Important: **Most of the analyses below use a counting window ('boxWidth') = 50 ms. This is the default that is used by 'VarVsMean'. This is a relatively short window, and is good for getting good temporal resolution, which was important to me when I first wrote this code.**

However, I would strongly suggest using a longer window (~100 ms) for most analyses. Longer time windows make real effects larger, and reduce the impacts of artifacts related to non-Poisson spiking. Just set 'fanoParams.boxWidth = 100;'

Of course, you should still run the examples below using the boxWidths indicated. But when you apply the code to your own data, I would use a 100 ms window unless you have a reason to go shorter.

For advice on individual functions: 'help functionName'.

If you have difficulty cutting and pasting lines into matlab from the .pdf (I do), try the .doc or the .txt file.

There is far more information in here than you likely need. Many of the exercises are devoted to demonstrating that the mean-matched FF is largely immune to various artifacts (or in a couple cases, to pointing out potential artifacts and how to avoid them). If you just want to get started quickly, Exercise #1 may be all you care about.

Exercise #1. Perform Fano Factor analysis on PMDdata

```
load PMDdata    % PMD dataset from monkey G
                  % target onset is at 400 ms
                  % all delays > =  400 ms
times = 200:25:850;  % from 200 ms before target onset until 450 ms after.
fanoParams.alignTime = 400;    % this time will become zero time
fanoParams.boxWidth = 50;     % 50 ms sliding window.
Result = VarVsMean(PMDdata, times, fanoParams);
plotFano(Result);
```

Black trace at bottom (with flanking 95% CI's) is the mean-matched population Fano-factor. Grey trace at top is the mean firing rate, across all trials/neuron/conditions. Black trace at top is the mean firing rate following mean-matching (see Exercise #2 to see distributions of firing rate counts before and after matching).

The left hand side of the horizontal calibration bar is the time of target onset.

Type 'help VarVsMean' for a description of how the data should be formatted.

Exercise #2 Scatterplots for three times

Do exercise #1 above, create 'Result'.

```
scatterParams.axLim = 'auto';
```

```
scatterParams.axLen = 5;
plotScatter(Result, -100, scatterParams);
text(2.5, 7, '100 ms before target', 'hori', 'center');
plotScatter(Result, 100, scatterParams);
text(2.5, 7, '100 ms after target', 'hori', 'center');
plotScatter(Result, 300, scatterParams);
text(2.5, 7, '300 ms after target', 'hori', 'center');
```

The scatterplot is of the variance of the spike count (in a window centered on the chosen time) versus the mean of the spike count. One dot per neuron/condition.

Grey distribution is for the mean count, and is on a log vertical scale. Black distribution is the 'core' distribution present for all times (all times in Result.times).

Black dots are those remaining after 'mean-matching': data points are thrown away until only the core distribution remains. This insures that the distribution of mean rates is virtually identical (within the resolution of the bins) for all the tested times.

Grey and black lines are the slope (via regression) for, respectively, all datapoints and the mean-matched datapoints. These lines are flanked by (difficult to see) 95% confidence intervals.

Exercise #3 Movie

Do exercise #1 above, create 'Result'.

```
ScatterMovie(Result);
```

Type 'help ScatterMovie' for advice on saving as .ai or .mgp

Exercise #4 The change in the Fano factor is not an artifact of non-Poisson spiking statistics.

One worries that the Fano factor may change simply because rates change, and spiking is not perfectly Poisson (e.g. the refractory period may regularize firing at high rates).

The mean-matched Fano factor avoids this problem, because the *analyzed* distribution of mean rates *does not change* (see Exercise #2 above).

To demonstrate this, we 'Fakerize' the data to remove any across trial variability in the underlying rate, leaving only spiking variability. Following this, the Fano factor should *not* drop (unless there is an artifact).

Do Exercise #1. Then:

```
Result = VarVsMean(Fakerize(PMDdata), times, fanoParams);  % takes a while
plotFanoParams.plotRawF = 1;
plotFano(Result, plotFanoParams);
```

The drop in the Fano factor goes away. This is true both for the mean-matched population Fano factor (black) and for the 'raw' population Fano factor (grey). This is reassuring, but not deeply surprising, as Fakerize produces Poisson statistics by default. It is more informative to try gamma (order 2) statistics, which are quite non-Poisson (much more regular for high rates / short windows).

```
Result = VarVsMean(Fakerize(PMDdata, 'gamma'), times, fanoParams);
plotFano(Result, plotFanoParams);
```

Gamma spiking statistics produce an artifactual drop in the Fano factor, as firing rates rise, but only for the 'raw' Fano factor (grey). The mean-matched Fano factor stays constant, as it should.

This is a good time to point out that the artifact seen above (in the raw Fano factor) also appears to plague the real (non-Fakerized) data:

```
Result = VarVsMean(PMDdata, times, fanoParams);
plotFano(Result, plotFanoParams);
```

Unless firing rates are changing dramatically (or spiking is very non-Poisson) a large drop in the raw Fano factor is unlikely to be entirely artifact. However, the exact timecourse of the raw Fano factor may not be trustworthy, especially if rates are changing the most at the most critical time. This is the central motivation for the mean-matched Fano factork, which distorts the timecourse minimally (see Exercises below).

Exercise #5 Analysis of simulated data I: the mean-matched Fano factor tracks the timecourse of variability.

```
% be patient, these take a minute or two
demoParams.decayRate = 50;
Demo(demoParams);
demoParams.decayRate = 150;
Demo(demoParams);
demoParams.decayRate = 400;
Demo(demoParams);
```

Each call to 'Demo' creates two plots. The first shows an example simulated neuron/condition (1 of 2000). There is a step change in mean rate at 400 ms. Each of the 50 trials has a different underlying 'true' rate. This rate is transformed into a spike train via a Poisson process (with a 1 ms refractory period enforced by the data format).

The second plot shows the mean-matched Fano factor, which approximately captures the time-course of the underlying variance (red, averaged across all simulated conditions). That timecourse is not quite as well captured by the raw (not mean-matched) Fano factor (grey).

To make a movie from the simulated data:

```
demoParams.decayRate = 250; % similar rate of decline to PMD dataset.
[SimData, SimResult] = Demo(demoParams);
ScatterMovie(SimResult);
```

Exercise #6 Analysis of simulated data II: the mean-matched Fano factor tracks the timecourse of variability, even when spiking statistics are non-Poisson.

```
% be patient, these take a minute or two
demoParams.decayRate = 250;
demoParams.process = 'poisson';  % (1 ms refractory period enforced by data format)
Demo(demoParams);
demoParams.process = 'refractory'; % 2 ms refractory period
Demo(demoParams);
demoParams.process = 'gamma'; % gamma order 2
Demo(demoParams);
```

The mean-matched Fano factor tracks the timecourse rather well in each case. The 'raw' non-mean-matched Fano factor does less well, especially when spiking statistics depart from Poisson.

Exercise #7 Analysis of simulated data III: the mean-matched Fano factor is constant when there is no across-trial variability (even if rates rise and spiking is non-Poisson)

```
% be patient, these take a minute or two
demoParams.noiseLevel = 0;
demoParams.process = 'poisson';  % (1 ms refractory period enforced by data format)
Demo(demoParams);
demoParams.process = 'refractory'; % 2 ms refractory period
Demo(demoParams);
demoParams.process = 'gamma'; % gamma order 2
Demo(demoParams);
```

The 'raw' Fano factor (grey) does show an artifactual drop due to rising firing rates when spiking statistics are non-poisson.

Exercise #8 Analysis of simulated data IV: the mean-matched Fano factor is constant when across-trial variability is constant.

```
% be patient, these take a minute or two
demoParams.noiseLevel = 1;
demoParams.decayRate = 10^6;  % variability stays high
demoParams.process = 'poisson';  % (1 ms refractory period enforced by data format)
```

```
Demo(demoParams);
demoParams.process = 'refractory'; % 2 ms refractory period
Demo(demoParams);
demoParams.process = 'gamma'; % gamma order 2
Demo(demoParams);
```

As in Exercise #7, the raw Fano factor shows an artifactual drop. This is because 1) spiking becomes more regular at high rates (esp. for 'refractory' and 'gamma') and 2) by computing the slope, we are effectively normalizing the 'true' variability. Normalization has a larger effect at higher rates. The mean-matched Fano factor avoids these problems.

Exercise #9 The timecourse of variability in PMd and MT.

Both PMd and MT show stimulus-driven drops in the Fano factor, but with different timecourses:

```
load PMDdata; load MTdata
times = 200:25:850;
fanoParams.alignTime = 400; fanoParams.boxWidth = 50;

Result_PMD = VarVsMean(PMDdata, times, fanoParams);
Result_MT =  VarVsMean(MTdata, times, fanoParams);

plotFano(Result_PMD);  plotFano(Result_MT);
```

Exercise #10 Movies for PMd and MT.

Do exercise #9 above, then:

```
ScatterMovie(Result_PMD);
ScatterMovie(Result_MT);
```

We can impose our own scalings & axis parameters.
For PMd:

```
scatterPlotParams.axLen = 5;   % passed through 'ScatterMovie' to 'plotScatter'
scatterPlotParams.axLim = 'auto';
ScatterMovie(Result_PMD, Result_PMD.times, scatterPlotParams);
```

For MT:

```
scatterPlotParams.axLen = 12;
scatterPlotParams.axLim = 'auto';
```

```
scatterPlotParams.HaxisP.tickLocations = 0:3:12;  % passed all the way to 'AxisMMC'
scatterPlotParams.VaxisP.tickLocations = 0:3:12;
ScatterMovie(Result_MT, Result_MT.times, scatterPlotParams);
```

Exercise #11 The mean-matched Fano factor can misbehave if the assumptions of the analysis are seriously violated.

```
demoParams.noiseLevel = 0;     % no real noise (we are looking for an artifact).
demoParams.process = 'gamma';     % very non-Poisson
demoParams.baselineRateRange = [10 11];    % similar for every neuron/condition
demoParams.finalRateRange = [20 21];        % no overlap with baseline
demoParams.numTrials = 5;     % few trials = poor estimate of actual mean count
demoParams.numConds = 5000;     % need more conditions (due to few trials)
Demo(demoParams)
```

Even though 'true' across trial variability is constant at zero, there is a small artifactual rise in the mean-matched Fano factor (and one in the opposite direction for the raw Fano factor).

The artifact in the mean-matched Fano factor occurs because we have violated two assumptions of the analysis. First, spiking-statistics are non-Poisson. Second, there is no overlap in the distribution of mean rates pre and post target (all neurons./conditions have almost the same pre-target rate, and almost the same post-target rate). The only overlap is apparent, due to sampling error resulting from the low trial count. Thus, the attempt at mean-matching fails.

The above artifact depends on violating both assumptions simultaneously. Violating one or the other is more benign.

After running the above, run:

```
demoParams.process = 'poisson';  % poisson instead of gamma, keep other params
Demo(demoParams)
```

Little artifact is produced (same is true if using 'refractory').

```
demoParams.process = 'gamma';   % back to (the problematic) gamma
demoParams.baselineRateRange = [10 15];    % but now with wider firing rate range…
demoParams.finalRateRange = [13 18];        % …and some overlap
Demo(demoParams)
```

Only a very small artifact is produced. There is no artifact if more trials are used:

```
demoParams.numTrials = 50;     % more trials = better estimate of actual mean count
demoParams.numConds = 2000;  % don't need as many with more trials
Demo(demoParams)
```

If you try this high trial count with the non-overlapping ranges ([10 11] & [20 21]) from the very first simulation above (the one with the sizeable artifact) then almost no points survive the mean-matching.

In summary, a sizable artifact is produced only if *all* of the following are satisfied: 1) spiking statistics are very non-Poisson, 2) firing rates are very similar for every neuron/condition, 3) there is little or no overlap in the distribution of firing rates pre and post-stimulus, 4) few trials/condition are collected (so that there appears to be overlap where there isn't really).

Of course, there may be other ways to produce an artifact, but it appears harder than one might initially think.

Exercise #12: Artifacts are possible when there is *constant* real variability *and* the trial count is low *and* rates rise sharply.

Mean-matching still makes things much better (relative to the raw FF) but can fail to completely eliminate artifacts due to rising rates E.g.,

First lets examine a situation where everything is still ok:
Rates rise sharply and trial count is low, but there is no true across-trial variability

```
demoParams.noiseLevel = 0;  % no problem when set low
demoParams.decayRate = 10^10;    % no decay
demoParams.baselineRateRange = [0 15];
demoParams.finalRateRange = [0 50];
demoParams.numTrials = 5;
demoParams.numConds = 5000;
Demo(demoParams);   % everything looks good (no change in mean-matched FF)
```

Thus low trial counts, on their own, are no problem.

To create an artifact that is not completely eliminated by the mean-matched FF:

```
demoParams.noiseLevel = 1;  % constant high variability
Demo(demoParams);   % now there is a sizeable artifact (worse with old  regression method)
```

The problem is due to the fact that the FF is (implicitly) normalized. Thus, the same amount of variability seems like less as rates rise. This is a serious problem for the raw FF. The mean-matched FF *should* eliminate this effect, but is imperfect for low trial-counts. This is because the mean cannot be perfectly matched when it cannot be accurately estimated.

For example, things are much better if we have higher trial counts and less extreme changes in rate:

```
demoParams.noiseLevel = 1;
```

```
demoParams.decayRate = 10^10;    % no decay
demoParams.baselineRateRange = [0 25];
demoParams.finalRateRange = [0 40];
demoParams.numTrials = 15;
demoParams.numConds = 5000;
Demo(demoParams);
```

The raw FF still shows an artifact. The artifact is much smaller (though not completely gone) for the mean-matched FF.

The take home lesson is: if you have low trial counts and large changes in mean rate, be skeptical of declines in the FF. Fortunately, you can act on your skepticism, and check the variance minus the mean (as opposed to the FF = variance / mean).

The (non mean-matched) variance minus the mean is not impacted by the above artifact. To see this run:

```
demoParams.noiseLevel = 1;  % these are the params that caused the problem above
demoParams.decayRate = 10^10;    % no decay
demoParams.baselineRateRange = [0 15];
demoParams.finalRateRange = [0 50];
demoParams.numTrials = 5;
demoParams.numConds = 5000;

[SimData, Result, trueVar] = Demo(demoParams);

fanoParams.includeVarMinusMean = 1;
fanoParams.alignTime = 400;
fanoParams.boxWidth = 100;
times = 200:25:850;
Result = VarVsMean(SimData, times, fanoParams);
% plot the variance minus the mean (not mean matched)
figure;
plot(Result.times, Result.VMMall_95CIs, 'k'); axis([-200 450 0 2]);
```

The variance minus the mean is not impacted by this artifact because it is not normalized.

Thus, if you are operating in the problematic regime (large increases in rate, but trial counts too low for mean matching to be effective), you should not trust and FF decline if there is no concurrent decline in the variance-mean.