

FinM 331/Stat 339 Financial Data Analysis

(Applied Statistical Analysis of Financial Data in MATLAB)

Winter 2009

Floyd B. Hanson, Visiting Professor

Email: t-9fhans@uchicago.edu

**Master of Science in Financial Mathematics Program
University of Chicago**

Lecture 6

6:30-9:30 pm, 09 February 2009, Ryerson 251 in Chicago

7:30-10:30 pm, 09 February 2009 at UBS Stamford

8:30-11:30 am, 10 February 2009, #02-01 Spring Singapore

6. More Maximum Likelihood Estimation:

6.0 Maximum Likelihood Estimation (MLE) Continued:^a

- ***6.1 MLE for Linear Diffusion with Variable Trading-Day Intervals:***

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.

^aContinuing 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,

$$\text{LR}_i = \Delta X_i = \Delta \log(A_i) = \tilde{m} \Delta t_i + \sqrt{\tilde{v} \Delta t_i} Z_i$$

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

The log-return distribution becomes,

$$\begin{aligned} F_{\Delta X_i}(\Delta x_i) &\stackrel{\text{alg}}{=} \text{Prob}[Z_i \leq (\Delta x_i - \tilde{m} \Delta t_i) / \sqrt{\tilde{v} \Delta t_i}] \\ &\stackrel{\mathcal{N}}{=} F_{Z_i}^{(n)} \left((\Delta x_i - \tilde{m} \Delta t_i) / \sqrt{\tilde{v} \Delta t_i}; 0, 1 \right) \end{aligned}$$

and upon differentiation with respect to Δx_i yields the i th likelihood function or density function form,

$$\begin{aligned} \text{LH}_i(\tilde{m}, \tilde{v}) = f_{\Delta X_i}(\Delta x_i) &= \frac{1}{\sqrt{\tilde{v} \Delta t_i}} f_{Z_i}^{(n)} \left(\frac{(\Delta x_i - \tilde{m} \Delta t_i)}{\sqrt{\tilde{v} \Delta t_i}}; 0, 1 \right) \\ &\stackrel{\mathcal{N}}{=} \frac{1}{\sqrt{2\pi \tilde{v} \Delta t_i}} \exp \left(-\frac{(\Delta x_i - \tilde{m} \Delta t_i)^2}{2\tilde{v} \Delta t_i} \right). \end{aligned}$$

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 $i = 1:n$,

$$\begin{aligned} \text{TLH}_n(\tilde{m}, \tilde{v}) &= f_{\Delta\vec{X}}(\Delta\vec{X}) \stackrel{\text{IID}}{=} \prod_{i=1}^n f_{\Delta X_i}(\Delta X_i) \\ &\stackrel{\mathcal{N}}{=} \prod_{i=1}^n \frac{\exp(-0.5(\Delta X_i - \tilde{m}\Delta t_i)^2 / (\tilde{v}\Delta t_i))}{\sqrt{2\pi\tilde{v}\Delta t_i}}. \end{aligned}$$

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

$$\begin{aligned} \text{LLH}_n(\tilde{m}, \tilde{v}) &= \log\left(f_{\Delta\vec{X}}(\Delta\vec{X})\right) = \sum_{i=1}^n \log(f_{\Delta X_i}(\Delta X_i)) \\ &= - \sum_{i=1}^n \left(\frac{(\Delta X_i - \tilde{m}\Delta t_i)^2}{2\tilde{v}\Delta t_i} + \frac{1}{2} \log(2\pi\tilde{v}\Delta t_i) \right), \end{aligned}$$

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

Seeking critical points^a,

$$\frac{\partial \text{LLH}_n}{\partial \tilde{m}}(\tilde{m}, \tilde{v}) = \sum_{i=1}^n \frac{(\Delta x_i - \tilde{m} \Delta t_i) \Delta t_i}{\tilde{v} \Delta t_i} \stackrel{*}{=} 0,$$

and

$$\frac{\partial \text{LLH}_n}{\partial \tilde{v}}(\tilde{m}, \tilde{v}) = \sum_{i=1}^n \left(\frac{(\Delta x_i - \tilde{m} \Delta t_i)^2}{2\tilde{v}^2 \Delta t_i} - \frac{1}{2\tilde{v}} \right) \stackrel{*}{=} 0,$$

gives the simultaneous estimates,

$$\hat{m} = \tilde{m}^* = \frac{\sum_{i=1}^n \Delta x_i}{\sum_{i=1}^n \Delta t_i} \equiv \overline{\Delta x} / \overline{\Delta t}$$

and

$$\begin{aligned} \hat{v} = \tilde{v}^* &= \frac{1}{n} \sum_{i=1}^n \frac{(\Delta x_i - \tilde{m}^* \Delta t_i)^2}{\Delta t_i} \\ &= \frac{1}{n} \sum_{i=1}^n \left(\frac{\Delta x_i}{\sqrt{\Delta t_i}} - \frac{\overline{\Delta x}}{\sqrt{\overline{\Delta t}}} \sqrt{\frac{\Delta t_i}{\overline{\Delta t}}} \right)^2 \equiv \hat{\sigma}_{\Delta x / \sqrt{\Delta t}}^2, \end{aligned}$$

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

^a*Correcting a typo by deleting a factor of 2 in the first equation of L5.40 that should have been cancelled by differentiation.*

Finally, converting back to standard model coefficient,

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

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

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

$$\hat{\sigma}_{\Delta x / \sqrt{\Delta t}}^2 = \hat{\sigma}_{\Delta x}^2 / \Delta t$$

so then

$$\hat{\sigma} = \hat{\sigma}_{\Delta x} / \Delta t \quad \& \quad \hat{\mu} = (\overline{\Delta x} + \hat{\sigma}_{\Delta x}^2 / 2) / \Delta t$$

like last lecture, but now with Δx instead of x before.

- **6.2 MLE for Linear Jump-Diffusion with Compound Poisson and Otherwise Constant Coefficients:**

Consider the asset price jump-diffusion model,

$$dA(t) = A(t)(\mu dt + \sigma dW(t)) + (1 - \delta_{dP(t),0}) \cdot \sum_{j=P(t)+1}^{(P+dP)(t)} \nu(Q_j) A(T_j^-),$$

where $\sigma > 0$, $\nu(Q) > -1$, $dP(t) = dP(t; Q)$,

$E[dP(t; Q)] = \lambda f_Q(q; a, b) dq dt > 0$, and T_j is the j th jump-time with $T_0 \equiv 0$. The Poisson jump-amplitude random variables Q_j are IID RVs with density $f_Q(q; a, b)$ on (a, b) . Using the 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

$$dX(t) = (\mu - \sigma^2/2) dt + \sigma dW(t) + (1 - \delta_{dP(t),0}) \cdot \sum_{j=P(t)+1}^{(P+dP)(t)} \log(1 + \nu(Q_j)).$$

However, the FinM 331 problem begins with the approximate discretized Gaussian-Poisson mixture model for sufficiently small time-steps Δt_i

^aHanson ('07) book, Chapters 4-5.

producing the log-return

$$\text{LR}_i = \Delta X_i = \widetilde{m} \Delta t_i + \sqrt{\widetilde{v} \Delta t_i} Z_i + (1 - \delta_{\Delta P_i, 0}) \cdot \sum_{j=1}^{\Delta P_i} Q_j,$$

where $\widetilde{m} \equiv \mu - \sigma^2/2$, $\widetilde{v} \equiv \sigma^2$ and $Q_j = \log(1 + \nu(Q_j))$ or $\nu(Q_j) = \exp(Q_j) - 1$ has been selected for underlying jump-amplitude simplicity. Also, we have dropped the initial Poisson count P_i from the jump-counter j for convenience of this one time-step model.

The log-return distribution is found using *General Probability Inversion (GPI)*^a and *Law of Total Probability (LTP)*^b,

^aHanson ('07) Online Appendix B, Lemma B.19, p. B13, which basically states that if X and Z are an RVs such that $X = g(Z)$ where g is increasing (so does not change the sign of the inequality) and invertible, then

$$\text{Prob}[X \leq x] = \text{Prob}[g(Z) \leq x] \stackrel{\text{GPI}}{=} \text{Prob}[Z \leq g^{-1}(x)].$$

^bHanson ('07) Online Appendix B, Sect. B.3.2, pp. B29-B30, which basically states that if X is a RV and $\{Y_k\}_{k=1}^{\infty}$ is a countable set of discrete RVs, then

$$\text{Prob}[X \leq x] \stackrel{\text{LTP}}{=} \sum_{k=1}^{\infty} \text{Prob}[X \leq x | Y_k] \text{Prob}[Y_k].$$

$$\begin{aligned}
F_{\Delta X_i}^{(jd)}(\Delta x_i) &\equiv \text{Prob}[\Delta X_i \leq \Delta x_i] \\
&= \text{Prob}\left[\tilde{m}\Delta t_i + \sqrt{\tilde{v}\Delta t_i}Z_i \right. \\
&\quad \left. + (1 - \delta_{\Delta P_i,0}) \cdot \sum_{j=1}^{\Delta P_i} Q_j \leq \Delta x_i\right] \\
&\stackrel{\text{GPI}}{=} \text{Prob}\left[Z_i \leq (\Delta x_i - \tilde{m}\Delta t_i \right. \\
&\quad \left. - (1 - \delta_{\Delta P_i,0}) \cdot \sum_{j=1}^{\Delta P_i} Q_j) / \sqrt{\tilde{v}\Delta t_i}\right].
\end{aligned}$$

This form is almost reduced to the normal distribution, but there are still the discrete Poisson RVs, ΔP_i , for which we can apply **LTP**, i.e,

$$\begin{aligned}
F_{\Delta X_i}^{(jd)}(\Delta x_i) &\stackrel{\text{LTP}}{=} \sum_{k=0}^{\infty} \text{Prob}\left[Z_i \leq (\Delta x_i - \tilde{m}\Delta t_i \right. \\
&\quad \left. - (1 - \delta_{\Delta P_i,0}) \cdot \sum_{j=1}^{\Delta P_i} Q_j) / \sqrt{\tilde{v}\Delta t_i} \mid \Delta P_i = k\right] \\
&\quad \cdot \text{Prob}[\Delta P_i = k],
\end{aligned}$$

but the underlying jump-amplitudes, Q_j , are usually continuous IID RVs and thus do not nicely fit into **LTP** form.

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

If the $Q_j = q_0$, where q_0 is a single fixed value and we have a *simple Poisson process* rather than a compound one, then, using the Poisson distribution

$$\text{Prob}[\Delta P_i = k] \equiv p_k(\lambda \Delta t_i) = \exp(-\lambda \Delta t_i) (\lambda \Delta t_i)^k / k!$$

and using the conditioning,

$$\begin{aligned} F_{\Delta X_i}^{(\text{jd})}(\Delta x_i) &\stackrel{\text{LTP}}{=} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \\ &\quad \cdot \text{Prob}[Z_i \leq (\Delta x_i - \tilde{m} \Delta t_i - k q_0) / \sqrt{\tilde{v} \Delta t_i}] \\ &\stackrel{\mathcal{N}}{=} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \\ &\quad \cdot F_{Z_i}^{(\text{n})}((\Delta x_i - \tilde{m} \Delta t_i - k q_0) / \sqrt{\tilde{v} \Delta t_i}; 0, 1) \\ &\stackrel{\text{ident.}}{=} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) F_{Z_i}^{(\text{n})}(\Delta x_i; \tilde{m} \Delta t_i + k q_0, \tilde{v} \Delta t_i) \end{aligned}$$

and differentiating produces in the i th likelihood function or density function mixture form,

$$\begin{aligned} \text{LH}_i(\tilde{m}, \tilde{v}, \lambda, q_0) &= f_{\Delta \mathbf{X}_i}^{(\text{jd})}(\Delta \mathbf{x}_i) \\ &= \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) f_{Z_i}^{(n)}(\Delta \mathbf{x}_i; \tilde{m} \Delta t_i + k q_0, \tilde{v} \Delta t_i), \end{aligned}$$

which depends on four (4) unknown parameters.

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

$$\begin{aligned} \text{TLH}_n(\tilde{m}, \tilde{v}, \lambda, q_0) &= f_{\Delta \vec{\mathbf{X}}}^{(\text{jd})}(\Delta \vec{\mathbf{x}}) \stackrel{\text{IID}}{=} \prod_{i=1}^n f_{\Delta \mathbf{X}_i}^{(\text{jd})}(\Delta \mathbf{x}_i) \\ &= \prod_{i=1}^n \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \\ &\quad \cdot f_{Z_i}^{(n)}(\Delta \mathbf{x}_i; \tilde{m} \Delta t_i + k q_0, \tilde{v} \Delta t_i) \end{aligned}$$

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

$$\begin{aligned} \text{LLH}_n(\tilde{m}, \tilde{v}, \lambda, q_0) &= \sum_{i=1}^n \log \left(f_{\Delta X_i}^{(\text{jd})}(\Delta x_i) \right) \\ &= \sum_{i=1}^n \log \left(\sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \right. \\ &\quad \left. \cdot f_{Z_i}^{(n)}(\Delta x_i; \tilde{m} \Delta t_i + k q_0, \tilde{v} \Delta t_i) \right), \end{aligned}$$

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

For numerical computing the infinite Poisson sum must be reduced to a finite sum, so we define the $(K + 1)$ -term truncated Poisson **log-likelihood**,

$$\begin{aligned} \text{LLH}_{n,K}(\tilde{m}, \tilde{v}, \lambda, q_0) &= \sum_{i=1}^n \log \left(\sum_{k=0}^K p_k(\lambda \Delta t_i) \right. \\ &\quad \left. \cdot f_{Z_i}^{(n)}(\Delta x_i; \tilde{m} \Delta t_i + k q_0, \tilde{v} \Delta t_i) \right. \\ &\quad \left. / P_K(\lambda \Delta t_i) \right), \end{aligned}$$

where $P_K(\lambda\Delta t_i) \equiv \sum_{\ell=0}^K p_\ell(\lambda\Delta t_i)$ is the renormalization factor that preserves probability and prevents peculiar behavior for small K , such as pathological critical points.

The simplest case is the $K = 0$ no-jump case.

$$\text{LLH}_{n,0}(\tilde{m}, \tilde{v}, \lambda, q_0) = \sum_{i=1}^n \log \left(f_{Z_i}^{(n)}(\Delta \mathbf{x}_i; \tilde{m}\Delta t_i, \tilde{v}\Delta t_i) \right),$$

but we already know the answer to that for pure diffusion problem where $(\hat{m}, \hat{v}) = \left(\overline{\Delta \mathbf{x}} / \overline{\Delta t}, \hat{\sigma}_{\Delta \mathbf{x} / \sqrt{\Delta t}}^2 \right)$.

For $K = 1$, the one-jump case and abbreviating $p_k = p_k(\lambda\Delta t_i)$ and $f_k = f_{Z_i}^{(n)}(\Delta \mathbf{x}_i; \tilde{m}\Delta t_i + kq_0, \tilde{v}\Delta t_i)$,

$$\text{LLH}_{n,1}(\tilde{m}, \tilde{v}, \lambda, q_0) = \sum_{i=1}^n \log((p_0 f_0 + p_1 f_1) / (p_0 + p_1))$$

and, for instance assuming p_1/p_0 is small enough to expand,

$$\begin{aligned}
 \frac{\partial \text{LLH}_{n,1}}{\partial \tilde{m}}(\tilde{m}, \tilde{v}, \lambda, q_0) &= \sum_{i=1}^n (p_0 f_0 \cdot (\Delta \mathbf{x}_i - \tilde{m} \Delta t_i) / \tilde{v} \\
 &\quad + p_1 f_1 \cdot (\Delta \mathbf{x}_i - \tilde{m} \Delta t_i - q_0) / \tilde{v}) \\
 &\quad / (p_0 f_0 + p_1 f_1) \\
 &= \frac{1}{\tilde{v}} \sum_{i=1}^n \left(\left(1 - \frac{p_1 f_1}{p_0 f_0} \right) \cdot (\Delta \mathbf{x}_i - \tilde{m} \Delta t_i) \right. \\
 &\quad \left. + \frac{p_1 f_1}{p_0 f_0} \cdot (\Delta \mathbf{x}_i - \tilde{m} \Delta t_i - q_0) \right) \\
 &= \frac{1}{\tilde{v}} \sum_{i=1}^n \left(\Delta \mathbf{x}_i - \tilde{m} \Delta t_i - \frac{p_1 f_1}{p_0 f_0} q_0 \right) \\
 &= \frac{1}{\tilde{v}} \left(\overline{\Delta \mathbf{x}} - \tilde{m} \overline{\Delta t} - \lambda q_0 \sum_{i=1}^n \Delta t_i \right. \\
 &\quad \left. \cdot \exp((2q_0(\Delta \mathbf{x}_i - \tilde{m} \Delta t_i) - q_0^2) / (2\tilde{v} \Delta t_i)) \right),
 \end{aligned}$$

where in the last two lines the terms in q_0 give the one-jump correction.

- **6.4 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_1 will be small compared to the zero-jump probability p_0 , recalling our multijump results for recent S&P 500 Index log-returns.

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

^aSee **help fminsearch** in MATLAB command window or search for **fminsearch** in MATAB 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 1: 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 ($f = \text{const}$, or $\min = -\infty$)	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 2: 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**,

$$x^* = \min_x [f(x)],$$

given an initial multivariable start **x0** and objective function **f** appearing as the first argument as the pointer or handle **@f** usually pointing to a subfunction within the main function m-file.

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. *{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. 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.
3. **Constraints** can be embedded in function directly or by sufficiently large values not mistaken for a minimum.
4. 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.
5. **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 solution (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.message** is exit message.

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

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

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 $F_j \leq F_{j+1}$;
2. The vector \vec{x}_{m+1} of the largest value is replaced by a *reflection*, $\vec{r} = 2\bar{x}_m - \vec{x}_{m+1}$, about the center, $\bar{x}_m = \sum_{k=1}^m \vec{x}_k / m$, of the rest of simplex;

^aSee 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. Optimization, 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*, $\vec{s} = 2\vec{r} - \bar{x}_m$ 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* ($\vec{c}\vec{c}$), *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.

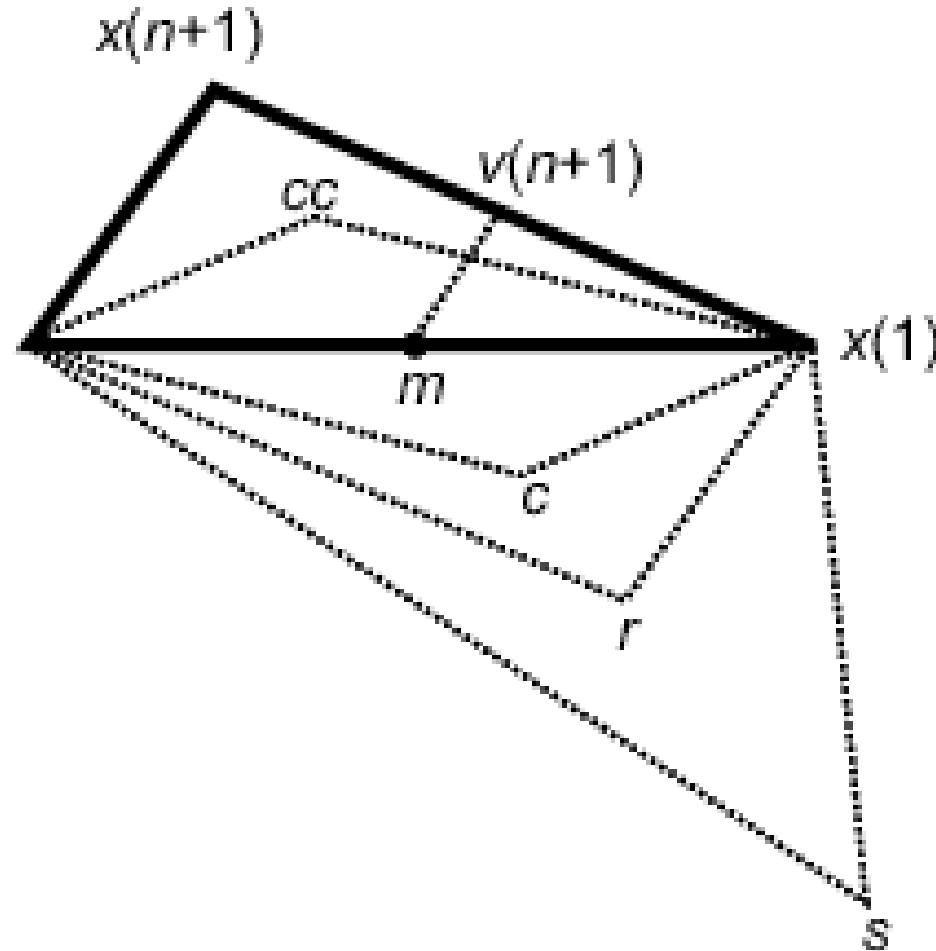


Figure 3: MATLAB Down-Hill Simplex Algorithm Illustration: *Fminsearch Algorithm, Unconstrained Nonlinear Optimization page, Optimization Toolbox, 2008. Triangular with $(x(1), x(2), x(n + 1) \approx x(3))$.*

- **6.5 MLE for Linear Jump-Diffusion with Compound Poisson Continued with Discretized Approximation to Jump-Amplitude Distribution Part:**

Previously we considered the partially reduced form to the normal distribution, but there are still the underlying Poisson jump-amplitude IID RVs, Q_j , in the jump-diffusion distribution,

$$F_{\Delta X_i}^{(jd)}(\Delta x_i) \stackrel{\text{LTP}}{\underset{\Delta P_i}{=}} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \text{Prob}[Z_i \leq (\Delta x_i - \tilde{m} \Delta t_i - (1 - \delta_{k,0}) \cdot \sum_{j=1}^k Q_j) / \sqrt{\tilde{v} \Delta t_i}],$$

where we have substituted, into the second equation on p. 9, the conditioning $\Delta P_i = k$ into the distribution for Z_i and the distribution $\text{Prob}[\Delta P_i = k] = p_k(\lambda \Delta t_i)$, but the underlying the Q_j are usually continuous IID RVs and thus do not nicely fit into **LTP** form.

The solution comes not from an **LTP** Poisson sum, but as an integral. Rather than write down the answer for FinM 331 and Stat 339, the result will be derived from the limit of a discrete approximation as a demonstration of an important technique that is often needed for complicated transformations of continuous RVs, in fact, the method underlies many results in stochastic calculus but is rarely revealed in the lattice of abstract theorems.

Consider the IID RV Q with a *discrete uniform distribution* on $[a, b]$, though it could be any reasonable distribution, but the uniform distribution is the most basic distribution of finite range and the market jump-amplitude distribution is essentially unknown.

Let

so $q_1 = a$, $q_N = b$ and $\Delta q_N = (b - a)/(N - 1)$, while

$$\text{Prob}[Q = q_i] = \frac{1}{N} = p_i^{(u)} = \frac{(N - 1)\Delta q_N}{N(b - a)} = \frac{(N - 1)}{N} f_Q^{(u)}(q; a, b)\Delta q_N,$$

so $\mathbf{E}_Q[1] = \sum_{i=1}^N 1/N = 1$, probability is conserved.

Next let $\vec{Q}^{(k)} = [Q_i^{(k)}]_{k \times 1}$ be a k -vector RV, $\vec{q}_j^{(k)} = [q_{i,j}^{(k)}]_{N \times 1}$ be the possible realizations of the j th RV for $j = 1:k$ and alternately $\vec{q}_\ell^{(k,N)} = [q_{\ell,i}^{(k,N)}]_{k \times 1}$ be the realizations of the ℓ th-vector RV for $\ell = 1:N$, Then, using a generalization of *LTP* with conditioning on an IID vector, being sure to cover all combinations (you can check the $k = 1$ case,

$$\text{Prob}[X] \stackrel{\text{IID}}{=} \prod_{j=1}^k \left(\sum_{\ell=1}^N \text{Prob} \left[Q_j^{(k)} = q_{\ell,j}^{(k,N)} \right] \right) \text{Prob} \left[X | \vec{Q}^{(k)} = \vec{q}_\ell^{(k,N)} \right]$$

and applying this model yields,

$$F_{\Delta X_i}^{(\text{jd})}(\Delta x_i) \stackrel{\text{IID}}{=} \lim_{N \rightarrow \infty} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \prod_{j=1}^k \left(\sum_{\ell=1}^N \text{Prob} \left[Q_j^{(k)} = q_{\ell,j}^{(k,N)} \right] \right) \text{Prob} \left[Z_i \leq (\Delta x_i - \tilde{m} \Delta t_i - (1 - \delta_{k,0}) \mathbf{1}^\top \vec{Q}^{(k)}) / \sqrt{\tilde{v} \Delta t_i} \mid \vec{Q}^{(k)} = \vec{q}_\ell^{(k,N)} \right].$$

Upon substituting the vector component conditioning and finite difference notation, then

$$\begin{aligned}
 F_{\Delta \mathbf{X}_i}^{(\text{jd})}(\Delta \mathbf{x}_i) = & \lim_{N \rightarrow \infty} \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \\
 & \cdot \left(\prod_{j=1}^k \left(\sum_{\ell=1}^N \frac{(N-1)}{N} \Delta q_N f_Q^{(\text{u})}(q_{\ell,j}^{(\text{k})}; a, b) \right) \right) \\
 & \cdot \text{Prob} \left[Z_i \leq (\Delta \mathbf{x}_i - \tilde{m} \Delta t_i \right. \\
 & \left. - (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}_\ell^{(\text{k},N)}) / \sqrt{\tilde{v} \Delta t_i} \right]
 \end{aligned}$$

and letting $N \rightarrow \infty$, $\Delta q_N \rightarrow dq_j$, an infinitesimal element of integration,

$$\begin{aligned}
 F_{\Delta \mathbf{X}_i}^{(\text{jd})}(\Delta \mathbf{x}_i) &= \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \left(\prod_{j=1}^k \int_a^b dq_j f_Q^{(\text{u})}(q_j; a, b) \right) \\
 &\quad \cdot \text{Prob}[Z_i \leq (\Delta \mathbf{x}_i - \tilde{\mathbf{m}} \Delta t_i \\
 &\quad - (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}) / \sqrt{\tilde{\mathbf{v}} \Delta t_i}] \\
 &= \sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \left(\prod_{j=1}^k \int_a^b dq_j f_Q^{(\text{u})}(q_j; a, b) \right) \\
 &\quad \cdot F_Z^{(\text{n})}(\Delta \mathbf{x}_i; \tilde{\mathbf{m}} \Delta t_i + (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}, \tilde{\mathbf{v}} \Delta t_i),
 \end{aligned}$$

after converting to the normal distribution and transforming parameters.

Further, differentiating to obtain the density and taking products over time index i to get the total likelihood (\mathbf{TLH}_n),

$$\begin{aligned}
 \mathbf{TLH}_n &\equiv \mathbf{TLH}_n(\tilde{m}, \tilde{v}, \lambda, a, b; \vec{\Delta x}) \\
 &= \prod_{i=1}^n f_{\Delta \mathbf{X}_i}^{(\text{jd})}(\Delta \mathbf{x}_i) \\
 &= \prod_{i=1}^n \left(\sum_{k=0}^{\infty} p_k(\lambda \Delta t_i) \cdot \left(\prod_{j=1}^k \int_a^b dq_j f_Q^{(\text{u})}(q_j; a, b) \right) \right. \\
 &\quad \left. \cdot f_Z^{(\text{n})}(\Delta \mathbf{x}_i; \tilde{m} \Delta t_i + (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}, \tilde{v} \Delta t_i) \right) \\
 &\simeq \mathbf{TLH}_n^{(\text{K})}(\tilde{m}, \tilde{v}, \lambda, a, b; \vec{\Delta x}),
 \end{aligned}$$

where the $(K + 1)$ truncation and Poisson renormalization of the infinite Poisson sum is

$$\begin{aligned} \mathbf{TLH}_n^{(K)} &\equiv \mathbf{TLH}_n^{(K)}(\tilde{m}, \tilde{v}, \lambda, a, b; \vec{\Delta x}) \\ &\equiv \prod_{i=1}^n \left(\sum_{k=0}^K \frac{p_k(\lambda \Delta t_i)}{P_K(\lambda \Delta t_i)} \cdot \left(\prod_{j=1}^k \int_a^b dq_j f_Q^{(u)}(q_j; a, b) \right) \right. \\ &\quad \left. \cdot f_Z^{(n)}(\Delta x_i; \tilde{m} \Delta t_i + (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}, \tilde{v} \Delta t_i) \right). \end{aligned}$$

In the Hanson book ('07) book the form is given in terms of the theory of convolutions which uses nested integral operators resembling autocorrelations. The approach here is more direct.

Note that when $K = 0$, we again get the *zero jump, normal (Gaussian) distribution, 2-parameter model*,

$$\mathbf{TLH}_n^{(0)}(\tilde{m}, \tilde{v}; \vec{\Delta x}) \equiv f_Z^{(n)}(\vec{\Delta x}; \tilde{m} \Delta t_i, \tilde{v} \Delta t_i),$$

so a nonlinear least squares problem when logarithms are use to transform $\mathbf{TLH}_n^{(0)}$ into the log-likelihood $\mathbf{LLH}_n^{(0)}$.

When $K = 1$, we get the *zero or one jump, Poisson-Gaussian mixture, 5-parameter model*, approximately valid if

$$P_1(\lambda\Delta t_i) = \exp(-\lambda\Delta t_i)(1 + \lambda\Delta t_i) \leq 1 - \alpha,$$

for positive and sufficiently small α , with

$$\begin{aligned} \text{TLH}_n^{(1)} &\equiv \text{TLH}_n^{(1)}(\tilde{m}, \tilde{v}, \lambda, a, b; \vec{\Delta x}) \\ &\equiv \prod_{i=1}^n \left(\frac{p_0(\lambda\Delta t_i)}{P_1(\lambda\Delta t_i)} f_{Z_i}^{(n)}(\Delta x_i; \tilde{m}\Delta t_i, \tilde{v}\Delta t_i) \right. \\ &\quad \left. + \frac{p_1(\lambda\Delta t_i)}{P_1(\lambda\Delta t_i)} \int_a^b dq_1 f_Q^{(u)}(q_1; a, b) \right. \\ &\quad \left. \cdot f_{Z_i}^{(n)}(\Delta x_i; \tilde{m}\Delta t_i + q_1, \tilde{v}\Delta t_i) \right). \end{aligned}$$

However, due to the analytical complexities of sums and integrals, taking logarithms for $\text{LLH}_n^{(1)}$ merely turns the product operation $\prod_{i=1}^n$ into a sum operation $\sum_{i=1}^n$ over essentially a single logarithm, but it should be better to work with the log-likelihood form due to the property of the logarithm to spread out the dependence.

The higher order approximations are similar, but with more Poisson terms and higher dimension jump-amplitude integrals. The log-likelihood approximation $\mathbf{LLH}_n^{(K)}$ allowing at most K jumps has the form

$$\begin{aligned} \mathbf{LLH}_n^{(K)} &= \mathbf{LLH}_n^{(K)}(\tilde{\mathbf{m}}, \tilde{\mathbf{v}}, \lambda, a, b; \vec{\Delta \mathbf{x}}) \\ &\equiv \sum_{i=1}^n \left(\log \left(p_{0,i} \cdot f_{Z_i,0}^{(n)} + \sum_{k=1}^K p_{k,i} \cdot \left(\prod_{j=1}^k \int_a^b dq_j f_{Q_j}^{(u)} \right) \cdot f_{Z_i,k}^{(n)} \right) \right. \\ &\quad \left. - \log(P_{K,i}) \right), \end{aligned}$$

where

$$p_{k,i} \equiv p_k(\lambda \Delta t_i); \quad P_{K,i} \equiv P_K(\lambda \Delta t_i);$$

$$f_{Z_i,k}^{(n)} \equiv f_{Z_i}^{(n)}(\Delta \mathbf{x}_i; \tilde{\mathbf{m}} \Delta t_i + (1 - \delta_{k,0}) \mathbf{1}^\top \vec{q}, \tilde{\mathbf{v}} \Delta t_i);$$

$$f_{Q_j}^{(u)} \equiv f_{Q_j}^{(u)}(q_j; a, b) \stackrel{\text{IID}}{=} f_Q^{(u)}(q_j; a, b);$$

approximately valid if

$$P_K(\lambda \Delta t_i) \equiv \exp(-\lambda \Delta t_i) \sum_{k=0}^K (\lambda \Delta t_i)^k / k! \leq 1 - \alpha, \text{ for}$$

positive and sufficiently small α , so $\log(P_{K,i}) \leq \log(1 - \alpha) \simeq -\alpha$.