1. **Introduction.** This is a “mock-up” of how the full power of literate programming might be useful to R programmers. It presents a simple (but interesting) function that computes the Kolmogorov–Smirnov statistic from empirical data. Then it shows some examples of that function in use.

(The author apologizes for any awkward coding, due to the fact that this is actually his very first attempt to program in R.)

\[\text{Define the function 5}\]
\[\text{Test the function 9}\]
2. The Kolmogorov–Smirnov statistic. The empirical distribution function \( F_n(x) \) of a set of samples \( X_1, X_2, \ldots, X_n \) is defined to be

\[
F_n(x) = \frac{\text{number of } X_1, X_2, \ldots, X_n \text{ that are } \leq x}{n}.
\]

For example, the jagged line shown here is an empirical distribution function taken from §3.3.1 of the book *Seminumerical Algorithms*:
§2   KS   THE KOLMOGOROV–SMIRNOV STATISTIC

3. The curved line in that illustration is another cumulative distribution function, $F(x)$. One way to measure goodness of fit, proposed by A. N. Kolmogorov in 1933 and modified by N. V. Smirnov in 1939, is to compute

$$K^+_n = \sqrt{n} \sup_{-\infty < x < +\infty} (F_n(x) - F(x));$$

$$K^-_n = \sqrt{n} \sup_{-\infty < x < +\infty} (F(x) - F_n(x)).$$

Here $K^+_n$ measures the greatest amount of deviation when $F_n$ is greater than $F$, and $K^-_n$ measures the maximum deviation when $F_n$ is less than $F$. The normalizing factor $\sqrt{n}$ ensures that $K^+_n$ and $K^-_n$ will converge to a limiting distribution as $n \to \infty$, if the $X$’s are independent samples from $F$. 
4. The given data comes from distribution $F$ if and only if $F(X_j)$ has the uniform distribution between 0 and 1.

Hence the obvious way to compute $K_n^+$ and $K_n^-$ is to start by sorting the $X$’s so that $X_1 \leq X_2 \leq \cdots \leq X_n$. Then we can compare $Y_j = F(X_j)$ to the “ideal” value $j/n$.

However, the time needed for sorting is of order $n \log n$. We will use an improved method suggested by T. Gonzalez, S. Sahni, and W. R. Franta [ACM Transactions on Mathematical Software 3 (1977), 60–64], who noticed that linear time suffices if we place the unsorted samples into $n$ bins.

Indeed, if we put $Y_j$ into bin $\lfloor Y_j \rfloor$, the final statistics depend only on the smallest and largest elements in each of the $n$ bins. Other elements in those bins don’t contribute to the extremes that are measured by $K_n^+$ and $K_n^-$. 
5. Coding details.

\[ \text{Define the function 5} \equiv \]
\[ \text{lit.ks.test=function(x,p) } \{ \]
\[ \quad n=\text{length}(x) \]
\[ \quad y=n*p(x) \]
\[ \quad \text{Return FALSE if } y \text{ doesn’t make sense} \]
\[ \quad \text{Insert } y \text{ into bins, remembering extreme values} \]
\[ \quad \text{Find the overall extreme values } lo, hi \]
\[ \quad \text{return } c(lo,hi)/\text{sqrt}(n) \]
\[ \} \]

This code is used in section 1.
The main idea is to keep track of \( lb[j] \) and \( ub[j] \), the smallest and largest element of bin \( j - 1 \), as well as \( count[j] \), the total number of elements in that bin, for \( 1 \leq j \leq n \).

(This is a situation where C programmers wish that R had 0-origin indexing.)

\[
\langle \text{Insert } y \text{ into bins, remembering extreme values } 6 \rangle \equiv
\]

\[
lb=1:n
ub=count=\text{array}(0,\text{dim}=n)
\]

\[
\text{for (j in 1:n) }
\]

\[
\text{ \quad yy=y[j]; k=yy} \% \% 1+1
\]

\[
\text{ \quad if (yy} < \text{lb[k]) } \text{lb[k]=yy}
\]

\[
\text{ \quad if (yy} > \text{ub[k]) } \text{ub[k]=yy}
\]

\[
\text{ \quad count[k]=count[k]+1}
\]

This code is used in section 5.
7. Here’s the key logic that makes it all work.

\[
\begin{aligned}
&\text{Find the overall extreme values } \text{lo, } \text{hi} \ 
\equiv \\
&\text{hi=lo=j=0} \\
&\text{for } (k \text{ in } 1:n) \{ \\
&\text{if (count[k]) } \\
&\quad \text{if (lb[k]-j>lo) lo=lb[k]-j} \\
&\quad j=j+\text{count[k]} \\
&\quad \text{if (j-ub[k]>hi) hi=j-ub[k]} \\
&\} \\
\end{aligned}
\]

This code is used in section 5.
8. Of course a good subroutine intended for general use will check to see that bad parameters haven’t been supplied. Otherwise the computer might hang up with subscripts out of range.

\( \text{Return \textbf{FALSE} if } y \text{ doesn’t make sense} 8 \) \( \equiv \)

\[
\begin{align*}
\text{OK} &= \text{FALSE} \\
\text{if (sum(y<0)) message(’That CDF has negative values!’)} \\
\text{else if (sum(y>n)) message(’That CDF has values > 1!’)} \\
\text{else if (sum(y==n)) }
\text{for (j in 1:n)} \\
\text{if (y[j]==n)} \\
\text{message(’x[’,j,’] is too high: ’,x[j])}
\text{else OK=TRUE }
\text{if (!OK) return(FALSE)}
\end{align*}
\]

This code is used in section 5.
9. **A few unit tests.** First let’s make sure that those error messages are properly issued.

\[
\begin{align*}
&\text{Test the function 9}\equiv \\
&\text{lit.ks.test(-1, function(x) return(x))} \\
&\text{That CDF has negative values!} \\
&[1] \text{ FALSE} \\
&\text{lit.ks.test(2, function(x) return(x))} \\
&\text{That CDF has values > 1!} \\
&[1] \text{ FALSE} \\
&\text{lit.ks.test(c(.6,.23,pi,-.1,1), punif)} \\
&x[3] \text{ is too high: 3.14159265358979} \\
&x[5] \text{ is too high: 1} \\
&[1] \text{ FALSE}
\end{align*}
\]

See also section 10.
This code is used in section 1.
10. And finally, we also want to obtain a valid result when we supply valid parameters.

(The built-in system function \textit{ks.test} computes the slightly different statistics $D_n^- = K_n^- / \sqrt{n}$ and $D_n^+ = K_n^+ / \sqrt{n}$.)

\begin{verbatim}
\{ Test the function 9 \}
\begin{verbatim}
    x=c(.999,.21,.64,.87,.22)
    lit.ks.test(x,punif)/sqrt(5)
\end{verbatim}
\end{verbatim}

\begin{verbatim}
[1] 0.27 0.18
\end{verbatim}

\texttt{ks.test(x,punif,alternative="less")}
\noindent $D^- = 0.27$, p-value $= 0.4134$

\texttt{alternative: the CDF lies below the null hypothesis}

\texttt{ks.test(x,punif,alternative="greater")}
\noindent $D^+ = 0.18$, p-value $= 0.651$

\texttt{alternative: the CDF lies above the null hypothesis}
11. Index.

count: 6, 7.
González-Arce, Teófilo
Francisco: 4.
hi: 5, 7.
Kolmogorov, Andrei
Nikolaevich: 3.
ks.test: 10.
lb: 6, 7.
lit.ks.test: 5, 9, 10.
lo: 5, 7.
OK: 8.
punif: 9, 10.
Sahni, Sartaj Kumar: 4.

Smirnov, Nikolai
Vasilievich: 3.

sqrt: 5, 10.
sum: 8.
ub: 6, 7.

yy: 6.
Define the function 5 Used in section 1.

Find the overall extreme values lo, hi 7 Used in section 5.

Insert y into bins, remembering extreme values 6 Used in section 5.

Return FALSE if y doesn’t make sense 8 Used in section 5.

Test the function 9, 10 Used in section 1.
KS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>The Kolmogorov–Smirnov statistic</td>
<td>2</td>
</tr>
<tr>
<td>Coding details</td>
<td>5</td>
</tr>
<tr>
<td>A few unit tests</td>
<td>9</td>
</tr>
<tr>
<td>Index</td>
<td>11</td>
</tr>
</tbody>
</table>