otherwise the statistical fitting will suffer.

For instance, for $K=2$, the log-likelihood takes the form:

$$\begin{aligned}\mathbf{LLH}_n(\vec{x};\vec{p}) = \sum_{i=1}^{n} \log\Big(\Big(f^{(n)}_{\Delta G_i}(x) + \Lambda\Big(f^{(n)}_{\Delta G_i} * f_Q\Big)(x) \\ + \tfrac{1}{2}\Lambda^2\Big(f^{(n)}_{\Delta G_i} * (f_Q * f_Q)\Big)(x)\Big)\Big/ C_2\Big),\end{aligned} \tag{6.62}$$

where $C_2 = 1+\Lambda+\frac{1}{2}\Lambda^2$. For $k=2$ jumps, the distribution is triangular on $(2a, 2b)$, double the original (a,b) interval,

$$(f_Q * f_Q)(x) = \frac{1}{(b-a)^2}\left\{\begin{array}{ll} x-2a, & 2a < x < a+b \\ 2b-x, & a+b \le x < 2b \\ 0, & otherwise \end{array}\right\}. \tag{6.63}$$

- **6.10 MLE Computational Issues:**
  - General MLE problems do not fit in with the generic **`mle` functions** for elementary distribution functions when there is a distribution mixture.
  - General MLE problems do not fit easily in with the **least squares (LS) functions** (e.g., **`lsqnonlin`**), especially for the log-likelihood formulation of non-separable terms.
  - One of the most general methods, **`fminsearch`**, is a basic **`MATLAB`** optimizer, so does not include the statistics output of MLE or LS functions.
  - The **`fminsearch`** function has to be reinterpreted for MLE: in the function **`help`**, the **`x`** is the MLE parameter vector, **`p=[mu,sigsq,lambda,a,b]`**, and so-called parameters are the input log-return data vector, **`y=[x,Dt]`**.
  - The fminsearch function does not take constraints, but **`{sigsq,lambda,-a,b}`** all need to be positive and **`MATLAB`** will not do it for the programmer, so test each parameter iterate at each call to function being minimized, resetting positivity new values.

- Hence, with the main code declared a function

```
function main
```
(6.64)

the negative log-likelihood subfunction

```
function f=NLLH(p)
```
(6.65)

could read, for example, as

```
global x Dt
f=-LLHn(x,p);
```
(6.66)

where the data can be passed by a global variables declared with the same names in both main and **NLLH** functions; the call to **fminsearch** in main should have the form,

```
[p,fval]=fminsearch(@NLLH,p0);
```
(6.67)

where **@NLLN** is the function handle for passing the function name as a calling function argument and **p0** is some good starting value for **p**, which can come by a pure Gaussian estimate using the log-return statistics, but **lambda** has to be positive as does **sigsq**.

- The **fminsearch** default options can be listed by a plain call to **optimset**, i.e., without input or output arguments.

** Reminder: Lecture 6 Homework Posted in Chalk Assignments, due by Lecture 7 in Chalk Assignments!*

*** Summary of Lecture 6:**

1. **Review of RobustFit Results**
2. **Review of N-Day and K-Day Problems**
3. **MLE Specialized Functions**
4. **MLE for Variable Δt_i Linear Diffusions**
5. **MLE for Linear Zero-One Jump-Diffusions**
6. **MLE for Linear One-Fixed Jump-Diffusions**
7. **Numerical Optimization and General `fminsearch`**
8. **MLE for Jump-Amplitude Distributed Jump-Diffusions**
9. **MLE for K-Jump Limited Jump-Diffusions**
10. **MLE Computational Issues**