In[1]:=

Kalman Folding [WORKING DRAFT]

Brian Beckman, Technology Fellow, GSI Technology, INC

Nov 2023

Abstract

Fold is a general function, available in most programming languages. *Fold* takes an accumulator function or *foldable* and a sequence of observations. *Fold* produces an aggregate output computed by the foldable accumulator function, starting with an initial value, a *prior* for the output.

Fold decouples the accumulation activity from the source of data. The same foldable accumulator function works over data in memory, over streams delivered on-the-fly, and over external observations delivered asynchronously. This is the great strength of *Fold*. It permits testing of accumulator functions in friendly environments, where results can be hardened and checked against ground truth, then deployment of these exact same functions, now trusted, in harsh environments.

We demonstrate this peculiar effectiveness of *Fold* over multiple instances of the Kalman filter, a foundational pieces of critical technology. It has not been appreciated before that Kalman filters have foldable form, but it is a great advantage that strengthens Kalman techniques even beyond their established utility and efficiency. We propose foldable Kalman filters as a new standard for linear Gaussian models: they run in constant memory and can be rigorously tested offline.

Here, we use the Wolfram language because it excels at concise expression of mathematical codes. The same ideas can be rendered in any contemporary programming language with minimal support for closures, such as Python, C++, and Java. The approach of the paper is pedagogical with many examples fully worked out.

Preliminaries

Definitions

Abbreviations

obl	observable
obr	observer
obn	observation

Experiment

Part 1 Preludes: Running Statistics

Running Count

Consider a sequence of samples of normally distributed noise with zero mean and unit variance. We calculate descriptive statistics of this sequence using *fold* expressions that decouple iteration and accumulation from the means and manner of accessing the observations.

Make a concrete example.

In[34]:=
```
ClearAll[zs];
zs = RandomVariate[NormalDistribution[], 10]
```

Out[35]=
```
{-2.17273, -0.0311523, -0.943119, -0.515608,
 -1.08543, 0.237478, 0.173215, 2.29913, 0.170264, 1.17193}
```

Running count, as a fold, is trivial.

In[36]:=
```
ClearAll[cume];
cume[n_, z_] := n + 1;
Fold[cume, 0, zs]
```

Out[38]=
```
10
```

Running Mean

Definition of *Mean*

As we accumulate data, one observation, z, at a time, write the new estimate of the mean as the ratio of (numerator) the current sum of all data plus the new observation z and (denominator) the new count, $n+1$. Write x for the current estimate of the mean. Write $\bar{z}$ for the current estimate of the mean.

Mean as a *Fold*

We use Mathematica's built-in *Fold* to update a list in curly braces. This list is a ***package*** of cumulative data. The first element of the package is x, the current estimate of the mean. The second

element is *n*, the current count of data. The third and last element is *sum*, the current sum of all input data. ***Fold*** accumulates the observations one-at-a-time, *z*, from the list of all data, *zs*. The function *cume* is the ***foldable accumulator*** function, or just *the foldable*. It depends only on the current package, {*x*, *n*, sum}, and on one *datum z*, from the list *zs*.

Fold takes three arguments: the *foldable cume*, an initial value for the output package, {0, 0, 0}, and the list, *zs*, of all data. *Fold* is flexible; we show later a version that takes data from a real-time stream rather than from a list of data in memory, but re-uses the same *cume*. Such re-use is the main message of this entire paper.

Why *Fold* is Important

The importance of this flexibility is just that: *Fold* can take data from anywhere. Here, *Fold* is in a friendly environment where we can test results in local computer memory against ground truth, say Mathematica's built-in functions like *Mean*. *Fold* can also take data from somewhere else, say from a hostile environment wherein we do not have ground-truth results to compare against. We can test *foldable* functions like *cume* in the friendly environment, build up trust in the foldable functions, and later deploy them in hostile environments with high confidence that they will not fail.

Folding the Definition of *Mean*

In[39]:=
```
ClearAll[cume];
cume[{x_, n_, sum_}, z_] :=
  {(sum + z)/(n + 1), n + 1, sum + z};
Fold[cume, {0, 0, 0}, zs]
```

Out[41]=
```
{-0.0696021, 10, -0.696021}
```

Check against *Mathematica* built-in *Mean* function:

In[42]:=
```
Mean[zs]
```

Out[42]=
```
-0.0696021
```

A Preferable Form

We don't need to track the running sum in the output package. Express the new estimate of mean-so-far as the sum of the old estimate of mean-so-far and a correction. The correction depends only on old values of *sum-of-data* and *count-of-data*, and on a new observational *datum*, *z*.

The right-hand side of the ***recurrence formula*** $\overline{z} \leftarrow \overline{z} + K \times (z - \overline{z})$ depends only on old values and the new datum *z*. The recurrence is not an equation because it has an arrow, ←, but it means the following equation: the new, $(n+1)^{\text{th}}$ estimate of the mean equals the current, n^{th} estimate plus a ***correction***. The correction is a ***gain***, *K*, times the ***residual***, $(z_{n+1} - \overline{z}_n)$: the difference between the current estimate, $\overline{z}_n$, and the new datum, z_{n+1}. In Equation 1 below, the mathematical notation for the recurrence is shown along with the "programming-language" formula, in which the current estimate is written ***x***.

$$\overline{z}_{n+1} = \overline{z}_n + K \times (z_{n+1} - \overline{z}_n) = x + K \times (z_{n+1} - x). \tag{1}$$

Next is a new foldable *cume* that expresses the new estimate as the old estimate plus the gain times the residual. We derive the gain factor immediately after checking this fold:

In[43]:=
```
ClearAll[cume];
cume[{x_, n_}, z_] :=
  With[{K = 1/(n + 1)},
   {x + K (z - x), n + 1}];
Fold[cume, {0, 0}, zs]
```

Out[45]=
```
{-0.0696021, 10}
```

Algebraically, we reckon the gain, K, as follows. The *sum* of all data so far is, by the original definition, the current estimate of the mean, x, times the count of data so far, n. We trust the earlier form of the new estimate of the mean, (sum + z)/($n+1$), so write it as $(x\,n+z)/(n+1)$. Set it equal to the new, desired form, x plus the correction $K(z-x)$, and solve for K:

In[46]:=
```
Solve[(x n + z)/(n + 1) == x + K (z - x), K]
```

Out[46]=
$$\left\{\left\{K \to \frac{1}{1+n}\right\}\right\}$$

We prefer this form because (1) it separates old quantities on the right from new quantities on the left, except for the new datum, z on the right, and (2) it is easy to memorize. It is an ***affine*** update: the old estimate plus a gain factor times the residual of the new observation from the old mean-so-far.

The Kalman Update

The form above will become a mantra for all kinds of estimates:

- ***The new estimate of any quantity is the sum of the old estimate and a gain times the residual of the old estimate and a new datum.***

Equation 1 is formally identical to the update phase of any Kalman-type filter. This paper builds up to several Kalman filters through a sequence of more elementary calculations that have similar forms.

Running Variance

Definitions

Variance is a measure of volatility or dispersion. It is the sum of squared residuals of data with respect to the running mean, divided by the count less one. Recall that the residual is the difference between an observation z and the old mean x. The "count less one" is ***Bessel's correction***, for sample statistics, not explained here.

Mathematica has a built-in ***Variance*** function that furnishes ground-truth for our calculations. Here is Mathematica's variance over our friendly data set, *zs*.

In[47]:=
```
Variance[zs]
```
Out[47]=
```
1.53129
```

Variance is also standard deviation squared, and Mathematica has a built-in, ground-truth function for standard deviation:

In[48]:=
```
StandardDeviation[zs]^2 === Variance[zs]
```
Out[48]=
```
True
```

Variance in Kalman Form

Sum of Squared Residuals

Here is the sum of squared residuals,

$$\Sigma_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} (z_i - \overline{z}_n)^2 \tag{2}$$

We require it to be the variance times the length-less-one. Here is a fold that computes the sum of squared residuals, *ssr*. Notice that we must precompute and pass along the final mean. This is undesirable, and we fix it soon.

In[49]:=
```
ClearAll[cume];
cume[{ssr_, finalMean_, n_}, z_] :=
  {ssr + (z - finalMean)^2, finalMean, n + 1};
Fold[cume, {0, Mean[zs], 0}, zs]
```
Out[51]=
```
{13.7816, -0.0696021, 10}
```

However, despite the undesirable form, we got it right, according to Mathematica:

In[52]:=
```
Variance[zs] * (Length[zs] - 1)
```
Out[52]=
```
13.7816
```

Covariance

The ***covariance***, for scalar data, is the same as the Variance. For vector data, the covariance is a matrix. We shall get there. For now, we just show that Mathematica's built-in, *Covariance*, yields what we expect:

In[53]:=
```
Covariance[zs]
```
Out[53]=
```
1.53129
```

In[54]:=
```
On[Assert]; Assert[Covariance[zs] === Variance[zs]]
```

Why Kalman Form is Important

We require a running covariance that does not refer to future observations (the final mean implicitly refers to all observations) and does not store the entire past, only a summary. In computer terms, storing only a summary of constant size, say in a slot in one of our data packages, is critically important. In this friendly example, we have ten data, and we can easily store them in memory. In a real-world, hostile environment, we will have an *infinite stream* of observations instead of our friendly *zs*. We might store a few million of them in memory and do fancy tricks with sliding windows over the data, but why do that when we can mathematically rewrite covariance in a foldable *cume* function so that we keep only a running estimate in constant memory, continuously updated one-observation-at-a-time? Then we can handle an infinite stream as easily as we handle a finite list in memory.

Kalman won the Draper Prize for this insight. The Draper Prize is the engineer's equivalent of the Nobel Prize. Kalman's Form enabled Aerospace Engineering and the entire Space Age to work on small computers in the 1960's. It survives today as a critical component in all high tech, such as smart-phone apps that continuously update estimates of position and speed of your phone and the position and speed of your Uber driver coming to meet you. The Kalman form for covariance enables position uncertainties depiced as disks that you can see in your apps around the car icons. As the Kalman Filter accumulates position data, the circles move and usually shrink. Shrinking shows continuous improvement in the estimate of position — continuous reduction in covariance. Similar Kalman Filters track the orientation of your phone from gyro data, as well as other non-geophysical quantities such as temperature and battery life, given only asynchronous observations, one-at-a-time.

Let's develop the Kalman form for covariance — the current estimate of covariance plus a gain times a residual — in steps. The steps will incorporate Welford's formula, not well-enough-known, but essential for computer numerics.

Brute-Force Variance

In the first step, accumulate variance directly from the definition, using Mathematica's built-in functions as crutches. This is useful for numerical checks. However, it assumes we know the length and the mean of the entire sequence `zs` of observations ahead-of-time, so it does not meet our requirement of being incremental. It needs computer memory for the entire sequence `zs`, which is a variable of unknown length, so does not meet our requirement of constant memory. We'll mitigate these defects step-by-step.

Start with a tiny example that we can check by hand.

In[55]:=
```
ClearAll[zs]; zs = {55, 89, 144};
```

In[56]:=
```
Variance[zs] // N
```

Out[56]=
```
2017.
```

The following *cume* mechanizes the mathematical definition of variance.

In[57]:=
```
ClearAll[cume];
cume[var_, z_] := var + (z - Mean[zs])^2 / (Length[zs] - 1);
FoldList[cume, 0., zs]
```

Out[59]=
```
{0., 840.5, 865., 2017.}
```

FoldList is the same as *Fold* except for displaying all intermediate results. Notice that only the final variance is valid. The intermediate variances are not valid because they do not refer to the "mean-so-far," but to the final mean.

School Variance

The following is much better, exploiting the "school formula." Please prove the school equation, Equation 3, to yourself, by expanding the square on the left.

Letting $\bar{z}_n = x$, our incremental running mean, the following yields the sum of squared residuals, `ssq`:

$$\sum_{i=1}^{n} (z_i - \bar{z}_n)^2 = \sum_{i=1}^{n} z_i^2 - n\,\bar{z}_n^2 \tag{3}$$

We already know that variance is `ssq` divided by $n-1$ when $n > 1$. When $n = 1$, the variance should be zero because there is no dispersion when there is only one datum. One might take *Infinity* for this special case, but zero is sensible and convenient. Write, assuming the same Kalman gain, $K = 1/(n+1)$, that we had before. All the intermediate variances are now valid.

In[60]:=
```
ClearAll[cume];
cume[{var_, ssq_, x_, n_}, z_] :=
  With[{n2 = n + 1},
   With[{K = 1/n2},
    With[{x2 = x + K (z - x), ssq2 = ssq + z^2},
     {(ssq2 - n2 x2^2)/Max[1, n], ssq2, x2, n2}]]];
FoldList[cume, {0, 0, 0, 0}, zs] // MatrixForm
```

Out[62]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3025 & 55 & 1 \\ 578 & 10\,946 & 72 & 2 \\ 2017 & 31\,682 & 96 & 3 \end{pmatrix}$$

The variance computed this way is incremental and runs in constant memoryIt tracks three auxiliary quantities: the sum of squares `ssq`, the mean-so-far `x`, and the count `n`. Notice, for the future, that `ssq` grows quickly, inviting numerical issues. We next address this defect.

Recurrent Variance

The school variance isn't a recurrence in Kalman Form. A Kalman recurrence would express the new variance as the old variance plus a correction, where the correction depends only on old values and one new datum, z. Let's derive that form.

Start with a recurrence for the sum of squared residuals, `ssr`, $\Sigma_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} (z_i - \bar{z}_n)^2$, which is not hard to find:

$$\Sigma \leftarrow \Sigma + \frac{n}{n+1} (z - \bar{z}_n)^2 \tag{4}$$

Everything on the right, except for the new observation z, is an old value.

Proof

Here's the old **ssq**:

$$S_n = \sum_{i=1}^{n} z_i^2 - n\,\bar{z}^2 \tag{5}$$

Here's the new **ssq**:

$$S_{n+1} = \sum_{i=1}^{n+1} z_i^2 - (n+1)\,\bar{z}_{n+1}^2 \tag{6}$$

Let's have a Mathematica symbolic function for the sum of squares up through the n^{th} datum: $\text{sz}_n^2 =$ `sz2[n]` $= \sum_{i=1}^{n} z_i^2$. Define a Mathematica function, `s[n]`, for the mathematical expression S_n above.

In[63]:=
```
ClearAll[s, zb, zbar];
s[n_] := sz2[n] - n zbar[n]^2
```

Define some Mathematica rules for $S_{n+1} - S_n = z^2$, the current observation, z, squared, an evident truth; and for $\bar{z}_{n+1} = \frac{n}{n+1}\bar{z}_n + \frac{z}{n+1}$, another evident truth:

In[65]:=
```
rulez$ = {sz2[1 + n] - sz2[n] → z^2, zbar[1 + n] → n/(n + 1) zbar[n] + z/(n + 1)};
```

Treat the sum of squares, `s[n]`, as an estimate like any other Kalman estimate, though it happens to be exact. Ask for the difference between the new estimate, `s[n+1]`, and the old estimate `s[n]`, and apply the rules:

In[66]:=
```
s[n + 1] - s[n] //. rulez$ // FullSimplify
```

Out[66]=

$$\frac{n\,(z - \text{zbar}[n])^2}{1 + n}$$

This is the correction term in the recurrence 4, with a gain of $n/(1+n)$.

Equivalence to Welford's

Welford's insufficiently famous recurrence is

$$\Sigma \leftarrow \Sigma + (z - \bar{z}_n)\,(z - \bar{z}_{n+1}) \tag{7}$$

where $\bar{z}_n$ is the last estimate of the mean and $\bar{z}_{n+1}$ is the new or current estimate. This is remarkable because it subtracts before squaring, as opposed to the last *cume* above, which squares before subtracting. Subtracting large numbers like squares invites *catastrophic cancelation*. Subtracting before squaring gives us the best possible defense against catastrophic cancelation in this case.

The correction term on the right of recurrence 7 is exactly equal to the correction term on the right of recurrence 4:

In[67]:=
```
(z - zbar[n]) (z - zbar[n + 1] //. rulez$ // FullSimplify)
```

Out[67]=

$$\frac{n\,(z - \text{zbar}[n])^2}{1 + n}$$

We invite you to do the algebraic manipulations by hand to accomplish this proof. In many years of giving this problem in job-interview questions, I have had a few candidates who succeeded in doing it by hand on the white-board.

Welford's is easier to memorize than recurrence 4, but its right-hand side cheats: it depends on both old and new values, whereas we want right-hand sides that depend only on old values (except for z), not least because they require fewer intermediate variables.

The following, using Welford's, has three levels of **With** because **ssr2** depends on the new mean **x2**, but we see it matches the school variance.

In[68]:=
```
ClearAll[cume];
cume[{var_, x_, n_}, z_] :=
  With[{K = 1/(n + 1)}, (* Kalman gain *)
   With[{x2 = x + K (z - x)}, (* update of the mean *)
    With[{ssr2 = (n - 1) var + (z - x) (z - x2)}, (* Welford's *)
     {ssr2/Max[1, n], x2, n + 1}]]];(* new variance, mean, and count *)
FoldList[cume, {0, 0, 0}, zs] // MatrixForm
```

Out[70]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 55 & 1 \\ 578 & 72 & 2 \\ 2017 & 96 & 3 \end{pmatrix}$$

Folding It

Recurrence 4 lets us get rid of one level of `With`, one level of dependency, because the new estimate of the sum of squared residuals, `ssr2`, does not depend on the new estimate of the mean, `x2`. We've satisfied all requirements: estimates in constant memory and Kalman form with only old estimates on the right-hand side! Please note that, while the gain for the mean is $K = 1/(n+1)$, the gain for the (co)variance is $nK = n/(n+1)$.

In[71]:=
```
ClearAll[cume];
cume[{var_, x_, n_}, z_] :=
  With[{K = 1/(n + 1)}, (* Kalman gain *)
   With[{x2 = x + K (z - x), (* update of the mean *)
     ssr2 = (n - 1) var + K n (z - x)^2}, (* no dependence on x2 *)
    {ssr2/Max[1, n], x2, n + 1}]];
FoldList[cume, {0, 0, 0}, zs] // MatrixForm
```

Out[73]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 55 & 1 \\ 578 & 72 & 2 \\ 2017 & 96 & 3 \end{pmatrix}$$

To LaTeX:

For presentation purposes, let's rewrite the output in LaTeX:

In[74]:=
```
ClearAll[cume];
cume[{var_, x_, n_}, z_] :=
  With[{K = 1/(n + 1)},
   With[{x2 = x + K (z - x),
     ssr2 = (n - 1) var + K n (z - x)^2},
    {ssr2/Max[1, n], x2, n + 1}]];
FoldList[cume, {0, 0, 0}, zs] // TeXForm
```

Out[76]//TeXForm=
```
\left(
\begin{array}{ccc}
 0 & 0 & 0 \\
 0 & 55 & 1 \\
 578 & 72 & 2 \\
 2017 & 96 & 3 \\
\end{array}
\right)
```

Windowed Statistics

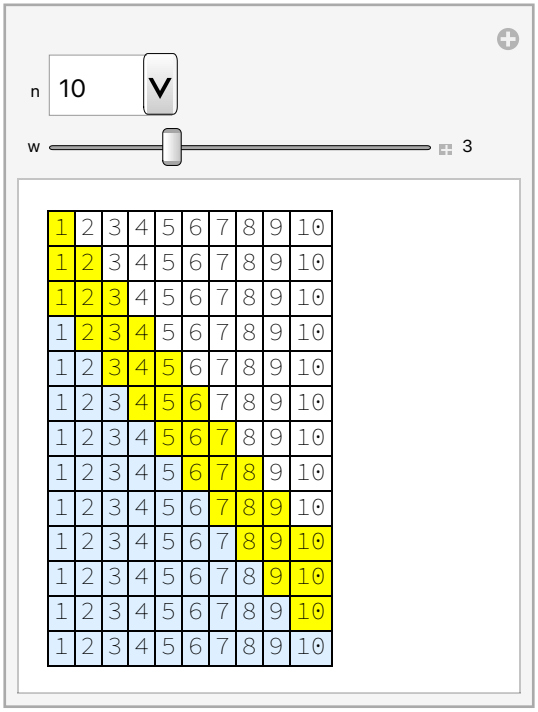
On the right-hand side above, we have only one datum, *z*, at a time. We can still stay in constant memory if we allow more than one datum and more than one estimate at a time, so long as they are constant numbers. We can still do this with Kalman-form recurrences.

Sometimes, we want the mean and variance of a limited amount of the past, say over a window of constant length *w*. In the following, manipulate the slider to change the size of the window, *w*, and

manipulate the drop-down menu to change the total count of data, *n*. We call the sequence of data arrivals, 1 in the first row, 2 in the second row, up through *n* in the n^{th} row, a *run* of data. Observe, in the rows, indices of data elements, ranging from 1 to *n*. In the first row, only one datum has arrived so far; in the second row, two data have arrived, and so on, but we keep empty, to-be-filled cells in white to the right to maintain the square appearance of the display and the structure of the sequential run. Observe also the yellow portion of each row, containing indices of the data in the window. The yellow portion is never more than *w* wide, though it is less than *w* wide near the beginning and near the end of the run. In light blue, see the indices of elements that have departed the window, never to be visited again. This display shows only indices, not data.

In[77]:=
```
Manipulate[
 Grid[Table[With[{l = i - w + 1},
   {Item[j, Background → LightBlue]  j < l
    Item[j, Background → Yellow]     j ≤ i
    j                                True}],
  {i, n + w}, {j, n}], Frame → All],
 {{n, 10}, 5 + Range[100]},
 {{w, 3}, Range[n + 1] - 1, Appearance → "Labeled"}]
```

Out[77]=



Suppose we are anywhere in the run -- on any row -- and we already have $1 \le w \le n$ data, *w* now meaning the row-dependent, varying width of the window, up to its maximum, the former meaning of *w*. The mean and variance of the data within the window are computed via the following formulae, which we do not derive, to save time and space:

$$R_{w,n} = \sum_{i=n-w+1}^{n} z_i \tag{8}$$

$$S_{w,n} = \sum_{i=n-w+1}^{n} z_i^2 - \frac{1}{w}\left(\sum_{i=n-w+1}^{n} z_i\right)^2 \tag{9}$$

Here is a symbolic expansion of the first few row, as a *FoldList*.

In[78]:=
```
Module[{data = Table[zi, {i, 4}], cume},
 cume[{n_, zm1_, x_, v_}, z_] :=
  {n + 1, z, x + 1/(n + 1) (z - x),
   ((n - 1) v + n/(n+1) (z - x)^2)/Max[1, n]};
 Grid[
  FoldList[cume, {0, 0, 0, 0}, data], Frame → All]]
```

Out[78]=

⊙	⊙	⊙	⊙
1	z_1	z_1	⊙
2	z_2	$z_1 + \frac{1}{2}(-z_1 + z_2)$	$\frac{1}{2}(-z_1 + z_2)^2$
3	z_3	$z_1 + \frac{1}{2}(-z_1 + z_2) + \frac{1}{3}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + z_3\right)$	$\frac{1}{2}\left(\frac{1}{2}(-z_1 + z_2)^2 + \frac{2}{3}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + z_3\right)^2\right)$
4	z_4	$z_1 + \frac{1}{2}(-z_1 + z_2) + \frac{1}{3}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + z_3\right) + \frac{1}{4}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + \frac{1}{3}\left(z_1 + \frac{1}{2}(-z_1 + z_2) - z_3\right) + z_4\right)$	$\frac{1}{3}\left(\frac{1}{2}(-z_1 + z_2)^2 + \frac{2}{3}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + z_3\right)^2 + \frac{3}{4}\left(-z_1 + \frac{1}{2}(z_1 - z_2) + \frac{1}{3}\left(z_1 + \frac{1}{2}(-z_1 + z_2) - z_3\right) + z_4\right)^2\right)$

The following is a unit-test of a function, *buttonize*, a GUI for running arbitrary, random data through arbitrary functions. We'll use that to stress-test our windowing statistics later on:

In[79]:=
```
ClearAll[buttonize];
buttonize[fnRandom_, fnResultFromRandom_, lbl_ : "RANDOMIZE!"] :=
  DynamicModule[{val = fnRandom[]},
Manipulate[fnResultFromRandom[val],
    Button[lbl, val = fnRandom[]]]];
```

In[81]:=
```
buttonize[RandomReal, Identity]
```

Out[81]=

Here we check, algebraically, a few expressions from the general formulae above:

In[82]:=
$$\frac{(n+1)\,mn-(n+1-w)\,ml}{w}\;//\;\text{FullSimplify}$$

Out[82]=
$$ml+\frac{(-ml+mn)\,(1+n)}{w}$$

In[83]:=
$$\frac{n\,\upsilon n+(n+1)\,mn^2-(n-w)\,\upsilon l-(n+1-w)\,ml^2-w\,mw^2}{w-1}\;//\;\text{FullSimplify}$$

Out[83]=
$$\frac{mn^2\,(1+n)-mw^2\,w+ml^2\,(-1-n+w)-n\,\upsilon l+w\,\upsilon l+n\,\upsilon n}{-1+w}$$

Data from the C program

We harvested some random data from a C program to present to our foldable windowing statistics. The foldable is presented, as usual, as a *cume* function that runs over the data, one observation at a time, updating a sizeable package of data. The elements of the package are explained in comments. The function is quite large and we do not derive the details, to save time and space. You are invited to check the formulae as an exercise.

In[84]:=
```
Module[{w = 3, u, l, mn, ml, mw, υn, υl, υw,
  data = {0.857454, 0.312454, 0.705325, 0.839363, 1.63781,
    0.699257, -0.340016, -0.213596, -0.0418609, 0.054705}, cume},
 cume[{_, n_, _, _, x_, xw_, _, _, v_, vw_, _, _}, z_] := (
  (* n : how many points seen before this one. *)
  (* z : current data point, at index n + 1. *)
  (* l :
   index of first point in window at least 1 wide and no more than w wide. *)
  (* w : width of window. *)
  (* u : number of points including z in the running window; converges to w. *)
  (* mn : the mean of all points through z_{n+1}. *)
  (* ml : the mean of all points through index l-1 = n+1-w,
  lagging w behind mn. *)
  (* mw : the mean of all points within the window. *)
  (* υn : variance of all points through z_{n+1}. *)
  (* υl : variance of all points through index l-1 = n+1-w,
  lagging w behind υn *)
  (* υw : variance of all points within the window. *)

  l = Max[1, n - w + 2]; u = n + 2 - l;
  mn = x + (z - x)/(n + 1);
  ml = If[n + 1 > w, xw + (data[[n + 1 - w]] - xw)/(n + 1 - w), 0];
  mw = ((n + 1) mn - (n + 1 - w) ml)/u;
  υn = ((n - 1) v + n/(n+1) (z - x)^2)/Max[1, n];
  υl = If[n + 1 > w, ((n - w - 1) υl + (n-w)/(n-w+1) (data[[n + 1 - w]] - xw)^2)/Max[1, n - w], 0];
  υw = (n υn + (n + 1) mn^2 - (n - w) υl - (n + 1 - w) ml^2 - u mw^2)/Max[1, u - 1];
  {l, n + 1, n + 2 - l, z,
   mn, ml, mw, Mean[data[[l ;; n + 1]]], υn, υl, υw, Quiet@Variance[data[[l ;; n + 1]]]}
 );
 Grid[
  Prepend[FoldList[cume, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, data],
   {"l", "n+1", "u", "z", "μ_{n+1}", "μ_l", "μ_w", "μ'_w", "υ_{n+1}", "υ_l", "υ_w", "υ'_w"}],
  Frame → All]]
```

Out[84]=

l	n+1	u	z	μ_{n+1}	μ_l	μ_w	μ_w'	v_{n+1}	v_l	v_w	v_w'
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0.8574… 54	0.8574… 54	0	0.8574… 54	0.8574… 54	0.	0	0.	Varian… ce[{0.85… 745… 4}]
1	2	2	0.3124… 54	0.5849… 54	0	0.5849… 54	0.5849… 54	0.1485… 13	0	0.1485… 13	0.1485… 13
1	3	3	0.7053… 25	0.6250… 78	0	0.6250… 78	0.6250… 78	0.0790… 86	0	0.0790… 86	0.0790… 86
2	4	3	0.8393… 63	0.6786… 49	0.8574… 54	0.6190… 47	0.6190… 47	0.0642… 035	0.	0.0749… 912	0.0749… 912
3	5	3	1.63781	0.8704… 81	0.5849… 54	1.06083	1.06083	0.2321… 51	0.1485… 13	0.2541… 69	0.2541… 69
4	6	3	0.6992… 57	0.8419… 44	0.6250… 78	1.05881	1.05881	0.1906… 07	0.0790… 86	0.2563… 38	0.2563… 38
5	7	3	-0.340… 016	0.6730… 92	0.6786… 49	0.6656… 84	0.6656… 84	0.3584… 15	0.0642… 035	0.9787… 94	0.9787… 94
6	8	3	-0.213… 596	0.5622… 56	0.8704… 81	0.0485… 483	0.0485… 483	0.40549	0.2321… 51	0.3215… 62	0.3215… 62
7	9	3	-0.041… 8609	0.4951… 32	0.8419… 44	-0.198… 491	-0.198… 491	0.3953… 54	0.1906… 07	0.0223… 952	0.0223… 952
8	10	3	0.0547… 05	0.45109	0.6730… 92	-0.066… 9173	-0.066… 9173	0.3708… 24	0.3584… 15	0.0184… 672	0.0184… 672

Here is a presentation of the results from the C program. Visually check that Mathematica matches the results.

Out[ ]=

l	n+1	u	z	μ_{n+1}	μ_l	μ_w	μ_w'	υ_{n+1}	υ_l	υ_w	υ_w'
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0.8574… 54	0.8574… 54	0	0.8574… 54	0.8574… 54	0.	0	0.	Varian… ce[{0.85… 745… 4}]
1	2	2	0.3124… 54	0.5849… 54	0	0.5849… 54	0.5849… 54	0.1485… 13	0	0.1485… 13	0.1485… 13
1	3	3	0.7053… 25	0.6250… 78	0	0.6250… 78	0.6250… 78	0.0790… 86	0	0.0790… 86	0.0790… 86
2	4	3	0.8393… 63	0.6786… 49	0.8574… 54	0.6190… 47	0.6190… 47	0.0642… 035	0.	0.0749… 912	0.0749… 912
3	5	3	1.63781	0.8704… 81	0.5849… 54	1.06083	1.06083	0.2321… 51	0.1485… 13	0.2541… 69	0.2541… 69
4	6	3	0.6992… 57	0.8419… 44	0.6250… 78	1.05881	1.05881	0.1906… 07	0.0790… 86	0.2563… 38	0.2563… 38
5	7	3	-0.340… 016	0.6730… 92	0.6786… 49	0.6656… 84	0.6656… 84	0.3584… 15	0.0642… 035	0.9787… 94	0.9787… 94
6	8	3	-0.213… 596	0.5622… 56	0.8704… 81	0.0485… 483	0.0485… 483	0.40549	0.2321… 51	0.3215… 62	0.3215… 62
7	9	3	-0.041… 8609	0.4951… 32	0.8419… 44	-0.198… 491	-0.198… 491	0.3953… 54	0.1906… 07	0.0223… 952	0.0223… 952
8	10	3	0.0547… 05	0.45109	0.6730… 92	-0.066… 9173	-0.066… 9173	0.3708… 24	0.3584… 15	0.0184… 672	0.0184… 672

Here are some more data from another run of the C program:

In[85]:=
```
Module[{w = 6, u, l, mn, ml, mw, υn, υl, υw,
  data = {0.857454, 0.312454, 0.705325, 0.839363, 1.63781,
    0.699257, -0.340016, -0.213596, -0.0418609, 0.054705, 1.10464,
    -0.387322, -0.00175018, 1.12034, 0.280948, -1.07877}, cume},
 cume[{_, n_, _, _, x_, xw_, _, _, v_, vw_, _, _}, z_] := (
  (* n : how many points seen before this one. *)
  (* z : current data point, at index n + 1. *)
  (* l :
   index of first point in window at least 1 wide and no more than w wide. *)
  (* w : width of window. *)
  (* u : number of points including z in the running window; converges to w. *)
  (* mn : the mean of all points through z_{n+1}. *)
  (* ml : the mean of all points through index l-1 = n+1-w,
```

```
  lagging w behind mn. *)
  (* mw : the mean of all points within the window. *)
  (* υn : variance of all points through z_{n+1}. *)
  (* υl : variance of all points through index l-1 = n+1-w,
  lagging w behind υn *)
  (* υw : variance of all points within the window. *)

  l = Max[1, n - w + 2]; u = n + 2 - l;
  mn = x + (z - x)/(n + 1);
  ml = If[n + 1 > w, xw + (data[[n + 1 - w]] - xw)/(n + 1 - w), 0];
  mw = ((n + 1) mn - (n + 1 - w) ml)/u;
  υn = ((n - 1) v + n/(n+1) (z - x)^2)/Max[1, n];
  υl = If[n + 1 > w, ((n - w - 1) υl + (n-w)/(n-w+1) (data[[n + 1 - w]] - xw)^2)/Max[1, n - w], 0];
  υw = (n υn + (n + 1) mn^2 - (n - w) υl - (n + 1 - w) ml^2 - u mw^2)/Max[1, u - 1];
  {l, n + 1, n + 2 - l, z,
   mn, ml, mw, Mean[data[[l ;; n + 1]]], υn, υl, υw, Quiet@Variance[data[[l ;; n + 1]]]}];

Grid[
 Prepend[FoldList[cume, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, data],
  {"l", "n+1", "u", "z", "μ_{n+1}", "μ_l", "μ_w", "μ_w'", "υ_{n+1}", "υ_l", "υ_w", "υ_w'"}],
 Frame → All]
```

Out[85]=

l	n+1	u	z	μ_{n+1}	μ_l	μ_w	μ_w'	υ_{n+1}	υ_l	υ_w	υ_w'
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0.8574… 54	0.8574… 54	0	0.8574… 54	0.8574… 54	0.	0	0.	Varian… ce[{0.85… 745… 4}]
1	2	2	0.3124… 54	0.5849… 54	0	0.5849… 54	0.5849… 54	0.1485… 13	0	0.1485… 13	0.1485… 13
1	3	3	0.7053… 25	0.6250… 78	0	0.6250… 78	0.6250… 78	0.0790… 86	0	0.0790… 86	0.0790… 86
1	4	4	0.8393… 63	0.6786… 49	0	0.6786… 49	0.6786… 49	0.0642… 035	0	0.0642… 035	0.0642… 035
1	5	5	1.63781	0.8704… 81	0	0.8704… 81	0.8704… 81	0.2321… 51	0	0.2321… 51	0.2321… 51
1	6	6	0.6992… 57	0.8419… 44	0	0.8419… 44	0.8419… 44	0.1906… 07	0	0.1906… 07	0.1906… 07
2	7	6	-0.340… 016	0.6730… 92	0.8574… 54	0.6423… 65	0.6423… 66	0.3584… 15	0.	0.4221… 67	0.4221… 67
3	8	6	-0.213… 596	0.5622… 56	0.5849… 54	0.5546… 91	0.5546… 91	0.40549	0.1485… 13	0.5377… 08	0.5377… 08
4	9	6	-0.041… 8609	0.4951… 32	0.6250… 78	0.43016	0.43016	0.3953… 54	0.0790… 86	0.5857… 35	0.5857… 35
5	10	6	0.0547… 05	0.45109	0.6786… 49	0.2993… 83	0.2993… 83	0.3708… 24	0.0642… 035	0.5599… 16	0.5599… 16
6	11	6	1.10464	0.5105… 03	0.8704… 81	0.2105… 22	0.2105… 22	0.3725… 71	0.2321… 51	0.3218… 51	0.3218… 51
7	12	6	-0.387… 322	0.4356… 84	0.8419… 44	0.0294… 25	0.0294… 25	0.4058… 75	0.1906… 07	0.3062… 06	0.3062… 06
8	13	6	-0.001… 7501… 8	0.4020… 36	0.6730… 92	0.0858… 027	0.0858… 027	0.3867… 71	0.3584… 15	0.2752… 89	0.2752… 89
9	14	6	1.12034	0.4533… 43	0.5622… 56	0.3081… 25	0.3081… 25	0.3938… 74	0.40549	0.4121… 02	0.4121… 02
10	15	6	0.2809… 48	0.44185	0.4951… 32	0.3619… 27	0.3619… 27	0.3677… 22	0.3953… 54	0.3842… 78	0.3842… 78
11	16	6	-1.078… 77	0.3468… 11	0.45109	0.1730… 14	0.1730… 14	0.4877… 25	0.3708… 24	0.7376… 97	0.7376… 97

Windowed Statistics from Random Inputs

Here, finally, are statistics for windowed, random data, available under GUI via *buttonize*. Change

the size of the window (it's 3, here) and the total count of data (it's 20, here). Press the button repeatedly to enjoy the computations. We shall soon see a Kalman-form of the same.

In[86]:=
```
DynamicModule[
 {w = 3, u, l, mn, ml, mw, υn, υl, υw, data = N@RandomInteger[{-100, 100}, 20], cume},
 cume[{_, n_, _, _, x_, xw_, _, _, v_, vw_, _, _}, z_] := (
   (* n : how many points seen before this one. *)
   (* z : current data point, at index n + 1. *)
   (* l :
    index of first point in window at least 1 wide and no more than w wide. *)
   (* w : width of window. *)
   (* u : number of points including z in the running window; converges to w. *)
   (* mn : the mean of all points through z_{n+1}. *)
   (* ml : the mean of all points through index l-1 = n+1-w,
   lagging w behind mn. *)
   (* mw : the mean of all points within the window. *)
   (* υn : variance of all points through z_{n+1}. *)
   (* υl : variance of all points through index l-1 = n+1-w,
   lagging w behind υn *)
   (* υw : variance of all points within the window. *)

   l = Max[1, n - w + 2];
   mn = x + (z - x)/(n + 1);
   ml = If[n + 1 > w, xw + (data[[n + 1 - w]] - xw)/(n + 1 - w), 0];
   mw = ((n + 1) mn - (n + 1 - w) ml)/w;
   υn = ((n - 1) v + n/(n+1) (z - x)^2)/Max[1, n];
   υl = If[n + 1 > w, ((n - w - 1) υl + (n-w)/(n-w+1) (data[[n + 1 - w]] - xw)^2)/Max[1, n - w], 0];
   υw = (n υn + (n + 1) mn^2 - (n - w) υl - (n + 1 - w) ml^2 - w mw^2)/(w - 1);
   {l, n + 1, n + 2 - l, z,
    mn, ml, mw, Mean[data[[l ;; n + 1]]], υn, υl, υw, Quiet@Variance[data[[l ;; n + 1]]]}
 );
 Manipulate[Grid[
   Prepend[FoldList[cume, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, data],
    {"l", "n+1", "u", "z", "μ_{n+1}", "μ_l", "μ_w", "μ'_w", "υ_{n+1}", "υ_l", "υ_w", "υ'_w"}],
   Frame → All],
  Button["RANDOMIZE!", data = N@RandomInteger[{-100, 100}, 20]]]]
```

Out[86]=

RANDOMIZE !

l	n+1	u	z	μ_{n+1}	μ_l	μ_w	μ'_w	υ_{n+1}	υ_l	υ_w	υ'_w
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	-100.	-100.	0	-33.3333	-100.	0.	0	3333.33	Variance[{-100.}]
1	2	2	14.	-43.	0	-28.6667	-43.	6498.	0	3865.33	6498.
1	3	3	-88.	-58.	0	-58.	-58.	3924.	0	3924.	3924.
2	4	3	41.	-33.25	-100.	-11.	-11.	5066.25	0.	4629.	4629.
3	5	3	-14.	-29.4	-43.	-20.3333	-20.3333	3873.8	6498.	4190.33	4190.33
4	6	3	-41.	-31.3333	-58.	-4.66667	-4.66667	3121.47	3924.	1746.33	1746.33
5	7	3	-53.	-34.4286	-33.25	-36.	-36.	2668.29	5066.25	399.	399.
6	8	3	55.	-23.25	-29.4	-13.	-13.	3286.79	3873.8	3504.	3504.
7	9	3	7.	-19.8889	-31.3333	3.	3.	2977.61	3121.47	2928.	2928.
8	10	3	-9.	-18.8	-34.4286	17.6667	17.6667	2658.62	2668.29	1109.33	1109.33
9	11	3	83.	-9.54545	-23.25	27.	27.	3334.87	3286.79	2416.	2416.
10	12	3	-84.	-15.75	-19.8889	-3.33333	-3.33333	3493.66	2977.61	6996.33	6996.33
11	13	3	-72.	-20.0769	-18.8	-24.3333	-24.3333	3445.91	2658.62	8676.33	8676.33
12	14	3	58.	-14.5	-9.54545	-32.6667	-32.6667	3616.27	3334.87	6201.33	6201.33
13	15	3	6.	-13.1333	-15.75	-2.66667	-2.66667	3385.98	3493.66	4281.33	4281.33
14	16	3	-55.	-15.75	-20.0769	3.	3.	3269.8	3445.91	3199.	3199.
15	17	3	-77.	-19.3529	-14.5	-42.	-42.	3286.12	3616.27	1849.	1849.
16	18	3	94.	-13.0556	-13.1333	-12.6667	-12.6667	3806.64	3385.98	8654.33	8654.33

17	19	3	-79.	-16.5263	-15.75	-20.6667	-20.6667	3824.04	3269.8	9862.33	9862.33
18	20	3	-21.	-16.75	-19.3529	-2.	-2.	3623.78	3286.12	7753.	7753.

Variance of the Mean

As observations accumulate, the running estimate $\overline{z}_n = \overline{z_n}$ of the abstract, true, constant population mean, א, improves, using Hebrew alef, א, for "abstract." Can we track the variance of $\overline{z}_n$ with respect to the abstract, true, constant, population mean, א? We can.

In almost all applications, outside of testing, we do not know the ground truth. In testing, we can check the percentage of residuals of data from true א that lie within one standard deviation of true א. If our computations behave properly, about 68% of residuals should be within one theoretical sigma of ground truth, א.

Variance Of the Theoretical Mean

The following derivation holds only when the ground truth is constant. We relieve this limitation below with the full, time-dependent Kalman filter.

Write this variance as an expectation value over the probability distribution of the assumed, modeled observation noise. There is no need for Bessels' correction because the distribution represents an infinite population and Bessel's correction pertains only to finite samples.

Consider the $(n+1)$-st estimate of the mean, $\overline{z_{n+1}}$, where z is the current observation:

$$\overline{z_{n+1}} = \overline{z_n} + K \times (z - \overline{z_n})$$

Its residual with respect to the unknown truth א is (subtract both sides from א):

$$(א - \overline{z_{n+1}}) = (א - \overline{z_n}) - K \times (z - \overline{z_n})$$

The observation z is the truth א plus a sample ζ_n of the random noise distribution (ζ is a random variable — not a number — with some distribution; ζ_n is the n-th sample of that distribution, a number with the same units of measure as z and א).

$$(א - \overline{z_{n+1}}) = (א - \overline{z_n}) - K \times (א + \zeta_n - \overline{z_n})$$

Because $K = 1/(n+1)$

$$(א - \overline{z_{n+1}}) = (א - \overline{z_n}) - \frac{(א + \zeta_n - \overline{z_n})}{n+1}$$

$$(א - \overline{z_{n+1}}) = \left(1 - \frac{1}{n+1}\right)(א - \overline{z_n}) - \frac{\zeta_n}{n+1}$$

$$(א - \overline{z_{n+1}}) = K\, n\, (א - \overline{z_n}) - K\, \zeta_n$$

Squaring:

$$(א - \overline{z_{n+1}})^2 = (K\, n\, (א - \overline{z_n}))^2 - \left[2\, K^2\, n\, (א - \overline{z_n})\, \zeta_n\right] + K^2\, {\zeta_n}^2$$

Now take the expectation value of both sides of this equation over the distribution of ζ. Assume this distribution has zero mean, so the middle term in square brackets vanishes. Assume the distribution of ζ has constant variance. Write this variance as capital zeta, Z.

Let us also write P_n for the n^{th} estimate of the *variance of the mean* with respect to $\varkappa$, namely $E\left[(\varkappa - \overline{z_n})^2\right]$; E is traditional notation. We get

$$P_{n+1} = K^2\left(n^2 P_n + Z\right)$$

This recurrence has a closed-form solution for $n > 0$, that being $P_n = Z/n$. We can show that

$$P_{n+1} = P_n - K^2 D$$

where $D = Z + P_n$. For $n = 0$, $P_1 = K^2 Z$.

This is the preferred form for the variance of the estimated mean with respect to the population mean $\varkappa$ because the correction is negative, reminding us that this variance decreases as observations accumulate. We see a similar form below where we develop the Kalman filter.

We can also see that $K = P/D$, which, again, is a preferred form that shows up in the Kalman filter itself.

Remark: Biased Observation Noise

This entire derivation fails if observations are biased, that is, ζ has non-zero mean. Mitigate by adding a constant term, the *bias*, and by estimating the bias, restoring the observations to a theoretical zero mean. This becomes clear with the full Kalman filter below.

Experimental and Theoretical

We now write a foldable accumulator function that tracks both the sample variance of the observations, subject to Bessel's correction, and the theoretical population variance of the mean, without Bessel's correction. This foldable defines D and expresses the gain K and the estimates using the preferred forms for recurrences identified above. We see these forms again in the full Kalman filter. Test this foldable numerically with 10,000 pseudo-random observations and an arbitrary truth mean and observation variance.

In[87]:=
```
ClearAll[cume];
cume[{var_, x_, n_, א_, Z_, P_}, z_] :=
  With[{D = Z + P},
   With[{K = P / D},
    With[{x2 = x + K (z - x)},
     With[{ssrz2 = (n - 1) var + K n (z - x)^2,
       P2 = If[n > 0, P - K^2 D, K^2 Z]},
      {ssrz2/Max[1, n], x2, n + 1, א, Z, P2}]]]];
ClearAll[zs, result];
zs = RandomVariate[NormalDistribution[42, 1.5], 10 000];
(result = Fold[cume, {0, 0, 0, 42, 1.5^2, 1 000 000 000.}, zs]);
{{"var", "mean", "count", "א", "Z", "P"}, result}^T // MatrixForm
```

Out[91]//MatrixForm=

var	2.21312
mean	42.0258
count	10 000
א	42
Z	2.25
P	0.000225

for LaTeX:

In[92]:=
```
ClearAll[cume];
cume[{var_, x_, n_, א_, Z_, P_}, z_] :=
  With[{D = Z + P},
   With[{K = P / D},
    With[{x2 = x + K (z - x)},
     With[{ssrz2 = (n - 1) var + K n (z - x)^2,
       P2 = If[n > 0, P - K^2 D, K^2 Z]},
      {ssrz2/Max[1, n], x2, n + 1, א, Z, P2}]]]];
ClearAll[zs, result];
zs = RandomVariate[NormalDistribution[42, 1.5], 10 000];
(result = Fold[cume, {0, 0, 0, 42, 1.5^2, 1 000 000 000.}, zs]);
{{"var", "mean", "count", "א", "Z", "P"}, result}^T // TeXForm
```

Out[96]//TeXForm=
```
\left(
\begin{array}{cc}
 \text{var} & 2.22644 \\
 \text{mean} & 42.0134 \\
 \text{count} & 10000 \\
 \aleph  & 42 \\
 \text{Z} & 2.25 \\
 \text{P} & 0.000225 \\
\end{array}
\right)
```

Sneak Preview: Kalman Calculates the Mean

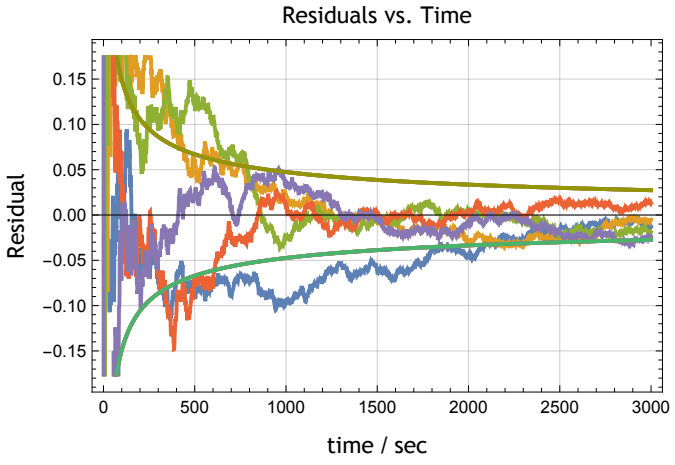
For now, the interesting thing to note is that Kalman, when seeded with 0 *a-priori* state and infinite *a-priori* covariance, computes the mean just as above. We present five different forms for the updated covariance. These forms are mathematically equivalent (we omit proofs), but not numerically equivalent, as we see next:

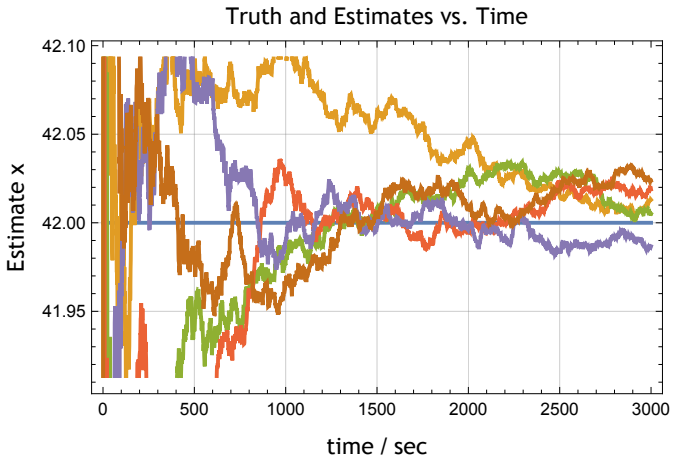
First, a run of 3000 data, showing the occasional, small numerical differences amongst the five formulae for covariance. Eventually, these differences become catastrophic and we must deal with them. In this small example, they're fine.

Note that *kalman* is an ordinary foldable once it has been fed a constant noise covariance, *Z*. The updatable data package is the mean *x* and its covariance *P*. The individual data packets are the pairs of model vector *A* and scalar observations *z*.

```
ClearAll[kalman];
kalman[z_][{x_, P_}, {A_, z_}] :=
  Module[{D, K, L, P1, P2, P3, P4, P5, precision = 1. * 10^-6},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = id[len[P]] - K.A;
   P1 = L.P; P2 = (L.P.L^T + K.z.K^T); P3 = (P - K.D.K^T); P4 = P.L; P5 = P.(L^T);
   If[Not[Round[P1, precision] === Round[P2, precision] ===
      Round[P3, precision] === Round[P4, precision] === Round[P5, precision]],
    Throw[<|"N" → Not[P1 == P2 == P3 == P4 == P5],
      "P1" → P1, "P2" → P2, "P3" → P3, "P4" → P4, "P5" → P5,
      "P1-P2" → (P1 - P2),
      "P2-P3" → (P2 - P3),
      "P1-P3" → (P1 - P3),
      "P1-P4" → (P1 - P4),
      "P1-P5" → (P1 - P5)|>]];
   (*Print[<|"N"→Not[P1==P2==P3],"P1"→P1,"P2"→P2,
      "P3"→P3,"P1-P2"→(P1-P2),"P2-P3"→(P2-P3),"P1-P3"→(P1-P3)|>];*)
   (*Print[<|"P1"→P1,"P2"→P2,"P3"→P3,
      "P1-P2"→(P1-P2),"P2-P3"→(P2-P3),"P1-P3"→(P1-P3)|>];*)
   {x + K.(z - A.x), P4}];(* P4 is an arbitrary favorite *)
With[{σz = 1.5, fooey$ = SeedRandom[43]},
 With[{aPrioriState = zero[1, 1],
   aPrioriCovariance = 1 000 000 000 id[1],
   z = col[{σz^2}]},
  experiment1[0, 3000, 1, aPrioriState, aPrioriCovariance, z,
   {dt, t} ↦ {id[1], col[{RandomVariate[NormalDistribution[42, σz]]}]},
   t ↦ 42.,
   5]]]
```

Out[99]=





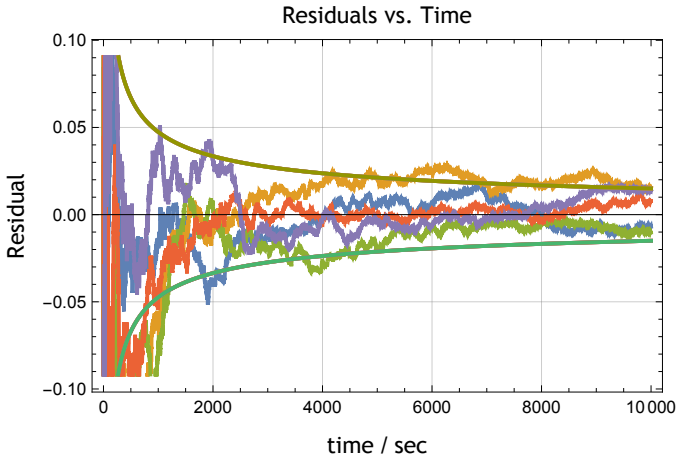
% position resids < σ (E >= 68)	76.8211
position resids >= σ	3478.
position resids < σ	11527.
average position resid	-0.00383109
theoretical std dev	0.0273816
std dev of position resids	0.0694131

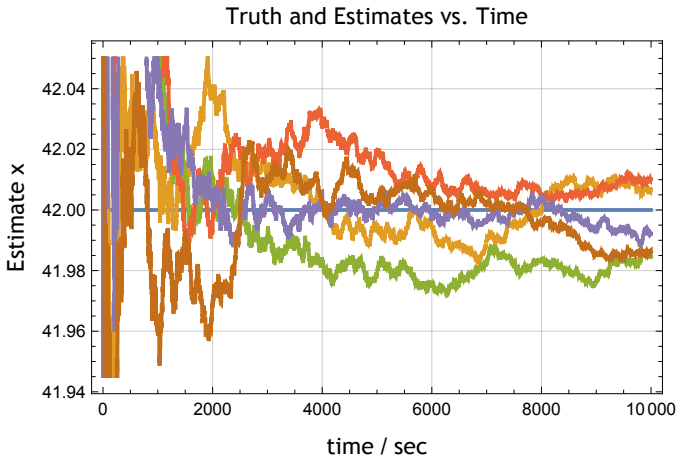
Now, a run of 10,000 data, picking P3 because it's the one most frequently seen in publications. It's often not the best because it can underflow, producing nonsensical negative covariance, as we shall see:

In[100]:=

```
ClearAll[kalman];
kalman[z_][{x_, P_}, {A_, z_}] :=
  Module[{D, K},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   {x + K.(z - A.x), P - K.D.K^T}];
With[{σz = 1.5},
 With[{aPrioriState = zero[1, 1],
   aPrioriCovariance = 1 000 000 000 id[1],
   z = col[{σz^2}]},
  experiment1[0, 10 000, 1, aPrioriState, aPrioriCovariance, z,
   {dt, t} ↦ {
     id[1],
     col[{RandomVariate[NormalDistribution[42, σz]]}]},
   t ↦ 42.,
   5]]]
```

Out[102]=





% position resids < σ (E >= 68)	83.7876
position resids >= σ	8107.
position resids < σ	41898.
average position resid	-0.00561872
theoretical std dev	0.0149993
std dev of position resids	0.0393936

Part 1: Time-Independent Model

The following is my favorite and consistent test-case. I use it for validating all other Kalman filters. It is a cubic function of time that depends linearly on four coefficients that we estimate. The ground truth for these coefficients is $\begin{pmatrix} -3 \\ 9 \\ -4 \\ -5 \end{pmatrix}$. Faked noisy data and partial derivatives of the model with respect to the coefficients, namely $\left[1, t, t^2, t^3\right]$, evaluated at times [−2, −1, 0, 1, 2] constitute the **testCase** below.

With Inverse

Inverting *D* is numerically hazardous and contraindicated. We present it, here, because it's, sadly, most common in publications.

In[103]:=
```
ClearAll[testCase];
m /@ (testCase = {
    {{{1, 0., 0., 0.}}, {-2.28442}},
    {{{1, 1., 1., 1.}}, {-4.83168}},
    {{{1, -1., 1., -1.}}, {-10.4601}},
    {{{1, -2., 4., -8.}}, {1.40488}},
    {{{1, 2., 4., 8.}}, {-40.8079}}})
```

Out[104]=
$$\left\{\begin{pmatrix}\{1, 0., 0., 0.\}\\ -2.28442\end{pmatrix}, \begin{pmatrix}\{1, 1., 1., 1.\}\\ -4.83168\end{pmatrix}, \begin{pmatrix}\{1, -1., 1., -1.\}\\ -10.4601\end{pmatrix}, \begin{pmatrix}\{1, -2., 4., -8.\}\\ 1.40488\end{pmatrix}, \begin{pmatrix}\{1, 2., 4., 8.\}\\ -40.8079\end{pmatrix}\right\}$$

In[105]:=
```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {A_, z_}] :=
  Module[{D, K},
   D = Z + A.P.A^T;
   K = P.A^T.Inverse[D];
   {x + K.(z - A.x), P - K.A.P}];
```

In[107]:=

```
m@(m/@#&/@
    Chop[FoldList[
      kalman[IdentityMatrix[1]],
      {col[{0, 0, 0, 0}],
       id[4]*1000.0},
      testCase]])
```

Out[107]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix}0\\0\\0\\0\end{pmatrix} & \begin{pmatrix}1000. & 0 & 0 & 0\\ 0 & 1000. & 0 & 0\\ 0 & 0 & 1000. & 0\\ 0 & 0 & 0 & 1000.\end{pmatrix} \\ \begin{pmatrix}-2.28214\\0\\0\\0\end{pmatrix} & \begin{pmatrix}0.999001 & 0 & 0 & 0\\ 0 & 1000. & 0 & 0\\ 0 & 0 & 1000. & 0\\ 0 & 0 & 0 & 1000.\end{pmatrix} \\ \begin{pmatrix}-2.28299\\-0.849281\\-0.849281\\-0.849281\end{pmatrix} & \begin{pmatrix}0.998669 & -0.332779 & -0.332779 & -0.332779\\ -0.332779 & 666.889 & -333.111 & -333.111\\ -0.332779 & -333.111 & 666.889 & -333.111\\ -0.332779 & -333.111 & -333.111 & 666.889\end{pmatrix} \\ \begin{pmatrix}-2.28749\\1.40675\\-5.35572\\1.40675\end{pmatrix} & \begin{pmatrix}0.998004 & 0 & -0.997506 & 0\\ 0 & 500.125 & 0 & -499.875\\ -0.997506 & 0 & 1.49676 & 0\\ 0 & -499.875 & 0 & 500.125\end{pmatrix} \\ \begin{pmatrix}-2.29399\\7.92347\\-5.34488\\-5.1154\end{pmatrix} & \begin{pmatrix}0.997508 & 0.49762 & -0.996678 & -0.498035\\ 0.49762 & 1.3855 & -0.829836 & -0.719881\\ -0.996678 & -0.829836 & 1.49538 & 0.830528\\ -0.498035 & -0.719881 & 0.830528 & 0.553787\end{pmatrix} \\ \begin{pmatrix}-2.97423\\7.2624\\-4.21051\\-4.45378\end{pmatrix} & \begin{pmatrix}0.485458 & 0 & -0.142778 & 0\\ 0 & 0.901908 & 0 & -0.235882\\ -0.142778 & 0 & 0.0714031 & 0\\ 0 & -0.235882 & 0 & 0.0693839\end{pmatrix} \end{pmatrix}$$

No Inverse

We get rid of the inverse, *pro forma*, via Mathematica's built-in *LinearSolve*, though inverse does not cause numerical issues in this tiny example:

In[108]:=

```
ClearAll[noInverseKalman];
noInverseKalman[Z_][{x_, P_}, {A_, z_}] :=
  Module[{PAT, D, KRes, KAP},
   PAT = P.A^T;
   D = Z + A.PAT;
   KRes = PAT.LinearSolve[D, z - A.x];
   KAP = PAT.LinearSolve[D, A.P];
   Chop@{x + KRes, P - KAP}];
```

In[110]:=

```
m@(m/@#&/@
    Chop[FoldList[
      noInverseKalman[IdentityMatrix[1]],
      {col[{0, 0, 0, 0}],
       id[4]*1000.0},
      testCase]])
```

Out[110]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix}0\\0\\0\\0\end{pmatrix} & \begin{pmatrix}1000. & 0 & 0 & 0\\ 0 & 1000. & 0 & 0\\ 0 & 0 & 1000. & 0\\ 0 & 0 & 0 & 1000.\end{pmatrix} \\ \begin{pmatrix}-2.28214\\0\\0\\0\end{pmatrix} & \begin{pmatrix}0.999001 & 0 & 0 & 0\\ 0 & 1000. & 0 & 0\\ 0 & 0 & 1000. & 0\\ 0 & 0 & 0 & 1000.\end{pmatrix} \\ \begin{pmatrix}-2.28299\\-0.849281\\-0.849281\\-0.849281\end{pmatrix} & \begin{pmatrix}0.998669 & -0.332779 & -0.332779 & -0.332779\\ -0.332779 & 666.889 & -333.111 & -333.111\\ -0.332779 & -333.111 & 666.889 & -333.111\\ -0.332779 & -333.111 & -333.111 & 666.889\end{pmatrix} \\ \begin{pmatrix}-2.28749\\1.40675\\-5.35572\\1.40675\end{pmatrix} & \begin{pmatrix}0.998004 & 0 & -0.997506 & 0\\ 0 & 500.125 & 0 & -499.875\\ -0.997506 & 0 & 1.49676 & 0\\ 0 & -499.875 & 0 & 500.125\end{pmatrix} \\ \begin{pmatrix}-2.29399\\7.92347\\-5.34488\\-5.1154\end{pmatrix} & \begin{pmatrix}0.997508 & 0.49762 & -0.996678 & -0.498035\\ 0.49762 & 1.3855 & -0.829836 & -0.719881\\ -0.996678 & -0.829836 & 1.49538 & 0.830528\\ -0.498035 & -0.719881 & 0.830528 & 0.553787\end{pmatrix} \\ \begin{pmatrix}-2.97423\\7.2624\\-4.21051\\-4.45378\end{pmatrix} & \begin{pmatrix}0.485458 & 0 & -0.142778 & 0\\ 0 & 0.901908 & 0 & -0.235882\\ -0.142778 & 0 & 0.0714031 & 0\\ 0 & -0.235882 & 0 & 0.0693839\end{pmatrix} \end{pmatrix}$$

With Random Data

In[111]:=

```
final$ = Chop[Fold[
    noInverseKalman[IdentityMatrix[1]],
    {col[{0, 0, 0, 0}],
     id[4]*1000.0},
    testCase]]
```

Out[111]=

```
{{{-2.97423}, {7.2624}, {-4.21051}, {-4.45378}},
 {{0.485458, 0, -0.142778, 0}, {0, 0.901908, 0, -0.235882},
  {-0.142778, 0, 0.0714031, 0}, {0, -0.235882, 0, 0.0693839}}}
```

DEBUG: Missing *aRow*

In[112]:=
```
With[{trials = 1},
 With[{ts = RandomReal[{-2, 2}, trials]},
  With[{as = aRow /@ ts},
   With[{zs = Table[a.groundTruth + RandomVariate[NormalDistribution[]], {a, as}]},
    Chop@Fold[
      noInverseKalman[IdentityMatrix[1]],
      finals$, MapThread[List, {as, zs}]]]]]]
```

Out[112]=
```
noInverseKalman[{{1}}][finals$,
 {aRow[-1.54337], 0.434697 + aRow[-1.54337].{{-3}, {9}, {-4}, {-5}}}]
```

Unpacked

In[113]:=
```
A$ = testCase[[1, 1]]
```

Out[113]=
```
{{1, 0., 0., 0.}}
```

In[114]:=
```
z$ = testCase[[1, 2]]
```

Out[114]=
```
{-2.28442}
```

In[115]:=
```
P$ = 1000. id[4]
```

Out[115]=
```
{{1000., 0., 0., 0.}, {0., 1000., 0., 0.}, {0., 0., 1000., 0.}, {0., 0., 0., 1000.}}
```

In[116]:=
```
PAT$ = P$.A$ᵀ
```

Out[116]=
```
{{1000.}, {0.}, {0.}, {0.}}
```

In[117]:=
```
Z$ = col[{1}]
```

Out[117]=
```
{{1}}
```

In[118]:=
```
D$ = Z$ + A$.PAT$
```

Out[118]=
```
{{1001.}}
```

In[119]:=
```
x$ = col[{0, 0, 0, 0}]
```

Out[119]=
```
{{0}, {0}, {0}, {0}}
```

In[120]:=
```
A$.x$
```
Out[120]=
```
{{0.}}
```

In[121]:=
```
Res$ = z$ - A$.x$
```
Out[121]=
```
{{-2.28442}}
```

In[122]:=
```
LinearSolve[D$, z$ - A$.x$]
```
Out[122]=
```
{{-0.00228214}}
```

In[123]:=
```
A$.P$
```
Out[123]=
```
{{1000., 0., 0., 0.}}
```

In[124]:=
```
Inverse[D$].A$.P$
```
Out[124]=
```
{{0.999001, 0., 0., 0.}}
```

In[125]:=
```
PAT$.Inverse[D$].A$.P$
```
Out[125]=
```
{{999.001, 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}}
```

Insensitivity to A-Priori State Covariance

Reminder that

$$\texttt{groundTruth} = \begin{pmatrix} -3 \\ 9 \\ -4 \\ -5 \end{pmatrix}$$

In the displays, *position* refers to the first coefficient, with ground truth −3, and *speed* refers to the second coefficient, with ground truth 9. These labels look forward to tracking examples with physical position and speed states of the system.

In[126]:=
```
m /@ Chop[Fold[kalman[id@1], {col[{0, 0, 0, 0}], id[4] * 1000.0}, testCase]]
```
Out[126]=
$$\left\{\begin{pmatrix} -2.97423 \\ 7.2624 \\ -4.21051 \\ -4.45378 \end{pmatrix}, \begin{pmatrix} 0.485458 & 0 & -0.142778 & 0 \\ 0 & 0.901908 & 0 & -0.235882 \\ -0.142778 & 0 & 0.0714031 & 0 \\ 0 & -0.235882 & 0 & 0.0693839 \end{pmatrix}\right\}$$

In[127]:=

```
m /@ Chop[Fold[kalman[id@1], {col[{0, 0, 0, 0}], id[4] * 1 000 000.0}, testCase]]
```

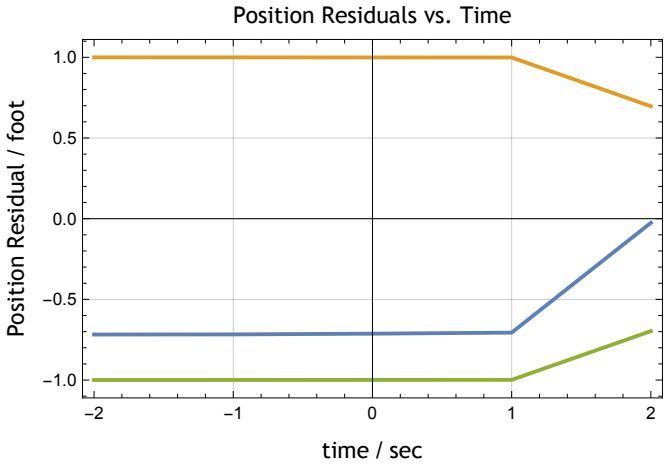
Out[127]=

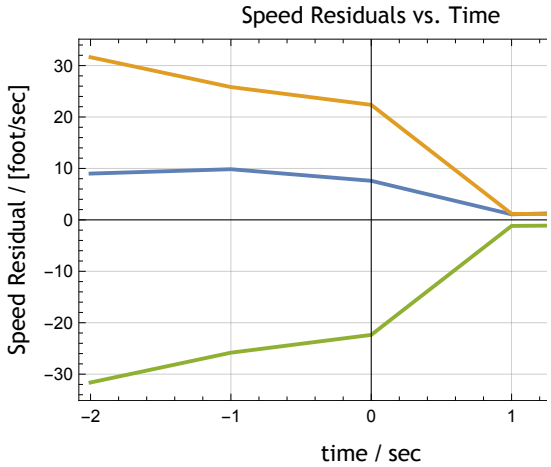
$$\left\{\begin{pmatrix} -2.97507 \\ 7.27 \\ -4.21039 \\ -4.4558 \end{pmatrix}, \begin{pmatrix} 0.485714 & 0 & -0.142857 & 0 \\ 0 & 0.902777 & 1.26198 \times 10^{-10} & -0.236111 \\ -0.142857 & 0 & 0.0714285 & 0 \\ 0 & -0.236111 & 0 & 0.0694444 \end{pmatrix}\right\}$$

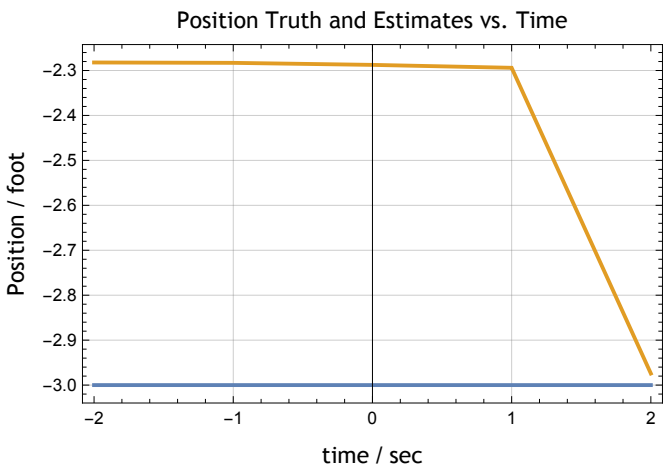
In[128]:=

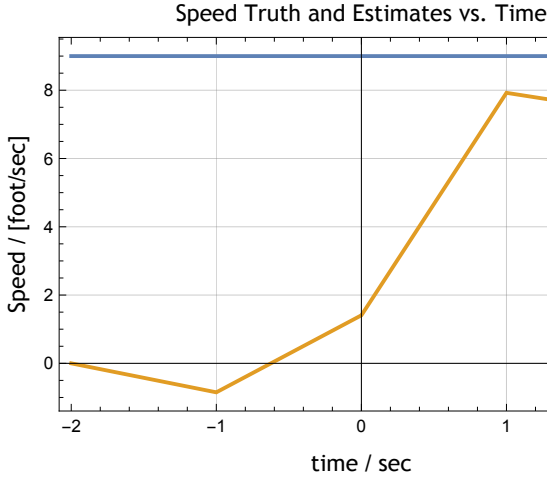
```
With[{σz = 1.},
 With[{aPrioriState = zero[4, 1],
   aPrioriCovariance = 1000 id[4],
   z = col[{σz^2}]},
  experiment[-2, 2, 1, aPrioriState, aPrioriCovariance, z,
   {dt, t} ↦ {testCase[[3 + t, 1]], testCase[[3 + t, 2]]},
   t ↦ groundTruth[[1, 1]],
   t ↦ groundTruth[[2, 1]],
   1]]]
```

Out[128]=









% position resids < σ (E >= 68)	100.
position resids >= σ	0.
position resids < σ	5.
average position resid	-0.575834
theoretical std dev	0.696748
std dev of position resids	0.307529

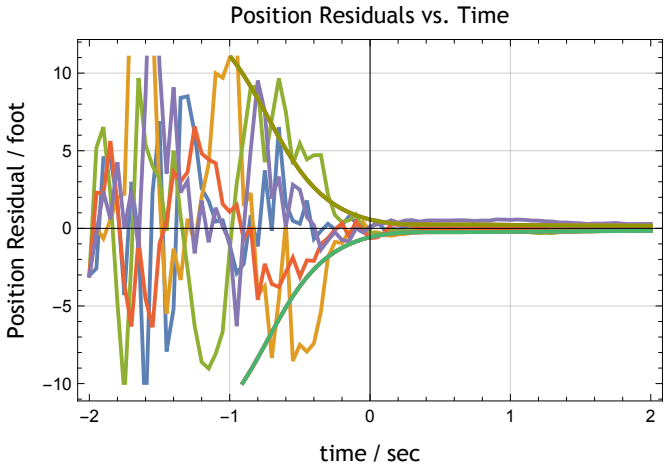
% speed resids < σ (E >= 68)	8
speed resids >= σ	
speed resids < σ	
average speed resid	5.8
theoretical std dev	0.9
std dev of speed resids	4.

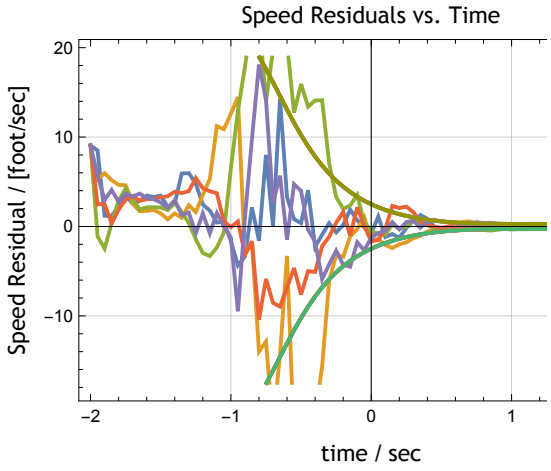
Do an experiment with a randomized version of the same model.

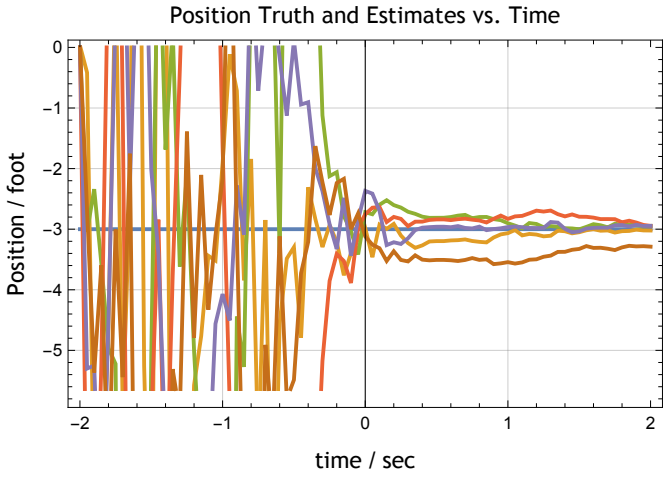
In[129]:=

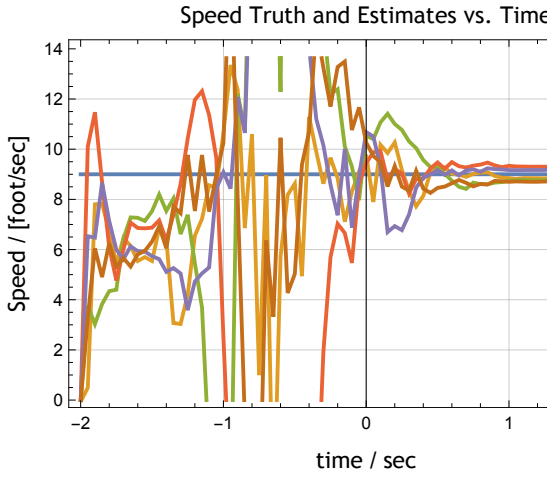
```
With[{σz = 1.},
 With[{aPrioriState = zero[4, 1],
   aPrioriCovariance = 1000[4],
   z = col[{σz^2}]},
  Module[{A},
   A[t_] := row[{1, t, t^2, t^3}];
   experiment[-2, 2, .05, zero[4, 1], 1000 id[4], 1.0,
    {dt, t} ↦ {A[t], scalar[A[t].groundTruth + gen[z]]},
    t ↦ groundTruth[[1, 1]],
    t ↦ groundTruth[[2, 1]],
    5]]]]
```

Out[129]=









% position resids < σ (E >= 68)	79.5062
position resids >= σ	83.
position resids < σ	322.
average position resid	0.586442
theoretical std dev	0.166685
std dev of position resids	3.85079

% speed resids < σ (E >= 68)	83
speed resids >= σ	
speed resids < σ	3
average speed resid	0.8
theoretical std dev	0.
std dev of speed resids	5.

Part 2: Tracking (Time-Dependent Model)

Process-Noise Matrix

Process noise enters the last state, and with a constant standard deviation. Integrate the process-noise covariance matrix over a discrete time interval and over a particular model of system dynamics to add its effects to the filter.

In[130]:=

```
ClearAll[Q, Φ, σx, t];
Q = {{0, 0, 0}, {0, 0, 0}, {0, 0, ξ^2}};
Φ[t_] := {{1, t, t^2/2}, {0, 1, t}, {0, 0, 1}};
Integrate[Φ[t].Q.Φ[t]^T, {t, 0, δt}] // m
```

Out[133]//MatrixForm=

$$\begin{pmatrix} \frac{\delta t^5 \xi^2}{20} & \frac{\delta t^4 \xi^2}{8} & \frac{\delta t^3 \xi^2}{6} \\ \frac{\delta t^4 \xi^2}{8} & \frac{\delta t^3 \xi^2}{3} & \frac{\delta t^2 \xi^2}{2} \\ \frac{\delta t^3 \xi^2}{6} & \frac{\delta t^2 \xi^2}{2} & \delta t\, \xi^2 \end{pmatrix}$$

As A Fold

- **kalman** is the Kalman filter with time-dependent dynamical process.
- At the bottom, we validate that **kalman** degenerates to the original static case.
- Ξ is the integrated process-noise matrix. It depend on time and on the size of the time step.
- Φ is the integral of the system-dynamics matrix F; more precisely, Φ is `Exp[F t]`.
- Γ is the time-step integral of system response G propagated by Φ.
- Ξ, Φ, Γ, and *u* may all depend on time and on the size of the time step.

In[134]:=

```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   {x2 + K.(z - A.x2), P2 - K.D.K^T}];
(* demonstrate that the filter degenerates to the earlier form given a time-
 independent model *)
m /@
 Chop[
  With[{
    Ξ = zero[4], Z = id[1],
    Φ = id[4], Γ = zero[4, 1], u = zero[1]},
   Fold[
    kalman[Z],
    {col[{0, 0, 0, 0}],
     id[4] * 1000.0},
    {Ξ, Φ, Γ, u} ⊕ # & /@ testCase
   ]]]
```

Out[136]=

$$\left\{\begin{pmatrix} -2.97423 \\ 7.2624 \\ -4.21051 \\ -4.45378 \end{pmatrix}, \begin{pmatrix} 0.485458 & 0 & -0.142778 & 0 \\ 0 & 0.901908 & 0 & -0.235882 \\ -0.142778 & 0 & 0.0714031 & 0 \\ 0 & -0.235882 & 0 & 0.0693839 \end{pmatrix}\right\}$$

In[137]:=

```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
   (*If[Not[PT1==PT2==PT3],
     Throw[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>]];*)
   (*Print[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
      "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>];*)
   (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"PT1-PT2"→Chop@(PT1-PT2),
      "PT2-PT3"→Chop@(PT2-PT3),"PT1-PT3"→Chop@(PT1-PT3)|>];*)
   (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"D"→D|>];*)
   {x2 + K.(z - A.x2), PT3}];
(* demonstrate that the filter degenerates to the earlier form given a time-
 independent model *)
m /@
 Chop[
  With[{
    Ξ = zero[4], Z = id[1],
    Φ = id[4], Γ = zero[4, 1], u = zero[1]},
   Fold[
    kalman[Z],
    {col[{0, 0, 0, 0}],
     id[4] * 1000.0},
    {Ξ, Φ, Γ, u} ⊕ # & /@ testCase
   ]]]
```

Out[139]=

$$\left\{\begin{pmatrix} -2.97423 \\ 7.2624 \\ -4.21051 \\ -4.45378 \end{pmatrix}, \begin{pmatrix} 0.485458 & 0 & -0.142778 & 0 \\ 0 & 0.901908 & 0 & -0.235882 \\ -0.142778 & 0 & 0.0714031 & 0 \\ 0 & -0.235882 & 0 & 0.0693839 \end{pmatrix}\right\}$$

Track a Falling Object

Reproduce examples from Zarchan and Musoff, *Fundamentals of Kalman Filtering, A Practical Approach*, Ch. 4, track a falling object, no air drag.

Three-State Falling Object

There are three states: the position x, velocity $\dot{x}$, acceleration $\ddot{x}$. The observations z directly measure the position state.

State-space form:

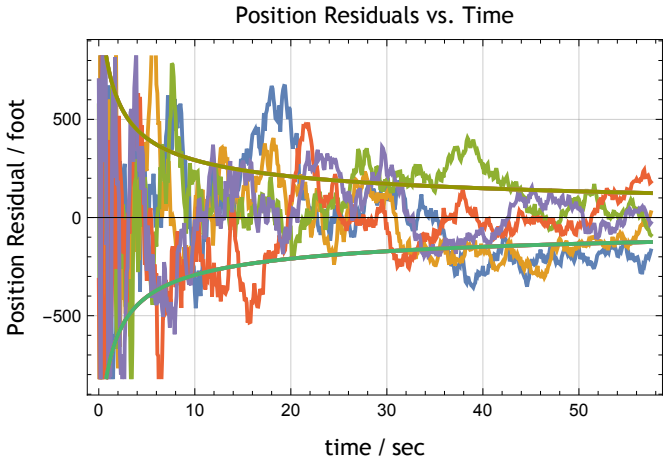
In[140]:=

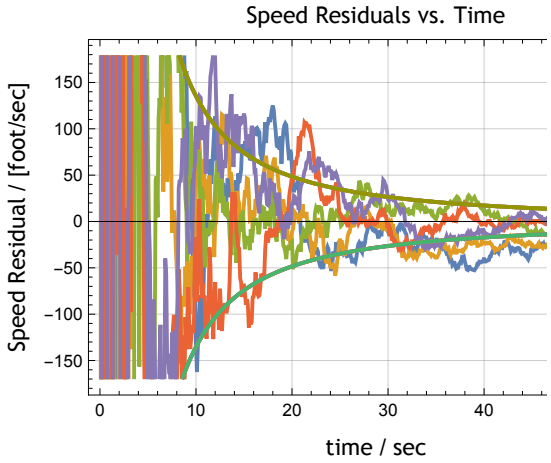
```
ClearAll[x, xd, xdd, Ψ, u, F, Φ, Ξc, Ξ, G, Γ, A];
u[t_] := col[{g}];
Ψ = {{x}, {xd}, {xdd}}; F = {{0, 1, 0}, {0, 0, 1}, {0, 0, 0}}; G = {{0}, {0}, {1}};
Φ[dt_] := {{1, dt, dt^2 / 2}, {0, 1, dt}, {0, 0, 1}}; Γ[dt_] := {{dt^2 / 2}, {dt}, {1}};
Ξc = {{0, 0, 0}, {0, 0, 0}, {0, 0, 1}};
Ξ[dt_] := {{dt^5/20, dt^4/8, dt^3/6}, {dt^4/8, dt^3/3, dt^2/2}, {dt^3/3, dt^2/2, dt}};
A[t_] := (1 0 0);
```

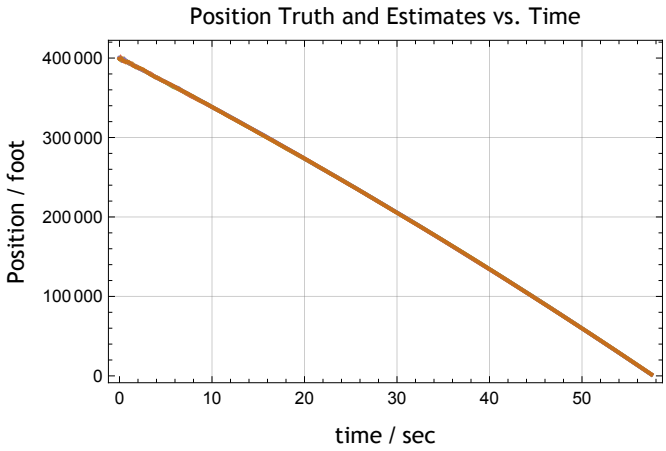
In[147]:=

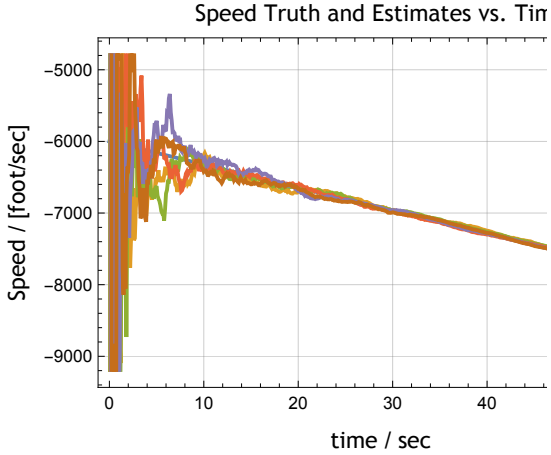
```
With[{x0 = 400 000., v0 = -6000., a0 = -32.2, σz = 1000., σx = 0},
 With[{aPrioriState = zero[3, 1](*col[{x0,v0,a0}]*),
   aPrioriCovariance = 1 000 000 000 000 id[3],
   z = col[{σz^2}]},
  SeedRandom[42];
  experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance, z,
   {dt, t} ↦
    {σx^2 ( dt^5/20  dt^4/8  dt^3/6
            dt^4/8   dt^3/3  dt^2/2
            dt^3/3   dt^2/2  dt     ), (* process noise *)
     ( 1  dt  dt^2/2
       0  1   dt
       0  0   1     ), (* system dynamics *)
     ( dt^2/2
       dt
       1     ), (* system response *)
     zero[1], (* external force *)
     (1 0 0), (* observation partials *)
     col[{x0 + v0 t + a0 t^2/2}] + gen[z]}, (* observation fake *)
   t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
   t ↦ v0 + a0 t (* true speed *), 5]]]
```

Out[147]=









% position resids < σ (E >= 68)	64.3056
position resids >= σ	1028.
position resids < σ	1852.
average position resid	5.99396
theoretical std dev	124.568
std dev of position resids	248.361

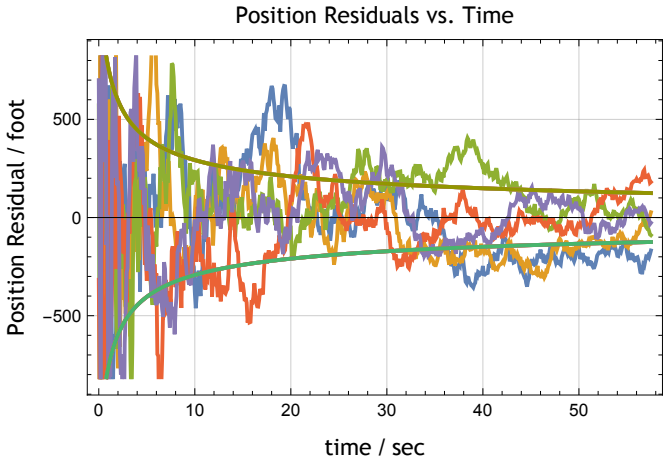
% speed resids < σ (E >= 68)	67
speed resids >= σ	9
speed resids < σ	1
average speed resid	−43
theoretical std dev	10
std dev of speed resids	23

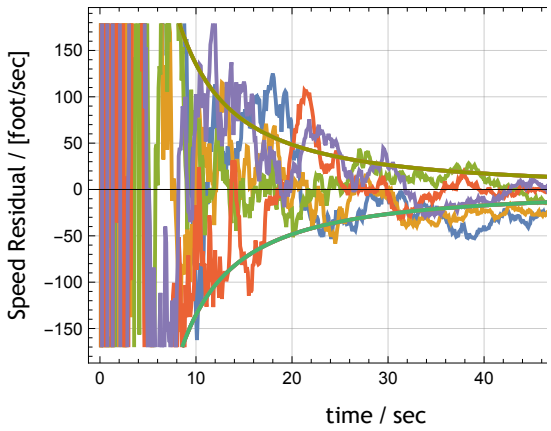
In[148]:=

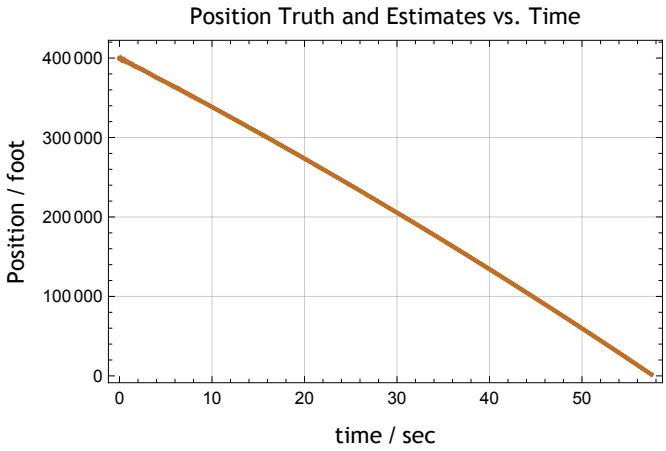
```
ClearAll[kalman];
kalman[z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.z.K^T); PT3 = (P2 - K.D.K^T);
   (*If[Not[PT1==PT2==PT3],
     Throw[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>]];*)
```

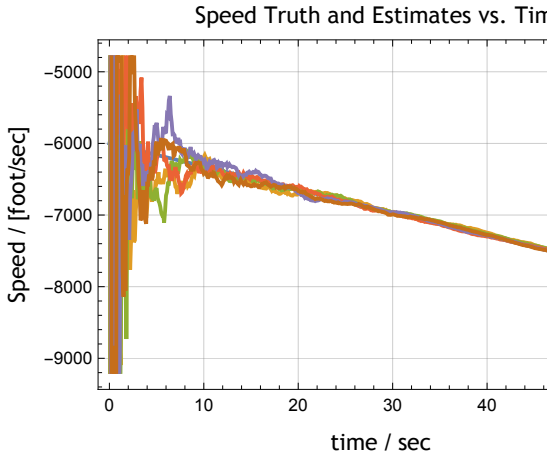
```
    (*Print[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>];*)
    (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"PT1-PT2"→Chop@(PT1-PT2),
       "PT2-PT3"→Chop@(PT2-PT3),"PT1-PT3"→Chop@(PT1-PT3)|>];*)
    (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"D"→D|>];*)
    {x2 + K.(z - A.x2), PT1}];
 With[{x0 = 400 000., v0 = -6000., a0 = -32.2, σz = 1000., σx = 0},
  With[{aPrioriState = zero[3, 1](*col[{x0,v0,a0}]*),
    aPrioriCovariance = 1 000 000 000 000 id[3],
    z = col[{σz^2}]},
   SeedRandom[42];
   experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance, z,
    {dt, t} ↦
     {σx^2 {{dt^5/20, dt^4/8, dt^3/6}, {dt^4/8, dt^3/3, dt^2/2}, {dt^3/3, dt^2/2, dt}}, (* process noise *)
      {{1, dt, dt^2/2}, {0, 1, dt}, {0, 0, 1}}, (* system dynamics *)
      {{dt^2/2}, {dt}, {1}}, (* system response *)
      zero[1], (* external force *)
      (1 0 0), (* observation partials *)
      col[{x0 + v0 t + a0 t^2/2}] + gen[z]}, (* observation fake *)
    t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
    t ↦ v0 + a0 t (* true speed *), 5]]]
```

Out[150]=









% position resids < σ (E >= 68)	64.3056
position resids >= σ	1028.
position resids < σ	1852.
average position resid	5.99389
theoretical std dev	124.567
std dev of position resids	248.361

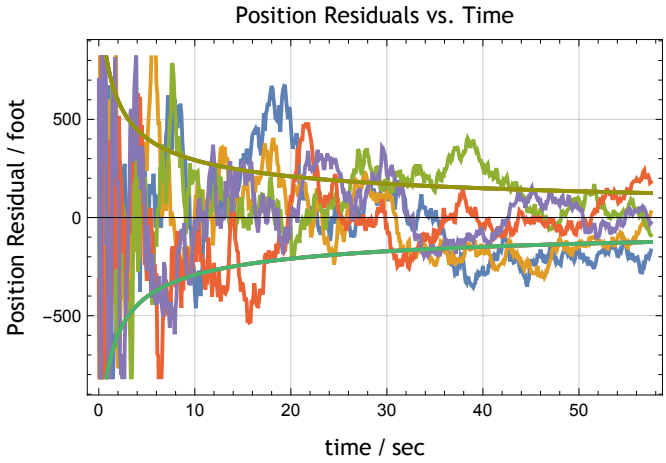
% speed resids < σ (E >= 68)	67
speed resids >= σ	9
speed resids < σ	1
average speed resid	-43
theoretical std dev	10
std dev of speed resids	23

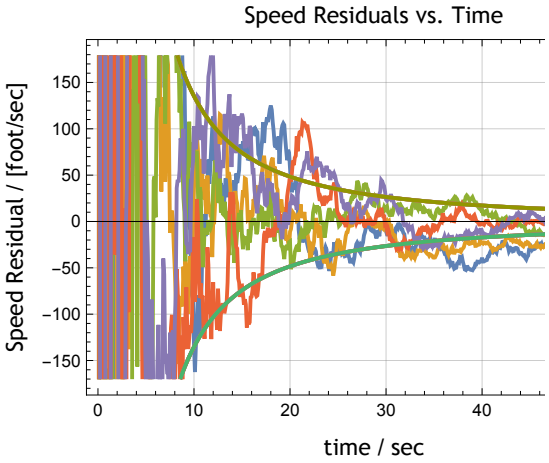
In[151]:=

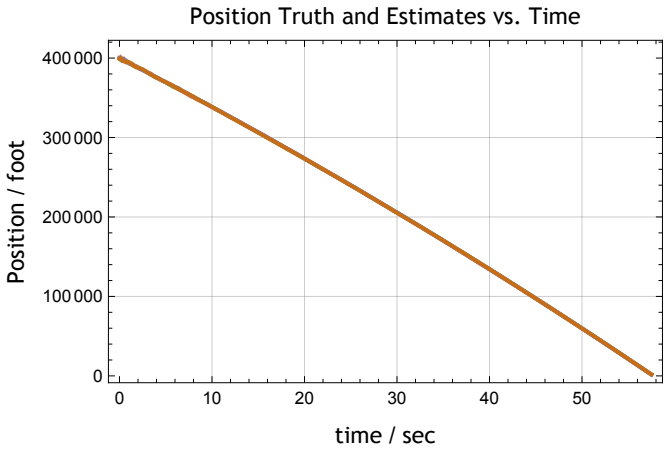
```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
   (*If[Not[PT1==PT2==PT3],
     Throw[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>]];*)
```

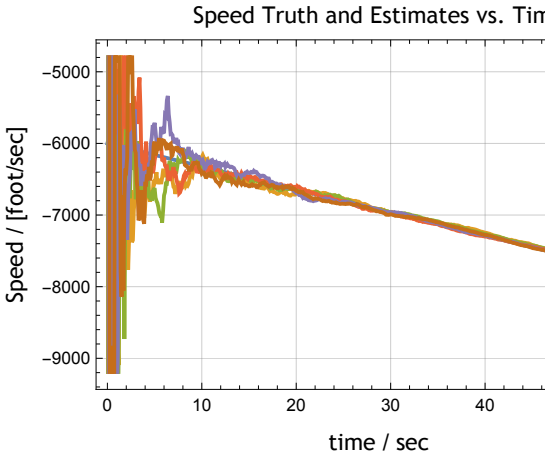
```
    (*Print[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>];*)
    (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"PT1-PT2"→Chop@(PT1-PT2),
       "PT2-PT3"→Chop@(PT2-PT3),"PT1-PT3"→Chop@(PT1-PT3)|>];*)
    (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"D"→D|>];*)
    {x2 + K.(z - A.x2), PT1}];
 With[{x0 = 400 000., v0 = -6000., a0 = -32.2, σz = 1000., σx = 0},
  With[{aPrioriState = zero[3, 1](*col[{x0,v0,a0}]*),
    aPrioriCovariance = 1 000 000 000 000 id[3],
    z = col[{σz^2}]},
   SeedRandom[42];
   experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance, z,
    {dt, t} ↦
     {σx^2 ( dt^5/20  dt^4/8  dt^3/6
             dt^4/8   dt^3/3  dt^2/2
             dt^3/3   dt^2/2  dt ), (* process noise *)
      ( 1  dt  dt^2/2
        0  1   dt
        0  0   1 ), (* system dynamics *)
      ( dt^2/2
        dt
        1 ), (* system response *)
      zero[1], (* external force *)
      (1 0 0), (* observation partials *)
      col[{x0 + v0 t + a0 t^2/2}] + gen[z]}, (* observation fake *)
    t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
    t ↦ v0 + a0 t (* true speed *), 5]]]
```

Out[153]=









% position resids < σ (E >= 68)	64.3056
position resids >= σ	1028.
position resids < σ	1852.
average position resid	5.99389
theoretical std dev	124.567
std dev of position resids	248.361

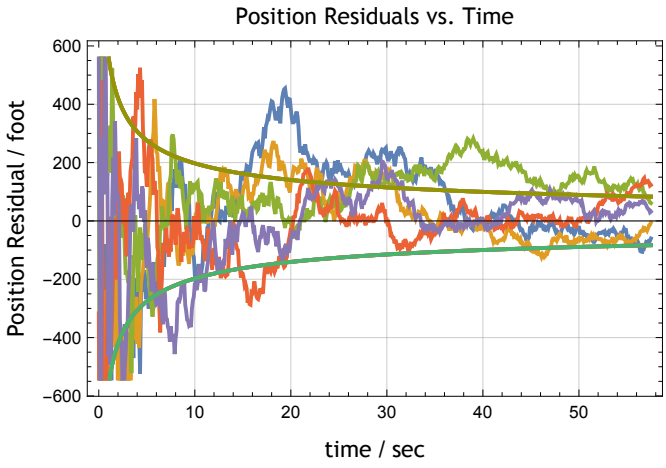
% speed resids < σ (E >= 68)	67
speed resids >= σ	9
speed resids < σ	1
average speed resid	−43
theoretical std dev	10
std dev of speed resids	23

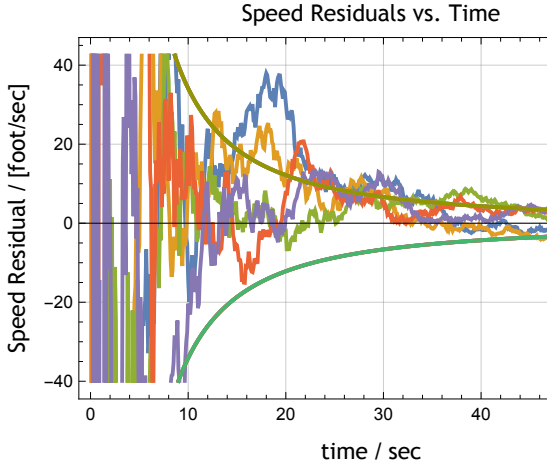
Two-State Falling Object

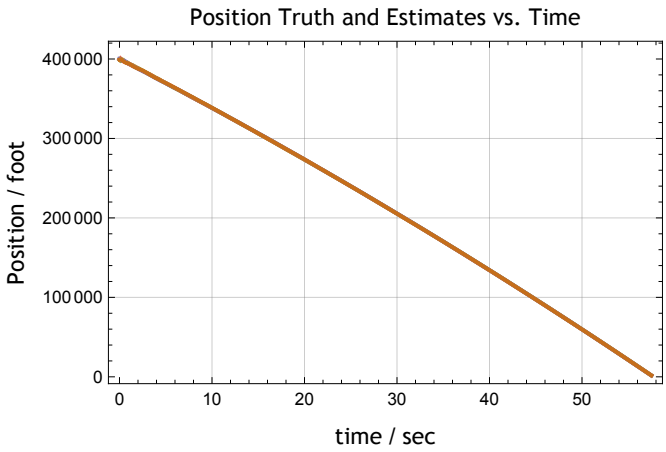
In[154]:=

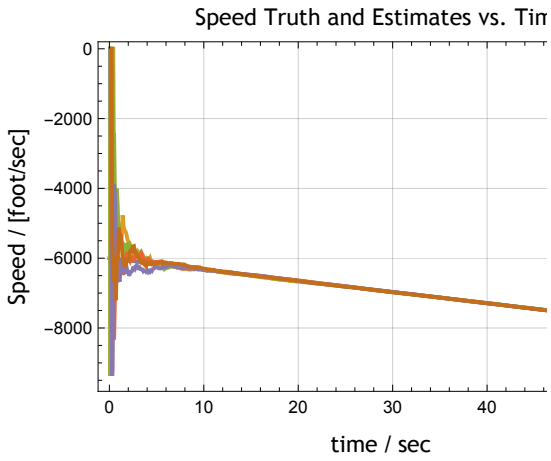
```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
   (*If[Not[PT1==PT2==PT3],
     Throw[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
       "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>]];*)
   (*Print[<|"N"→Not[PT1==PT2==PT3],"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,
      "PT1-PT2"→(PT1-PT2),"PT2-PT3"→(PT2-PT3),"PT1-PT3"→(PT1-PT3)|>];*)
   (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"PT1-PT2"→Chop@(PT1-PT2),
      "PT2-PT3"→Chop@(PT2-PT3),"PT1-PT3"→Chop@(PT1-PT3)|>];*)
   (*Print[<|"PT1"→PT1,"PT2"→PT2,"PT3"→PT3,"D"→D|>];*)
   {x2 + K.(z - A.x2), PT1}];
With[{x0 = 400 000, v0 = -6000, g = -32.2, σz = 1000., σx = 0.},
 With[{aPrioriState = zero[2, 1],
   aPrioriCovariance = 1 000 000 000 000 id[2],
   Z = col[{σz^2}]},
  SeedRandom[42];
  experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance,
   Z, (* observation-noise covariance *)
   {dt, t} ↦
    {σx^2 {{dt^3/3, dt^2/2}, {dt^2/2, dt}}, (* process noise *)
     {{1, dt}, {0, 1}}, (* system dynamics *)
     {{dt^2/2}, {dt}}, (* system response *)
     col[{g}], (* external force *)
     (1 0), (* observation partials *)
     col[{x0 + v0 t + g t^2/2}] + gen[Z] (* observation fake *)},
   t ↦ x0 + v0 t + g t^2/2, (* true position *)
   t ↦ v0 + g t (* true speed *), 5]]]
```

Out[156]=









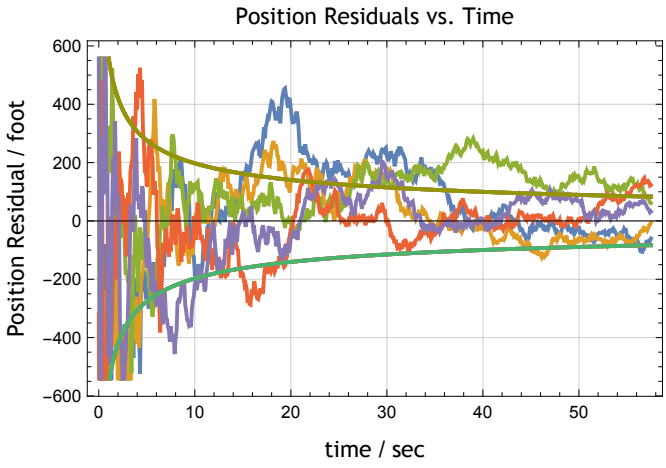
% position resids < σ (E >= 68)	67.5694
position resids >= σ	934.
position resids < σ	1946.
average position resid	23.3763
theoretical std dev	83.2249
std dev of position resids	176.477

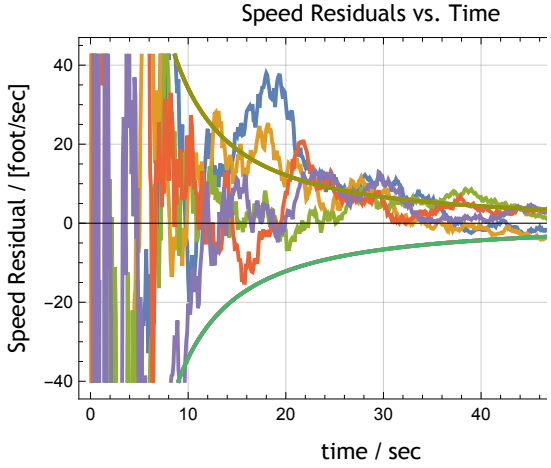
% speed resids < σ (E >= 68)	71
speed resids >= σ	8
speed resids < σ	2(
average speed resid	-11
theoretical std dev	2.5
std dev of speed resids	19

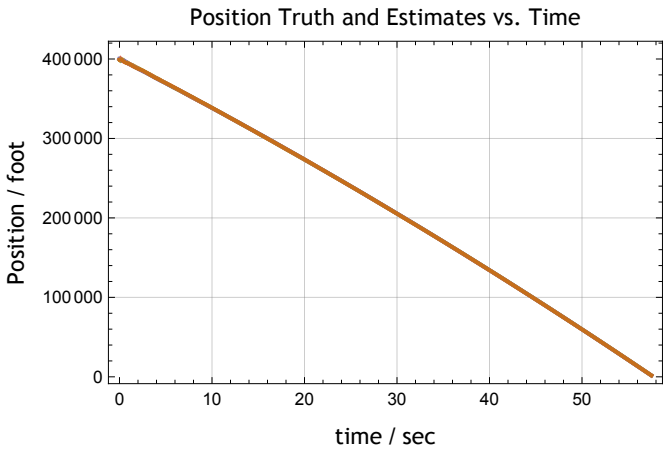
In[157]:=

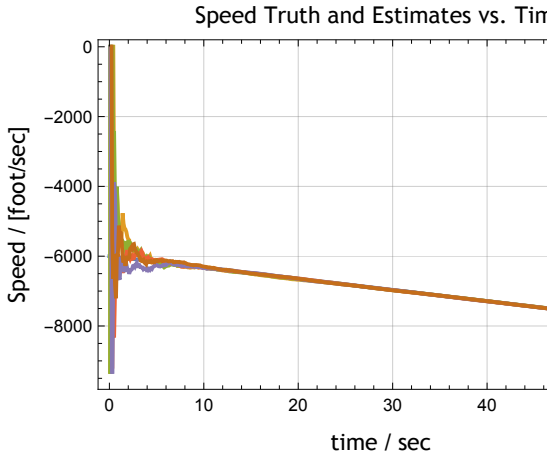
```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
   {x2 + K.(z - A.x2), PT2}];
With[{x0 = 400 000, v0 = -6000, g = -32.2, σz = 1000., σx = 0.},
 With[{aPrioriState = zero[2, 1],
   aPrioriCovariance = 1 000 000 000 000 id[2],
   Z = col[{σz^2}]},
  SeedRandom[42];
  experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance,
   Z, (* observation-noise covariance *)
   {dt, t} ↦
    {σx^2 ((dt^3/3  dt^2/2
            dt^2/2  dt), (* process noise *)
     ((1 dt
       0  1), (* system dynamics *)
     (dt^2/2
        dt), (* system response *)
     col[{g}], (* external force *)
     (1 0), (* observation partials *)
     col[{x0 + v0 t + g t^2/2}] + gen[Z] (* observation fake *)},
   t ↦ x0 + v0 t + g t^2/2, (* true position *)
   t ↦ v0 + g t (* true speed *), 5]]]
```

Out[159]=









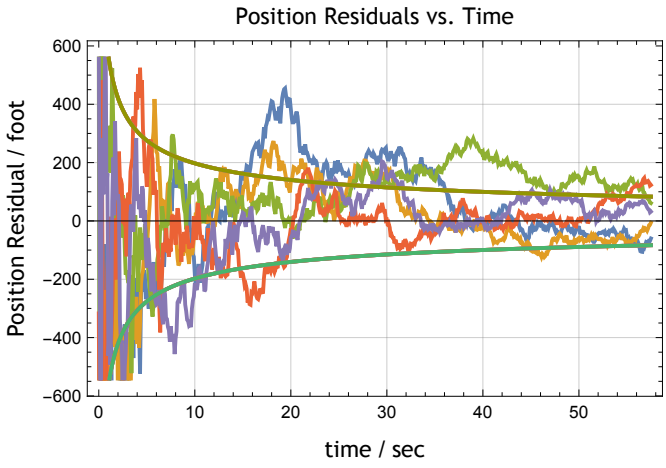
% position resids < σ (E >= 68)	67.5694
position resids >= σ	934.
position resids < σ	1946.
average position resid	23.3763
theoretical std dev	83.2249
std dev of position resids	176.477

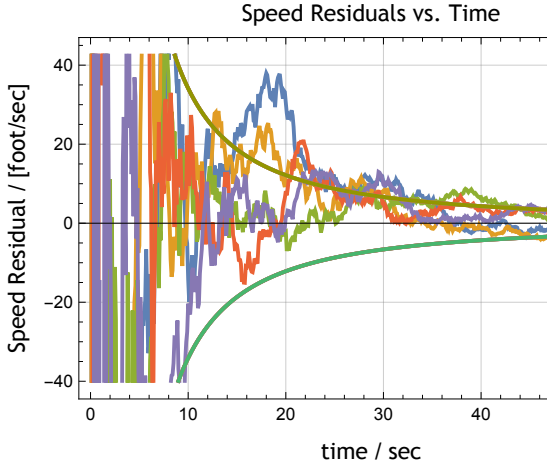
% speed resids < σ (E >= 68)	71
speed resids >= σ	8
speed resids < σ	2(
average speed resid	-11
theoretical std dev	2.5
std dev of speed resids	19

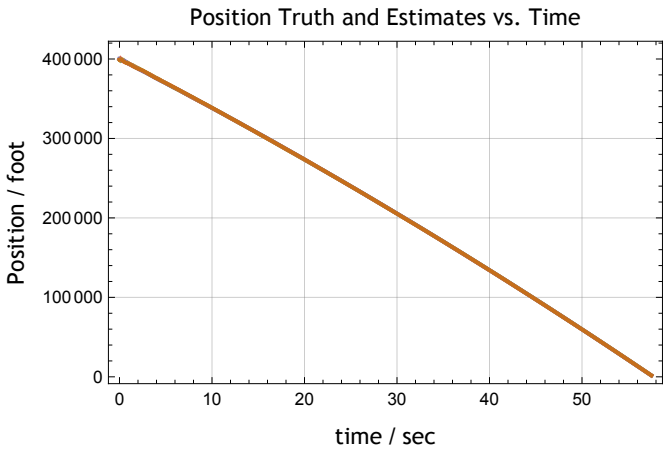
In[160]:=

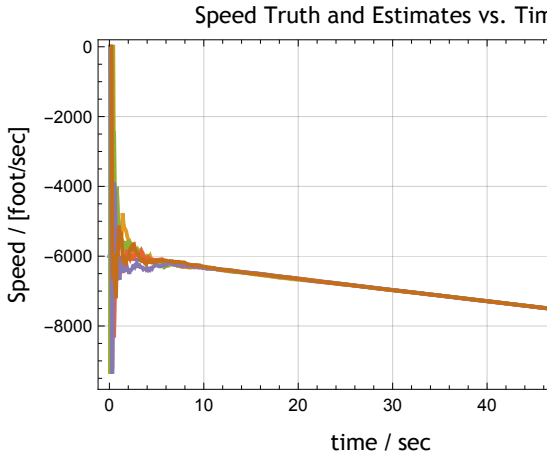
```
ClearAll[kalman];
kalman[Z_][{x_, P_}, {Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   L = id[len[P]] - K.A;
   PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
   {x2 + K.(z - A.x2), PT3}];
With[{x0 = 400 000, v0 = -6000, g = -32.2, σz = 1000., σx = 0.},
 With[{aPrioriState = zero[2, 1],
   aPrioriCovariance = 1 000 000 000 000 id[2],
   Z = col[{σz^2}]},
  SeedRandom[42];
  experiment[0, 57.5, 0.1, aPrioriState, aPrioriCovariance,
   Z, (* observation-noise covariance *)
   {dt, t} ↦
    {σx^2 ((dt^3/3  dt^2/2
            dt^2/2  dt), (* process noise *)
     ((1 dt
       0  1), (* system dynamics *)
     (dt^2/2
        dt), (* system response *)
     col[{g}], (* external force *)
     (1 0), (* observation partials *)
     col[{x0 + v0 t + g t^2/2}] + gen[Z] (* observation fake *)},
   t ↦ x0 + v0 t + g t^2/2, (* true position *)
   t ↦ v0 + g t (* true speed *), 5]]]
```

Out[162]=









% position resids < σ (E >= 68)	67.5694
position resids >= σ	934.
position resids < σ	1946.
average position resid	23.3763
theoretical std dev	83.2249
std dev of position resids	176.477

% speed resids < σ (E >= 68)	71
speed resids >= σ	8
speed resids < σ	2(
average speed resid	-11
theoretical std dev	2.5
std dev of speed resids	19

Part 1: Time-Dependent Observation Noise (TDZ)

KalmanTDZ

TODO: Limitation: scalar observations only.

In[163]:=
```
(* make a foldable kalman filter that takes Zeta as an extra input *)
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {Z_, Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K},
   x2 = Φ.x + Γ.u;
   P2 = Ξ + Φ.P.Φ^T;
   D = Z + A.P2.A^T;
   K = P2.A^T.inv[D];
   (*Print[<|"D"→D,"K"→K,"x"→x,
      "Z"→Z,"P"→P,"xn"→x2+K.(z-A.x2),"Pn"→P2-K.D.K^T|>];*)
   {x2 + K.(z - A.x2), P2 - K.D.K^T}];
(* demonstrate that the filter degenerates to the earlier form given a time-
 independent model *)
m /@
 Chop[
  With[{
    Ξ = zero[4], Z = id[1],
    Φ = id[4], Γ = zero[4, 1], u = zero[1]},
   Fold[
    kalmanTDZ,
    {col[{0, 0, 0, 0}],
     id[4] * 1000.0},
    {Z, Ξ, Φ, Γ, u} ⊕ # & /@ testCase
   ]]]
```

Out[165]=
$$\left\{\begin{pmatrix} -2.97423 \\ 7.2624 \\ -4.21051 \\ -4.45378 \end{pmatrix}, \begin{pmatrix} 0.485458 & 0 & -0.142778 & 0 \\ 0 & 0.901908 & 0 & -0.235882 \\ -0.142778 & 0 & 0.0714031 & 0 \\ 0 & -0.235882 & 0 & 0.0693839 \end{pmatrix}\right\}$$

Three-State Accelerometer (TIS, TDZ)

Time-Independent State, Time-Dependent Observation Noise

sqrt

Sow values if sqrt is attempted on a negative variance.

In[166]:=
```
ClearAll[sqrt];
SetAttributes[sqrt, Listable];
sqrt[z_] := With[{zc = Identity[z]},
   If[zc < 0, Sow[<|"z" → z, "zc" → zc|>, "sqrt"];
    Sqrt[zc], Sqrt[zc]]];
```

stir

Move labels around in a labels packet. TODO: This can probably be removed, but it make labels packet easier if we apply stir to its result. This is a minor annoyance at best. Don't bikeshed too much.

In[169]:=
```
ClearAll[stir];
stir[l_] := (l[[1 ;; len@l ;; 2]]) ⊕ (l[[2 ;; len@l ;; 2]])
```

labelsPacket2

Can handle any number of states.

In[171]:=
```
ClearAll[labelsPacket2, labelsPacket2];
labelsPacket2[_, {}] := {};
labelsPacket2[
   j : {indNym_String, indUnits_String}, {{nym_String, units_String}, more___}] :=
  Module[{brk, fst},
   brk[q_String] := " / [" <> q <> "]";
   fst[q_String] := nym <> q <> brk@units;
   With[{ind = indNym <> brk@indUnits,
     res = " Residual", resvs = " Residuals vs. ",
     trtvs = " Truth and Estimates vs. "},
    {{{fst[res], ""}, {ind, nym <> resvs <> indNym}},
     {{fst[""], ""}, {ind, nym <> trtvs <> indNym}}} ⊕
     labelsPacket2[j, {more}]
   ]];
```

ExperimentTDZ

TODO: Poorly named; this is a full replacement for `experiment`.

In[174]:=

```
ClearAll[experimentTDZ];
experimentTDZ[startTime_, endTime_, timeIncrement_,
    aPrioriState_, aPrioriCovariance_, fakeFn_,
    nStates_Integer : 2, truthFns_List : {Identity, Identity},
    labelsPacket_List : stir@labelsPacket2[
       {"Independent Variable", "?dim"}, {{"State1", "?dim"}, {"State2", "?dim"}}],
    statsLabels_List : {"State1", "State2"}, iterations_Integer : 1] :=
  Module[{t, timesIter, times, fakes, estss,
    trueStates, estimatedStatess, residualStatess,
    stddevStatess, theoStateVarss, statsStates, styler},
    styler = myStyle /@ # &;
    timesIter = {t, startTime, endTime, timeIncrement};
    times = Table[t, Evaluate@timesIter];
    trueStates = Table[truthFns[[n]][t], {n, nStates}, Evaluate@timesIter];
    fakes[] := Table[fakeFn[timeIncrement, t], Evaluate@timesIter];
    estss = Table[
      FoldList[kalmanTDZ, {aPrioriState, aPrioriCovariance}, fakes[]],
      iterations];
    (* starts at 2;; to skip the a-priori *)
    estimatedStatess = Table[estss[[ ;; , 2 ;;, 1, n, 1]], {n, nStates}];
    residualStatess = Table[
      Identity[trueStates[[n]] - # & /@ estimatedStatess[[n]]],
      {n, nStates}];
    stddevStatess = Table[sqrt@estss[[ ( ;; ), (2 ;;), 2, n, n]], {n, nStates}];
    theoStateVarss = Table[Last[estss[[1]]][[2, n, n]], {n, nStates}];
    statsStates = Table[
      stats[statsLabels[[n]],
       f1@residualStatess[[n]],
       f1@stddevStatess[[n]],
       theoStateVarss[[n]]], {n, nStates}];
    Grid@{
      Table[
       ListLinePlot[(withTimes[times, #] & /@ residualStatess[[n]]) ⊕
          (withTimes[times, #] & /@ stddevStatess[[n]]) ⊕
          (withTimes[times, #] & /@ -stddevStatess[[n]]),
        Frame → True, FrameLabel → styler /@ labelsPacket[[n]],
        GridLines → All, ImageSize → Medium], {n, nStates}],
      Table[
       ListLinePlot[{withTimes[times, trueStates[[n]]]} ⊕
          (withTimes[times, #] & /@ estimatedStatess[[n]]),
        Frame → True, FrameLabel → styler /@ labelsPacket[[n + nStates]],
        GridLines → All, ImageSize → Medium], {n, nStates}],
      statsStates}];
```

ExperimentTDZ with Three-State Accelerometer

With $P \leftarrow L.P.L^{\intercal} + K.Z.K^{\intercal}$

In[176]:=

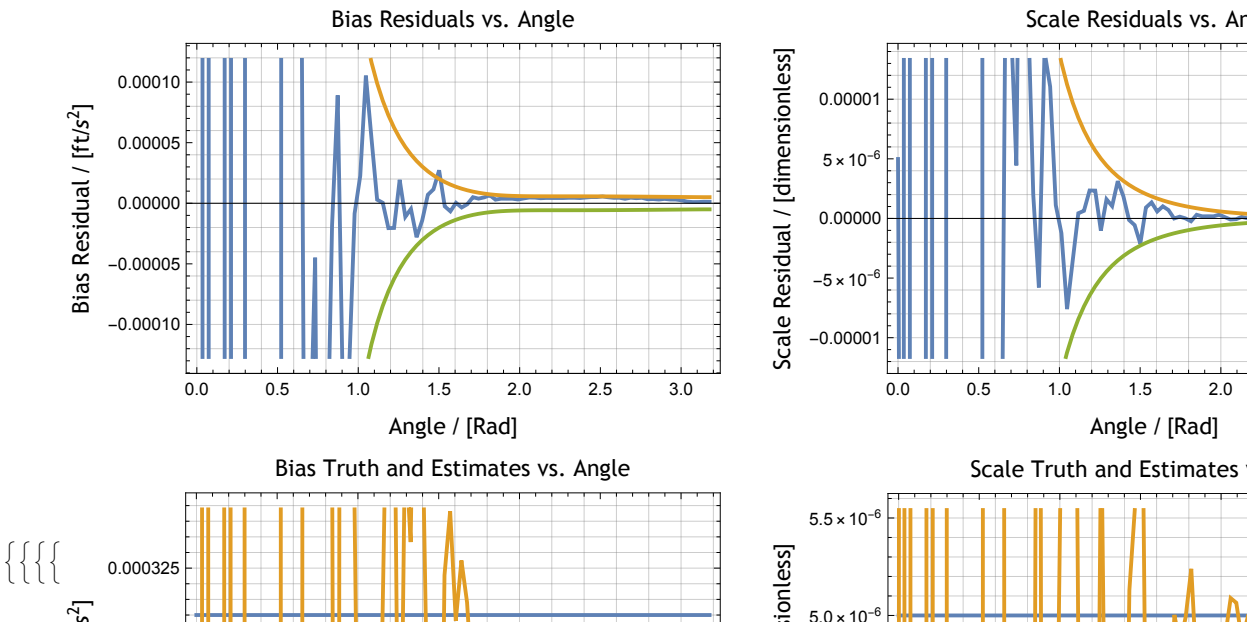
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {z_, A_, z_}] :=
  Module[{D, K, L, P1, P3, P4},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = (id[len[P]] - K.A);
   P1 = L.P;
   P3 = L.P.L^T + K.z.K^T;
   P4 = P - K.D.K^T;
   (*Print[<|"P3"→P3|>];*)
   Sow[Sqrt[(P1)[[1, 1]]], "std1"];
   Sow[Sqrt[(P1)[[2, 2]]], "std2"];
   Sow[Sqrt[(P1)[[3, 3]]], "std3"];
   {x + K.(z - A.x), P3(*P2-K.D.K^T*)}];
```

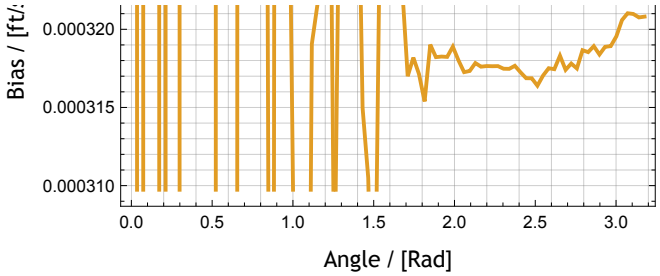
Error in accelerometer is bias + scale * g cos(θ) + drift * (g cos(θ))^2. Three states are bias, scale, and drift. Indpendent variable is θ.

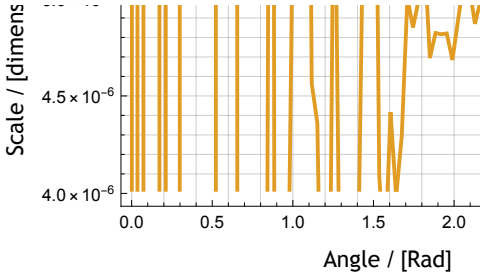
In[178]:=

```
With[{σx = 0, σθ = 1. × 10.^-6, σθtheoretical = 1 × 10^-6,
  g = 32.2, δθ = 2. °(*1°/60*), θ0 = 0., θ1 = 180. °},
 With[{bias0 = 10. × 10^-6 Abs[g], scale0 = 5. × 10^-6, drift0 = 1. × 10^-6 / Abs[g]},
  With[{groundTruth = {bias0, scale0, drift0},
    aPrioriState = zero[3, 1], aPrioriCovariance = 99 999 999 999 id[3]},
   Module[{z, zees, partials}, z[θ_] := col[{ σθtheoretical^2 g^2 (Sin[θ])^2}];
    partials[θ_] := ( 1 g Cos[θ] (g Cos[θ])^2 );
    SeedRandom[42];
    {Reap[Reap[Reap[
        experimentTDZ[θ0, 91 δθ, δθ, aPrioriState, aPrioriCovariance,
         {dθ, θ} ↦
          With[{θnoisy = θ + randn[σθ]},
           {z[θnoisy], (* observation noise *)
            partials[θnoisy], (* observation partials *)
            partials[θnoisy].groundTruth + {g Cos[θnoisy] - g Cos[θ]}}],
        3, {θ ↦ bias0, θ ↦ scale0, θ ↦ drift0},
        stir@labelsPacket2[{"Angle", "Rad"},
          {{"Bias", "ft/s^2"}, {"Scale", "dimensionless"}, {"Drift", "s^2/ft"}}],
        {"Bias", "Scale", "Drift"}, 1],
       "std1", ListLinePlot[Flatten[#2], ImageSize → Medium,
         Frame → True, GridLines → All, PlotLabel → #1] &],
      "std2", ListLinePlot[zees = Flatten[#2],
        ImageSize → Medium, Frame → True, GridLines → All, PlotLabel → #1] &],
     "std3", ListLinePlot[Flatten[#2], ImageSize → Medium,
       Frame → True, GridLines → All, PlotLabel → #1] &]}]]]]
```

Out[178]=

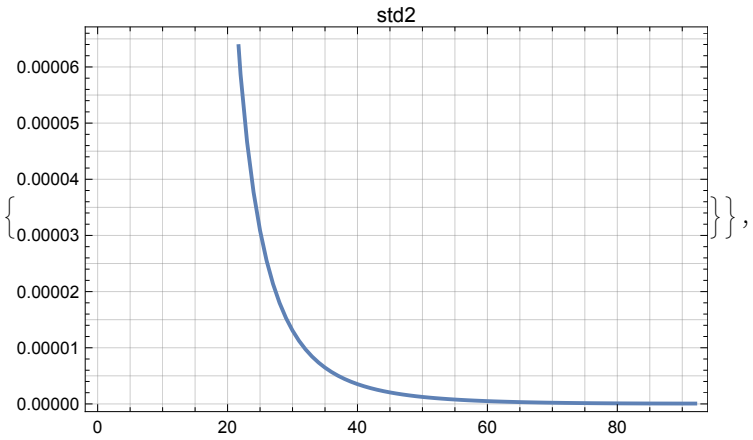


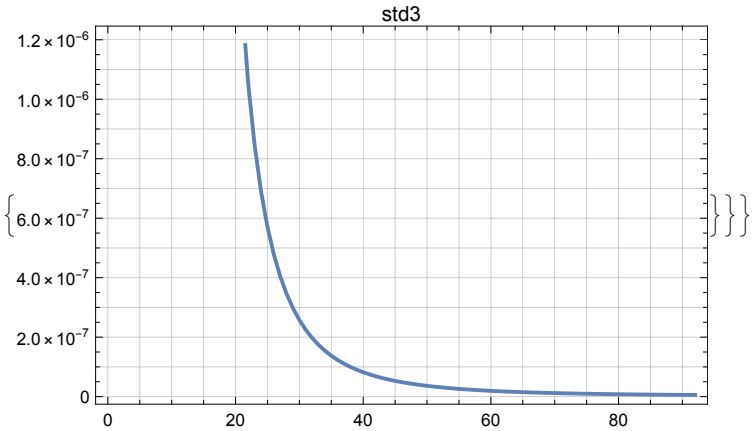




% Bias resids < σ (E >= 68)	83.6957
Bias resids >= σ	4.
Bias resids < σ	77.
average Bias resid	0.00358082
theoretical std dev	5.02418×10^{-6}
std dev of Bias resids	0.0923306

% Scale resids < σ (E >= 68)	
Scale resids >= σ	
Scale resids < σ	
average Scale resid	-
theoretical std dev	6
std dev of Scale resids	





With *P* ← *L.P*

In[179]:=

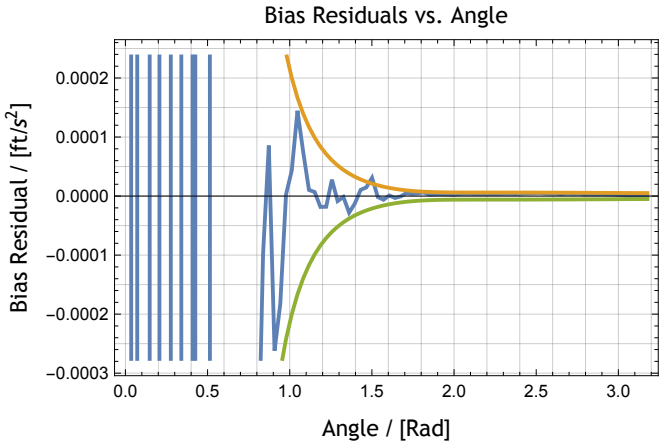
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {z_, A_, z_}] :=
  Module[{D, K, L, P1, P3, P4},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = (id[len[P]] - K.A);
   P1 = L.P;
   P3 = L.P.L^T + K.z.K^T;
   P4 = P - K.D.K^T;
   (*Print[<|"P3"→P3|>];*)
   Sow[Sqrt[(P1)[[1, 1]]], "std1"];
   Sow[Sqrt[(P1)[[2, 2]]], "std2"];
   Sow[Sqrt[(P1)[[3, 3]]], "std3"];
   {x + K.(z - A.x), P1(*P2-K.D.K^T*)}];
```

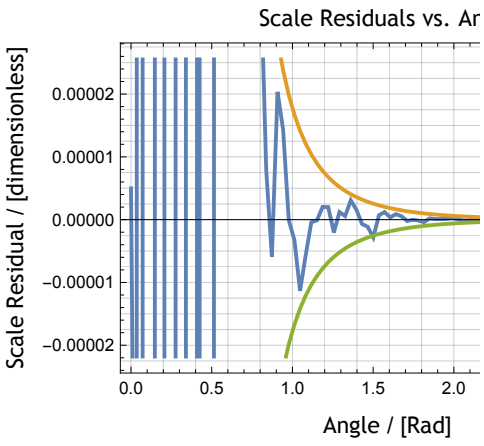
Error in accelerometer is bias + scale * g cos(θ) + drift * (g cos(θ))^2. Three states are bias, scale, and drift. Indpendent variable is θ.

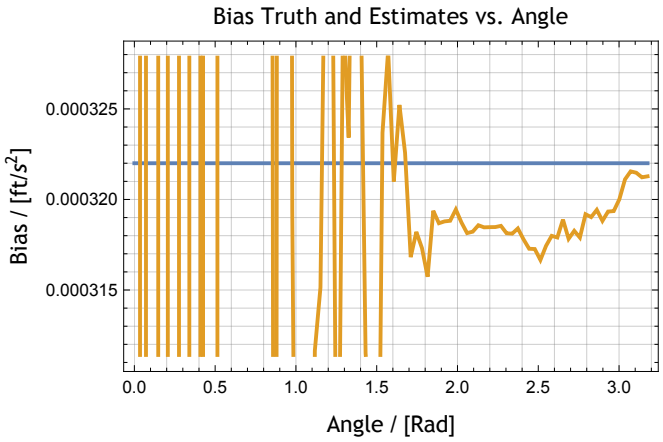
In[181]:=

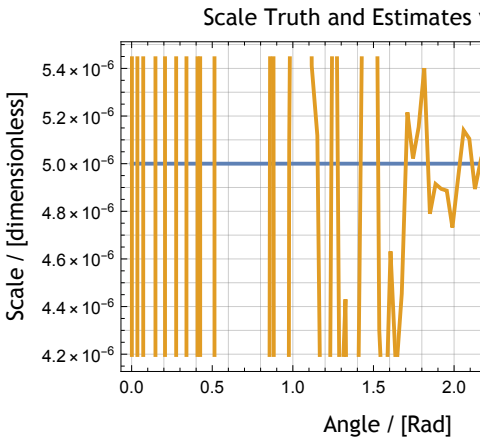
```
With[{σx = 0, σθ = 1. × 10.^-6, σθtheoretical = 1 × 10^-6,
  g = 32.2, δθ = 2. °(*1°/60*), θ0 = 0., θ1 = 180. °},
 With[{bias0 = 10. × 10^-6 Abs[g], scale0 = 5. × 10^-6, drift0 = 1. × 10^-6 / Abs[g]},
  With[{groundTruth = {bias0, scale0, drift0},
    aPrioriState = zero[3, 1], aPrioriCovariance = 99 999 999 999 id[3]},
   Module[{z, zees, partials}, z[θ_] := col[{σθtheoretical^2 g^2 (Sin[θ])^2}];
    partials[θ_] := (1 g Cos[θ] (g Cos[θ])^2);
    SeedRandom[42];
    {Reap[Reap[Reap[
        experimentTDZ[θ0, 91 δθ, δθ, aPrioriState, aPrioriCovariance,
         {dθ, θ} ↦ With[{θnoisy = θ + randn[σθ]},
           {z[θnoisy], (* observation noise *)
            partials[θnoisy], (* observation partials *)
            partials[θnoisy].groundTruth + {g Cos[θnoisy] - g Cos[θ]}}],
         3, {θ ↦ bias0, θ ↦ scale0, θ ↦ drift0},
         stir@labelsPacket2[{"Angle", "Rad"},
           {{"Bias", "ft/s^2"}, {"Scale", "dimensionless"}, {"Drift", "s^2/ft"}}],
         {"Bias", "Scale", "Drift"}, 1],
        "std1", ListLinePlot[Flatten[#2], ImageSize → Medium,
          Frame → True, GridLines → All, PlotLabel → #1] &],
       "std2", ListLinePlot[zees = Flatten[#2],
         ImageSize → Medium, Frame → True, GridLines → All, PlotLabel → #1] &],
      "std3", ListLinePlot[Flatten[#2], ImageSize → Medium,
        Frame → True, GridLines → All, PlotLabel → #1] &]}]]]]
```

Out[181]=





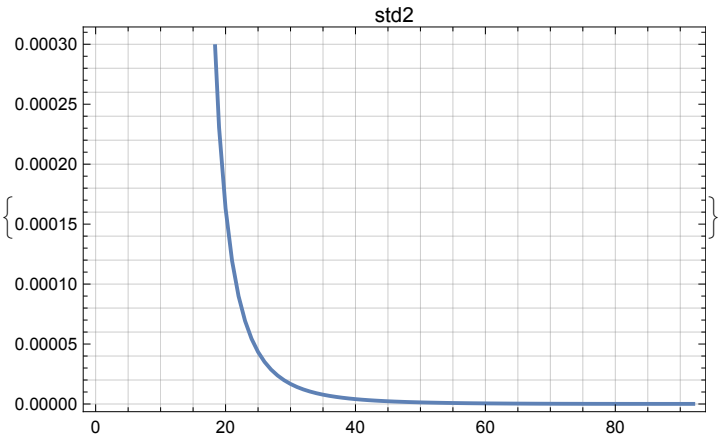




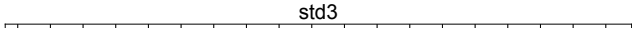
{{{{

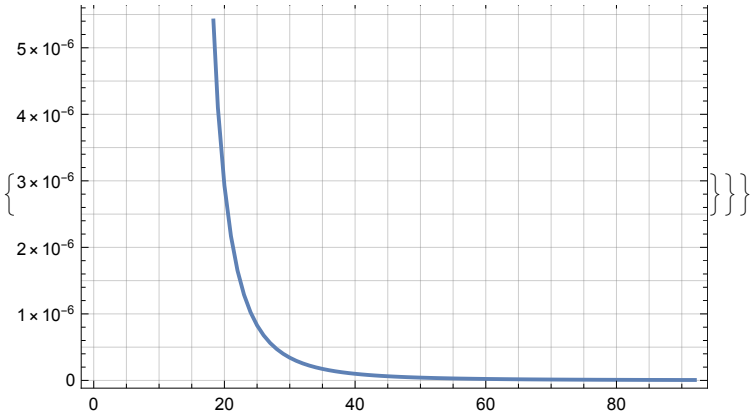
% Bias resids < σ (E >= 68)	86.9565
Bias resids >= σ	7.
Bias resids < σ	80.
average Bias resid	0.0113483
theoretical std dev	5.08884×10^{-6}
std dev of Bias resids	0.114917

% Scale resids < σ (E >= 68)	
Scale resids >= σ	
Scale resids < σ	
average Scale resid	-
theoretical std dev	9
std dev of Scale resids	



}}},





With $P \leftarrow P - K.D.K^{\intercal}$

In[182]:=

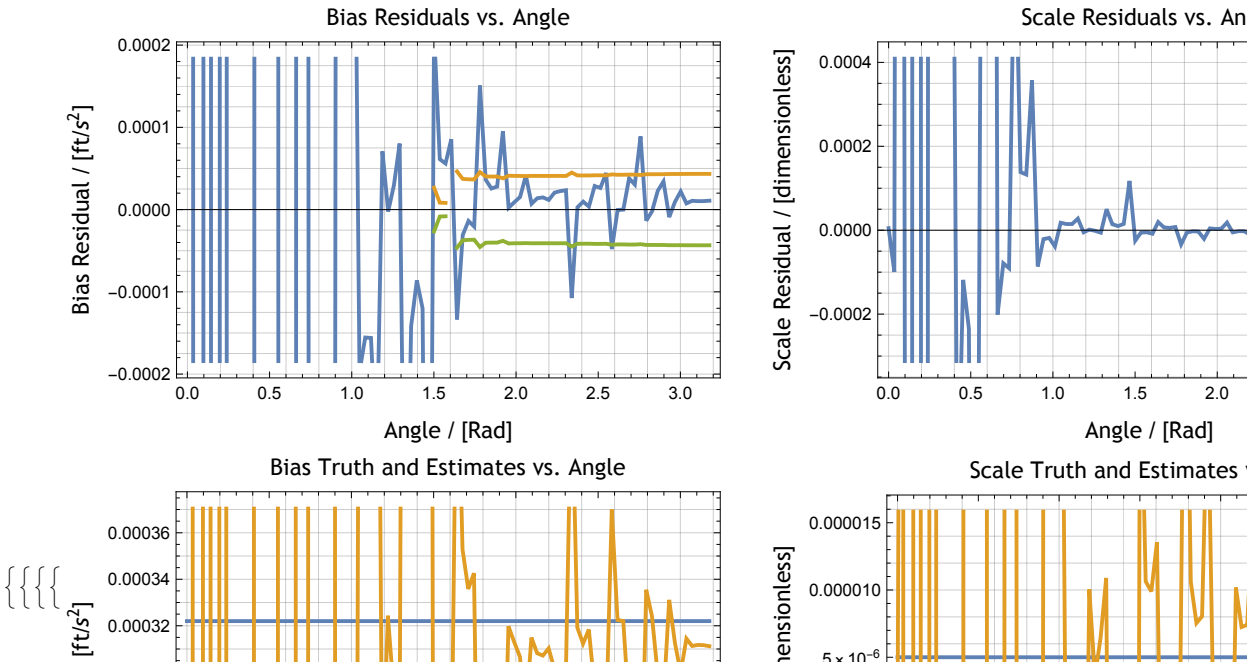
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {z_, A_, z_}] :=
  Module[{D, K, L, P1, P3, P4},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = (id[len[P]] - K.A);
   P1 = L.P;
   P3 = L.P.L^T + K.z.K^T;
   P4 = P - K.D.K^T;
   (*Print[<|"P3"→P3|>];*)
   Sow[Sqrt[(P1)[[1, 1]]], "std1"];
   Sow[Sqrt[(P1)[[2, 2]]], "std2"];
   Sow[Sqrt[(P1)[[3, 3]]], "std3"];
   {x + K.(z - A.x), P4(*P2-K.D.K^T*)}];
```

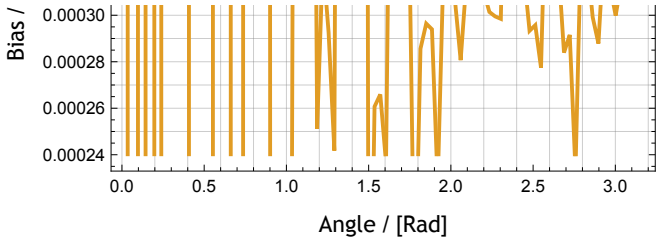
Error in accelerometer is bias + scale * g cos(θ) + drift * (g cos(θ))^2. Three states are bias, scale, and drift. Indpendent variable is θ.

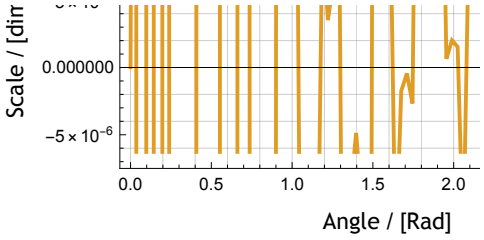
In[184]:=

```
With[{σx = 0, σθ = 1. × 10.^-6, σθtheoretical = 1 × 10^-6,
  g = 32.2, δθ = 2. °(*1°/60*), θ0 = 0., θ1 = 180. °},
 With[{bias0 = 10. × 10^-6 Abs[g], scale0 = 5. × 10^-6, drift0 = 1. × 10^-6 / Abs[g]},
  With[{groundTruth = {bias0, scale0, drift0},
    aPrioriState = zero[3, 1], aPrioriCovariance = 99 999 999 999 id[3]},
   Module[{z, zees, partials}, z[θ_] := col[{σθtheoretical^2 g^2 (Sin[θ])^2}];
    partials[θ_] := (1 g Cos[θ] (g Cos[θ])^2);
    SeedRandom[42];
    {Reap[Reap[Reap[
       experimentTDZ[θ0, 91 δθ, δθ, aPrioriState, aPrioriCovariance,
        {dθ, θ} ↦ With[{θnoisy = θ + randn[σθ]},
          {z[θnoisy], (* observation noise *)
           partials[θnoisy], (* observation partials *)
           partials[θnoisy].groundTruth + {g Cos[θnoisy] - g Cos[θ]}}],
        3, {θ ↦ bias0, θ ↦ scale0, θ ↦ drift0},
        stir@labelsPacket2[{"Angle", "Rad"},
          {{"Bias", "ft/s^2"}, {"Scale", "dimensionless"}, {"Drift", "s^2/ft"}}],
        {"Bias", "Scale", "Drift"}, 1],
       "std1", ListLinePlot[Flatten[#2], ImageSize → Medium,
         Frame → True, GridLines → All, PlotLabel → #1] &],
      "std2", ListLinePlot[zees = Flatten[#2],
       ImageSize → Medium, Frame → True, GridLines → All, PlotLabel → #1] &],
     "std3", ListLinePlot[Flatten[#2], ImageSize → Medium,
      Frame → True, GridLines → All, PlotLabel → #1] &]}]]]]
```

Out[184]=

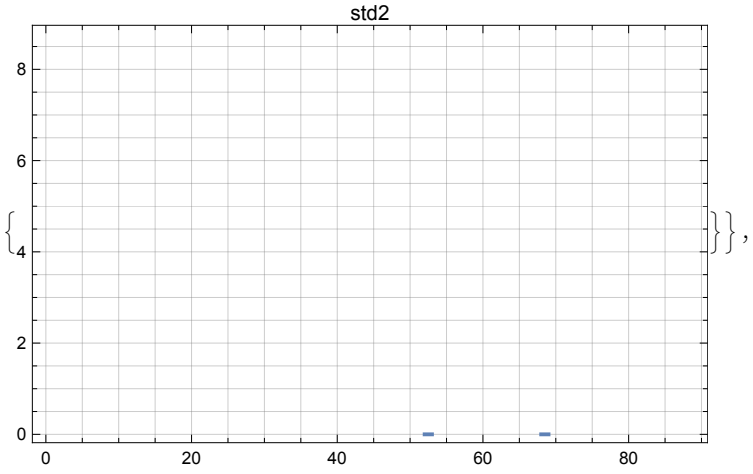


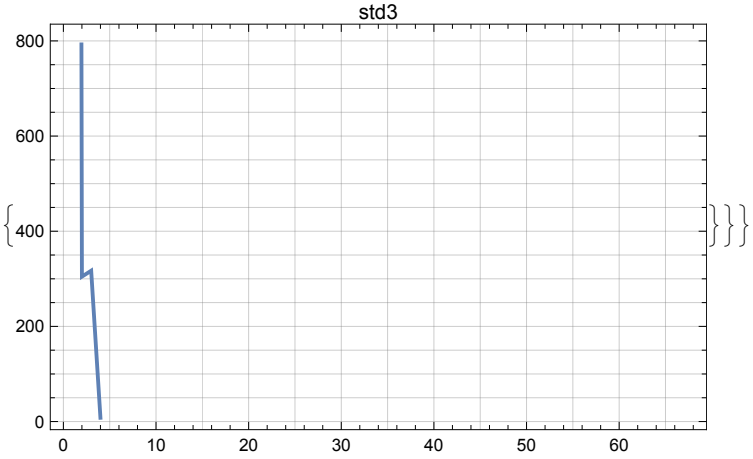




% Bias resids < σ (E >= 68)	48.913
Bias resids >= σ	13.
Bias resids < σ	45.
average Bias resid	-0.00498231
theoretical std dev	0.0000434082
std dev of Bias resids	0.0729717

% Scale resids < σ (E >= 68)	
Scale resids >= σ	
Scale resids < σ	
average Scale resid	
theoretical std dev	0.
std dev of Scale resids	





Comparison with Zarchan & Musoff's Three-State

Accelerometer

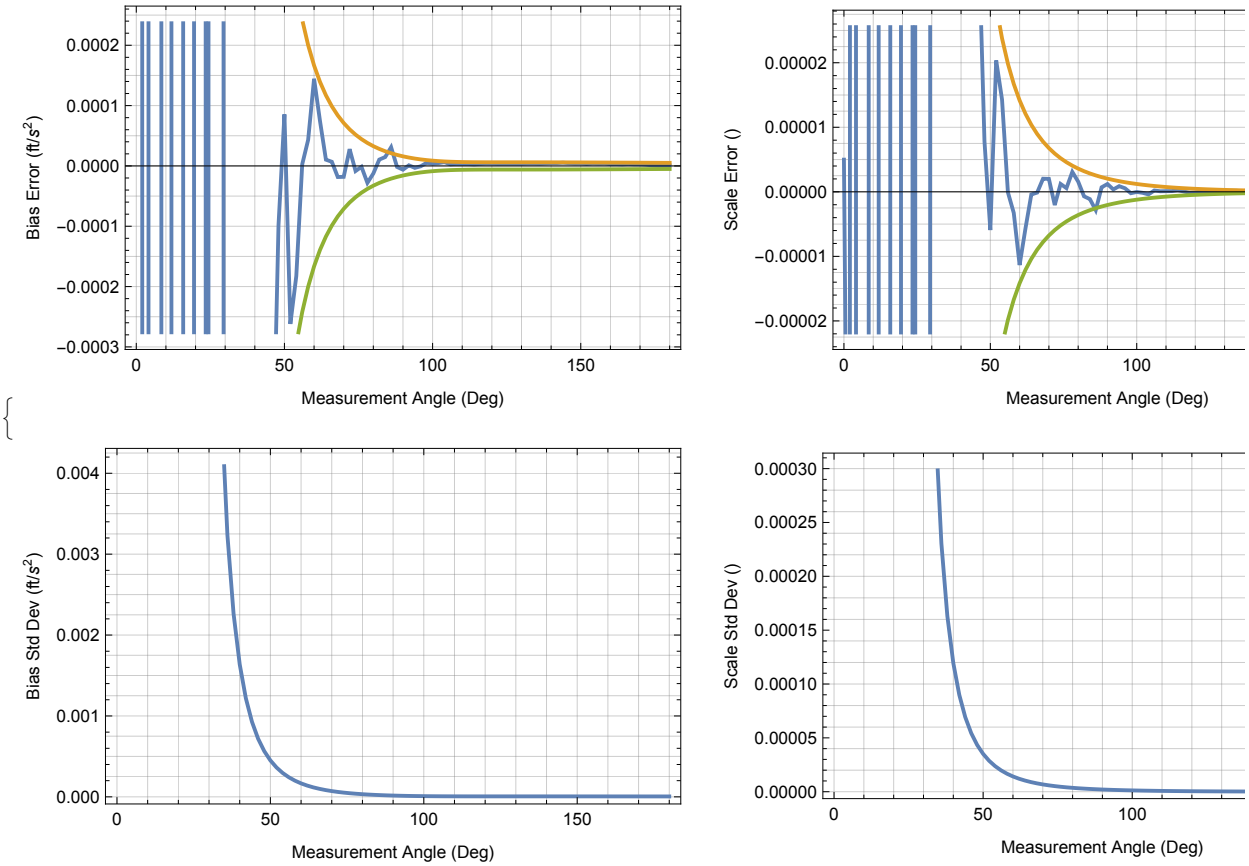
In[185]:=

```
Module[{
  order = 3,
  bias = .00001 * 32.2,
  sf = .000005,
  xk = .000001 / 32.2,
  sigth = .000001,
  g = 32.2,
  biash = 0,
  sfh = 0,
  xkh = 0,
  signoise = .000001,
  s = 0,
  q = 0 IdentityMatrix[3],
  phi = IdentityMatrix[3],
  idnp = IdentityMatrix[3],
  p = 99 999 999 999 IdentityMatrix[3],
  ppri,
  count = 0,
  thetdeg, thet, thetnoise, thets, hmat, r, m, k, z, res, sp11, sp22,
  sp33, sp11p, sp22p, sp33p, biaserr, sferr, xkerr, actnoise, sigr, sigrp,
  arraythetdeg, arraybias, arraymhmatt, arrayk11, arrayz, arrayres,
  arraybiash, arraysf, arraysfh, arrayxk, arrayxkh, arraybiaserr,
  arraysp11, arraysp11p, arraysferr, arraysp22, arraysp22p, arrayxkerr,
  arraysp33, arraysp33p, arrayactnoise, arraysigr, arraysigrp},
 SeedRandom[42];
 {arraythetdeg, arraybias, arraymhmatt, arrayk11, arrayz, arrayres,
   arraybiash, arraysf, arraysfh, arrayxk, arrayxkh, arraybiaserr,
   arraysp11, arraysp11p, arraysferr, arraysp22, arraysp22p, arrayxkerr,
   arraysp33, arraysp33p, arrayactnoise, arraysigr, arraysigrp} =
  Reap[For[
     thetdeg = 0, thetdeg ≤ 180., thetdeg += 2.,
     Sow[thetdeg, "thetdeg"];
     Sow[bias, "bias"];
     thet = thetdeg * (1.0 °) (*/57.3*);
     thetnoise = randn@signoise;
     thets = thet + thetnoise;
     hmat = {{1, g Cos[thets], (g Cos[thets])^2}};
     r = (g Sin[thets] sigth)^2 IdentityMatrix[1];
     m = phi.p.(phi^T) + q;
     k = m.(hmat^T).Inverse[hmat.m.(hmat^T) + r];
```

```
    Sow[(m.(hmat^T))[[1, 1]], "hmatt"];
    Sow[k[[1, 1]], "k11"];
    ppri = p;
    p = (idnp - k.hmat).m;
    (*Print[<|"P"→p|>];*)
    Sow[
     z = bias + sf g Cos[thets] + xk (g Cos[thets])^2 - g Cos[thet] + g Cos[thets], "z"];
    Sow[res = z - biash - sfh g Cos[thets] - xkh (g Cos[thets])^2, "res"];
    Sow[biash = biash + k[[1, 1]] res, "biash"];
    Sow[sf, "sf"];
    Sow[sfh = sfh + k[[2, 1]] res, "sfh"];
    Sow[xk, "xk"];
    Sow[xkh = xkh + k[[3, 1]] res, "xkh"];
    Sow[biaserr = bias - biash, "biaserr"];
    Sow[sp11 = Sqrt[p[[1, 1]]], "sp11"];
    Sow[sp11p = -sp11, "sp11p"];
    Sow[sferr = sf - sfh, "sferr"];
    Sow[sp22 = Sqrt[p[[2, 2]]], "sp22"];
    Sow[sp22p = -sp22, "sp22p"];
    Sow[xkerr = xk - xkh, "xkerr"];
    Sow[sp33 = Sqrt[p[[3, 3]]], "sp33"];
    Sow[sp33p = -sp33, "sp33p"];
    Sow[actnoise = g Cos[thets] - g Cos[thet], "actnoise"];
    Sow[sigr = Sqrt[r[[1, 1]]], "sigr"];
    Sow[sigrp = -sigr, "sigrp"];
    count = count + 1;
   ]][[2]];
{Grid[
  {{ListLinePlot[{{arraythetdeg, arraybiaserr}^T,
     {arraythetdeg, arraysp11}^T, {arraythetdeg, arraysp11p}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel →
     {{"Bias Error (ft/s^2)", ""}, {"Measurement Angle (Deg)", ""}}],
   ListLinePlot[{{arraythetdeg, arraysferr}^T,
     {arraythetdeg, arraysp22}^T, {arraythetdeg, arraysp22p}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel → {{"Scale Error ()", ""}, {"Measurement Angle (Deg)", ""}}],
   ListLinePlot[{{arraythetdeg, arrayxkerr}^T,
     {arraythetdeg, arraysp33}^T, {arraythetdeg, arraysp33p}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel →
     {{"Drift Error (s^2/ft)", ""}, {"Measurement Angle (Deg)", ""}}]},
```

```
  {ListLinePlot[{{arraythetdeg, arraysp11}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel →
     {{"Bias Std Dev (ft/s^2)", ""}, {"Measurement Angle (Deg)", ""}}],
   ListLinePlot[{{arraythetdeg, arraysp22}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel → {{"Scale Std Dev ()", ""}, {"Measurement Angle (Deg)", ""}}],
   ListLinePlot[{{arraythetdeg, arraysp33}^T},
    ImageSize → Medium, GridLines → All, Frame → True,
    FrameLabel →
     {{"Drift Std Dev (s^2/ft)", ""}, {"Measurement Angle (Deg)", ""}}]
  }}],
arraysp22}]
```

Out[185]=



{5.05559, 0. + 3.00108 i, 0.588137, 0. + 0.0330254 i, 0. + 0.010045 i, 0.00594021,
0.00188567, 0.000929563, 0.000537759, 0.000341305, 0.000230592, 0.000163069,
0.000119432, 0.0000899436, 0.0000692955, 0.0000544077, 0.0000434091, 0.0000351136,
0.0000287444, 0.0000237773, 0.0000198506, 0.0000167087, 0.0000141673,
0.0000120918, 0.0000103818, 8.96171×10^{-6}, 7.77381×10^{-6}, 6.77353×10^{-6},
5.9261×10^{-6}, 5.20414×10^{-6}, 4.58587×10^{-6}, 4.05386×10^{-6}, 3.59403×10^{-6},
3.19493×10^{-6}, 2.84721×10^{-6}, 2.54316×10^{-6}, 2.27641×10^{-6}, 2.04164×10^{-6},
1.8344×10^{-6}, 1.65096×10^{-6}, 1.48817×10^{-6}, 1.34336×10^{-6}, 1.21425×10^{-6},
1.0989×10^{-6}, 9.95628×10^{-7}, 9.03015×10^{-7}, 8.19816×10^{-7}, 7.44959×10^{-7},
6.77515×10^{-7}, 6.16673×10^{-7}, 5.61725×10^{-7}, 5.12053×10^{-7}, 4.67117×10^{-7},
4.26439×10^{-7}, 3.89601×10^{-7}, 3.56234×10^{-7}, 3.26012×10^{-7}, 2.98647×10^{-7},
2.73886×10^{-7}, 2.515×10^{-7}, 2.31291×10^{-7}, 2.13078×10^{-7}, 1.96703×10^{-7},
1.82021×10^{-7}, 1.68902×10^{-7}, 1.57228×10^{-7}, 1.4689×10^{-7}, 1.37784×10^{-7},
1.29815×10^{-7}, 1.22888×10^{-7}, 1.16915×10^{-7}, 1.11806×10^{-7}, 1.07476×10^{-7},
1.03841×10^{-7}, 1.0082×10^{-7}, 9.83365×10^{-8}, 9.6318×10^{-8}, 9.46973×10^{-8},
9.34134×10^{-8}, 9.2412×10^{-8}, 9.16466×10^{-8}, 9.10857×10^{-8}, 9.08009×10^{-8}}}

ExperimentTDZ Estimating the Mean

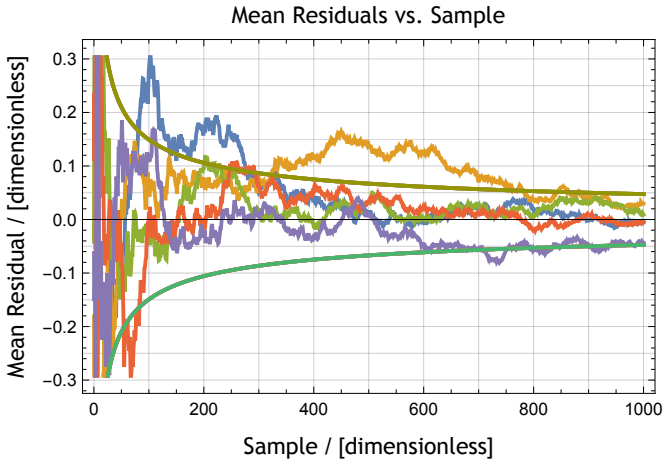
Reminder of new experiment signature; closes over **kalmanTDZ** , which takes **{z, A, z}**

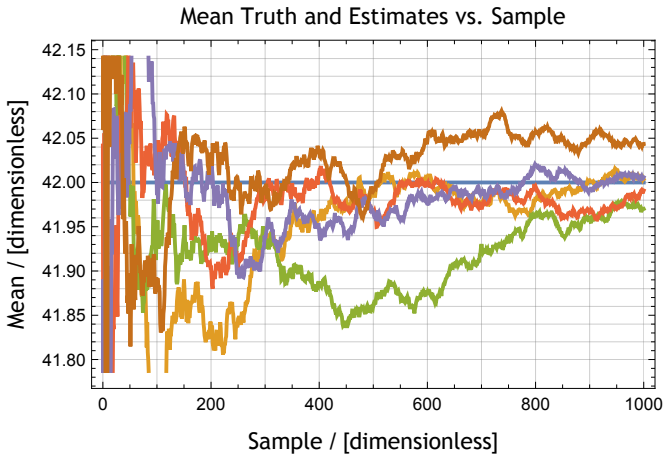
```
experimentTDZ[t₀, t₁, δt,
x̃, P̃, {δt, t} ↦ {z, A, z},
nStates : 2, truthFns : {Ident, Ident},
labelsPacket : "a basic default; don't forget to stir",
statsLabels_List : {"state1", "state2"}, iterations : 1]
```

In[186]:=

```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {z_, A_, z_}] :=
  Module[{D, K, L, P1, P3, P4},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = (id[len[P]] - K.A);
   {x + K.(z - A.x), L.P}];
With[{σz = 1.5},
 With[{aPrioriState = zero[1, 1],
   aPrioriCovariance = 1 000 000 000 id[1],
   z = col[{σz^2}]},
  experimentTDZ[0, 1000, 1, aPrioriState, aPrioriCovariance,
   {dt, t} ↦ {
     z,
     id[1],
     col[{RandomVariate[NormalDistribution[42, σz]]}]},
   1,
   {t ↦ 42.},
   stir@labelsPacket2[{"Sample", "dimensionless"},
     {{"Mean", "dimensionless"}}],
   {"Mean"},
   5]]]
```

Out[188]=





% Mean resids < σ (E >= 68)	81.3986
Mean resids >= σ	931.
Mean resids < σ	4074.
average Mean resid	0.019131
theoretical std dev	0.0474105
std dev of Mean resids	0.114848

ExperimentTDZ Estimating the Three-State Falling Object with No Drag

Reminder of new experiment signature; closes over **kalmanTDZ** , which takes **{z, A, z}**

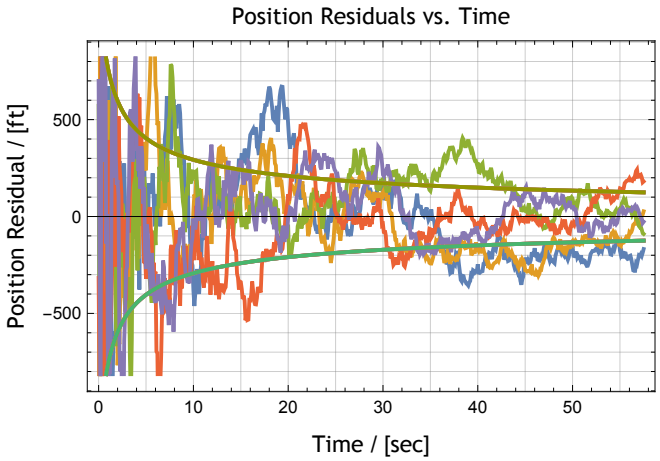
```
experimentTDZ[t0, t1, δt,
x̃, P̃, {δt, t} ↦ {Z, Ξ, Φ, Γ, u, A, z},
nStates : 2, truthFns : {Ident, Ident},
labelsPacket : "a basic default; don't forget to stir",
statsLabels_List : {"state1", "state2"}, iterations : 1]
```

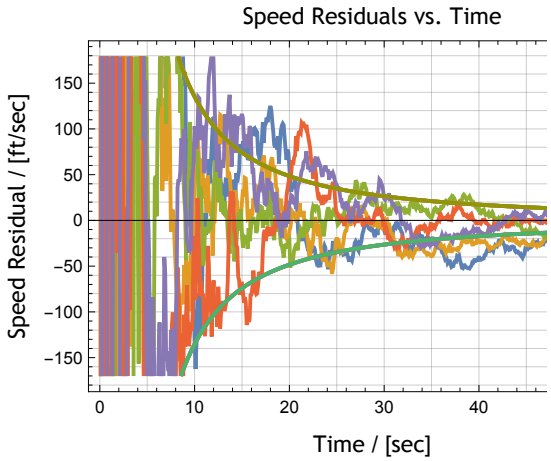
In[189]:=

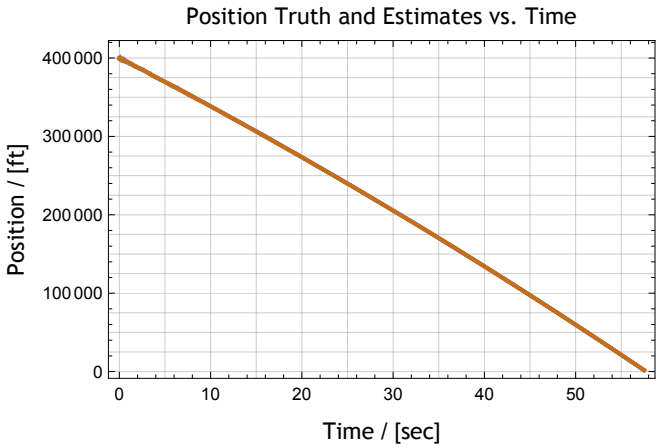
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {Z_, Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L, PT1, PT2, PT3},
```

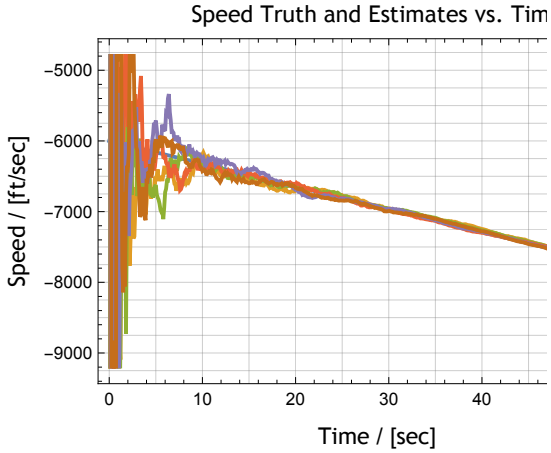
```
    x2 = Φ.x + Γ.u;
    P2 = Ξ + Φ.P.Φ^T;
    D = Z + A.P2.A^T;
    K = P2.A^T.inv[D];
    L = id[len[P]] - K.A;
    PT1 = L.P2; PT2 = (L.P2.L^T + K.Z.K^T); PT3 = (P2 - K.D.K^T);
    {x2 + K.(z - A.x2), PT3}];
With[{x0 = 400 000., v0 = -6000., a0 = -32.2, σz = 1000., σx = 0, nStates = 3},
 With[{aPrioriState = zero[nStates, 1],
   aPrioriCovariance = 1 000 000 000 000 id[nStates],
   Z = col[{σz^2}]},
  SeedRandom[42];
  experimentTDZ[0, 57.5, 0.1,
   aPrioriState, aPrioriCovariance,
   {dt, t} ↦
    {Z,
     σx^2 {{dt^5/20, dt^4/8, dt^3/6},
           {dt^4/8, dt^3/3, dt^2/2},
           {dt^3/3, dt^2/2, dt}}, (* process noise *)
     {{1, dt, dt^2/2},
      {0, 1,  dt},
      {0, 0,  1}}, (* system dynamics *)
     {{dt^2/2},
      {dt},
      {1}}, (* system response *)
     zero[1], (* external force *)
     (1 0 0), (* observation partials *)
     col[{x0 + v0 t + a0 t^2/2}] + gen[Z]}, (* observation fake *)
   nStates,
   {t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
    t ↦ v0 + a0 t (* true speed *),
    t ↦ a0 (* true acceleration *)},
   stir@labelsPacket2[{"Time", "sec"},
     {{"Position", "ft"}, {"Speed", "ft/sec"}, {"Acceleration", "ft/sec^2"}}],
   {"Position", "Speed", "Acceleration"},
   5(* iterations *)]]]
```

Out[191]=









% Position resids < σ (E >= 68)	64.3056
Position resids >= σ	1028.
Position resids < σ	1852.
average Position resid	5.99396
theoretical std dev	124.568
std dev of Position resids	248.361

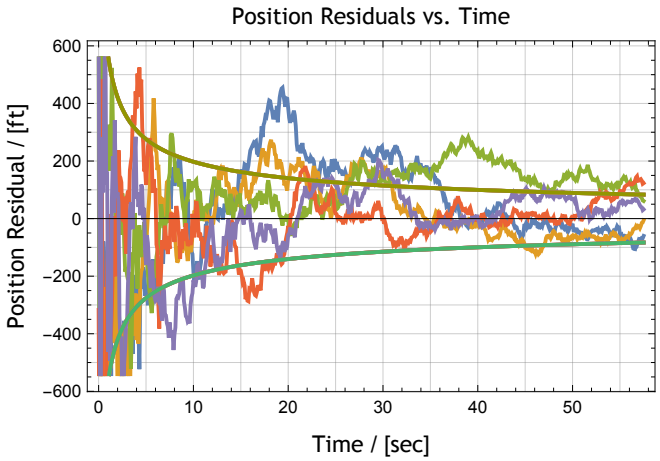
% Speed resids < σ (E >= 68)	67
Speed resids >= σ	9
Speed resids < σ	19
average Speed resid	-43
theoretical std dev	10
std dev of Speed resids	239

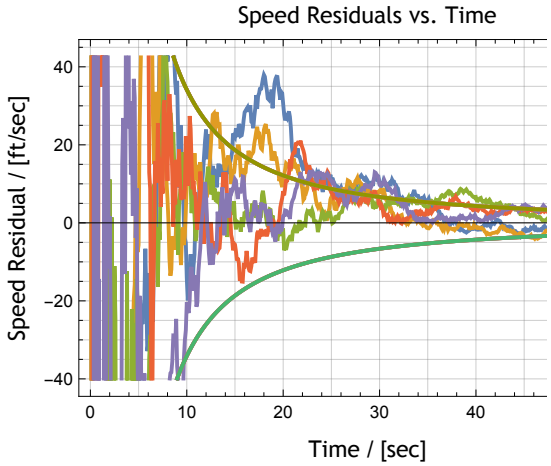
ExperimentTDZ Estimating the Two-State Falling Object with No Drag

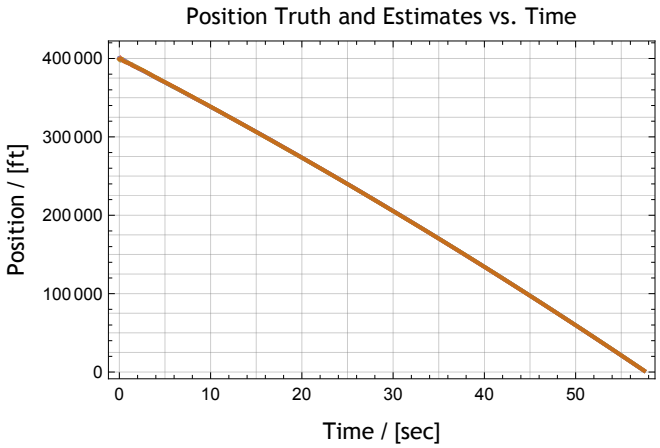
In[192]:=

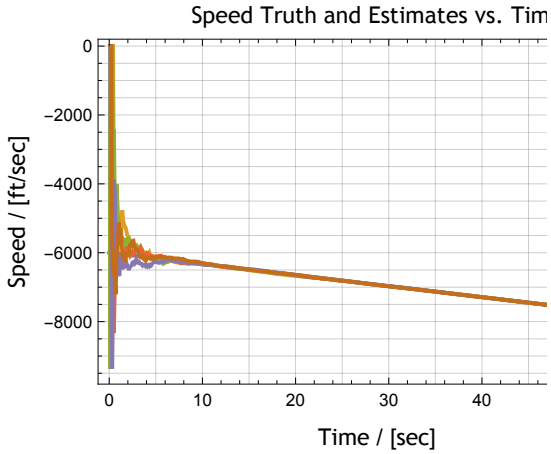
```
With[{x0 = 400 000., v0 = -6000., a0 = -32.2, σz = 1000., σx = 0, nStates = 2},
 With[{aPrioriState = zero[nStates, 1],
   aPrioriCovariance = 1 000 000 000 000 id[nStates],
   z = col[{σz^2}]},
  SeedRandom[42];
  experimentTDZ[0, 57.5, 0.1,
   aPrioriState, aPrioriCovariance,
   {dt, t} ↦
    {z,
     σx^2 ((dt^3/3  dt^2/2
            dt^2/2  dt), (* process noise *)
     (1 dt
      0 1), (* system dynamics *)
     (dt^2/2
      dt), (* system response *)
     col[{a0}], (* external force *)
     (1 0), (* observation partials *)
     col[{x0 + v0 t + a0 t^2/2}] + gen[z] (* observation fake *)},
   (* observation fake *)
   nStates,
   {t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
    t ↦ v0 + a0 t (* true speed *)},
   stir@
    labelsPacket2[{"Time", "sec"}, {{"Position", "ft"}, {"Speed", "ft/sec"}}],
   {"Position", "Speed"},
   5(* iterations *)]]]
```

Out[192]=









% Position resids < σ (E >= 68)	67.5694
Position resids >= σ	934.
Position resids < σ	1946.
average Position resid	23.3763
theoretical std dev	83.2249
std dev of Position resids	176.477

% Speed resids < σ (E >= 68)	71
Speed resids >= σ	8
Speed resids < σ	2(
average Speed resid	-11
theoretical std dev	2.5
std dev of Speed resids	198

Part 5: EKF

Custom Integrator

eulerStep, rkStep

Quote from old paper:

> Write differential equations in first-order, state-space form. In particular, Lagrange's equations (or Newton's) look like this

$$\frac{d}{dt}\begin{bmatrix} q(t) \\ \dot{q}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ F(q, \dot{q}, t) & G(q, \dot{q}, t) \end{bmatrix}\begin{bmatrix} q(t) \\ \dot{q}(t) \end{bmatrix}$$

We can write this in shorthand; let $\Psi(t) = [q(t), \dot{q}(t)]^T$, $\Phi(q, \dot{q}, t) = [[0, 1], [F(q, \dot{q}, t), G(q, \dot{q}, t)]]$, then

$\frac{d}{dt}\Psi(t) = \Phi(q, \dot{q}, t)\,\Psi(t)$

For instance, consider a spring with constant $-k$ and a damper with constant $-v$ on a mass m in one dimension. Write the equations as

$$\begin{bmatrix} \dot{q}(t) \\ \ddot{q}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -v/m \end{bmatrix}\begin{bmatrix} q(t) \\ \dot{q}(t) \end{bmatrix}$$

In general, as above, the derivative matrix is a function of the q-vector and of the time, but this one is constant

The following is a completely general, fourth-order Runge-Kutta integrator that takes four arguments:

- ψ: a state vector at any time
- ϕ: a function of ψ and t that returns a state-space matrix as above
- t: the current time
- dt: the interval of time to the next time

To produce a sequence of integrands, just *FoldList* this over a *Range* of times, as shown immediately below:

The following only works on 'linear' models where dψdt = F.ψ

In[193]:=

```
ClearAll[rk2step, rk4step];
rk2step[ψ_, ϕ_, t_, dt_] :=
  (With[{dψ1 = dt ϕ[ψ, t].ψ},
    With[{dψ2 = dt With[{ψ1 = ψ + .5 dψ1}, ϕ[ψ1, t + .5 dt].ψ1]},
     ψ + (dψ1 + dψ2) / 2.]]);
rk4step[ψ_, ϕ_, t_, dt_] :=
  (With[{dψ1 = dt ϕ[ψ, t].ψ},
    With[{dψ2 = dt With[{ψ1 = ψ + .5 dψ1}, ϕ[ψ1, t + .5 dt].ψ1]},
     With[{dψ3 = dt With[{ψ2 = ψ + .5 dψ2}, ϕ[ψ2, t + .5 dt].ψ2]},
      With[{dψ4 = dt With[{ψ3 = ψ + dψ3}, ϕ[ψ3, t + dt].ψ3]},
       ψ + (dψ1 + 2. dψ2 + 2. dψ3 + dψ4) / 6.]]]]);
```

The following works on arbitrary models where dψdt = Dψ depends on ψ and t

In[196]:=

```
ClearAll[eulerStepA];
eulerStepA[ψt_, Dψ_(*[ψ,t]*), t_, dt_] :=
  ψt + dt Dψ[ψt, t];
```

In[198]:=
```
ClearAll[rk2stepA, rk4stepA];
rk2stepA[ψt_, Dψ_(*[ψ,t]*), t_, dt_] :=
  (With[{dψ1 = dt Dψ[ψt, t]},
    With[{dψ2 = dt Dψ[ψt + .5 dψ1, t + .5 dt]},
     ψt + (dψ1 + dψ2) / 2.]]);
rk4stepA[ψt_, Dψ_(*[ψ,t]*), t_, dt_] :=
  (With[{dψ1 = dt Dψ[ψt, t]},
    With[{dψ2 = dt Dψ[ψt + .5 dψ1, t + .5 dt]},
     With[{dψ3 = dt Dψ[ψt + .5 dψ2, t + .5 dt]},
      With[{dψ4 = dt Dψ[ψt + dψ3, t + dt]},
       ψt + (dψ1 + 2. dψ2 + 2. dψ3 + dψ4) / 6.]]]]);
```

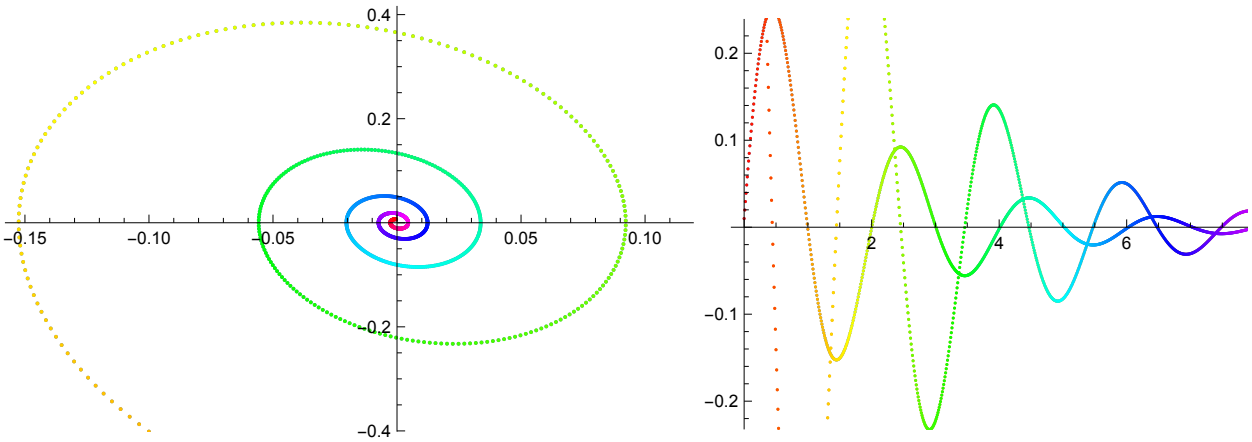
General Second-Order Linear ODE

Demonstration that rk4stepA matches NDSolve:

In[201]:=

```
Module[{dt = 0.01, t0 = 0, tN = 10, times},
 times = Range[t0, tN, dt];
 With[{soln = Module[{m = 1, k = 10, ν = 1, mat}, mat = {{0, 1}, {-k / m, -ν / m}};
     FoldList[
      rk4stepA[#1, {ψ, t} ↦ mat.ψ, #2, dt] &, (* calculate one step *)
      {0, 1}, (* initial state *)
      times]]},
  Grid[{
    {ListPlot[MapIndexed[Style[#1, Hue[#2 / len@soln]] &]@soln, ImageSize → Medium],
     ListPlot[{
       MapIndexed[Style[#1, Hue[#2 / len@times]] &]@{times, d1[pick[1] /@ soln]}^T,
       MapIndexed[Style[#1, Hue[#2 / len@times]] &]@{times, d1[pick[2] /@ soln]}^T},
      ImageSize → Medium]}
   }]]]
```

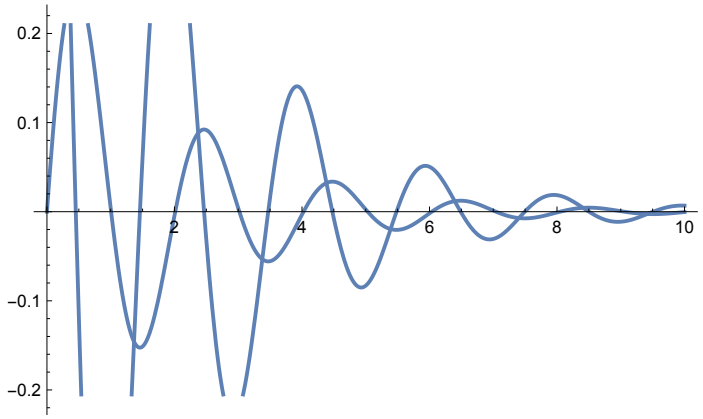
Out[201]=



In[202]:=

```
Module[{dt = 0.01, t0 = 0, tN = 10, times},
 times = Range[t0, tN, dt];
 With[{soln = Module[{m = 1, k = 10, ν = 1},
     NDSolve[{x''[t] == -k x[t] / m - ν x'[t] / m, x[0] == 0, x'[0] == 1}, x, {t, 0, 10}]]},
  Grid[{{Plot[{x[t], x'[t]} /. soln, {t, 0, 10}, ImageSize → Medium]}}]]]
```

Out[202]=

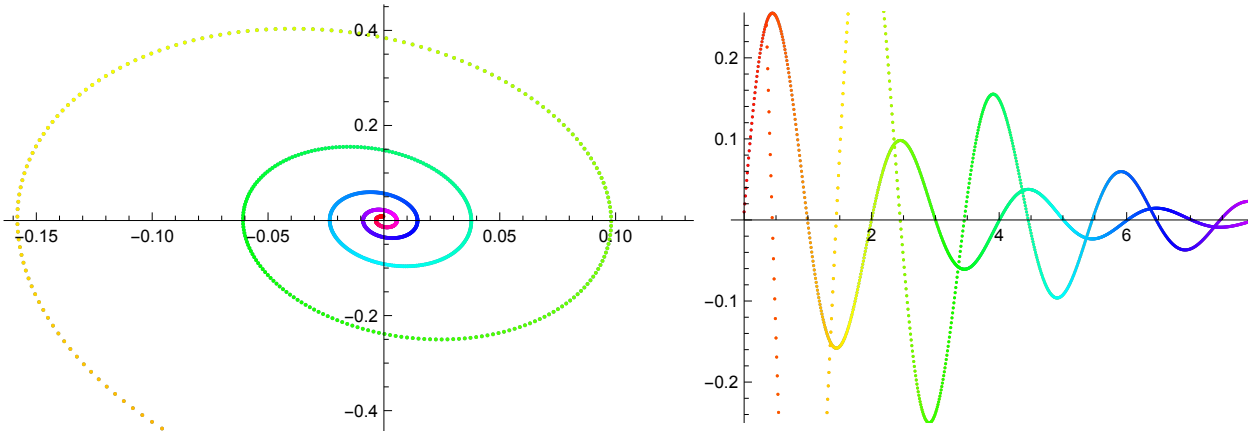


Demonstration that rk2step also matches NDSolve.

In[203]:=

```
Module[{dt = 0.01, t0 = 0, tN = 10, times},
 times = Range[t0, tN, dt];
 With[{soln = Module[{m = 1, k = 10, ν = 1, mat, ϕ},
     mat = {{0, 1}, {-k / m, -ν / m}};
     ϕ = Function[{ψ, t}, mat];
     FoldList[rk2step[#1, ϕ, #2, dt] &, {0, 1}, times]]},
  Grid[{{
     ListPlot[MapIndexed[Style[#1, Hue[#2 / len@soln]] &]@soln, ImageSize → Medium],
     ListPlot[{
       MapIndexed[Style[#1, Hue[#2 / len@times]] &]@{times, d1[pick[1] /@ soln]}^T,
       MapIndexed[Style[#1, Hue[#2 / len@times]] &]@{times, d1[pick[2] /@ soln]}^T},
      ImageSize → Medium]}
  }]]]
```

Out[203]=



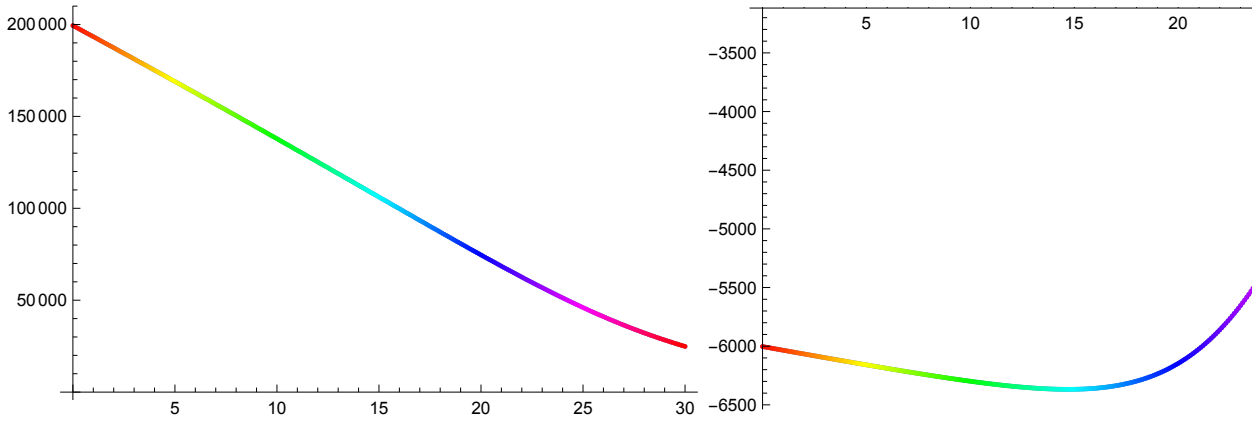
Falling Object with Drag

My version of integrating the falling-object equation of motion with FoldList and rk2stepA, matching Zarchan & Musoff.

In[204]:=

```
With[{g = 32.2},
 With[{x0 = 200 000., v0 = -6000., a0 = -g, t0 = 0., t1 = 30., dt = .1},
  Module[{times = Range[t0, t1, dt], timeStyler},
   timeStyler[xs_] := MapIndexed[Style[#1, Hue[#2 / len@xs]] &];
   With[{soln =
      Module[{ρ, qp, β, drag, Dψ},
       ρ[x_] := 0.0034 Exp[-x / 22 000];
       (* English units/g; Zarchan, P., 1998 *)
       qp[{x_, v_}] := 0.5 ρ[x] v^2;
       β = 500;
       drag[ψ_] := Sow[qp[ψ] g / β];
       Dψ[ψ : {x_, v_}, t_] := {v, a0 + drag[ψ]};
       FoldList[
        eulerStepA[#1, Dψ, #2, dt] &,
        {x0, v0},
        times]]},
    Grid[{
      {ListPlot[
        {timeStyler[times]@{times, d1[pick[1] /@ soln]}^T}, ImageSize → Medium],
       ListPlot[
        {timeStyler[times]@{times, d1[pick[2] /@ soln]}^T}, ImageSize → Medium]}
     }]]]]]
```

Out[204]=

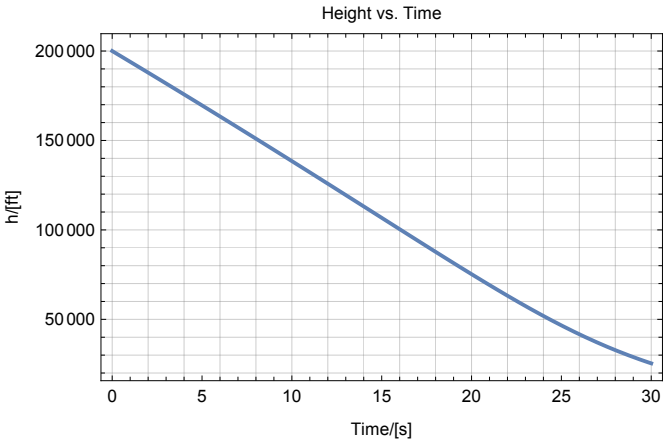


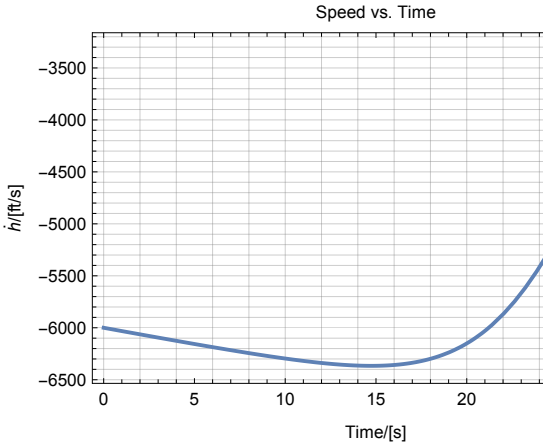
Via NDSolve

In[205]:=

```
With[{g = 32.2, A = 0.0034, k = 22 000., beta = 500.},
 With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30.},
  With[{soln = NDSolve[{h''[t] == -g + A g (h'[t])^2 Exp[-h[t] / k] / (2. beta),
       h[0] == x0, h'[0] == v0}, h, {t, t0, t1}]},
   Grid[{{Plot[h[t] /. soln, {t, t0, t1},
      ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
     Plot[h'[t] /. soln, {t, t0, t1},
      ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}]}}]]]]
```

Out[205]=





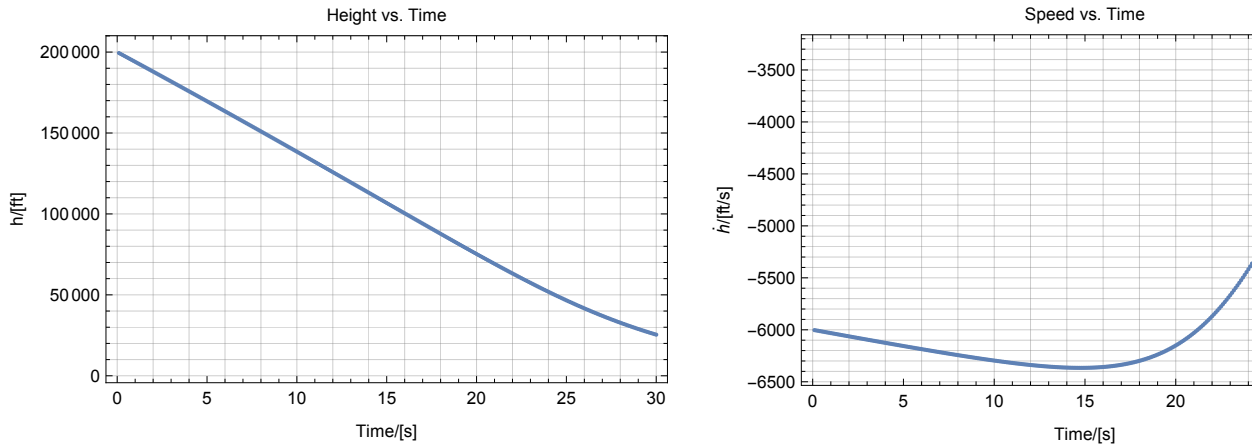
Via Zarchan & Musoff

Page 229, fourth edition.

In[206]:=

```
Module[{
  g = 32.2,
  x = 200 000., xd = -6000., xdd,
  xold, xdold, d,
  beta = 500.,
  ts = .1, tf = 30., t = 0, s = 0, h = .001,
  count = 0,
  arrayt, arrayx, arrayxd, arrayxdd},
 {arrayt, arrayx, arrayxd, arrayxdd} =
  Reap[
    While[t ≤ tf,
     xold = x;
     xdold = xd;
     xdd = 0.0034 g xd xd Exp[-x / 22 000.] / (2. beta) - g;
     x = x + h xd;
     xd = xd + h xdd;
     t = t + h;
     xdd = 0.0034 g xd xd Exp[-x / 22 000.] / (2. beta) - g;
     x = 0.5 (xold + x + h xd);
     d = 0.5 (xdold + xd + h xdd);
     s = s + h;
     If[s > ts - .00001,
      s = 0;
      count = count + 1;
      Sow[t, "t"];
      Sow[x, "x"];
      Sow[xd, "xd"];
      Sow[xdd, "xdd"]];]][[2]];
 Grid[{{ListPlot[{arrayt, arrayx}ᵀ,
     ImageSize → Medium, GridLines → Full, Frame → True,
     FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
    ListPlot[{arrayt, arrayxd}ᵀ,
     ImageSize → Medium, GridLines → Full, Frame → True,
     FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}],
    ListPlot[{arrayt, arrayxdd}ᵀ,
     ImageSize → Medium, GridLines → Full, Frame → True,
     FrameLabel → {{"ḧ/[ft/s²]", ""}, {"Time/[s]", "Acceleration vs. Time"}}]}}]]
```

Out[206]=



Via Stream

Streams Library

integersFrom

Let's make folds for potentially infinite, lazy, iteractive [sic] collections (***streams***). Represent a stream as a pair of a value and a thunk; the thunk must produce another stream.

Here's an example that produces the entire infinite set of integers:

In[207]:=

```
integersFrom[n_Integer] := {n, integersFrom[n + 1] &}
```

extract :: stream ⟶ value

Extract is a (tail-recursive) way to get the 1-indexed, n^{th} value from any stream.

By convention, a finite stream has a *Null* thunk at the end. Thus, the empty stream, obtained by invoking such a thunk, is `Null[]`. Extracting anything from such a stream produces, `Null`, which does not print in the notebook and looks like nothing.

In[208]:=

```
ClearAll[extract];
(* defaulting n: *)
extract[{v_, thunk_}] := extract[{v, thunk}, 1];
(* base case: *)
extract[Null[], _] := Null; (* somebody invoked my null thunk! *)
(* do this if you just want the first value; it's the default *)
extract[{v_, thunk_}, 1] := v;
(* tail recursion: *)
extract[{v_, thunk_}, n_Integer /; n > 1] := extract[thunk[], n - 1];
(* catch-all: *)extract[___] := Null;
```

Now we can get the 630 000-th integer efficiently (if not terribly quickly):

In[214]:=
```
Block[{$IterationLimit = Infinity},
  extract[integersFrom[1], 630 000]] // AbsoluteTiming
```
Out[214]=
```
{0.621006, 630 000}
```

take :: stream ⟶ index ⟶ stream

Take a finite number of elements from a stream and produce another stream.

In[215]:=
```
ClearAll[take];
take[stream_] := take[stream, 1];
take[_, 0] := Null[];
take[Null[], _] := Null[];
take[{v_, thunk_}, 1] := {v, Null};
take[{v_, thunk_}, n_Integer /; n > 1] := {v, take[thunk[], n - 1] &};
```

Produce a finite stream of three integers; extract the first value:

In[221]:=
```
extract[take[integersFrom[1], 3], 1]
```
Out[221]=
```
1
```

and the last value:

In[222]:=
```
extract[take[integersFrom[1], 3], 3]
```
Out[222]=
```
3
```

If we extract too far into a finite stream, we get Null, which doesn't print to the notebook:

In[223]:=
```
extract[take[integersFrom[1], 3], 4]
```

takeUntil :: stream ⟶ (value ⟶ Boolean) ⟶ stream

Take elements until some predicate evaluates to True, excluding that element.

In[224]:=
```
ClearAll[takeUntil];
takeUntil[Null[], _] := Null[];
takeUntil[{v_, thunk_}, predicate_] /; predicate[v] := Null[];
takeUntil[{v_, thunk_}, predicate_] := {v, takeUntil[thunk[], predicate] &};
```

The example needs `reify`.

reify :: stream ⟶ list

Reify a stream to a list, invoking all thunks (not tail-recursive; might have to do better if we overflow the stack):

In[228]:=
```
ClearAll[reify];
reify[Null[]] := {};
reify[{v_, Null}] := {v};
reify[{v_, thunk_}] := Join[{v}, reify[thunk[]]];
```

In[232]:=
```
reify[take[integersFrom[1], 3]]
```

Out[232]=
```
{1, 2, 3}
```

In[233]:=
```
reify[takeUntil[integersFrom[1], # > 9 &]]
```

Out[233]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

disperse :: list ⟶ stream

In[234]:=
```
ClearAll[disperse];
disperse[{}] := Null[];
disperse[{x_}] := {x, Null};
disperse[{v_, xs__}] := {v, disperse[{xs}] &};
```

In[238]:=
```
reify[disperse[{1, 2, 3}]]
```

Out[238]=
```
{1, 2, 3}
```

In[239]:=
```
reify[disperse[{}]]
```

Out[239]=
```
{}
```

last :: stream ⟶ element

In[240]:=
```
ClearAll[last];
last[Null[]] := Null;
last[{v_, thunk_} /; thunk[] === Null[]] := v;
last[{v_, thunk_}] := last[thunk[]];
```

In[244]:=
```
last@disperse[{}]
```

In[245]:=
```
last@disperse[{1, 2}]
```

Out[245]=
```
2
```

In[246]:=
```
Block[{$IterationLimit = Infinity},
  last@take[integersFrom[1], 630 000]] // AbsoluteTiming
```
Out[246]=
```
{1.86111, 630 000}
```

Fibonaccis as a Stream

Direct, non-tail recursive, fibonacci stream (with *memoization*, i.e., dynamic programming)

In[247]:=
```
ClearAll[fibonaccis];
fibonaccis[0] = {0, fibonaccis[2] &};
fibonaccis[1] = {1, fibonaccis[3] &};
fibonaccis[n_] :=
 (* Here's the dynamic
  programming: create a static value for a particular argument. *)
 fibonaccis[n] =
  {(* value *)
   extract[fibonaccis[n - 1]] + extract[fibonaccis[n - 2]],
   (* thunk *)
   fibonaccis[n + 1] &}
```

In[251]:=
```
reify[take[fibonaccis[0], 11]]
```
Out[251]=
```
{0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89}
```

This is efficient, modulo stack overflow: it can find large Fibonacci numbers quickly:

In[252]:=
```
extract[fibonaccis[0], 200] // AbsoluteTiming
```
Out[252]=
```
{0.000807, 280 571 172 992 510 140 037 611 932 413 038 677 189 525}
```

In[253]:=
```
extract[fibonaccis[200]] // AbsoluteTiming
```
Out[253]=
```
{1. × 10^-6, 280 571 172 992 510 140 037 611 932 413 038 677 189 525}
```

foldStream :: f ⟶ initState ⟶ stream ⟶ stream

In[254]:=
```
ClearAll[foldStream];
foldStream[f_, s_, Null[]] := {s, Null};
foldStream[f_, s_, {z_, thunk_}] :=
  {s, foldStream[f, f[s, z], thunk[]] &};
```

In[257]:=
```
allFibs =
  foldStream[
   {s, z} ↦ {s[[2]], s[[1]] + s[[2]]},
   {0, 1},
   integersFrom[0]];(* lazy stream: all ints *)
Transpose@reify@take[allFibs, 11] //
 TeXForm(* don't take them all; that won't finish *)
```

Out[258]//TeXForm=
```
\left(
\begin{array}{ccccccccccc}
 0 & 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 \\
 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 & 89 \\
\end{array}
\right)
```

Here is another Fibonacci stream:

In[259]:=
```
MatrixForm /@
 Module[{f = {{0, 1}, {1, 1}}, fs},
  fs[fk_] := {fk, fs[fk.f] &};
  reify@take[fs[IdentityMatrix@2], 11]]
```

Out[259]=
$$\left\{\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, \begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}, \begin{pmatrix}1 & 1\\ 1 & 2\end{pmatrix}, \begin{pmatrix}1 & 2\\ 2 & 3\end{pmatrix}, \begin{pmatrix}2 & 3\\ 3 & 5\end{pmatrix}, \begin{pmatrix}3 & 5\\ 5 & 8\end{pmatrix}, \begin{pmatrix}5 & 8\\ 8 & 13\end{pmatrix}, \begin{pmatrix}8 & 13\\ 13 & 21\end{pmatrix}, \begin{pmatrix}13 & 21\\ 21 & 34\end{pmatrix}, \begin{pmatrix}21 & 34\\ 34 & 55\end{pmatrix}, \begin{pmatrix}34 & 55\\ 55 & 89\end{pmatrix}\right\}$$

mapStream :: stream ⟶ unaryFunction ⟶ stream

In[260]:=
```
ClearAll[mapStream];
mapStream[Null[], _] := Null[];
mapStream[{v_, thunk_}, f_] := {f[v], mapStream[thunk[], f] &};
```

In[263]:=
```
reify@mapStream[disperse@{1, 2, 3}, #^2 &]
```

Out[263]=
```
{1, 4, 9}
```

In[264]:=
```
reify@mapStream[take[integersFrom[1], 3], #^2 &]
```

Out[264]=
```
{1, 4, 9}
```

forEach :: stream ⟶ unaryFunction ⟶ Null

forEach takes a stream and a unary function and invokes the function on every member of the stream

for side effect.

Don’t invoke this on infinite stream, as with *reify*.

In[265]:=
```
ClearAll[forEach];
forEach[Null[], lambda_] := Null;
forEach[{v_, thunk_}, lambda_] :=
  (lambda[v](* discard lambda's result *);
   forEach[thunk[], lambda];);
```

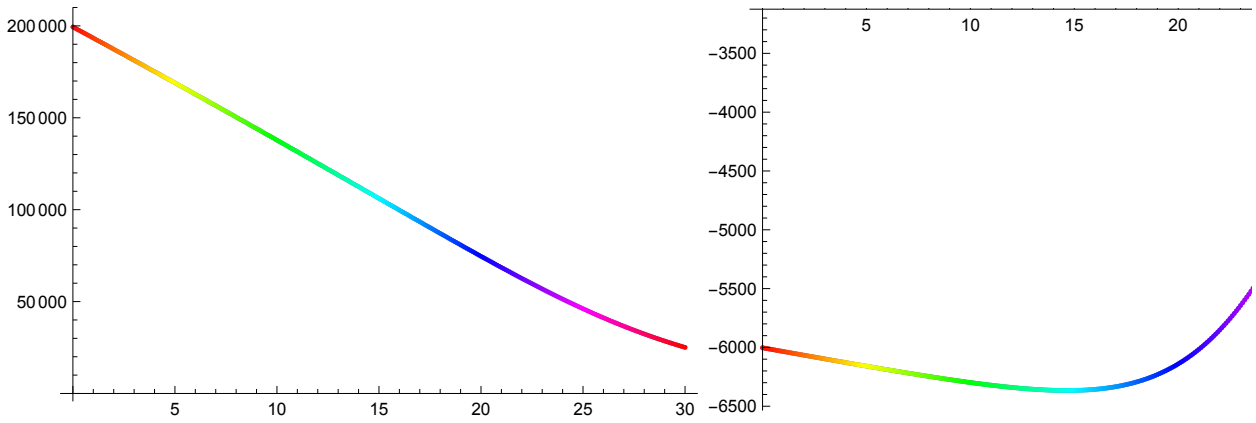
an example, side-effecting the notebook via *Print*:

In[268]:=
```
forEach[take[allFibs, 11], Print]
```

```
{0, 1}
{1, 1}
{1, 2}
{2, 3}
{3, 5}
{5, 8}
{8, 13}
{13, 21}
{21, 34}
{34, 55}
{55, 89}
```

In[269]:=
```
With[{g = 32.2, A = 0.0034, k = 22 000., beta = 500.},
 With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30., dt = .1},
  Module[{times = Range[t0, t1, dt], timeStyler},
   timeStyler[xs_] := MapIndexed[Style[#1, Hue[#2 / len@xs]] &];
   With[{soln =
      Module[{Dψ},
       Dψ[ψ : {x_, v_}, t_] := {v, -g + A g Exp[-x / k] v^2 / (2 beta)};
       FoldList[
        rk4stepA[#1, Dψ, #2, dt] &,
        {x0, v0},
        times]]},
    Grid[{
      {ListPlot[
        {timeStyler[times]@{times, d1[pick[1] /@ soln]}^T}, ImageSize → Medium],
       ListPlot[
        {timeStyler[times]@{times, d1[pick[2] /@ soln]}^T}, ImageSize → Medium]}
     }]]]]]
```

Out[269]=



In[270]:=
$$\frac{\text{slug}}{\text{ft}^3}\left(\frac{\text{ft}}{\text{sec}}\right)^2$$

Out[270]=
$$\frac{\text{slug}}{\text{ft sec}^2}$$

In[271]:=
$$\frac{\text{slug ft}}{\text{sec}^2} \Big/ \text{ft}^2$$

Out[271]=
$$\frac{\text{slug}}{\text{ft sec}^2}$$

Without a stream integrator

In[272]:=

```
ClearAll[dragStream];
(* Let-Over-Lambda style: *)
With[{g = 32.2, A = 0.0034, k = 22 000., beta = 500.},
  With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30., dt = .1},
   Module[{Dψ},
    Dψ[ψ : {x_, v_}, t_] := {v, -g - A g Exp[-x / k] v^2 / (2 beta)};
    dragStream[ψ : {δt_, t_, x_, v_}] :=
     {ψ, dragStream[{δt, t + δt} ⊕ rk4stepA[{x, v}, Dψ, t, δt]] &}]]];
```

Abstracting a non-stream integrator

In[274]:=

```
ClearAll[dragStream];
With[{integrator = eulerStepA, g = 32.2, beta = 500},
  Module[{Dx}, Dx[{x_, v_}, t_] := {v, g (0.0034 Exp[-x / 22 000] v^2 / (2. beta) - 1)};
   dragStream[psi : {dt_, t_, x_, v_}] :=
    {psi, dragStream[{dt, t + dt} ~Join~ integrator[{x, v}, Dx, t, dt]] &}]];
```

In[276]:=

```
With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30., dt = .1},
 reify[
  takeUntil[
   dragStream[{dt, t0, x0, v0}], #[[2]] > .3 &]]]
```

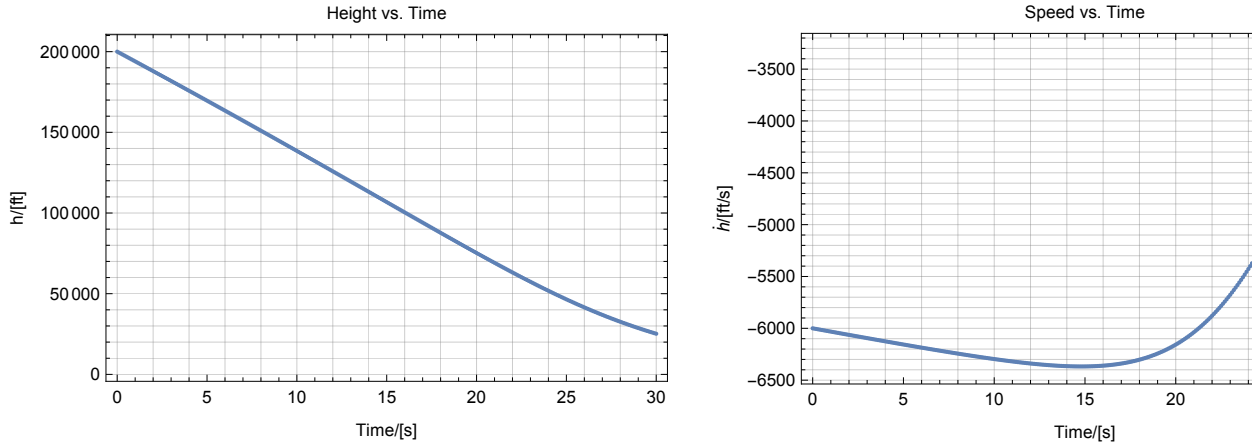
Out[276]=

```
{{0.1, 0., 200 000., -6000.}, {0.1, 0.1, 199 400., -6003.18},
 {0.1, 0.2, 198 800., -6006.35}, {0.1, 0.3, 198 199., -6009.52}}
```

In[277]:=

```
With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30., dt = .1},
 Module[{ts, xs, vs},
  {ts, xs, vs} = Transpose[#[[2 ;; 4]] & /@
     reify[
      takeUntil[
       dragStream[{dt, t0, x0, v0}], #[[2]] > 30. &]]];
  Grid[{
    {ListPlot[{{ts, xs}^T}, ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
     ListPlot[{{ts, vs}^T}, ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}]}}]]]
```

Out[277]=



Can we take a stream that implements differentials, *foldStream* an integrator over it, and get a stream that implements the integrated differential equation?

In[278]:=

```
ClearAll[eulerAccumulator, rk2Accumulator, rk4Accumulator, dragD, dragDStream];
eulerAccumulator[{t_, x_}, {dt_, t_, Dx_}] :=
  {t + dt, x + dt Dx[x, t]};
rk2Accumulator[{t_, x_}, {dt_, t_, Dx_}] :=
  With[{dx1 = dt Dx[x, t]},
   With[{dx2 = dt Dx[x + .5 dx1, t + .5 dt]},
    {t + dt, x + (dx1 + dx2) / 2.}]];
rk4Accumulator[{t_, x_}, {dt_, t_, Dx_}] :=
  With[{dx1 = dt Dx[x, t]},
   With[{dx2 = dt Dx[x + .5 dx1, t + .5 dt]},
    With[{dx3 = dt Dx[x + .5 dx2, t + .5 dt]},
     With[{dx4 = dt Dx[x + dx3, t + dt]},
      {t + dt, x + (dx1 + 2. dx2 + 2. dx3 + dx4) / 6.}]]]];
With[{g = 32.2, A = 0.0034, k = 22000., beta = 500.}, (* let-over-lambda style *)
  dragD[{x_, v_}, t_] := {v, g (A Exp[-x / k] v^2 / (2. beta) - 1)}];
dragDStream[Δ : {dt_, t_, Dx_}] := {Δ, dragDStream[{dt, t + dt, Dx}] &};
```

In[284]:=

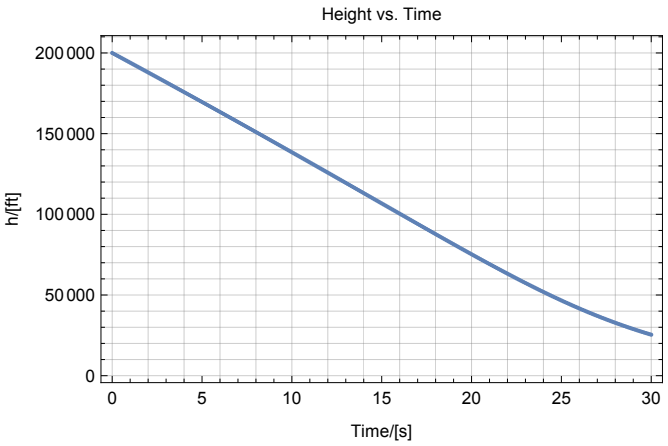
```
reify[
 With[{x0 = 200000., v0 = -6000., t0 = 0., t1 = 0.3, dt = .1},
  takeUntil[
   foldStream[
    rk4Accumulator,
    {t0, {x0, v0}},
    dragDStream[{dt, t0, dragD}]
   ], First[#] > t1 &]]]
```

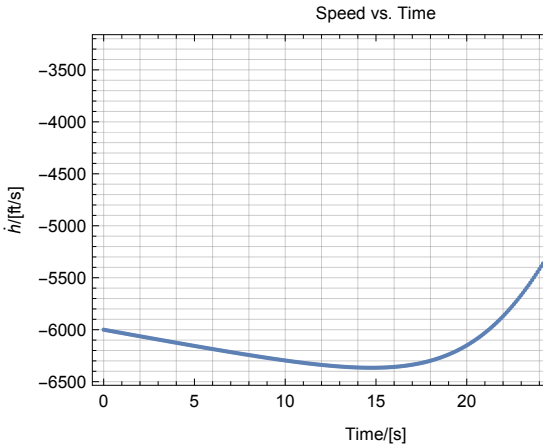
Out[284]=

```
{{0., {200000., -6000.}}, {0.1, {199400., -6003.17}},
 {0.2, {198799., -6006.35}}, {0.3, {198199., -6009.52}}}
```

In[285]:=
```
With[{soln =
   reify[
    With[{x0 = 200 000., v0 = -6000., t0 = 0., t1 = 30., dt = .1},
     takeUntil[
      foldStream[
       rk4Accumulator,
       {t0, {x0, v0}},
       dragDStream[{dt, t0, dragD}]
      ], First[#] > t1 &]]]},
 Module[{ts, ps, xs, vs},
  {ts, ps} = Transpose[soln];
  {xs, vs} = Transpose[ps];
  Grid[{
    {ListPlot[{{ts, xs}^T}, ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
     ListPlot[{{ts, vs}^T}, ImageSize → Medium, GridLines → Full, Frame → True,
      FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}]}]}]]]
```

Out[285]=





Linearization

In[286]:=
```
(ρ[x] v^2 / (2 β) - 1) g
```

Out[286]=
$$g\left(-1+\frac{v^2\,\rho[x]}{2\,\beta}\right)$$

In[287]:=
```
ρRule = {ρ[x_] :> 34/10000 Exp[x/22000]}
```
Out[287]=
$$\left\{\rho[x\_] :\to \frac{34\ \text{Exp}\left[\frac{x}{22\,000}\right]}{10\,000}\right\}$$

In[288]:=
```
D[((ρ[x] v^2)/(2 β) - 1) g /. ρRule, x]
```
Out[288]=
$$\frac{17\ e^{x/22\,000}\ g\ v^2}{220\,000\,000\,\beta}$$

In[289]:=
```
D[((ρ[x] v^2)/(2 β) - 1) g /. ρRule, v]
```
Out[289]=
$$\frac{17\ e^{x/22\,000}\ g\ v}{5000\,\beta}$$

In[290]:=
```
ρRule = {ρ[x_] :> A Exp[-x/k]}
```
Out[290]=
$$\left\{\rho[x\_] :\to A\ \text{Exp}\left[-\frac{x}{k}\right]\right\}$$

In[291]:=
```
D[((ρ[x] v^2)/(2 β) - 1) g /. ρRule, x] // TeXForm
```
Out[291]//TeXForm=
```
-\frac{A g v^2 e^{-\frac{x}{k}}}{2 \beta  k}
```

In[292]:=
```
D[((ρ[x] v^2)/(2 β) - 1) g /. ρRule, v] // TeXForm
```
Out[292]//TeXForm=
```
\frac{A g v e^{-\frac{x}{k}}}{\beta }
```

In[293]:=
```
f21 = D[((ρ[x] v^2)/(2 β) - 1) g /. ρRule, x]
```
Out[293]=
$$-\frac{A\ e^{-\frac{x}{k}}\ g\ v^2}{2\,k\,\beta}$$

In[294]:=
```
<< "Units`"
```

In[295]:=
```
f21 /. (unitRules =
    {x → Foot, k → Foot, g → Foot/Second^2, v → Foot/Second, A → Slug/Foot^3, β → Slug Foot/Second^2 / Foot^2})
```
Out[295]=
$$-\frac{1}{2\,e\,\text{Second}^2}$$

In[296]:=
```
f22 = D[(ρ[x] v^2/(2 β) - 1) g /. ρRule, v]
```
Out[296]=
$$\frac{A\,e^{-\frac{x}{k}}\,g\,v}{\beta}$$

In[297]:=
```
f22 /. unitRules
```
Out[297]=
$$\frac{1}{e\,\text{Second}}$$

In[298]:=
```
(ρ v^2/(2 β) - 1) g /. unitRules /. {ρ → Slug/Foot^3}
```
Out[298]=
$$-\frac{\text{Foot}}{2\,\text{Second}^2}$$

In[299]:=
```
ClearAll[Φ];
Φ[τ_] := (id[2] + (F = ({{0, 1}, {F21, F22}})) τ)
```

In[301]:=
```
Integrate[Φ[t].{{0, 0}, {0, 1}}.Φ[t]^T, {t, 0, dt}] // FullSimplify // MatrixForm
```
Out[301]//MatrixForm=
$$\begin{pmatrix} \frac{dt^3}{3} & \frac{1}{6}\,dt^2\,(3+2\,dt\,F22) \\ \frac{1}{6}\,dt^2\,(3+2\,dt\,F22) & dt+dt^2\,F22+\frac{dt^3\,F22^2}{3} \end{pmatrix}$$

In[302]:=
```
Integrate[Φ[t].{{0, 0}, {0, 1}}.Φ[t]^T, {t, 0, dt}] // MatrixForm
```
Out[302]//MatrixForm=
$$\begin{pmatrix} \frac{dt^3}{3} & \frac{dt^2}{2}+\frac{dt^3\,F22}{3} \\ \frac{dt^2}{2}+\frac{dt^3\,F22}{3} & dt+dt^2\,F22+\frac{dt^3\,F22^2}{3} \end{pmatrix}$$

EKF Interactive

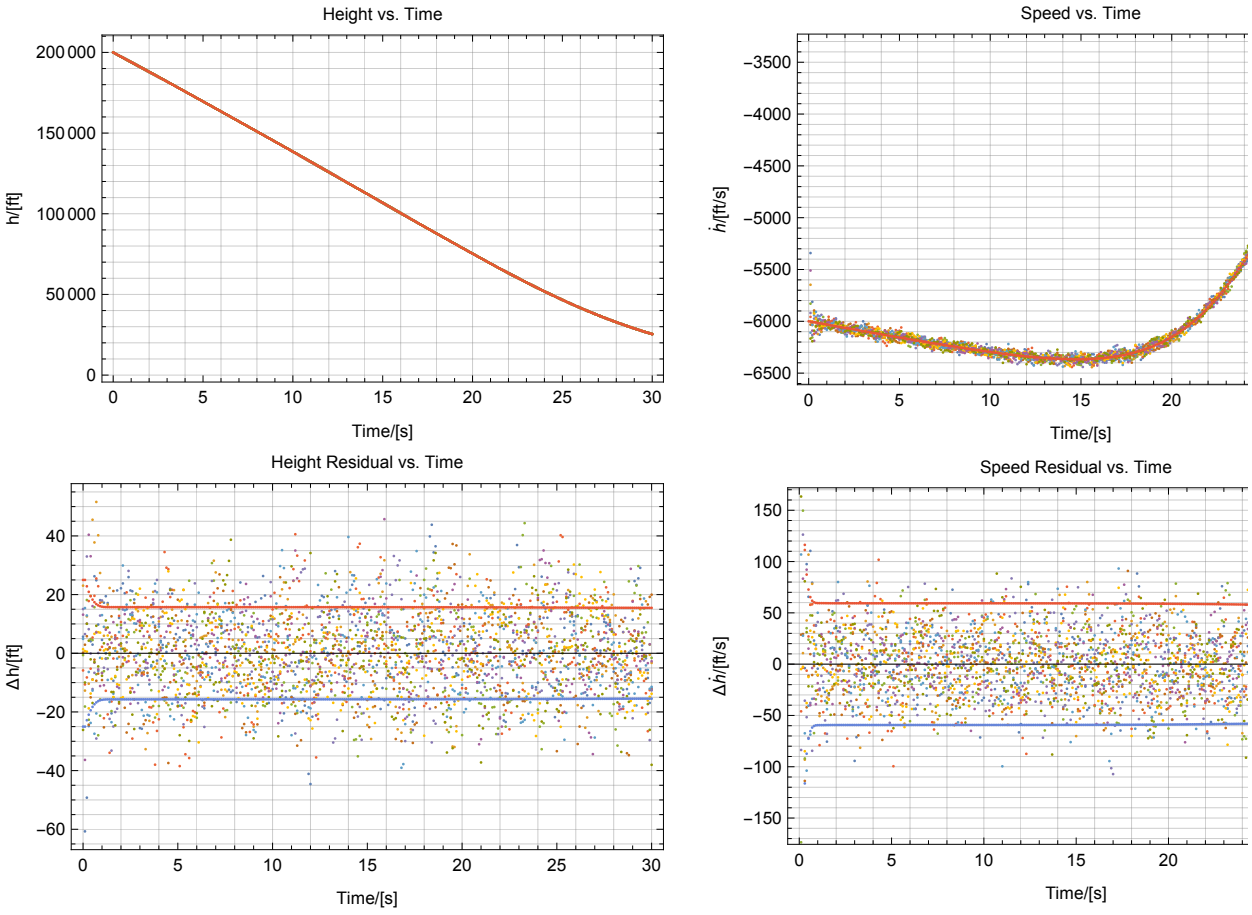
Experiment by hand

In[303]:=
```
ClearAll[Phi, Xi];
```

In[304]:=
```
With[{g = 32.2, A = 0.0034, k = 22000., beta = 500.},
 Module[{dragD, dragDStream, F21, F22, Phi, Xi, EKFDrag},
  dragD[{x_, v_}, t_] := {v, g (A Exp[-x / k] v^2 / (2. beta) - 1)};
  dragDStream[Delta : {dt_, t_, Dx_}] := {Delta, dragDStream[{dt, t + dt, Dx}] &};
  F21[x_, v_] := -A Exp[-x / k] g v^2 / (2. k beta);
  F22[x_, v_] := A Exp[-x / k] g v / beta;
  Phi[dt_, {x_, v_}] := {{1, dt}, {dt * F21[x, v], 1 + dt * F22[x, v]}};
  Xi[dt_, {x_, v_}] := With[{f = F22[x, v]},
    {{dt^3 / 3, (dt^2 * (3 + 2 * dt * f)) / 6},
     {(dt^2 * (3 + 2 * dt * f)) / 6, dt + dt^2 * f + (dt^3 * f^2) / 3}}];
  EKFDrag[sigmaXi_, Zeta_, integrator_, fdt_, idt_][{x_, P_}, {t_, A_, z_}] :=
   Module[{x2, P2, D, K},
    x2 = last[takeUntil[foldStream[integrator, {t, x},
          dragDStream[{idt, t, dragD}]],
         First@# > t + fdt &]][[2]];
    P2 = sigmaXi^2 Xi[fdt, x] + Phi[fdt, x].P.Phi[fdt, x]^T;
    D = Zeta + A.P2.A^T;
    K = P2.A^T.inv[D];
    {x2 + K.(z - A.x2), P2 - K.D.K^T}];
  With[{nStates = 2, nIterations = 10},
   With[{sigmaZeta = 25., sigmaXi = 100.0,
     t0 = 0., t1 = 30., filterDt = 0.1, integrationDt = 0.1},
    With[
     {x0 = 200000, v0 = -6000, Z = sigmaZeta^2 id[1], P0 = 1000000000000 id[nStates]},
     Module[{fakes},
      fakes[] := foldStream[rk4Accumulator, {t0, {x0, v0}},
        dragDStream[{filterDt, t0, dragD}]];
      SeedRandom[44];
      Module[{ffs, rffs, ts, xvs, xss, vss, txs, tvs, ps, σxs, σvs},
       xss = ConstantArray[0, nIterations];
       vss = ConstantArray[0, nIterations];
       ffs = takeUntil[fakes[], First@# > t1 &];
       rffs = reify@ffs;
       ts = pick[1] /@ rffs;
```

```
    txs = pick[2, 1] /@ rffs;
    tvs = pick[2, 2] /@ rffs;
    {xss, vss} = Transpose@Map[
        ({xvs, ps} = Transpose@Rest@reify@foldStream[
                EKFDrag[sigmaXi, z, rk2Accumulator, filterDt, integrationDt],
                {{0, 0}, P0},
                mapStream[ffs, {#[[1]], ( 1 0 ), #[[2, 1]] + gen[z]} &]];
          σxs = If[Not[ValueQ[σxs]], Sqrt[pick[1, 1] /@ ps], σxs];
          σvs = If[Not[ValueQ[σvs]], Sqrt[pick[2, 2] /@ ps], σvs];
          Transpose@xvs) &,
        Range[nIterations]];
    Grid[{
      {ListPlot[Transpose /@ ({ts, #} & /@ xss) ⊕ {{ts, txs}^T},
        ImageSize → Medium, GridLines → Full, Frame → True,
        FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
       ListPlot[Transpose /@ ({ts, #} & /@ vss) ⊕ {{ts, tvs}^T},
        ImageSize → Medium, GridLines → Full, Frame → True,
        FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}]},
      {ListPlot[Transpose /@ ({ts, txs - #} & /@ xss) ⊕ {{ts, σxs}^T, {ts, -σxs}^T},
        ImageSize → Medium, GridLines → Full, Frame → True,
        FrameLabel →
         {{"Δh/[ft]", ""}, {"Time/[s]", "Height Residual vs. Time"}}],
       ListPlot[Transpose /@ ({ts, tvs - #} & /@ vss) ⊕ {{ts, σvs}^T, {ts, -σvs}^T},
        ImageSize → Medium, GridLines → Full, Frame → True,
        FrameLabel →
         {{"Δḣ/[ft/s]", ""}, {"Time/[s]", "Speed Residual vs. Time"}}]}}]
  ]]
]]]]]
```

Out[304]=



Manipulator

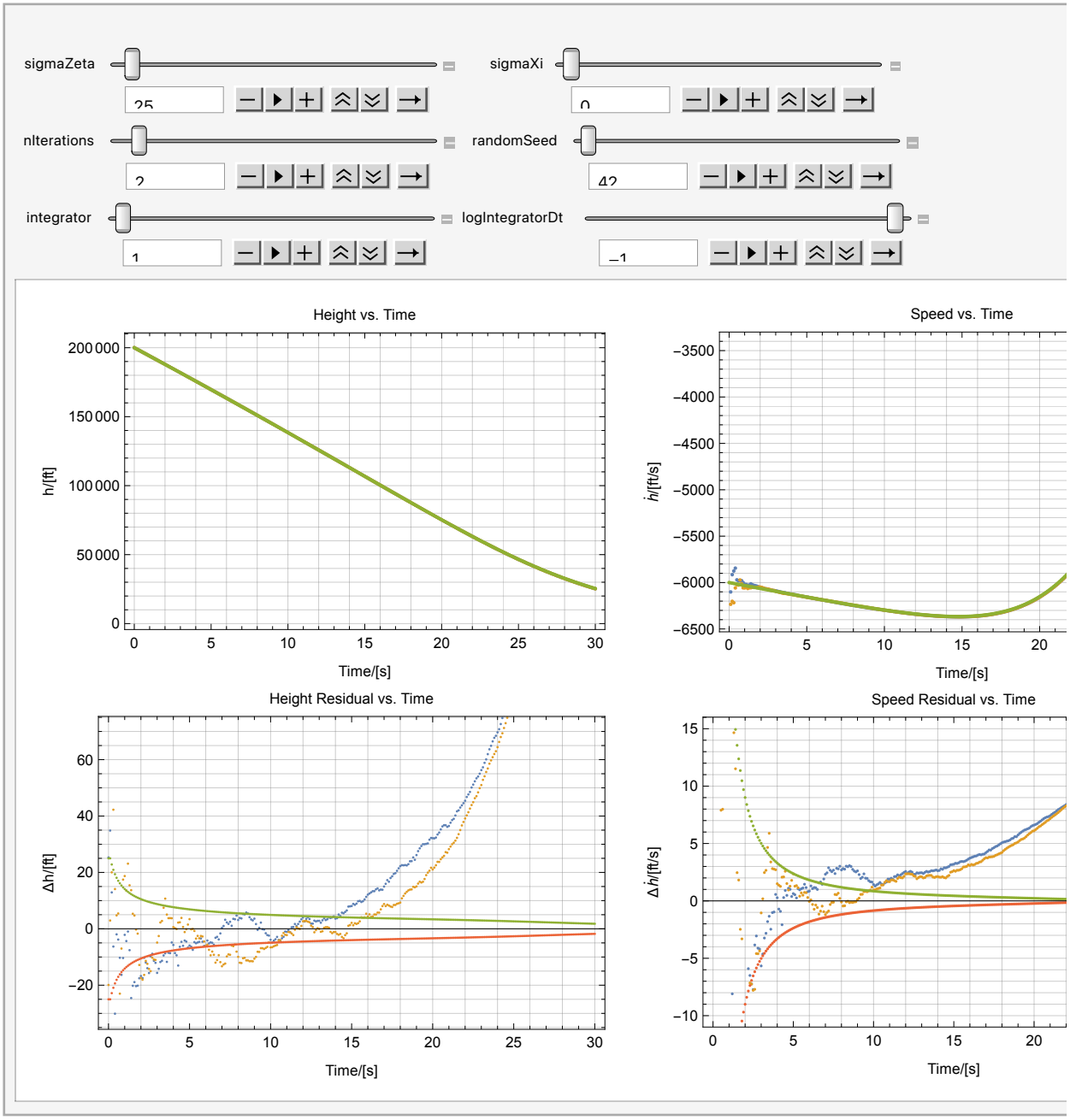
In[305]:=

```
Manipulate[
 With[{g = 32.2, A = 0.0034, k = 22000., beta = 500.},
  Module[{dragD, dragDStream, F21, F22, Phi, Xi, EKFDrag},
   dragD[{x_, v_}, t_] := {v, g (A Exp[-x / k] v^2 / (2. beta) - 1)};
   dragDStream[Delta : {dt_, t_, Dx_}] := {Delta, dragDStream[{dt, t + dt, Dx}] &};
   F21[x_, v_] := -A Exp[-x / k] g v^2 / (2. k beta);
   F22[x_, v_] := A Exp[-x / k] g v / beta;
   Phi[dt_, {x_, v_}] := {{1, dt}, {dt * F21[x, v], 1 + dt * F22[x, v]}};
   Xi[dt_, {x_, v_}] := With[{f = F22[x, v]},
     {{dt^3 / 3, (dt^2 * (3 + 2 * dt * f)) / 6},
      {(dt^2 * (3 + 2 * dt * f)) / 6, dt + dt^2 * f + (dt^3 * f^2) / 3}}];
   EKFDrag[sigmaXi_, Zeta_, integrator_, fdt_, idt_][{x_, P_}, {t_, A_, z_}] :=
    Module[{x2, P2, D, K},
     x2 = last[takeUntil[foldStream[integrator, {t, x},
           dragDStream[{idt, t, dragD}]],
```

```
      First@# > t + fdt &]][[2]];
  P2 = sigmaXi^2 Xi[fdt, x] + Phi[fdt, x].P.Phi[fdt, x]^T;
  D = Zeta + A.P2.A^T;
  K = P2.A^T.inv[D];
  {x2 + K.(z - A.x2), P2 - K.D.K^T}];
Block[{$IterationLimit = Infinity},
 With[{nStates = 2},
  With[{t0 = 0., t1 = 30., filterDt = 0.1},
   With[{x0 = 200 000, v0 = -6000,
     z = sigmaZeta^2 id[1], P0 = 1 000 000 000 000 id[nStates]},
    Module[
     {fakes, integrators = {eulerAccumulator, rk2Accumulator, rk4Accumulator},
      integrationDt},
     fakes[] := foldStream[rk4Accumulator, {t0, {x0, v0}},
       dragDStream[{filterDt, t0, dragD}]];
     integrationDt = 10.0^logIntegratorDt;
     SeedRandom[randomSeed];
     Module[{ffs, rffs, ts, xvs, xss, vss, txs, tvs, ps, σxs, σvs},
      xss = ConstantArray[0, nIterations];
      vss = ConstantArray[0, nIterations];
      ffs = takeUntil[fakes[], First@# > t1 &];
      rffs = reify@ffs;
      ts = pick[1] /@ rffs;
      txs = pick[2, 1] /@ rffs;
      tvs = pick[2, 2] /@ rffs;
      {xss, vss} = Transpose@Map[
          ({xvs, ps} = Transpose@Rest@reify@foldStream[
                 EKFDrag[sigmaXi, z,
                  integrators[[integrator]], filterDt, integrationDt],
                 {{0, 0}, P0},
                 mapStream[ffs, {#[[1]], (1 0), #[[2, 1]] + gen[z]} &]];
            σxs = If[Not[ValueQ[σxs]], Sqrt[pick[1, 1] /@ ps], σxs];
            σvs = If[Not[ValueQ[σvs]], Sqrt[pick[2, 2] /@ ps], σvs];
            Transpose@xvs) &,
          Range[nIterations]];
      Grid[{
        {ListPlot[Transpose /@ ({ts, #} & /@ xss) ⊕ {{ts, txs}^T},
          ImageSize → Medium, GridLines → Full, Frame → True,
          FrameLabel → {{"h/[ft]", ""}, {"Time/[s]", "Height vs. Time"}}],
         ListPlot[Transpose /@ ({ts, #} & /@ vss) ⊕ {{ts, tvs}^T},
          ImageSize → Medium, GridLines → Full, Frame → True,
          FrameLabel → {{"ḣ/[ft/s]", ""}, {"Time/[s]", "Speed vs. Time"}}]},
```

```
            {ListPlot[Transpose /@ ({ts, txs - #} & /@ xss) ⊕ {{ts, σxs}^T, {ts, -σxs}^T},
              ImageSize → Medium, GridLines → Full, Frame → True,
              FrameLabel →
               {{"Δh/[ft]", ""}, {"Time/[s]", "Height Residual vs. Time"}}],
             ListPlot[Transpose /@ ({ts, tvs - #} & /@ vss) ⊕ {{ts, σvs}^T, {ts, -σvs}^T},
              ImageSize → Medium, GridLines → Full, Frame → True,
              FrameLabel →
               {{"Δḣ/[ft/s]", ""}, {"Time/[s]", "Speed Residual vs. Time"}}]}]]]
       ]]
    ]]]]],
Grid[
 Control[{{sigmaZeta, 25}, 1, 1000, 1, Appearance → "Open"}]     Control[{{sigmaXi, 0]
  Control[{{nIterations, 2}, 1, 25, 1, Appearance → "Open"}]    Control[{{randomSeed, 4
   Control[{{integrator, 1}, 1, 3, 1, Appearance → "Open"}]  Control[{{logIntegratorD
 ]]
```

Out[305]=



Via Zarchan & Musoff [UNFINISHED]

Not much point in finishing this as I reproduced their results with an Euler integrator!

In[306]:=

```
(*Module[{signoise,x,xd,beta,xh,xdh,order,ts,tf,phis,t,s,h,p,idnp,hmat,
  rmat,count,xold,xdold,xdd,rho,f21,f22,f,phi,q,m,k,xnoise,xddb,xdb,xb,
  res,errx,sp11,errxd,sp22,sp11p,sp22p,arrayt,arrayx,arrayxh,arrayxd,
  arrayxdh,arrayerrx,arraysp11,arraysp11p,arrayerrxd,arraysp22,arraysp22p},
 signoise=1000.;
 x=200000.;
 xd=-6000.;
 beta=500.;
 xh=200025.;
 xdh=6150.;
 order=2;
 ts=.1;
 tf=30.;
 phis=0./tf;
 t=0.;
 s=0.;
 h=0.001;
 p=(signoise^2   0
      0.     20000);
 idnp=id[order];
 hmat=(1 0);
 rmat=signoise^2;
 count=0;
 {arrayt,arrayx,arrayxh,arrayxd,arrayxdh,arrayerrx,
   arraysp11,arraysp11p,arrayerrxd,arraysp22,arraysp22p}=
  Reap@While[t≤tf,
     xold=x;
     xdold=xd;
     xdd=.0034 32.2 xd xd Exp[-x/22000.]/(2.beta)-32.2;
     x=x+h*xd;
     xd=xd+h xdd;
    ][[2]];
]*)
```

Non-Dimensionalization

In[307]:=

```
ClearAll[x, v, t, A, F, f11, f12, f21, f22];
```

In[308]:=

```
DSolve[{Dt[x[t], t] == F x[t], x[0] == A}, x, t]
```

Out[308]=

$$\{\{x \to \text{Function}[\{t\}, A\, e^{F\, t}]\}\}$$

In[309]:=
```
soln$ = DSolve[x''[t] == a x[t] + b x'[t], x, t]
```
Out[309]=
$$\left\{\left\{x \to \text{Function}\left[\{t\},\ e^{\frac{1}{2}\left(b-\sqrt{4\,a+b^2}\right)t}\,c_1 + e^{\frac{1}{2}\left(b+\sqrt{4\,a+b^2}\right)t}\,c_2\right]\right\}\right\}$$

In[310]:=
```
D[(x /. soln$)[[1]][t], {t, 2}] -
  a (x /. soln$)[[1]][t] - b D[(x /. soln$)[[1]][t], t] // FullSimplify
```
Out[310]=
$$0$$

In[311]:=
```
DSolve[Dt[{{x'[t]}}, t] == (a b).{{x[t]}, {x'[t]}}, x, t]
```
Out[311]=
$$\left\{\left\{x \to \text{Function}\left[\{t\},\ e^{\frac{1}{2}\left(b-\sqrt{4\,a+b^2}\right)t}\,c_1 + e^{\frac{1}{2}\left(b+\sqrt{4\,a+b^2}\right)t}\,c_2\right]\right\}\right\}$$

In[312]:=
```
DSolve[
```

ExperimentEKF

In[312]:=
```
ClearAll[experimentEKF];
experimentEKF[filter_,
   startTime_, endTime_, timeIncrement_,
   aPrioriState_, aPrioriCovariance_, fakeFn_,
   nStates_Integer : 2, truthFns_List : {Identity, Identity},
   labelsPacket_List : stir@labelsPacket2[
      {"Independent Variable", "?dim"}, {{"State1", "?dim"}, {"State2", "?dim"}}],
   statsLabels_List : {"State1", "State2"}, iterations_Integer : 1] :=
  Module[{t, timesIter, times, fakes, estss,
    trueStates, estimatedStatess, residualStatess,
    stddevStatess, theoStateVarss, statsStates, styler},
   styler = myStyle /@ # &;
   timesIter = {t, startTime, endTime, timeIncrement};
   times = Table[t, Evaluate@timesIter];
   trueStates = Table[truthFns[[n]][t], {n, nStates}, Evaluate@timesIter];
   fakes[] := Table[fakeFn[timeIncrement, t], Evaluate@timesIter];
   estss = Table[
     FoldList[filter, {aPrioriState, aPrioriCovariance}, fakes[]],
     iterations];
   (* starts at 2;; to skip the a-priori *)
   estimatedStatess = Table[estss[[ ;; , 2 ;;, 1, n, 1]], {n, nStates}];
   residualStatess =
    Table[trueStates[[n]] - # & /@ estimatedStatess[[n]], {n, nStates}];
   stddevStatess = Table[sqrt@estss[[ ( ;; ), (2 ;;), 2, n, n]], {n, nStates}];
   theoStateVarss = Table[Last[estss[[1]]][[2, n, n]], {n, nStates}];
   statsStates = Table[stats[statsLabels[[n]],
      f1@residualStatess[[n]],
      f1@stddevStatess[[n]], theoStateVarss[[n]]], {n, nStates}];
   Grid@{Table[ListLinePlot[(withTimes[times, #] & /@ residualStatess[[n]]) ⊕
       (withTimes[times, #] & /@ stddevStatess[[n]]) ⊕
       (withTimes[times, #] & /@ -stddevStatess[[n]]),
      Frame → True, FrameLabel → styler /@ labelsPacket[[n]],
      GridLines → All, ImageSize → Medium], {n, nStates}],
    Table[ListLinePlot[{withTimes[times, trueStates[[n]]]} ⊕
       (withTimes[times, #] & /@ estimatedStatess[[n]]),
      Frame → True, FrameLabel → styler /@ labelsPacket[[n + nStates]],
      GridLines → All, ImageSize → Medium], {n, nStates}],
    statsStates}];
```

Three-State Accelerometer

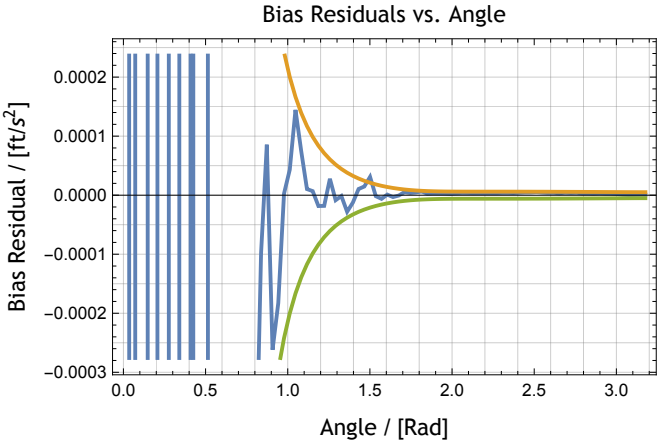
With $P \leftarrow L.P$

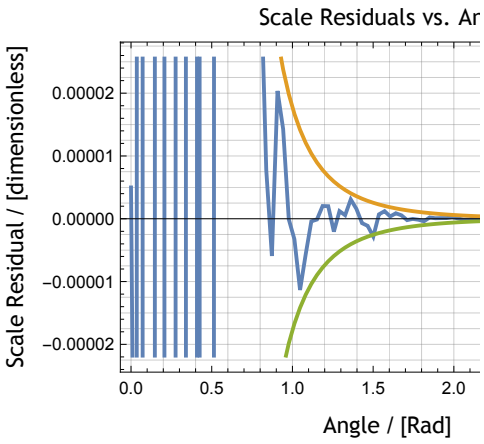
In[314]:=

```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {z_, Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{D, K, L, P1, P3, P4},
   D = z + A.P.A^T;
   K = P.A^T.inv[D];
   L = (id[len[P]] - K.A);
   P1 = L.P;
   P3 = L.P.L^T + K.z.K^T;
   P4 = P - K.D.K^T;
   (*Print[<|"P3"→P3|>];*)
   Sow[Sqrt[(P1)[[1, 1]]], "std1"];
   Sow[Sqrt[(P1)[[2, 2]]], "std2"];
   Sow[Sqrt[(P1)[[3, 3]]], "std3"];
   {x + K.(z - A.x), P1(*P2-K.D.K^T*)}];
With[{σx = 0, σθ = 1. × 10.^-6,
  σθtheoretical = 1 × 10^-6, g = 32.2, δθ = 2. °, θ0 = 0., θ1 = 180. °,
  nStates = 3, nIterations = 1, nDisturbances = 1},
 With[{bias0 = 10. × 10^-6 Abs[g], scale0 = 5. × 10^-6, drift0 = 1. × 10^-6 / Abs[g]},
  With[{Ξ = zero[nStates], Φ = zero[nStates],
    Γ = zero[nStates, nDisturbances], u = zero[nDisturbances]},
   With[{groundTruth = {bias0, scale0, drift0},
     aPrioriState = zero[nStates, 1], aPrioriCovariance = 99 999 999 999 id[nStates]},
    Module[{z, zees, partials},
     z[θ_] := col[{σθtheoretical^2 g^2 (Sin[θ])^2}];
     partials[θ_] := (1 g Cos[θ] (g Cos[θ])^2);
     SeedRandom[42];
     {Reap[Reap[Reap[
         experimentTDZ[θ0, 91 δθ, δθ, aPrioriState, aPrioriCovariance,
          {dθ, θ} ↦
           With[{θnoisy = θ + randn[σθ]},
            {z[θnoisy], Ξ, Φ, Γ, u,
             partials[θnoisy], (* observation partials *)
             partials[θnoisy].groundTruth + {g Cos[θnoisy] - g Cos[θ]}}],
          nStates, {θ ↦ bias0, θ ↦ scale0, θ ↦ drift0},
          stir@labelsPacket2[{"Angle", "Rad"},
            {{"Bias", "ft/s^2"}, {"Scale", "dimensionless"}, {"Drift", "s^2/ft"}}],
          {"Bias", "Scale", "Drift"}, nIterations],
        "std1", ListLinePlot[Flatten[#2], ImageSize → Medium],
```

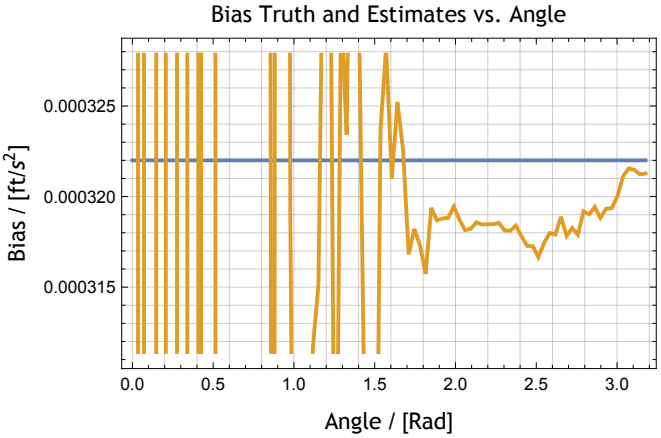
```
      Frame → True, GridLines → All, PlotLabel → #1] &],
   "std2", ListLinePlot[zees = Flatten[#2],
     ImageSize → Medium, Frame → True, GridLines → All, PlotLabel → #1] &],
  "std3", ListLinePlot[Flatten[#2], ImageSize → Medium,
    Frame → True, GridLines → All, PlotLabel → #1] &]}]]]]]
```

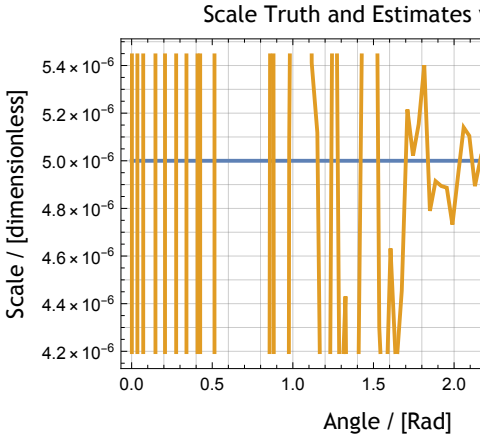
Out[316]=





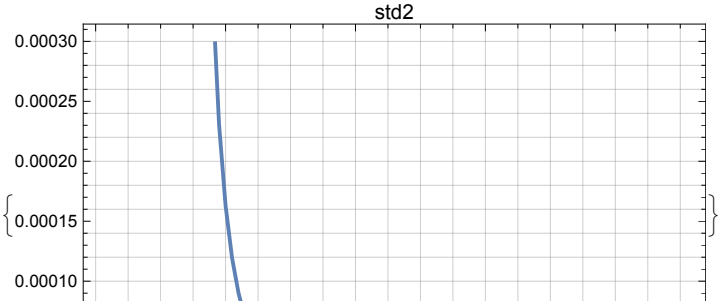
{{{{



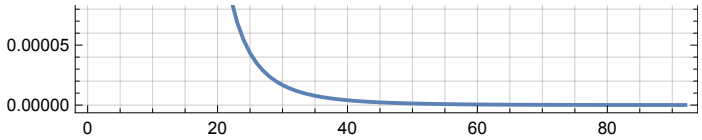


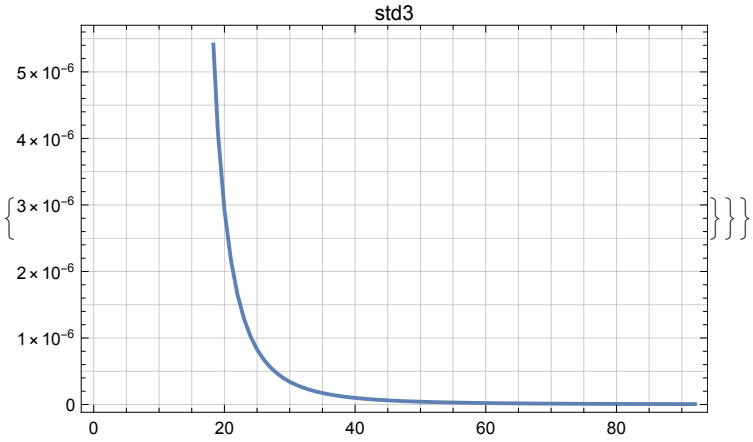
% Bias resids < σ (E >= 68)	86.9565
Bias resids >= σ	7.
Bias resids < σ	80.
average Bias resid	0.0113483
theoretical std dev	5.08884×10^{-6}
std dev of Bias resids	0.114917

% Scale resids < σ (E >= 68)	
Scale resids >= σ	
Scale resids < σ	
average Scale resid	-
theoretical std dev	9
std dev of Scale resids	



}}},



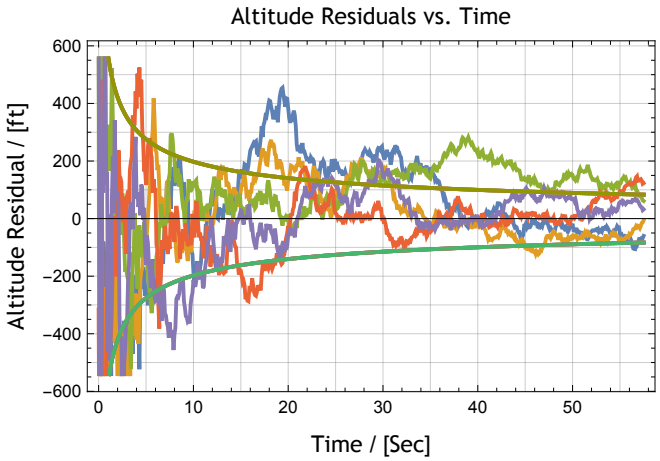


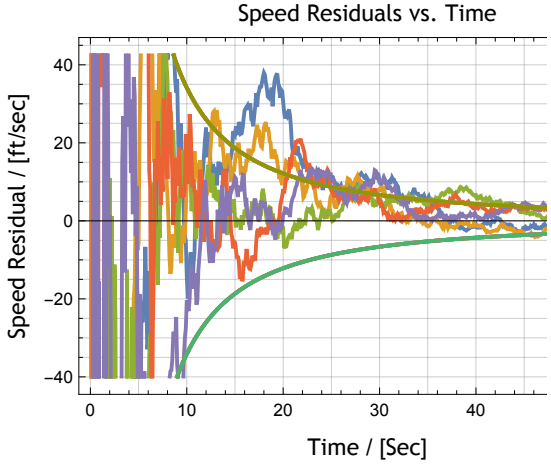
Two-State Falling Object

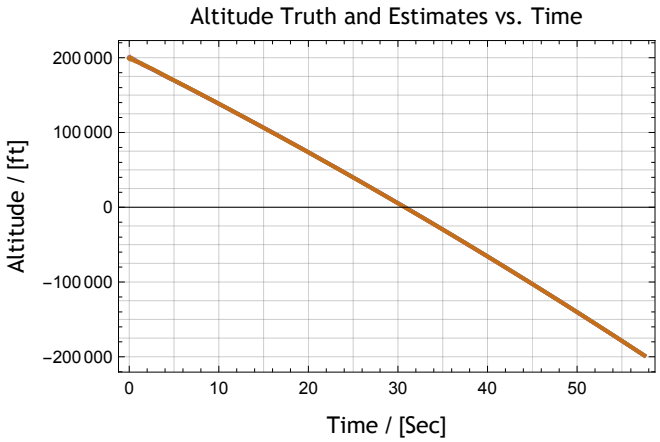
In[317]:=

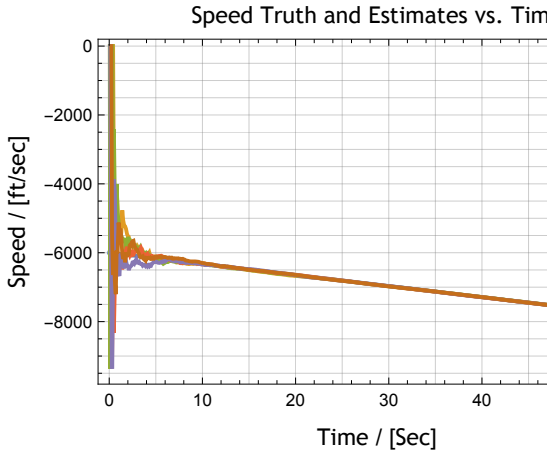
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {Z_, Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L},
   x2 = Φ.x + Γ.u; P2 = Ξ + Φ.P.Φ^T; D = Z + A.P2.A^T; K = P2.A^T.inv[D];
   L = (id[len[P]] - K.A);
   {x2 + K.(z - A.x2), L.P2}];
With[{g = 32.2, αsh = 22 000(* atmospheric scale height [ft] *)},
 With[{x0 = 200 000, v0 = -6000, a0 = -g, σz = 1000.,
   σx = 0., nStates = 2, nDisturbances = 1, nIterations = 5},
  Module[{ρ, qp, β, drag, Dψ},
   ρ[x_] := 0.0034 Exp[-x / αsh];
   (* English units/g; Zarchan, P., 1998 *)
   qp[{x_, v_}] := 0.5 ρ[x] v^2; β = 500; drag[ψ_] := qp[ψ] g / β;
   Dψ[ψ : {x_, v_}, t_] := {v, a0 + drag[ψ]};
   Module[{f21, f22},
    f21[ψ : {x_, v_}] := -ρ[x] v^2 / (2 αsh β);
    f22[ψ : {x_, v_}] := ρ[x] x / β;
    With[{Ξ = zero[nStates], Φ = zero[nStates],
      Γ = zero[nStates, nDisturbances], u = zero[nDisturbances]},
     With[{aPrioriState = zero[nStates, 1],
       aPrioriCovariance = 99 999 999 999 id[nStates],
       Z = col[{σz^2}]},
      SeedRandom[42];
      experimentTDZ[0, 57.5, 0.1, aPrioriState, aPrioriCovariance,
       {dt, t} ↦
        {Z, σx^2 {{dt^3/3, dt^2/2}, {dt^2/2, dt}}, {{1, dt}, {0, 1}}, {{dt^2/2}, {dt}}, col[{a0}], (1 0),
         col[{x0 + v0 t + a0 t^2/2}] + gen[Z]}, nStates,
       {t ↦ x0 + v0 t + a0 t^2/2, (* true position *)
        t ↦ v0 + a0 t},
       stir@
        labelsPacket2[{"Time", "Sec"}, {{"Altitude", "ft"}, {"Speed", "ft/sec"}}],
       {"Altitude", "Speed"}, nIterations]]]]]]]
```

Out[319]=









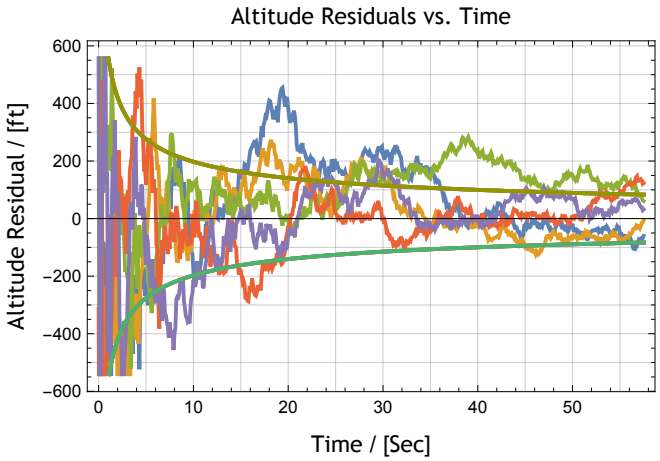
% Altitude resids < σ (E >= 68)	67.5694
Altitude resids >= σ	934.
Altitude resids < σ	1946.
average Altitude resid	23.3447
theoretical std dev	83.2249
std dev of Altitude resids	176.498

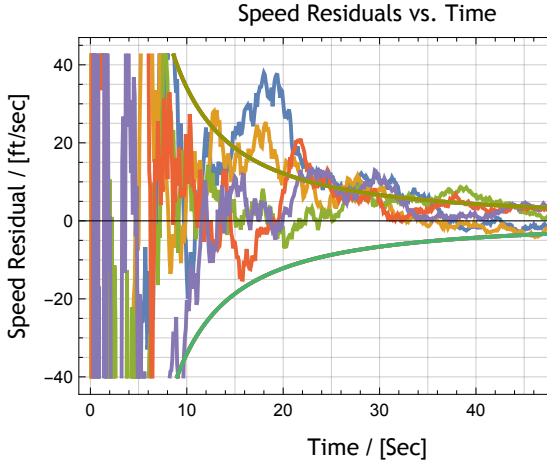
% Speed resids < σ (E >= 68)	71
Speed resids >= σ	8
Speed resids < σ	2(
average Speed resid	−76
theoretical std dev	2.5
std dev of Speed resids	12

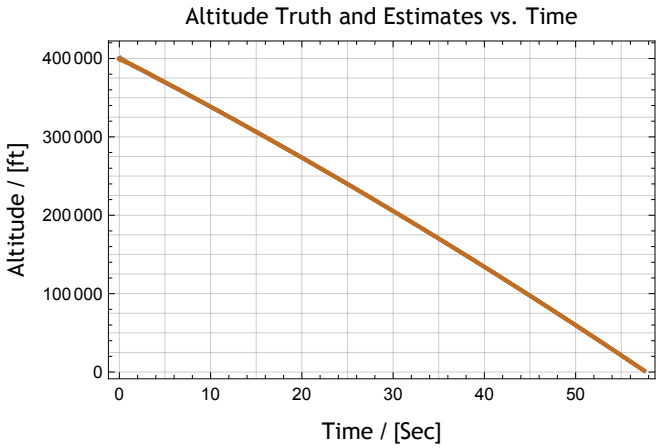
In[320]:=

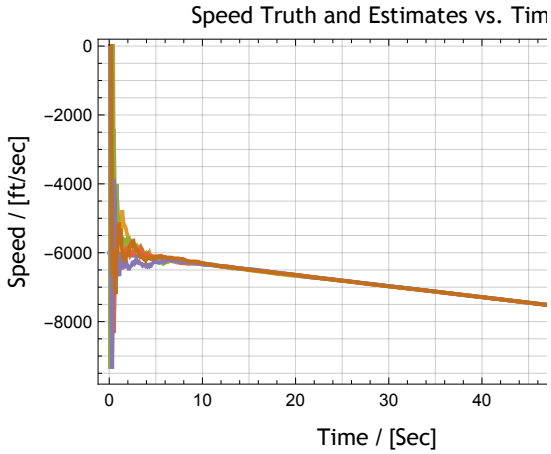
```
ClearAll[kalmanTDZ];
kalmanTDZ[{x_, P_}, {Z_, Ξ_, Φ_, Γ_, u_, A_, z_}] :=
  Module[{x2, P2, D, K, L},
   x2 = Φ.x + Γ.u; P2 = Ξ + Φ.P.Φ^T; D = Z + A.P2.A^T; K = P2.A^T.inv[D];
   L = (id[len[P]] - K.A);
   {x2 + K.(z - A.x2), L.P2}];
With[{x0 = 400 000, v0 = -6000, g = -32.2,
  σz = 1000., σx = 0., nStates = 2, nDisturbances = 1, nIters = 5},
 With[{Ξ = zero[nStates], Φ = zero[nStates],
   Γ = zero[nStates, nDisturbances], u = zero[nDisturbances]},
  With[
   {aPrioriState = zero[nStates, 1], aPrioriCovariance = 99 999 999 999 id[nStates],
    Z = col[{σz^2}]},
   SeedRandom[42];
   experimentTDZ[0, 57.5, 0.1, aPrioriState, aPrioriCovariance,
    {dt, t} ↦
     {Z, σx^2 {{dt^3/3, dt^2/2}, {dt^2/2, dt}}, {{1, dt}, {0, 1}}, {{dt^2/2}, {dt}}, col[{g}], (1 0),
      col[{x0 + v0 t + g t^2/2}] + gen[Z]}, nStates,
    {t ↦ x0 + v0 t + g t^2/2, (* true position *)
     t ↦ v0 + g t},
    stir@
     labelsPacket2[{"Time", "Sec"}, {{"Altitude", "ft"}, {"Speed", "ft/sec"}}],
    {"Altitude", "Speed"}, nIters]]]]
```

Out[322]=









% Altitude resids < σ (E >= 68)	67.5694
Altitude resids >= σ	934.
Altitude resids < σ	1946.
average Altitude resid	23.3069
theoretical std dev	83.2249
std dev of Altitude resids	176.536

% Speed resids < σ (E >= 68)	71
Speed resids >= σ	8
Speed resids < σ	2(
average Speed resid	-11
theoretical std dev	2.5
std dev of Speed resids	198

Junkyard

Recurrent Linear Regression

Try to find best-fit m, b, where $z = m\,x + b$.

Ground-Truth

In[323]:=

```
ClearAll[m, b];
{m, b} = {0.5, -1. / 3.};
```

Model Partials

In[325]:=
```
ClearAll[nData, min, max];
nData = 150; min = -1.; max = 3.;
ClearAll[partials];
partials = Table[{x, 1}, {x, min, max, (max - min) / (nData - 1)}];
Length[partials]
```
Out[329]=
```
150
```

Fake Data

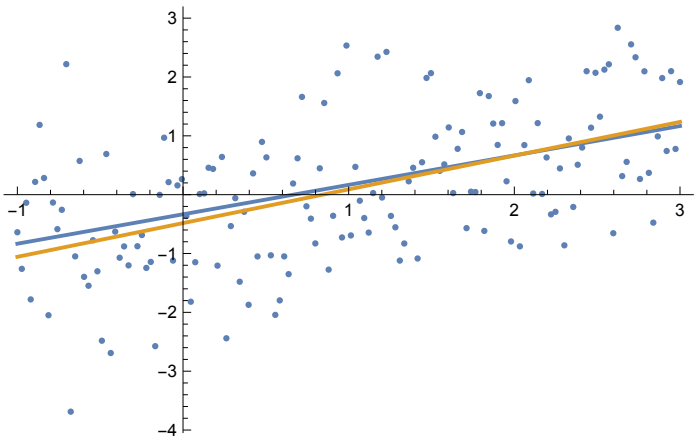
In[330]:=
```
ClearAll[data, noiseBias, noiseSigma];
noiseBias = 0.0; noiseSigma = 1.0;
data = Table[RandomVariate[NormalDistribution[noiseBias, noiseSigma]]
     + partials[[i]].{m, b}, {i, nData}];
Short[data]
```
Out[333]//Short=
```
{-0.638891, <<148>>, 1.91458}
```

Model

In[334]:=
```
ClearAll[model];
model = LinearModelFit[{partials[[All, 1]], data}^T, ξ, ξ];
Normal[model]
```
Out[336]=
```
-0.483186 + 0.573192 ξ
```

In[337]:=
```
Show[
 ListPlot[{partials[[All, 1]], data}^T],
 Plot[{m ξ + b, model[ξ]}, {ξ, min, max}]]
```

Out[337]=



Normal Equations

In[338]:=
```
Inverse[partials^T.partials].partials^T.data
```

Out[338]=
```
{0.573192, -0.483186}
```

In[339]:=
```
{data[[1 ;; 3]], partials[[1 ;; 3]]}^T
```

Out[339]=
```
{{-0.638891, {-1., 1}}, {-1.26067, {-0.973154, 1}}, {-0.137545, {-0.946309, 1}}}
```

Recurrent Form

In[340]:=
```
ClearAll[update];
update[{ψ_, Λ_}, {ξv_, Av_}] :=
  With[{ξ = {{ξv}}, A = {Av}},
   With[{Π = (Λ + A^T.A)},
    {Inverse[Π].(A^T.ξ + Λ.ψ), Π}]];
```

In[342]:=
```
Fold[update, {{{0}, {0}}, ({{1.0*^-6, 0}, {0, 1.0*^-6}})}, {data, partials}^T]
```

Out[342]=
```
{{{0.573192}, {-0.483186}}, {{352.685, 150.}, {150., 150.}}}
```

State Space

System Dynamics Equation in state-space form.

In[343]:=
$$\frac{\partial\, m[\Psi]}{\partial t} == m[F.\Psi + G.u]$$

Out[343]=
$$\frac{\partial 0.5[\{\{x\}, \{xd\}, \{xdd\}\}]}{\partial t} == 0.5[F.\{\{x\}, \{xd\}, \{xdd\}\} + \{\{0\}, \{0\}, \{1\}\}.u]$$

Mathematica can easily solve it if we unpack the matrix forms.

In[344]:=
```
DSolve[{x'[t] == xd[t], xd'[t] == -g}, {x[t], xd[t]}, t]
```

Out[344]=
$$\left\{\left\{x[t] \to -\frac{g\, t^2}{2} + c_1 + t\, c_2,\ xd[t] \to -g\, t + c_2\right\}\right\}$$

We can also solve it on sight or by repacking the above in matrix forms.

In[345]:=
```
Φk.Ψ + Gk.u
```

Out[345]=
```
Gk.u + Φk.{{x}, {xd}, {xdd}}
```

In[346]:=
```
RSolve[(n + 1)^2 p[n + 1] == n^2 p[n] + z, p[n], n]
```

Out[346]=
$$\left\{\left\{p[n] \to \frac{z}{n} + \frac{c_1}{n^2}\right\}\right\}$$

In[347]:=
```
integersFrom[0][[2]][][[2]][][[2]][]
```

Out[347]=
```
{3, integersFrom[3 + 1] &}
```

In[348]:=
```
ClearAll[cume];
cume[{x_, n_}, z_] :=
  With[{K = 1 / (n + 1)}, {x + K * (z - x), n + 1}];
Fold[cume, {0, 0}, {55, 89, 144}]
ClearAll[cume];
```

Out[350]=
```
{96, 3}
```

In[352]:=
```
ClearAll[cume];
cume[{var_, x_, n_}, z_] := With[{K = 1 / (n + 1)}, With[
    {x2 = x + K (z - x), ssr2 = (n - 1) var + K n (z - x)^2}, {ssr2 / Max[1, n], x2, n + 1}]];
FoldList[cume, {0, 0, 0}, {55, 89, 144}]
ClearAll[cume];
```

Out[354]=
```
{{0, 0, 0}, {0, 55, 1}, {578, 72, 2}, {2017, 96, 3}}
```