

## 2.2 Why Statistical Arbitrage Trades Break Down

The way a pairs trade is supposed to work is that a trade entry is signaled by a significant divergence in the spread between the prices of a correlated pair of stocks. The higher value stock is sold while the lower value stock is purchased, with various schemes used to set the hedge ratio (matching dollar value, or beta-neutral, for example). As the spread begins to converge, the higher value stock declines relative to the lower value stock, yielding a profit on the trade. The trade is exited when the spread converges to some specified level, typically a (lower) multiple of its standard deviation.

One of the risks of statistical arbitrage strategies is that individual pairs trades break down quite frequently. Even in a successful trade the spread will often continue to diverge before ultimately converging to produce an overall profit. But sometimes the trade breaks down altogether - the spread continues to diverge until the market-to-market losses force the trader to unwind the position.

The risk of breakdown in an individual pairs trade is mitigated by trading a large portfolio of pairs, together with sensible stop-loss limits and risk management protocols. But the losses on pairs trades that break down can be painful, especially in periods when several pairs diverge at the same time.

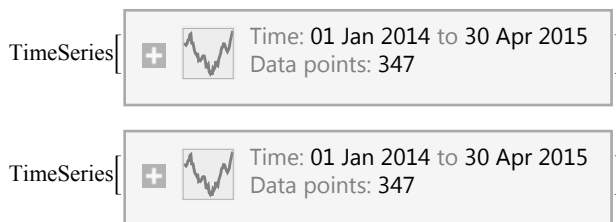
In this section our aim is to understand why this happens and what can be done to mitigate the risk.

### 2.2.1 Example: Chevron Corp. vs. Exxon Mobil Corp.

Let's motivate the discussion with an example of a pairs trade in the NYSE stocks Chevron Corp. (CVX) and Exxon Mobil Corp. (XOM).

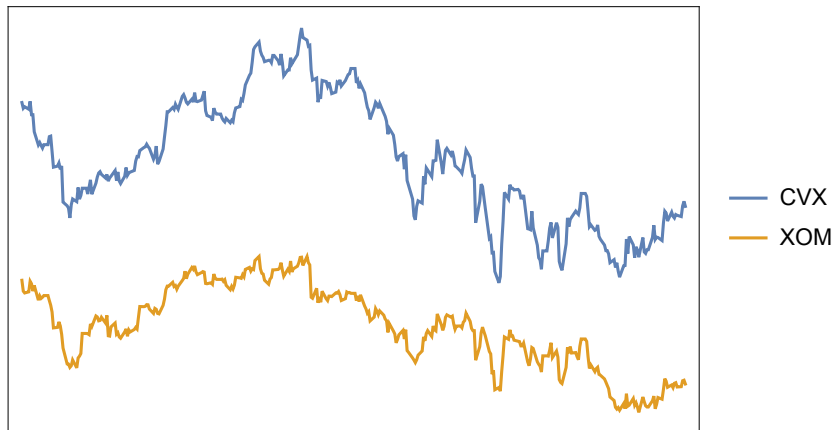
- We first load daily close prices for the two stocks over the formation period from Jan 2014 to Apr 2015:

```
CVX = Entity["Financial", "NYSE:CVX"] [EntityProperty["Financial", "Close",  
    "Date" → {DateObject[{2014, 1, 1}], DateObject[{2015, 4, 30}]}]]  
XOM = Entity["Financial", "NYSE:XOM"] [EntityProperty["Financial", "Close",  
    "Date" → {DateObject[{2014, 1, 1}], DateObject[{2015, 4, 30}]}]]
```



- We plot the two series, from which it is evident that their movements are closely aligned:

```
DateListPlot[{CVX, XOM}, PlotLegends -> {"CVX", "XOM"}]
```



- The CVX-XOM pair appears to be a promising candidate for pairs trading, based on the high correlation between the price series:

```
Correlation[CVX, XOM]
```

```
0.936033
```

- We first extract the prices from the two time series:
 

```
cvx = QuantityMagnitude[CVX["Values"]];
xom = QuantityMagnitude[XOM["Values"]];
```
- Then we proceed with a standard Dickey-Fuller unit root test of each price series. The high probability level of the test statistic indicates that, as expected, we are unable to reject the null hypothesis of a unit root in the price process for each stock. Hence the prices series are integrated, order 1, as is typically the case for asset price processes:

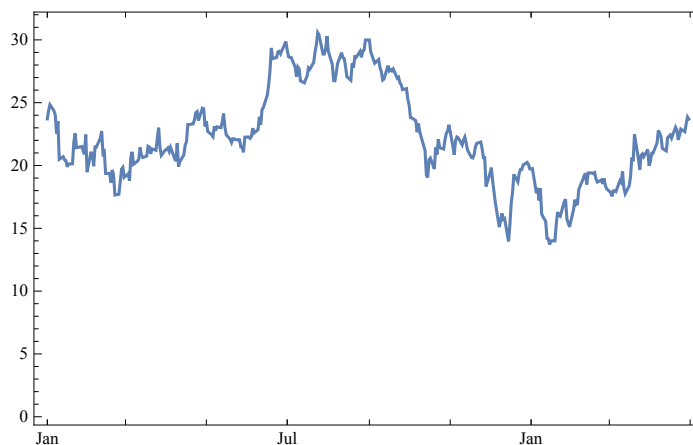
```
UnitRootTest /@ {cvx, xom}
```

```
{0.651376, 0.646736}
```

### 2.2.1.2 Modeling the Cointegration Relationship

- A time series plot of the spread indicates that it is relatively stable over the formation period:

```
DateListPlot[CVX - XOM]
```



- We fit a linear regression model that relates the price of CVX to XOM, with an estimated beta of 0.606:

```
lm = LinearModelFit[Transpose[{cvx, xom}], x, x]
```

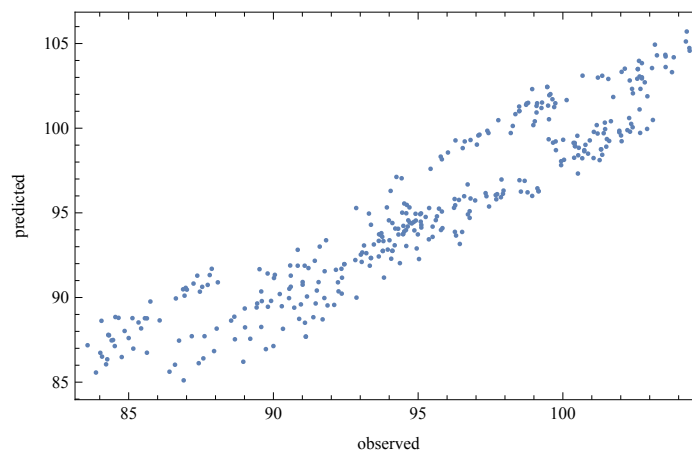
```
FittedModel[0.606104 x + 23.9765]
```

- The model prices provide a good fit to the observed prices for CVX:

```
lm["RSquared"]
```

```
0.876158
```

```
ListPlot[Transpose[lm[{"Response", "PredictedResponse"}]],  
FrameLabel -> {"observed", "predicted"}, Frame -> True, Axes -> False]
```



### 2.2.1.3 Cointegration Test

- We now demonstrate that a cointegration relationship exists between CVX and XOM. Firstly, the earlier Dickey-Fuller tests failed to reject the null hypothesis that the two price series contain a unit root and are therefore integrated with order 1.
- Applying the same test to the model residuals, we reject the null hypothesis of a unit root, confirming that the residual process is not integrated:

```
UnitRootTest[lm["FitResiduals"]]
```

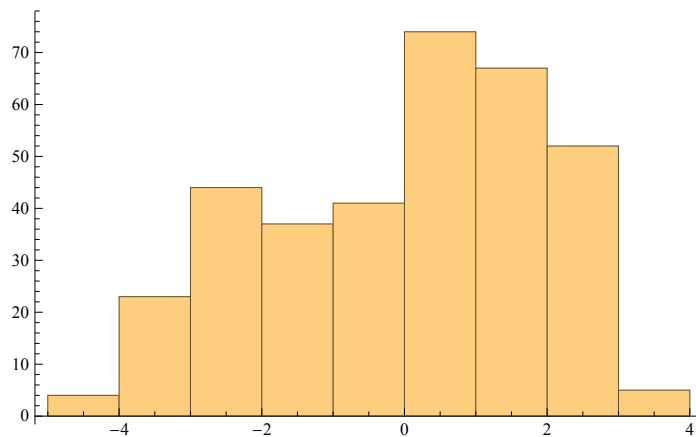
```
0.00833189
```

- By demonstrating that a linear combination of the nonstationary price series for CVX and XOM is itself not integrated, we have shown that the series are cointegrated order 1, with cointegrating vector  $\{-1, 0.606\}$ .

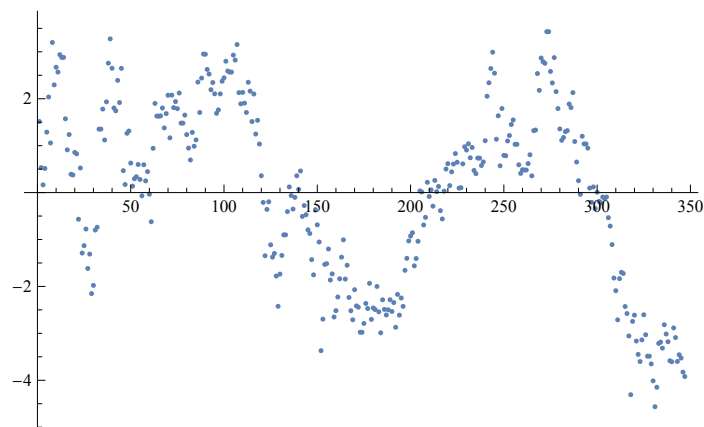
### 2.2.1.4 Modeling the Spread Residuals

- We can take the analysis further by fitting an ARMA model to the residuals of the linear model:



Histogram[`lm["FitResiduals"]`]



ListPlot[`lm["FitResiduals"]`]



`spread = TimeSeriesModelFit[lm["FitResiduals"]]`

TimeSeriesModel[   Family: ARMA  
Order: {1, 1} ]

- The parameters of the spread ARMA model are given by:

**Normal [spread]**

`ARMAProcess[ $1.766 \times 10^{-16}$ , {0.975909}, {-0.126808}, 0.324408]`

- Mathematica selects the best-fitting ARMA model based on the Akaike Information Criterion, a standard measure of goodness-of-fit. The table below lists of all the models considered, together with their AIC:

`spread["CandidateSelectionTable"]`

	Candidate	AIC
<b>1</b>	<b>ARMAProcess(1, 1)</b>	<b>-382.636</b>
2	ARMAProcess(2, 1)	-378.435
3	ARMAProcess(1, 2)	-376.388
4	ARMAProcess(2, 2)	-376.37
5	ARProcess(1)	-335.989
6	MAProcess(1)	236.704
7	MAProcess(0)	453.578

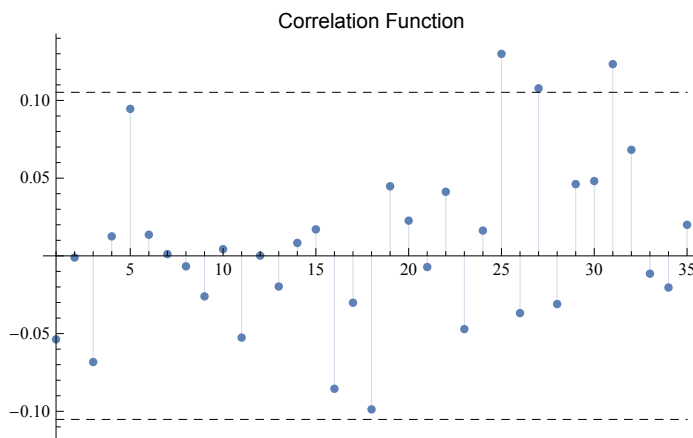
- There are a large number of properties of the selected ARMA model that the analyst can explore and we will look at one or two of them:

`spread["Properties"]`

```
{ACFPlot, ACFValues, AIC, AICc, BIC, BestFit, BestFitParameters,
CandidateModels, CandidateModelSelectionValues, CandidateSelectionTable,
CandidateSelectionTableEntries, CovarianceMatrix, ErrorVariance,
FitResiduals, ForecastStandardErrors, InformationMatrix,
LjungBoxPlot, LjungBoxValues, ModelFamily, PACFPlot,
PACFValues, ParameterConfidenceIntervals, ParameterTable,
ParameterTableEntries, PredictionLimits, Properties, SBC,
SelectionCriterion, StandardizedResiduals, TemporalData}
```

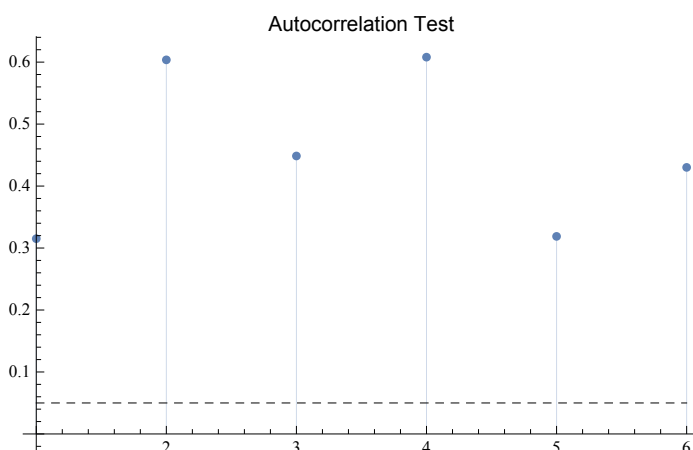
- The ACF plot appears to show no lack of fit in the ARMA model residuals, with insignificant autocorrelations up to 24 lags:

`spread["ACFPlot"]`



- The LjungBox portmanteau test confirms that there is no evidence of lack of fit in the residual autocorrelations:

`spread["LjungBoxPlot"]`



### 2.2.1.4 Stationarity of the Spread Process

Recall that an ARMA process may or may not be stationary, depending on its autoregressive parameters.

- The conditions for stationarity are given by:

```
WeakStationarity[ARMAProcess[{\alpha_1, \alpha_2}, {\beta_1, \beta_2}, \nu]]
```

$$1 - \alpha_2^2 > 0 \wedge (1 - \alpha_2^2)^2 + (-\alpha_2 \alpha_1 - \alpha_1)(\alpha_2 \alpha_1 + \alpha_1) > 0$$

- In this case, the spread ARMA model satisfies the conditions for stationarity:

```
WeakStationarity[Normal[spread]]
```

```
True
```

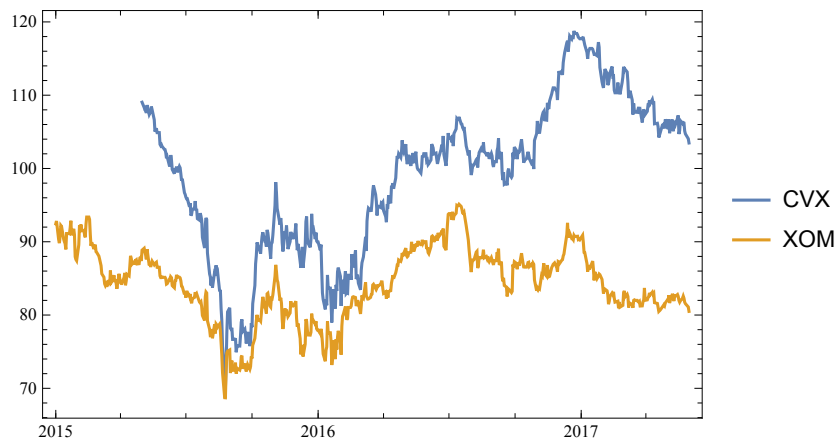
### 2.2.1.5 CVX-XOM Pairs Trade: The Story So Far

Our analysis so far appears promising: we have demonstrated that the CVX-XOM price series are not only highly correlated, but are also cointegrated. Furthermore, we have succeeded in fitting a stationary ARMA model to the residuals of the linear spread model.

This additional step is helpful because it provides further confirmation of the stability of the spread. But it is also useful to set the entry/exit levels of the trade. Firstly, the variance of the ARMA model can be used to determine the levels to go long or short the spread. Alternatively, we can use the ARMA model to make explicit forecasts, and apply these as entry/exit signals instead.

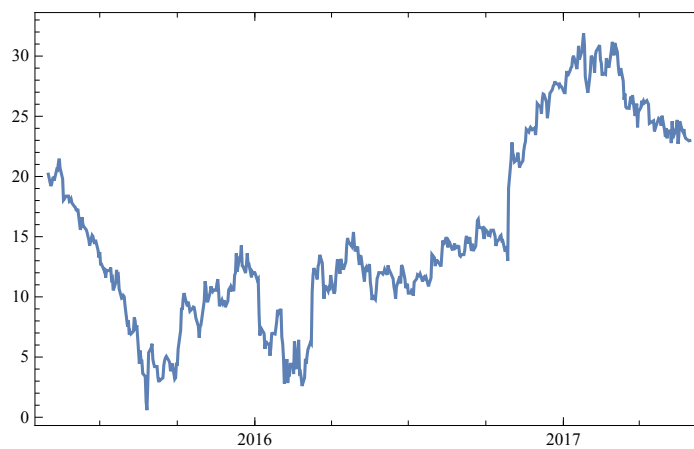
However, before going any further down this road we should take a look at what actually happened to CVX and XOM after the formation period, in the ensuing two years from May 2015 to May 2017:

```
CVXos =
  Entity["Financial", "NYSE:CVX"] [EntityProperty["Financial", "Close",
    "Date" -> {DateObject[{2015, 5, 1}], DateObject[{2017, 5, 31}]}]];
XOMos = Entity["Financial", "NYSE:XOM"] [
  EntityProperty["Financial", "Close",
    "Date" -> {DateObject[{2015, 1, 1}], DateObject[{2017, 5, 31}]}]];
DateListPlot[{CVXos, XOMos}, PlotLegends -> {"CVX", "XOM"}]
```



- Far from remaining stable at around \$20, the CVX-XOM spread first converges almost to zero towards the end of 2015, then proceeds to widen back to over \$30 over the ensuing 18 months.

DateListPlot [CVXos - XOMos]




- A detailed analysis is not required to conclude that the out-of-sample performance of our pairs trade would have been very poor. Regardless of how we had set the entry/exit levels, the trade would have been stopped out (probably several times) as the spread first converged to zero and, then widened back out over the period from 2016-2017.  
So what went wrong?

### 2.2.1.6 The Low Power of Unit Root Tests

The key lesson from this example is that Dickey-Fuller and other unit root tests, on which cointegration analysis depends, have extremely low power.

- To see this, let's consider a data sample generated from a stationary ARIMA process with a large autoregressive coefficient of 0.98 at lag 1:

```
data = RandomFunction[ARIMAProcess[{0.98}, 0, {0.2}, 1], {250}, 1000]
```

TemporalData [  Time: 0 to 250  
Data points: 251 000 Paths: 1000 ]

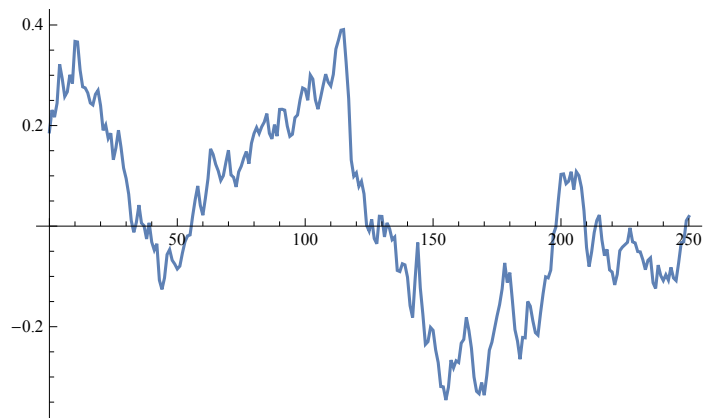
- We know that the underlying process is stationary:

```
WeakStationarity[ARIMAProcess[{0.98}, 0, {0.2}, 1]]
```

True

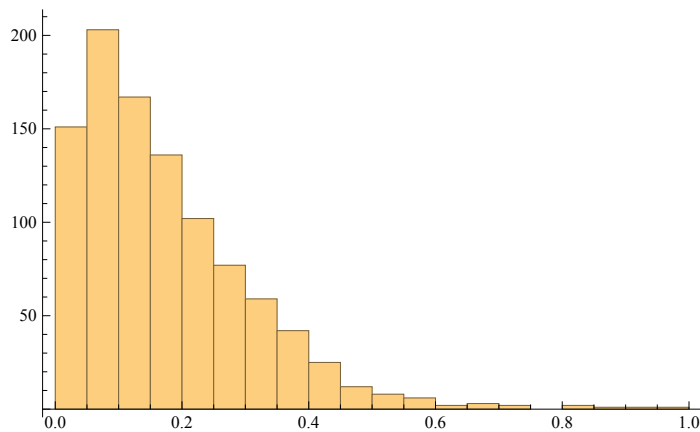
- If we track the mean of the 1,000 samples generated from the process we can see that, while it wanders quite far from its starting point, it eventually reverses course and reverts:

```
ListLinePlot[TimeSeriesThread[Mean, data]]
```



- However, when we conduct unit root tests on the samples we find that in only 15% of cases do the tests reject the null hypothesis of a unit root. Put another way, in approximately 85% of cases we would conclude incorrectly that the process was nonstationary!

```
Histogram[URProbs = UnitRootTest /@ data["ValueList"]]
```



```
Count[URProbs, x_ /; x < 0.05] / Length[URProbs] // N
```


```
0.151
```

- Now let's consider the scenario of a nonstationary process, with unit root:

```
WeakStationarity[ARIMAProcess[{-0.9}, 1, {0.2}, 1]]
```

```
False
```

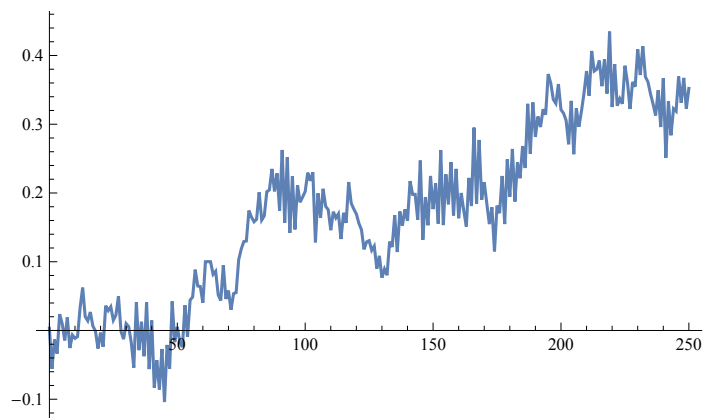
```
data = RandomFunction[ARIMAProcess[{-0.9}, 1, {0.2}, 1], {250}, 1000]
```

```
TemporalData [ +  Time: 0 to 250  
Data points: 251 000 Paths: 1000 ]
```

- This time we see that the mean of the data samples trends away from its initial value, without reverting:

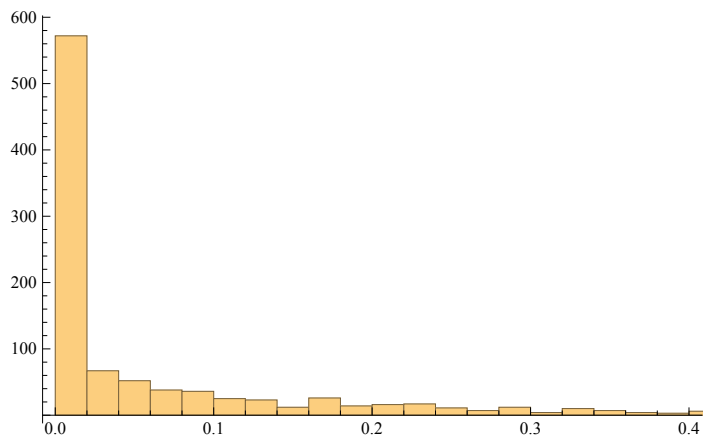


ListLinePlot[TimeSeriesThread[Mean, data]]



- However, when we conduct unit root tests on this data sample we find that in 2/3 of cases the Dickey-Fuller test rejects the null hypothesis of a unit root, leading to the conclusion that the process is stationary!

Histogram[URProbs = UnitRootTest /@ data["ValueList"]]



Count[URProbs, x\_ /; x < 0.05] / Length[URProbs] // N

0.667

### 2.2.1.7 Conclusion

The standard tests used to identify suitable candidates for pairs trades are intrinsically unreliable, and likely to result in a mis-specification of the underlying stochastic processes. This will result in pairs trades breaking down frequently, since the fundamental characteristics of the trade are so poorly defined. Consequently, effective risk controls are vitally important to successful statistical arbitrage and these include, at the very least, stop loss limit on spread trades that fail to converge within a pre-specified time period.